UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI DI PARMA

DOTTORATO DI RICERCA IN
“TECNOLOGIE DELL’INFORMAZIONE”

CICLO XXXIV

DECISION-MAKING, PLANNING AND CONTROL USING DEEP REINFORCEMENT LEARNING FOR AUTONOMOUS DRIVING VEHICLES

Coordinatore:
Chiar.mo Prof. Marco Locatelli

Tutore:
Chiar.mo Prof. Massimo Bertozzi

Dottorando: Alessandro Paolo Capasso

Anni 2018/2021

Alla mia Famiglia

Summary

List of Figures

Introduction

The autonomous driving field has become a worldwide topic of interest in the last years both for academia and industrial purposes; its emergence is a direct consequence of the technological growth of recent years, but it mainly derives from the desire to reduce human error in motor vehicle crashes, which is estimated around 90% ([1]). However, this innovative evolution is not the result of the last few years, but it has been a process started decades ago which concerned the evolution of hardware and software systems especially in recent years thanks to the rapid growth of Artificial Intelligence.

In this chapter, a history of vehicle driving automation is given showing such evolution over the decades as well as a brief overview of the research work carried out in this three years, explaining the purpose of the project and the technologies used to implement it.

0.1 History of Vehicle Automation

The history of vehicle automation started about a hundred years ago with Francis Houdina, which in 1925 developed a radio-controlled car which received radio signals via an antenna that controlled its speed and direction [2]. The modified Chandler Sedan drove through the streets of Manhattan without anyone at the steering wheel, followed by a second vehicle containing the car's operator trailed.

By the 1950s, the Radio Corporation of America (RCA) replaced radio communication with guide wires embedded in the road surface to control the vehicle [3].

(a) *RCA* Lab car

(b) Stanford Lab Cart

Figure 1: (1a) shows the test performed by *RCA* Labs in which the autonomous vehicle receives signals from electrical wires embedded in the road, while (1b) represents the Stanford Lab cart able to drive without external support

This test proved the feasibility of the autonomous driving, but the cost to implement such infrastructure was unsustainable (Figure 1a). For this reason, by the 1960s Stanford University researches focused their attention on building droids (Figure 1b) that could drive autonomously without external assistance, using cameras to observe the

(a) Mercedes van

(b) Mercedes 500 SEL

Figure 2: Mercedes van (2a) and Mercedes 500 SEL (2b) set up by Ernst Dickmanns and his team

environment and computers to process information, following white lines and avoiding obstalces [4].

This new approach combined with the technological progress in CPUs and computer vision, sped up the process of vehicle driving automation. In 1977 the engineers at the University of Tsukuba's Mechanical Engineering Lab developed the first self-driving passenger vehicle able to travel at speeds up to 20 miles per hour on Japanese roads using two cameras to detect street markings [5]. In 1986 the German engineer and professor at Bundeswehr University Munich Ernst Dickmanns with his research group set up a Mercedes van (Figure 2a) with a computer and cameras testing the system on streets without traffic; as a continuation of such project, they developed a new self-driving Mercedes 500 SEL (Figure 2b) in 1994 able to detect and track obstacles and performed more complex maneuvers as lane change [6]. In 1995 the robotic department of Carnegie Mellon University developed a modified 1990 Pon-

Figure 3: Self-driving Lancia Thema set up by professor Alberto Broggi and his team at the University of Parma in 1996 driving autonomously along italian highways in a 6-day trip of 1900 km

tiac Trans Sport driving autonomously for the 98.2% of the trip from Pittsburg to Los Angeles [7]. The vehicle was equipped with a computer, a camera and a GPS receiver: it was the first time an autonomous vehicle used this technology.

In those years, also in Italy the first tests of autonomous driving were being carried out. In particular, in 1996 a group of researchers of the University of Parma guided by professor Alberto Broggi, developed the ARGO project by setting up a Lancia Thema with two cameras (Figure 3). The project, consisting of a 6-day trip of 1900 km along italian highways, was named *Mille Miglia in Automatico (One Thousand Automatic Miles)* during which the vehicle was able to drive autonomously for the 94% of the journey with an average speed of 90 km/h [8].

An important contribution was given by the Defense Advanced Research Projects Agency (DARPA), which between 2004 and 2007 set three autonomous driving challenges attracting several researchers and companies from all over the world. In the first two competitions autonomous vehicles have to drive autonomously for 150 miles in the Mojave Desert in California; in the first one [9] none completed the challenge, while the second one was won by Stanford University with a Volkswagen Touareg [10] (Figure 4). The last competition was helded in a US air force base set up as urban area and it was won by the Tartan Racing team using a Chevrolet Tahoe [11] (Figure 5).

Another milestone was reached again by the University of Parma in 2010 through the group named VisLab. They ran the Vislab Intercontinental Autonomous Challenge (VIAC) [12], travelling from Parma to Shanghai for 9900 miles in 100 days with four driverless vehicles equipped with stereo cameras, laser scanners and a GPS receiver (Figure 6). Given the complexity of the trip, it would be impossible for the vehicles determine the route; for this reason, in the first vehicle the operators take control only in the case a decision on the road had to be taken, collecting data for the whole duration of the trip; the second one drove 100% autonomously, following the route defined by the first vehicle sent to the following one as GPS waypoints.

The recent advances in artificial intelligence, in particular in Deep Learning techniques [13], have further sped up the automation process for self-driving cars. Nowadays we can observe cars equipped with driving support algorithms and even cars

Figure 4: Volkswagen Touareg winner of the DARPA Grand Challenge 2005

Figure 5: Chevrolet Tahoe winner of the DARPA Grand Challenge 2007

Figure 6: The four self-driving Piaggio Porters developed by VisLab which travelled from Parma to Shanghai

able to drive autonomously on highways, on low-traffic roads and in specific and predetermined areas. However, the question concerning on when autonomous cars will be able to drive on all types of roads and under all kinds of environmental and traffic conditions, still remains without a certain answer and it seems to be related more to road infrastructures and human driving habits than to the technological progress.

0.2 Research Overview

The research project consists in the development of Deep Learning techniques and in particular of Deep Reinforcement Learning [14] algorithms for the implementation of maneuver execution systems using both discrete and continuous action-space; both approaches are trained in simulation, but always with a particular focus on the deployment of such systems in real-world contexts. In particular, we started implementing a module capable of performing the roundabout immission safely choosing discrete actions to modulate longitudinal behavior of the vehicle and then testing it in real scenarios on board of real sefl-driving car; afterwards, we focused on the development of an algorithm able to drive the car through intersections predicting both longitudinal and lateral vehicle behaviors. We prove that the development of decision-making systems through the use of Reinforcement Learning techniques aims at solving the main limitations of rule-based approaches, typically used for the resolution of such tasks; indeed, the performance of these systems are strongly inefficient in dense traffic conditions where negotiation and interaction with other road users and a good understanding of the surroundings dynamics are essential in order to avoid unwanted behavior.

Finally, in the last part of the project we developed a Reinforcement Learning planner, implementing a beta version able to drive smoothly and safely in obstacle-free real-world environments.

Chapter 1

Autonomous Driving Architecture

The development of an autonomous vehicle is a process that involves several steps, from the architecture of the car itself at hardware level to the design and the implementation of the software able to drive the vehicle safely. Indeed, considering that this process began almost one hundred years ago with a radio-controlled car, it is clear that as well as the techniques involved in the development of self-driving cars, also the architecture has been subjected to several variations. In this chapter, an explanation of the levels of vehicle automation is given together with a description of a typical autonomous driving stack.

1.1 Autonomous Driving Levels

The Society of Automotive Engineers (SAE) defines six levels of automation, from the *Level 0* in which there is no driving automation to *Level 5*, the full autonomous vehicle; in particular:

- *Level 0*: there is no automation in this level; the human is fully responsible for the behavior of the vehicle, even if there could be some driver support system as the emergency breaking or lane-keeping assistance.
- *Level 1*: it is the lowest level of automation. This level provides a single auto-

mated system for the steering or braking and acceleration; an example is the adaptive cruise control, that allows the car to adapt its speed depending on the distance to the vehicle ahead, or the lane following assistance.

- *Level 2*: it consists in advanced driving assistance systems, also called as ADAS. In this level the vehicle can take the control of both steering and acceleration/breaking in some specific scenarios like highways, but the driver must always supervised the system all the time. Examples of this technology are the Tesla Autopilot or the Cadillac Super Cruise systems.

- *Level 3*: it is also known as conditional driving automation and combines several advanced driving assistance and artificial intelligence systems in order to make complex decision depending on the environmental changes around the vehicle. Also this level requires that a human driver must remain alert to take the control of the vehicle in case of system failures.

- *Level 4*: in this level the human driver supervision is not required in most circumstances since the vehicle can also handle system failures; however, the driver has the possibility to take the control of the car. The autonomous vehicle is able to drive in a specific regions under certain weather conditions and this is the highest level currently present on some roads. An example is the Waymo robotaxi (Figure 1.1) able to drive autonomously in the specific Phoenix metropolitan area.

- *Level 5*: this is the full driving automation in which the vehicle can drive autonomously in all the environment under all weather condition and there is no need that the driver takes the control of the vehicle.

We are still far away to see *Level 5* autonomous cars on our streets, not only for a technological limitation but also for a social acceptability [15]. This confidence must be achieved not only by the final customer who will use the autonomous vehicle, but also by the whole society, from the other road users to the local governments and authorities that should provide appropriate laws and infrastructures in order to

Figure 1.1: The Waymo robotaxi able to drive autonomously through the streets of Phoenix

provide the adequate tools for testing and deploying autonomous vehicles on the streets.

1.2 Autonomous Car Architecture

From the brief historical background given in Section 0.1, it is clear that the technological progress affected also the autonomous car architecture, starting from a simple one (Figure 1b) to a more complex architecture (Figure 1.1). It is possible to define two kind of architectures: generally the first one involves the use of sensors, perception systems, localization, planning and control, while the second one is the end-to-end approach in which the information provided by sensors is mapped into a direct control of the vehicle.

Later in the thesis we will explain how the systems developed in this research project are embedded in the autonomous driving architecture; before explaining these two different architectures, we will present the sensor part typically used on board of the autonomous car.

1.2.1 Sensors

Even if the two architectures are rather different, they both need sensors. The most used sensors embedded in autonomous vehicles are:

- Cameras: they provide the vehicle the ability to observe the environment ant its users. The technological progress in both hardware and software allows the use of high quality resolution cameras processing frames in real time and performing tasks like object detection and classification. The use of camera pairs provide the possibility to use the stereo vision obtaining accurate 3D measurements ([16], [17]). However, the quality of images is strongly dependent on weather and light conditions.

- Lidars (light detection and ranging): they scan the environment emitting laser beams, measuring the return time of the reflected pulse after hitting a physical surface; in this way, it is possible to determine the distance between the autonomous vehicle and the objects in the scenario. It can be used to generate dense point clouds to perform tasks like 3D object detection and tracking ([18], [19]), but they could produce inaccurate results with bad weather conditions or with particular sun lights angle.

- Radars (radio detection and ranging): they emit electromagnetic waves in specific directions helping the autonomous car retrieving surrounding object measurements like their angles, ranges and speeds. Radars can work under any weather condition but it is difficult to use them alone; indeed, they are generally coupled with cameras or lidars performing more accurate object detection task ([20]).

- Sonar (sound navigation and ranging): they are used to detect short-range objects sending sound pulses and listening the return echo from the physical surfaces ([21]). They work well in bad weather conditions and low light, but they hardly detect small objects or multiple objects moving at fast speeds.

- Inertial Navigation Systems (INS): it is a navigation device used to calculate position, orientation, and velocity of a self-driving car, combining an Inertial

Measurement Unit (IMU), typically composed by accelerometers, gyroscopes and magnetometers, a global navigation satellite system receiver (GNSS) and a microprocessor that is used to perform real-time sensor fusion ([22]).

- Global Position Systems (GPS): it is a navigation system using satellites, a receiver and algorithms to provide location, speed and time ([23]). The accuracy can vary from 30 centimeters to 5 meters like in urban scenarios where buildings and other obstacles may disturb the signal.

1.2.2 Typical Architecture

The typical architecture of an autonomous driving car is composed by a set of sensors, perception algorithms, prediction of the behavior of other road users, localization and mapping, planning and control. The scheme proposed in Figure 1.2 aims to illustrate a simplified architecture, but some of the proposed modules may be unnecessary or others may be added to this pipeline based on the tasks to be performed. In Section 1.2.1 we provided a description of the main sensors embedded in a typical autonomous driving car, while in the next ones we focus on the remaining modules illustrated in Figure 1.2.

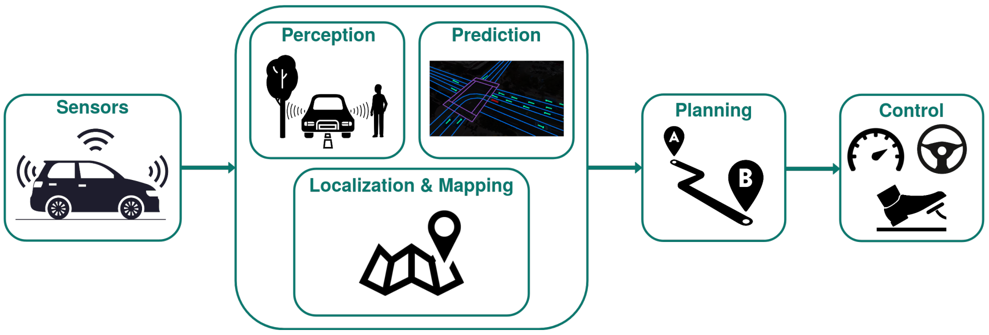


Figure 1.2: Main components of an autonomous driving pipeline

Perception

This module processes data coming from sensors in order to extract the most interesting features understanding the surrounding environment. The *Perception* module is able to extract features in order to retrieve several environmental information like lane markings ([24], [25]), curbs ([26], [27]) or traffic signs ([28], [29]), also predicting also the free-space in which the autonomous car could navigate. Moreover, object detection and classification ([30], [31]) can be performed retrieving road users inside the scenario and their features (size, position, heading, speed). All these information can be embedded over time in order to perform obstacle tracking task: in this way it is possible to know the past positions of a road occupant trying to predict their future actions.

Prediction

Another important module in the development of an autonomous car concerns the prediction of other road users behavior, like pedestrians and vehicles. This is particularly useful to understand the dynamics of the scenario in order to perform safe actions also based on the predicted future behaviors of other road users ([32]). However, a perfect prediction of other road users behavior is quite impossible to achieve, especially for pedestrians which have more degrees of freedom than vehicles, and for this reason it is necessary to include uncertainties on the predictions.

Localization & Mapping

As well as sensors and perception algorithms play a significant role in understanding the surrounding scenario, also the localization task is essential to navigate safely since a localization error of a few meters could have catastrophic consequences. Since the measurements coming from Global Navigation Satellite Systems are often inaccurate, they should always be flanked by algorithms extracting environmental features performing tasks like visual odometry ([33], [34]) or SLAM (*Simultaneous Localization and Mapping*) ([35], [36]). Indeed, the more accurate is the estimation of the autonomous car position in the world, the more precise will be its relationship with the other elements in the environment (center lane, curbs, traffic sings, obstacles, road users). In this way, the planning and control modules will know the best possible measurements between the autonomous car and the other elements in the scenario, performing the correct maneuver safely.

Another element that can dramatically improve localization are HD Maps (High Definition Maps [37]), containing not only the information about road topology, but other relevant data like the exact positions of traffic lights, traffic signs, pedestrian crossings, stoplines, intersections and much more. In this way, the autonomous vehicle drives in a familiar environment making navigation less complex since it already knows what and where the static elements in the scenario will be.

Planning

Given the information retrieved and processed in the earlier modules, the autonomous car should be able to establish which maneuver to perform predicting how to behave in the short-term future. It is clear that the performance of this module is strictly related to the measurements coming from the previous modules that should be as accurate as possible in order to allow the *Planning* module to compute the path the vehicle should follow taking the correct high-level decision like lane keeping, lane changes, overtakings, insertions and many others.

Control

If the *Planning* module defines what to do in a certain situation, the *Control* module determines how to do it. Indeed, it transforms the high-level maneuver coming from the *Planning* module in a sequence of low-level actions, controlling steering angle, throttle and breake, driving the autonomous car smoothly and comfortably.

1.2.3 End-to-End Architecture

A different type of architecture consists of directly mapping sensor data into actions (acceleration, breaking and steering angle), without using additional modules to process environmental features. The typical implementation is based on Deep Learning

([13]) in which a neural network learns the best features automatically in order to solve the task efficiently; however, the approach could be rather different based on the learning process we want to use.

Imitation Learning

The first approach is based on the typical supervised learning in which the system learns and develops new skills observing an expert behavior; this implementation is called Imitation Learning (IL) ([38]). The first promising results were achieved by NVIDIA ([39]), in which they collected a dataset composed by images and steering angle commands driving manually for several hours. Training a neural network with this data, the vehicle was able to control the steering angle autonomously on urban roads, on highways and also in parking lots or on unpaved roads. In this way, they proved that an end-to-end approach for autonomous driving is possible avoiding to develop intermediate tasks like lane detection or obstacle detection and thus using a smaller and more portable system than the one illustrated in Figure 1.2. However, this implementation suffers from several problems: the learned behavior is strictly related to the demonstration data collected by the expert and for this reason the system performance deteriorates in those states outside the dataset; indeed, even without considering the effort in terms of manual driving hours for collecting data, it is almost impossible to include all the corner cases in the dataset.

Learning from Trials and Errors

A different approach is based on Deep Reinforcement Learning (DRL) ([14]), consisting in the combination of artificial neural networks and the Reinforcement Learning (RL) framework ([40]) in which the neural network is trained without observing expert demonstrations in order to learn a policy through a process of trials and errors. In particular, this system involves the interaction between one or more agents with the environment and each time step they execute an action receiving a reward signal based on the action chosen. The goal of the agent is to maximize the total amount of discounted reward obtained during the episode.

However, also this approach suffers from several limitations: simulators are typically required for training Reinforcement Learning algorithms thus increasing the complexity of deploying the system in real world because of the gap between real world and simulated data. Moreover, another important limitation is related to the so called *reward shaping* ([41], [42]): indeed, the training process is strictly related to the reward signal and the design of such function is often tricky especially when trying to solve complex task as in the autonomous driving field.

In the next chapter, the key concepts of Deep Reinforcement Lerarning are explained in order to understand the theory behind the algoritms used in the implementation of the system developed during this research project.

Chapter 2

Deep Reinforcement Learning

The use of Deep Reinforcement Learning techniques has grown exponentially in recent years achieving impressive results not only for the resolution of games like Atari [43] and Go [44] for discrete control space problems, but also for continuous ones [45] especially in robotics field. In recent years, Deep Reinforcement Learning has also proved the ability to solve autonomous driving tasks, in particular those ones related to the planning and control systems, thus capturing the attention of the industrial sector.

As we discussed in Section 0.2, Deep Reinforcement Learning algorithms have been developed and tested in this research project in order to implement maneuver execution techniques and systems able to control steering angle and acceleration driving the vehicle autonomously. In this section we will start with a brief introduction on Machine Learning ([46], [47]) and Deep Learning ([48], [49]) theory and then focusing on the theory of the Deep Reinforcement Learning techniques used in this project.

2.1 Machine Learning

Machine Learning (ML) represents one of the fundamental areas of Artificial Intelligence (AI) and deals with the realization of systems based on data for the synthesis of new knowledge. Starting from observations, data or experiences, that are typically called *training sets*, the system is able to automatically learn without being explicitly programmed and without human assistance. Moreover, the performance achieved should guarantee good generalization capabilities in order to obtain good results also for those data outside the *training set*. 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

The *supervised learning* consists in the use of labeled dataset (the *training set*) in which each example or observation is composed by an input and its output value (the label). The input data is given to the model continuously in order to train it predicting the desired output. Using this approach it is possible to solve two different problems:

- Classification, in which the model is trained to predict the class/label of specific data points. It consists in finding a mapping function starting from input data and assigning the probability that such data belongs to a specific category. In autonomous driving field this process is mostly used to recognize objects (pedestrians, vehicles, bicyclists and many others).
- Regression, in which continuous value are predicted learning a curve that explains the distribution of the given data points; such curve is then used to make predictions for new data.

Typically, having a large dataset makes the system more robust and should also increase the generalization capibility; however, finding labeled data or the labeling proccess itself, is often expensive and time-consuming.

2.1.2 Unsupervised Learning

In this process the *training set* is not labelled, means that we train the model using the input data but without having a prior knowledge of the desired outcome. The system should discover interesting features and information for its own, group the data based on similarities and learning a compressed format of the data. Clustering or anomaly detection are two use cases of this learning method. However, the training process is harder than the *supervised learning* and the model could require more time to converge to an optimal solution.

2.1.3 Semi-supervised Learning

Semi-supervised learning is a combination of *supervised* and *unsupervised learning* in which the *training set* is typically composed by a small amount of labeled data and a large part of unlabeled observations, thus solving the problem of labeling.

2.1.4 Reinforcement Learning

Reinforcement Learning consists of learning to behave from experience without using labeled data. One or more agents interact with an environment, performing actions and noticing the consequences of their decisions and thus receiving a reward signal that could be positive or negative depending on the action performed. This framework is well suited to problems in which actions must be performed in sequence and the goal is long-term like in game-playing or robotics.

2.2 Deep Learning

Deep Learning is a subset of Machine Learning based on learning from input data and predicting the desired output, using neural networks which try to imitate human brain mechanisms as we will explain in the next section. The network is composed by several layers (called *hidden layers*) able to extract and learn the interesting features and thus using this knowledge to generalize to data outside the *training set*. Deep Learning is particularly useful having large data to process; indeed, the combination of Big Data with the technological progress in computational computing (GPUs) and research related to the Artificial Intelligence field, have contributed to make Deep Learning popular in the last few years.

2.2.1 Artificial Neural Network

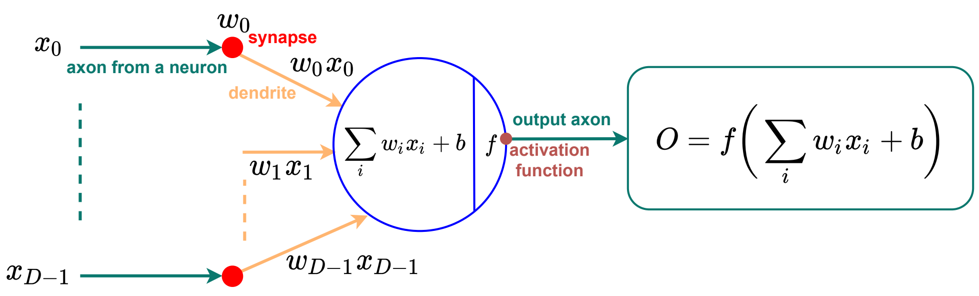


Figure 2.1: Artificial neuron

The fundamental element of a neural network is the artificial neuron, whose schematic representation is illustrated in Figure 2.1. As explained in the previous section, a neural network tries to imitate a human brain and so the artificial neuron do the same with biological neuron; the latter receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result, and then outputs the final outcome. All biological neurons have the same four basic components: dendrites, soma, axon, and synapses. Dendrites act like input channels receiving the input through the synapses of other neurons; then, the soma processes these incoming signals producing an output that is sent out to other neurons through the axon and synapses. In the same way, an artificial neuron is composed by:

- Input D (dendrites);
- Weights (synapses): w_i with $i = 0, 1, \cdots, D-1$ are adjusted continuously during the training phase in order to obtain the correct output.
- A function (soma) that combines inputs and weights defining the behavior of the neuron. Moreover, a non-linear function f (called the *activation function*) is applied to simulate the behavior of biological neuron and to avoid the concatenation of simple linear transformation that would not be able to approximate complex functions. Considering all these elements, the final output of the neu-

ron will be:

$$O = f\left(\sum_{i=0}^{D-1} x_i w_i + b\right) \tag{2.1}$$

where b is the bias, that is learned as well as the weights and describes how far the predictions are from their intended values.

- Output O (axon).

Moreover, the most used *activation functions* are:

- Rectified Linear Unit (ReLU), defined as $ReLU(x) = max(0, x)$;
- Hyperbolic Tangent (tanh), defined as $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$;
- Sigmoid, defined as $S(x) = \frac{1}{1+e^{-x}}$.

The artifical neurons are clustered together trying to reproduce what happens in the human brain, such that information can be processed in a dynamic, interactive, and self-organizing way. In artificial neural networks it occurs by creating connected layers, as shown in Figure 2.2.

Although there are useful networks which contain only one layer, or even one element, most applications require networks that contain at least three types of layers:

- Input Layer: it receives data either from observations (that could be provided by a dataset or by a simulator as for the case of Reinforcement Learning) or directly from electronic sensors in real-time applications.
- Output Layer: it is the last layer of the neural network and produces the end result in order to communicate it to the external world, to another process or to other devices like mechanical control systems.
- Hidden Layers: they are middle layers between the input and the output layers; in most networks, each neuron in a hidden layer receives the signals from all of the neurons of the preceding layer; after a neuron performs its function, it passes its output to all of the neurons in the next layer, providing a feedforward path to the output.

