Simulation of the Control Center for the Design of an Eco-Friendly Mobility Service

Thesis by

Ambrogio Delli Colli

Master of Science

in

Mechatronic Engineering

Politecnico di Torino Supervisor : Prof. M. Violante

E.N.S.M.M. Supervisor : Prof. J.M. Nicod

IRT SystemX Supervisor : A. Koudri

POLITECNICO DI TORINO

Turin, Italy

2018/2019

ABSTRACT

Simulation tools for autonomous vehicles are in high demand today in the research field, indeed simulations provides experimentation less expensive than a real test of the system. With a simulation we are able to gain in terms of time for the realization of the system, anticipate problems that could effect it, and measure the impact of the system's actions on his environment.

During the internship we integrated two different types of simulators : a robotics simulator and a microscopic traffic simulator. Therefore, with this simulation system, the autonomous vehicle can be inserted in a real traffic situation to study the interaction of the sensors and the actuators of the system with the environment and the others vehicles. Starting with the development of behavioral models of the autonomous vehicle and with the simulation of these, we achieved the verification and the improvement of the reference architecture ARC-IT (Architecture Reference for Cooperative and Intelligent Transportation), used in the EVA project.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

CONTEXT OF THE PROJECT

1.1 IRT SystemX

Launched in 2012, the IRT SystemX is a technological research institute specialized in the digital engineering of complex systems for the future. The IRT SystemX has three principal aims :

- **To meet the challenges of the companies** with four operational fields : Agile Industry, Autonomous Transport, Smart Territories, Internet of Trust.
- **To find scientific answers** evolving the research in four scientific axes : data science and interaction, optimization and scientific computing, system engineering and software, infrastructure and networks.
- **To share this knowledge** using technological platforms developed for the research project.

1.2 Paris-Saclay Autonomous Lab Project

Groupe Renault, the Transdev Group, IRT SystemX, VEDECOM, and the University of Paris-Saclay committed their work to the Paris-Saclay Autonomous Lab project. It was launched under the acronym EVAPS[1] with the support of the French government's Investments for the Future program (PIA) entrusted to ADEME[2], the Établissement Public d'Aménagement Paris-Saclay[3], the Paris-Saclay urban community, the Essonne Department, and Ile-deFrance Mobilités.
The main purpose of the Paris-saclay Autonomous Lab is to devise and test a mobility service based on the utilization of : autonomous electric vehicles, Renault ZOE Cab prototype cars, autonomous Transdev i-Cristal shuttle, and a connected infrastructure to achieve a smarter and more efficient transportation system.
Inaugurated on May 15 2019, the Paris-Saclay Autonomous Lab project offers:

[1]1 Eco-mobilité par Véhicules Autonomes sur le territoire de Paris-Saclay – autonomous vehicle eco-mobility in the ParisSaclay area.

[2]French environment and energy management agency

[3]Paris-Saclay urban development agency

- A night transportation service using the Transdev autonomous shuttles.
- A daytime on-demand car service using Renault ZOE Cab prototype vehicles on the Paris-Saclay urban Campus.

1.3 Role of IRT SystemX in the Autonomous Lab Project

IRT SystemX supplies its expertise in providing specifications and recommendations for this system of systems. The ambitions of the EVA project are:

- To master the architecture and the mobility system life cycle using MBSE (model-based systems engineering).
- To consolidate the safety of the mobility system by ensuring the compliance with the Level 4 safety requirements of the SAE standard.
- To protect the system against malicious acts by providing a PKI (Public Key Infrastructure) solution.

Chapter 2

INTRODUCTION

2.1 Mobility Challenges

In the last three centuries, urbanization has experienced a significant growing trend. With regards to the French territory, INSEE estimates that today more than 75% of the population live in cities. In addition, large urban communities concentrate not only the population but also the bulk of economic activities. The first truly visible effect of this important urbanization is the complexification of transport systems which is essential to satisfy citizens's mobility needs. Indeed, the efficiency of transportation systems for both people and goods, is an important parameter that measures the socio-economic health of a territory, whatever its scale is. With this regard, the increasing densification of cities creates a number of problems such as, in a particular way, road congestion. Last year, the average time wasted by Parisian drivers in traffic jams was 65 hours [1]. Moreover, this time can likely increase due to the fact that the construction of the roads occurred when the car circulating were significantly less in number. As a consequence, the current load is dangerously out of size because unpredictable until recently.

Another significant effect of urbanization is a relevant rise in real estate prices, especially in inner cities. This has led most of the urban dwellers to seek more affordable housing, usually on outskirts of the city. As a consequence, these people tend to get away from their working area. At the same time, the price of real estate has also pushed companies to settle outside cities and away from their employees. Therefore, now many people drive to get to their working place which could be on the opposite side of the city by passing through crowded roads.

Due to these phenomena, recently, public authorities have started to invest in improvement of public transport systems in order to decongest road traffic. As for roads, public transportation infrastructures rely on a certain history and have not always been dimensioned to support such a demographic evolution. In addition, a key aspect of workers traffic is that it is saturated during peak hours. As a result, many people prefer to take their vehicle without taking care of environment and safety. Moreover, since evolution and maintenance of transport services is increasingly costly for local authorities, some maintenance operations are postponed sine die for lack of means and unfortunate or even catastrophic consequences can be easily met. Road congestion, as well as its impact on the

environment and safety, is indeed one of the major problems of urbanization.
Regarding this last point, it can be noticed that the more traffic increases, the more people depend on cars. The reason of this phenomenon can be found on security aspects because, in traffic jam situation, people generally feel safer inside a vehicle. To address this problem of increasing traffic, several solutions have been experimented. Examples include private vehicle sharing (a service offered by BlaBlaCar or Uber), reserved lanes which optimize public transport and reduce vehicles, and finally the development of a powerful multi-modal transport service. This last option is the one that the partners of EVA project have chosen to consider.

2.2 The Use of Autonomous Vehicles

During the last decade, autonomous vehicles have become firmly established in our streets, especially in the United States where 26 states have adopted legislation for autonomous vehicles [2]. On the contrary, the trend experienced in Europe appears to be different and slower. This technology can radically change the way in which transportation systems work. Although the impacts of autonomous vehicles on road safety and congestion have been anticipated in some detail, the potential behavioral has changed and the resulting environmental impacts have received little attention [3]. A solution like robotaxis brings benefits to the city of the future by decreasing car parks and owned cars. In addition, if autonomous vehicles are electric, a reduction in CO2 emissions is achieved as well. Furthermore, every unoccupied vehicle is able to get to more advantageous areas to make the next trip quicker as well.
In addition, from a user point of view, the use of mobile applications to access every kind of service is becoming widespread. We live in a service society and transportation is definitely not an exception to the rule. The acceptance of new modes of transport is subject to a set of measurable and effective criteria: safety, cost, availability, and efficiency.
Lastly, the economic aspect is obviously an important factor. If the price of this service as a mean to get to work or to go shopping exceeds that of owning and maintaining a personal vehicle, convincing people would be difficult even with security arguments. In this context, vehicle sharing is a serious argument for lowering costs. In fact, a special attention needs to be paid on the access to the service in order to be quick, efficient, and available at any time (24/7).

2.3 Workplan

The objective of the EVA (Ecomobility via Autonomous Vehicles) project is to study the feasibility of an eco-mobility solution based on the use of autonomous vehicles and a dedicated infrastructure.

The following chapter presents the study carried out by the EVA Project SoS (System of Systems), the tools used for the simulations, and a brief theoretical presentation of the deep reinforcement learning algorithm that has been implemented. Moreover, chapter 4 is dedicated to the description of the work process. The first part shows the development of autonomous vehicle models and simulation, secondly the validation of requirements and functions using simulations is explained, and finally, in the last part, the use of the deep reinforcement learning technique for autonomous vehicle navigation is detailed.

Chapter 3

STATE OF ART

3.1 The EVA Project

The purpose of the EVA project is to exploit advances in digital technologies by promoting rapid decision-making process in order to achieve a significant reduction in costs and delays, and avoid a poor quality of mobility systems. With this regard, the quality aspects, which are taken care by a collaborative engineering, have the capacity to reconcile different points of view of the stakeholders throughout the life cycle of the system in particular during its conception, and the ability to be flexible at changes and different configurations. This methodology makes partners co-design the SoS (System of Systems) with the imperatives: to respect the development strategies, to protect the intellectual property and the know-how of each, to authenticate and secure the exchange of information between organizations in design activities, and to provide an approach based on changes, inconsistency detection, and reconciliation. This methodology is developed by a combination of 3 different approaches:

- A top-down approach starting from the need and establishing high level requirements on the operational, functional, or organic parts of the SoS.
- A bottom-up approach characterizing the existing modelss at a level of abstraction which is needed to enable efficient collaboration.
- A "meet-in-the-middle" approach focusing on a gap analysis that compares what is desired and what exists.

Several models analyzing these aspects of the system of systems are associated at each of these levels.
From a technical perspective, the design of a complex system driven by scenarios can be helpful if it allows the consideration of any relevant combination of features (events, behaviors, conditions, and so on) to really understand how the system should behave.

3.2 Generation of Analysis Scenarios

As shown in Figure 3.1 the definition of scenarios is performed by experts. To improve the quality and the accuracy of the choice of scenarios, the proposed approach is based on the utilization of a knowledge extraction, using natural language processing techniques, from the literature [4] in order to capitalize it into an ontology.

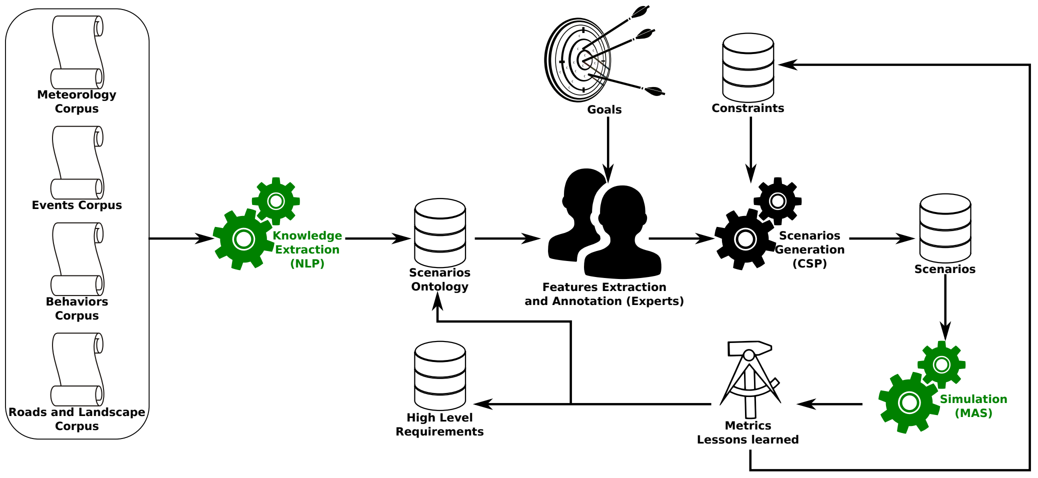


Figure 3.1: Overview of the scenarios management.

