NUST COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING

Auto-Parking of Vehicle by Using Supervised Learning Approach

A PROJECT REPORT

DE-40 (DEE)

Submitted by

NC Mustaqeem Ashraf
40-EE (246700)

PC Tooba Anwar
40-EE (280794)

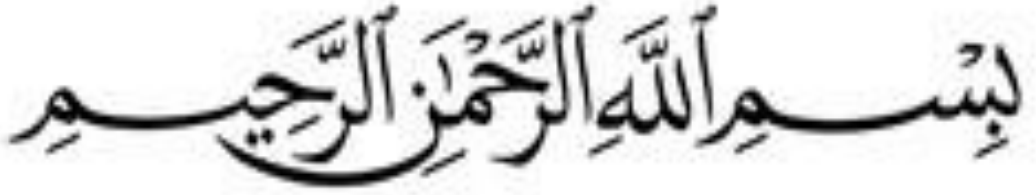
PC Shamsa Kanwal
40-EE (280790)

GC Mazahir Hussain
40-EE (280865)

PROJECT SUPERVISOR

Dr. Muwahida Liaquat

BACHELORS
IN
ELECTRICAL ENGINEERING
YEAR 2022


بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

Auto-Parking of Vehicle by Using Supervised Learning Approach

A Thesis Presented To

In partial fulfillment of the requirement for the degree of

B.S. Electrical Engineering

By

NC Mustaqeem Ashraf
40-EE (246700)

PC Tooba Anwar
40-EE (280794)

PC Shamsa Kanwal
40-EE (280790)

GC Mazahir Hussain
40-EE (280865)

PROJECT SUPERVISOR

Dr. Muwahida Liaquat

DEDICATION

This project is dedicated to our parents, their efforts and all the prayers were with us during the project. It is also dedicated to our teachers and all the technical staff who really helped us for the completion of this project.

CERTIFICATE OF APPROVAL

It is to certify that the project "**AUTO-PARKING OF VEHICLE BY USING SUPERVISED LEARNING**" was done by **NC Mustaqeem Ashraf, PC Tooba Anwar, PC Shamsa Kanwal, GC Mazahir Hussain** under the supervision of **Dr. Muwahida Liaquat.**

This project is submitted to **Department of Electrical Engineering**, College of Electrical and Mechanical Engineering (Peshawar Road Rawalpindi), National University of Sciences and Technology, Pakistan in partial fulfilment of requirements for the degree of Bachelors of Engineering in Electrical engineering.

Students:

1- Mustaqeem Ashraf

NUST ID: _____ Signature: _____

2- Tooba Anwar

NUST ID: _____ Signature: _____

3- Shamsa Kanwal

NUST ID: _____ Signature: _____

4- Mazahir Hussain

NUST ID: _____ Signature: _____

APPROVED BY:

Project Supervisor: _____ Date: _____

Dr. Muwahida Liaquat

DECLARATION

We hereby declare that no portion of work referred to in this Project Thesis has been submitted in support of an application for another degree or qualification of this of any other university or other institute of learning. If any act of plagiarism found, we are fully responsible for every disciplinary action taken against us depending upon the seriousness of the proven offence, even the cancellation of our degree.

1- Mustaqeem Ashraf

NUST ID: _____ Signature: _____

2- Tooba Anwar

NUST ID: _____ Signature: _____

3- Shamsa Kanwal

NUST ID: _____ Signature: _____

4- Mazahir Hussain

NUST ID: _____ Signature: _____

COPYRIGHT STATEMENT

ACKNOWLEDGEMENT

This project has come to an end by great effort of many people, first of all we want to thank Allah for helping us knowledge, strength, determination to complete this project.

We want to thank our supervisor Dr. Muwahida Liaquat for giving us helpful advises, time, knowledge and for guiding us in every difficult situation.

We can't thank enough our department, **Department of Electrical Engineering**, for nurturing us over the years and helping us excel in our field. We are really appreciative of all the faculty members and staff, who have played their roles impeccably in ensuring that we are provided with the best facilities and treatment.

Finally, we want to express our gratitude for our parents without whose prayers and encouragement we would not have been able to complete this project.

ABSTRACT

When parking a vehicle, it's critical to make sure the vehicle maintains a consistent approach to the parking spot, maintains a good heading angle, and avoids excessive line pressure losses. A supervised learning-based automatic parking model is proposed. A parking kinematics model will be created to calculate the various states of its movement. The car's autonomous parking will be attained through training, and a thorough examination of the many stages and scenarios in the parking process is provided. Multi-objective optimization and comfort, which includes safety, efficiency of car parking, and final parking performance must be considered in automatic parking motion design.

The majority of present research depends on parking data from skilled drivers or human previous information. This project provides a model-based Neural Network approach that iteratively execute the data generation, data assessment, and training network to learn the parking policy of the data. We can mainly eliminate human experience and learn parking strategies autonomously and quickly using this technology. A DNN-based end-to-end parking algorithm is suggested for autonomous parking. The first few layers of DNN provide general parking trajectory planning knowledge for all types of vehicles, however the latter few layers of DNN can be quickly modified to adapt to different types of vehicles. The model may learn and accumulate the experience from several parking attempts and then demand ideal steering wheel angle and speed of vehicle at various parking spaces. Errors caused by path tracking can be prevented by using this end-to-end auto parking approach. Finally, a real-world vehicle test shows that the suggested strategy can achieve a better parking attitude than other methods.

We were able to implement our controller over a given parking area and optimize it such that it is be able to generate empty spaces in parking area that is followed by the system with maximum accuracy over CARLA Simulator. The programming is done in python using any available python IDE and was run parallel to the CARLA interface. The results over multiple were compared to show the efficiency of the controller.

Table of Contents

Table of Figures

CHAPTER 1

1. INTRODUCTION

The installation of automatic parking benefits drivers in terms of both safety and comfort. To park a vehicle securely in a place in a limited amount of time, the vehicle's automatic parking system must complete a variety of complex tasks. These tasks include detecting the surroundings, steering, accelerating, braking, and shifting gears while the vehicle is in motion. Numerous studies on automatic parking have been conducted; nonetheless, accomplishing automatic parking of a car inside a small space is a challenge that must be conquered before advanced driver-assistance systems or autonomous driving can be implemented. Numerous studies on automatic parking have been conducted.

To address this issue, a number of methodologies have been developed, with the trajectory planning method being one of the most widely used. It considers a valid vehicle trajectory from the given initial location and orientation to the stated final position and orientation to be an equivalent construction of the parking control issue. This is accomplished by moving from the beginning and final positions and orientations given. Noticing that the specific parking control approach will define a trajectory, it is clear that if valid trajectories are planned, steering actions can be decided subject to some action space limits. (Serna, 2014) This is due to the fact that proper trajectories must be planned before valid steering operations can be determined. In most circumstances, these methods can be classified into two categories:

- Direct trajectory planning
- Indirect trajectory planning

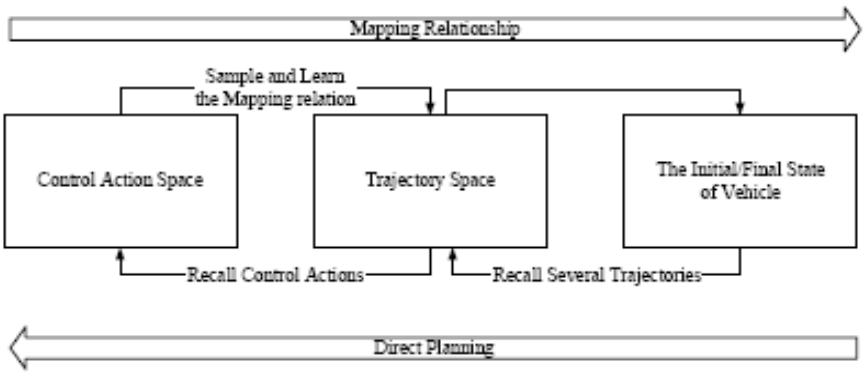


Figure 1: Trajectory Planning

The direct trajectory planning solution framework for autonomous parking.

The goal of this project is to develop a fundamental prototype of an auto-parking vehicle. We trained a Convnet (CNN) to map the raw pixels extracted from the images captured by a front-facing camera to the steering commands. This allowed us to steer the vehicle using only the images captured by the camera. With only a limited amount of training data, the system is able to learn how to park the vehicle.

The CARLA simulator was used after the execution of this project to generate dataset sets in an artificial environment. This environment was created using CARLA. The CARLA simulator is used to run experiments on various algorithms related to the autonomous cars. CARLA delivers open digital assets that were built specifically for this purpose and are free to use. CARLA provides open digital assets in addition to open-source code and protocols. The simulation platform allows for the flexible definition of sensor suites and environmental variables.

1.1 Background

In recent years, the rapid development of artificial intelligence (AI) has been closing the gap in capability that previously existed between humans and machines. Numerous researchers and enthusiasts of artificial intelligence have been focusing their efforts in this particular field for an extensive amount of time in order to make new developments. The primary motivation for doing so is to give machines the ability to perceive various things in the real world in the same way that humans do. The field of computer vision is one example of this. The Convolutional Neural Network is a type of algorithm that,

as a result of developments in deep learning and computer vision, has been refined and improved to the point where it functions flawlessly.

The architecture of convolutional neural networks (CNN) is comparable to that of the human neural network, which is constructed with synapses (weights) and neurons. From this vantage point, it is possible to acquire advanced skills through the use of the network. This makes use of CNN in conjunction with previously developed architectures to determine in real time whether or not a parking spot is vacant.

Convolutional Neural Network, also known as ConvNet or CNN, is a deep learning algorithm that is most commonly used for visual imagery. This algorithm takes images as input and assigns them their importance in the form of learnable weights and biases in order to differentiate between different images (Saha, 2018). Because CNN's preprocessing is much simpler than that of other deep learning algorithms, it performs well even when applied to large amounts of data. The formation of the visual cortex served as the inspiration for the architecture of ConvNet, which is modelled after the pattern of connectivity found in the human brain's neurons. [1]

Regularized versions of CNNs, multilayer perceptron are typically fully connected networks. This means that each neuron in one layer is connected to all of the neurons in the layer below it. Multilayer perceptron are regularized versions of CNNs (Convolutional neural network, 2019). Because neural networks are quite complicated and have a tendency to overfit, the solution to this issue is to make minute adjustments to the learning algorithms, which is also referred to as regularization. This will result in an improvement in the performance of the modal. CNN approaches regularization in such a way that they are at the lower end of the connectivity and complexity spectrum as a result of taking advantage of the hierarchical pattern in the data in order to use simpler and smaller patterns in their filters to create patterns of increasing complexity. This allows CNN to be at the lower end of the connectivity and complexity spectrum (Convolutional neural network, 2019).