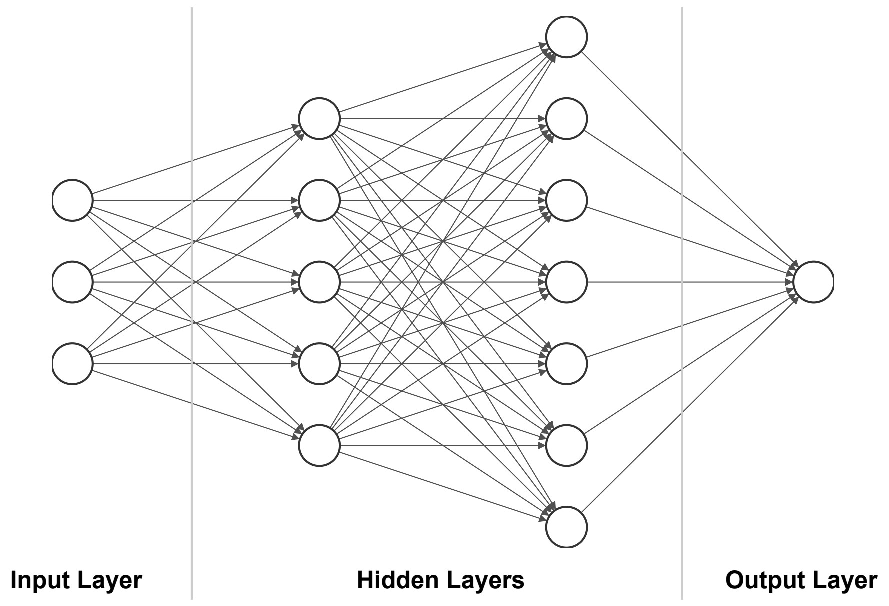


Figure 2.2: Schematic representation of a neural network architecture showing the input, hidden and output layers

The activation functions and the layers of a neural network have to be differentiable to allow the use of the *backpropagation* technique [50], that is the key mechanism by which the neural network weights are learned automatically. In this way, it is possible to compute the partial derivatives of the loss function with respect to each network parameters (that is nothing but the prediction error of the model) and thus using a gradient-descent optimization algorithm [51] in order to update the weights of the neural network.

Fully Connected Layer

One of the most used layer implementing a neural network is the *Fully Connected Layer*. Each neuron of a layer is connected to all neurons of the previous one, such that the input/output signals pass through all the neurons of the neural network. Typically, the last layers of a neural network are fully connected in order to process the data extracted by previous layers to produce the final outcome.

Convolutional Neural Network

One of the most important breakthroughs in Computer Vision using Deep Learning has been made through the use of Convolutional Neural Networks (CNNs). They are typically composed by a sequence of convolutional layers, in which the first ones

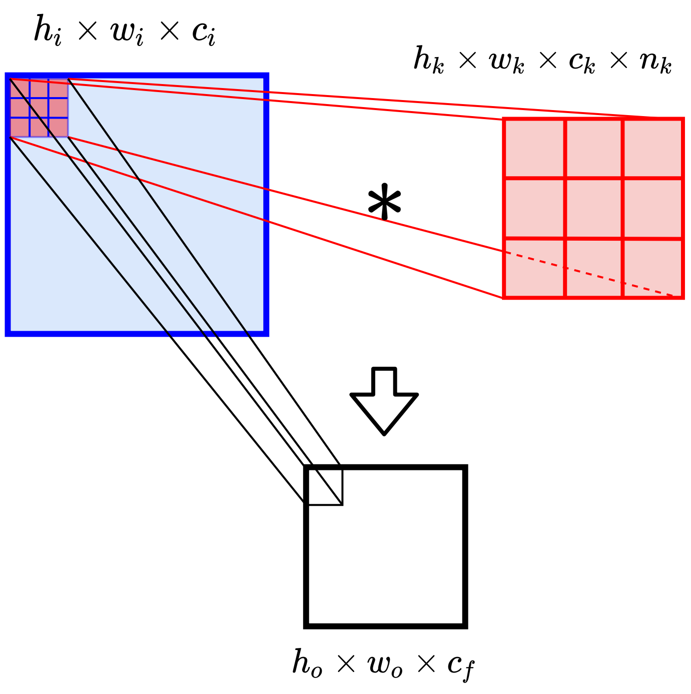

$h_i \times w_i \times c_i$
$h_k \times w_k \times c_k \times n_k$
$*$
$h_o \times w_o \times c_f$

Figure 2.3: Convolution operation over images. A 3×3 filter (the red one) is scrolled over the whole image (blue square) computing the convolution operation in order to produce a new feature map (black square). h, w and c represent the heights, widths and number of channels of the input, output and the filter used for the convolution operation, while n is the number of filters we want to use

extract low-level features and finally high-level ones are extracted by last layers, producing the final *feature map* that gives the network a better understanding of the image.

Also this algorithm is inspired by the behavior of the human brain in which neurons are able to focus on restricted region of the visual field known as *receptive field*; the entire visual field is covered by partially overlapping the *receptive fields* of different neurons. In the same way CNNs have the ability to assign importance to objects or specific regions inside an image, extrapolating spatial and temporal dependencies through the use of *filters/kernels*. These are scrolled over the image applying the convolutional operation, such that a new matrix with different dimensions and size is produced.

The values of the *filters* are the parameters that the network has to learn and since their dimensions are independent of the image size, the number of learned parameters are lower than those required by a fully connected layer that performs general matrix multiplications. Moreover, the learned filters are able to recognize patterns independently on the position in the image, thus increasing generalization capabilities.

Figure 2.3 shows how the convolution operation between filters and images works. The 3×3 filter in figure is sliding over the whole image moving of a stride s at a time, and the convolution operation is applied to calculate each output element generating a new *feature map*. Moreover, a padding p could be applied in order to increase the size of the image and to avoid loosing information. In the example of Figure 2.3, h_i, w_i and c_i represent the height, width and the number of channels of the input image (the blue one) respectively. c_i is equal to one in the example, but in a typical RGB image it will be equal to 3. h_k, w_k and c_k are the height, width and the number of channels of the kernel respectively. The latter must be equal to the number of channels of the input image, while n_k represents the number of filters we want to use. Finally, h_o, w_o and c_o are the height, width and the number of channels of the output feature map respectively and while c_0 will be equal to n_k, h_o and w_o are computed as follows:

$$\begin{aligned} h_o &= \left\lfloor \frac{h_i + 2p - h_k}{s} + 1 \right\rfloor \\ w_o &= \left\lfloor \frac{w_i + 2p - w_k}{s} + 1 \right\rfloor \end{aligned} \tag{2.2}$$

Finally, a pooling layer is typically applied in order to downsample the feature map produced by the convolutional layer. A patch (generally of 2×2 size) is sliding over the feature maps, performing one of the following function:

- Average Pooling, which calculates the average value for each patch on the feature maps.
- Max Pooling, which computes the maximum value for each patch on the feature maps.

In this way, a downsampling of the feature maps is achieved giving the model the capability to be invariant to local translation, since small changes in the location of the input features will result in the same location of the feature map as output.

2.3 Reinforcement Learning Theory

As well as neural networks, and thus artificial neurons, try to imitate the operations and connections that occur in the human brain, also Reinforcement Learning is inspired by how the human being learns his first vital functions from birth and in many cases without supervision. Indeed, as well as children learn to walk or to play simply by trying and observing the consequences of their actions, so in Reinforcement Learning an agent is left free to take sequential actions within an environment without any supervision. The only element the agent receives is a reward signal that gives an idea about the goodness of the action chosen; the goal of the agent is to maximize the sum of these future rewards.

It is clear that this approach strongly differs from supervised learning in which each observation/data is labeled such that we always have the correct prediction during the training time. However, Reinforcement Learning also differs from unsupervised learning since there is still a signal that aims to evaluate the behavior of the agent, while in unsupervised learning the purpose is to discover unknown structures in the data.

The term Deep Reinforcement Learning comes from the combination of Reinforcement Learning and Deep Learning techniques. In this way, the interaction agent-environment and the Reinforcement Learning theory (further explained in this chapter) combined with a neural network that predicts the action the agent has to perform, are the key elements that allow agent to achieve the optimal behavior in order to solve the desired tasks.

2.3.1 Markov Decision Process

Reinforcement Learning involves the interaction between one or more agents and an environment. Each time step t, the agent performs an action a_t in the state s_t and receives a reward signal r_t that is typically a numerical value, and as a result of the action performed it will find itself in another state s_{t+1}. In this way, the environment will change as a consequence of the action taken by the agent or because of its natural evolution. An example of this interaction is illustrated in Figure 2.4.

The Reinforcement Learning process can be defined as a Markov Decision Process (MDP), defined as $M = (S, A, P, r, \gamma)$, where:

- S is the set of states, that could be fully or partially observable as in the case of Partially Observable Markov Decision Process (POMDP);
- A is the set of actions and they could be discrete, choosing high-level commands like turn left/right or move forward/backward, or they could be continue, useful for those tasks in which we want the agent learns not only which the best action is for a specific state, but also how to perform it.
- P is the state transition probability $P(s_{t+1}|s_t, a_t)$ in which, as defined by the Markov property, the next state of the environment only depends on the current state and the action choosen from the agent;
- r represents the reward function;
- γ is the discount factor ($[0, 1]$) that modulates the importance of future rewards; the closer will be to 0 the more interested the agent will be in maximizing immediate rewards.

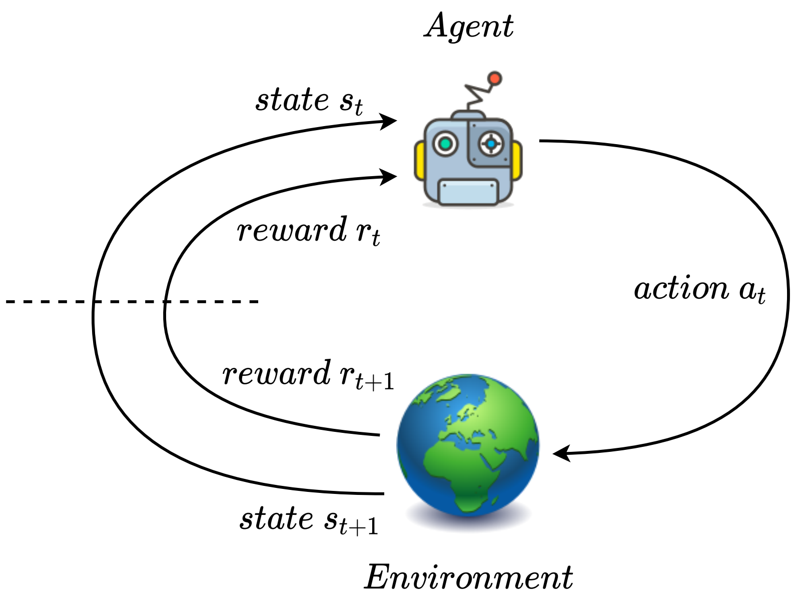


Figure 2.4: Markov Decision Process illustrating the interaction between the agent and the environment. Each time step the agent performs an action a_t observing a state s_t and receives a reward signal r_{t+1}, finding itself in a different state s_{t+1} as a consequence of the performed action. This interaction continues until a terminal state is achieved by the agent

In Reinforcement Learning, the episode refers to a sequence of states, actions and rewards which ends with a terminal state at time T. The aim of the agent is to find the best policy π that maximizes the discounted sum of the rewards, that is called *expected return* defined as follows:

$$R_t = \sum_{t}^{T} r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t} r_T \tag{2.3}$$

The policy $\pi(a|s)$ is nothing but a mapping function that for each state $s_t \in S$ predicts an action $a_t \in A$ in order to maximize future discounted rewards, thus achieving its goal. This does not mean that the agent must always perform what it believes is the best action, but a good trade-off between exploration and exploitation has to be achieved. Exploration consist in taking the action that is not optimal for the agent, that could be more dangerous but it may reveal new and possibly better behaviors; on the other hand exploitation consists of taking the best assumed action with respect to the observed state, namely the safest action for the agent. It is clear that we do not have a prior knowledge about when it is optimal to explore rather than exploit and for this reason it is crucial to delineate a strategy with a good trade-off between these two behaviors. An example is the ε-greedy algorithm in which the agent explores choosing a random option with ε probability and exploits with $1-\varepsilon$ probability.

In the following sections of this chapter we will give further explanation on the basic theory of Reinforcement Learning algorithms and techniques related to those ones used and implemented in this research project.

2.3.2 Value Function

An important element in Reinforcement Learning is the *value function* V_π which evaluates how good is a certain state for the agent to be in. This estimation is defined in terms of expected future rewards or expected return with respect to the policy π the agent will follow. Mathematically speaking the *state-value function* V_π is defined as:

$$V_\pi(s) = \mathbb{E}_\pi(R_t|s_t) \tag{2.4}$$

It is possible to estimate which is the best policy, comparing the *state-value functions* obtained with different policies. In this way the optimal policy π^* will be the one that estimates the optimal *value function* V_{π^*}, namely the one with the maximum value compared to all the other *value functions*, such that:

$$V_{\pi^*}(s) = \max_\pi V_\pi(s) \tag{2.5}$$

In the same way, in *action value function* methods such as *Q-learning* [52], Q_π gives the expected return under a policy π after performing an action a_t in a state s_t:

$$Q_\pi(s_t,a_t) = \mathbb{E}_\pi(R_t|s_t,a_t) \tag{2.6}$$

Also in this case it is possible to estimate the optimal *action value function* Q_{π^*} following the optimal policy π^*:

$$a_t = \max_{a \in A} Q_{\pi^*}(s_t, a) \qquad (2.7)$$

An important property is that for any policy π and state s_t it is possible to define the *state value function* V_π iteratively, using the value of the next state. This property is called *Bellman equation* and is defined as:

$$V_\pi(s_t) = \mathbb{E}_\pi(R_t | s_t) = \mathbb{E}_\pi(r_{t+1} + \gamma V_\pi(s_{t+1})) \qquad (2.8)$$

The same idea can be applied to the *action value function* Q_π:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi(R_t | s_t, a_t) = \mathbb{E}_\pi(r_{t+1} + \gamma Q_\pi(s_{t+1} | a_{t+1})) \qquad (2.9)$$

2.3.3 Monte Carlo and Temporal Difference learning

Monte Carlo (MC) learning is suitable only for the so called episodic MDP, namely for those tasks that have a terminal state. Indeed, in MC methods all the rewards are stored such that at the end of the episode the estimations of the *value functions* (or *action value functions*) are updated in all the visited states. The update formula used in MC learning is defined as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \qquad (2.10)$$

where α is the learning rate and R_t is the sum of discounted rewards as defined in Equation 2.3.

Instead, Temporal Difference (TD) learning does not require the end of the episode to update the *value function* of a given state, but it only needs the next time step as in the *Bellman equation* (Equation 2.8). Indeeed, the simplest TD method known as TD(0) uses the obtained reward r_{t+1} and the estimate $V(s_{t+1})$ to update $V(s_t)$:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \qquad (2.11)$$

where the part in the brackets is called *TD error*. Estimating a value starting from another estimation is called *bootstrapping*.

2.3.4 Model-free vs Model-based Reinforcement Learning

As explained in the previous section, the agent performs an action a_t after observing a state s_t, receives a reward r_{t+1} and thus finding itself in another new state s_{t+1}. For complex environment, the states perceived by the agent may change not only because of its behavior, but also for other external factors making impossible having a full knowledge of the environment and thus predicting how the state will evolve from s_t to s_{t+1} and what would be the future reward r_{t+1}. These two distributions represent the *model* of the environment and in *Model-based Reinforcement Learning* it is learned by the agent, thus predicting how will be the future state s_{t+1} and future reward r_{t+1} given the action performed in s_t; in this way, the number of trials and errors the agent has to execute to reach an optimal policy is reduced, increasing the sample efficiency of the solution.

A different approach is represented by *Model-free Reinforcement Learning* in which the *model* is not learned and the only aim of the agent is to perform an action observing the state s_t without making any assumption on s_{t+1} and r_{t+1} and the dynamics of the environment is implicitly learned by the model.

2.3.5 On-policy, Off-policy and Offline Reinforcement Learning

On-policy, *Off-policy* and *Offline* Reinforcement Learning are different methods that generate data in different ways. The firts one aims at improving the latest learned policy that has been used to collect data and take actions; an example is *SARSA* (state, action, reward, state', action'), where state' and action' are the new state-action pair. The method is explained in the pseudo-code in Algorithm 1.

In *Off-policy* methods the agent appends its experience in a buffer (or replay buffer) collecting this data with different policies π_k such that the buffer is composed by observations retrieved with $\pi_0, \pi_1, \cdots, \pi_k$; data are sampled from the buffer and used to obtain the new policy π_{k+1}. An example is *Q-learning*, whose pseudo-cose is illustrated in Algorithm 2.

Both *On-Policy* and *Off-policy* methods belong to the category of the so called *Online* Reinforcement Learning, since in both cases the agent interacts with the envi-

Algorithm 1 SARSA, On-policy TD learning

```
1: Initialize discount factor $\gamma$ and learning rate $\alpha$
2: Initialize $Q(s,a)$ for all $s \in S$, $a \in A$
3: for each episode do
4:     Initialize $s$
5:     Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
6:     for each step of the episode do
7:         Take action $a$, observe $r$ and $s'$
8:         Choose $a'$ from $s'$ using policy derived from Q (e.g., $\varepsilon$-greedy)
9:         $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
10:        $s \leftarrow s'$
11:        $a \leftarrow a'$
12:    end for
13:    Until $s$ is terminal
14: end for
```

Algorithm 2 Q-learning, Off-policy TD learning

```
1: Initialize discount factor $\gamma$ and learning rate $\alpha$
2: Initialize $Q(s,a)$ for all $s \in S$, $a \in A$
3: for each episode do
4:     Initialize $s$
5:     for each step of the episode do
6:         Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
7:         Take action $a$, observe $r$ and $s'$
8:         $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
9:         $s \leftarrow s'$
10:    end for
11:    Until $s$ is terminal
12: end for
```

ronment updating the policy using the experience gathered during the training phase. A completely different approach is represented by *Offline* Reinforcement Learning, which involves a static dataset of fixed interactions used by the agent to learn the best policy. In this case there is no more interaction with the environment and the dataset is not updated with the agent experience.

2.3.6 Policy Gradient methods

In contrast to *value function* or *action value function* methods that improve the *value* estimation improving the policy implicitly, *Policy Gradient* algorithms follow gradients with respect to the policy thus updating the policy itself.

In *Policy Gradient* methods, that belong to *Model-free* Reinforcement Learning algorithms, the policy π is defined by a set of parameters θ by which $\pi(a|s,\theta)$ represents the probability of taking action a in state s following the policy π parametrized with θ. Then, the aim of such methods is to update such parameters θ in order to find the optimal policy $\pi^*(a|s)$.

Considering the parametrized policy π_θ with the purpose of maximizing the expected return through the objective function $J(\pi_\theta)$:

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} r_t\right] = \mathbb{E}_{\pi_\theta}[R_\tau] \tag{2.12}$$

It is possible to maximise the objective function $J(\pi_\theta)$ in order to maximise the expected return adjusting the parameters θ of the policy. The gradient of the objective function is $\nabla J(\pi_\theta) = \nabla \mathbb{E}_{\pi_\theta}[R_t]$ and the update formula is defined as follows:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta) = \theta_t + \alpha \nabla \mathbb{E}_{\pi_\theta}[R_\tau] \tag{2.13}$$

We can rewrite the gradient in this form:

$$\begin{aligned}
\nabla J(\theta) &= \nabla \mathbb{E}_{\pi_\theta}[R_\tau] \\
&= \nabla_\theta \sum_\tau P(\tau|\theta) R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau|\theta) R(\tau) \\
&= \sum_\tau P(\tau|\theta) \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} R(\tau) \\
&= \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) \\
&= \mathbb{E}_{\pi_\theta}(\nabla_\theta \log P(\tau|\theta) R(\tau))
\end{aligned} \tag{2.14}$$

in which the log-derivative trick ($\nabla_\theta \log(z) = \frac{\nabla_\theta z}{z}$) is used to obtain $\nabla_\theta \log P(\tau|\theta)$. Then, considering that:

$$\begin{aligned}
\nabla_\theta \log P(\tau|\theta) &= \nabla_\theta \log p(s_0) + \sum_{t=0}^{T-1} (\nabla_\theta \log P(s_{t+1}|s_t, a_t) + \nabla_\theta \log \pi_\theta(a_t, s_t)) \\
&= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t)
\end{aligned} \tag{2.15}$$

and replacing the result of the Equation 2.15 in Equation 2.14 we obtain the final formula of the gradient of the objective function:

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t) R(\tau) \right) \tag{2.16}$$

Finally, the update formula in Equation 2.13 can be written as follows:

$$\theta_{t+1} = \theta_t + \alpha R(\tau) \frac{\nabla_\theta \pi_\theta(a_t|s_t, \theta)}{\pi(a_t|s_t, \theta)} \tag{2.17}$$

This update uses the gradient ascent rule since we want to increase the probability of taking action a_t in the observed state s_t; moreover, this increment is proportional to the obtained return, but it is normalized by the probability of the action in order to compensate the fact that more likely actions will be taken more often.

Algorithm 3 Advantage Actor-Critic

```
1:  Initialize parameters $\theta^{\pi}$ and $\theta^{v}$
2:  Initialize step counter t ← 1
3:  Initialize episode counter E ← 1
4:  Set $n$ as the number of steps required to perform an update
5:  while $E < E_{max}$ do
6:      $\Delta_{\pi} = 0, \Delta_{v} = 0$
7:      $t_{up} = t$
8:      get $s_t$
9:      while $s_t$ not terminal and $(t - t_{up}) < n$ do
10:         Execute $a_t$ following the policy $\pi(a_t|s_t, \theta^{\pi})$
11:         Get the reward $r_t$ and new state $s_{t+1}$
12:         $t \leftarrow t + 1$
13:     end while
14:     $R = \begin{cases} 0, & \text{if } s_t \, is terminal \\ V(s_t, \theta^{v}), & \text{otherwise} \end{cases}$
15:     for $i \in t-1, \cdots, t_{up}$ do
16:         $R = r_i + \gamma R$
17:         $\Delta_{\pi} = \Delta_{\pi} + \nabla_{\theta^{\pi}} \log \pi(a_i|s_i, \theta^{\pi})[R - V(s_i, \theta^{v})]$
18:         $\Delta_{v} = \Delta_{v} + \nabla_{\theta^{v}} R - V(s_i, \theta^{v})^2$
19:     end for
20:     $\theta^{\pi} = \theta^{\pi} + \alpha_{\pi}\Delta_{\pi}$
21:     $\theta^{v} = \theta^{v} + \alpha_{v}\Delta_{v}$
22:     $E \leftarrow E + 1$
23: end while
```

2.3.7 Actor-Critic Methods

An approach that combines the *action value* algorithms with the policy gradient learning is the *Actor-Critic* method. In such case, the policy is parametrized and optimized, as in policy gradient methods (Section 2.3.6), but at the same time the state value function is estimates thus reducing the variance of the updates and permitting bootstrapping using the TD-learning algorithm.

Moreover, the name *actor-critic* is related to the fact that two networks are used: one to predict the action, namely the policy network (the *actor*) parametrized as θ^{π}, and the other one to predict the *state value function* (the *critic*) parametrized as θ^{v}.

In the so called *Advantage Actor-Critic*, the return R_{τ} used in Equation 2.17 is replaced by th *Advantage* (A_b), that tell us if a state is better or worse than expected, namely how is the executed action with respect to the expected action:

$$A_b = r_t + \gamma r_{t+1} + \cdots + \gamma^{b-1} r_{t+b-1} + V(s_{t+b}, \theta^{v}) - V(s_t, \theta^{v}) \tag{2.18}$$

where b is the number of the obtained reward before bootstrapping. If the *Advantage* is greater than zero it means that the chosen action is better than expected and the updates will be directed towards a more frequent execution of that action, otherwise we will encourage the agent to take the opposite of such action. The update formulas of *actor* and *critic* parameters are defined as:

$$\theta_{t+1}^{\pi} = \theta_t^{\pi} + \alpha_{\pi} A_b \frac{\nabla_{\theta^{\pi}} \pi(a_t|s_t, \theta^{\pi})}{\pi(a_t|s_t, \theta^{\pi})} = \theta_t^{\pi} + \alpha_{\pi} A_b \nabla_{\theta^{\pi}} \log \pi(a_t|s_t, \theta^{\pi}) \tag{2.19}$$

$$\theta_{t+1}^{v} = \theta_t^{v} + \alpha_v A_b \nabla_{\theta^{v}} V(s_t, \theta^{v}) \tag{2.20}$$

where α_{π} and α_v are the learning rates for the *actor* and *critic* respectively. Algorithm 3 shows the pseudo-code of the *Advantage Actor-Critic* in which the updates defined in Equation 2.19 and 2.20 are performed each n-steps or when a terminal state is reached.

