UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione
XXXII Ciclo

Giulio Bacchiani

MICROSCOPIC TRAFFIC SIMULATION AND MANEUVER EXECUTION WITH MULTI-AGENT DEEP REINFORCEMENT LEARNING

DISSERTAZIONE PRESENTATA PER IL CONSEGUIMENTO
DEL TITOLO DI DOTTORE DI RICERCA

Anni 2016/2019

UNIVERSITÀ DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXXII Ciclo

MICROSCOPIC TRAFFIC SIMULATION AND MANEUVER EXECUTION WITH MULTI-AGENT DEEP REINFORCEMENT LEARNING

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Massimo Bertozzi

Dottorando: *Giulio Bacchiani*

Anni 2016/2019

Ad Ante, Domenico e Rosario

A Gianni

Summary

List of Figures

List of Tables

Introduction

The development of vehicles driven by computers could be seen as a step further in the substitution of humans by an artificial player for the execution of tasks which were previously done by themselves: this process is typically referred to as automation. The range of applications involved in this process is extremely broad and constantly changing, and it goes from simple devices that help us in our everyday life, such as washing machines, to more complicated controllers used in modern industrial plants.

0.1 History of automation

This process is not a 21^{st} century novelty, but it started way back in the past. A noteworthy example of ingenious automation dates back to Ptolemaic Egypt (around 270 *B.C.*), when the Greek inventor Ctesibius ([1]), led by the desire of keeping an accurate track of time, built a clock based on the collection of water coming from a constant water-flow. Every hour the container of this *water clock* was emptied by the use of a siphon and a mechanical counter was incremented. Moreover, since at that time Greek and Romans were considering 12 daylight hours independently on the period of the year, therefore with varying duration, the clock was self-adapting the quantity of water needed to 'fill' an hour. In a few words, this clock was able to give the exact time throughout a year with no human interventions.

Around the 17^{th} century, innovations in grain mills, furnaces, boilers and engines make the use of new automatic control systems necessary, as temperature, pressure and float regulators; short time later, in 1745, Jacques de Vaucanson invented the first automated loom. The automatic telephone switchboard was invented in 1892, bringing

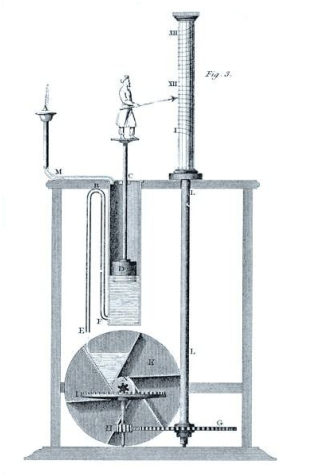


Figure 1: Ctesibius' water clock

a growth in the call volume which was feared to consume all American electricity production and inducing Bell Labs to research on the transistor ([2]).

However, the word *automation* was not popular before 1947, when the 'automation department' was established at the Ford Motor Company, increasing the average production by 20% ([3]). At that time, automation was described this way:

"Where previously, in the course of producing a unit, manual work supplemented each mechanical performance, the entire process becomes mechanized. Electronic controls built into the machines also inspect the work at various stages and approve it or correct it."

Since then, the degree of automation in the industrial sector has grown steadily and automated devices are of common use and often indispensable.

0.2 Self-driving cars development

Today the research on self-driving cars is registering an unprecedented development, but we are wrong if we think that it started in the last decades; indeed, almost a century has passed after the first tests on driverless cars.

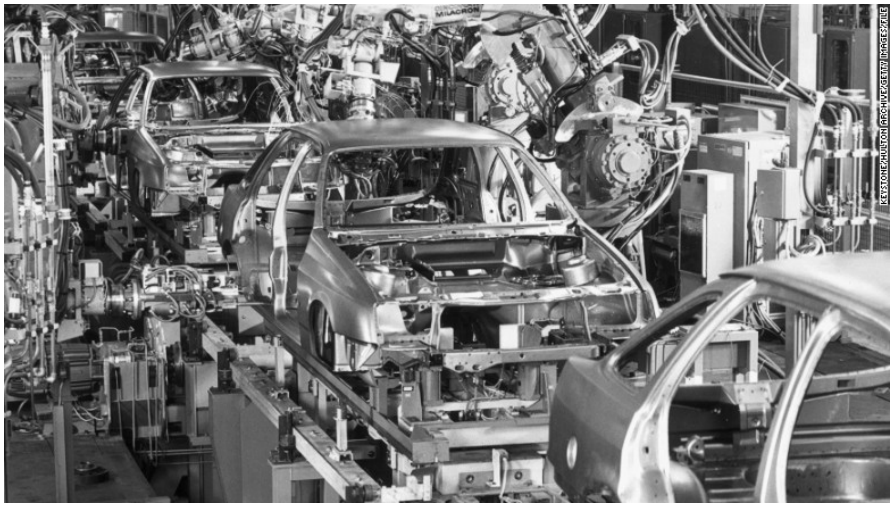


Figure 2: There are no humans in sight as robots weld the bodies of Ford Sierras in 1984

In 1925 Francis P. Houdina[1] of the Houdina Radio Control company showed the *American Wonder* along the streets of New York City, which was probably the first radio-operated car. Unfortunately, it seems that this demonstration ended with the American Wonder crashing into a vehicle of photographers documenting the event ([5]).

In order to reduce the human intervention needed for piloting the car, the following researches went in the direction of integrating additional equipments in the roads to control the trajectory of the cars, like magnetic cables and sensors buried in the pavement. Between the '50s and '60s some demonstration came from the RCA Labs ([6]), whose automated vehicle was demonstrated on a public highway in Nebraska, and the United Kingdom's Transport and Road Research Laboratory, which developed a modified Citroën DS that went through a test track up to 130 km/h even in presence of

[1]Not to be confused with the magician Harry Houdini. However, even if the illusionist was not involved in the development of this almost magic car, he is somehow related to the story. Indeed it seems that the magician, bothered by the similarity between his name and the one of the engineer, protested violently in the office of the company causing him a summon to court ([4]).

Figure 3: Houdina's *American Wonder*

snow ([7]).

Premature works on visual navigation came in the late '60s from the students of the Stanford Artificial Intelligence Lab, that built a cart made of bicycle wheels and featuring a front-facing camera. The cart was initially able to follow a white line at a speed of 1.3 km/h ([8]); in the late '70s, the same cart was able to avoid obstacles[2] ([10]).

Real-time driving through vision arose with the group of Ernst Dickmanns of Bundeswehr University Munich in the late '80s, when they equipped a Mercedes-Benz van with cameras so that it was able to navigate in normal roads without traffic, reaching a maximum speed of 96 km/h ([11]); the system made use of saccadic vision, Kalman filters and parallel computing in order to cope with resource constraints of that time.

[2]The cart was able to plan 1-meter step every 15 minutes, and crossed a room filled with chairs in 5 (five!) hours without human intervention. The final version of the project included also a slider for the camera in order to enable stereo-vision. This and other interesting stories about the cart can be found in [9]

Figure 4: Test on the driverless car developed by *RCA Labs*, which used sensors to detect an electrical cable embedded in the road carrying warning signals

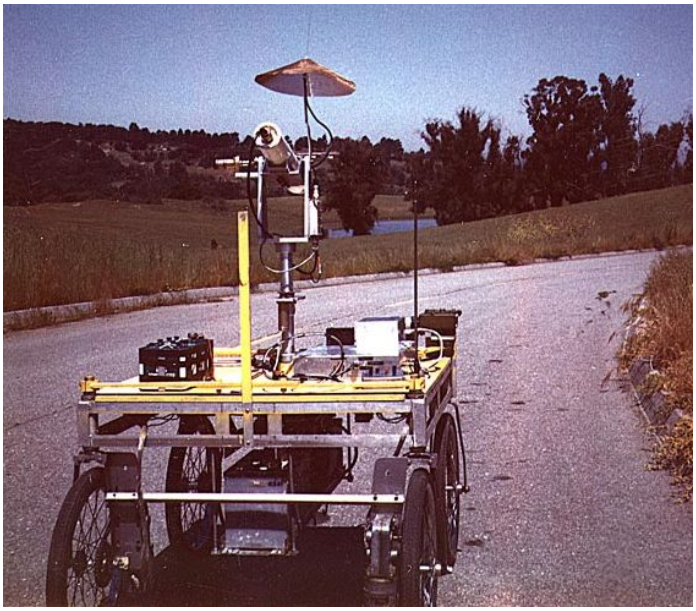


Figure 5: Stanford Artificial Intelligence Lab cart driving on the road around the laboratory

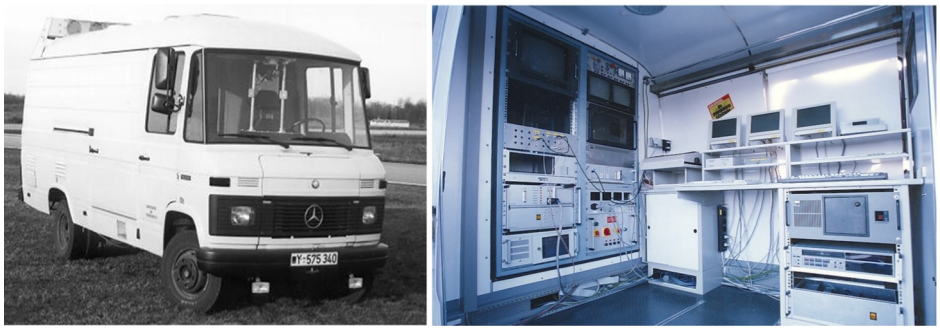


Figure 6: VaMoRs, the 5-ton van developed by the Bundeswehr University Munich and its not-too-portable interior

Future developments brought in mid '90s to an autonomous S-class Mercedes Benz able to navigate on highway traffic and address maneuvers such as lane change, overtaking and free lanes driving; this car succeeded on a 1678 km trip with around 95% of them in autonomous driving, reaching a maximum speed of around 175 km/h[3] ([12]). Meanwhile, in 1989, first steps on the use of artificial neural networks for autonomous vehicles control were made by Dean Pomerlau of the Carnegie Mellon University ([13]) in a land vehicle equipped with cameras and laser sensors. This efforts brought to a self-steering car which accomplished a 5000km journey dubbed *No Hands Across America* ([14]).

The University of Parma played an active role already in 1996, when, under the guide of professor Alberto Broggi, the ARGO project was presented, consisting on a modified Lancia Thema with two black-and-white cameras performing stereo vision. The project culminates with a 6-day journeys of 1900km along some Italian highways under the name of *Mille Miglia in Automatico* (*One Thousand automatic miles*), with the 94% of the journey in autonomous driving (the longest stretch being 55 km) and an average speed of 90 km/h ([15]).

Important contributions on autonomous vehicles came from the Darpa Grand Challenges ([16]) held between 2004 and 2007, which brought funding and interests on the field by many universities. In the first two competitions, vehicles had to finish

[3]This speed was achieved in a German Autobahn which does not feature general speed limit

Figure 7: Argo, the autonomous Lancia Thema developed at the University of Parma

an off-road, traffic-free course with no human interventions, while the 2007 edition was held in a simulated urban environment with real traffic.

From those days, with an increased attention on the topic and improvements on chips computational power, many research projects blossomed. The University of Parma showcased again important developments in the field thanks to the group called VisLab. In 2010, this group ran the *Vislab Intercontinental Autonomous Challenge* (VIAC), a 15900 km test drive from Parma to Shanghai that lasted 100 day. During this journey, an electric van using lane-keeping and obstacle avoidance techniques preceded another one which was performing vehicle-following algorithms ([17]).

The same group, in 2013, conducted *PROUD* (*Public ROad Urban Driverless*, [18]), a test in a urban scenario in which the vehicle had to negotiate junctions and traffic lights.

In the same year, Daimler and FZI/KIT Research Centers accomplished a 100 km autonomous drive on the historic Bertha Benz Memorial Route with a Mercedes-Benz S-class with close-to-production sensors ([19]).

Nowadays, many others car companies either own research groups or team up with self-driving start-ups, investing large amounts of money in what it seems every

Figure 8: Stanley from Stanford University, the winner of the 2005 Darpa Grand Challenge held in the Mojave Desert

Figure 9: Boss from Carnegie Mellon University, the winner of the 2007 Darpa Urban Challenge held in a simulated urban scenarios with traffic; vehicles had to avoid obstacles and negotiate intersections

Figure 10: The electric Piaggio Porter autonomous vans which took part to the Vislab Intercontinental Autonomous Challenge parade in the Shanghai Expo 2010

Figure 11: Vislab's autonomous car PROUD at the end of the public demonstration held in Parma in 2013

day more a race against the first, fully-autonomous car. While cars equipped with systems able to autonomously steer, accelerate and brake in 'simple' situations, such as highways or not crowded secondary roads, are already available on the market, the path towards cars designed to drive on public roads without humans behind the wheels and able to cope with every condition, is still long and uncertain: not only from a technological point of view, but also because of social acceptance and road regulation constraints ([20]).

0.3 Social impact of autonomous cars

As previously said, the aim of automation is to replace tasks which are generally done by humans; because of this, it has been argued that its extensive application in the industry will cause worldwide unemployment ([21]).

Since the beginning, automatic machines has been seen by many as a threat for their jobs by making their skills unnecessary and often drove to organized protests. Luddite ([22]), for instance, were members of a 19th-century band who started destroying textile machinery which were seen as the reason of the deterioration of their lifestyle; movements of this kind have arisen periodically ever since, even directed against self-driving cars ([23]).

Surely, automation is already contributing substantially to unemployment, especially in case of 'low-skilled' jobs; at the same time it has also been stated that the positive effects of technology will outweigh the negative ones of workers displaced by automation, paving the way to new jobs, better productivity and increasing the level of public services ([24]). While this debate will remain open for several years before receiving certain answers, some deeper consideration should be done for the self-driving domain.

It is out of doubts that the self-driving technology, when applied to taxi transportation or truck deliveries, will substitute the jobs of many human drivers. This should not be underestimated, and governments have to be prepared to support those people and find new employments. However, on the other side, the benefits of automatic driving may be of essential importance in many battles of the modern mankind.

First of all, nowadays being inside a vehicle is a danger for the life of its occupants. The number of people loosing their lives in the world's roads reached 1.35 millions in 2016 only ([25]): this means one person every 25 seconds. Even in high-income countries, where progresses are made in terms of legislation, vehicle standards and access to post-crash care, road traffic deaths are among the first causes of death for young adults.

Technology has the power to be of help in this battle: this was already clear to Charles Adler Jr. in the 1920s, when he proposed several ideas to improve automobile safety such as an automatic speed-control system able to regulate vehicles speed on dangerous location of the road like railroad crossing, intersections and streets with schools ([26]). Unfortunately, Adler was ahead of his time and his inventions often remained unheard.

Nowadays, car makers are introducing constantly new safety features (airbags, anti-lock brakes, electronic stability control, Advanced Driver-Assistance Systems), which are initially sold as additional features of their luxury lines and then often becomes part of the standard equipment thanks to the inclusion of new road regulations by the governments. Following this path and Adler's motto which says *"If they can save lives, I want everyone to have them"*, if the artificial driver will prove to be more reliable than the human and able to save lives, it will be mandatory to spread its capabilities.

Figure 12: Charles Adler Jr. showing one of his safety systems, which triggers a light when a car pass over a sound detector

Clearly, there are many difficulties along this path: at the beginning, self-driving systems will be affordable only by wealthy people and will have to be able to drive safely in roads populated mainly by human-driven vehicles, which are far less predictable than computers-driven ones; moreover, it will take time before this technology will be mature, and testing autonomous vehicles on public road will expose additional risks to road users (fatal accidents already happened [27]).

Furthermore, autonomous vehicles may enhance the freedom of moving independently for many people that are now elder or hampered by disabilities and health restrictions, which could be the cause of reduced economic opportunities or isolation that bring a decrease on their quality of life ([28]): reducing this problem will be an enormous success in our societies. In addition, also people lacking license would benefit of this improved mobility. Possibly in the future, when the technology will be evolved enough so that no human attention is required in the process of driving, people could spend the time that was once used to drive at will: this amounts, in average, to

around 4 years for a person in Europe ([29]).

Finally, self-driving cars may play a not-marginal effect on the environmental impact played by means of transportation. Artificial drivers might feature an efficient driving style, avoiding useless braking and acceleration, and they will be able to follow cost-effective routes. However, easiness of transportation coming from driverless cars could also increase the total number of kilometers traveled by vehicles, counteracting the positive effects coming from an increased efficiency ([30]). An ingenious solution to avoid this negative implication, together with awareness-raising of people to favor cycling or walking when possible, could potentially come from the ride-sharing possibility: private cars, which generally stay 95% of their lifetime parked ([31]), may no longer exist in cities and could be replaced by a fleet of autonomous vehicles designed for combined transport of people. Their job will be to tirelessly pick up people at request and take them to their destination; when some of these cars are not needed or need maintenance they could go autonomously to a strategic deposit, leaving our streets empty from parked vehicles. I leave the reader to imagine how our cities would be like without parked cars on every corner..

Chapter 1

Self-driving Basics

Autonomous vehicles will not pop up on roads all of a sudden once they will be ready to solve the full problem, namely to be able to cope with every situation without human interventions; indeed, cars will be gradually featuring more advanced algorithms and sensor suites while the technology keeps maturating.

Somehow, this process was already started with safety equipments which helps the human in some specific situations, as with the Anti-lock Braking Systems (ABS) and the Electronic Stability Control (ESP). As today, it is common to meet on the roads vehicles capable of autonomously parking, following lanes and executing overtakes on highways: the range of possible applications will be constantly extended as the degree of caution needed for assuring safe behaviors will decrease.

In this chapter, a very brief overview of the broad self-driving domain is given, together with an introduction on the main topics which has to be solved in order to obtain a complete self-driving vehicle.

1.1 Levels of driving automation

The organization Society of Automotive Engineers (SAE) has established a taxonomy regarding automated driving systems, so that the degree of autonomy of a vehicle is mapped in a range from 0 to 5:

- **Level 0 - no automation**: the full-driving task is executed by the human driver, even when enhanced by warning or intervention systems; this means that even cars equipped with a lane-departure warning or driving fatigue detection fall in this category;

- **Level 1 - driver assistance** (*hands on*): the control of the vehicle can be shared between the driver and the automated system, which is capable to control either steering or acceleration/braking functions, but not both; an example is a car equipped with Adaptive Cruise Control (ACC);

- **Level 2 - partial automation** (*hands off*): the system can take the full control of the vehicle in specific situations, but the driver must be prepared to intervene immediately at any time in case of improper behaviors; Tesla autopilot ([32]) is considered to belong to this level.

- **Level 3 - conditional automation** (*eyes off*): if the conditions are favorable, the driver can give full control to the system and turn its attention to tasks which does not involve driving because it is not needed its immediate intervention; the vehicle will handle every anomaly, but still the driver has to be prepared to intervene within some limited time;

- **Level 4 - high automation** (*mind off*): the driver does not have to be prepared in order to ensure safety since the vehicle is able to safely stop the trip if no human intervenes, making a human driver behind the wheel unnecessary while the system is on; this capability is supported only in specific situations, like geofenced areas or traffic jams;

- **Level 5 - full automation** (*steering wheel off*): no human driver is required at all; every aspect of driving can be handled autonomously under every scenario.

As of 2019, no Level 4 (or more) vehicles are available on the market, and Level 3 of autonomy is reached only in very structured situations such as traffic jams or restricted areas ([33]).

1.2 Sensors

A variety of different sensors are available in order to provide multiple types of information useful for understanding the environment around the self-driving vehicle, each one with its strengths and weaknesses.

1.2.1 Cameras

Images captured with cameras provide a dense input which can be potentially used to compute many of the data needed to solve the problem of driving, from free space detection to obstacle recognition. However, this rich representation is often problematic to process: vision is strongly dependent on weather conditions (rain, fog, etc.) and light changes (night, sun's glares, etc.), undermining the reliability of these sensors and requiring advanced robust algorithms. Cameras can also be used in pair to enable stereo vision, thus giving the possibility of extracting precise 3D measurements.

1.2.2 Radars

Radars (RAdio Detections And Ranging) work by emitting radio waves in predetermined directions and analyzing the reflected signal, so that both distance and speed of objects can be estimated. Radars have no problem in bad weather conditions or at night, and can achieve long operating distances. At the same time, radar detections are strongly dependent on the reflection strength of objects, making metallic target such as cars much more easy to detect than pedestrians; moreover, the definition of the detections is coarse-grained due to the relative long wavelength of the emitted radio signals, thus making difficult to estimate the shape of the detected objects.

1.2.3 Laser scanners

The principle behind laser scanners is the same as of radars, but instead of radio signals the transmitter emits laser pulses whose wavelength is much shorter, permitting to build an exact three-dimensional representation of the scene including small objects and precise shapes. However, light signals are not as robust as radar against bad weather

conditions, such as fog and heavy rain, reducing the reliability of the information. Furthermore, these devices are very expensive at the moment; for this reason, some developers of self-driving vehicles are building systems which does not equip laser scanners.

1.2.4 Global Navigation Satellite System

Global Navigation Satellite Systems (GNSS), of which GPS is the most famous, are systems which makes use of satellites to provide absolute positioning. The accuracy can be relatively high, reaching 30 centimeters in case of GPS. However, even when paired with augmentation services aimed to increase the accuracy, the precision downgrades rapidly when used in urban scenario, due to the presence of buildings and other reflective obstacles around the vehicle.

1.2.5 Inertial Navigation System

Inertial Navigation Systems (INS) are devices which continuously calculate position, orientation and velocity of the vehicle using sensors, such as accelerometers and gyroscopes, which does not require external references; therefore, their information is accessible even when there is no signal from satellites, for example when the vehicle is situated inside a gallery. Since the output is computed by dead reckoning, namely using previously determined output, the process is subject to cumulative errors; for this reason, coupling GNSS and INS is a typical solution used to reach robust position estimation.

1.2.6 Ultrasonic transducers

Ultrasonic sensors are a type of Sonar (SOund NAvigation Ranging) that, as radars and laser scanners, works by emitting pulses and analyzing the reflected signal; however, they use the propagation of the sound instead of that of electromagnetic waves, with a frequency which is higher than that audible to the human ear. Their field of use is limited to specific situations such as parking assistance, due to the slower propagation of sound which makes their signal not timely enough for critical applications.

1.3 Self-driving tasks

The entire problem of autonomous navigation can be disentangled in several different subproblems, each of them requiring a dedicated approach and proper tools to be solved; a simplified pipeline is shown in Figure 1.1, but some of the tasks may be unnecessary or additional subtasks may be required depending on the design of the system.

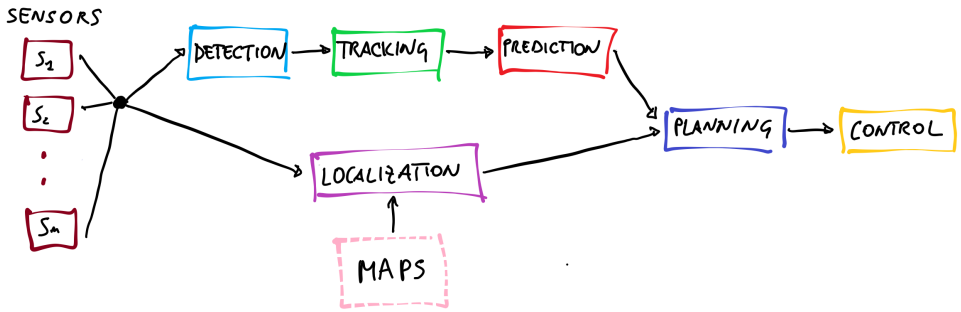


Figure 1.1: Pipeline of subtasks which constitutes the full autonomous navigation problem

1.3.1 Detection

Detection, or perception, aims at extracting all the useful information from the latest output of the sensors, for both obstacle detections and navigation. One of the goal of detection is to recognize and localize other road occupants, such as cars and pedestrians, and environmental features, such as lane markings, traffic signs and curbs. Moreover, high-level information of crucial importance may be also deduced: free-space detection, namely the detection of space in which is possible to navigate, and the identification of lane-centers of the road, are two important examples.

1.3.2 Tracking

Tracking deals with grouping together those detections of different time intervals which belong to the same object, so that the trace followed by an object can be fully

reconstructed. This permits to highlight which obstacles are moving and which are static, making it possible to infer their future positions.

Tracking gives also an additional benefit: past motions of an object can be used to refine those detections which are not correct, because of either imperfections on the perception system or occlusions in the detections; similarly, false positive detections which have no correlation with the past may be excluded.

1.3.3 Prediction

Once the past positions of an obstacle are known, the next step can be nothing else than predict its future positions. In fact, knowing where all the other road users will be in the future gives the possibility of planning a safe path which avoids collisions and dangerous situations. However, perfect prediction of future poses is not possible, since human moves generally in a non-deterministic manner: for this reason, it is necessary to take into account uncertainties on the estimations.

There are works in the literature that goes further the simple estimation of future position of cars or pedestrian based on the past history, and try also to infer the intention of people by looking at high level features such as their gaze directions ([34]), hand gestures ([35]) or mutual relations ([36]).

1.3.4 Localization

The aim of localization is to estimate the position of the vehicle within few centimeters, that is less than what Global Navigation Satellite Systems could give. For this reason, specific algorithms need to be implemented to flank GNSS, which may use both detected features of the environment and odometry information coming from inertial navigation systems.

An important aid to the task can be given by high definition maps (*HD maps*), which are maps that include not only the topology of the road network, but also additional precise information like the position of lane boundaries, curbs, traffic signs and landmarks, which can be matched to the information coming from the perception system for triangulating the position of the vehicle. The use of HD maps is very

attractive, since makes the localization problem more tractable; however, it poses additional challenges and limitations: recording this kind of maps is an heavy task since they need to be constantly updated to follow environment changes; moreover, a vehicle strongly dependent on precise mapping will not be able to navigate in unmapped environments.

1.3.5 Planning

The goal of a planner is to design a path made of the desired positions of the vehicle at future time instants such that the proposed trajectory will be feasible and avoids collision or road rules violations. The output of the planner strongly depends on the estimations coming from the precedent modules: a good prediction on the position of the other road occupants is essential, and its uncertainty has to be taken into account so that the risk of undesired situation is minimized; at the same time, a precise localization inside the environment is fundamental for a safe navigation.

Implicitly or explicitly, the planner must to take into account the sequence of the high-level maneuvers that the vehicle has to follow, such as lane changes, overtakings or insertions on intersections.