CNNs require very little preprocessing in comparison to other deep learning image classification algorithms. This means that through automatic learning, the network learns to optimize filters or kernels, whereas in other algorithms, the filters were typically engineered by hand. In addition, CNNs require less memory than other deep

learning image classification algorithms. It turned out to be a significant advantage due to the fact that it did not rely on previous information in the process of feature extraction.

1.2 Problem Statement

Autonomous vehicles are intended to be capable of performing the majority of driving tasks that are normally performed by human drivers. Among all of these tasks, the parking problem continues to be an important one. This is not only due to the fact that it is a common scenario that each and every driver encounters in their day-to-day life, but also due to the fact that it reflects fundamental relationship that is at the heart of all driving tasks: a relationship between the control actions and trajectories that are constrained by the dynamics of the vehicle. [2]

Controlling a vehicle in parking can be defined as finding an appropriate strategy or a series of precise control actions that guide a vehicle from its initial position to its required final position and orientation, in general.

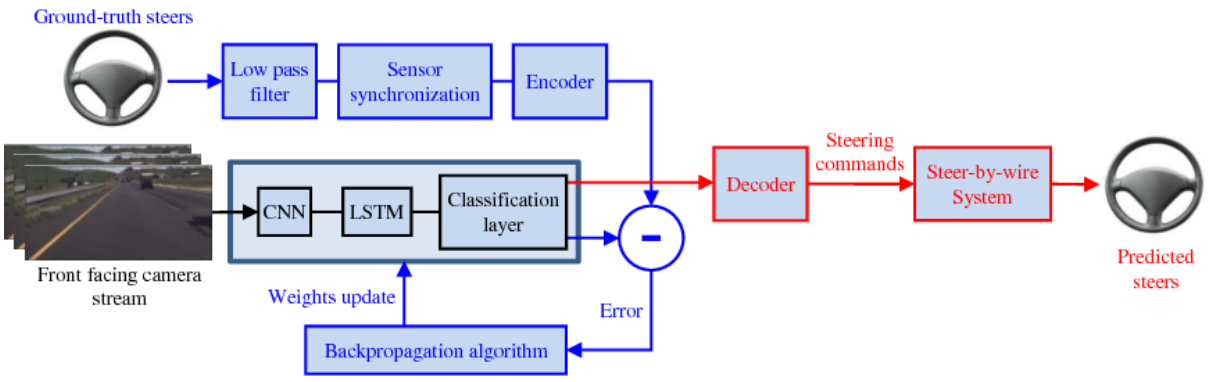


Figure 2: Overview of problem statement

Figure 3: Depiction of problem

1.3 Solution Approach

For the most fundamental modification of the auto-parking car, we utilized an end-to-end learning strategy, which consists of training Convolutional Neural Network (CNN) with assistance of the images. The steering commands are mapped directly to the raw pixels that are extracted from the images obtained by a single front-facing camera. This deep learning approach helped us in a way that it learns the weights and kernels automatically, with the exception of the fact that only the hyperparameters would be known. This is because there is no approach that is considered to be classical in nature for computer vision. The following is a summary of the project's primary concept:

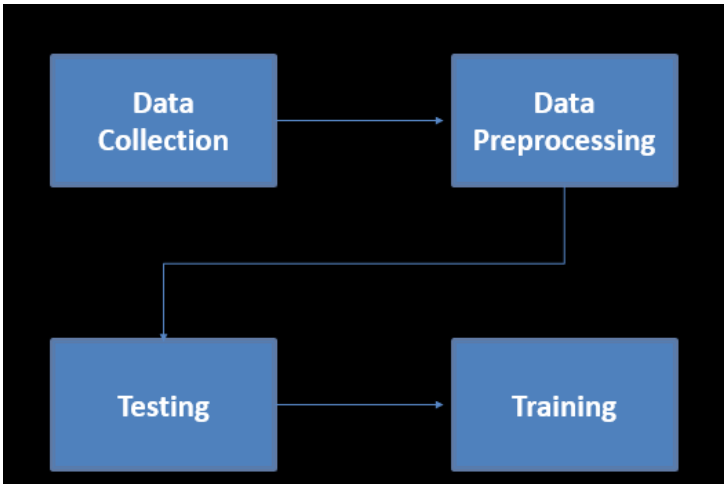


Figure 4: Overview of problem statement

Work	Explanation
Analyze and Preprocess Data	Creation of Dataset Preprocessing of data Labeling of Dataset Processing of data
Model Compilation and Training	Building of CNN model for training Training of Model Training and tuning the parameters
Testing and Visualization	Testing of the trained Model Visualizing the tested Model

1.4 Motivation

It's possible that finding parking is the most time-consuming part of driving. Even for veteran urban dwellers, perpendicular parking can be a challenge. And nobody takes pleasure in driving around in circles in a packed shopping center parking lot, vying with other aggravated drivers for the few available parking spots. People who shop at sprawling shopping centers may find this to be a time-saver, and drivers who struggle with physical limitations may find this to be of great assistance. A system like this could also be used to pack more vehicles into areas that have a restricted amount of space, such as a small parking lot in an area that is known for its lively nightlife.

Our primary objective was to design a method that would enable us to accurately park the car at any empty space available in the parking area while minimizing the amount of error that occurred in the process. This was done because although work of this nature is in high demand, the majority of businesses keep the details of their programs extremely confidential; consequently, doing work of this nature will also prove to be very cost-effective in the future.

1.5 Goals and Objectives

- Studying and Learning UGV model
- Learning python and Deep learning libraries (Open CV, Tensor flow, scikit-learn, Keras)
- Preprocessing of Dataset
- Labelling and creation of Dataset using CARLA simulator
- Training of CNN model on CARLA simulator
- Design and implementation of Controller

1.6 Technologies/Software Used

- CARLA simulator: version 0.9.10 (we didn't used latest model of CARLA because of the requirement of high-end GPU)
- We used older versions of libraries to connect with version of CARLA
- To train our network on Graphical Processing Unit (GPU), we used CUDA framework and drivers.
- For training our network we used Python 3.10
- PyCharm version: 2021.1.3
- Libraries: TensorFlow, Open CV, scikit learn, Keras
- CNN model: NVIDIA
- GPU: NVIDIA GeForce GTX 1650 (4GB)
- Anaconda/Conda version: 4.13.0
- CUDA toolkit Version 11.6
- CuDNN Version: latest version 8.3

CHAPTER 2

2. LITERATURE REVIEW

As it gets increasingly difficult to park on our own, automatic parking is becoming increasingly important in everyday life. Some of the causes are as follows. To begin with, typical car drivers struggle to accurately assess the vehicle's surrounds, particularly at the rear. Furthermore, parking is a tough action for drivers to do quickly by moving the steering wheel, accelerating, or braking. Finally, an increase in the number of cars in a confined and congested parking lot necessitates a higher level of skill. The University of Michigan Transportation Research Institute investigates and analyses parking accidents. The results of the poll suggest that there are Approximately 10,000 car accidents occur each year as a result of entering or exiting the country. [3]

Automatic parking of car is driving an infinite attention in the field of automotive engineering because of huge number of traffic incidents and the requirements of drivers. Also, the research and the development of autonomous parking of a car is being conducted by a wide range of car manufacturers and research institutions.

2.1 Significance of Auto-Parking

Auto-parking of car can increase the safety of driving. It helps drivers to learn environment around the vehicle during parking process through the various sensors. Drivers are prompted to make correct operations by sound or video displays of human-computer interaction system. Vehicle collisions caused by a lack of human expertise can be prevented by using automatic parking to regulate steering wheel actions. To park the vehicle into a parking space, the driver merely needs to regulate the accelerate and brake. It has the potential to reduce the driver fatigue and improve much of the driving comfort. It is also used extensively in military applications. It is one of the requirements for ground mobile robots. Meanwhile, it is one of the technical basis for human - robot interaction and can be extended to more complex tactical planning. Furthermore, automatic car parking is an important component of an intelligent traffic system (ITS). ITS is a transportation system that uses information technology, computer connectivity, and sensor technology to improve highways, urban roads, public transportation, and vehicle parking in a safer and more comfortable manner. [4]

2.2 End-to-End Learning for Self-Driving Cars by NVIDIA

To convert raw pixels to steering commands, we used a Convolutional Neural Network and a front-facing camera in the project. The system has been trained to drive in different parking spots that are either empty or mostly full. The system was never explicitly trained to detect; instead, it learns the critical processing operations, such as detecting road objects like lane markers, naturally. To train, NVIDIA Torch7 was employed, and the NVIDIA (PX) autonomous vehicle computer was used to determine where to drive. CNNs revolutionized pattern recognition jobs that previously required complicated methodologies. CNNs are excellent tools because they learn characteristics from training samples automatically. Furthermore, because of convolution kernels, picture recognition requires extremely few parameters. The CNN learns to steer in this experiment.