2.3.8 Asynchronous Advantage Actor-Critic (A3C)

The asynchronous version of the *Advantage Actor-Critic* (A3C) [53] involves the use of multiple instances of the environment in which the actor-learners are inde-

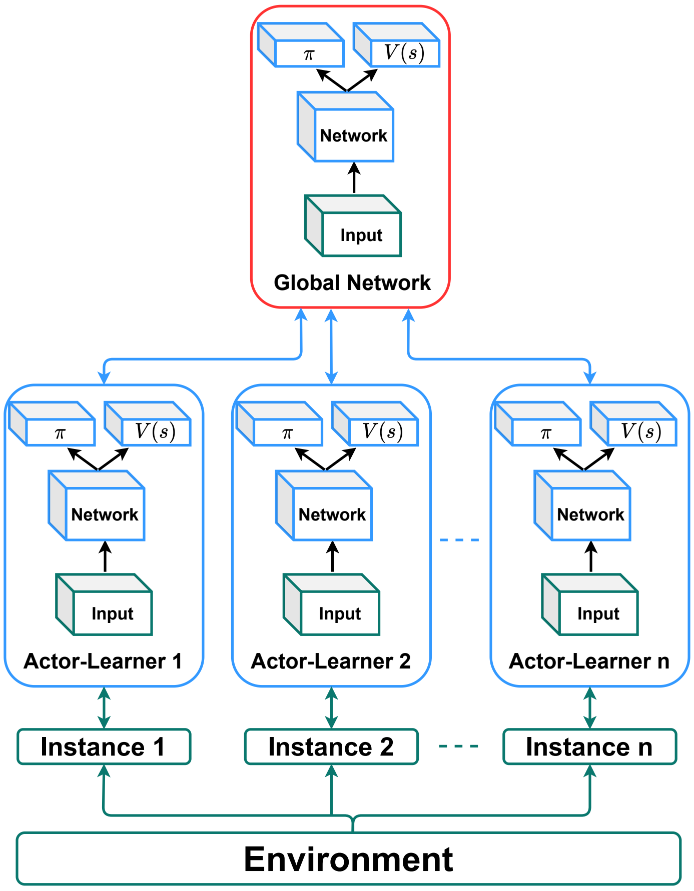


Figure 2.5: *Asynchronous Advantage Actor-Critic* scheme in which it is possible to notice the different instances of the environment that allow actor-learners to collect update not correlated to each other and to have an indipendent environment configuration. The agents retrieve the last version of the global network at the beginning of each episode: this is executed asynchronously as well as the policy updates during the episode

pendent with each other; they only share a global network, pulling the last version of it (both the *actor* and the *critic*) at the beginning of each episode. Each actor-learner computes the accumulated gradients during the episode and send them to the global network asynchronously every $n-$steps: in this way, the agents involved in the learning phase may have different version of the global network thus increasing the exploration process. This element combined with a different and independent environment configuration perceived by each agent that makes the actor-learners updates not correlated to each other, increases the learning process stability. A scheme of this learning process is illustrated in Figure 2.5, while a pseudo-code of A3C is shown in Algorithm 4.

2.3.9 Multi-Agent Deep Rinforcement Learning

In previous sections we only considered the case in which a single agent interacts with the environment or with a single independent instance of it. In this way, the environment perceived by the agent changes as a result of the agent actions or accordingly to deterministic rules. For example, an environment could be populated by agents not involved in the learning process, following a rule-based behavior; predicting, even implicitly, the behavior of those entities is not a difficult task for the actor-learner, but designating an environment governed by deterministic rules makes the system unsuitable for the real wolrd, in which uncertainty elements will surely be encountered.

On the other hand, a Multi-Agent approach consists in the use of two or more agents sharing the same environment or the same instance of it and they are both involved in the learning process. The approach could be *centralized* or *decentralized*: in the first case a single brain/controller is deployed across all the actor-learners and it has a full knowledge of the environment; the action predicted by the single brain will be broadcast to all agents and possibly decoded into different sub-actions. Instead, in the *decentralized* approach the agents act independently using different policies or slightly different versions of the same shared policy or global network, as illustrated in Figure 2.6. In this case, the environment will change not only as a consequence of a single agent actions, but also because of other actor-learners, that

Algorithm 4 Asynchronous Advantage Actor-Critic (A3C)

```
 1: // The global shared parameters are $\theta_\pi$ (actor) and $\theta_v$ (critic)
 2: // The specific actor-learner parameters are $\theta'_\pi$ (actor) and $\theta'_v$ (critic)
 3: Initialize parameters $\theta_\pi$ and $\theta'_v$
 4: Set $n_{inst}$ as the number of environment instances
 5: Set $n$ as the number of steps required to perform an update
 6: Run $n_{inst}$ environment instances
 7: for environment e = 1, $\cdots$, $n_{inst}$ do [running concurrently until termination]
 8:     Initialize step counter t $\leftarrow$ 1
 9:     $\theta'_\pi = \theta_\pi$, $\theta'_v = \theta_v$
10:     $\Delta'_\pi = 0$, $\Delta'_v = 0$
11:     while $s_t$ not terminal do
12:         $t_{up} = t$
13:         Get $s_t$
14:         while $s_t$ not terminal and $t - t_{up} < n$ do
15:             Execute $a_t$ following the policy $\pi(a_t|s_t, \theta'_\pi)$
16:             Get the reward $r_t$ and new state $s_{t+1}$
17:             $t \leftarrow t + 1$
18:         end while
19:         $R = \begin{cases} 0, & \text{if } s_t \, is terminal \\ V(s_t, \theta'_v), & \text{otherwise} \end{cases}$
20:         for $i \in t-1, \cdots, t_{up}$ do
21:             $R = r_i + \gamma R$
22:             $\Delta'_\pi = \Delta'_\pi + \nabla_{\theta'_\pi} \log \pi(a_i|s_i, \theta'_\pi)[R - V(s_i, \theta'_v)]$
23:             $\Delta'_v = \Delta'_v + \nabla_{\theta'_v} R - V(s_i, \theta'_v)^2$
24:         end for
25:         $\theta^\pi = \theta^\pi + \alpha_\pi \Delta'_\pi$
26:         $\theta^v = \theta^v + \alpha_v \Delta'_v$
27:     end while
28: end for
```

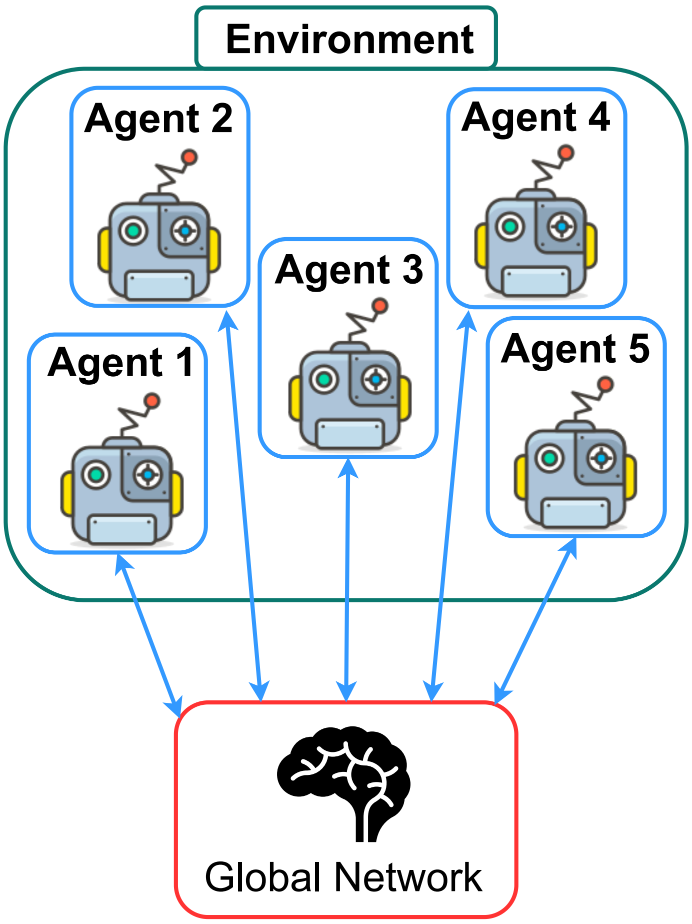


Figure 2.6: Multi-Agent Deep Reinforcement Learning using a *decentralized* approach in which an arbitrary number of agents (5 in the proposed example) interact with the same environment and with each other. The actor-learners involved in the training process share a global network and each agent will behave following a slightly different version of it, but updating and improving the same policy

will not follow deterministic rules thus increasing the complexity of predicting other agent intentions. However, the introduction of noise and uncertainty elements (as the non-deterministic behavior of the actor-learners populating the environment) will surely reduce the gap between simulated and real-world scenario, thus increasing the robustness of the model.

In Multi-Agent configuration, agents interact not only with the environment, but also with each other; especially in those cases where a common goal has to be reached and a cooperative behavior should emerge during the training process, a communication or a negotiation should be achieved among the actor-learners in order to perform actions safely and thus reaching the goal. Depending on the task, a communication channel or a shared memory accessible by the agents, could be provided in order to share information about the environment or their internal states, thus facilitating the interaction and negotiation processes. However, when this communication protocol is not provided, the agents have to implicitly learn to negotiate and predict other agent intentions.

In this research project, both single-agent and multi-agent approaches have been developed, providing a novel implementation of the *Asynchronous Advantage Actor-Critic* (A3C). Moreover, as explained in Section 2.3.4, the learning phase of *Model-free* methods is based on the trials and errors process and for this reason the use of simulators is required for training agents. In the next chapter a brief overview on the use of simulator in Reinforcement Learning is given as well as a description of the one used in this reasearch project.

Chapter 3

Simulators in Deep Reinforcement Learning

A key role in the Reinforcement Learning training pipeline is played by simulators that represent nothing but the environment, or the set of environments, in which one or more agents are trained. As we will explain in the next sections of this chapter, there are different kind of simulators and the choice of which one to use is strictly related to the task we want to solve: in the game theory they are simply virtual environments on which agents are trained to solve a specific task or game, while in robotics field they are used to simulate real-world scenarios (sometimes using a synthetic representation of them) or to provide a physics engine.

Moreover, as explained in the previous chapter, the typical Reinforcement Learning training is based on the process of trails and errors, in which agents learn how to behave in specific environments through experience, performing actions and observing the consequences of them. In Deep Reinforcement Learning this process is supported by a neural network that has to learn the relevant features of the environment, understanding its dynamics and then figuring out how to act properly, thus achieving the desired behavior. Moreover, during the training phase we want the agent takes actions that lead to catastrophic outcomes, since only observing the consequences of different actions it can learn how to behave correctly and thus reaching an optimal policy. Especially in robotics field, training a robot in a real environment is not always feasible for different reasons:

- The trials and errors process requires thousands or millions of episodes, but depending on both the hardware we have at disposal and clearly the task we want to solve, this process could require only few hours in simulation. Instead, a training performed in a real-world environment is more time and resource consuming since a human supervision on the robot would be required throughout training.
- The actions taken by a policy during the training process may be catastrophic causing serious damages to the users involved in the environment and to the real robot itself.

In the following sections we will analyze the different types of simulators, showing the most used ones in particular in the autonomous driving field, and finally presenting the simulators used in this research project.

3.1 OpenAI Gym

OpenAI Gym [54] is a collection of environments, from games to physics-based ones, provided for developing and comparing reinforcement learning algorithms in order to increase their reproducibility. This is a very useful tool since researchers have to implement only the code of the algorithm they wants to test without being concerned about the environment implementation, and most importantly a detailed benchmark for each environment is provided in order to compare the performances of their algorithm with the existing ones.

There are different available tasks to solve on OpenAI Gym:

- Classic control games, like Acrobot [55], CartPole [56], MountainCar [57] and Pendulum (Figure 3.1), and toy text.
- Algorithmic calculations.

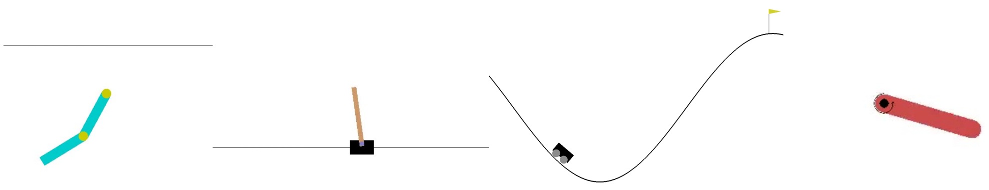


Figure 3.1: Classic control environments provided by OpenAI Gym (from left to the right: Acrobot, CartPole, MountainCar and Pendulum)

- Atari games, including Pong, Space Invaders, Breakout and many others (Figure 3.2).

Figure 3.2: Some of Atari games widely use for training and testing Reinforcement Learning algorithms

- 2D and 3D Robot control using a physics engine called MuJoCo (Multi-Joint dynamics with Contact [58]), that allows testing complex dynamic systems with contact-rich behaviors.

3.2 Autonomous Driving Simulators

Simulators play an essential role also in the development of autonomous driving systems. They allow to observe the behavior of an algorithm before testing and deploying its final version on the real self-driving car, thus saving time and resources in the development phase; in this way, if we notice unexpected behavior in simulation, we keep improving the algorithm without spending time on testing non-functioning systems in real-world and thus ensure the safety of other road users as well as self-driving drivers. Moreover, in simulation it is possible to reproduce particular condition (or traffic condition) that in real life are rare or may require a long time.

Simulators are not only used for testing autonomous driving systems before deploying them in real-world environments, but also for training Reinforcement Learning algorithms or even for the creation of datasets used for supervised or unsupervised learning.

However, some approaches based on Reinforcement Learning do not use simulators for training agent performing planning and control tasks, but they only provide scalar parameters containing environmental information like positions, velocities and headings of other road users, or information related to the topology of the road like positions or the number of lanes. This kind of approach strongly reduces the complexity of what would be the agent surroundings and thereby simplifying the learning process, but exposes the system to a high risk of overfitting on the training scalar parameters. Indeed, the achievements of the last decades on convolutional neural networks (Section 2.2.1) allow the agent to learn relevant features of the environment, achieving a better understanding of its dynamics and thus better generalizing to environments unseen during the training phase.

In this section we focus on those simulators used for training Reinforcement Learning agents and in particular illustrating two different types of simulators: realistic graphic simulators that aim to reproduce real-world scenarios and those ones based on a synthetic representation of the reality, namely reproducing the information of the environment in a different and simplified scheme and thereby reducing the complexity of the model input. Finally, an explanation on simulators used in this reasearch project is given showing their major components and functionalities.

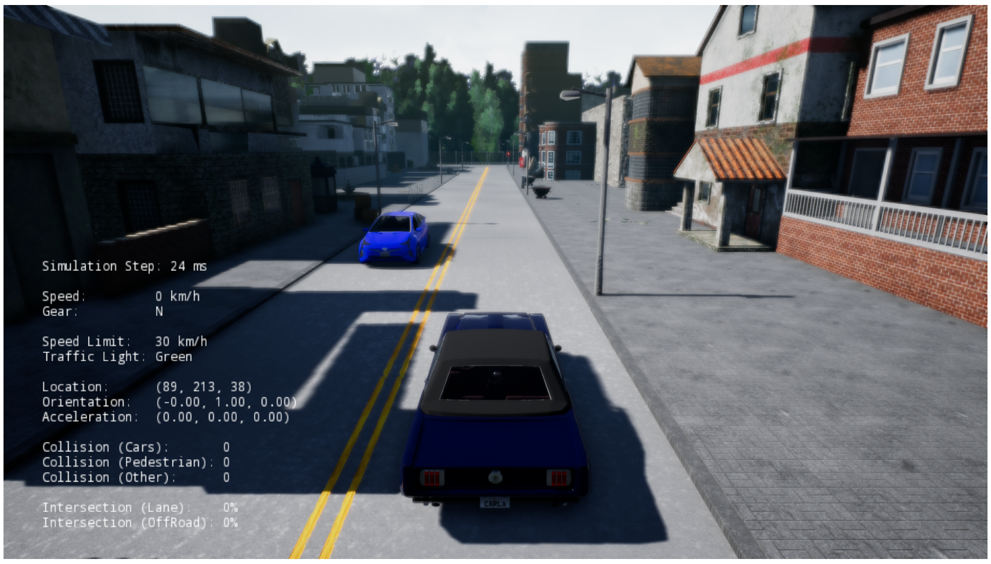
3.2.1 Realistic Graphic Simulators

Realistic graphic simulators aim at reproducing realistic scenarios trying to provide camera images as similar as possible to real images. Thanks to its very realistic physics engine, *Grand Theft Auto V* (GTA V) video game is one of the most realistic graphics simulators [59] widely used to create pseudo-realistic labeled datasets reducing time and cost due to the collection and labelization of real images. However, this kind of simulators do not facilitate the environment customization and they do not allow to use an arbitrary number of sensors on the same vehicle. In addition, it does not provide feedback regarding erroneous driving behaviors or traffic rules violations.

At this purpose, the open-source simulator *CARLA (CAR Learning to Act)* [60] (Figure 3.3) has been developed upon the Unreal game engine in order to support the development, training, and validation of autonomous driving systems. This simulator allows to collect data, setting different traffic driving behaviors, but it can also be used to train and test planning algorithms. Moreover, it supports a flexible customization of vehicles and scenarios, enabling the use of different sensors and the creation of several scenarios and weather conditions.

However, despite the realistic graphics, a domain gap between simulated scenarios and real images is a relevant factor, especially if the final purpose is to deploy the training system in real-world environments. At this purpose, several works aim at reducing the difference between the two domains (composed by simulated and real images) through the use of *Generative Adversarial Neural Networks* (GAN) [61]. This still remains an open problem even if recent achievements have shown promising results ([62]).

Moreover, especially for the training of Reinforcement Learning algorithms for planning and control tasks, an other relevant gap is related to the behaviors between real and simulated traffic cars; these latter often follow rule-based behaviors, making difficult the prediction of human driving intentions for the system when deployed in real-world scenarios. Finally, the trained agent dynamics in simulation should be as



```
Simulation Step: 24 ms

Speed:          0 km/h
Gear:           N

Speed Limit:    30 km/h
Traffic Light:  Green

Location:       (89, 213, 38)
Orientation:    (-0.00, 1.00, 0.00)
Acceleration:   (0.00, 0.00, 0.00)

Collision (Cars):       0
Collision (Pedestrian): 0
Collision (Other):      0

Intersection (Lane):    0%
Intersection (OffRoad): 0%
```

Figure 3.3: CARLA simulator

similar as possible to the one on board of the real self-driving car used for real tests; however, trying to replicate the real vehicle dynamics in simulation is not always an easy task and for example *CARLA* simulator allows the user to set up some vehicle physics parameters.

3.2.2 Simulators based on Synthetic Representation of Reality

A different approach involves the use of simulators which do not try to reproduce real-world scenarios, but to provide a synthetic representation of the reality. These kind of tools are useful for the development of planning and control algorithms, traffic modeling, maneuver execution systems and motion forecasting. In this way, agents will learn a simplified version of the environment, no longer characterized by features derived from complex scenarios or particular elements involved in the simulation, but rather a schematic representation of information processed by sensors on board of the real vehicle.

Typically, the environment populated by agents will be characterized by a sim-

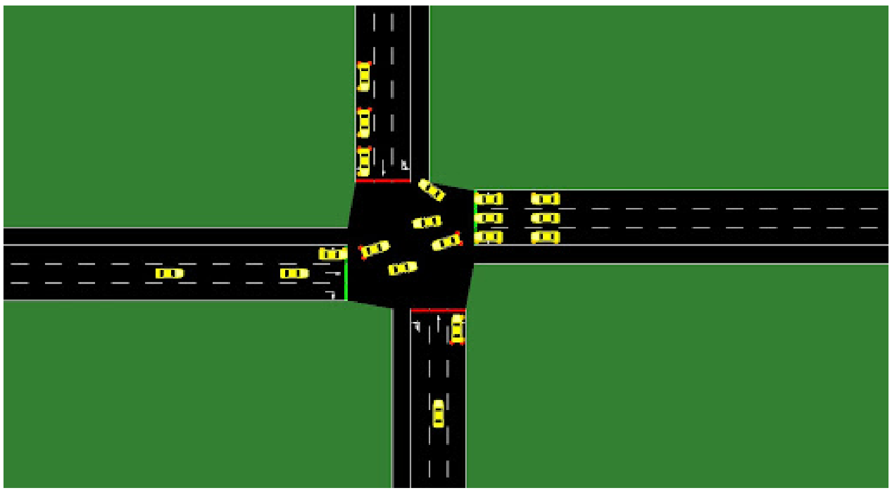


Figure 3.4: SUMO simulator

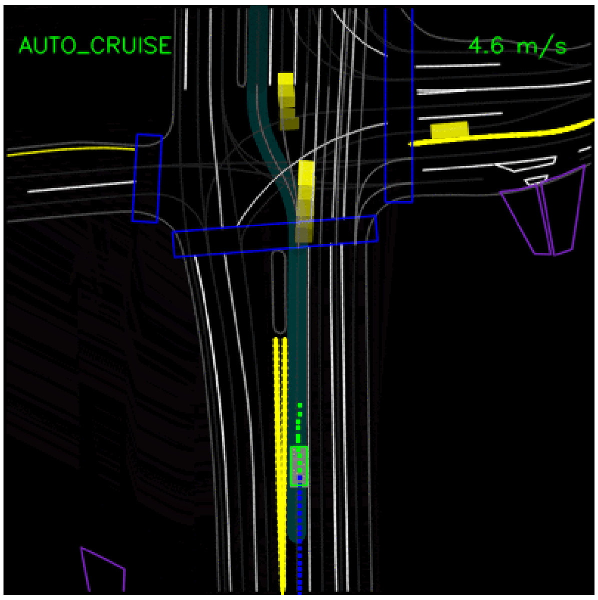


Figure 3.5: Waymo ChauffeurNet

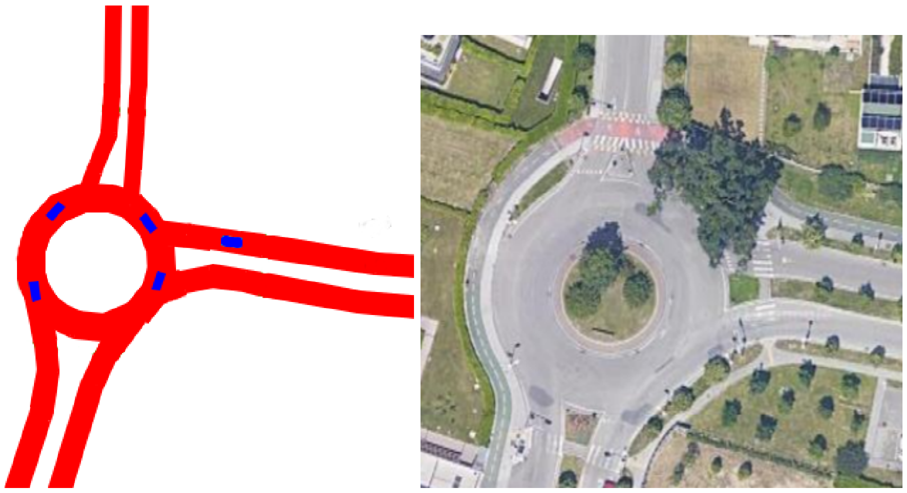
plified top-view representation as in the case of *SUMO (Simulation of Urban MObility)* [63] (Figure 3.4), that is an open-source, microscopic traffic simulator mostly used for traffic forecasting, including the evaluation of traffic lights and route selection, but it also used for testing vehicular communication systems. Another example of simplified simulator is given by Waymo ChauffeurNet [64] (Figure 3.5) in which, as in SUMO, the vehicles are represented by two-dimensional oriented boxes such that the learning process of the agent will be mostly focused on the behavior to solve a specific task instead of learning complex environmental features.

In this way, the domain gap between simulated and real images no longer exists, but if we are interested in deploying our system in real-world scenarios we need to focus on the problems related to the difference between real and simulated traffic behaviors and the dynamics of the trained agents. Such problems are mostly relevant for Online Reinforcement Learning methods since the agent is entirely trained with simulated data, while in other approaches like Imitation Learning (Section 1.2.3), the agents are trained observing realistic data both related to other road users and ego-vehicle behaviors.

In the next section we will present the microscopic traffic simulator developed in [65] and its upgraded version based on High-Definition Maps developed in the last part of this research project.

3.2.3 Multi-Agent Microscopic Traffic Simulator

In the first part of this research project, we started using the simulator implemented in [65] developed with Cairo graphic library [66] used to reproduce synthetic representations of a real environment as illustrated in Figure 3.6. The simulator was developed for training agents to drive safely inside the roundabout illustrated in Figure 3.6a, thus achieving an intelligent traffic road users. The purpose indeed, was to reduce the gap between real and simulated traffic behavior and thereby overcoming the lack of interaction and negotiation that could occurr when using agents following rule-based rules. Indeed, agents were trained in a multi-agent fashion on several instances of the same environment, such that actions taken by an actor-learner may affect not only its own state but also those of other agents, as explained in Section 2.3.9.



(a) Synthetic roundabout representation (b) Real roundabout

Figure 3.6: On the left (3.6a) is illustrated the synthetic representation of a real roundabout in Parma (3.6b), implemented with Cairo graphic library for training agents (the blue boxes in 3.6a) to navigate safely inside the roundabout

As a result, agents learn to cooperate and negotiate implicitly in order to achieve the common goal, driving safely without crashing with each others. In particular, the Asynchronous Advantage Actor-Critic (A3C) (Section 2.3.8) algorithm has been used, training a neural network that predicts discrete actions in order to control the longitudinal behavior of the vehicle, while the lateral control was not learned since agents simply follow the waypoints representing the center lanes of the route.

Agents follow the kinematic bicycle model which consists in a 4-wheel model combining the front and the rear wheels to form a two-wheeled model (or bicycle model). However, we can use such model making some assumptions:

- No-slip condition, namely that there is no lateral or longitudinal slip in the wheels.
- The vehicle has a lumped mass acting at the center of mass.
- Vehicle are moving on a 2D pane.