Therefore, firstly an ontology was developed to classify the various climatic conditions, the different types of roads and their states, the different types of behaviors of the users both pedestrians or people in vehicles, and the various types of vehicle. Secondly, the subset of the ontology which was considered of interest for experimentation, has been extracted. Moreover, the Paris-Saclay plateau and the topological data of the experimental plot with OpenStreetMap have been used as key initial data for our simulations.
In accordance to the ontology subset and the ISO26262, a probability level and a criticality one, from A to E, has been defined with the partners for each parameter. The objective is to calculate the set of possible scenarios resulting from the combination of these parameters. However. since we need that these scenarios are coherent, any conjunction of parameters which have been considered unrealistic were excluded from this combinatory. In addition, several constraints have been defined as logical rules, especially exclusion rules or implication rules, for example:

- $\neg(crossRoad \land roundAbout)$ exclusion rule based on the fact that a scenario cannot have both an intersection and a roundabout.
- $fog \longrightarrow lowGrip$ implication rule which affirms that the presence of fog implies poor adhesion.

Therefore, considering 45 parameters, 22 constraints have been set and the final number of scenarios has resulted 1012. Subsequently, since it is not possible to analyze all the scenarios, these have been reduced with a classification at the level of probability and criticality based on the following two formulas:

$$P_g = \prod P_i \tag{3.1}$$

$$C_g = \sum 2^{C_i} \tag{3.2}$$

The first Formula (3.1) aims at calculating the total probability for each scenario and the second 3.2 the total criticality. Considering five probability and criticality components, 165 scenarios were distributed on a Cartesian coordinate system with the criticality in the x axis and the probability in y axis, and finally only 1134 scenarios were selected for the analysis on the pareto front, Figure 3.2. Once the relevant scenarios were selected, we could simulate them to gain a more accurate understanding by observing emerging behaviors resulting from interactions between system actors.

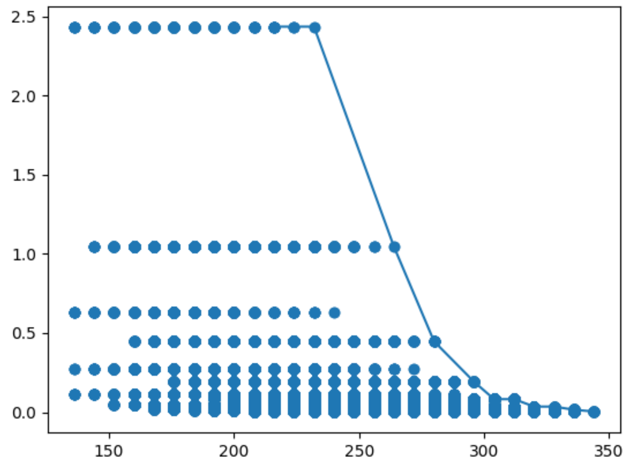


Figure 3.2: Pareto front of the generated scenarios.

3.3 SUMO and Webots

For the scenario simulation, we chose to use a multiagent system integrated with a more detailed simulation of the vehicle. The use of such a system is relevant when we need to describe unsupervised systems where the behavior is unpredictable [5].

In our mobility service, semi-supervised actors are responsible for making decisions based on their perceptions of the environment. However they can also receive orders to execute. This is the case of autonomous vehicles which can interact with the supervision center. On the other hand, some different actors remain without external control and their behavior produces situations that affect our autonomous vehicle [6].

SUMO Simulation of Urban Mobility Overview

SUMO is a microscopic traffic simulator open-source, which allows to model and simulate inter modal traffic configurations consisting on pedestrians, various types of vehicles, buses, and cycles. A SUMO network is structured in:

- An oriented graph whose nodes represent the intersections and edges model unidirectional ways between them.
- An edge is composed of lanes (at least one) which have separately their own authorized vehicle types and speed limit.
- Links between edges at junctions are modeled with connections between their lanes.

SUMO networks are encoded in XML files, which contain all the edges of the network, the list of the lanes for each edge, and the junctions with their priority definition and connections. Therefore, SUMO consists of different modules that allow to process the data. For instance, one of these modules is SUMO-netconverter. In details, this application can import OSM (OpenStreetMap) files [7]. Moreover, it is possible to modify the network imported with the SUMO-netedit module which provides an intuitive graphical interface. Once we have generated the network, we need to add the description of the vehicles that will be presented in our simulation. In this way, we will define what is called traffic demand. This step can be carried out in two different ways : specifying all the edges that make up the traveling path of the vehicle has to travel, or specifying the starting and destination edges (potentially intermediate edges) so that the simulator is responsible for calculating the shortest path in terms of distance. Then, the road of a vehicle can be changed while running using reroutes.

Furthermore, instantiating a vehicle is implemented in SUMO by specifying its type (a default type can also be used), its path, and the parameters whose values we want to be different from those defined in the type (color , number of places. . .). It is also possible to define regular vehicle fluxes. Furthermore, SUMO allows each vehicle to be assigned a class of emission based on four predefined models including an electric vehicle model (important element for the analysis of the simulation results).

Moreover, it is possible to simulate public transport in SUMO by importing OpenStreetMap's public transport data for the selected area, define or import bus stop positions, and let the vehicles stop there for a determined time. However, it is necessary to specify pedestrian access points at each bus stop from other network locations. SUMO pedestrians are treated differently because of their different actions compared to vehicles. Pedestrians can move through the network by walking or using a vehicle. Since the definition of a pedestrian in the simulation includes all its stages, its action was classified by type [8] :

- Walking: specifying the arcs to go (at least one lane for pedestrians).
- Stopping: specifying the breakpoint and its duration.
- Taking a public transport vehicle: specifying the arrival point (assuming the person is already at the point of climb), the transport line, and the type of vehicle used.

TraCI

TraCI is a python API which is able to have online interaction with SUMO simulation. It uses a TCP client / server architecture in which SUMO takes over the role of the server and TraCI acts as a client. Moreover, it is possible to control parallel simulations with the same TraCI script or the same simulation from different clients connected simultaneously to the same server. Specifically, the simulation is controlled by TraCI in the sense that the python script gives the order at each time step. TraCI commands are divided into 13 different domains according to the simulation elements they allow to control (vehicle, person, traffic light, . . .) [9]. For our purposes, it is interesting to study the possibility to introduce dynamic adaptation of the public transport and dangerous vehicle behaviors in order to create and deal with critical situations.

Why Webots?

Unfortunately, SUMO does not take into account the physical aspects that could affect the behavior of vehicles, such as wet roads, fog, sudden pedestrian crossing, and so on. To overcome this problem, we decided to interface SUMO with a robotic simulator. After comparing the various existing solutions (see next chapter) [10], we selected a simulator called Webots. Indeed, the integration of these two tools was useful to combine the macroscopic or physical view provided by Webots and the microscopic view provided by SUMO. Thus, we are able to perform simulations which can take into account the physical aspects of the road (variation adhesion, inclination. . .) as well as the perception of autonomous vehicles by using the characteristics of their sensors.

Comparison of Simulators

The connection of SUMO to a robotic simulator has led us to comparing different solutions in order to determine which one would be the most suited to meet our requirements. Therefore, we have defined four relevant comparison criteria:

- Driving environment: it is an indication of the possibility to modify the simulation environment from real maps.
- Non-Player Characters Control: it shows the ability to control the simulation actors with whom the vehicle interacts.
- Documentation: it has a detailed documentation on Internet as well as an active developer community.
- Interface with SUMO: the presence of an interface which is already developed with SUMO.

Then, to compare the four best open-source robot simulators from [10] according to these criteria : CARLA [11], AirSim [12], Webots [13], and Gazebo [14].

Simulator	Driving environment	Non-Player Characters Control	Documentation	Interface with SUMO
CARLA	NO	NO	Poor	NO
Airsim	YES	NO	Poor	NO
Webots	YES	YES	Poor	YES
Gazebo	YES	YES	Rich	NO

Table 3.1: Comparison of robotics simulators.

Table 3.1 shows that the the most accurate simulators result to be Webots and Gazebo. In conclusion, Webots was chosen as the most appropriate one especially for the flexibility of the programming languages that can be used. Indeed, Webots is compatible with C, C++, Java, Python, Matlab and RO [13].Moreover, the presence of in interface with SUMO was an additional key point for this simulator decision.

Webots

Webots is an open-source robotic simulator with efficient tools that allow to develop autonomous vehicle control models. Furthermore, it is possible to generate complete simulation 3D environments with real road networks imported directly from OpenStreetMap [7] which are translated into a data structure suitable by Webots. The OSM file describing the OpenStreetMap map has various types of road segments, called ways, which are composed by nodes and properties. The most important properties used in the conversion OSM $\longrightarrow$ Webots are: type of segment (from highways to residential streets), direction of road, and number of ways . These properties are used for generating the Webots edges by connecting the lanes at intersections. Webots, as indicated previously, offers the possibility to modify a posteriori the simulation environment by adding variable meteorological conditions such as visibility reduced by the presence of fog, variation of the coefficient of friction caused by the presence of water on the road (or even snow), and variable lighting conditions. Therefore, in conclusion, the interface with SUMO ensures the introduction of a large number of vehicles in Webots which recreate a real traffic situation. SUMO vehicles are created and moved to Webots by using vehicle information retrieved from SUMO with TraCI and the Webots Supervisor API. Figure 3.3 shows how the interface SUMO-Webots works. This is also used to simulate human intervention [15]. Moreover, the Supervisor API is helpful to restart or end the simulation, to read or change the position of the various actors of the simulation, and to read the status red lights. On the other hand, the autonomous vehicle controlled in Webots is moved to SUMO using TraCI.