Figure 1 below depicts a block diagram of DAVE-2 training data collecting. Three cameras record video while the human driver controls the steering angle. To avoid singularity when driving straight, vehicle's Controller Area Network (CAN) is utilized to get steering command, and the inverse of turning radius (1/r) is employed to represent steering angle. (Mariusz Bojarski, 2016)

To generate training data, images from the video are sampled and matched with steering orders. To the training data, additional photographs of a car in various shifts from middle of lane and rotations from direction of the road are included. Because the data from human driver is insufficient and the network is prone to errors, more photographs have been included. [5]

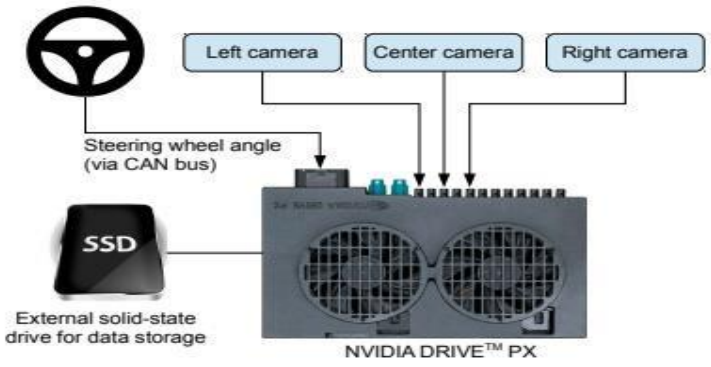


Figure 5: High-level view of the data collection system

The training system of the vehicle is depicted in the block diagram below. CNN analyses the images which are received by CNN and determines the best driving command. This driving command is then compared to desired command for that image, and the changes are made to CNN's weights to match the output of CNN with the desired output. Weight modification is accomplished by the use of back propagation.

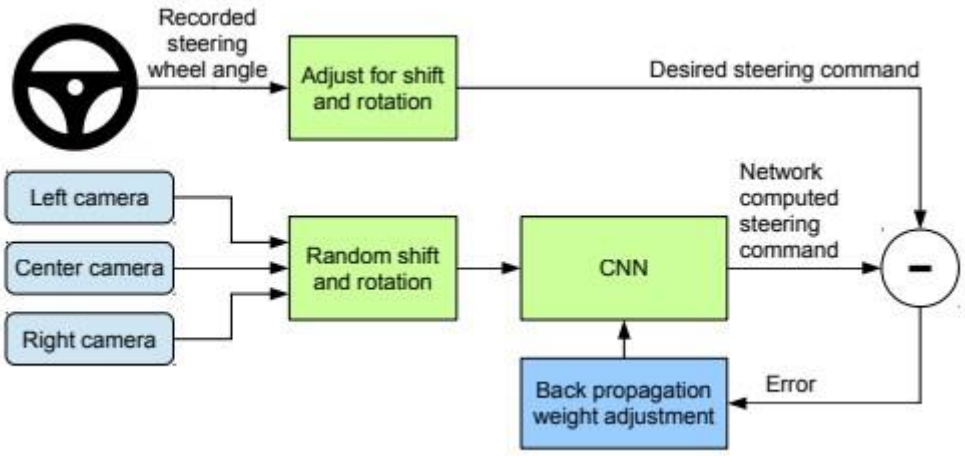


Figure 6: Training the neural network

The network, once trained, generates steering from video pictures of middle camera. Figure 3 shows this configuration.

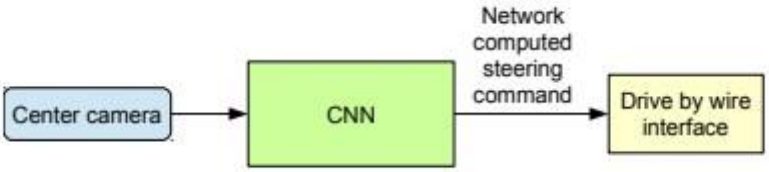


Figure 7: The trained network to generate steering commands

The training data was obtained in a variety of situations, including several roadways, weather conditions, and lighting conditions. The vast majority of the road data was gathered in New Jersey, Michigan, Illinois, New York, and Detroit. Unpaved roads, freeways, residential roads,

parking lots, and tunnels were among the road types. Different weather situations, such as foggy, snowy, rainy, night and day, were also considered. Highly skilled drivers were assigned the task of driving with complete focus. Driving data was gathered for approximately 72 hours.

The weight training was carried out in such a way that mean square error between the steering command output and either the corrected steering command for off-center or the human driver was minimized. The network has nine layers, including a normalization layer, the convolutional layers, and the three fully-connected layers. Input photos are divided into YUV planes and sent to the network. The graph is presented below. 2016 (Mariusz Bojarski)

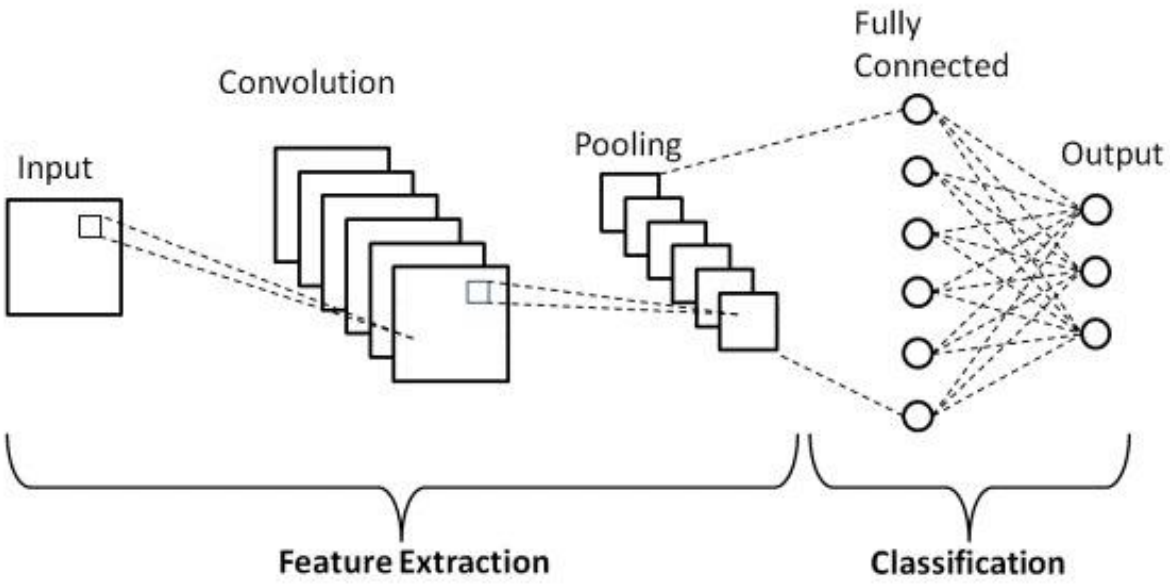


Figure 8: CNN architecture

The first layer of CNN is in charge of image normalization.

Convolutional layers are responsible for feature extraction. Strided convolutions with 2x2 stride and 5x5 kernel size were used in the first three convolutional layers. While non-strided convolutions with 3x3 kernel size were utilized in the last two convolutional layers. There are around 27 million connections and 250 thousand characteristics in the network.

First, we select frames to use for data selection. The data is organized into three categories: road type, weather condition, and driver activity. The video is sampled at 10 frames per second since greater frames per second result in similar image usage, which is inefficient. [6]

Artificial rotations and shifts are introduced to the final set of frames to educate the network how to recover from bad locations and orientation. The data is taken at random from normal distribution which have mean of zero and standard deviation of twice that we measured with human drivers.

The network is first evaluated on simulation before test on road. Figure 5 demonstrates the block diagram of the simulated system.

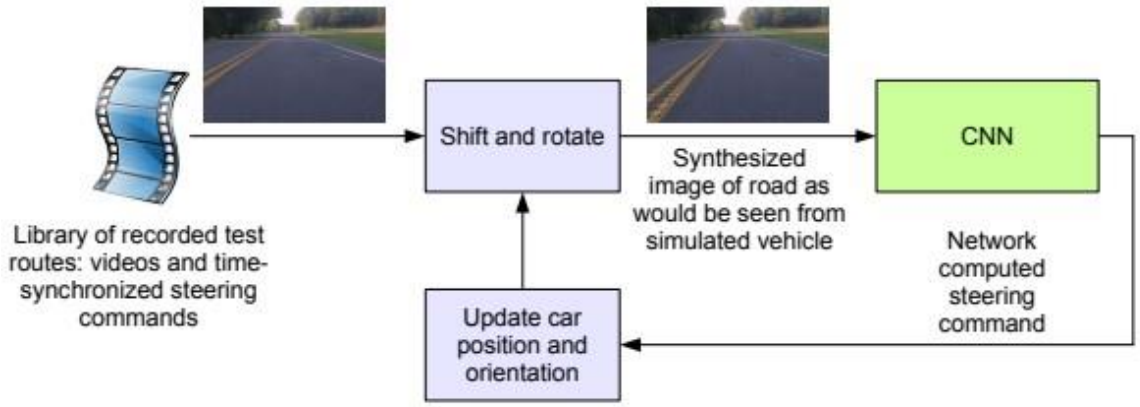


Figure 9: Block diagram of drive simulator

The simulator analyzes human-driven data from the front camera and compares it to what would happen if CNN drove the vehicle. The center lane is manually adjusted because the driver cannot always drive in the center. This is known as "ground truth." The simulator alters the frames such that the vehicle appears to be positioned from CNN's steering command. The process is continued until a virtual vehicle's location and orientation match the original video's ground truth.

The simulation is used first, followed by the real-world road test.

The percentage of automotive autonomy is calculated during the simulation test. The formula for autonomy is as follows:

$$\text{autonomy} = (1 - \frac{(\text{number of interventions}) \cdot 6 \text{ seconds}}{\text{elapsed time [seconds]}}) \cdot 100$$

The autonomy is 90 percent after 10 trials and six hundred seconds.

The network is entered into DRIVE PX for the on-road test, and the test vehicle is driven. This is the amount of time that the car drives autonomously. According to our data, the car drives itself 98 percent of the time.

Finally, we would like to state that we have successfully proved that CNNs can learn entire tasks without the need for human assistance. The training worked in a variety of conditions despite having very little data. Without any specific training, the algorithm was able to learn and recognize road outlines. 2016 (Mariusz Bojarski)

2.3 COMMERCIAL APPLICATION STATUS

Volkswagen debuted a new parking assist system concept on Hannover in April 2008 called "PAV" (Park Assist Vision). An auto-parking vehicle has a number of features with cameras in the reversing mirrors on left and the right to identify the location of parking. The Collision Detection System (CDS) is in charge of detecting collisions by using the front and back cameras and ultrasonic sensors of the vehicle. It makes use of a high-performance computer with a clock speed of 2 GHz, process the data from the sensors Making use of this transformation system, steering wheel, the accelerator, and brake action, and the door locks can both be controlled automatically.

