Al Imam Muhammad ibn Saud Islamic University
College of Computer and Information System
Computer Science Department

CS492
Senior Project in Computer Science 1
Graduation Project Final Report

Autonomous vehicle using IoT-enabled AI technologies

By:

Anas Walid Omar Bajnaid (435021105)

Abdullah Sami Abdullah Alabadlah (435031568)

Yasser Abdalmohsen Ali Alboraidi (437017295)

Supervisor: Dr. Qaisar Abbas Muhammad Abbas

Instructor: Dr. Qaisar Abbas Muhammad Abbas

Final Report | Second Semester

APR 14, 2021 | Ramadan 2 , 1442H

Abstract

The world's autonomous vehicles in the field of mobility can not be stopped. However, the change from conventional vehicles to autonomous vehicles will not happen in a short time, instead, it will be a process that takes many years in which new independent features are gradually prepared and added to customer vehicles, and these autonomous functions are now called ADAS. (Advanced driver assistance systems). This project aims to collect the various functions of ADAS, to make the vehicle move freely on the highway independently but at the same time, the traffic rules and the requirements of legal regulations are followed, as well as ensuring the safety of vehicles and pedestrians on the road. For this project to be successful, the proposed approach must be followed, which is to implement a learning method and reinforce a step-by-step approach to policy to determine the appropriate job in all situations. Through this project, the regulatory framework will be explained for road safety and an understanding of ADAS functions and the possibility of developing them in the future. The algorithm will be tested using a dedicated simulator on large or small roads, chosen after several studies on the latest simulators of autonomous vehicles. Through the LIDAR data sent from the sensor in the car, it will be directed and controlled (one of the sensors in the autonomous vehicles). The vehicle's performance will be shown through the experience and evaluated using the simulator, and in this project as well, we aspire to develop vehicles that rely on artificial intelligence (AI) techniques to deal with the autonomous driving of vehicles and demonstrate their importance to humans and provide the best safe and comfortable solutions to the user at a low financial cost through the development of devices. Sensing and linking them to self-driving vehicles to facilitate driving and mobility, and methods are evaluated in different scenarios, and their performance is evaluated through the standards and criteria provided by CARLA, and the simulation platform includes sensors that are required and important in smart cities and can fully control autonomous vehicles in all fixed directions and Moving to create a way to control the car on the road to ensure the vehicle’s path is monitored based on the global positioning device inside the car, and the first wireless connection between the host vehicle and a group of target vehicles are monitored.

Abstract in Arabic

لا يمكن إيقاف التغيير الى المركبات المستقلة في العالم في مجال التنقل، ومع ذلك فإن التغيير من المركبات التقليدية إلى المركبات المستقلة لن يحدث في وقت قصير، وبدلاً من ذلك ستكون عملية تستغرق سنوات عديدة يتم فيها إعداد ميزات مستقلة جديدة تدريجياً وإضافتها إلى مركبات العملاء، وتسمى هذه الوظائف المستقلة (أنظمة مساعدة السائق المتقدمة). يهدف هذا المختلفة، لجعل السيارة تتحرك بحرية على الطريق السريع بشكل مستقل ولكن في نفس ADAS المشروع إلى جمع وظائف الوقت يتم اتباع قواعد المرور ومتطلبات اللوائح القانونية، وكذلك ضمان سلامة المركبات والمشاة على الطريق. لكي ينجح هذا المشروع، يجب اتباع النهج المقترح، وهو تنفيذ طريقة التعلم وتعزيز نهج خطوة بخطوة للسياسة لتحديد الوظيفة المناسبة في وإمكانية ADAS جميع المواقف. من خلال هذا المشروع، سيتم شرح الإطار التنظيمي للسلامة على الطرق وفهم وظائف تطويرها في المستقبل. سيتم اختبار الخوارزمية باستخدام جهاز محاكاة مخصص على الطرق الكبيرة أو الصغيرة، يتم اختياره المرسلة من المستشعر في LIDAR بعد عدة دراسات على أحدث أجهزة محاكاة المركبات ذاتية القيادة. من خلال بيانات السيارة، سيتم توجيهه والتحكم فيه (أحد أجهزة الاستشعار في المركبات ذاتية القيادة). سيتم عرض أداء السيارة من خلال التجربة وتقييمها باستخدام جهاز المحاكاة، وفي هذا المشروع أيضًا نطمح إلى تطوير مركبات تعتمد على تقنيات الذكاء للتعامل مع القيادة المستقلة للمركبات وإثبات أهميتها للبشر وتقديم أفضل الحلول الآمنة والمريحة للمستخدم (AI) الاصطناعي بتكلفة مالية منخفضة من خلال تطوير الأجهزة. استشعارها وربطها بالمركبات ذاتية القيادة لتسهيل القيادة والتنقل، ويتم تقييم ، وتشمل منصة CARLA الأساليب في سيناريوهات مختلفة، ويتم تقييم أدائها من خلال المعايير والمعايير التي توفرها المحاكاة أجهزة الاستشعار المطلوبة والمهمة في الذكاء. . المدن ويمكن التحكم الكامل في المركبات ذاتية القيادة في جميع الاتجاهات الثابتة والتحرك لإنشاء طريقة للتحكم في السيارة على الطريق لضمان مراقبة مسار السيارة بناءً على جهاز تحديد .المواقع العالمي داخل السيارة، وأول اتصال لاسلكي بين السيارة المضيفة ويتم رصد مجموعة من المركبات المستهدفة

Keywords

(CARLA) the name of simulation

(AI) artificial intelligence

(API) Application Programming Interface

(ADAS) Advanced Driver Assistance Systems

(RSS) Responsibility sensitive safety

(LDAR) Light Detection and Ranging

(GPs) Global Positioning System

Table of Contents

Contents

Table of Figures

List of tables

1 Introduction Chapter

1.1 Introduction

In 2050, the world's population in large cities will be approximately seven out of ten people. These cities represent more than 70% of global carbon emissions and 60% to 80% of energy consumption., frequent road accidents, and related health issues. Governments can build smart and sustainable cities for their citizens by using information and communication technologies and other important technologies. Smart cities are cities that were created to take advantage of information technology to improve life and civilized services provided to citizens and the ability to meet the needs of citizens in the future in all economic, social, cultural, and environmental aspects،although the cities in which all civil systems and services are connected do not exist yet, many of cities are on their way to becoming sustainable and smart cities. And it relies more on information and communication technology to enhance energy efficiency, waste management, health care development, improve housing and traffic, alert police about crimes, and improve water and sanitation networks for each city. With the help of the Internet of Things (IoT) to communicate with each other's computer devices and exchange data at high speed, including sensors and software, it enables many devices and objects equipped with smart sensors to communicate with each other, collect information in real-time, and send this data, via wireless communication,

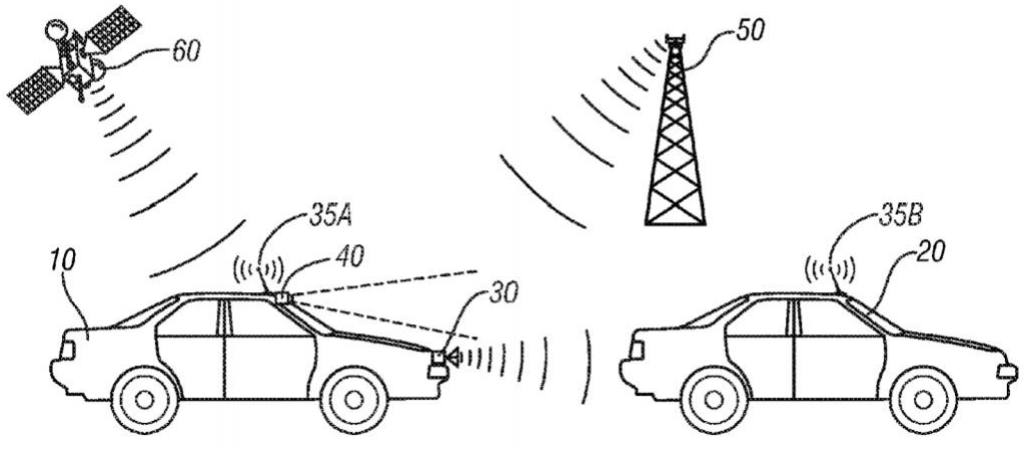

Figure 1: collect information in real-time, and send this data, via wireless communication [2]

To central control systems. These, in turn, manage traffic, reduce energy use, and improve a wide range of urban processes and services. Artificial intelligence allows extremely large data sets to be analyzed mathematically to reveal patterns that are used to inform and enhance the decision-making process in smart cities. By using the CARLA simulator, which is responsible for all the simulations used to provide physical sensors, update the global situation, actors, etc., because it aims to achieve positive and realistic results, it is better to run the server with a powerful GPU dedicated to this, especially when using machine learning, and the client consists Units that control the actors inside the emulator and set global conditions, and this is achieved by using the CARLA simulator and developed in the Python language,

rendering, computation of physics, updates on the world-state and its actors and much more. As it aims for realistic results, the best fit would be running the server with a dedicated GPU, especially when dealing with machine learning. The client side consists of a sum

of client modules controlling the logic of actors on scene and setting world conditions. This is achieved by leveraging the CARLA API in (Python), a layer that mediates between server and client that is constantly evolving to provide new functionalities.

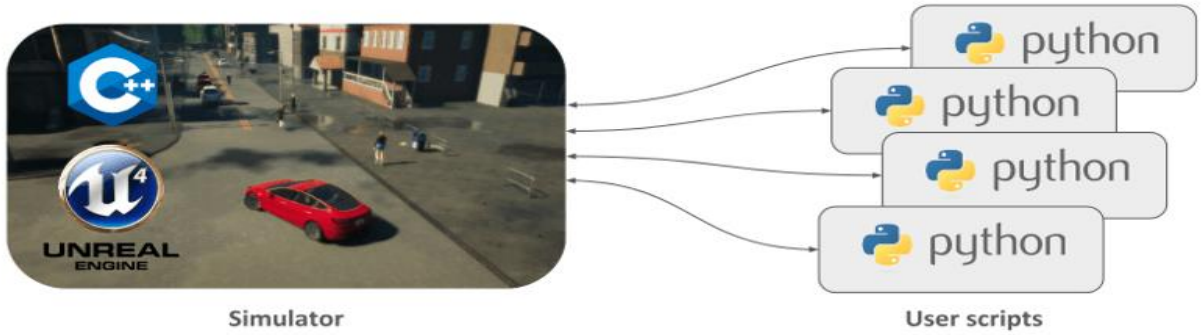

Figure 2: CARLA API in (Python),

This is a summary of the structure of the simulator, and CARLA has many different features and elements, and some of these features are mentioned below to obtain the capabilities and advantages that CARLA provides and what it can achieve.

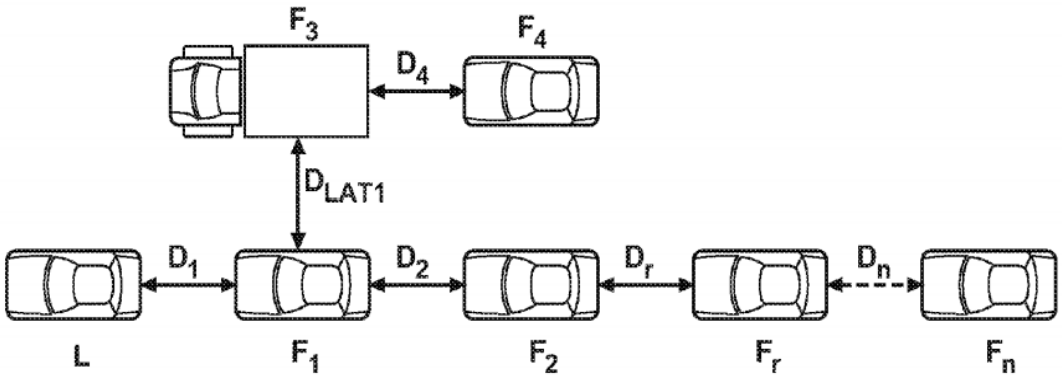

Figure 3: Calculate the distance between vehicles [8]

In this project, we aspire to develop a city that relies on artificial intelligence (AI) technologies to deal with self-driving vehicles and their importance to humans and provide the best solutions with easy effort and reduce the financial cost through the development of sensors. And linking them with self-driving vehicles to facilitate transportation, choose the best roads, and avoid congestion and dangerous problems.

1.2 Problem Definition

Various methodological techniques have been developed to discover and predict the best paths for autonomous vehicles through advanced sensor processing, machine learning, and artificial intelligence algorithms. The purpose of this project is to first investigate basic smart algorithms that are being applied to discover the best approaches for autonomous and autonomous vehicles. Then, using advanced technologies to accurately detect traffic congestion, choose the appropriate road, and have less time while driving. Although many survey articles have been written to address this issue, according to our limited knowledge, none of them have built a hybrid system with cameras and sensors to study the road and provide the best solutions. There is a recent investigative article to detect and predict accidents before they happen, but that study indicated that there are still many improvements required in this area. Accordingly, in this project, we will also develop an integrated e-sharing technology for predicting best methods and incidents before they happen that supports AI and the Internet of Things. "E-Share: Autonomous share public transit using IoT-enabled AI technologies.

1.3 Aims and Objectives

Aim

The aim of this project is to design, implement and test autonomous system in the CARLA simulator to share autonomous vehicles on the same road, calculate the distance between cars, increase safety and reduce congestion through artificial intelligence and augmented learning algorithms for autonomous vehicles, we will use the Internet of Things - based on technologies, and sensors to detect conditions. Environmental independent. We use CARLA to study the performance of three autonomous driving methods: a classic modular pipeline, an end-to-end model trained through simulated learning, and a comprehensive model trained by reinforcement learning. The methods are evaluated in controlled scenarios, and their performance is examined through the metrics provided by CARLA, in addition a passenger can request a car.

The objectives

1- Training on Carla simulator and studying the understudying technologies

2- To develop a smart simulator, incorporate artificial intelligence (AI) technologies to handle tourism with low-cost solutions by studying autonomous cars.

3- To develop a completely autonomous solutions in a CARLA simulator by E-share autonomous vehicles on the same route It will help to build a smart city by decreasing traffic congestion and money spend to reach to the destination.

4- To build a set of algorithms that can make smart decisions based on machine algorithms for predicting the best and profitable route by the autonomous vehicle. To complete this E-share system**.**

5- To different IoT-based technologies will be used to detect autonomous environmental conditions.

6- To develop our technical writing skills by writing a technical report for our project.

1.4 Methodology

We will develop a self-driving vehicle linked with sensors with help of Carla simulator to overcome some of the problems resulting from traffic congestion and avoid accidents to save the lives of others and choose the best way to reduce time and exert effort, and the Carla system provides open-source systems in Building smart cities and sensor systems and linking them to autonomous vehicles, for learning, training, planning, and design, it can be used freely.

The simulation platform also supports all the sensors needed in smart cities and has full control in all static and dynamic directions in creating maps, Using the RSS library, implements a mathematical model for safety assurance. It receives sensor information and provides restrictions to the controllers of a vehicle. To sum up, the RSS module uses the sensor data to define situations. A situation describes the state of the ego vehicle with an element of the environment. For each situation, safety checks are made, and a proper response is calculated. The overall response is the result of all the combined. For specific information on the library, you can create very complex choreographies," This will be of great help in the field of human-machine interaction, for example, to help improve the field of prediction. Prediction is one of the most important features. He must be able to anticipate what some pedestrian or nearby car will do in the next few seconds, will he cross or stay on the sidewalk? So, you must have some very advanced sensors to avoid accidents.

