



MODULAR APPROACH BASED SELF-DRIVING-CAR

A THESIS

SUBMITTED TO

THE DEPARTMENT OF ELECTRONICS AND ELECTRICAL
COMMUNICATIONS

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

BACHELOR OF SCIENCE IN ELECTRONICS AND ELECTRICAL
COMMUNICATIONS ENGINEERING

AMAL MOHAMED MOHAMED

MOHAMED MAHMOUD ALI

MOHAMED MOSTAFA ABDELATY

MOSTAFA ADEL KAMAL

OMAR MOHAMED ABDALLAH

ZAINAB KHALED HUSSIEN

UNDER SUPERVISION OF

DR. HASSAN MOSTAFA DR. SAMAH EL-TANTAWY

TECHNICALLY SPONSORED BY

VALEO EGYPT INC.

AUGUST 2020

Abstract

Artificial intelligence is a technology which enables a machine to simulate human behavior. Machine learning is a subset of AI which is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. But deep learning refers to a set of machine learning techniques that utilize neural networks with many hidden layers for tasks, such as image classification, speech recognition, language understanding. Many fields now depend heavily on AI & machine learning together to achieve the best technology based results, and one of the most important of them is the automotive industry. We have many potential applications in the automotive domain both inside the vehicle, e. g. advanced driving assistance systems (ADAS), and outside the vehicle, e.g. during development, manufacturing and sales & after sales processes. For the past few years, the task of building self driving vehicles has received a lot of concern and research, as it introduces a very efficient solution for many problems, by decreasing the driver probable mistakes, and letting the artificial intelligence be responsible for the driving decision making such as the steering angle adjustment, the required speed determination, the need for the brakes, etc.. ,these vehicles can help reduce the increasing number of car accidents drastically and save many lives, and also cut down the existence of traffic jams and saving us much time and effort.

Acknowledgments

This dissertation would not have been possible without the support of many people. First and foremost, we would like to thank our advisors and role models Dr. Hassan Mostafa, Dr. Samah EL-Tantawy, Eng. Mahmoud El Khateeb, and Eng. Mohamed Abdou.

Dr. Hassan Mostafa & Dr. Samah EL-Tantawy, we would like to thank you both for your patience, guidance, support and the great supervision you offered us through the whole project, and equipment you helped us access and use easily.

Eng. Mahmoud El Khateeb & Eng. Mohamed Abdou , you both provided perfect technical guidance, through the whole project, that helped us reach our desired objective, address various complex problems we faced, perform several advanced techniques. We would specially thank you for the tremendous amount of support and guidance you provided us.

Table of Contents

1	Introduction.....	7
1.1	What is a Self-Driving Car?	7
1.2	Levels of Automation of Self-Driving Cars:	7
1.2.1	Level 0 – No driving automation:	8
1.2.2	Level 1 – Driving assistance:	8
1.2.3	Level 2 – Partial driving automation:	8
1.2.4	Level 3 – Conditional driving automation:	8
1.2.5	Level 4—highly automated driving:.....	9
1.2.6	Level 5—Complete automation:	9
1.3	How they work:	9
1.4	Benefits of Self-Driving Cars:	11
1.4.1	Reduced emissions:	11
1.4.2	Road safety:	11
1.4.3	Less traffic:	12
1.5	Downsides of Self-Driving Cars:.....	12
1.5.1	Loss of jobs:.....	12
1.5.2	Moral machine:	12
1.5.3	Criminal hacking:	13
1.6	Objectives of The project:	13
2	Background	14
2.1	Simulators:.....	14
2.1.1	Popular private simulators:	14
2.1.2	Open source simulators:.....	15
2.2	Machine Learning:	17
2.2.1	Techniques of Machine Learning:.....	17
2.2.2	History of machine learning:.....	18
2.3	Deep learning:.....	19
2.3.1	History of Deep learning:	19
2.4	Computer vision:.....	19
2.4.1	History of computer vision:	20

2.5	Sensor Fusion:	21
2.5.1	Early Fusion:	21
2.5.2	Late Fusion:	22
2.5.3	Middle Fusion:	22
2.6	Sensor Fusion for 3D object detection:	23
2.6.1	Sensors used in Sensor Fusion:	23
2.7	Autonomous Driving Approaches:	25
2.7.1	End-to-End Approach:	25
2.7.2	Modular Approach:	25
3	Related work	27
3.1	The state of practice:	27
3.1.1	GM Motors:	27
3.1.2	Waymo (Google):	27
3.1.3	Yandex:	28
3.1.4	Tesla:	28
3.1.5	Almotive:	28
3.2	Related work similar to our scale part:	29
3.2.1	SIM-TO-REAL CONDITIONAL END-TO-END SELF-DRIVING VEHICLE THROUGH VISUAL PERCEPTION:	29
3.2.2	Autonomous Vehicle Control: End-to-end learning in Simulated Urban Environments: ..	30
4	Methodology	31
4.1	Project overview:	31
4.2	Modules:	32
4.2.1	Perception:	32
4.2.2	Localization:	53
4.2.3	Planner:	56
4.2.4	Mapping:	57
4.2.5	Controller:	62
4.2.6	Agent:	65
5	Results	69

Table of Figures

Figure 1: Early fusion	21
Figure 2: Late Fusion.....	22
Figure 3: Middle Fusion	22
Figure 4: End-to-End Approach.....	25
Figure 5: Modular Approach.....	26
Figure 6: Modules block diagram	31
Figure 7: RGB Sample	34
Figure 8: Depth Sample	35
Figure 9: LiDAR point cloud	35
Figure 10: 3D Bounding Boxes sample	36
Figure 11: Cartesian and spherical coordinates.....	37
Figure 12: LiDAR Point cloud after filtration.....	37
Figure 13: 2D Bounding Boxes.....	40
Figure 14: LaserNet Architecture.....	43
Figure 15: LaserNet block architecture	43
Figure 16: sample of prediction output from pre-trained YOLO	46
Figure 17: Tiny YOLO loss vs epochs	47
Figure 18: Distance Estimation	48
Figure 19: Distance estimation sample	50
Figure 20: decide if the road isn't open	51
Figure 21: the road is open	52
Figure 22: the road in the left isn't open	52
Figure 23: the road in the left is open	53
Figure 24: Road layout in OpenDrive	58
Figure 25: Lane ID in OpenDrive	58
Figure 26: Section ID for same separation in OpenDrive.....	59
Figure 27: Section ID for different separation in OpenDrive	59
Figure 28: PID Block Diagram	65
Figure 29: Agent block diagram.....	66

1 Introduction

1.1 What is a Self-Driving Car?

Self-driving vehicles are cars or trucks within which human drivers are never required to require control to securely operate the vehicle. Also referred to as autonomous or “driverless” cars, they combine sensors and software to regulate, navigate, and drive the vehicle.

Though still in its infancy, self-driving technology is becoming increasingly common and will radically transform our transportation (and by extension, our economy and society).

Currently, there are not any legally operating, fully autonomous vehicles. There are, however, partially autonomous vehicles /cars and trucks with varying amounts of self-automation, from conventional cars that have privileges such as brake and lane assistance to independent self-driving prototypes.

1.2 Levels of Automation of Self-Driving Cars [1]:

Self-driving cars are graduating through several levels of automation, each of which incorporates a different level of technological input starting from zero (no autonomous features) to full autonomous (not needing a driver). The SAE (Society of Automotive Engineers) International - organization within the field of engineering- created a customary for levels of automation. Each automation level relates to two key metrics:

1. *The autonomous capabilities of the car.*
2. *The kind of interaction between the car and therefore the driver.*

The five levels of automation are:

1.2.1 Level 0 – No driving automation:

A human driver is fully chargeable for operating the vehicle. However, the system may send alerts to warn the motive force of any perceived danger.

1.2.2 Level 1 – Driving assistance:

An automated driving system (ADS) assists the human driver by performing the driving task. The ADS can do small tasks like steer the wheel or control the driving speed, however it cannot do both tasks at the same time . Adaptive control (ACC) falls into this category, because it handles just braking (to keep a specified distance from the car before of you), but not steering.

1.2.3 Level 2 – Partial driving automation:

A human driver and one or more ADS features share the responsibility for operating the car. The ADS can steer the wheel and control the driving speed simultaneously while the motive force monitors the performance of the ADS and cannot take their eyes off the road.

1.2.4 Level 3 – Conditional driving automation:

While a driver remains required to be within the car, they are not expected to, bear in mind of everything in any respect times like with Level 2 and 1 automation. The ADS can drive the car and monitor the environment. Human drivers are unengaged to take their eyes off the road but have to remain within the driver's seat to reply to any ADS request and intervene.

1.2.5 Level 4—highly automated driving:

The ADS can drive the car, monitor the environment, and does not require human interference. However, at this level, the ADS is proscribed to geo-fenced areas or special circumstances like those that traffic jams. In other words, you will not be required to require over when all the conditions are right. However, if it is raining or snowing, then the vehicle will not allow you to interact self-driving.

1.2.6 Level 5—Complete automation:

The car is fully autonomous .It requires no human intervention and might drive on roads and in diverse environmental conditions. There is not even a wheel. The earliest Level 5 vehicle is already on the road! In addition, this can be the most topic of our project. Top of Form

1.3 How they work:

Google, Uber, Tesla, Nissan, and other major automakers, researchers, and technology companies develop various self-driving technologies [2].

While design details vary, most self-driving systems create and maintain an interior map of their surroundings, supported a good array of sensors, like radar. Uber’s self-driving prototypes use sixty-four laser beams, together with other sensors, to construct their internal map; Google’s prototypes have, at various stages, used lasers, radar, high-powered cameras, and sonar.

GPS systems, sensors, and radars help the car get a transparent and live picture of the road. Laser Illuminated Detection and Ranging (LiDAR) technology helps the vehicle orient and interact with surrounding objects and infrastructure sensors. Cars equipped

with Dedicated Short Range Communications (DSRC) can connect with one another via an ad-hoc network.

Advanced control systems process the geospatial information and make decisions supported the data. For instance, once the car receives information about the road conditions, the advanced system charts a route and tells the vehicle actuators the way to drive. Some control systems have intelligent object detection methods that help the system navigate through obstacles.

Software then processes those inputs, plots a path, and sends instructions to the vehicle's "actuators," which control acceleration, braking, and steering. Hard-coded rules, obstacle avoidance algorithms, predictive modeling, and "smart" object classification that help the software obey traffic rules and navigate through obstacles.

Partially autonomous vehicles may require somebody's driver to intervene if the system encounters uncertainty; fully autonomous vehicles might not even offer a wheel.

Self-driving cars is further distinguished as being "connected" or not, indicating whether or not they can communicate with other vehicles and/or infrastructure, like next generation traffic lights. Most prototypes do not currently have this capability.

1.4 **Benefits of Self-Driving Cars:**

Self-driving cars can have tremendous benefits for people and societies. Here are three of the most benefits [1].

1.4.1 **Reduced emissions:**

The US transportation sector produces 30% of all U.S warming emissions. Despite the dire environmental consequences of emissions, more vehicles are deployed daily, partly thanks to the demand for more delivery trucks for last-mile delivery. Traffic jams, excessive speed, and braking and re-accelerating contribute to pollution.

Self-driving cars help reduce the pollution emitted by vehicles. Autonomous capabilities like consistent driving speeds and keeping a measured distance between vehicles can reduce unnecessary braking and re-acceleration. Electronic models of self-driving cars with an electrical or hybrid engine further reduce pollution by eliminating or lessening the employment of fuel.

1.4.2 **Road safety:**

Road accidents end in 1.25 million deaths and 20-50 million injuries worldwide. If nothing changes, road traffic injuries may become the fifth leading reason for death by 2030.

Self-driving cars is also the answer to road accidents. Since human error is that, the reason for around 90% of traffic accidents, delegating most or the entire vehicle operating responsibilities to the ADS can increase road safety. While fully automated vehicles have not been commercially adopted yet, many cars today are equipped with sensors and advanced systems that alert drivers to dangers.

1.4.3 Less traffic:

Research by INRIX ranks the U.S because the fifth most congested country, with Thailand within the lead for the title of “World’s Most Congested Country”. Drivers within the U.S spend a median of 41 hours a year in traffic jams.