Model of the Vehicle in Webots

The model of the vehicle used in Webots is composed by a body and four wheels. The body of the vehicle can move with 6 degrees of freedom, the wheels can turn and move with the body, and the steering wheels can also steer. To summarize, the model has 16 degrees of freedom in total. The model incorporates basic rigid dynamics properties including Ackermann's steering dynamics. Figure 3.4 shows the evolution of the speed during the acceleration and deceleration of the vehicle. Figure 3.4a shows the evolution of the speed in the simulation for target cruising speed for the vehicle of $100\frac{km}{h}$. Meanwhile Figure 3.4b

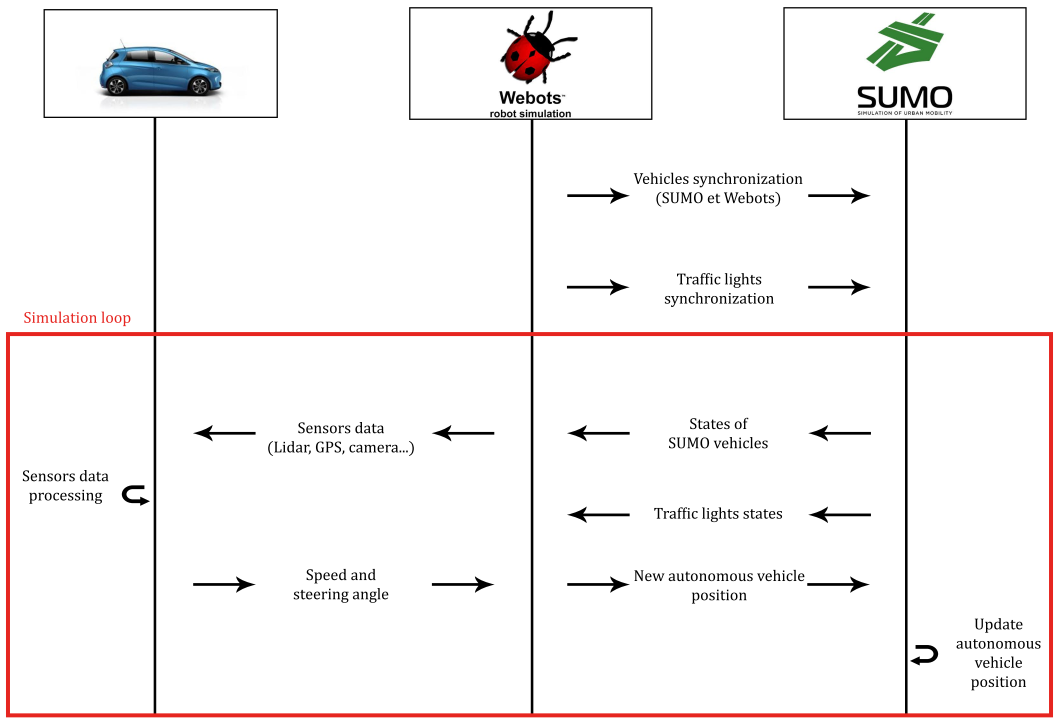

Webots robot simulation
SUMO
Vehicles synchronization (SUMO et Webots)
Traffic lights synchronization
Simulation loop
Sensors data (Lidar, GPS, camera...)
States of SUMO vehicles
Sensors data processing
Traffic lights states
Speed and steering angle
New autonomous vehicle position
Update autonomous vehicle position

Figure 3.3: Scheme of the interaction between autonomous vehicle and SUMO via Webots.

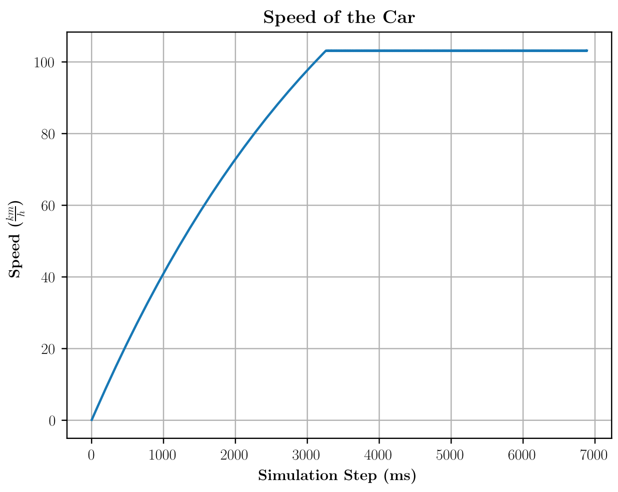
shows the evolution of the speed when the desired speed is set to $0\frac{km}{h}$ and the brake intensity is maximum.

In addition, a wide range of sensors is available and useful in Webots to operate the autonomous vehicle (GPS, Camera, Lidar, distance sensors, and so on). Furthermore, the numerous available different configurations give the system a great flexibility.

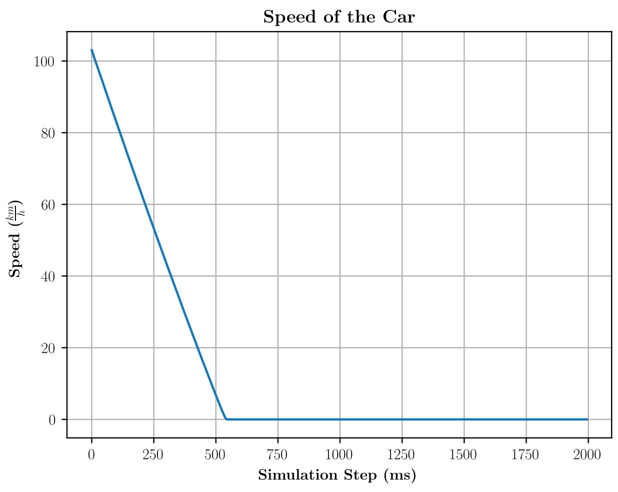
3.4 Reinforcement Learning

Reinforcement learning [16] is one of the several sub-domains that contributes to the vast domain of machine learning. A well-structured operation of the reinforcement learning algorithm involves the use of an agent that performs actions (a). These actions change the environment, changing the agent's state (s).
Whenever the agent change its state, it receives a reward (r). The reinforcement learning framework is shown in Figure 3.5.

To determine the next action to perform, the choice of the agent is based on a policy π. So π gives to the agent a mapping from states to actions.



(a) Vehicle acceleration.



(b) Vehicle deceleration.

Figure 3.4: Evolution of the speed in the simulation.

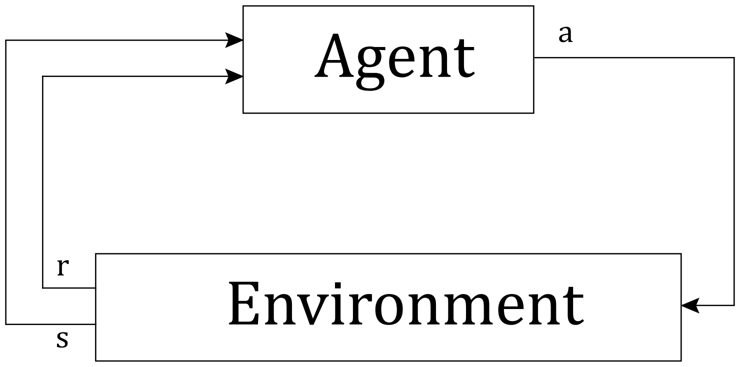


Figure 3.5: Reinforcement learnign framework with : reward (r), state (s) and action (a).

The learning of the policy is split into episodes. For each learning episode, the ultimate goal is to find a policy π^*, using the state s and the rewards r, to optimize the estimate of the final reward [17] :

$$R(\pi, r) = \mathbb{E}_\pi[\sum_{t=0} \gamma^t r(s_t, a_t)] \tag{3.3}$$

Formula 3.3 is useful to find the value of the reward, where $\mathbb{E}_\pi$ is the expected value obtained following the policy π. To get the policy, the reinforcement learning algorithms are separated into two big groups: model-based methods which require a model of the environment where the agent acts, and model-free method which do not require a description of the environment. In our case we chose to study the model-free algorithm and, in particular, the Q-learning.

Q-Learning

For a Q-learning [18] algorithm the function $Q(s, a)$ shows the quality of the action a when the agent is in the state s. A learning episode is defined as (s, a, r, s_{t+1}) in which the agent is in the state s, performs an action a, receives a reward r, and moves in the sate s_{t+1}. For each time step, the update of the $Q(s, a)$ function is made by knowing that the optimal policy is given by the choice of the action a^* such that :

$$a^* = arg \max_a Q(s, a) \tag{3.4}$$

The key point is the approximation of the Q function. In this way the action a_{t+1} at the time step t is chosen with the actual optimal policy :

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \tag{3.5}$$

Formula 3.5 is the Bellman Equation [19], from which can be derived :

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{3.6}$$

In Formula 3.6, α is defined as the learning rate. The function is the Bellman function is $\alpha = 1$, meanwhile the update converges to the Bellman equation, but in a slower way, if $\alpha < 1$.

3.5 Deep Reinforcement Learning

The deep reinforcement learning algorithms was born with the study of Deep Q-Networks (DQNs) [20]. Indeed, these networks integrate deep convolutional neural network with the Q-learning algorithm. In this way, one can use a neural network to estimate the Q function.

Deep Q-Learning

A deep Q-learning algorithm is obtained by combining a Deep Q-network with a replay experience [21], which is a simple buffer that saves old experiences and uses them for learning. With the network, we estimate the value of the rewards for the actions $Q(s, a)$. At each step of the simulation, starting from the state s_t, first we choose an action a_t with a policy derived from Q, then we observe the state s_{t+1} and the reward r_{t+1} obtained by applying such action. The update of the weights aims at the minimization of the loss function, the difference between the estimation of the reward, and its real value, as shown in 3.7:

$$loss = (r + \gamma^* \max_a \hat{Q}(s, a') - Q(s, a))^2 \tag{3.7}$$

where the first term represents the target and the second term the prediction.

Chapter 4

WORK DONE

Since we did not have access to both the autonomous vehicle control model and the specifications of the sensors used by the EVA project industrial partners, we decided to develop perception, navigation, and control algorithms for each scenario in order to study the interactions of the systems. In addition, for the EVA System of Systems task, the autonomous vehicle was considered a black box so the task aimed at the study of the impact of the mobility service on the overall system (traffic, issue of CO_2, communication systems,. . .). In all scenarios, we equipped the autonomous vehicle with:

1 Lidar SICK LMS 291 [22] : a single-layer lidar with a range of up to 80 meters and a perception angle of 180 degrees. In Webots, the Sick lms 291 features a scalable spherical projection and Gaussian noise that we have set to zero for our model. Then, the Sick returns a vector size N (N defined by the sensor resolution), and each element of the vector indicates the distance measured by the laser at a specific angle.

1 camera : a generic camera that allows us to take pictures and then process them. All camera features (resolution, viewing angle, noise. . .) are adjustable. It is also possible to enable the object recognition feature for the camera. This function is very interesting because it allowed us to detect pedestrians, traffic lights, and cars in the simulation without developing a segmentation algorithm.