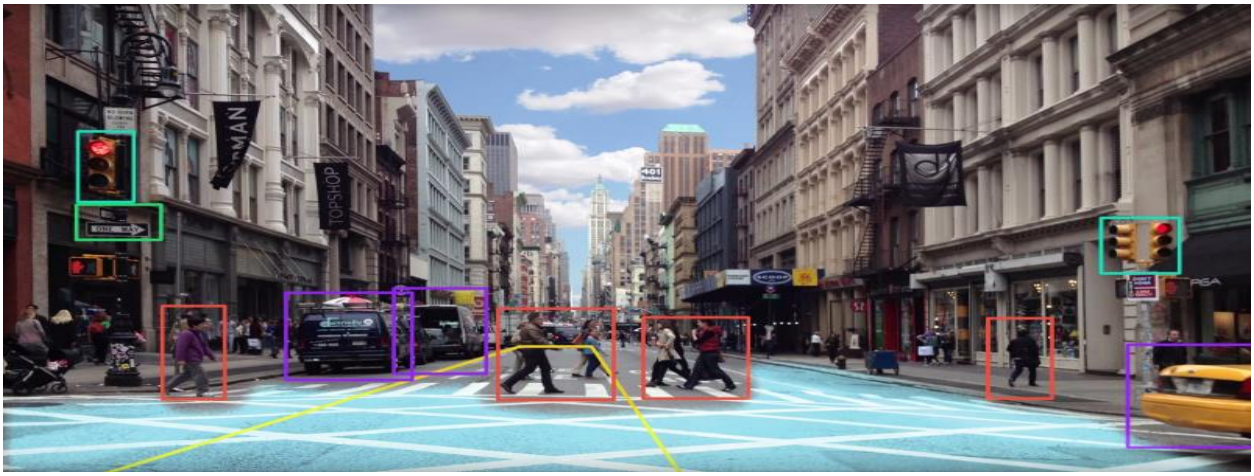

Figure 4: Distinguishing objects in the autonomous vehicle system

The RSS sensor implements a mathematical model for safety assurance. It receives sensor information and provides restrictions to the controllers of a vehicle. To sum up, the RSS module uses the sensor data to define situations. For each situation, safety checks are made, and a proper response is calculated.

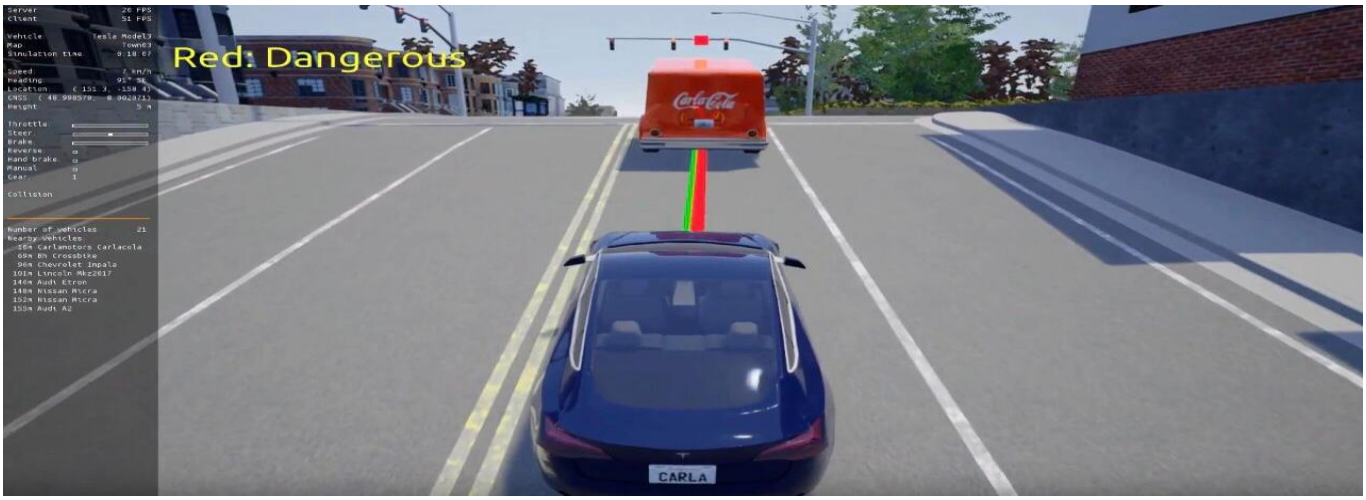

Figure 5:RSS sensor on implementation

1.5 Team Qualifications

- Abdullah Alabadlah **:**

  Java programming language , A+, Network+, Cybersecurity

- Anas Bajunaid **:**

  Java programming language, C++ , CCNA Corus

- Yasser Alboraidi**:**

  Java programming language, python, SQL

1.6 Conclusion

The project had the main objective of making an Autonomous Vehicle navigate on a highway by combining artificial intelligence and the Internet of things functions using a CARLA simulator. After the execution of this project, the objective has been achieved has been possible by the specific objectives set at the beginning. To have a better understanding of how to develop vehicle's control systems, the Regulatory Framework has been explained we made a review about CARLA simulator and automated functions. The first step was to find a simulator that fit with our project. It could be seen that there exist several simulators depending on their purpose; from professionals use simulators, to educational. After a comparison between ten of these simulators, the CARLA simulator was found to be the best for the project highway simulator, can add more vehicles on the road and all sensors can be implemented, to facilitate future implementations, Finally, once all the sub-objectives were achieved, this project will provide some appropriate solutions to traffic congestion problems, reduce accidents, and choose the best routes through the autonomous Vehicle associated with the required sensors.

2 Literature Review Chapter

2.1 Introduction

The technologies have developed systems with the help of software and hardware technologies to discover the best way around by sharing information with other vehicles and predict accidents before they happen. These systems were based on information processing provided by advanced sensors and machine learning technologies. The information is reviewed using systems linked to the sensors and they study the road and provide the best information using artificial intelligence. Also, another way to spot accidents is by applying an RSS library to a mathematical model to ensure safety. It receives sensor information and provides restrictions for vehicle control units. In short, the RSS unit uses sensor data to identify situations. The position describes the state of the ego vehicle with the environment component. For each case, safety checks are performed, and an appropriate response is calculated. Several studies including recent studies have been thoroughly examined and the characteristics of the proposed methods used for classification along with special classifiers, limitations, and advantages (shortcomings), the accuracy of the methods as well as their experimental preparation.

2.2 Background

CARLA is a simulator created to develop open-source autonomous vehicles [1]. It was created as a programming interface to address the range of tasks involved in autonomous driving problems. One of CARLA's main goals is to assist in research and development in the field of autonomous driving, as it acts as a tool that developers can easily attract and customize to work on. The simulator should provide requirements for different uses within self-driving problems in different ways (for example, educational leadership policies, training perception algorithms). CARLA relies on a simulation engine and uses the Open DRIVE (1.4) standard to define paths and cultural settings. The simulation is controlled by an API which is handled in Python [2] and C ++ which is implemented as the project does. Many algorithms for autonomous vehicles have been developed in previous studies to be able to predict before accidents happen on congested roads, and this allows the development of future systems for autonomous vehicles. Traditional autonomous systems such as fuzzy logic, statistical models, and decision trees as well as modern deep learning algorithms are widely fired in automated systems in advance of an accident prediction. Therefore, these algorithms are carefully chosen to present and overcome these systems. In this project report, [3] the Responsibility Sensitive Safety Model (RSS) was implemented as a safety that could be developed for autonomous vehicle behaviors such as safe distance. However, dealing with poor and changing situations and uncertainty may significantly reduce vehicle durability, and in some cases safety cannot be guaranteed. It identifies complications from vehicle conditions, road engineering, and environmental standards. Especially difficult situations occur if these algorithms are changed during road accident avoidance maneuverings such as severe braking and sudden cornering. As part of our analysis, we modified the distance equation in the RSS sensor to calculate safe distances that ensure that potential highway collisions do not occur through braking and speed reduction.

2.3 Related Work

The daily routine that people encounter on the roads while on the go has become a problem with every passing day, as people get late and face accidents. [4] The self-driving car model aims to solve these problems by keeping people away from the problems of delay, congestion, and accidents so that they do not have to drive anymore, and the risks of accidents and delays can be reduced, and traffic congestion can be reduced to the lowest possible degree. Others, spotting obstacles, taking sharp bends and turns, tracking traffic.

Some research focuses on the high and tactical [5] level for making decisions related to autonomous vehicles in a complex and dynamic urban environment with PreScan,.[6] PreScan is a simulation environment for developing Advanced Driver Assistance Systems (ADAS) and Intelligent Vehicle Systems (IV). It is a platform that can be used to build 3D traffic. And the creation of vehicles, pedestrians, traffic lights, and other control units. It also provides an interface for Simulink to allow users to easily control and operate the simulation. PreScan comes with a powerful graphics processor, an advanced 3D visual display, and standard MATLAB / Simulink connectivity. It consists of various main units. An array of sensors where the readings of multiple sensors are simulated and captured; Multiple sides can be photographed in one scene, and an integrated view can be viewed and viewed on a moving camera, with dynamics and control models simulated to gather.

By delving into previous research and studies, deep reinforcement [7] learning has led us to newer capabilities in solving complex control and mobility-related tasks. The research provides several solutions through the autonomous navigation system to enhance deep learning and avoid the obstacles of self-driving cars, which was applied with the [8] Deep Q Network to simulate a car in an advanced environment. Two types of sensor data were used: the camera sensor and the laser sensor located in front of the vehicle. It's also a cost-effective high-speed prototype car that can run the same algorithm in real-time. Also, the design uses a Hokuyo Lidar Camera and a sensor on the front of the vehicle.[9] The integrated graphics processing unit (Nvidia-TX2) is used to run deep learning algorithms based on the input of the sensors

And we have noticed through real-life that with traffic [10] congestion some unexpected accidents happen. We also encountered these problems with vehicles. A previous study of some solutions to this problem was presented. To avoid these accidents by providing a means to detect vehicles in the surroundings of the autonomous vehicle and issuing alerts to avoid a potential collision. With her on the road, through two stages. It is implemented in the Android Operating System (ROS). The first part is used to discover vehicles located near an autonomous vehicle. The YOLO v2 algorithm, which was developed on a set of newly created images, was used. The YOLO v2 algorithm is designed to detect three classes of objects: car, truck, and bus. The second part of the proposed method is a ROS node to evaluate the distance to avoid collision by creating two ROS nodes to evaluate distance; ROS node is used to evaluate distance in Carla simulator, other ROS node is used to evaluate the real-world distance. And display the results presented from the previous two parts.

there was another study on avoiding traffic congestion [11], which is based on detecting traffic signals and recognizing them through the camera installed in the car and to increase safe driving on the road, the camera plays an essential role in automatic detection and recognition of traffic lights (lights) in smart vehicles using assistive systems Using [12] advanced driver (ADAS). However, detecting and recognizing traffic lights is a challenging challenge due to their small size and colors (red, yellow, green) which may be similar to other things, lighting diversity, environmental conditions, etc. With the study and proposed work by FBL., [13] They detected traffic lights using a region-based convolutional neural network (R-CNN) and recognized traffic lights based on forked learning from Grassmann. They also used the learning transfer on the VGG16 to extract features from the detected traffic lights and use these features to create subspaces for each traffic signal (red, yellow, and green). These subspaces are located in the Grassmann manifold and include variations in different states of the same traffic light expected during detection.

Traffic congestion is a serious global problem, and to avoid this in the future [14], a study in 2017 introduced Enhanced or Deep Learning Algorithms in the Traffic Simulation Study. And he relied on improved learning on a repetitive search algorithm, and the main goal of the algorithm was to find optimal vehicle lanes and avoid traffic congestion. The Deep Q Learning algorithm was able to learn appropriate strategies, taking into account the dynamic and sudden changes in traffic at the intersection of highways. In particular, the target network of Q learning has created a mobile system that is able to make this smart method based on reinforcement learning an effective tool for improving vehicle tracks, including the autonomous driving system.

And the problem of self-driving car control and its failure to deviate [15] from the lane and collision with other cars driven by humans and sharing the road with humans without any problems in the future was posted. A previous study provided a solution to this problem, which are pre-emptive collision avoidance algorithms that take into account the intention of drivers of vehicles the advanced algorithm uses multi-stage Gaussian operations (GPs) to determine the transmission model for each vehicle by studying the driver's intent. It also updates its lane predictions online to provide a collision avoidance-based lane prediction that adapts to different driving styles, whether the vehicle is driven by a human or non-human being, road and weather conditions. The effectiveness of this concept is evidenced by a variety of simulations that use data derived from real human driving in various scenarios including highway intersection. Experiments are conducted through sophisticated driving simulations and highly realistic simulations of third-party vehicles

2.4 Conclusion

After completing this section of our project, we conclude that the vehicle car can solve some problems such as intensifying congestion and reduce the risk of traffic accidents, and on the other hand some research focuses on making decisions related to autonomous vehicles in a civilized and dynamic environment, and we have touched PreScan, which is one Driver development and assistance systems and intelligent vehicle systems, and among the proposed methods is the ROS node used to assess the distance in the CARLA simulator, and in the next Chapter we will complete customer requests and create program design and choose a set of requirements that can be verified once the program is built

3 System Analysis Chapter

3.1 Introduction

We conclude that the vehicle car can solve some problems such as intensifying congestion and reduce the risk of traffic accidents, and on the other hand, some research focuses on making decisions related to autonomous vehicles in a civilized and dynamic environment, and we have touched on PreScan, which is one Driver development and assistance systems and intelligent vehicle systems, and among the proposed methods is the ROS node used to assess the distance in the CARLA simulator, and in the next Chapter we will complete customer requests and create program design and choose a set of requirements that can be verified once the program is built , In this chapter, we include a description of the requirements Specifications such as (Specific Requirements, User Requirements, System Requirements), User Characteristics, and Gantt chart.

3.2 Software Requirements Specification

We will collect the required data for A Computerized System, there are different methods for gathering the requirements for the system, we will start by conducting a requirements analysis process to collect different features that need to be included in the developed computer system.

3.2.1 User Characteristics

In general, our users are of all ages. They must know how to deal with the computer and can understand the way the program works. Often the user needs special knowledge or experience in some of the programming languages to be able to design and develop better. So, our software can be used by computer students, developers, and programmers. However, we expect most of our young users. At the corporate level, our users are public transport development companies

3.2.2 Specific Requirements

This section is the most important section of the system, which explains all functional and non-functional requirements. This section gives us a detailed description of the system.

3.2.2.1 External Interface Requirements

This section provides a detailed description of all inputs and outputs from the system. It also provides hardware descriptions, software interfaces and connections, and provides basic user interface models

3.2.2.2 User interfaces

To access the Car system the user must have a key. This key will connect the user to a main frame containing information about the Car as well as the login screen through which he can access the system Not everyone can access the system only who has an account -key that can access the system there are a group of users who can access the system and deal with each user who has the power to interact with certain interfaces

3.2.2.3 Hardware interfaces

- 64-bit Windows/Linux/MacOS
- 8-GB RAM or more
- Quad core Intel or AMD processor, 2.5 GHz or faster
- NVIDIA GeForce 470 GTX or AMD Radeon 6870 HD series card or higher
- 10GB of hard drive space for the simulator setup

3.2.2.4 Software interfaces

To run the interface, following software are required.