1.3.6 Control

The purpose of the control phase is to translate the desired set points coming from the planner to commands for the actuators controlling the vehicle, that are those controlling acceleration, breaking and steering angle. A good controller will provide a smooth and comfortable driving in ordinary situation, as well as rapid and efficient commands when a danger occurs.

1.4 Approaches to self-driving

As previously said, the tasks needed to be solved in order to develop an efficient autonomous vehicle are well defined. Despite this, the possible approaches that can be followed to tackle them may be very different.

1.4.1 Traditional stack

The traditional approach consists on developing separately the modules which solve each different task, whose output are then linked together in order to form a pipeline like the one shown in Figure 1.1. Each module is trained independently from the others, so that a specific objective is optimized; possibly, each of them would be developed by a different group of people specialized in that precise task.

When adopting this approach, the output of the full system is explicable, since allf the intermediate outputs can be analyzed and errors easily detected. However, having independent modules means that a mistake of a single module is sufficient for causing problems on the final output, because each module is allowed to have as input only the output of the precedent one; for the same reason, a win-in-one module does not necessary translate to a win-in-overall performance. Moreover, having different objectives makes uncertainties difficult to propagate, making difficult to evaluate the final output of the system.

A further drawback is the resource efficiency: indeed, computation is not shared between modules, making the full system generally heavy to run. Finally, additional work may be needed when the different modules will be put together in order to let adjacent blocks communicate successfully.

1.4.2 End-to-end learning

A completely different approach is end-to-end learning, in which a unique system is responsible for all the tasks as a whole. Ideally, this module learns a perfect mapping between the raw sensor input and the control outputs of the vehicle by applying gradient-based learning to a dataset containing expert demonstration.

The module is generally represented by a neural network, so that the best features for the task are automatically learned. The first appearance of a solution of this kind dates back to 1989 ([13]), while a recent leap forward was shown by NVIDIA ([37]). This approach is very tempting, since it reduces the problem of solving all the navigation tasks to the single problem of collecting a good dataset. Moreover, a system of this kind would be extremely time-efficient, since the time needed to

compute commands for the vehicle will be only those required for a forward pass of a neural network.

However, there are two main issues which affect the end-to-end approach. Firstly, its performances are strictly related to the goodness of the dataset, and it would strongly deteriorate in situations which are outside the training demonstrations: even if the dataset would seem very large and varied, it is not possible to store all the possible corner cases which may be encountered while driving, and additional efforts would be require for tackling the generalization capabilities of the system. Secondly, the output will be not explicable, since there are not intermediate output which helps on understanding how the system reasoned for compute that precise output. This introduces challenges on understanding what went wrong, bringing difficulties on estimating the uncertainties and guaranteeing the safety of the system.

1.4.3 Interpretable end-to-end learning

Recently, an hybrid approach has been proposed in order to combine the advantages of the end-to-end approach and the interpretability of the traditional stack ([38]). This method aims at building an holistic model able to solve several of the tasks needed for the autonomous navigation by learning from complete human trajectories with annotated ground truth examples for each task, so that intermediate outputs can be provided and used for the computation of the loss function which has to be minimized. This way, the full stack can be trained from representation in one single pass and still its decisions can be understood by analyzing the output of each subtask.

The idea is that all the components of the model will be optimized for reaching the final goal instead of maximizing their isolated performances; as example, the detection phase may give more importance to detect close vehicles which are interpreted as crucial for a safe navigation. This is in contrast with traditional accuracy metrics used for systems dedicated to obstacle detections which give equal importance to every vehicle independently from the global task.

Chapter 2

Deep Reinforcement Learning

As the name suggests, Deep Reinforcement Learning is the fusion of two broad areas of Machine Learning, namely the well-studied Reinforcement Learning Theory with the new achievements in the Deep Learning for function approximation.

In this chapter, both these topics are introduced in their guidelines, so that the specific themes of this thesis can be better understood. However, I let to dedicated books (such as [39] and [40]) the goal of clarifying all the aspects of these very interesting subjects.

2.1 Machine learning

Machine learning is a subfield of artificial intelligence in which knowledge is acquired through data, observations and active interaction with the world; the goal is to use this knowledge to interpret correctly new data, namely to generalize, so that what it has been learned from data can be successfully deployed in new situations. The objective of a machine learning algorithm can be of different types, but the majority of the models can be seen as regressors or classificators.

In regression, the goal is to unveil the relationship among different variables: generally speaking, the interest is the estimation of the value of a subset of the variables dependently on that of the others, which constitute the input of the system.

Thus, regression output is typically continuous and not restricted to predetermined values. An example of this kind could be the prediction of the amount of rain that will fall tomorrow by using satellite information about cloud placement and type. Similarly, a regression problem related to this work is determining the acceleration and steering values of a car in order to avoid accidents, giving the position of the other cars and the point that has to be reached.

Classification, on the other hand, aims at identifying which category is described by the input values. Contrarily to regression, the cardinality of the categories is finite, and there should be examples in the training set for each one of them. Instead of estimating the amount of rain that will fall tomorrow, the goal of the classifier would be to predict if it will be a rainy rather than a sunny day. Similarly, in an autonomous vehicle the job of a classifier could be to tell to the planner when beginning to cross an intersection rather than waiting for another car to pass.

2.1.1 Type of learning

Depending on the type of task and the data at disposal, different classes of learning algorithms can be used.

Supervised learning

In the supervised learning setting, the available data are pairs of observations and labels, where observations are sample of the input variables and labels are the expected output related to the observations. In a regression problem labels would represent the value of the set of variables to be estimated, while in a classification task would be the index of the category of the input. Intuitively, the larger the dataset, the more robust the system will be; unfortunately, large enough dataset are often impossible or very expensive to collect. One of the solutions, as will be explained more in detail throughout this thesis, is the use of simulators which try to mimic real world data synthetically.

Other means to extend the dataset are using the so called semi-supervision or weak supervision: in the semi-supervised scenario, the observation-label pairs are

accompanied by unlabeled observations which are used to extract more information and possibly improve the learning efficiency; weak supervision, instead, is the use of low-quality training data, which can be imprecise or noisy, but much more efficient to collect, as in the case of data labeled by heuristic rules instead of by hand.

Unsupervised learning

Contrarily to supervised learning, in the unsupervised setting no labels are provided for the observations: instead of learning a mapping between input and output, the model objective is to find hidden structures in the data. Clustering algorithms, such as *K-Means*, fall into this category.

Reinforcement learning

Reinforcement learning, as better explained in the next section, stays in between the unsupervised and supervised scenarios: while there is not a specific label for the observations, it is available a feedback signal that is correlated with the output given by the system, which in this case are the actions taken from an agent interacting with an environment. This signal can be the result of several actions, some of which could have been taken even way back in the past but can still be used to learn a mapping between observations and optimal actions.

2.2 Reinforcement Learning Theory

Human beings does not learn to accomplish tasks by receiving constant feedbacks for their actions by some experts; little infants even does not have a well defined language to communicate with their parents. Nonetheless, they learn to crawl and to waves their arms; children learn to play sports and to whistle just following their senses and talents. Growing they learn to run bicycles, holding conversations and deepen their skills, often without any trainer aside, for the rest of their lives.

What it drives them is the delight on seeing their desires come true: for the little infant that would be reaching that beautiful toy 10 meters away or receiving food

from its mum, a child will feel great when it become important for the team; some people feel good when they reach outstanding careers or when they know how to make people laugh. Of equal importance are the advices and suggestions received from the others, especially if they are better than us or experts in that field; at the same time, disappointments, failures and negative events are extremely useful to decide what to do in the same circumstances in the future.

In all that situations, there is not a unique and absolute way of acting in order to satisfy everyone's dreams, and the same goals could be reached by different paths: as example, a new language could be learn either by looking TV-series or by traveling abroad. In a few words, we are driven by the interactions with the environment in which we are leaving and by the effects that these interactions raise inside us.

The same idea underlies *reinforcement learning*, in which a learning agent is free of taking actions inside an environment. The agent is not told which action is the best or which it should be avoided, neither which are its effects. All it has is a numerical reward signal which generally gives an idea about the goodness of the actions taken until that time, like the feelings of the baby when he reach the toy or when a child succeed on scoring a basket; this signal may also be an indicator of a bad ending of a strategy, like the pain a child receive when she falls off the bike. Thus, the goal of a reinforcement learning agent is to maximize the sum of future reward signals that it is going to receive.

Now it is clear that, like in real life, the interaction with the environment is crucial in this scenario. The action taken at a specific time instant may not lead to an immediate reward, but still be fundamental to obtain good rewards in the future, since it will modify the agent conditions or the environment itself. This setting brings to the two main distinguishing features of reinforcement learning, namely the need of exploring solutions with the risk of making mistakes, called trial-and-error search, and the fact that the only supervision we have, the reward, may be available only in the future and only in particular occurrences. This setting differs from supervised learning, in which each sample of the dataset is a full descriptions of the situation, and its label tells exactly what the agent should do in that. At the same time, reinforcement learning does not match with what is called unsupervised learning: while it is true that there

is no labels for the observations, there is still an indicator of the behavior coming from the rewards; the goal indeed is to maximize them instead of discovering hidden structures in the data.

The theory pills given in the following are written taking inspiration mainly from the well-documented book from Richard Sutton and Andrew Barto called *Reinforcement Learning, an Introduction* ([39]).

2.2.1 Exploration versus exploitation

As previously explained, in order to efficiently solve reinforcement learning problem it is fundamental to be curious: trying unknown paths, even if they didn't seem promising and could lead to mistakes, may be the key for discovering the best solutions. This because the goal is to maximize rewards in the long run, and sometimes this implies to wait for the hen instead of taking the egg immediately.

Ideally, the optimal solution will be exactly found if all the possible paths were previously tried (several times in case of stochastic tasks): indeed, it would be sufficient to store the rewards obtained for each of them in a look-up-table and retrieve the one which will lead to higher expected rewards.

However, apart from very simple problems, this strategy would be infeasible, since the number of possible paths is tremendously high, if not infinite. For this reason it's important to focus the exploration toward areas that proved to be potentially good, and avoid those which led to catastrophic results; this means that it is needed what is generally called exploitation, namely the use of what the agent has already experienced in the past in order to reach prolific configuration of the environment.

Now it is clear that a balance between exploration and exploitation is essential to succeed in the task: pure exploitation will not unveil actions which are better than expected, while pure exploration would be extremely inefficient. The right proportion of this balance is a long-standing yet unresolved problem.

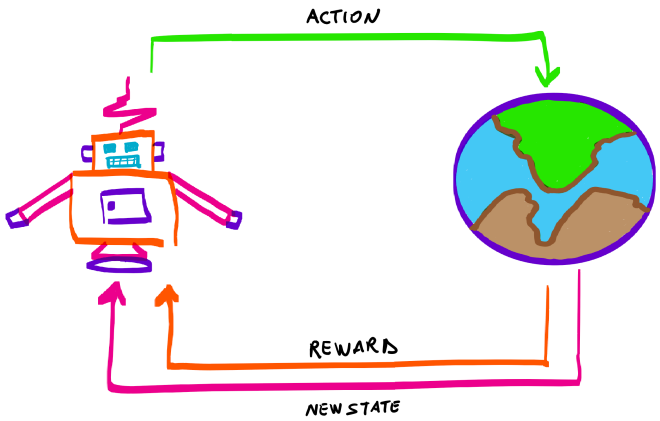


Figure 2.1: Cycle of a Markov Decision Process: the agent take actions based on its environment observations; as a consequence, it receives a reward signals and the new environment configuration

2.2.2 Markov Decision Processes

The sequential decision problem tackled by reinforcement learning algorithms, in which the task is to learn by interaction how to achieve a goal, can be generally formalized as a *Markov Decision Process* (MDP, [41]). In this mathematical framework, the decision maker, which is called the *agent*, takes actions inside an *environment*, which comprises everything outside itself. This interaction can be viewed as a cycle, as shown in Figure 2.1: the agent executes an action chosen by looking at the environment (or part of it in case of partially observable MDPs), and it will possibly receive a numerical feedback that we already called *reward*. Then, a new round can start, in which the agent will interact with the new configuration of the environment, which could have been modified by the action of the agent or its own natural evolution.

Going mathematically, the action a_t, taken at time t, is drawn from the range of action A, which could be either discrete or continuous; the agent chooses a_t by receiving an observation, eventually partial, of the configuration of the environment at that time, namely its *state*, referred to as s_t, which is part of the set of possible states S.

The subsequent time step, potentially as a consequence of the action taken, the agent receives the reward r_{t+1} and finds itself in the new state s_{t+1}.

An MDP, to be such, has to be memory-less and so guaranteeing the validity of the so called *Markov property*: namely the next state of the environment has to be dependent only on the current state and the action taken from the agent, that is $s_{t+1} = f(s_t, a_t)$. Goal of the agent is to maximize the expectation of a function of its future rewards, the *expected return*, which in its basic form is defined as:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T = \sum_{k=t+1}^{T} r_k \tag{2.1}$$

where T is the final time step which ends the trial of the process, either because the goal is reached or a terminal event happened. However, that definition makes sense only when the learning process is divided into episodes of finite length, namely episodic tasks; indeed, reinforcement learning can be applied also to continuing tasks which does not break into episodes, such as the control of a power plant or a robot throughout its life. In continuing tasks, the value of equation 2.1 could be unbounded, since T would be infinite; for this reason, it is generally used the expected *discounted return*, which aims at regulating the importance of future rewards in the current time step:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-t-1} r_T = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \tag{2.2}$$

where γ is called *discount rate* and its value is between 0 and 1. If γ is close to 0 the agent will be mainly interested on maximizing immediate rewards, while, if γ is close to 1, the agents will increase its concern on rewards far in the future. Discounting, besides permitting to tune the agent concerning for the future, makes the reward finite even in continuing tasks; in fact, considering a constant reward of +1 at every step:

$$R_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} r_k = \sum_{k=0}^{\infty} \gamma^k r_k = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma} \tag{2.3}$$

which is a finite value when γ is smaller than 1.

The agent chooses its action following what it is called its *policy*, which simply is a mapping between states and actions. The policy can be either deterministic, in which every state is associated to a specific action, or stochastic, where the mapping is between states and action probabilities (or probability densities in the continuous-action case).

2.2.3 Value functions

An important role in reinforcement learning is played by the *value functions*, which are functions intended to tell the goodness of states and actions when the agent follows a specific policy. Calling $\pi(a|s)$ the probability of taking action a in given the state s, the *state-value function* (2.4) of π gives the expected return when, starting from state s, the policy π is followed:

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

The state-value function permits to rank policies: we can say that policy π_1 is better than π_2 if $V_{\pi_1}(s) > V_{\pi_2}(s)$ for every s; for this reason we call *optimal policy* the policy that is better or equal to all other policies, namely the policy π^* for which $V_{\pi^*}(s) = \max_\pi\{V_\pi(s)\}$ for every s.

Similarly, the *action-value function* (2.5) gives the expected return when the agent follows the policy π after having taken action a in state s:

$$Q_\pi(s_t, a_t) = \mathbb{E}(R_t|s_t, a_t) \tag{2.5}$$

The difference is that the action-value function gives the expected return for every possible actions taken in s, while the value function is a property of the state itself. This means that if the *optimal action-value function* Q_{π^*}, that is the action-value function of the optimal policy, would be known, it would be possible to follow the optimal policy simply by taking action $a_t = \max_{a\in A}\{Q_{\pi^*}(s_t, a)\}$. Trying to estimate this function is the goal of those algorithms which fall under the category of *action-value methods*, as it will be explained in sections 2.2.6 and 2.2.7.

An useful tool for estimating the value functions are the so called *Bellman equations*, which permit to describe the value of a state (or a state-action pair) iteratively in terms of the value of its successor states (or state-action pairs). The Bellman equation for the state-value says:

$$V_\pi(s_t) = \mathbb{E}(R_t|s_t) = \mathbb{E}_{a_t}(r_{t+1}) + \gamma\, \mathbb{E}_{a_{t+1},s_{t+1}}(V_\pi(s_{t+1})) \tag{2.6}$$

Similar considerations bring to the Bellman action value function:

$$Q_\pi(s_t|a_t) = \mathbb{E}(R_t|s_t,a_t) = \mathbb{E}(r_{t+1}) + \gamma\, \mathbb{E}_{s_{t+1}}(Q_\pi(s_{t+1}|a_{t+1})) \tag{2.7}$$

2.2.4 Model-based versus model-free

Equations 2.6 and 2.7 average over the probability distributions of the immediate reward r_{t+1} and the future state s_{t+1}. If those distributions, which are called the *model* of the environment, would be known to the agent, it would be possible to find the optimal policy exactly, since the problem would assume a closed form.

Nonetheless, the cardinality of the state space typically makes this closed form evaluation computationally infeasible; for this reason, iterative methods based on dynamic programming are more suitable in these situations.

However, in the majority of cases the dynamics of the environment completely unknown to the agent, and different approaches have to be considered. In *model-based* reinforcement learning, the model is explicitly learned by the agent by observing the effects of its actions to the environment. One of the advantages of this approach is the possibility to inject prior knowledge of these dynamics inside the model and exploit more extensively the experience of the agent; this translates into an increased sample efficiency of the solution, that means diminishing the number of trial-and-error search which the agent has to perform before reaching a satisfactory policy.

Another approach is the *model-free* reinforcement learning, in which the agent does not take into account explicitly how the environment will evolve and it learns directly to predict the best actions given a state of the environment; this does not mean

that the agent would be unaware of the effects of its actions, since the environment dynamics are implicitly considered on the policy it is optimizing.

While model-based learning is more sample-efficient, it is also more computationally demanding, due to an larger state space which includes also the to-be-learned dynamics. Since in this research it has been developed a simulator where agents can be trained and able to generate large quantities of data, the focus has been directed toward model-free approaches which guarantee higher computational efficiency.

2.2.5 On-policy versus off-policy methods

As discussed in subsection 2.2.1, the agent needs to make mistakes in order to discover better policies: indeed it is trying to learn the optimal policy while continuing to explore new strategies, meaning that sometimes the action taken is not what it expects to be the optimal one.

On-policy approaches tackle this issue by letting the agent exploit its current knowledge most of the time (exploitation), while in the remaining part it selects actions which are not thought as the best actions (exploration); the agent is continuously updating its policy, which needs to be followed in order to evaluate directly its results. This means that the agent is learning action values not for the optimal policy, but rather for a near-optimal policy that still explores. Several strategies can be used to select exploring actions: the simpler one is called $\varepsilon - greedy$, and it consists on acting greedily in the $(1-\varepsilon)\%$ of the cases while sampling actions randomly with equal probability in the remaining $\varepsilon\%$; other techniques, such as *upper confidence bound* (*UCB*, [42]) action selection, gives different priority to actions based on their expected values and related uncertainties.

Since the agent has to exploit its current knowledge, on-policy methods have to be applied online, making it not possible to learn from a recorded set of state-action pairs obtained following a different policy from the one which is currently updated.

Off-policy methods follow a different path: in this approach, behavioral and target policies can be different, since the experience obtained by the behavioral policy are translated into updates for the optimal policy estimation; this way the agent is learning the target policy while following a more exploratory one. Since the two policies does

not match, off-policy methods tends to have greater variance and slower convergence rate; moreover, they generally need additional precaution such as importance sampling ([43]) or deterministic target policies ([44]).

However, off-policies approaches learn directly the true optimal policy and give the freedom of learning offline, as from human experts or recorded data in which it was not possible to influence the behavioral policy.

2.2.6 Monte Carlo methods

Monte Carlo methods is a family of algorithms for solving reinforcement learning problems which is based on averaging the returns obtained by the agent in order to estimate the value functions, which permit to predict future returns and estimating the best actions. Since returns are needed to update the estimation, this type of approach is valid only for episodic tasks, namely all episodes have to eventually terminate at some point. Once the terminal state is reached and all the rewards obtained throughout the episode are stored, it is possible to update the estimation of the value functions in all the visited states.

An example of on-policy Monte Carlo algorithm is shown in Algorithm 1, in which the policy being optimized is of type $\varepsilon - greedy$. In this approach, the credit of large rewards obtained during the episode is spread across all the actions taken from that time until the beginning of the episode; this means that this method is somehow more suitable for environments where many actions contribute to the goal achievement, such as problems with inertia in which is needed a repeated sequence of similar actions (e.g. the control of the water flow of a dam) instead of problems in which a precise action, or a short set of actions, is required at some point.

2.2.7 Temporal-difference learning

Temporal-difference (TD) learning algorithms, unlike Monte Carlo ones, are not forced to wait the end of the episode before updating the estimation of value functions. Indeed, instead of using the true final returns, they use estimated values of future states together with recent obtained rewards as a 'fresher' estimation of the optimal value functions.

Algorithm 1 On-policy Monte Carlo algorithm using $\varepsilon - greedy$ policy

initialize discount factor γ
initialize $\varepsilon - greedy$ policy $\pi(a,s)$
initialize action value function $Q(s,a)$ for all $s \in S$, $a \in A$
initialize $returns(s,a)$ as a empty list
for every episode lasting from $t = 0$ to T **do**
    behave following π generating sequence: $(s_0,a_0,r_1),...,(s_{T-1},a_{T-1},r_T)$
    $R = 0$
    for $t = T-1, T-2, .. \ 0$ **do**
        $R = \gamma \cdot R + r_{t+1}$ {with γ discount factor}
        append R to $returns(s_t,a_t)$
        set $Q(s_t,a_t)$ as the arithmetic mean of $returns(s_t,a_t)$
        $a^* = argmax_a\{Q(s_t,a)\}$
        for every action a **do**
            $$\pi(s_t,a) = \begin{cases} 1-\varepsilon+\varepsilon/|A| & \text{if } a = a^* \\ \varepsilon/|A| & \text{if } a \neq a^* \end{cases}$$
        end for
    end for
end for

This technique of learning 'a guess from a guess' is called *bootstrapping*.

It is believed that for several real-world problems, temporal difference methods require less memory and computation complexity and produce more accurate prediction than Monte Carlo methods ([45]).

Since real rewards are needed only for the immediate future, this approach is perfectly suitable for continuing task which does not have a terminal state. Algorithm 2 shows the pseudo-code for *Q-learning* ([46]), an off policy TD-learning algorithm which uses the estimation of the action-value function of the immediate subsequent state and the reward obtained in order to update the action-value of the present state.

Algorithm 2 Off-policy TD-learning algorithm (*Q-Learning*)

initialize discount factor γ and learning rate α
initialize desired behavioral policy $\pi_b(a,s)$
initialize action value function $Q(s,a)$ for all $s \in S$, $a \in A$
for every episode **do**
  get initial state s
  repeat
    for each step of the episode
    execute action $\hat{a}$ drawn from $\pi_b(a,s)$
    get reward $\hat{r}$, state $\hat{s}$
    $Q(s,\hat{a}) = Q(s,\hat{a}) + \alpha \cdot [\hat{r} + \gamma \cdot \max_a Q(\hat{s},a) - Q(s,\hat{a})]$
    $s = \hat{s}$
  until $\hat{s}$ is terminal
end for

In this solution, the effects (namely, the reward) of each single action will be directly associated to it, since it involves the comparison between the expected rewards before and after this same action: this means that, in contrast to Monte Carlo learning, short term dynamics will be ideally learned faster (e.g. finding which one out of a set of levers gives higher reward). At the same time, Q-learning would struggle if the reward of a specific action is delayed in time: in fact, all the intermediates state-action pairs have to be updated before that the value of the action which really caused that

reward would be affected.

It might be possible that neither Monte Carlo Learning nor Q-Learning matches well with the dynamics of a particular task. In order to span between these two extremes it is sufficient to augment the bootstrapping interval, leading to the so called *n-step* methods, as the one that will be shown in Algorithm 3.

2.2.8 Policy gradient methods

Both Monte Carlo and TD-learning methods estimate the value functions, which are then used for the action selection, that is for shaping their policies. Policy gradient methods, instead, learn directly a parametrized policy, which does not need to take into account the value functions to be used.

This approach may give advantages in some peculiar situations. In fact, when the action space is continuous, using the action-value function as a guide for the policy requires an optimization over a continuous function at every time step, which is likely to be very inefficient. Moreover, maximizing the action-value function would not be optimal also in those cases in which the best policy is stochastic: it would be very easy to win at *rock-paper-scissors* against a deterministic opponent after few matches...

The simplest example of a policy gradient algorithm is *Reinforce* ([47]), an on-policy algorithm in which the parameters of the policy are updated so that probabilities of actions that yielded a better return at the end of an episode are increased with respect to the lower-return actions. Parameterizing the optimal policy $\pi^*(a|s)$ with a function $\pi(a|s,\theta)$, the updates are:

$$\theta_{t+1} = \theta_t + \alpha \cdot R_t \cdot \frac{\nabla_\theta \pi(a_t|s_t,\theta)}{\pi(a_t|s_t,\theta)} \tag{2.8}$$

The equation states that each increment takes the direction of the gradient of the policy in the parameter space, which corresponds to the direction that most increases the probability of the action taken (a_t) in the visited state (s_t), and is proportional to the obtained return. At the same time, it is normalized by the action probability, in order to avoid the risk that actions which are selected more frequently win out even if they do not yield the best return.

2.2.9 Actor-Critic methods

Between action-value methods, which build the optimal policy upon value functions as in case of Monte Carlo or TD-learning, and policy methods, which directly estimate the optimal policy, there is a class of algorithms called *actor-critic*.

In this approach, the policy is still directly optimized and does not depend on the value functions, as in policy gradient methods; however also the state-value function are estimated, so that a double benefit is obtained: firstly it helps in reducing the variance of the updates ([48]); furthermore, it permits bootstrapping, thus allowing the use of TD-learning techniques for the estimation of the expected return as well as the use of this methods in continuing tasks.