11 infra-red distance sensors : distance sensors that detect collisions with environmental objects. What is interesting with the sensors provided by Webots is the possibility to specify the values returned by the sensor as a function of the measured distance.

4.1 Scenario 1

For the first scenario, we decided to simulate a critical situation for the autonomous vehicle in terms of decision-making and vehicle-control communication center. Figure 4.1 shows a helicopter view of this scenario:

- The road presents a bottleneck, which is interesting to see how the autonomous vehicle, with its catches, ensures the safe lane change and, if possible, respects the different priorities.

- A broken car in the curve, which limits visibility and therefore the ability to detect the vehicle in the opposite lane.
- The need to cross the yellow line to avoid and pass the obstacle.

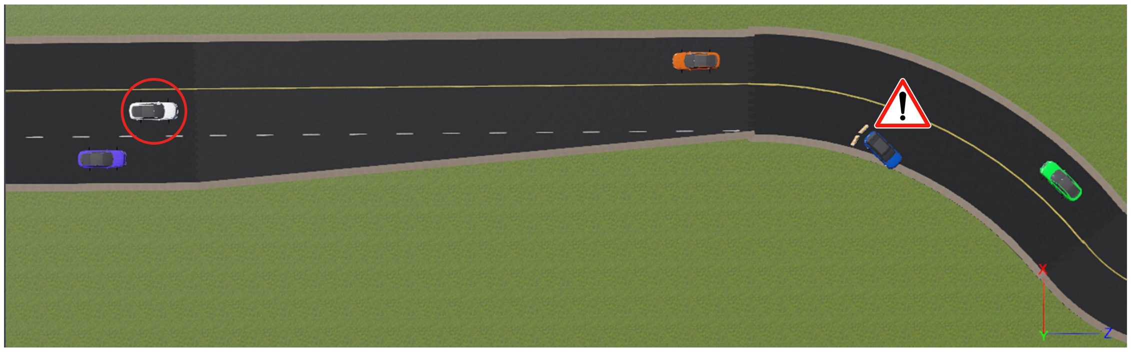


Figure 4.1: Screenshot from the simulation of the first scenario, we have highlighted: the autonomous vehicle (Red circle) and the broken vehicle (Danger signal).

Algorithms Developed for the Scenario 1

Detection and Following of the Yellow Line

The yellow line tracking algorithm (Appendix A.3) was based on the image's pixels analysis of the camera. The camera image was coded as a sequence of three entries represented by the blue, green, and red level of the pixel. Since the pixels were stored in horizontal lines, we obtained a matrix $\mathbb{R}^{WxH}$, where W and H are respectively the width and height of the image expressed in pixels.

The developed algorithm analyzed the image of the camera at each simulation step by searching for all pixels that have a BGR value in the range of the BGR line values. Therefore, we took the three values for the yellow line and then we fixed the threshold values between ±10. Furthermore, we considered a pixel P_{coleur} belonging to yellow line to follow if :

$$193 < P_B < 213$$
$$177 < P_G < 197$$
$$85 < P_R < 105$$

Once the pixels belonging to the yellow line were determined, their average position in the image was computed. Then, we calculated the distance between this mean and the center of the image in order to apply equation (4.1) which links the horizontal linear distance to the angle between the center of the image and the yellow line as shown in Figure 4.2.

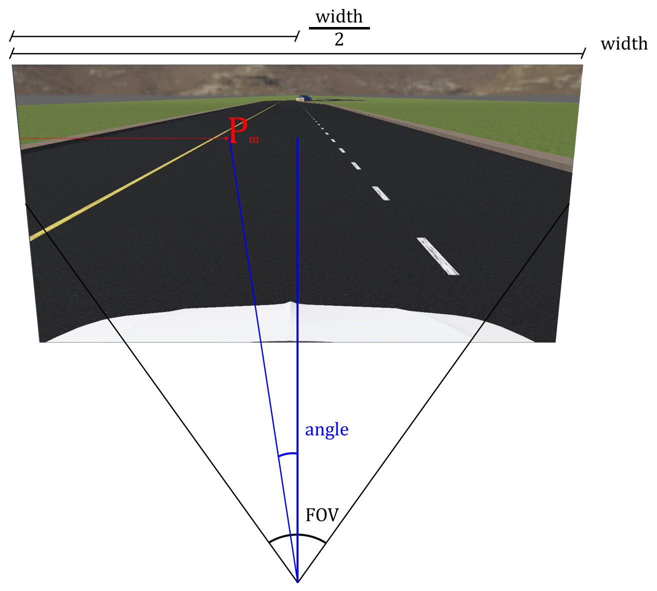


Figure 4.2: Field of view of the camera with the geometric components.

$$\frac{angle}{FOV} = \frac{P_m - \frac{width}{2}}{width} \tag{4.1}$$

FOV is the field of view and $(P_m - \frac{width}{2})$ is the distance between the mean position of the yellow pixels and the center of the image.

Once we determined the angle between the yellow line and the vehicle, we could correct the position of the vehicle in the lane by applying an angle control via PID (Appendix A.1) by comparing the angle found with a fixed angle called target_angle. The block scheme for the PID controller is shown in Figure 4.3. The target_angle is, therefore, fixed in order to maintain the vehicle in the center of the lane.

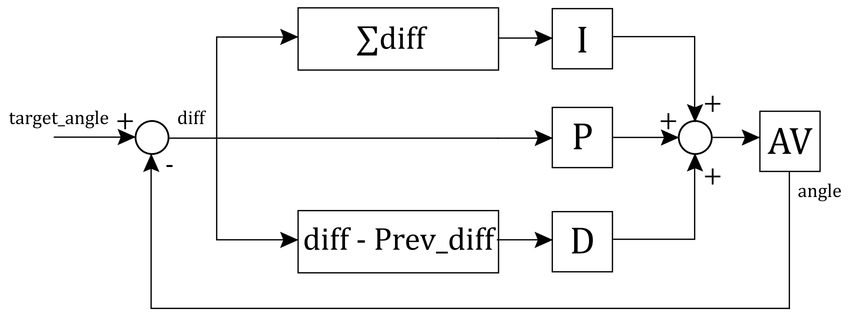


Figure 4.3: Block scheme for the PID regulation.

Algorithm 1 Algorithm to find the angle and to apply the PID

```
Set the thresholds for yellow
$number\_yellow\_pixels = 0$
Set the values P, I et D
set the value of the target_angle
**for** Each pixel of the image **do**
    **if** $P_B$ and $P_G$ and $P_R$ in the thresholds **then**
        Save of the relative position with respect to the width of the image
        Add the position to the collected sum of past positions
        $number\_yellow\_pixels = number\_yellow\_pixels + 1$
    **end if**
**end for**
Compute the average position
Compute the angle of the line
Compute the difference $Diff = target\_angle - angle$
$Integral = \sum Diff$
Compute the input angle :
$input\_angle| = P * Diff + I * Integral + D * (Diff - PreviousDiff)$
Save of the difference $PreviousDiff = Diff$
```

Obstacles Detection

The algorithm for the detection of the obstacles (Appendix A.2, A.4) is based on the analysis of the vector of values ordered from left to right. The field of vision of the SICK is 180 Degrees, but we reduced the arc of analysis to 130 degrees as shown in Figure 4.4. The algorithm calculated the average distance of the distances given by the lidar that are less than 15 meters (we considered these points to belong to the same obstacle). Subsequently, we calculated the angle of the object in relation to the vehicle by applying Formula 4.2 analogous to the previous:

$$\frac{angle}{FOV} = \frac{X_m - \frac{width}{2}}{width} \tag{4.2}$$

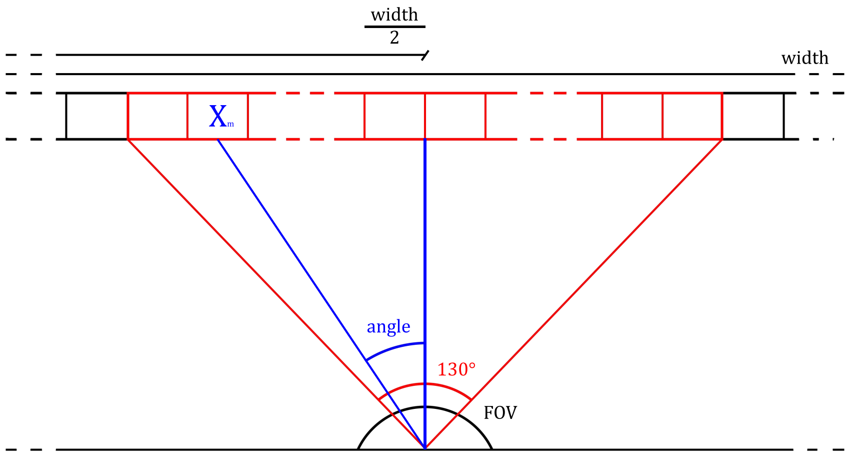


Figure 4.4: Field of view of the SICK with the geometric components.

In Formula (4.2), $(X_m - \frac{width}{2})$ is the average distance of the obstacle from the lidar sensor. The angle obtained and the mean distance of the object were used to calculate the control angle needed to avoid the obstacle, as shown in 4.3:

$$angle = actual\_steering \pm \frac{obstacle\_angle + 1}{obstacle\_distance} \tag{4.3}$$

In Formula (4.3) the choice of ± depends on the angle sign of the object. Indeed, with Formula (4.2), one gets a positive angle if the obstacle is on the right, and a negative angle of control for the car to turn left. If the angle obtained is negative, the obstacle is on the left and therefore a positive control angle is required to turn right.

Algorithm 2 Algorithm to find the angle of the obstacle and the input angle to avoid the obstacle

```
Set the minimum distance
$analysed\_zone = 130$
Initialization of collision points
Initialization of distances sum
for each entry of the vector in the range of ($\frac{width\_lidar}{2} \pm analysed\_zone$) do
    if Value in the range of distance then
        Save position in the vector
        Sum of the collision points
        Sum the distances
    end if
end for
Compute mean distance
Compute the angle of the object
Compute the input angle as function of the obstacle angle and the mean distance
```

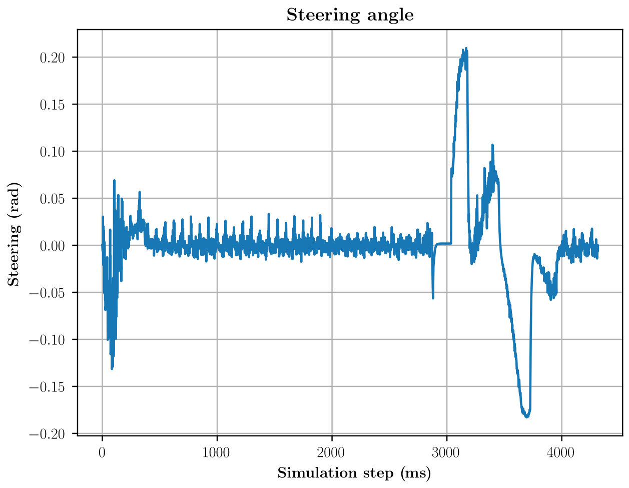
The simulation of the scenario, with various functions offered by Webots and SUMO, allowed us to analyze the autonomous vehicle's path in terms of performances of the model developed and in relation to the impact of these decisions on the global systems. During the simulation of the first scenario, we studied the evolution of the control angle in order to improve the vehicle's response to the changes in its position in the lane. Therefore, by analyzing the results described in Figure 4.5, one can see that if we choose the three values indicated in Figure 4.5a for our PID regulator we got a response with less oscillations.

