

Autopilot for Carla using end-to-end learning for lane centering, global map planning and LIDAR based occupancy grid mapping

Tuan Duc Nguyen

Supervisor: Prof. Dr. Andreas Becker
2nd Supervisor: Sebastian Ehren

A master thesis presented for the degree of
Master of Engineering

Faculty of Information Technology
Fachhochschule Dortmund
Germany
April 2021

Abstract

The aim of this project is to study the different technologies and methodologies applied in autonomous driving, this involves the use of Convolutional Neural Network, Model Predictive Control, LiDAR, Global Navigation Satellite System and Dijkstra's algorithm. Each application is described in each chapter, which includes the theories behind it, the implementation and the final result separately. The project was done using Carla simulator, an open-source platform for developing, testing applications for autonomous driving.

Contents

1	Introduction	7
1.1	Project structure	7
1.2	CARLA	9
2	End-to-end learning	13
2.1	Artificial Intelligent	13
2.2	Machine Learning	13
2.3	Deep Learning	15
2.4	End-to-end learning	26
2.5	End-to-end learning in autonomous driving	27
2.6	Implementation in Carla	38
2.6.1	Environment setup	38
2.6.2	Data collection	38
2.6.3	Model training	40
2.6.4	Training results	42
3	Model predictive control	46
4	GNSS for intersection detection	57
5	LiDAR for objects avoidance	60
5.1	Data retrieval and interpretation	60
5.2	Application	63
6	Global path planning using Dijkstra's algorithm	69
7	Conclusion	78
8	Encountered problems	79
8.1	Data dimension	79
8.2	Versions Inconsistency	79
9	Future improvements	81

List of Figures

1.1	Example of the front camera view for end-to-end learning based algorithm[3]	8
1.2	Autonomous system software architecture	8
1.3	Autonomous system process flow	9
1.4	CARLA view	10
1.5	View of map town02	11
1.6	Sensor examples	11
1.7	Vehicle model examples	12
2.1	Data set structure	14
2.2	Clustering technique	15
2.3	An example of an artificial Neural Networks structure	15
2.4	MNIST data samples	16
2.5	Image in neurons representation	16
2.6	Input layer example	17
2.7	Sigmoid function[8]	18
2.8	Example of cost function	19
2.9	Second example of cost function	19
2.10	Example of cost function in 3D representation	19
2.11	Connection of elements	22
2.12	An example of a network with indices	24
2.13	Sigmoid Function	25
2.14	Tanh Function	25
2.15	Rectified Linear Unit (ReLU) Function	26
2.16	Leaky ReLU Function	26
2.17	Typical speech recognition process	26
2.18	Speech recognition process with end-to-end learning applied	27
2.19	Functional architecture example of autonomous driving system[15]	28
2.20	Training system for self-driving car[3]	29
2.21	CNN architecture for self-driving car[3]	30
2.22	Input	31
2.23	Output of normalization layer	31
2.24	Output of the first convolutional layer	32
2.25	Output of the second convolutional layer	32
2.26	Output of the third convolutional layer	33
2.27	Output of the fourth convolutional layer	33
2.28	Output of the fifth convolutional layer	34
2.29	Output of the dropout layer	34

2.30	Zoomed in output of the dropout layer	35
2.31	Output of the first dense layer	35
2.32	Output of the second dense layer	36
2.33	Output of the third dense layer	36
2.34	Final output	37
2.35	Example of recorded frames	39
2.36	Recorded labels	39
2.37	Steering values distribution	39
2.38	End-to-end learning neural network architecture	40
2.39	The original sample	41
2.40	A flipped sample	41
2.41	Random translated samples	41
2.42	Random shadow samples	42
2.43	Final processed samples for the training	42
2.44	Training result	43
2.45	E2E test visualization	44
2.46	E2E control parameters	45
3.1	MPC controller in autonomous driving system	46
3.2	Simulated path generated in T time steps	47
3.3	Car model uses control options to simulate possible path	47
3.4	Matrices definition	52
3.5	Coordinates of the vehicle on the defined courses	53
3.6	Optimized path at every single time step	54
3.7	Throttle and velocity of the vehicle	55
3.8	Steering value and steering angle of the vehicle	56
4.1	How GNSS module is used for switching driving mode	57
4.2	Control modules transition testing	59
5.1	Point clouds generated by Lidar with different channels	61
5.2	Point clouds generated by Lidar with different amount of points per second	62
5.3	Lidar data in 3D representation	63
5.4	Lidar data in 3D representation after ground removal	64
5.5	Lidar data in 2D representation after ground removal	65
5.6	Stages of forming safety region	66
5.7	Visualization of the dynamic region in its active and inactive state . .	66
5.8	Lidar test scenario	67
5.9	Test visualization	68
6.1	Example graph	69
6.2	Example graph	70
6.3	Example graph	71
6.4	Example graph	72
6.5	Example graph	73
6.6	Example graph	74
6.7	Town02 road segments	74
6.8	Example of direction on a straight segment	75

6.9	Result of first trial of global path finding using Dijkstra's algorithm	76
6.10	Result of second trial of global path finding using Dijkstra's algorithm	77

List of Tables

2.1	System setup	38
2.2	Training parameters	42
3.1	MPC parameters	52
4.1	GNSS parameters	58
4.2	GNSS returned values	58
5.1	Velodyne specifications	60
5.2	Chosen LiDAR specifications	61
6.1	Tracking table	70
6.2	Tracking table updated at node A	71
6.3	Tracking table updated at node D	72
6.4	Tracking table updated at node E	72
6.5	Tracking table updated at node B	73

Chapter 1

Introduction

1.1 Project structure

Autonomous vehicles, self-driving cars and driver-less car are the terms used to imply vehicles driving themselves without human interaction. They are no longer the products of science fiction as it is estimated that more than 30 million self-driving vehicles will be sold each year by 2040 [1]. The ability of self-driving is the result from many different sensors, applications, algorithms and technologies, working together to enable the car to sense its surrounding environment. Accordingly, it is able to visualize the surrounding terrain, classify objects within its field of view, self-navigation while following the transportation rules.

There are many tech giants, who are already involved in the topic that can be named, such as BMW, Audi, Daimler, Waymo, Uber, Udacity, Tesla, etc. BMW and Daimler released their level 4 self-driving car concept and stated that it can handle itself on freeway without human intervention [2]. Uber and Waymo rely on technologies such as LiDAR, camera, ultrasonic sensors and radar while Tesla relies heavily on cameras. Each of them has their own approaches but the vehicles themselves are not yet considered fully autonomous driving (level 5).

The final goal of this master thesis is to develop an autonomous driving system in CARLA Simulator that enables a vehicle to travel from one point to another. In general, the system involves end-to-end learning for self-driving cars method from NVIDIA, an AI-based algorithm; model predictive control, obstacles detection using LiDAR point cloud and Dijkstra's algorithm for path finding.

Each of the modules is responsible for a different task. The end-to-end learning method, also referred to as E2E module, uses a trained model to extract the features from the image frames recorded from a front camera and make predictions of steering values based on those features. The approach had proven to work and NVIDIA claimed that the network model is able to recognise and learn the features that are important for driving, although it had never been told to recognise those features specifically. Details about the method will be discussed in chapter 2.

Chapter 3 explains model predictive control algorithm module (MPC module), which is used to maneuver the vehicle at the intersections so that it stays close to the path defined by the Dijkstra's algorithm. Chapter 4 demonstrates how the LiDAR

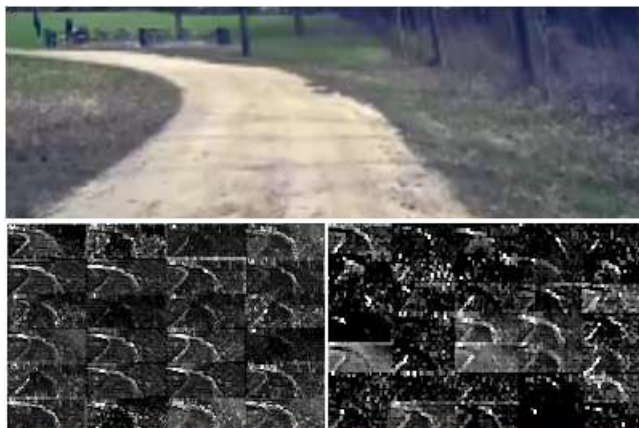


Figure 1.1: Example of the front camera view for end-to-end learning based algorithm[3]

module was used to detect obstacles in the predicted path at a certain range to avoid collision. Finally, chapter 5 explains how the Dijkstra's algorithm works to define the global path. The general structure of the autonomous driving system is described in figure 1.2 and the main process flow between the modules is described in figure 1.3.

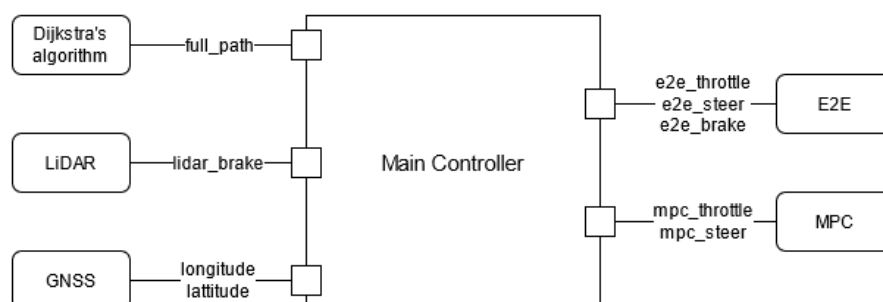


Figure 1.2: Autonomous system software architecture

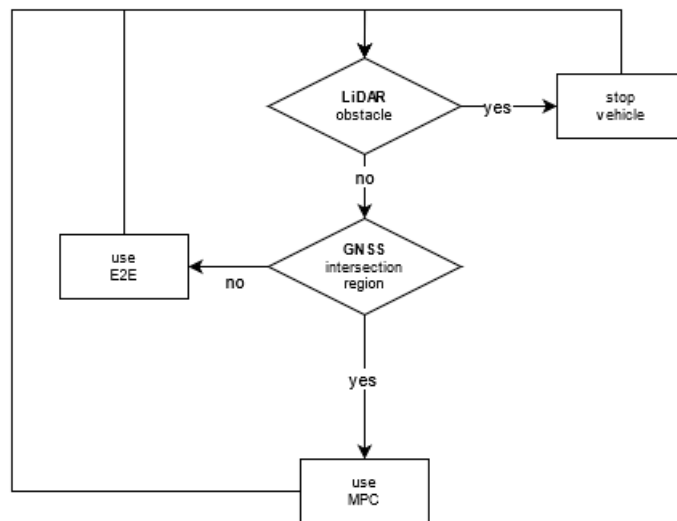


Figure 1.3: Autonomous system process flow

1.2 CARLA

As introduced on their home page, CARLA is a simulator created to support the development, training and validation of autonomous driving systems. It provides open digital assets such as urban layouts, buildings, vehicles for creating scenarios and provides a flexible specification of sensor suites, environmental conditions, the users also have full control of all static and dynamic actors, maps generation, etc.

CARLA features many maps with different scenarios, a few of those are displayed in figure 1.4 below.

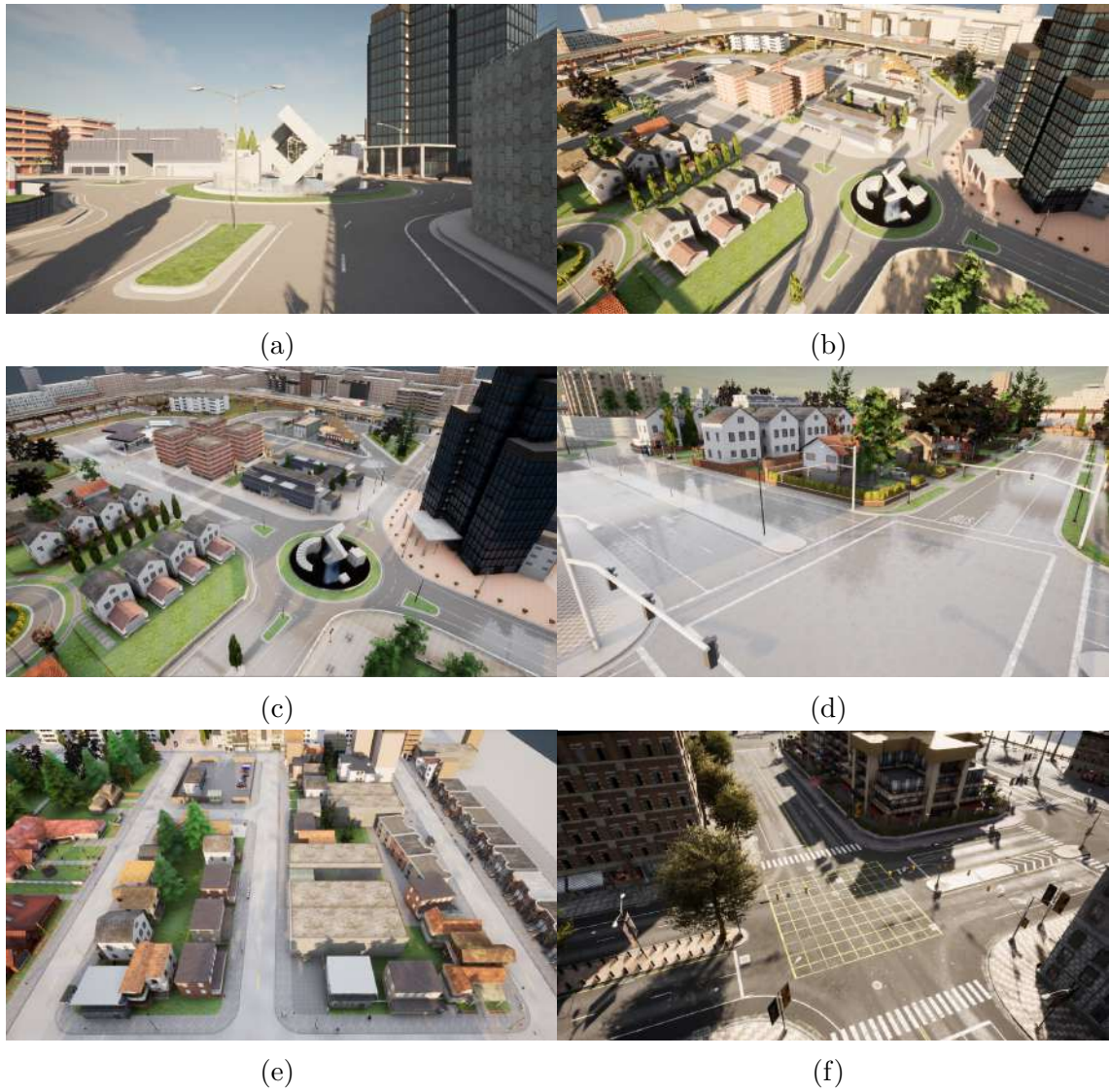


Figure 1.4: CARLA view

Map town02 is used as the main map for development and testing for this project. It features almost every aspect of the basic urban traffic such as two-lane road with broken lane, intersections with traffic lights, etc. The bird-eye view of the map is shown in figure 1.5.



Figure 1.5: View of map town02

The user can choose to spawn any sensor supported by CARLA library with custom specifications, figure 1.6 shows some examples of camera, depth camera, semantic segmentation camera.



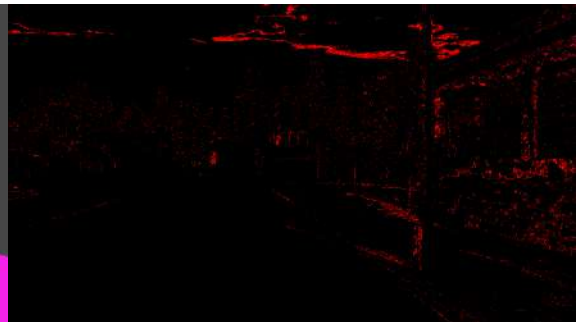
(a) camera



(b) depth camera



(c) semantic segmentation camera



(d) dynamic vision sensor

Figure 1.6: Sensor examples

Moreover, the user also has the freedom to choose the wanted model for the vehicle

such as Mercedes, BMW, Tesla, etc., as in figure 1.7.



(a) Vehicle model 1

(b) Vehicle model 2

Figure 1.7: Vehicle model examples

Chapter 2

End-to-end learning

2.1 Artificial Intelligent

Artificial Intelligent (AI) implies the intelligence of machines that mimics human learning processes. These processes include information gathering and processing, reasoning, and self-improving. In particular, it allows the machine to learn from experiences and can be trained to complete specific tasks just like humans by learning from a large amount of data provided.

AI is being used widely in almost every profession and industry, from medical, education, military to stock exchange, money transaction, security, etc. It only just became well-known and received huge attention thanks to Big Data. Big Data simply provides data and AI can be seen as a tool to analyze and extract information from such data and in the end, giving more data. Without each other, they are not pretty useful. Humans apparently can analyze data but when speaking of efficiency, it cannot be compared with the machine, especially with a large amount of data that may take decades to gather. One great thing about AI is that it is also able to recognize patterns that are undetected under human supervision. This makes AI become the right tool for data analysis.

2.2 Machine Learning

Machine Learning is a division of AI where the machine is not hard-coded but improves itself by repeating the same task a large number of times. It will tweak the process each time until it meets the desired output, which is quite similar to the way human learns. There are various algorithms of machine learning which require different types of input and produce different types of output as well. The choice of the appropriate algorithm depends on the application or the task that it is used for.

The first type is Supervised Learning. With input (x), output (y), a function (f) is used for mapping input to output and an algorithm is used to learn this function. In specific, a data set provides examples of situations and is already labeled with the possible outcomes. The machine will then analyze the training data to produce a model that can provide a prediction for the new input based on what it learnt

from the training data. Some use cases of supervised learning are fraud detection based on analyzed bank data of transactions or new songs recommendation based on user's music choices and song features. Netflix stated on their website that they have applied supervised learning to give out movie recommendations for users based on their preferences and moreover, to optimize the production of Netflix Original Series by learning the characteristics that make a movie successful[4]. Supervised learning can be categorized into two smaller algorithms which are "Classification" and "Regression". Classification is related to categorized output, such as positive or negative, profitable or valueless, or for object classification. Regression is more about real value, such as height, weight, amount of money or age, etc. A data set can be visualized in figure 2.1 as follows.

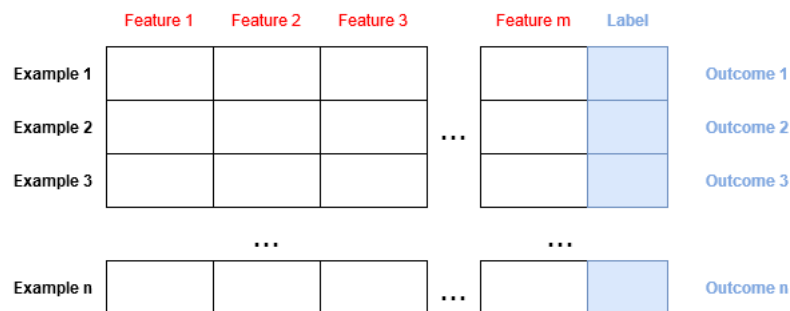


Figure 2.1: Data set structure

One simple example of supervised learning is fruit classification. The features being used here are the shape and color of the fruit. For instance, if a fruit has a curvy shape and is yellow, there is a high chance that it is a banana. If the fruit has a round shape, it can either be an apple or an orange; based on the color, it is an apple if it is red and it is an orange if it has the color orange. The more features given, the more accurate the prediction will be.

The second type is known as Unsupervised learning. Unlike Supervised Learning, Unsupervised Learning uses data set without output, meaning the examples provided have no corresponding outcomes. In particular, the training data used in this case are not labeled. What the machine does, in this case, is grouping data that have similarities without prior training, from there it will recognise the pattern of data groups and will be able to model the distribution of data. Unsupervised learning can be categorized into the following techniques:

- **Clustering:** a technique that discovers similarities between data points and finds a structure, a pattern in such data so that they can be classified into specific groups, as demonstrated in figure 2.2. A simple example of clustering is classifying customers into separate groups based on their similarities in shopping behavior. There is a variety of clustering techniques that can be named, such as K-means, hierarchical clustering, k-nearest neighbor, etc.
- **Association:** the technique find interesting relationships, dependencies in large databases. These discoveries provide valuable information that is often used to improve decision making, to create plan. Some use-cases of association rules are market-basket data analysis, cross-marketing, data preprocessing, etc. [5].

One crucial thing about features is weighing them, which means evaluate their ef-

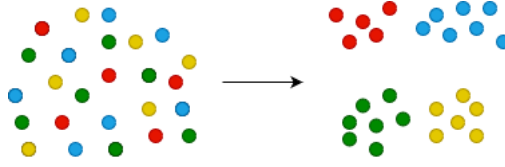


Figure 2.2: Clustering technique

fects. Considering a movie that can be judged based on numerous criteria such as character, plot, visuals, structure, etc. For a good movie to be made, one should not focus on any, separately. A movie with fancy visual effects does not make it a great one without a good plot or having a nice plot but a confused structure does not make it better anyway. Nevertheless, these criteria have different levels of influence based on the movie and one way to measure their importance is by doing weighing. The general value of a film can be estimated using the formula 2.1 where w represents the assigned weight and x_n indicates the given score.

$$value = w_{character}x_1 + w_{plot}x_2 + w_{visuals}x_3 + w_{structure}x_4 \quad (2.1)$$

Mathematically speaking, weight is the Gradient in linear algebra and is also known as the steepness of the linear function. The higher the Gradient is at a point, the steeper the line is at that point. Assuming $w_{content} = 4$ and $w_{cover} = 2$, their influences will now be different despite being given the same score.

2.3 Deep Learning

Deep Learning is a subset of Machine Learning which is inspired by the human brain structures. It mimics how the human brain processes and classifies information by filtering inputs through layers and layers to learn to predict the outcome[6]. These layers structure are referred to as artificial neural networks, a network that contains multiple layers and each layer itself contains a large number of neuron nodes. Each neuron in one layer is connected to other neurons in the next layer, which creates a structure that is similar to what is shown in figure 2.3.