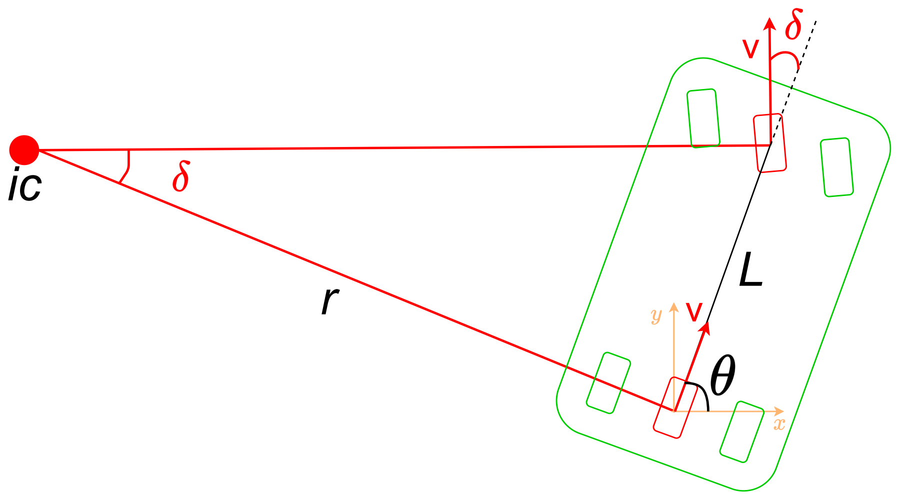


Figure 3.7: An example scheme of the bicycle model

A simple example is illustrated in Figure 3.7 in which *ic* represents the instantaneous centre of rotation, namely the point around which the car is rotating at a given instant, and it is positioned at the intersection of the perpendiculars of the two wheels. We can define the x and y components of the velocity v and the angular velocity $\dot{\theta}$ as follows:

$$\begin{aligned}\dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= \frac{v}{r}\end{aligned} \tag{3.1}$$

where r is the radius of rotation. However, we know that the angle intersecting the front and rear perpendicular lines is still the steering angle δ, and for this reason we can also define $\tan(\delta) = \frac{L}{r}$ ad so $r = \frac{L}{\tan(\delta)}$, thus writing:

$$\dot{\theta} = \frac{v}{r} = v\frac{\tan(\delta)}{L} \tag{3.2}$$

Another way to calculate the angular velocity consists in using the curvature κ defined as the inverse of the radius of rotation r; finally, we can define the final model with acceleration a and curvature k as inputs:

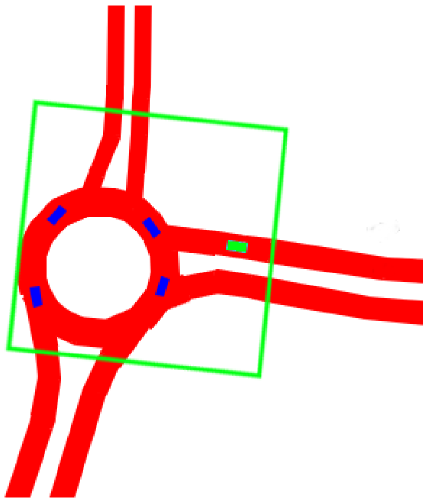
$$\begin{aligned}
\dot{x} &= v\cos(\theta) \\
\dot{y} &= v\sin(\theta) \\
\dot{\theta} &= \frac{v}{r} = v\kappa \\
\dot{v} &= a
\end{aligned} \tag{3.3}$$

3.2.4 State Space

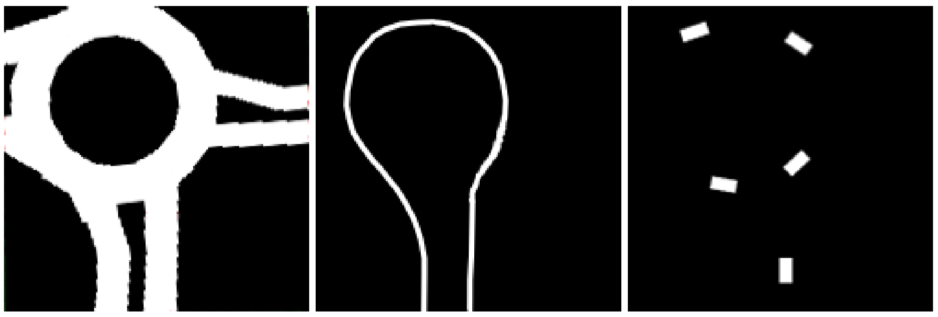
Each actor-learner involved in the environment perceives a surrounding of 50×50 meters, illustrated by the green square in the example in Figure 3.8, observing 40 meters forward, 25 left and right and 10 backward; in the first version of the simulator developed in [65], such surrounding is split in different channels of 84×84 pixels composed by:

- Navigable space (Figure 3.8b) inside which the agent can drive.
- Path (Figure 3.8c) the agents have to follow.
- Obstacles (Figure 3.8d) represented by vehicles inside the 50×50 meters surrounding (blue boxes in Fgure 3.8) included the ego-agent itself (the green one).

The information illustrated in Figure 3.8 can be retrieved by data perceived by sensors and processed by perception and localization systems embedded on the self-driving: the navigable space can be provided by topological maps (or HD maps) known a priori as well as the path the agent should follow, but they could also be computed by perception systems. Finally, the obstacle channel can be reproduced using object detection systems able to detect and track the road users involved in the surrounding of the ego-agent; in this simulators, only vehicles are embedded in the obstacle channel, however any static or dynamic obstacles could be included in such channel by resizing the boxes according to the type of object.



(a) Simulated roundabout



(b) Navigable space (c) Path (d) Obstacles

Figure 3.8: Synthetic representation of a real roundabout in Parma (3.8a) in which the green square represents the 50×50 meters surrounding perceived by the green agent; anyway, each actor-learner involved in the scenario perceived this kind of view. The information contained in such area is split in three different channels representing the navigable space (3.8b), the path (3.8c) the agent will follow, and the obstacles (3.8d) including the ego-vehicle (the green agent)

3.2.5 HD Simulator

For the last part of the research project, we developed a new version of the simulator illustrated in Figure 3.6 using HD Maps (developed internally by the research team), in order both to speed up the creation process of scenarios and to have more information available during the training process, like stop line positions, speed limits, lane boundaries, traffic signs and much more. An example is illustrated in Figure 3.9 showing the full view of the scenario (Figure 3.9a) and the agent surrounding (Figure 3.9b); the information contained in such view is then split in different channels as illustrated in Figure 3.8.

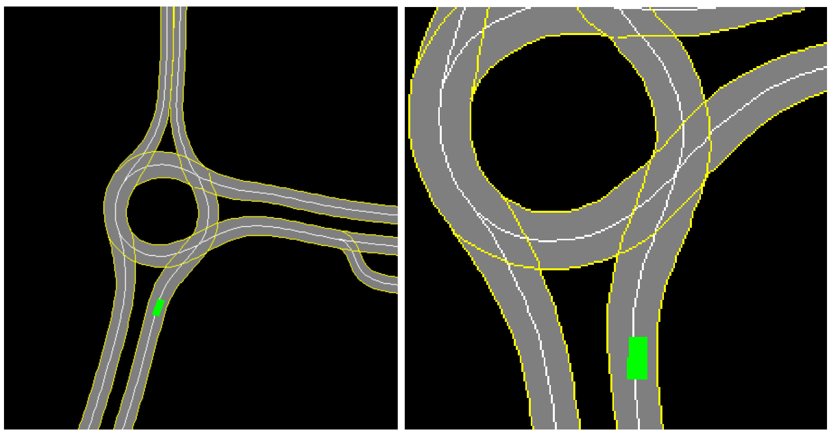

(a) Full view (b) Agent view

Figure 3.9: (3.9a) represents the synthetic top-view representation of the real roundabout in Figure 3.6b created using HD Maps, while in (3.9b) is illustrated the 50×50 meters surrounding perceived by the green agent. White and yellow lines represent centers and lane boundaries respectively

Chapter 4

Intelligent Roundabout Insertion

In this chapter we will explain the first system developed in such research project, which consists in the implementation of a module able to perform the immission maneuver in roundabout scenarios [67]. At the beginning, we started exploring the capabilities of a novel version of the *Asynchronous Advantage Actor-Critic* (A3C) (Section 2.3.8), training agents performing such maneuver in the roundabout illustrated in Figure 3.6a through the training of a neural network that predicts a discrete output value to modulate the longitudinal acceleration of the vehicle. The lateral control is not learned since the agent simply follow the center lanes of the route. Then, we continue increasing the generalization capabilities of the system, thus deploying and testing the model on a real-world roundabout on-board of a self-driving vehicle.

4.1 Roundabout Insertion

Typical approaches that attempt to solve the immission maneuver in roundabout scenarios are based on rule-base methods including time slots and space reservation ([68], [69]); however, such strategies often lead to undefined waits because of their overly cautious behavior especially in case of heavy traffic conditions. Other methods are based on vehicle-to-vehicle (v2v) communication system as in [70], but this is unsuitable in those scenarios also populated by manually driven vehicles.

For this reason, we develop a system based on Deep Reinforcement Learning such that agents learn to perform the roundabout insertion safely, implicitly reaching negotiation and communication capabilities and thus achieving better performances compared to rule-based methods. The traffic agents populated the scenario are represented by those ones trained in [65] which do not follow rule-based methods, but have a more realistic behavior since they have the capability to negotiate and interact implicitly among them driving safely inside the roundabout. In this way, we obtain a scenario fully populated by intelligent agents which are able to negotiate the immission in the roundabout as well as human drivers do in real roundabout scenarios. We will refer to the traffic agents as *passive* agents while those ones specifically trained to perform the immission maneuver as *active* agents.

Moreover, as well as each human driver has its own driving style, the model driving behavior is tuned through the use of a learned parameter that simulates the impetus of the maneuver; in this way, the model learns to perform the immission with different levels of impetus, proving that agents with higher aggressiveness tend to take more risks.

4.2 Training Setup

During the training process we create multiple instances of the scenarios, such that *active* agents learn how to enter in the three-entry-roundabout of Figure 3.6a from all the entries simultaneously; in this way, we achieve a sufficient amount of agents for the learning process to be stable, allowing multiple copies of the agents to learn from every entry in indipendent instances of the roundabout.

In each instance of the environment, a single *active* agent performs the immission starting from the same entry lane and finishing the episode after completing the immission maneuver: in this way, the *active* vehicle can focus on learning the specific entry maneuver. Instead, *passive* vehicles start their episodes from the other two lanes choosing one of them randomly as well as the exit lane (one of the three available). So, each instance of the scenario will be populated by only one *active* agent and at most eight *passive* cars simultaneously.

4.2.1 Input and Output Space

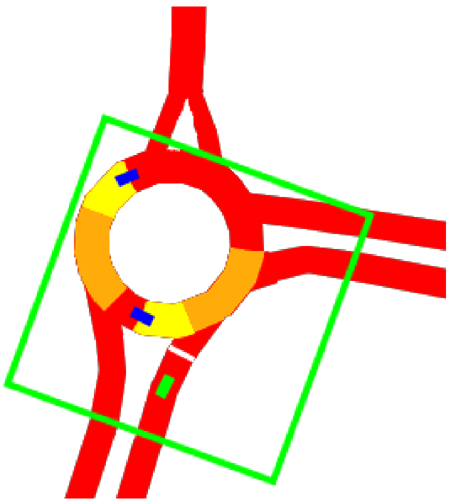
Compared to the channels illustrated in Figure 3.8 and used in [65], for the development of the entry maneuver task we add an additional channel representing the stop line (Figure 4.1e), namely the position where the *active* agent should stop in case it has to give way to other *passive* cars inside the roundabout. In this way, the 50×50 meters surrounding of the *active* vehicle will be split in four channels 84×84 pixels (navigable space, path, obstacles and stop line) as illustrated in Figure 4.1.

The agents are trained using a neural network illustrated in Figure 4.2, that receives two different kind of input: a visual and a numerical one. The visual input consists in a sequence of four temporal frames of the four channels illustrated in Figure 4.1 of size $4 \times 4 \times 84 \times 84$, while the non-visual sensory channel is composed by four elements:

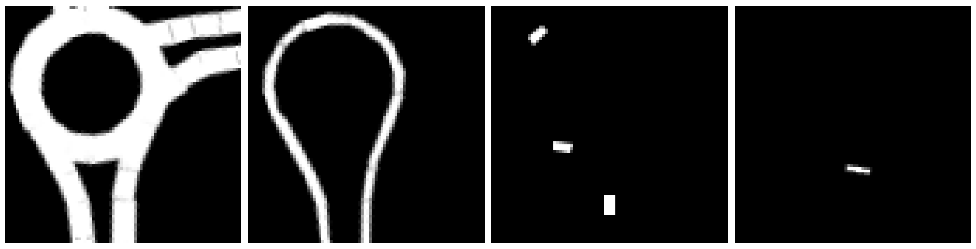
- Agent speed: the current velocity of the agent.
- Target speed: the maximum velocity that agent should not exceed; it can be interpreted as the speed limit of the road and it is sampled randomly at the beginning of each episode.
- Aggressiveness: it is the parameter learned from the model to tune the agent behavior. Such value is kept fixed during the whole episode and it is chosen randomly between $[0, 1]$.
- Last action: the last output value predicted by the neural network.

The neural network outputs the probabilities of three possible actions together with the state-value function estimation. The discrete actions the agent could perform are:

- Permitted: in this case the agent perceives the entry area of the roundabout as free and the neural network predicts that performing the immission will be safe. This state results in a comfort maximum acceleration a_{max} $(+2\frac{m}{s^2})$ unless the target speed is reached.



(a) Simulated roundabout



(b) Navigable space (c) Path (d) Obstacles (e) Stop line

Figure 4.1: Full view of the synthetic simulator (4.1a) illustrating the *active agent* (the green one) and the *passive* ones. The 50×50 meters surrounding of the *active* is split in the four different 84×84 pixels channels representing the navigable space (4.1b), path (4.1c), obstacles (4.1d) and stop line (4.1e)

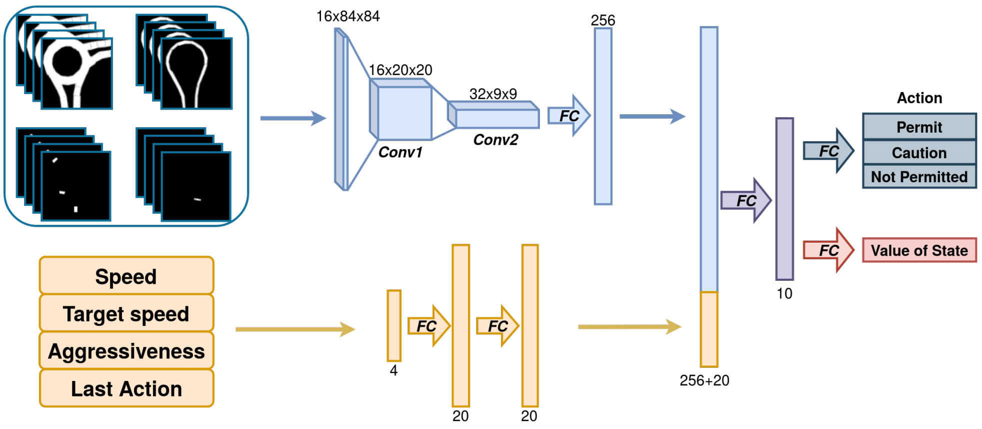


Figure 4.2: The neural network architecture used to train agents performing the immission maneuver task. It receives the last four frames of the four channels illustrated in Figure 4.1 and some scalar parameters, predicting the probabilities of three possible actions together with the state-value function estimation

- Not Permitted: the entry area is perceived as busy by the agent and the immission would be dangerous; the results will be a deceleration computed as $min(d_{max}, d_{stop\_line})$, where d_{max} is the maximum deceleration permitted following feasible constraints and d_{stop_line} is the decelaration applied to stop the *active* vehicle at the stop line. If the agent has already passed the line, the deceleration will assume the d_{max} value.

- Caution: the agent predicts the entry area as not completely free such that the behavior should be cautious, approaching the roundabout with prudence both to improve the agent view and to negotiate the entering with a *passive* car populating the roundabout. In such state the maximum speed permitted is $\frac{1}{2} targetspeed$ and the acceleration a can be:

$$a = \begin{cases} \frac{a_{max}}{2}, & \textbf{if } agent\ speed < \frac{target\ speed}{2} \\ \frac{d_{max}}{2}, & \textbf{if } agent\ speed > \frac{target\ speed}{2} + h \\ 0, & otherwise \end{cases} \tag{4.1}$$

whre h is set to 0.5.

4.2.2 Autonomous Driving Architecture

Given the structure of the simulator used to develop the proposed maneuver planning system, the architecture used on board of self-driving car will be similar to the one explained in Section 1.2.2, since we need a pipeline able to reproduce a topological map of the environment, to localize the car on such map and to detect the obstacles in the surroundings of the autonomous driving car.

In this way, in the particular case of the roundabout insertion, the proposed maneuver planning system replace the decision-making part of the *Planning* module (Section 1.2.2), and its output will be one of the input of the *Control* module (Section 1.2.2), resulting in an architecture similar to the one illustrated in Figure 4.3.

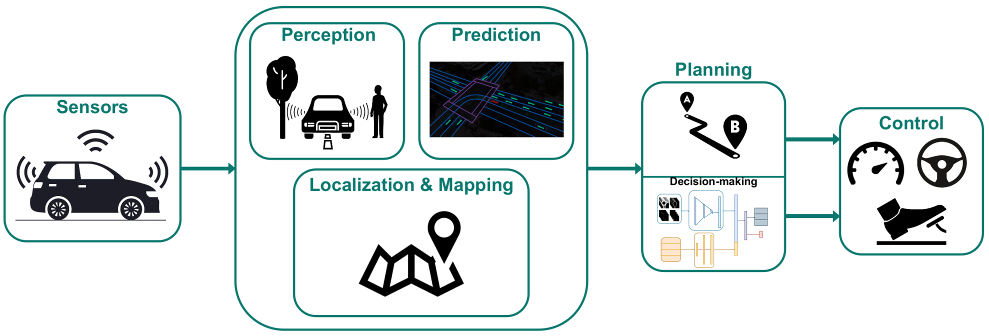


Figure 4.3: The autonomous driving architecture pipeline used for the specific task of the roundabout insertion. The decision making module output is represented by the neural network prediction that will be one of the input of the *Control* module

4.2.3 Delayed Asynchronous Advantage Actor-Critic (D-A3C)

We designed and developed a new version of the original *Asynchronous Advantage Actor-Critic* (A3C) (Section 2.3.8) called Delayed-A3C (D-A3C). In such configuration, each agent begins the episode with a local copy of the last version of the global network, while the system collects all the actors' contributions; the agent updates their local copy of the network at fixed time intervals but all the updates are sent to the global network only at the end of the episode, while in the classical A3C algorithm this exchange is performed at fixed time intervals. *Active* agents act in separate multiple instances of the scenario performing the maneuver form the three different entries simulataneously, but they could have a slightly different version of the same policy, thus increasing the exploration and at the same time reducing the synchronization burden, since the number of parameter exchanges diminishes compared to the classical A3C.

In Section 4.4.1, we demonstrate that D-A3C achieves better performance in the proposed tasks than both the typical A3C and A2C [71], that is the synchrounous variant of A3C.

4.3 Reward Function

The reward function designated for the proposed task consists of several factors, and it can be described by the following formula:

$$r_t = r_{danger} + r_{terminal} + r_{indecision} + r_{speed} \tag{4.2}$$

r_{danger} is a penalization related to dangerous behaviors of the *active* agent, and it is defined as:

$$r_{danger} = -w_{d_s} \cdot \alpha \cdot d_s - w_{c_f} \cdot \alpha \cdot c_f \tag{4.3}$$

where,

- d_s is a binary variable which is equal to 1 when the *active* vehicle violates the safety distance that is computed as the space traveled from the *active* agent in one second, and it is represented by the yellow region in Figure 4.1. When the safety distance is mantained the value of d_s is 0;

- c_f is a binary variable and it is set to 1 when the *active* agent passes in the front area of a *passive* vehicle; this area is equal to three times the distance traveled from the *passive* vehicle in one second and it is illustrated by the orange region in Figure 4.1. If the actor learner does not cut in front a *passive* car, the value of c_f is 0.

- α is related to the aggressiveness level of the *active* agent and it is defined as $\alpha = (1 - aggressiveness)$. For the whole episode the *aggressiveness* is kept fixed and chosen randomly between 0 and 1: higher values of *aggressiveness* should lead the agent to perform the maneuver with more impetus but consequently, dangerous actions will be less penalized.

- w_{d_s} and w_{c_f} are constants set to 0.002 and 0.005 respectively.

$r_{terminal}$ depends on the terminal state of the agent, namely how the episode ends, and it can assume the following values:

- +1: the *active* agent ends the episode safely without crashing with other *passive* vehicles;

- $-\beta - \gamma \cdot \alpha$: in this case a crash occurred between the *active* car and another traffic agent. β is a costant set to 0.2 and γ is the weight of α set to 1.8. Moreover, in this terminal state we modulate $r_{terminal}$ based on the *aggressiveness*, as explained for r_{danger}.

- -1: the time available to end the episode expires.

$r_{indecision}$ is a penalization given to the agent when performing frequent changes of conflicting actions in two consecutive time steps. In particular, we penalize the *active* agent when the action passes from *Permitted* to one of the others. Calling $L1$ and $L2$ the last and the second to last outputs respectively, $r_{indecision}$ can be designed in the following way:

$$r_{indecision} = \begin{cases} -0.05, & \textbf{if} \quad L2 = Permitted \\ & \quad\quad \text{and } L1 = Caution \\ -0.15, & \textbf{if} \quad L2 = Permitted \\ & \quad\quad \text{and } L1 = Not\ Permitted \\ 0, & \quad\quad otherwise. \end{cases} \tag{4.4}$$

Finally, r_{speed} is a positive reward which encourages the *active* agent to increase the speed reaching the target speed, and it is defined as:

$$r_{speed} = \psi \cdot \frac{current\ speed}{target\ speed} \tag{4.5}$$

where ψ is set to 0.0045.

As explained in Section 1.2.3, the reward shaping is a well-known problem in literature and even if the function designed for such task is the result of several trials, it is also true that it allowed the achievement of the desired comfort behavior, learning the agent to follow the basic road rules like the right of way and the safety distance.

4.4 Preliminary Results

4.4.1 Algorithms Comparison

We compared our implementation of D-A3C with the classical A3C and A2C, in order to prove if our approach improves the learning performances of the system for the proposed task. From Figure 4.4, we can notice how D-A3C reaches an optimal solution in less episodes compared to A3C, while A2C converges on a suboptimal solution, consisting on always outputting the *Permitted* state independently on the occupancy of the roundabout.

Some examples of immissions performed by *active* agents in the simulated roundabout are illustrated at the following video[1].

4.4.2 Aggressiveness Test

As explained in Section 4.2.1, the aggressiveness parameter aims at simulating different driving style behavior, as well as each human driver as its own, from the most cautious to the most impetuous. Using such parameter as one of the input of the neural network and shaping the reward based on such value, the system learns to modulate its behavior accordingly to the aggressiveness value; this is proved in Figure 4.5, in

[1]https://youtu.be/qVwRCad5K9c

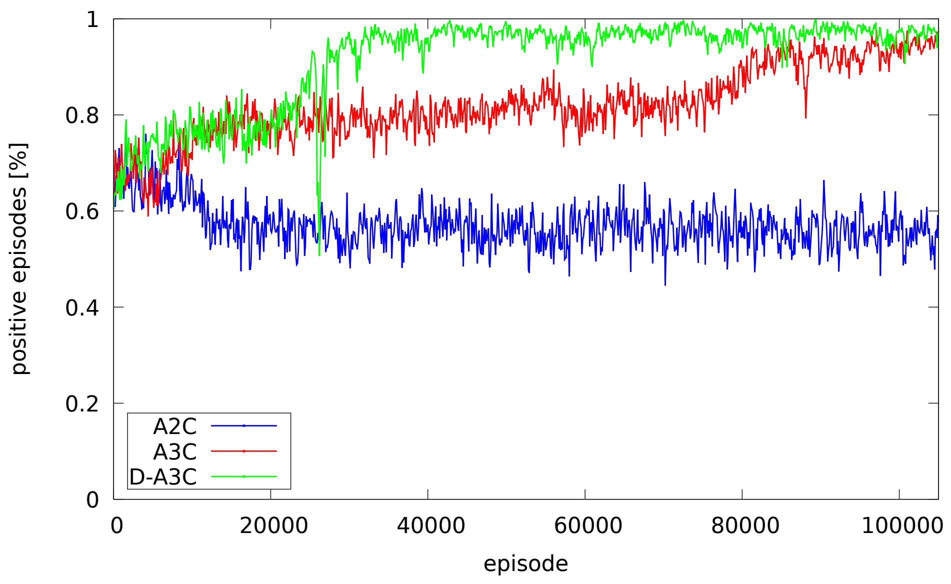


Figure 4.4: Learning curves representing the percentages of episodes ended successfully training the system with D-A3C (green), A3C (red) and A2C (blue)

which the D-A3C system is tested on a busy roundabout varying the aggressiveness level.

Each test is composed by 3000 episodes and we can notice that tests performed with higher values of aggressiveness increase the impetus of the *active* vehicle which tends to take more risks, rising the percentages of crashes with a consequent decrease of the positive episodes ratio, but also increasing the average speed values. Moreover, it is interesting to notice that the behavior of the system is consistent also for those aggressiveness values outside the range used during the training phase ($[0, 1]$).