The advantage of using Webots coupled with SUMO was that once the simulation is complete, we could retrieve a series of information about the journey of each car in the scenario. By examining the output information in SUMO, we learnt how, in Figure 4.6, the presence (and the detection) of an obstacle on the road influences the variation in the speed of the autonomous vehicle. This is also translated into a change in the length of the journey.

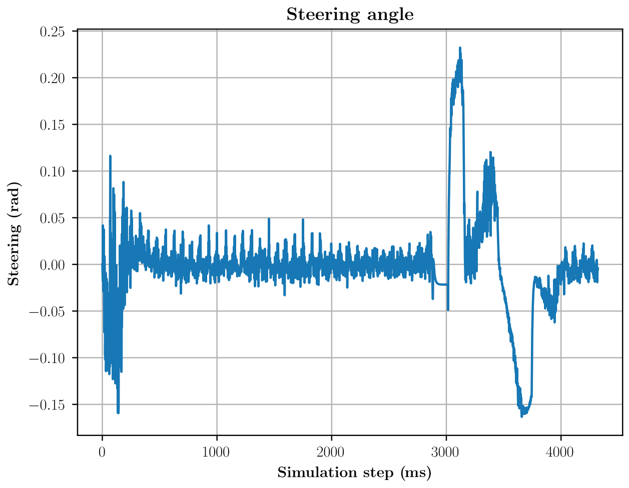
The simulation of this scenario was presented in the demo of the EVA project's Sos task in occasion of the DigiHallDay 2019, an event in which different participants demonstrated their latest innovations in the fields of artificial intelligence, cybersecurity, cyber-physical systems, and industry of the future.

Improvement on the Scenario 1

Subsequently, we added two additional functions to the Autonomous Vehicle behavioral model. Indeed, a discussion with the partners underlined that the most natural behavior for



(a) Using $P = 0.15$, $I = 0.006$, $D = 10$.



(b) Using $P = 0.25$, $I = 0.006$, $D = 15$.

Figure 4.5: Variation of the input angle in the simulation for different parameters of the PID regulator.

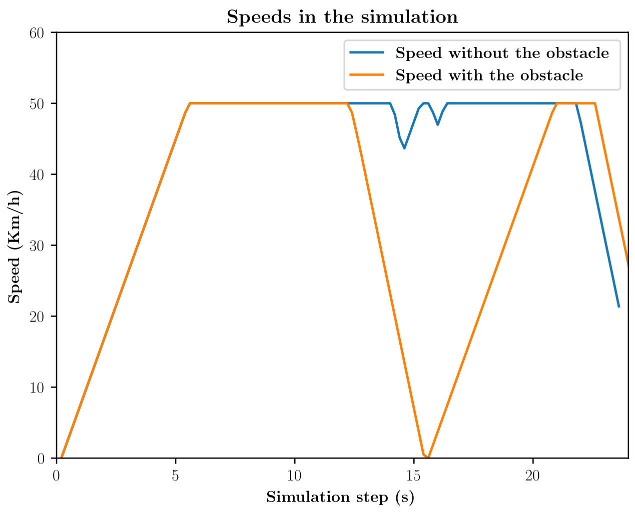


Figure 4.6: Comparison of autonomous vehicle speeds in SUMO.

the vehicle, in case of critical situation or incorrect operation, was to stop, communicate the critical situation to the control center, and wait that it communicates the action to take. In this perspective, we added the possibility to control the vehicle by using the keyboard to better simulate the intervention of the supervision center.

We used the sensors to calculate the distance between the obstacle and the car which is, then, compared with the calculated braking distance as :

$$d = \frac{v^2}{250 * f} \tag{4.4}$$

where v is the speed of the vehicle and f is the coefficient of friction, that we considered equal to 0.8. At this distance we added a gap of $2m$ to avoid an abrupt braking. Therefore, when the distance from the obstacle is less than d, in Formule 4.4. The control speed goes to zero and the braking intensity is adjusted according to the distance between the vehicle and the obstacle.

By comparing Figures 4.7 and 4.6, we can observe the main difference between the two behaviors in the time interval between 10 and 15, which corresponds to the autonomous vehicle passing in the neighborhood of the obstacle. The time period in which the vehicle speed is less than $10\frac{km}{h}$ corresponds to the time required to start the communication with the supervision center, which evaluates the situation and returns the action to perform to the vehicle. We could also simulate the same situation with different communication delays and notice how this impacts the traffic situation in terms of CO_2 emissions, battery consumption, and travel time.

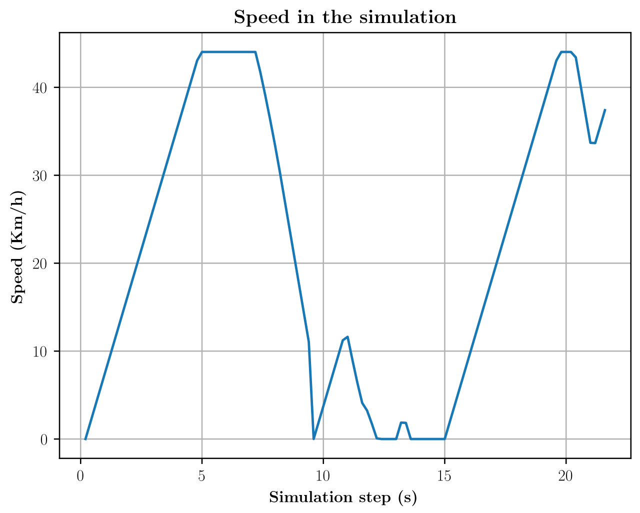


Figure 4.7: Speed of the autonomous vehicle with the simulation of the supervision center.

The simulation of Scenario 1 with all the improvements to the vehicle model was exposed at the Autonomous LAB project' stand during the SPRING 2019 day [23].

4.2 Scenario 2

In order to bring the simulation of the scenarios closer to the reality, we chose to analyze the behavior of the vehicle in a simulation environment which corresponds to the experimental path of autonomous vehicles on the Plateau Paris-Saclay. The area analyzed was divided into two different routes. The first road goes from Massy-Palaiseau to the roundabout of Camille Claudel reserved for the experimentation of the Transdev shuttles, the second road settles in the École Polytechnique campus reserved for Renault Zoe autonomous vehicles. Therefore, we extracted the experimental path from the OSM file of the Plateau de Paris-Saclay that we previously edited using JOSM [24]. Since the speed of the simulation depends on the complexity of the 3D environment, we only kept the roads that concern us by including the informations on the bus lines and the traffic lights. Once the OSM file was modified, we converted it into 3D world by using the tool provided by Webots, result in Figure 4.8.

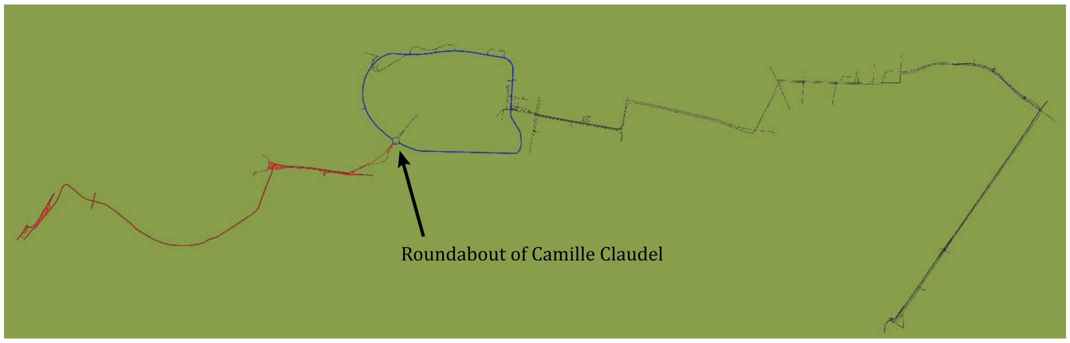


Figure 4.8: Screenshot of the simulation for the the second scenario, the path of the bus 91.06 in Webots. In red the path reserved for the Transdev autonomous shuttles and in blu the path for the Renault Robotaxis.

The generated environment presented the following problems :

- The roads, especially the main of 91.06 [25], normally two-lane, were exported as a single lane.
- All the traffic lights present in the selected route were not imported.
- Variable slope areas were not imported correctly.

The resolution of these problems required manual intervention on the simulation environment to modify the various uncorrected elements. Once these changes were completed, we synchronized the traffic lights with SUMO so that SUMO could control them. Moreover, for the scenario 2 we integrated a partial risk warning system, following [26], adding the functionality for the vehicle to send a message displayed on the camera screen, whenever a critical situation is detected.

Developed Algorithms for the Scenario 2

Traffic Lights Color Detection

The detection of the traffic light state (Appendix A.5) in our model was implemented using the object recognition function (Appendix A.6), already developed in Webots camera. This is able to differentiate between pedestrians, vehicles, traffic lights, signals, and all others available objects. It is also possible to find the distance of these objects from the vehicle. In this case the camera returns a vector $\mathbb{R}^{3x1}$ corresponding to the three coordinates $[x\ y\ z]$ of the object in the three-dimensional reference frame having the origin in the camera. We calculated the distance as the vector module. Once the traffic light was detected, if the distance was less than a certain threshold, the algorithm processed the image to identify the color of the traffic light. If it was red, the braking action was generated according to the procedure used in scenario 1 described in Formule (4.4).

