POLITECNICO DI TORINO

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATIONS

MASTER'S DEGREE COURSE IN MECHATRONIC ENGINEERING

FINAL PROJECT WORK

A COMPLETE END-TO-END SIMULATION FLOW FOR AUTONOMOUS DRIVING FRAMEWORKS

Supervisor:

Prof. Massimo Violante

Supervisor (Teoresi):

Eng. Andrea Marchese

Candidate:

Salvatore Filippo Santonato

245451

Academic Year 2019/2020

In partnership with:

Teoresi S.p.A.

Abstract

The research on automotive industry in the last year focused on three main aspects: electric engine, advanced infotainment and autonomous driving.

Nowadays autonomous driving represents one of the hottest topics in the automotive field, since technology advancement in these last years, especially with the new discovery on Artificial Intelligence, allowed to lay the foundations for the development of fully autonomous vehicles and revolutionize the concept of autonomous mobility.

Since building a real vehicle equipped with the needed sensors is very expensive, it is critical to test every component, both hardware and software, considering all the possible scenarios where the vehicle may operate. It is clear how Software and Hardware in the Loop play a key role for the development, allowing a preliminary validation of the vehicle before road testing. However, this approach, to be reliable, requires a virtual environment that mimics the conditions of the real world. The goal of this thesis is to create an automated system for the generation of three-dimensional maps starting from simple 2D maps. The building of a structured simulation ready map, a detailed map containing information of the environment, starting from OpenStreetMap data is the key for reduce costs and time for fast prototyping of this new generation of algorithms. The map generation is accomplished by using CARLA simulator, an Unreal Engine-based simulator. After this, the map is converted into a specific format suitable for an eventual field test by using a bridge between CARLA and the ROS-based platform Autoware. The last part is dedicated to the evaluation of the achieved result considering possible limitations and improvements.

Acknowledgements

First of all, I would like to thank my parents and my brothers, I am really grateful for their support in all these years. A special thought goes to my late grandmother and all the happy memories with her, which I will cherish forever.

Lastly, I thank all my friends and colleagues, who helped me to grow as a person.

Contents

List of Figures

List of Tables

Chapter I

1 Introduction

An autonomous car, as the name suggests, is a vehicle capable of operating without the human action by making use of several sensors tasked with sensing its environment to safely drive. While sensors are essential for the correct vehicle operation, they are not the only important components. A combination of hardware and software components is required to ensure the self-driving capability. The entire process behind autonomous driving can be summarized by the below scheme, as shown in Figure 1.1, which shows the fundamental components in an autonomous vehicle (AV).

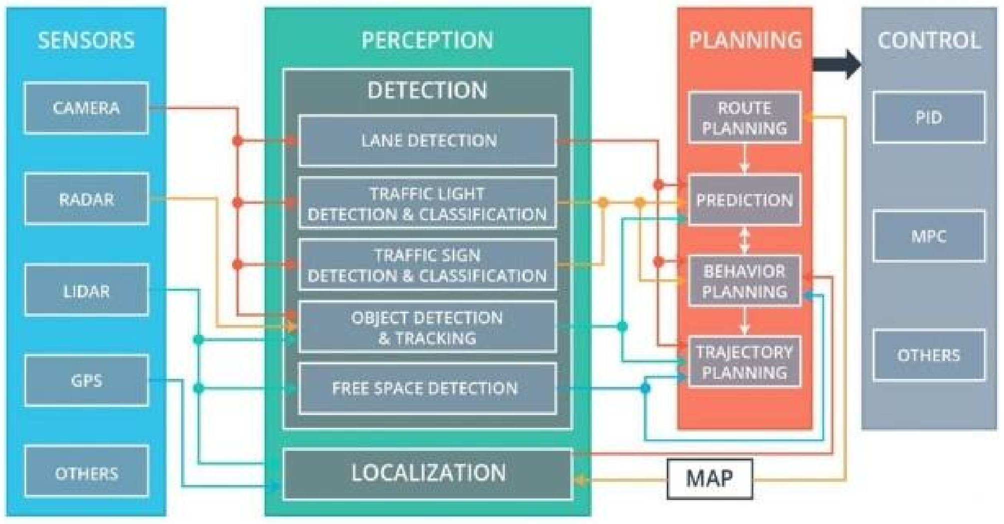


Figure 1.1 Typical autonomous vehicle system overview

More specifically:

- **Sensors:** they represent the eye of the vehicle and collect information about the environment. Sensor fusion of the data from different sources allows the vehicle to perceive the environment more accurately.
- **Perceptions:** the vehicle analyses the data received by the sensors to detect and classify the objects around itself and also to locate its position.

- **Planning:** this is the process of planning the route of the vehicle. The decision is made considering all the information about the environment and the road in order to choose the best path.
- **Control:** the combination of actions needed to follow the path previously planned. These actions regard the control over the steering angle and the acceleration, or the brake amount issued by the planning process.

AV technology has the potential to improve daily life and not only but the presence of all these components, both hardware and software, requires a high level of reliability as prerequisite since safety has the top priority. Industry is still far away from achieving fully autonomous driving but in the meantime its development is proceeding rapidly thanks to the number of goals reached by technology in these last years. All companies involved in autonomous driving development are facing several challenges related to many aspects and in doing so their vehicles are undergoing testing to collect essential data and improve their systems. Data represent indeed a powerful resource for AV development and particularly every information regarding the environment is useful to build an infrastructure able to help the vehicle by communicating with the last one which will know the status of the surrounding world. For this purpose, in these last years a new generation of maps built for AVs is gaining foothold in the industry. These so-called HD maps bring with them high road accuracy and more information of the surroundings supporting AV navigation. From the previous considerations, it is clear why running tests and gathering data is essential.

This thesis work aims to develop a methodology to build a simulation environment for testing purpose by means of real geodata and simulation software. In the next chapter the state of art of simulation is discussed, considering the most important simulators available in this field. This is followed by an overview of the HD map features. Then, in the Methodology chapter, the workflow to obtain such simulation environment is proposed and discussed.

Chapter II

2 Simulation for autonomous driving

2.1 The importance of simulation

Nowadays, the automotive industry is offering models of car equipped with Advanced driver-assistance systems (ADAS), electronic systems capable of assisting the driver in many situations but still, they are not sufficient to offer full self-driving. Despite the continuous technological progress and the development of the infrastructure to support the testing, the maturity necessary for the landing these technologies in the current productions has not yet been reached. Reliability matters above all when safety is considered the top priority for drivers who have to rely on intelligence inside a car. It is generally assumed that no fully autonomous vehicle has already hit the market because the desired level of safety is still not guaranteed. The big question that every companies involved in AV development is asking is: "How safe if safe enough?". Answering this is not simple or simplified. The only way to measure safety for the companies is to continuously test their vehicles gathering useful data to prove the vehicle reliability. According to a report of Rand Automotive, Autonomous vehicles would have to be driven hundreds of billions of miles to demonstrate their reliability [1]. The objective is to obtain a vehicle able to drive at least as good as a human driver. Testing on real road is fundamental, especially in real life scenarios to study how the vehicle act and to locate eventual issues. The problem of road testing is that it requires a lot of resources and money. Making a vehicle can be very expensive due the presence of many components, particularly the ones used by ADAS. For this reason, many companies are investing their time and effort in building simulators to be used for testing. Virtual testing allows to evaluate the vehicle behaviour in a virtual environment where every sort of scenario can be simulated. This is a great opportunity for carmakers to improve their systems making the vehicle safer before road testing but also to shorten the time of development. Besides being a cost-effective solution, a simulator has the advantage to give full control over the simulated world. This is extremely important since in real world tests it is nearly impossible to cover all kinds of possible scenarios, particularly those that are less likely to happen. Hence, it is not simply the number of miles that matters but the overall quality of the data obtained by simulating and this is even more true when it comes to AI training. For these reasons, simulation represents a linchpin for the autonomous driving development.

2.2 Features and applications of Autonomous Driving Simulators

Simulators can be a valuable source for data gathering and development but to be useful they must offer realistic simulations. Other than an accurate dynamic model of the vehicle, a good simulator requires several important features, such as support for different sensors or traffic handling and so on, in order to ensure high-fidelity simulation environments. The number of these features depends on the functionality to be tested but in general, if the aim is to test the full solution stack, a simulator must include all the needed features and models with high accuracy to reproduce a real world scenario. “Fidelity” represents an important concept in AV simulation since it can tell how well a simulation matches the real world. While for vehicle dynamics modelling it is easy to achieve a high-fidelity level, for environment representation it is a different matter. Environment model contains information about roads, surrounding infrastructures and actors. It must be highly detailed, trying to recreate the realism of a possible scenario, in particular the behaviour of possible actors interacting with the vehicle. Detailed maps, such as HD maps, can be of great help for the environment realization. Also, the physics used in the simulation plays a key role since all the dynamic elements of the virtual world depend on it. These last ones have a great impact on the vehicle perception system. The vehicle knows how to act only if it is capable to sense its surroundings correctly. Perception requires a high computational effort since it has to process a large amount of data and this is due in part to the high level of details of the represented environment. The previous features can be considered as the foundations of a simulator, but they are not the only features. From the point of view of the user, a simulator must be easy to use and offer a good level of controllability. This allows the users to test several aspects in the simulation by simply setting values in the scenario. Another interesting feature is flexibility since it is very useful interfacing the simulator with third party software or hardware elements. This is the case of Software in the Loop (SIL) and Hardware in the Loop (HIL). These two methods of simulation give the possibility to test and validate components which will be implemented in the final platform. The more features are present in a simulator the more applications can find, but the choice of a specific simulator depends mainly on the purpose of the test. In fact, in the industry are available all kinds of simulators, some of them are suitable to test a particular functionality, as can be traffic management or path planning, while other are more advanced and allow to test every elements related to the autonomous driving, from simple algorithm testing to test and validation of software and hardware components. Hence, defining the applications of a specific simulator without considering the testing purpose makes little sense.

2.3 State of the Art in Autonomous Driving Simulators

Leading companies in autonomous driving have already developed their own simulating platform for a few years thanks to their resources but now that technology has become accessible to everyone, many minor companies or even research groups are working on their own solution. Not all of these simulators are available on the market, but instead they are only utilised for internal research and this is in general the case of big companies. On the other hand, many simulators are available as open-source software, thus allowing the democratization of autonomous driving research and development [2]. A special mention goes to the video game industry which offers the possibility to use game engine technology for the creation of photorealistic environments.

The rest of this section is dedicated to an overview of the most common and advanced simulators or platforms available in the industry, also taking into consideration the aim of the case study of this thesis work.

2.3.1 Uber platform

At Uber the Advanced Technologies Group (ATG) with the help of the Data Visualization Team is working on its platform focusing on the interpretation of the collected data [3]. The team is building this platform by designing it as a browser-based service (Figure 2.1). Thanks to the data collected from the real world, the simulator recreates the roads with high precision which can be used for future simulations.

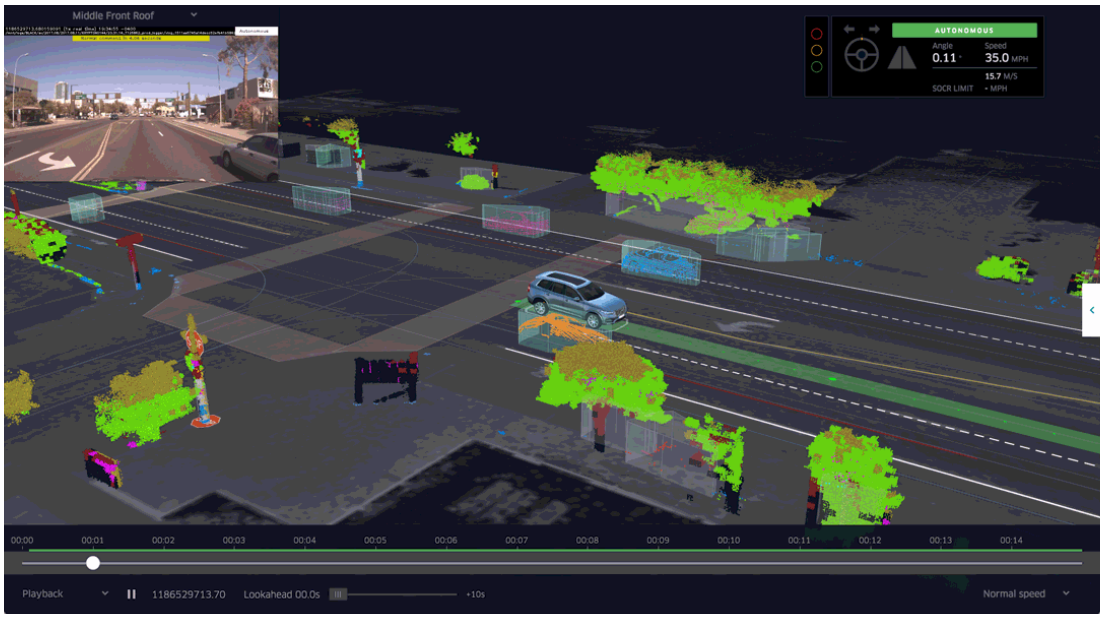


Figure 2.1 An example of a pre-processed 3D map

Perception is fundamental to properly predict the position of all the objects the vehicle can came across. This information allows to compute the motion planning algorithms. Besides, last year Uber made available an open-source version of its Autonomous Visualization System [4], hoping to create a standard for engineers and developers who can share data.

2.3.2 Cruise

At the beginning of the year, Cruise [5], a subsidiary company of General Motors, unveiled its new electric vehicle called Origin, devoid of manual controls. After the announcement, the company began to share new information about the progress achieved. Like other competitors, Cruise has invested a lot of its time to gather data by driving many miles. To further improve the quality of the data, the company relies on 3D simulation. The team developed an end-to-end simulator based on the Unreal Engine [6] (Figure 2.2), which the employees call "The Matrix", capable to recreate high detailed cities [7].

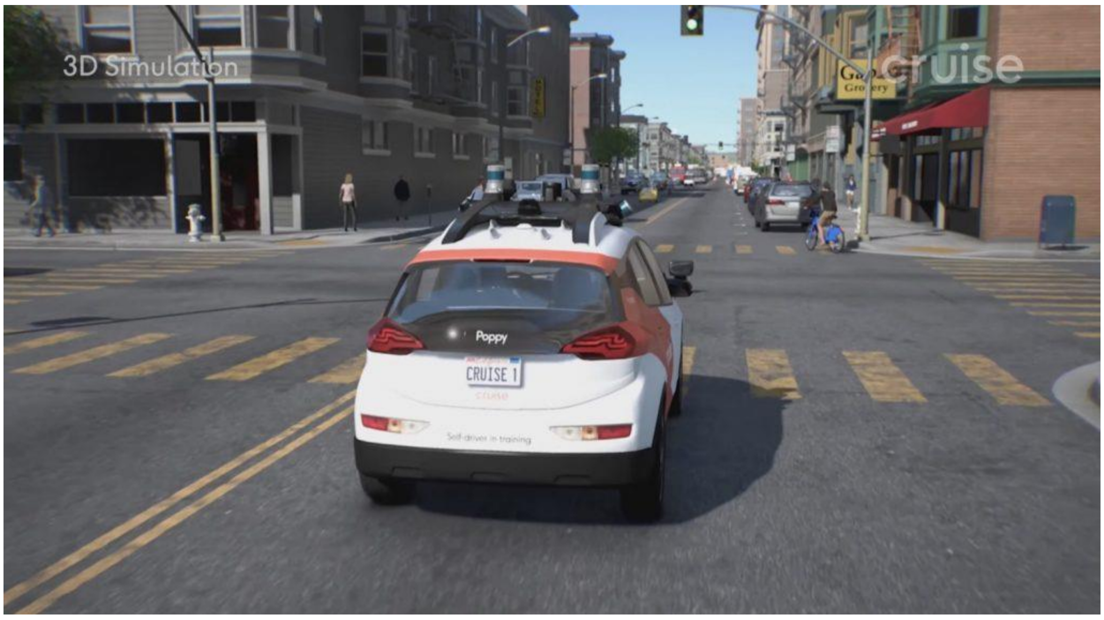


Figure 2.2 The Matrix: a glimpse of the simulation

Moreover, Cruise has made available its open source visualization tool Worldview [8], a web browser-based 2D and 3D scene renderer (Figure 2.3).This tool allows to visualize, explore, and analyse data collected in the simulation and on the road.

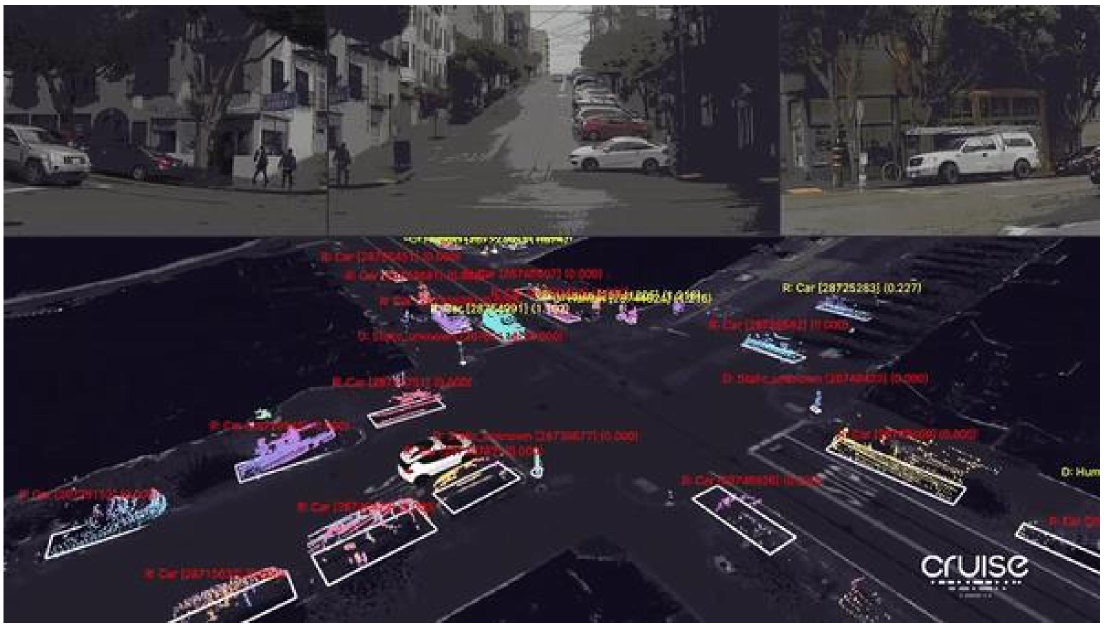


Figure 2.3 An example of a scene on Worldview

2.3.3 Carcraft (Waymo)

Waymo, a subsidiary owned by Alphabet Inc., developed its own simulation platform called Carcraft (Figure 2.4). Thanks to the simulator, all the virtual AV vehicles drive billion miles per day [9].