- Model Predictive Control Toolbox
- Simulink Control Design
- Carla 2.0
- Python 3.7
- ROS
- RSS

3.2.2.5 Communications interfaces

Communication between different parts of the system is important since they depend on each other. However, the way the connection is achieved is not important to the system and is therefore handled by the core operating systems via command line. Also, a good internet

3.2.3 User Requirements

3.2.3.1 Functional requirements

This section includes the requirements that specify all the fundamental actions of the software system

- Feature: Admin -login
  Scenario: When the system administrator to enter the system must use the user's password and password of the administrator “key”
  because he has all the powers that help him to manage the system in the way required

- Feature: Driver - Record Saved data
  Scenario: The driver logs on to the system and goes to the Destinations data recording screen It then He can choose one of the previously saved places like home, work

- Feature: Driver - login to the system
  Scenario: The driver can access the system then the driver can do some operations on the system

- Feature: driver – Estimate distance
  Scenario: when the user goes at high speed and the vehicle nearly another vehicle automatically the system decreases the speed

- Feature: driver – Localization Stage.
  Scenario: When the user drives on the highway he turns on the Localization button that does select the best path.

- Feature: driver – auto pilot
  Scenario: When the user wants to take a rest, he activates through the control panel the auto-driving mood.

3.2.3.2 Non-Functional Requirements

- The search feature should be prominent and easy to find for the user so that the user can find the search feature easily.
- Different search options must be clear, simple, and easy to understand. so that the user can perform a search easily.
- The results displayed in the list view should be easy to use and easy to understand. Selecting an item in the results list should only take one click. For the user to use the list view easily.
- The requested car must be the nearest.
- The simulator should give accurate results.

3.2.4 System Requirements

3.2.4.1 Functional requirements

- The simulator uses the ROS sensor it detects vehicles and calculates the distance between vehicles

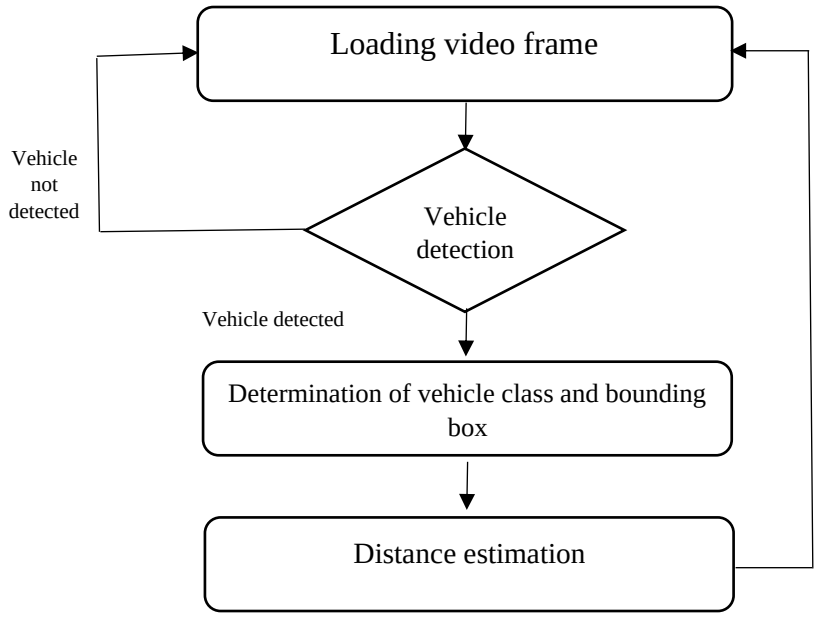

Figure 6: the diagram of ROS sensor

- The simulator uses the RSS sensor, when the vehicle approaches another vehicle, the sensor sends a signal to the system and the system reduces the speed

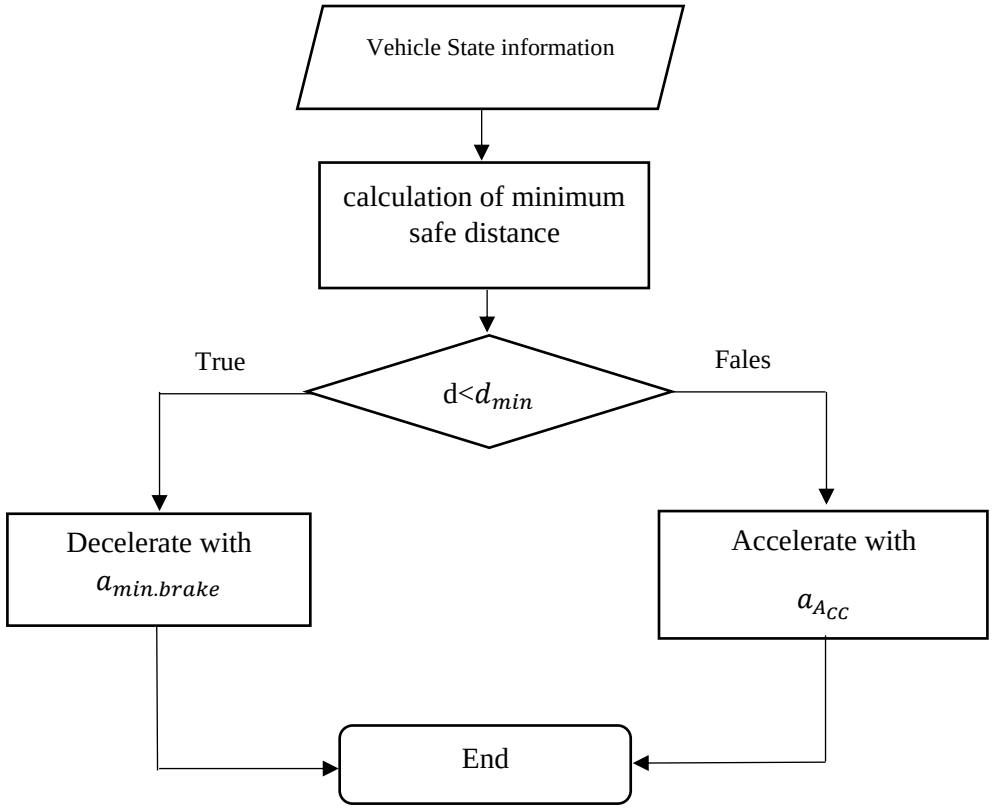

Figure 7: the diagram of RSS sensor

- The simulator should be developed by High-level programming language using " python'
- The simulator should build by using some CARLA libraries.

3.2.4.2 Non-functional Requirement

- Performance: autonomous cars driving systems is built on fast platforms to respond in due rate time.

- Tolerance: the develop software will not have any errors because it will handle through exceptions

- Useability: Use of integrated graphics processing unit is used to run algorithms based on the input of the sensors

- Reliability: The system reads surrounding vehicles of the autonomous vehicle and issuing alerts to avoid a potential collision.

3.3 Project Management Plan

Figure 8: the timeline of project

3.4 Conclusion

In this chapter, we faced challenges in collecting data and determine the user's needs. In this chapter, we explain the main points of interaction between the system and the user. We have mentioned in detail the software and hardware requirements specifications. Then we have explained the analysis of functional and non-functional requirements.

4 Design Chapter

4.1 Introduction

Different dimension of CARLA simulator is going to be discussed in this section including system and database design along with modular decomposition. This section will help to understand the working infrastructure of CARLA simulator. As CARLA is being extensively used for the purpose of development, training and validation testing of autonomous driving systems. CARLA not only provides the open-source code but also urban maps and digital vehicles that can be used by developers freely and flexibly along with sensors, weather and environmental controls. Salient features of CARLA include scalability through multi-client-based algorithms, easy to customize traffic scenarios and map generation, etc.

4.2 System Architecture

The CARLA simulator is based upon easy-to-use client-server architecture. It should be noted that the server is solely responsible for each and everything about simulation including the sensor controls, physical parameters as well as map and weather states. It is recommended to use dedicated GPU for this process as it helps to generate more realistic results when using machine learning. On the other hand, the client side is the sum of different modules being used at the client end. Client is allowed to communicate with the server through CARLA API based upon Python or C++ modules. This API is constantly being modified to provide more functionality. Some of the features are described hereunder in order to create better understanding about the structure of CARLA simulator.

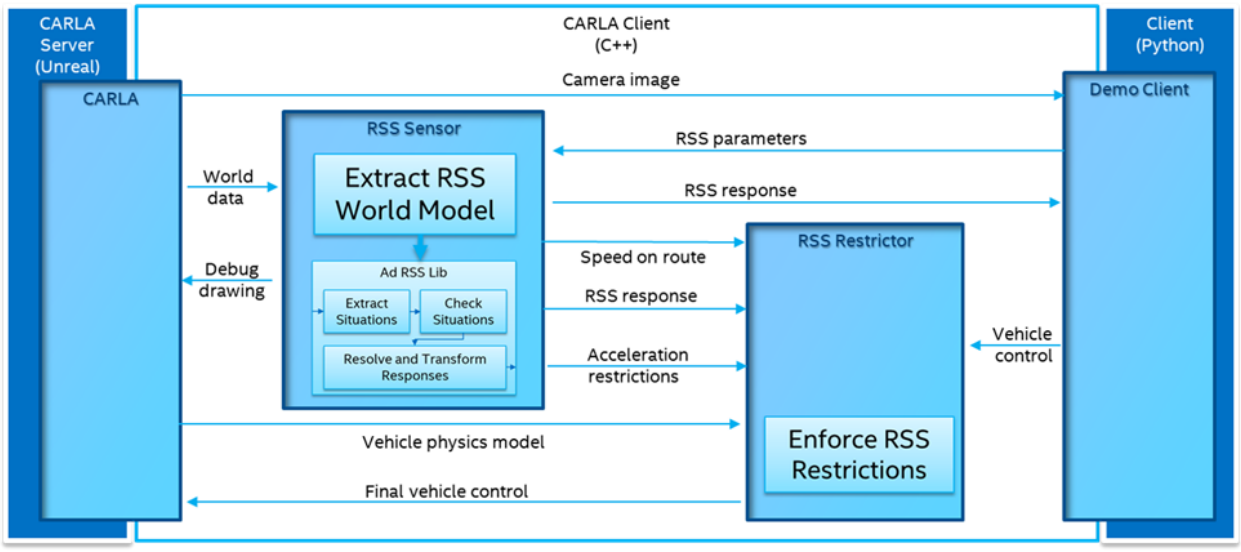

Figure 9: CARLA architecture

- **Traffic Scenario Generator** includes build-in functionality in order to control the vehicle being used for the purpose of learning along with other vehicles in traffic mode. This generate urban like traffic scenario.
- **Sensors** in CARLA include a range of sensors like Camera, Radar and Lidar, etc. These sensors help to fetch information from the environmental surrounding, store and retrieve that information from the database to improve the process.
- **Recording Tool** helps to record every incident in the simulation world, performed by different actors making the tracking possible.
- **ROS Bridge and Automation Implementation** using different integration protocols to communicate between different environments. These tools make simulation more interactive and responsive.
- **Free Resources** are provided by CARLA simulator like urban infrastructure, traffic, weather conditions, blueprints, etc. The simulator allows all of these controls to be customized being client friendly.
- **Scenario Runner** helps to improve the learning process of vehicles, the simulator introduces different built-in routes that help developer to test its model.

4.3 Database Design

Efficient working of CARLA simulator is dependent upon data retrieval in the form of raw data from sensors and processes data from computer. This document introduces set of entities that are collected as simulation data. By default, the simulation starts with standard settings of traffic, weather, etc. Ego vehicle roam around the city having basic sensors. On the basis of data recorded on these sensors new simulation can be generated, sensors can beaded and many more customizations can be performed. CARLA simulator entities include simulation, traffic, ego vehicle, basic sensors data (including camera and RGB sensors, etc.) and advanced sensors (depth camera, semantic segmentation camera, RADAR and LIDAR sensor). All these datasets are recorded, manipulated and retrieved for accurate simulation results. Map setting includes the setting related with area and weather as show below.

Figure 10: CARLA weather entity

Each entity has dataset of its attributes that depends upon the following information:

- Cameras
- Image size
- Number of cars used in each view/screen
- Number of pedestrians in each view/screen
- Lateral and Longitudinal noise level
- Weather set

To store the data of each episode a separate folder is generated, that contains json file with following dataset.

- Number of spawned pedestrians and vehicles
- Speed of pedestrians as well as vehicles, normally it stores the average speed assigned by CARLA simulator
- Weather data of episode

It should be noted here that sensor data is stored in PNG format and measurements are stored in json file format.

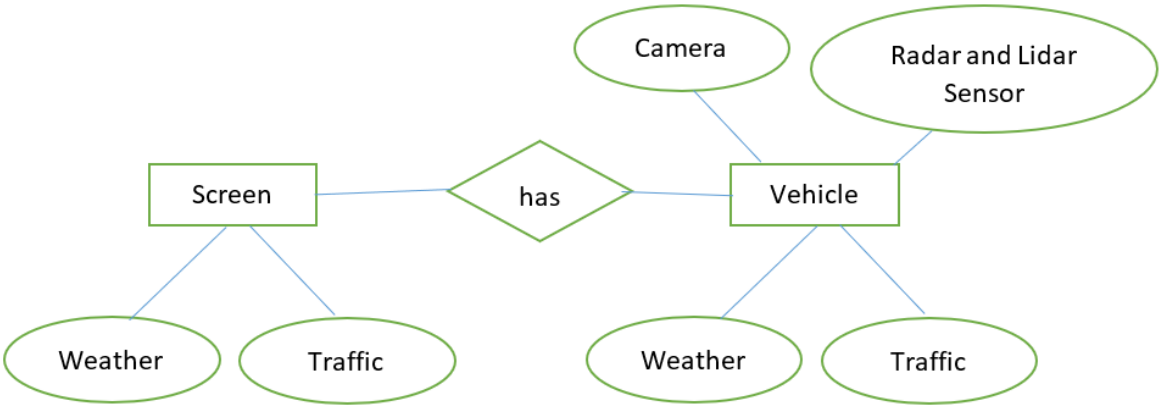

Figure 11: ER Diagram

4.4 Modular Decomposition

The following figure depicts different modules of CARLA simulator architecture that help to understand the working of the overall system. Each module is assigned a different task. These modules are set to predefined inputs according to which they perform their function and are interconnected through ROS interfaces. These allows the user to understand and interchange the modules with same function, input and output. On the other hand this modular approach allows to test and verify different settings and solutions of test environment.

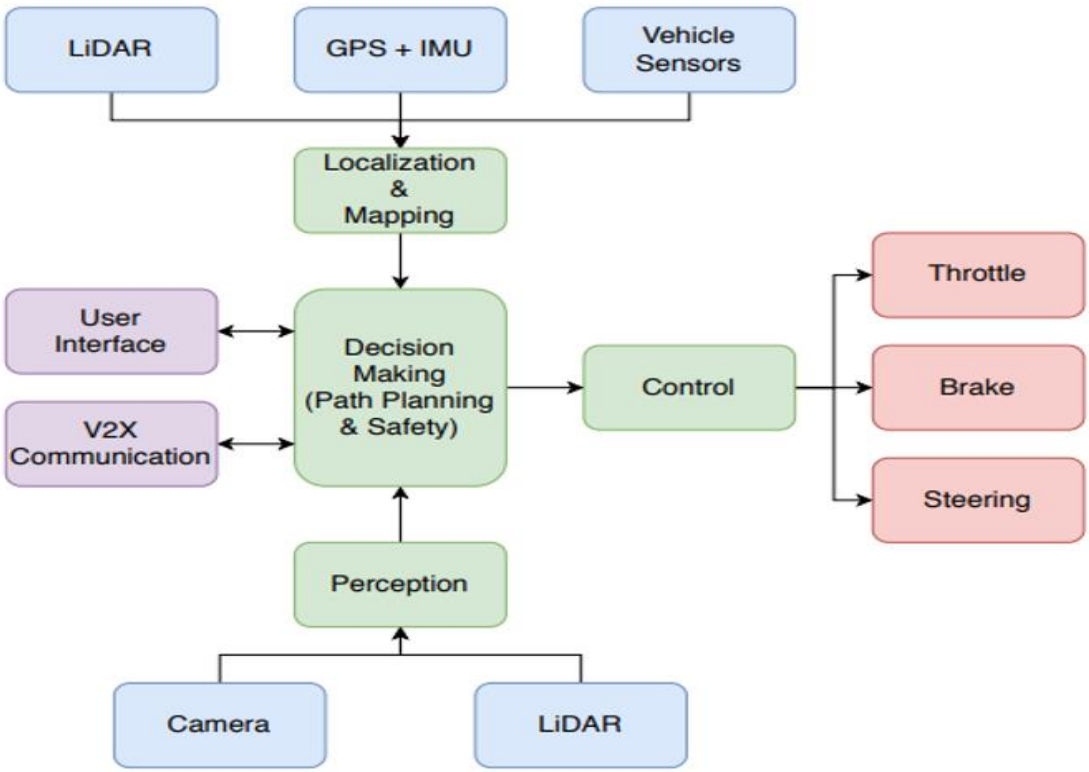

Figure 12: Decomposition of CARLA system in modular Components

According to above shown layout data flows from sensors and processor to different modules. These modules then control the vehicle in terms of steering, accelerating and braking. User can communicate with vehicle through Graphical User Interface, through which aimed destination can be achieved. Additionally, V2X module helps to communicate with other entities of the model.