4.4.3 Comparison with a Rule-based Method

We also demonstrate that our approach based on D-A3C achieves better performances than a classical rule-based approach consisting in setting four different tresholds (25, 20, 15 and 10 meters) corresponding to the minimum distances required between a

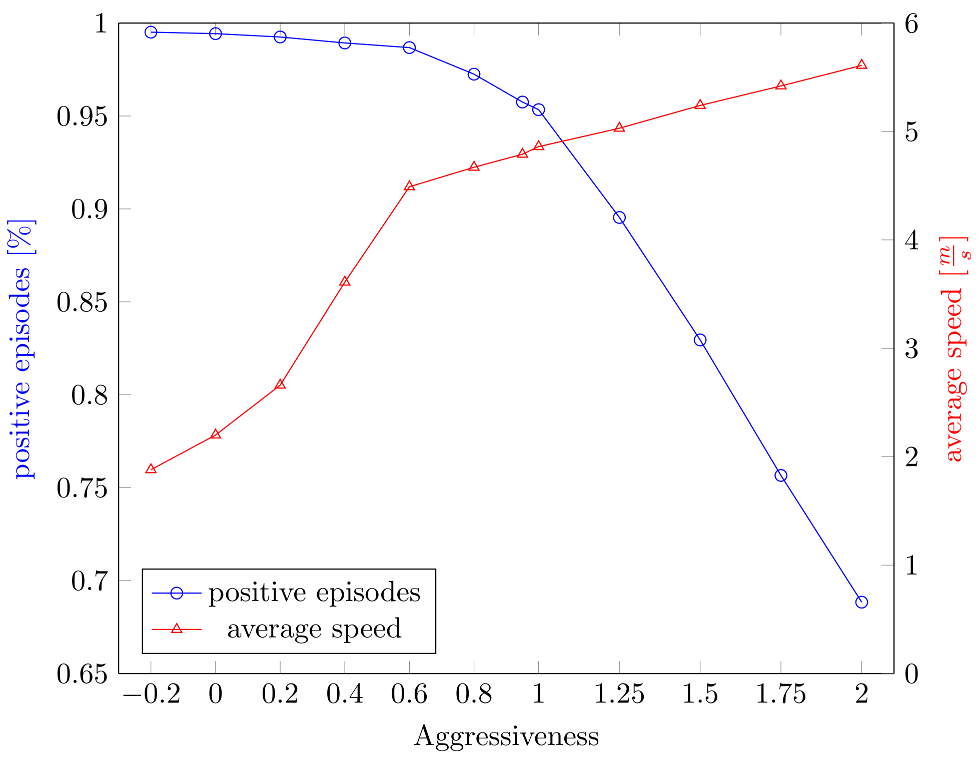


Figure 4.5: Average speed values (red curve) and percentages of episode ended successfully (blue curve) based on the aggressiveness level used for each test

passive vehicle and the *active* one for starting the entering maneuver.

The metrics used to evaluate the performances of the proposed approaches are: *Reaches*, *Crashes*, *Time-overs* corresponding to the percentage of episodes ended successfully, with a crash or when the available time has expired respectively. Every test is composed by three different experiments composed by 3000 episodes each, and using three different traffic conditions: low, medium and high, corresponding to a maximum number of *passive* vehicles populating the roundabout of 4, 6 and 8 respectively. The Table 4.1 shows the average percentages of the metrics used for this test, and we can notice that even the percentages of *Crashes* achieved by the rule-based method are rather low, such approach could lead to undefined waits due to its lack of negotiation and interaction capabilities that allows the vehicle to perform the immission maneuver only when the roundabout is completely free.

	Rule-based				D-A3C
	25m	20m	15m	10m	
Reaches %	0.456	0.732	0.831	0.783	0.989
Crashes %	0.0	0.002	0.012	0.100	0.011
Time-overs %	0.544	0.266	0.157	0.117	0.0

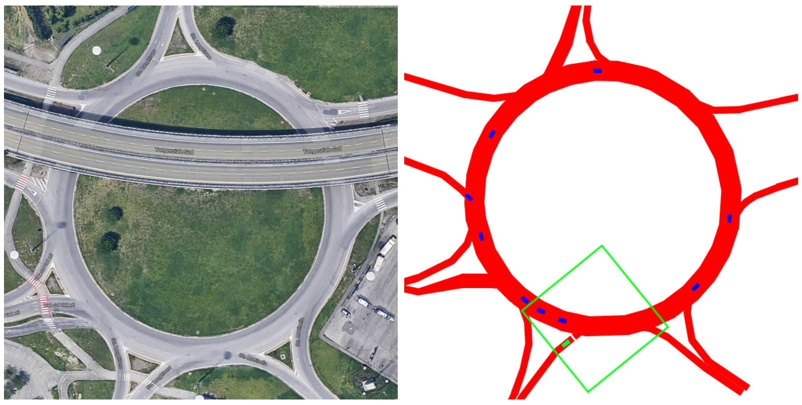
Table 4.1: Comparison between the results achieved by our D-A3C model and a classical rule-based algorithm.

4.4.4 Results on Unseen Scenarios

To evaluate the generalization capabilities on onseen scenarios, we tested the system on a new test roundabout (Figure 4.6), not seen during the training phase, comparing the results achieved by our D-A3C with two different baselines: the first one obtained using a policy that always outputs the *Permitted* state independently of road occupancy, while the second one using random actions. Given the larger area of such roundabout, we consider as low, medium and high traffic conditions a maximum number of 10, 15, and 20 *passive* cars respectively, populating the roundabout simultaneously. From Table 4.2, we can notice that the system features some generalization capabilities, but a crash percentage of 8.5% makes the module unsuitable for testing it in real scenarios where a great generalization capability is required given the uncertainty and the novel situations encountered in real-world environment [72].

	D-A3C	**Random**	**Permitted**
Reaches %	0.910	0.684	0.676
Crashes %	0.085	0.270	0.324
Time-overs %	0.003	0.046	0.0

Table 4.2: Results on the unknown roundabout of Figure 4.6.



(a) Real test roundbout (b) Synthetic test roundabout

Figure 4.6: Real roundabout of Parma (4.6a) and its synthetic representation (4.6b) unseen during the training phase and used to evaluate the generalization capability of D-A3C system

4.5 Generalization Techniques

An important problem in Deep Reinforcement Learning algorithms is related to the generalization capabilities of such systems when tested in unknown environments; moreover, also the deployment of a DRL algorithm in a real-world context requires a high ability to handle new and unseen situations given the uncertain nature of the elements populated real scenarios.

Many approaches create different training levels of increasing difficulty showing that in this way they reduce the risk of overfitting ([73], [74]); however, they also show that agents have a high capacity of memorizing levels of the training set but the performances on test scenarios depend on the complexity of levels or the training set size. Moreover, especially if we are interested in deploying the system in real-world environments as in robotics field, techniques aim at reducing the gap between simulated and real world should be considered, since the states observed by an agent in simulation could be very different from those perceived in real scenarios.

For this reason, we try to increase the generalization capability of our system combining two main ideas:

- The design of a *Multi-environment System* using a pipeline similar to the one adopted in the typical supervised learning approach consisting in training, validation and test scenarios.

- The injection of different sources of noise in the simulator trying to reduce the difference between the states observed during the training phase and on board of real self-driving vehicle.

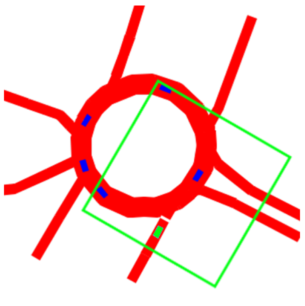
We prove that in this way we achieve impressive results in unseen scenarios, making the system more robust to uncertainties introduced by real-world environments and thus testing such system on a real roundabout on board of an autonomous vehicle [75].

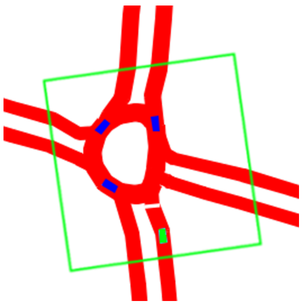
4.5.1 Multi-environment System

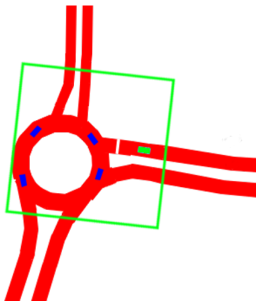
The first improvement introduced regards the implementation of a *Multi-environment System* inspired by the typical supervised learning pipeline consisting of splitting the scenarios in training, validation and test set. All the environments are a synthetic representation of real roundabouts in Parma, and they are composed by four training roundabouts (Figure 4.7), one used as validation (Figure 4.8) during the training phase to evaluate the performance of the policy, and finally a test scenario (Figure 4.6) that is the same used in Section 4.4.4.

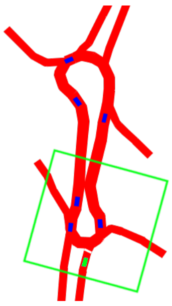
During the training phase, we evaluate the performance of the policy on the validation roundabout (Figure 4.8a) in order to save the best model based on the results obtained on this scenario and thus reducing the risk of overfitting on the training environments; the *active* vehicles acting in the validation roundabout do not compute or send back the gradient, but they perform the maneuver with the latest network weights pulled at the beginning of each episode. A simple scheme of the learning process is illustrated in Figure 4.9.

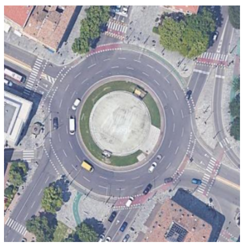
Given the diversity of the scenarios in terms of shape and length, we also need traffic vehicles able to behave as well as possible in such roundabouts. For this reason,


(a) Training roundabout 1

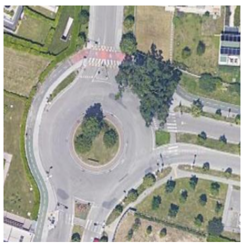

(b) Training roundabout 2


(c) Training roundabout 3


(d) Training roundabout 4


(e) Real roundbout 1

(f) Real roundbout 2


(g) Real roundbout 3

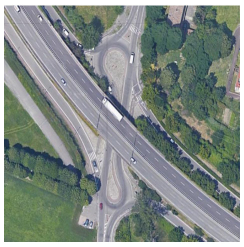
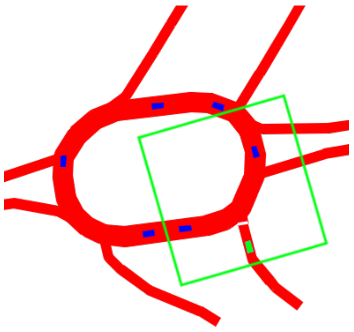

(h) Real roundbout 4

Figure 4.7: Synthetic representation of real roundabouts used for training scenarios in the *Multi-environment System* setting

we also retrained the *passive* agents in all the environments (training, validation and test roundabouts) following the same settings used in [65]. We did not apply the generalization techniques proposed for training the *active* agents since we are interested in obtaining traffic agents that behave as well as possible in all the environments.


(a) Validation roundabout

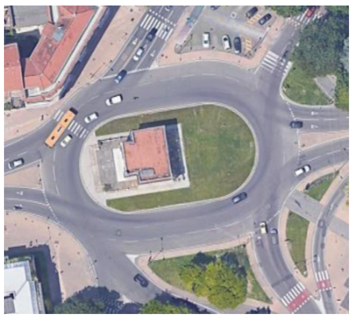

(b) Real roundabout

Figure 4.8: Real roundabout of Parma (4.8b) and its synthetic representation (4.8a) used as validation scenarios

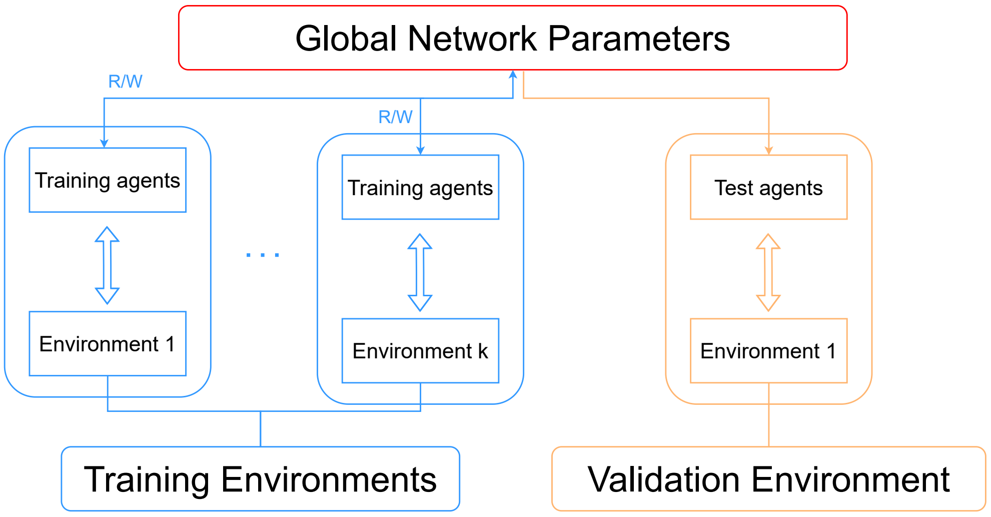


Figure 4.9: Training pipeline used for the *Multi-environment System* setting. *Active* agents performing the immission on the training scenarios update the global network following the D-A3C algorithm (Section 4.2.3), while those ones acting on the validation roundabout do not compute the gradient and they are not involved in the update of global network parameters

However, it is clear that the same generalization techniques can be also applied to the *passive* vehicles.

The maximum numbers of *passive* vehicles populating each instance of the environments are illustrated in Table 4.3 and can vary depending on the shape and the length of the roundabout, allowing the *active* agent to observe and handle different traffic conditions. For each environment we create as many instances as the number of entry lanes such that in every instance only one *active* agent performs the immission from a specific entry, as explained in Section 4.2. Indeed, the number of *active* cars involved in the five roundabouts (four training environments and one for validation) during the training phase is 19, interacting with 119 *passive* vehicles, namely the sum of the maximum number of traffic cars used for every single instance of each environment (Table 4.3) multiplied for the number of entry lanes of each roundabout.

Roundabout Instance	**#Passives**
Training roundabout 1	6
Training roundabout 2	3
Training roundabout 3	6
Training roundabout 4	6
Validation roundabout	9

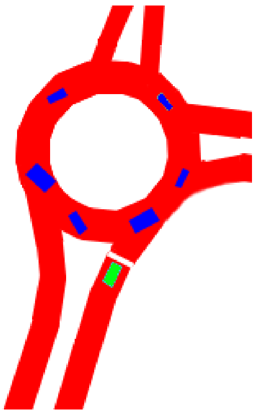
Table 4.3: Maximum number of traffic cars allowed in each instance of training (Fig. 4.7) and validation (Fig. 4.8) environments.

4.5.2 Noise Injection

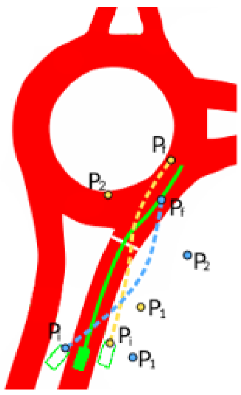
The second improvement adopted aim at reducing the gap between simulated and real-world observations is the injection of noise during the training phase. Indeed, the data distributions observed in simulation can be very different from those perceived in real scenarios, both for the stochastic nature of real world environments and for the errors that perception or localization systems on board of the autonomous driving car may introduce. In order to reproduce such noise and thus training our agents to handle the uncertainties of real world elements, we introduce different artificial elements in the simulation:

- Perception noise: this noise aims at simulating possible errors introduced by perception systems on board of self-driving car, or simply to reproduce imperfect human driving behaviors. Indeed, we inject noise in the position (x, y coordinates), size (width and height) and heading of *passive* vehicles perceived from the *active* one as illustrated in Figure 4.10a. Finally, we also introduce detection errors of *passive* cars: each time step a traffic car has a probability of not being detected by the *active* one, such that the four sequential obstacle channel images (Figure 4.1d) given as input to the neural network may contain some detection errors.

- Localization noise: this element has been introduced both to simulate localization errors and to train the *active* agent to observe the state not always from the center lane, but also being sligthly away from a perfect position. At this purpose, we generate a new virtual path starting from the original one and using Cubic Bézier curves [76] as illustrated in Figure 4.10b.



(a) Perception noise



(b) Localization noise

Figure 4.10: Examples of perception (4.10a) and localization (4.10b) noise aim at simulating errors that systems on-board vehicle may introduce

Finally, we introduce an additional noise element by simply modifying the shape of the entry lanes every 1000 episodes, in order to train the *active* agent to perform the immission observing different navigable spaces.

4.6 Generalization Results

We test the system in the unknown scenario of Figure 4.6b in order to evaluate the generalization capabilities of the system. As for the previous tests, we performed three experiments, each one composed by 3000 episodes with different traffic conditions, low, medium and high, corresponding to a maximum number of *passive* vehicles populating the test roundabout to 10, 15 and 20 respectively. The metrics used in this test is the same used in Section 4.4.4, except for the *Time-overs* metric that has been replaced by *Total Steps*, namely the average number of steps required to end the episode; indeed, for such test we let the agent performing the insertion without time limit. For this reason, we also performed a new test with the model obtained training the system on the single roundabout of Figure 4.1, which we will call *Single_env*, using the new metrics and using the updated version of traffic vehicles.

Moreover, in order to prove the improvements introduced by the generalization techniques exaplained in Section 4.5, we analyzed the generalization performances on the test roundabout using several models obtained with different training configurations explained as follows:

- **Single_env**: such model is obtained training the system on the single roundabout of Figure 4.1, without using any generalization techniques (Section 4.5).
- **Five_envs**: the agents are trained both on the four training roundabouts (Figure 4.7) and on the one of Figure 4.8, without using the latter as validation scenario and without introducing noise (Section 4.5.2) in the simulation.
- **Five_envs+noise**: this module is trained in the same way as *Five_envs*, but with the injection of noise (Section 4.5.2) during the training phase.

- **Multi_env**: the system is trained following the *Multi-environment System* setting (Section 4.5.1) thus using the roundabout of Figure 4.8 as a validation scenario, but without introducing noise (Section 4.5.2) in the simulation.

- **Multi_env+noise**: such module is obtained using both the generalization techniques exaplained in Section 4.5, thus using the roundabout of Figure 4.8 as validation scenario and the noise injection during the training.

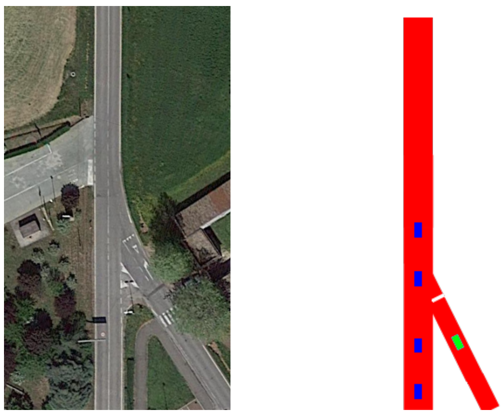
From the results illustrated in Table 4.4 we can state that the simple addition of training environments as performed in *Five_envs* does not guarantee better generalization performances; on the contrary, the percentage of episode ended successfully by *Single_env* overcome the one obtained by *Five_envs* even if it reaches excellent results in the training scenarios ($> 98\%$ of *Reaches*). Finally, we can observe that best results are achieved by such model trained using the *Multi-environment System* setting with the validation scenario and the noise injection; indeed, even if the introduction of unpredictable elements were designed to reduce the gap between simulated and real data, they also allow the system to achieve better performances on the test scenario.

Further demonstrations on the capabilities achieved by *Multi_env+noise* are rep-

	Reaches %	**Crashes %**	**Total Steps**
Single_env	0.907	0.093	103.489
Five_envs	0.891	0.109	100.460
Five_envs+noise	0.979	0.021	116.356
Multi_env	0.952	0.048	108.237
Multi_env+noise	0.991	0.009	137.438

Table 4.4: Results obtained on the test scenario (Fig. 4.6b) achieved by the different modules on the test roundabout (Figure 4.6b). The values reported in the table are computed as the average of the three experiments with different traffic conditions (low, medium, high).

resented by tests performed on the validation scenario and on a completely different type of environment that is the junction scenario illustrated in Figure 4.11. We compare the performances achieved by *Multi_env+noise* and *Single_env* to show how the generalization techniques (Section4.5) introduces in the model dramatically improve the capabilities of the system.



(a) Real junction (b) Synthetic junction

Figure 4.11: Junction scenario used as additional test to compare the performances obtained by *Multi_env+noise* and *Single_env* models.

In both tests, we conducted the experiments using the same setup and metrics of the previous ones, namely performing three tests with different traffic conditions, each one composed by 3000 episodes. In such cases, low, medium and high traffic conditions correspond to a maximum number of *passive* vehicles populating the scenarios simultaneously of 6, 12 and 18 for the validation roundabout (Figure 4.8a) and 2, 4 and 6 for the highway in the junction scenario (Figure 4.11). Moreover, to better evaluate the maneuver in the junction scenario, we reshape its entry lane at the beginning of each episode such that the junction angle could assume different values during tests. Table 4.5 and Table 4.6 illustrate the performance comparison between *Multi_env+noise* and *Single_env* obtained on the validation roundabout and on the junction scenario respectively. We can notice how the generalization techniques explained in Section 4.5 dramatically improve the capabilities of the system even in those scenarios completely different from roundabout environments.

	Single_env	**Multi_env+noise**
Reaches %	0.920	0.994
Crashes %	0.080	0.006
Total Steps	88.906	115.741

Table 4.5: Performances achieved by *Single_env* and *Multi_env+noise* on the validation roundabout (Fig. 4.8a).

	Single_env	**Multi_env+noise**
Reaches %	0.919	0.970
Crashes %	0.081	0.030
Total Steps	97.011	128.182

Table 4.6: Comparison between *Single_env* and *Multi_env+noise* models on the junction scenario (Fig 4.11).

4.7 Real-World Test

The final experiment was conducted deploying *Multi_env+noise* and *Single_env* models on board of the real self-driving car illustrated in Figure 4.12, testing them in the real roundabout of Figure 4.7g. We chose this scenario for two main reasons: the first one is related to the fact that both the tested models were trained on such roundabout in simulation, and secondly because at the time of such tests this roundabout was included in one of the few areas in which autonomous driving tests were allowed in Parma.

The autonomous driving pipeline developed to perform the experiments is the one shown in Figure 4.3: in particular, combining a topological map with localization and

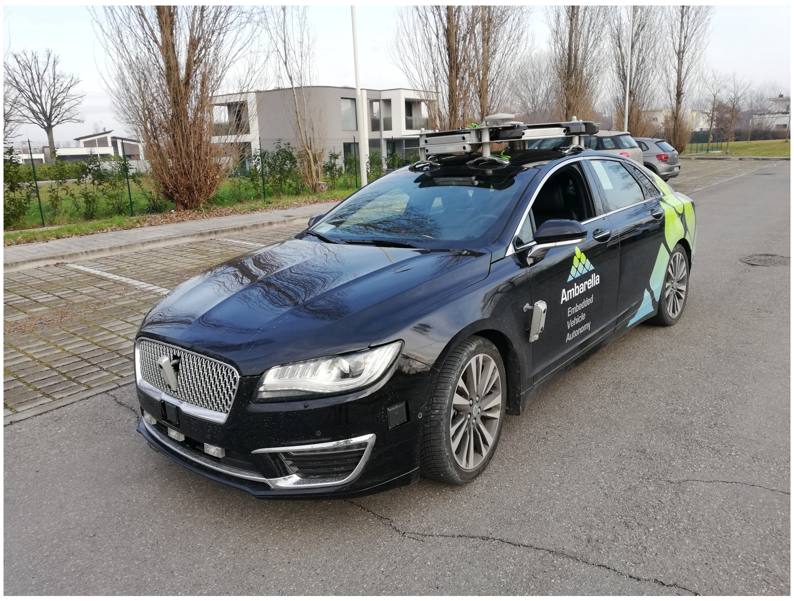


Figure 4.12: The real autonomous driving car used to perform tests on the real roundabout of Figure 4.7g.

detection systems embedded on the self-driving car, we extract the visual channels illustrated in Figure 4.6.

The comparison between the two tested models on the real roundabout allows us to understand the real impact of the generalization techniques (Section 4.5) of the system. Indeed, even if *Single_env* model was trained only on the synthetic representation of such scenario, it was infeasible to perform the immission using such model since its behavior was unsure, with continuous changes of output due to uncertainty and noise introduced by localization and perception systems on board of the self-driving vehicle. On the contrary, *Multi_env+noise* has proved to be more robust to these uncertain elements, behaving safely and performing more than 100 immissions in the real roundabout from the three different entry lanes. The following video[2] shows some immission maneuvers performed by the self-driving vehicle equipped

[2]https://youtu.be/QmgB0YH2BdQhttps://youtu.be/QmgB0YH2BdQ

with the *Multi_env+noise* model.

4.8 Conclusions

The proposed approach proved the feasibility of Reinforcement Learning techniques deployment in real-world scenarios, in which the whole training phase was totally performed with simulated data using a non-graphic simulator (Figure 4.1). The results obtained represent a promising starting point for the development of maneuver execution systems through the use of Reinforcement Learning techniques and their corresponding deployment on board of real self-driving vehicles; moreover, such approach also proved to achieve excellent generalization capabilities both in unseen and real-world scenarios without the need of using real data during the training phase.

However, one of the main limitation of the system proposed in this chapter lies in the use of discrete actions; indeed, even if the aggressiveness input parameter modulates the behavior of the agent during the whole episode making the agent more cautious or more impetuous, it is also true that the output of the neural network architecture (Figure 4.2) makes impossible to modulate the impetus of the acceleration during the immission maneuver, thus using the same acceleration (or deceleration) independently on the traffic conditions. At this purpose, as a natural extension of such system, we developed a new system explained in the next chapter, based on continuous actions and trained on intersection scenarios.

Chapter 5

Continuous Control Actions Handling Intersection Scenarios

In the previous chapter we explored the capability of a Deep Reinforcement Learning algorithm in solving the specific task of roundabout insertion, training a neural network which predicts discrete actions to control the longitudinal behavior of the sel-driving car. As a natural evolution of such approach, we try to implement a more complex model which directly outputs both longitudinal and lateral continuous control values and able to handle intersections in which only traffic signs are provided [77]. In this way, we overcome the limitation analyzed in the previous chapter for which discrete actions do not allow the system to modulate its behavior depending on the observed state.