Self-driving car technology, like connected cars, may offer an answer to clogged roads. Connected cars can communicate with one another can help optimize routes for every individual vehicle, creating a network of data that helps distribute traffic flow.

1.5 Downsides of Self-Driving Cars:

Like all kickshaws, autonomous vehicles have a downside. Here are three reasons autonomous cars may encourage be detrimental or perhaps dangerous.

1.5.1 Loss of jobs:

The ultimate goal of self-driving cars is to delegate the responsibility of driving to a machine. When fully autonomous self-driving cars are sold commercially, people who depend on driving for his or her income may lose this way of livelihood. Those who manage a fleet of drivers is also terminated, alongside dedicated training facilities and services like driving lessons and licenses.

1.5.2 Moral machine:

Fully autonomous self-driving cars have to be ready to make moral decisions, together with technical driving decisions.

The first question is self-driving cars capable of constructing swift moral judgment calls to the identical level of somebody's being. If someone jumps before of the car, and the car is left with the dilemma of hitting the person or swerving to the side, what is going to

the car choose. The second question is whether if the car is indeed capable of creating moral decisions, should it has allowed trying to do so?

1.5.3 Criminal hacking:

Car thieves once used physical objects to unlock a car; nowadays thieves use technology to hack the car system.

Ironically, as cars become more sophisticated, their systems provide cybercriminals with more hacking opportunities. A cyber-attack on a completely automated self-driving car may result in traffic jams, fatalities thanks to traffic accidents, or maybe function an entry point to local traffic systems bottom of form

1.6 Objectives of The project:

The goal at the end of this project is to be able to implement a fully functional and completely automated Self-Driving car (Level 5) that should not require any human interference to achieve the driving task successfully. This project depends heavily on the AI and Computer vision fields that are core in any Self-Driving car project and with the help of Machine & Deep learning this can be achieved very efficiently, as they can lead to very fast and accurate estimations of the surrounding environment to make the critical decisions that are required to fulfill the driving task.

2 Background

2.1 Simulators:

Engineers will need to meet high standards to get the autonomous vehicles on the road. We cannot use a real car on the road because it will take time and large budgets so that we need simulators to test different approaches and sensor outputs in autonomous driving .

We can divide simulators into two types:

- Companies' simulators, which is company specific and is not publicly available.
- Open source simulators, which anyone can download it and use it.

2.1.1 Popular private simulators:

2.1.1.1 Waymo:

Waymo performs a mix of simulation, closed course testing in their facility and real world testing. Their simulation environment does not cover the perception problem because they do not use realistic graphics but they have 4 generations of self-driving vehicles with a lot of cities and self-driven miles simulated .

2.1.1.2 Uber:

Uber provides a simulation environment for autonomous vehicles. Their simulation environment does not cover the perception problem similar to Waymo but Uber provides open source visualization toolsets for analyzing mobility and geography.

2.1.1.3 NVIDIA:

NVIDIA provides expertise in simulation, hardware and data analysis. NVIDIA provides simulation technology for autonomous systems level 2-5. They have support from many big cars' companies like Toyota and Audi.

2.1.2 Open source simulators:

2.1.2.1 AirSim:

AirSim is open source simulator for autonomous systems created by Microsoft. The simulator is useful for AI, deep learning, computer vision and reinforcement learning. It provides realistic vehicle dynamics and environments for cars, and more through Unreal Engine and Unity game engines.

2.1.2.2 CARLA:

CARLA is open source simulator for self-driving cars developed in partnership between Intel Labs, Toyota Research Institute and the Computer Vision Center, Barcelona. It is a service oriented, high fidelity, realistic graphics environment built on top of Unreal Engine. The simulation environment allows testing from perception to vehicle control. There are another open source simulators like gazebo but it is less efficient or does not specialize in autonomous driving industry [3].

We choose Carla as our simulator in this project because it is easy to use, its documentation easy to understand and has many functions to facilitate simulation and testing.

CARLA simulates a dynamic world and provides a simple interface between the world and an agent that interacts with the world. To support this functionality, CARLA is

designed as a server-client system, where the server runs the simulation and renders the scene. The client API is implemented in Python and is responsible for the interaction between the autonomous agent and the server via sockets. The client sends commands and meta-commands to the server and receives sensor readings in return. Commands control the vehicle and include steering, accelerating, and braking. Meta-commands control the behavior of the server and are used for resetting the simulation, changing the properties of the environment, and modifying the sensor suite. Environmental properties include weather conditions, illumination, and density of cars and pedestrians. When the server is reset, the agent is re-initialized at a new location specified by the client.

After talking about why Carla is the best choice for our work? Now it is the time to choose the version of Carla. The two versions of Carla was Carla 0.8.x and Carla 0.9.x

2.1.2.2.1 Carla 0.8.x:

It was the start of Carla deployment as it has many sensors like LIDAR and camera and many shapes of vehicles and pedestrians. The version developed later to fix issues like improvement of how vehicles are spawned to better handle spawning failures or improvement of increasing the fps. However, it misses also many features so we did not use this version in our work and we headed to Carla 0.9.x version.

2.1.2.2.2 Carla 0.9.x:

It is the new version of Carla and has many new features which added to the simulator like new shapes of vehicles and pedestrians which are more realistic, possibility to add and remove at any time any vehicle or camera and adding the synchronous mode which we used in our work. In addition, it has very important feature and it is essential

point in our work which it has multiple lanes so we have the ability to change the lane of our car. We choose Carla 0.9.6 as our simulator in our work because of its new features, which make our work easy and more realistic.

2.2 Machine Learning:

As a definition, machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Another definition of ML is that a computer learns to do some task T with a measure of performance P through training on dataset the computer's performance is improved going through this data in a process called training typically machine learning has four methods.

2.2.1 Techniques of Machine Learning:

2.2.1.1 Supervised machine learning:

When the data used for training is labeled, we call this supervised learning. As the computer will see a sample of an input and its corresponding output and from this it can learn that whenever it sees this input or something alike the output would be similar to the output of the training input by going through sufficient amount of data the computer will learn to generalize.

2.2.1.2 Unsupervised machine learning:

We usually go to unsupervised machine learning when we do not have labeled data, in this process the computer learns through the hidden structure of the input (training) data. In this case, the computer does not give labels to the data instead, it draws boundaries

between the different subsets of the data in conclusion it can separate them but it does not give them labels

2.2.1.3 Semi-supervised machine learning:

When we do not have enough labeled data and we then use semi-supervised algorithms, in which we have both labeled and unlabeled data we make a model by the few labeled data. Then we go through the unlabeled group of data producing labels to them and then train the model again. Then go to new data and so on. There is not a way to make sure that the model gives accurate labeling and surely, it will be less accurate than the rest of unsupervised algorithms.

2.2.1.4 Reinforcement learning:

Is a learning method that interacts with its environment by producing actions and discovers errors or rewards i.e. The model make action then by reward signal it can know wither it is the best action or not and by trial and error, it can reach the best action.

2.2.2 History of machine learning:

The first case of neural networks was in 1943, when neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper about neurons, and how they work. They decided to create a model of this using an electrical circuit, and therefore the neural network was born. Machine learning's improvement stops because of the lack of data and hardware resources until the beginnings of the 21 century when we had huge amount of data and good hardware resources and new algorithms.

2.3 Deep learning:

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces. Generally deep learning is a process that try to imitate the thinking process by extracting features (which is some special things about the inputs) as we go through layers which can simulate the we human abstract feature

2.3.1 History of Deep learning:

It is close to the history of machine learning it most came out to the light when In 1970's, back propagation, was developed which uses errors into training deep learning models. Back propagation became popular when Seppo Linnainmaa wrote his master's thesis

2.4 Computer vision:

Computer vision is a field that deals with how much the computer can gain high level understanding of an image or a video; it takes into consideration how humans themselves understand those things so it is some sort of interdisciplinary field. It seeks to automate tasks that the human visual system can do by the development of a theoretical and algorithmic basis to achieve automatic visual understanding. Computer vision is concerned with the theory behind artificial systems that extract information from the tasks of computer vision are either object detection which is knowing exactly what objects do you have in an image and where are them of object classification which is

only knowing what are the objects in the image typically the detection problem is much harder

Images, in computer vision there are two techniques.

1. Classical techniques, in which you can extract features yourself there are many classical techniques that can do this very accurate but its problem is that you have to choose which features to look for.
2. Deep learning and AI techniques in which you use the deep learning models to extract hidden features in an image so you can classify and detect objects in it.

2.4.1 History of computer vision:

Computer vision started to take shape as a field in the 1960s, with the goal of simulating the human visual system and tell us what they see, i.e. automating the process of image analyses which was the precursor to artificially intelligent image recognition

In 1959 we could deal with the first digital image ever when Russell Kirsch and his colleagues developed an apparatus that allowed transforming images into grids of numbers—the binary language machines could understand .then through the years we developed algorithms to enhance the ability of computer to describe what it sees.

2.5 Sensor Fusion:

Sensor fusion is an approach for combining data delivered from disparate sources (sensors in autonomous driving) such that the coherent information is created [4],[5],[6]. The resulting information is more accurate than when these sources were used individually. For example, on the autonomous vehicle, it is important to have a camera in order to clone a human vision, but the information of the obstacle distance will be the best gained through the sensors as LiDAR or radar. For that reason, sensor fusion of camera with LiDAR or radar data is very important since there are complementary. On the other hand, combining information from LiDAR and radar will provide more certain information about the distance of the obstacle ahead of the vehicle or general distance of the objects in the environment.

2.5.1 Early Fusion:

The idea of the Early Fusion approach is that the neural network should learn how to fuse the different sensor data without any human engineered features or special net configurations. Instead, the sensor data are combined to a common input tensor and processed together in the entire network.

The disadvantage of Early Fusion is that the outputs of the different sensor types vary, and each sensor has sensor specific properties. These sensor specific properties might be lost by an early fusion.

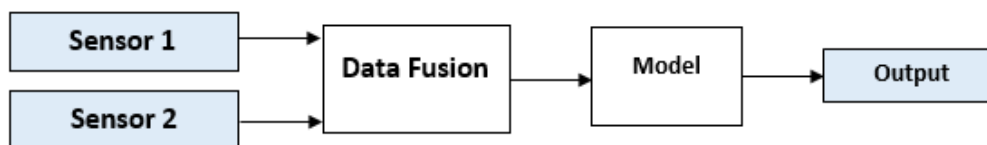


Figure 1: Early fusion

2.5.2 Late Fusion:

Another net architecture is considered which overcomes Early Fusion problem by first processing the sensor data separately before they are finally fused. The Late Fusion approach, also allows the use of different dimensions of input data. Late fusion often gives better performance because errors from multiple models are dealt with independently.

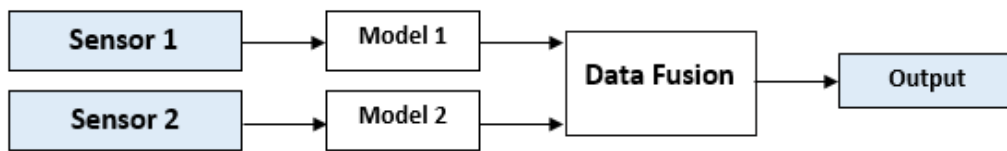


Figure 2: Late Fusion

2.5.3 Middle Fusion:

Middle Fusion is a combination of Early Fusion and Late Fusion. By this approach, the distinct sensor data are first processed independently, and are concatenated at a Middle stage. Then, the concatenated sensor data are processed further by network.

The Middle Fusion approach enables the network to learn sensor specific properties and then how to combine these properties to suitable features for a good and robust object detection.