Automatic parking system of Audi's signifies the modern research— "one key type of parking". By using application program of intelligent mobile phone, operation of car parking can be achieved automatically. For example, by operating only one key on his/her mobile phone, he/she will be able to make their car find and reach the parking space automatically. And if "out of the garage" command is given by the application program, the car can get out of the parking space or garage and find another suitable place automatically. However, this whole parking method is based on laser scanners arranged in parking lot. The person can keep track with his car by wireless fidelity, and can make sure that the vehicle cannot deviate from its trajectory and collide with obstacles. [7]

2.4 FINDINGS AND DISCUSSIONS

Presently, research being carried out on automatic vehicle parking focuses on visual perception, ultrasonic sensor and radar technology, trajectory planning, neural nets, image processing and identification technology, and the technology of digital signal processing.

2.4.1 Visual Perception

Human errors, such as lack of awareness, distraction, drowsiness, insufficient training, and exhaustion, cause the majority of collision incidents. With the help of system of advanced driver assistance (ADAS), by keeping an eye on driving surroundings and alerting drivers to be aware of the future threats, human errors can be reduced. A driver assistance system can help you fix the images that have been altered through electronic distortion correction functions, and present a clear visual of what's going on behind the wheel to drivers. It can also make use of image processing to evaluate the distance among cars and other objects with precision. With the help of the trajectory lines, the real-time pictures and videos were superimposed on both static and dynamic level, they provide drivers with the Parking viewpoints that are accurate.

2.4.2. Image Processing and recognition technology

With the advancement of image processing and recognition technologies, several researchers have begun to investigate the use of image sensors in automated parking of vehicle. They use image and pattern in their work for recognizing the parking spaces. Sensors collect continuous pictures, which are then used to in order to create a three-dimensional virtual environment to find suitable parking spots. How to park a car is discussed by Daxwanger and Schmidt (1995). By driving into its parking spot, the vehicle etched out flag lines by the data obtained through the use of cameras. The cameras are set up and positioned on the vehicle's back so that indexes lines can be captured. Moreover, the controller's input data can be accessed through image captured filtering, and edge detection, reduced resolution as well as other types of pretreatments.

Li and Chang (2003) capture photos with cameras and apply real-time image processing to extract information about the posture and position of the vehicle including the parking space. After receiving the vehicle's steering angle adjustment information, to execute the parallel parking and vertical parage parking, fuzzy sliding and path tracking mode control are

employed. The disadvantage of this strategy is that it requires independent computer control due to the lack of integrated DSP chips, as well as a greater cost. [8]

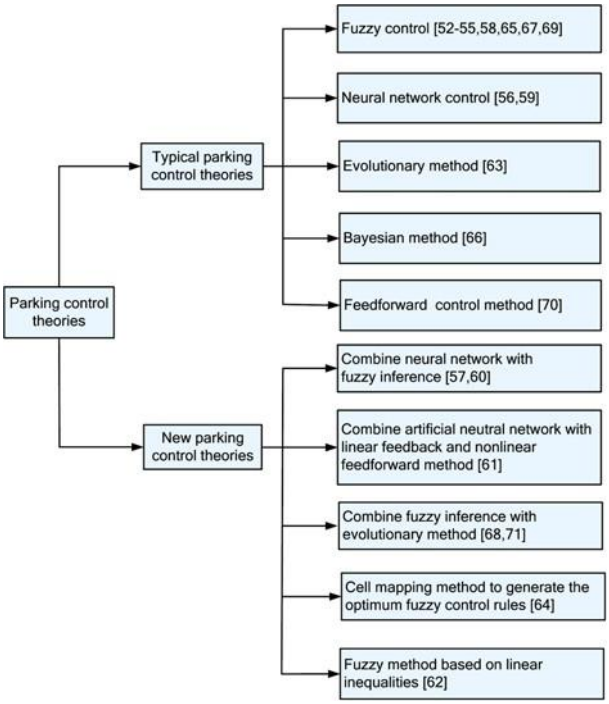


Figure 10: Comparison of typical and new parking control theories

There are many more comparable studies implemented by Chan-Hong and Ozkul et al. An automatic parallel parking method based on visual technology was developed by Chan-Hong and Cheng Hsiang (2005). Primarily, from omnidirectional visual pictures gathered by cameras, an image processing method is employed for the filtering of noise and to extract feature point coordinates of parking spaces. Then, using the characteristic information, the position of the car with relation to the parking spot is assessed, and valid frames of reference parking paths are provided. Lastly, a fuzzy logic controller is employed for the control of the steering wheel of parallel parking and tracking of reference path.

During a parking procedure, image processing and recognition refers to technology that uses ultrasonic sensors or CCD cameras for the record and processing of image data around the automobile. The required distance information and parking position can be recovered using CPU filtering, line extraction, and edge detection. Finally, by regulating the steering wheel to follow a reference path, automatic parking of a vehicle can be accomplished.

2.5 SUMMARIZATIONS AND REASONABLE CONCLUSIONS

To conclude the literature review, it is stated that path planning and the fuzzy control have key roles in related research. Following a survey of the primary literatures on autonomous parking, it was discovered that there are still some gaps that need to be investigated further. These gaps propose new study directions for the future. The automatic parking system and automatic driving have gotten a lot of attention and investigation. However, very little, if any, work is done on vehicle parking automation. Furthermore, a data collection for automatic car parking is not available. There is much more research and effort to be done on this side because there is a significant gap in the industry linked to automatic parking of a vehicle, thus we established a project and are working on automatic parking of a car to close this gap to some extent.

CHAPTER 3

3. METHODOLOGY

To design our controller, we listed the tasks in a specific order to achieve our objectives required. Tasks are in the order that every task was pre requisite of the previous target.
Perpendicular parking involves parked vehicles next to each other perpendicular to the wall or anything else. Perpendicular parking is thought to be the easiest of all parking strategies, including parallel, angle, and reverse parking. However, parking a car successfully with this strategy might be difficult for inexperienced drivers, especially when the available space is between two other vehicles. The driver should preserve enough space between his vehicle and the row of other parked cars when using this parking technique, turning wheel all the way on one side and proceeding to parking slot. And after reaching the halfway in the parking slot, when driver reaches the exact parking position, he should stop and straighten the wheels of the car.
We are using Deep Convolutional Neural Network (CNN) in our project work to tackle the problem of the perpendicular parking of autonomous vehicle. Now-a-days End-to-End learning has various applications and most of research work conducted is concerned to deep learning which is enhancing the accuracy and even replacing the conventional systems. It uses a great amount of data when training the CNNs from the scratch or pre-training of CNNs and that amount of data depends on the architecture of network and the problem we are handling. We searched a lot for the dataset for autonomous parking, but there is no dataset available. The datasets which are available were mostly related to safe, secure and emergency parking locations identification, public parking areas empty spaces, or detection of parking space. Collecting data is not an easy task, collection of data by ourselves is expensive as it requires an experienced and patient driver who drives the car hundreds or even thousands of times collecting photos and steering angles that translate to efficient parking behavior. On other hand, repeating this process hundreds or even thousands of the times may have an impact on the quality of the data acquired. Unmanned Ground Vehicle (UGV) model will be used to collect dataset and investigate the performance of trained model.

3.1 Deep Learning Model

Our strategy is motivated by NVIDIA's work on autonomous vehicles and self-driving cars. The following layers comprise our deep learning model:

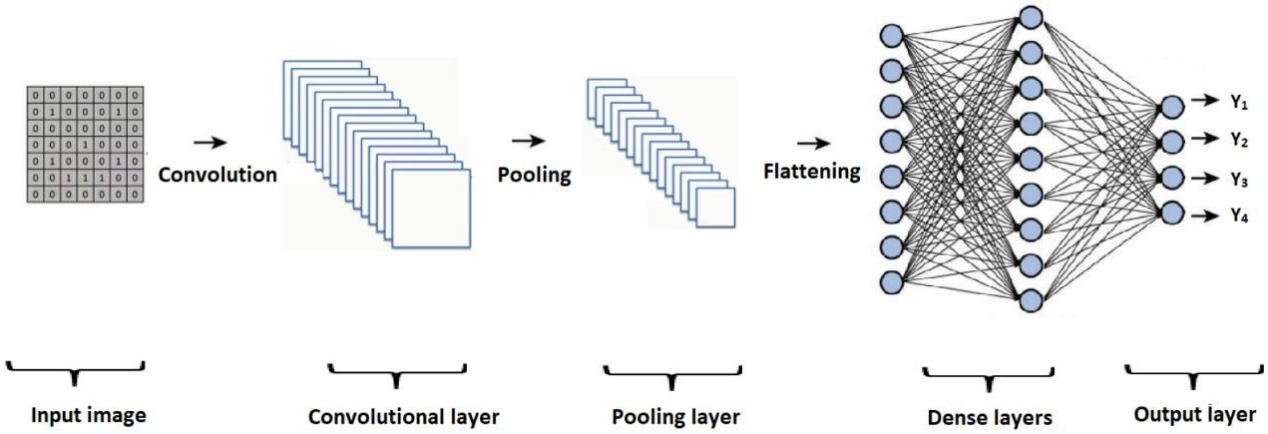


Figure 11: CNN layers

3.1.1 Convolutional Layer of CNN:

Convolutional layer is the first layer of ConvNet that uses convolution operations to apply filters to input images to generate an output that is a three-dimensional activation of neurons. The convolution operation generates a single value from all pixels in an image's receptive field. This layer aids in categorization by detecting various picture features, and its final output is a vector. The number of convolutional layers in a network change depending on how this layer is applied. The first layers of a network's convolutional layer detect low-level visual features, whereas the remaining layers detect high-level feature detection.

3.1.2 Maxpooling Layer of CNN:

The max or maximum pooling layer is an operation that calculates the largest or maximum value present in each patch of the feature map and down samples the output of the pooling.

Apart from max pooling, there are various pooling methods such as average, global, and L-2 norm pooling, however max pooling is the most commonly employed.

This pooling or down sampling shortens the training time without sacrificing the precise information of image features and lowers the dimensionality. The max pooling layer down samples an image by pooling neurons and produces a picture that is smaller than the input image (Shraddha Manchekar).

3.1.3 Dense Layer of CNN:

The fully connected layer or dense layer is the last or final and most typically utilized layer of a convolutional neural network. In dense layer, all neurons in a layer are connected to each neuron in the preceding layer (Shraddha Manchekar), which means that each neuron receives input from every neuron in the previous layer.