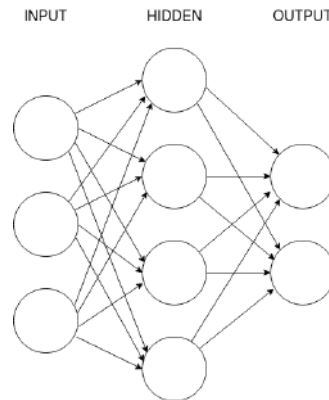


Figure 2.3: An example of an artificial Neural Networks structure

Many layers of nonlinear processing units are used to form a hierarchy of multi-level abstraction. Each level is the output of the previous layer and will be used as the

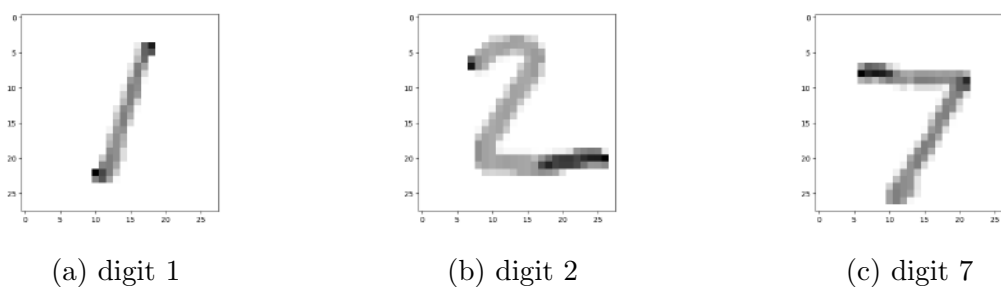


Figure 2.4: MNIST data samples

input for the next layer. The iterations continue until the data is transformed into a more abstract representation, which means it has reached an acceptable level of accuracy. The process where the data passing through numerous layers explains the inspiration of the word "deep" in deep learning.

One famous example that can be used to demonstrate clearly how neural network works is Handwritten Digit Recognition. The database used in this example is MNIST, short for Modified National Institute of Standards and Technology, a massive database of handwritten digits. It features 2 sets of images, a training set with 60000 images, and a test set with 10000 images with the shape of 28x28 pixels. A few samples are shown in figure 2.4.

Each pixel is considered as a neuron node and each carries a color value between 0 and 255. In grayscale, the value will be between 0.00 and 1.00, which is the value each neuron will hold, also known as "Activation". The higher the value, the brighter the neuron is, and the lower the value, the darker it will be. An example of the digit 2 is shown in figure 2.5.

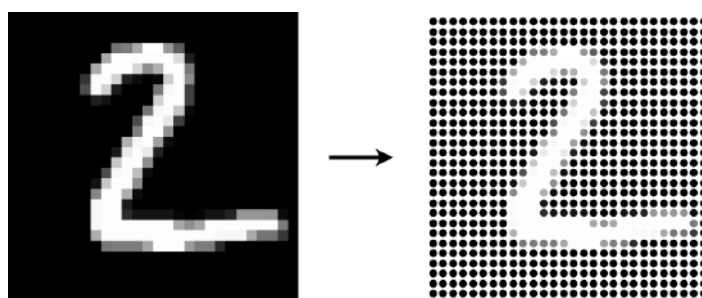


Figure 2.5: Image in neurons representation

At first, the neural network starts with the input layer with the number of neurons corresponding to 28 times 28 pixels, which is 784 neurons. For a simpler demonstration, an image with the size of 3x3 pixels will be flattened into a layer with the shape of 9x1 neurons, as shown in figure 2.6.

These 784 neurons form the first layer (input layer) of the network. As for the last layer (output layer), there will be 10 neurons corresponding to 10 digits ranging from 0 to 9, each neuron will also hold a value between 0.00 and 1.00. When feeding in an image of a digit, the neuron in the last layer with the highest value indicates the number that the network thinks the input shares the resemblance. What happened here is that when an image is fed in the network, the brightness value of all 784

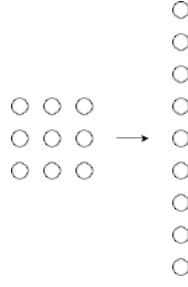


Figure 2.6: Input layer example

neurons in the first layer will create a pattern for the next layer and the iteration continues until the last layer. The output of a layer is the input for the next layer, as mentioned above. The layers in the middle are referred to as hidden layers and how the pattern is created will be explained in detail below.

As introduced at the beginning of this chapter, each neuron is linked with all other neurons in the layer after it. This multi-layer concept is sometimes referred to as *multilayer perceptrons* or *MLPs*[7]. Each link and each connection, instead of a feature, are weighted and this is similar to what is explained with the formula (2.1) of value estimation above. The Activation of an individual neuron in the next layer is the weighted sum of all the connection it has with the previous layer, which can be expressed as follows:

$$value = a_1w_1 + a_2w_2 + a_3w_3 + \dots + a_nw_n \quad (2.2)$$

With a_n is the activation of each pixel and w_n is the weight of each connection accordingly. The weight can be negative or positive depending on the importance of the pixel. This helps define a certain, small region of interest as the brightness of the group pixels in that region is positive. The edge of the area could also be detected by adding negative weights to the pixels surrounding it.

Computing the weighted sum of a node also means computing the probability of a feature. When using the formula (2.2) above, the result may vary over a large range. Probability values exist only between 0 and 1, therefore the result must be squashed using the sigmoid function. As described by deepai.org, a sigmoid function, also known as the squashing function, is an activation function. It squeezes the output so that the outcome lies within the range from 0 to 1 as desired. The sigmoid function is expressed as follows:

$$\sigma(x) = \frac{1}{1 + e^{-z}} \quad (2.3)$$

Additionally, the activation function is a way that allows linear functions, such as function (2.2), to be transformed into nonlinear functions and its purpose is to decide whether a neuron should fire (or activate) or not. There are many different types of activation functions, such as Sigmoid function, Tanh function, ReLU, leaky ReLU, etc. Each of those has a different output range with its own advantages and disadvantages. This will be discussed deeply later in this chapter and for now, the Sigmoid function is kept to explain the basic concept of the neural network.

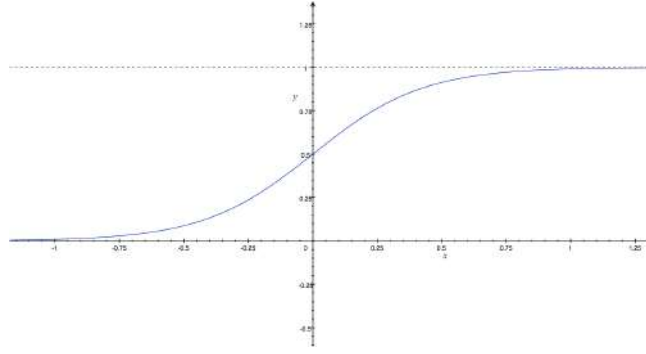


Figure 2.7: Sigmoid function[8]

The expected activation level of each pixel is, however, different. At a certain value, some nodes will light up while the others will not. For this to be done, a value called bias is added to the weighted sum before it is squished. From what is discussed above, the formula (2.2) can now be rewritten as:

$$n_{activation} = \sigma(a_1w_1 + a_2w_2 + a_3w_3 + \dots + a_nw_n + b) \quad (2.4)$$

Once again, the formula above is just for computing the activation of one neuron in the next layer. As described previously, each neuron in the current layer have connections to every neuron in the layer before it, this means for each neuron, there will be different sets of weights and biases. What the neural network will do is to find the right weights and bias so that it will behave as expected. This process is referred to as learning in deep learning. From the example of 3blue1brown[9], assuming there are 2 middle layers, each has 16 nodes and this creates approximately 13000 weights and bias to be tweaked in this neural network, each change will make it behave differently.

Initially, the weights and biases are generated randomly and this eventually makes the output of the last layer become inaccurate. At this point, a cost function C is used to measure the accuracy of the output. What is calculated is the sum of the square of the differences between the current value of each node and its expected result. Apparently, the sum is small when the result is near as expected and vice versa. The average cost of all training data is then computed and it represents the network performance at that moment, the lower the average cost, the better the performance and in order to achieve the desired output, the weights and biases need to be adjusted to the appropriate value.

Mathematically speaking, the cost function is just a normal function with the goal is to find the minimum value. Imagine a simple cost function $C(\omega)$, as shown in figure 2.8, which depends on only one variable ω . At first, the weight is initiated randomly, therefore the cost might be one of the 4 dots on the sides. Assume that the current cost is the blue dot and the desired cost is the red dot with the slope at that point is 0. The slope indicates how weight should be adjusted. More specifically, the weight value should be shifted to the left if the slope is positive and to the right, if the slope is negative. Assuming the current point is the blue dot and its negative slope implies the weight value should increase.

Nevertheless, the graph of the cost function could be something shown in figure 2.9, a function with multiple local minimums. This suggests that there are multiple

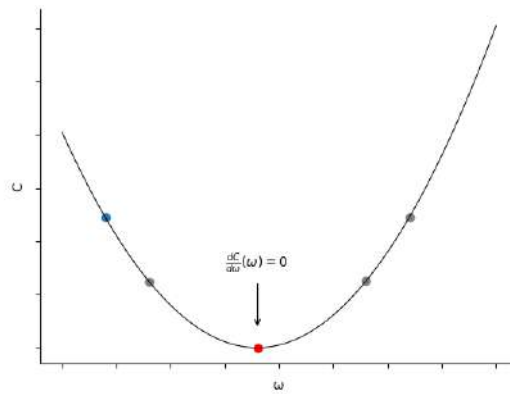


Figure 2.8: Example of cost function

possible values for the cost function and this depends on the random starting point that is initiated. Assuming that a local minimum has been chosen, there is no guarantee that it is the smallest cost that is achievable.

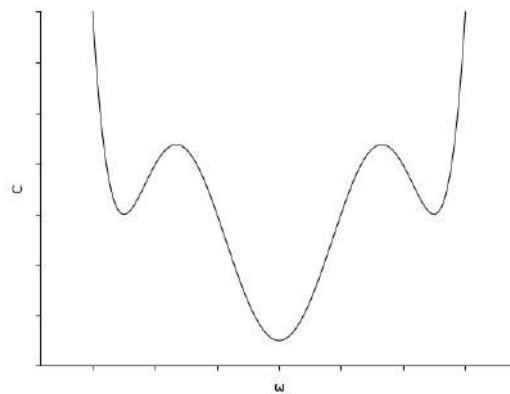


Figure 2.9: Second example of cost function

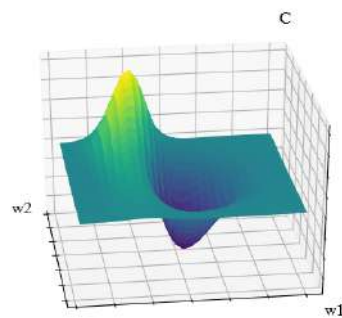


Figure 2.10: Example of cost function in 3D representation

In 3D representation, the cost function may look like what is presented in figure 2.10. In this case, the two axes x and y represent two corresponding variables while z axis represents the cost C . To minimize the cost, a technique called Gradient Descent is used.

As explained in [7], suppose the cost function C_v has multiple variables with $v = v_1, v_2, \dots, v_n$. For now, having only v_1 and v_2 is simpler for 3D demonstration. By using derivatives, the change of C can be approximated as follows:

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \quad (2.5)$$

Δv_1 and Δv_2 represent the amounts of changes in the x and y direction accordingly. Since the goal is to minimize C , it is obvious that ΔC should be negative and this depends on the value of Δv_1 and Δv_2 . From the equation, there are two elements that can be defined. The first one is the vector of changes in v , denoted as Δv :

$$\Delta v \equiv (\Delta v_1, \Delta v_2)^T \quad (2.6)$$

The second element is gradient C , which is the vector of partial derivatives, denoted as ∇C :

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T \quad (2.7)$$

By what are defined above, equation (2.5) can now be rewritten as equation (2.8). This shows that ∇C relates changes in v to changes in C [7] and this make it possible to adjust Δv as desired to make ΔC become negative.

$$\Delta C \approx \nabla C \cdot \Delta v \quad (2.8)$$

Suppose Δv is chosen as:

$$\Delta v \approx -\eta \nabla C \quad (2.9)$$

With η is a small and positive number, known as the learning rate. At this point, equation (2.8) can be rewritten as follows:

$$\Delta C \approx -\eta \|\nabla C\|^2 \quad (2.10)$$

Since $\|\nabla C\|^2$ and η are both positive, the statement $\Delta C < 0$ is asserted. This implies that as long as the expression (2.9) is hold true, C will always have a negative change, which can be understood that it will always decrease. Each time Δv is computed, the cost is updated and this process repeats until the desired minimum cost is achieved. The new value of v' is updated as:

$$v \rightarrow v' = v - \eta \nabla C \quad (2.11)$$

One notice is that the value of η is flexible, this suggests that its value should be chosen properly in order for the gradient descent algorithm to work efficiently.

Returning to weights and and biases, the following expressions happens when the interpretation (2.11) is applied:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \quad (2.12)$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \quad (2.13)$$

The cost function of the one training cycle is defined as:

$$C(w, b) = \frac{1}{2n} \sum_x \|a - y(x)\|^2 \quad (2.14)$$

Where a are the network predictions and $y(x)$ is the desired value. This can also be expressed as in (2.15), where $C_x = \frac{\|a - y(x)\|^2}{2}$ is the cost of one training sample. Similarly, gradient ∇C , expressed in (2.16), is also computed by taking the average of all the gradient ∇C_x , with each computed from an individual training sample.

$$C = \frac{1}{n} \sum_x C_x \quad (2.15)$$

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x \quad (2.16)$$

However, this might be time-consuming in case the training data is enormous. One solution to this problem *stochastic gradient descent*, what it does is selecting an amount of m samples randomly and these amounts are known as *mini-batch*. Denote ∇C_{Xj} as the cost for each training sample in the current batch and how it relates to gradient ∇C is expressed in (2.17). What is expected here is that the average value of ∇C_{Xj} of a mini-batch is just about the average value of ∇C_x of the entire data set.

$$\frac{\sum_{j=1}^m \nabla C_{Xj}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C \quad (2.17)$$

In terms of weights and biases in the neural network system, equation (2.12) and (2.13) can be presented as:

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{Xj}}{\partial w_k} \quad (2.18)$$

$$b_l \rightarrow b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{Xj}}{\partial b_l} \quad (2.19)$$

The mini-batches are chosen randomly until the training data is exhausted, which means a training epoch has been completed and a new epoch will begin.

What to be discussed next is the algorithm used to compute the gradient of the cost function. Such an algorithm is known as *back-propagation* and it illustrates how the cost is influenced by changing the weights and biases.

First of all, assuming a context where there are only two neurons. The activation of the first and second neuron are defined as a^{L-1} and a^L respectively. The superscript (L) , $(L-1)$ are not understood as exponents, they indicate the layer of the respective neurons with L is the total amount of layers in the network. The cost of a single training example is similar to expression (2.14) which is:

$$C_0 = \frac{1}{2}(a^{(L)} - y)^2 \quad (2.20)$$

The activation of the neuron in layer L is computed as:

$$a^{(L)} = \sigma(z^{(L)}) \quad (2.21)$$

With $z^{(L)}$ is denoted as the weighted sum:

$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)} \quad (2.22)$$

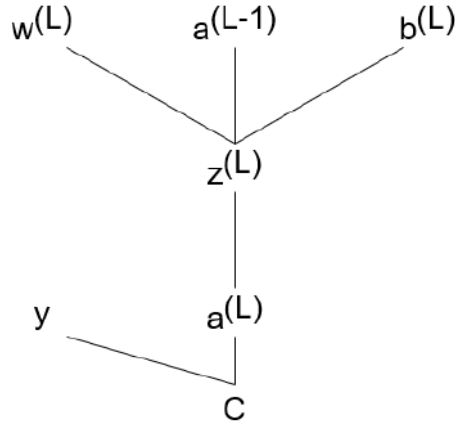


Figure 2.11: Connection of elements

The three expressions above can be visualized as in figure 2.11. These connections are the base to evaluate the sensitivity of the cost function when the three elements, weight, bias and activation are adjusted. Under mathematical expression, this means finding the derivative of C with respect to $w^{(L)}$, $\frac{\partial C_0}{\partial w^{(L)}}$. Based on how the weight is linked with the weighted sum $z^{(L)}$, the activation $a^{(L)}$ and the cost C_0 , the derivative can be expressed as:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} \quad (2.23)$$

Based on the expression (2.20), (2.21) and (2.22) above, those derivatives can be computed individually as follows:

$$\frac{\partial C_0}{\partial a^{(L)}} = a^{(L)} - y \quad (2.24)$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)}) \quad (2.25)$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)} \quad (2.26)$$

Assume the derivative of the Sigmoid function is signified as $\sigma'(z^{(L)})$. The gradient component in expression (2.23) can now be rewritten as:

$$\frac{\partial C_0}{\partial w^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) (a^{(L)} - y) \quad (2.27)$$

This is just of course one gradient component of a training example. The general gradient will be the average result of all training examples:

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_k^{n-1} \frac{\partial C_k}{\partial w^{(L)}} \quad (2.28)$$

Similarly, the sensitivity of the cost function to the bias is as follows:

$$\frac{\partial C_0}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} \quad (2.29)$$

As the derivative of the weighted sum with respect to the bias, $\frac{\partial z^{(L)}}{\partial b^{(L)}}$, is equal to 1, the expression (2.29) will now be:

$$\frac{\partial C_0}{\partial b^{(L)}} = 1 \sigma'(z^{(L)}) (a^{(L)} - y) \quad (2.30)$$

Finally, the sensitivity of the cost based on the last element, the activation of the last layer is defined as:

$$\frac{\partial C_0}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} \quad (2.31)$$

The derivative of the weighted sum with respect to the activation in the last layer is $\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = w^{(L)}$. The above expression is rewritten as:

$$\frac{\partial C_0}{\partial a^{(L-1)}} = w^{(L)} \sigma'(z^{(L)}) (a^{(L)} - y) \quad (2.32)$$

By repeating these computations backward all the way through the network, the sensitivity of the cost function with respect to every weight and bias can be kept track of. This is the main idea behind the term *back-propagation*.

As assumed at the beginning, these gradient components are only between two neurons. In a complex network (figure 2.12), these components can be expressed in the matrix form. The components are provided with subscripts to indicate the connections between which neurons to which as this forms matrices with those indices. For the activation component, the subscript marks the order in the current layer. As for the weight, there are two separate subscripts j and k which implies the neuron

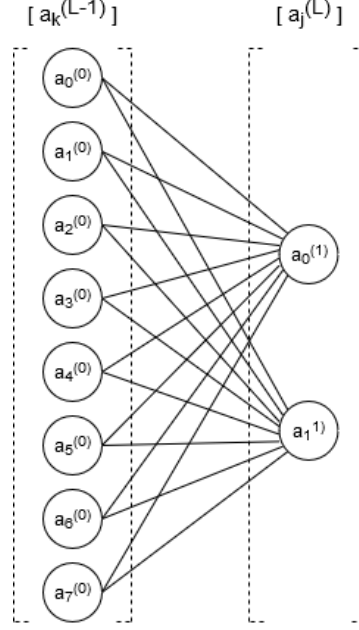


Figure 2.12: An example of a network with indices

that it is linked with and the one it is connected from accordingly. It is written as $w_{jk}^{(L)}$.

The only difference when the above derivatives are applied is the derivative with respect to the activation of the last layer. According to figure 2.12, one neuron in layer 1 now influences 2 neurons in layer 2. The expression (2.31) changes simply by taking the sum over all neurons as:

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}} \quad (2.33)$$

At this point, the algorithm for the network to learn to adjust by itself to reduce the cost function to a minimum value has been explained.

Besides, as mentioned above, there are many different activation functions that can be used for a neural network. Each of those has certain impacts on computational complexity and the convergence of models[10]. Due to the output range between 0 and 1, the Sigmoid function (figure 2.13) is especially used for models that predict the probability[11]. Nevertheless, this function is only useful in the mid-region where a small change of x causes a large change in y . As explained above, the neural network learns based on the derivative of the function and when the gradient is close to 0, the network learns slowly because the weights are only updated with a small increment. This problem is known as the vanishing gradient.[10].

Similar to Sigmoid function, the Tanh function (figure 2.14) has the output range constrained between -1 and 1. This helps it converge faster than the Sigmoid function but still suffering from the vanishing gradient problem[10].

Rectified Linear Unit function (figure 2.15) is different from Sigmoid function and Tanh function, the output of the ReLU function is equal to its input in the range

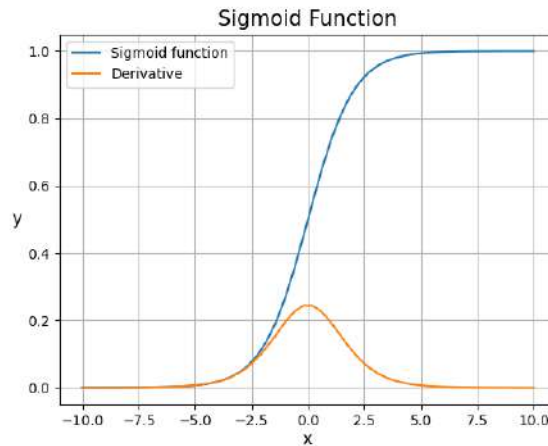


Figure 2.13: Sigmoid Function

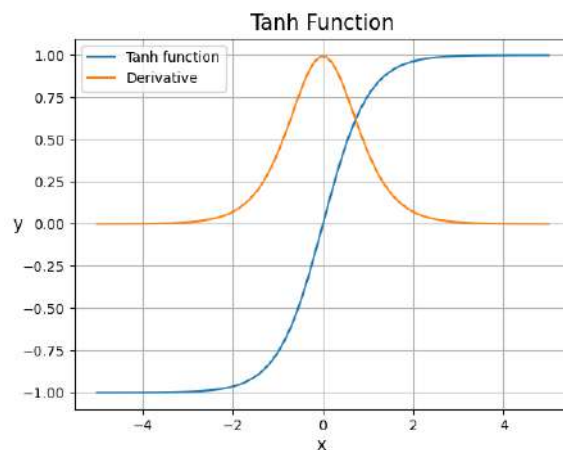


Figure 2.14: Tanh Function

that is greater than 0 and is equal to 0 in the remained region. Nonetheless, with the derivative being 1 when $x > 0$ and being 0 when $x \leq 0$, it creates another problem where the weights are never updated during back-propagation, causing the network unable to learn for negative input values. This is known as dying ReLU[10].