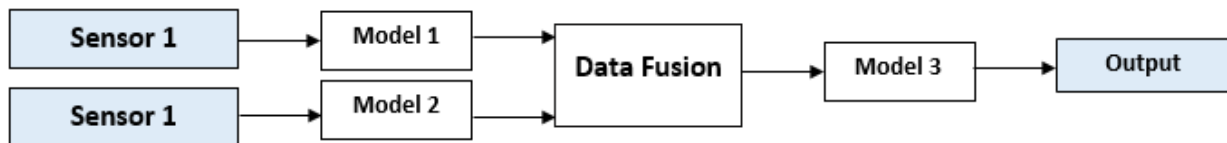


Figure 3: Middle Fusion

2.6 Sensor Fusion for 3D object detection:

Current trends in autonomous vehicles development showed increased usage of the LiDAR. Sensor fusion from camera and LiDAR data gives an optimal solution in terms of hardware complexity of the system, only two types of sensors are integrated, and the system coverage, camera for vision and LiDAR for obstacle detection complement each other. Here the image data is fused with 3D point cloud data, and as a result, 3D box hypothesis and their confidence are predicted [7].

2.6.1 Sensors used in Sensor Fusion:

2.6.1.1 Camera:

While perception in autonomous vehicles is achieved with many sensors and sensor systems, camera was one of the first types of sensors to be used in driverless vehicles and is currently the main choice for car manufacturers. The domain of application includes perception, semantic segmentation, end-to-end autonomous driving, and many others.

Camera enables an autonomous vehicle to literally visualize its surroundings. They are very efficient at the classification of texture interpretation, are widely available, and more affordable than radar or LiDAR.

The disadvantage of a camera is that it has a limited FOV and gives us no depth information.

2.6.1.2 LiDAR:

LiDAR stands for light detection and ranging, is a sensor that throws out laser rays and provides points in the environment around based on the light rays that reflects on the surroundings and come back. Lidar uses an infrared laser beam to determine the distance between the sensor and a nearby object.

LiDAR provides a 360° horizontal field of view (FOV) and a limited vertical FOV. The main advantage of a LiDAR is that it gives highly accurate depth values. But, it doesn't give a very high resolution output.

Lidar has a much higher spatial resolution than radar because of the more focused laser beam, the larger number of scan layers in a vertical direction, and the high density of LiDAR points per layer. This type of LiDAR cannot measure the velocity of objects directly and have to rely on the different position between two or more scans. LiDAR are more affected by weather conditions and by dirt on the sensor.

2.7 Autonomous Driving Approaches:

2.7.1 End-to-End Approach:

End-to-end autonomous driving is defined as a single, self-contained driving system that carries out all processes automatically, from mapping based on sensor input, such as a front-facing camera, to the actions necessary for driving, such as steering, braking and acceleration. An end-to-end autonomous driving system is often designed to learn from expert demonstrations rather than depend upon manually-designed tasks and modules.

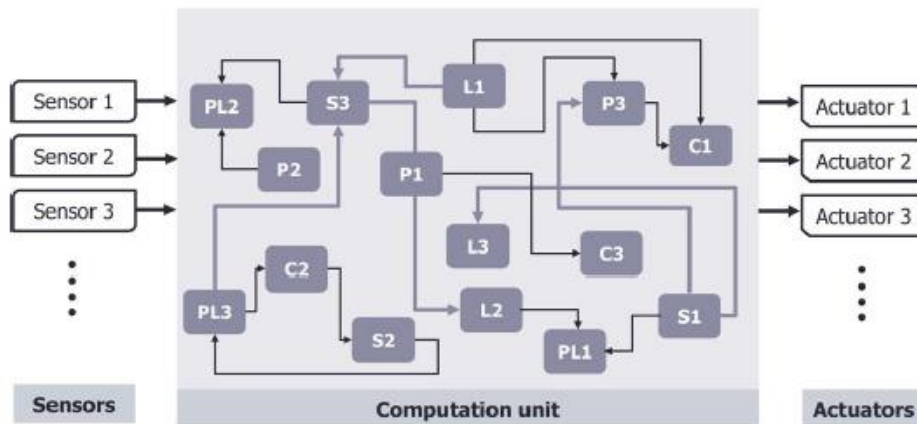


Figure 4: End-to-End Approach

2.7.2 Modular Approach:

The Modular approach subdivides the system into sub modules (Perception, Localization, Planning, Prediction and control). Then, each individual module is constructed by using a deep learning approach. Integrating all these modules together gives the vehicle the ability to take decisions on its own without human interference.

There are many benefits of the Modular approach compared with the End-to-End. The computational complexity of the system can be reduced and the computational stability and efficiency of the system can improve. Therefore, we intend to use the Modular approach to build our self-driving car.

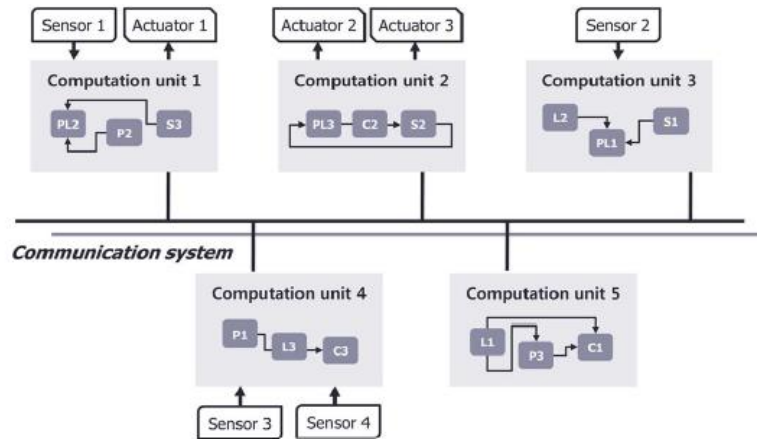


Figure 5: Modular Approach

3 Related work

We can divide the related work of our project into two parts: the state of practice part, which we will talk about what the big companies did in the autonomous industry and related work similar to our scale part, which we will talk about research papers similar to us in scale and testing [8].

3.1 The state of practice:

The competition between the top car companies to launch their first self-driving car is on. In 2009, when Google announced to start their research in self-driving, the concept became majorly popular. Here is some of the latest update on their technologies and business strategies.

3.1.1 GM Motors:

They are currently working on Chevrolet Cruise AV; it is a fully autonomous car without intervention of human. They have been testing many of their versions in the traffic of New York City and Michigan. Chevrolet Cruise AV works on the intelligence of the processors on board in accelerating, braking and making other essential driving decisions. It also allows the passengers to engage with the car.

3.1.2 Waymo (Google):

Waymo is testing their cars in Phoenix, Arizona. The Waymo self-driving car will have LIDARS, cameras, radars and microphones to detect different sounds like ambulance sirens. Waymo depends on its deep learning algorithms to make the perception module in

their autonomous cars. The touch screen behind the front seats let travelers see the vehicles around in blue.

3.1.3 Yandex:

Yandex is a company, which is known for its expertise in machine learning, cloud-based technologies and mapping techniques, it is one of the pioneers in this field. Recently, it announced the possible release of a self-driving robot called Rover capable of delivering goods automatically without any assistance by next year. Yandex cars are currently running in the heavy traffic of Moscow and Las Vegas. They have completed more than 4000 passenger rides and have crossed one million miles of autonomous driving.

3.1.4 Tesla:

Tesla cars use powerful radar and ultrasonic sensors to view their surroundings. The cameras and deep learning algorithms do the rest of the work. Tesla is way ahead of all the other competitors with completion of 1.88 billion autonomous miles this October. Another thing to look out for is the combination of laser, ultrasonic and passive video, which is cheaper than LIDARS. Tesla cars have thus a real competitive advantage over other companies and are set to launch a self-driving car.

3.1.5 AIotive:

There are also big companies, which released software stacks for autonomous driving like AIotive Company. AIotive, the autonomous technology service provider has announced the next generation of its AI-powered self-driving full-stack software, aiDrive 2. Through a highly modular design similar to our work but on a practice scale, aiDrive2 will serve as a platform for deeper collaboration in the autonomous industry. aiDrive2's

modular design will open the door to a wide range of implementations reducing the time to market of automated driving solutions.

3.2 Related work similar to our scale part:

There are many projects, which are similar to us in scale and use Carla as a testing environment. They used different metrics to measure the performance of their work like number of successful trips or number of collisions. The papers talk about end to end approach unlike our approach and modular approach papers almost not present but the metrics of measuring performance of autonomous driving doesn't differ from approach to another, so let's talk about some of this paper and what was their work.

3.2.1 SIM-TO-REAL CONDITIONAL END-TO-END SELF-DRIVING VEHICLE THROUGH VISUAL PERCEPTION:

This project was a graduation project to students from faculty of engineering Cairo University. The project was mainly about testing and evaluating an end-to-end approach trained using simulated data on a simulated environment, targeting level 5 autonomy and achieving a state-of-the-art success rate in different driving scenarios, without any pre or post processing. To prove their approach effectiveness, they used CARLA simulator that offer several trips with gradual increasing difficulty. Their evaluating metric is the success rate, which is the number of successful trips to the total number of defined trips. They compared our results over 50 trips in different towns and different weather conditions and the results were very satisfying.

3.2.2 Autonomous Vehicle Control: End-to-end learning in Simulated Urban Environments[9]:

This paper examines end-to-end learning for autonomous vehicles in simulated urban environments containing other vehicles, traffic lights, and speed limits. Furthermore, the paper explores end-to-end systems' ability to execute navigational commands and examines whether improved performance can be achieved by utilizing temporal dependencies between subsequent visual cues. They trained two different nets CNN and CNN-LSTM for the project and after 100 epochs of training, both models' training and validation loss had stabilized. The models were then evaluated on the unseen test set from CARLA's second town. On the test set, the CNN was able to predict a control signal with an average error of 0.023, a 43% increase compared to its validation error. The CNN-LSTM was able to predict a control signal with an average error of 0.022, a 57% increase compared to its validation error. The results were good and indicate that end-to-end systems are able to operate autonomously in simulated urban environments.

4 Methodology

4.1 Project overview:

The figure below shows the abstract block diagram of our project. The perception module will take the inputs of the camera and LiDAR sensors as RGB image and point cloud then by using the concept of the sensor fusion the output from this module will be the Lane follow and the lane change flags which will help the car to know what are the objects surrounding it and where are they. Then the output of perception module will be the input to our agent which will take also inputs from localization like current location for my car and my destination. The agent will output the next start and destination waypoints of my road and gives them to planner module which will output the PID parameters used in the controller module. The last module will be the controller which will use the PID controller to output the steering and throttle angles to move the car.

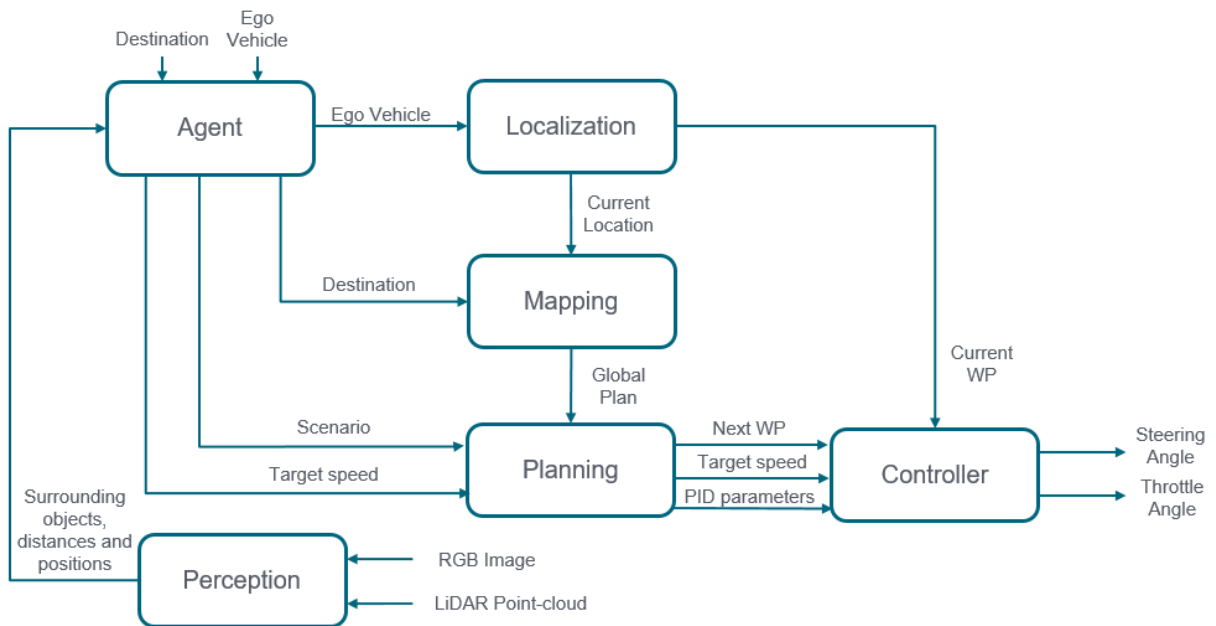


Figure 6: Modules block diagram

4.2 Modules:

4.2.1 Perception:

Perception is a fundamental module to help the autonomous vehicles to collect information about the driving environment, including the free drivable areas and surrounding obstacles' locations, velocities, and even predictions of their future states. Based on the sensors implemented, the environment perception task can be tackled by using LiDARs, cameras, or a fusion between these two kinds of devices. Some other traditional approaches may also involve the use of short/long-range radars and ultrasonic sensors. We used LiDAR and camera to achieve the tasks we want from perception module, so we need to prepare a good data from these sensors and use the data for our module.