In the on-policy version called *Advantage Actor-Critic*, instead of using the return R_t as in case of Reinforce, the probability of actions is modulate with the *Advantage*, which is an estimation of the benefits obtained following the executed actions with respect to the reward that it would have been obtained following the current policy:

$$A_b = r_t + \gamma r_{t+1} + ... + \gamma^{b-1} r_{t+b-1} + v(s_{t+b}, \theta^v) - v(s_t, \theta^v) \tag{2.9}$$

where b is the number of obtained rewards taken into account before bootstrapping. Calling the parameterized policy $\pi(a|s, \theta^\pi)$ and state-value function $v(s, \theta^v)$, the updates of these two functions can now be defined as:

$$\theta^\pi_{t+1} = \theta^\pi_t + \alpha_\pi \cdot A_b \cdot \frac{\nabla_{\theta^\pi} \pi(a_t|s_t, \theta^\pi)}{\pi(a_t|s_t, \theta^\pi)} = \theta^\pi_t + \alpha_\pi \cdot A_b \cdot \nabla_{\theta^\pi} \log \pi(a_t|s_t, \theta^\pi) \tag{2.10}$$

$$\theta^v_{t+1} = \theta^v_t + \alpha_v \cdot A_b \cdot \nabla_{\theta^v} v(s_t, \theta^v) \tag{2.11}$$

in which the first one is the direct evolution of Equation 2.8 and the second one simply updates the parameters of the state-value function estimation so that the advantage converges toward zero.

Algorithm 3 shows a n-step Advantage Actor Critic algorithm, in which both long and short-term bootstrapping happens: every n time steps (or whenever a terminal state is reached) a sequence of updates like those in Equations (2.10) and (2.11) take

place, one for each time step. In every update all the available subsequent rewards are used to compute a more precise estimation of the value of the current state.

Algorithm 3 *n-step* Advantage Actor Critic

initialize parameters θ_π of $\pi(a|s,\theta_\pi)$
initialize parameters θ_v of $V(a|s,\theta_v)$
$t \leftarrow 1$
while *learning is active* **do**
  get s_t
  $t_{last\_up} = t$
  while $t - t_{last\_up} < n$ **and** s_t *is not terminal* **do**
    execute action a_t drawn from $\pi(a|s,\theta_\pi)$
    get reward r_t, state s_{t+1}
    $t \leftarrow t+1$
  end while
  $R = \begin{cases} 0 & \text{if } s_t \text{ is terminal} \\ V(s_t,\theta_v) & \text{otherwise} \end{cases}$
  $\Delta_\pi = 0,\ \Delta_v = 0$
  for $i \in \{t-1,..,t-t_{last\_up}\}$ **do**
    $R = r_i + \gamma R$
    $\Delta_\pi = \Delta_\pi + \nabla_{\theta_\pi}\{log\pi(a_i|s_i,\theta_\pi)\}[R - V(s_i,\theta_v)]$
    $\Delta_v = \Delta_v + \nabla_{\theta_v}\{R - V(s_i,\theta_v)^2\}$
  end for
  get $\alpha_\pi,\ \alpha_v$ from the optimization algorithm
  $\theta_\pi = \theta_\pi + \alpha_\pi \cdot \Delta_\pi$
  $\theta_v = \theta_v + \alpha_v \cdot \Delta_v$
end while

2.3 Deep learning

Machine learning algorithms, as explained at the beginning of this chapter, aim at extracting knowledge from raw data, so that it can be used to interpret correctly new data. This knowledge, or *representation*, is encoded in pieces of information called *features*. For example, a machine learning algorithm can be used to decide where to build a new hospital in a city, in which the features describing the scenario may be the position of the other hospitals, the inhabitants of every district and its densities.

However, finding the features needed to accomplish a task might be an hard job: for some problems, like pedestrian recognition, years of work of the scientific community was needed to shape good enough features. Let's think to the task concerning the classification of images, e.g. recognizing if a picture contains a monkey or not. This task is very easy for humans, but the election of features useful for a computer to solve the problem is very challenging: monkeys may have different breeds and dimension, while the picture may be taken in a zoo or in the jungle.

Representation learning is a set of methods used to solve this problem by designing the algorithm to learn by itself which are the most relevant features for accomplishing its task. *Deep learning* methods, more specifically, build this representation in terms of other simpler representations using architectures called neural networks, due to their partial adherence to the human brain principles. The term 'deep' comes from the fact that these architectures are composed by a sequence of *layers*, each one transforming its input into a higher level representation, such that it becomes possible to learn complex functions ([49]).

A layer is made of many neurons: each neuron takes as input a portion of the signals ($\mathbf{i}$) coming from the preceding layer, and applies a transformation which in the simplest form is of the type:

$$l = \mathbf{a} \cdot \mathbf{i} + b \tag{2.12}$$

where $\mathbf{a}$ and b are the vector of weights and bias value respectively, and are the parameters that have to be learned in order to describe the desired function which gives the neuron scalar output l.

However, if layers of neural networks would execute simple linear transformations,

their concatenation, no matter how long, would lead to an overall linear function, which would be of limited usefulness. For this reason, neurons contain an additional component that has to be applied to their linear output, that is the *activation function* (*af*), whose aim is to determine the degree of activation of that neuron, making the final output of the type:

$$o = af(\mathbf{a} \cdot \mathbf{i} + b) \tag{2.13}$$

Different types of functions can be used as activation function, with the only required feature of containing a non-linearity; this way, a cascade of non-linear layers, even if simple taken singularly, has the power of describing non-linear functions with almost arbitrary complexity.

Some of the most used activation functions are:

- **Sigmoid**, defined as $\mathbf{S}(x) = \frac{1}{(1+e^{-x})}$
- **Hyperbolic tangent**, defined as $\mathbf{tanh}(x) = \frac{2}{(1+e-2x)} - 1 = 2 \cdot sigmoid(2x) - 1$
- **Rectified Linear Unit**, defined as $\mathbf{ReLU}(x) = max(0,x)$

Another constraint for the activation functions and the layers of a neural network is that they have to be differentiable[1], in order to permit the use of the technique called *backpropagation* ([50]), which is the essence of the automatic learning of the weights of a neural network.

Indeed, being the layers of a network deterministic and differentiable, it is possible to compute the partial derivatives of the loss function with respect to each parameter, allowing the application of a gradient-descent approaches for their update.

The term "backpropagation" comes from the way derivatives are computed, that is in a backward propagating sweep through the layers of the neural network, in which the partial derivatives of the loss function respect to the weights of a layer are defined in terms of the derivative of the loss respect the input of the subsequent layer; this corresponds to an opposite order respect to the one followed for the inference, which is from the input towards the final output.

[1] Sometimes, as in case of the ReLU, activation functions may be non-differentiable; however, things still work since the non-differentiability is restricted in few specific points.

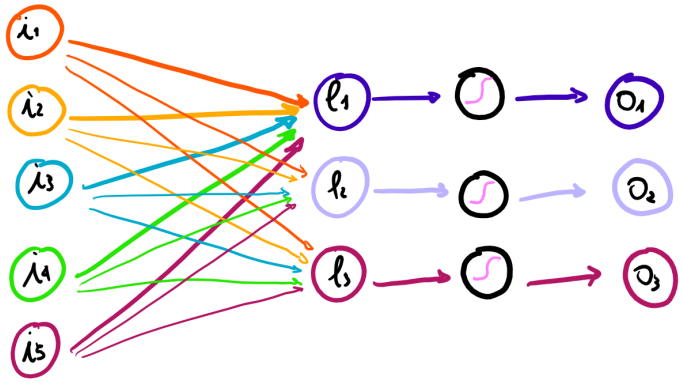


Figure 2.2: Outline of a fully-connected layer: every neuron is a linear transformation of all the inputs, to which is then applied the non-linearity (in this case a sigmoid function)

to compute the final output

Different input patterns given to the neurons, together with their internal disposition, give origin to different types of layers, each one specialized for a specific task. The two architectures used in the development fo this project are the *fully-connected* and *convolutional* layers, and will be explained briefly in the following.

2.3.1 Fully-connected layers

Fully-connected layers, as their name suggests, are layers in which neurons take in input all the output signals of the preceding layer. This type of layer is expensive in terms of the amount of parameters needed; indeed each neuron will have one weight for each neuron of the precedent output.

A layer of this kind is used when the output of a neuron has to be dependent on the totality of the layer input: for example, a layer of this kind is typically used as the last layer of a neural network which output the probabilities of a set of actions: indeed, each neuron will output a probability for a specific action, and all the neurons have to take into account all (and the same) output of the precedent layer.

2.3.2 Convolutional layers

Convolutional layers were thought specifically for processing images, while are also a valid choice in those cases where the input to the network features some spatial pattern to be decoded, like speech recognition or time-series data.

In fact, this architecture was biologically inspired and the connectivity pattern among neurons resembles that of animal visual cortex, in which single neurons respond to stimuli coming from a small region of the visual field called *receptive field*.

For this reason, in contrast to fully-connected layers that perform general matrix multiplications, convolutional layers are made of a sequence of fixed size patterns, generally called *filters*, which are scrolled in a convolutional manner over the set of input images, so that a new set, possibly with different image dimensions and set cardinality, is produced.

What has to be learned in this case, are the values of each element of the filters: since their dimension is independent to the image dimensions, thanks to the convolutional operation, the number of parameter needed for these layers is generally several order of magnitude lower. For example, if 64 filters of size $5x5$ are used to process an input made of 16 images having dimensions $100x100$, the number of parameters of this layer would be equal to:

$$num\_filters * (filter\_dims + bias) = 64 \cdot (5 \cdot 5 \cdot 16 + 1) = 25664 \tag{2.14}$$

while, for a fully-connected layer featuring the same input and output sizes, this would be:

$$(num\_in + bias) * num\_out = (100^2 \cdot 16 + 1) \cdot (100^2 \cdot 64) = 102,4 \cdot 10^9 \tag{2.15}$$

In addition to the parameter-efficiency, convolutional layers have also the capability of being translation invariant: indeed, a learned filter will be able to recognize patterns independently on their positions in the image, further increasing the learning scalability.

When one or more layers of a neural network are convolutional layers, the network architecture is called *Convolutional Neural Network* (CNN). CNNs are generally

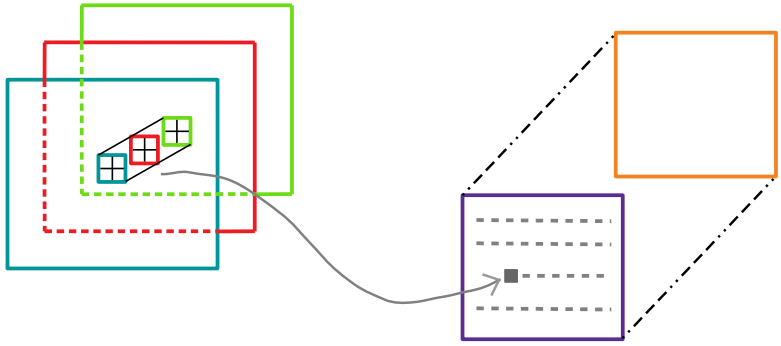


Figure 2.3: Outline of a convolutional layer: in this case a 3-channel input image is processed by a filter of dimension 2x2, producing a feature map as output. The output image will have as many channels as the number of filters being used

structured in a pyramidal shape, with the initial input dimensions shrinking from one layer to the other. The input-dimensions reduction is possible thanks to the use of *stride* and *max-pooling* layers. Stride controls the scrolling step-size of the filters around the input volume: a value for the stride higher than one means that some positions in which the convolution can take place are skipped.

A max-pooling layer consists on down-sampling the input volume by selecting only the maximum out of each restricted region of the input; typically this is done for each 2x2 blocks of every feature map, resulting on a scaling factor of 2. This layer, besides reducing the input dimensions, gives also invariance to local translation, since small shifts on the input features will result in the same feature map as output.

An example of a simple CNN is shown in Figure 2.4.

2.4 The flourish of deep reinforcement learning

Reinforcement learning, as previously seen, requires the computation of either a value function, or a policy, sometimes both. Apart from well-structured simple cases, it is

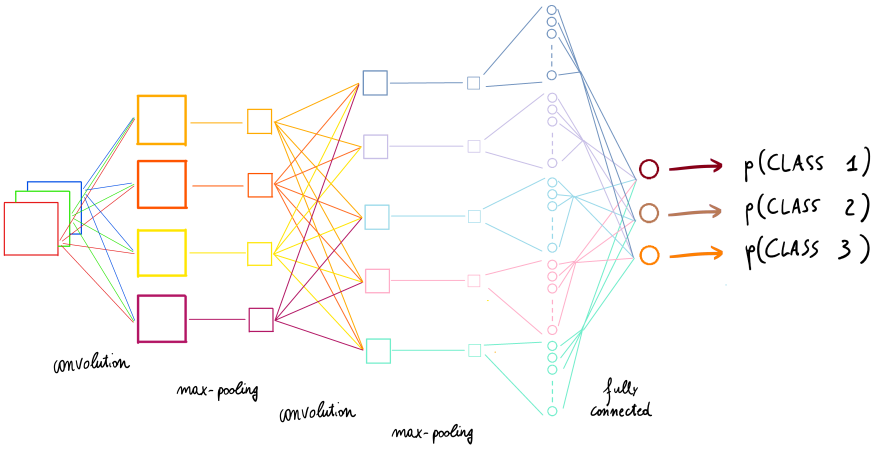


Figure 2.4: Scheme of a simple Convolutional Neural Network which takes as input a 3-channel image and output the probability associated to three different classes. The network features 2 convolutional layers, each one followed by a max-pooling layer, and a final fully-connected layer that computes the score for each class

not possible to reach an exact representation of the optimal functions, due to the *curse of dimensionality*: very large state space dimensions are rapidly reached, and learning would become prohibitively slow. For example, the ancient game of *GO*, which is played in a chess-like grid having dimensions $19x19$, features a state-space complexity of 10^{120}, which is more than the total number of atoms in the universe. It is obvious that tabular methods as the ones shown in Algorithms 1 and 2 cannot be applied in such cases. Indeed, in such large state spaces is not possible to visit every single state, indeed it becomes essential the capability of generalizing what was previously learned to unseen states.

In order to reach this goal becomes necessary the use of function approximators for learning sub-optimal but more compact representations of the optimal functions that have to be learned: several approximators were initially used for this purpose such as tile coding ([51]), decision trees ([52]), locally weighted regression ([53]) and radial basis functions ([54]). Besides, in order to flee the need of hand-design features, the idea of using neural networks as function approximators seems trivial.

However, until few years ago, the fusion of reinforcement learning and deep learning was limited to few simple cases (e.g the *Tower of Hanoi* or pole balancing problems as in [55]): this was mainly due to the complications arising from training neural networks with limited computational resources and, more importantly, from the difficulties of making their weights converge. Indeed, delayed rewards and not independent and identically distributed data afflicted badly the learning procedure.

Things changed recently, when authors of [56] have been able to train a neural network in order to play several *ATARI 2600* games at human level and beyond. The crucial factor was the introduction of the *experience replay*, a mechanism consisting on continuously storing a large amount of past experiences made of tuples (s_t, s_{t+1}, r_t, a_t): during the learning phase, instead of using fresh data, off-policy learning was computed on a batch of past experiences randomly sampled from the experience replay buffer[2]; this way, the updates came from a distribution which gets closer to i.i.d. data, helping the convergence of the parameters.

Since then, copious works flourished about the application of deep reinforcement learning to new and more complicated tasks; in this work it has been developed a novel version of the deep reinforcement learning algorithm called *A3C*.

2.5 Asynchronous Advantage Actor-Critic (A3C)

The use of experience replay, while powerful under some aspects, places some limitations. First of all, storing all that data is expensive in terms of memory occupation. More importantly, adopting the experience replay means using data which may have been acquired long time before: if the environment dynamics are not static, its efficiency would quickly decrease until becoming harmful, because updates would be computed using obsolete experiences. The same problem arises, as explained in Section 2.6, when using a deep reinforcement learning approach in a multi-agent scenario: indeed, agents' behaviors changes through time making the stored tuples out-of-date.

These limitations have been overcome in [57], where instead of using the experience replay, the uncorrelation between updates is increased by letting several

[2]The authors of [56] used a version of *Q-Learning*, which was shown in Algorithm 2

independent agents run concurrently in different copies of the same environment producing asynchronous updates.

In particular, the asynchronous variant of the Advantage Actor Critic algorithm (which was explained in Section 2.2.9) involves the use of two different deep neural networks, one for defining the agent policy (actor part), and the other for estimating the value of states (critic part), both of which are simultaneously updated from all the agents concurrently.

Indeed, every agent owns a local copy of the parameters of these shared networks, and uses it for taking actions for a limited amount of time T; after T, possibly bootstrapping if a termination state is not reached, the updates of the policy and state estimation networks are computed using Equations 2.10 and 2.11 and sent to the shared global networks. In return, each agent updates its networks with the parameters of the global networks, therefore adopting also the improvements which were already sent to the global collector from the other agents, as shown in Figure 2.5a. Since each agent is experiencing a different and independent environment configuration, their updates are not correlated, thus augmenting the process stability.

Algorithm 4 shows the asynchronous variant of Advantage Actor-Critic which was shown in Algorithm 3.

2.6 Multi-agent deep reinforcement learning

Until now we have considered reinforcement learning scenarios in which a single agent has to deal with an environment, which has to be quasi-static so that the optimal policy and/or value functions can be gradually learned. The agent is modeling itself, the environment and the reciprocal interaction of these two components. Possibly, other players are taking actions in the same environment, but they are considered from the learning agent as a part of the environment; these are single-agent scenarios, because only a unique agent is evolving its behavior.

In the same category fall all those problems containing more than one learning agent but featuring a central controller that knows exactly the full global state; indeed, the same system can be viewed as a single-agent problem in which the agent is made

Algorithm 4 *n-step* Asynchronous Advantage Actor-Critic (A3C)

get n_{env} as the number of environment instances
initialize parameters θ_π^g of $\pi(a|s,\theta_\pi)$
initialize parameters θ_v^g of $V(a|s,\theta_v)$
run n_{env} environment instances
for *environment* $e = \{1,..,n_{env}\}$ **do** {*execute concurrently until termination*}
  $t \leftarrow 1$
  copy parameters $\theta_\pi^e = \theta_\pi^g$
  copy parameters $\theta_v^e = \theta_v^g$
  $\Delta_\pi^e = 0$, $\Delta_v^e = 0$
  while s_t *is not terminal* **do**
    get s_t
    $t_{last\_up} = t$
    while $t - t_{last\_up} < n$ **and** s_t *is not terminal* **do**
      execute action a_t drawn from $\pi(a|s_t,\theta_\pi^e)$
      get reward r_t, state s_{t+1}
      $t \leftarrow t+1$
    end while
    $$R = \begin{cases} 0 & \text{if } s_t \text{ is terminal} \\ V(s_t,\theta_v^e) & \text{otherwise} \end{cases}$$
    for $i \in \{t-1,..,t-t_{last\_up}\}$ **do**
      $R = r_i + \gamma r$
      $\Delta_\pi^e = \Delta_\pi^e + \nabla_{\theta_\pi^e}\{log\pi(a_i|s_i,\theta_\pi^e)\}[R - V(a|s_i,\theta_v^e)]$
      $\Delta_v^e = \Delta_v^e + \nabla_{\theta_v^e}\{R - V(a|s_i,\theta_v^e)\}^2$
    end for
    update θ_π^g using Δ_π^e
    update θ_v^g using Δ_v^e
  end while
end for

of several components to be controlled; an action taken from the central controller can be decoded in several sub-actions, one for each component; this configuration is called *centralized*.

In multi-agent systems, on the other hand, several agents are simultaneously learning in the same world, modeling each other's goal and actions. The presence of multiple independent agents, all of which are constantly exploring then behaving unpredictably, makes the environment dynamic and constantly changing from the agent's perspective, adding a further uncertainty to that inherent to the domain ([58]): this means that the state is not dependent anymore only on its precedent state and the action of the single agent, that is the state does not satisfy the Markov property.

For instance, let's consider two agents, a and b, living on the same environment: a, by taking action a_t^a at time step t based on policy $\pi(a_t^a|s_t^a)$, may modify the subsequent state s_{t+1}^b of the agent b. Agent b, consequently, will react accordingly taking action $\pi(a_{t+1}^b|s_{t+1}^b)$, which may modify in turn s_{t+2}^a. Therefore, learning the optimal policy for an agent means taking into account not only how the environment is affected by its own actions, but also what mutual behaviors are induced to other agents. Systems of these kind are called *decentralized*, and the agents *independent learners*.

In some multi-agent scenarios, agents may be aided in the task of understanding each other intentions by the possibility of sharing information through communication protocols. This information may travel by direct messages between agents or lay in a shared memory storage publicly accessible, so that it can be used to maximize their shared utility. In Reinforced Inter-Agent Learning (RIAL) ([59]), for example, each agent uses a neural network to output both the Q-values (as in standard Q-learning shown in Algorithm 2) and a message to communicate to other agents.

However, agents are not always endowed with communication capabilities and sometimes a communication channel is not available: in these situations agents have to learn how to infer the intentions of the others. Dependently on the nature of the task, agents may evolve either competitive or cooperative behaviors, or a mixed approach between these two.

In multi-agent problems, traditional single-agent algorithm may be used considering, from the agent's perspective, the other players as part of the environment instead

of a source of uncertainty coming from their evolving policies; at the same time the extension of single-agent solutions to multi-agent scenarios has to be done cautiously, since their underlying assumptions, the requirement of a Markovian state, will be violated and there are no theoretical converging guarantees.

In spite of this, decentralized approaches of this kind have been used with satisfying results even when a centralized solutions would have been possible, leading to a better computational efficiency thanks to an higher scalability. Indeed, even if centralized approaches does not suffer from non-stationarity, their state space increases exponentially with the number of agents and may become infeasible to process, while decentralized approaches can maintain it confined.

In [60], some single-agent deep reinforcement learning algorithms, such as deep Q-Learning, TRPO ([61]) and DDPG ([62]), have been extended to multi-agent domains, highlighting some of the limitations which arose in the new setting. In particular, the use of experience replay in multi-agent systems proved to be unreliable, due to the incompatibility between the replay buffer and the non-stationarity of multi-agent systems: indeed, the constantly changing policies of the agents make the stored experiences out-of-date, jeopardizing the stability of the system[3].

In order to dim this problem, an importance-sampling technique has been used in [64] in order to weight differently new and old experiences. For the same purpose, authors of [65] used a centralized approach during the learning phase, storing in the replay buffer not only the action taken from each independent learner, but also the actions of the others, making the learning process stationary: the replay buffer is needed only at training time, and agents can behave independently during execution time; this solution, called centralized learning with decentralized execution, requires fully knowledge of the system in the training phase, thing that is not always available and reduces the scalability of the system.

Asynchronous approaches, on the other hand, substitutes the use of the replay buffer by multiplying the number of environments in which independent copies of the same agent learn simultaneously; as a consequence, we believe that the extension of this solution to multi-agent scenarios leads to an increased stability of the learning

[3]This was already known in 1992, as explained by Lin in [63]

process. In this setting, agents are not independent anymore, but share the environment with copies of other learning agents, making it possible to learn reciprocal interactions.

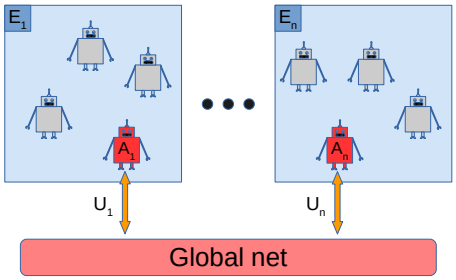
Agents might be of different types and have each one its own specific task: for example, if the agents would be members of a firefighters team, there could be an agent in charge of driving the fire track, an agent in charge of extinguishing the fire, and another agent specialized in saving people trapped in hostile environments. In that case, it is advisable to have each agent with its own structure designed optimally for its own task; that is, having its own architecture and parameters indicating the policy to follow. If this is the case, in each environment instance would be present a copy of each agent typology as shown in Figure 2.5b. Algorithm 5 shows a basic multi-agent version of the A3C algorithm for such scenarios.

Different is the case in which all agents are similar and have tasks of the same typology, as in case of virtual cars of a traffic simulator, where all artificial drivers have to behave following the same traffic rules and have comparable goals. In these cases, it is possible to use a technique called *parameter sharing*, that is using the same architecture and parameters for each and every agent. This way, the acquired experience of every single agent is added up to that of the others in order to improve a shared single policy, thus increasing the amount of training samples available to train a single neural network ([66]).

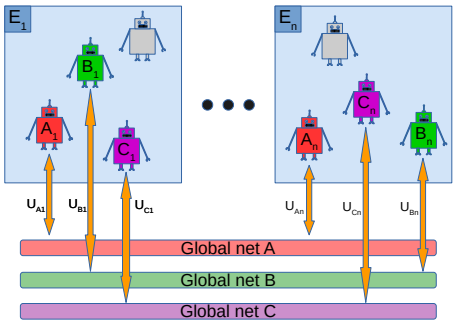
This solution permits to scale up effectively to problem with a large number of homogeneous agents without problems of memory occupation and time-efficiency because the computational complexity is linear in the number of agents, but also the amounts of exploration grows linearly. Dependently on the application, it may be unnecessary to use several copies of the environment, since the diversity in the agent experiences could arise from different copies of the same agent seeing different observations of the same environment, as shown in Figure 2.5c.

Sharing parameters does not necessarily mean that agents behave in the same way taking the same actions: indeed, even if they share a single policy, each agent is receiving different observations of the environment, and it will behave accordingly to a different situation.