4.5 System Organization

As mentioned previously CARLA provides autonomous driving solutions. Basically, it was built with modular approach and scalable APIs, in order to tackle problems associated with autonomous driving. Along with other features CARLA aims to support democratization of simulation process, being a tool that can be easily customized by the user. To achieve this goal CARLA simulator depends upon driving learning rules along with perception algorithm depending upon semantic segmentation. The structure of CARLA simulator is based upon Unreal Engine along with Open Drive that support various functionalities like roads, urban and weather settings. Simulation control is given to the user through API that are handled through python or C++ programming. Additionally, to support smooth driving, development and testing process CARLA continues to evolve becoming a huge and more interactive project build by the community of CARLA simulator. With the passage of time CARLA simulator is providing more options in order to be more efficient tool of autonomous driving and environment testing. All these extended features and modules are provided in open-source format. Simulator acts as white box with extended access to anybody letting them use these features and customize them accordingly.

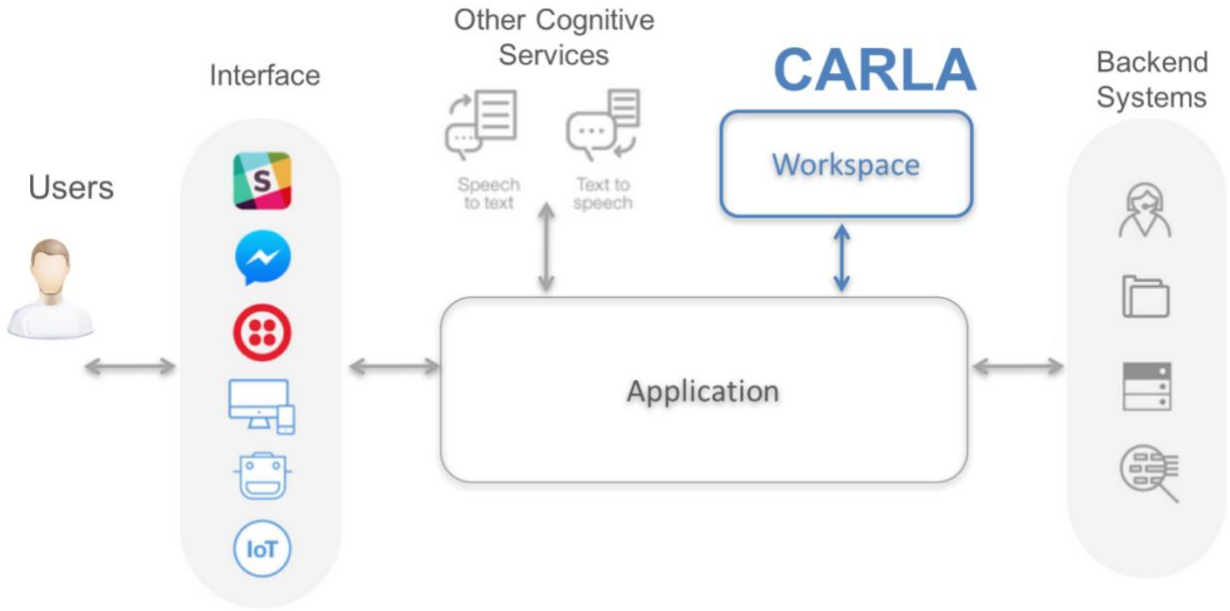

Figure 13: Organization

It should be noted here that autonomous driving in urban traffic environment is difficult task due to various reasons specifically the involvement of high-speed intruders. CARLA being high fidelity simulator is based upon crowd detection and driving through algorithms. The simulator offers Open Street Map database that has tendency to detect the heterogeneous motion of multi agents in urban traffic in open-source environment. This summit is built in accordance with the laws of physics as well as autonomous driving realism. In this way CARLA architecture offers a variety of services including perception and planning mechanism, vehicle control system, learning approach mechanism, etc. in real time traffic behaviour detection and optimum performance.

4.6 Algorithms

Autonomous driving is dependent upon different algorithm being used by CARLA simulator. These algorithms include modular pipeline, imitation learning and reinforcement learning. First approach decomposes the driving mechanism into different sub-systems namely perception, planning and consecutive control. No input in terms of map is provided so map perception becomes a critical task. It should be noted that perception approach utilizes semantic segmentation technique to estimate the road boundaries, sideways, static and dynamic objects along with hazards. A local planner utilizes rule-based state machine to help in the implementation of predefined rules and regulation of urban traffic environment. Consecutive and efficient control is performed with the help of PID control unit that helps to accurate the steering and braking mechanism.

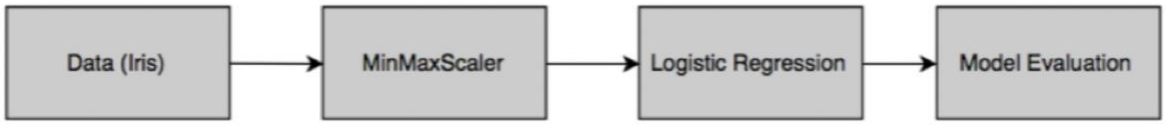

Figure 14: a simple pipeline [19]

Second approach is conditional imitation learning. This approach depends upon comparatively higher commands along with perception input. This method uses datasets in the form of tuples {oi, ci, a} (observation: oi, command: c and action: a). commands are given by the drivers, which are collected during data collection process. Datasets can suggest the future turns of the driver from perception approach. Most commonly used commands include follow lane, go straight till next intersection, turn left and turn right. The observations include images from cameras.

Figure 15: Imitation Learning Algorithm

Third approach is includes deep enforcement learning. This methodology involves reward signal with no human involvement. For this purpose Asynchronous Advantage Actor Algorithm is used. This algorithm has shown excellent performance in three dimensional fields against positional changes. Due to its asynchronous approach it can allow running multiple sequences at the same time and handling the complexity of the reinforcement learning.

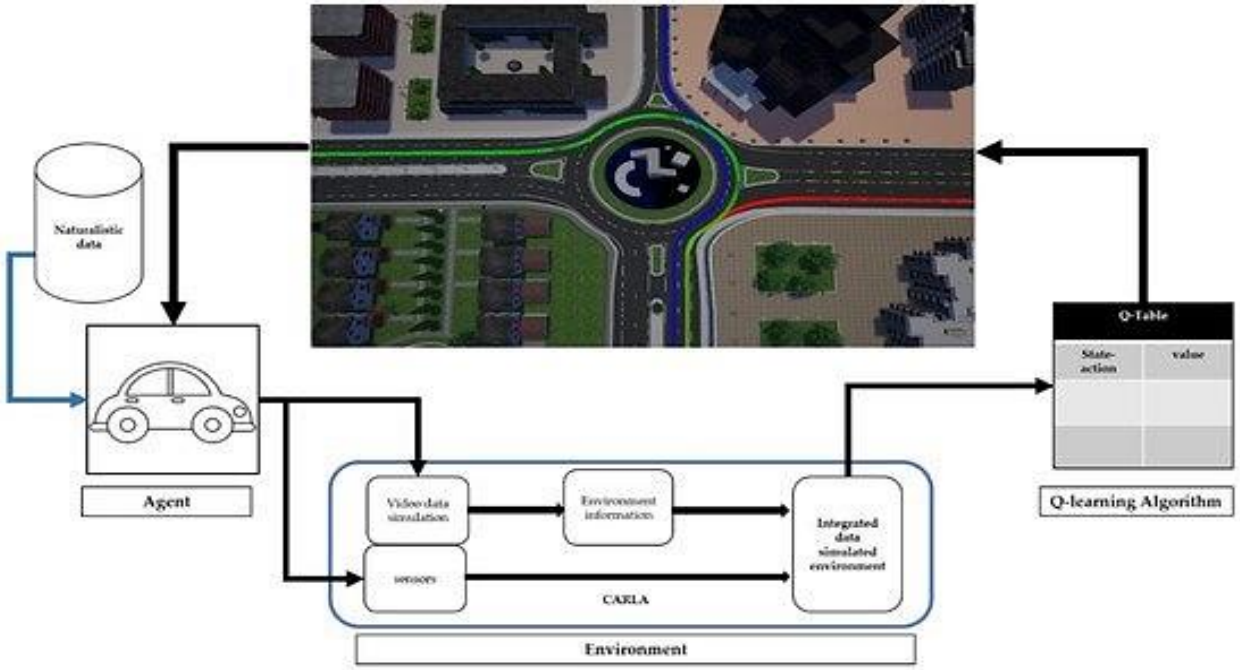

Figure 16: Learning Algorithm [18]

4.7 Alternative Designs/Methods

CARLA offers a variety of designs and features to test and verify the autonomous driving mechanism. There are other simulators that claim to provide simulation feature but not any of them have features even comparable to CARLA. These alternatives include auto ware, AirSim and TORCS which are based upon Gazebo, UE4 & Unity and OpenGL architectures. Additionally there are Udacity Simulator, Donkey Car Simulator, which are simpler and use unity based architecture. In comparison to CARLA, AirSim also uses the same technology but it has very less features and user controls.

Figure 17: 8AirSim GUI

4.8 Graphical User Interface Design

At start of CARLA GUI allows the user to create simulation in URBAN (intersection) mode as well as free mode (freeway) scenarios, as shown in following figure.

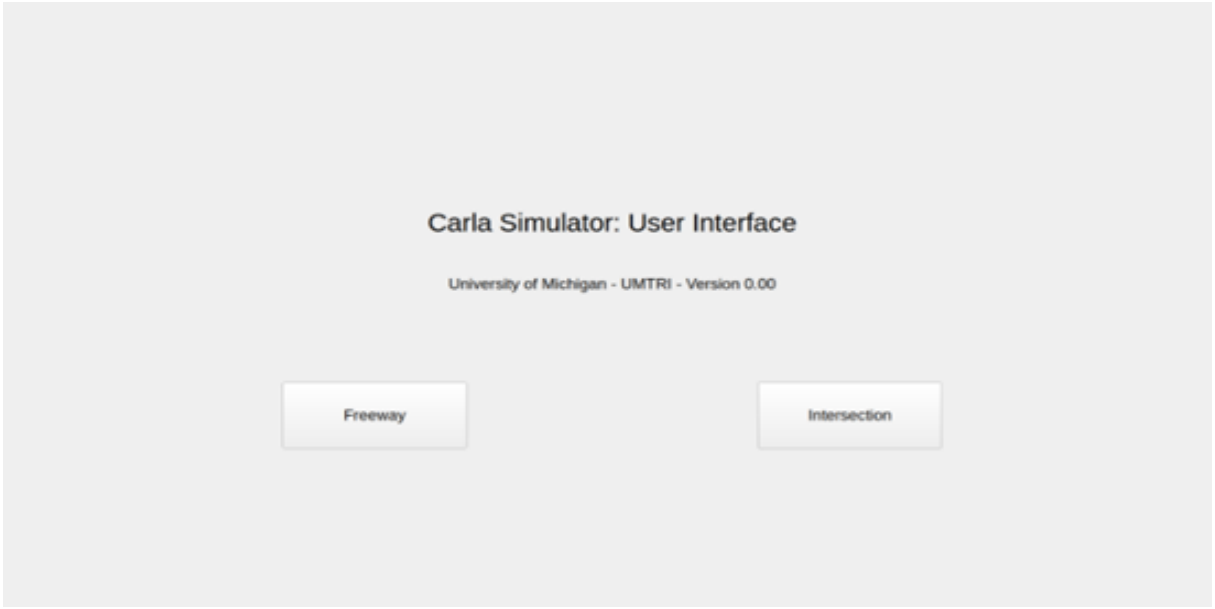

Figure 18: Starting GUI

As mentioned previously urban scenario is still under the process of development. Its GUI is shown in following figure that depicts the back-end feature of URBAN scenario.

Figure 19: Urban scenario

The following figure depicts the back-end feature of freeway scenario.

Figure 20: Freeware Scenario

5 Implementation Chapter

5.1 Introduction

This chapter has two parts: implementation requirements and implementation details. The implementation requirements will mention all hardware and software requirements as well as programming languages.

5.2 Implementation Requirements

5.2.1 Hardware Requirements

- Any modern PC or laptop
- Core I7 equivalent or above CPU
- 8 GB of Ram or above

5.2.2 Software Requirements

Windows

- **Windows 10:** operating system
- **Cmake:** Creates standard files using simple configuration files.
- **Git:** is a release control system for CARLA repository management.
- **Make:** Create executable files.
- **Python3:** is the main programming language for developing the CARLA simulator.
- **Unreal Engine 4.24 :**is a state-of-the-art real-time engine and editor that features photorealistic

Linux

- **Ubuntu 18.04:** operating system
- **Python 3.7:** python is a versatile programming language that we choose as the language to program in.
- **Pygame:** to create graphics directly with Python.
- **NumPy**: is the fundamental package for array computing with Python.
- **CarlaUE4:** is a state-of-the-art real-time engine and editor that features photorealistic rendering, dynamic physics, and effects, lifelike animation, robust data translation.
- **PyCham community Edition:** text editor used for developing the software and to run it.

5.2.3 Programming Language(s)

Python and C ++ were chosen by the developers of Carla to implement all the positive aspects of the work, and these are some of the most important packages.

Table 1: the most important packages

Packages	Description
Actor	Used to represents an actor in the simulation
ActorAttribute	Used to cast the value given
ActorBlueprint	Contains all the necessary information for spawning an Actor
ActorList	Used to find an actor by id & Filters a list of Actor by id
BlueprintLibrary	Works as a list but it is a map
Client	Construct a Carla client.
Map	map and location
Sensor	Register a callback to be executed each time a new measurement is received.
Vehicle	Used to control of vehicle.
Walker	Used to control of Walker
World	world control and information
Waypoint ld	Returns an unique Id identifying this waypoint.
RssSensor	forward declaration of the RssDynamics structure

5.2.4 Tools and Technologies

- **Carla simulation tool**

The Carla Simulator was developed to support developers to design and verify autonomous driving systems. The Carla platform provides some open-source protocols to support the development and improvement of systems for autonomous driving and also provides a set of sensors and environmental conditions and fully controls them freely.

- **Unreal Engine (UE4)**

is a complete suite of creation tools for game development, architectural and automotive visualization, live event production, training and simulation, and other real-time applications.