We prepared our data after two stages: data harvesting and data filtering.

4.2.1.1 Data Collection:

As we are making our model work in a simulation environment, we had to train the model with dataset from the same distribution of this environment. Therefore, we had to make our own annotated dataset from the same environment that we expect our model to work in. Data collection was divided into three main phases, Data harvesting, Data filtering, and Data preprocessing and we will discuss each phase in the next section.

The annotated data is desired to be RGB image with size $(1920 \times 640 \times 3)$, and

LiDAR point-cloud described as a list of points of format (x, y, z) in Cartesian coordinates, referenced from the center of the LiDAR sensor on the top of the vehicle.

The annotation is desired to be n lists where n is the number of classes that we want our

model to train to predict them, each list is describing the position of each object in the RGB image as a bounding box described as $(x_{min}, x_{max}, y_{min}, y_{max})$ where x is the horizontal number of the pixel and y is the vertical number of the pixel. So each annotation list will be of size $(m \times 1 \times 4)$, where m is number of existed objects of the class in the RGB image.

4.2.1.1.1 Data Harvesting:

In this stage, all valuable information about the scene was captured by using Carla API's without any filtrations. The RGB camera, LiDAR, and depth camera were spawned, on the top of the vehicle. And the data acquisition process is launched by choosing an ego vehicle in the world, spawns the sensors on top of it, capture about 10 frames, then chooses another random ego vehicle in the scene and repeat the process. Each frame we capture we save the RGB image, the depth image, the LiDAR point-cloud, the bounding boxes of all objects in the world, referenced to the point of view of the ego vehicle, the speed of the ego vehicle and the time-stamp of the frame. Figure 7: RGB Sample is a sample of the RGB image captured. Figure 8: Depth Sample is sample of depth image captured.

is a sample of the LiDAR point cloud captured, and Figure 10: 3D Bounding Boxes sample is a sample of bounding boxes obtained from Carla engine for all vehicles in the scene, transformed into the coordinates of the ego vehicle.



Figure 7: RGB Sample



Figure 8: Depth Sample

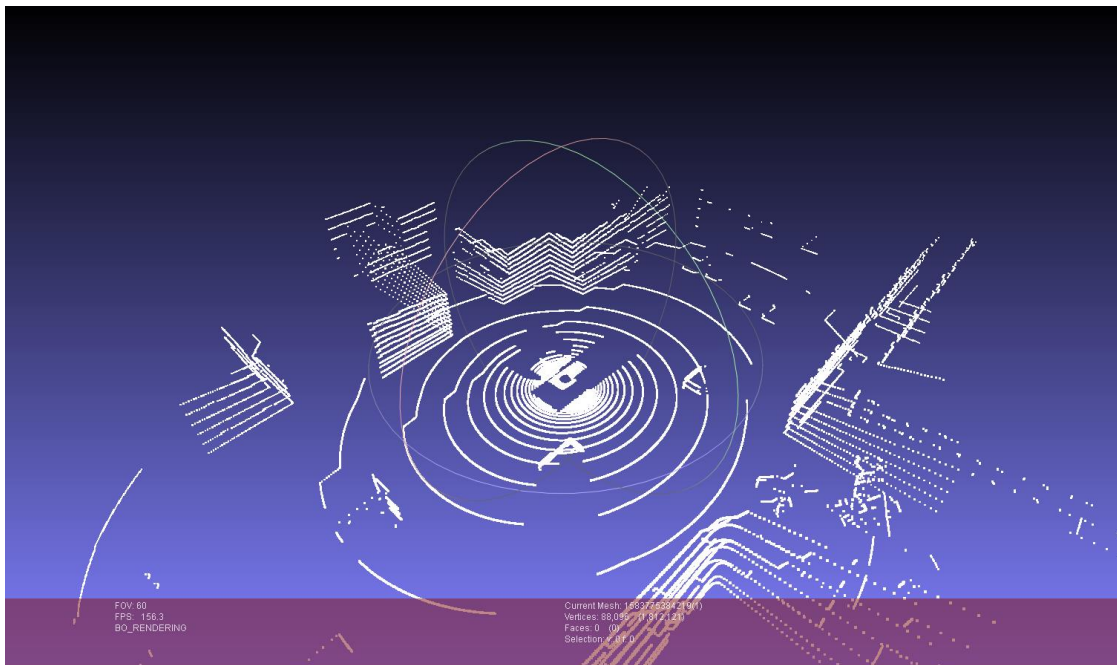


Figure 9: LiDAR point cloud

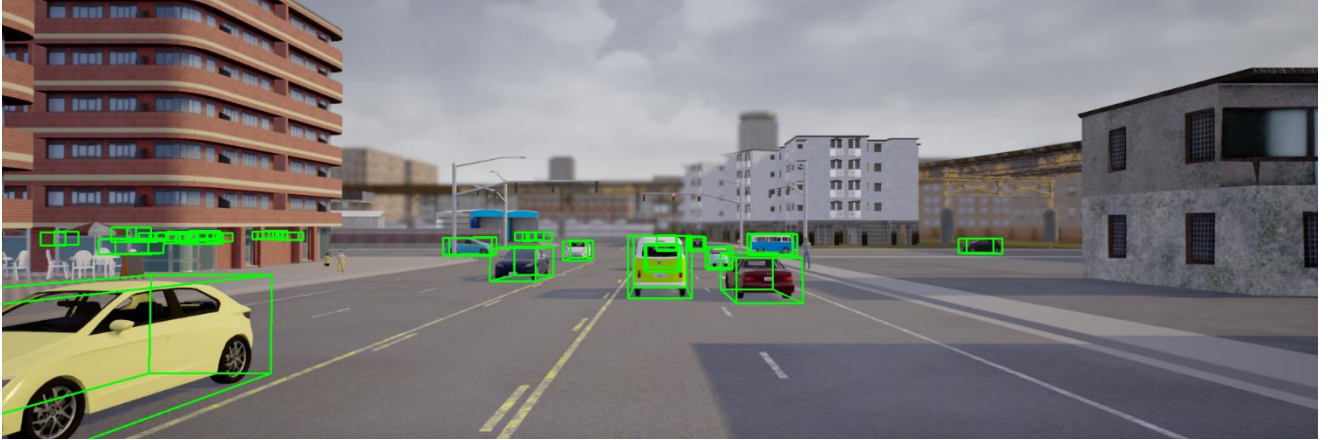


Figure 10: 3D Bounding Boxes sample

4.2.1.1.1.1 Data Filtering:

In this stage, we filter out the unwanted information about the scene, and this filtration is done for bounding boxes and the LiDAR point-cloud.

4.2.1.1.1.2 Filtering LiDAR point-cloud:

LiDAR points obtained from Carla LiDAR sensors are described in Cartesian format where the origin of the cloud is the sensors center. The point-cloud has some repeated points that need to be filtered to reduce the size of the list to ease the operations on the cloud. In addition, points that are outside the field-of-view of the RGB camera will not be used for the sensor fusion stage; therefore, they must be filtered out. First, we transform all points in the cloud to the spherical representation (r, θ, ϕ) where this representation describes the point as shown in 11 where:

$$r = \sqrt{x^2 + y^2 + z^2} \quad (1)$$

$$\theta = \tan^{-1}\left(\frac{y}{x}\right) \quad (2)$$

$$\phi = \cos^{-1}\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right) \quad (3)$$

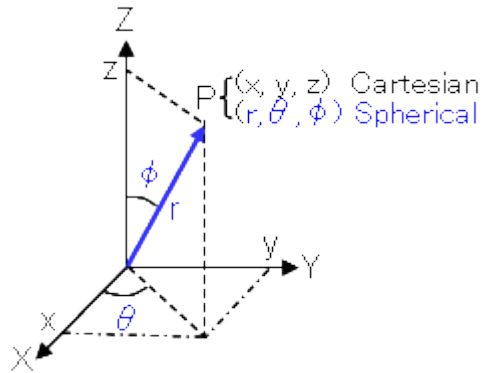


Figure 11: Cartesian and spherical coordinates

After converting to spherical coordinates, we start filtering points by its θ , so if θ is within the range of the camera FOV this point is valid and if not, it will be filtered out. Results of this stage is shown in Figure 12: LiDAR Point cloud after filtration

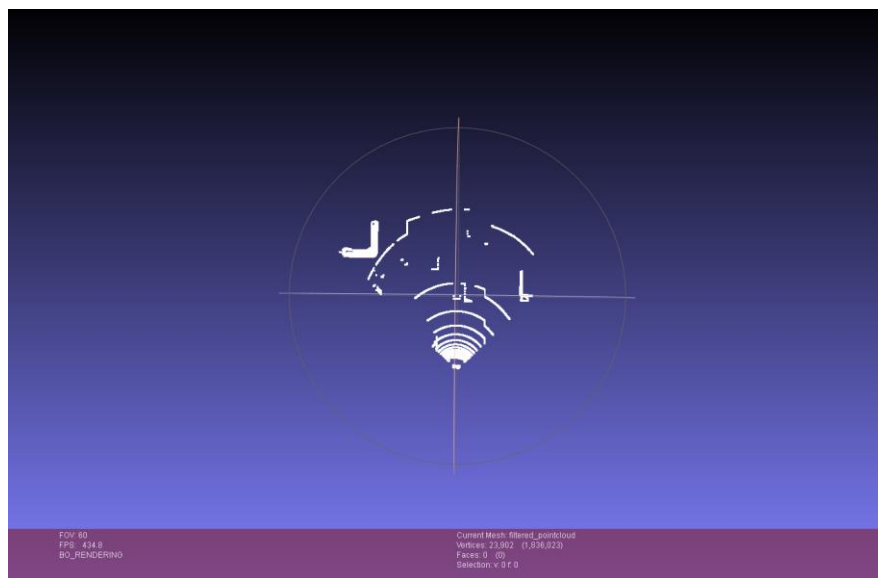
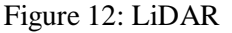


Figure 12: LiDAR Point cloud after filtration

4.2.1.1.1.3 Filtering bounding boxes:

As Carla engine returns the 3D bounding boxes of all objects in the world, and referenced to the point of view of the RGB camera, this led to having some invalid bounding boxes that needed to be filtered out as shown in  Figure 12: LiDAR Point cloud after filtration

All bounding boxes is in the form of a list of size (8×3) where 8 is the number of corners of the 3D bounding box is, and 3 is the number of coordinates (x, y, z) of each corner.

First, we filter out all bounding boxes that lay outside the FOV of the RGB camera, by checking if all corners of a box are outside the FOV or at least one point is inside the FOV then this box must be filtered.

Second, we use the information from the depth camera where, the depth image has the same size of the RGB image, and each pixel has a value in range of $[0: 255]$ that represents the depth distance in the scene. So we get the position of each corner for each bounding box in the pixel coordinates in the RGB image, and look at the same position in the depth image and get the depth value(d), and compare it to the z value of this corner, if $z < d$ then then this corner is in the visible view of the camera. If a box has five or more visible corners then it will be a valid box, otherwise it will be filtered.