The background operation of a fully connected layer is matrix vector multiplication, in which the trainable parameters are the matrix values, and we can update these parameters utilizing the phenomena of back propagation (Sharma, 2020). The following are some dense layer parameters:

- Activation: The activation parameter applies an element-wise activation function.
- Unit: When it comes to the size of the weight matrix with bias vector, this parameter plays a part by indicating the layer output size with a positive integer value.
- Use-bias: Using a bias vector for calculation is determined by the use-bias parameter, which is initially set to true.
- Constraints: The constraints dictate the values of a weight matrix or a bias vector. (Sharma, 2020)

3.1.4 Batch Normalization Layer of CNN:

Deep neural networks are difficult to train since they are typically sensitive to learning the algorithm's initial setup and weights. Such a problem can develop when weights update after each small batch affects the distribution of the inputs to the layer. When we are training a deep

neural network, we apply batch norm normalization. It basically tackles the problem by standardizing the inputs to the layer for each mini batch and therefore drastically lowering the number of epochs by stabilizing the learning process for deep networks. This normalization is done in such a way that the output standard deviation remains close to one and mean output remain close to zero.

3.2 Neural Network Architecture

In our tests, we used feed forward neural networks and backpropagation neural networks. We employ networks with convolutional layers as the first layers because we are working with picture inputs. The first layer is the input layer, which takes images with dimensions of 160X320X3, where 160X320 is the image height and width and 3 is the number of channels (RGB), and then there are 5 convolution layers with 24, 36,48,64,64 neurons, with the first three having filter sizes of 5X5 and a jump at each filter pass of 2X2 pixels (strides of size 2X2), and the final two layers having filter sizes of 3X3 with no strides. The strategy of increasing the number of neurons at each layer aids in the extraction of visual information and features. Elu is the activation function for all convolution layers because it converges to zero faster than Relu and produces more accurate results. A dropout layer is put after the convolution layers to prevent overfitting. The dropout layer implements it by randomly picking and deactivating a specific number of neurons (50 percent in our case). The flatten layer would then convert the multidimensional output of the convolution layers to a one-dimensional output. The last layer of the network is a 5 dense layer network with 100, 100, 50,10,3 neurons apiece, all having elu activations except for the output layer, which has a linear activation because this is a regression problem. These layers are all Sequential models. [9]

The photos are used as input data for the model, and the steering angle of vehicle, throttle, and brake are used as labels in the form of Numpy arrays. The main parameters (number of layers, number of neurons, steps per epoch, number of epochs, learning rate, and batch size) are determined empirically, and the ideal values obtained after a number of attempts are:

Batch size =32
Number of epochs = 50

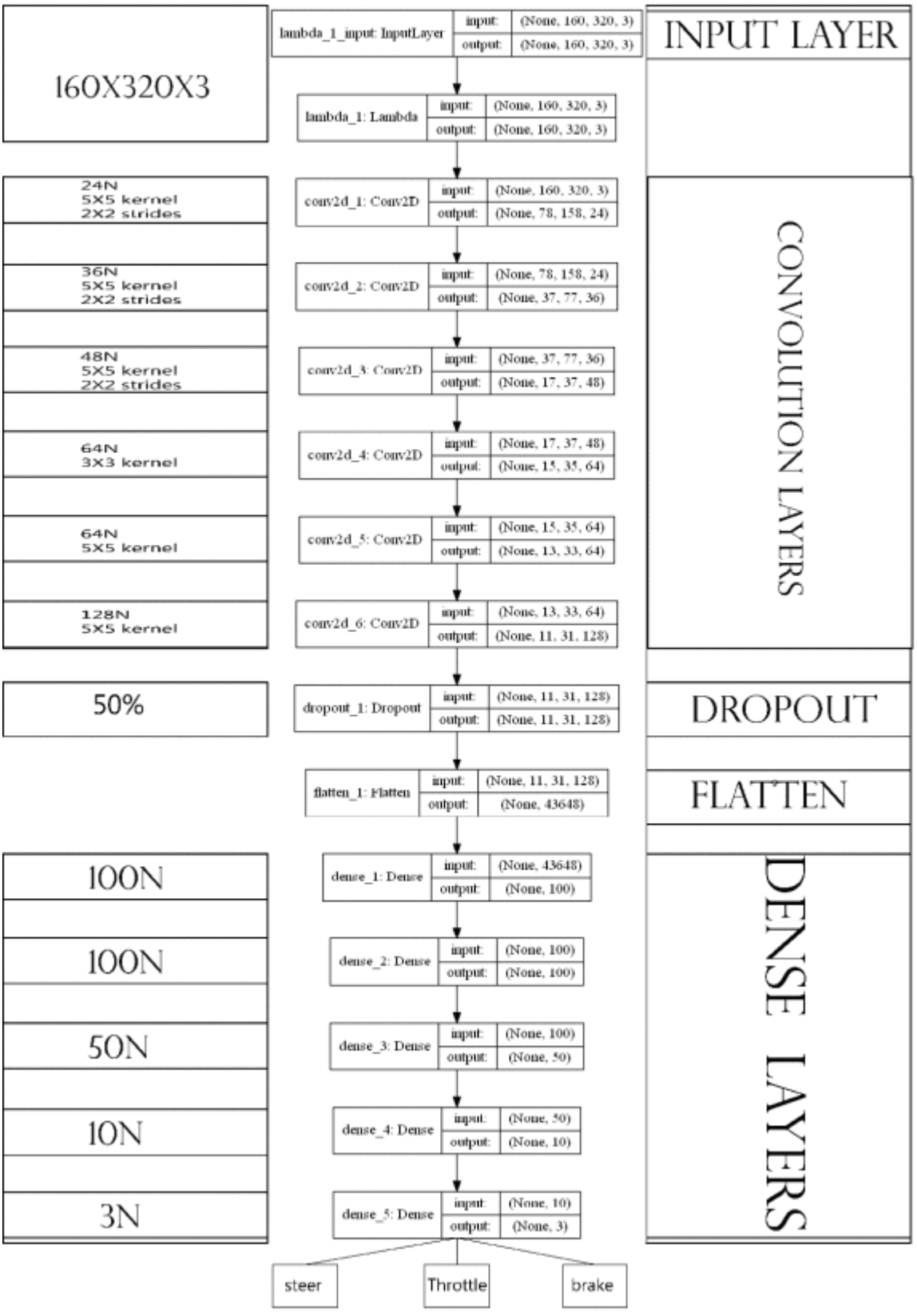


Figure 12: NN Model Architecture

3.3 NVIDIA Model

By making roadways safer and more efficient, self-driving cars are transforming the way we live, work, and play. These breakthrough advantages demand massive computational power and experience in large-scale manufacturing software. NVIDIA has created a software-defined, end-to-end platform for the transportation industry that allows for continuous improvement and deployment via over-the-air updates, building on decades of experience in high-performance computing, image processing, and artificial intelligence. It includes everything needed to construct large-scale autonomous vehicles.

The NVIDIA model basically is inspired from CNN in NVIDIA end to end dev op paper that came in 2016.

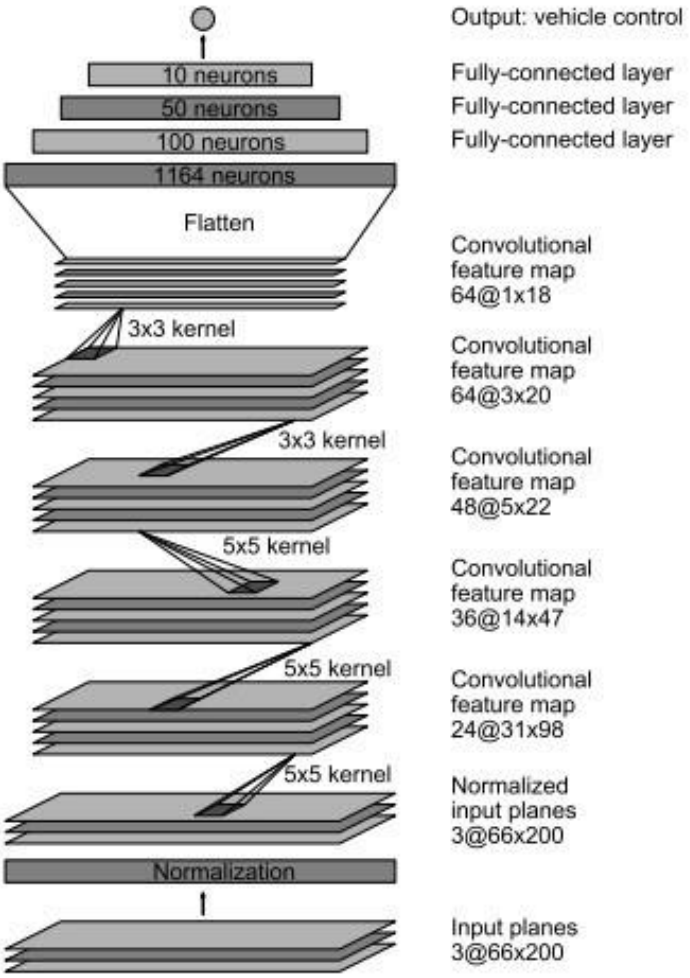


Figure 13: NVIDIA Model Architecture

NVIDIA model has total 9 layer and following parameters:

- Normalization layer
- 5 Convolutional Layers
- 3 Dense/fully-connected layers
- Optimizer ‘Adam’
- Epochs = 50
- Batch Size = 32
- Activation function of each layer ‘elu’
- Loss ‘Mean Square Error’

CHAPTER 4

4. DESCRIPTION OF SOFTWARE

We used CARLA Simulator, a Microsoft development for testing latest innovations, for generating our data set. CARLA simulator is an open-source simulator for self-driving research. CARLA simulator was built from the ground up to aid in the development, training, and validation of the autonomous urban driving systems. It can be used as a modular and adaptable API to address a variety of activities associated with the problem of autonomous driving.