Algorithm 3 Algorithm for the detection of the traffic light color

```
Set the three thresholds for the three colors red, yellow and green
if Distance of the traffic light less than fixed distance then
    Image conversion from BRG to HSV
    Application of three different masks to highlight the current color
    if the color is not green then
        Brake the vehicle
    end if
end if
```

Slope Detection

For the slope detection, we initially used the information about the component of the vehicle's speed along the y axis, that was possible to compute by knowing the angle α of the car with respect to the z axis, easy to find using the supervisor functions of Webots applying Formule (4.5).

$$v_y = v \sin(\alpha) \tag{4.5}$$

Finally, we chose to simplify the algorithm from a computational point of view, by analyzing the information that comes from GPS sensor. Specifically, we obtained the position of the vehicle in the simulation environment expressed by a three dimensional vector $[p_x \; p_y \; p_z]$ as show in Figure 4.9.

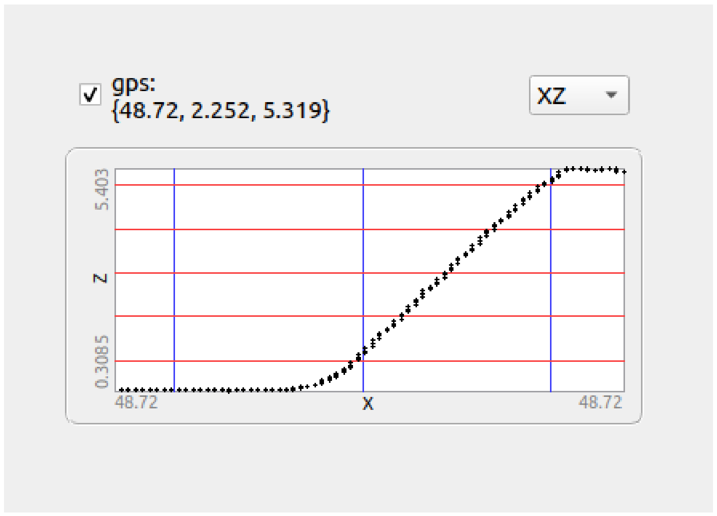


Figure 4.9: Acquired GPS data in presence of a slope, plotted in a xz reference frame.

Zebra Crossing Detection and Pedestrians Detection

For the detection of zebra crossing and pedestrians in the simulation, we started with the same procedure used for the traffic lights, using the recognition function of the Webots camera. Once we identified the pedestrian, we analyzed five different situations:

1. Pedestrian crossing respecting the traffic light.
2. Pedestrian crossing violating the traffic light indication.
3. Pedestrian crossing on a zebra crossing not regulated by a traffic light.
4. Pedestrian crossing out of the zebra crossing.
5. Pedestrian waiting the right indication from the traffic light.

Situations 2 and 4 were considered critical for the vehicle. In these cases, we estimated that the vehicle needs to wait instructions from the control center. To distinguish cases 1 and 2 from case 5, we analyzed the position of the pedestrian in relation to the camera. In particular, the position along the x axis of the reference frame associated with the camera, for the simulation step $t + 1$ a greater value of x could be translated with a shift of the pedestrian position, so the pedestrian was considered to be crossing the road.

Pedestrian Behavioral Model

For the implementation of pedestrian behavior (Appendix A.7) we used the functions of the supervisor as it makes it easy to obtain information on the distance between the vehicle and the pedestrian and information about the current color of the traffic light. It also allowed pedestrians to move around the simulation environment. Therefore, we developed an algorithm for the pedestrian to cross the street both when the traffic light was red and green for vehicles and the autonomous vehicle was close to the pedestrian.
The main problem regarding pedestrian simulation was to transfer them from Webots to SUMO. Indeed, the interface developed did not allow us to import the pedestrian from Webots to SUMO, so we modified the code of the coupling interface in order to have the pedestrian, although considered as a small vehicle, in the SUMO simulation.

Requirements and Functions Simulation and Validation

From the reference architecture ARC-IT [27], adapted to the context of the system of systems of the EVA project and in agreement with the partners, we defined the functional areas to be addressed. Mainly the focus was pointed on the relationship between the vehicle and its on-board systems such as: the ability of the vehicle to provide information based on the data collected and to process that data, automatic vehicle condition and emergency management.
Under the behavioural model assumptions, the function validation was performed using the Webots Window Robot, which monitors the data received and the status of the vehicle. The process of importing ARC-IT data to CAPELLA [28] and the next verification and improvement by simulation is shown in Figure 4.10.

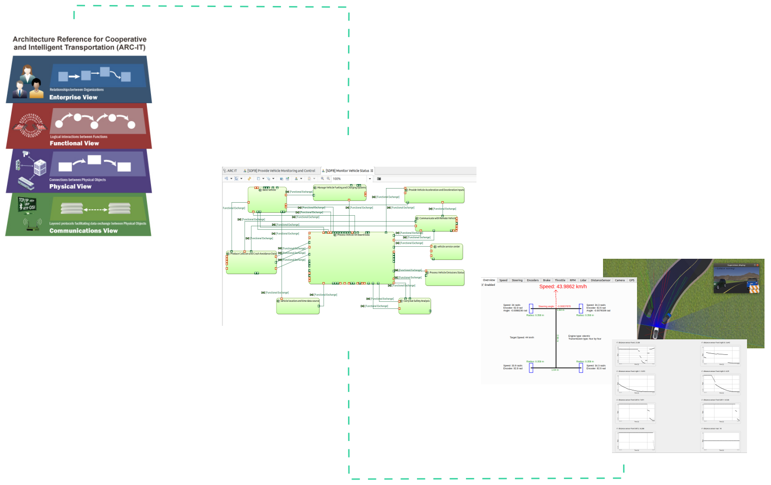

Architecture Reference for Cooperative and Intelligent Transportation (ARC-IT)
Enterprise View
Functional View
Physical View
Communications View

Figure 4.10: Verification and improvements process for the ARC-IT functions.

4.3 Deep Reinforcement Learning for the Autonomous Vehicle

In parallel to the implementation of the scenarios to simulate the behaviour of the vehicle and the development of the algorithms needed, we implemented a deep Reinforcement learning algorithm for the navigation and obstacle avoidance of the autonomous vehicle. By using a tool like Webots, we could set up a system that allowed us to play simulations with a trained model and at the same time improve it with continuous learning. The training system was composed by of two parts:

- The deep reinforcement learning algorithm that controls the autonomous vehicle agent.
- The Webots simulator that provides the learning environment.

Controller Architecture

The central part of the model was a controller based on a deep Q-network, which receives in inputs the data of the Lidar SICK lms 291, which are centered and normalized to have values in the interval [-1, 1] and average equal to zero.

Figure 4.11 provides a better understanding of our model architecture, key components, and interaction with the simulation environment.

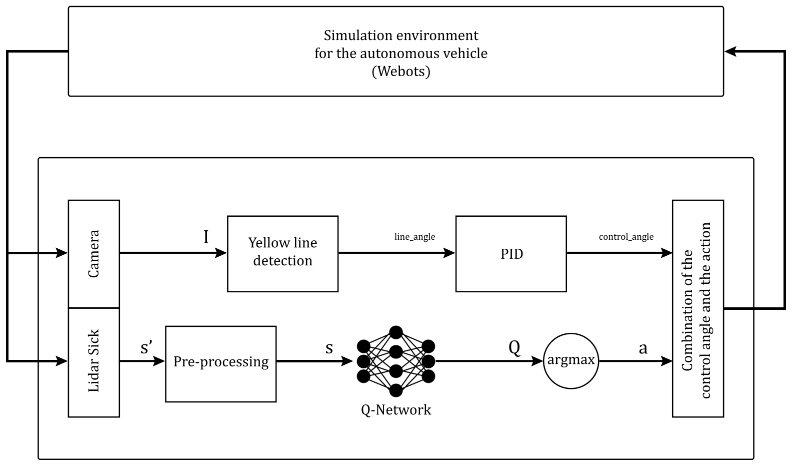


Figure 4.11: Controller scheme.

The controller sends the order to the simulated vehicle. Then, the deep Q-network takes the Lidar values as a state and produces an estimate of the rewards associated with each possible action.

Training Scheme

The training was done in episodes. In order to avoid a biased model, we made the training independently from the starting position of the vehicle, by randomly defining it at the beginning of each episode. The same procedure was applied to the obstacles positions. The speed of the vehicle during the training was fixed at 20 $\frac{km}{h}$, while the resolution of the Lidar was fixed to 360. The DQN network also took as input the current steering angle of the vehicle. In this way, we obtained a state with a dimension of 361. Recalling that the main purpose of the model is to drive the vehicle on a trajectory and if necessary avoid obstacles, we combined the controller with the PID developed for tracking the yellow line.
For the determination of the actions to be chosen from the network, we discretized the continuous set of possible turning angles [-1, 1] into five values:

1. **THL** Turn Hard Left : in this case the obstacle is very close to the vehicle and on its

right side. The vehicle brakes to decrease its speed and the control angle is set equal to $(PID\_angle - 0.65)$.

2. **TSL** Turn Soft Left : when the obstacle is on its right side and the, the control angle is set equal to $(PID\_angle - 0.2)$.

3. **THR** Turn Hard Right : the obstacle is very close to the vehicle and on its left side. The vehicle brakes to decrease its speed and the control angle is set equal to $(PID\_angle + 0.65)$.

4. **TSR** Turn Soft Right : when the obstacle is on the vehicle's right side and the control angle is set equal to $(PID\_angle + 0.2)$.

5. **S** Straight : not obstacles to avoid.

For collision detection we equipped the vehicle with two additional Lidar each of 4 levels on the right and on left of the vehicle. A collision was detected if the measured distances were less than 1 m.
Our DQN network consisted of 3 hidden layers with 600 neurons in the first layer and 300 for the other two layers. These layers used a ReLu activation function [29], while the output one, with 5 neurons, used a linear activation function.

Specifically, in order to improve the results, we applied change in the traditional DQN architecture. We implemented an additional network, called target network, to make the learning more robust with an estimation of the actions of the agent [30]. In this way we have two networks. The Q-network calculated $Q(s', a)$ with s', for all the actions a, after we chose the action a^* applying the argmax for $Q(s', a)$. Indeed, finally the quality of $Q(s', a^*)$ was selected for the calculation of the target, shown in Figure 3.7. So the target-network determined $Q(s', a^*)$, while the Q-Network the action a^*.
We tested several functions for the reward policy in order to implement different vehicle behaviors. Finally, at each step, the reward for learning was determined according to the Table system 4.1.

Condition	Reward
Collision	-10
Target position	+10
Less target distance[1]	+0.05
Bigger target distance[2]	-0.05

Table 4.1: Reward system.

For our learning algorithm we set the replay experience dimension to 100, the discount rate γ to 0.95, and the learning speed α to 0.001.

We modified the algorithm by changing the loss function for the weight update and we finally used the Huber loss function [31].

Training Evaluation

To evaluate the progress of the learning we could study the evolution of the reward for each episode as well as the distance covered by the vehicle (computed as the length of the vector between the initial position and the final position).