After filtering the boxes in the scene and determine the valid bounding boxes, we transform the 3D valid bounding boxes into 2D bounding boxes as follow:

$$x_{min} = \min(\text{all } x \text{ locations of all corners})$$

$$y_{min} = \min(\text{all } y \text{ locations of all corners})$$

$$x_{max} = \max(\text{all } x \text{ locations of all corners})$$

$$y_{max} = \max(\text{all } y \text{ locations of all corners})$$

Third, if a box has some corner outside the size of the camera RGB image or outside the visible view, then we had to handle this case by limiting the bounding box to fit only the visible area of the car. This case may happen if a part of the vehicle was behind a building or only a part of it is visible.

Fourth, if there are two cars that are very close to each other and their boxes has too much IOU "Intersection over Union", then they can be treated as one vehicle, and put only one bounding box that contains them both.

Finally, if a vehicle is too far so that its box is very small, this case will not be very useful to train our model over it. Therefore, we will filter the very far boxes, by calculating the area of the box with the following formula:

$$\text{Area} = (x_{\max} - x_{\min}) \times (y_{\max} - y_{\min})$$

In addition, if the area is smaller than some threshold, that was calculated with trial-and-error technique. Then this box has to be filtered.

Figure 13: 2D Bounding Boxes, shows the 2D filtered bounding boxes.

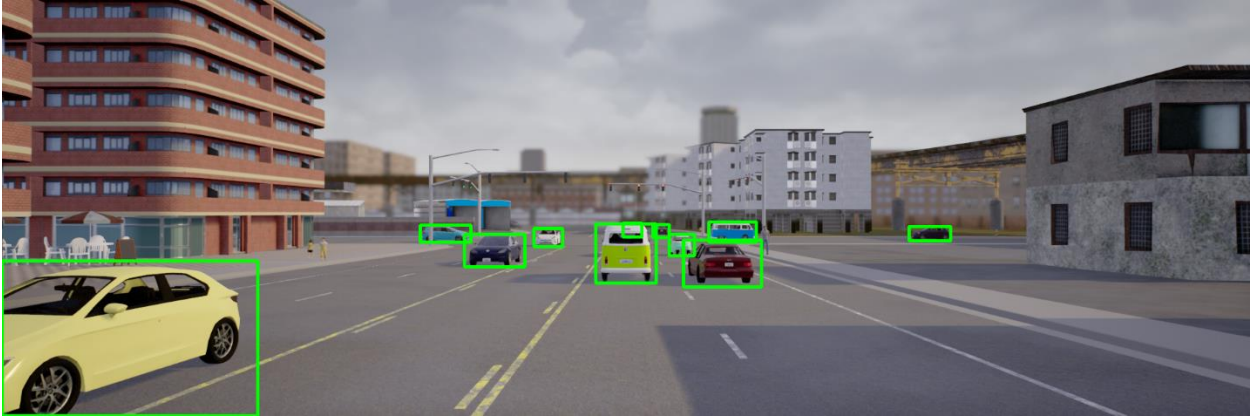


Figure 13: 2D Bounding Boxes

4.2.1.2 Data preprocessing:

Before feeding the ground-truth into the network for training, we have to put each sample of the data in the form of x "input" and y "annotation" of the network.

The input of the network will be simply the RGB image as a matrix of size $1920 \times 640 \times 3$

The output will be a feature map of size $20 \times 60 \times (4 + 2 + 1)$ as described in the next section.

4.2.1.3 Model Architecture:

For this task, we used many algorithms like sensor fusion between camera and LIDAR to benefit from the huge capability of the camera catching the features of object to classify and detect it and second to benefit from LIDAR's great capability of catching the distance to other objects in the scene, YOLO algorithm with just camera sensor as it is the state of art in 2D object detection.

4.2.1.3.1 **First attempts:**

In perception module we needed to fully understand and describe the surrounding environment we decided to use deep learning for the task .

First approach we attempt was to use a model for object detection using RGB image and another one for LIDAR to detect the distance we first looked at mobile net which was a SSD (single shot detector).

It has a lot of advantages like small number of parameters, which would make it fast. We compared it with efficient-det which was a modification on Yolo by increasing depth, width and resolution of the network with certain weights depending on the running machine. This approach is called the late fusion in which we combine the results of the camera and the LIDAR after processing each of them separately. Then in searching for the quest of having better algorithm we found a paper from Uber that suggests a network that makes middle fusion, which is the process of combining the LIDAR features and the RGB features after extracting some higher level feature map from both.

4.2.1.3.1.1 LaserNet [10],[11],[12],[13]:

This network name was Lasernet; laser net suggests working on both the LIDAR image and the RGB image after extracting some features from both of them to deal with a LIDAR data you mainly deal with two kinds of inputs representations.

4.2.1.3.1.2 Input representation:

BEV and RV representation, each one of these two representations has its own advantages and disadvantages, as we will describe.

- **RV**: In the RV, the sensor data is dense, but the perceived size of objects varies with range which would harden the process of training and the detection but the RV preserves occlusion information which is lost when projecting the data into the BEV with sufficient data this will significantly improve the detection and training and reduce the training and processing time as RV representation is often represented in smaller data sizes.
- **BEV**: BEV representation we process the data as if we see it from above. In the BEV, the data is sparse, but the size of objects remains constant regardless of range. This consistency adds a strong prior to the prediction making the problem easier to learn.

4.2.1.3.1.3 Related work for LaserNet:

As in ordinary object detection networks that perform only on the RGB images there are two approaches in detection SSD (single shot detectors) and RGN (**Region**

Proposal Network)

- **SSD**: Single shot based methods produce detections with a single stage
- **RPN**: in RGN method we have two stages: the first stage proposes plausible regions containing objects, and the second stage extracts features from the proposed regions and uses them to produce detections

Many models deal with LIDAR data with different architectures like, VeloFCN, MV3D, VoxelNet, and PointNet. Each of them deals with it in RV like what we will do. But these nets mainly discretize the azimuth angle and the elevation angles and then operate the detections over voxels of the feature map. But in LaserNet we operate

the detection on the row input itself generally other approaches voxelize the feature map then perform the detection on these voxels.

4.2.1.3.1.4 LaserNet architecture:

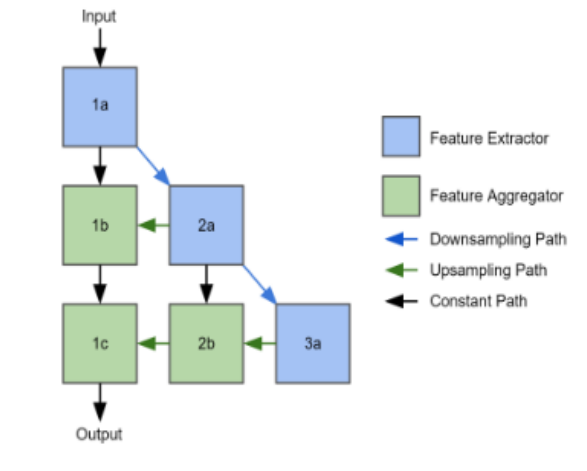


Figure 14: LaserNet Architecture

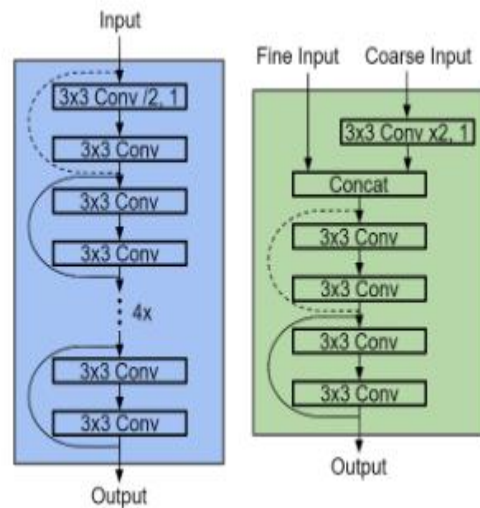


Figure 15: LaserNet block architecture

The input LIDAR image consists of different objects with different sizes the aggregation block is used to combine different features with different feature scales.

The input goes into sequence of aggregation and feature extraction blocks to extract features then combining them then predictions are performed as shown in figure 13

The network was designed by Uber to predict class probability distribution for each point over bounding boxes, but as we were not interested in semantic segmentation and as Uber never provided neither data samples nor output layer example and because laser-net is single shot detector just like YOLO.

We decided to take the network architecture and use the yolo anchors technique in performing detection.

4.2.1.3.1.5 Final layer:

We divide the image into 20×60 cell, which is called a grid. In every cell lies a center for five anchors with different shapes to detect different objects with different shapes, then we take the probability that an object lies in the anchor \square as the probability that there is an object multiplied by the probability of that this object from class \square multiplied by IOU (the percent of an object match the anchor).

4.2.1.3.1.5.1 IOU (intersection over union):

Each one of the five anchors in each cell has a score that defines how much its area intersected with the different anchors so that if more than one object's center in the same cell we could differentiate between them.

4.2.1.3.2 YOLO (You Only Look Once) Training [14],[15]:

YOLO is the state of the art object detection for 2D RGB image.

YOLO system deals with the detection problem as a single regression problem unlike other algorithm which uses complex pipe lined processes to perform detection like RCNN (RPN based method) which first propose potential regions for detection then perform classification then after classification processing is done to eliminate duplicate images and to fine tune the boxes

Or DPM which perform classification at different areas with different scales in an image then find the best IOU, YOLO process the whole problem at once as mentioned as a single regression problem.

In order to compare our work we intended to use YOLO and compare its results with LaserNet.

4.2.1.3.2.1 YOLO Detection:

As mentioned before in previous section, we divide the image into cells equal in area called grids. In each cell lies a center of five anchor boxes. This is to detect more than one object. If its center in the same cell then as single regression problem we get the dimensions of each bounding box and we get the probability that there is an object and class probability and IOU multiplying all those together to perform detection.

4.2.1.3.2.2 Pre-trained YOLO:

First we used a pre-trained YOLO on different datasets from what we have. When we tried to predict the objects to our datasets we found that it predicts almost well. We then used this pre-trained YOLO without any training as our perception model for the project. The figure below shows a sample of a prediction output from this model to our dataset.



Figure 16: sample of prediction output from pre-trained YOLO

4.2.1.3.2.3 Tiny YOLO Training:

To increase the accuracy of our perception model we trained YOLO on our datasets. We used tiny YOLO network on our training step as it has lower parameters compared to the regular YOLO so that it helped us to collect less data [16]. Our training was simple as we annotated the list of data images with labels of cars and pedestrians to predict them. To insure that we reach to the optimal loss if we found that the loss after three consecutive epochs was almost constant we decreased the learning rate by 10 times. Also we used early stopping algorithm to ensure reaching to the best loss. After many training efforts we reached to training loss of 37 and validation loss of 40. The two below images show our last performance with epochs and the final prediction outputs.

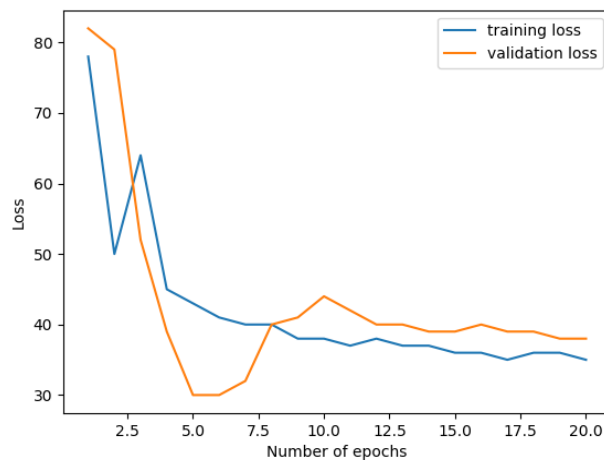


Figure 17: Tiny YOLO loss vs epochs

4.2.1.4 Distance Estimation:

The approach used for objects' distance estimation depends on LiDAR and camera late fusion summarized in the following process flow [17].

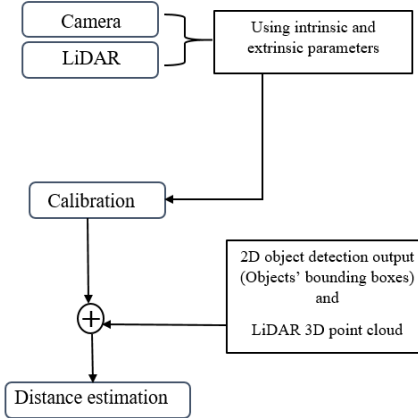


Figure 18: Distance Estimation

Here is the illustration of the each step in the algorithm in details:

For all objects in the camera field of view (Obj_i), get the bounding box four attributes:

X_{min} , X_{max} , Y_{min} , Y_{max} from which the box four corners could be abstracted .