We used TensorFlow with Keras as the backend because a huge quantity of processing power was required for a large amount of data. TensorFlow is an open-source machine learning framework with a vast and versatile tool system, community resources, and a variety of libraries that operates from start to finish. With the advancement of state-of-art machine learning, this allows researchers and the developers to quickly create and deploy Machine Learning applications (An end-to-end opensource machine learning platform, 2021). TensorFlow 2.9.1 was utilized in conjunction with Keras. Keras is a high-level TensorFlow API that is easy to use. It is a high productivity interface that is used to address difficulties linked to machine learning, with a concentration on current deep learning. It provides some building pieces and key concepts for constructing and launching high iteration rate machine learning systems (About Keras, 2021). Many engineers and researchers nowadays make full use of TensorFlow's cross-platform functionality, as it can run on enormous clusters of graphics processing units and be exported to a mobile device or a browser.

TensorFlow's three primary advantages are, first and foremost, its ease of use when it comes to model creation. Second, it delivers reliable machine learning anyplace, and third, it offers powerful experimentation for research purposes. TensorFlow makes it simple to train machine learning models, especially with its high-level Kera API, and it can be done regardless of the programming language used. It features a basic yet adaptable design that allows us to turn our ideas into code and models (An end-to-end opensource machine learning platform, 2021).

There is a small limitation in TensorFlow in that part work must be done in a version such as 2.4.0 and the other half must be done in a version such as 2.4.1, but both tasks must be completed. This cannot be done at the same time. We utilized the Anaconda Package to solve this problem. This allows us to construct multiple environments at once. The Anaconda is a well-known platform among Professionals of data science and information technology. It's an open-source platform with several tools for data collection using AI and machine learning. It is extremely manageable when it comes to the building and deployment of environments and is commonly used for machine training, development, and testing on a machine (Python Anaconda Tutorial, 2021).

We used NVIDIA's CUDA framework/toolkit to train our network on a Graphical Processing Unit. CUDA is a GPU-accelerated DNN (deep neural network) library, while cuDNN is a GPU-accelerated deep neural network primitive's library. CuDNN is a GPU accelerator of high-performance that is commonly used by the deep learning researchers and professionals around the world. Instead of wasting time on low-level GPU performance, it assists users in developing and training network models and designing software applications. CuDNN (NVIDIA cuDNN, 2021) accelerates deep learning frameworks like as TensorFlow, PyTorch, MATLAB, Keras, and others. The deep learning library handles standard operations including pooling, forward and backward convolution, activation layers, and normalization (NVIDIA cuDNN Documentation, 2021).

It features a basic yet adaptable design that allows us to turn our ideas into code and models (An open-source end-to-end machine learning platform, 2021). Another package we utilized in our project was Open CV, which we combined with other libraries like NumPy and Python to analyze Open CV array structures. OpenCV is the largest and most widely used open-source library for Machine Learning, Computer Vision, and Image Processing. Its real-time functioning aids in the recognition of faces, objects, and handwriting identification. Open CV was created with the primary objective of providing real-time access to information to make computations more efficient. It is also free for users and works with iOS, Python, C, Java, and other interfaces are available for Linux, Mac OS, Android, and Windows (OpenCV overview, 2019).

Our software section of the project covers five major steps that are illustrated as follow:

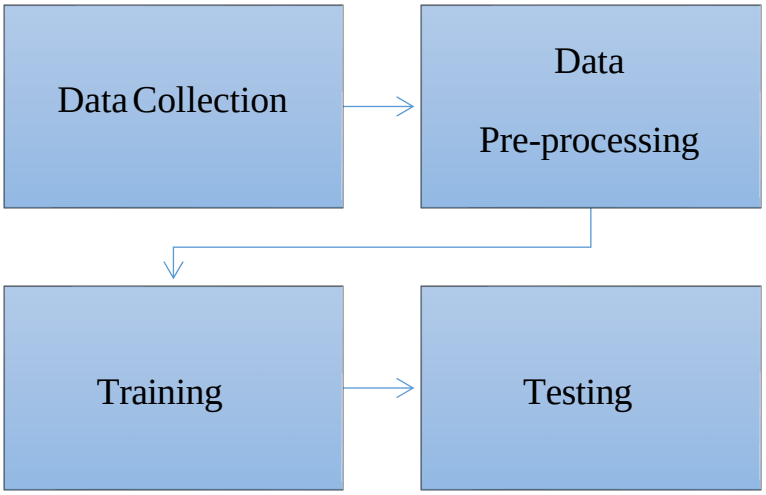


Figure 14: Stepwise Software Setup

4.1 Data Collection

4.1.1 Carla Simulator

We use CARLA Simulator, a Microsoft development for testing latest innovations, for generating our data set. CARLA simulator is an open-source simulator self-driving research. CARLA simulator was built from the ground up to aid in the development, training, and validation of the autonomous urban driving systems. It can be used as a modular and adaptable API to address a variety of activities associated with the problem of autonomous driving.

Carla is virtual environment that was created from the ground up with the help of top software engineers and graphic artists. It includes a number of tool sets that assist with validation, training, and testing of self-driving cars This is not only an open-source project, but it also generates exquisite digit models of numerous cars and urban cities that can be used and implemented according to one's preferences. [10]

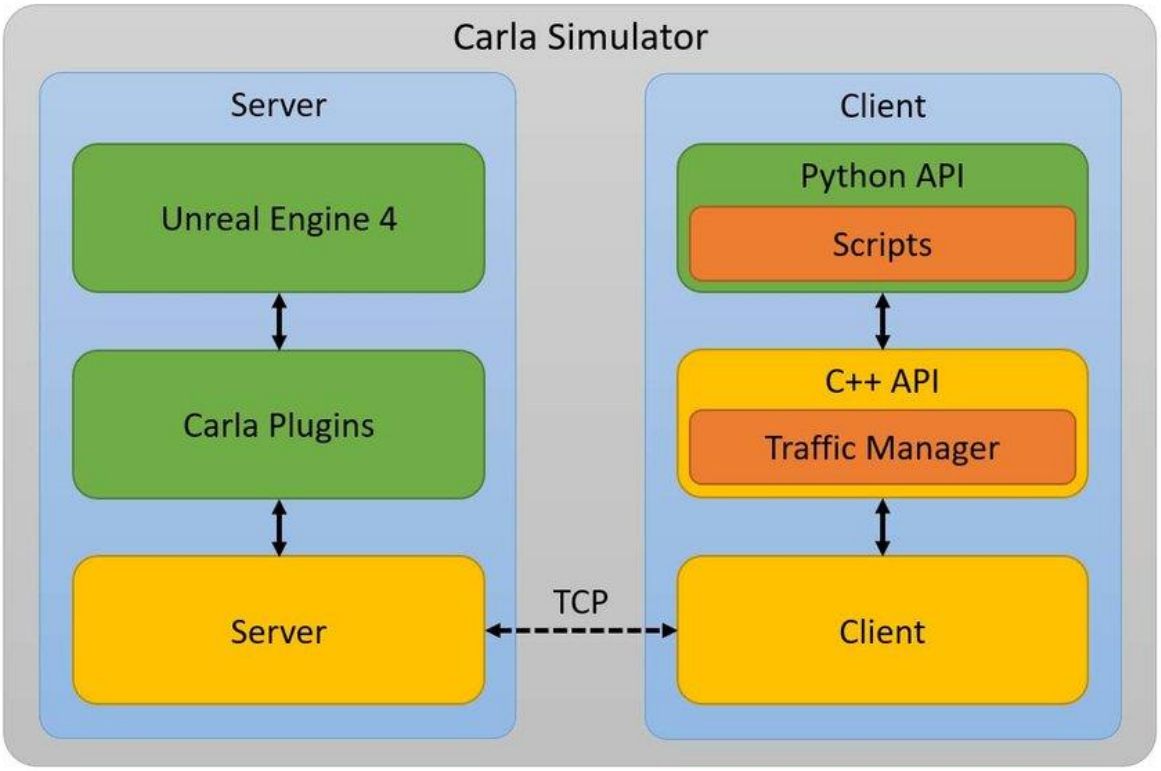


Figure 15: CARLA Simulator System Architecture Pipeline

4.1.1.1 Main Features

CARLA SIMULATOR provides us the following features

- Strong and robust API that allows us to control and design every part of simulation.
- Contains a large architecture with several clients connected at the same time to control various models that are present in the environment.
- Sensor libraries that assist autonomous vehicles in their operation.
- Various rendering presets allow you to pick between quality and speed.
- It comes with a toolbox that allows you to make your own maps for testing.
- Different traffic and pedestrian scenarios can be created.

- But, more crucially, CARLA is prospering; it has a vibrant community, is well-documented, and is (relatively) simple to use.

Figure 16:A Street view from Town 2 of Carla

4.1.1.2 Unreal Engine

It's built on Unreal Engine 4, which isn't the most recent version available right now (version 5 is the latest iteration). Considering the amount of raw processing, the graphical strength and realism of this engine are extremely impressive. It requires a lot of power to create and simulate the environment (Carla recommends 4gb Graphical processing unit memory).

Carla is adaptable in terms of allowing the environment to interact with the agent in a variety of ways in the made-up universe The simulation communicates with the server using Python, and the server then renders the results. The commands are delivered back to the client once the image and data collected from sensors has been processed. The agent is then simulated as needed**.** UE4 allows for the development of pretty complicated features without the need to use C++. Blueprints, a specific visual scripting language in UE4, is designed for this purpose. Blueprints were utilized to construct the

race track. In our example, the data collected from the server aids in controlling the car's acceleration, steering, and braking as needed. Environment parameters such as weather, pedestrian and car concentrations, and lightning conditions are also adjusted accordingly.

4.1.1.3 Environment & Sensors

Carla offers a wide range of choices for world creation, including pedestrians, other vehicles, and managing elements like as lighting. Pedestrians' movement and spawn are determined by the map on which they are permitted to use sidewalks and zebra crossings. In addition, we may evaluate collision avoidance properties and other features by simulating random movement on the road and check other thing related with computer vision data collected via optical sensors.

Figure 17: CARLA Environment with racetrack map

Figure 18: Daytime Parking Area with empty parking slots

In terms of sensors, it offers a wide range of optical sensors, including stereo vision for depth data with t The entire RGB spectrum is present. additional sensors such as radars, LIDARS, and so on are tested by developers and with each new update that is given out to users is added to the library. It offers the vehicle's geolocation in addition to sensors, and different measurements can be generated via graphs and trajectories.