5.3 Implementation Details

5.3.1 Deployment and Installation

There are several steps to getting started with the Carla emulator. For this project, we used Windows 10, and we need to type some commands on the machine to download programs and libraries. It is possible to use the Ubuntu or Linux, but the Windows version does not support all the features.

First: download the current release (CARLA 0.9.11)

this release has recent fixes and features, previous releases, and a nightly build with all the developmental fixes and features.

1 - to build the Unreal Engine, write this command in "command prompt" and run it.

```
git clone --depth=1 -b 4.24 https://github.com/EpicGames/UnrealEngine.git
```

2 - Install PIP for Python packages

These commands depend on your version. In this project.

```
pip3 install --user setuptools
```

3 - after download the source file should install the patch and apply it.

```
cd UnrealEngine

powershell -Command "(New-Object
System.Net.WebClient).DownloadFile('https://carla-releases.s3.eu-west-
3.amazonaws.com/Backup/UE4_patch_wheels.patch', 'UE4_patch_wheels.patch')"

git apply UE4_patch_wheels.patch
```

Second: Run the configuration scripts

The Carla simulator runs on Unreal Engine 4.24.

```
Setup.bat

GenerateProjectFiles.bat
```

Three: Clone the CARLA repository

In this command it is take you to the project files where you can find the official repository and you download from and extract it.

```
git clone https://github.com/carla-simulator/carla
```

5.3.2 Data Structures Description

This data structure what we have used in this project

A dataset directory looks like the structure in the figure below, and it will be explained in the next sections.

```
<dataset_name>
│   dataset_metadata.json
│
└─── episode_00000
│   │   episode_metadata.json
│   │   <Camera1_name>_00000.png
│   │   ...
│   │   <Camera2_name>_00000.png
│   │   ...
│   │   <Lidar_name>_00000.png
│   │   ...
│   │   measurements_00000.json
│   │   ...
└─── episode_00001
```

Dataset metadata

Each database contains metadata file with:

- Used cameras.
- The cars number to be used in each episode.
- The range of cars number to be used in each episode.
- The range of the number of pedestrians to be used in each episode.
- The percentage of long noise.
- The percentage of longitudinal noise.
- The set of weathers to be sampled from.

Each episode is stored on a different folder and for each collected episode we generate a json file containing its general aspects which is:

1. Count of walkers: The total count number of spawned walkers.
2. Count of Vehicles: The total count number of spawned vehicles.
3. Spawning for the walkers & vehicles: it's random Spawn used for the CARLA simulator spawning process.
4. Weather condition: It is the weather used, that is, the one that was applied in the episode.

Every episode lasts for few minutes partitioned in simulation steps of 100ms, in each step data stored divided categories and saved into two different pats, first one sensor data is stored as PNG images, second one measurement data are stored as json files.

Sensor Data

All collected images are stored as PNG, all lidar sensors collected are store as PLY files. The collected sensors are the described in the configuration file.

Measurements

Measurements represent all floating data collected for each simulation step, Each measurement is associated with its own sensor data. The units of measure are in SI format, otherwise we specify the format, all measurements are stored in json files and contain:

- Simulation step Number: the current simulation step, starts at zero and it increased by one for every new simulation step.
- Timestamp: the time that has passed since the simulation has started was expressed in milliseconds.

- Position: the world position of the ego-vehicle. It is expressed as a three-dimensional vector (x, y, z) in meters.
- Orientation: the orientation of the vehicle in the frame was Expressed as Euler angles (row, pitch and yaw).
- Acceleration: the acceleration of the E-vehicle in vector-based.
- Forward Speed: expressing the speed of the e-vehicle in linear path.
- Intentions: a signal that is proportional to the effect that the dynamic objects on the scene are having on the ego car actions. uses three different action signals: stop of car and stop in traffic lights and slowing down when coming much closer from another objects.
- High Level Commands: the high-level command means you can tills what the e-vehicle must do in the next intersection or situation: like go straight or turn left or turn right or do nothing (the ego vehicle could pick any option). These commands are encoded in as an integer number.
- Waypoints: it is a set containing the ten future positions weher the vehicle can be.
- Steering Angle: the angle of the vehicle's steering wheel and it direction.
- Throttle: the current mod of pressure on the throttle pedal.
- Brake: the current mod of pressure on the brake pedal.
- Hand Brake: the current mod of the hand brake.
- Steer Noise: the current mood of steering angle in the car considering the noise function.
- Throttle Noise: the current mood pressure on the throttle pedal of the car considering the noise function.
- Brake Noise: the current mood pressure on the brake pedal in the car considering the noise function.

For each type of the non-player agents such as Walker's, Cars, traffic light), the following information is provided:

- Unique ID: unique identifier for each.
- Type: Type of the object.
- Position: the position of the agent in the city. And It is expressed as a three-dimensional (x, y,z) vector in meters.
- Orientation: the orientation of the agent with respect to the map. Are expressed as Euler angles (row, pitch and yaw).
- Forward Speed: a scalar expressing the linear speed of the car.
- State: it only for traffic lights, which contains the state of the traffic light, if it is either red, yellow etc.

5.3.3 Procedures Description

In the tables, we will provide a description for the important methods.

- **Actor**

Table 2: Actor methods

Name of Methods	**Description**
add_angular_impulse	To determine the angle of mass of an object for the actor
add_force	To apply force over a period of time on the cycle
add_impulse	instantaneous momentum to the actor
add_torque	Apply torque over a specified period of time
destroy	Used to permanently delete the actor
disable_constant_velocity	is used in case the developer wants to disable the actors'
enable_constant_velocity	It is used to adjust the speed of the car to make it stable

- **Actor attribute**

Table 3: Actor attribute methods.

Name of Methods	Description
as_bool	To reads the attribute as Boolean value
as_color	To reads the attribute as Carla.Color
as_float	To reads the attribute as float
as_int	To reads the attribute as int
as_str	To reads the attribute as string.

- **Actor Blueprint**

Table 4: Actor Blueprint methods.

Name of Methods	Description
has_attribute	To returns True if the blueprint contains the attribute id
has_tag	To returns True if the blueprint has the specified tag listed
match_tags	To returns True if any of the tags listed for this blueprint matches

- **Actor list**

Table 5: Actor list methods

Name of Methods	Description
filter	Filters the list of the Actors matching specific pattern against their variable type_id which identifies the blueprint used to spawn them
find	Finds an actor using its identifier and returns it or ignore if it is not there

- **Client**

Table 6: Client methods

Name of Methods	Description
apply_batch	To execute a list of commands in one step
load_world	To download a new world
reload_world	To reload the world
show_recorder_collisions	View the recorded collisions
show_recorder_file_info	View the recordings saved in a file
start_recorder	Start recording and save it to a file
stop_recorder	Stop recording

- **Map**

Table 7: Map methods

Name of Methods	Description
generate_waypoints	Used to return a list of coordinates with a specified distance for each lane to create waypoints.
save_to_disk	To save the path of the map to the OpenDRIVE file
to_opendrive	Returns the path saved in OpenDRIVE to the map
convert_to_geolocation	To change the saved path on OpenDRIVE

- **Sensor**

Table 8: Sensor methods

Name of Methods	Description
listen	it's the function that the sensor will call it in every time it receive a new measurement
Stop	Commands the sensor to stop listening for any data

- **Vehicle**

Table 9: Vehicle methods

Name of Methods	Description
apply_control	Applies a control object on the car , containing driving parameters such as throttle, steering or gear shifting
apply_physics_control	Applies a physics control object in the next tick containing the parameters that define the vehicle as a corporeal body
enable_carsim	Enables the CarSim physics solver for this particular vehicle
is_at_traffic_light	Vehicles will be affected by a traffic light when the light is red, and the vehicle is inside its bounding box
use_carsim_road	Enables or disables the usage of CarSim vs terrain file

- **Blueprint Library**

Table 10: Blueprint Library methods

Name of Methods	Description
filter	Filters a list of blueprints matching the wildcard_pattern against the id , and returns the result as a new one
find	Returns the blueprint identifier.

- **World**

Table 11: World methods

Name of Methods	Description
apply_settings	When simulation running and returns the ID of the frame they were implemented.
cast_ray	Casts a ray from the specified firs location to final location, and all geometries intersecting the ray and returns a list of carla.LabelledPoint .
enable_environment_objects	This method enables or disables a set of Environment Object defined by the identifier, and there is a possibility that these objects will appear or disappear from the level.
freeze_all_traffic_lights	Freezes and unfreezes all traffic lights, the frozen traffic lights can be modified by the user and it will not be updated them until unfrozen.
ground_projection	The one will cast a ray from current location in the direction (0,0, -1) downwards and returns to Carla, If no geometry is found in the search distance range
load_map_layer	Loads the selected layers to the level
on_tick	Used in asynchronous mode, it is called every time the server ticks, used remove_on_tick() to stop the callbacks.
project_point	Displays the specified point in the desired direction, the function throws a ray from the location in another direction and returns the Carla
remove_on_tick	A callback is not triggered for callback_id started with on_tick().
reset_all_traffic_lights	Reset all traffic lights in the map to the first state
spawn_actor	In this method it will create and return and spawn an actor. Also, it will need a schema for the site to be created and converted to location
tick	used in synchronous mode, next frame will send the tick, it returns the ID of the new frame directly by the server
try_spawn_actor	returns None on failure instead of throwing an exception
unload_map_layer	Empties the selected layers to the level, of the layer is not loaded the call will have no effect on it
wait_for_tick	the next frame is computed and the server will tick and return a snapshot which describe the new state

- **Waypoint**

Table 12: Waypoint methods

Name of Methods	Description
next	Returns the list of coordinates at a certain approximate distance from the current point, focusing on the expected deviations without changing the lane, if the list is empty; this means that the lane is not connected to the other lane
next_until_lane_end	Returns coordinates to the end of the lane without a specific marker
previous	Not to return the coordinates used by the actor, but in the opposite direction of the path, and the focus is on the road and its possible deviations without any change in the lane
previous_until_lane_start	Return the list of coordinates from the beginning of the lane without any connection to another lane

- **RSS sensor**

Table 13: RSS sensor methods

Name of Methods	Description
append_routing_target	Determine a new route for the vehicle
drop_route	to Ignore the current route, can create a new route using targets otherwise create a new random path
register_actor_constellation_callback	Register the callback to customize the result of carla.RssActorConstellation, and the RSS parameter settings are finished without affecting the settings
reset_routing_targets	Delete the targets that are in the route

- **Walker**

Table 14: Walker methods

Name of Methods	Description
apply_control	to move the walkers in a specific direction and at a standard speed, how many steps can be done
apply_control	Select the control that will be applied to the walker's skeleton

5.3.4 Graphical User Interface Description

In this part, we will show some pictures of when the CARLA simulator is running and explain the details about it.