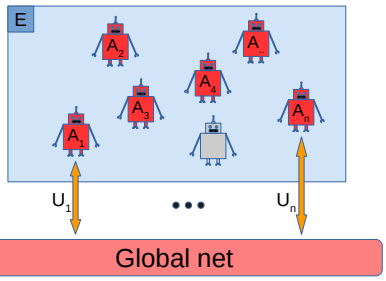
Nevertheless, sharing the same policy poses some restrictions: even if agents will



(a) Single-agent setting



(b) Multi-agent setting with non-homogenous agents



(c) Multi-agent setting with parameter sharing

Figure 2.5: Comparison between single and multi-agent A3C settings. Colored robots are active agents learning in the environment copy E_i, while gray robots are passive agents. Each active agent owns a copy of the global network related to its category x ($x \in A, B, C..$), and it contributes to its evolution by updates $U_x i$. In **(a)** learning agents are independent from each other, while in **(b)** active agents belonging to different category can sense each other, allowing them to learn how to interact. In **(c)** homogeneous agents are sharing parameters inside the same environment instance, contributing to the update of the same global network

Algorithm 5 Multi-agent version of A3C with non-homogeneous agents

get n_{env} as the number of environment instances
get I as the set of different agent typologies
initialize parameters $\theta^g_{\pi_i}$ of $\pi_i(a|s,\theta_{\pi_i})$ for every agent typology $i \in I$
initialize parameters $\theta^g_{v_i}$ of $V_i(a|s;\theta_{v_i})$ for every agent typology $i \in I$
run n_{env} environment instances
for $e = \{1,..,n_{env}\}$ **do**
  for $i = \{1,..,|I|\}$ **do**
    create agent A^e_i
  end for
end for
for every *agent* $A^{e=\{1,..,n_{env}\}}_{i=\{1,..,|I|\}}$ **do** {*executed concurrently until termination*}
  $t \leftarrow 1$
  copy parameters $\theta^A_\pi = \theta^g_{\pi_i}$
  copy parameters $\theta^A_v = \theta^g_{v_i}$
  $\Delta^A_\pi = 0$, $\Delta^A_v = 0$
  while s_t *is not terminal* **do**
    get s_t {*dependent on the* $(|I|-1)$ *agents of the same environment* e}
    $t_{last\_up} = t$
    while $t - t_{last\_up} < n$ **and** s_t *is not terminal* **do**
      execute action a_t drawn from $\pi(a|s_t,\theta^A_\pi)$
      get reward r_t, state s_{t+1}
      $t \leftarrow t+1$
    end while
    $$R = \begin{cases} 0 & \text{if } s_t \text{ is terminal} \\ V(s_t,\theta^A_v) & \text{otherwise} \end{cases}$$

for all $i \in \{t-1,..,t-t_{last\_up}\}$ **do**
  $R = r_i + \gamma r$
  $\Delta_\pi^A = \Delta_\pi^A + \nabla_{\theta_\pi^A}\{\log \pi(a_i|s_i,\theta_\pi^A)\}[R - V(a|s_i,\theta_v^A)]$
  $\Delta_v^A = \Delta_v^A + \nabla_{\theta_v^A}\{R - V(a|s_i,\theta_v^A)\}^2$
end for
update $\theta_{\pi_i}^g$ using Δ_π^A
update $\theta_{v_i}^g$ using Δ_v^A
end while
end for

behave accordingly to their unique observations, they will have the same personality: different agents, in similar situations, will behave in the same way. This condition may represent a problem in some applications: coming back to the traffic simulator example, parameter sharing would lead to drivers with exactly the same driving style, while we all know that every one has its own unique personality behind the wheel. If the desire is to have a traffic simulator that mimics real-world situations, this solution is not suitable anymore.

For this reason, in this work has been analyzed a novel approach in which parameter-sharing agents can behave according to different styles by enriching the observation of each agent with information devoted to tuning its behavior, as explained in Section 3.4.

Chapter 3

Microscopic Traffic Simulation using Multi-Agent Deep Reinforcement Learning

3.1 Simulation in reinforcement learning

Deep Reinforcement Learning, as explained in Section 2, gathers very powerful methods useful to solve sequential tasks without the need of supervision or annotated data, generating agents capable of generalizing in high-dimensional state spaces.

However, in order to achieve good performances, a large amount of training experience is needed. Indeed, agent's intelligence consists on a neural network function approximator, whose strength and weakness is the ability of automatically selecting the features needed to understand the input data. I said strength because its outstanding skills are due to the super-human capability of finding the best patterns to represent the data, and I said weakness because many samples are needed to shape those features correctly ([67]).

Moreover, the agent not only has to understand the input its receiving, as it would be the case of a classifier module, but it has also to act properly once having understood its environment. This means that the functions approximating the policy

or the value functions has also to encode the behavior of the agent and the dynamics of the environment.

This is possible thanks to a trial-and-error approach, which requires the agent to explore many states and try different actions, in order to understand their effects. It is clear that such brute-force solution requires many samples to discover efficient policies.

Now the question is: how to get all this experience in real-life situations? Will the task I have to solve allow it? Unfortunately often this is not the case; let's think, as example, to all those tasks involving a real robot learning to navigate: will the robot be able to collect also negative-ending trajectories which may involve crashing? Or, even more, if the robot has to navigate in an environment populated by humans, will it be a danger for them? Nevertheless, even in case of controlled environments and robust robots, the human work required to follow the robot throughout its learning phase would be often out of reach.

For these reasons, frequently the only possible solution is the use of a synthetic environment in which a model of the agent is free of taking actions repetitively with no risks of getting damaged or being harmful. Moreover, this synthetic environment may ran on computers and be extremely more time-efficient, making it possible to acquire millions of trajectories in limited amount of time.

Several works addressed the problem of learning efficiently real-world tasks from simulation. Initially, hybrid approaches were proposed for simple tasks using standard reinforcement learning algorithms, using a combination of many synthetic trials and few real world trials: for example, in [68] and [69] an approach of this kind was used to command a robotic car using a low-dimensional state spaces consisting on agent position, velocities and heading angles.

More recently, hybrid solutions were proposed also for deep reinforcement learning algorithms dedicated to high-dimensional input made of raw visual images, as in [70] in which progressive nets[1] were used for learning a task in gradually more difficult simulations, or in [72] in which real and synthetic examples are used together

[1] Progressive networks are explicitly designed for supporting the transfer across sequences of tasks by leveraging prior knowledge via lateral connections to previously learned features ([71]).

to reduce a cross-domain adaptation loss.

Furthermore, some works tried to learn real-world tasks using only synthetic data, as in case of [73] in which a drone were taught to fly using camera images as input, even if it had never seen a real image before. This was achieved by designing a simulator with randomize textures, so that the agent could learn domain agnostic actions.

3.2 Simulation in the self-driving domain

Learning navigation tasks as lane following in real scenarios using deep learning has been tried since the '80s with ALVINN ([13]). More recently, thanks to development in the field and increased computation capabilities, it has been achieved using raw camera images as input ([37]). These works were learned in a supervised fashion: the neural network was fed with examples of the task accomplished by a human expert, whose actions were taken as label associated to each input signal.

Even if these achievements were breakthroughs for the scientific field, highlighting the possibility of learning self-driving tasks from demonstration instead of manually designing the whole navigation pipeline, they were not usable in practice without a human which was ready to intervene in case of undesired behaviors. Indeed, policies learned in a supervised fashion are weak against situations not seen in the training dataset due to the way in which they are learned ([74]): the agent is told exactly what to do in the training cases, and the neural network builds a mapping between input and actions without thinking to the policy as a whole. For this reason, in order to make imitation learning successful, further considerations have to be taken into account, such as the perturbation with synthesized trajectories in order to expose the agent to undesired situations not present in the dataset and learn how to recover from them ([75]).

Reinforcement learning, on the other hand, leaves the agent free of exploring different behaviors other than the optimal one, implicitly making him more robust against undesired situations: throughout the learning phase the agent will hopefully discover those maneuvers which restore it to a safe state from dangerous situations,

caused by wrong choices or unexpected events, in order to avoid the negative reward that will be received in case of unhappy ending. Policies are not anymore a direct mapping between single states and actions, but take into account the long-term effect of the decisions made, in order to maximize also the rewards far in the future.

However, teaching a self-driving car to drive in a reinforcement learning fashion directly on public roads is not a good idea. Indeed, accidents and dangerous maneuvers have to happen in order to learn the negative effects they will produce. This is clearly not possible, due to safety reasons and for the impossibility of breaking a car at every mistake. There are some works, as [76] and [77], in which agents learned simple tasks directly on real life with few exploration trajectories. However, the environment was limited to simple cases that did not consider the presence of other agents. For these reasons, in order to successfully learn complex behaviors in multi-agent scenarios a simulator is often essential.

Some works, as [78], use parametric simulators in which agents learn to perform maneuvers by observing coordinates, velocities and other numerical variables such as the lane number of the other agents, and output high-level commands that have to be translated into driving actions. These graphics-less simulators have the advantage of providing a very low-dimensional state space which greatly simplify the learning process. However, a different description of the scene has to be adopted for every specific case, making them inadequate to generalize to unseen scenarios.

Instead, more flexibility can be obtained exploiting recent developments on Convolutional Neural Networks architectures (described in Section 2.3.2), which give the possibility of training agents directly from raw images, making it possible to build agents best suited to generalization; depending on the desired approach, a different type of visual simulator has to be adopted.

3.2.1 Simulators with realistic graphics

If the goal is to build an agent able to navigate using raw camera images, the graphics of the simulator has to get as much as close to the camera one. Several attempts have been made using the *Grand Theft Auto V* (*GTA*) video game, thanks to its very high-definition and realistic physic engine ([79]). For example, in [80], it has been

demonstrated that a large number of images captured from GTA is more efficient than fewer yet real images for object detection purposes.

However, entertainment programs are not made for algorithm development purposes, so do not facilitate benchmarking and environment customization: they do not permit extensive sensor simulation (e.g. several cameras mounted on the same vehicle) and they do not provide detailed feedback upon dangerous driving behaviors and violation of traffic rules.

In order to fulfill those requirements, authors of [81] developed a simulator called *CARLA* (*CAr Learning to Act*) build upon the Unreal game engine ([82]); this simulator is build from the ground for supporting autonomous driving models, and for this reason it was endowed with flexibility on the setup of simulated sensor suites and with the possibility of providing feedback signals useful for training autonomous agents, such as GPS coordinates, motion details as speed and acceleration, and information about collisions and infractions committed. Moreover, it gives the possibility of designing urban scenarios at will and specifying weather conditions and time of the day.

Some works, as [83] and [84], evaluated the possibility of transferring policies learned in CARLA and TORCS ([85]) simulator respectively to the real world, using as intermediate level the segmentation of the scene obtained from the camera in order to diminish the domain gap.

However, experiments involve simple tasks without considering the participation of other agents in the process. It remains an open question if it would be possible to extend such solutions to more complex multi-agent tasks.

3.2.2 Simulators with simplified graphics

A different approach is taken from simulators that does not try to mimic what we see with our eyes, but instead build a semantic representation of the scene, thus reducing its sample complexity. Indeed, the visual representation of a car does not depend anymore on its brand or color, but its fixed a priori: only the crucial appearance-independent information are maintained, such us its position on the road, encumbrance and motion dynamics. At the same time, an agent learning in such an environment does not have to learn that a road may be paved or cobblestoned, or even covered with snow, since

its representation in all those cases would be the same.

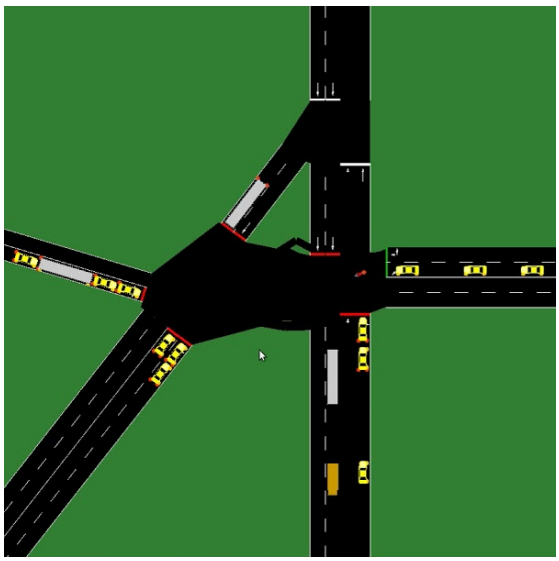
Often, a top-view representation of the scene perceivable from the agent is used instead of first person observations, as in case of SUMO ([86], Figure 3.1a) or Waymo's ChauffeurNet ([75], Figure 3.1b) simulators: this way, a car would be drawn as an oriented two-dimensional box, meaning that also the three-dimensional perspective is removed from the interpretative burden in charge of the agent. Therefore, the idea is to simplify as much as possible the representation of the environment, so that the agent can focus mainly on learning a good-enough policy needed to accomplish its task instead of understanding also the semantic of what it is seeing around it.

The drawback of this mid-level representation comes from the fact that the real world is indeed complex and incredibly various, thus very far from the representation used. For this reason, in order to transfer the policy learned in simulation into real driving it is necessary to use detection systems able to reconstruct a representation similarly to that seen in simulation. This means that the self-driving car has to be equipped with object detectors able to recognize cars, pedestrians and other relevant information from the input coming from its sensors; at the same time, it would need high-level maps and localization systems in order to be able to reconstruct the correctly the scenario.

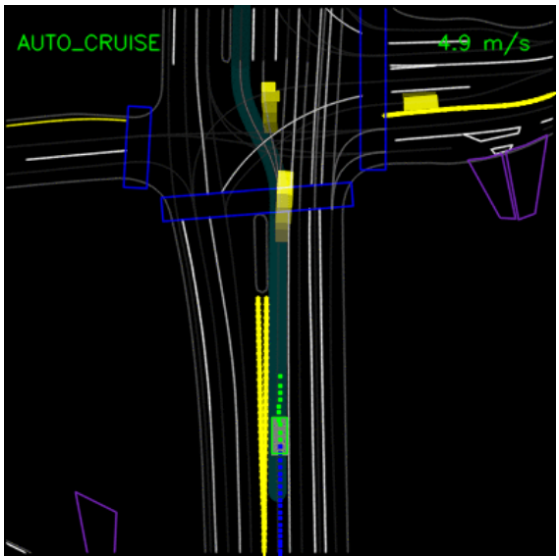
An agent trained in a simulator of this kind does not feature anymore the full end-to-end characteristic in which everything is taken into account from a big neural network which solve all the fundamental tasks using raw sensor data as explained in Section 1.4.2, but it will require a suite of systems in charge of several limited tasks. On the other hand, the domain gap between simulation and reconstructed scenes can be much smaller than the gap obtained when using simulators with realistic graphics, since it is not needed anymore to get closer to the visual appearance of the scene.

3.3 Why simulating with multi-agent deep reinforcement learning

What is often missing in the simulators used to train agents able to perform high-level maneuvers, whether featuring a realistic or simplified graphics, is the realism on the



(a) SUMO



(b) ChaffeurNet

Figure 3.1: Snapshots of the visualization used by SUMO **(a)** and Waymo's Chaffeurn-Net **(b)** simulators

behavior of the agents populating the simulated scenario. For example, in [87], an intersection-handling model is developed in a SUMO-based environment, in which vehicles' motion follows a deterministic follow-the-leader model called *Intelligent Driver Model* (IDM, [88]), while in [89] all vehicles behave aggressively following a preprogrammed conduct.

Similarly, a lane change system is developed in [90] in an environment whose vehicles follow a simple lane-keeping behavior with collision avoidance, or in [91] in which they are able to overtake relying on hard-coded rules.

In all these works, vehicles behaves deterministically making their future trajectories easy to predict: however, we all know that humans behind the wheel have very high variability in their driving styles, depending on their personal temper and external factors; let's think for example how a caring mother would drive while carrying her baby to the kindergarten comparing to a young student running because he is late for an important exam. Moreover, also physical factors influence the way of driving: elderly often react more slowly to external stimulus and people with poor eyesight might be more imprecise on their trajectories. Agents trained in environments in which all vehicles behaves similarly are not expected to behave well in real-world unpredictable scenarios.

There are some works which propose methods aimed at increasing the inter-driver variability: in [92] different speed trajectory are assigned to every IDM agents; in [93] also the time gap, bumper-to-bumper distance and acceleration parameters are personalized. Again, in [94] drivers can have one of four different hard-coded behaviors which try to mimic hesitating rather than aggressive drivers.

Even if assessing, albeit partially, the variability in the driving styles, none of the aforementioned solutions consider agents capable of complex nondeterministic behaviors such as the ability to negotiate. In [78] this is achieved by training the agents populating a merging scenario by imitation learning, using data collected from human drivers. However, this solution forces all the agents to follow the same policy, and, more importantly, it is very expensive in terms of required training data and directed to a specific scenario.

These reasons pushed us to think a new method for creating a simulation environ-

ment in which vehicles motion resembles more accurately the real-world dynamics: the idea is to let agents learn in a reinforcement-learning fashion how to navigate inside it, instead of giving them deterministic behaviors. Moreover, in order to learn also how to interact with other vehicles, an agent needs to learn how to navigate the environment while other agents are simultaneously driving (and learning) on it, that is learning in a multi-agent scenario: for this purpose, it has been designed a novel multi-agent deep reinforcement learning algorithm called *Delayed Multi-Agent A3C*.

The proposed approach, as explained with more details in the next sections, gives the possibility of channel the training experiences of every agent into the improvement of a single shared control unit (a neural network) by using the parameter sharing technique; at the same time, even if all agents are driven by the same policy, it permits to assign a unique personal behavior to each one of them, thanks to the use of a hybrid input made by environment observations and parameters defining the desired behavior of the agent.

In order to obtain a flexible platform easily expandable to a multiplicity of different scenarios, it has been used an input composed of visual observations of the environment instead of specific parametric features; at the same time, it has been employed a mid-level representation instead of a realistic one, enhancing the sample complexity of the state space and limiting the computational burden required to train several agents simultaneously.

The developed simulator is based on our work published in ([95]).

3.4 Delayed multi-agent A3C

In this project it has been developed a novel variant of a deep reinforcement learning algorithm called Asynchronous Advantage Actor Critic (A3C, Section 2.5) for simulating the traffic flow at vehicle level. This algorithm, which was initially proposed for solving single-agent problems, uses multiple parallel independent learners in order to diminish the correlation between updates of the neural network which drives the agent policy. Indeed, a well-known difficulty residing on the use of deep neural networks as reinforcement learning function approximators is the instability of the learning

process, which is sequential by nature.

Road traffic vehicles, generally speaking, can be seen as homogeneous agents because they all obey the road laws (more or less..), have similar goals (connect point A with point B) and a similar way to achieve it (they don't fly or have strange motion dynamics). For this reason it has been adopted the parameter sharing technique, that is the use of a single central policy which drives every agent, and, as a consequence, can be updated from all the agents at the same time (see Section 2.6).

However, if we look more closely, in real world driving each driver has its own peculiar way of driving, which is changing throughout its life time and is dependent on external factors such as being in a hurry. For this reason, even if sharing a unique single policy, the system has been designed so that the personal behavior of each agent can be shaped by modifying some of its dedicated input, as explained in the next section.

In the proposed setting, an agent, initialized with its behavioral parameters, explore and learn in an environment which is not anymore aseptic as in the classical single-agent version, but it is populated by several other similar agents, each one with its unique driving style. Therefore, agents have to learn how to reach their goal in a situation in which cooperation is needed; since they are not endowed with direct communication capabilities, the unique way to reach the goal is implicitly negotiating trough driving actions, as it happens in real world driving. Our belief is that agents, if given proper reaction time and motion dynamics, will discover human-like techniques to understand the best way to proceed: for example, a timid driver will be likely to give way to a more aggressive one in an intersection, after having tried with scarce results to insert.

In order to achieve these behaviors proper rewards have to be shaped, both to let agents respect traffic laws, as the right of way, and to penalize agents that does not follow their assigned driving style. Every agent is given a path to follow leading to its assigned goal position, and it will be rewarded positively if it would be reached without incurring in accidents; the explicit rewards used for the specific case of the roundabout insertion are explained in Section 3.7.3.

Agents take their actions at common time instances, so that race conditions are

avoided and their neural network inference steps are executed together. Dependently on the scenario, it might happen that the number of agents simultaneously learning on it is not sufficient to guarantee enough differentiation on the updates of the policy. For example, in case of intersections or roundabouts, the number of learning agents is limited by the number of incoming branches; since the updates coming from different but interacting agents is not independent anymore, the overall correlation between updates may be excessive.

In order to cope with this problem, several independent environments have been simulated simultaneously as in A3C, permitting to multiply the number of parallel learners, though maintaining them shared so that inter-agent interaction can still be learned; the effectiveness of the use of a multi-environment configuration are analyzed in the ablation studies of Section 3.7.7.

Another novelty of the proposed multi-agent algorithm respect to single-agent A3C, is the addition of delay on the exchanges of updates between agents and the global collector. Indeed, even if agents compute *n-step* updates as before, they do not send them immediately but wait a longer time in which they collect them locally, after which they send a unique bigger update; in particular, in the proposed approach, updates are sent at the end of each agent episode.

The reason of this choice is twofold: on the one hand, it reduces the synchronization burden because every exchange of parameters requires time for the transfer; since agents compute updates asynchronously but have a shared time scale, it is sufficient that only one of them is computing its backpropagation for getting all the others stuck, slowing down the full process; reducing the number of exchanges by bundling several updates together mitigates this overload. On the other hand, the presence of delay on the local policy updates leads to an augmentation on the diversity of agents behaviors; indeed each one of them will own a copy of the global networks related to more distant time intervals, which is also evolving independently until the next exchange of updates; this added noise is potentially beneficial for the exploration by letting agents investigate slightly different policies. The pseudo-code for multi-agent delayed A3C in case of discrete action-space is given in Algorithm 6.

In some cases, it could be advantageous to use a technique called *action repeat*,

Algorithm 6 multi-agent delayed A3C for discrete action-spaces

get n_{env} as the number of environment instances
get n_{ag} as the number of agents for each environment
initialize parameters θ_π^g of $\pi(a|s,\theta_\pi)$
initialize parameters θ_v^g of $V(a|s,\theta_v)$
run n_{env} environment instances
for $e = \{1,..,n_{env}\}$ **do**
    for $i = \{1,..,n_{ag}\}$ **do**
        create agent A_i^e
    end for
end for
for every *agent* $A_{i=\{1,..,|I|\}}^{e=\{1,..,n_{env}\}}$ **do** {*executed concurrently until termination*}
    $t \leftarrow 1$
    copy parameters $\theta_\pi^A = \theta_\pi^g$
    copy parameters $\theta_v^A = \theta_v^g$
    $\Delta_\pi^A = 0$, $\Delta_v^A = 0$

16: **while** s_t *is not terminal* **do**
17: get s_t {*dependent on the* $(n_{ag}-1)$ *agents of the same environment*}
18: $t_{last\_up} = t$
19: **while** $t - t_{last\_up} < n$ **and** s_t *is not terminal* **do**
20: execute action a_t drawn from $\pi(a_t|s_t, \theta_\pi^A)$
21: wait other agents to finish their actions
22: get reward r_t, state s_{t+1}
23: $t \leftarrow t+1$
24: **end while**
25: $R = \begin{cases} 0 & \text{if } s_t \text{ is terminal} \\ V(s_t, \theta_v^A) & \text{otherwise} \end{cases}$
26: **for all** $i \in \{t-1, .., t-t_{last\_up}\}$ **do**
27: $R = r_i + \gamma r$
28: $\Delta_\pi^A = \Delta_\pi^A + \nabla_{\theta_\pi^A}\{\log \pi(a_i|s_i, \theta_\pi^A)\}[R - V(a|s_i, \theta_v^A)]$
29: $\Delta_v^A = \Delta_v^A + \nabla_{\theta_v^A}\{R - V(a|s_i, \theta_v^A)\}^2$
30: **end for**
31: **end while**{*episode finished*}
32: **update** θ_π^g using Δ_π^A
33: **update** θ_v^g using Δ_v^A
34: **end for**

which means repeating the same action for several subsequent frames without computing the updates on them, as done in [56]: besides increasing the simulation speed (backpropagation is computed in a fraction of the frames), it has been demonstrated that sometimes the use of action repeat improves learning ([96]). This is due to an increased capability of learning associations between temporally distant state-action pairs resulting from the incremented power given to each action for affecting the state. The effects of this technique for the proposed simulator is analyzed in the ablation studies of Section 3.7.7.

3.5 State space

As already said, having a visual state space increases the flexibility of the simulator; however, the use of convolutional layers makes the system not suitable for handling independent scalar inputs, which could turn useful to enhance the visual input with additional information (e.g. the exact speed of the agent) or to tune the desired behavior of the agent. Processing this kind of input would be possible using fully-connected layers, but it would become inefficient to directly process the visual input by this mean, both in terms of resources (the size of the neural network would grow exponentially) and in terms of sample efficiency.

In order to obtain a system able to exploit both types of informative streams, the module has been shaped so that both types of input, **visual** and **numerical**, are processed by a dedicated pipeline before being merged together, as elucidated in section 3.6.

The hint of mixing two different input streams came from [97], in which a low-dimensional vector of *measurements* is used to define the goals of the agent and drifts the training phase toward a supervised setting by learning how these values are affected by the performed actions. This solution fits problems which embed naturally the concept of relevant and observable set of measurements linked to the agent goal (such as health and score in a video-game), but it is not suitable in scenarios with only sparse and delayed feedback. Hence, in this work it has been used the numerical input simply as an additional input, and followed standard temporal-difference learning to

train the agents.

3.5.1 Visual input

The agent field of view is limited by the range of its perception sensors, which very probably does not cover the whole environment but only a portion of it, called its *view*. Since it has been used a mid-level representation, the input image does not have to be raw *RGB* images, but it is divided in multiple single-channel semantic layers, each one carrying a specific type of information, such as the navigable space, the route and the obstacles. Having a segmented input clearly reduces the input complexity, and permits to add different semantic layer at will dependently on the task. The layers used in the application case of this work are shown in Figure 3.4.

Furthermore, a single snap-shot of the surrounding of the agent is not sufficient for a full comprehension of the situation: indeed, speed and acceleration of other vehicles, for example, cannot be deduced unless the module has memory (e.g. using recurrent neural networks) or past motion is encoded in the views, as in [98].

A different solution, adopted in this work, is the use of the frame-stacking technique as in [75]: instead of using a single view as input, it is given to the system a sequence of the v most recent views seen from the agent; we set $v = 4$ as in [56] so that relative speeds and accelerations estimation of the other agents becomes feasible.

3.5.2 Numerical input

The numerical input component has a twofold motivation; in the first place it permits the enrich the environment description by augmenting it with information which are not directly perceivable from the sequence of views, such as the absolute speed of the agent or the distance to the goal position, allowing a better understanding of the scene.

Secondly and more importantly, the numerical input can be used to shape agents behaviors by defining the value of some driving-style parameters. This is achieved by teaching to the agent a correlation between these inputs and the to-be-obtained rewards, permitting a partial perturbation of the agent state at will. For example, if the agent would learn that whenever a dedicated parameter is set it would be rewarded

positively for making accidents, this would enable the possibility of adding suicidal drivers to the simulation.

In this work, something more soft has been used to create different driving styles, that is the possibility to set a value of aggressiveness and a desired cruising speed for the agent, as explained in Sections 3.7.5 and 3.7.6 respectively.

3.6 Network architecture

As previously explained, the different structure of the two input streams requires that two different pipelines are initially used to process each type of information, producing two feature-vectors containing an initial separate description of the numerical and visual part. The visual input pipeline consists on two convolutional layers followed by a fully-connected layer, while the numerical input is processed by two fully-connected layers.