Figure 19: A view from front camera of car in Carla

Figure 20: A view from front camera of car in Carla

4.1.1.4 Implementation of Autonomous Driving

Carla offers three potential techniques to automating vehicle movement for the implementation of our driving objective. The following are some of them:

●Modular Pipeline (One we are going to implement)

● DNN-based on imitation learning

● DNN-based on reinforced learning

Figure 21: A view of car in Carla detecting the empty parking slot

Figure 22: Auto parking of car in CARLA

Figure 23: Auto parking of car in CARLA

We chose the PID controller because of its flexibility, simplicity, and quick response time, which is one of the most important needs of any autonomous vehicle: it must react

to data without error or with the bare minimum of faults in order to give a safe and pleasant driving experience. We give our controller the waypoint, which is the location of the trajectory on a map, the speed at that location, and the vehicle's orientation. All of these factors are taken into account for vehicle longitudinal and lateral control. [11]

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Selection of Network Architecture:

The following are the reasons why we eliminated various models and chose NVIDIA with Deep Learning as our final model for training our convolutional neural network.

Why not ResNet 101?

ResNet 101 Model was discarded for the following reasons:

- One main reason is that training a deeper network typically takes weeks, making it almost impossible to use in real-world applications.
- It enabled the training of deep neural network, up to thousands of layers, making the model more complex to train.

Why Not Alex Net?

Alex Net was superior to NVIDIA, but it had its own set of problems, including:

- Shallow Network: The Alex Net is a very shallow network with fewer layers and fewer feature layers, such as the Convolutional and Maxpooling layers.

- When compared to other models, convergence to high loss.
- Low Accuracy: On our dataset, the Alex Net model has a low accuracy of 35.59 percent.

Why Not Customized Model?

The customized model was created with both Alex Net and NVIDIA architecture, but it was abandoned for the following reasons:

- Accuracy is good (60.67 percent), but not exceptional.
- Overfitting may be indicated by the difference between training and validation loss.

Why NVIDIA:

The following are the reasons for choosing this network:

- Their data collection included a wide range of variables, such as weather conditions, paved and unpaved roads, lane markers, parking lots, and freeways.
- High accuracy: On our dataset, the NVIDIA model provided a high accuracy of 95%.
- It enabled the training of very deep Networks, up to thousands of layers.

5.1 Accuracy:

As it can be seen from the following graph that the accuracy finally achieved was around 95%.

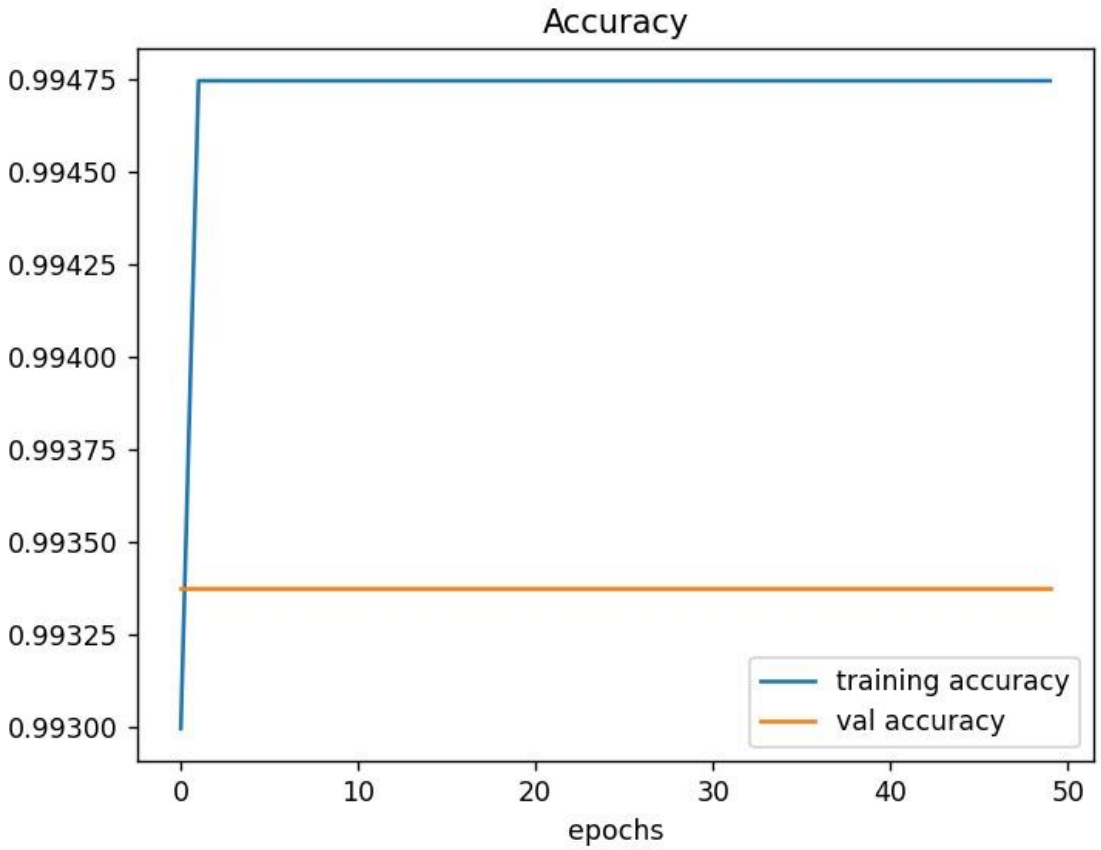


Figure 24: Accuracy of our model

5.2 Loss:

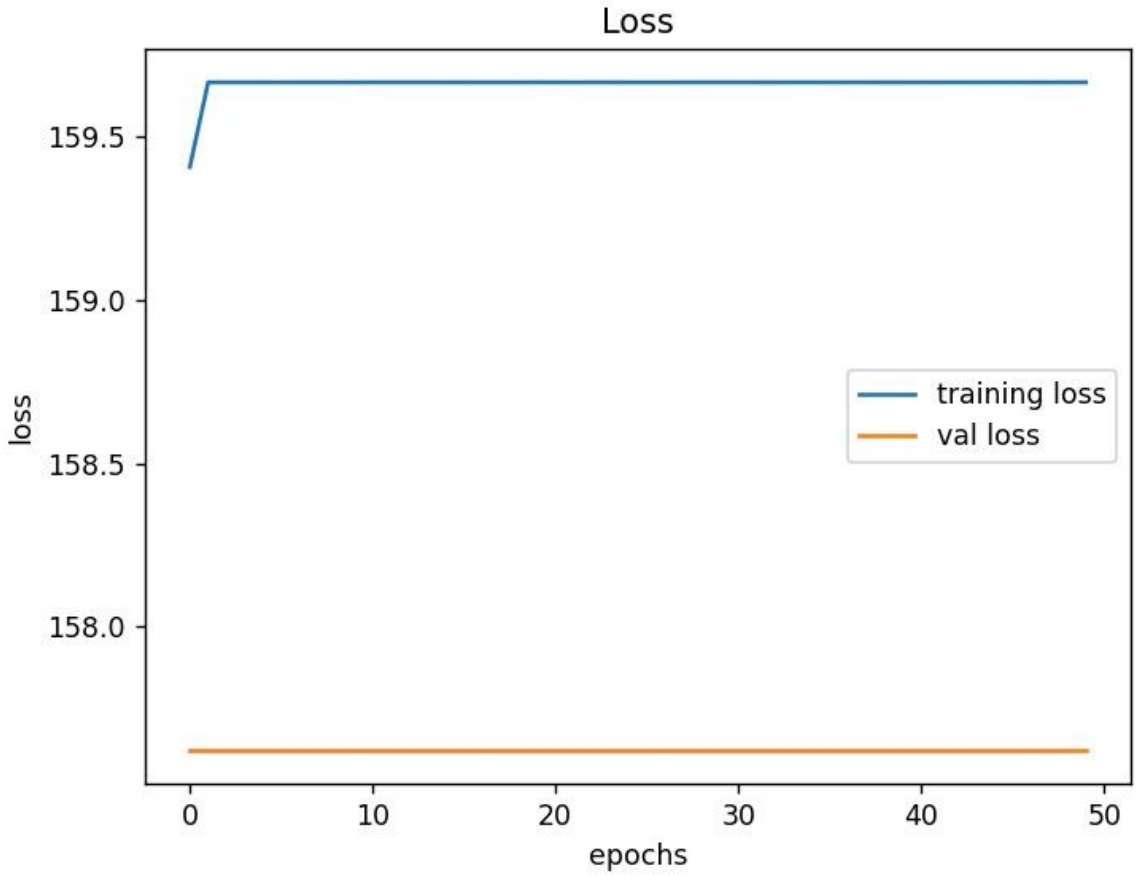


Figure 25: Loss of our model

The view from car and of the car aiming to the parking slot

Figure 26: view from car and of the car aiming to the parking slot

Figure 27: View from car and of the car parked in the parking slot

5.3 Limitations and Drawbacks of our work:

- Training would take an excessive duration without Graphical Processing Unit (GPU) processing. This means that the DNN's development, testing, and functioning were highly dependent on GPUs.
- To increase the accuracy, high computational power is required
- Driving speed must be below 13 km/h
- Auto-parking may not work with textured road surfaces such as cobblestone or brick.
- Auto-parking performance depends on the ability of the cameras (as we are using single camera) to determine the vehicle's proximity to curbs, objects, and other vehicles
- While raining, droplets on camera may affect the result of camera

CHAPTER 6

6. Conclusions

- The major goal of this project is to park a car automatically via a video sensor without the need for a human driver to interfere. To do so, we looked at current solutions and sought to incorporate as many structures and libraries as possible.
- We used supervised learning approach for the training of the model, that is, training a CNN with the help of images for basic adaptation of an Auto-Parking Vehicle, in order to replicate model with human's behavior and park the car in different environments and conditions without making mistakes.
- The neural networks employed in this study are regression neural networks that predict the steering angle output.
- We can mostly eliminate human experience and quickly and autonomously learn parking techniques.
- We used Deep ConvNet to solve the problem of perpendicular parking in this project work (CNNs).
- End-to-end learning has shown itself in a variety of applications, and the majority of current research is focused on using deep learning to improve the accuracy of traditional systems or even replace them.
- There is no dataset available for autonomous parking that we are aware of. The majority of the datasets accessible are connected to identifying the safe, secure and immidiate parking sites, occupancy of public parking places, parking signs, and parking space detection, thus we used a camera sensor to collect data from the CARLA SIMULATOR.
- Our model is based on NVIDIA's work on autonomous vehicles and self-driving cars. We've used this to train our model.
- The proposed method's excellent reliability and efficiency have been demonstrated through testing. By mapping the input visuals from the front camera into corresponding actions, the model was able to accurately replace the human driver's behavior. The model was able to determine useful road characteristics on its own using only human steering angle, accelerator, and braking amount applied by the human driver as labels.