View of the map from the top

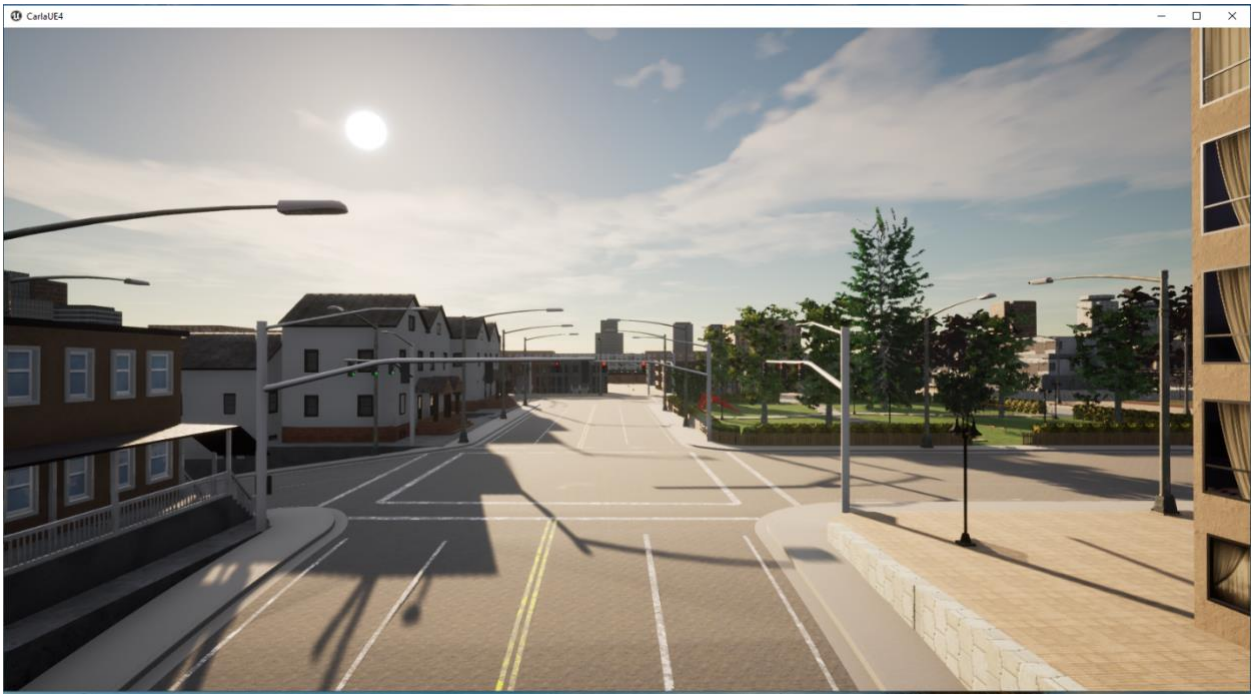

Figure 21:View of the map from the top

After adding the vehicles and walkers on the map

Figure 22: adding vehicles and walkers

run's manual control to drive

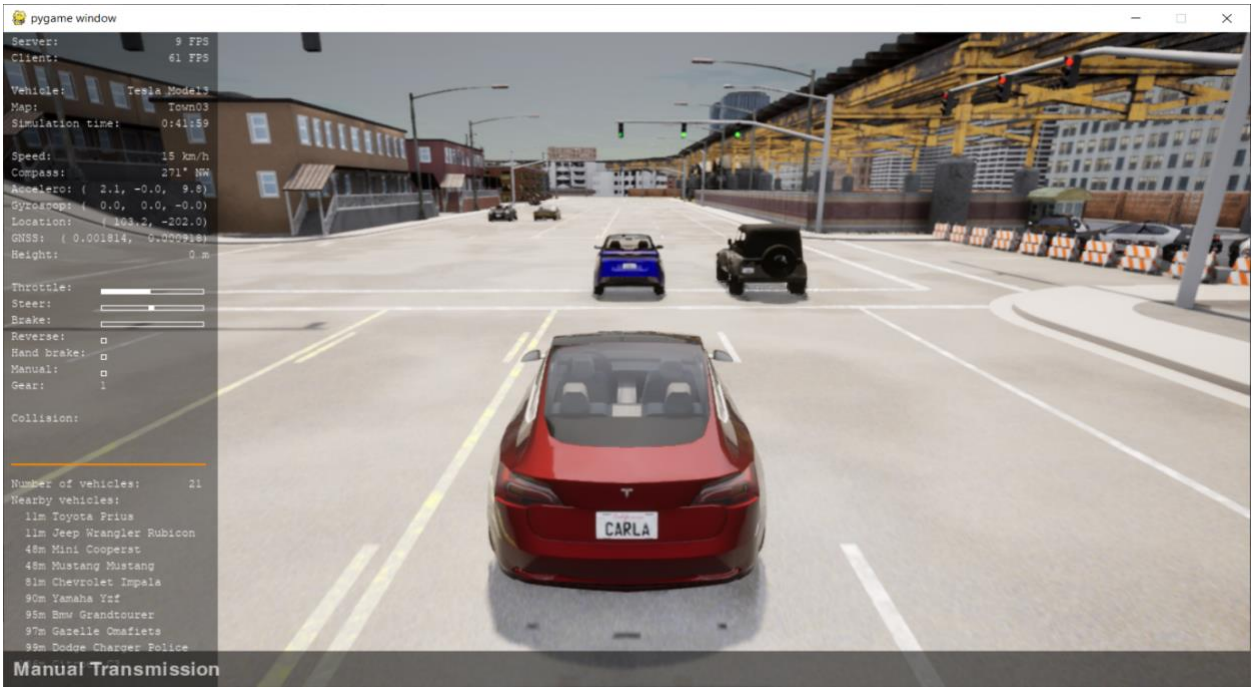

Figure 23: manual control

A list of details to help with driving on the CARLA simulation

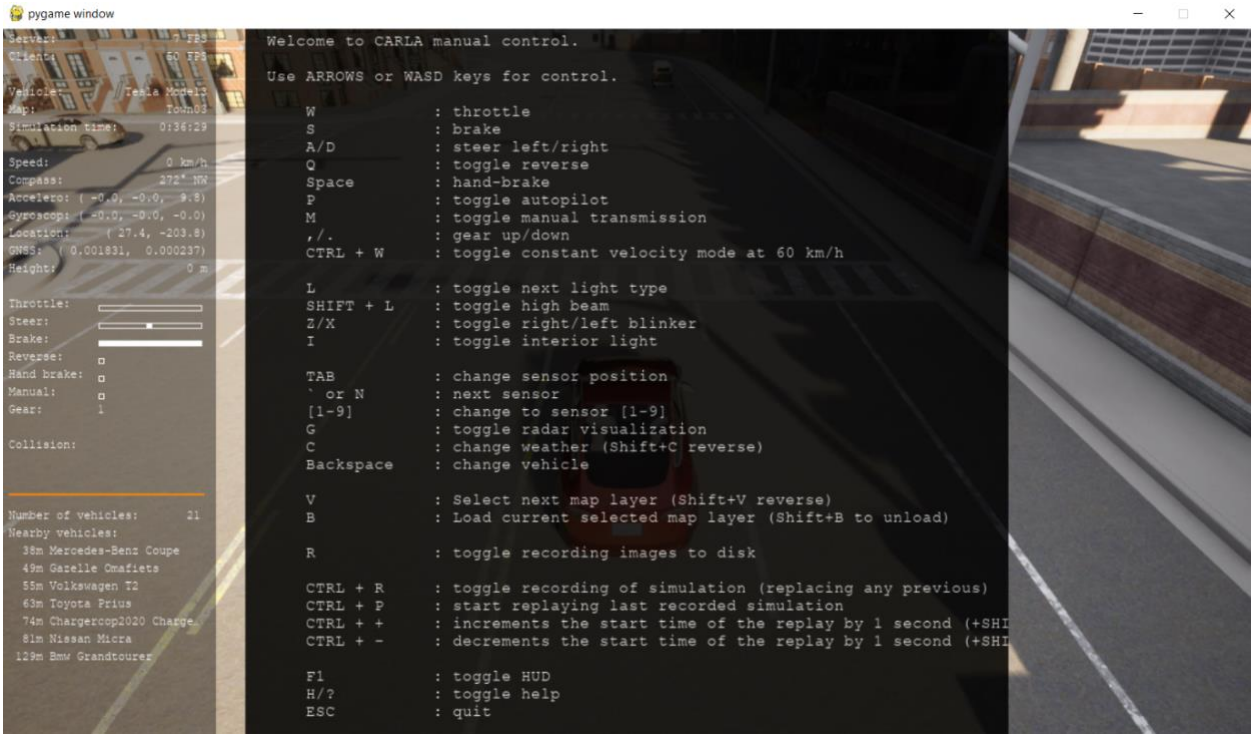

Figure 24: list of details to help with driving

Change weather on the map

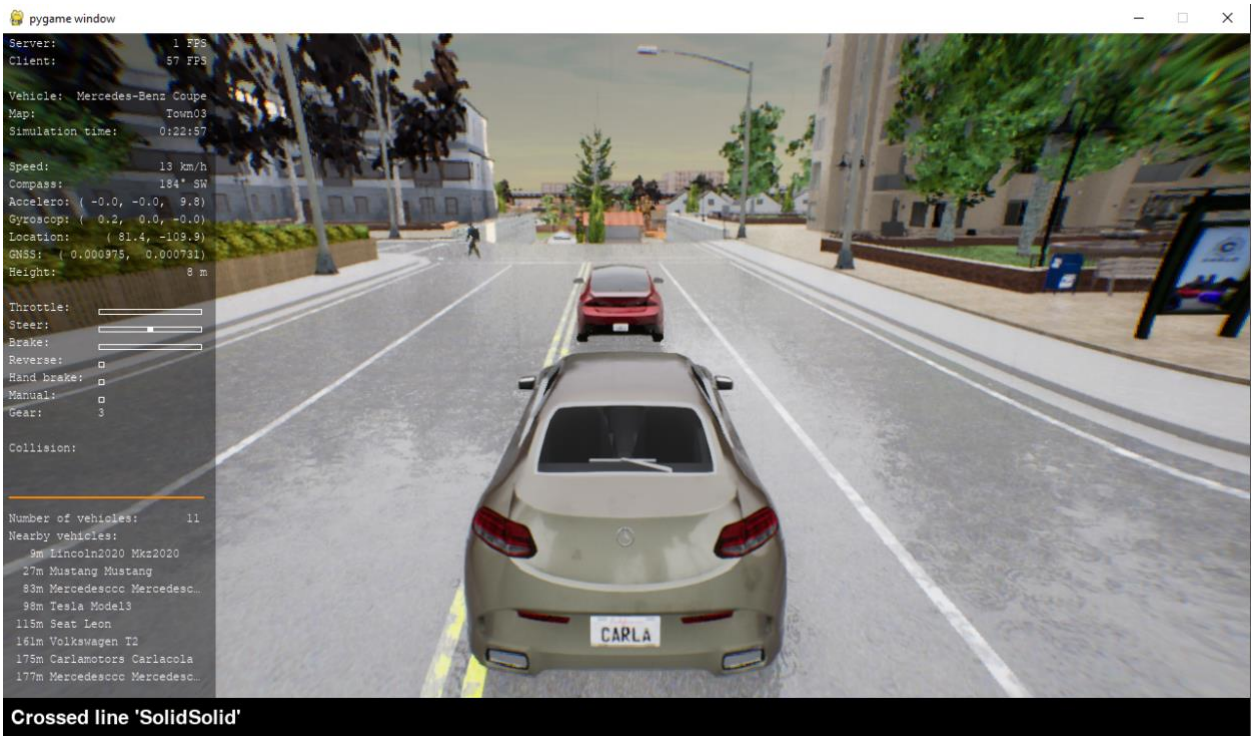

Figure 25: Rainy weather

When the vehicle off the lane, details appear at the bottom of the screen

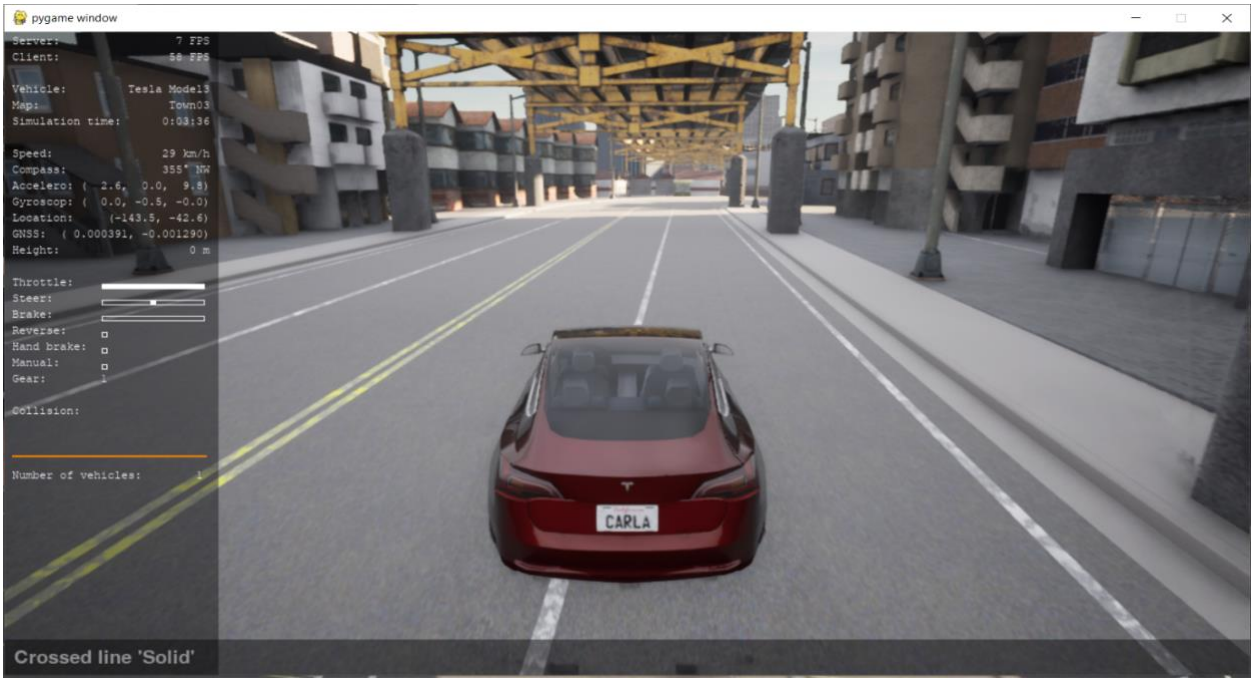

Figure 26: crossed line.

Upon collision, collision details appear at the bottom of the screen “Manual control “mode.

Figure 27: collision with vehicle

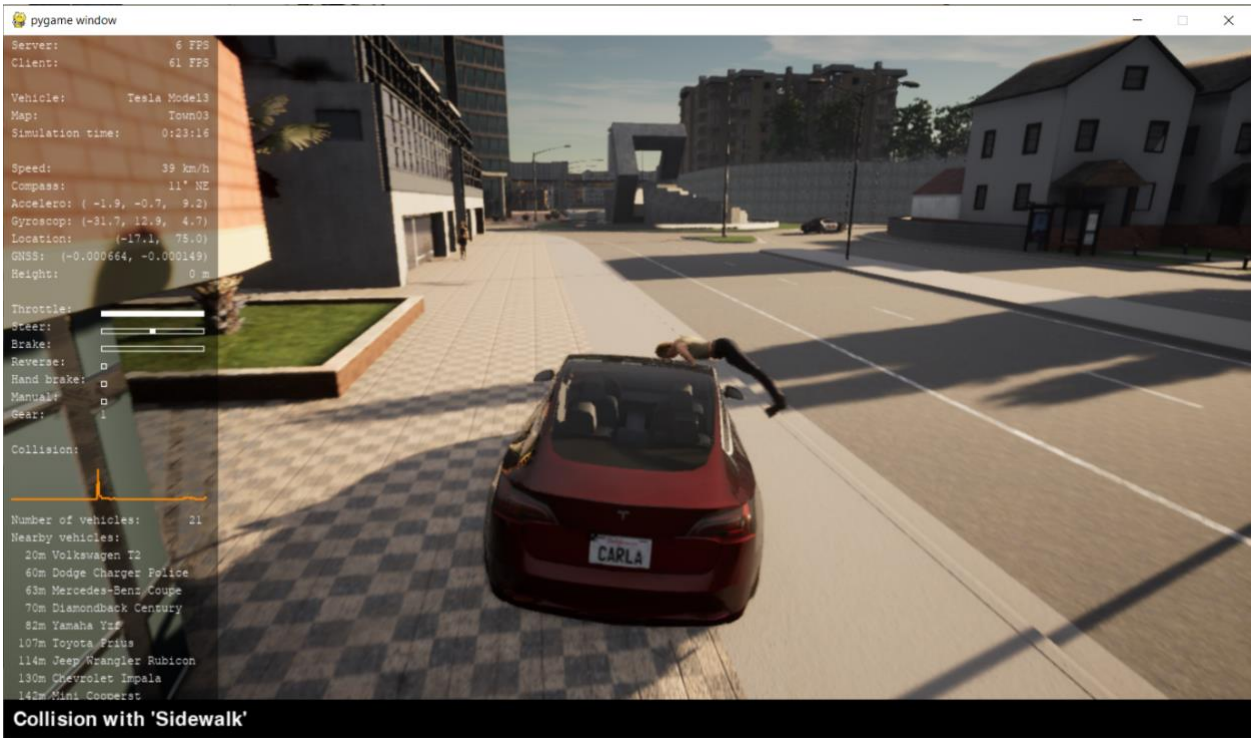

Figure 28: collision with sidewalk

Figure 29: collision with wall

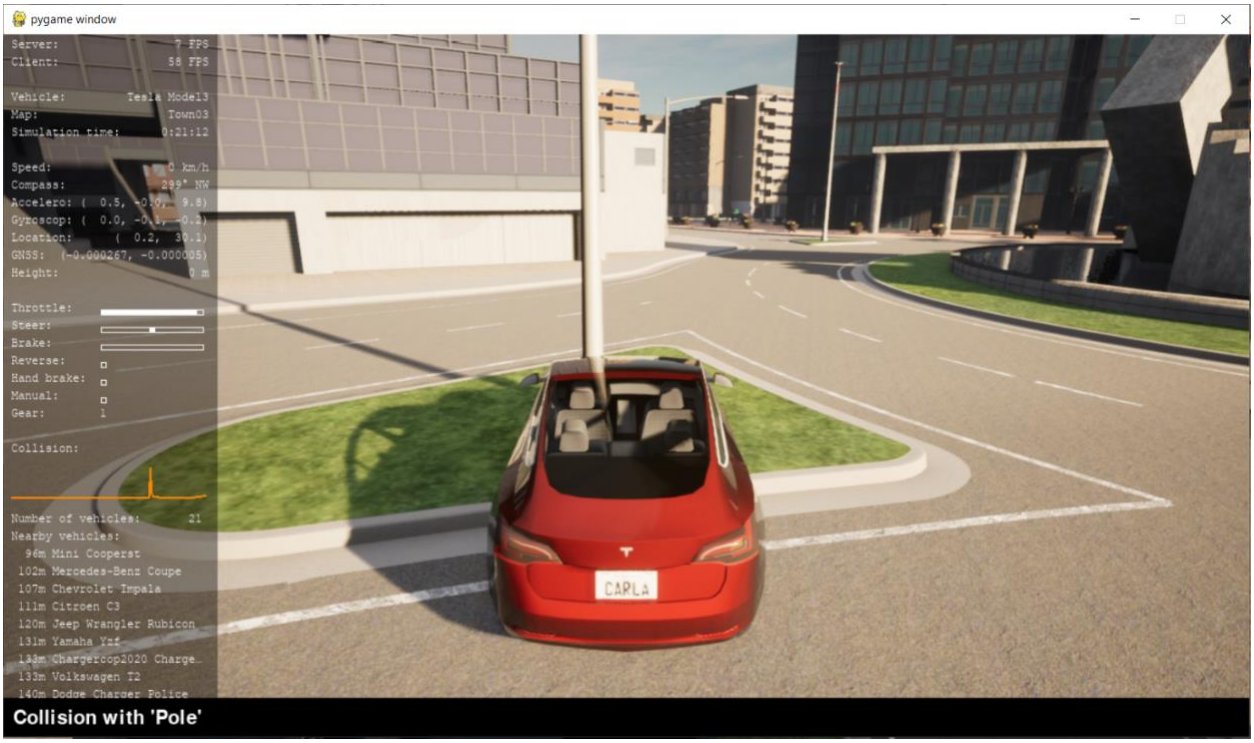

Figure 30: collision with pole

run's the "Autopilot ", it is means drive by use the sensors 'RSS' on the road.