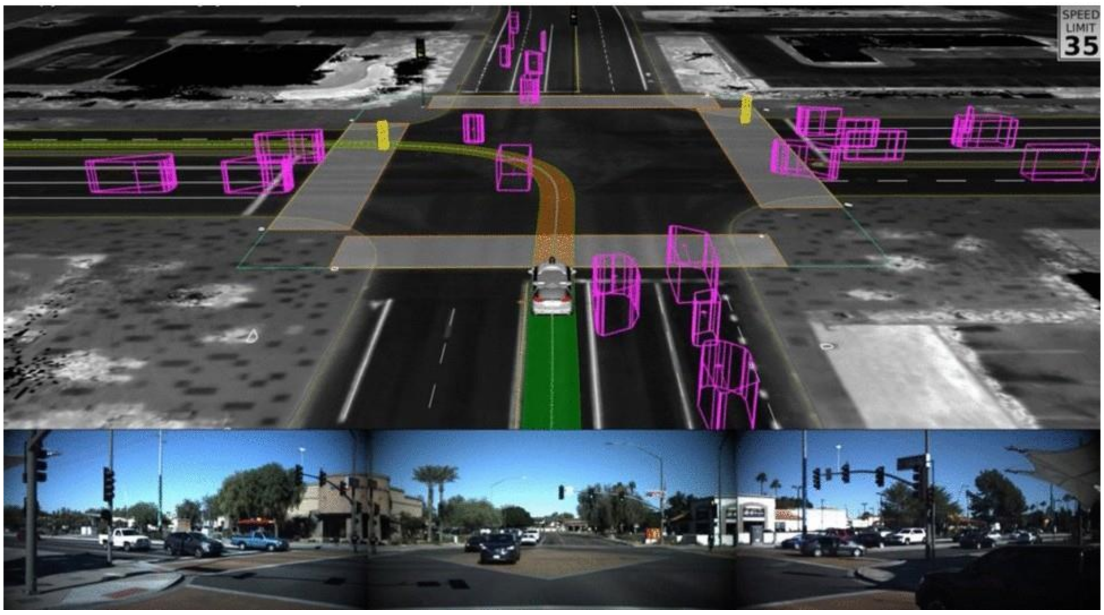


Figure 2.4 A scene of Carcraft

When Google unveiled Carcraft, a real scene, which really happened in the presence of a roundabout, was recreated to show off the potential of the simulator. It has been stated that the simulator has the capability to employ a fleet of thousands of virtual vehicles. Moreover, Waymo has recently begun to make use of AI to simulate AV sensor data. By using a SurfelGAN network, it is possible to recreate realistic camera images through simulation and data [10].

2.3.4 NVIDIA DRIVE Constellation

Nvidia, one of the leading companies in GPU manufacturing, has developed its AV platform called NVIDIA DRIVE [11]. This platform provides both hardware and software solutions, including DRIVE Constellation, an advanced driving simulation software stack. DRIVE Constellation is a Hardware in the loop simulation-based platform which allows to develop and validate autonomous vehicles through virtual test. The platform consists of three main components:

- **DRIVE Sim**: it is the 3D simulation software which recreate a realistic virtual environment.
- **DRIVE Constellation Simulator**: it is a powerful GPU server which runs DRIVE Sim generating data to send to the AV hardware
- **DRIVE Constellation Vehicle**: it is the processing system, hosting AV hardware and software, which receives the data generated in the simulation.

The simulation provides a high level of fidelity to recreate realistic scenarios. In order to reach such level, Nvidia DRIVE platform combines photorealistic physics and HD Maps as shown in Figure 2.5.

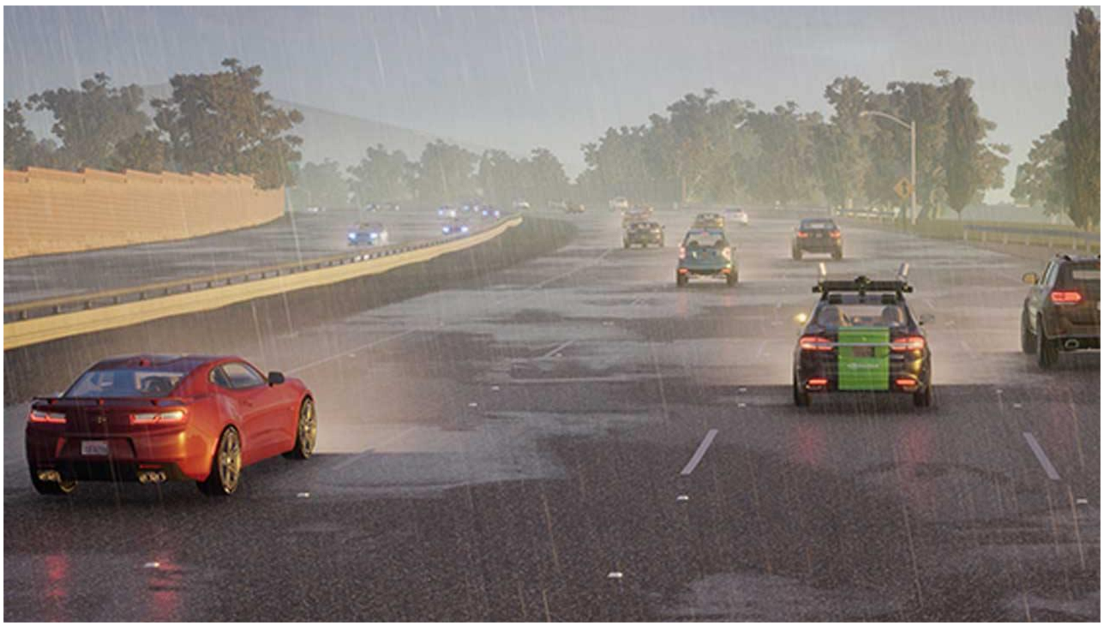


Figure 2.5 an example of Nvidia DRIVE Sim

2.3.5 rFpro

rFpro [12] provides a simulation platform which allows to test and validate vehicles in photorealistic environments (Figure 2.6). The company offers LIDAR-scanned road surfaces and supports the HD map format to build detailed environments.

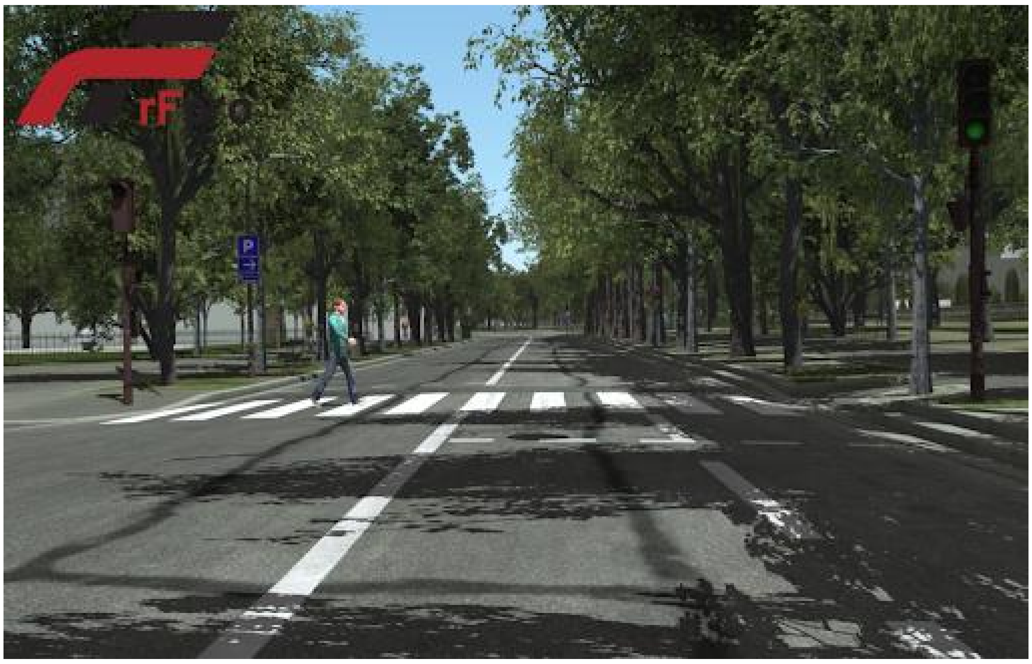


Figure 2.6 Model of a photorealistic scanned road

Another feature of the platform is its support to Simulink® [13] which allows communication between Simulink models and external applications.

The high level of accuracy offered by the platform has meant that many top automotive OEMs have adopted its solutions.

2.3.6 Cognata simulator

Cognata [14], an Israeli start-up company, has developed a simulation platform which leverages real world map data to recreate accurate and realistic virtual roads, as shown in Figure 2.7. The idea of the company is to speed up testing and validation by building 3D city models as detailed as possible. According to Dan Atsmon, Cognata CEO, the result is achieved in three steps [15]:

- a first layer is built, containing the assets of every static element, such as the buildings, the roads, the traffic sign or even the trees.
- A second layer is built, containing all the dynamic information and models such as the traffic behaviour of vehicles and pedestrians and the weather conditions previously recorded in the real world.
- Finally, a third layer is dedicated to the simulation of all the sensor interactions with the surroundings.

Additionally, the company makes use of AI and deep learning to further improve fidelity of the generated virtual environments.

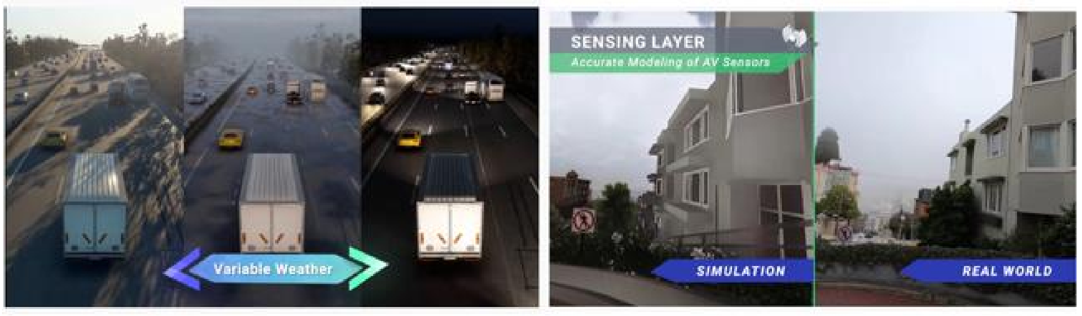


Figure 2.7 Cognata simulator:on the left is shown the graphic quality while on the right a comparison between simulation and reality

2.3.7 Metamoto simulator

Metamatoto [16] provides its simulation platform as Service to train, test, debug, and validate autonomous vehicles. The platform comprises three main components, as shown in Figure 2.8:

- **Designer**: a development tool to build scenarios and configure vehicle
- **Director**: the core of the platform, which schedules and runs simulations for every scenario.
- **Analyzer**: an autonomous system software debugging tool to assess how a vehicle acted in a specific scenario.

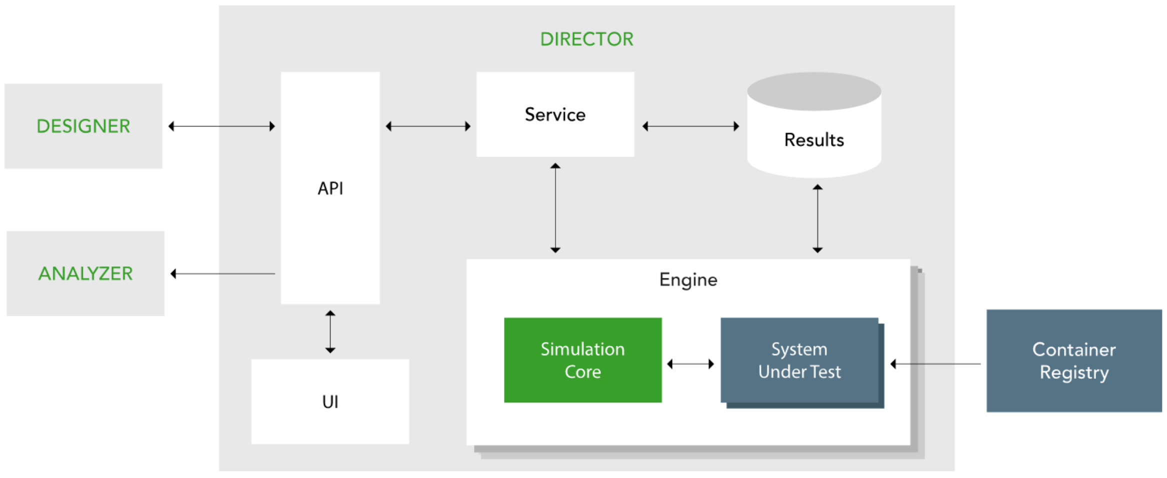


Figure 2.8 The platform structure

The company goal is to build a platform that is scalable and highly parameterizable. In addition, the simulation is characterized by a realistic physics and along with the possibility to support HD maps, a high level of fidelity can be ensured.

2.3.8 Vires Virtual Test Drive

Virtual Test Drive [17] is a platform which provides the possibility to create, configure, simulate and customize virtual environments for AV development. Figure 2.9 shows an overview of different scenarios generated in the simulator.

The platform provides a Road Network Editor which allows to design complex road and rail networks with the addition of the OpenDRIVE [18] support for the network logic. The open and modular design of the platform offers the possibility to interface the simulation with external applications to enable X in the loop (where X could be: Software, Vehicle, Hardware etc.).

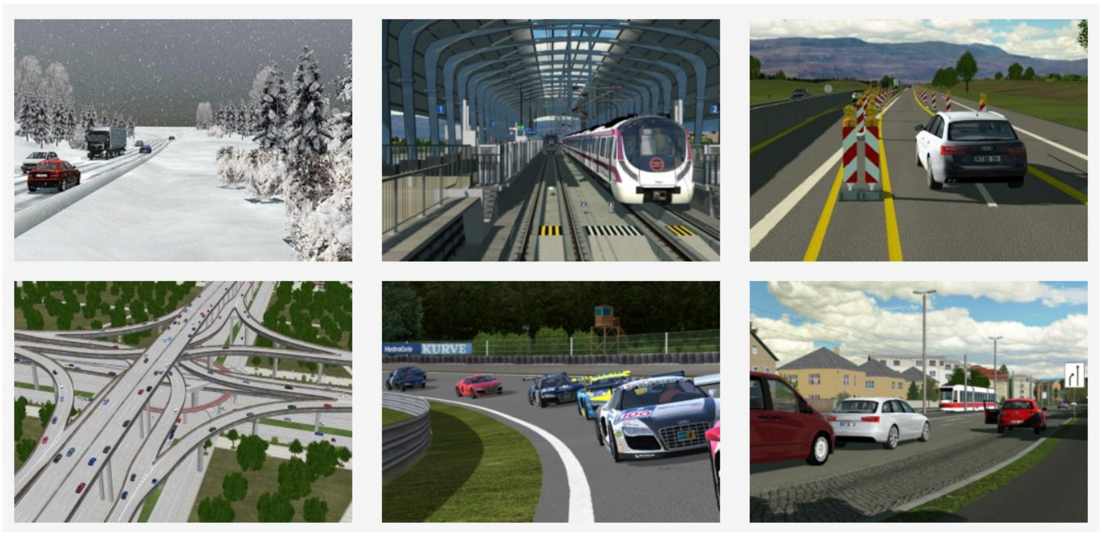


Figure 2.9 Vires VTD simulation

2.3.9 AutonoVi-Sim

AutonoVi-Sim is presented as a novel high-fidelity simulation platform for autonomous driving data generation and driving strategy testing by the development team [19]. The platform supports multiple vehicles and non-vehicle traffic participants, and weather control along with support of several sensors allow to recreate all kinds of scenarios useful for data generation. The modular structure of the platform, as shown in Figure 2.10, facilitates the construction of advanced scenarios by enabling the configuration of every aspect of the simulation.

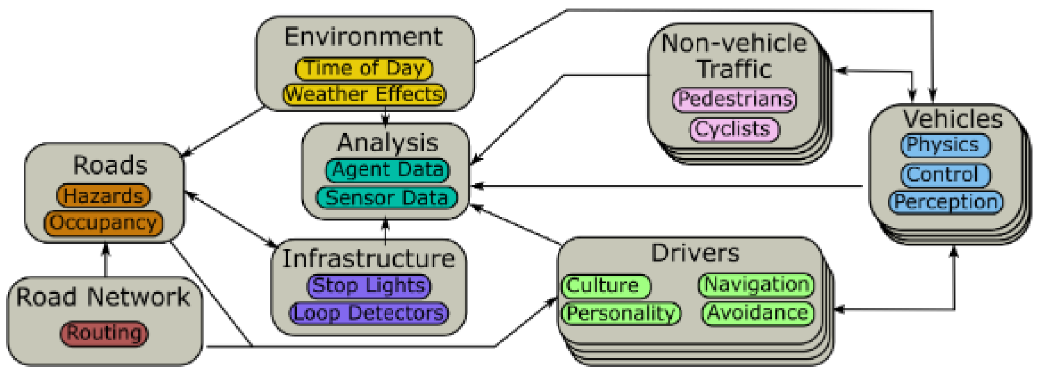


Figure 2.10 AutonoVi-Sim Platform Overview

The simulation platform is still in development and has some limitations, but it is promising, especially for its modular nature.

2.3.10 IPG CarMaker

CarMaker [20] is a virtual test-driving platform (Figure 2.11) developed specifically for testing passenger cars and light-duty vehicles. The platform provides the possibility to recreate realistic scenarios thanks to the accurate representation of the vehicle dynamics and traffic models. The road generation tool allows to import real road networks from external sources such as HERE maps [21] and supports multiple standard file formats including OpenDRIVE.

Figure 2.11 a scene of the CarMaker simulation

2.3.11 PreScan

PreScan [22] is a physics-based simulation platform (Figure 2.12) used for developing and testing ADAS with the possibility to integrate XIL and third-party applications. The platform also supports Vehicle-to-everything (V2X) communication applications. Scenarios can be quickly built by using real information from external sources such as OpenStreetMap, Google Earth and GPS navigation device.

Figure 2.12 an example of a PreScan simulation

2.3.12 VRXPERIENCE

VRXPERIENCE [23] of ANSYS is a driving simulator used to create virtual scenarios powered by SCANeR [24]. Driving scenario generation support HD map importing to recreate high-fidelity environments. The level of realism is further improved by the possibility to simulate lighting systems and sound sources. Another feature is the presence of a tool dedicated to the Human-Machine-Interface (HMI) assessment. Moreover, it is possible to interface the platform with third-party applications and run XIL tests. In Figure 2.13 are shown different scenes taken from the simulation.

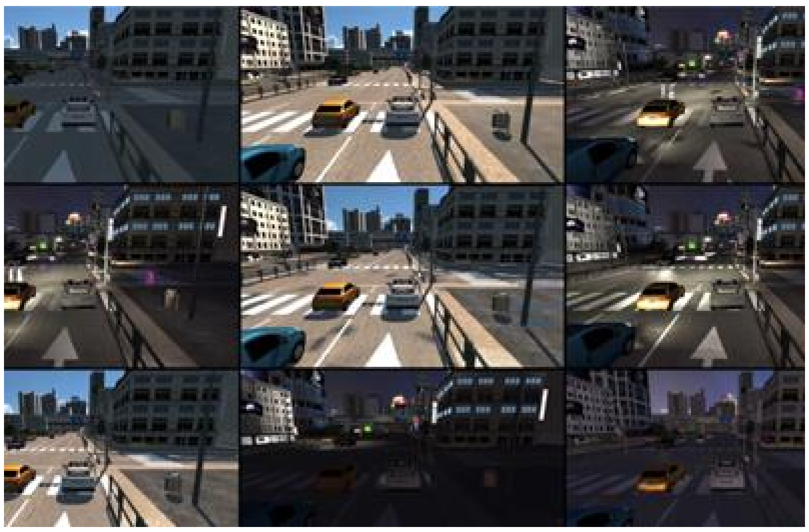


Figure 2.13 several examples of simulations through VRXPERIENCE

2.3.13 Deepdrive

Deepdrive [25] is an open-source self-driving car simulator which supports deep reinforcement learning and Unreal API integrated with python. The goal of Voyage, the company behind this simulator, is to facilitate research and development through machine learning and with regard to this, the company has decided to launch a leaderboard where anyone can submit their own agent to see how it stacks up against others. Regarding the virtual environments available, the simulator features realistic 3D worlds built by Parallel Domain

[26], a platform for automated generation of virtual worlds. Figure 2.14 shows a snapshot taken from the simulator.