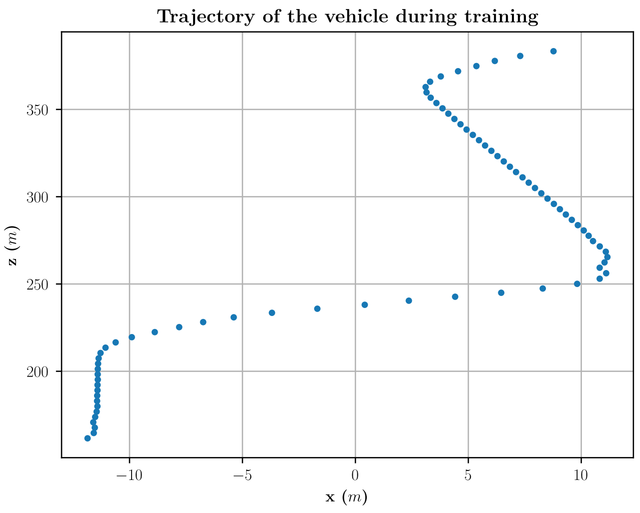


Figure 4.12: Trajectory of the vehicle without obstacles in the training environment.

The path shown in Figure 4.12 was obtained in an obstacle-free situation. In this case, the episode ends when the vehicle reaches the desired final position with a reward of 370 and a covered distance of 229.63 meters. Of course, the measured distance and the trajectory are influenced by the presence of a random GPS noise.

Figure 4.13 shows the average reward obtained on 10 episodes. The total number of episodes for learning amounted to 8446 episodes. The greatest reward was 365 and it was

[1] $d_{t+1} < d_t$, target position closer.
[2] $d_{t+1} > d_t$, target position farther.

obtained at episode 2457. By analysing also Figure 4.14 we can find a correspondence between the progression of the reward and the progression of the distance.
In both cases, we got different results for different initializations of the position of the vehicle and the obstacles. From the convergence of rewards and distances, we can state that the vehicle is learning how to follow the yellow line and how to avoid obstacles.

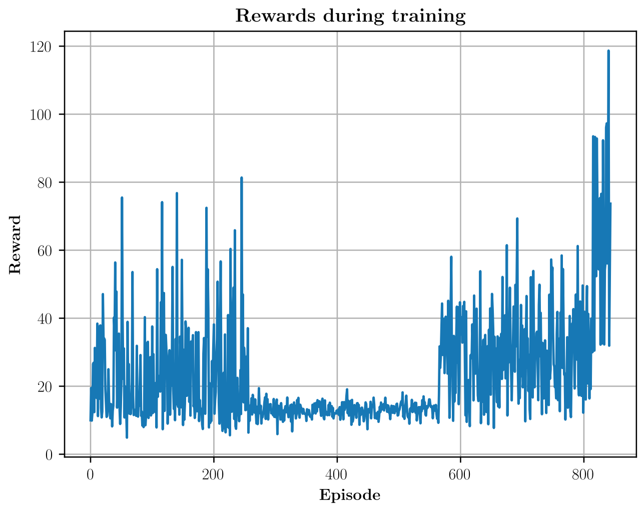


Figure 4.13: Average reward on 10 episodes for 8446 episodes.

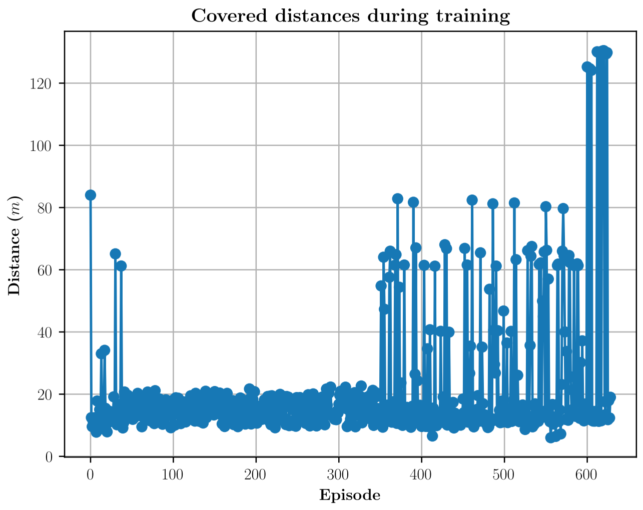


Figure 4.14: Covered distance during the training.

Finally, looking at the obtained results for the training, we concluded that to improve the training, it is preferable to use a DDPG (Deep Deterministic Policy Gradient) architecture [32]. This allows us to work directly with a continuous action space by eliminating the approximations provided for the discretization applied on the steering angle space.

Chapter 5

CONCLUSION

5.1 Discussion and Results

We studied a simulation system that integrates a microscopic view, provided by SUMO with a vision that addresses all physical aspects of our simulation environment and their interaction led by Webots. The simulations obtained combining the two tools, provided us important results, which involve the single autonomous vehicle and the entire traffic situation. The results steered decisions for the design of the system, choice of architecture for the mobility service, and finally the consolidation of the base of the requirements of the EVA project.

Scenario simulations are capable of providing comparable measurements, starting from the same assumptions in order to test vehicle data recorders [33].The study of the scenarios presented in sections 4.1 and 4.2 allowed us to analyze and validate vehicle functions such as:

- Ability of the vehicle to change its speed, with accelerations and decelerations, depending on the perception of the environment.
- Vehicle capability to collect and process sensor data.
- Ability of the vehicle to send control signals based on the data analyzed.
- Ability of the vehicle to provide effective communication of critical situations.

Moreover, with other simulations, especially on SUMO, we analyzed the impact of the autonomous agent on the overall situation in terms of accidents, pollution, and transport performance such as passengers travel and waiting times. The results were discussed in a paper submitted to the prestigious RE19 conference.

5.2 Future Works

Future work on simulation in the EVA project will be directed towards the development and creation of other scenarios. These scenarios should include new critical situations (fog, different slopes, night . . .) with the aim to consider new systems functions. This implementation of the remaining functions as well as their verification, will help to consolidate the ARC-IT database [27] used in the project. The simulation of the scenarios will also supply the functional analyses as well as the performance and integrity analyses of the system.

BIBLIOGRAPHY

[1] Juliette Mickiewicz. *Trafic routier : ou passe t on le plus de temps dans les embouteillages?* 2017. URL: `http://www.lefigaro.fr/actualite-france/2017/02/20/01016-20170220ARTFIG00268-trafic-routier-o-passe-t-on-le-plus-de-temps-dans-les-embouteillages.php`.

[2] NCSL. *Autonomous Vehicles | Self-Driving Vehicles Enacted Legislation*. 2019. URL: `http://www.ncsl.org/research/transportation/autonomous-vehicles-self-driving-vehicles-enacted-legislation.aspx`.

[3] Daniel Fagnant and Kara Kockelman. "The travel and environmental implication of shared autonomous vehicles using agent-based model scenarios". In: *Transportation Research Part C: Emerging Technologies* 40 (Mar. 2014). DOI: `10.1016/j.trc.2013.12.001`.

[4] Monika Gope and MMA Hashem. "Knowledge Extraction from Bangla Documents using NLP: A Case Study". In: *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*. IEEE. 2019, pp. 1–5.

[5] Andrej Lúčny and DAI FMFI. "Advantages of multi-agent approach to building of monitoring systems". In: Informatics. 2007.

[6] Sabina Alazzawi et al. "Simulating the impact of shared, autonomous vehicles on urban mobility-a case study of Milan". In: *SUMO User Conference*. 2018.

[7] Mordechai Haklay and Patrick Weber. "Openstreetmap: User-generated street maps". In: *IEEE Pervasive Computing* 7.4 (2008), pp. 12–18.

[8] Pablo Alvarez Lopez et al. "Microscopic Traffic Simulation using SUMO". In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: `https://elib.dlr.de/124092/`.

[9] Jane C Bare. "TRACI: The tool for the reduction and assessment of chemical and other environmental impacts". In: *Journal of industrial ecology* 6.3-4 (2002), pp. 49–78.

[10] Jeff Craighead et al. "A survey of commercial & open source unmanned vehicle simulators". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE. 2007, pp. 852–857.

[11] Alexey Dosovitskiy et al. "CARLA: An open urban driving simulator". In: *arXiv preprint arXiv:1711.03938* (2017).

[12] Shital Shah et al. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[13] Olivier Michel. "Cyberbotics Ltd. Webots: professional mobile robot simulation". In: *International Journal of Advanced Robotic Systems* 1.1 (2004), p. 5.

[14] Nathan Koenig and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[15] Webots. *http://www.cyberbotics.com*. Ed. by Cyberbotics Ltd. Commercial Mobile Robot Simulation Software. URL: `http://www.cyberbotics.com`.

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

[17] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. "Model-free Deep Reinforcement Learning for Urban Autonomous Driving". In: *arXiv preprint arXiv:1904.09503* (2019).

[18] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[19] Leemon Baird. "Residual algorithms: Reinforcement learning with function approximation". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[20] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[21] Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine Learning* 8.3 (1992), pp. 293–321. ISSN: 1573-0565. DOI: `10.1007/BF00992699`. URL: `https://doi.org/10.1007/BF00992699`.

[22] SICK Sensor Intelligence. URL: `https://www.sick.com/tw/en/detection-and-ranging-solutions/2d-lidar-sensors/lms2xx/lms291-s05/p/p109849`.

[23] Spring Day. URL: `https://paris-saclay-spring.com`.

[24] JOSM editor. URL: `https://josm.openstreetmap.de/`.

[25] Albatrans. URL: `http://www.albatrans.net/les-lignes-les-horaires/`.

[26] Thomas A Dingus et al. "Human factors design issues for crash avoidance systems". In: *Human factors in intelligent transportation systems* 3 (1998).

[27] United States Department of transportation. *ARC-IT*. URL: `https://local.iteris.com/arc-it/index.html`.

[28] Polarsys. URL: `https://www.polarsys.org/capella`.

[29] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions". In: *arXiv preprint arXiv:1710.05941* (2017).

[30] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[31] Shixiang Gu et al. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3389–3396.

[32] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[33] Josef Krems and Tibor Petzoldt. “Tools and procedures for measuring safety-relevant criteria”. In: *The safety of intelligent driver support systems. Design, evaluation and social perspectives* (2011), pp. 93–109.