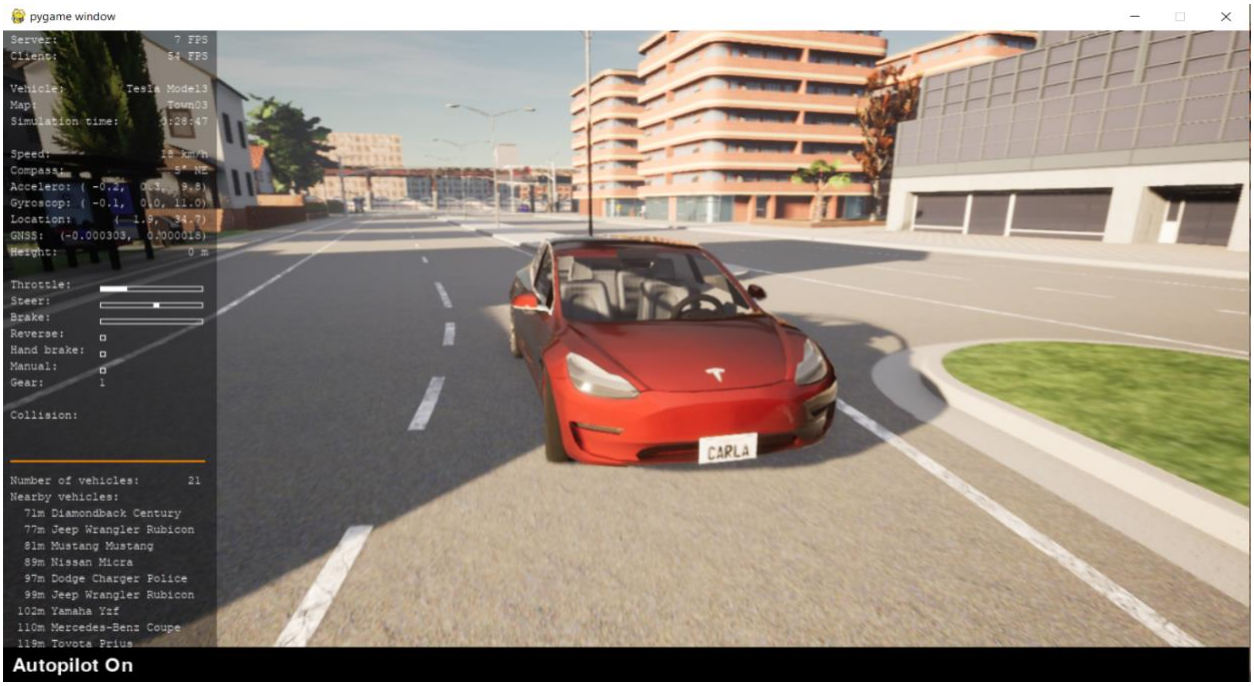

Figure 31: run auto pilot.

Stand completely on paths, pedestrians, and intersections and make sure of them before setting off, after run's '' autopilot " mode.

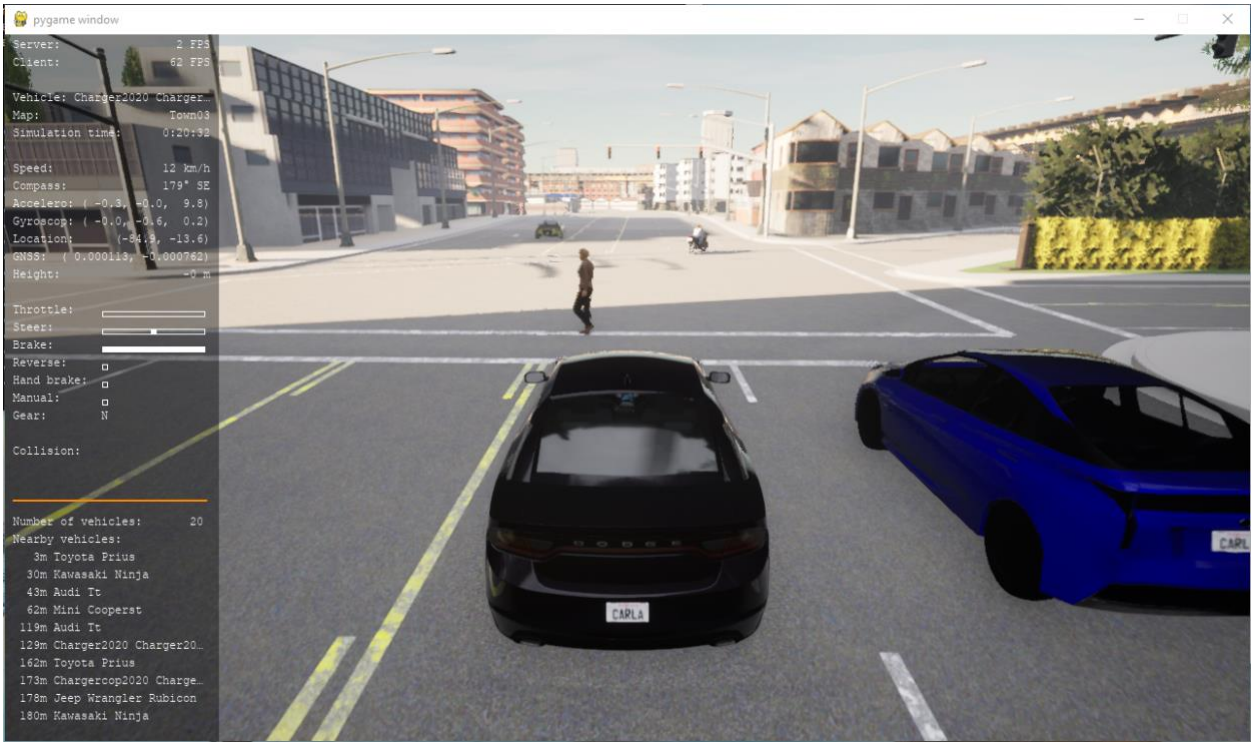

Figure 32: Stand completely on paths

Maintaining the path and not leaving it after operating the "autopilot" mode.

Figure 33: not leaving the path.

Client bounding boxes to alert him to vehicles moving around him.

Figure 34: Client bounding boxes

Derail in curves of the road at a speed above 60 km / h “Autopilot “mode.

Figure 35:Derail in curves of the road at a speed above 60 km / h (1)

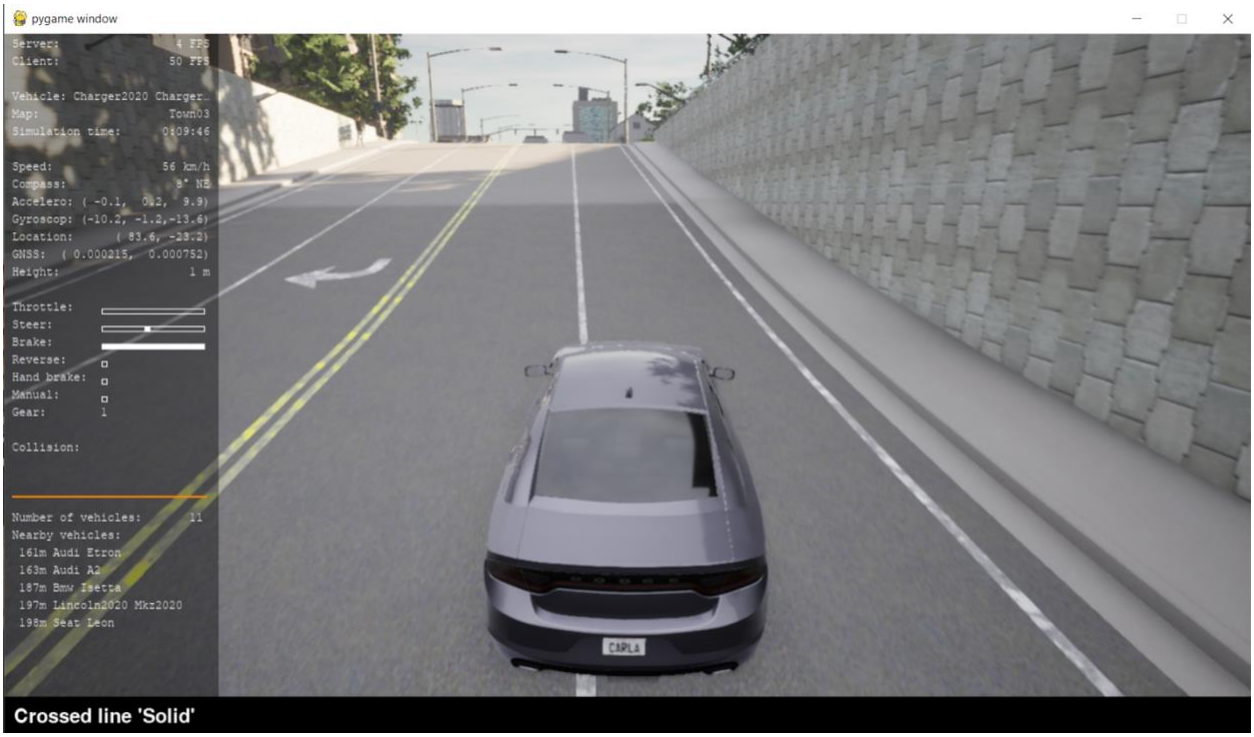

Figure 36:Derail in curves of the road at a speed above 60 km / h (2)

Ignore stop before turning at intersections, at above 60 km / h “Autopilot “mode.

Figure 37:Ignore stop before turning at intersections, at above 60 km / h (1)

Figure 38:Ignore stop before turning at intersections, at above 60 km / h (2)

Not to stop at traffic lights when they are red at above 60 km / h “Autopilot “mode.

Figure 39:Not to stop at traffic lights when they are red at above 60 km / h (1)

Figure 40:Not to stop at traffic lights when they are red at above 60 km / h (2)

Rendering mode visualizer

Figure 41: rendering mode

6 Testing Chapter

6.1 Introduction

In this chapter, we will try the process of testing commands to ensure the software is working as expected, by applying the used software techniques.

Table 15: verify running the simulator

Step name: Simulator running.				
Description: Testing the simulator.				
Pre-condition: should have the systems and software requirement.				
Step number	**Test steps**	**Expected result**	**Actual result**	**Statues**
1	Running the simulator Carla UE4	The system should display main frame	The map in simulator is displayed	Pass
2	Opening the pythonAPI then examples file and running the Command prompt "	The command prompt window should display	command prompt window displayed	Pass
3	Running python spawn_npc.py	The system should display default operating test	The system operations work	Pass
Post-conditions: All system requirements have been validated and simulator are working successfully.				

Table 16: verify adding cars and choosing a type of car and choosing gear mood

Step name: Car's modification				
Description: Adding car, car type, gear type				
Pre-condition: should have the systems requirement and simulator server connection.				
Step number	**Test steps**	**Expected result**	**Actual result**	**Statues**
1	Opening the examples file and running the"Command prompt" running python spawn_npc.py -n 20	The system should display 20 cars on the map	The system displayed 20 cars on the map	Pass
2	Choosing a type of car using press button "backspace"	System should show you different car type with each press	System shows you different car type with each press	Pass
3	Choosing gear mood using press button "M"	System should change the gear mood from Auto to manual mood	System changing the gear mood from Auto to manual mood	Pass
Post-conditions: All system changes have been validated and applied on the simulator successfully.				

Table 17: verify driving control, auto drive, weather dynamic

Step name: Controls options				
Description: driving control, auto drive, weather dynamic				
Pre-condition: should have the systems requirement and simulator server connection.				
Step number	**Test steps**	**Expected result**	**Actual result**	**Statues**
1	Opening the examples file and running the"Command prompt" running python manual_control.py	The system should open new frame for manual driving using "w,a,s,d" press buttons	The system opened new frame for manual driving, can be control using "WASD" press buttons	Pass
2	Changing the driving mood from manual to Autopilot mood using press button "P"	The car should move by itself	The car moved by itself	Pass
3	Opening the examples file and running the"Command prompt" running python dynamic_weather.py	System should change the weather dynamically	System changed the weather dynamically and can be control using "C" press buttons	Pass
Post-conditions: All system changes have been validated and applied on the simulator successfully.				

- **Performance comparison**

Table 18: Performance comparison.

Cases	**Manual control mood**	**Autopilot mood**
Keep on track, at a speed of 60 km / h	Up to the driver	Going on the right track
Keep on track, at a speed above 60 km / h	Up to the driver	Derail in curves of the road
Stop at the pedestrian crossing at a speed of 60 km / h	Up to the driver	Completely stop on paths
Stop at the pedestrian crossing at above 60 km / h	Up to the driver	Ignores the crossed path
Stop before turning at intersections at a speed of 60 km / h	Up to the driver	Completely stop on paths
Stop before turning at intersections, at above 60 km / h	Up to the driver	Ignores the intersections

6.2 Conclusion

We applied the testing cases using the specific techniques of software operating on the Carla simulator and it has been explained in detail in the above tables.

7 Conclusion Future work Chapter

7.1 Future work

An autonomous vehicle is a vehicle or truck that can sense its environment and control its movements without human intervention. With this type of vehicle, the human driver will be responsible for all driving tasks, and these vehicles include some driver assistance features such as lane-keeping assist or adaptive stability control Speed, the car can combine two or more automatic tasks such as steering and acceleration simultaneously, and the car can drive from point A to point B without human intervention, but only in certain circumstances. The driver still needs to be prepared to take on the task at any time because the system will require a human to intervene in critical situations. The vehicle is fully autonomous in most, but not all, driving conditions. In general, the car will be able to drive on its own and not require human intervention to complete the journey. In this project, we presented a model for the development of autonomous vehicles through the Carla simulator, and the development of conditions similar to reality to overcome the problems faced by autonomous vehicles on the roads by developing sensors and linking them to the vehicles for experiment and drawing plans to develop them in the future more precisely. This experience with the Kala simulator had a clear effect on the study and development of autonomous vehicles. Many people strive to be the first to develop a self-driving car that can drive in all driving conditions. Big automakers and tech giants are looking for a way to get to the front of the race. They run a bunch of experiments on the road with their self-driving cars, and the data gathered from these experiments help cars navigate a world where unexpected things happen all the time. In the future of autonomous driving, cars will not only take on many of our tasks, but our societies will undergo a massive redesign. The cars will drive you to the destination and continue to serve the next customer. Then there would be no need for sprawling parking and underground parking, and self-driving cars could provide people with disabilities with the basic freedom to come and go as they like.

7.2 Conclusion

The transition from conventional cars to autonomous vehicles has not yet occurred, however, modern artificial intelligence technologies and the development of machine learning are making rapid leaps forward, and our project was one of the examples of modern artificial intelligence techniques to develop machine learning, and at the end of the chapter, we talked about future work for Autonomous vehicles.