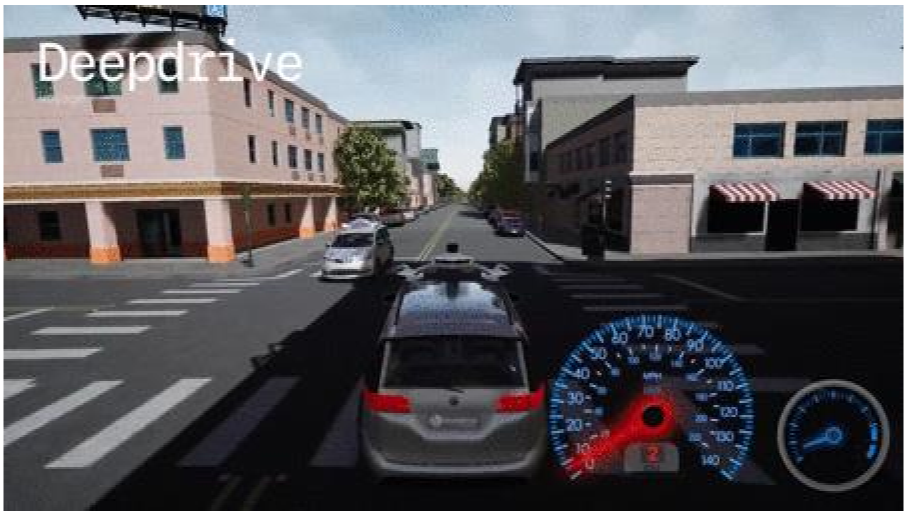


Figure 2.14 a snapshot from Deepdrive

2.3.14 Udacity Simulator

Udacity has developed its self-driving car simulator [27] for research and teaching purposes and made the code available on GitHub as open source software. The simulator is built on Unity [28] engine but it does not offer a photorealistic graphics, as compensation, however, it has a user-friendly interface. In the main screen it is possible to choose a track and a mode as shown in Figure 2.15.

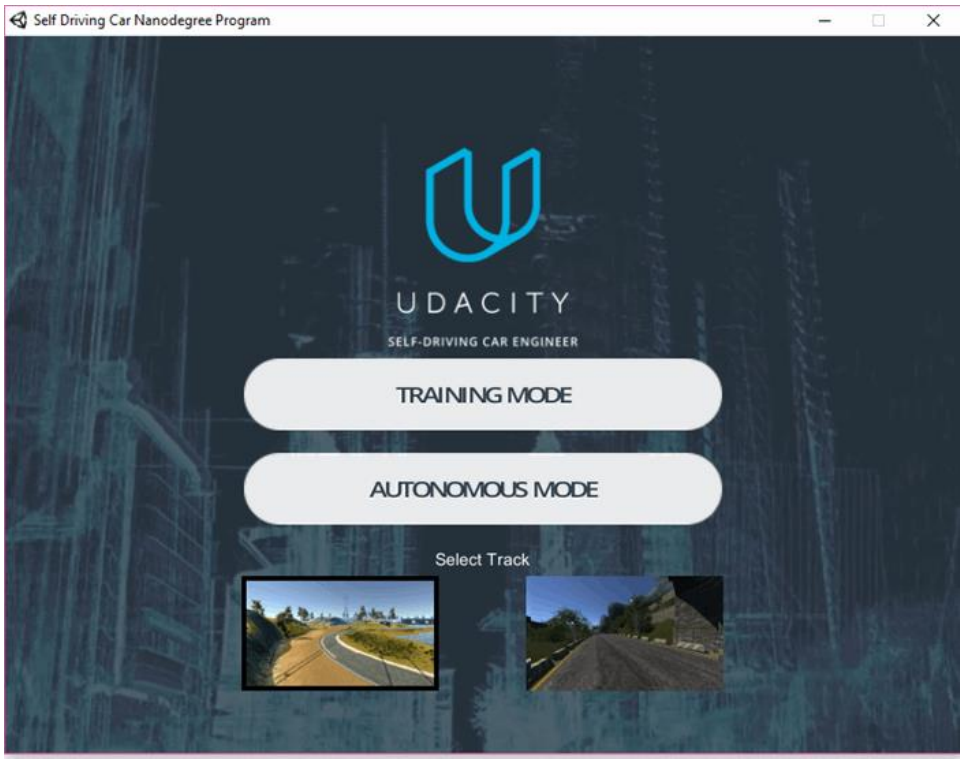

Self Driving Car Nanodegree Program
UDACITY
SELF-DRIVING CAR ENGINEER
TRAINING MODE
AUTONOMOUS MODE
Select Track

Figure 2.15 Simulator main screen

The training mode gives the possibility to record the driving behaviour, useful for training purpose. The autonomous mode can be used to test machine learning models. Through Unity editor it is possible to modify existing scenes. Figure 2.16 shows a snapshot taken from the simulator.

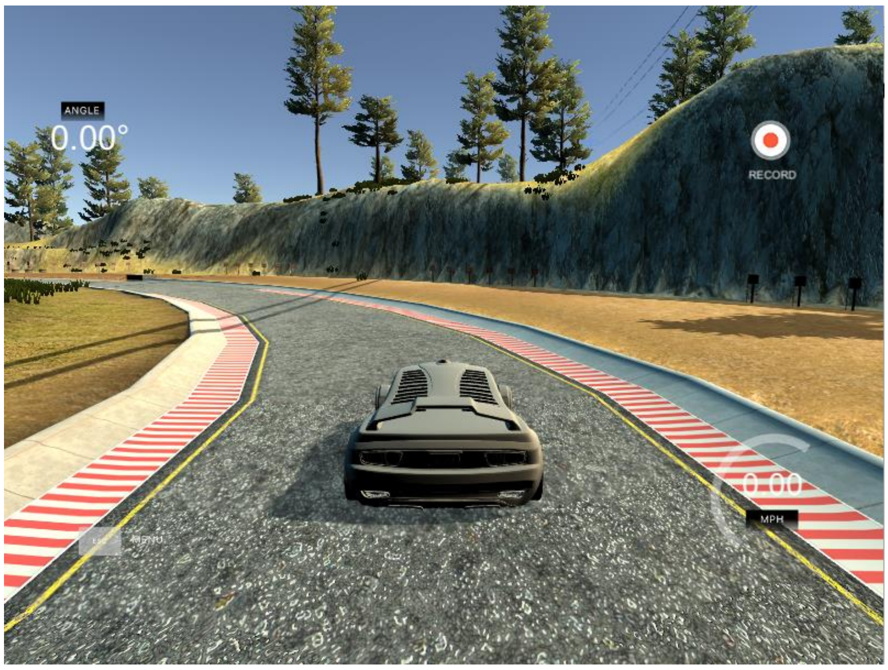


Figure 2.16 a snapshot from the simulator

2.3.15 AirSim

AirSim [29] is an open-source simulator built on Unreal Engine (an experimental release based on Unity is available) for drones, cars and other vehicles (Figure 2.17). The main goal of this platform is to support research by developing and testing deep learning and reinforcement learning algorithms [30]. The simulator features high-fidelity photorealism, in particular thanks to the advanced physics, and supports HIL testing and external peripherals, such as a flight controller, to improve simulation results.

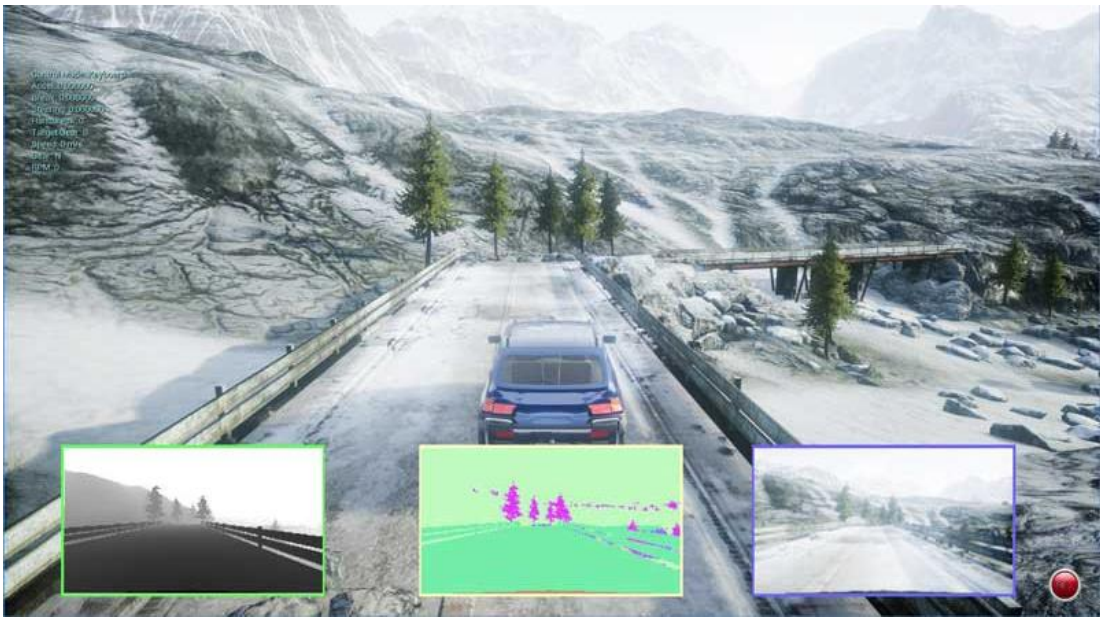


Figure 2.17 a snapshot from AirSim

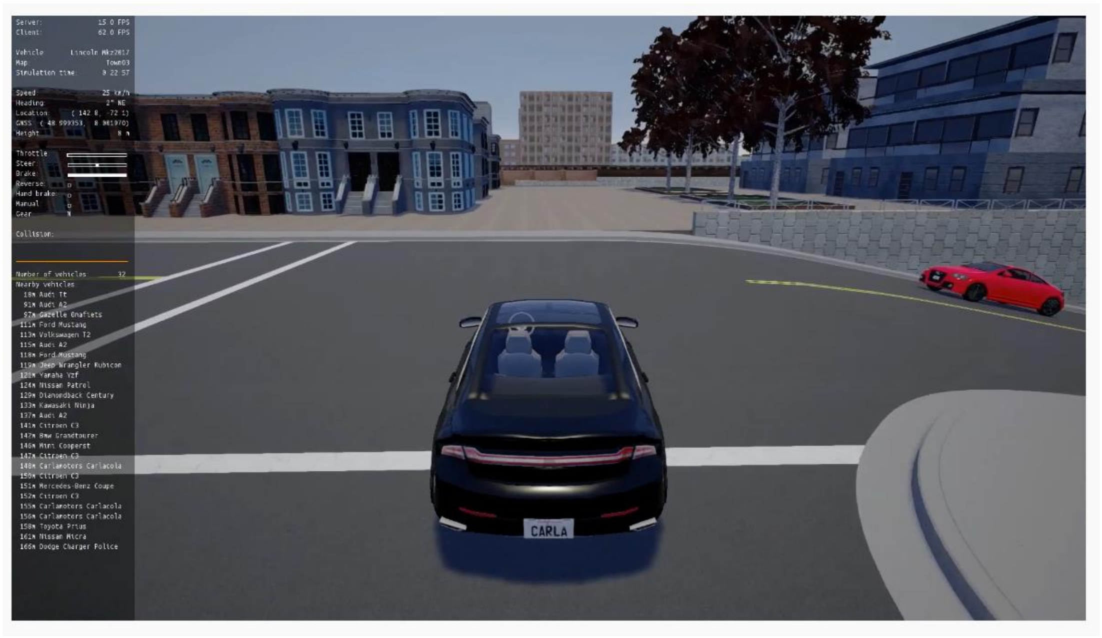
AirSim currently supports the following sensors for data gathering:

- Camera
- Barometer
- Imu
- Gps
- Magnetometer
- Distance Sensor
- Lidar

It is available a detailed 3D urban environment which can be customized in the Unreal Engine editor, but it is also possible to import new environments.

2.3.16 CARLA simulator

CARLA is an open-source simulator developed for autonomous driving research [31]. The simulator is built on Unreal Engine to provide high-fidelity simulations. In Figure 2.18 is shown a scene of CARLA.



```
Server:                15.0 FPS
Client:                62.0 FPS

Vehicle:      Lincoln Mkz2017
Map:                   Town03
Simulation time:      0:22:57

Speed:                25 km/h
Heading:                 2° NE
Location:       ( 142.8, -72.1)
GNSS:  ( 48.999353,  8.001970)
Height:                   8 m

Throttle:
Steer:
Brake:
Reverse:     □
Hand brake:  □
Manual:      □
Gear:        N

Collision:

Number of vehicles:        32
Nearby vehicles:
  18m Audi Tt
  91m Audi A2
  97m Gazelle Omafiets
 111m Ford Mustang
 113m Volkswagen T2
 115m Audi A2
 118m Ford Mustang
 119m Jeep Wrangler Rubicon
 121m Yamaha Yzf
 124m Nissan Patrol
 129m Diamondback Century
 133m Kawasaki Ninja
 137m Audi A2
 141m Citroen C3
 142m Bmw Grandtourer
 146m Mini Cooperst
 147m Citroen C3
 148m Carlamotors Carlacola
 150m Citroen C3
 151m Mercedes-Benz Coupe
 152m Citroen C3
 155m Carlamotors Carlacola
 156m Carlamotors Carlacola
 158m Toyota Prius
 161m Nissan Micra
 166m Dodge Charger Police
```

Figure 2.18 a snapshot from CARLA

It is possible to recreate complex and realistic scenarios thanks to traffic and agent management. Data gathering can rely on the presence of several kinds of sensors and cameras. CARLA offers flexibility and gives good control over the simulator through Python API. As stated by the team, the simulator can be used to evaluate three approaches to autonomous driving [31]:

- **Classic modular pipeline**: this approach focuses on dedicated subsystems for perception, planning and control.
- **Imitation learning**: this approach utilizes a dataset of human driving records to train a deep network.
- **Reinforcement learning**: this method trains a deep network to complete a task, based on a reward signal, within an environment.

Roads and urban information are defined according to the OpenDRIVE standard. The simulator is still in development and currently the latest released version is 0.9.9 [32], which added support to several third-party software.

2.3.17 LGSVL Simulator

LGSVL simulator is a high-fidelity simulation platform built on Unity engine (Figure 2.19) and its source code is freely available on GitHub [33], [34]. The platform is designed to be open and interfaceable with Autonomous Driving stacks with support of SIL and HIL testing [34]. Users takes control over the simulator by employing Python API. A default set of sensors is available, but it is possible to implement custom sensors as plugins. 3D environments support HD map annotations to handle agents in the simulation.

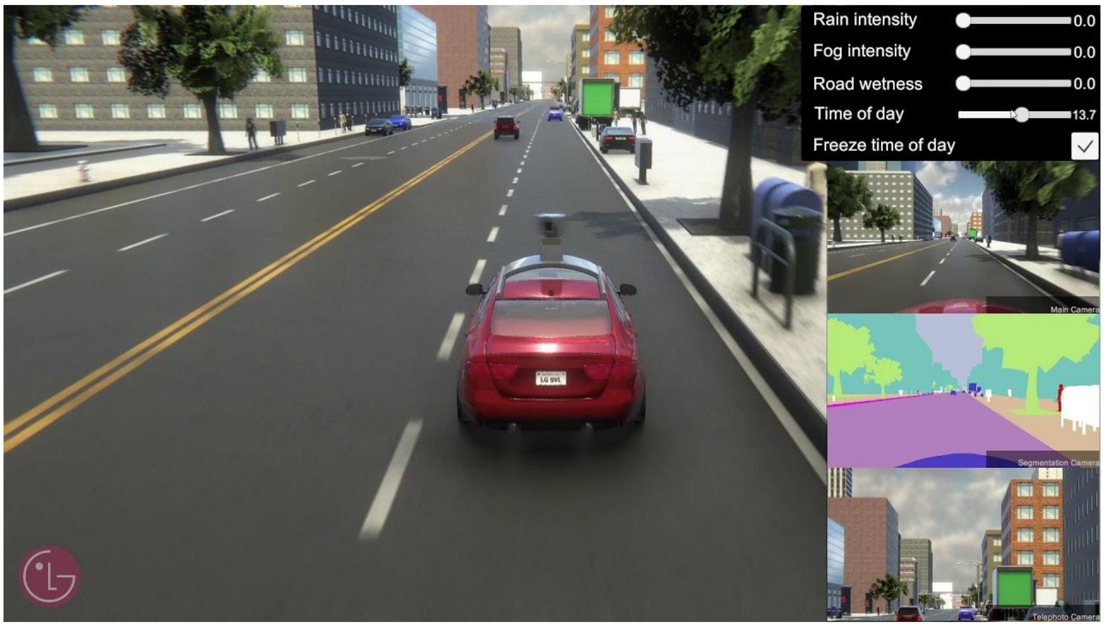

Rain intensity 0.0
Fog intensity 0.0
Road wetness 0.0
Time of day 13.7
Freeze time of day ✓

Figure 2.19 a snapshot from LGSVL

Regarding applications, the simulator allows to test machine learning and reinforcement algorithms as well as V2X algorithms.

2.3.18 Sim4CV

Sim4CV is a photo-realistic simulator (Figure 2.20) built on Unreal Engine used to develop and test autonomous driving for cars and autonomous flying for aerial vehicles [35].

Figure 2.20 a snapshot from Sim4CV

Multiple sensors can be set up to collect data which can be used to train vehicles by exploiting a deep neural network. This neural network is then employed to predict waypoints ahead the vehicle and build control algorithms. Virtual environments are built by means of an external software tool starting from a 2D map with overhead view.

2.4 HD maps

Nowadays High-Definition maps are widely used in the autonomous driving industry and are now considered essential in ensuring safety on the road. HD maps can be seen as the necessary evolution of the classic maps and they fill the void left by the lack of accuracy of the GPS technology. Considering a width of 10 cm for lane markings, it is clear that autonomous vehicles need centimeter level accuracy to avoid collisions by wrongly assessing their position [36]. But high precision is not the only feature of HD maps (Figure 2.21). In fact, these maps provide rich geometric information as well as semantic data about road network and surroundings [37].