After this first processing, the two feature vectors are joined together and processed by an additional fully-connected layer. This way, features coming from visual and numerical streams are fused and processed together, producing a feature vector which describes also the joined interaction between these two different information sources. Every layer is followed by a *ReLU* non-linearity function which was described in Section 2.3.

The delayed multi-agent A3C algorithm, explained in Section 3.4, requires that both action probabilities and value of the current state will be estimated. Instead of using two similar but separate networks for accomplishing these two tasks, it has been used a unique neural networks with two different heads, each one destined to a single task; this is possible since the input of the networks would be the same, and so are the majority of the features to be learned. Using a single network, the number of parameters to be trained is halved, leading to speed up in the learning process.

A visual representation of the network architecture is shown in Figure 3.2.

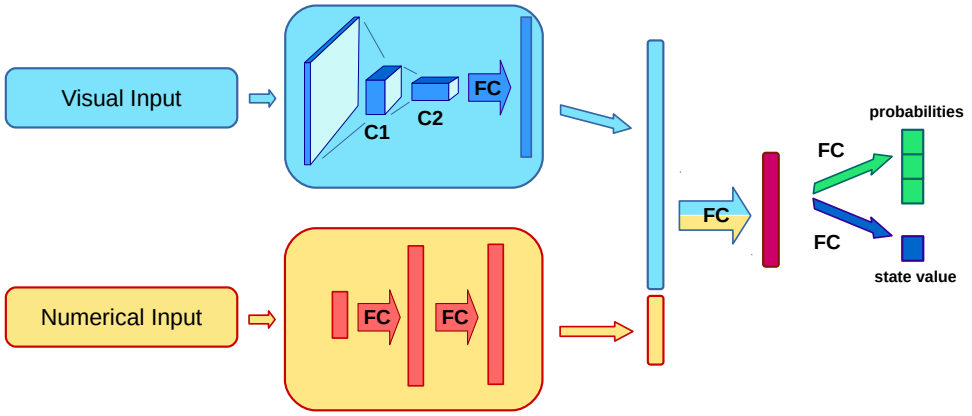
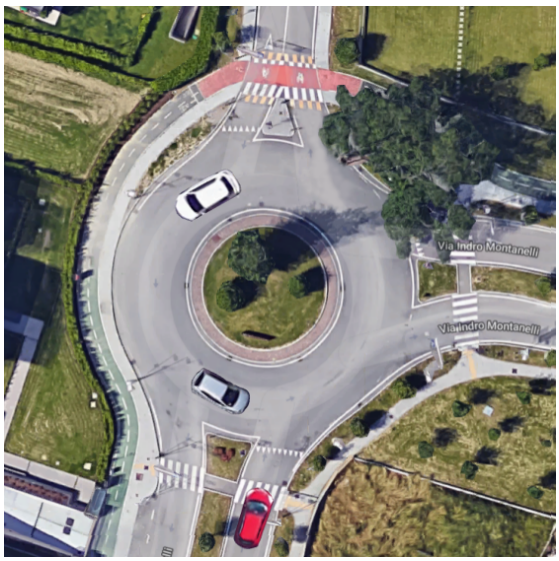


Figure 3.2: Architecture of the neural network used for computing action probabilities and estimating the value of the state

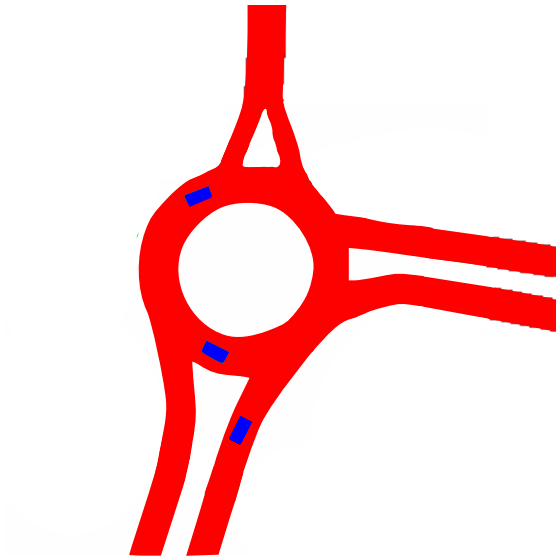
3.7 Case of study: roundabout scenario

Among the several cases in which the proposed approach can be used, in this work it has been adopted for simulating vehicles driving in a roundabout. This scenario requires extensive cooperation between participants, especially in case of busy traffic conditions, where a simple and cast-iron following of traffic laws may lead to undefined waits for vehicles without right of way. In fact, in those cases, human drivers tend to push their way after having waited long enough in an intersection, in order to avoid blocking it; on the other side, drivers with the right of way let sometimes vehicles pass: this implicit negotiation generally happens without accidents (unluckily not always..).

However, learning and simulating such behaviors is a very challenging control task, especially if it is needed a mathematical model of humans intentions. For this reason, experiments trying to tackle this problem in a multi-agent reinforcement learning fashion represent an interesting and practically useful research area for the self-driving car development.



(a) Top-view of the real roundabout



(b) Synthetic representation

Figure 3.3: Top-view of the roundabout used as application case for the discrete action-space setting. The real roundabout and its vehicles **(a)** are translated into the synthetic representation **(b)**

3.7.1 State space

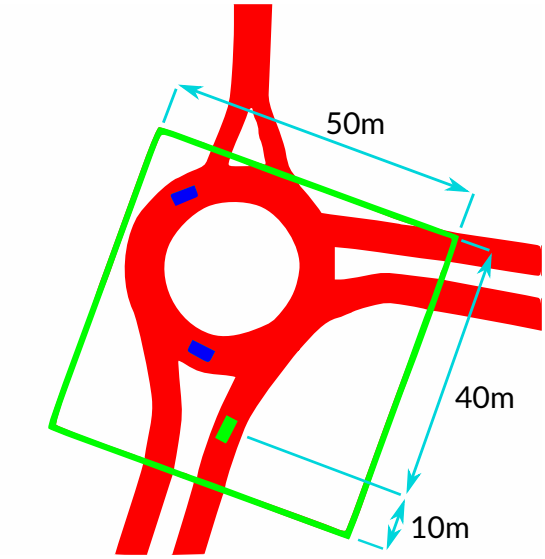
The environments used for the experiments are synthetic representations of real roundabouts, as the example shown in Figure 3.3. Agents, as explained in section 3.5, are given fixed-size synthetic images of their sensed surrounding used to interpret the scene. For this reason, a proper field of view has to be set, meaning that some of the detected vehicles might be outside of the input image. For the roundabout simulation, a $50m \; x \; 50m$ field of view has been used, positioned around the vehicle as shown in Figure 3.4a.

These surrounding regions are translated into three-channel images, where each channel corresponds to a different semantic information content; in these experiments the layers included are:

1. **navigable space layer**, containing the portion of the surrounding of the vehicle in which vehicles are allowed to drive, and is generally extracted from high-level maps; this layer is shown in Figure 3.4b;
2. **obstacles layer**, which is made of all vehicles and obstacles seen from the agent, including the agent himself; this information is obtained from the perception module of the self-driving vehicle and an example is give in Figure 3.4c;
3. **path layer**, that shows to the agent its assigned path, and it is given from the high-level planner module of the vehicle; this layer is shown in Figure 3.4d.

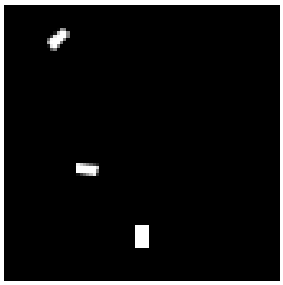
In addition to the semantic layers, the agent has in input some numerical parameters, useful for both enriching the information about the perceived scene and shaping the agent behavior. The numerical parameters adopted for the simulation are:

1. **agent speed**, the instantaneous speed of the agent;
2. **target speed**, the desired cruising speed that would be reached unless prudence is needed;
3. **elapsed time ratio**, the ratio between the elapsed time from the beginning of the episode and the fixed time limit for reaching the goal.
4. **distance to the goal**, the distance to be traveled for reaching the agent' goal.

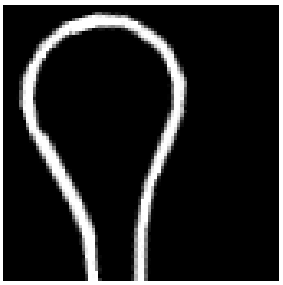


(a) Area perceived from the green vehicle

(b) Navigable space channel



(c) Obstacles channel



(d) Path channel

Figure 3.4: **(a)** shows the region perceived from the green agent, which is translated into three different semantic layers which are used as input to the system: the navigable space layer **(b)**, the obstacles layer **(c)** and the path-to-be-followed layer **(d)**

3.7.2 Action space

The output of the system can be the values of acceleration and steering-wheel angle for the agent' vehicle. The selected actions would be executed for a fixed time interval and have to be picked up from either a discrete or a continuous set. In the training phase, the selected action is repeated for several frames in order to improve and stabilize the learning process (the so called *action repeat* technique); at test time, however, actions are not repeated, so that they can be selected with a finer resolution and thus guaranteeing better results, as explained in the ablation studies in Section 3.7.7.

Two different settings have been analyzed, the first one aimed at controlling the longitudinal motion along a predetermined route, in which the agent can pick up its actions from a discrete set, and the second one in which the agent has full control of the vehicle and can select acceleration and steering angles from a continuous set of actions.

Discrete setting

In the discrete setting the agent has at disposal three actions, each one linked to a different acceleration value, while the steering angle is not selectable and corresponds to the value needed to follow the assigned road; this means that the agent is free of deciding its longitudinal motion along its route, but it is not allowed to vary the lateral motion. The actions at disposal allow the agent to maintain its actual speed, accelerate or decelerate; thus, the policy head of the neural network gives the probabilities of executing each of these actions. Using a stochastic policy means that all actions have a non-zero probability dependent on how good they are expected to be, making good actions more probable but at the same time letting it possible to execute actions which are not believed to be the best, so that exploration is guaranteed even if on-policy learning is adopted.

The policy head of the network is designed to output log-probabilities instead of pure probabilities, in order to simplify the updates as shown in Equations 2.10 and 2.11; this is achieved by applying a *logarithmic soft-max function*[2] to the output of the

[2]Calling $o_i..o_n$ the output of the last layer of the network linked to the actions $1..n$, the logarithmic

last layer.

Continuous setting

In the continuous setting, the agent can decide both its longitudinal acceleration and steering angle, meaning that it is free of deciding the route to follow. In case of continuous action spaces it is not possible to output directly the action probabilities due to the infinite many possible actions. In [62] this is solved by using a deterministic output, but it requires the use of an off-policy reinforcement learning algorithm for guaranteeing exploration, which brings the already mentioned problems related to the use of experience replay in the multi-agent domain.

In this work, it has been followed the hint given in [57] for extending classical A3C to the continuous domain: instead of outputting action probabilities, the policy head of the network provides a vector μ representing the mean of a multivariate *Gaussian* distribution with spherical covariance which is used as probability density function for selecting actions in a continuous multidimensional set, where each dimension correspond to a different controllable output; this is achieved by applying to each output of the last layer of the policy head of the network the *hyperbolic tangent* function (see Section 2.3) with amplitude equal to the range of the action it controls, instead of the *logarithmic soft-max* used for discrete probabilities. The standard deviation of each output, σ, which is closely related to the exploration pursued by the agent, is diminished throughout the learning phase until a lower bound representing a quasi-deterministic behavior. This way, when at the beginning the agent has no clues about the environment dynamics, it selects actions following an almost random policy, so that every action has the same chance to be tried; meanwhile the agent is acquiring knowledge of the environment and its dynamics, actions would be selected every time closer (in terms of probability) to the mean predicted by the system, which represents the *greedy* action.

---

soft-max function is defined as:

$$\log p_i = \log \frac{e^{o_i}}{\sum_{j=1}^{n} e^{o_j}} \tag{3.1}$$

As just said, in this continuous setting the neural network is not estimating action probabilities, but rather a set of variables, the vector mean μ, which affects the probability distribution over the actions; for this reason Equation 2.10, which defines the updates of the neural network parameters, has to be redesigned. Knowing that the probability density of the Gaussian distribution is given by:

$$N(x|\mu,\sigma)) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.2}$$

the gradient of the logarithm of the density function respect to the network parameters, needed for computing Equation 2.10, becomes:

$$\nabla_{\theta^\mu}\{log\pi(x|\mu(\theta^\mu),\sigma)\} = \nabla_\mu\{logN(x|\mu,\sigma)\}\cdot\nabla_{\theta^\mu}\mu(\theta^\mu) =$$

$$\nabla_\mu\{log(\frac{1}{\sqrt{2\pi}}) - log(\sqrt{\sigma^2}) - \frac{(x-\mu)^2}{2\sigma^2}\}\cdot\nabla_{\theta^\mu}\mu(\theta^\mu) = \frac{(x-\mu)}{\sigma^2}\cdot\nabla_{\theta^\mu}\mu(\theta^\mu) \tag{3.3}$$

where $\mu(\theta^\mu)$ is the output of the policy head of the neural network after the hyperbolic tangents, whose allowed ranges are given in the implementation details of Section 3.7.4.

3.7.3 Reward design

The reinforcement learning appeal arises from the fact that it is not needed to tell the agent at every time instant what is the right action to execute, but rather give to the agent rewards, either positive or negative, whenever an important event takes place, such us accomplish its goal or ending up in an unwanted situation. These rewards could even be very sparse and computed automatically without any human intervention, and the agent will learn step by step a good policy to obtain good rewards without falling into negative situations.

Depending on how the reward is shaped, agents may develop either competitive or cooperative conducts ([99]): if agents are awarded for impeding the others to reach their goal, competitive behaviors are likely to emerge. On the other hand, if agents are penalized when obstructing an other agent, cooperative behavior will be the best way to maximize their rewards; moreover, when parameters are shared among agents, this

would be even enforced, since adopting cooperative actions by an agent versus the others, means that also the other agents will cooperate with the same agent.

The latter scenario is the one adopted in the roundabout application case: agents are rewarded positively whenever they reach theirs goals, while both agents involved in an accident are rewarded negatively.

However, this weak supervision cannot be limited to positive and negative rewards related to the maneuver outcome, but some additional secondary rewards have to be taken into account in order to make it possible for the agent to learn also those behaviors which cannot be deduced by simple trial-and-error, such as the respect of traffic laws and the driving style shaping.

Discrete setting

In the discrete setting, the goal was to learn the interaction between vehicles for negotiating the insertion even in busy traffic conditions; since agents cannot drift from the assigned trajectory, the test-case was a small single-lane roundabout. For this task, the reward received at a generic time step t is made of three main terms:

$$r_t = r_{terminal} + r_{violation} + r_{speed} \tag{3.4}$$

$r_{terminal}$ is the main reward linked to the final outcome of the agent, and it differs from zero only if t is a terminal state for the current episode; the values it assumes are:

- $+1$: if the agent reaches its goal;
- -1: if the agent crashes against another agent;
- -1: if the available time expires;
- 0: if t is a non-terminal state.

The negative reward given in case of expiration of the available time has been added in order to avoid agents to stay stopped indefinitely for avoiding possibly negative rewards; moreover, it has been used to shape the aggressiveness as explained in Section 3.7.5.

$r_{violation}$ is a soft penalization given to the agent whenever it breaks some traffic laws. The aim of this negative reward is to let the agent understand that these situations are unwanted even if they will not end up in accidents or other terminal states: for example, it is used to avoid the agent to stay too close to the vehicle in front and thus breaking the safety distance, which would result in a simulation with unnatural vehicles behavior. At the same time, it is used to penalize the agent whenever it enters the roundabout cutting off in front of an already inserted vehicle in an unsafe manner, thus inducing in the learning process the concept of right of way: whenever two agents would have to negotiate the insertion, the one with the right of way would have more chances to pass first, unless the one without right of way speeds up in order to let enough space. Calling d_v the distance traveled from the vehicle v in one second, the $r_{violation}$ penalization values for the agent a are:

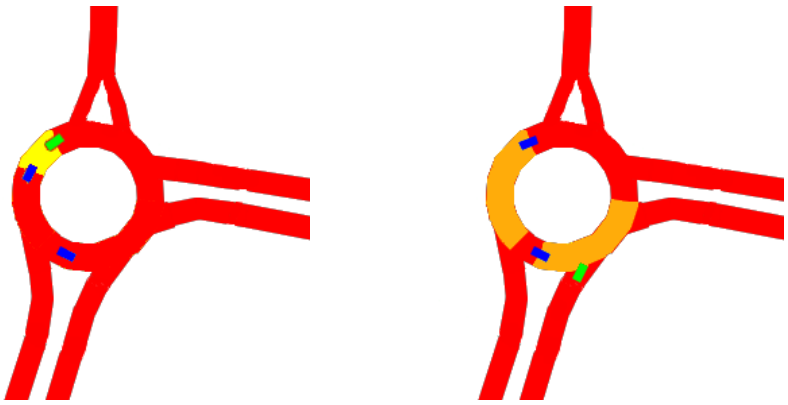
- $-k_{sd}$: if a violates the safety distance with the vehicle in front, unless the latter is entering the roundabout and it is cutting in front of a. The safety distance in this case is taken equal to d_a, and the associated region is depicted in yellow in Figure 3.5a;

- $-k_y$: if the agent fails to yield when entering the roundabout to an already inserted vehicle v. This happens when a crosses the region in front of v whose length is taken equal to $3 \cdot d_v$, as shown from the orange region in the example of Figure 3.5b;

- 0: if none of the above occur.

The last term, r_{speed}, is a reward dependent on the speed of the agent; it is maximized when the current agent speed (s_a) is equal to its assigned desired cruising speed (s_t) according to the following criterion:

$$r_{speed}(s_a, s_t) = \begin{cases} \frac{s_a}{s_t} \cdot k_{sp}^{+} & \textbf{if } s_a \leq s_t \\ k_{sp}^{+} - \frac{s_a - s_t}{s_t} \cdot k_{sp}^{-} & \textbf{if } s_a > s_t \end{cases} \tag{3.5}$$



(a) Safety distance violation (b) Right of way violation

Figure 3.5: Situations which lead to a r_{danger} penalization for the green vehicle due to violations of traffic laws. In **(a)**, the agent receives a negative reward due to a safety distance breaking, while in **(b)** it receives a penalization due to a right of way violation

Continuous setting

Learning to drive in the continuous task is considerable more difficult than in the discrete scenario, since agents not only have to decide how to execute the assigned trajectory, but also have to learn how to select the trajectory itself. Indeed, agents are given only the area in which they are allowed to navigate instead of a precise trajectory, consisting of the navigable space around their assigned routes[3].

To better evaluate this setting, the test-case chosen is a wide three-lane roundabout, so that agents are able to overtake each other and choose among a larger pool of possible trajectories. Thus, the learning efforts for this scenario have been directed towards the navigation phase instead of the induction of behaviors which mimic precisely those of humans. For this reason the aim of the rewards is to guide the agent exploration by inserting external knowledge which penalizes zigzag trajectories. The

[3]The route has to be seen just as an indication of the direction to follow, like the information given to the driver by a common GPS, instead of a precise trajectory as in the discrete setting

full reward is given by this equation:

$$r_t = r_{terminal} + (r_{angle} + r_{center}) \cdot r_{speed} \tag{3.6}$$

$r_{terminal}$, as before, is the reward received by the agent when the episode terminates, either because the agent accomplished its goal or because it ended in a not acceptable state, as in case of an accident. Its value is the same as in the discrete setting, with the addition of a negative (-1) reward given if the agent drives outside its assigned navigable space.

r_{angle} encourages the agent to maintain its heading direction close to the direction of the virtual trajectory made by the center points of the assigned route, as shown in Figure 3.6a; calling θ_r and θ_{car} the heading of the center-lane trajectory and that of the agent respectively, its value amounts to:

$$r_{angle} = (|\theta_r - \theta_{car}|) \cdot k_{angle} \tag{3.7}$$

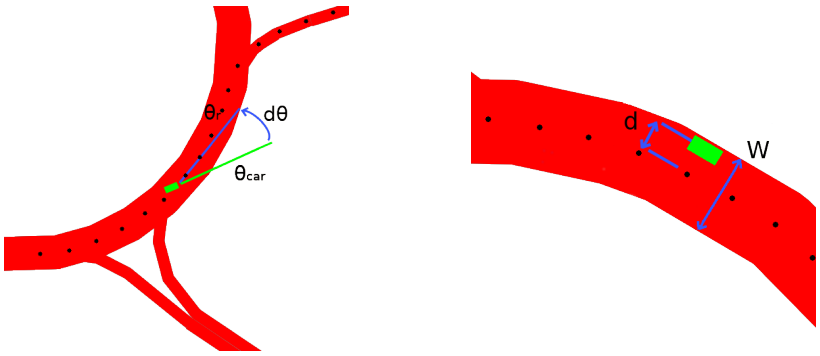
Similarly, r_{center} encourages the agent to stay close to the center of the lane, thus penalizing trajectories which go at the margins of the road as shown in Figure 3.6b; its value follows this function:

$$r_{center} = (1 - \frac{|d|}{W/2}) \cdot k_d \tag{3.8}$$

where d is the distance from the center of the lane and W is the width of the lane given from the high-level maps.

Finally, r_{speed} is a multiplying factor which modulates the importance of the sum of the previous two depending on the current speed of the agent (s_a) respect to the target speed (s_t); it equals:

$$r_{speed}(s_a, s_t) = \begin{cases} \frac{s_a}{s_t} & \textbf{if } s_a \leq s_t \\ 1 - \frac{s_a - s_t}{k_{speed}} & \textbf{if } s_a > s_t \end{cases} \tag{3.9}$$



(a) Heading angle shift (b) Lane-center shift

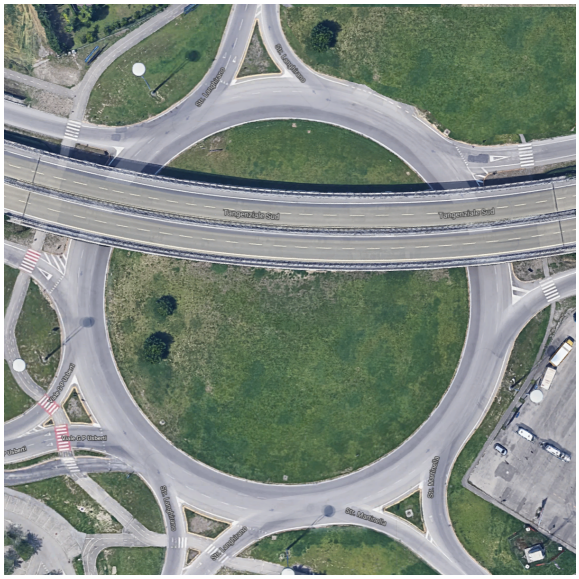
Figure 3.6: **(a)**: angular shift between the heading of the vehicle and the direction of the trajectory passing for the lane-center points. **(b)**: positional shift between the vehicle and the lane-center

3.7.4 Implementation details

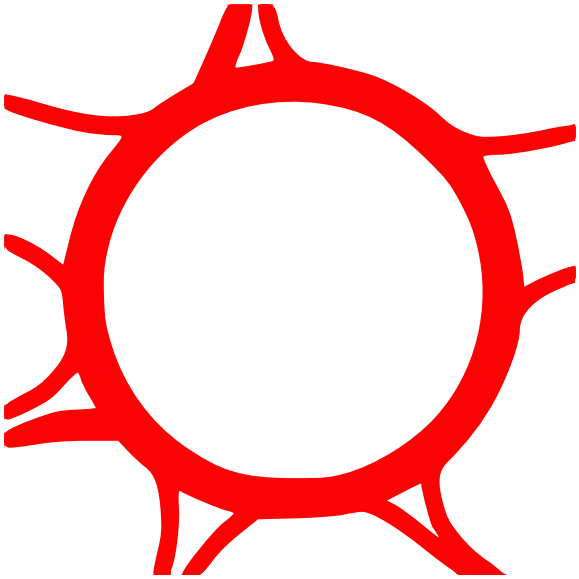
The roundabout taken as application case for the discrete task is a three-entries roundabout located in the district *Parma Mia* of the Italian city of Parma, at coordinates $N\,44°\,47'\,02.3$", $E\,10°\,18'\,04.5$". A top view of the roundabout is shown in Figure 3.3a, while in Figure 3.3b is shown its synthetic counterpart. For the continuous setting, the simulation was based on the wider 6-entries roundabout located in the Campus area of the same city, at coordinates $N\,44°\,46'\,01.4$", $E\,10°\,19'\,24.6$", shown in Figure 3.7.

The $50m\ x\ 50m$ surrounding region of each vehicle (Figure3.4a), is scaled into images having dimension $WxH = 84x84$ which, as explained in Section 3.7.1, are made of $c = 3$ separated semantic channels, namely the navigable space layer, the obstacles layer and the path layer; these images are obtained using the Cairo graphics library ([100]) which permits to directly draw on raw memory sections with high efficiency. Considering that the input is formed by the last $m = 4$ perceived surroundings, and that there are $p = 4$ additional input parameters, the full input dimensions are:

$$m\ x\ c\ x\ (layer\ W)\ x\ (layer\ H)\ +\ p = 4\ x\ 3\ x\ 84\ x\ 84 + 4 \tag{3.10}$$



(a) Top-view of the real roundabout



(b) Synthetic representation

Figure 3.7: Real **(a)** and synthetic representations **(b)** of the roundabout used as application case for the continuous setting

module	type	input dimension	channels	kernel size	stride
visual pipeline	C	84 *x* 84 *x* 3 *x* 4	16	8 *x* 8	4
	C	20 *x* 20 *x* 16	32	4 *x* 4	2
	FC	$9 \cdot 9 \cdot 32$	256	–	–
numerical pipeline	FC	3	20	–	–
	FC	20	20	–	–
shared layer	FC	$256 + 20$	10	–	–
actor output	FC	10	3	–	–
critic output	FC	10	1	–	–

Table 3.1: Values of the network hyperparameters used for the simulator

The inputs are processed by a convolutional neural network whose architecture is shown in Figure 3.2: the hyperparameters of each layer, both convolutional and fully-connected, are shown in Table 3.1.

In the discrete action-space setting (3.7.2), agents have at disposal three different actions, which correspond to values for accelerating, decelerating or maintaining the current speed; these values are set to be comfortable values for humans, and are $a_a = (1m/s^2)$ in case of acceleration and $a_d = (-2m/s^2)$ in case of deceleration. The number of parallel environments used for increase the number of agents learning simultaneously was set to $n_{env} = 4$; during learning, the density of agents present in each environment was changing through time so that different traffic condition, from zero to very busy traffic, were encountered. Agents started each episode from a random position inside one of the three entry lane, and finished it once reached the end of their randomly assigned exit lane.