Appendix A

SCRIPTS OF THE ALGORITHMS

All the algorithms were developed starting from Webots code examples, adding improvements and functionalities.

```
def apply_PID(angle, targetAngle):
    """Apply the PID controller and return the angle command."""
    P = 0.15
    I = 0.006
    D = 10
    diff = angle - targetAngle
    if apply_PID.previousDiff is None:
        apply_PID.previousDiff = diff
    apply_PID.integral += diff
    input_angle = P * diff + I * apply_PID.integral + D * (diff - 
apply_PID.previousDiff)
    apply_PID.previousDiff = diff
    return input_angle
# PID parameters initialization
apply_PID.integral = 0
apply_PID.previousDiff = None
```

Listing A.1: Compute of the input angle

```
def process_lidar_data(lidar_data):
    HALF_AREA = 130
    sumx = 0
    collision_count = 0.0
    obstacle_dist = 0
    for x in range(widthFront / 2 - HALF_AREA, widthFront / 2 + HALF_AREA
):
        lidar_range = lidar_data[x]

        if lidar_range<15:
            sumx = sumx+x
            collision_count = collision_count + 1.0
            obstacle_dist = obstacle_dist+lidar_range

    if collision_count == 0:
        return None

    obstacle_dist = obstacle_dist/collision_count
    collision_count = collision_count
    obstacle_angle = ((sumx/(collision_count)/widthFront) - 0.50)*fovFront

    return obstacle_dist, obstacle_angle
```

Listing A.2: Process of the lidar data

```
def process_camera_image(image):
    num_pixels = camera.getWidth()*camera.getHeight()
    sumx = 0
    pixel_count = 0

    for x in range(0,camera.getWidth()):
        for y in range (0,camera.getHeight()):

            if  (abs(image[x][y][0]-203)<10 and abs(image[x][y][1]-187)<10 and abs(image[x][y][2]-95)<10):
                sumx = sumx + (x%camera.getWidth())
                pixel_count = pixel_count + 1.0

    if pixel_count == 0:
        return None

    line_angle = ((sumx/(pixel_count)/camera.getWidth()) - 0.50)*camera.getFov()


    return line_angle
```

Listing A.3: Process of the camera's image to detect the yellow line's angle

```
if distFront[1]<0:

                        if not (obstacle_front_right and obstacle_rear_right and obstacle_right) and distFront[0]!=0:
                            obstacle_steering = input_angle + (distFront[1] + 0.5) / distFront[0]
                            driver.setCruisingSpeed(10)
                            driver.setBrakeIntensity(0.7)
                            driver.setSteeringAngle(obstacle_steering)
                        elif not obstacle_rear:
                            s = driver.getCurrentSpeed()
                            breaking_distance = (s*s)/(250*0.8)
                            if distFront[0]<=breaking_distance:
                                driver.setCruisingSpeed(30)
                                driver.setBrakeIntensity(1)

                    if distFront[1]>0:

                        if not (obstacle_front_left and obstacle_rear_left and obstacle_left) and distFront[0]!=0:
                            obstacle_steering = input_angle - (distFront[1] + 0.5) / distFront[0]
                            driver.setCruisingSpeed(10)
                            driver.setBrakeIntensity(0.7)
                            driver.setSteeringAngle(obstacle_steering)
                        elif not obstacle_rear and distFront[0]!=0:
                            s = driver.getCurrentSpeed()
                            breaking_distance = (s*s)/(250*0.8)
                            if distFront[0]<=breaking_distance:
                                driver.setCruisingSpeed(30)
                                driver.setBrakeIntensity(1)


                    if distFront[1]==0:

                        if not (obstacle_front_left and obstacle_rear_left and obstacle_left) and distFront[0]!=0:
                            obstacle_steering = input_angle - (distFront[1] + 0.5) / distFront[0]
                            driver.setCruisingSpeed(10)
                            driver.setBrakeIntensity(0.7)
                            driver.setSteeringAngle(obstacle_steering)
                        elif not obstacle_rear:
                            s = driver.getCurrentSpeed()
                            breaking_distance = (s*s)/(250*0.8)
                            if distFront[0]<=breaking_distance:
                                driver.setCruisingSpeed(30)
                                driver.setBrakeIntensity(1)
```

Listing A.4: Avoidance of obstacles

```python
color_def = {0:'green',1:'yellow',2:'red'}
if trafficL:
            x = positionTL[0]
            y = positionTL[1]
            image1 = image[y-(sizeTL[0]):y+(sizeTL[0]), x-(sizeTL[1]):x+(sizeTL[1])]
            hsv = cv2.cvtColor(image1, cv2.COLOR_BGR2HSV)


            # green color mask
            lower = np.array([40, 50, 60])
            upper = np.array([70, 255, 255])
            green_mask = cv2.inRange(hsv, lower, upper)
            # yellow color mask
            lower = np.array([20, 100, 100])
            upper = np.array([30, 255, 255])
            yellow_mask = cv2.inRange(hsv, lower, upper)
            # red color mask
            lower_red = np.array([0,70,50])
            upper_red = np.array([10,255,255])
            mask0 = cv2.inRange(hsv, lower_red, upper_red)
            # red upper mask (170-180)
            lower_red = np.array([170,70,50])
            upper_red = np.array([180,255,255])
            mask1 = cv2.inRange(hsv, lower_red, upper_red)
            red_mask = mask0+mask1

            # combine the mask
            green_yellow_mask = cv2.bitwise_or(green_mask, yellow_mask)
            green_yellow_red_mask = cv2.bitwise_or(green_yellow_mask, red_mask)
            masked = cv2.bitwise_and(image1, image1, mask = green_yellow_red_mask)

            img_pred = cv2.addWeighted(masked,1.0,image1,0.2, 0)
            array_colors = [np.mean(green_mask),np.mean(yellow_mask),np.mean(red_mask)]
            color_detected = color_def[np.argmax(array_colors)]


            if not color_detected == 'green':
                s = driver.getCurrentSpeed()
                breaking_distance = (s*s)/(250*0.8)

                if trafficL_distance -8<=breaking_distance:
                    driver.setCruisingSpeed(0)
                    driver.setBrakeIntensity(1)
```

Listing A.5: Detection of the traffic light's color

```
objects = camera.getRecognitionObjects()
        if len(objects)>0:
            for object in objects:
                ped = 0
                model = object.get_model().decode()
                if model == 'traffic light':
                    trafficL_pos = object.get_position()
                    if trafficL_pos[0]>=0:
                        trafficL = True
                        trafficL_distance = math.sqrt(trafficL_pos[0]*
    trafficL_pos[0]+trafficL_pos[1]*trafficL_pos[1]+trafficL_pos[2]*
    trafficL_pos[2])
                        if trafficL_distance<old_TL:
                            old_TL = trafficL_distance
                            positionTL = object.get_position_on_image()
                            sizeTL = object.get_size_on_image()
                if model == 'pedestrian crossing':
                    Xped = True
                    Xped_pos = object.get_position()
                    Xped_distance = math.sqrt(Xped_pos[0]*Xped_pos[0]+
    Xped_pos[1]*Xped_pos[1]+Xped_pos[2]*Xped_pos[2])
                if model == 'pedestrian':
                    pedestrian =  True
                    ped = ped+1
                    if oldPed_pos == None:
                        ped_pos = object.get_position()
                        oldPed_pos = ped_pos
                    else:
                        oldPed_pos = ped_pos
                        ped_pos = object.get_position()
                    ped_distance = math.sqrt(ped_pos[2]*ped_pos[2])
                if model in CAR_MODEL:
                    car = True
```

Listing A.6: Objects recognition with the camera

```
node1 = self.getFromDef("WEBOTS_VEHICLE0")
```

```
            d = node1.getField("translation").getSFVec3f()
            node2 = self.getFromDef("WEBOTS_PED1")
            d1 = node2.getField("translation").getSFVec3f()
            dx =math.sqrt((d[2]-d1[2])*(d[2]-d1[2])+(d[0]-d1[0])*(d[0]-d1
    [0])+(d[1]-d1[1])*(d[1]-d1[1]))

            nodeTL = controller.getFromDef("TLS_137572230_1")
            TLstate = nodeTL.getField("recognitionColors").getMFColor(1)
            if dx<=16 and TLstate == [1.0,0.0,0.0]:

                    #The pedestrian crosses the road
```

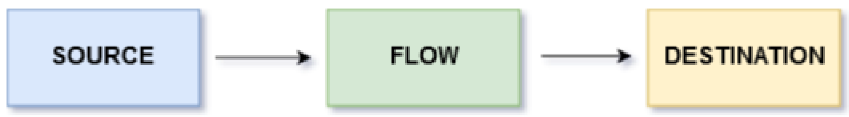
Listing A.7: Behavior of the pedestrians in the presence of the autonomous vehicle and the traffic light

Appendix B

INTRODUCTION TO ARC-IT

The architecture reference for cooperative and cooperative intelligent transportation (ARC-IT) is a common framework for planning, defining and integrating intelligent transportation systems. This reference architecture includes a set of interconnected components that are organized into four different views :

- **Enterprise View** is focused on the relationships between organizations and users.The enterprise uses enterprise objects to define responsabilites. For each physical object, in the physical view, and role combination there is an enterprise object.
  The enterprise view is focused on the relationship between enterprise objects.
- **Functional View** addresses the analysis of abstract elements and their logical interactions. Here ARC-IT is depicted as a set of processes(activities and functions) organized hierarchically. Here there is a data flows that move between logical processes in the functional flow. These data flows are related to information flows in the physical view, so the are related to the informations exchanged between physical objects that encapsulate related processes. Functional view defines processes to control and manage system behavior such as monitoring and other control elements.
- **Physical View** describes the transportation systems and the informations exchanged that support ITS. Here we have physical objects that interact and exchange informations to support the architecture service package. These physical objects are divided into **Subsystems** that are part of the overall intelligent transportation system and provide the functionality that is inside the boundary of ITS and **Terminators** that lie at the boundary they receive informations from ITS and supply function for the ITS. The informations exchanged between physical objects are organized in **Triples**.



- **Communication View** describes the protocols necessary to provide interoperability between physical objects. Each triple is mapped to a set of standards or published specifications that togheter can be used to build an interoperable implementation. The protocols are organized in a series of layers owing the necessary hierarchical nature of the relationships.

Appendix C

ARTICLE FOR RE19 CONFERENCE

Abstract

The focus of this paper consists in addressing requirements' quality issues while conceiving complex systems. Indeed, project stakeholders have to share the same problems' understanding allowing to undertake rational and optimal decisions. We propose an approach based on Natural Language Processing (NLP) techniques jointly with a Multi Agent System (MAS) method to improve systems' quality requirements such as consistency and completeness. We assessed the performance of our approaches through experimentations and highlighted feedbacks to projects' stakeholders and players.

Index Terms — System of systems, Requirements Elicitation and Validation, Natural Language Processing, Multi-Agent Systems.