We develop a multi-agent system, using a continuous, model-free Deep Reinforcement Learning algorithm training a neural network through the multi-agent version of D-A3C (Section 4.2.3) for predicting both the acceleration and the steering angle values in order to safely navigate in intersection scenarios. In particular, we create three different training environments (Figure 5.1) corresponding to three different difficulty levels (easy, medium and hard), using CAIRO graphic library [66] as for the roundabout scenarios, training the agents to follow the basic rules needed to handle intersections; moreover, we no longer use the distinction between *active*

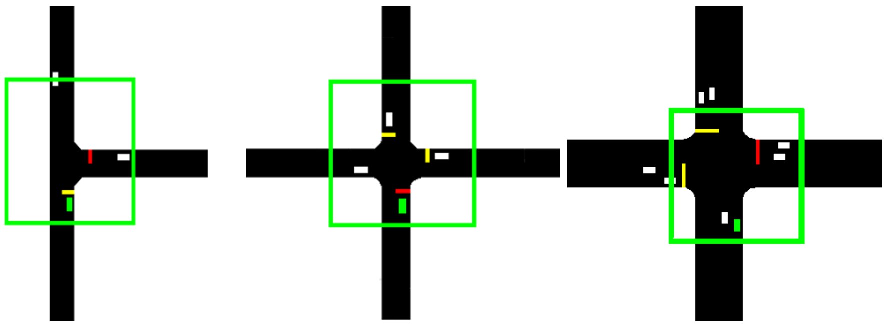

(a) Easy intersection (b) Medium intersection (c) Hard intersection

Figure 5.1: Synthetic representations of three intersections representing three different difficulty levels: easy (5.1a), medium (5.1b) and hard (5.1c). As in the roundabout case, each agent in the scenario observes an area of 50×50 meters: green squares show some examples of these surroundings perceived by green vehicles

and *passive* agents as in the roundabout scenarios, since in this case all the agents inside the environments are involved in the training phase updating the global network following the D-A3C algorithm.

We also demonstrate that agents learn both to understand the priorities of other vehicles involved in the environment and to drive safely in order to reach safely the common goal, namely crossing the intersection without causing accidents. Moreover, we prove that our approach achieves better performance compared to a classical rule-based method especially with dense traffic conditions and finally, we test our module using real recorded traffic data on different scenarios, proving that the proposed system is able to generalize both to unseen environments and to different traffic conditions.

5.1 Intersection Handling

We focus on intersections regulated only by traffic signs since those ones ruled by traffic lights could be solved by simple rule-based methods in which the autonomous vehicle behavior is closely related to the traffic light states; for those cases, interesting solutions consist in mitigating traffic congestions through reservation-based systems [78] or intelligent traffic lights ([79], [80], [81]). However, our work does not focus on solving the problem related to the traffic congestion, but rather to the direct control of the autonomous vehicle.

At this purpose, a typical rule-based method widely used to handle intersection scenario is base on the time-to-collision (TTC) algorithm [82] that could be useful for simple cases but it has several limitations; for example, it assumes costant speed of traffic vehicles, it does not understand the dynamic of the scenario and the possible intentions of other agents, leading consequently to unnecessary delays. Instead, in [83] authors propose a multi-agent approach in which road rules emerge as optimal solution to the traffic flow problem; in [84] and [85] a single agent is trained to handle unsignalized intersections using Deep Q-Networks (DQNs) [86]. However, in such works a single agent is trained predicting discrete actions facing environments populated by traffic vehicles which follow the Intelligent Driving Model (IDM) algorithm [87] to control their speed. In this way, the trained agent cannot learn to interact and negotiate the immission since the traffic vehicles do not have such ability, which instead is an innate behavior of human drivers. In this way, the trained agent will observe traffic behaviors different from those ones adopted by human drivers, thus increasing the gap between simulated and real data making difficult the deployment of such system in real-world environments.

Instead, our system is based on a multi-agent training such that vehicles implicitly learn to interact and negotiate among them, as the actions of an agent affect the state of others and viceversa as explained in Section 2.3.9. Moreover, in [84] and [85] it is assumed that the trained agent has always the lowest priority compared to the other traffic cars, such that it should only choose a discrete action in the right time step to perform the maneuver correctly; for this reason, such system is not suitable to handle more complex intersections where vehicles should understand the priorities of other cars in order to perform the immission efficiently both in terms of time and safety. Indeed, in our approach the agents learn to understand the priorities of all the vehicles involved in the environment, since they are trained to follow the priority to the right rule, that is a right of way system typically used in countries with right-hand traffic:

in this way, when two vehicles are approaching the intersection and their trajectories may intersect, the car coming from the right has the priority in the case both agents have the same traffic sign or when no sign is present; otherwise, the priority is defined by traffic signs.

5.2 Environment

The environments in which agents are trained are illustrated in Fig. 5.1, in which the easy and medium levels correspond to two different single-lane crosses, while the hard scenario is a double-lane intersection. Since agents control both longitudinal and lateral behaviors, they learn to drive inside their paths (that is computed at the beginning of each episode), such that they may end the episode in one of the following terminal states:

- Reaches: the agent crosses the intersection safely and driving without going out of its path reaching the goal position.
- Crashes: the vehicle crashes with another vehicle during the crossing maneuver.
- Off-roads: the agent goes out of its path.
- Time-overs: the time available to finish the episode expires.

Moreover, even if the hard scenario is a double-lane intersections, we do not allow lane changes, meaning that if the agent goes out of its path we consider the episode ended as a failure. In addition, only one agent can start from each entry lane, such that the maximum number of vehicles involved simultaneously in the environment is equal to the number of entry lanes of the scenarios: 3 cars for the easy intersection, 4 for the medium and 8 for the hard one. In order to let agent perceiving different state configurations, a new path and a new traffic sign are randomly computed at the beginning of each episode; however, since the hard scenario is a double-lane intersection, we assume that both lanes of the same branch have the same traffic sign and for this reason we randomly change the traffic signs at fixed time intervals: for this reason, we restart the episodes of all the learners in such scenario simultaneously, avoiding changing the traffic signs configuration during the execution of the crossing maneuver.

5.2.1 Input Space

As for the roundabout case, each agent involved in the environment perceives a 50×50 meters surrounding which is split in four different channels of 84×84 pixels representing navigable space, path, obstacles and traffic sign as illustrated in Figure 5.2, that will be one of the input of the neural network. Given the narrow view of the agent, it is almost impossible handling larger intersections; for this reason future works may be directed towards the use of a more powerful encoder like Variational Autoencoder (VAE) [88] in order to achieve a satisfactory compression of larger areas.

The choice of using grayscale images allows us to embed more information in the traffic sign and obstacle channels. Indeed, in order to train agents to understand the different types of traffic signs it may encounter at intersections, these could assume different values representing a:

- Stop sign: it will be drawn as a red segment in the top-view of the simulator and as a black segment in the traffic sign view (Figure 5.2e);
- Yield sign: it is drawn as a yellow segment in the simulator and as a white segments in the traffic sign channel (Figure 5.2e);
- None: no traffic sign is provided as if such lane was the main road; it means that the agent has the priority on vehicles with the stop or yield signs; however, it must always give the priority to those cars without traffic signs coming from its right (as expected by the priority to the right rule).

Obstacle channel provides information about both the positions of the vehicles in the surrounding view of the ego-agent and their priority levels. Indeed, the ego-vehicle could perceive the other cars in two different ways in the obstacle channel:

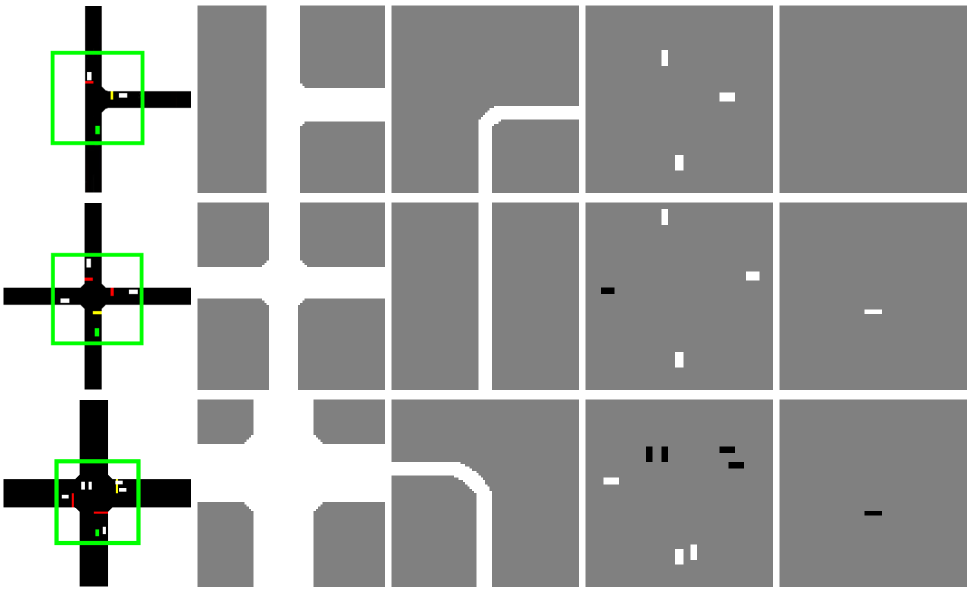


Figure 5.2: Examples of views perceived by learners in the three different intersections. The first column (5.2a) shows the top-views of the simulator, while (5.2b), (5.2c), (5.2d) and (5.2e) represent the information contained in the green squares, namely navigable space (map), path, obstacles including the ego-vehicle (green cars in 5.2a) and the traffic sign of the green agents respectively. The traffic sign could be black or white representing the stop line and yield line respectively; however, if it is not present it means that no traffic sign is present, as if that lane belonged to the main road. Also the obstacles could be black, in case of those cars with higher priority than the ego-vehicle, or white representing both the ego-agent and those ones which should give the right of way to the ego-car

- White agents: they represent both the ego-agent and those vehicles on which the ego-car has the right of way.
- Black agents: they are the vehicles with higher priority than the ego-car.

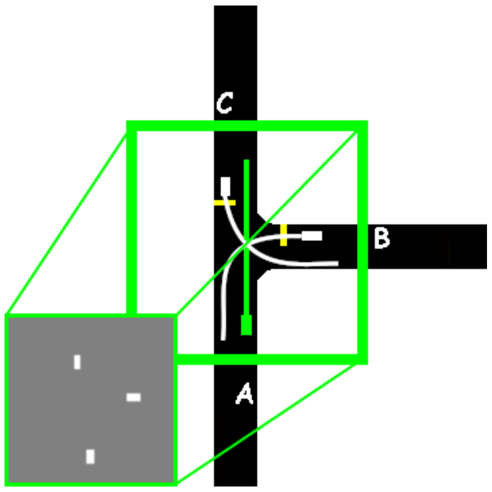
This prior knowledge about vehicle priorities is computed depending on the traffic signs and the priority to the right rule, and it is embedded in the obstacle channel (Figure 5.2d), such that agents can learn this basic rule in order to handle the crossing maneuver, negotiating the immission with other vehicles with different priority levels. To clarify how the priority to the right rule works, Figure 5.3 illustrates some simple examples with the corresponding obstacle channel views perceived by the green agents. In the first example (Figure 5.3a), the vehicle A has the highest priority since both B and C have the yield sign; for this reason, agent A will perceive the other two vehicles as white obstacles in the dedicated channel. Then, even if agents C and B have the same traffic signs, C should pass before B as established by the priority to the right rule[1]. In the second example (Figure 5.3b), agent C has the lowest priority since it is the only one with the stop sign. Then, car A should wait that B has passed before crossing the intersection, as defined by the priority to the right rule.

Moreover, such prior knowledge contained in the traffic sign and obstacle channels can be easily retrieved on board of a self-driving car using perception systems and high-deifinition maps, thus obtaining information related to the road topology and vehicle priorities, the latter computed based on traffic signs and the right of way rule.

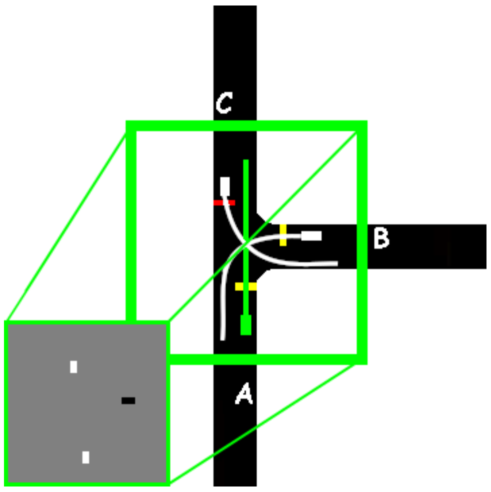
5.3 Training Setup

As for the roundabout case, agents follow the kinematic bicycle model using values in the interval $[-0.2, 0.2]$ for the steering angle and $[-3.0\frac{m}{s^2}, +3.0\frac{m}{s^2}]$ for the acceleration. At the beginning of the episode, each agent starts driving in its path with different speed, randomly chosen inside the range $[3.0\frac{m}{s}, 6.0\frac{m}{s}]$ and different target speed ($[5.0\frac{m}{s}, 8.0\frac{m}{s}]$) that is the maximum speed the agent should not exceed. Since we do not use traffic vehicles, but instead the traffic flow is represented by the actor-learners, each agent waits a random delay ($[delay_{min}, delay_{max}]$) before starting a new episode, thus ensuring to use different traffic density during the training phase. Given the

[1]When two vehicles are approaching the intersection and their trajectories may intersect, the car coming from the right has the priority in the case both agents have the same traffic sign or when no sign is present; otherwise, the priority is defined by traffic signs.



(a) Example 1



(b) Example 2

Figure 5.3: Three examples illustrating how the priorities of the vehicles are embedded in the obstacle channel of each agent. The surrounding views proposed in these examples are those one perceived by the green agents, but the same considerations can be done also for all the other cars involved in the simulation

difference in terms of space and lengths between the three intersections (Figure 5.1), this delay range can assume different values depending on the the intersection: $[0,30]$, $[0,50]$ and $[0,100]$ seconds for the easy (Figure 5.1a), medium (Figure 5.1b) and hard (Figure 5.1c) scenario respectively.

5.3.1 Neural Network Architecture

The goal of the agents is to handle intersection scenarios learning the priority to the right rule in order to perform the maneuver efficiently in terms of time and safety; they perform such maneuver learning to drive along their paths through the training of a neural network that control the longitudinal and lateral behaviors of the agent, predicting both the acceleration and the steering angle every 100 milliseconds.

The neural network architecture is illustrated in Figure 5.4 and it is composed by

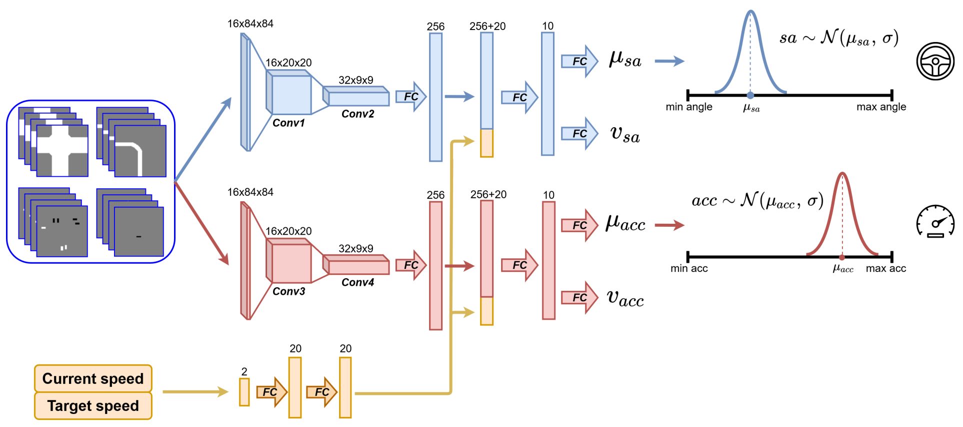


Figure 5.4: The neural network architecture used to train agents to cross the intersection and navigate safely along their paths. The net is composed by two different sub-modules receiving the same four visual input, each one composed by the last four sequential frame images 84×84 pixels in order to give the agent a history of the past states in order to understand how the environment is changing. The neural network produces two different outputs corresponding to the means (μ_{acc}, μ_{sa}) of two different Gaussian distribution from which the acceleration and the steering angle performed by the agent are sampled using a standard deviation σ that decrases linearly from 0.65 to 0.05 during the training phase

two main sub-modules, the first one to handle the acceleration (*acc*) and the second one the steering angle (*sa*) output. The two sub-modules receive the same visual input, consisting in four sequential frames for each channel (navigable space, path, obstacles and traffic sign) of size 84×84 pixels. Together with this visual input, the network receives two scalar parameters representing the speed of the agent and the target speed. In order to guarantee exploration, the actions performed by the agent are sampled by two Gaussian distributions centerd on the outputs of the two sub-modules (μ_{acc}, μ_{sa}), such that the acceleration and the steering angle are computed as:

$$acc = \mathcal{N}(\mu_{acc}, \sigma) \tag{5.1}$$

$$sa = \mathcal{N}(\mu_{sa}, \sigma) \tag{5.2}$$

where σ is a tunable parameter that decreases linearly from 0.65 to 0.05 during the training phase. Finally, along with μ_{acc} and μ_{sa}, the network produces the corresponding state-value estimations (v_{acc} and v_{sa}) using two different reward functions $R_{acc,t}$ and $R_{sa,t}$ related to the acceleration and steering angle respectively. In this way, the state-value estimations can be defined as $v_{acc}(s_t;\theta^{v_{acc}}) = \mathbb{E}(R_{acc,t}|s_t)$ and $v_{sa}(s_t;\theta^{v_{sa}}) = \mathbb{E}(R_{sa,t}|s_t)$ and the update formula for the actor parameters (Equation 2.19) results as follows:

$$\begin{aligned}\theta_{t+1}^{\mu_{acc}} = \theta_t^{\mu_{acc}} + \alpha \cdot A_{acc,t}\frac{\nabla\pi(a_t|s_t;\theta_t^{\mu_{acc}})}{\pi(a_t|s_t;\theta_t^{\mu_{acc}})} &= \\ \theta_t^{\mu_{acc}} + \alpha \cdot A_{acc,t}\frac{\nabla\mathcal{N}(acc|\mu_{acc}(\theta_t^{\mu_{acc}}))}{\mathcal{N}(acc|\mu_{acc}(\theta_t^{\mu_{acc}}))} &= \\ \theta_t^{\mu_{acc}} + \alpha \cdot A_{acc,t}\frac{acc-\mu_{acc}}{\sigma^2}\nabla\mu_{acc}(\theta_t^{\mu_{acc}})&\end{aligned} \tag{5.3}$$

$$\begin{aligned}\theta_{t+1}^{\mu_{sa}} = \theta_t^{\mu_{sa}} + \alpha \cdot A_{sa,t}\frac{\nabla\pi(a_t|s_t;\theta_t^{\mu_{sa}})}{\pi(a_t|s_t;\theta_t^{\mu_{sa}})} &= \\ \theta_t^{\mu_{sa}} + \alpha \cdot A_{sa,t}\frac{\nabla\mathcal{N}(sa|\mu_{sa}(\theta_t^{\mu_{sa}}))}{\mathcal{N}(sa|\mu_{sa}(\theta_t^{\mu_{sa}}))} &= \\ \theta_t^{\mu_{sa}} + \alpha \cdot A_{sa,t}\frac{sa-\mu_{sa}}{\sigma^2}\nabla\mu_{sa}(\theta_t^{\mu_{sa}})&\end{aligned} \tag{5.4}$$

where the two *Advantages* $A_{acc,t}$ and $A_{sa,t}$ are defined as:

$$A_{acc,t} = R_{acc,t} + v_{acc}(s_{t+1};\theta^{v_{acc}}) - v_{acc}(s_t;\theta^{v_{acc}}) \tag{5.5}$$

$$A_{sa,t} = R_{sa,t} + v_{sa}(s_{t+1};\theta^{v_{sa}}) - v_{sa}(s_t;\theta^{v_{sa}}) \tag{5.6}$$

5.4 Reward

In the previous section, we defined two different state value estimations depending on two different reward signals to evaluate the acceleration ($R_{acc,t}$) and the steering angle ($R_{sa,t}$) in order to obtain a more precise estimation of the performed actions. Since we do not allow lane changes as explained in Section 5.2, we assume that a crash between two vehicles just depends on the acceleration, while the off-road case only on the steering angle output.

In this way, $R_{acc,t}$ and $R_{sa,t}$ functions could be defined as follows:

$$R_{acc,t} = r_{speed} + r_{terminal} \tag{5.7}$$

$$R_{sa,t} = r_{localization} + r_{terminal} \tag{5.8}$$

r_{speed} is a positive reward given to $R_{acc,t}$ in order to encourage the agent to reach the target speed and it is defined as:

$$r_{speed} = \xi \cdot \frac{current\ speed}{target\ speed} \tag{5.9}$$

where ξ is a constant set to 0.005.

$r_{terminal}$ is assigned both to $R_{acc,t}$ and $R_{sa,t}$ and it depends on the terminal state achieved by the agent (Section 5.2):

- Goal reached: the agent achieves the goal position driving and crossing the intersection safely, without going outside its path or without crashing with another vehicle. In this case, the value assigned to $r_{terminal}$ is $+1.0$ both for $R_{acc,t}$ and $R_{sa,t}$ signals.

- Crash: an accident between two vehicles has occurred. Since the lane change is not allowed, we assume that this is only cause by an inaccurate estimation of the acceleration output and for this reason $r_{terminal}$ will be 0 for $R_{sa,y}$. Instead, for $R_{acc,t}$ signal, we modulate $r_{terminal}$ value based on the coding explained in Section 5.2.1, by which an agent should give the right of way to those ones represented as black cars in the obstacle channel, but it has the priority on white vehicles (Figure 5.2d). By using such information, we are able to assign the penalization depending on the priorities of the vehicles involved in the accident, setting the value of $r_{terminal}$ to -1.0 for the agent with the lowest priority, namely if it crashes with a black car, otherwise it will be -0.5.

- Off-road: the agent goes off its path and we assume that it is caused by 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: the time available to end the episode expires and this is strictly related to a conservative acceleration behavior; $r_{terminal}$ will be 0.0 for $R_{sa,t}$ and -1.0 for $R_{acc,t}$.

Finally, $r_{localization}$ is a penalization assigned to $R_{sa,t}$ and given to the agent when its position (x, y coordinates) and its heading (h_a) differs from the center lane and the heading (h_p) of the path respectively, such that:

$$r_{localization} = \phi \cdot \cos(h_a - h_p) + \psi \cdot d \tag{5.10}$$

where ϕ and ψ are constants set to 0.05 and d is the distance between the position of the agent and the center of the path it has to follow.

5.5 Results

We analyze the results achieved on the training intersections (Figure 5.1), using the metrics illustrated in Section 5.2: *Reaches*, *Crashes*, *Off-Roads* and *Time-overs* corresponding to the percentages of the episodes ended successfully, with a crash, with the agent off its path and with the depletion of available time respectively. Moreover, we also study a further metrics, namely the *Average Speed* of the agents in the three different intersections based on the traffic signs assigned to their lane.

We performed several experiments, analyzing how the agents behave differently depending on the traffic signs and thus proving that they learn the right of way in order to reach the common goal safely. Each test is composed by 3000 episodes using different traffic conditions depending on the random delay $[delay_{min}, delay_{max}]$ that agents must wait before starting a new episode, such that for lower values of delay range the traffic conditions will be denser. The values of $delay_{min}$ and $delay_{max}$ are set to 0 and 10 respectively and for each test we increase $delay_{max}$ of 10, performing as masy tests as $delay_{max}$ achieves the maximum value used during the training phase. Recalling that the $delay_{max}$ values used in the three intersections during the training phase are 30, 50 and 100 for the easy (Figure 5.1a), medium (Figure 5.1b) and hard scenario (Figure 5.1c) respectively, we performed three experiments for the easy intersection, five for the medium one and ten for the hard scenario.

	Easy Intersection		
	No Sign	Yield Sign	Stop Sign
Reaches %	0.998	0.995	0.992
Crahes %	0.002	0.005	0.008
Off-roads %	0.0	0.0	0.0
Time overs %	0.0	0.0	0.0
Average Speed $[\frac{m}{s}]$	8.533	8.280	8.105
	Medium Intersection		
	No Sign	Yield Sign	Stop Sign
Reaches %	0.995	0.991	0.992
Crahes %	0.005	0.009	0.008
Off-roads %	0.0	0.0	0.0
Time overs %	0.0	0.0	0.0
Average Speed $[\frac{m}{s}]$	8.394	7.939	7.446
	Hard Intersection		
	No Sign	Yield Sign	Stop Sign
Reaches %	0.997	0.991	0.972
Crahes %	0.003	0.009	0.012
Off-roads %	0.0	0.0	0.0
Time overs %	0.0	0.0	0.016
Average Speed $[\frac{m}{s}]$	8.224	7.365	5.855

Table 5.1: Results obtained in the training intersections (Figure 5.1) with the three different traffic signs.

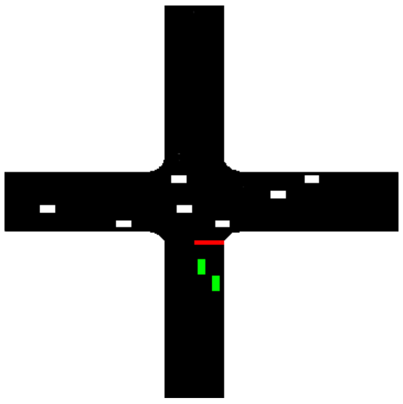
Table 5.1 shows the results achieved by the agents in the three different intersections facing the three different traffic signs proving that agents learn to handle those environments driving and performing the crossing maneuver safely. We can notice that handling the intersection with the stop sign is the hardest task since the vehicle has always the lowest priority. The *Average Speed* metric shows how agents modulate their behavior based on their traffic signs and hence on their priorities, while the *Off-roads* cases never happen proving that agents learn to drive correctly without going off their paths. The following video[2] shows how the agents handle the three intersection scenarios facing different traffic signs and traffic conditions.