6.1 SWOT Analysis:

• **Strengths:** End-to-End Deep Learning approach.

• Weaknesses: to train the model, a lot of data is required.

• **Opportunities:** Using more sensors like LIDAR, RADAR and Ultrasonic sensors accuracy can further be increased.

• **Threats:** no threats so far

6.2 Suggestions for Future work:

Some of the recommendations that would be beneficial in the future include:

- Dataset will be available for people who want to work on similar projects in the future.
- More data of the training in multiple tracks with varying the conditions of light is recommended to improve the capabilities of our CNN. This will undoubtedly increase the precision of car navigation. Additional capabilities, such as avoiding encountered obstructions on the track, can also be incorporated to the model.
- Our future study will include the real-time implementation of our suggested system with additional features such as efficiently searching for nearby parking spaces.
- The suggested solution tackles both the difficulty of perpendicular parking and the feasibility of using CNN to achieve autonomous parking. The approach can be utilized in self-driving cars that employ the same localization technology, or it can be used in connection with GPS and high-definition (HD) maps.
- With this project, users can gain other parking techniques, such as parking in the existence of static and dynamic obstacles, such as pedestrian, other automobiles, and so on.
- Accuracy can be improved by using more sensors LIDAR, RADAR, and ultrasonic sensors etc.
- By adding more convolutional layers to the same model, the accuracy can be improved.
- Using the Convolutional Neural Network on a car physically could have a significant impact on accuracy of model. Training the model on a simulator is insufficient because,

while the simulator is designed to be as realistic as feasible, it may overlook some of the real-world variables and specifications.

References

[1] K. a. R. N. O'Shea, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458 (2015).*

[2] W. J. A.-M. a. D. R. Schwarting, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems,* 2018.

[3] W. e. a. Wang, "Automatic parking of vehicles: A review of literatures," *International Journal of Automotive Technology,* 2014.

[4] H. e. a. Ye, "Linear model predictive control of automatic parking path tracking with soft constraints," *International Journal of Advanced Robotic Systems,* 2019.

[5] M. Bojarski, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316,* 2016.

[6] M. D. a. R. F. Zeiler, "Visualizing and understanding convolutional networks," *European conference on computer vision,* 2014.

[7] J. Zhao, "Parking, intelligent parking system," *019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC),* 2019.

[8] D. S. K. B.-H. K. J. G. a. H. K. T.-h. K. Pal, "Image Processing, and Pattern Recognition," 1998.

[9] A. J. L. E. a. M. K. Guez, "Neural network architecture for control," *IEEE control systems Magazine,* 1988.

[10] D. R. a. B. C. V. Niranjan, "Deep learning based object detection model for autonomous driving research using carla simulator," *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC),* 2021.

[11] D. R. a. B. C. V. Niranjan, "Deep learning based object detection model for autonomous driving research using carla simulator," *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC),* 2021.

[12] A. a. B. M. Serna, "Detection, segmentation and classification of 3D urban objects using mathematical morphology and supervised learning.," *ISPRS Journal of Photogrammetry and Remote Sensing 93*, 2014.

APPENDICES

CNN Model Code:

```
import keras.backend

import numpy

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import cv2

import tensorflow as tf

import os

from sklearn.model_selection import train_test_split

# from tensorflow.keras.utils import to_categorical

from keras.models import Sequential, load_model

from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

import csv


model_runer = 1


cur_path = os.getcwd()

# Retrieving the images and their labels

path = 'C:/FYP/Dataset/Images/New_data/'

# path = 'C:/FYP/Dataset/Images/Temp/'


path1 = os.path.join(path + 'dataset.csv')
```

```
# path1 = os.path.join(path + 'Test_images.csv')
temp = open(path1)
file = csv.reader(temp)
new_row = []
for row in file:
    new_row.append(row[0])
for j in range(0, 10):
    data = numpy.zeros((1024, 320, 160, 3), dtype=np.float16)
    labels = numpy.zeros((1024, 2))
    for i in range(0, 1024):

        path2 = path + 'output/'
        # path2 = path + 'Test_images/'

        path3 = path2 + new_row[(j*1024)+i]
        try:
            image = cv2.imread(path3, -1)
            # resize image
            image1 = cv2.resize(image, (160, 320), interpolation=cv2.INTER_AREA)
            # print(np.shape(image1))
            image2 = np.array(image1, dtype=np.float16)
            image2 = image2 / 255

            # sim = Image.from array(image)
            data[i, :, :, :] = image2
            labels[i, 0] = (float(row[1])) * 50
```

```
        labels[i, 1] = (float(row[2])) * 1
    except:
        print("Error in loading image!")
# Converting lists into numpy arrays
data = np.array(data)
labels = np.array(labels)
print(data.shape, labels.shape)
# Splitting training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.05, random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# Converting the labels into one hot encoding
# y_train = to_categorical(y_train, 15)
# y_test = to_categorical(y_test, 15)

# Building the model
if model_runer == 1:
    model = Sequential()

    model.add(Conv2D(filters=24, kernel_size=(5, 5), strides=(2, 2), input_shape=X_train.shape[1:]))
    model.add(Conv2D(filters=36, kernel_size=(5, 5), strides=(2, 2)))
    model.add(Conv2D(filters=48, kernel_size=(5, 5), strides=(2, 2)))
    model.add(Conv2D(filters=64, kernel_size=(3, 3)))
    model.add(Conv2D(filters=64, kernel_size=(5, 5)))
    model.add(Conv2D(filters=128, kernel_size=(5, 5)))
```

```
        model.add(Dropout(rate=0.5))


        model.add(Flatten())


        model.add(Dense(100, activation='elu'))
        model.add(Dense(100, activation='elu'))
        model.add(Dense(50, activation='elu'))
        model.add(Dense(10, activation='elu'))
        model.add(Dense(2, activation='linear'))
        model_runer = 0
    if model_runer == 0:
        model = load_model('C:/FYP/my_model12.h5')
    # Compilation of the model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


    epochs = 50
    history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
    model.save("my_model12.h5")
    keras.backend.clear_session()
    del model



# plotting graphs for accuracy
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
```

```
plt.title('Accuracy')

plt.xlabel('epochs')

plt.ylabel('accuracy')

plt.legend()

plt.show()


plt.figure(1)

plt.plot(history.history['loss'], label='training loss')

plt.plot(history.history['val_loss'], label='val loss')

plt.title('Loss')

plt.xlabel('epochs')

plt.ylabel('loss')

plt.legend()

plt.show()
```

Code for controlling speed of vehicle:

```
model = load_model('C:/FYP/my_model1.h5')

speed_a = 0

angle_a = 0

key_auto = 0


var = 1

angle = 0
```

```
speed = 0

key_no = 0



def autopark(image):

    image = cv2.resize(image, (160, 320), interpolation=cv2.INTER_AREA)

    image = np.expand_dims(image, axis=0)

    image = np.array(image)

    image = image / 255

    pred = model.predict(image)

    c = pred[0]

    speed = c[0]

    angle = c[1]

    speed = speed

    angle = angle * -1

    speed = speed - 33

    angle = angle - 21

    speed = speed/50

    angle = angle/10

    global speed_a

    speed_a = speed

    global angle_a

    angle_a = angle



def process_data(image, speed1, angle1):
```

```
    global var
    var = var + 1
    a = str(var)
    cv2.imwrite("C:/CARLA_0.9.10/WindowsNoEditor/PythonAPI/examples/Dataset/output/" + a + ".png", image)
    f = open('C:/CARLA_0.9.10/WindowsNoEditor/PythonAPI/examples/Dataset/dataset.csv', 'a', newline="")
    b = a + '.png'
    tup = (b, speed1, angle1)
    writer = csv.writer(f)
    writer.writerow(tup)
    f.close()
```

Code for saving images and reading images by CNN model:

```
def _parse_vehicle_keys(self, keys, milliseconds):
        if key_auto == 0:
            if keys[K_UP] or keys[K_w]:
                self._control.throttle = min(self._control.throttle + 0.005, 0.45)
            else:
                self._control.throttle = self._control.throttle / 1.005

            if keys[K_DOWN] or keys[K_s]:
                self._control.brake = min(self._control.brake + 0.2, 1)
                if self._control.throttle > 0:
                    self._control.throttle = self._control.throttle - 0.01
```

```
        if self._control.throttle < 0:
            self._control.throttle = 0

else:
    self._control.brake = 0

steer_increment = 5e-4 * milliseconds
steer_increment = steer_increment

if keys[K_LEFT] or keys[K_a]:
    if self._steer_cache > 0:
        self._steer_cache = 0
    else:
        self._steer_cache -= steer_increment
elif keys[K_RIGHT] or keys[K_d]:
    if self._steer_cache < 0:
        self._steer_cache = 0
    else:
        self._steer_cache += steer_increment
elif self._steer_cache > 0:
    if self._steer_cache > steer_increment:
        self._steer_cache -= steer_increment
    else:
        self._steer_cache = 0
elif self._steer_cache < 0:
    if self._steer_cache < -steer_increment:
```

```
            self._steer_cache += steer_increment
        else:
            self._steer_cache = 0
    else:
        self._steer_cache = 0
    self._steer_cache = min(0.7, max(-0.7, self._steer_cache))
    global angle
    angle = self._steer_cache
    global speed
    speed = self._control.throttle
else:
    self._control.throttle = speed_a
    self._steer_cache = angle_a
    self._control.brake = 0
self._control.steer = round(self._steer_cache, 1)
self._control.hand_brake = keys[K_SPACE]
```