Project the input LiDAR 3D points $F_{lidar,3D}(x,y,z, id)$ to $F_{lidar,2D}(x,y, id)$ using the

calibration matrix $M_{calibration}$ where:

$$M_{calibration} = C_{intrinsic} \times M_{extrinsic}$$

The intrinsic matrix $C_{intrinsic}$ represents the camera intrinsic parameters: FOV and captured image size.

$$C_{intrinsic} = \begin{pmatrix} f_x & 0 & u_x \\ 0 & f_y & u_y \\ 0 & 0 & 1 \end{pmatrix}$$

Where f_x is the camera focal length in the horizontal direction and f_y is the camera focal length in the vertical dimension:

$$f_x = \frac{\text{Image width}}{2 * \tan\left(\frac{\text{Camera FOV} * \pi}{360}\right)}$$

$$f_y = \frac{\text{Image height}}{2 * \tan\left(\frac{\text{Camera FOV} * \pi}{360}\right)}$$

u_x, u_y Is the image center point:

$$u_x = \frac{\text{Image width}}{2}$$

$$u_y = \frac{\text{Image height}}{2}$$

The extrinsic matrix, $M_{\text{extrinsic}}$, represents the camera and LiDAR relative positions in terms of rotation and translation parameters between them.

$$M_{\text{extrinsic}} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Because we have the camera and LiDAR are in the same position on the ego vehicle, so

$M_{\text{extrinsic}} = I_{4 \times 4}$. So, we can consider that the intrinsic matrix only represents the calibration matrix.

$$M_{calibration} = C_{intrinsic} = \begin{pmatrix} f_x & 0 & u_x \\ 0 & f_y & u_y \\ 0 & 0 & 1 \end{pmatrix}$$

$$\& \quad F_{lidar,2D} = M_{calibration} \times F_{lidar,3D}$$

Extract the indices $id_{extracted}$ of the $F_{lidar,2D}$ that lies within each boundary box limits in both horizontal and vertical directions represented in $(X_{min} \ X_{max} \ ,Y_{min} \ Y_{max})$.

Extract the 3D points from $F_{lidar,3D} (x, y, z, id)$ corresponding to the extracted indices $id_{extracted}$.

Finally, compute the distance for each object from the extracted 3D points as:

$$\text{Distance} = \min. \{\text{Vectors (r's) from ego vehicle to each extracted point}\}$$

The following figures represent this algorithm results using YOLO V2 for object classification task.



Figure 19: Distance estimation sample

4.2.1.5 Edge detection:

We used very basic classical technique to determine if the car able to change its lane or not and it consists of two decisions:

4.2.1.5.1 Decide if the road is open or not:

The centre of the frame is used to know if there is a car front of me or not by check if X_{min} of the boundary box is less than the centre or not and if X_{max} of the boundary box is greater than the centre or not.

If $X_{min} \leq \text{centre}$ and $X_{max} \geq \text{centre}$ there is a car front of me as shown in the following figure so the road is not open.

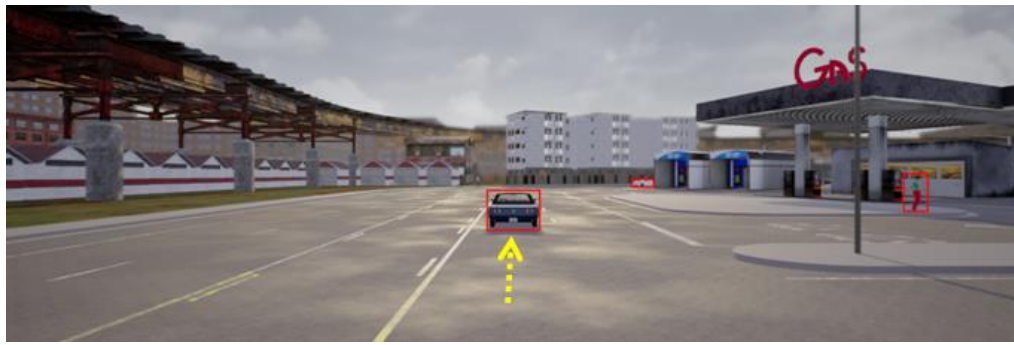


Figure 20: decide if the road isn't open

If $X_{min} \leq \text{centre}$ and $X_{max} \leq \text{centre}$ or if $X_{min} \geq \text{centre}$ and $X_{max} \geq \text{centre}$ the road is open as shown in the following figure.

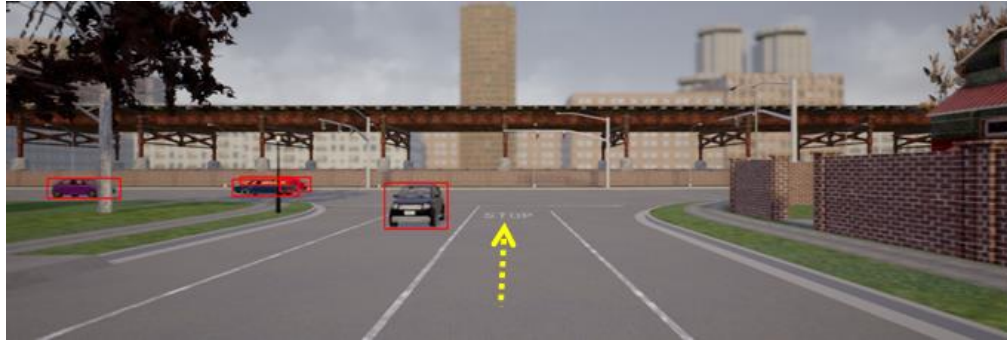


Figure 21: the road is open

4.2.1.5.2 Switching lanes:

When the road is not open we need to change the lane right or left according to the road.

- The road in the left is not empty but the road in the Right is empty:

If $X_{\min} \leq (\text{centre} - \delta)$ and $X_{\max} \geq (\text{centre} + \delta)$ there is a car in the left as shown in the following figure so we cannot change the lane in the left.

But the lane in the right is empty so we can change the lane in the right after check from Carla if the road in the right in the same direction or not.

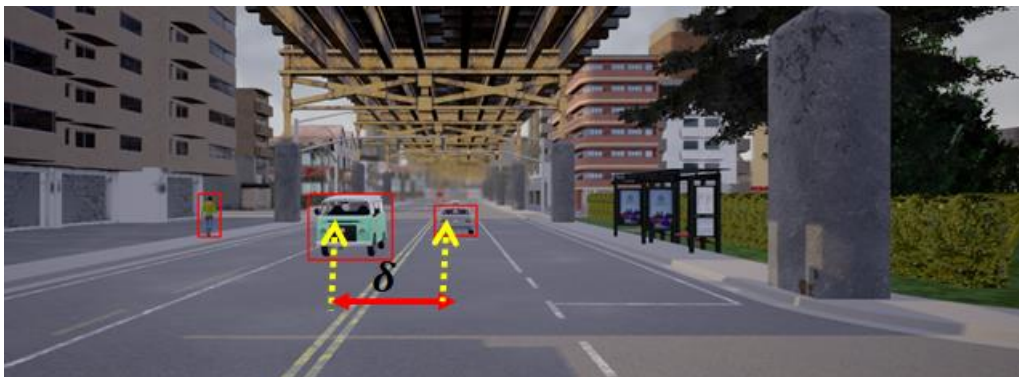


Figure 22: the road in the left isn't open

- The road in the right is not empty but the road in the left is empty:

if $X_{\min} \leq (\text{centre} + \delta)$ and $X_{\max} \geq (\text{centre} + \delta)$ there is a car in the Right as shown in the following figure so we cannot change the lane in this case.

But the lane in the left is empty so we can change the lane to the left after check from Carla if the road in the left in the same direction or not.

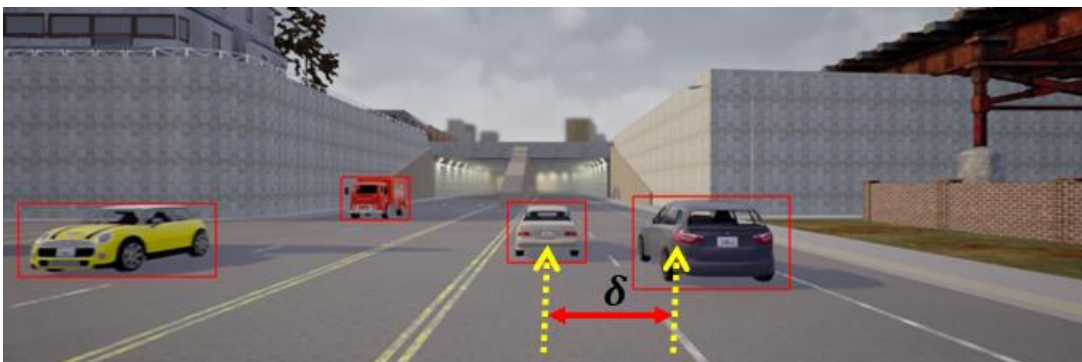


Figure 23: the road in the left is open

4.2.2 Localization:

Generally, Localization is how a car figures out where it is in the world, it is a critical capability for autonomous vehicles, making it possible to pinpoint their location within centimeters inside a map. Typically, car has Global and local position.

- **Global position:** where is the car with respect to the world i.e. which country , city , town it is important as we need to determine global starting point and global ending point
- **Local position:** This is considered the hardest part of localization. Local positioning is determined with a centimeters error factor. It is detecting the exact position of the car in the lane and the exact distance between the car and

the different landmarks. So we can relative to those landmarks determine exactly the car's location, which is so critical as according to localization we may determine to overtake lane or to change speed or to maneuver another vehicle .

In self-driving cars this typically done by combining different sensors, these sensor collect information about every object in the surroundings

By collecting information about the surroundings like the distance between the cars and other cars and between the car and the different landmarks in a road the car can exactly know its location.

We define local coordinate system, which is a series of (X, Y, Z) points in every node in the model to help us keeping track of where is the car's local position where we chose the origin to be any point of the system like start or end point or any arbitrary image.

Local coordinate system information:

- position with respect to the local coordinate system
- velocity vector with respect to the local coordinate system
- acceleration
- orientation ,Roll, Pitch ,Yaw angels with respect to the local coordinate

Landmarks in the road are things like schools mosques churches stop signs different areas where speed limit changes knowing exactly where is the car with respect to all of those things can effectively enhance its ability to take critical decisions.

4.2.2.1 Sensors used for localization:

Sensors used to determine the absolute position of the car relative to the global map (used in determining the global position) like GNSS.

Other sensors used to determine the car's location relative to the object detected in the surroundings (used in determining the local position) like IMU, LiDAR, Ultrasonic, Camera, and Radar.

4.2.2.2 Accuracy in localization:

For the importance of the measurements of the sensors used in localization task it is important to make sure, it is as accurate as possible so sensor calibration must be done.

4.2.2.3 Localization in software:

As it was too hard in CARLA's environment to get map with landmarks on it and because of the absence of some sensor and the lack of time.

We used the definition of localization in real life and implemented it on the simulation with the help of the provided information from CARLA's environment.

CARLA provides a local map that has a known fixed origin, and provides waypoints that connects point to point from one node to another. And it provides the ego vehicle local location with respect to the world fixed point and we consider this also the global location in CARLA's environment. So we used the information collected from CARLA engine like the exact lane and the exact current waypoint to guide the car and doing the mission of localization.

4.2.3 Planner:

In this module we connect between the behavioral planning and localization and control modules to implement the different driving scenarios.

4.2.3.1 Path planning:

Path planning enables transport to find the safest, most convenient, most economically beneficial routes and generate way points from point A to point B. way point defines the position along the reference line, measured in meters from the beginning of the track, calculated in the XY-plane. Global and Local Path Planning are planning methods used in the autonomous vehicles.