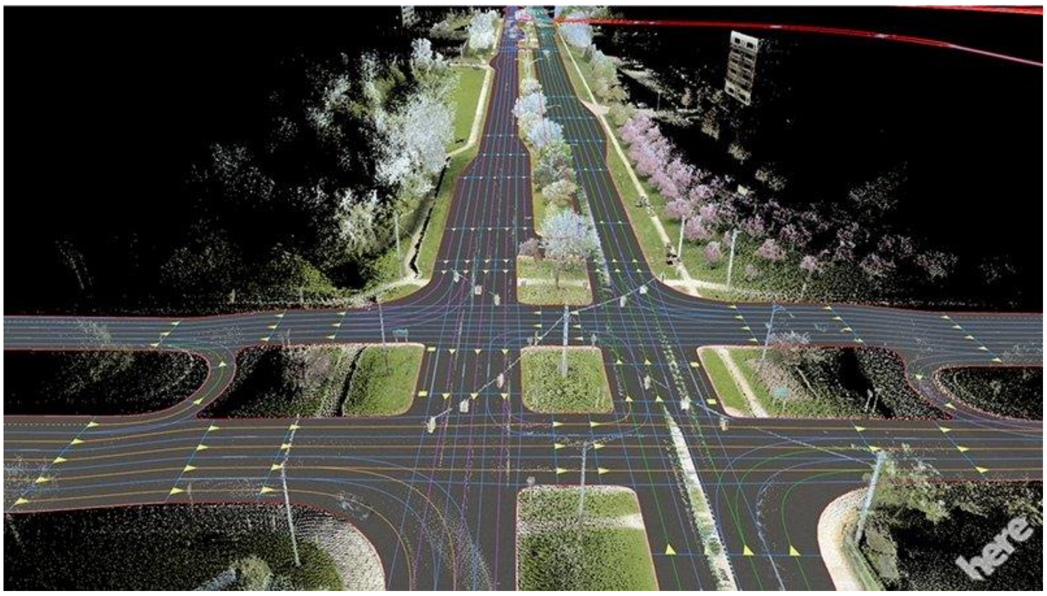


Figure 2.21 an example of how a HD map appears

Having a priori information about the environment is critical in assisting sensors, reducing accordingly computational effort. From a structural point of view, a HD map is organized in multiple layers. Lyft gives an idea about the anatomy of this kind of map, although it may slightly change depending on the HD map provider [36]. These layers are: the base map layer, the geometric map, the semantic map, map priors, and real-time knowledge as shown in Figure 2.22.

- **Geometric Map Layer**: this layer contains 3D information about the world. It is composed of collected raw sensor data. Data, in the form of 3D point cloud, is post-processed to produce derived map objects.
- **Semantic Map Layer**: this layer, built on the geometric map layer, includes semantic objects.
- **Map priors layer**: this layer contains information about dynamic elements and also human driving behaviour (e.g. the order in which traffic lights change).
- **Real-time knowledge layer**: this is a layer designed to be continuously updated and contains information about real-time traffic.

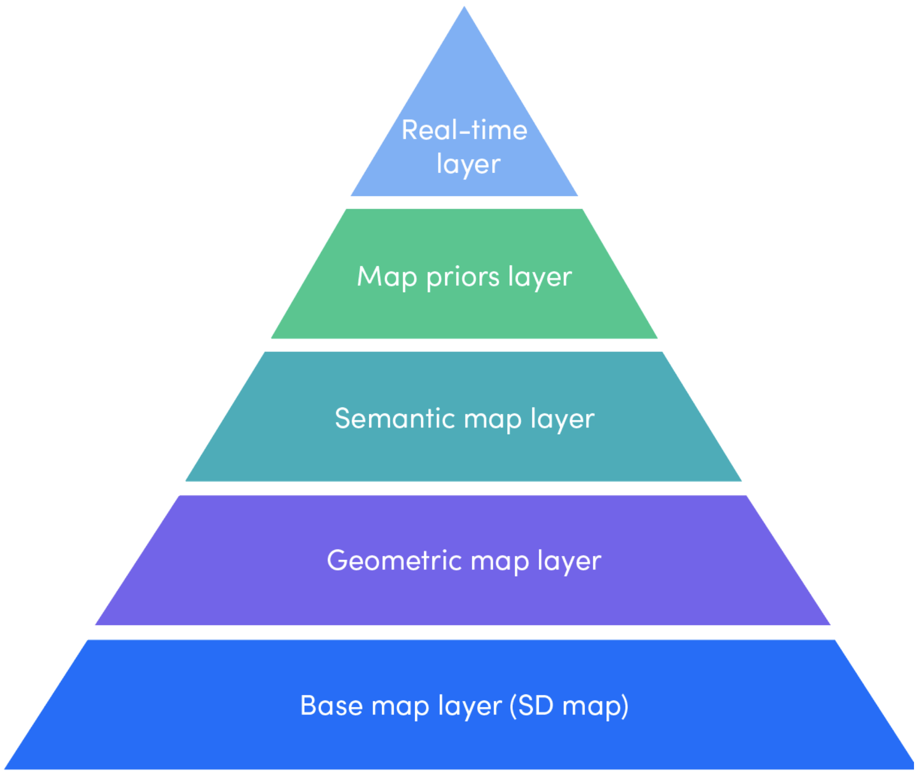


Figure 2.22 the layer-based structure of HD maps

HD maps creation requires great effort and resources but due to their importance, all the companies involved in this field are investing their time to optimize the process of map generation seeking to lower the cost. For this reason, companies adopt different approaches, some of them rely on crowdsourcing by encouraging drivers to gather data while others invest on computer visions to avoid the high cost of sensors such as LiDAR [38]. Some of the top key player operating in the industry are TomTom, HERE, Waymo, NVIDIA, and NavInfo, all of them being big companies, a further evidence of the required effort for HD map creation.

The importance of having HD maps to improve the level of safety and reliability in AVs has led many companies to integrate them in their simulation platforms ensuring high-fidelity. The necessity to speed up autonomous driving development is pushing towards the realization of a standard format for HD maps. Currently, the most popular formats supported by several companies are:

- **OpenDRIVE**: is an open format based on XML syntax for the description of road networks in driving simulation applications [39]. The data describes the geometry of roads as well as features along the roads that influence the logics (e.g. lanes, signs, signals) with the format organized in a hierarchical structure.

- **Lanelet2**: this format has been developed as an extension of the map format Liblanelet and was designed to be representable on the XML-based OSM (OpenStreetMap) data format [40]. Lanelet2 divides the world into a hierachical structure of six different primitives: Points, linestrings, polygons, lanelets, areas and regulatory elements.
- **Apollo Map format**: Baidu uses a modified version of the OpenDRIVE format for its Apollo platform [41]. Roads elements are represented by sequences of points.
- **Autoware Vector format**: Autoware [42] is one of the most popular and advanced open-source frameworks for autonomous driving vehicles and its HD map format is now widely supported. This format has been developed by the Japanese company Aisan Technology [43] but no official documentation is available.
- **Navigation Data Standard (NDS)**: NDS is a standardized format for automotive-grade navigation database, developed by automotive OEMs and suppliers [44]. It is not an open format, but it is widely supported by companies. Since it requires a purchase of a license, this format is not ideal for academic research.

The diffusion of these formats has led to the development of tools capable of map format conversion to facilitate the process of creation. Among the above-mentioned formats, OpenDRIVE can be considered the de facto standard due to its wide and increasing support from many companies and research teams.

2.4.1 Apollo and Autoware: two open source platforms strongly related to HD maps

The process of HD map creation consists of two steps: data collection and map generation. Data collection is a critical stage and the quality of data can make the difference for the final result. Two approaches can be adopted for data gathering: by collecting sensor data through vehicle or by using available sources of existing map data. The second step requires a software tool to give shape to the map itself. To this end, Apollo and Autoware represent a good choice for HD map development. These two platforms are both built on ROS [45] (Cyber RT for Apollo 3.5 and after), and in terms of open source software, they offer the state-of-the-art of the autonomous driving software stack and therefore their HD map formats are now popular. Figure 2.23 shows an overview of the architecture of both the platforms.

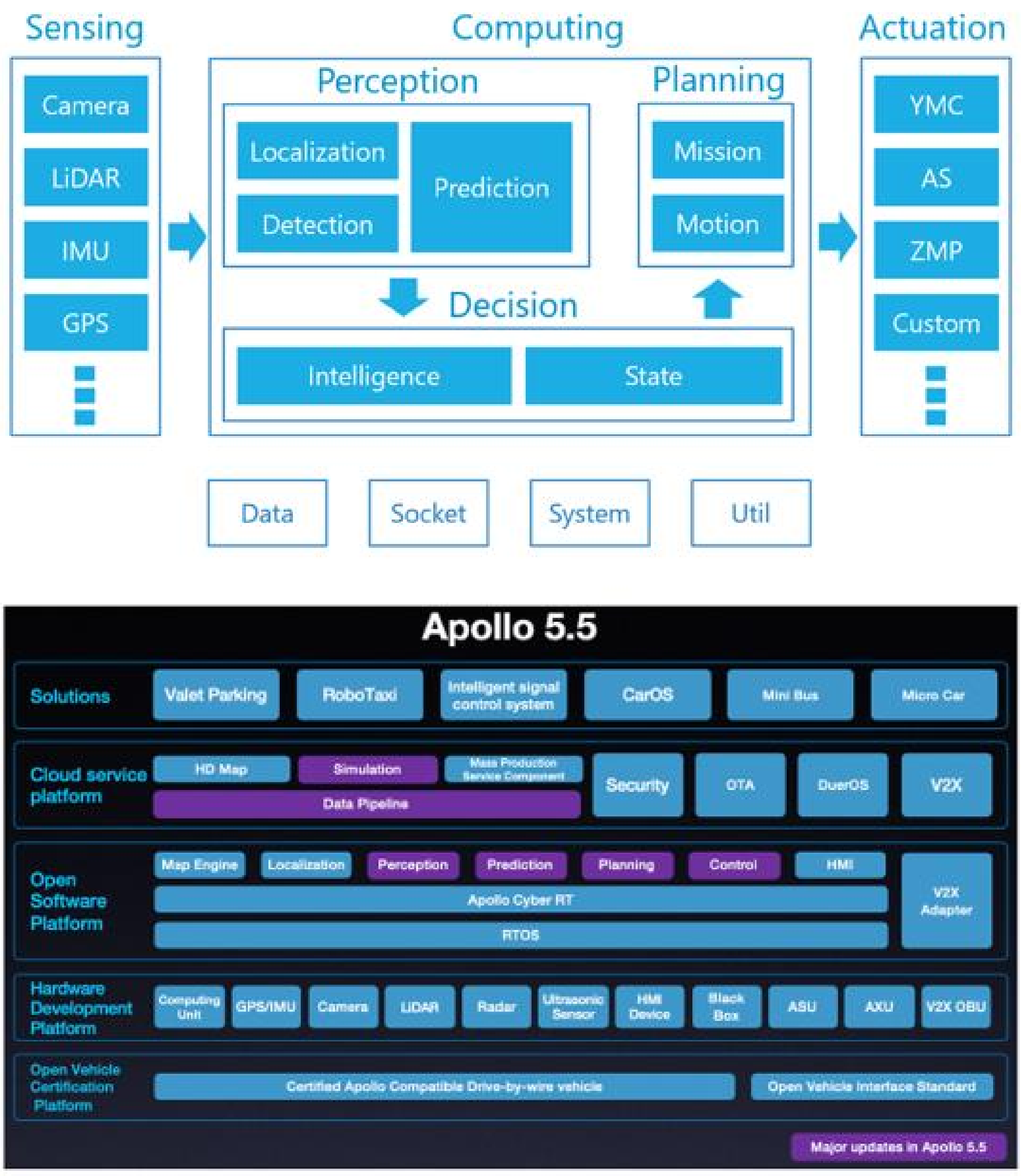


Figure 2.23 Overview of the two architectures: on top is shown Autoware architecture while on the bottom is shown Apollo architecture

As can be seen, both Apollo and Autoware share similar functionalities. HD maps are fundamental for localization and planning, and for this reason, in the next chapter the possibility to generate HD maps will be discussed by eventually exploiting mapping functionalities of one of these platforms.

2.5 Related works

Currently, the main trend for generating 3D scenario for AV simulation is mainly based on sensors fusion and manual editing. Geodata is still hardly used as the only source of information for scenario generation and only in recent years the automotive industry showed interest in speeding up the development and diffusion of reliable map database of the real world. The ongoing standardization and diffusion of specific formats for HD maps is particularly helping the academic world which is focusing the attention on developing useful tools for this area of the automotive industry. OpenStreetMap, being the only valid data source freely available, is currently supported by several platforms and tools but at the time of writing there are still no available works which outlined a similar methodology, fully based on the implementation of GIS data. Worth mentioning is Christoffer Wilhelm Gran's work [46]. Gran has developed a methodology for generating HD maps compatible with Apollo platform using OpenStreetMap data. The generated maps are very simple and not accurate but the work is still interesting since Apollo map format was investigated and the obtained results can be potentially useful for possible future works.

Chapter III

3 Methodology

The goal of this work is to define a process to automatically generate realistic 3D scenarios, using real world data from an open data source, to be implemented in driving simulation. The logical description of road networks is then used to build HD maps in accordance with the main standard formats. In this chapter, the methodology to define the process is presented.

3.1 Selecting the suitable simulator

This section focuses on the selection of the most suitable simulator for this case study, keeping in mind the fact that the choice is influenced by the necessity to use only an open source simulator. Moreover, the final choice dates back to the end of the last year, after conducting the literature review and some tests with different software.

The search, for the simulator to be used, can be further narrowed down by setting the following requirements as filters:

- Constant and long-term support from the development team
- Good documentation
- High-fidelity level of simulation
- Easy for users to customize the simulator according to their own needs
- Possibility to integrate the simulator with other platforms, especially with Apollo and Autoware
- Support to HD maps and formats for the logical description of road networks, such as the OpenDRIVE or lanelet2 formats.

The only 3D simulators that satisfy these requirements are CARLA and LGSVL.

For this reason, in the following tables only the main features of these two simulators are considered for a direct comparison.

The first table (**Table 3.1**) shows a comparison of the main sensors available in both simulators. The most essential sensors are implemented in both simulators.

Table 3.1 Sensors comparison

	SENSORS						
	RGB camera	Depth camera	Segmentation camera	Lidar	Radar	GPS	IMU
CARLA	+	+	+	+*	+	+	+
LGSVL	+	+	+	+	+	+	+

Table legend:(+) implemented, (-) not implemented

**: Lidar in CARLA does not generate intensity for the points nor semantic labels. A semantic Lidar is planned to be included in the next release.*

Table 3.2 shows instead the standard formats and AD platforms supported by the simulators.

Table 3.2 Standard format support

	AD Platforms		Formats		
	Autoware	Apollo	OSM	OpenDRIVE	Lanelet2
CARLA	+	-	+*	+	-
LGSVL	+**	+**	-	+	+

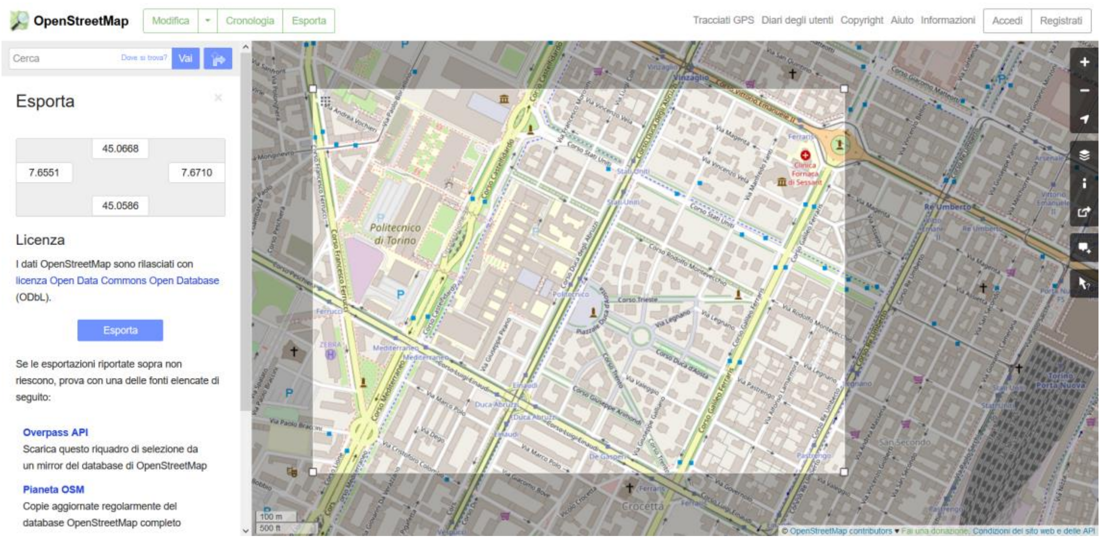
Table legend:(+) implemented, (-) not implemented

**: Osm support is planned to be included in the next release. **: Lgsvl allows to export HD maps in the Apollo and Autoware formats.*

Looking at both tables, it is evident that lgsvl simulator offers more functionalities but for both simulators it is possible to generate a HD map by defining a pipeline which exploits several tools. It was eventually decided to choose CARLA simulator since this one is release under the MIT license, giving more freedom than the proprietary license of lgsvl simulator.

3.2 Data source

The first step is to identify an open source database offering geodata. There are several companies and agencies that made available for free their database, but this is limited to certain areas only. The data must contain information about roads, in particular their geometry as well as their features. OpenStreetMap is currently the only valid free option to offer the necessary data, good enough to be used for HD map generation. The growing popularity of this project attracted millions of users, who support the project via crowdsourcing, and many big companies giving the opportunity to compete with giants like Google. This success allowed the .osm file format to become a standard widely used. The official website of OpenStreetMap [47] offers the possibility to export an .osm file of a selected area as shown in Figure 3.1.



Esporta

45.0668
7.6551
7.6710
45.0586

Licenza

I dati OpenStreetMap sono rilasciati con licenza Open Data Commons Open Database (ODbL).

Esporta

Se le esportazioni riportate sopra non riescono, prova con una delle fonti elencate di seguito:

Overpass API
Scarica questo riquadro di selezione da un mirror del database di OpenStreetMap

Pianeta OSM
Copie aggiornate regolarmente del database OpenStreetMap completo

Figure 3.1 Map exporting from a selected area

A valid alternative is to export osm data from the website BBBike [48] which allows to select different file formats for the road data to be exported with the possibility to manually set the boundary box shape. The osm file is mainly distributed in the XML format. All the features regarding the map are defined by the following three primitives:

- **A node**: this primitive defines points in space and includes its coordinates
- **A way**: this primitive defines linear features and area boundaries
- **A relation**: this primitive defines a relationship between two or more data elements.

Many available editors allow to visualize road networks derived from extracted osm files and eventually make changes. This can be useful to check the quality of the road data.

3.3 Conversion to OpenDRIVE format

Having chosen CARLA as simulator, it is fundamental to have available data formatted to the OpenDRIVE format since CARLA requires this type of file. For this reason, a conversion from osm to xodr (OpenDRIVE file extension) is required. It is possible to exploit the conversion tool included in SUMO (Simulation of Urban MObility) to generate an OpenDRIVE file from the osm data.

3.3.1 SUMO netconvert

The netconvert [49] tool allows to import road networks from different sources and then generates as output a new file formatted as specified in the command line. It is possible to easily install SUMO on Windows, Linux and macOS and then use its command line applications by setting the necessary parameters.

SUMO is an open source microscopic traffic simulator which allows to simulate and manage road traffic [50]. It is provided as a package containing useful tools for the generation, validation and evaluation of traffic scenarios.

In this specific case for the conversion from osm to OpenDRIVE, the command to be run inside the shell is the following:

```
netconvert --osm <FILE>.osm --opendrive-output <FILE>.xodr
```

The converted file is generated according to the version 1.4 of the OpenDRIVE specification. This version of the OpenDRIVE standard is currently the most supported one with the 1.6 released in March 2020.