In the continuous setting it was sufficient to use a single environment instance ($n_{env} = 1$), since the test-case roundabout is much wider thus permitting the presence of enough agents simultaneously. Agents are free of choosing theirs accelerations from a continuous set within the same bounds of the discrete setting, namely $a \in [a_a, a_d]$.

For modeling the vehicle motion it has been adopted the kinematic bicycle model ([101]) because of its simplicity and adherence to the non-holonomic constraints of

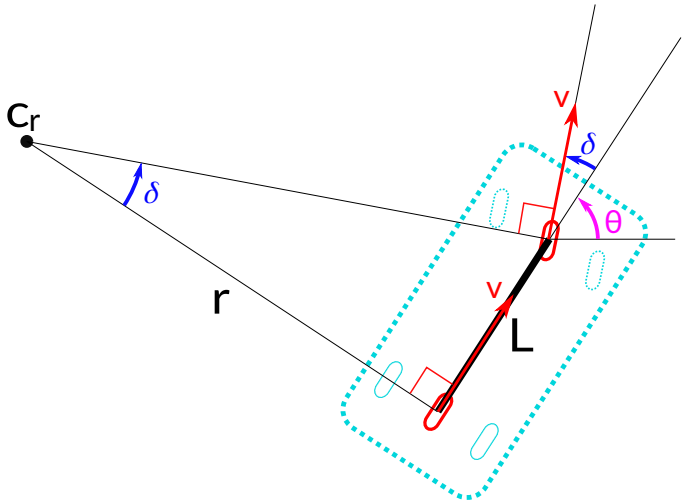


Figure 3.8: Representation of the bicycle model, in which each pair of wheels of the axles of the car (shown in light blue) are lumped into a single virtual wheel positioned in the middle of the axle (shown in red). The steering angle δ is strictly related to the radius of rotation r, defined as the distance between the instantaneous center of rotation c_r and the virtual posterior non-steering wheel; c_r is placed at the intersection of the perpendiculars of the two virtual wheels

a car. In this model the two wheels of each axle, front and rear, are lumped into a single wheel positioned in the middle of the axle; moreover it is assumed valid the so called no-slip condition, meaning that the wheels can not move laterally or slip longitudinally either.

In this simplified scenario, shown in Figure 3.8, the vehicle is rotating around an instantaneous center of rotation (c_r) positioned at the intersection of the perpendiculars of the two wheels, with a radius of rotation equal to the distance between c_r and the

rear non-turning wheel. Thanks to this model it is possible to write:

$$\tan\delta = \frac{L}{r} \tag{3.11}$$

$$\dot{\theta} = \frac{v}{r} = \frac{v \cdot \tan\delta}{L} \tag{3.12}$$

The equations show that the angular velocity $\dot{\theta}$ can be easily mapped with the radius of rotation r instead of the steering angle δ; a more clever choice is the use of the so called *curvature* κ, defined as the inverse of the radius of rotation, which constitutes a bounded and continuous control variable instead of r, which may go from $-\infty$ to $+\infty$ in case of passing from a left to a right turn; in the experiments the allowed interval for κ has been chosen as the average interval of a medium size car, which is $\kappa \in [-0.2, 0.2]$.

Therefore, the motion equations of the system, having as inputs acceleration a and curvature κ and as origin the center of the rear axle, are:

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \frac{v}{r} = \kappa \cdot v \\ \dot{v} = a \end{cases} \tag{3.13}$$

The control inputs, in both discrete and continuous settings, are maintained for the duration of a time step, which in the simulation amounts to $t_{step} = 0.1s$; in order to improve the learning process, in the training phase action are repeated for the subsequents 4 time steps following the action repeat technique.

Regarding the reward functions, in Table 3.2 are given the values of the constants used in the experiments.

3.7.5 Aggressiveness tuning

During training, agents learn to reach their goal within a time limit which is known to them thanks to the elapsed time ratio input; moreover, another additional information is provided to the agents in order to enrich their knowledge and thus regulate their

Discrete task				Continuous task		
$\mathbf{k_{sd}}$	$\mathbf{k_y}$	$\mathbf{k_{sp}^+}$	$\mathbf{k_{sp}^-}$	$\mathbf{k_{angle}}$	$\mathbf{k_d}$	$\mathbf{k_{speed}}$
0.05	0.05	0.001	0.03	0.005	0.005	2

Table 3.2: Values of the reward constants used in the simulation experiments

speed for accomplishing the task, that is the distance to the goal. The presence of these two inputs permits to modify the temperament of every agent, and simulate drivers with different aggressivenesses.

It has been tried to achieve the same results using the time-left input only, therefore encouraging the agent to speed up whenever the time limits is approaching to expire, without providing any information about the distance which has still to be traveled; however, the results were scarce: indeed, it was not feasible for the agent to learn to speed up without the direct knowledge that this acceleration makes the goal closer, information which is not provided from the images. Nevertheless, this became possible when the input has been enforced with the remaining distance information, confirming the power of the visual-numerical coupling.

The idea is that, at training time, agents are given a fixed interval of time for reaching their goals, at the end of which a negative reward is given and the episode terminated prematurely if the goal has not been reached; agents can see both the elapsed time and the distance to-be-traveled, thus learning to adapt their policy to fit the schedule. However, at test time, both these inputs can be kept fixed and used to shape the behavior of the agent: in particular, it has been explored the scenario in which the distance to the goal is maintained always constant, while the elapsed time ratio is varied in the $[0, 1]$ interval and kept fixed for the whole episode.

Values close to 1 induce the agent to drive faster in order to avoid the expected negative return caused by running out of time. For the same reason, values close to 0 tells the driver that it still has much time for reaching its goal, therefore it is not a problem yielding to other vehicles. This way, the elapsed time ratio acts as an 'aggressiveness level' for the agent.

An experiment has been set up in order to check the validity of this approach

for the discrete setting: an agent, having a fixed value of aggressiveness a^*, is tested for several episodes inside the roundabout, which was populated by 6 other vehicles, value which has been chosen in order to avoid traffic jams albeit guaranteeing enough traffic for the test. These vehicles featured random and variable-through-time values of aggressiveness, so that different scene configurations can take place. In order to determine the behavior of agents with different values of a^* two different features have been registered: the ratio of positive episodes r_p in which the agent reached the goal without accidents, and the average speed s_a maintained during the episodes; the results are shown in the graph in Figure 3.9.

From the graph it is evident that, by increasing $a*$, the driver behavior will shift towards a more risky configuration, since both average speed and probability of accidents grow. Moreover, it is interesting to note that values of aggressiveness outside the training interval $[0, 1]$ produce consistent effects to the agent conduct, intensifying its behavior even further.

3.7.6 Target speed tuning

A similar approach to the aggressiveness tuning has been adopted in order to modify the desired cruising speed of every single agent. Indeed, at training time, a random value of target speed s_t is assigned to each agent, which is encouraged to follow it thanks to reward term r_{speed} introduced in Section 3.7.3; s_t was drawn at every episode from a uniform distribution between $[5, 8]\frac{m}{s}$, but agent may still decide to surpass it if they believe it would be advantageous, such as accelerating for stopping the insertion of another vehicle by a particular aggressive driver.

In order to evaluate the effects of such target speed tuning, an evaluation scenario similar to the aggressiveness tuning one has been set up: agents with different values of s_t, but same aggressiveness level equal to 0.5, are tested along several episodes in a roundabout populated by 6 vehicles with random aggressiveness and target speed values. As before, values for the positive episodes ratio r_p and average speed s_a are recorded and shown in the graph of Figure 3.10, which show an almost linear dependency between s_a and the target speed input, while r_p remains substantially unchanged, meaning that the risks taken does not vary. Even in this case, it is interesting

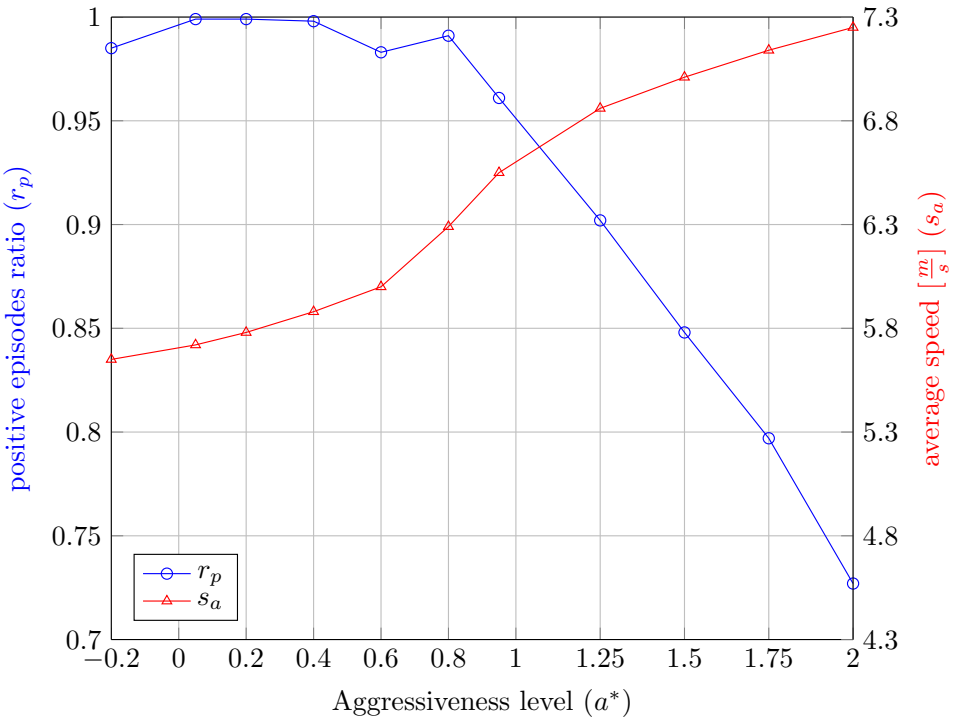


Figure 3.9: Effects of the aggressiveness input on the behavior of the agent. The blue plot shows the positive-ending episodes ratio varying the aggressiveness input; similarly, the red plot shows how this input affect the average speed maintained by the agent

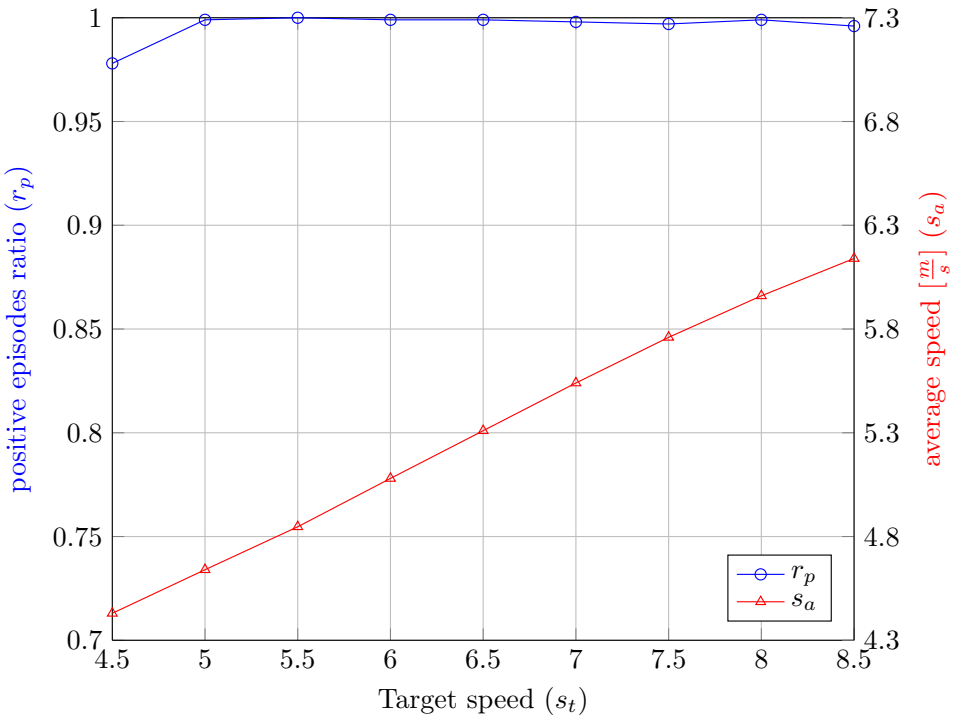


Figure 3.10: Effects of the target speed input on the behavior of the agent. The red plot shows the positive-ending episodes ratio varying the target speed; similarly the blue plot shows how this input affect the average speed maintained by the agent

to note that values of s_t outside the training range $[5,8]$ induce consistent agent behaviors.

3.7.7 Ablation studies

Multiple environment instances

It has been studied the effectiveness of the use of multiple environment instances for the discrete setting, in order to understand if the addition of independent learners improves or destabilizes the learning process. For this purpose, it has been compared the learning evolution of an agent trained in a single environment with another trained

simultaneously in 4 copies of the environment. In order to have a feedback of this evolution, it has been tracked the positive episodes ratio r_p by an *exponential moving average* $\overline{r_p}$ updated at the end of each episode e, which is defined as:

$$\overline{r_p}(e) = (1-\alpha)\cdot\overline{r_p}(e-1) + \alpha\cdot outcome(e) \tag{3.14}$$

where $outcome(e)$ is a binary variable equal to 1 if the episode ended successfully, and 0 otherwise, while the smoothing factor α has been set to 0.01. The resulting curves, shown in Figure 3.11, tell that when using multiple instances the learning process is more stable and leads to slightly better overall performances; even if in both cases the model converged, they confirm that the reduction of the correlation between updates brings a positive effect.

It is interesting to note that in the multi-instance scenario agents start improving their performances at a later stage, but, once started, with a faster pace. Our interpretation for this behavior is that the higher rate of asynchronous updates coming from pseudo-random policies initially makes the learning process more arduous, since the global policy receives simultaneous updates from several agents whose behavior is still infant; however, when the policies of agents start to gain sense, their updates will be directed strongly toward a common goal due to the reduced search space.

Action repeat

A similar analysis to the multi-environment study has been carried out for the evaluation of the effects of using the action repeat technique. Again, an agent is trained in the scenario featuring a discrete action space without repeated actions, both in the single and multi-instance cases. The learning curves of such agents, shown in Figure 3.12, demonstrate that in the single environment setting the agent is still able to learn the task without evident dissimilarities; however, when multiple environment are used, the agent is not able to learn the task at all, making the use of the action repeat technique necessary.

The reason of this inability is due to the pseudo-random exploration of the agents in the initial phase, which makes the global policy unstable. This weakness affects mainly the multi-instance scenario, in which the number of asynchronous agents

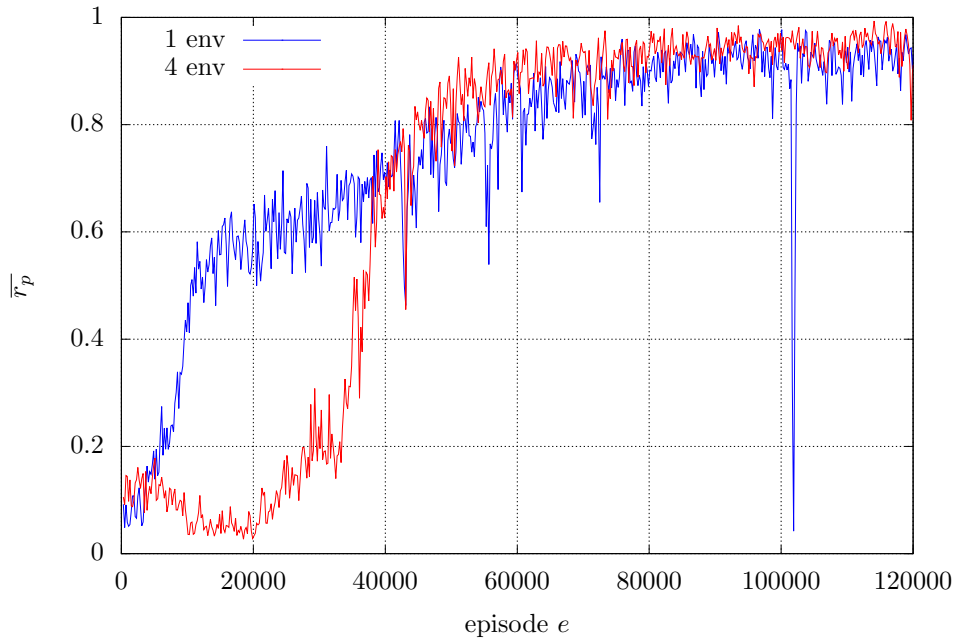


Figure 3.11: Moving average $\overline{r_p}$ of the positive episodes ratio when adopting a single instance (in blue) and 4 concurrent instances (in red); in both learning processes action were repeated 4 times. The exponential moving average is obtained using Equation 3.14

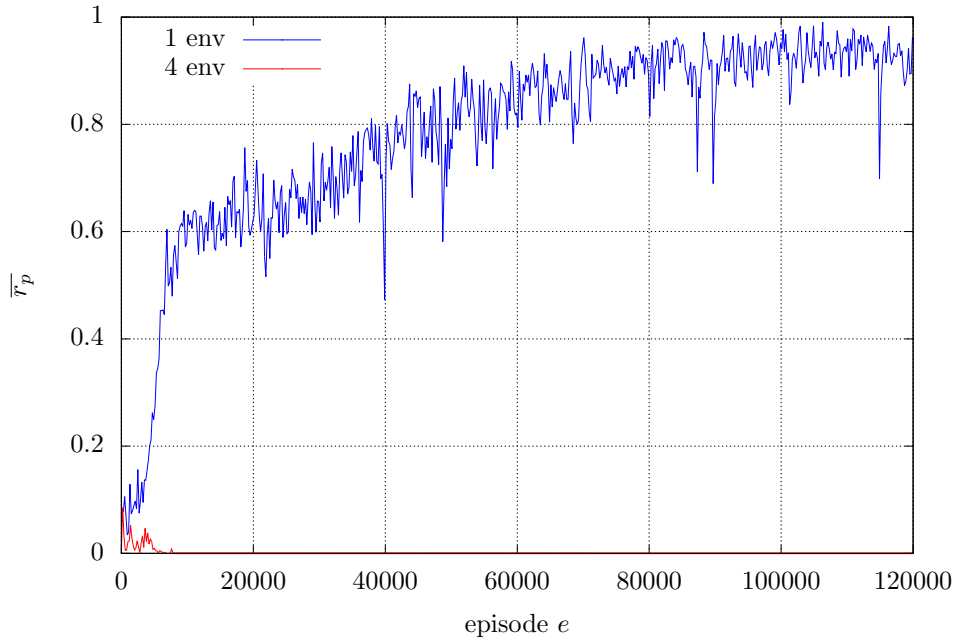
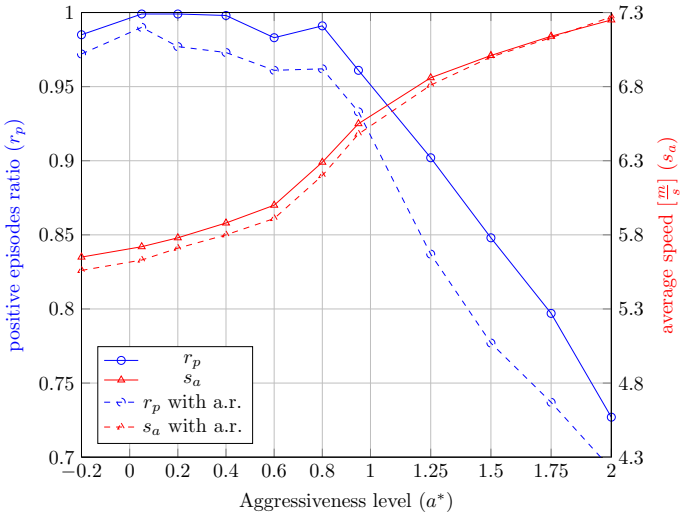


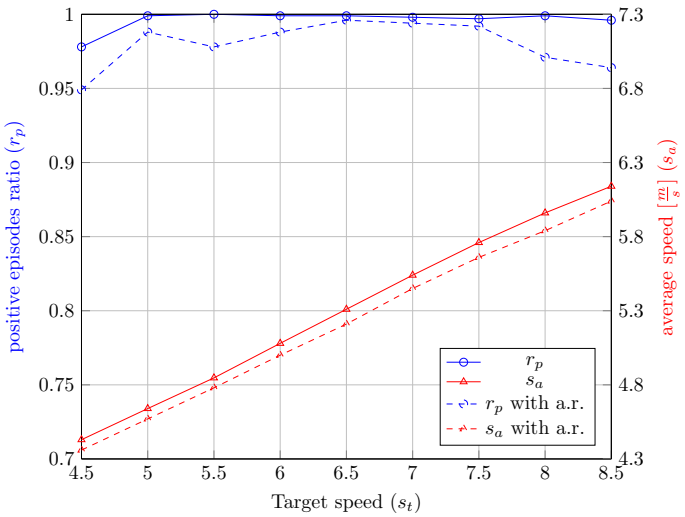
Figure 3.12: Moving average $\overline{r_p}$ of the the positive episodes ratio when actions are not repeated, both in single (in blue) and multi-instance (in red) scenarios. The exponential moving average is obtained using Equation 3.14

is greater, causing a deterioration of the policy up to the point of suppressing any exploration behavior, therefore blocking the learning process; instead, the use of action repeat helps in turning those exploration policies less schizophrenic and more effective, thus permitting to learn a policy which gradually gains sense.

At test time, it is not longer needed to stabilize the exploration policies, and repeating actions would only diminish the frequency at which agents are allowed to change actions; indeed, the empirical results obtained varying both aggressiveness and target speed, shown in the graphs of Figure 3.13, tells that the positive episode ratio increases if actions are not repeated, while the average speed trend remains very similar. This result has been achieved for an agent trained with action repetition, meaning that it is possible to take advantage of both approaches.



(a) Aggressiveness tuning



(b) Target speed tuning

Figure 3.13: Effects of using repeated actions in the test-time performances. In **(a)** are shown positive episode ratio r_t and average speed s_a values for different aggressiveness levels, both with (dotted line) and without (solid line) repeated actions; analogous results are given in **(b)** for different target speed values. From the graph it is clear that the use of action repeat at test time affects negatively the performance of the system

Chapter 4

Maneuver Execution using Deep Reinforcement Learning

Chapter 3 introduced a modern approach for a realistic simulation of vehicular traffic in circumscribed scenarios, with extensive analysis placed for roundabout junctions. This solution, based on a novel multi-agent deep reinforcement learning algorithm, permits to build a simulation platform in which virtual drivers are endowed with unique and personal driving capabilities; moreover, drivers feature non-deterministic behaviors, giving an important role to the inter-driver negotiation which arises implicitly through driving actions.

Such a simulator, while needing efforts made of time and resources to be trained, make it possible to develop models suitable for the interaction with real human drivers. Indeed, in this chapter it is evaluated the development of a high-level module for self-driving vehicles able to guide the lower levels of the driving planner on the execution of particular maneuvers involving active interactions with other agents.

This module is based on a neural network similar to the one used for controlling agents inside the simulator, but this time trained with the single-agent version of the Delayed A3C algorithm exposed in Section 3.4, since only the policy of the agent executing the maneuver is learned while that of the vehicles populating the environment is kept fixed.

The application case taken as experiment is again the roundabout insertion ([102]), whose vehicle driving on it are trained following the procedure explained in Section 3.7.

4.1 Intersection handling: the state of the art

The strategies adopted for autonomous vehicles for dealing with those maneuvers which require to take into account the interaction with other drivers are typically based on safety indicators such as *time-to-collision* ([103]) and *headway* ([104]), as in case of the Darpa Urban Challenge winner Boss ([105]) and [106].

The high-level decisions of such systems are typically made by deterministic rules based on such indicators which let the vehicle execute the maneuver only if it is absolutely safe considering the worst case scenario; these solutions lead to excessively cautious behaviors due to a lack of interpretation of the situation and the capability of interacting with other vehicles.

These limitations suggest the use of machine learning approaches in order to infer the intentions of the other drivers. Several works considered this interpretation by modeling the intersection task as a *Partially Observable Markov Decision Process* (POMDP, [107]), in which the intentions of the other vehicles are mapped into the non-observable part of the state (as in [108] and [109]).

In order to model more complex behaviors, deep reinforcement learning approaches are applied inside simulated environments, as in case of [91] for overtakings and [92] for lane-changes, in which the agent observes a discrete set of parameters related to a fixed a priori amount of surrounding vehicles. Similarly, [87] and [90] extended reinforcement learning based solution in order to process visual input, thus making it possible to consider an arbitrary amount of surrounding vehicles and exploiting the use of Convolutional Neural Network for image-based inputs.

4.2 The importance of negotiation

However, as previously explained in Section 3.3, the use of learning based approaches inside a simulated environment whose vehicles behave following predetermined rules is not expected to give robust results when adopted in real world, due to the high variability in the human behaviors and inter-drivers dependency. For this reason, we believe that using a traffic simulator in which virtual drivers are endowed with their own unique personality and are able to negotiate with each other, varying their behaviors depending on the particular situation, is fundamental for training machine learning modules for solving high-level driving tasks.

Indeed, in the developed simulator, vehicles are able to interact with each other while following their assigned conduct, giving rise to a continuously changing and interactive scenario; even further, it is also not guaranteed that agents populating the simulation will not end up crashing or behaving over-aggressively, thus encoding in the simulation also those dangerous corner cases.

As a result, models developed in such an environment will end up featuring implicit negotiation and able to find the best policy to accomplish the task without incurring in accidents or harmful situations; moreover they can be trained so that the determination for accomplishing the maneuver can be tuned, since the effects of such behaviors ca be directly evaluated. For example, agents will be able to try the insertion even if it is not certain that there will be enough room to accomplish the maneuver as humans do in certain cases, in contrast to rule-based methods in which this would not have tried; maybe, the approaching agent will exit the roundabout, or it will slow down, and the maneuver could be completed successfully, or, in the other case, the agent has to block its maneuver prematurely. This ability of negotiating through actions can be paramount in scenarios such as busy traffic intersections, in which traditional approaches will end up in indefinite waiting times; indeed, by modifying the impetus to be used for the maneuver, this capability can be used to solve such situations efficiently and avoid deadlocks.