Dying ReLU can be solved by returning a very small value in the region where $x \leq 0$, such function is known as Leaky ReLU, an improved version of ReLU function.

There are many other activation functions and they are chosen depending on their characteristics and the general purpose. However, with being used widely nowadays, ReLU is the one considered as a good general activation function[12].

The next section introduces the end-to-end learning approach, a small branch of deep learning, and presents the network structure that is used for the self-driving vehicle.

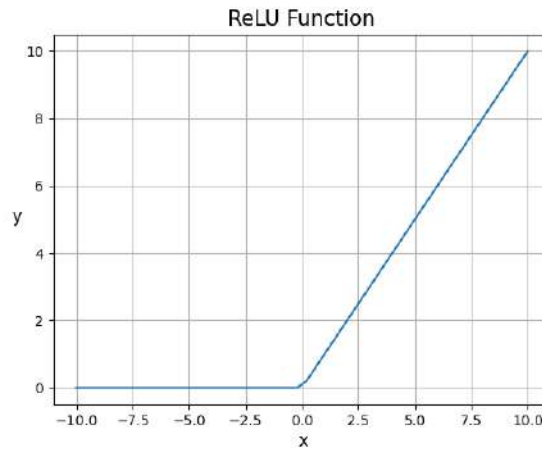


Figure 2.15: Rectified Linear Unit (ReLU) Function

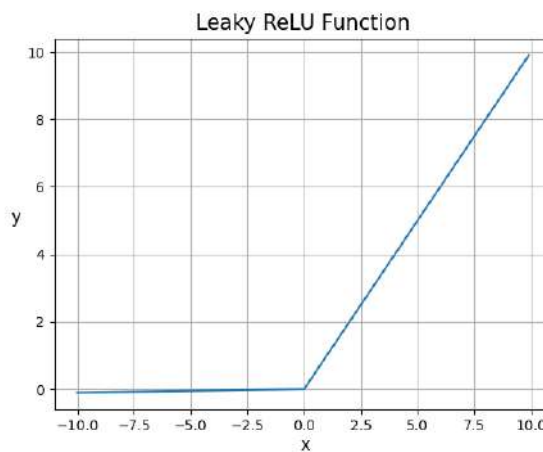


Figure 2.16: Leaky ReLU Function

2.4 End-to-end learning

There are simple problems that the output predictions can be generated directly from the input, but there are also numerous problems that require a complex system of multiple processing stages in the middle before returning the predictions. For example, the well-known speech recognition where the input, an audio clip, is mapped to a transcript, which is the final output, and of course the audio must go through some middle processing stages before returning the output. A typical speech recognition approach includes the following stages: feature extraction, phonemes detection, word composition, and finally, transcript formation. The optimization for these modules above must be done separately under certain criteria. The pipeline is visualized as follows:

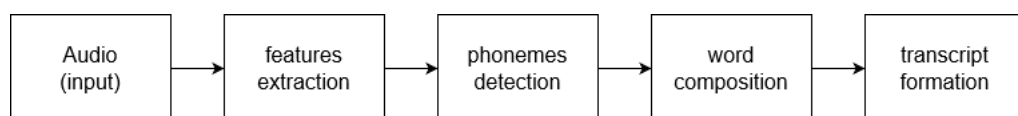


Figure 2.17: Typical speech recognition process

End-to-end learning is one of many approaches of deep learning process where all the parameters in the process are trained jointly, rather than step-by-step[13]. What makes it different is that it bypasses all the intermediate stages presented in the original pipeline, replacing those with a neural network, which makes the optimization becomes simpler. The above pipeline is now simplified to as:

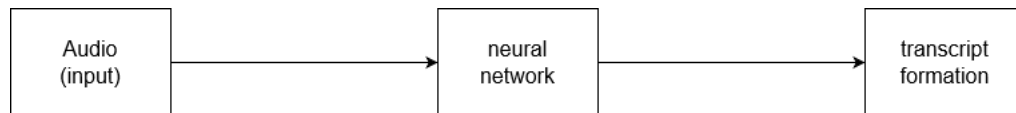


Figure 2.18: Speech recognition process with end-to-end learning applied

By neglecting the intermediate steps, the deep knowledge of the problem could be temporarily ignored and task-specific engineering could be avoided. This is expressed by R. Collobert and his colleagues in the article *Natural Language Processing (Almost) from Scratch*[14] as they also desired of avoiding task-specific engineering by using their unified neural network architecture. Nevertheless, this approach requires a large amount of data, larger than the usual amount when using the regular pipeline. Moreover, the data need to be proper with the network architecture so that the net can extract useful information and make the decision based on it and this leads to cases where the provided data are insufficient or inefficient for the end-to-end learning approach.

2.5 End-to-end learning in autonomous driving

Autonomous driving can be considered as a classic and the most suitable example for end-to-end learning as the pipeline includes multiple intermediate layers that can be shortened. In the article *A Standard Driven Software Architecture for Fully Autonomous Vehicles*, A. Serban and his colleagues introduced an autonomous driving system architecture which composed of different layers for different purposes [15].

From what is displayed in figure 2.19, the leftmost block represents the sensors' abstraction. It shows a list of sensors that are used to gather the input data, those include cameras, LiDAR, radars, ultrasonic sensors, etc. These data are then processed in the sensor fusion layer to extract features, to detect and classify different types of objects, to position the vehicle. This information is needed to create a world model, which is what happens in the next layer, a virtual world model that simulates the surrounding external environment of the vehicle. At this stage, the system creates the possible predictions of the environment and the vehicle, along with the predefined goals, multiple behaviors are developed and the best ones are then selected.

To realize the behavior, the vehicle must perform a certain set of maneuvers. The objectives of the planning layer are to create a path that must avoid any possible collisions and accidents and these can be divided into smaller sub-tasks such as lane-keeping, lane change, collision avoidance, etc. They are monitored and coordinated so that the maneuvers are executed as the vehicle is kept in the trajectory path.

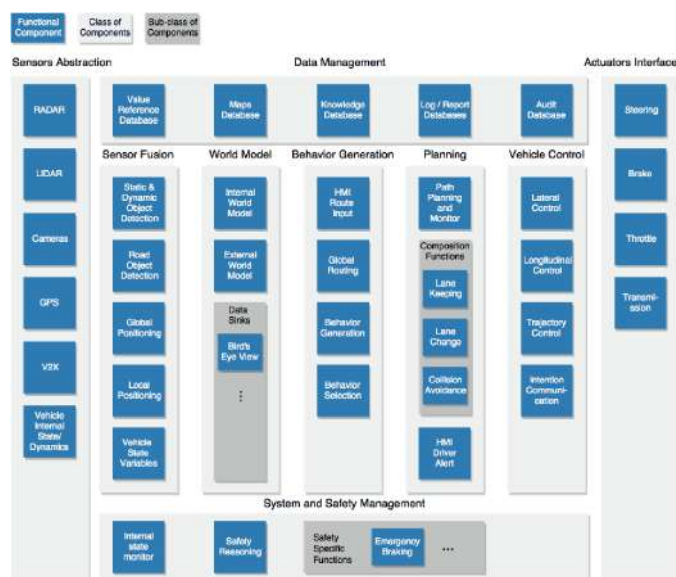


Figure 2.19: Functional architecture example of autonomous driving system[15]

Finally, the control values needed for those functions are sent to the actuators in the vehicle control layer.

This is an ideal example for the application of end-to-end learning, the whole pipeline can be shortened to a simple process where the data has the typical form of (x, y) with x is the raw input data from the sensor abstraction layer and y is the control signal for the last layer, vehicle control. Undoubtedly, a special network architecture is needed to realize the process and luckily, this had been accomplished by M. Bojarski and his colleagues at NVIDIA Corporation. In their paper *End to End Learning for Self-Driving Cars*, they proposed a system that is capable of controlling the vehicle autonomously using the raw data images from the attached cameras. More specifically, they trained a convolutional neural network to map the raw pixels from the camera directly to the steering commands[3].

They claimed that it can automatically learn internal representations without explicit training and the result was fascinating as the vehicle was able to drive itself in traffic on roads with or without lane markings. The block diagram shown in figure 2.20[3] describes how the system gathers data and how the CNN is trained. At first, the images from the three cameras are fed into the CNN and the CNN will compute a proposed steering command. This value is then compared with the desired steering command from the human driver to obtain the error between them. By using the backpropagation technique mentioned in chapter 1, the error is then used to adjust the weights of the CNN so that the returned final output is as desired. Once the training is complete and the model is obtained, it will be used to generate steering command using the images from the camera directly.

As displayed in figure 2.21, the network is formed of 9 layers, which includes one normalization layer, five convolutional layers, one flatten layer, and three fully connected layers.

Normalization is a technique that is said to accelerate the model's training process. In particular, A. Bindal stated in his journal[16] that by normalizing each feature,

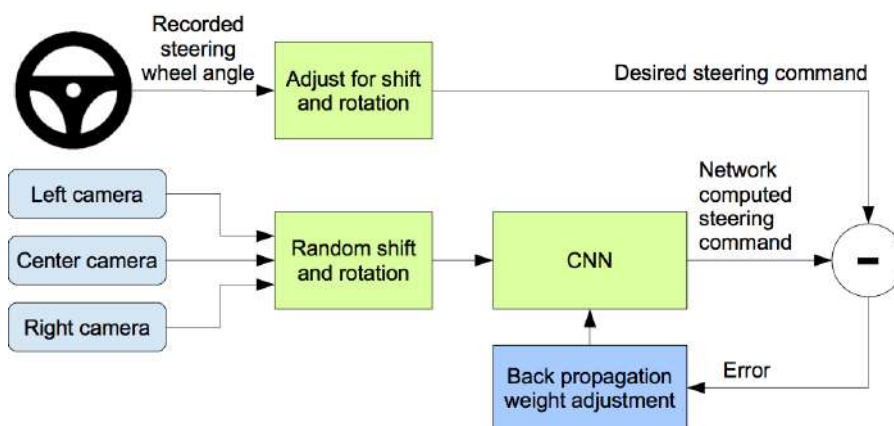


Figure 2.20: Training system for self-driving car[3]

the contribution of every feature is maintained, making the network unbiased even though some feature has higher numerical value than others. Another benefit is that it helps to reduce internal covariate shift, the change in the distribution of network activations caused by the change in network parameters during training. Finally, it increases the optimization process as it prevents the weights from exploding and restricts them to a certain range[16].

Multilayer perceptron neural network (MLP), as described in chapter 1, is one of the traditional neural networks. They are considered to be inefficient in image processing[17], especially large images with 3 colour channels which create a vast amount of weights to be trained and managed which may potentially result in overfitting. Moreover, the MLP adjusts the weights differently although it is the same input but at different angles. In an image, nodes that are close together define the features of the image at some points, nevertheless, as described in chapter 2.3, the hidden layer is flattened as this resulted in the loss of information. A solution to this problem is by using a convolutional neural network. Basically, it applies a filter (normally with a size of 3x3 or 5x5) to the input images and moves across it from top left to bottom right. Using the convolution operation, a value is calculated for each point on the image where the filter is placed. In the paper[3], the strided convolutions with a 2x2 stride and 5x5 kernel are used in the first three layers and a non-strided convolution with a 3x3 kernel is used in the last two convolutional layers. Here, the term "stride" describes the steps that the filter moves.

During training there are chances that the model models the training data too well and this is known as overfitting. This happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. A way to avoid this problem is by applying a regularization method, known as dropout, to approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "dropped out." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "view" of the configured layer. The fraction of the input units to drop is set as 0.5

The Flatten layer is used to reshape the input to a 1D-array with the length is the

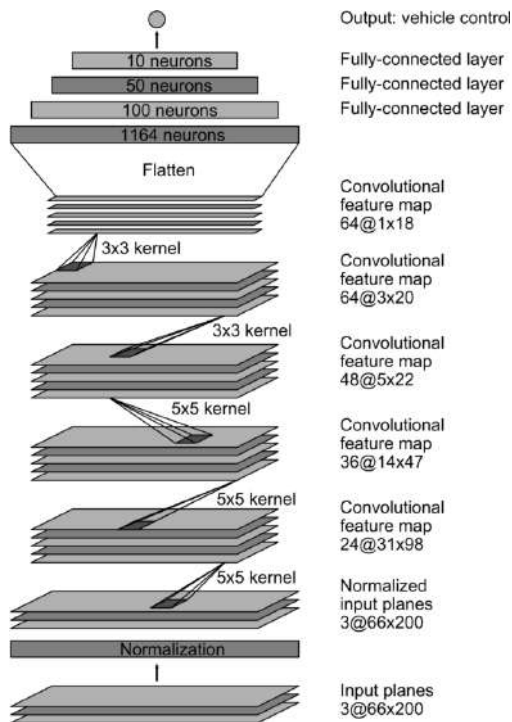


Figure 2.21: CNN architecture for self-driving car[3]

number of element of the input. This is required before going to the fully connected layers.

Lastly, the fully connected layers are used to connect all the inputs from one layer to every activation unit of the next layer and as in [3], these layers are designed as a controller for steering.

A series of figures are shown below for the visualization of the output of each layer using a sample image.