From the official website of OpenDRIVE, a basic viewer can be downloaded to visualize the OpenDRIVE data as a map [51]. Figure 3.2 shows the representation on the viewer of the converted data.

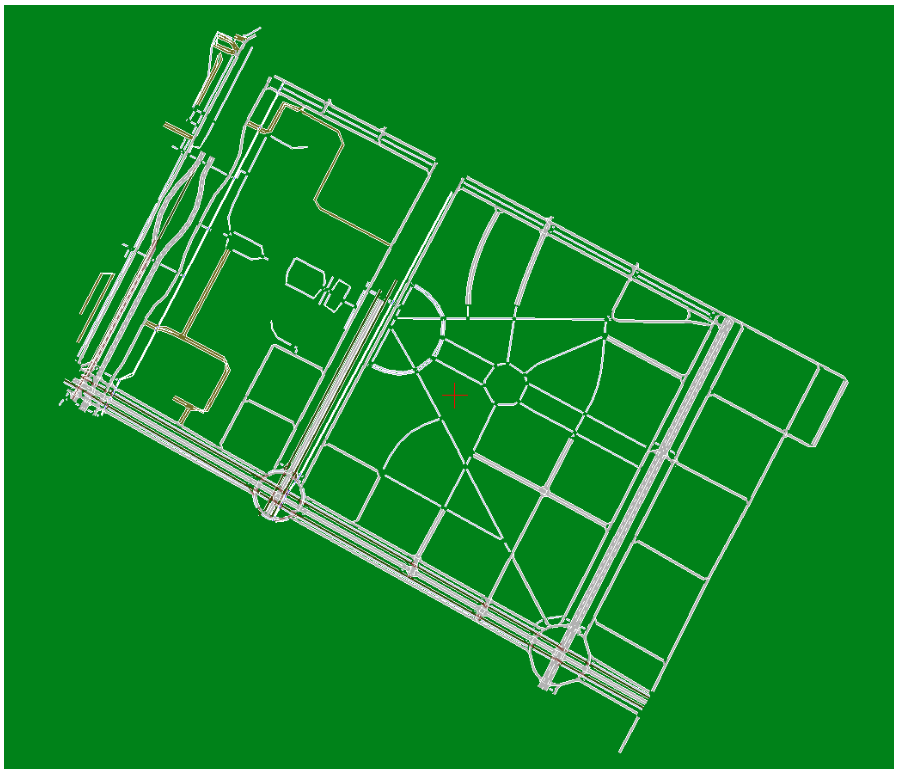


Figure 3.2 Visualization of the OpenDRIVE data - Crocetta district

3.4 3D Map generation

CARLA simulator is built on top of the Unreal Engine 4, hence map generation requires that all the generated assets must be compatible with the engine in case an external editor is used. Two approaches are possible to generate a map by exploiting osm data:

- Generate the map inside Unreal Engine
- Generate the map by using third-party editors

Currently, there are not many available solutions offering the possibility to automatically generate maps from geographic data. In general, the main trend in the industry is to manually build a map and only recently the focus is shifting towards automatic generation processes. On the web, some plugins for the Unreal Engine capable of generating maps by importing osm data are available or in development but only one turned out to be potentially interesting during the initial investigation. In the end, after some considerations about the desired results, the choice fell on the RoadRunner [52] editor, which represents the best solution, since it offers native support for CARLA simulator. The use of

RoadRunner was possible thanks to the collaboration between TeoresiGroup and MathWorks. In next subsection the procedure to generate a map from osm data by using the Unreal Engine plugin is still briefly shown since it represents a valid free option to RoadRunner.

3.4.1 StreetMap

The StreetMap [53] plugin, available for free on GitHub, allows to import osm data into Unreal Engine rendering streets and buildings. The plugin can be built alongside with CARLA to generate drivable roads almost out of the box. CARLA assets can be used to improve the results though the plugin still has obvious limitations:

- Roads are very simple; they do not present lane markings and traffic directions are not rendered
- Textures are not always correctly rendered
- The lack of an OpenDRIVE file does not allow to enable autopilot and cause some issues with the Python API

In Figure 3.3 is shown an example of map generated via plugin

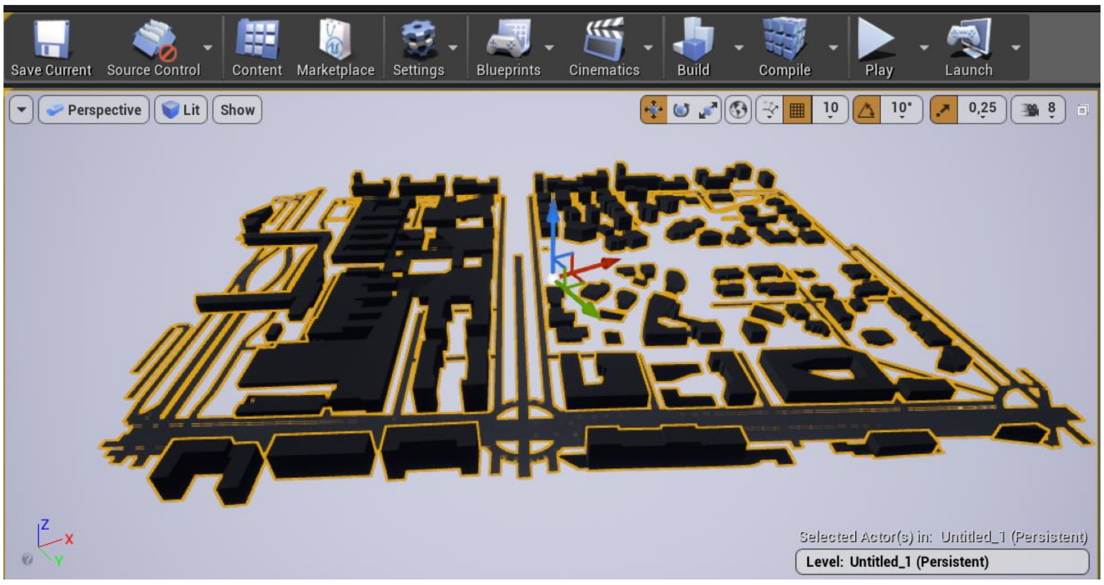


Figure 3.3 Map created with StreetMap plugin

Tests with this approach were limited to run the manual control python script after making some changes to the script.

For the reasons set out above, the plugin requires some improvements and some changes to the python scripts of CARLA need to be made but anyway the plugin is still interesting for its open source nature.

3.4.2 RoadRunner

RoadRunner editor lets to design 3D scenarios for simulating and testing AD systems. Among the various features, such as the possibility to import gis data for visual reference, is of particular interest the one that allows to import OpenDRIVE files. This last one is very useful since it is possible to automatically generate roads from an OpenDRIVE file by simply clicking the **Convert to roads** button. RoadRunner currently supports OpenDRIVE 1.4, the same version supported by CARLA. Importing OpenDRIVE files has some limitations as indicated in the official documentation and therefore the resulting map may have some inconsistencies that require manual intervention. The degree of interventions can also be affected by the quality of the initial imported data. For any changes made afterwards to the scene, the OpenDRIVE file is updated to include the new information. Once the map is ready, it is possible to generate the necessary files to be exported to CARLA. The option to export the map can be selected by clicking **File > Export > CARLA (.fbx + .xml + .xodr)** in the menu. In Figure 3.4 is shown one of the generated maps successfully imported in CARLA.

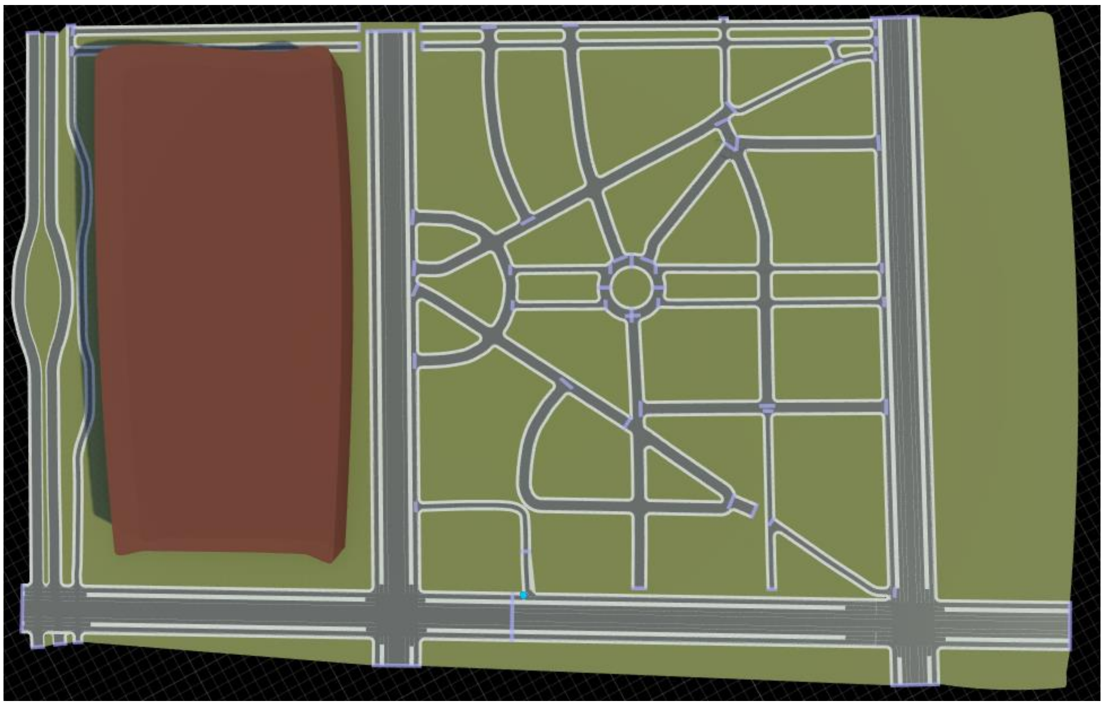


Figure 3.4 Map generated in RoadRunner - Crocetta district

The generated map could possibly require manual interventions to achieve a good result and avoid possible sources of crashes in CARLA. Special attention is required for junctions and maneuver roads within them. RoadRunner offers several junction tools to edit geometry and traffic behaviour.

3.5 Setting up the simulation environment

Before running the actual experiment to generate the HD map related to the 3D scene previously obtained in RoadRunner, it is necessary to set up the simulation environment by building all the required software and dependencies.

Figure 3.5 shows the design of the pipeline related to the simulation environment.

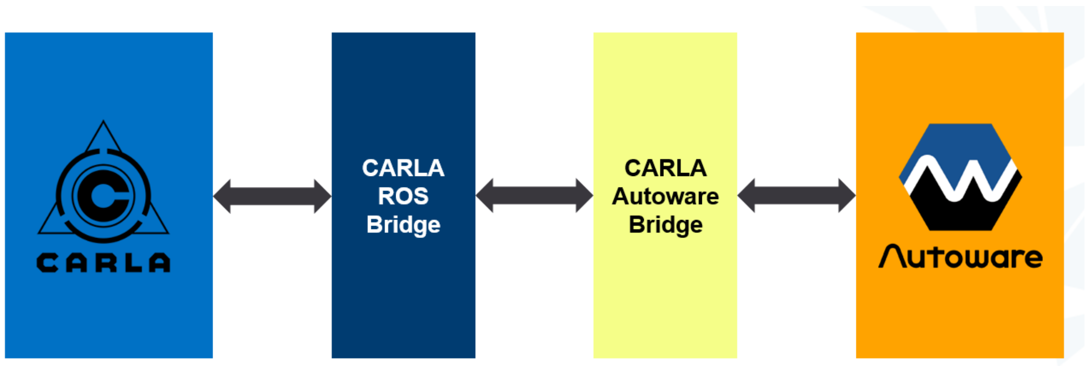


Figure 3.5 Design of the followed pipeline

As shown in the previous figure, ROS is used as middleware to link CARLA to Autoware.

The setup chosen for the case study is the following:

- Ubuntu 18.04
- Unreal Engine 4.24
- CARLA 0.9.9 (built from source)
- ROS melodic
- Autoware 1.14 (built from source)
- CARLA-ROS bridge
- CARLA-Autoware bridge

In addition to these ones, there are other software required for the proper functioning of all the environment. All these requirements are properly indicated in the documentation of every software listed above.

3.5.1 CARLA setup and map importation

CARLA can be easily built by running the following two **make** commands in the terminal:

1. **make launch** to compile the server simulator and launch Unreal Engine

2. **make PythonAPI** to compile the API client, necessary for simulation control

It is possible to use a pre-packaged version of CARLA which does not require to be build. This version has lower system requirements, but it is not possible to customize maps inside the editor and it requires the use of the Docker image of Unreal Engine to automatically ingest a map. After building CARLA, the process to ingest a map is very simple. The files previously generated in RoadRunner are placed in the Import folder inside CARLA. Once placed them, running the **make import** command is enough to start the importing process. A possible alternative to this method is to use the Unreal Engine plugin of RoadRunner which turned out to be more reliable in some cases. Figure 3.6 shows the new map correctly ingested in the Unreal Engine editor.

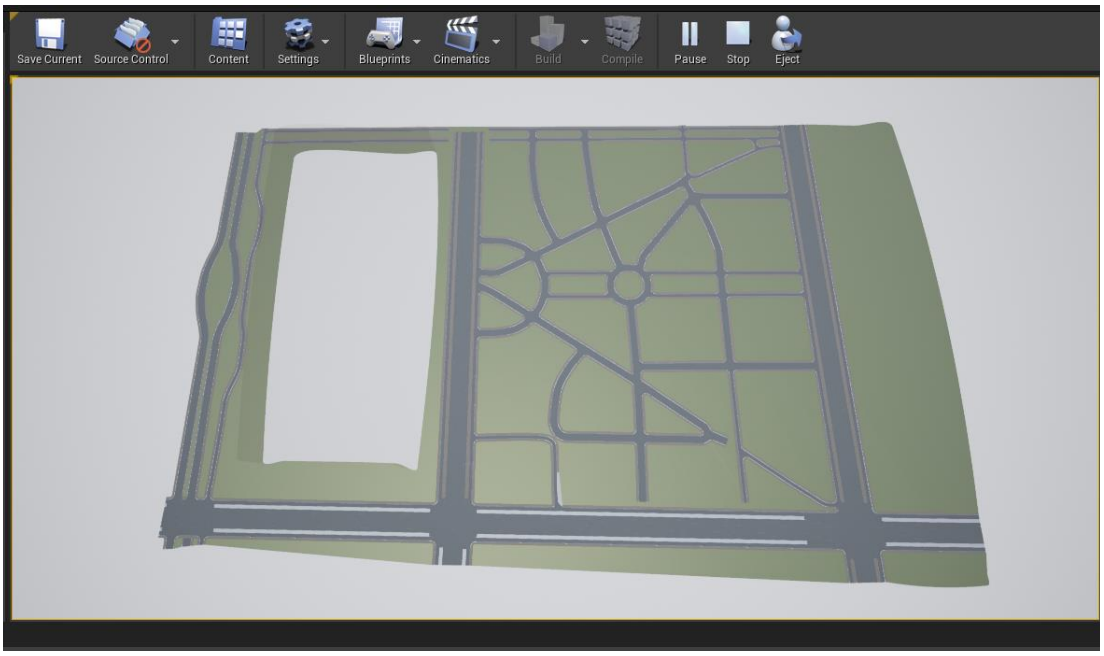


Figure 3.6 Map successfully imported in Unreal Engine

It is possible to check the correct functioning of the new map by running a python script of CARLA such as the **manual_control.py** from the Examples folder.

3.5.2 Bridge to Autoware

Once CARLA is ready, it is possible to proceed with the installation of ROS and Autoware. With the latest releases is now easier to build the bridge thanks to the collaboration between the two teams behind CARLA and Autoware. As matter of fact, Autoware 1.14 natively supports the bridge with CARLA. To launch the bridge, it is necessary to run the following commands on different terminals:

1. Run CARLA:

```
$ make launch-only
```

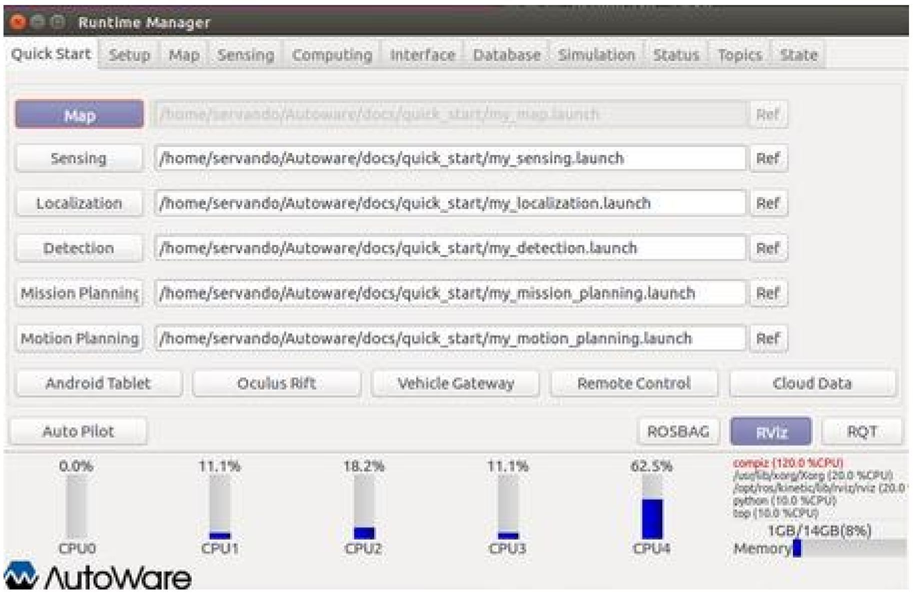
2. Run bridge:

```
$ roslaunch carla_autoware_bridge carla_autoware_bridge.launch
```

3. Run the agent:

```
$ roslaunch runtime_manager runtime_manager.launch
```

The Runtime Manager GUI (Figure 3.7) allows to operate functions and load 3D point cloud/vector map. These two kinds of maps are fundamental and therefore in the next section is shown how to generate these files. In the absence of a vector map is still possible to drive by following a pre-recorded route consisting of waypoints which can be made by using the **waypoint_saver** functionality available in the computing tab of the Runtime Manager interface. However, this functionality is beyond the scope of this work.



Map: /home/servando/Autoware/docs/quick_start/my_map.launch
Sensing: /home/servando/Autoware/docs/quick_start/my_sensing.launch
Localization: /home/servando/Autoware/docs/quick_start/my_localization.launch
Detection: /home/servando/Autoware/docs/quick_start/my_detection.launch
Mission Plannin: /home/servando/Autoware/docs/quick_start/my_mission_planning.launch
Motion Planning: /home/servando/Autoware/docs/quick_start/my_motion_planning.launch

```
compiz (120.0 %CPU)
/usr/lib/xorg/Xorg (20.0 %CPU)
/opt/ros/kinetic/lib/rviz/rviz (20.0
python (10.0 %CPU)
top (10.0 %CPU)
```

Figure 3.7 Runtime Manager Interface

3.6 Autoware HD map generation

As explained above, Autoware requires a 3D point cloud map and a vector map to correctly work. The former is used for localization while the latter contains all the information inherent to the road, such as lanes, traffic lights, signs and intersections.

There are different possible methods to obtain such kinds of map, some of which involve tools available on the web. The following subsections focus on showing the recommended and tested methods.