4.3 Case of study: roundabout scenario

4.3.1 State space

As for the roundabout simulator, an hybrid state space is used in order to take advantage of both visual and numerical type of streams: the former is more suitable to represent spatial features such as vehicles position and road structure, while the latter permits to add additional discrete information and tuning the behavior of the agent.

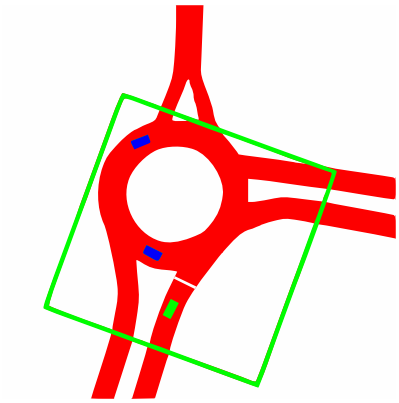
Visual input

In addition to the three semantic layers previously used, consisting on the navigable space in which the navigation is allowed, the obstacles around the agents and its assigned path, a further layer is included, whose aim is to let the agent know the position of the stop line related to the intersection to be handled. This layer contains only the stop line along the path assigned to the agent, as shown in Figure 4.1.

Numerical input

The numerical input features the speed of the agent and the desired cruising speed which it should reach and maintain unless prudence is needed for avoiding dangerous situations; however, instead of receiving the elapsed time ratio and the distance to goal for selecting its aggressive, the agent has a single input dedicated to the impetus of the maneuver (which it will be referred to as the *impetus* input), which is directly related to the reward function observed by the agent, as explained in Section 4.3.3.

Moreover, an additional set of inputs is added in order to let the agent know which was its output at the preceding time instant; this way, by giving a specific cost to each action transition, a more fluent maneuver execution is achieved, avoiding undesired flickers due to the lack of memory of the agent. This memory input is structured as a one-hot vectors with as many entries as the number of actions determining the maneuver (in this case 4): the entry related to the last executed output is set to 1 while the others are filled with 0.



(a) Area perceived from the green vehicle

(b) Navigable space

(c) Obstacles

(d) Path

(e) Stop line

Figure 4.1: **(a)** shows the region perceived from the agent executing the entering maneuver, which is translated into four semantic layers: the navigable space layer **(b)**, the obstacle layer **(c)**, the path layer **(d)** and the stop line layer **(e)**

4.3.2 Action space

The actions at disposal of the agent for entering in the roundabout are high-level commands which are linked to the values of acceleration for the agent dependently on the position of the stop line and its assigned target speed, simulating the the action which would be taken by a planner module. The allowed actions are:

- **stop**: which means that the agent decided to halt the vehicle. The action corresponds to a negative value for the agent acceleration a, which depends on whether the agent has surpassed the stop line or not; if the agent has not yet passed the stop line, a would be equal to the constant value of a_{sl} needed to reach the stop line with zero velocity, namely to halt at the stop line, unless a_{sl} goes over the maximum deceleration a_d, in which case $a = a_d$; if the agent has already surpassed the stop line, a would be equal to a_d. Summarizing, calling v the current velocity of the agent, and d_{sl} the distance that has to be traveled to reach the stop line:

$$\begin{cases} d_{sl} = v_i\, t + \dfrac{1}{2}\, a\, t^2 = \dfrac{v\, t}{2} \\ a_{sl} = \dfrac{v_f - v_i}{t} = \dfrac{-v}{t} \end{cases} \rightarrow a_{sl} = \frac{-v_i^2}{2\, d_{sl}} \tag{4.1}$$

$$a = \begin{cases} \max(a_{sl}, a_d)^1 & \textbf{if } \text{agent before stop line} \\ a_d & \textbf{if } \text{agent after stop line} \end{cases} \tag{4.2}$$

- **go with caution**: which means that the agent decided to proceed with prudence, namely with smaller values of acceleration until a target speed point equal to s_c is reached; the value of acceleration is equal to a_c^+ if the current speed s is below s_c and equal to a negative value a_c^- if it is above, taking in consideration an hysteresis defined by the constant h:

$$a = \begin{cases} a_c^+ & \textbf{if } s < s_c + \frac{h}{2} \\ 0 & \textbf{if } s_c - \frac{h}{2} < s < s_c + \frac{h}{2} \\ a_c^- & \textbf{if } s < s_c - \frac{h}{2} \end{cases} \tag{4.3}$$

[1] remember that a_{sl} and a_d are negative values, so the absolute acceleration value is kept bounded

- **go**: meaning that the insertion can take place freely without the risk of incurring on accidents: the value of acceleration is set to the comfort value a_a unless the current speed is above the assigned target speed s_t:

$$a = \begin{cases} a_a & \textbf{if } s < s_t \\ 0 & \textbf{if } s \geq s_t \end{cases} \tag{4.4}$$

4.3.3 Reward design

The reward function features some novelties from the one adopted for the simulator. In the first place there is the additional term, called r_{change}, representing the cost the agent has to pay when it changes the action from the last time step. Moreover, instead of using the elapsed time ratio and the distance to the goal inputs as a means of determining the behavior of the agent as explained in Section 3.7.5, this time the reward function is directly a function of the *impetus* input which determines the determination on the maneuver execution; this input, at training time, is kept inside the range $[0,1]$. Thus, the reward received at the time step t can be defined as:

$$r_t = r_{terminal} + r_{violation} + r_{speed} + r_{change} \tag{4.5}$$

where the $r_{terminal}$ and $r_{violation}$ are now dependent on the *impetus*.

$r_{terminal}$, indeed, is defined similarly to the function used for the simulator, with the only difference that the penalization for a crash is now dependent on the *impetus* in the following way:

$$r_{terminal} = -\beta - \zeta \cdot (1 - impetus) \qquad \#\ in\ case\ of\ a\ crash \tag{4.6}$$

where β and ζ are constants, the former defining the fixed part of the penalty while the latter weights the impetus dependency. This way, low values for the *impetus* mean a stronger penalization in case of accidents, and vice-versa.

Similarly, $r_{violation}$ is now defined as:

- $-k_{sd} \cdot (1 - impetus)$: when the agent violates the safety distance from the vehicle in front, as shown in Figure 3.5a;

- $-k_y \cdot (1 - impetus)$: when the agent fails to yield, entering the roundabout in front of an already inserted vehicles, as depicted in Figure 3.5b;
- 0: if none of the above occur.

Again, the penalty is more severe for lower values of *impetus*, thus more brave maneuvers are discouraged when the maximum of the prudence is required.

The newly added term r_{change}, has been designed to encourage the agent to choose the *go* output only if it is adequately sure that there will not be any impediment in the maneuver, thus assuring a smooth insertion; consequently, the agent is expected to negotiate with other vehicles through the *stop* and *go with caution* outputs, before allowing the insertion once it is sure about a successful end for the maneuver. Going into details, this penalty is given whenever the agent leaves the *go* condition for a new different state, called N, in this way:

$$r_{change} = \begin{cases} -k_{caution} & \textbf{if } N = caution \\ -k_{stop} & \textbf{if } N = stop \end{cases} \tag{4.7}$$

where $k_{caution} < k_{stop}$ in order to give a stronger penalization for switching between the two opposite outputs.

Finally, the r_{speed} term is used for rewarding the agent as much as it gets closer to its assigned target speed; however, since the agent is now bounded in its maximum speed dependently on the selected output, this term it does not have anymore the function of penalizing the agent when the target speed is surpassed. Its value equals:

$$r_{speed} = k_{sp} \cdot \frac{s}{s_t} \tag{4.8}$$

4.3.4 Implementation details

The neural network hyperparameters used are similar to the ones used for the discrete setting of the simulator and explained in Section 3.7.4; the only difference is in the input structure, which in this case is composed by 4 semantic layers and 7 numerical inputs, namely the current and target speeds, impetus level, and the one-hot vector with 4 entries, one for every action.

β	zeta	k_{sd}	k_y	$k_{caution}$	k_{stop}	k_{sp}
0.2	1.8	0.02	0.05	0.05	0.15	0.0045

Table 4.1: Values of the reward constants used in the maneuver execution experiments

The negative acceleration values are selected inside the range $[a_d, 0]$, with $a_d = -2\frac{m}{s^2}$; the acceleration used in case of output *go* was $a_a = 1\frac{m}{s^2}$, while for *go with caution* was $a_c = 0.5\frac{m}{s^2}$.

The values of the constants used for the reward function are given in Table 4.1

4.4 Experiments and results

For the experiments, unless stated otherwise, has been used a setting similar to the one used for the simulator experiments in the roundabout of Figure 3.3, which are described in Section 3.7.5.

4.4.1 Algorithms comparison

The learning phase, executed with the single-agent version of the Delayed A3C algorithm, is compared with to the ones obtained using classic A3C and its synchronous variant A2C, in which the updates coming from the different agents are computed at common time intervals. To each independent copy of the agent is assigned one of the three entries of the roundabout, creating three slightly different learning environments instead of only 1 as in classical A3C, in order to obtain a more robust policy which was not designed for a specific situation.

This evaluation has been obtained tracking the positive episodes ratio by the exponential moving average of Equation 3.14, called $\overline{r_p}$, for each learning phase executed with the three different algorithms.

The resulting curves, shown in Figure 4.2, tell that the delayed version of A3C (D-A3C) features a faster convergence rate respect to the classical A3C, while the synchronous version, A2C, does not solve successfully the task, since it converges on

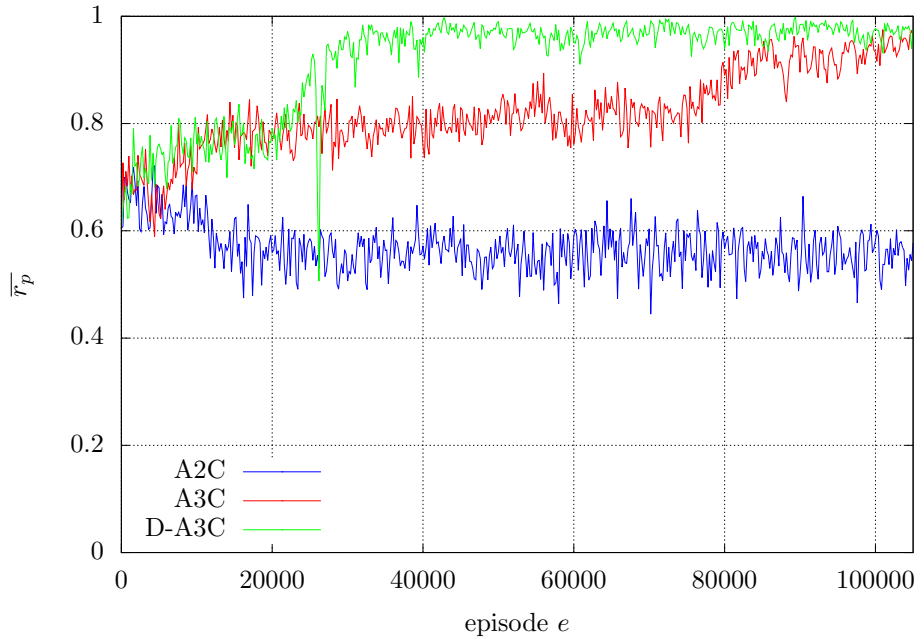


Figure 4.2: Moving average $\overline{r_p}$ of the positive episodes ratio during the learning phase of Delayed A3C (green), classic A3C (red) and A2C (blue)

a suboptimal solution consisting on allowing always the insertion.

4.4.2 Maneuver impetus tuning

The effects of the impetus input, obtained by using a reward function dependent on its value as explained in Section 4.3.3, has been evaluated by recording the positive episodes ratio r_p for different values of impetus i^*, computed averaging the results over several episodes. The graph, represented in Figure 4.3, shows that the augmentation of the impetus brings a decrease of r_p and an increase of the average speed, indicating that more determination is given to the maneuver which reduces the waiting time.

Interestingly, value of the impetus outside the training range $[0,1]$, affects the behavior of the agent in a consistent way. Another curiosity is that, for impetus values

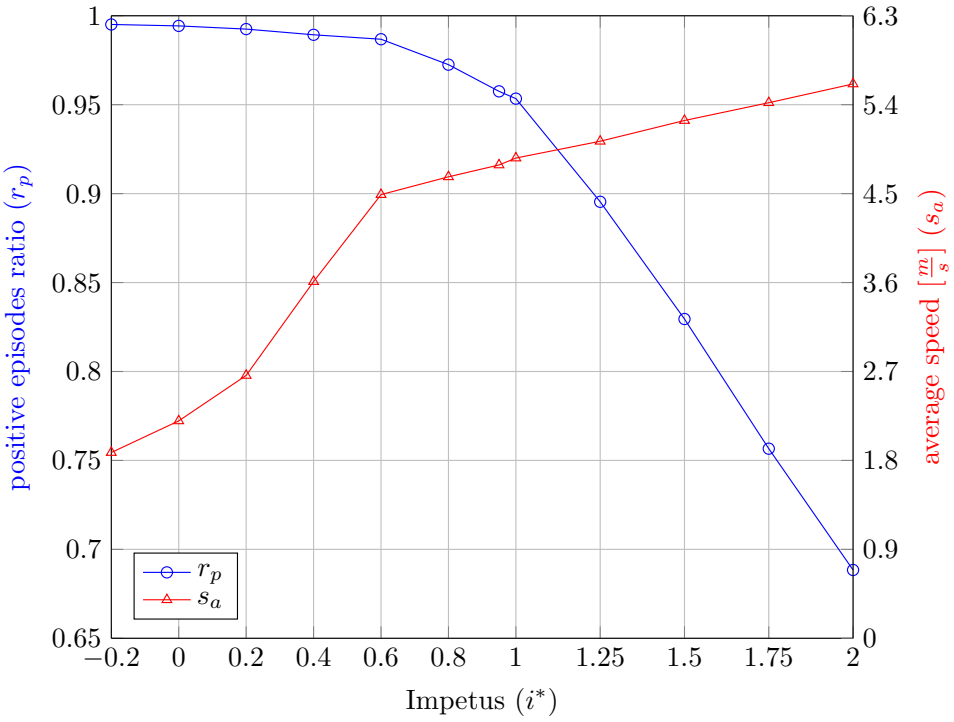


Figure 4.3: Effects of the aggressiveness input on the behavior of the agent. The blue plot shows the positive-ending episodes ratio varying the aggressiveness input; similarly, the red plot shows how this input affect the average speed maintained by the agent

above 0.6, the average speed follows a linear trend, while, for values under 0.6, the trend becomes quasi-quadratic.

4.4.3 Robustness augmentation

One of the goals of this work is the use of the developed high-level modules in real world, where also other road occupants are allowed to navigate: for this reason, the proposed simulator features agents whose behaviors resemble those of human-driven vehicles; moreover, the synthetic representation adopted, as explained in Section 3.2.2,

was focused on the capability of being created with either simulated data and real-time information coming from the vehicle's perception and maps, so that algorithms developed upon the simulator can be also used for the self-driving vehicle.

However, on-board algorithms dedicated to obstacle perception, localization and mapping are clearly not perfect, and the two representations inevitably feature a domain gap; thus, in order to diminish it, some state perturbations have been introduced inside the simulation: one related to the agents trajectories, and the other for simulating perception flaws.

Moreover, some considerations have been taken into account in order to improve the generalization capabilities of the systems, leading to broadening the range of real situations in which these modules may be useful.

Trajectory perturbation

Ideally, the trajectory followed by the autonomous vehicle would match perfectly with the path assigned to the agent; similarly, this path would be positioned exactly at the center of the lane given as input to the agent. If this is the case, by overlapping all the semantic layers the result would be as the one shown in Figure 4.1.

However, imprecisions in the localization or external causes such as the presence of an obstacle on the road, may cause the agent to follow trajectories which does not match exactly with the assigned one; moreover, also the high level maps used to understand the road structure might contain some inaccuracies, bringing misalignment between the agent position, the path to-be-followed and the navigable space.

For this reason, the trained agent policies have been fine-tuned in an environment in which the trajectories of the agents were perturbed, resulting in a two-step learning phase which can be seen as a sort of *curriculum learning*[2] approach. The trajectory assigned to the agent, which passes for the points positioned in the middle of the lane, is transformed into a cubic Bézier curve ([111]) whose control points are randomly selected relying on the original route, so that agents follow constantly new random

[2]Curriculum learning techniques are those in which a learning system, in this case a neural network, is trained in gradually more difficult tasks in order to exploit the knowledge acquired in easier task for solving more complex tasks [110]

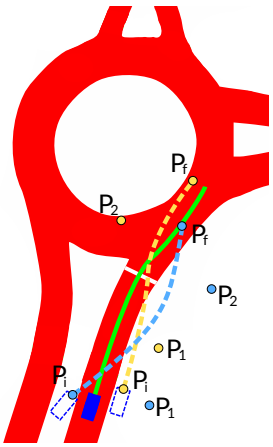


Figure 4.4: The green (solid) trajectory is the original route passing through the lane center points and based on the map structure; the yellow and light blue (dashed) curves are the perturbed trajectories obtained using the Bézier curves. P_i, P_1, P_2 and P_f are the control points of the Bézier curves attained adding noise to the original route

trajectories. In Figure 4.4 are shown the original trajectory and two related perturbed trajectories.

Perception noise simulation

As for localization and mapping, also the perception systems on-board of autonomous vehicles are affected by non-idealities which cause false detections and errors in the obstacle positioning and dimensions.

Because of that, noise in the obstacle perception has been injected in the simulated environment in order to let it get closer to the real-world representation. This perturbation has been obtained by injecting Gaussian noise to the bounding box dimensions and positions of each vehicle detected in the agent surrounding, which changes at every time-step and may feature a non-zero mean for including also bias errors. Moreover, noise is added also in the heading of the vehicles: the type and amount of the injected noise has been designed in order to simulate the same typology of obstacle detections obtained with the self-driving vehicle used for the real-world experiment.

noise-free environment			
	% **reaches**	% **crashes**	% **timeovers**
noise-free	0.989	0.011	0.000
noise-augmented	0.964	0.012	0.024

Table 4.2: Comparison between models trained with and without noise injected during the training phase over 3000 episodes carried out in a ideal environment with no perturbation on trajectories and obstacle perception. The performances are evaluated tracking the ratio of episodes terminated either with a success, with a crash or because the allowed time was expired

Evaluation

For evaluation purposes, two similar models, the first one trained in a noise-free environment while the second in an environment featuring noise inserted both in the trajectories of the agents and on the perceived obstacles, have been compared along 3000 episodes for three different traffic conditions on the aforementioned roundabout with the presence of random noise in the detections and agent trajectories. For determining a score for the agent behavior, the ratio of episodes terminated with success (*% reaches*), that is with a safe maneuver ending, has been computed, together with the ratio of episodes terminated with crashes (*% crashes*) and in which the agent expired the time at disposal (*% time overs*)).

When the two models are compared over an ideal noise-free environment, the noise-free model performs better, due to a perfect matching between training and evaluation conditions, as it is shown from Table 4.2.

However, when noise is injected in the evaluation, Table 4.3 tells that the performances of the noise-free model drops significantly, while the noise-augmented module maintain similar result to the one obtained in the noise-free condition, confirming the robustness against perturbations on the agent trajectory and on the obstacle detections.

	noisy environment		
	% **reaches**	% **crashes**	% **timeovers**
noise-free	0.899	0.043	0.058
noise-augmented	0.967	0.021	0.012

Table 4.3: Comparison between models trained with and without noise injected during the training phase over 3000 episodes carried out in a environment featuring perturbation in the trajectory of agents and on its perceived obstacles. It can be noted that the noise-augmented version is more robust against those perturbations

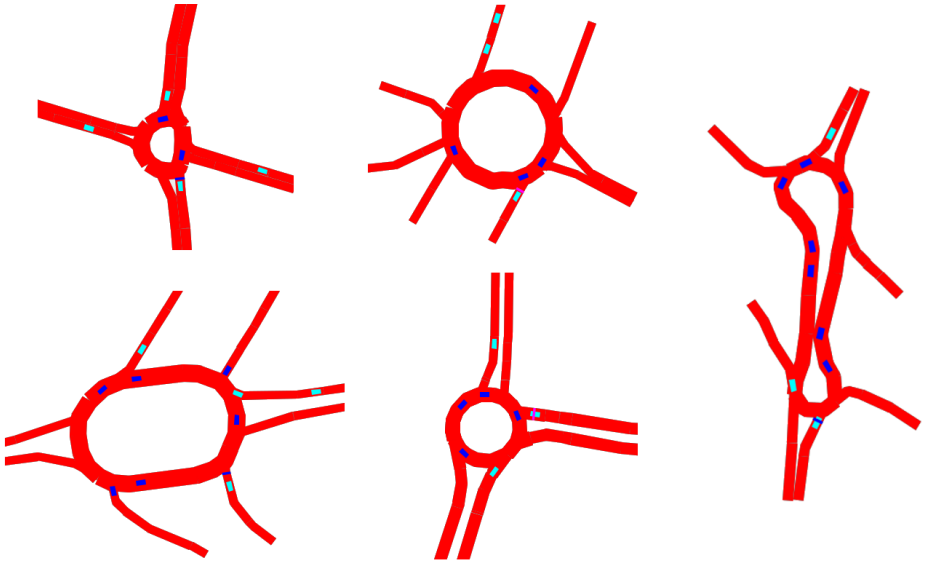
4.4.4 Generalization improvement

The high-level module would be useful in practice if it could be deployed not for just a single specific situation, such as a single roundabout, but for the majority of the situations for which it has been designed. For this reason, generalization tests have been carried out in order to evaluate the generalization capabilities of such a system.

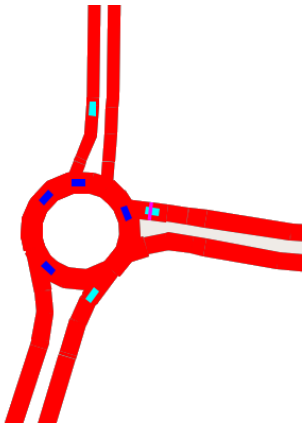
In particular, it has been set up a simulation in which agents are simultaneously trained on different environments, so that the experiences gathered consider more various and diverse situations. In particular, the set of roundabouts which has been used for training are shown in Figure 4.5a; the traffic conditions for each environment were varying during training and evaluation phases.

The performances of the obtained high-level module have been tested along episodes over an unseen roundabout, shown in Figure 4.5c, in order to stress the ability of the system to deal with unknown situations. The results have been compared with an equivalent module trained only in the single roundabout of Figure 4.5b, and with two dummy models useful as a baseline comparison: the first one is a module whose actions are selected randomly, while the second is a module with a constant output consisting on the *go* decision.

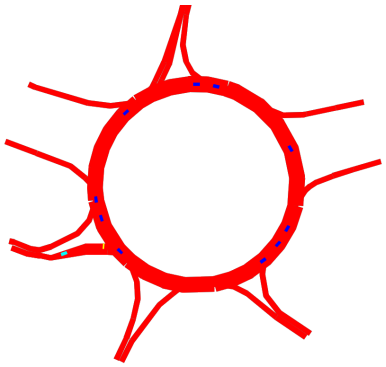
As can be noted in the results given in Table 4.4, the module trained on a single roundabout suffer the new environment, which causes a significant decrement in the performances, although the output is still consistent compared to the baselines.



(a) Training set multi-environment



(b) Training set single-environment



(c) Test roundabout

Figure 4.5: **(a)** shows the structure of the roundabouts used in the training phase for the model trained simultaneously in different scenarios, while **(b)** the one used for the model trained in copies of the same roundabout. **(c)** is the roundabout which was not seen from the agents and used as evaluation case

results on an unseen roundabout			
	% **reaches**	% **crashes**	% **timeovers**
always go	0.676	0.324	0.000
random	0.684	0.270	0.046
single-environment	0.910	0.085	0.003
multi-environment	0.947	0.051	0.002

Table 4.4: Comparison between models trained on a single and multiple scenarios; in the table are also shown the performances of baseline models featuring random or constant (*go*) outputs. The data confirm that agents learning simultaneously on different scenarios features stronger generalization capabilities

Furthermore, the model trained on a variety of different scenarios outperforms the single-environment model, obtaining results which are not far from the performances obtained in a roundabout present in the training set.

4.4.5 Evaluation with real-world data

A preliminary validation of the proposed approach in real-world has been done recording data with a test vehicle. In these first tests, the system has been evaluated only in sequences in which the vehicle was in a static position, so that it has been possible to compare the output of the system with those of humans, and it has been avoided to put people at risk; unfortunately, this means that the negotiation capabilities of the system are marginally verified since the actuation loop has not yet been closed.

Equipment

For acquiring real-world data it has been used a Volkswagen Tiguan equipped with different sensors, shown in Figure 4.6. In particular, obstacles detection has been performed using algorithms based on information coming from the Smartmicro UMRR-11 ([112]) radar positioned in the front bumper of the vehicle; moreover, live images have been recorded using the Ambarella SuperCam ([113]) camera mounted on the rack

Figure 4.6: Volkswagen Tiguan used as test vehicle for acquiring real-world data; in red it is highlighted the radar used for detecting obstacles, in blue the camera for recording images and in green the antennas of the GNSS-INS positioning system

positioned on the vehicle rooftop.

The localization of the vehicle has been achieved by the Applanix POS LV ([114]) high-precision GNSS with INS augmentation (see Section 1.2.5), featuring two antennas for removing constant offset errors in the estimated position and consenting to track the heading of the vehicle.

Data collection

Data have been collected in the three-entries roundabout shown in Figure 3.3a, and consist of both output of the perception and localization systems and video sequences.

The video sequences has been shown to 7 different people at normal speed, and to each of them were asked to select the action they would have performed at every

instant, which could be either to enter the roundabout or to wait at the stop line. Different driver profiles have been emulated by varying the threshold of humans which would have entered at every time-step: a prudent driver which would enter the roundabout only if above the 75% of the users agree with this choice, an average driver with the threshold set to 50% and an aggressive one which would perform the entry if at least one of the users would have done it.

Results

The three driving profiles have been compared with the output of the developed high level module: again, it has been simulated the three different driving style by setting the impetus of the maneuver to -1.0 for the prudent driver, 0.0 for the average driver and 1.0 for the aggressive one.

The result of this empiric comparison in one sequence is shown in Figure 4.7; the average match between the output of the module and the decisions of the human drivers is around 85%, which is comparable with the matching ratio obtained between two different human outputs. For this reason, these tests can be seen as a positive index for the deployment of the module in real world, and future tests will be directed towards closing the loop and using the output of the module for commanding the vehicle.