5.5.1 Comparison with the TTC Algorithm

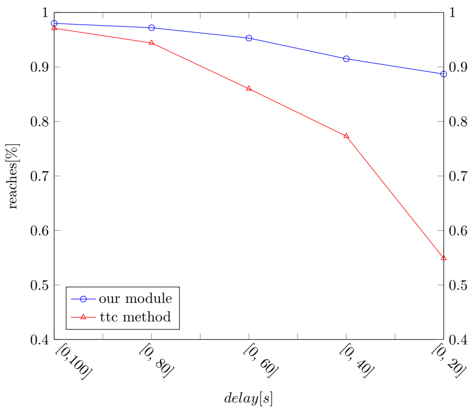
To prove that the obtained model behaves efficiently not only in terms of safety but also of time, we compared our system with the time-to-collision (TTC) algorithm in the hard intersection. In particular, we use such scenario performing two different experiments: the first one testing the agents trained with our D-A3C algorithm, while the second one with agents which simply follow the center of the path, choosing the right time to cross the intersection using the TTC algorithm. Both the tested agents (the green ones in Figure 5.5a) start their episodes from the branch with the stop sign as illustrated in Figure 5.5a, while traffic vehicles start from the two adjacent lanes following the center lane and using the Intelligent Driving Model (IDM) algorithm to modulate their speed. In this configuration, the green agents performing the crossing maneuver have always the lowest priority compared to the traffic cars.

In the TTC experiments, agents start the episode decelerating in order to reach the stop sign with zero speed; then, a single threshold is used to estimate when accelerate to cross the intersection, such that considering an imaginary line aligned with the longitudinal axis of the agent, the time-to-collision is computed as the time a traffic vehicle reaches this line, assuming that it drives with constant speed. In this way, if we find that the lowest TTC of a traffic car exceeds a specific threshold, the agent accelerates using a value of $3.0\frac{m}{s^2}$, otherwise it continues waiting at the stop line. This threshold is chosen in order to obtain the best results with the TTC method and zero percent of *Crashes*.

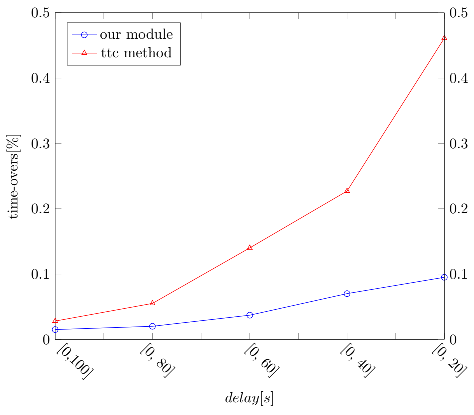
[2]https://www.youtube.com/watch?v=x28qRJXiQfo



(a) Test scenario



(b) Reaches



(c) Time-overs

Figure 5.5: (5.5a) shows the intersection used as a test scenario to compare the performance achieved by our module with those obtained by the TTC method. The tested agents (the green ones) perform the maneuver while traffic vehicles (the white obstacles) follow the center of the path using the Intelligent Driver Model (IDM) algorithm to control their speeds. (5.5b) and (5.5c) show the comparison between the percentages of episodes ended successfully and the percentage of time-overs obtained by ourmodule (the blue curves) and the TTC algorithm (the red curves) respectively, showing that reducing the delay range the traffic vehicles have to wait (and thus increasing the traffic congestion) the difference between the two appraches becomes more relevant

Both the learned agents and those ones following the TTC algorithms have the same available time to finish the episode, and as the previous experiments we use different traffic condition levels: *low, medium* and *high*, corresponding to a maximum number of traffic vehicles involved simultaneously in the environment to 4, 8 and 12 respectively.

We performed several experiments, in which traffic agents should wait different delay ranges illustrated in the graphics of Figure 5.5 in order to gradually increase the traffic congestion and thus the difficulty of the crossing maneuver. The results obtained using low and medium traffic levels by the TTC method and the D-A3C algorithm are impressive since they both reach almost 99% of episodes ended successfully for every delay range configuration. Instead, with the *high* traffic condition, the performance achieved by TTC method drops leading the agent to unnecessary waits. The comparison between the results obtained by the D-A3C algorithm and TTC is illustrated in Figure 5.5, and we can notice how the difference between the two approaches becomes more and more evident increasing the traffic level of the environment, namely reducing the delay range traffic cars must wait before starting a new episode.

5.5.2 Testing the Right of Way Rule

Even if the results illustrated in Table 5.1 obtained in the training scenarios and the comparison achieved between our module and the TTC method show that our agents learned to drive handling the intersections predicting both the acceleration and the steering angle safely in terms of time and safety, we have not yet proved that our agents learn the priority to the right rule.

At this purpose, we performed a test on the training scenarios in which agents start the episode using the same current and target speed such that they approach the intersection almost simultaneously. As for the training phase, only one agent can start from a single lane such that the number of vehicles involved each episode is equal to 3, 4 and 8 for the easy, medium and hard intersection respectively. Only in this way we can understand if agents with lower priorities give the right of way to those ones with higher priorities; for this test we consider the episode ended when all the agents involved reach a terminal state.

	Easy Intersection	Medium Intersection	Hard Intersection
No Infraction %	0.985	0.990	0.863
Infraction %	0.015	0.010	0.137

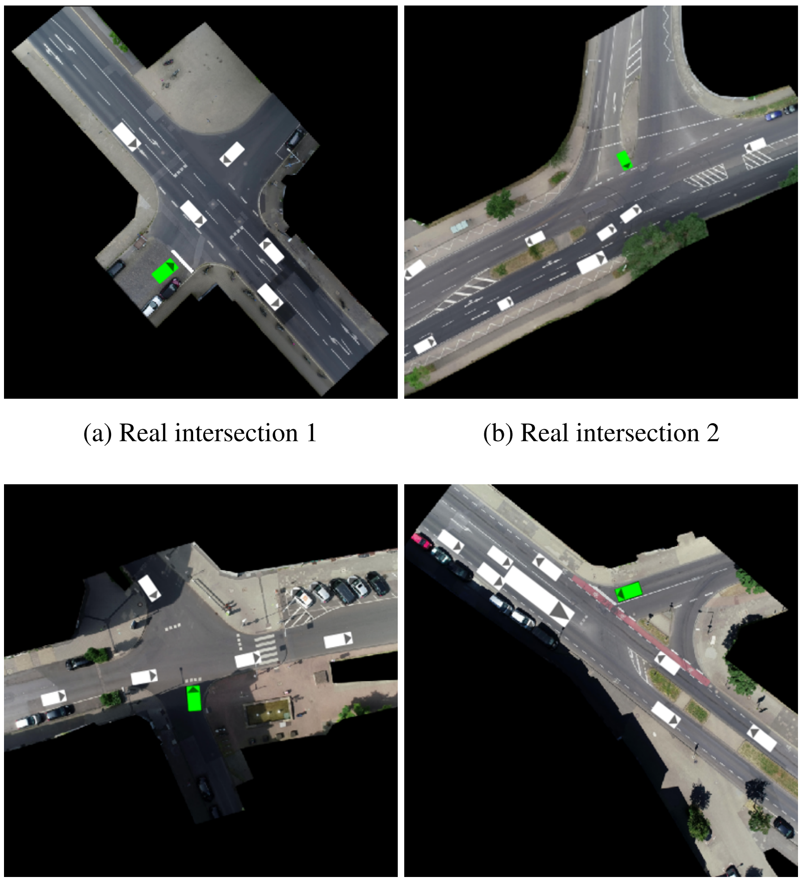
Table 5.2: Percentages of episodes ended following the right of way rule by all the agents involved in the episode (*No infraction*) and with at least an infraction (*Infraction*).

For each intersection we perfomed 9000 episodes and in order to properly evaluate the behavior adopted by agents we consider an episode ended without infractions (*No Infraction*) if and only if all the agents involved in the episode respect the priority to the right rule, otherwise it will be considered as a failure (*Infraction).* From the results obtained and collected in Table 5.2, we can notice that agents follow the priority to the right rule in most of the cases and only in the hard intersection there is a slight decrease in the performances since the larger spaces of such environment allow vehicles to risk the maneuver more frequently.

5.5.3 Test on Real Data

Finally, we tested our module on real data using the inD dataset ([89]), containing 33 real traffic sequences in four different intersections (Figure 5.6) resulting in a total of 10 hours of recorded data. The dataset contains the tracking and classification of more than 11500 static and dynamic road users including cars, trucks, busses, bicyclists and pedestrians collected with a camera-equipped drone. However, we only consider dynamic road users for such tests, and since pedestrians and byciclists are not used during the training phase we do not include them during such tests.

During these tests, we assume that the ego-agent (green cars in Figure 5.6) has always the lowest priority since it is not sensed by traffic vehicles. Moreover, it always starts from the same lane (the least busy), choosing a random exit each episode; for this reason, we do not include those traffic vehicles starting from the same lane of the


(a) Real intersection 1 (b) Real intersection 2

(c) Real intersection 3 (d) Real intersection 4

Figure 5.6: Real intersections contained in the inD dataset [89] and used for testing our agents (green cars) using the recorded traffic data (white vehicles)

ego-agent. In this way, the total number of traffic road users (cars, trucks and busses) populating the four real scenarios is 7386, and the episodes performed by our agents are 2702.

We were able to reproduce the four intersections in simulation with CAIRO grahic library [66] using the recorded trajectories of the dynamic traffic vehicles.

Using the code provided by [89] on their github page[3], we saved data related to our agent positions, speeds and headings in order to project them on the real scenarios (Figure 5.6).

The percentage of episodes ended successfully is greater than 99%, with 0% time-over and 0% off-road cases, and some examples of the behavior adopted by our agents are illustrated at the following video[4]. Observing the video we can notice that some risky maneuvers performed by our agents are emphasized as it is not sensed by traffic vehicles; nevertheless, these represent promising results, especially considering the diversity of scenarios and traffic behaviors observed during these tests compared to those used during the training phase.

5.6 Conclusions

Even if for the specific task of intersection handling, the development of a module able to predict continuous actions related both to the acceleration and the steering angle, represents the first implementation of a Reinforcement Lerning planner of such research project. In particular, the system was able both to drive the car along its predetermined path safely and to avoid crashing with other vehicles at the intersection, modulating the steering angle and the acceleration properly. Training the agents on the intersections of Figure 5.2a, the model learned to follow the priority to the right rules, achieving impressive results facing intersections and traffic conditions unseen during the training phase (Section 5.5.3).

However, in order to test such system on board of a real self-driving vehicle we also need to consider to achieve a smoother driving style to obtain a comfort and feasible behavior. At this purpose, before testing such module on real world environments, we first focus on improving the driving style of the system, training agents using a new simulator in obstacle-free environments as we will explain in the next chapter.

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

[4]https://youtu.be/SnKUk2k9YCg

Chapter 6

Tackling Real-World Planning and Control using Deep Reinforcement Learning

The results obtained in the previous chapter proved the great capabilities of the D-A3C algorithm in handling intersection scenarios training agents in a *Multi-agent* fashion, in which they learned both to navigate safely following their paths and to observe the priority to the right rule.

However, before testing such module on board of a real self-driving car, we focused on increasing the quality of the driving style obtained by the previous policy. At this purpose, we developed a Reinforcement Learning planner training agents in a new simulator based on HD maps (Section 3.2.5), achieving a comfort and safe driving style both in simulation and in real-world scenarios.

Finally, to obtain simulated agents that execute a target action as similar as possible the real vehicle would behave, a simple model based on a neural network has been developed in order to simulate as likely as possible the real self-driving vehicle dynamic behavior.

6.1 Environment

As explained in Section 3.2.5, for the last part of this research project we used a new simulator based on the HD maps illustrated in Figure 6.1 representing a low-traffic neighborhood that is included in one of the areas designated for autonomous driving testing in Parma. The blue streets are the mapped areas from which it can be possible to create an arbitrary number of scenarios used for the training phase.

The environments extracted from the mapped area and used for training agents, are those ones circled in red in Figure 6.1 and an example of the full-view of a scenario and the surrounding view of the agent are represented in Figure 6.2.

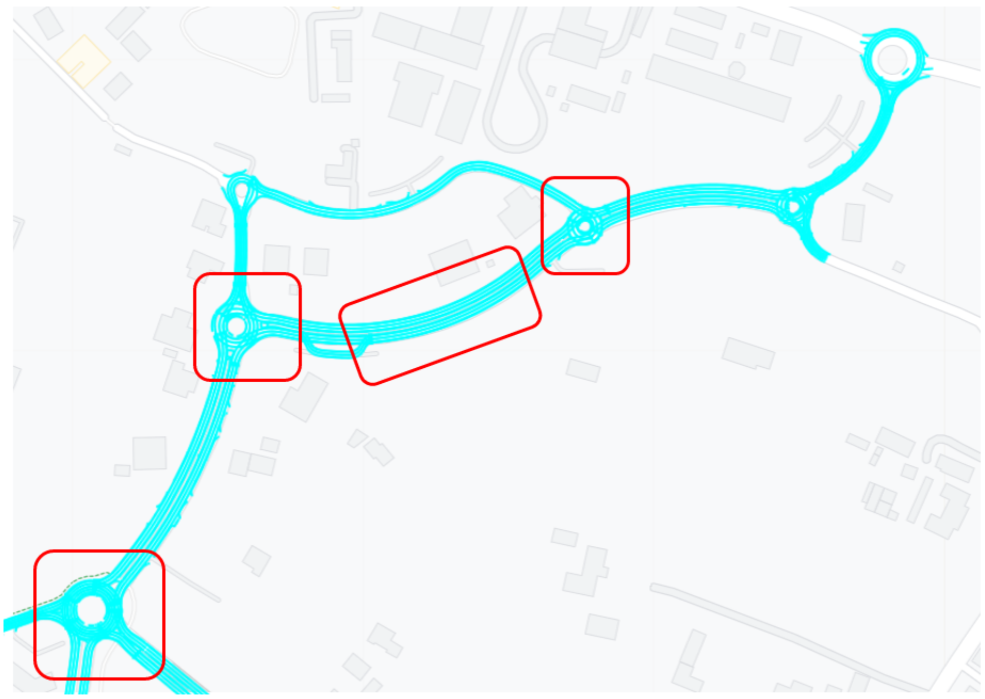


Figure 6.1: The figure shows the mapped area (the blue roads) of a neighborhood of Parma. The four parts circled in red represent the scenarios used for training agents to drive following their paths predicting both acceleration and steering angle

Moreover, in addition to speeding up the creation process of the training scenarios (that however remain subordinated to the availability of HD maps), during the training phase it is possible to easily retrieve several information about the external environments such as positions, headings, widths of the lanes and much more. However, apart from the information related to the reconstruction of the navigable space, path and stop line channels which can be easily achievable using only topological maps, we only use the HD maps to retrieve the additional information related to the road speed limits.

At this stage of work we only focused on the quality of the driving style, trying to optimize a policy that behaves as well as possible both in simulation and in real-world scenarios on board of the real self-driving car illustrated in Figure 4.12. For

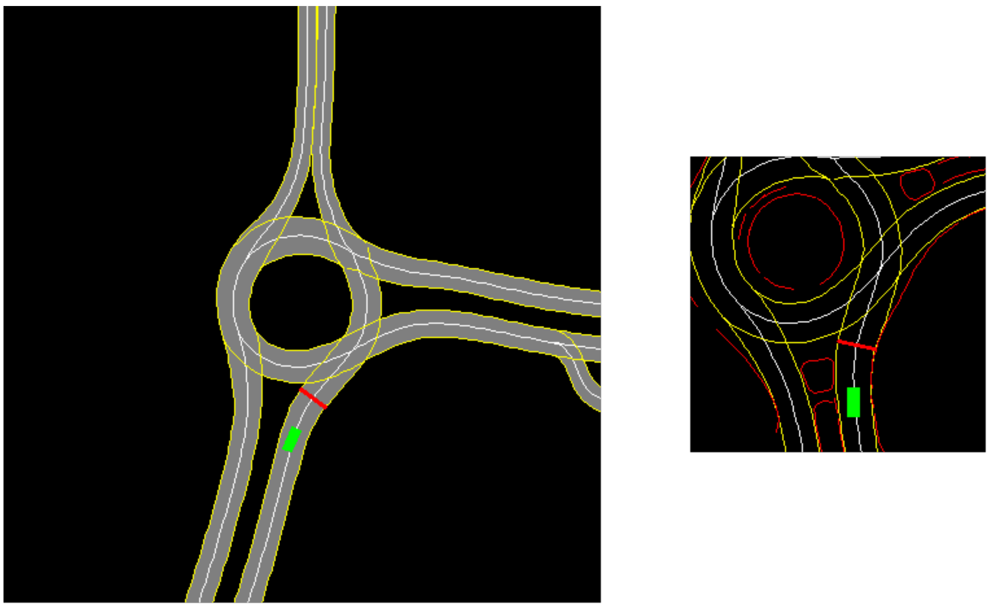


Figure 6.2: On the left of the figure is represented an example of the top-view of a training scenario, while on the right the 50×50 meters surrounding perceived by the agent, that is then split in the four channels (navigable space, path, obstacles and stop line) as explained in Figure 5.2

this reason, the agents are trained on static scenarios without using obstacles or other traffic vehicles, but they only focus on driving smoothly and safely following their paths and observing the speed limits obtained by HD maps.

6.2 Training Settings

The agents are trained in the four scenarios illustrated by the circled regions in Figure 6.1, using the D-A3C algorithm explained in Section 4.2.3. Each episode the agent starts with a random initial speed $[0.0, 8.0]$, learning to drive smoothly and safely along its predetermined path following the target speed, which is retrieved by the road speed limits of HD maps that could vary during the episode as in the case of roundabout scenarios. The speed limit range in the proposed environments is $[4.0, 8.3]\frac{m}{s}$.

Agents are trained through a neural network similar to the one implemented in Section 5.3.1 which predicts acceleration and steering angle every 100 milliseconds.

Since the training is performed in static scenarios, namely without using obstacles and traffic vehicles, the episode could end in one of the following terminal states:

- Goal reached: the final goal position is achieved by the agent without going outside its path.
- Off-road: the episode ends with the agent outside its path, because of erronoeous predictions of the steering angle.
- Time-over: the available time to finish the episode expires due to a overly cautious behavior of the agent in the prediction of acceleration outputs.

6.2.1 Neural Network Architecture

The neural network used for training agents to navigate smoothly and safely along their path is similar to the one illustrated in Figure 5.4 thus using two separate branches predicting the steering angle and the acceleration every 100 milliseconds and sharing the same feature vector extracted from scalar parameters. The main difference from the network architecture illustrated in Figure 5.4 consisted in using 5 scalar parameters including the target speed (the road speed limit retrieved by HD maps), the 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. Moreover, instead of using a predetermined standard deviation σ which linearly decreases as the number of episodes increases, we let the network predicts and thus modulates such value during the training phase, such that this prediction can also be interpreted as an estimation of the model uncertainty.

6.2.2 Simulating Self-Driving Car Behavior

One of the main problems related to the use of simulators consists in the difference between simulated and real data, that in the case of graphic simulators this gap is related both to environmental features and the behavior of the road users. However, we strongly reduce such difference creating synthetic representations of realities, then reproducing the same channels used in simulation on board of the real self-driving car using perception, localization and mapping systems. Nevertheless, even in these non-graphic simulators, techniques aim at simulating some unexpected behaviors of real prediction or localization systems embedded on the autonomous car are often essential in order to reduce the gap between simulated and real data (Section 4.5).

Moreover, another relevant problem is related to the difference in how simulated agents perform a target action compared to how the autonomous car would behave executing that command. Indeed, a target action computed at time t can ideally be executed with immediate effect at the same precise instant of time in simulation (blue curve in Figure 6.3 representing steering angle values), but this not happens on board of a real vehicle in which such target action will be performed with some delay and its complete effect will occur at time $t+\delta$. For this reason, it is necessary to simulate such response time in simulation, in order to train agents to handle such delay on board of real autonomous car.

In order to achieve such behavior, we initially trained agents adding a fixed delay in the execution of a target action predicted by the neural network as illustrated by the green curve in Figure 6.3. However, as we can notice from the figure, the difference in the response time between simulated and real vehicle (the orange curve in Figure 6.3) still remains relevant, causing unexpected and erroneous behaviors when testing the system on board of the autonomous car.

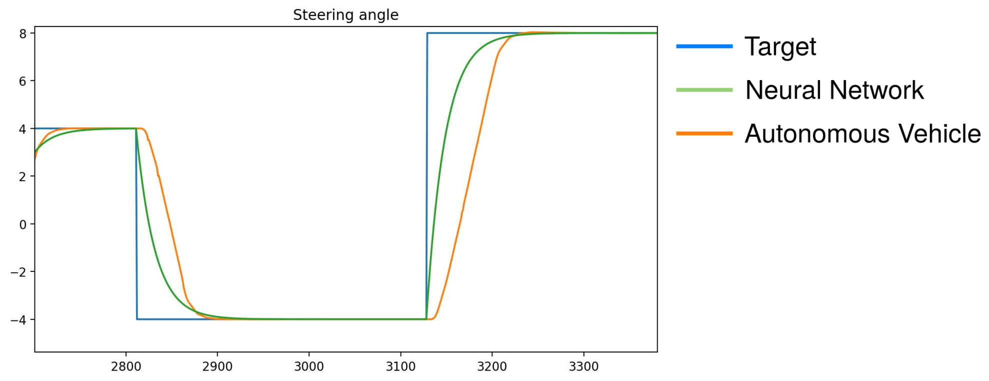


Figure 6.3: The proposed example shows the difference in the response time execution of the steering angle between the simulated agent and the real autonomous car. The blue curve represents the target action that could be feasible to execute in simulation but not on board of the vehicle which will behave accordingly to the orange curve. The green curve represents the agent behavior in simulation applying a fixed delay in the execution of the target action, implemented to reduce the gap between autonomous vehicle and simulated agent response time. However, even if such gap is reduced compared to the difference between the target action and real vehicle curves, it is still not sufficient to achieve a smooth behavior when testing the whole system on board of the self-driving car. For this reason, a small neural network aim at imitating the vehicle behavior has been developed and its trend in the proposed figure would basically overlap the orange curve. The same idea has been applied for the acceleration output

For this reason, a model consisting in a small neural network (which we will call *deep_response*) has been developed in order to reproduce as realistically as possible, the response of the real vehicle to a target action; in other words, the neural network is trained using a dataset collected on board of the autonomous car (Figure 4.12) composed by random target actions and its actual evolution executed by the vehicle over time. In this way, we embedded such model in the simulator in order to evolve the action predicted by agents, thus having a more scalable system aims at reproducing the autonomous vehicle behavior compared to the usage of fixed thresholds. A plot of the proposed neural network behavior will be very similar to the orange curve of Figure 6.3 (the one representing the real self-driving 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 is applied to the acceleration output.

Given the complexity introduced by such module, a recurrent layer and in particular a *Long short-term memory* (LSTM) [90] has been added at the end of the two fully connected layers that receive as input the scalar parameters in order to give agents a memory related to the past performed actions and not just the last one. The introduction of such layer allows the agent to mitigate the delay introduced by the *deep_response* model, thus achieving a smooth and safe behavior testing the whole system in real-world scenarios. The final architecture of the neural network is illustrated in Figure 6.4.

6.2.3 Reward Function

As for the systems described in Chapter 4 and Chapter 5, a reward shaping is essential to achieve a policy with a smooth and safe behavior. Moreover, also in this case two reward functions have been designed in order to give a more specific signal to the acceleration ($R_{acc,t}$) and to the steering angle ($R_{sa,t}$). However, since agents are trained in static scenarios, without interacting with obstacles or traffic vehicles, we do not consider a penalization related to accidents between agents as instead it was defined in Section 5.4. In this way, the two reward functions can be defined as:

$$R_{acc,t} = r_{speed} + r_{acc\_indecision} + r_{terminal} \tag{6.1}$$

$$R_{sa,t} = r_{localization} + r_{sa\_indecision} + r_{terminal} \tag{6.2}$$

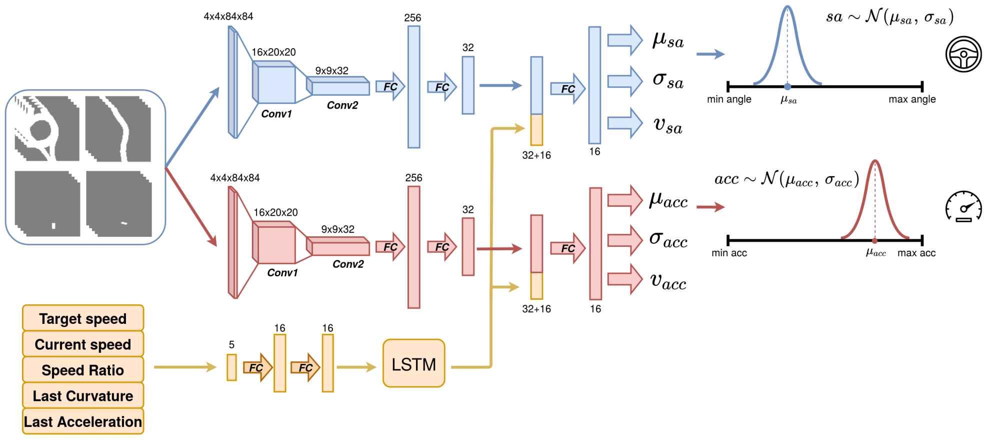


Figure 6.4: The final neural network architecture used for training agents to drive following their paths and observing the road speed limits. The main differences compared to the neural netwok of Figure 5.4, consist in the introduction of the LSTM layer after the two fully connected layers that receive the scalar parameters as input and in the prediction of the standard deviations (σ_{sa} and σ_{acc})

r_{speed} is a signal given to the agent in order to encourage it to reach but not to overcome the target speeds, which represent the speed limits retrieved by HD maps; this function depends on the ratio between the current speed and the target speed ($speed\_ratio = \frac{curr\_speed}{target\_speed}$) and it is defined as:

$$r_{speed} = \begin{cases} speed\_ratio \cdot \zeta, & \textbf{if } speed\_ratio < 1.0 \\ (speed\_ratio - 1.0) \cdot \zeta, & \textbf{otherwise}. \end{cases} \tag{6.3}$$

where ζ is constant set to 0.009.