3.6.1 Point cloud map

ROS bridge [54] offers the possibility to create point cloud maps from CARLA maps. By driving around the map, point clouds are created and then stored in the temporary folder. The entire process makes use of pcl tools [55]. After executing CARLA, the command to be run is the following:

```
$ roslaunch pcl_recorder pcl_recorder.launch
```

This command gives control over the vehicle and starts the point cloud capturing. It is highly recommended to enable the autopilot to make easier the capturing.

Once the capture is done, the size of the acquired data can be reduced by running:

#create a single point cloud file

```
$ pcl_concatenate_points_pcd /tmp/pcl_capture/*.pcd
```

#filter duplicates

```
$ pcl_voxel_grid -leaf 0.1,0.1,0.1 output.pcd map.pcd
```

A check of the result can be done by launching the pcl viewer.

3.6.2 Vector map generation

The procedure to generate vector map in the Autoware format consists of the following two steps:

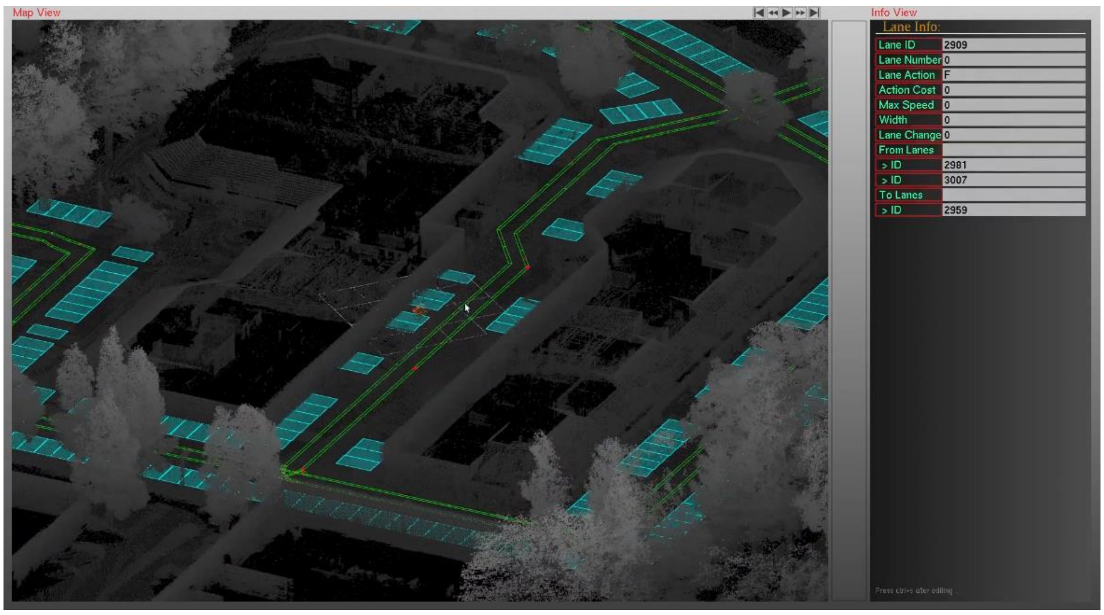
1. Conversion from Opendrive to lanelet2
2. Conversion from lanelet2 to Autoware vector map

The choice to use lanelet2 as intermediate step is motivated by the fact that Autoware is now supporting this format by also including a conversion tool among the available utilities.

The first step requires to install ASSURE mapping tools [56], a useful tool for viewing, editing and saving road network maps. This tool supports the following formats:

- OpenPlanner[1] [57] map format .kml (import/export)
- Google Earth .kml (export)
- Vector map for Autoware (import)
- Opendrive .xodr (import)
- Lanelet2 .osm (import/export)

Point cloud data can be imported and used as reference to guide the process of editing, which relies on a minimalist GUI (Figure 3.8).



Lane Info:	
Lane ID	2909
Lane Number	0
Lane Action	F
Action Cost	0
Max Speed	0
Width	0
Lane Change	0
From Lanes	
> ID	2981
> ID	3007
To Lanes	
> ID	2959

Figure 3.8 Appearance of the tool Interface. Point cloud data is used as visual reference as shown

[1] https://github.com/Autoware-AI/core_planning/tree/master/op_global_planner

Once loaded the Opendrive file generated from RoadRunner, it is possible to proceed with the conversion and obtain the equivalent lanelet2 file.

The obtained file can be now converted into Autoware Vector map by means of the lanelet aisan converter [58] available in the utilities folder of Autoware.

The conversion can be done by running the following command:

```
rosrun lanelet_aisan_converter lanelet2aisan _map_file:=<path to lanelet map>
_origin_lat:=<latitude> _origin_lon:=<longitude> _save_dir:=<path to save the
map>
```

At the end of the conversion, several .csv file containing semantic information about the road network. Conversion is not 100% accurate, hence the obtained vector map may require manual adjustments to achieve the desired level of quality. Tier IV, the company behind Autoware, offers a free web-based tool to manually build or edit vector maps [59]. Using the vector map builder tool can fill in the gaps due to the previous conversion. Although this process could require substantial manual intervention, largely going against the research goal of obtaining an automatic process, it is shown anyway, since it is currently the most valid available option to build a vector map for Autoware. The tool requires to sign up for an account before using it. Once logged in, the tool features a simple Interface as shown in Figure 3.9.

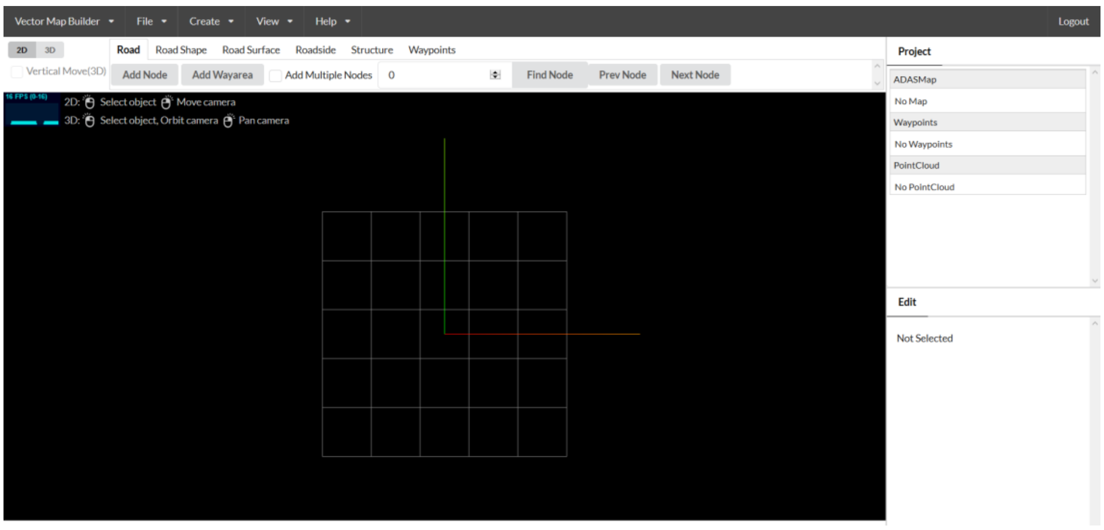


Figure 3.9 Interface of Vector Map Builder

The first step is to load the .pcd file, previously obtained by using the **pcl recorder** command. Since the tool can only read files in the binary format, it is necessary to convert the file from ASCII to binary. This can be done by running the following command which exploits pcl tools:

```
$ pcl_convert_pcd_ascii_binary <map.pcd> <bin-map.pcd> 1
```

After also importing the vector map obtained from the lanelet2aisan conversion, it is possible to edit the map and make the final adjustments. Eventually, the map is ready to be used in Autoware.

3.7 The overall pipeline – from OpenStreetMap to Autoware

In this Chapter, the focus has been put on defining the methodology to be adopted for generating a 3D scenario for autonomous driving simulation by exploiting free GIS data. To summarize, the process begins with OpenStreetMap data extraction to obtain an .osm file which is then converted in the OpenDRIVE format (.xodr) through SUMO netconvert tool. The converted .xodr file is imported in RoadRunner to automatically generate the road network. In this phase it is possible to customize the generated map, if desired, before exporting it to CARLA. CARLA simulator is finally connected to Autoware through a ROS bridge. To run the simulation, an HD map is required. Such HD map is obtained by combining two files: a point cloud map and a Vector map. The former is generated by collecting point cloud data of the 3D map while the latter is generated in two steps: a conversion of the OpenDRIVE file in the lanelet2 format by means of ASSURE mapping tools, which is then converted in the Vector map format through one of the Autoware utilities. Once obtained and loaded these files into Autoware, it is possible to eventually run the simulation. Figure 3.10 shows the methodology workflow.

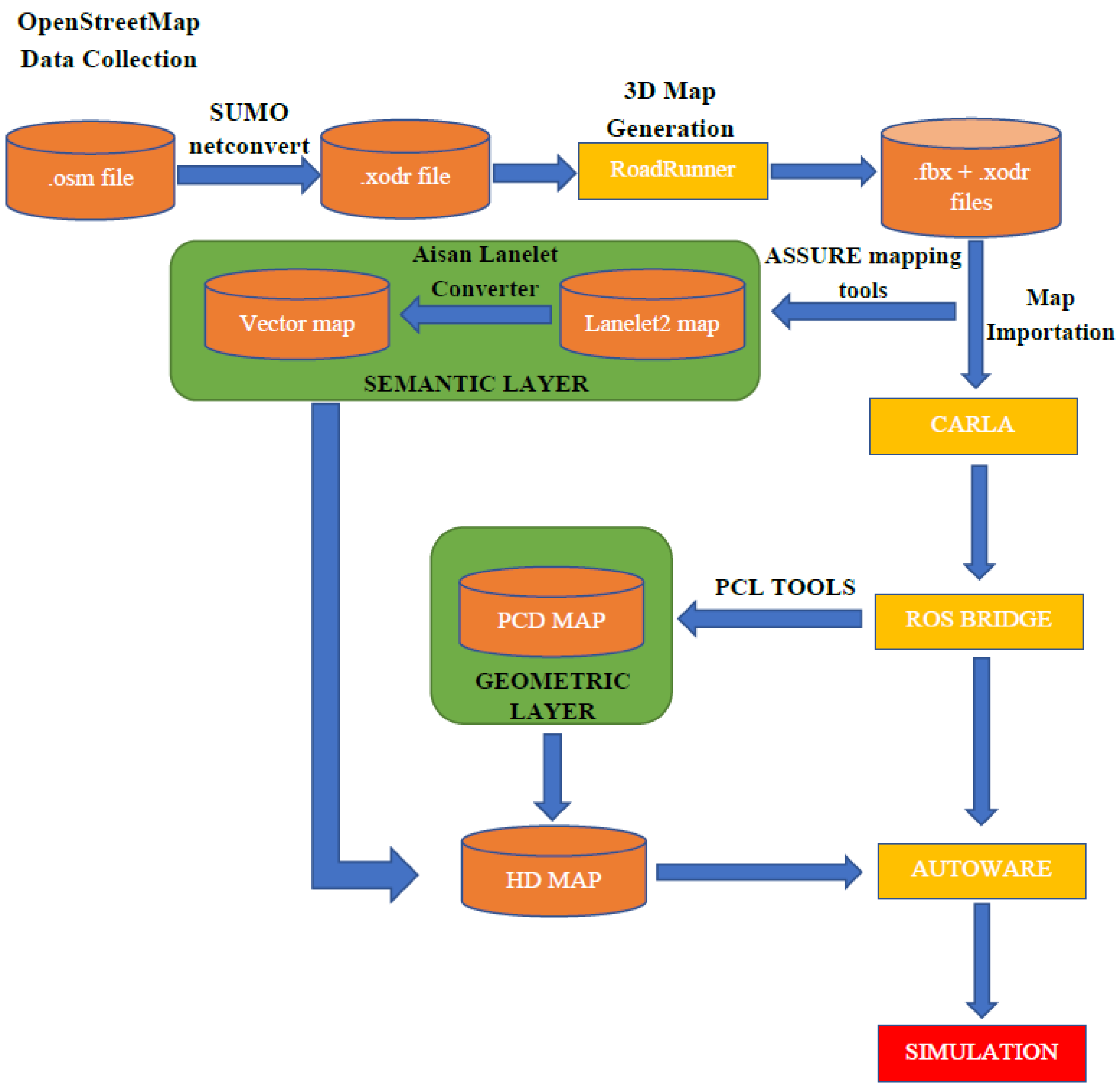


Figure 3.10 The entire workflow

Chapter IV

4 Results

This chapter is dedicated to the presentation of the results obtained by following the above outlined methodology. Specifications of the laptop, utilized for this study, are listed below as reference:

- CPU: i7-7770HQ, 2.80GHz 4 cores
- RAM: 16 GB
- GPU: NVIDIA GTX 1070, 8GB GDDR5
- OS: Ubuntu 18.04

4.1 OSM data

OpenStreetMap database is a valuable source of road network data but it has one significant limitation, map accuracy is not always guaranteed since the service relies on crowdsourcing. This led to a preliminary quality check of the osm data to be used. Several tests were carried out on different areas of the city of Turin to find a suitable one for the map generation. The analysis of the data extracted from the OpenStreetMap database was conducted by means of SUMO netedit [60]. This tool is a visual network editor, built on top of netconvert, which allows to visualize the road network. Figure 4.1 shows a comparison between the original map and the corresponding road network visualized in netedit.

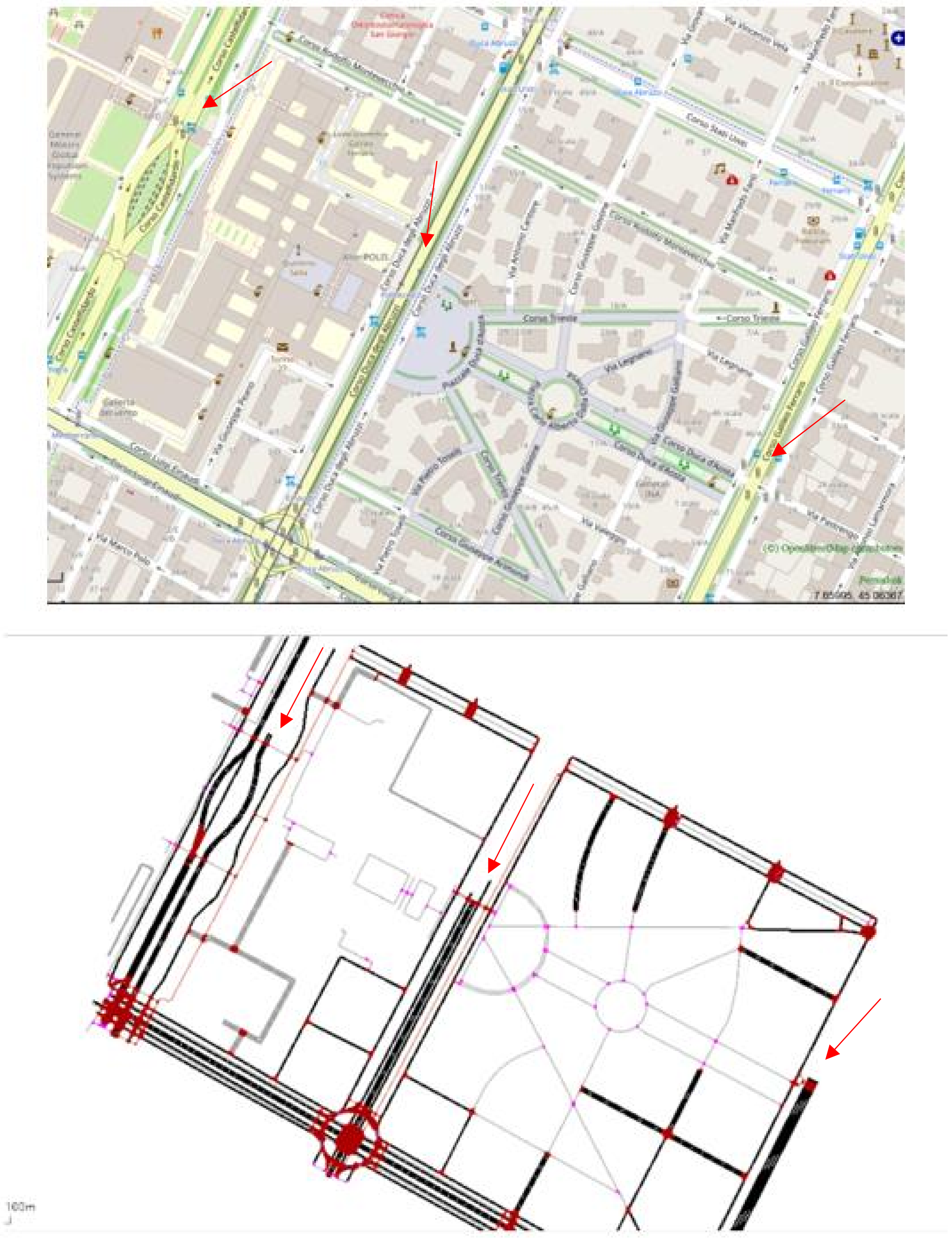


Figure 4.1 A comparison between a selected area of Turin and its corresponding extracted road network

By looking at Figure 4.1, it is possible to notice some inconsistencies (indicated by the red arrows) between the two maps. The absence of some stretches of road is due to the aforementioned crowdsourcing model adopted for mapping. The use of netedit tool allowed to check the extracted data before proceeding with the conversion into OpenDRIVE format. In the end, it was decided to use, as reference for the experiment, an area from the Aurora district after assessing the quality of the corresponding osm data as shown in Figure 4.2.

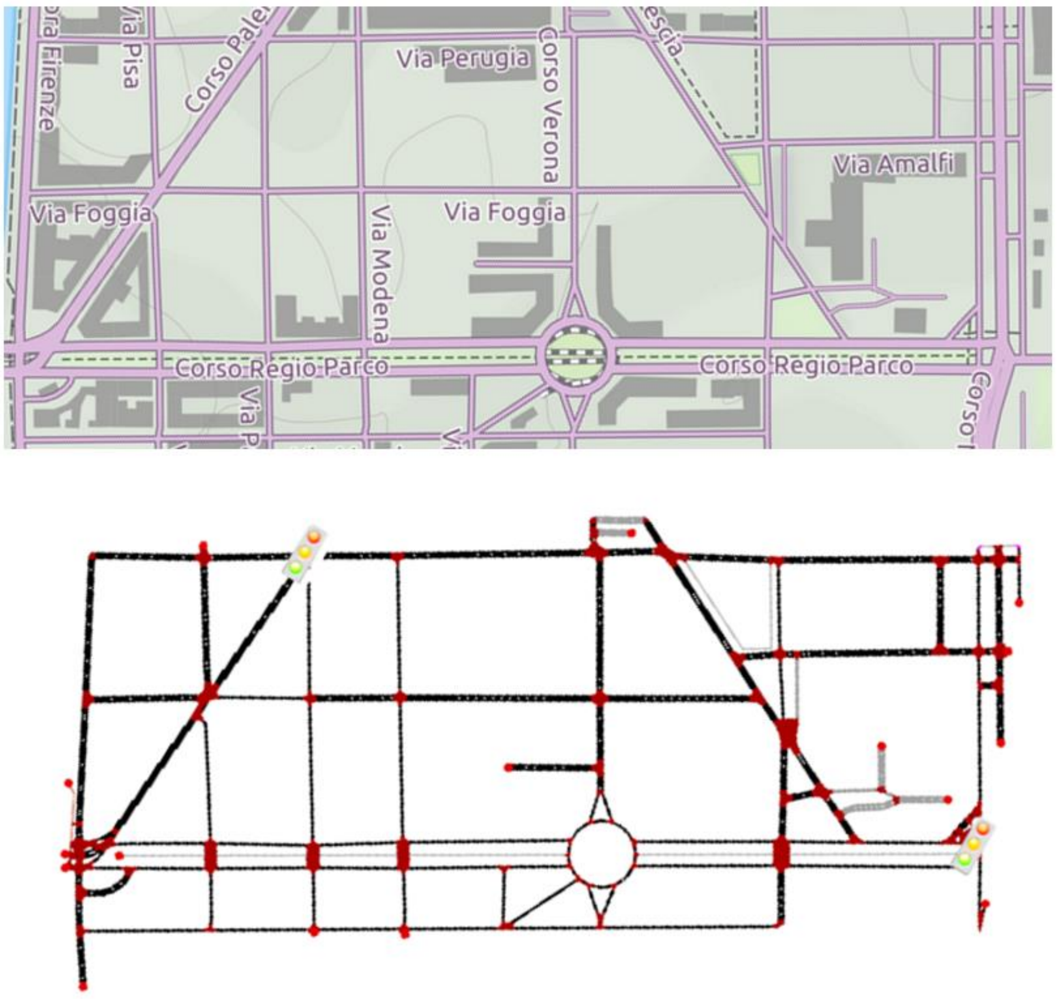


Figure 4.2 The area chosen to conduct the experiment

Figure 4.2 shows how the extracted road network and the map perfectly match, turning out to be a good choice.