The planner inputs are current location from localizer, objects in the surrounding from perception and recommended path from mapping.

Planner takes these inputs and decides which scenario should be performed (Follow Lane, Lane Change or Emergency Stop).

4.2.3.2 Global Planner:

The global planner requires a map of the environment to calculate the best route by compute trajectories from an origin to target waypoints.

The global route planner has received the following updates:

1. Lane change connections for routing with lane changes
2. Basic agents can now perform lane changes in accordance with the planned route

4.2.3.3 Local Planner:

The local planner uses to initialize controller's parameters and computes next waypoints taking into consideration the dynamic obstacles and the vehicle constraints.

Local planner is buffering the waypoints then controller uses it and the target speed for moving car to next point.

4.2.4 Mapping:

Mapping module is responsible for determining the best route to take based on the current town. The input for the mapping module is the town, which is represented as a Carla world, the start point, and the destination point. Mapping module first build a map for the town then choose the best route to take from the start point to the destination point.

4.2.4.1 Building Map:

In Carla 0.9.6, a world is a large map of waypoints that are connected to each other. Each waypoint has a unique ID, and a type based on where is the waypoint located. A waypoint is located in a lane in the map, and its type is the type of the lane where it is located. There are several types of lanes in Carla 0.9.6 like Driving lane, sidewalk, parking lane, and biking. Each waypoint also has a road id and a section id based on the representation of OpenDrive description for roads, lanes, and sections. OpenDrive representation for road layout states that, each road consists of features, lanes, and reference line as described in figure.16.

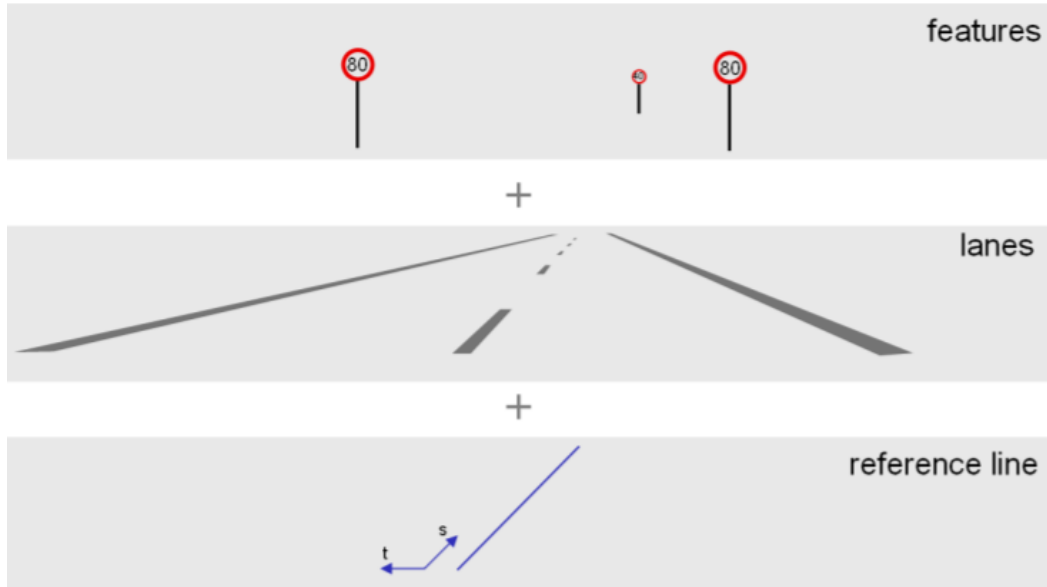


Figure 24: Road layout in OpenDrive

Lanes in the road layout has a unique id based on the position of the lane referenced from the reference line as shown in figure.17.

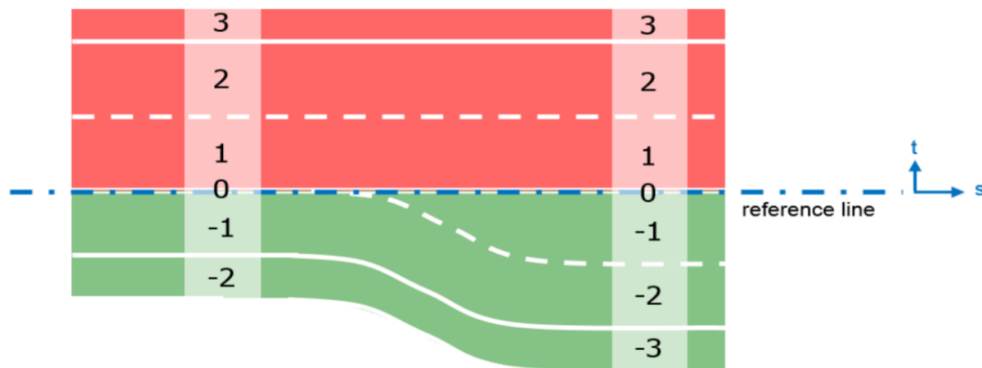


Figure 25: Lane ID in OpenDrive

A lane can be divided into sections based on the type and number of lanes in each side of the road. Figure shows the section division in a road by dividing the two sides of the road for the same sections. Figure shows the section division in the road by dividing each side with independent sections.

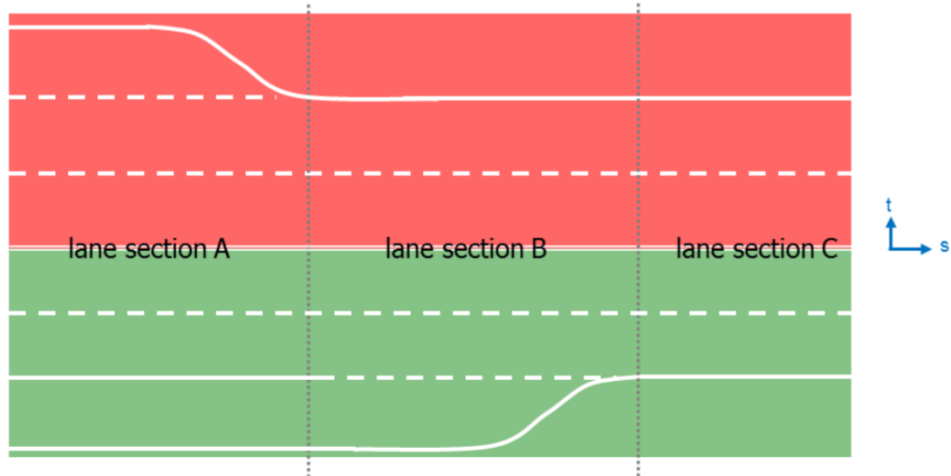


Figure 26: Section ID for same separation in OpenDrive

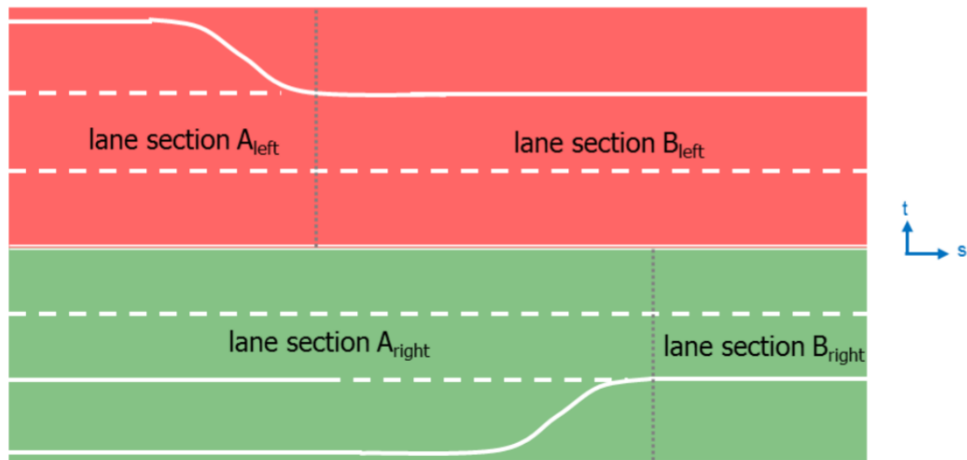


Figure 27: Section ID for different separation in OpenDrive

Mapping module then starts to build a map of connected waypoints. It first takes the start waypoint and obtains its road id and section id. Then from Carla engine, it gets the waypoint in the end of the road in which the start waypoint is. It gets then all the waypoints in between the start waypoint and the waypoint at the end of the road. Then gets the waypoints in the start and end of each road that is connected to the current road. Then get all waypoints in between the start and end waypoints in each road and so on.

After these steps, the mapping module will build a map of connected waypoints for a single lane trip. Then it will get back to the start waypoint and check if a lane change is available at this waypoint by checking if the lanes beside the start lane are drivable and on the same side of the start lane from the reference. If lane change is available, then it will obtain the left or right waypoint, consider it as the start waypoint, and repeat the steps mentioned above.

After these steps, mapping module has built a map of connected waypoints that represents the whole map. The final step in building the map is to put a special flag for the waypoints at the end of the map that has no connected roads and consider as locked roads.

4.2.4.2 Routing:

After building a map of connected waypoints, mapping module will build a tree structure of nodes and edges where the nodes are the waypoints and every two connected waypoints have an edge between them. Building a network tree of nodes and edges ease the process of routing from the start node to the destination node by applying one of the routing algorithms. Then after choosing the sequence of nodes to take from the start node to the destination node, these nodes are converted back to a list of waypoints ordered from the first next waypoint and ended with the destination waypoint.

Routing algorithms are algorithms used to choose the shortest path taken from the start node in the graph, down to the destination node through the best set of nodes and edges. Graphs are classified into weighted graphs and unweighted graphs.

4.2.4.2.1 **Weighted graphs:**

Weighted graphs have cost weights in each link between every two nodes. The total cost of a path is computed by summing the cost of all links that were taken from the start node to the destination node. Consequently, the best path in this type of graphs is the path with the minimum total cost, regardless the number of nodes taken from the first to the last node. There are several algorithms to compute the best path in this type of graphs and we will discuss two of the most commonly used algorithms.

4.2.4.2.1.1 Dijkstra Algorithm:

For a given node, Dijkstra algorithm finds the shortest path to all other nodes in the graph. Therefore, Dijkstra can find the shortest path from the source node to a specific node by stopping the algorithm once the desired path is calculated. The complexity of the Dijkstra algorithm is considered to be $\Theta(|V| + |E| \log|V|)$ where $|V|$ the number of nodes is and $|E|$ is the number of edges. In addition, the performance can be optimized by using the Fibonacci heap min-priority queue to have complexity of $\Theta(|E| + |V| \log|V|)$. However, Dijkstra algorithm can only deal properly with graphs with all positive weighted cost on all edges.

4.2.4.2.1.2 Bellman-Ford Algorithm:

Bellman-ford algorithm is very similar to Dijkstra algorithm as they both works by the relaxation method, by replacing the correct answer of the best path by each iteration. Despite this, Bellman-ford is more complex and slower than Dijkstra as it has a complexity of $\Theta(|E| \cdot |V|)$ in the worst case. However, its advantage that it presents a robust solution for dealing with graphs with negative weights in some of its edges.

4.2.4.2.2 **Unweighted graph:**

Unweighted graphs –also known as Traversal Graphs - do not have cost weights in the links between the nodes. Therefore, the best path in this type of graphs is the path with the minimum number of links taken from the start node until the destination node. The most commonly used algorithm used in this type of graphs is the A-star algorithm.

A-Star Algorithm:

An algorithm that is used for search, traverses through the unweighted and weighted graphs, and find out the best path between any two nodes in the graph. It is used very commonly because of its completeness and optimal efficiency where its complexity is $O(|E|)$.

4.2.5 **Controller:**

After we used perception module to help the car to know its environment, the Localization module to know its position and the Planning module to make decisions and generate trajectories, now the only thing that remains is to move the vehicle. The controller module is responsible of moving the vehicle by generating an angle for the steering wheel and throttle.

There are three approaches for controlling the steering of autonomous vehicles:

- **AI Approach:** The general idea of this approach is to gather training data by driving the car in a simulator, then train the deep neural network with that data, and in the end let the car be driven by the model generated by the deep neural network.