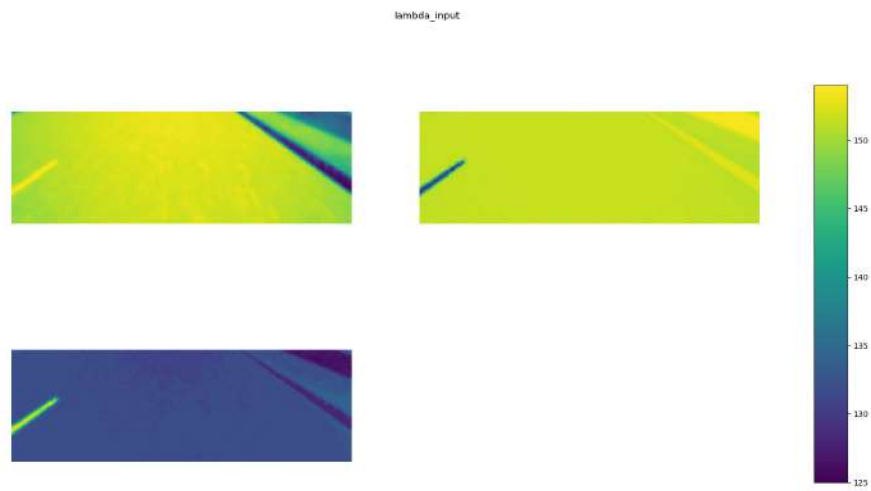


Figure 2.22: Input

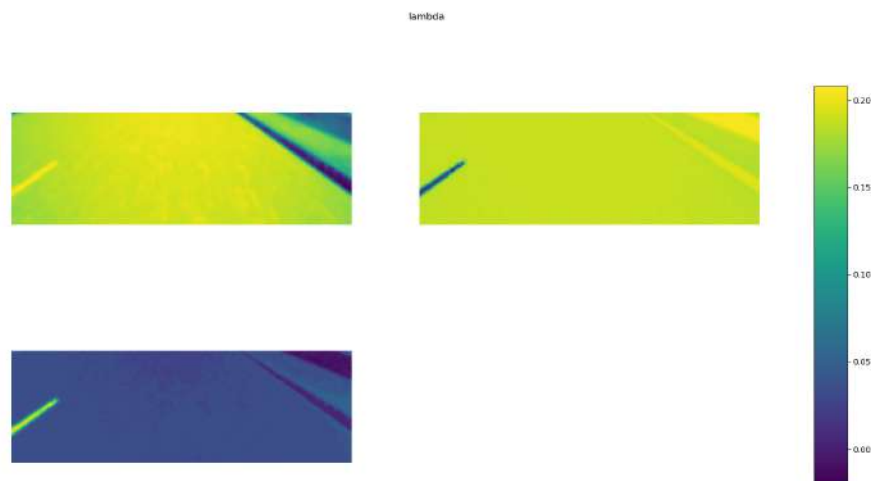


Figure 2.23: Output of normalization layer

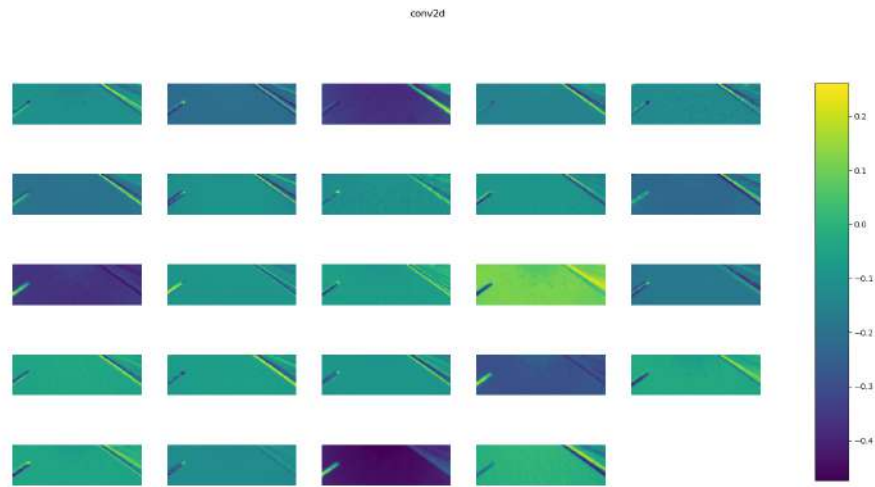


Figure 2.24: Output of the first convolutional layer

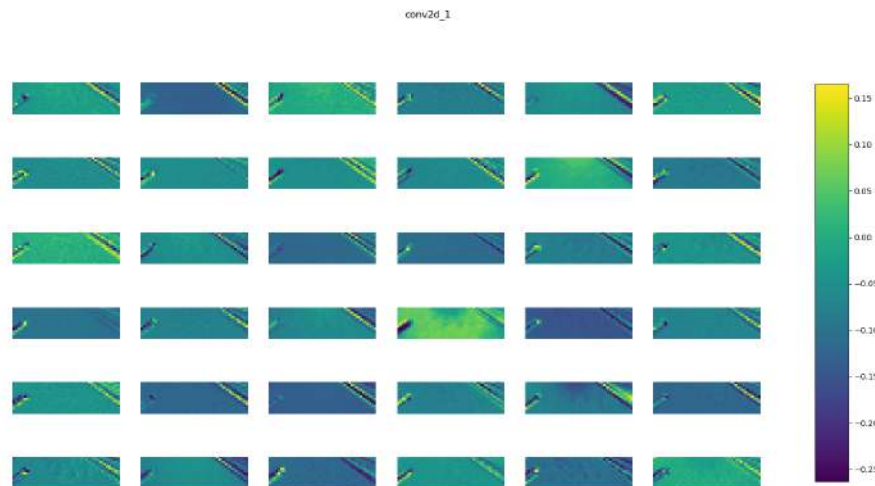


Figure 2.25: Output of the second convolutional layer

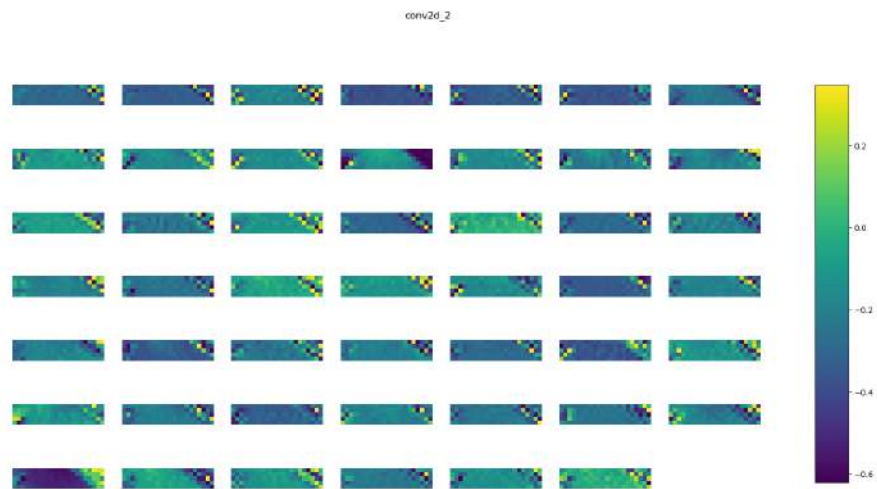


Figure 2.26: Output of the third convolutional layer

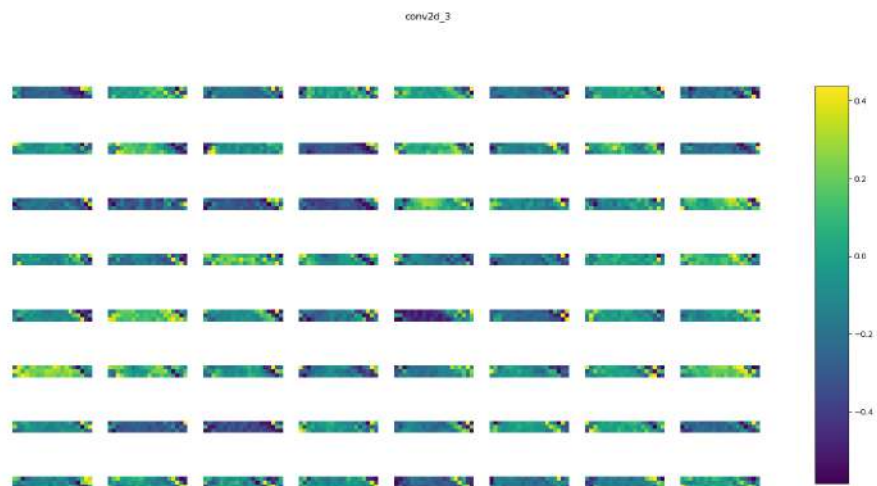


Figure 2.27: Output of the fourth convolutional layer

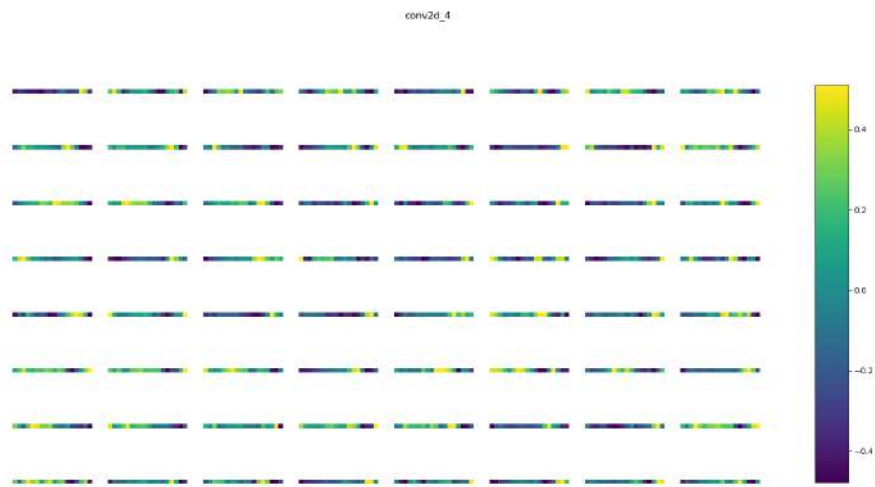


Figure 2.28: Output of the fifth convolutional layer

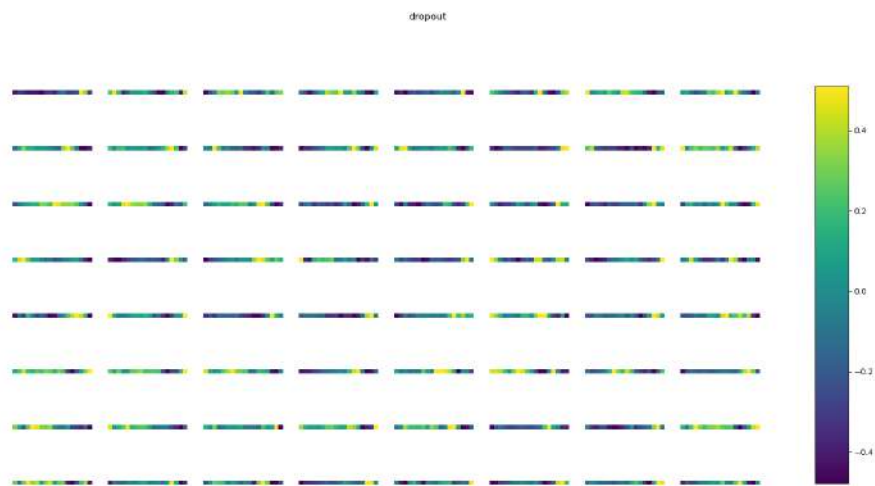


Figure 2.29: Output of the dropout layer

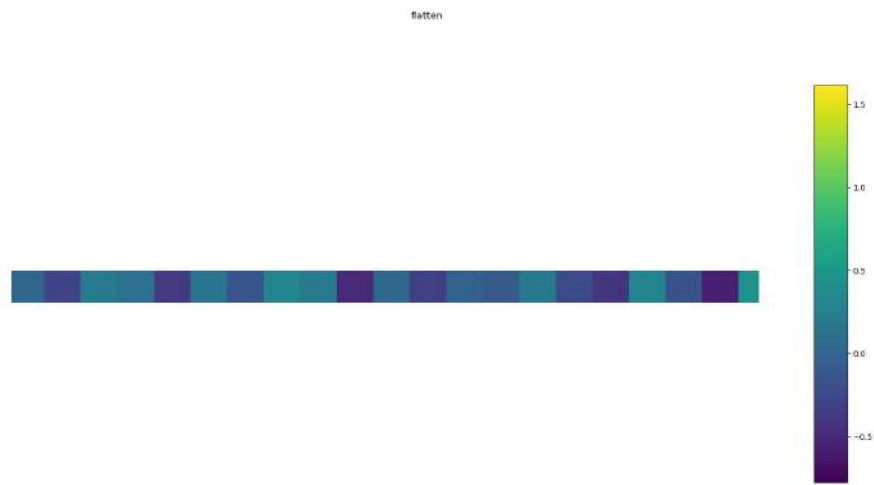


Figure 2.30: Zoomed in output of the dropout layer



Figure 2.31: Output of the first dense layer

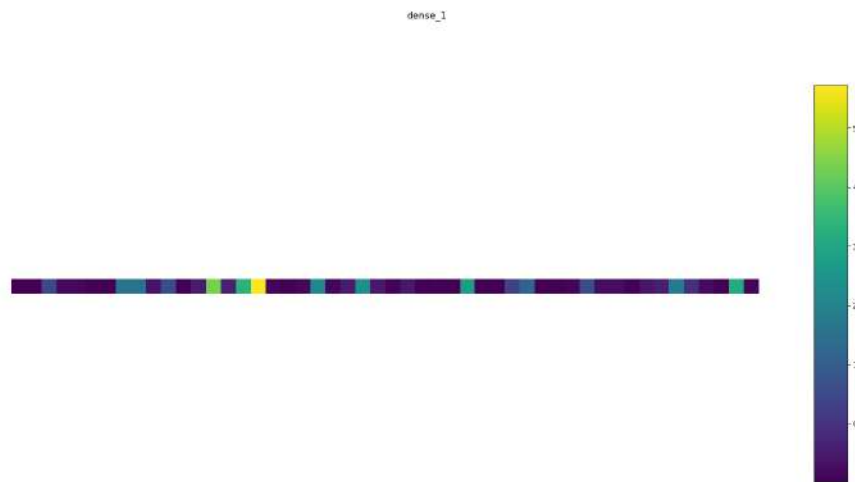


Figure 2.32: Output of the second dense layer



Figure 2.33: Output of the third dense layer

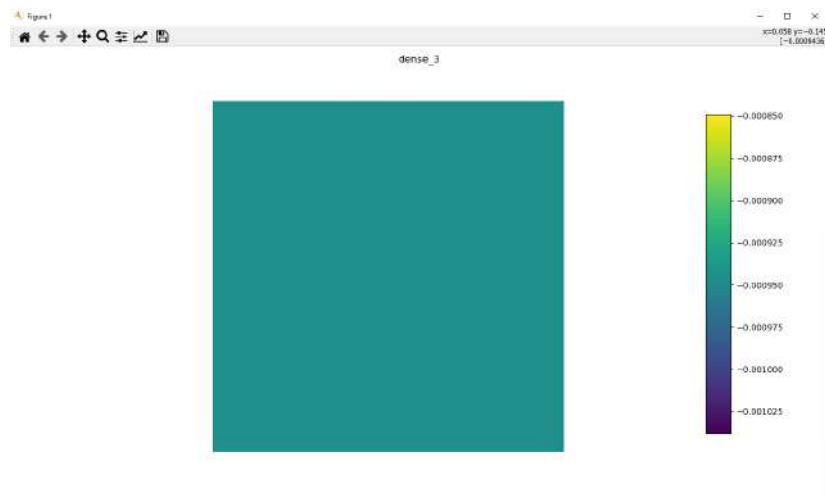


Figure 2.34: Final output

2.6 Implementation in Carla

The standard process for the entire pipeline consists of the following steps:

- environment setup
- data gathering
- model training
- model testing

2.6.1 Environment setup

A drawback of using CARLA is that it requires an adequate GPU for realistic simulations. The specification of the system used for this project is listed in table 2.1.

Table 2.1: System setup

Specifications	
Processor	Intel(R) Core(TM) i5-7300HQ CPU @ 2.5GHz
RAM	8.00 GB
System type	Windows 10 (64-bit)
GPU	GeForce GTX 1050
Graphic Card Driver	456.38
CUDA	10.1
cuDNN	7.6.5
Environments	
Python	3.7.7
Anaconda	4.8.4
Libraries	
numpy	1.19.1
scipy	1.5.0
matplotlib	3.3.0
opencv	4.4.0
tensorflow-gpu	2.1.0
eventlet	0.26.1
flask	1.1.2
socketio	4.5.1
pillow	7.2.0

2.6.2 Data collection

As introduced in chapter 1, the map used for developing the end-to-end learning algorithm is town02. A single camera is spawn at the bonnet of the vehicle with a yaw value of -10° to capture the view of the path with the dimension of 320x160 in width and height respectively, some recorded samples are shown in figure 2.35.

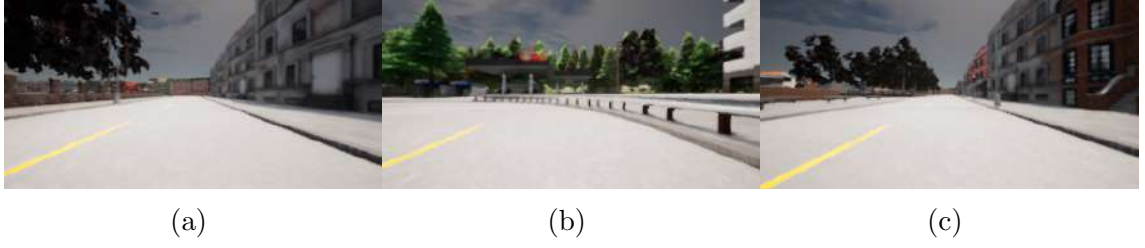


Figure 2.35: Example of recorded frames

Each of the captured frames is labeled with a group of values, which are xy coordinate, boolean value to check whether the vehicle is near an intersection, left and right lane type, throttle, brake and steer values. The labeled frames are shown in figure 2.36.

IMG/1614008828962.png	17.694077	105.3763	FALSE	Broken	NONE	0.3793725	0	-0.003681
IMG/1613938937795.png	11.149892	105.81202	FALSE	Broken	NONE	0.3518459	0	-0.01065
IMG/1614009320676.png	107.13837	106.50024	FALSE	Broken	NONE	0.6432453	0	0.0004918
IMG/1613934892902.png	41.902718	279.11606	FALSE	Broken	NONE	0.3793711	0	0.0010172
IMG/1613930179504.png	193.65031	129.57231	FALSE	Broken	NONE	0.379368	0	0.0002033
IMG/1613930093669.png	193.72644	163.33652	FALSE	Broken	NONE	0.3748122	0	-0.001613
IMG/1613912695839.png	69.993317	306.53836	FALSE	Broken	NONE	0.379419	0	0.0021774
IMG/1613938923815.png	147.51463	106.15892	FALSE	Broken	NONE	0.7	0	0.0033904
IMG/1614009283937.png	129.77277	106.38396	FALSE	Broken	NONE	0.7	0	0.0001606
IMG/1613929528115.png	74.239708	110.42683	FALSE	Broken	NONE	0.6617471	0	-0.000505

Figure 2.36: Recorded labels

A data collection session starts with a vehicle spawned at a random point on the map with the said camera, roaming around the map freely using Carla built-in autopilot. Although the user is able to use the 4 keys W, A, S, D to control the vehicle, the difficulty and disruption make the built-in autopilot mode a better choice. After few hours, approximately 12000 samples were collected and ready for the training. Figure 2.37 shows the distribution of the steering values of the collected data set.

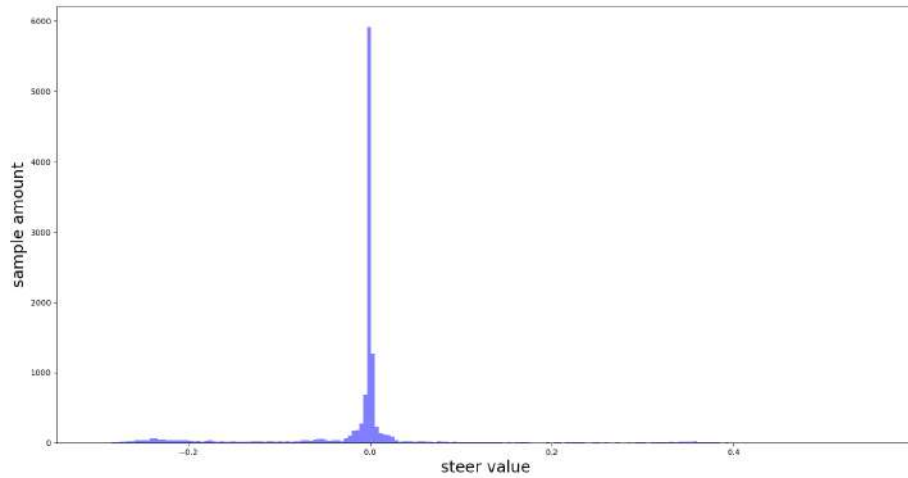
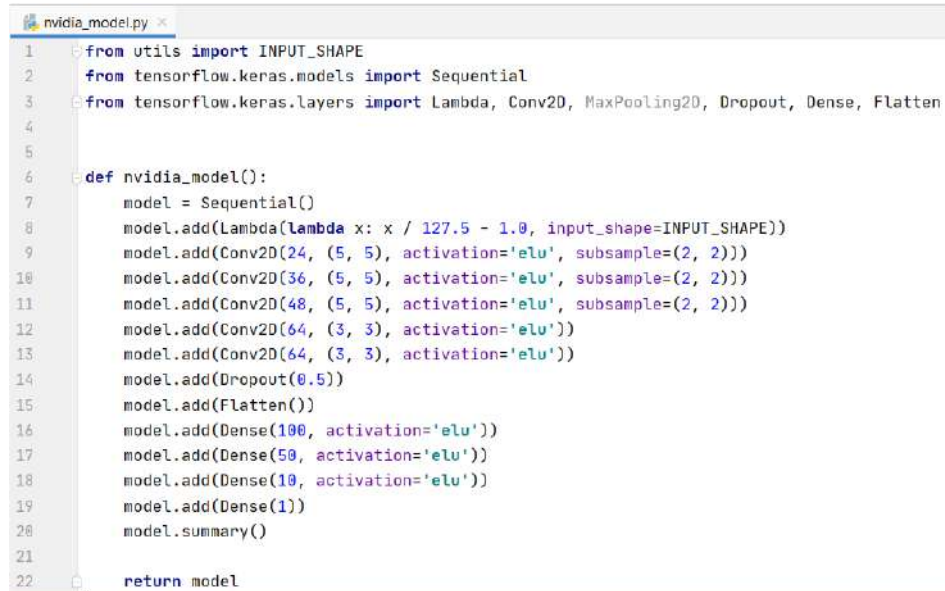


Figure 2.37: Steering values distribution

2.6.3 Model training

The initial thing before starting the training is to have a training network. Using the library Tensorflow Keras, the PilotNet mentioned in section 2.5 is built as in figure 2.38. The script for the network building is placed in the file *nvidia_model.py*.



```

1 from utils import INPUT_SHAPE
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Lambda, Conv2D, MaxPooling2D, Dropout, Dense, Flatten
4
5
6 def nvidia_model():
7     model = Sequential()
8     model.add(Lambda(lambda x: x / 127.5 - 1.0, input_shape=INPUT_SHAPE))
9     model.add(Conv2D(24, (5, 5), activation='elu', subsample=(2, 2)))
10    model.add(Conv2D(36, (5, 5), activation='elu', subsample=(2, 2)))
11    model.add(Conv2D(48, (5, 5), activation='elu', subsample=(2, 2)))
12    model.add(Conv2D(64, (3, 3), activation='elu'))
13    model.add(Conv2D(64, (3, 3), activation='elu'))
14    model.add(Dropout(0.5))
15    model.add(Flatten())
16    model.add(Dense(100, activation='elu'))
17    model.add(Dense(50, activation='elu'))
18    model.add(Dense(10, activation='elu'))
19    model.add(Dense(1))
20    model.summary()
21
22    return model
  
```

Figure 2.38: End-to-end learning neural network architecture

For starting, the csv file is loaded and split into features and labels. The features are the names of the images while the label is only the steering value. They are then split into 4 different groups named *x_train*, *x_valid*, *y_train*, *y_valid*, which stand for training and validation. The validation group is used to validate the model while being trained.

As mentioned in chapter 2, the end-to-end learning approach requires a huge amount of data and 12000 samples are not enough for the model to be generalised. This is when data augmentation is applied to create augmented data from the original ones. The processed that were applied are image random flipping, image random translation and random shadow. A reference sample of the original image is shown with the original steering value set as the title is shown in figure 2.39.

-0.086176112



Figure 2.39: The original sample

The flipping process flips the image and changes the sign of the steer value at the same time. However, this should only be used if the road is one way, without the middle lane markings. Being used in a 2 lanes traffic system will accidentally lead to a result where the vehicle is trained to run against traffic.

0.086176112



Figure 2.40: A flipped sample

The random translation process translates the image up and down, right and left randomly and the respective steering value is also adjusted according to adjusted distance in the x-axis.



Figure 2.41: Random translated samples

The random shadow randomly creates a dark mask the cover the image at a random

region, this helps the model to ignore the affect of reflection or shadow that might appear on the road.



Figure 2.42: Random shadow samples

Finally, the image is cropped, resized to 200x66, and converted from RGB to YUV to meet the requirements of the training input.



Figure 2.43: Final processed samples for the training

Once the processing steps are finished, the training can begin. Some parameters for the training is displayed in table 2.2.

Table 2.2: Training parameters

batch_size	40
learning rate	1.0e-4
number of epoch	50
steps per epoch	10000
metric	Mean Squared Error
optimizer	Adam

With 10000 steps per epoch, it will probably take weeks of training to complete 50 epochs on the system specified in table 2.1. The dataset was moved to another system equipped with the Tesla V100 PCIe 32GB GPU and as a result, the full training time reduced to less than 16 hours. Once the training is complete, an *.h5* file is exported. This is the model, or more precisely, the network with all the updated weights that can be used for the prediction of steering value.

2.6.4 Training results

The training and validation losses of 50 epochs training is shown in figure 2.44. The model of epoch 50 with the validation loss of 0.000090 was chosen for the final testing.

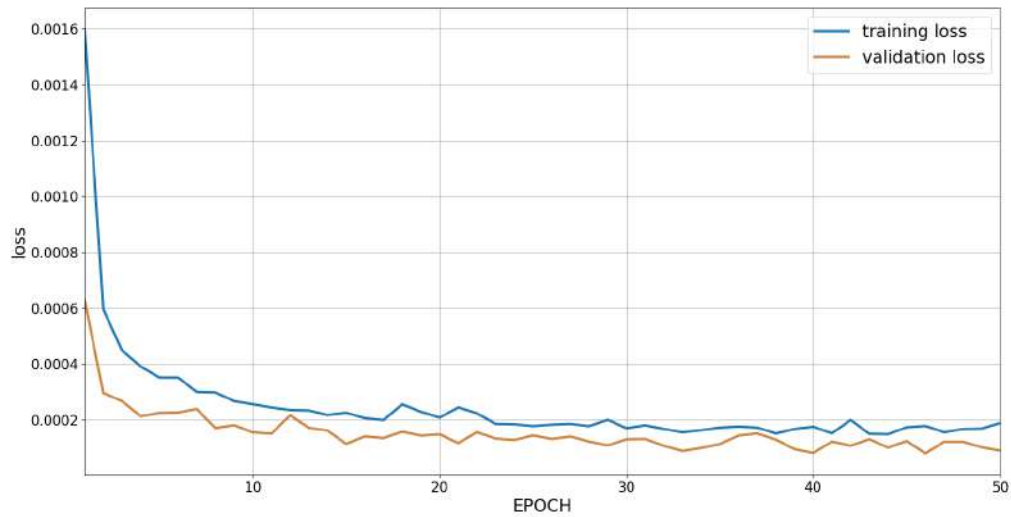
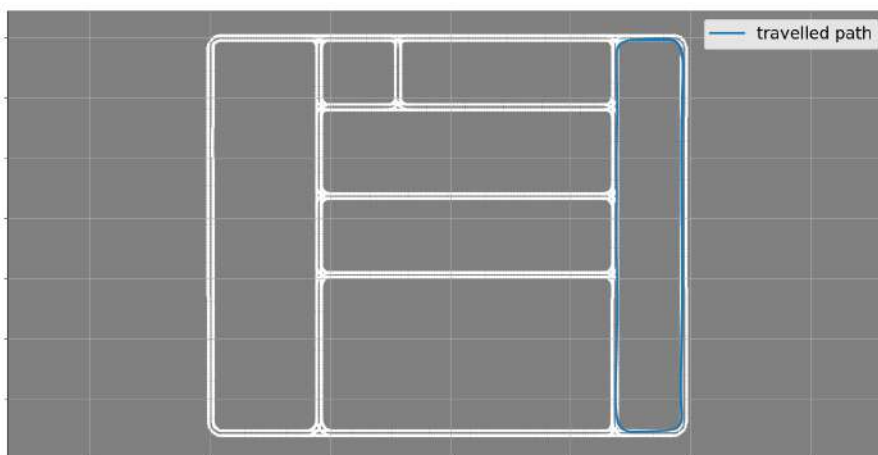
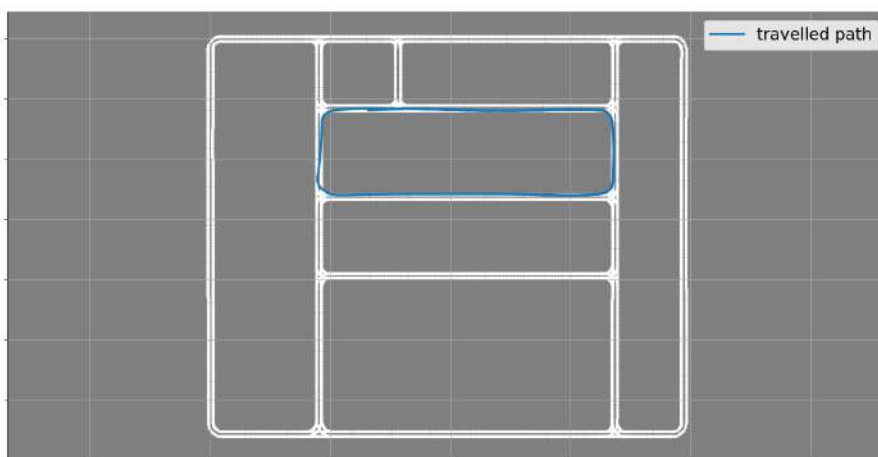


Figure 2.44: Training result

It would be meaningless to perform the test on the same track where the data were collected; therefore, the map town01 with similar road features was used instead. The result of 2 tests are shown in figure 2.45.