8 Appendix chapter

Appendix

Spawn NPCS into the simulation

```
import glob
import os
import sys
import time
try:
sys.path.append (glob.glob (../ carla / dist / carla - *% d.% d-% s.egg '%)
sys.version_info.major,
sys.version_info.minor,
win-amd64 'if os.name = "nt' else" linux-x86_64)) (0)
except IndexError:
pass
import carla
from carla import VehicleLightState as vls
import argparse
import logging
from numpy import random
def main ():
argparser - argparse.ArgumentParser (
description = doc_)
```

```
argparser.add_argument (
'--host',
metavar = 'H',
default = '127.0.0.1',
help = IP of the host server (default: 127.0.0.1) ')
argparser.add_argument (
'-p', '--port',
metavar = 'P',
default = 2000,
type = int,
help = "TCP port to listen to (default: 2000) ')
argparser.add_argument (
'-n', '-number-of-vehicles',
metavar = 'N',
default = 10,
type = int,
help = 'number of vehicles (default: 10) "')
argparser.add_argument (
'-w', '--number-of-walkers',
metavar = 'W',
default = 50,
type = int,
help = "number of walkers (default: 50))
argparser.add_argument (
'--safe',
action = "store_true ',
help = 'avoid spawning vehicles prone to accidents')
argparser.add_argument (
--filterv ',
metavar = "PATTERN ', default =' vehicle. *", help = 'vehicles filter (default: "vehicle. *")')
```

```
  argparser.add_argument (
-filterw ',
metavar = "PATTERN ',
default = 'walker.pedestrian. * ",
help = "pedestrians filter (default:" walker.pedestrian. * ")")
argparser.add_argument (
--tm-port ',
metavar = "P ',
default = 8000,
type = int,
help = "port to communicate with TM (default: 8000) ')
argparser.add_argument (
'--sync',
action = "store_true ',
help = "Synchronous mode execution ')
argparser.add_argument (
--hybrid ',
action = "store_true ',
help = "Enanble ')
argparser.add_argument (
'-s', '--seed',
metavar = 'S',
type = int,
help = "Random device seed ')
argparser.add_argument (
'--car-lights-on',
action = 'store_true',
default = False,
```

```
help = "Enanble car lights')
args = argparser.parse_args ()
logging.basicConfig (format = "% (levelname) s:% (message) s, level = logging.INFO)
vehicles_list = []
walkers_list = []
all id = 0
client = carla.Client (args.host, args.port)
client.set_timeout (10.0)
synchronous_master = False
random.seed (args.sced if args.seed is not None else int (time.time ()))
try:
world = client.get_world ()
traffic_manager = client.get_trafficmanager (args.tm_port)
traffic_manager.set_global_distance_to_leading_vehicle (1.0)
if args.hybrid:
traffic_manager.set_hybrid_physics_mode (True)
if args.seed is not None:
traffic_manager.set_random_device_seed (args.seed)
if args.sync:
settings = world.get_settings ()
traffic_manager.set_synchronous_mode (Truc)
if not settings.synchronous_mode:
synchronous_master = True
settings.synchronous_mode = True
settings.fixed_delta_seconds = 0.05
world.apply_settings (settings)
```

```
else:
synchronous_master = False
blueprints = world.get_blueprint_library (.filter (args.filterv)
blueprints Walkers = world.get_blueprint_library (). filter (args.filterw)
if args.safe:
blueprints = (x for x in blueprints if int (x.get_attribute ('number_of_wheels')) = 4)
blueprints = [x for x in blueprints if not x.id.endswith (isetta '))
blueprints = [x for x in blueprints if not x.id.endswith (carlacola '))
blueprints = [x for x in blueprints if not x.id.endswith (cybertruck '))
blueprints = [x for x in blueprints if not x.id.endswith (2 '))
blueprints = sorted (blueprints, key = lambda bp: bp.id)
spawn_points = world.get_map (). get_spawn_points ()
number_of_spawn_points = len (spawn_points)
if args.number_of_vehicles <number_of_spawn_points:
random.shuffle (spawn_points)
elif args.number_of_vehicles> number_of_spawn_points:
msg = requested% d vehicles, but could only find% d spawn points'
logging.warning (msg, args.number_of_vehicles, number_of_spawn_points)
args.number_of_vehicles = number_of_spawn_points
# @todo cannot import these directly.
SpawnActor = carla.command.SpawnActor
SetAutopilot = carla.command.SetAutopilot
SetVehicleLightState = carla.command.SetVehicleLightState
FutureActor = carla.command.FutureActor
```

Spawn vehicles.

```
batch = []
for n, transform in enumerate (spawn_points):
ifn> = args.number_of_vehicles:
break
blueprint = random.choice (blueprints)
if blueprint.has_attribute ('color'):
color = random.choice (blueprint.get_attribute ('color'). recommended_values)
blueprint.set_attribute ('color', color)
if blueprint.has_attribute ('driver_id'):
driver_id = random.choice (blueprint.get_attribute ('driver_id'). recommended_values)
blueprint.set_attribute ('driver_id', driver_id)
blueprint.set_attribute (role_name ',' autopilot ')
# prepare the light state of the cars to spawn
light_state = vls.NONE
if args.car_lights_on:
light_state = vls.Position | vls.LowBeam | vls. LowBeam
# spawn the cars and set their autopilot and light state all together
batch.append (SpawnActor (blueprint, transform)
.then (SetAutopilot (FutureActor, True, traffic_manager.get_port ())
then (SetVehicleLightState (FutureActor, light_state)))
for response in client.apply_batch_sync (batch, synchronous_master):
if response.error:
logging.error (response.error)
else:
vehicles_list.append (response.actor_id)
```

Spawn walker.

```
percentagePedestriansRunning = 0.0 # how many pedestrians will run
percentagePedestriansCrossing = 0.0 # how many pedestrians will walk through the road
# 1. Take all the random locations to spawn
spawn_points = []
for i in range (args.number_of_walkers):
spawn_point = carla. Transform ()
loc = world.get_random_location_from_navigation ()
if (loc! = None):
spawn_point.location = loc
spawn_points.append (spawn_point)
# 2. We spawn the walker object
batch = []
walker_speed = []
for spawn_point in spawn_points:
walker_bp = random.choice (blueprints Walkers)
# set as not invincible
if walker_bp.has_attribute ('is_invincible'):
walker_bp.set_attribute ('is_invincible', 'false')
# set the max speed
if walker_bp.has_attribute ('speed'):
if (random.random ()> percentagePedestriansRunning):
# walking
walker_speed.append (walker_bp.get_attribute ('speed'). recommended_values [1])
else:
# running
walker_speed.append (walker_bp.get_attribute ('speed'). recommended_values [2])
```

```
else:
print ("Walker has no speed")
walker_speed.append (0.0)
batch.append (SpawnActor (walker_bp, spawn_point))
results = client.apply_batch_sync (batch, True)
walker_speed2 = []
for i in range (len (results)):
if results [i] .error:
logging.error (results [i] .error)
else:
walkers_list.append ({"id": results [i] .actor_id})
walker_speed2.append (walker_speed [i])
walker_speed = walker_speed2
# 3. We spawn the walker controller
batch = []
walker_controller_bp = world.get_blueprint_library (.find ('controller.ai.walker')
for i in range (len (walkers_list)):
batch.append (SpawnActor (walker_controller_bp, carla.Transform), walkers_list [i] ["id"]))
results = client.apply_batch_sync (batch, True)
for i in range (len (results)):
if results [i] .error:
logging.error (results [i] .error)
else:
walkers_list [i] ["con"] = results [i] .actor_id
```

Weather.

```
import glob, import os, import sys
try:
sys.path.append (glob.glob ("./ carla / dist / carla - *% d.% d-% s.egg '%)
sys.version_info.major,
sys.version_info.minor,
'win-amd64' if os.name == 'nt' else 'linux-x86_64)) [0])
except IndexError: pass
import carla
import argparse
import math
def clamp (value, minimum = 0.0, maximum = 100.0):
return max (minimum, min (value, maximum))
class Sun (object):
def_init _ (self, azimuth, altitude):
self.azimuth = azimuth
self.altitude = altitude
self._t = 0.0
def tick (self, delta_seconds):
self._t + = 0.008 * delta_seconds
self_t% = 2.0 * math.pi
self.azimuth + = 0.25 * delta_seconds
self.azimuth% = 360.0
self.altitude = (70 * math.sin (self._1)) - 20
def_str (self):
return 'Sun (alt:% .2f, azm:% .2f)'% (self.altitude, self.azimuth)
```

```
class Storm (object):
def_init_ (self, precipitation):
self_ t = precipitation if precipitation> 0.0 else -50.0
self._increasing = True
self.clouds = 0.0
self.rain = 0.0
self.wetness = 0.0
self.puddles = 0.0
self.wind = 0.0
self.fog = 0.0
def tick (self, delta_seconds):
delta = (1.3 if self._increasing else -1.3) * delta_seconds
self_t = clamp (delta + self._t, -250.0, 100.0)
self.clouds = clamp (self._t + 40.0, 0.0, 90.0)
self.rain = clamp (self_t, 0.0, 80.0)
delay = -10.0 if self._increasing else 90.0
self.puddles = clamp (self._t + delay, 0.0, 85.0)
self.wetness = clamp (self_t * 5, 0.0, 100.0)
self.wind = 5.0 if self.clouds <= 20 else 90 if self.clouds> = 70 else 40
self.fog = clamp (self._t - 10, 0.0, 30.0)
if self._t = -250.0:
self._increasing = True
if self_t = 100.0:
self, increasing = False
def_str (self):
return 'Storm (clouds =% d %%, rain =% d %%, wind =% d %%)'% (self.clouds, self.rain,
self.wind)
```

```
class Weather (object):
def_init __ (self, weather):
self.weather = weather
self._sun = Sun (weather.sun_azimuth_angle, weather.sun_altitude_angle)
self._storm = Storm (weather.precipitation)
def tick (self, delta_seconds):
self._sun.tick (delta_seconds)
self._storm.tick (delta_seconds)
self.weather.cloudiness = self._storm.clouds
self.weather.precipitation = self._storm.rain
self.weather.precipitation_deposits = self._storm.puddles
self.weather.wind_intensity = self._storm.wind
self.weather.fog_density = self_storm.fog
self.weather.wetness = self_storm.wetness
self.weather.sun_azimuth_angle = self._sun.azimuth
self.weather.sun_altitude_angle = self._sun.altitude
def_str_ (self):
return "% s% s'% (self._sun, self._storm)
```

collision sensors.

```
class CollisionSensor (object):
def_init_ (self, parent_actor, hud):
Constructor method
self.sensor = None
self.history = []
self._parent = parent_actor
self.hud = hud
world = self._parent.get_world ()
blueprint = world.get_blueprint_library (). find (sensor.other.collision ')
self.sensor = world.spawn_actor (blueprint, carla.Transform (), attach_to = self._parent)
# We need to pass the lambda a weak reference to
# self to avoid circular reference.
weak_self = weakref.ref (self)
self.sensor.listen (lambda event: CollisionSensor._on_collision (weak_self, event))
def get_collision_history (self):
w "Gets the history of collisions".
history = collections.defaultdict (int)
for frame, intensity in self.
history [frame] + = intensity
return history
@staticmethod
```

```
def_on_collision (weak_self, event):
On collision method
self = weak_self ()
if not self:
return
actor_type = get_actor_display_name (event.other_actor)
self.hud.notification ('Collision with% r'% actor_type)
impulse = event.normal_impulse
intensity = math.sqrt (impulse.x * 2 + impulse.y * 2 + impulse.z ** 2)
self.history.append ((event.frame, intensity))
if len (self.history)> 4000:
self.history.pop (0)
```

9 References

[1] Thibault Buhet; Emilie Wirbel; Xavier Perrotton , Conditional Vehicle Trajectories Prediction in CARLA Urban Environment , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/9022290 , 28.Oct/2019

[2] Tamara Naumovic; Marijana Despotovic-Zrakic; Božidar Radenkovic; Lazar Zivojinovic; Ivan Jezdovic , Development of a Continuous System Simulation Engine in Python Programing Language , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/9066334 , 20.March/2020

[3] Chen Chai; Xianming Zeng; Xiangbin Wu; Xuesong Wang , Safety Evaluation of Responsibility Sensitive Safety (RSS) on Autonomous Car-Following Maneuvers Based on Surrogate Safety Measurements , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/8917421 , 30.Oct/2019

[4] W. Farag , Z. Saleh , An Advanced Vehicle Detection and Tracking Scheme for Self-Driving Cars , https://digital-library-theiet-org.sdl.idm.oclc.org/content/conferences/10.1049/cp.2019.0222 , 26.March/2019

[5] Yan-Jyun Ou; Xiang-Li Wang; Chien-Lung Huang; Jinn-Feng Jiang; Hung-Yuan Wei; Kuei-Shu Hsu , Application and Simulation of Cooperative Driving Sense Systems Using PreScan Software , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/8479296 , 20.Nov/2017

[6] Xue-Mei Chen , Yi-Song Miao , Driving Decision-Making Analysis of Car-Following for Autonomous Vehicle Under Complex Urban Environment ,https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/7830353 , 9.May/2016

[7] Abdur R. Fayjie ,Sabir Hossain , Doukhi Oualid , Driverless Car Autonomous Driving Using Deep Reinforcement Learning in Urban Environment , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/8441797 , 30.Jun/2018

[8] Tian Tan, Tianshu Chu,Jie Wang , Multi-Agent Bootstrapped Deep Q-Network for Large-Scale Traffic Signal Control ,https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/9206275 . 26.Aug/2020

[9] Tanya Amert, Nathan Otterness, Ming Yang,James H. Anderson; F. Donelson Smith , GPU Scheduling on the NVIDIA TX2: Hidden Details Revealed , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/8277284 , 8.Dec/2017

[10] Mario Gluhaković : Marijan Herceg ; Miroslav Popovic ; Jelena Kovačević , vehicle Detection in the Autonomous Vehicle Environment for Potential Collision Warning , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/9161791 , 27.May/2020

[11] Any Gupta: Ayesha Choudhary, A Framework for Traffic Light Detection and Recognition using Deep Learning and Grassmann Manifolds, https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/8814062 , 12. Jun /2019

[12] Aleksandra Simić: Ognjen Kocić; Milan Z. Bjelica; Milena Milošević , Driver monitoring algorithm for advanced driver assistance systems , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/7818908 , 23.Nov. 2016

[13] Koji Kashihara: Deep Q learning for traffic simulation in autonomous driving at a highway junction , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/8122738 , 8.Oct / 2017

[14] Any Gupta: Ayesha Choudhary, A Framework for Traffic Light Detection and Recognition using Deep Learning and Grassmann Manifolds, https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/8814062 , 12.Jun / 2019

[15] Shaochi Hu: Huijing Zhao ; Mathieu Moze ; Francois Aioun ; Franck Guillemard , Human-like Highway Trajectory Modeling based on Inverse Reinforcement Learning , https://ieeexplore-ieee-org.sdl.idm.oclc.org/document/8916970 . 30.Oct / 2019

[16] Hussein, A.: García, F.; Armingol, J.M.; Olaverri-Monreal, C. P2V and V2P communication for Pedestrian warning on the basis of Autonomous Vehicles. In Proceedings of the 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, 1–4 November 2016; pp. 2034–2039.

[17] Alvarez, W.M.: Moreno, F.M.; Sipele, O.; Smirnov, N.; Olaverri-Monreal, C. Autonomous Driving: Framework for Pedestrian Intention Estimationin a Real World Scenario. arXiv 2020, arXiv:2006.02711.

[18] Dosovitskiy, A.: Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An Open Urban Driving Simulator. arXiv 2017, arXiv:1711.03938.

[19] Krajzewicz, D.: Hertkorn, G.; Feld, C.; Wagner, P. SUMO (Simulation of Urban MObility)—An open-source traffic simulation. In Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM20002); SCS Europe: Sharjah, UAE, 2002.

[20] De Miguel, M.Á.: Fuchshuber, D.; Hussein, A.; Olaverri-Monreal, C. Perceived Pedestrian Safety: Public Interaction with Driverless Vehicles. In Proceedings of the2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France, 9–12 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 90–95.

[21] Alvarez, W.M.: de Miguel, M.Á.; García, F.; Olaverri-Monreal, C. Response of Vulnerable Road Users to Visual Information from Autonomous Vehicles in Shared Spaces. In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 3714–3719.