4.2 OpenDRIVE conversion

Conversion from osm to OpenDRIVE format was performed through netconvert tool from SUMO. It is possible to visualize the map, corresponding to the obtained OpenDRIVE file, through the viewer available online (Figure 4.3).

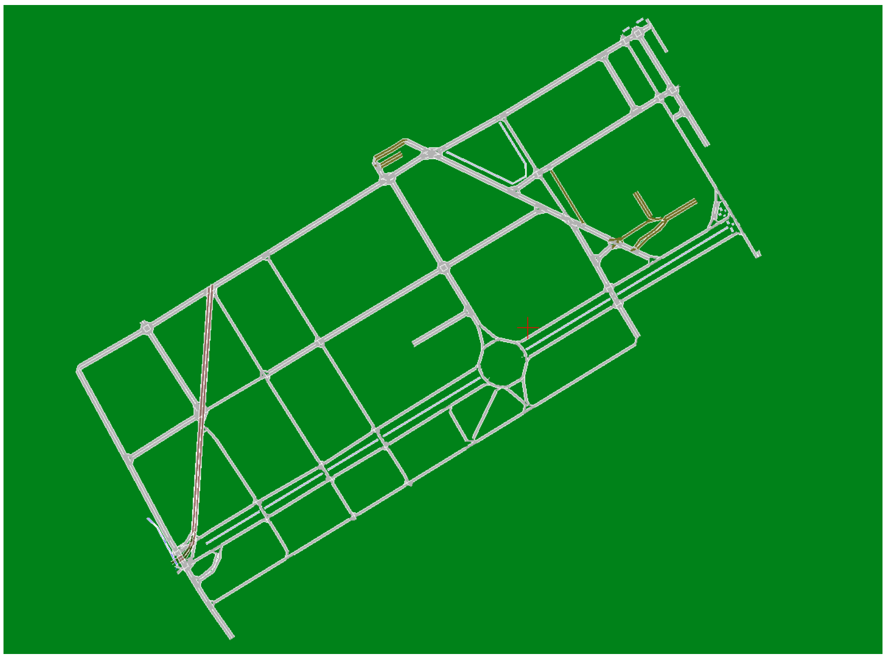


Figure 4.3 Map visualized in the OpenDRIVE viewer

A quick comparison between the map in Figure 4.2 and the one in Figure 4.3 shows that the conversion generated an acceptable map, at least as regards the road network. A more in-depth analysis allows to notice the absence of some elements in the map such as the traffic lights and traffic signs or some geometric anomalies in junctions. These errors are directly caused by the netconvert tool which has significant limitations.

4.3 Generated map for CARLA

The OpenDRIVE generated from the conversion was imported in RoadRunner to automatically generate the road network. Figure 4.4 shows the road network automatically generated in RoadRunner.

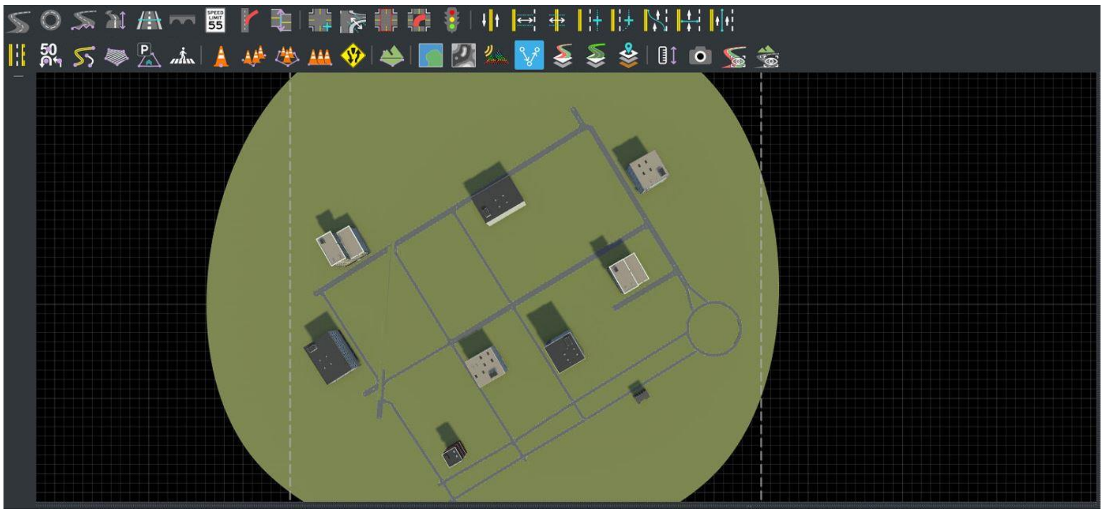


Figure 4.4 A snapshot of the map generated in RoadRunner

When importing OpenDRIVE files, RoadRunner builds junctions based on the overlap of roads while dead-end roads are readjusted with the addition of a U-turns placed as junctions at the end. The map was centered in (0,0) as indicated in the CARLA documentation before exporting [61]. Once the map was ready, the scene was exported in CARLA by embedding all the textures in the .fbx file. The map was then successfully imported in CARLA where it was eventually used to generate the corresponding point cloud map in the .pcd format by means of the **pcl_recorder** tool included in the ROS bridge. In Figure 4.5 is shown the generated pcd map.

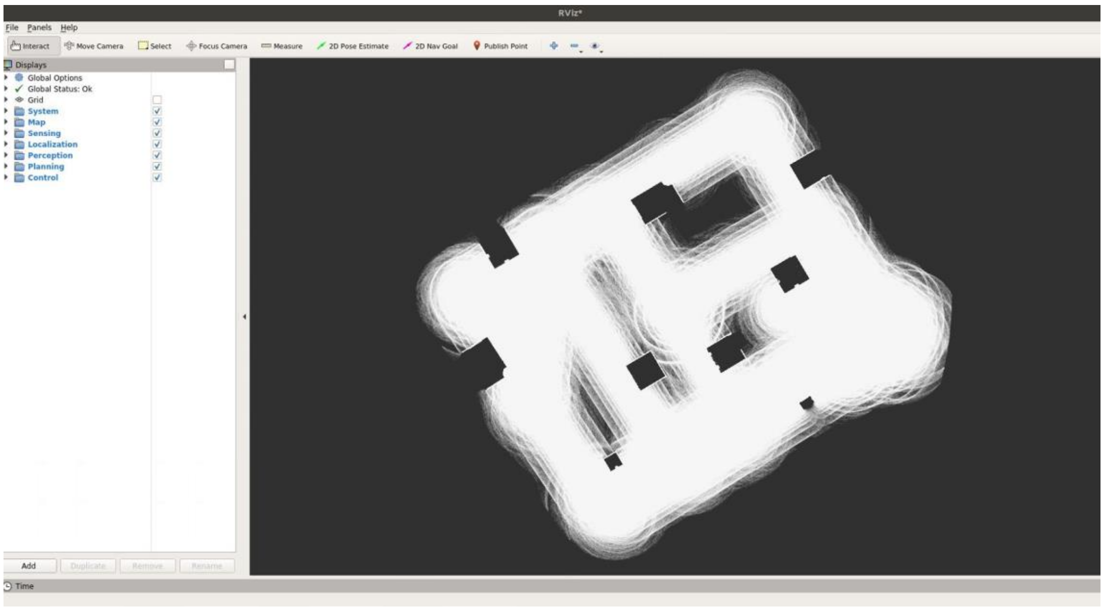


Figure 4.5 The generated pcd map successfully loaded in Autoware

4.4 HD map for Autoware

HD map generation was achieved by means of two different tools. The first tool, ASSURE mapping tools, allowed to convert the OpenDRIVE file in lanelet2. Figure 4.6 shows the resulting map in the lanelet2 format.

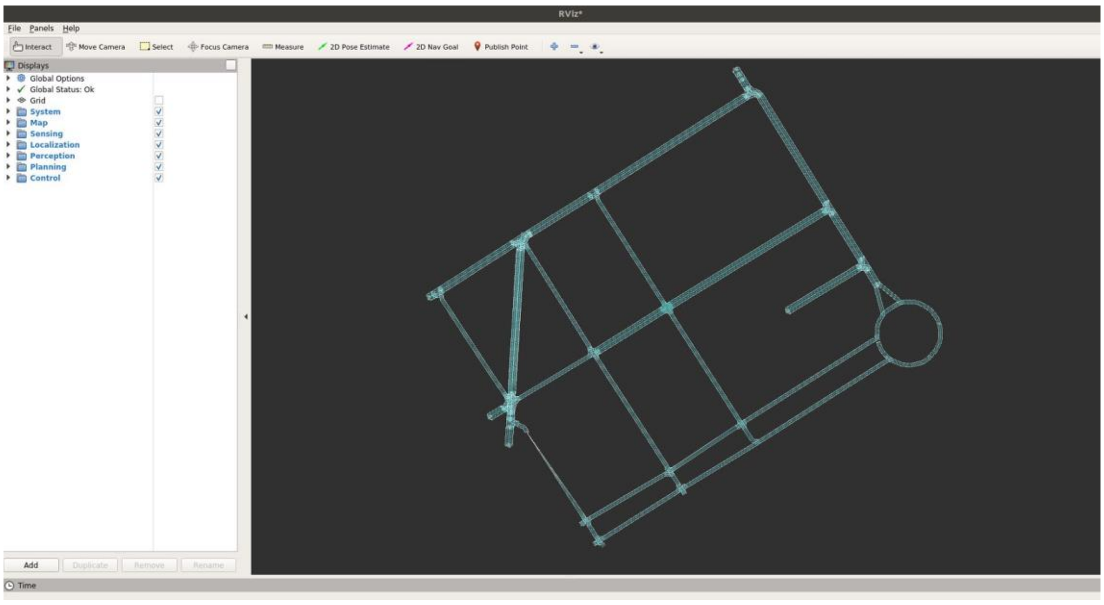


Figure 4.6 The generated lanelet2 map successfully loaded in Autoware

The generated map contains simple information about the roads, and elevation tags are always present. No information about traffic signs or lane markings are included.

The lanelet2 map was then converted in the Vector map by means of one of the Autoware utility. The conversion process generated the following series of .csv files:

- area.csv
- dtlane.csv
- intersection.csv
- lane.csv
- line.csv
- node.csv
- point.csv
- wayarea.csv
- whiteline.csv

After obtaining the necessary maps, it was possible to load the two maps and finally build the desired HD map. Both maps contain information about their coordinates. A specific tf launch file is required to provide the necessary transforms for the different coordinate frames. Figure 4.7 shows how the pcd map and the lanelet2 perfectly match. The HD map was thereby ready.

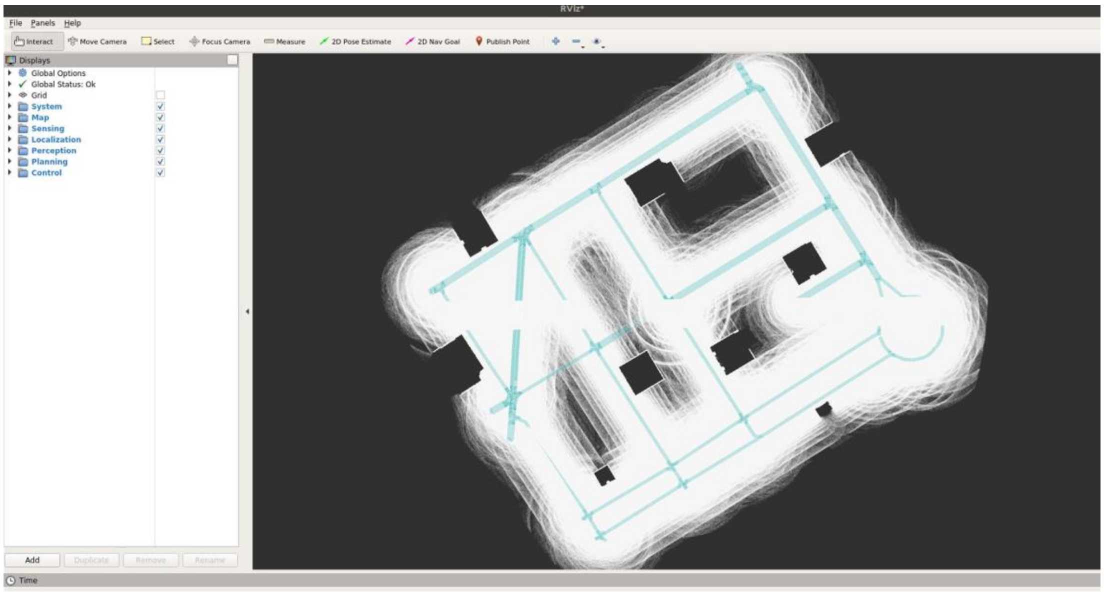
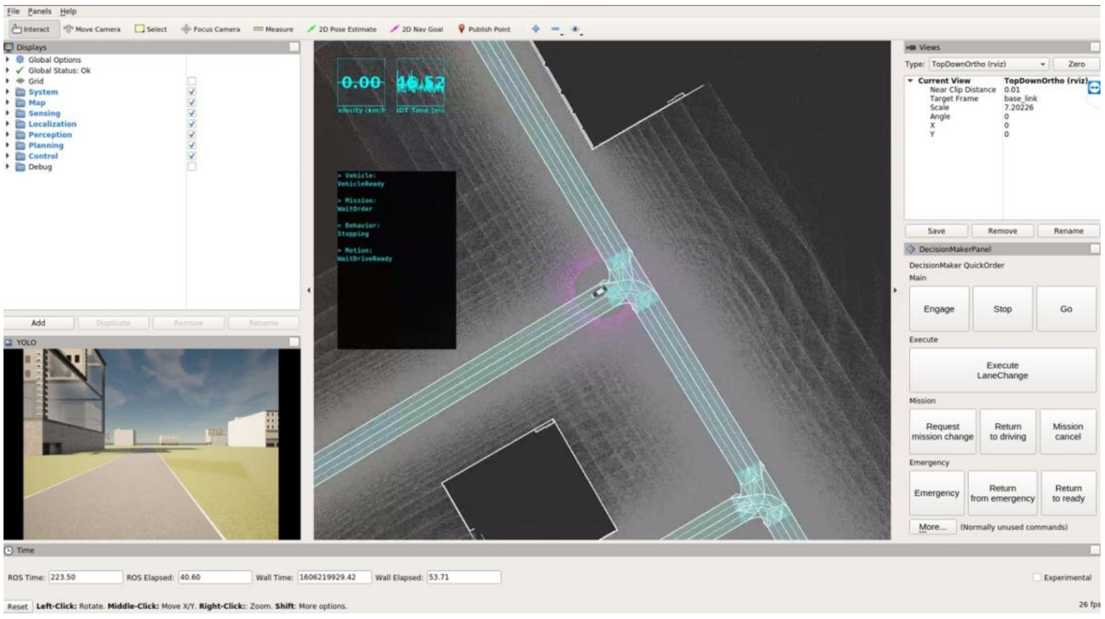


Figure 4.7 Matching between pcd map and lanelet2 map

4.5 Running the final simulation

In the end, having obtained the HD map, it was possible to launch the CARLA-Autoware bridge. Control over the simulation can be taken by directly interacting with Autoware or by enabling the **manual control** functionality provided by CARLA PythonAPI module through ROS bridge. In Figure 4.8 is shown the simulation successfully performed while the bridge is in execution.



```
> Vehicle:
VehicleReady

> Mission:
WaitOrder

> Behavior:
Stopping

> Motion:
WaitDriveReady
```

Type: TopDownOrtho (rviz)

Current View	TopDownOrtho (rviz)
Near Clip Distance	0.01
Target Frame	base_link
Scale	7.20226
Angle	0
X	0
Y	0

ROS Time: 223.50 ROS Elapsed: 40.60 Wall Time: 1606219929.42 Wall Elapsed: 53.71

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift**: More options.

Figure 4.8 A snapshot of the simulation running on Autoware and CARLA via bridge

Chapter V

5 Discussions

This chapter focuses on discussing and making observations on the findings of this work, which aims to define an automatic process for generating a realistic 3D simulation scenario featuring semantic information extracted from real world road networks. Moreover, limitations are underlined to address the weaknesses of this study and highlight all the aspects of the methodology which require further development to improve results. The previous chapter showed how it is possible to generate 3D scenarios by means of several tools, eventually deploying them into an end-to-end driving simulation.

5.1 Map data

As seen in the previous chapters, OpenStreetMap represents the only valid road network data source freely available to be used in map generation. The osm file format structure is very intuitive, and tags allow to quickly analyse and change any code section. However, data are not always good and during the research work a few limitations has come to light. Anyway, available data is in general quite accurate and traffic information included in the database still represents a valuable asset which can be exploited in simulation. Turin was chosen as the reference area for demonstration purposes only. All areas which were presenting inconsistencies or missing data were avoided in order to have a good map as starting point. This step was crucial since SUMO conversion from osm to OpenDRIVE is not perfectly accurate and therefore further issues can emerge. In this regard, SUMO netconvert turned out to be the most critical phase of the entire workflow due to the limitations of the conversion tool. Converted files are mainly afflicted by geometrical issues concerning junctions, which in some cases require to manually intervene. Furthermore, the tool is not able to maintain all the information derived from the osm file such as traffic signs. It is possible to find missing data by directly examining the code of the converted file. Below a snippet of the code shows the absence of information about signs in the converted file, as can be seen under the Signal tag at the end of the snippet.

```
<road name="Corso Verona" length="44.32704031" id="1391" junction="-
1">
        <link>
            <predecessor elementType="junction" elementId="3"/>
            <successor elementType="junction" elementId="1"/>
        </link>
        <type s="0" type="town"/>
        <planView>
            <geometry         s="0.00000000"         x="433.71754369"
y="427.84956072" hdg="-1.01856079" length="44.32704031">
                <line/>
            </geometry>
        </planView>
        <elevationProfile>
            <elevation s="0" a="0.00" b="0" c="0" d="0"/>
        </elevationProfile>
        <lateralProfile/>
        <lanes>
            <laneSection s="0">
                <center>
                    <lane id="0" type="none" level="true">
                        <link/>
                        <roadMark      sOffset="0"      type="solid"
weight="standard" color="standard" width="0.13"/>
                    </lane>
                </center>
                <right>
                    <lane id="-1" type="driving" level="true">
                        <link/>
                        <width  sOffset="0"  a="3.20"  b="0"  c="0"
d="0"/>
                        <roadMark      sOffset="0"      type="solid"
weight="standard" color="standard" width="0.13"/>
                        <speed sOffset="0" max="13.89"/>
                    </lane>
                 </right>
            </laneSection>
        </lanes>
        <objects/>
        <signals/>
    </road>
```

By further analysing and comparing the code with the corresponding osm file, it is possible to spot all the missing data. This allows to get a sense of the expected result on RoadRunner. Limitations of the SUMO netconvert tool had a significative impact on the overall result of the outlined workflow and possible alternatives were investigated during the initial phase of the study. Unfortunately, there are only few minor projects available on GitHub but none have given satisfactory results. Recently, CARLA team has started developing an internal conversion tool based on SUMO netconvert but for now results does not differ much from the ones obtained with SUMO [62].