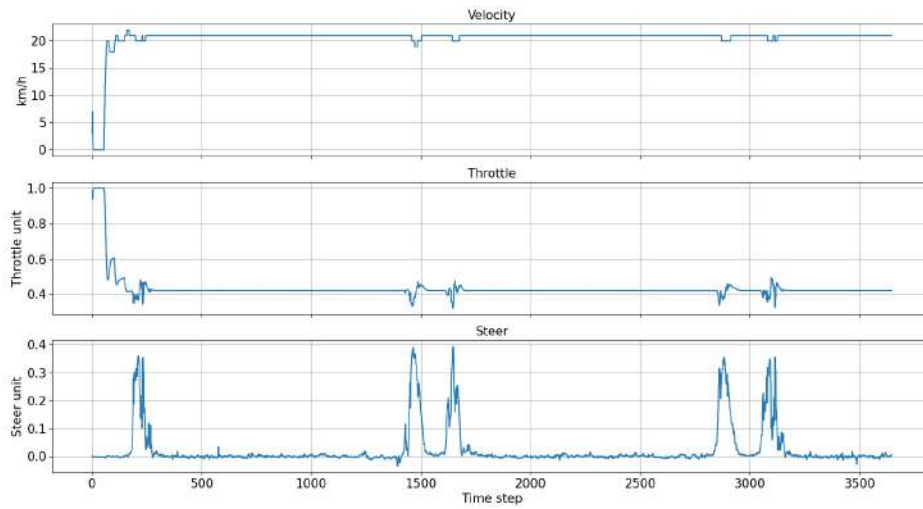
(a) Test 1



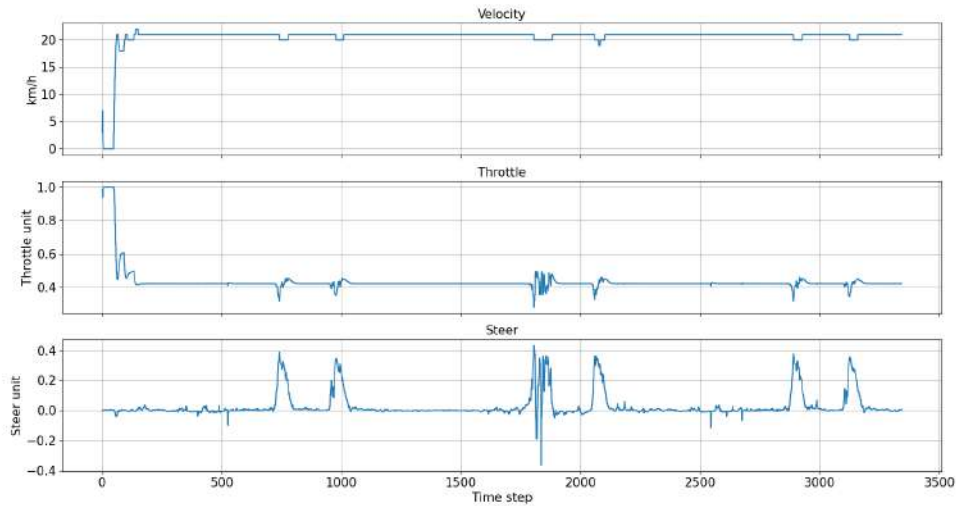
(b) Test 2

Figure 2.45: E2E test visualization

The velocity and control parameters of each test are shown in figure 2.46a and 2.46b.



(a) Test 1



(b) Test 2

Figure 2.46: E2E control parameters

The conclusion is the vehicle responds well when approaching the corner without serious oscillations although it has never been trained with the road sample of town01. This proves that the resulted model is generalised.

Chapter 3

Model predictive control

Despite the performance of the end-to-end learning approach, it is unable to handle navigation, particularly when travelling from one place to another while passing through multiple intersections. With the goal of developing a fully-autonomous system, the vehicle should be able to navigate through intersections, and the best way is by using Model Predictive Control (MPC) with the help of Dijkstra's algorithm path finding.

As described at the opening, the MPC module will take over the E2E controller whenever the vehicle is close to an intersection. At this point, the vehicle will drive as close to the predefined way points as possible, which will help it overcome the intersection and navigate correctly. The goal of the MPC module is to keep the car as close to the predefined path as possible, particularly, this is done by solving the optimization problem of the steer and throttle values to reduce the cost difference between the predicted way point and the actual way point.

Model Predictive Control is a feedback control algorithm, that uses a model to make predictions about future output of a process based on the constraints of the system. For this particular project, it makes predictions of the future trajectories based on all the possible choices of control and chooses the one that keeps the vehicle as close to the desired trajectory as possible.

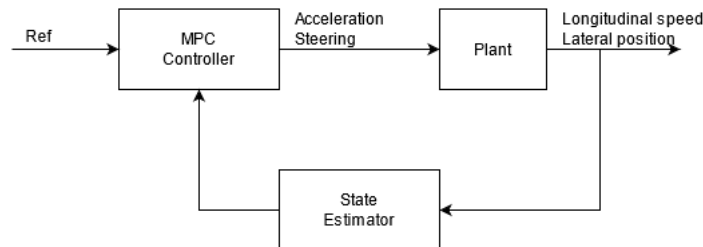


Figure 3.1: MPC controller in autonomous driving system

One of the reasons why MPC is used is because it is a multi-variable controller that controls the output simultaneously by taking into account all the interactions between system variables. Another reason is the ability to handle constraints of the system to prevent violations that might lead to consequences. The typical

constraints are maximum and minimum speed, maximum steering limit, acceleration limit, etc.

The predictions of the plant outputs are made using a model of the plant and an optimizer, which ensures that the predicted future plant output tracks the desired reference. The car model is used to simulated the vehicle's path in a range of time steps with the input is a set of control options.

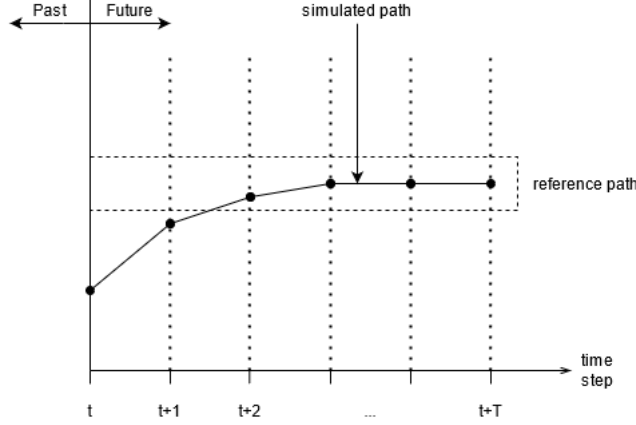


Figure 3.2: Simulated path generated in T time steps

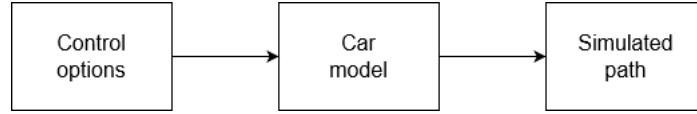


Figure 3.3: Car model uses control options to simulate possible path

The value T represents the length of the time into the future or the number of future time steps for measuring how far ahead the MPC looks into the future and is referred to as the prediction horizon. After simulating all possible simulated paths, the MPC controller needs to find the best one that is closest to the reference, whose responsibility belongs to the optimizer. In an example where only steering is taken into account, the optimizer tries to minimize the error between the reference path and the predicted path of the vehicle, and the changes in the steering wheel from this time step to the next one to avoid motion sickness while ensuring the vehicle sticks to the reference path under the given constraints. The cost function J of this problem is presented as:

$$J = \sum_{i=1}^T w_e e_{k+i}^2 + \sum_{i=0}^{T-1} w_{\Delta u} \Delta u_{k+i}^2 \quad (3.1)$$

where:

e : the error between the reference and the predicted path

w_e : weight of the error of each time step

Δu : change in steering

$w_{\Delta u}$: weight of the change of steering of each time step

At time step t , the path with the smallest J is chosen as the optimal path and the sequence of steering that generates the path is used as the final result; however, only the first control set of the sequence is applied with the rest is disregarded. With the chosen steering, the car moves to the predicted position at time step $t + 1$ and the process is repeated. Additionally, there might be a difference between the actual position and the predicted position due to unaccounted external disturbance such as wind, slippery road surface, etc.

The parameters needed for the design of the MPC controller are the sample time T_s , the prediction horizon, the control horizon, the constraints and weights. The choosing of sample time decides the rate the controller execute the algorithm. A large sample time cause slowness in reaction when unexpected disturbances happen. On the other hand, a small sample time allows fast reaction but it causes an excessive computational load. The recommended range is $\frac{T_r}{20} \leq T_s \leq \frac{T_r}{10}$ where T_r is the time needed for the response to rise from 10% to 90% of the steady-state response.

The choosing of the prediction horizon should also cover the dynamics of the system as well such as a full stop requires at least 3 seconds since the application of the brake, the recommendation is to have $T \cdot T_s \geq T_{settling}$ where $T_{settling}$ is the time needed for the error to fall to within 2% of the reference value.

The next design parameter is the control horizon, the set of future control actions that leads to the predicted output. A small control horizon may be better for the computation load but might not give the best possible manoeuvre. The control horizon might be chosen to be the same as the prediction horizon; since only the first few moves have a significant impact on the predicted output while the rest have a minor effect, choosing a larger control horizon is redundant, the recommendation is setting it to 10% to 20% of the prediction horizon.

During the optimization, it might take a large number of iterations to find the most optimal solutions. This is problematic, especially when this happens to every time step, which may cause the computation time to exceed the controller sample time. A solution is putting a limit on the number of iterations and find a sub-optimal solution instead, the number is determined by performing tests on the system and compare with the sample time to find the best option.

For this project, the vehicle model is considered as the bicycle model and the state of the model is defined as follows:

$$z = [x, y, v, \phi] \quad (3.2)$$

where x is the x-position, y is the y-position, v is the velocity and ϕ is the yaw angle. The input vector includes two components, which are acceleration a and steering rate δ , and it is defined as:

$$u = [a, \delta] \quad (3.3)$$

From equation 3.1 and according to [18], the cost function for this project is developed as:

$$\min(Q_f(z_{T,ref} - z_T)^2 + Q\Sigma(z_{t,ref} - z_t)^2 + R\Sigma u_t^2 + R_d\Sigma(u_{t+1} - u_t)^2) \quad (3.4)$$

with:

R : input cost matrix

R_d : input difference cost matrix

Q : state cost matrix

Q_f : state final matrix

The term $Q_f(z_{T,ref} - z_T)^2$ defines the difference between final time step reference state and the actual state. The term $Q\Sigma(z_{t,ref} - z_t)^2$ defines the difference between reference states and the actual states throughout the horizon. The term $R\Sigma u_t^2$ makes sure the inputs comply with the constraint (3.6) and (3.8); and finally, the term $R_d\Sigma(u_{t+1} - u_t)^2$ constrains the rate of change between the current inputs and the next inputs. The cost function (3.4) is put under the following constraints:

- Maximum steering speed

$$|u_{t+1} - u_t| < du_{max} \quad (3.5)$$

- Maximum steering angle

$$|u_t| < u_{max} \quad (3.6)$$

- Maximum and minimum speed

$$v_{min} < v_t < v_{max} \quad (3.7)$$

- Maximum and minimum input

$$u_{min} < u_t < u_{max} \quad (3.8)$$

The linearisation begins with:

$$\dot{x} = v\cos(\phi) \quad (3.9)$$

$$\dot{y} = v\sin(\phi) \quad (3.10)$$

$$\dot{v} = a \quad (3.11)$$

$$\dot{\phi} = \frac{v\tan(\delta)}{L} \quad (3.12)$$

The state equation of the system is written as:

$$\dot{z} = \frac{\partial}{\partial z} z = f(z, u) = A'z + B'u \quad (3.13)$$

A' is the state matrix, which expresses the dependency of the state on the changes of each element, where:

$$A' = \begin{bmatrix} \frac{\partial}{\partial x} v \cos(\phi) & \frac{\partial}{\partial y} v \cos(\phi) & \frac{\partial}{\partial v} v \cos(\phi) & \frac{\partial}{\partial \phi} v \cos(\phi) \\ \frac{\partial}{\partial x} v \sin(\phi) & \frac{\partial}{\partial y} v \sin(\phi) & \frac{\partial}{\partial v} v \sin(\phi) & \frac{\partial}{\partial \phi} v \sin(\phi) \\ \frac{\partial}{\partial x} a & \frac{\partial}{\partial y} a & \frac{\partial}{\partial v} a & \frac{\partial}{\partial \phi} a \\ \frac{\partial}{\partial x} \frac{v \tan(\delta)}{L} & \frac{\partial}{\partial y} \frac{v \tan(\delta)}{L} & \frac{\partial}{\partial v} \frac{v \tan(\delta)}{L} & \frac{\partial}{\partial \phi} \frac{v \tan(\delta)}{L} \end{bmatrix} \quad (3.14)$$

$$= \begin{bmatrix} 0 & 0 & \cos(\bar{\phi}) & -\bar{v} \sin(\bar{\phi}) \\ 0 & 0 & \sin(\bar{\phi}) & \bar{v} \cos(\bar{\phi}) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\tan(\bar{\delta})}{L} & 0 \end{bmatrix}$$

Matrix B' is the input matrix, which expresses the dependency of the state on the changes of each unput element, where:

$$B' = \begin{bmatrix} \frac{\partial}{\partial a} v \cos(\phi) & \frac{\partial}{\partial \delta} v \cos(\phi) \\ \frac{\partial}{\partial a} v \sin(\phi) & \frac{\partial}{\partial \delta} v \sin(\phi) \\ \frac{\partial}{\partial a} a & \frac{\partial}{\partial \delta} a \\ \frac{\partial}{\partial a} \frac{v \tan(\delta)}{L} & \frac{\partial}{\partial \delta} \frac{v \tan(\delta)}{L} \end{bmatrix} \quad (3.15)$$

$$= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & \frac{\bar{v}}{L \cos^2(\bar{\delta})} \end{bmatrix}$$

z_{k+1} is expressed in discrete-time mode with Forward Euler Discretization with sampling time dt as follows:

$$z_{k+1} = z_k + f(z_k, u_k) dt \quad (3.16)$$

By using first degree Taylor expansion:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!} (x - a)^n \quad (3.17)$$

The expression (3.16) can be written with \bar{z} and \bar{u} as:

$$z_{k+1} = (I + dt A') z_k + (dt B') u_k + (f(\bar{z}, \bar{u}) - A' \bar{z} - B' \bar{u}) dt \quad (3.18)$$

$$\begin{aligned}
 z_{k+1} &= z_k + (f(\bar{z}, \bar{u}) + \begin{pmatrix} A' \\ B' \end{pmatrix}((z_k + u_k) - (\bar{z} + \bar{u})))dt \\
 z_{k+1} &= z_k + (f(\bar{z}, \bar{u}) + A'z_k + B'u_k - A'\bar{z} - B'\bar{u})dt
 \end{aligned}$$

and this can be rewritten to be the linearised equation of the vehicle model as:

$$z_{k+1} = Az_k + Bu_k + C \quad (3.19)$$

where:

$$A = (I + dtA') = \begin{bmatrix} 1 & 0 & \cos(\bar{\phi})dt & -\bar{v}\sin(\bar{\phi})dt \\ 0 & 1 & \sin(\bar{\phi})dt & \bar{v}\cos(\bar{\phi})dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{\tan(\bar{\delta})}{L}dt & 1 \end{bmatrix} \quad (3.20)$$

$$B = dtB' = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ dt & 0 \\ 0 & \frac{\bar{v}}{L\cos^2(\bar{\delta})}dt \end{bmatrix} \quad (3.21)$$

and C is computed as:

$$\begin{aligned}
 C &= (f(\bar{z}, \bar{u}) - A'\bar{z} - B'\bar{u})dt \\
 &= dt \left(\begin{bmatrix} \bar{v}\cos(\bar{\phi}) \\ \bar{v}\sin(\bar{\phi}) \\ \bar{a} \\ \frac{\bar{v}\tan(\bar{\delta})}{L} \end{bmatrix} - \begin{bmatrix} \bar{v}\cos(\bar{\phi}) - \bar{v}\sin(\bar{\phi})\bar{\phi} \\ \bar{v}\sin(\bar{\phi}) + \bar{v}\cos(\bar{\phi})\bar{\phi} \\ 0 \\ \frac{\bar{v}\tan(\bar{\delta})}{L} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ \bar{a} \\ \frac{\bar{v}\bar{\delta}}{L\cos^2(\bar{\delta})} \end{bmatrix} \right) \\
 &= \begin{bmatrix} \bar{v}\sin(\bar{\phi})\bar{\phi}dt \\ -\bar{v}\cos(\bar{\phi})\bar{\phi}dt \\ 0 \\ -\frac{\bar{v}\bar{\delta}}{L\cos^2(\bar{\delta})}dt \end{bmatrix}
 \end{aligned}$$

The definition of matrices A , B and C was done as in figure 3.4.

```

def get_linear_model_matrix(v, phi, delta):

    A = np.zeros((NX, NX))
    A[0, 0] = 1.0
    A[1, 1] = 1.0
    A[2, 2] = 1.0
    A[3, 3] = 1.0
    A[0, 2] = DT * math.cos(phi)
    A[0, 3] = - DT * v * math.sin(phi)
    A[1, 2] = DT * math.sin(phi)
    A[1, 3] = DT * v * math.cos(phi)
    A[3, 2] = DT * math.tan(delta) / WB

    B = np.zeros((NX, NU))
    B[2, 0] = DT
    B[3, 1] = DT * v / (WB * math.cos(delta) ** 2)

    C = np.zeros(NX)
    C[0] = DT * v * math.sin(phi) * phi
    C[1] = - DT * v * math.cos(phi) * phi
    C[3] = - DT * v * delta / (WB * math.cos(delta) ** 2)

    return A, B, C

```

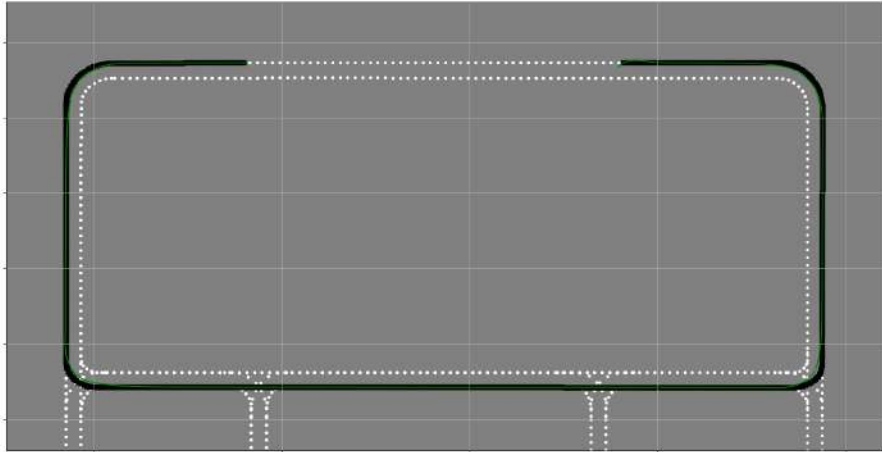
Figure 3.4: Matrices definition

The optimization is done using the Python library `cvxpy`[19] that returns the optimized values, which includes the acceleration, steering rate, x and y location, steering angle and velocity. The parameters set for the implementation are shown in table 3.1:

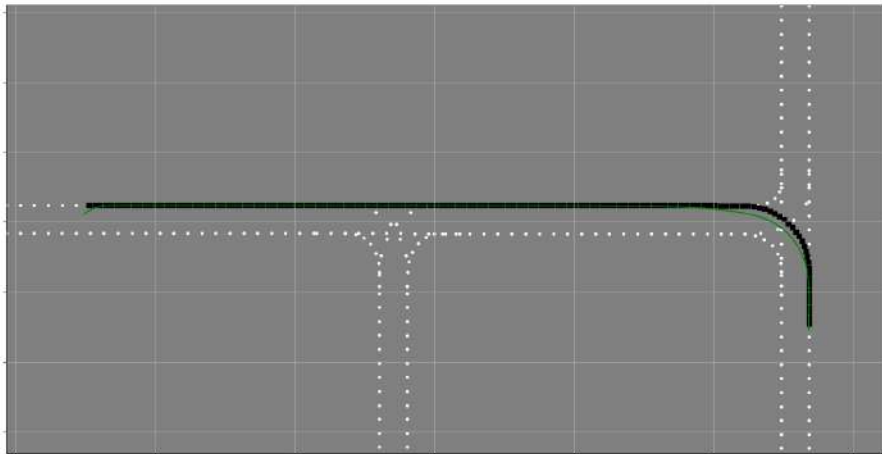
Table 3.1: MPC parameters

Horizon length	6
Max iteration	3
Max speed (km/h)	45
Min speed (km/h)	0
Max steering (degree)	45
Max steering speed (degree/s)	30
Max acceleration	1.0

Below are the results of 2 tests performed on predefined random course. Figure 3.5 shows the coordinates of the vehicle (marked in green) during the process of staying close to the predefined path (marked in black).



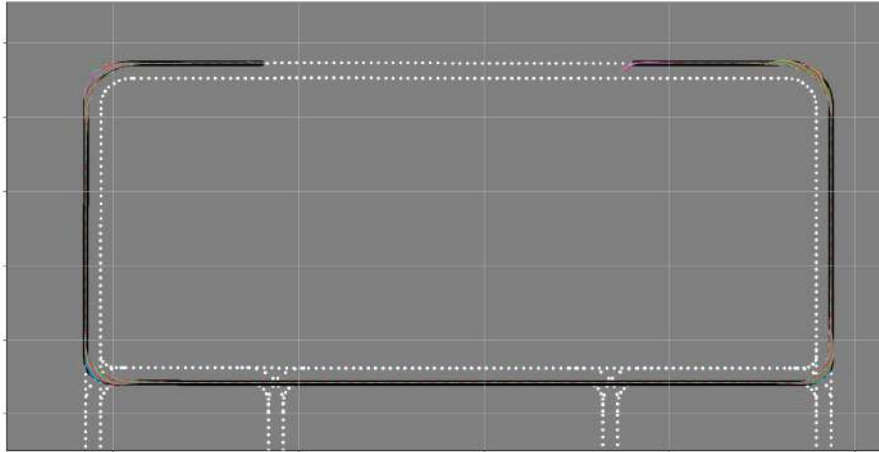
(a) Trial 1



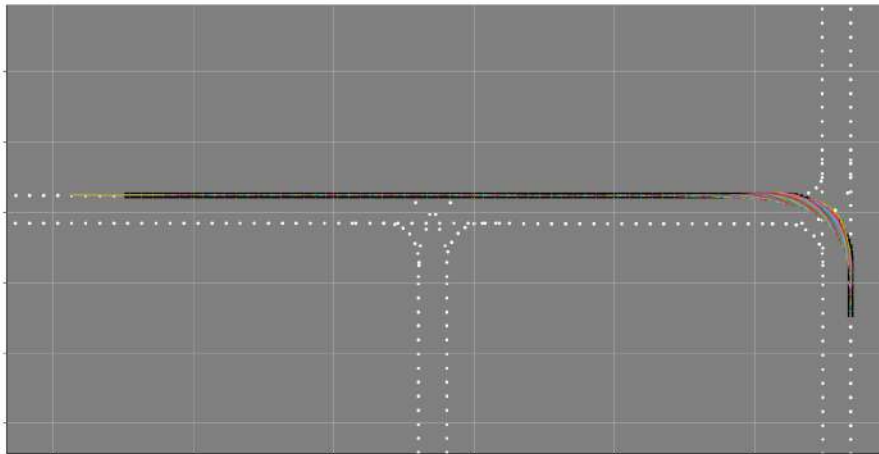
(b) Trial 2

Figure 3.5: Coordinates of the vehicle on the defined courses

Figure 3.6 shows all the possible optimized path computed during the travelling process at every single time step. Each colored line is a different path.