$r_{acc\_indecision}$ aims at penalizing the agent when the difference between two consecutive accelerations ($\Delta_{acc} = |acc(t) - acc(t-1)|$) differs from a fixed threshold ($\delta_{acc} = 1.0\frac{m}{s^2}$), in order to obtain a smooth longitudinal behavior of the vehicle. In this way, $r_{acc\_indecision}$ is defined as:

$$r_{acc\_indecision} = \psi \cdot \min(0.0, (\delta_{acc} - \Delta_{acc})) \tag{6.4}$$

where ψ is a constant set to 0.1.

$r_{localization}$ is associated to $R_{sa,t}$ and it is a negative signal given to the agent when its heading and position (x, y coordinates) differs from those of the HD maps, such that:

$$r_{localization} = \phi \cdot \cos(h_a - h_p) + \chi \cdot d \tag{6.5}$$

where h_a and h_p are the heading of the agent and the point of the HD maps respectively, ϕ and χ are constants set to 0.05 and d is the lateral distance between the position of the agent and the nearest point of the HD maps.

In order to obtain a smooth lateral behavior of the vehicle, $r_{sa\_indecision}$ is designed to penalize the agent when the difference of two consecutive predictions of the steering angle ($\Delta_{sa} = |sa(t) - sa(t-1)|$) differs from a fixed threshold ($\delta_{sa} = 0.05$), such that:

$$r_{sa\_indecision} = \lambda \cdot \min\left(0.0, (\delta_{sa} - \Delta_{sa})\right) \tag{6.6}$$

where λ is a constant set to 0.01.

Finally, the only common member between $R_{acc,t}$ and $R_{sa,t}$ is $r_{terminal}$, and it depends on the terminal state achieved by the agent:

- Goal reached: the agent achieves the final position safely and $r_{terminal}$ will be $+1.0$ both for $R_{acc,t}$ and $R_{sa,t}$.

- Off-roads: the agent ends the episode going outside its path due to erroneous predictions of steering angle outputs; for this reason $r_{terminal}$ will be -1.0 for $R_{sa,t}$ and 0.0 for $R_{acc,t}$.

- Time-over: the available time to finish the episode expires because of cautious behavior related to the acceleration output; for this reason, $r_{terminal}$ will be -1.0 for $R_{acc,t}$ and 0.0 for $R_{sa,t}$.

6.3 Real-World Test

The architecture for testing the system on board of the autonomous vehicle will be similar to the one illustrated in Figure 6.5, thus composed by sensors to retrieve data from the environment, perception, localization and HD maps systems in order to reproduce the four channels related to navigable spaces, paths, obstacles and stop lines that are given as input to the neural network. Finally, the final parts related to the planning and control tasks are completely replaced by the neural network that predicts acceleration and steering angle, resulting in a more simple and scalable architecture (Figure 6.5).

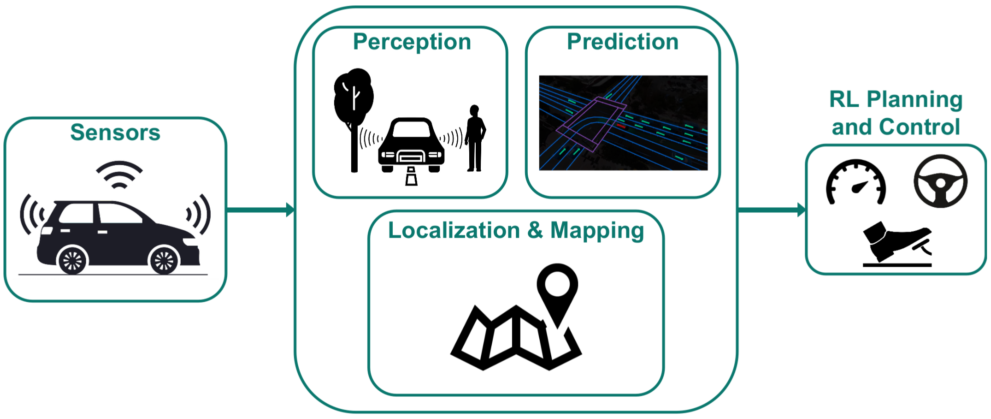


Figure 6.5: The final architecture implemented and tested on board of the autonomous car in which the typical parts related to planning and control are represented by a single neural network that predicts acceleration and steering angle using the information processed by the localization, mapping and perception systems

In particular, we tested two different policies over the entire mapped area (the blue roads in Figure 6.1) analyzing the obtained behaviors:

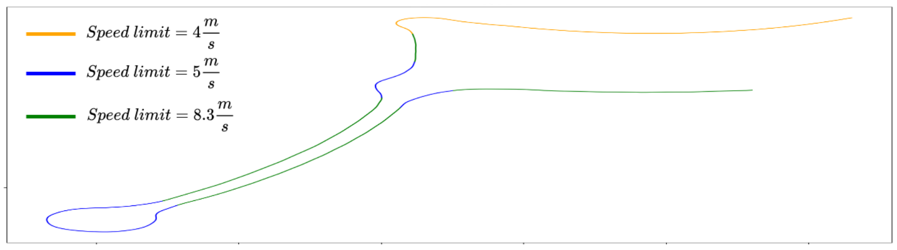
- Policy 1: it is trained without using both the *deep_response* model (Section 6.2.2) and the LSTM layer, but using a fixed delay in the execution of the actions predicted by the agent as explained in Section 6.2.2 and illustrated by the green curve of Figure 6.3.
- Policy 2: in this case the system is trained using the *deep_response* module (Section 6.2.2) and introducing the LSTM layer in the network architecture as illustrated in Figure 6.4.

Both policies behave smoothly and safely in simulation allowing the agents to achieve 100% of episodes ended successfully both in training scenarios (those ones circled in Figure 6.1) and in those parts of the mapped area different from the training environments.

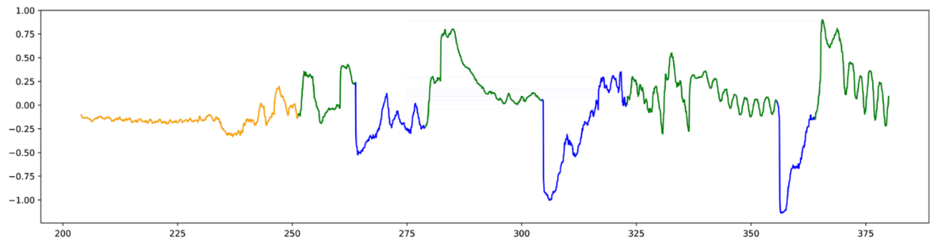
However, the behavior observed testing *Policy 1* in real-world scenarios was noisy and uncomfortable since it was not able to handle the delay introduced by the autonomous car as explained in Section 6.2.2 (Figure 6.3). This brings the autonomous car to continuous and unexpected corrections especially of the steering angle, often leading the huma driver to regain the control of the vehicle to avoid running off the road. This mainly happens because the actions predicted by the neural network on board of the autonomous car do not have the same effect that they would have in simulation, thus changing its internal state and the environment unexpectedly.

On the contrary, *Policy 2* behaves correctly both in simulation and in real scenarios, driving safely for the 100% of tests performend on the whole mapped area (Figure 6.1) without the need for the human driver to regain the control of the vehicle, with the exception of those cases in which it was necessary to avoid other road users; we did not consider these cases as failures since both *Policy 1* and *Policy 2* are trained in obstacle-free scenarios.

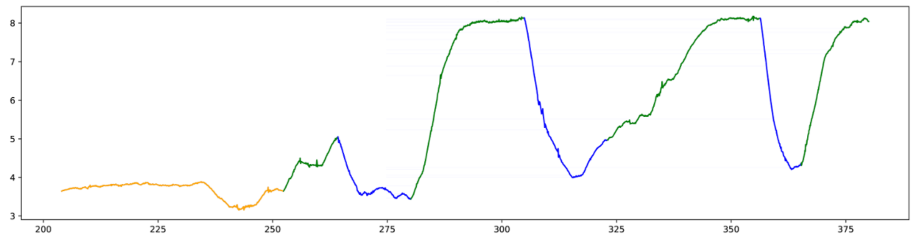
Moreover, the system learned to modulate the velocity accordingly to the speed limits of the map. Indeed, in order to show a more detailed example of the longitudinal behavior of the real car, we plot the accelerations predicted by the network (Figure 6.6b) and the vehicle speeds (Figure 6.6c) obtained driving along the route of Figure 6.6a. In particular, the paths drawn as orange, blue and green of Figure 6.6a represent those parts of the mapped area in which the speed limits are set to 4, 5 and 8.3 $\frac{m}{s}$ respectively, and following the same color-based rule, we can notice how the neural network modulates the acceleration and thus the vehicle speed based on the route speed limits retrieved by the HD map. Finally, it is also important to notice that even if the acceleration output seems noisy, the difference between two consecutive predicted values never exceeds the threshold δ_{acc} used in the reward function (Section 6.2.3), resulting in a smooth and comfort longitudinal behavior perceived on-board vehicle.



(a) Vehicle positions (x, y coordinates) and road speed limits



(b) Acceleration predicted by the neural network



(c) Speed values

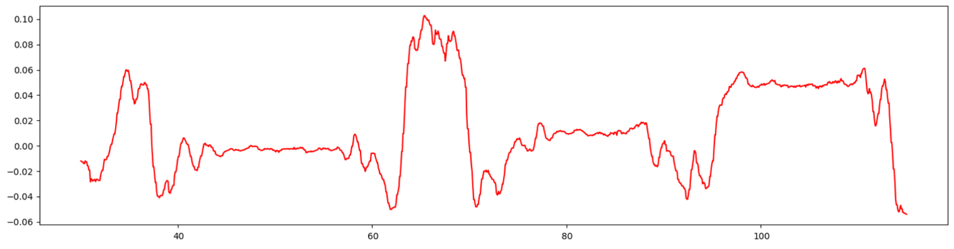
Figure 6.6: A test example showing the longitudinal behavior of the Reinforcement Learning planner deployed in the real autonomous vehicle. In particular, the illustrated data refer to the car behavior driving along the route (6.6a), in which in the orange, blue and green paths the road speed limits are set to 4, 5, and 8 $\frac{m}{s}$ respectively. Following the same color-based rule, (6.6b) shows the acceleration values predicted by the neural network in the different parts of the route and the corresponding vehicle speeds (6.6c), proving that the system is able to modulate the longitudinal behavior in order to follow the road speed limits

Finally, we also analyzed the lateral behavior achieved by *Policy 2*, proving that the introduction of the *deep_response* module and the LSTM layer in the network architecture are useful to learn agents to handle the delay introduced by the autonomous car. In particular, the red curve in Figure 6.7 represents the curvature values predicted by the neural network every 100 milliseconds, while the red and blue curves in Figure 6.7b represent the corresponding predicted values to be actuated and the effective values performed by the vehicle respectively. It is easily observable the delay introduced by the autonomous car, which however was learned by *Policy 2* since it was trained using *deep_response* in order to replicate in simulation such delay and the behavior of the real car as realistically as possible in the execution of target actions. In this way, the delay observed in Figure 6.7b does not represent an issue for *Policy 2* that, unlike the results achieved by *Policy 1* during the real tests, is able to navigate safely and smoothly over all the mapped area of Figure 6.1.

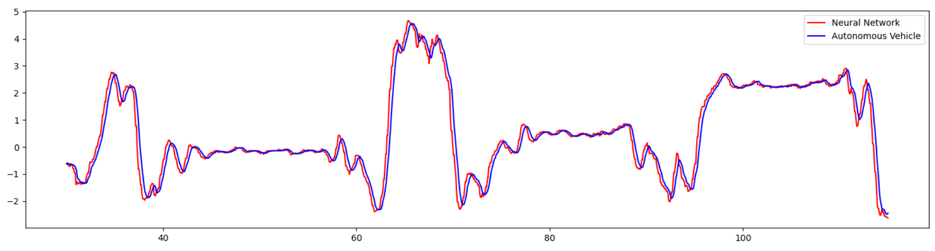
Moreover, as we can see from the following video[1], the vehicle is able to drive correctly both in the training areas and also in those ones unseen during the training phase, proving that the system does not overfit on the only training scenarios and showing good generalization capabilities.

The results achieved at this stage of work represent promising results for future developments which will surely consist in introducing traffic vehicles and multiple actor-learners sharing the same environment thus achieving a multi-agent training.

[1]https://drive.google.com/file/d/1gRqkd1R59Qh96f0kWLyh3L0z19h6w_5i/view?usp=sharing



(a) Curvature predicted by the neural network



(b) Curvature commands (red) and actual executed values (blue)

Figure 6.7: Curvatures predicted by the neural network (6.7a) and the corresponding actuation values (red curve in 6.7b). The blue curve represents the effective commands performed by the vehicle showing the delay introduced by the autonomous car (Section 6.2.2)

Chapter 7

Conclusions and Future Developments

The main purpose of this research project consisted in exploring Deep Reinforcement Lerning algorithms applied to autonomous driving field, with a particular focus on the performance obtained by such techniques in real-world scenarios and with the final goal of developing a Reinforcement Learning planner able to drive safely and smoothly both in simulation and in real world.

We started implementing a maneuver planning system able to handle the roundabout immission, consisting in training a neural network that predicts a discrete signal to modulate the longitudinal behavior of the trained agents together with the *state value function*. We demonstrated that the module obtained was able to perform the maneuver safely negotiating the immission with the othe traffic vehicles, but its generalization capabilities had to be increased in order to test it in real scenarios. At this purpose, we implemented a *Multi-environment System* setting, developing the typical pipeline of supervised approaches consisting in training, validation and test set (in our case representing by roundabout scenarios). Moreover, in order to reduce the gap between simulated and real-world data and thus to obtain a more robust policy able to handle the uncertainties and the novel situations encountered in the real-world, we injected noisy elements inside the simulator to reproduce errors that perception and localization systems on board of the autonomous car may introduce. In this way, we increased the generalization capability of the policy, which was able to behave safely and correctly both in unseen simulated environments and in a real roundabout used as a test scenario.

As a natural extension of such system, we increased the difficulty of the task to solve developing a neural network that predicts continuous values to control both lateral and longitudinal behaviors. Moreover, at this stage of the research project we trained the neural network to predict the means of two Gaussian distributions used to sample the corresponding acceleration and steering angle values with a predetermined standard deviation which linearly decreased during the training phase. In particular, through the use of such module, agents were trained in a *Multi-agent* fashion on several intersection scenarios, learning both to drive safely along their designated paths and to observe the priorities of other actor learners computed according to the traffic sign states and the priority to the right rule. We proved that agents learn such basic rules driving and handling correctly the proposed intersections and also featuring some generalization capabilities testing the system on environments unseen during the training phase and facing real recorded traffic data.

However, the obtained driving style was not as smooth to test the system on board of the real vehicle, and for this reason the last part of the research project was conducted on the implementation of a system able to drive smoothly both in simulation and in real world scenarios. At this purpose, we developed a system able to simulate the delay introduced by the real vehicle in the execution of target actions. The embedding of such model in simulation, allowed us to obtain agents that behave similarly to the real vehicle training them through a neural network that predicts both the means and the standard deviations of two Gaussian distributions used to sample the acceleration and steering angle values, and adding an LSTM layer in order to give agents a memory of the past performed actions. However, at this stage of work, we focused on the quality of the driving style thus using obstacle-free training environments in which agents were only trained to follow their paths and to observe the road speed limits.

We tested the system on a low-traffic neighborhood included in one of the areas designated for autonomous driving testing in Parma and we observed that the system was able to drive smoothly both in those parts of the area used as training scenarios and in unknown areas, proving that the system did not overfit on the only training environments.

The behavior observed on board vehicle represents an excellent starting point in developing a full Reinforcement Learning planner able to handle different situations and tasks. However, increasing the complexity of the task to be solved and thus adding obstacles, traffic vehicles or simply performing *Multi-agent* training, the capabilities of the system has to be improved in order to handle multiple tasks simultaneously. In this way, the neural network architecture could be redesigned and it may be composed by different sub-modules, thus handling different tasks in a hierarchical fashion, such as obstacle avoidance, overtaking, immission maneuvers and many others.

Moreover, another main limitation of the system is related to the size of the input images used for training the neural network. Indeed, as explained in Chapter 3, each agent in the simulator perceives a surrounding view of 50 meters, observing 40 meters forward, 25 left and right and 10 backward, obtaining input images of size 84×84 pixels; this limitation is even more significant if we consider that cameras on board of the self-driving vehicle (Figure 4.12) used for testing the systems described in this thesis have a range of 100/150 meters. For this reason, future developments are also directed towards increasing image input size, allowing the agent to observe and handle wider areas and thus using higher speed values. However, the increase of the input size will surely require the use of a more powerful encoder like the Variational Autoencoder (VAE), that could be useful to speed up the training process of the reinforcement learning agents: indeed, once the VAE weights are optimized, there may be no need to retrain the convolutional layers every time, thus updating only the last layers of the network, typically composed by fully connected layers.

Bibliography

[1] Santokh Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. 2015.

[2] Carl Engelking. The 'driverless' car era began more than 90 years ago [online]. 2017. URL: `https://www.discovermagazine.com/technology/the-driverless-car-era-began-more-than-90-years-ago`.

[3] Evan Ackerman. Self-driving cars were just around the corner - in 1960 [online]. 2016. URL: `https://spectrum.ieee.org/tech-history/heroic-failures/selfdriving-cars-were-just-around-the-cornerin-1960`.

[4] Taylor Kubota. Stanford's robotics legacy [online]. 2019. URL: `https://news.stanford.edu/2019/01/16/stanfords-robotics-legacy/`.

[5] Aditya Bhat. Autonomous vehicles: A perspective of past and future trends. *International Journal of Engineering Technology Science and Research*, Volume 4, Issue 10, 2017.

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

[7] Dean Pomerleau and Todd Jochem. No hands across america journal [online]. 1995. URL: `https://www.cs.cmu.edu/~tjochem/nhaa/Journal.html`.

[8] A. Broggi, M. Bertozzi, and A. Fascioli. Argo and the millemiglia in automatico tour. *IEEE Intelligent Systems and their Applications*, 14(1):55–64, 1999.

[9] R. Behringer. The darpa grand challenge - autonomous ground vehicles in the desert. *IFAC Proceedings Volumes*, 37(8):904–909, 2004. IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 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] Alberto Broggi, Paolo Medici, Paolo Zani, Alessandro Coati, and Matteo Panciroli. Autonomous vehicles control in the vislab intercontinental autonomous challenge. *Annual Reviews in Control*, 36(1):161–171, 2012.

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

[14] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[15] Eva Fraedrich and Barbara Lenz. *Societal and Individual Acceptance of Autonomous Driving*, pages 621–640. 05 2016.

[16] 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. IEEE, 2000.

[17] 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. IEEE, 2005.

[18] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[19] Muhammad Sualeh and Gon-Woo Kim. Dynamic multi-lidar based multiple object detection and tracking. *Sensors*, 19(6):1474, 2019.

[20] Takeo Kato, Yoshiki Ninomiya, and Ichiro Masaki. An obstacle detection method by fusion of radar and motion stereo. *IEEE transactions on intelligent transportation systems*, 3(3):182–188, 2002.

[21] Dirk Langer and Charles E Thorpe. Sonar based outdoor vehicle navigation and collision avoidance. In *IROS*, pages 1445–1450. Citeseer, 1992.

[22] Oliver J Woodman. An introduction to inertial navigation. Technical report, University of Cambridge, Computer Laboratory, 2007.

[23] Bernhard Hofmann-Wellenhof, Herbert Lichtenegger, and James Collins. *Global positioning system: theory and practice*. Springer Science & Business Media, 2012.

[24] ZuWhan Kim. Robust lane detection and tracking in challenging scenarios. *IEEE Transactions on intelligent transportation systems*, 9(1):16–26, 2008.

[25] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. Lane detection and tracking using b-snake. *Image and Vision computing*, 22(4):269–280, 2004.

[26] Yihuan Zhang, Jun Wang, Xiaonian Wang, and John M Dolan. Road-segmentation-based curb detection method for self-driving via a 3d-lidar sensor. *IEEE transactions on intelligent transportation systems*, 19(12):3981–3991, 2018.

[27] Wijerupage Sardha Wijesoma, KR Sarath Kodagoda, and Arjuna P Balasuriya. Road-boundary detection and tracking using ladar sensing. *IEEE Transactions on robotics and automation*, 20(3):456–464, 2004.

[28] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 international joint conference on neural networks (IJCNN)*, pages 1–8. Ieee, 2013.

[29] Arturo De La Escalera, Luis E Moreno, Miguel Angel Salichs, and José María Armingol. Road traffic sign detection and classification. *IEEE transactions on industrial electronics*, 44(6):848–859, 1997.

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[32] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11525–11533, 2020.

[33] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. Ieee, 2004.

[34] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[35] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[36] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[37] 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. IEEE, 2016.

[38] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[39] 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.

[40] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[41] Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign, 2004.

[42] Marek Grzes and Daniel Kudenko. Plan-based reward shaping for reinforcement learning. In *2008 4th International IEEE Conference Intelligent Systems*, volume 2, pages 10–22. IEEE, 2008.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland,

Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[44] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[45] Alexander Pritzel Nicolas Heess Tom Erez Yuval Tassa David Silver Timothy P. Lillicrap, Jonathan J. Hunt and Daan Wierstra. Continuous control with deep reinforcement learning. 2016.

[46] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[47] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[48] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[50] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[51] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[52] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[53] 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. PMLR, 2016.

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

[55] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044, 1996.

[56] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[57] Andrew William Moore. Efficient memory-based learning for robot control. 1990.

[58] 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.

[59] 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.

[60] 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. PMLR, 2017.

[61] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[62] Alex Bewley, Jessica Rigley, Yuxuan Liu, Jeffrey Hawke, Richard Shen, Vinh-Dieu Lam, and Alex Kendall. Learning to drive from simulation without real

world labels. In *2019 International conference on robotics and automation (ICRA)*, pages 4818–4824. IEEE, 2019.

[63] Daniel Krajzewicz, Georg Hertkorn, Christian Rössel, and Peter Wagner. Sumo (simulation of urban mobility)-an open-source traffic simulation. In *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*, pages 183–187, 2002.

[64] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

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

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

[67] Alessandro Capasso, Giulio Bacchiani, and Daniele Molinari. Intelligent roundabout insertion using deep reinforcement learning. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,*, pages 378–385. INSTICC, SciTePress, 2020.

[68] Stefano Masi, Philippe Xu, and Philippe Bonnifait. Roundabout crossing with interval occupancy and virtual instances of road users. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[69] Stefano Masi, Philippe Xu, and Philippe Bonnifait. Adapting the virtual platooning concept to roundabout crossing. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1366–1372. IEEE, 2018.

[70] Lejla Banjanovic-Mehmedovic, Edin Halilovic, Ivan Bosankic, Mehmed Kantardzic, and Suad Kasapovic. Autonomous vehicle-to-vehicle (v2v) decision making in roundabout using game theory. *Int. J. Adv. Comput. Sci. Appl*, 7(8):292–298, 2016.

[71] Yuhuai Wu et al. Openai baselines: ACKTR & A2C. 2017.

[72] Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, et al. The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4-5):405–420, 2018.

[73] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[74] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[75] 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. IEEE, 2020.

[76] Huynh Ngoc Phien and Nattawit Dejdumrong. Efficient algorithms for bézier curves. *Computer Aided Geometric Design*, 17(3):247–250, 2000.

[77] Alessandro Paolo Capasso, Paolo Maramotti, Anthony Dell'Eva, and Alberto Broggi. End-to-end intersection handling using multi-agent deep reinforcement learning. *arXiv preprint arXiv:2104.13617*, 2021.

[78] Kurt Dresner and Peter Stone. Multiagent traffic management: A reservation-based intersection control mechanism. In *Autonomous Agents and Multiagent Systems, International Joint Conference on*, volume 3, pages 530–537. IEEE Computer Society, 2004.

[79] Marco A Wiering. Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158, 2000.

[80] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2496–2505, 2018.

[81] Deepeka Garg, Maria Chli, and George Vogiatzis. Deep reinforcement learning for autonomous traffic light control. In *2018 3rd ieee international conference on intelligent transportation engineering (icite)*, pages 214–218. IEEE, 2018.

[82] Richard Van Der Horst and Jeroen Hogema. Time-to-collision and collision avoidance systems. In *Proceedings of the 6th ICTCT workshop: Safety evaluation of traffic systems: Traffic conflicts and other measures*, pages 109–121. Citeseer, 1993.

[83] Avik Pal, Jonah Philion, Yuan-Hong Liao, and Sanja Fidler. Emergent road rules in multi-agent driving environments. In *International Conference on Learning Representations*, 2021.

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

[85] 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. IEEE, 2018.

[86] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

[87] Arne Kesting, Martin Treiber, and Dirk Helbing. Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4585–4605, 2010.

[88] Y. Pu, Zhe Gan, Ricardo Henao, X. Yuan, C. Li, A. Stevens, and L. Carin. Variational autoencoder for deep learning of images, labels and captions. In *NIPS*, 2016.

[89] 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. IEEE, 2020.

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