5.2 RoadRunner as main choice

Scenario generation for autonomous driving requires significant effort to develop high-fidelity simulations. In this case, where automatic scenario generation is sought after, an editor capable of correctly parsing map data files to build the corresponding scenario was necessary. At the beginning of this work, it was decided to use only open source software, and in fact initial tests were conducted with the StreetMap plugin. A custom version of the plugin was also developed with minor changes to the source code. However, maps generated via plugin were incomplete and required too much manual intervention every time. Besides, several crashes and incompatibility issues with ROS bridge occurred. For these reasons, it was eventually decided to use RoadRunner thanks to the partnership between MathWorks and TeoresiGroup company. The use of RoadRunner allowed to come under the objective of automatically generating a 3D scenario. The only manual intervention required is simply related to the customization of the map through the addition of props such as buildings. The strength of RoadRunner is that maps generated in such a way will be perfectly compatible with all the functionalities provided by the Python module. To be more specific, world and actor settings and blueprints are specifically created to properly work with CARLA. By correctly matching these files, it is possible to avoid issues related to the Python API and consequently it is not necessary to change the code every time for each generated map.

5.3 What the generated maps look like

As seen above, maps imported in CARLA are generated by converting OpenDRIVE files in roads through the built-in option in RoadRunner. Road generation supports signals importation. Due to the loss of information about signals derived from SUMO netconvert tool, all road networks imported in RoadRunner did not contain any signals. However, the editor offered the possibility to automatically signalize junctions. Ideally, RoadRunner would then be capable of generating satisfying scenarios without intervention if a good OpenDRIVE file is used. Leaving aside these limitations, all imported maps were fully driveable, and autopilot worked properly. Maps were also tested with generated traffic to simulate a more complex scenario. In addition, ROS bridge had no issues with maps generated from RoadRunner. The editor proved to be the best choice and it is possible to assess its potential by looking at CARLA official maps which were created through it. In the end, the major limitation results from the converted OpenDRIVE which affects the final results.

5.4 Generated HD maps

The methodology presented in this thesis specifically focuses on HD map generation compatible with Autoware. The latest releases (Autoware v1.13 and v1.14) currently support Vector map and lanelet2 as official formats. The Autoware Foundation will fully migrate from Vector map format to lanelet2 in order to avoid relying on the proprietary map format of Aisan Technology company. In spite of this, it was preferred to still include in this work a possible method for generating a Vector map for a simple reason, older versions (v1.12 and older), which do not support lanelet2, are still widely used. As previously stated in Chapter III, the process for Vector map generation consists of two steps to obtain the desired map format starting from the OpenDRIVE file related to the imported map in CARLA. Conversion from OpenDRIVE to lanelet2 relies on ASSURE mapping tools. The developer behind these tools is an active contributor of the Autoware community and he is currently a PhD student at Nagoya University, the place where the Autoware Foundation started. While interesting and useful, these tools are still in early development and they are not exempt from bugs. Nevertheless, lanelet2 maps obtained from these tools are specifically generated to be compatible with Autoware. Maps generated in such a way slightly differ from the default format and contain specific tags which are mandatory for Autoware. It is possible to have a better understanding about the lanelet2 structure specific for Autoware by looking at the documentation available on GitHub [63]. The operation to obtain the lanelet2 file was quite simple and immediate. In this phase, it is possible to choose between the UTM and MGRS coordinate system, therefore it is important to pay attention to this point to avoid a possible mismatch between pcd and lanelet2 map. In this case, matching between the two types of map is easy to achieve through the tf.launch file since for every map imported into CARLA was chosen the point (0,0,0) as the world origin. All tests were conducted using only the OpenDRIVE files generated with SUMO netconvert or RoadRunner, which did not contain any traffic signs. More tests would be needed to see how the tool handles the conversion when using OpenDRIVE files containing more information. However, it was possible to use the code of CARLA default maps, as reference, to find out that information about traffic signs was missing in the lanelet2 file whereas the OpenDRIVE file contained it. Checks on these files showed that the tool is able to only generate simple road networks through conversion and manual editing is needed if additional details in the map are required. Conversion from lanelet2 to Vector map was the last step. The conversion tool as part of the Autoware utilities is simple to use but, as said before, it is not 100% accurate and part of the information is lost with the conversion process. It is possible to get an idea of the missing features by looking at the script **create_feature.cpp**, where when a certain feature is not supported, a comment like the following one is shown:

```
“missing.attribute” = 0;  // Not supported
```

In the generated .csv file these missing features will display a value equal to 0. If necessary, it is still possible to use the Vector Map Builder tool to improve the resulting map. In any case, it is recommended to use the lanelet2 format, if possible, for two simple reasons: it is an open source format, whereas the Vector map format is proprietary and no clear documentation is available, and by avoiding a further conversion process it is possible to maintain more information.

5.5 Analysis of the final simulation

After generating the HD map, it was finally possible to test the bridge between CARLA and Autoware. Before launching the bridge, a few changes in the code of the bridge files were made. Once made all necessary preparations, the bridge was successfully launched and the generated map was tested. The correct functioning of the simulation was assessed by trying the autopilot mode and the **2D Nav Goal** function on Rviz. The simulation performance can be affected by the map derived from the OpenDRIVE. It is suggested to intervene in the map creation on RoadRunner, if necessary, by fixing those areas of the map afflicted by geometric anomalies which can make things difficult for the autopilot. As mentioned above, possible issues are mainly related to junctions or road ends which do not present U turns, so when the vehicle approaches a critical zone a crash could occurs. Regardless, the entire methodology proved to give good results and currently it represents a valid alternative to classical methods which rely on sensor fusion and manual editing.

Chapter VI

6 Conclusion

The goal of this thesis work was to define a methodology to automatically build 3D scenarios generated from available map data of the real world as means of implementing them in an autonomous driving framework. It was decided to rely on open source software to accomplish the task with the only exception of RoadRunner, which was implemented in the workflow thanks to the collaboration between TeoresiGroup and MathWorks. The entire process consisted of two main phases. The first phase focused on generating a 3D scene by importing an existing road network previously extracted from the OpenStreetMap database. In the second phase, the OpenDRIVE file, which contains semantic road information, was converted in the lanelet2 and Vector map formats to exploit them as HD map in simulations. The indicated methodology points out all the necessary tools and software needed to reach the desired goal. The strength of this methodology is that the development of each software takes into account the progress of the others to keep up with them. This leads to a simple integration of these software within the workflow. Moreover, the adoption of open source formats, which are now de facto standards, simplifies research and development, especially thanks to available clear documentation. Results of this work are promising since having the possibility to automatically generate a 3D scenario without resorting to a demanding manual editing can be considered an important achievement. The outlined methodology has large room for improvement, but it already represents a valid means for speeding up the development of autonomous driving platforms.

6.1 Future work

This thesis work has shown how it is possible to build maps to support autonomous driving by exploiting open source map data. Although OpenStreetMap has proven to be a valid solution for map generation, the use of SUMO **netconvert** tool had an impact on the outcome. The next step to improve the proposed methodology would be to work on the source code of SUMO in order to overcome the limitations imposed by the **netconvert** tool. Moreover, it would be beneficial to monitor developments in the Autoware support for lanelet2 since the team is still working on it. This must be kept in mind since in the future it may be necessary to make some changes to ASSURE mapping tools. Eventually, after solving all the issues related to SUMO, a field test with a real car will be considered to move on the next step of the development of an autonomous driving system.

References

[1] N. Kalra and S. Paddock, *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?*: RAND Corporation, 2016.

[2] *CARLA democratizes autonomous vehicle R&D with free open-source simulator - Unreal Engine.* Accessed: Jun. 7 2020. [Online]. Available: https://www.unrealengine.com/en-US/spotlights/carla-democratizes-autonomous-vehicle-r-d-with-free-open-source-simulator

[3] *Engineering Uber's Self-Driving Car Visualization Platform for the Web.* Accessed: Jun. 7 2020. [Online]. Available: https://eng.uber.com/atg-dataviz/

[4] *AVS Home.* Accessed: Sep. 17 2020. [Online]. Available: https://avs.auto/#/

[5] *Cruise.* Accessed: Sep. 17 2020. [Online]. Available: https://www.getcruise.com/

[6] *The most powerful real-time 3D creation platform - Unreal Engine.* Accessed: Sep. 17 2020. [Online]. Available: https://www.unrealengine.com/en-US/

[7] *GM's Cruise is preparing for a self-driving future in the cloud | VentureBeat.* Accessed: Jun. 7 2020. [Online]. Available: https://venturebeat.com/2019/04/20/gms-cruise-is-preparing-for-a-self-driving-future-in-the-cloud/

[8] *Worldview.* Accessed: Sep. 17 2020. [Online]. Available: https://webviz.io/worldview/#/

[9] *'Carcraft' is Waymo's virtual world for autonomous vehicle testing | Engadget.* Accessed: Jun. 7 2020. [Online]. Available: https://www.engadget.com/2017-08-23-waymo-virtual-world-carcraft.html?guccounter=1

[10] Z. Yang *et al.,* "SurfelGAN: Synthesizing Realistic Sensor Data for Autonomous Driving," 2020.

[11] *NVIDIA DRIVE - Autonomous Vehicle Development Platforms | NVIDIA Developer.* Accessed: Sep. 17 2020. [Online]. Available: https://developer.nvidia.com/drive

[12] *Driving Simulation | rFpro.* Accessed: Sep. 17 2020. [Online]. Available: http://www.rfpro.com/driving-simulation/

[13] *Simulink - Simulation and Model-Based Design - MATLAB & Simulink.* Accessed: Sep. 17 2020. [Online]. Available: https://www.mathworks.com/products/simulink.html

[14] *Cognata | Autonomous and ADAS Vehicles Simulation Software.* Accessed: Sep. 17 2020. [Online]. Available: https://www.cognata.com/

[15] *The spring of self-driving cars. - Towards Data Science.* Accessed: Jun. 7 2020. [Online]. Available: https://towardsdatascience.com/the-spring-of-self-driving-cars-e37f76c11586

[16] *Metamoto.* Accessed: Sep. 17 2020. [Online]. Available: https://www.metamoto.com/

[17] *Virtual Test Drive (VTD) – Complete Tool-Chain for Driving Simulation.* Accessed: Sep. 17 2020. [Online]. Available: https://www.mscsoftware.com/product/virtual-test-drive

[18] *OpenDRIVE - Home.* Accessed: Sep. 17 2020. [Online]. Available: http://www.opendrive.org/index.html

[19] A. Best, S. Narang, L. Pasqualin, D. Barber, and D. Manocha, "AutonoVi-Sim: Autonomous Vehicle Simulation Platform with Weather, Sensing, and Traffic Control," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Salt Lake City, UT, Jun. 2018 - Jun. 2018, pp. 1161–11618.

[20] *CarMaker | IPG Automotive.* Accessed: Sep. 17 2020. [Online]. Available: https://ipg-automotive.com/products-services/simulation-software/carmaker/

[21] *HD Maps for Autonomous Driving and Driver Assistance | HERE.* Accessed: Sep. 21 2020. [Online]. Available: https://www.here.com/platform/automotive-services/hd-maps

[22] *PreScan | TASS International.* Accessed: Sep. 17 2020. [Online]. Available: https://tass.plm.automation.siemens.com/prescan

[23] *Ansys VRXPERIENCE Driving Simulator | Ansys.* Accessed: Sep. 17 2020. [Online]. Available: https://www.ansys.com/products/systems/ansys-vrxperience/driving-simulator

[24] *SCANeR studio - AVSimulation.* Accessed: Sep. 17 2020. [Online]. Available: https://www.avsimulation.com/scanerstudio/

[25] *Deepdrive from Voyage - Push the state-of-the-art in self-driving.* Accessed: Sep. 17 2020. [Online]. Available: https://deepdrive.voyage.auto/

[26] *Home - Parallel Domain.* Accessed: Sep. 17 2020. [Online]. Available: https://paralleldomain.com/

[27] *GitHub - udacity/self-driving-car-sim: A self-driving car simulator built with Unity.* Accessed: Sep. 17 2020. [Online]. Available: https://github.com/udacity/self-driving-car-sim

[28] *Unity Real-Time Development Platform | 3D, 2D VR & AR Engine.* Accessed: Sep. 17 2020. [Online]. Available: https://unity.com/

[29] *GitHub - microsoft/AirSim: Open source simulator for autonomous vehicles built on Unreal Engine / Unity, from Microsoft AI & Research.*

Accessed: Sep. 17 2020. [Online]. Available: https://github.com/microsoft/AirSim

[30] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,” pp. 621–635, 2017.

[31] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An Open Urban Driving Simulator,” 2017.

[32] *GitHub - carla-simulator/carla: Open-source simulator for autonomous driving research.* Accessed: Sep. 17 2020. [Online]. Available: https://github.com/carla-simulator/carla

[33] *GitHub - lgsvl/simulator: A ROS/ROS2 Multi-robot Simulator for Autonomous Vehicles.* Accessed: Sep. 17 2020. [Online]. Available: https://github.com/lgsvl/simulator

[34] G. Rong *et al.,* “LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving,” 2020.

[35] M. Müller, V. Casser, J. Lahoud, N. Smith, and B. Ghanem, “Sim4CV: A Photo-Realistic Simulator for Computer Vision Applications,” *International Journal of Computer Vision*, vol. 126, no. 9, pp. 902–919, 2017, doi: 10.1007/s11263-018-1073-7.

[36] *Rethinking Maps for Self-Driving - Lyft Level 5 - Medium.* Accessed: Jun. 9 2020. [Online]. Available: https://medium.com/lyftlevel5/https-medium-com-lyftlevel5-rethinking-maps-for-self-driving-a147c24758d6

[37] J. Jiao, “Machine Learning Assisted High-Definition Map Creation,” in *Proceedings - International Computer Software and Applications Conference*, 2018, pp. 367–373.

[38] *The Golden Age of HD Mapping for Autonomous Driving.* Accessed: Jun. 9 2020. [Online]. Available: https://medium.com/syncedreview/the-golden-age-of-hd-mapping-for-autonomous-driving-b2a2ec4c11d

[39] M. Dupuis, M. Strobl, and H. Grezlikowski, “OpenDRIVE 2010 and Beyond-Status and Future of the de facto Standard for the Description of Road Networks,”

[40] F. Poggenhans *et al.,* “Lanelet2: A high-definition map framework for the future of automated driving,”

[41] *Apollo.* Accessed: Sep. 17 2020. [Online]. Available: https://apollo.auto/

[42] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, “An open approach to autonomous vehicles,” *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015, doi: 10.1109/MM.2015.133.

[43] *AISAN TECHNOLOGY.* Accessed: Sep. 18 2020. [Online]. Available: https://www.aisantec.co.jp/english/

[44] *Navigation Data Standard (NDS) - The worldwide standard for map data in automotive eco-systems.* Accessed: Oct. 30 2020. [Online]. Available: https://nds-association.org/

[45] M. Quigley *et al.,* “ROS: an open-source Robot Operating System,” [Online]. Available: http://stair.stanford.edu/

[46] C. Wilhelm Gran, “HD-Maps in Autonomous Driving,” 2019.

[47] *OpenStreetMap.* Accessed: Sep. 23 2020. [Online]. Available: https://www.openstreetmap.org/

[48] *BBBike extracts OpenStreetMap.* Accessed: Sep. 18 2020. [Online]. Available: https://extract.bbbike.org/

[49] *netconvert - SUMO Documentation.* Accessed: Aug. 19 2020. [Online]. Available: https://sumo.dlr.de/docs/netconvert.html #supported_network_formats

[50] P. A. Lopez *et al.,* “Microscopic Traffic Simulation using SUMO,” in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018, pp. 2575–2582.

[51] *OpenDRIVE - Downloads.* Accessed: Sep. 11 2020. [Online]. Available: http://www.opendrive.org/download.html

[52] *RoadRunner - MATLAB & Simulink.* Accessed: Sep. 17 2020. [Online]. Available: https://www.mathworks.com/products/roadrunner.html

[53] *GitHub - ue4plugins/StreetMap: Import OpenStreetMap data into Unreal Engine 4.* Accessed: Sep. 17 2020. [Online]. Available: https://github.com /ue4plugins/StreetMap

[54] *GitHub - carla-simulator/ros-bridge: ROS bridge for CARLA Simulator.* Accessed: Sep. 23 2020. [Online]. Available: https://github.com/carla-simulator/ros-bridge

[55] *pcl/tools at master \textperiodcentered GitHub.* Accessed: Sep. 17 2020. [Online]. Available: https://github.com/PointCloudLibrary/pcl/tree/master/tools

[56] *GitHub - hatem-darweesh/assuremappingtools: Desktop based tool for viewing, editing and saving road network maps for autonomous vehicle platforms such as Autoware.* Accessed: Sep. 17 2020. [Online]. Available: https://github.com/hatem-darweesh/assuremappingtools

[57] H. Darweesh *et al.,* “Open Source Integrated Planner for Autonomous Navigation in Highly Dynamic Environments,” *JRM*, vol. 29, no. 4, pp. 668–684, 2017, doi: 10.20965/jrm.2017.p0668.

[58] *utilities/lanelet_aisan_converter at master \textperiodcentered GitHub.* Accessed: Sep. 17 2020. [Online]. Available: https://github.com/Autoware-AI/utilities/tree/master/lanelet_aisan_converter

[59] *VectorMapBuilder.* Accessed: Sep. 17 2020. [Online]. Available: https://tools.tier4.jp/feature/vector_map_builder/

[60] *netedit - SUMO Documentation.* Accessed: Sep. 7 2020. [Online]. Available: https://sumo.dlr.de/docs/netedit.html

[61] *Add a new map - CARLA Simulator.* Accessed: Sep. 15 2020. [Online]. Available: https://carla.readthedocs.io/en/latest/tuto_A_add_map/
[62] *Generate maps with OpenStreetMap - CARLA Simulator.* Accessed: Oct. 16 2020. [Online]. Available: https://carla.readthedocs.io/en/latest/tuto_G_openstreetmap/
[63] *common/lanelet2_format_extension.md at master \textperiodcentered GitHub.* Accessed: Nov. 29 2020. [Online]. Available: https://github.com/Autoware-AI/common/blob/master/lanelet2_extension/docs/lanelet2_format_extension.md