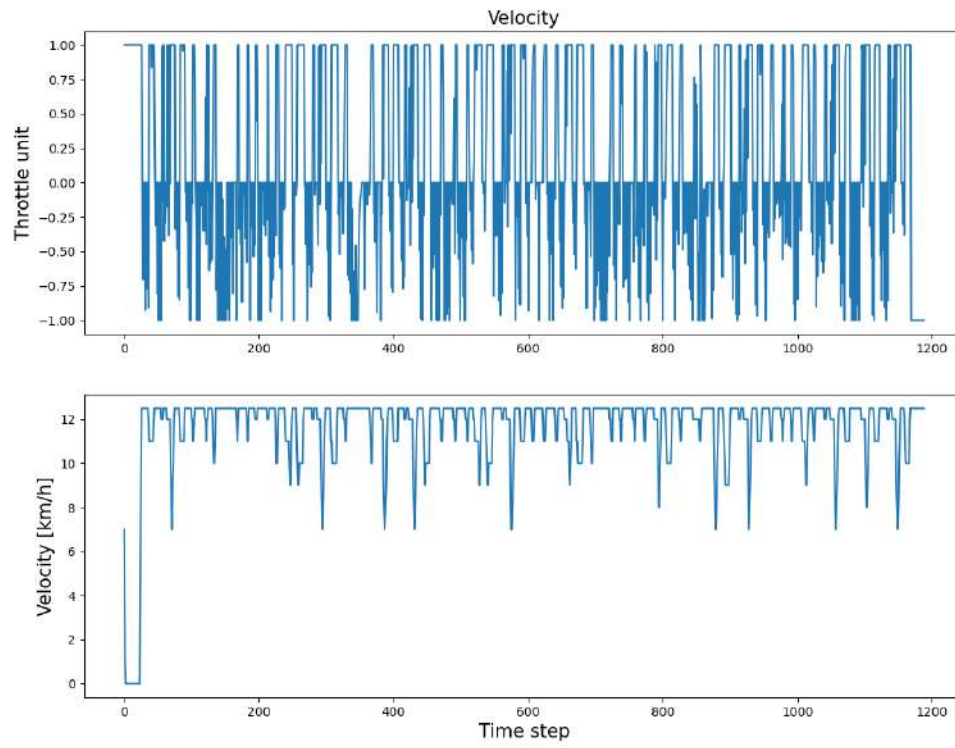
(a) Trial 1



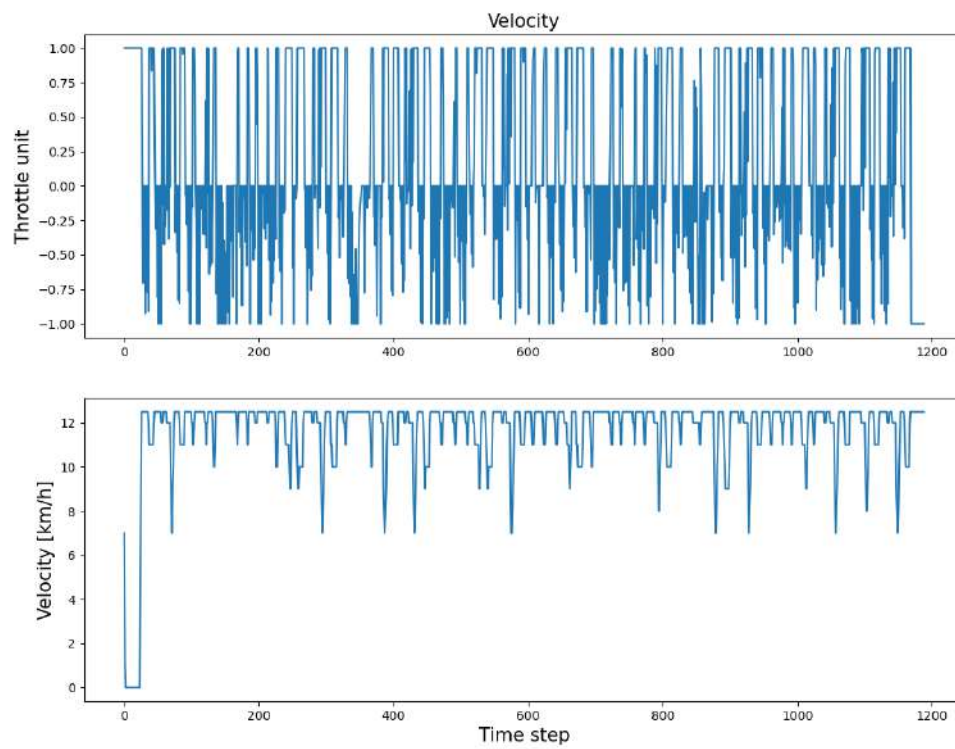
(b) Trial 2

Figure 3.6: Optimized path at every single time step

The throttle and velocity of the vehicle are shown in figure 3.7, while the steer value and steer angle are shown in figure 3.8.

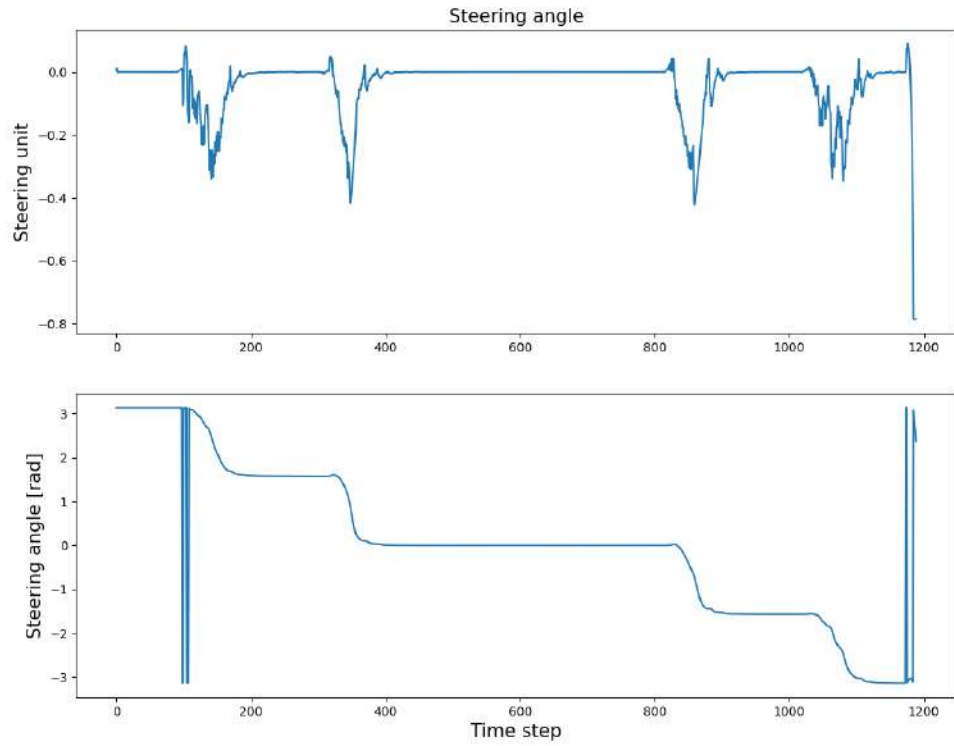


(a) Trial 1

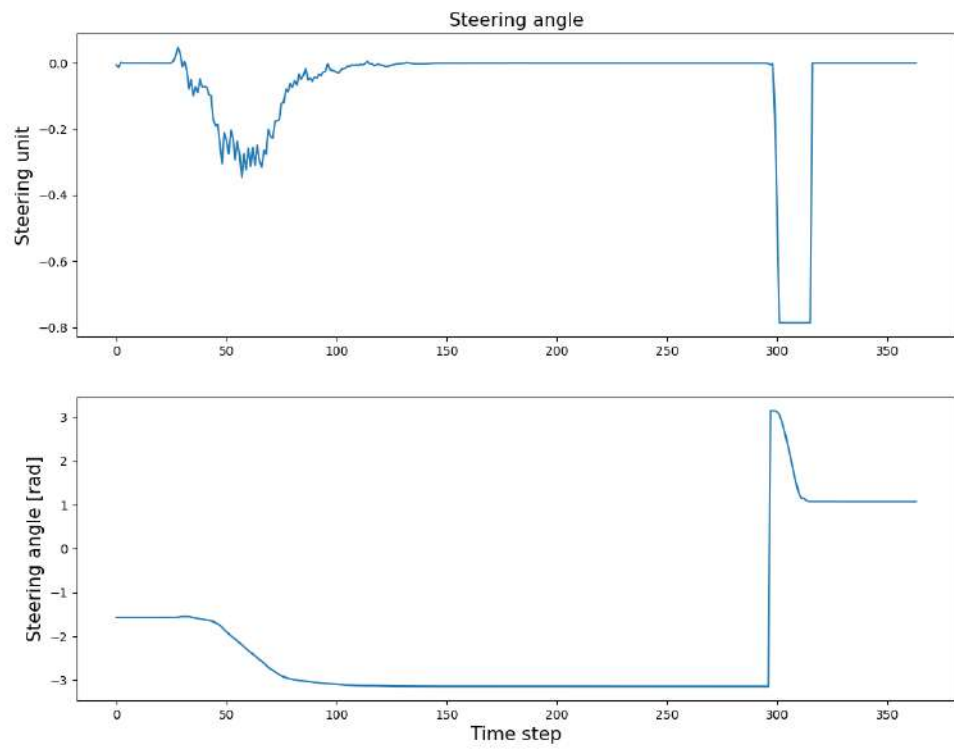


(b) Trial 2

Figure 3.7: Throttle and velocity of the vehicle



(a) Trial 1



(b) Trial 2

Figure 3.8: Steering value and steering angle of the vehicle

Chapter 4

GNSS for intersection detection

As mentioned in chapter 3, the E2E module cannot handle the entire travelling autonomously. It encounters difficulties at intersections where exist multiple options. An ideal solution is to let the E2E module drive on regular path and the MPC module will take over when the vehicle is approaching the intersection. This requires location awareness of the vehicle, which is supported by CARLA using its Global Navigation Satellite System module. The implementation structure is expressed in figure 4.1.

Since the main aim of this project does not deal with GNSS heavily, the noise impact is ignored and all the noise parameters are set to zero. The specification for the GNSS module is set as in table 4.1

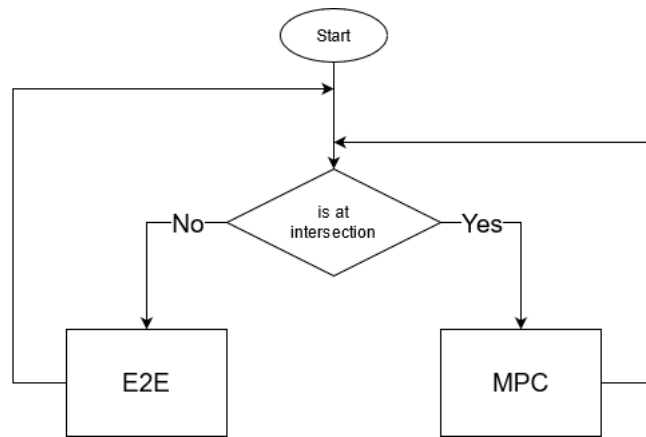


Figure 4.1: How GNSS module is used for switching driving mode

Table 4.1: GNSS parameters

<code>noise_alt_bias</code>	0
<code>noise_alt_stddev</code>	0
<code>noise_lat_bias</code>	0
<code>noise_lat_stddev</code>	0
<code>noise_lon_bias</code>	0
<code>noise_lon_stddev</code>	0
<code>noise_seed</code>	0
<code>sensor_tick</code>	0.05

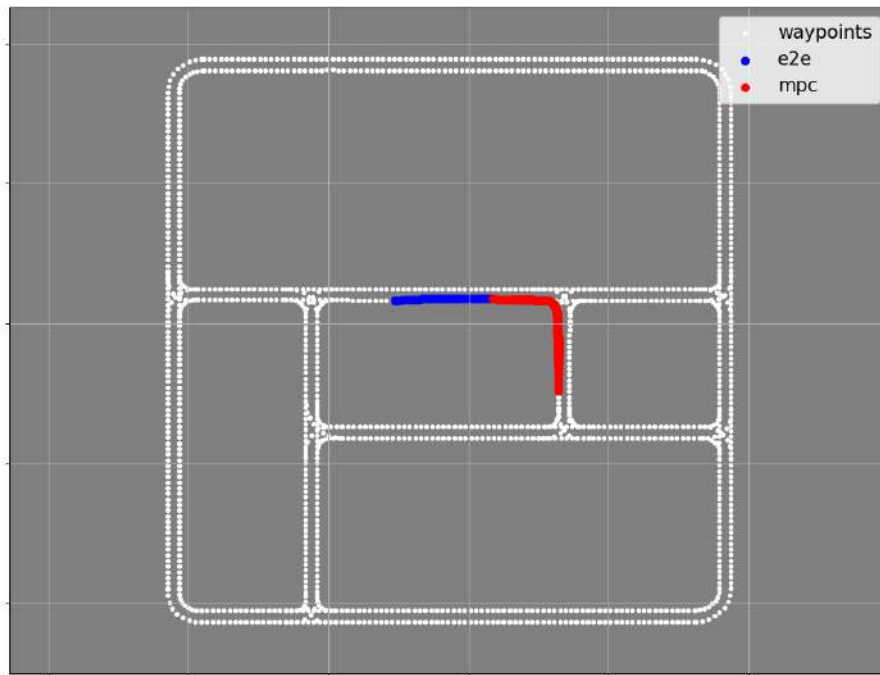
The GNSS module returns 3 geographic coordinate system values: longitude, latitude and altitude and they have the format shown in table 4.2

Table 4.2: GNSS returned values

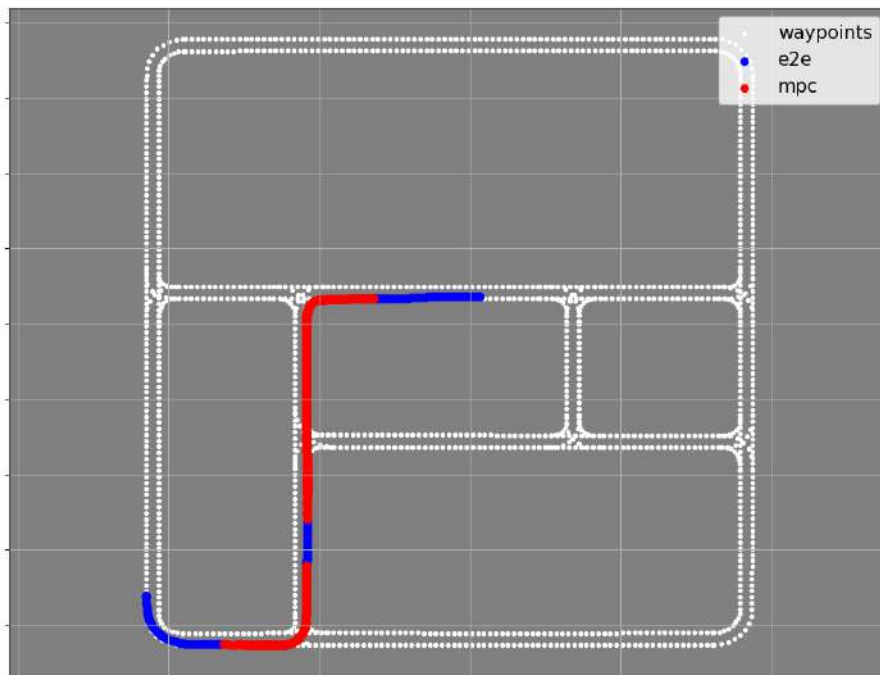
Latitude	Longitude	Altitude
-0.002088251	0.001222914	3.246121645
-0.002045393	0.001223012	3.017500877
-0.002038851	0.001223041	3.020994186
-0.002013239	0.001222989	3.017291784
-0.001990859	0.001222949	3.017678738
-0.00188926	0.001222967	3.018906593

The intersection is defined manually by taking the centre coordinates and set a radius, which returns a circle region. The geographic coordinate of the car is being checked repeatedly to see if it falls into the region; in case it does, the vehicle controller will take the MPC module output. When the vehicle is out of the region, the vehicle controller will take the E2E module output.

The result is that the vehicle is able perform smooth transitions between the two modes. Figure 4.2 shows the result of two separate testings with the red path indicates when the controller is using the outputs from MPC module, the blue path is when the outputs from E2E module is being used.



(a) Trial 1



(b) Trial 2

Figure 4.2: Control modules transition testing

Chapter 5

LiDAR for objects avoidance

This chapter aims to explain how Lidar was used for object avoidance and how it cooperates with the autonomous system. In particular, it includes data retrieval and interpretation, and application.

5.1 Data retrieval and interpretation

Lidar is known as a type of sensor which emits pulses of light and use the measured time of flight of the round trip when it bounces off an object to estimate the distance to such object. This can be easily done when the time of flight and the speed the pulses are known.

The Lidar equipped for the vehicle in Carla shares similar specifications with Velodyne Lidar, which has the capability to sample around a million data points per second, according to A. Kell’s blog[20], its specifications are briefly summarised in table 5.1. For this project, a slightly lower version of the Velodyne Lidar is created with its full specification listed in table 5.2.

Table 5.1: Velodyne specifications

Velodyne specifications	
channels	64
range	50
points_per_second	1000000
fov	26.8

Table 5.2: Chosen LiDAR specifications

LiDAR specifications	
channels	64
range	50
points_per_second	100000
rotation_frequency	10
upper_fov	10
lower_fov	-30
atmosphere_attenuation_rate	0.004
dropoff_general_rate	0.45
dropoff_intensity_limit	0.8
dropoff_zero_intensity	0.4
sensor_tick	0.2
noise_stddev	0.0

Users have the freedom to adjust the above parameters the way they prefer. The figure 5.1 below are the comparisons of the resulted point cloud with different Lidar specs, which helps to understand the effects of those parameters. This was done by running several test with differences in Lidar channels, points per second, the drop-off values and noise.

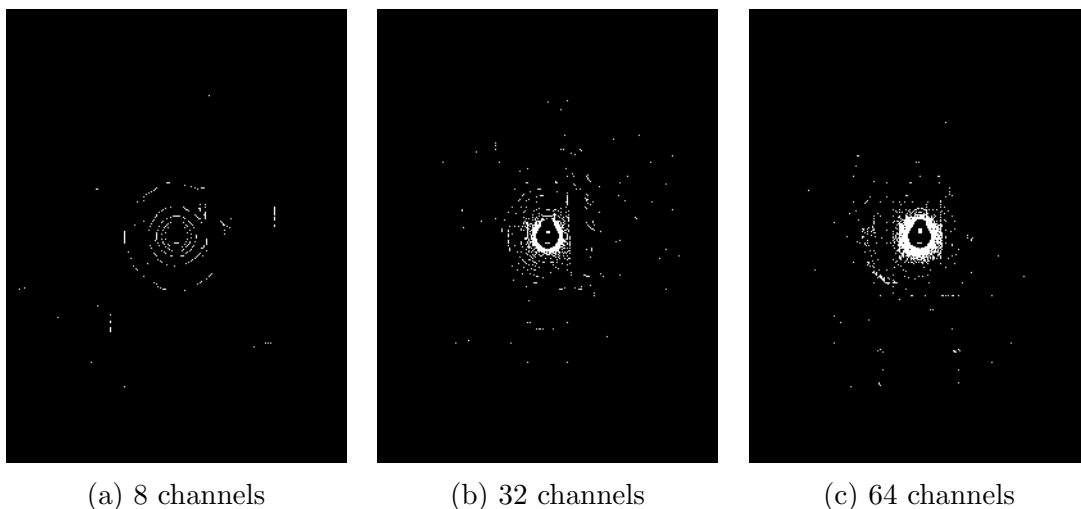


Figure 5.1: Point clouds generated by Lidar with different channels

As can be seen, the greater the amount of channels, the greater the amount of points captured. As explained by A. Kell in his blog, an increase in the number of channels will lead to a rise in cost, which is not very effective. Therefore, the chosen number of channels will be 64, as same as Velodyne's.