- **Non-AI Approach:** In this approach, we use control theory and math to calculate the steering angle.
- **Combining both approaches:**

We headed into the second approach in our project due to its less complexity, easier to implement and fast. A large number of controllers exist to move a vehicle or a robot. They are more or less complex depending on the problem that we want to solve. The most famous two controllers in non-AI approach are MPC (Model Predictive Control) and PID (Proportional Integral Derivative).

4.2.5.1 Model Predictive Controller "MPC":

The general idea of this controller is to take into account the forces that apply to the vehicle and the characteristics of the vehicle. The controller defines some variables called actuators which are the elements to move the vehicle. A car has three actuators: a steering wheel, an accelerator pedal and a brake pedal. The objective of an MPC is to play on these actuators by varying the angle of the steering wheel, the pressure on the accelerator pedal or on the brake pedal. The controller has two types of implementation models: kinematic model which means that our vehicle realizes the implementation of mathematical formulas to define the movement and trajectory of the vehicle, dynamic model which takes into account the fundamental principle of dynamics and therefore the forces applied to the vehicle. An MPC controller solves an optimization problem. Its purpose is to calculate several pairs (angle, acceleration) and choose the one that causes the lowest error. The MPC controller is very powerful and accurate but very difficult to implement so we headed to use the PID controller.

4.2.5.2 PID controller:

The PID controller is an algorithm that calculates a value like a steering wheel angle or speed from an error calculation. The error is the difference between the trajectory that we must adopt and the one we actually adopt. PID controller is the controller we used in the project as it is the simplest controller and most common in the world. It has the advantage of being implemented quickly and operating in simple situations.

The Control step consists of following the trajectory generated as faithfully as possible. A path is a sequence of waypoints each containing a position (x; y) an angle (yaw) and a speed (v). We divided our controller into two main parts: longitudinal control which takes current speed and target speed to produce throttle as an output and lateral control which takes current location as it's an output from localization module and target way point as an output from planning module and produce steering angle as an output to move the vehicle to the next wanted waypoint.

The PID controller has three main elements in either lateral or longitudinal controls:

P (proportional): This term applies a correction to the steering angle or the throttle proportional to the error. If we are too far from the goal, we turn the wheel in the other direction. The disadvantage of a single P Controller is that it causes a constant oscillation. Depending on the frequency at which the algorithm calculates the error, the oscillation is more or less important. The coefficient K_p indicates the degree of oscillation desired ($a = -K_p e$).

D (Derivative): The purpose of the term D is to suppress this oscillation effect by adding a damping term to the formula. This term is the change of error. The PD

controller understands that the error decreases and slightly reduces the angle it adopts to approach a smooth path.

I (Integral): The last term is used to correct a mechanical error that causes us to turn the wheel more or less strong depending on the vehicle to stay upright. So we add a last term to penalize the sum of cumulative errors.

We therefore have a sum of three components allowing the vehicle to follow a trajectory efficiently in real time. The different K_p , K_d , K_i are coefficients that we must find in order to optimize driving. Figure.20 shows the block diagram of PID controller.

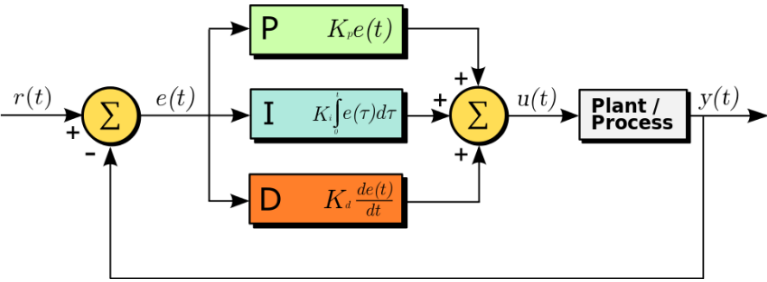


Figure 28: PID Block Diagram

4.2.6 Agent:

Agent module shares the task of system integration with the expert module that we are going to discuss later on. It initializes the system by passing the parameters required for each module: planner, controller, localizer and perception. Agent’s output is the best route to reach the destination by determining at each instant which driving scenario to follow according to the road status: open way, congested way, blocked way while lane change is available lane change or blocked way but lane change is not available.

4.2.6.1 Initialization:

The agent is initialized by the ego vehicle itself, passes the PID parameters to initialize the planner, and initializes the localizer and perception modules.

4.2.6.2 Agent block diagram:

The following block diagram summarizes the agent module task.

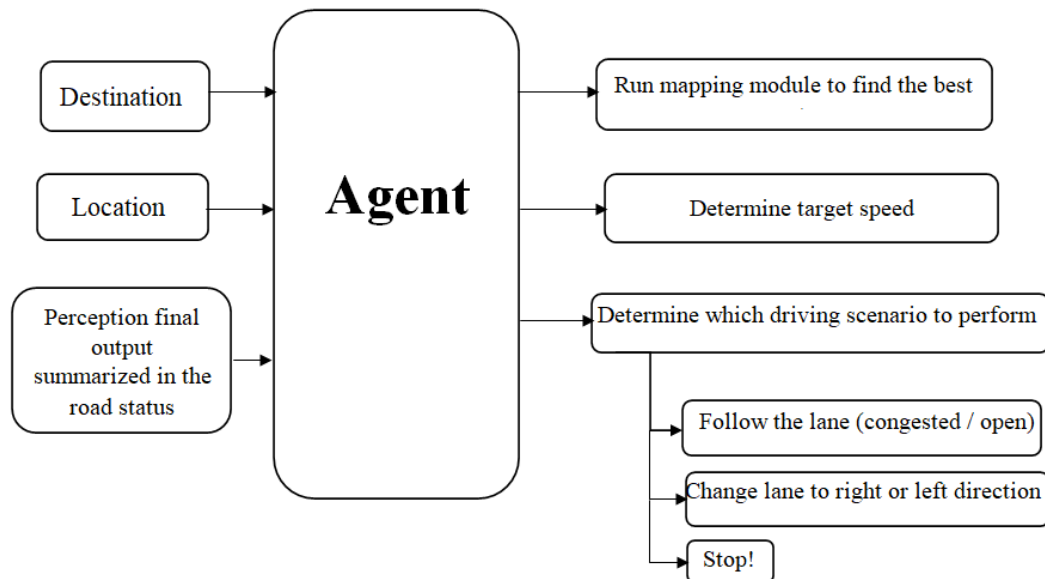


Figure 29: Agent block diagram

The process of finding best route based on a certain routing algorithm is discussed in the mapper, in the following sub sections we are going to discuss how the agent process run to decide the most suitable driving scenario and target speed to follow .

4.2.6.3 Agent process flow:

- First of all, it adjusts the ego vehicle speed to a constant value equals 20 km/h .
- Running perception module , the system has information about each object class in the scene , distance to it in meters and whether it's in same lane with the ego vehicle or not .
- By manipulating this information, the perception module provides the agent with the road status ahead: open way, congested way or blocked way and the availability of lane change to the right or left directions.
- In the following sub section, we are going to discuss each driving scenario.

4.2.6.4 Driving scenarios:

We are going to discuss four driving scenarios: To follow the open way lane, follow the congested lane, change lane to right or left or emergency stop.

4.2.6.5 Scenario one: Follow your open way lane:

If the way ahead the ego vehicle is open which means that distance between ego vehicle and the nearest vehicle is not less five meters, so the ego vehicle has to keep the lane with a constant speed equals 20 km/h with the same PID controller parameters given in the initialization.

4.2.6.6 Scenario two: Follow your lane, but it is congested!

In case that the way ahead is congested by other vehicles, the ego vehicle has to follow its current lane with a speed value less than the default one (20 km/h). This new speed value is determined according to the distance to the nearest object ahead and its speed.

4.2.6.7 Scenario three: Change lane to right or left:

If the current lane is blocked, the ego vehicle has to change lane to the left if available, or to right if available with a speed equals 10 km/h .

4.2.6.8 Scenario four: Emergency stop!

In case of blocked way ahead and lane change is not available to either right direction or left direction, the vehicle has to stop!

5 Results

To evaluate our performance, we used our own metric to measure the total distance travelled in each town, and also measure the total distance without collision for each, then dividing the later by the former, we calculated the accuracy of the whole system as follows :

	Town1	Town2	Town3	Town4	Town5
Total distance	2590 meters	1880 meters	1680 meters	500 meters	1640 meters
Distance without collision	2150 meters	1480 meters	1300 meters	500 meters	1340 meters
Accuracy	83.01%	78.7%	77.3%	100%	81.7%

In addition to these accuracy values , the vechile was still able to complete its trip successfully and reach its destination even if a collision had already occurred. This means that the system's rule based modules can complete their tasks successfully.

Notice that this metric may not be an enough representative to evaluate our performance, in addition we wanted to compare our results with the latest state of the art results available, however we had a problem picking a proper one due to the fact that we found no benchmark available for the modular approach method to compare our results too, also there was no available benchmarks to compare with that worked on carla .9.x versions.

6 Conclusion

We have tested and evaluated a modular approach system using simulated data, on a simulated environment, targeting level 5 autonomy, we managed to successfully build all of the rule based modules and achieved an acceptable accuracy on the deep learning perception module.

We proved that the modular approach is better in comparison with the End-to-End approach in terms of reliability and modularity.

For the future work that can be done for this project, we can retry to colmpete and train the Sensor fusion model, as it can achieve higher levels of accuracy and efficiency, also we can take in consideration traffic lights and traffic signs which will provide even more scenarios to add to the planner module, also we can collect and use more data to train the model better and reach a higher accuracy.

7 References

- [1] Self-Driving Car: Levels, Benefits And Constraints, <https://mobility.here.com/learn/smart-transportation/self-driving-car-levels-benefits-and-constraints?fbclid=IwAR1NwbL0Tp5fBctN56TYIeQG5qb8LtYWWL5i9g-by-QCuoQhOXrbk8YzqSk>
- [2] Self-Driving Cars Explained: How do self-driving cars work—and what do they mean for the future?, <https://www.ucsusa.org/resources/self-driving-cars-101>
- [3] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, Vladlen Koltun, CARLA: An Open Urban Driving Simulator.
- [4] Sensors and Sensor Fusion in Autonomous Vehicles, <https://ieeexplore.ieee.org/document/8612054>
- [5] Andreas Pfeuffer, and Klaus Dietmayer, Optimal Sensor Data Fusion Architecture for Object Detection in Adverse Weather Conditions
- [6] INTRODUCTION TO DATA FUSION , <https://medium.com/haileleol-tibebu/data-fusion-78e68e65b2d1>
- [7] Sensor Fusion of LiDAR and Camera — An Overview, <https://medium.com/@navin.rahim/sensor-fusion-of-lidar-and-camera-an-overview-697eb41223a3#:~:text=Sensor%20Fusion,image%20or%20object%20detection%20applications.>
- [8] Autonomous Vehicle Technology Companies to Watch out For, <https://www.iotforall.com/autonomous-vehicle-technology-companies/>

- [9] Hege Haavaldsen, Max Aasboe, Frank Lindseth, Autonomous Vehicle Control: End-to-end Learning in Simulated Urban Environments
- [10] Gregory P. Meyer, Jake Charland, Darshan Hegde, Ankit Laddha, Carlos Vallespi-Gonzalez, Sensor Fusion for Joint 3D Object Detection and Semantic Segmentation.
- [11] Gregory P. Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, Carl K. Wellington, LaserNet: An Efficient Probabilistic 3D Object Detector for Autonomous Driving
- [12] Fisher Yu, Dequan Wang, Evan Shelhamer, Trevor Darrell, Deep Layer Aggregation
- [13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition
- [14] Joseph Redmon, Ali Farhadi, YOLO9000: Better, Faster, Stronger
- [15] Joseph Redmon, Ali Farhadi, YOLOv3: An Incremental Improvement
- [16] Wangpeng He *, Zhe Huang, Zhifei Wei, Cheng Li and Baolong Guo, TF-YOLO: An Improved Incremental Network for Real-Time Object Detection
- [17] G Ajay Kumar , Jin Hee Lee, Jongrak Hwang, Jaehyeong Park, Sung Hoon Youn and Soon Kwon, LiDAR and Camera Fusion Approach for Object Distance Estimation in Self-Driving Vehicles