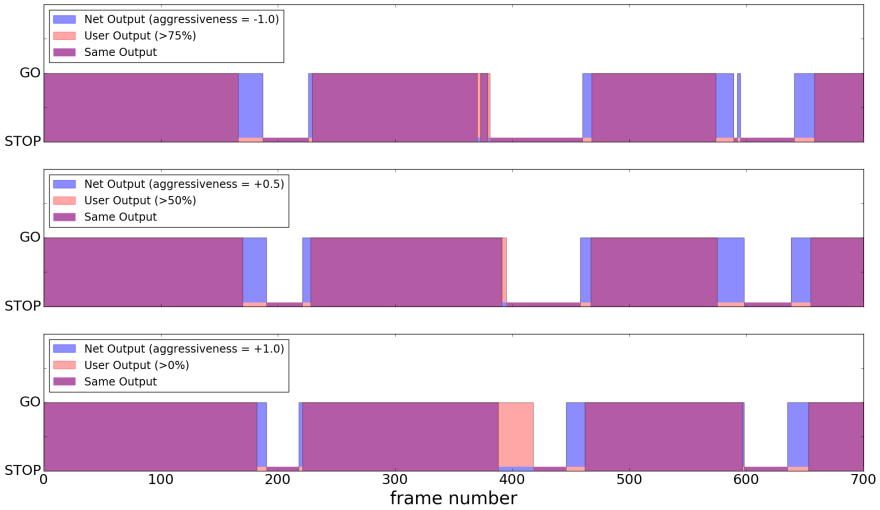


Figure 4.7: Comparison between the actions taken by the human-based driving profiles and the outputs of the maneuver module for performing an entry in the roundabout shown in Figure 3.3a; three different driving profiles, based on the ratio of human drivers which would have perform the entry at every instant ($> 75\%, > 50\%, > 0\%$)), have been compared with the output of the maneuver module with three different impetus level ($-1.0, 0.0, 1.0$).

Chapter 5

Conclusions and Future Developments

The goal of this project was the development of high-level modules for self-driving vehicles useful to coordinate the execution of complex maneuvers; in particular, such modules are useful in situations in which classical planning algorithms fail to accomplish the maneuver in an efficient way. For example, focus has been placed in the roundabout insertion, where, for a smooth execution, it is required a prediction over the intentions of the other road occupants as well as interaction capabilities in order to negotiate the insertion.

For this task, a machine learning approach has been followed, in which the maneuver is learned and optimized throughout a training phase instead of manually designed. In particular, the adopted solution used an algorithm of the family called *Deep Reinforcement Learning*, which goes beyond pure imitation learning and learns the maneuver by exploring actions through a trial-and-error approach.

This solution, however, requires many data to lead to proper behaviors, and it would be needed a fleet of vehicles to acquire enough data.

In order to achieve the goal with the limited resources at disposal, the maneuver has been learned in simulation instead of in real world; however, state-of-the-art driving simulators, even if available with a broad range of graphic realism, are populated by

vehicles whose behaviors are very far from that of humans.

For this reason, it has been developed a simulator based on a novel *multi-agent* Deep Reinforcement Learning algorithm, in which vehicles learn during a training phase how to negotiate with each other. These agents learn to take actions having as input synthetic images of their surroundings, making this approach scalable and adoptable in a broad range of situations without the need of hand-designed input features; furthermore, the proposed solution permits to simulate drivers with infinite many driving styles, in order to include inside the simulation prudent as well as aggressive drivers. This simulator has been developed having in mind the possibility of building the agent synthetic inputs with the data coming from the perception systems on-board self-driving vehicles, so that modules developed upon the simulator can be deployed in real-world with limited domain gap.

Once developed the simulator and trained the agents to mimic human behaviors, this synthetic environment has been used to learn a module able to give high-level maneuver for a safe entry in roundabouts, using a similar approach to the one followed to train agents inside the simulator; the learned module is capable of estimating future trajectories of the other road occupants and implicitly negotiating through driving actions with them, so that an efficient entering is obtained. Moreover, the impetus of the maneuver can be tuned at will, so that even difficult situations, such as entering a very busy roundabout, can be accomplished successfully, when classical approach would have led to undefined waits.

Besides, additional considerations have been taken into account in order to make the system suitable for real-world driving and reducing even further the domain gap between simulation and real driving situations: indeed, noise has been injected both in the obstacle perception and in the agents trajectories, in order to make the system more robust to imperfections which are natural in a real self-driving environment.

Empirical tests have been conducted in order to validate the proposed approach by acquiring real world data and comparing the output of the system with those of human drivers; in order to make this comparison possible, the recorded sequences have been taken with the vehicle in a static position, that is without actuating the output decisions.

Further tests will be directed toward closing the actuation loop and executing at real-time the actions predicted by the module so that also the negotiation capabilities can be measured.

Finally, the same approach will be followed to develop high-level maneuver module for other critical maneuvers, such as intersection handling and overtaking.

Bibliography

[1] Ctesibius' life from *Dreamers of History of Computers* [online]. URL: https://history-computer.com/Dreamers/Ctesibius.html.

[2] George Constable and Bob Somerville. *A Century of Innovation: Twenty Engineering Achievements that Transformed our Lives*. Joseph Henry Press, Washington, DC, 2003.

[3] Allan Nevins. *Ford: The Times, The Man, The Company*. Scribners, New York, 1954.

[4] Dean Carnegie. Houdini Going Postal [online]. 11-02-2012. URL: http://www.themagicdetective.com/2012/02/houdini-going-postal.html.

[5] Carl Engelking. The 'Driverless' Car Era Began More Than 90 Years Ago [online]. 13-12-2017. URL: http://blogs.discovermagazine.com/d-brief/2017/12/13/driverless-car-houdina-houdini/.

[6] Evan Ackerman. Self-Driving Cars Were Just Around the Corner—in 1960 [online]. 31-08-2016. URL: https://spectrum.ieee.org/tech-history/heroic-failures/selfdriving-cars-were-just-around-the-cornerin-1960.

[7] Sunday Times Driving. 1960s Citroën DS Driverless Car Test [online]. URL: https://www.youtube.com/watch?v=MwdjM2Yx3gU.

[8] Rodney Albert Schmidt Jr. *A study of the real-time control of a computer-driven vehicle*. PhD thesis, Stanford University, 1971.

[9] Reuben Hoggett. 1960 – stanford cart [online]. URL: `http://cyberneticzoo.com/cyberneticanimals/1960-stanford-cart-american/`.

[10] Hans Peter Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. PhD thesis, Stanford University, 1980.

[11] Ernst Dieter Dickmanns and Volker Graefe. Applications of dynamic monocular machine vision. *Journal of Machine Vision and Application*, 1(4):241–261, 1988.

[12] Ernst Dieter Dickmanns, Reinhold Behringer, Dirk Dickmanns, Thomas Hildebrandt, Markus Maurer, Frank Thomanek, and Joachim Schiehlen. The seeing passenger car 'VaMoRs-P'. In *Proceedings of the 1994 IEEE Intelligent Vehicles Symposium (IV)*, pages 68–73, 1994.

[13] Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1 (NIPS)*, pages 305–313. 1989.

[14] Dean Pomerleau and Todd Jochem. No Hands Across America [online]. 1995. URL: `http://www.cs.cmu.edu/afs/cs/usr/tjochem/www/nhaa/nhaa_home_page.html`.

[15] Alberto Broggi, Massimo Bertozzi, Alessandra Fascioli, and Gianni Conte. *Automatic Vehicle Guidance: The Experience of the ARGO Autonomous Vehicle*. World Scientific Publishing Co. Pte. Ltd., 1999.

[16] Defense Advanced Research Projects Agency. The DARPA Grand Challenge: Ten Years Later [online]. 13-03-2014. URL: `https://www.darpa.mil/news-events/2014-03-13`.

[17] Massimo Bertozzi, Luca Bombini, Alberto Broggi, Michele Buzzoni, Elena Cardarelli, Stefano Cattani, Pietro Cerri, Alessandro Coat, Stefano Debattisti, Rean Fedriga, Mirko Felisa, Luca Gatti, Alessandro Giacomazzo, Paolo Grisleri, Maria Laghi, Luca Mazzei, Paolo Medici, Matteo Panciroli, Pier Paolo Porta, Paolo Zani, and Pietro Versari. VIAC: an Out of Ordinary Experiment. In *Proceedings of the 2011 IEEE Intelligent Vehicle Symposium (IV)*, pages 175–180, 2011.

[18] Alberto Broggi, Pietro Cerri, Stefano Debattisti, Maria Chiara Laghi, Paolo Medici, Daniele Molinari, Matteo Panciroli, and Antonio Prioletti. PROUD—Public Road Urban Driverless-Car Test. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):3508–3519, December 2015.

[19] Julius Ziegler, Philipp Bender, Markus Schreiber, Henning Lategahn, Tobias Strauß, Christoph Stiller, Thao Dang, Uwe Franke, Nils Appenrodt, Christoph Gustav Keller, Eberhard Kaus, Ralf G. Herrtwich, Clemens Rabe, David Pfeiffer, Frank Lindner, Fridtjof Stein, Friedrich Erbs, Markus Enzweiler, Carsten Knöppel, Jochen Hipp, Martin Haueis, Maximilian Trepte, Carsten Brenk, Andreas Tamke, Mohammad Ghanaat, Markus Braun, Armin Joos, Hans Fritz, Horst Mock, Martin Hein, and Eberhard Zeeb. Making Bertha Drive—An Autonomous Journey on a Historic Route. *IEEE Intelligent Transportation Systems Magazine*, 6:8–20, 2014.

[20] Nidhi Kalra and Susan M. Paddock. Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability? Technical report, Rand Corporation, 2016. URL: `https://www.rand.org/pubs/research_reports/RR1478.html`.

[21] Jeremy Rifkin. *The End of Work: The Decline of the Global Labor Force and the Dawn of the Post-market Era*. G.P. Putnam's Sons, 1995.

[22] Encyclopaedia Britannica, editor. Luddite [online]. URL: `https://www.britannica.com/event/Luddite`.

[23] Simon Romero. Wielding rocks and knives, arizonans attack self-driving cars [online]. 31-12-2018. URL: https://www.nytimes.com/2018/12/31/us/waymo-self-driving-cars-arizona-attacks.html.

[24] World Development Report 2019: The Changing Nature of Work. Technical report, World Bank Group, 2019. URL: https://www.worldbank.org/en/publication/wdr2019.

[25] Global Status Report on Road Safety 2018. Technical report, World Health Organization, 2018. URL: https://www.who.int/violence_injury_prevention/road_safety_status/2018/en/.

[26] Lee Vinsel. The Man Who Invented Intelligent Traffic Control a Century Too Early. *IEEE Spectrum*, 2016. URL: https://spectrum.ieee.org/tech-history/dawn-of-electronics/the-man-who-invented-intelligent-traffic-control-a-century-too-early.

[27] List of self-driving car fatalities [online]. URL: https://en.wikipedia.org/wiki/List_of_self-driving_car_fatalities.

[28] Self-driving cars: the Impact on People with Disabilities. Technical report, Ruderman Family Foundation, 2017. URL: https://rudermanfoundation.org/wp-content/uploads/2017/08/Self-Driving-Cars-The-Impact-on-People-with-Disabilities_FINAL.pdf.

[29] Our Lives Inside Our Cars — European Survey. Technical report, Citroën and CSA Research, 2016. URL: http://media.citroen.be/file/64/4/.pdf?_ga=1.246496516.1879363606.1433151684.

[30] Justin Worland. Self-Driving Cars Could Help Save the Environment—Or Ruin It. It Depends on Us [online]. 8-11-2016. URL: https://time.com/4476614/self-driving-cars-environment/.

[31] Paul Barter. "Cars are parked 95% of the time". Let's check! [online]. 22-02-2013. URL: https://www.reinventingparking.org/2013/02/cars-are-parked-95-of-time-lets-check.html.

[32] Wikipedia. Tesla autopilot [online]. URL: https://en.wikipedia.org/wiki/Tesla_Autopilot.

[33] Reuters. Waymo unveils self-driving taxi service in arizona for paying customers [online]. URL: https://www.reuters.com/article/us-waymo-selfdriving-focus/waymo-unveils-self-driving-taxi-service-in-arizona-for-paying-customers-idUSKBN1O41M2.

[34] Eike Rehder, Horst Kloeden, and Cristoph Stiller. Head detection and orientation estimation for pedestrian safety. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 2292–2297, 2014.

[35] Henrik Kretzschmar and Jiajun Zhu. Cyclist hand signal detection by an autonomous vehicle, 2014. US Patent 9014905B1. URL: https://patents.google.com/patent/US9014905B1.

[36] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *IEEE International Conference on Computer Vision (ICCV)*, pages 261–268, 2009.

[37] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars. 2016. URL: http://arxiv.org/abs/1604.07316.

[38] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-To-End Interpretable Neural Motion Planner. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[39] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018. URL: `http://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf`.

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

[41] David Silver. Markov decision processes [online]. URL: `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf`.

[42] Rajeev Agrawal. Sample Mean Based Index Policies with O(log n) Regret for the Multi-Armed Bandit Problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995.

[43] Herman Kahn and Andrew W. Marshall. Methods of Reducing Sample Size in Monte Carlo Computations. *Journal of the Operations Research Society of America*, 1(5):263–278, 1953.

[44] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning (ICML)*, pages 387–395, 2014. URL: `http://dl.acm.org/citation.cfm?id=3044805.3044850`.

[45] Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9–44, 1988. URL: `https://doi.org/10.1023/A:1022633531479`.

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

[47] Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.*, 8(3-4):229–256, May 1992.

[48] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992. URL: https://doi.org/10.1007/BF00992696.

[49] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. URL: https://doi.org/10.1038/nature14539.

[50] David E. Rumelhart, Geoffrey E. Hinton., and Ronald J. Williams. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. volume 1, chapter Learning Internal Representations by Error Propagation, pages 318–362. 1986.

[51] Richard S. Sutton. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems 8 (NIPS)*, pages 1038–1044. MIT Press, 1996.

[52] Larry D. Pyeatt and Adele E. Howe. Decision Tree Function Approximation in Reinforcement Learning. Technical report, In Proceedings of the Third International Symposium on Adaptive Systems: Evolutionary Computation and Probabilistic Graphical Models, 1998.

[53] Justin A. Boyan and Andrew W. Moore. Generalization in Reinforcement Learning: Safely Approximating the Value Function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7 (NIPS)*, pages 369–376, 1995. URL: http://papers.nips.cc/paper/1018-generalization-in-reinforcement-learning-safely-approximating-the-value-function.pdf.

[54] R. M. Kretchmar and C. W. Anderson. Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In *Proceedings of International Conference on Neural Networks (ICNN)*, volume 2, pages 834–837, 1997.

[55] Charles Williams Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems*. PhD thesis, University of Massachusetts, 1986.

[56] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. URL: `http://dx.doi.org/10.1038/nature14236`.

[57] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning (ICML)*, pages 1928–1937, 2016.

[58] Peter Stone and Manuela Veloso. Multiagent Systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[59] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-agent Reinforcement Learning. In *Advances in Neural Information Processing Systems 29 (NIPS)*, pages 2145–2153, 2016. URL: `http://dl.acm.org/citation.cfm?id=3157096.3157336`.

[60] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative Multi-agent Control Using Deep Reinforcement Learning. In *AAMAS 2017: Autonomous Agents and Multiagent Systems*, pages 66–83, 2017.

[61] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning, (ICML)*, pages 1889–1897, 2015. URL: http://proceedings.mlr.press/v37/schulman15.html.

[62] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceeding of the 4th International Conference on Learning Representations (ICLR)*, 2016. URL: http://arxiv.org/abs/1509.02971.

[63] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, 1992. URL: https://doi.org/10.1007/BF00992699.

[64] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1146–1155, 2017. URL: http://proceedings.mlr.press/v70/foerster17b.html.

[65] Ryan Lowe, YI WU, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems 30 (NIPS)*, pages 6379–6390, 2017. URL: http://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments.pdf.

[66] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. Is multiagent deep reinforcement learning the answer or the question? A brief survey. 2018. URL: http://arxiv.org/abs/1810.05587.

[67] Alon Halevy, Peter Norvig, and Fernando Pereira. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 24:8 – 12, 2009.

[68] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using Inaccurate Models in Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 1–8, 2006. URL: http://doi.acm.org/10.1145/1143844.1143845.

[69] M. Cutler, T. J. Walsh, and J. P. How. Reinforcement learning with multi-fidelity simulators. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3888–3895, 2014. URL: http://acl.mit.edu/papers/Cutler14_ICRA.pdf.

[70] Andrei A. Rusu, Matej Vecerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-Real Robot Learning from Pixels with Progressive Nets. In *1st Annual Conference on Robot Learning (CORL)*, pages 262–270, 2017. URL: http://proceedings.mlr.press/v78/rusu17a.html.

[71] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive Neural Networks. 2016. URL: http://arxiv.org/abs/1606.04671.

[72] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. Adapting Deep Visuomotor Representations with Weak Pairwise Constraints. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2016. URL: https://arxiv.org/pdf/1511.07111.pdf.

[73] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real Single-Image Flight Without a Single Real Image. In *Robotics: Science and Systems XIII*, 2017. URL: http://www.roboticsproceedings.org/rss13/p34.html.

[74] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, 2011. URL: http://proceedings.mlr.press/v15/ross11a.html.

[75] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. 2018. URL: http://arxiv.org/abs/1812.03079.

[76] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to Drive in a Day. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 8248–8254, 2019. URL: https://doi.org/10.1109/ICRA.2019.8793742.

[77] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-Supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, 2018. URL: https://doi.org/10.1109/ICRA.2018.8460655.

[78] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. 2016. URL: http://arxiv.org/abs/1610.03295.

[79] Mark Martinez, Chawin Sitawarin, Kevin Finch, Lennart Meincke, Alex Yablonski, and Alain L. Kornhauser. Beyond Grand Theft Auto V for Training, Testing and Enhancing Deep Learning in Self Driving Cars. 2017. URL: http://arxiv.org/abs/1712.01397.

[80] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the Matrix: Can virtual

worlds replace human-generated annotations for real world tasks? In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 746–753, 2017. URL: https://doi.org/10.1109/ICRA.2017.7989092.

[81] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio López, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *1st Annual Conference on Robot Learning (CORL)*, pages 1–16, 2017. URL: http://proceedings.mlr.press/v78/dosovitskiy17a.html.

[82] Epic Games. Unreal Engine 4. URL: https://www.unrealengine.com/en-US/features.

[83] Matthias Mueller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. Driving Policy Transfer via Modularity and Abstraction. In *2nd Annual Conference on Robot Learning (CORL)*, pages 1–15, 2018. URL: http://proceedings.mlr.press/v87/mueller18a.html.

[84] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to Real Reinforcement Learning for Autonomous Driving. In *British Machine Vision Conference (BMVC)*, 2017. URL: https://www.dropbox.com/s/qqb0qvt1q18dp9e/0221.pdf?dl=1.

[85] Bernhard Wymann, Christos Dimitrakakisy, Andrew Sumnery, and Christophe Guionneauz. Torcs: The open racing car simulator, 2015. URL: https://sourceforge.net/projects/torcs/.

[86] Daniel Krajzewicz, Georg Hertkorn, Christian Feld, and Peter Wagner. SUMO (Simulation of Urban MObility); An open-source traffic simulation. In *4th Middle East Symposium on Simulation and Modelling (MESM2002)*, pages 183–187, 2002.

[87] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating Occluded Intersections with Autonomous Vehicles Using Deep Reinforcement Learning. In *IEEE International Conference on Robotics*

and Automation (ICRA), pages 2034–2039, 2018. URL: https://doi.org/10.1109/ICRA.2018.8461233.

[88] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested Traffic States in Empirical Observations and Microscopic Simulations. *Physical Review E*, 62:1805–1824, 2000.

[89] Chris Paxton, Vasumathi Raman, Gregory D. Hager, and Marin Kobilarov. Combining Neural Networks and Tree Search for Task and Motion Planning in Challenging Environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6059–6066, 2017.

[90] Jingchu Liu, Pengfei Hou, Lisen Mu, Yinan Yu, and Chang Huang. Elements of Effective Deep Reinforcement Learning towards Tactical Driving Decision Making. 2018. arXiv:http://arxiv.org/abs/1802.00332.

[91] Xin Li, Xin Xu, and Lei Zuo. Reinforcement Learning Based Overtaking Decision-Making for Highway Autonomous Driving. In *International Conference on Intelligent Control and Information Processing (ICICIP)*, pages 336–342, 2015.

[92] Carl-Johan Hoel, Krister Wolff, and Leo Laine. Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning. In *21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2148–2155, 2018. URL: https://doi.org/10.1109/ITSC.2018.8569568.

[93] Arne Kesting, Martin Treiber, and Dirk Helbing. Agents for Traffic Simulation. In *Multi-Agent Systems - Simulation and Applications*, pages 325–356. 2009. URL: https://doi.org/10.1201/9781420070248.ch11.

[94] Wei Liu, Seong-Woo Kim, Scott Pendleton, and Marcelo H. Ang. Situation-aware Decision Making for Autonomous Driving on Urban Road using Online POMDP. In *Proceegings of the 2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1126–1133, 2015.

[95] Giulio Bacchiani, Daniele Molinari, and Marco Patander. Microscopic Traffic Simulation by Cooperative Multi-agent Deep Reinforcement Learning. *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1547–1555, 2019. URL: `http://www.ifaamas.org/Proceedings/aamas2019/pdfs/p1547.pdf`.

[96] Alex Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. Frame Skip Is a Powerful Parameter for Learning to Play Atari. In *AAAI-15 Workshop on Learning for General Competency in Video Games*, 2015.

[97] Alexey Dosovitskiy and Vladlen Koltun. Learning to Act by Predicting the Future. In *International Conference on Learning Representations (ICLR)*, 2017.

[98] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, and Jeff Schneider. Motion Prediction of Traffic Actors for Autonomous Driving using Deep Convolutional Networks. 2018. URL: `http://arxiv.org/abs/1808.05819`.

[99] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent Cooperation and Competition with Deep Reinforcement Learning. 2015. `arXiv:1511.08779`.

[100] Keith Packard and Carl Worth. Cairo graphics library. `https://www.cairographics.org/`, 2003–2019.

[101] Rajesh Rajamani. *Vehicle Dynamics and Control*. Springer, 2006.

[102] Alessandro Capasso, Giulio Bacchiani, and Daniele Molinari. Intelligent Roundabout Insertion using Deep Reinforcement Learning. *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART)*, 2020.

[103] Richard van der Horst and Jeroen Hogema. Time-To-Collision and Collision Avoidance Systems. In *6th International Co-operation on Theories and Concepts in Traffic safety Conference (ICTCT) workshop*, 1994.

[104] Cem Hatipoglu, Umit Ozguner, and Martin Sommerville. Longitudinal headway control of autonomous vehicles. *Proceeding of the IEEE International Conference on Control Applications*, pages 721–726, 1996.

[105] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William "Red" Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *Journal of Field Robotics - Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(8):425–466, 2008. URL: `https://onlinelibrary.wiley.com/doi/epdf/10.1002/rob.20255`.

[106] Akansel Cosgun et al. Towards Full Automated Drive in Urban Environments: A Demonstration in GoMentum Station, California. In *Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1811–1818, 2017.

[107] Matthijs T. J. Spaan. Reinforcement Learning. Adaptation, Learning, and Optimization. volume 12, chapter Partially Observable Markov Decision Processes, pages 387–414. 2012.

[108] Weilong Song, Guangming Xiong, and Huiyan Chen. Intention-Aware Autonomous Driving Decision-Making in an Uncontrolled Intersection. *Mathematical Problems in Engineering*, 2016:1–15, 2016. URL: `http://dx.doi.org/10.1155/2016/1025349`.

[109] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. Intention-Aware Motion Planning. In *Proceed-*

ings of the Tenth Workshop on the Algorithmic Foundations of Robotics, pages 475–491, 2013.

[110] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 41–48, 2009. URL: http://doi.acm.org/10.1145/1553374.1553380.

[111] Wikipedia. Bézier curve [online]. 14-09-2019. URL: https://en.wikipedia.org/wiki/B%C3%A9zier_curve.

[112] Smartmicro UMRR Datasheet [online]. URL: http://www.smartmicro.de/fileadmin/user_upload/Documents/Automotive/UMRR_Automotive_Type_132_Data_Sheet.pdf.

[113] Ambarella Introduces CV1 4K Stereovision Processor with CVflow Computer Vision Architecture [online]. 08-01-2018. URL: https://www.ambarella.com/news/ambarella-introduces-cv1-4k-stereovision-processor-with-cvflow-computer-vision-architecture/.

[114] Applanix POS LV Datasheet [online]. URL: https://www.applanix.com/downloads/products/specs/POS_LV_Datasheet.pdf.

Ringraziamenti

Un grazie a tutti i compagni del VisLab con cui ho condiviso questo percorso: ognuno, a modo suo, magari involontariamente, ha saputo trasmettermi qualcosa e contribuito alla mia esperienza; spero che questo contributo possa essere restituito nel breve futuro.

Un altro tipo di ringraziamento va peró a quelle persone che hanno reso possibile il raggiungimento di questo 'traguardo' e che hanno arricchito questi tre anni di incontri, condivisioni, amicizie, e reso ogni giorno diverso dal successivo.

Penso all'esperienza di Tandem, che é riuscita a stanarmi dalla casa mobile con l'opportunitá di coabitare una situazione tanto utopica quanto normale, e al suo naturale proseguimento nelle case Koudoumani e Djecafo.

Penso alle persone incontrate nel porto di Via Duca, e al continuo scambio di emozioni, spesso folli.

Penso agli amici di Arte Migrante, che con una semplicitá e una naturalezza che hanno dell'incredibile hanno reso questa cittá migliore; sicuramente l'impronta lasciata da questo gruppo é piú profonda e inaspettata di quella che sperassi lasciare al mio arrivo.

Penso a tutte le altre persone con cui ho condiviso speranze, sogni, frustrazioni: sono tutte queste persone che mi hanno dato entusiasmo ed hanno donato senso alla mia migrazione; una menzione speciale per la neo-dottoressa Giulia, che da incontro inaspettato é diventata fedele compagna di avventure.

Un ringraziamento speciale va infine alla mia famiglia, che non ha mai smesso di esserci da 28 anni a questa parte :)