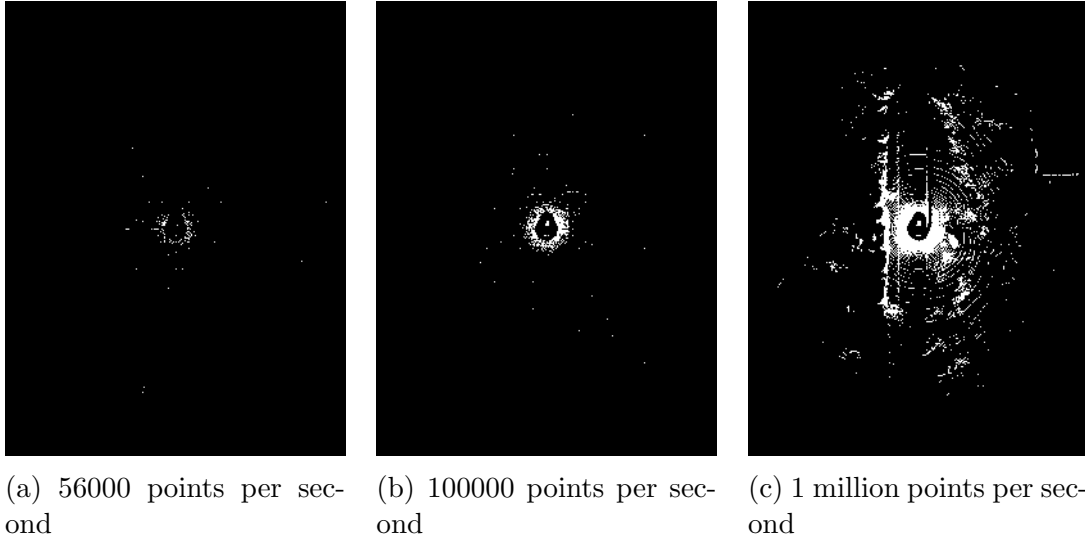
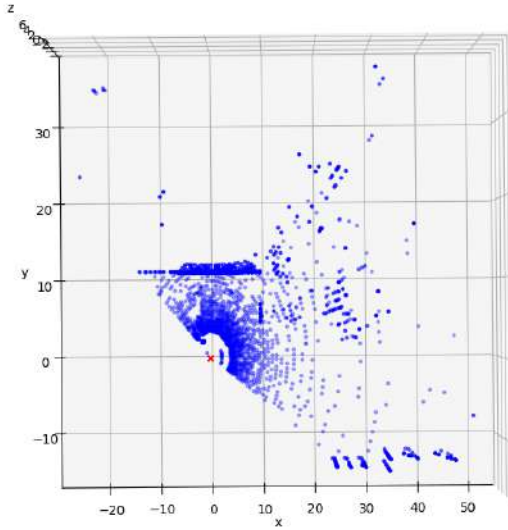


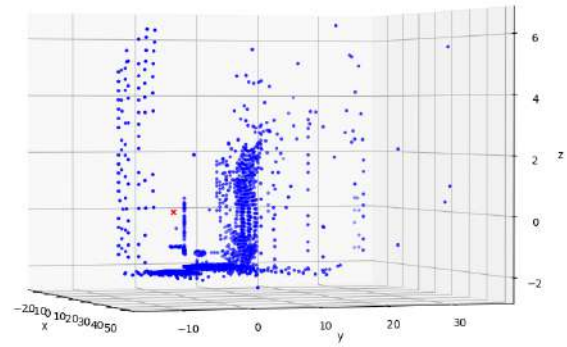
Figure 5.2: Point clouds generated by Lidar with different amount of points per second

As for the parameter "points per second", the demonstration in figure 5.2 shows that the option with 1 million points per second gives a better visual and more information than the other two: nevertheless, it comes with its own limitation. Unlike the number of channels, the choosing of "points per second" is limited because of its huge impact on the overall processing performance. In other words, higher points per seconds leads to lower processing performance, which also affect the other computation processes in the autonomous system. The best number that satisfies the need and the performance limitation is 100000 points per second.

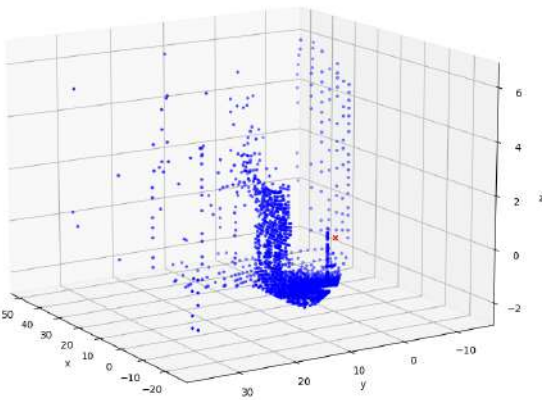
The noise of the Lidar used for this project is assumed to be removed as the original purpose was to use good data to detect and avoid obstacles. The different point of views of the retrieved Lidar data in 3D representation are shown in figure 5.3 with the red cross represents the location of the Lidar sensor placed on the vehicle and the blue points are the data points.



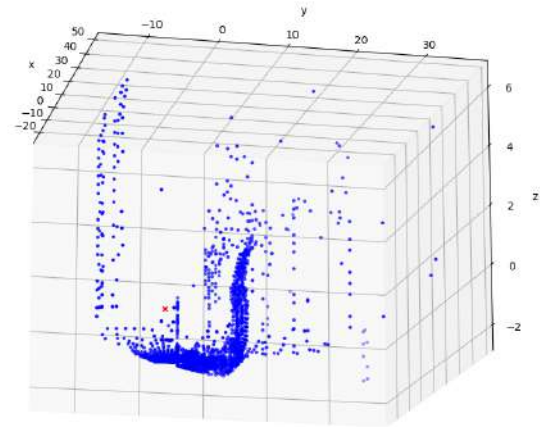
(a) POV 1



(b) POV 2



(c) POV 3

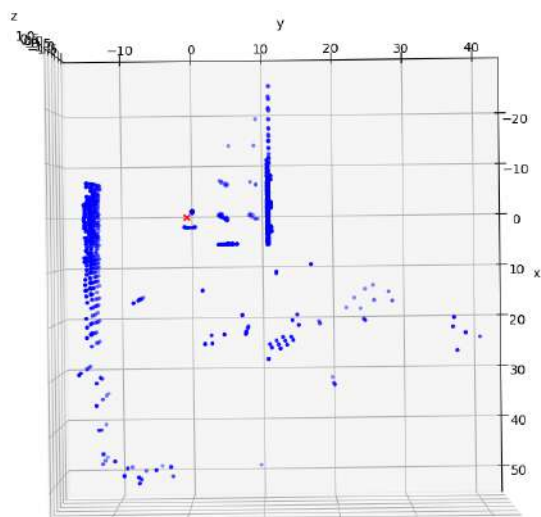


(d) POV 4

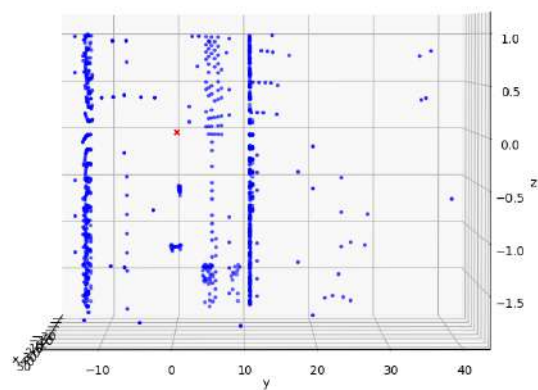
Figure 5.3: Lidar data in 3D representation

5.2 Application

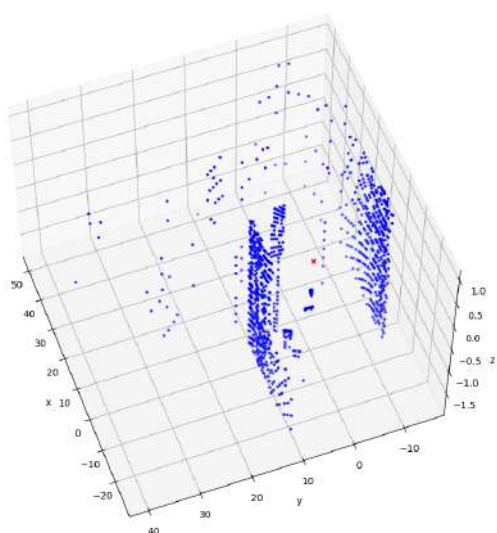
For the main purpose of using lidar to avoid obstacles, the points which represent the ground surface must be removed and the points above which are kept. The points are selected based on their z values whether they are above a certain threshold or not. Figure 5.4 represents the remaining after ground removal.



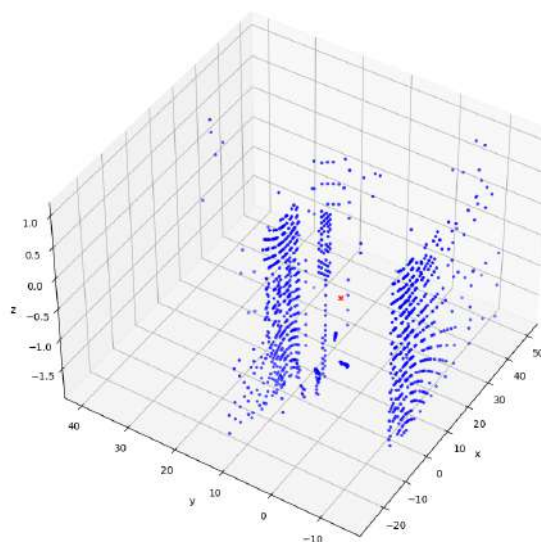
(a) POV 1



(b) POV 2



(c) POV 3



(d) POV 4

Figure 5.4: Lidar data in 3D representation after ground removal

Since a 2D perception of the received is enough for the task, the next step is removing the z-axis out of the picture and the results are shown in figure 5.5. The green box represents the vehicle's bounding box, the red cross marks the location of the Lidar sensor and the blue points are the Lidar points.

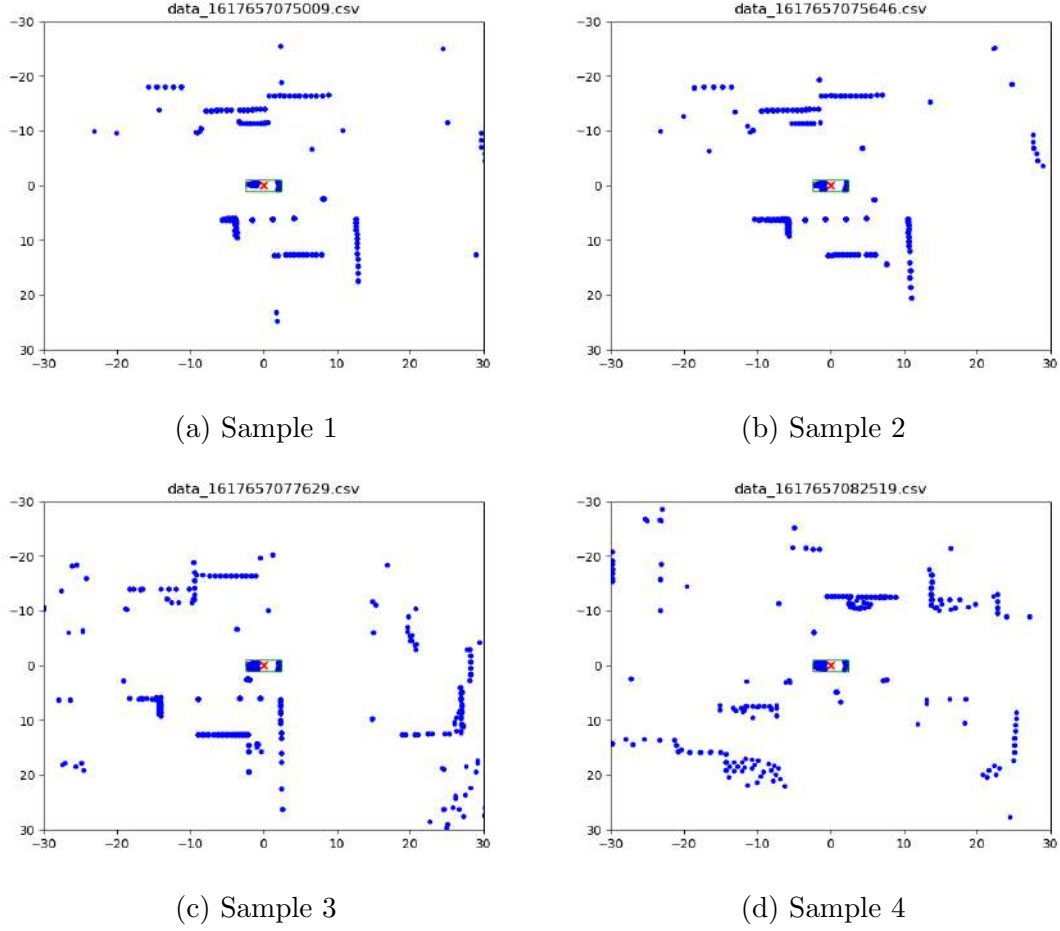


Figure 5.5: Lidar data in 2D representation after ground removal

The next step is to define regions of interest which will trigger a brake attempt if there are points that fall into either one of these region or both.

The first region's dimension is dynamic, its width is kept at a constant, which is slightly larger than the width of the vehicle, the length of this region is determined by utilizing the predicted future path retrieved by using model predictive control algorithm, as explained in chapter 3.

Initially, the future points' coordinates are related to the map's coordinate system. Therefore, they are converted to the ego vehicle's coordinate system using the current location of the vehicle to define the local safety region. A pair of imagine points are then created from each predicted way-point, the distance between the two marks the width of the safety region and the result is as in stage 1 in figure 5.6. Using the yaw information provided, the line is rotated with respect to the center point as what is shown in stage 2. The final stage is to connect the two endpoints of each pair together to form the safety region, as shown in stage 3.

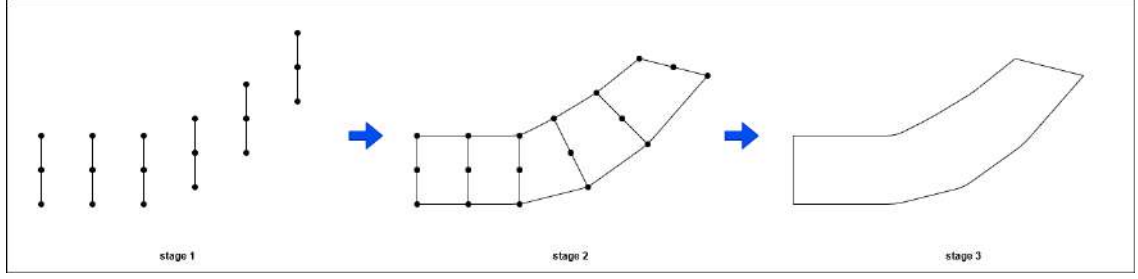


Figure 5.6: Stages of forming safety region

The second region's dimension is fixed and established at the front of the vehicle as its purpose is to make a full brake attempt if any point gets too close to its front. The width of the region is the same as the first region's width, its length is defined short to avoid capturing unwanted points that are away from the main travelling path. Its purpose is to prevent the vehicle from moving forward in case the 1st region is inactive. Figure 5.7 shows what the dynamic region looks like when it is active in inactive, where the yellow rectangle is the fixed region and the green rectangle is the dynamic region.

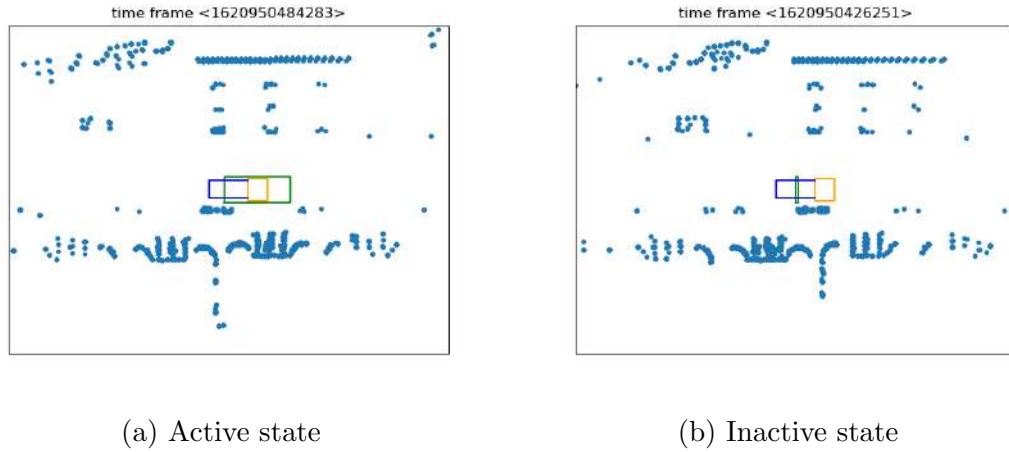


Figure 5.7: Visualization of the dynamic region in its active and inactive state

The explanation of using both regions is using only the dynamic region will cause the vehicle to move forward as the region is collapsed, or inactive. On the other side, using only the fixed region will make it difficult to stop the vehicle while travelling in high speed as the length of the region is too small and increasing the length will cause unnecessary braking attempt because of the capture of the unwanted points as mentioned above.

A test scenario is set up at a corner as shown in figure 5.8, where the ego vehicle is approaching the corner with an obstacle vehicle blocking the way.

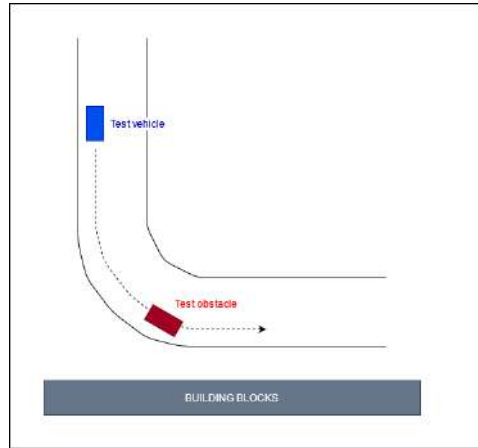


Figure 5.8: Lidar test scenario

The visualization of the result is shown in figure 5.9. The first (a) and second (b) time frames show what it looks like during the run. When approaching the corner, the green region was shaped into a spline like shape, which was also the region of the predicted future path. The third time frame was when it detected the point cloud of the obstacle, painted in purple, and returned the brake value of 1, which caused the vehicle to stop. When the dynamic region became inactive, the fixed region prevented the vehicle to continue moving forward.

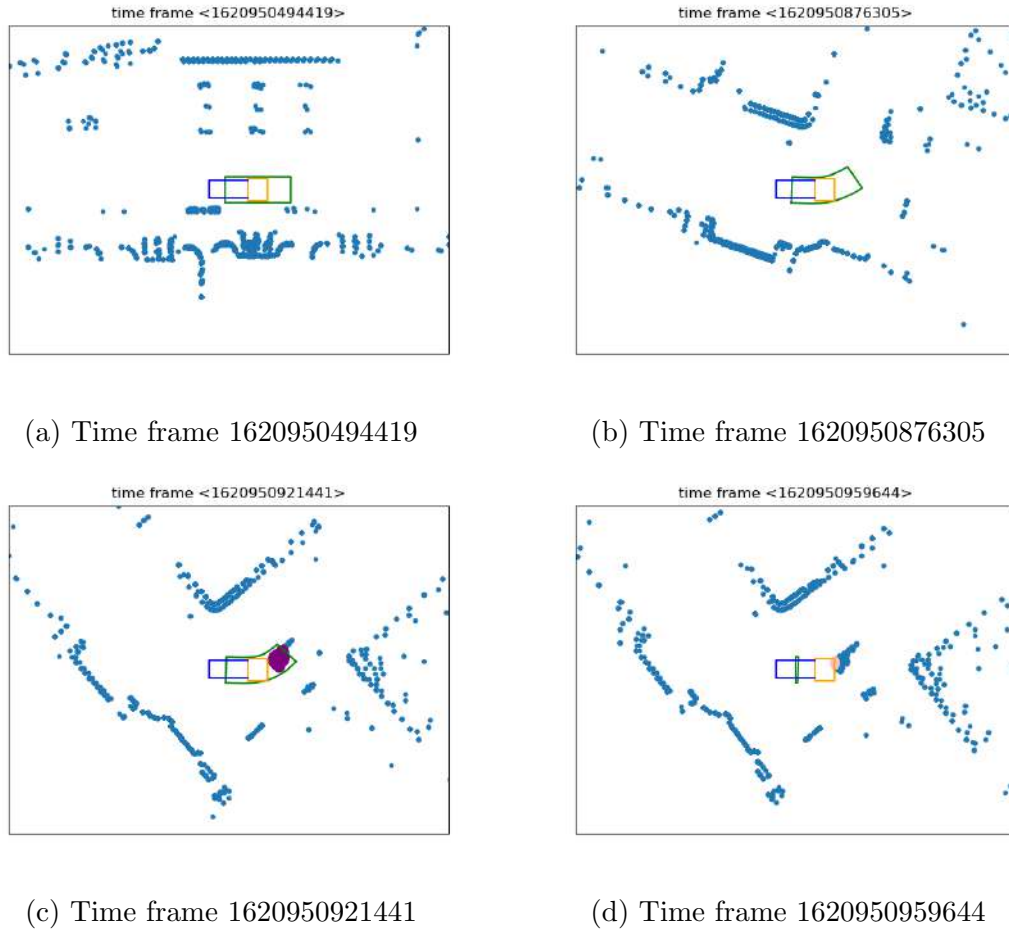


Figure 5.9: Test visualization

In case the obstacle moves out of the way, the pink points in the fixed region will disappear and the ego vehicle will attempt for a move again.

Chapter 6

Global path planning using Dijkstra's algorithm

For the final part, global path planning, the Dijkstra's algorithm is used. The Dijkstra's algorithm, named after the computer scientist Edsger W. Dijkstra, is used to find the shortest path between nodes in a graph. The basic components of Dijkstra's algorithms include the nodes, which represent the way points, the distances between the nodes and the graph which holds all of those components. This algorithm is explained using a example in a video made by the Youtube channel Computer Science[21], the example used a graph to demonstrate the connections and the distances between the nodes and a table to track the path finding progress with the goal is to find the shortest path from node A to C. The graph and the table are shown as in figure 6.1 and table 6.1.

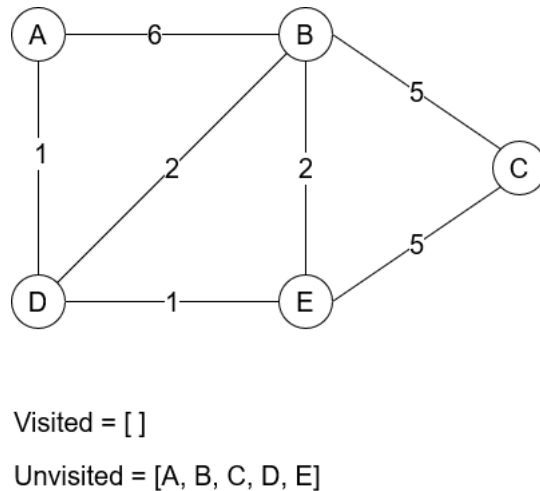
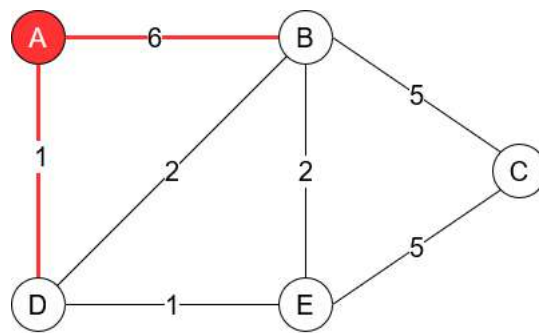


Figure 6.1: Example graph

Node	Shortest distance from A	Previous node
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

Table 6.1: Tracking table

There are 2 lists, which stores the visited nodes and the unvisited nodes. The table includes 3 columns, the name of the nodes, the shortest distance from the starting node and the previous node that connects to it. The progress starts by filling in the table the shortest distance from every node to node A and for node A, it is 0. For the other nodes, they are temporarily left as infinity. The next step is to evaluate node A and the distances to the unvisited neighbouring nodes. Node A has 2 neighbours, which are node B and node D with the distances from node A are 6 and 1 respectively, if the calculated distance is less than a known distance, the shortest distance will then be updated. At this point, the shortest distances of B and D are both less than infinity; therefore, table 6.1 is updated to table 6.2 and node A is added as the previous node of node B and D and it is moved to the visited list and will not be evaluated again.



Visited = [A]

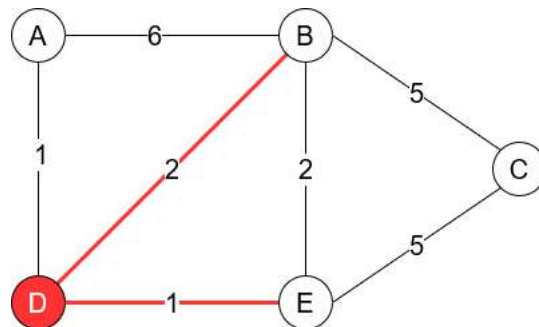
Unvisited = [B, C, D, E]

Figure 6.2: Example graph

Node	Shortest distance from A	Previous node
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

Table 6.2: Tracking table updated at node A

At this point, the next node selected for the evaluation is the one with the shortest distance from node A, which is node D. Node D has two neighbours in the unvisited list, which are node B and E. The distances from node A to these neighbouring nodes are computed again by taking the sum of the distances from node A to D and the distances from node D to its neighbours. The sums of distances for the path A-D-E and A-D-B are 2 and 3 respectively, if the new distance is shorter than the currently existed one, it will be replaced. In this case, the shortest distance from A to B is updated as 3 and the previous node for node B is node D. The new table is updated as in table 6.3 and node D is moved to the visited list.



Visited = [A, D]

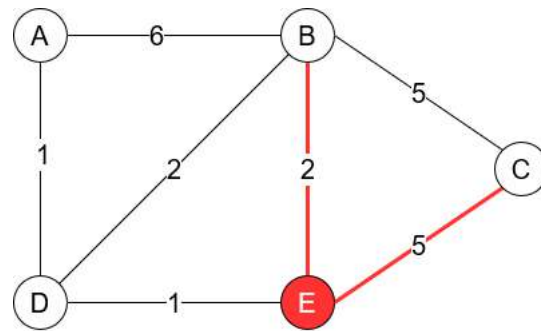
Unvisited = [B, C, E]

Figure 6.3: Example graph

Node	Shortest distance from A	Previous node
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

Table 6.3: Tracking table updated at node D

The process continues with node E as the unvisited node with the shortest distance from A. From node E, there are two unvisited nodes, B and C and the distance for the path A-D-E-B is 4 and for the path A-D-E-C is 7. The results are compared with the current values in the table and only the shortest distance for node C is updated with node E is assigned as the previous node of node C, the details are shown in table 6.4. Node E is then removed from the unvisited list.



Visited = [A, D, E]

Unvisited = [B, C]

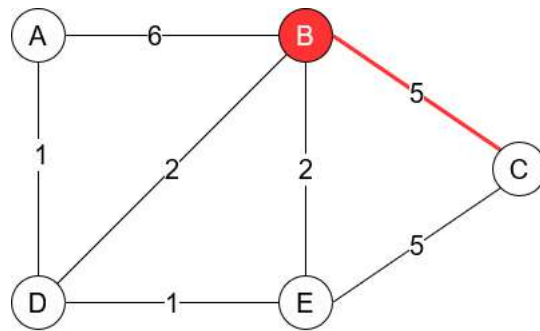
Figure 6.4: Example graph

Node	Shortest distance from A	Previous node
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Table 6.4: Tracking table updated at node E

Then next node in the unvisited list is node B, from here there is only 1 unvisited

neighbour that is node C. The distance of the path A-D-B-C is 8, which is longer comparing to 7 of the path A-D-E-C computed earlier; therefore, nothing is updated and node B is moved to the visited list.



Visited = [A, D, E, B]

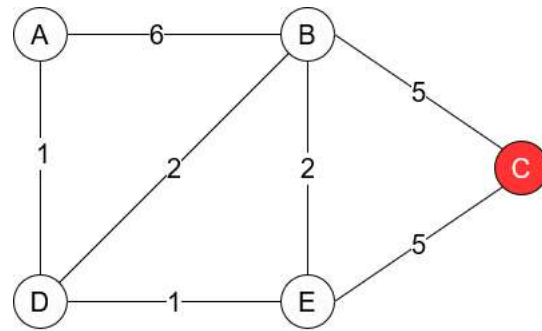
Unvisited = [C]

Figure 6.5: Example graph

Node	Shortest distance from A	Previous node
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Table 6.5: Tracking table updated at node B

The last node C does not have any unvisited neighbour and the only thing to do is to moved it to the visited list and table 6.5 is the final table used for tracking down the shortest path from A to C. By tracking down the sequence of the previous node, starting from node C, the sequence is C-E-D-A; therefore, the shortest path from A to C is A-D-E-C with the global distance of 7.



Visited = [A, D, E, B, C]

Unvisited = []

Figure 6.6: Example graph

To apply this in CARLA, all the road segments of map Town02 were retrieved using a CARLA built-in function and they are stored in a list. Each of the segments is the list of way points with a distance of two metres between each. The visualization of the road segments are shown in figure 6.7.

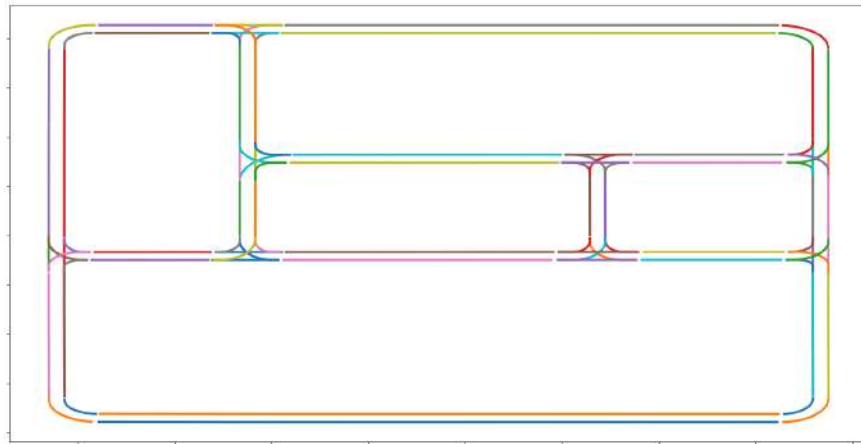


Figure 6.7: Town02 road segments

The testing scenario is two random points are chosen as the starting point and the destination. The constraints for the vehicle include:

- Unable to travel on the wrong side
- Unable to travel in reverse
- Unable to make a U-turn

Normally, the Dijkstra's algorithm can easily find the shortest path from one end to another; however, it is most likely that the path is against the traffic laws. This explains the need for the constraints.

The way points in each segments are in the correct order of the travelling direction. To obey the constraints, the nodes are connected in one direction to prevent the later nodes from connecting to the previous nodes. This is easily imaginable with an example of a straight segment as in figure 6.8. Figure 6.8a describes the scenario of with two nodes belong to the same segment. The allowed direction is from A to B. In the case when A is chosen as the starting point and B is chosen as the destination, the travelling path is set as in figure 6.8b, which is the same direction with the rule. On the other side, if B is chosen as the starting point, A is chosen the destination and B is allowed to connect to the node before it, the situation is figure 6.8c will happens. In this case, node B should search for the next node in line that to connect to and find a path that helps it reach A in the correct direction, as in figure 6.8d.

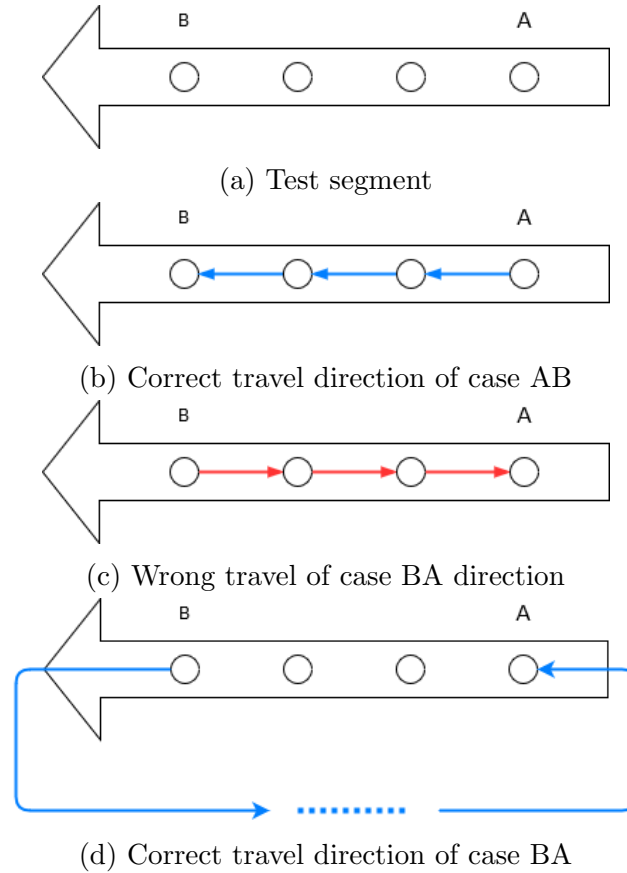
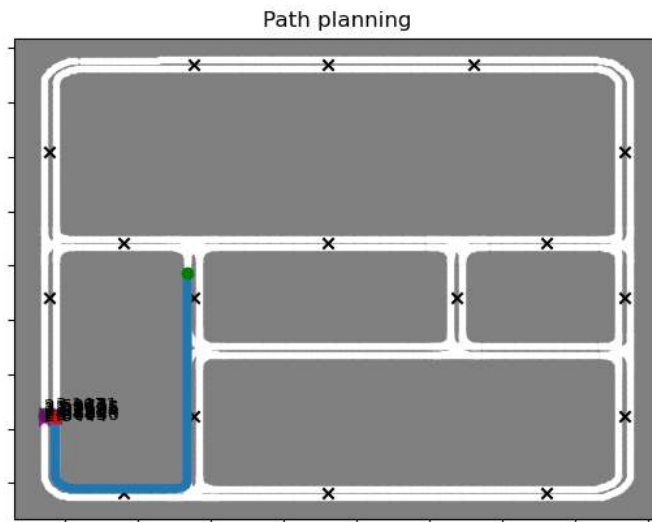
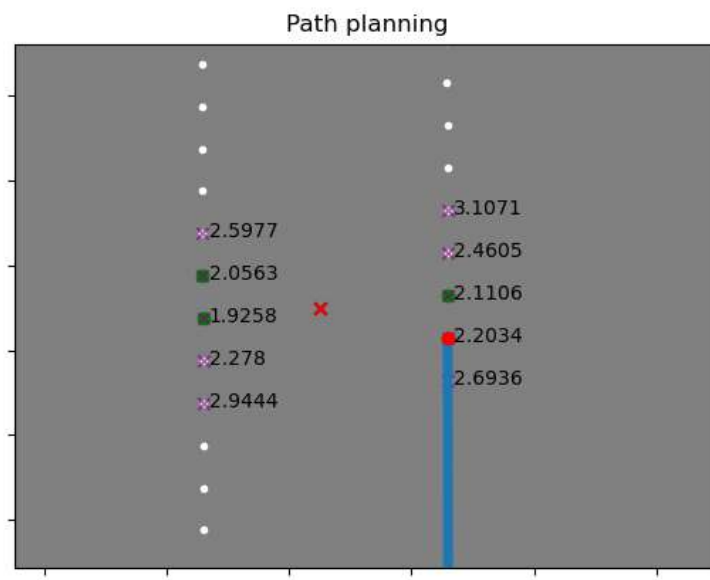


Figure 6.8: Example of direction on a straight segment

To make the application more practical, the destination are chosen as a point with random coordination on the map and this point does not exist in the database. Initially, four points from the database that are closest to the random point are selected, the Dijkstra's algorithm is then applied to find the shortest path to every single option. In the end, the point with the shortest path are selected as the final path. Some of the results are shown in figure 6.9 and 6.10.

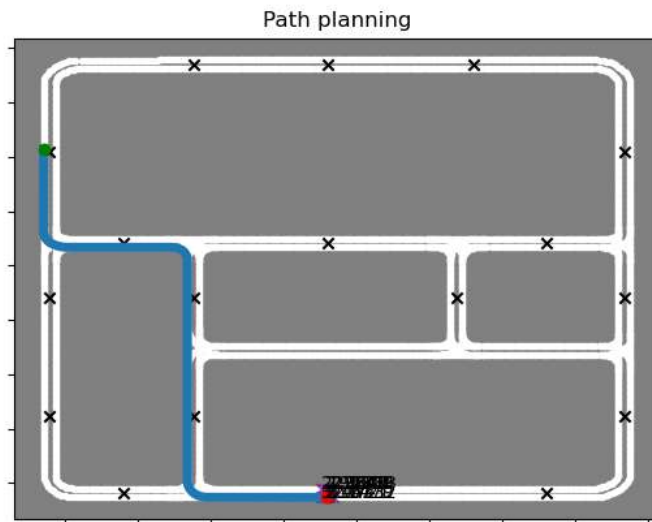


(a) Global path

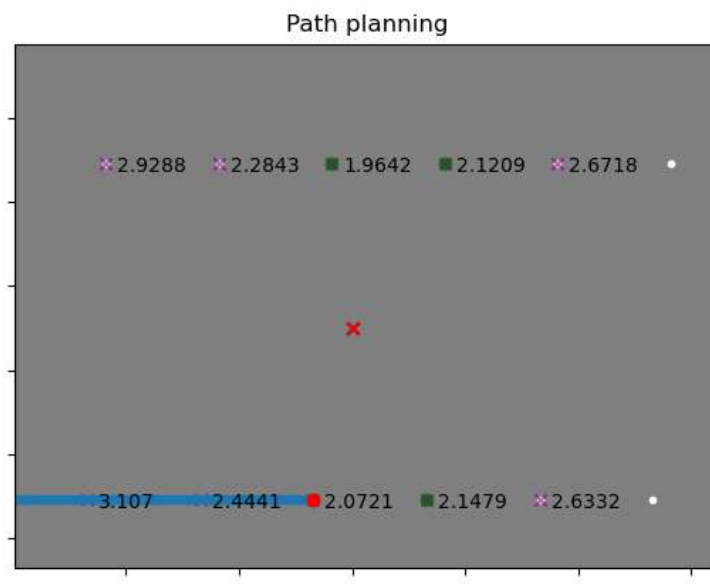


(b) Destination node zoomed in

Figure 6.9: Result of first trial of global path finding using Dijkstra's algorithm



(a) Global path



(b) Destination node zoomed in

Figure 6.10: Result of second trial of global path finding using Dijkstra's algorithm

The green dot in sub-figure "a" of each figure represent the starting point. In sub-figure b, there are a group of points in the database selected to calculate the distance to the red "x", the distances are shown right next to each point. Four of them, which are the green dots with the purple x, are selected for the final path finding and the result is the red dot that represents the destination node with the blue path drawn from the green dot to the red dot. In both cases, the blue paths satisfied the constraints listed before and they are also the shortest path as can be seen in the map.

Chapter 7

Conclusion

In conclusion, all of the named applications were implemented successfully and the results were acceptable. Nevertheless, due to the hardware constraint, it is not possible to implement the LiDAR module together with the rest.

Chapter 8

Encountered problems

There were a few problems during the implementation of this project and they can be listed as:

- Data dimension
- Inconsistency between versions of libraries

8.1 Data dimension

During the process of implementing the end-to-end learning for self-driving cars, the size of the collected images were initially set at 600x400 in width and height. With this setting, the size of each data set with around 10000 samples could be up to 10 GB, this caused difficulties in transportation of the data between different platforms and systems. Moreover, the training time of 6 epochs with these images were up to approximately 12 hours. This was difficult for model testing and tuning; therefore, the size of the images was reduced to 320x160 in width and height. The result is as mentioned in the chapter above.

8.2 Versions Inconsistency

Throughout the time when the project was in the process, the code was changed a few times to adapt to the Tensorflow API. At first, due to hardware restriction, the code was written in a context that followed Tensorflow 1. At this point, the GPU was not utilized and therefore the training was slow. When the option to use the graphics card with Tensorflow 2 is in use, every part that uses this library was rewritten; however, the time consumed for the changes is negligible. The main problem of the version switching was the installation of the required drivers and dependencies, namely the compatible driver for the GPU, Cuda Toolkit, and cuDNN library. The version of Cuda Toolkit and cuDNN must also be compatible with each other and must meet the requirements of the current system. Moreover, there were errors happened when the drivers were installed and it only happened only to such specific version, the recommended solution was to install the lower version, which led to the uninstalling of the driver and reinstall from the beginning, part of the reason is that the instruction for the installation is complicated as it is unclear about

the proper versions to use. These occur in both Windows 10 and Ubuntu and one recommended solution is by installing Anaconda and create a conda environment with Tensorflow-GPU, all the needed drivers and dependencies will also be installed at once. The command line for this is as follows:

```
conda create --name tfgpu37 tensorflow-gpu python=3.7 [1]
```

Chapter 9

Future improvements

This project might be taken to another level by implementing most of the applications in real life. This does not necessarily mean an actual car but could be a radio-controlled car. The challenge will be to handle the real life constraints, disturbance and perform tuning on the controller. Moreover, the data gathered will probably be noisier, which will requires more data processing knowledge before the data can be used.

In case the applications are constantly used, there will be a need to develop an automatic pipeline for data collection, data storage, data processing and data analyzing; especially the end-to-end learning application that requires new data whenever changes of the structure are made.

Bibliography

- [1] Chris Neiger. Jun 25, 2019. URL: <https://eu.usatoday.com/story/money/2019/06/25/autonomous-vehicles-4-hard-to-believe-driverless-car-facts/39586719/>.
- [2] Darrell Etherington. *BMW and Daimler partner on autonomous driving, first results of team-up in market by 2024*. July 4, 2019. URL: <https://techcrunch.com/2019/07/04/bmw-and-daimler-partner-on-autonomous-driving-first-results-of-team-up-in-market-by-2024/>.
- [3] Mariusz Bojarski. “End to End Learning for Self-Driving Cars”. In: (Apr 25, 2016).
- [4] Netflix. *About*. URL: <https://research.netflix.com/research-area/machine-learning>.
- [5] Kurgen L.A. Cios K.J. Swiniarski R.W. *Unsupervised Learning: Association Rules*. 2007.
- [6] Anne Bonner. *What is Deep Learning and How Does it Work?* Sep 7, 2019. URL: <https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-f7d02aa9d477>.
- [7] Michael Nielsen. *Chapter 1, Using neural nets to recognize handwritten digits*. Dec 2019. URL: <http://neuralnetworksanddeeplearning.com/chap1.html>.
- [8] Tran The Anh. *[ML – 15] From Sigmoid function to Rectifier Linear function*. Apr 26, 2017. URL: <https://labs.septeni-technology.jp/machine-learning/ml-15-from-sigmoid-function-to-rectifier-linear-function/>.
- [9] 3Blue1Brown. *But what is a Neural Network? — Deep learning, chapter 1*. Youtube. URL: <https://www.youtube.com/watch?v=aircAruvnKk&t=762s>.
- [10] Soner Yıldırım. *Activation Functions in Neural Networks*. Jun 15, 2020. URL: <https://towardsdatascience.com/activation-functions-in-neural-networks-eb8c1ba565f8>.
- [11] Sagar Sharma. *Activation Functions in Neural Networks*. Sep 6, 2017. URL: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>.
- [12] Avinash Sharma V. *Understanding Activation Functions in Neural Networks*. Mar 30, 2017. URL: <https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0>.
- [13] Franck Dernoncourt. *What does “end to end” mean in deep learning methods?* Jul 2016. URL: <https://stats.stackexchange.com/questions/224118/what-does-end-to-end-mean-in-deep-learning-methods>.

- [14] R.Collobert J.Weston L.Bottou M.Karlen K.Kavukcuoglu P.Kuksa. “Natural Language Processing (Almost) from Scratch”. In: *Journal of Machine Learning Research* (Aug 12, (2011): 2493-2537).
- [15] Alexandru Constantin Serban, Erik Poll, and Joost Visser. “A Standard Driven Software Architecture for Fully Autonomous Vehicles”. In: *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)* (2018).
- [16] A. Bindal. *Normalization Techniques in Deep Neural Networks*. Feb 10, 2019. URL: <https://medium.com/techspace-usict/normalization-techniques-in-deep-neural-networks-9121bf100d8>.
- [17] M. Stewart. *Simple Introduction to Convolutional Neural Networks*. Feb 27, 2019. URL: <https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac>.
- [18] AtsushiSakai. *model_predictive_speed_and_steer_control*. Jul 16, 2018. URL: github.com/AtsushiSakai/PythonRobotics/blob/master/PathTracking/model_predictive_speed_and_steer_control/model_predictive_speed_and_steer_control.py.
- [19] *cvxpy*. URL: www.cvxpy.org.
- [20] Adam Kell. *Engineer Explains: Lidar*. Apr 5, 2016. URL: blog.cometlabs.io/engineer-explains-lidar-748f9ba0c404.
- [21] Computer Science. *Graph Data Structure 4. Dijkstra’s Shortest Path Algorithm*. URL: <https://www.youtube.com/watch?v=pVfj6mxhdMw>.