DrivingMatter: An Autonomous RC Car using Raspberry Pi

Thesis · May 2018

DOI: 10.13140/RG.2.2.22396.85128

3 authors:

Syed Owais Ali Chishti
National University of Computer and Emerging Sciences, Peshawar
4 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Muhammad Bilal Zaib
National University of Computer and Emerging Sciences
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Sana Riaz
National University of Computer and Emerging Sciences
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

DrivingMatter: An Autonomous RC Car using Raspberry Pi

Project Team

Syed Owais Ali Chishti p14-6011
Hafiz M. Bilal Zaib p14-6099
Sana Riaz p14-6114

Session 2014-2018

Supervised by

Dr. Mohammad Nauman

Department of Computer Science

National University of Computer and Emerging Sciences
Peshawar, Pakistan

May, 2018

Student's Declaration

We declare that this project titled "*DrivingMatter: An Autonomous RC Car using Raspberry Pi*", submitted as requirement for the award of degree of Bachelors in Computer Science, does not contain any material previously submitted for a degree in any university; and that to the best of our knowledge, it does not contain any materials previously published or written by another person except where due reference is made in the text.

We understand that the management of Department of Computer Science, National University of Computer and Emerging Sciences, has a zero tolerance policy towards plagiarism. Therefore, we, as authors of the above-mentioned thesis, solemnly declare that no portion of our thesis has been plagiarized and any material used in the thesis from other sources is properly referenced.

We further understand that if we are found guilty of any form of plagiarism in the thesis work even after graduation, the University reserves the right to revoke our BS degree.

Syed Owais Ali Chishti Signature: _____

Hafiz M. Bilal Zaib Signature: _____

Sana Riaz Signature: _____

_____

Verified by Plagiarism Cell Officer
Dated:

Certificate of Approval

The Department of Computer Science, National University of Computer and Emerging Sciences, accepts this thesis titled *DrivingMatter: An Autonomous RC Car using Raspberry Pi*, submitted by Syed Owais Ali Chishti (p14-6011), Hafiz M. Bilal Zaib (p14-6099), and Sana Riaz (p14-6114), in its current form, and it is satisfying the dissertation requirements for the award of Bachelors Degree in Computer Science.

Supervisor

Dr. Mohammad Nauman Signature: _____

Shakir Ullah

FYP Coordinator
National University of Computer and Emerging Sciences, Peshawar

---

Dr. Omar Usman Khan

HoD of Department of Computer Science
National University of Computer and Emerging Sciences

Acknowledgements

We are very thankful to Almighty Allah who is most merciful and most beneficent for giving us strength and courage to choose such challenging project and for completing Final Year Project report with great dignity and grace. We have put great deal of effort in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We would like to express deep gratitude to our project supervisor Dr. Mohammad Nauman, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. We would also like to thank Director of FAST NUCES Peshawar and Head of Computer Science Department, for providing budget and space for conducting our experiments. Our grateful thanks also extended to Associate Professor Dr. Omar Usman Khan and assistant professor Fazl-e-Basit, for being kind and helpful throughout the project. We are highly indebted to FAST NUCES university Peshawar for providing us platform to showcase our work. Thanks and appreciations also go to the colleagues in developing the project and people who have willingly helped us out with their abilities.

Finally, our sense of gratitude to one and all, who directly or indirectly, have lent their support in this project.

Syed Owais Ali Chishti
Hafiz M. Bilal Zaib
Sana Riaz

Abstract

DrivingMatter is an experiment carried out to understand the deeper side of an autonomous car. In 1900s, idea was to drive car on Moon from Earth. This was initial motivation which grow from there and now expanding to complex system of road in our world.

Using the hardware and software aspect of an autonomous car, built a book-sized autonomous car via Raspberry Pi with three on-board cameras coupled with three ultrasonic sensors. In addition to this, the authors designed a circuit and did the engineering related work on the car. Software side was accomplished by developing a Python based library for controlling and communicating with car over a network or locally within the car.

For environment learning we practiced two methodologies; *Supervised learning:* Drove the car on an environment/road and collected 3,000+ data-points, based on this we trained a CNN model which achieved 73% test 89% train accuracy. *Reinforcement learning:* We trained car for three different road signs; Stop, No left, and traffic light using deep Q-learning with existing CNN model.

Figure 1: Demo videos - YouTube

Figure 2: Source code - GitHub

Contents

List of Figures

List of Tables

Chapter 1

Introduction

DrivingMatter, A Raspberry Pi based RC car is autonomous or self-driving vehicle, which have capability to learn its path and environment to achieve the set goals.

An autonomous or self-driving car is a vehicle that must be able to navigate, to certify as autonomous, without human input to a pre-established destination over a path that have not been adapted for its use. It makes its own driving decisions, able to cope with all situations and continuously keep learning from its environment and decisions on that environment to maximize its output over time. A fully automated car is able to handle all driving tasks in all driving mode and under all environmental conditions, just like human drivers.

Autonomous cars development is uprising rapidly in automotive industry. They are increasingly catching attention worldwide because prospective of this technology is clear as it will dramatically change transportation by minimizing traffic jam, increasing efficiency and allowing faster speed. Autonomous cars are predicted to increase traffic flow and lower fuel consumption which will reduce contamination in urban areas by improving driving and significantly reduce needs for parking space. In addition, autonomous cars will speed up people and freight transportation, as well as increase the security, specifically a significant reduction in traffic collisions by reducing the human error.

Since governments, industries, and research institutions are investing vast amounts of both human-time and money, we can make an idea of the interest that autonomous driving is raising and pushing its limits. Fully autonomous vehicles have become a reality, as for example by November 2017, it has revealed that Waymos self-driving cars had driven more than 4 million miles on public roads. The Institute of Electrical and Electronics Engineers (IEEE) predicts that driver-less cars will account for up to 75% of vehicles on the road by the year 2040.[6]

Despite the various benefits to increased vehicle automation, there is a lot more work to be done before self-driving cars are ready for the mainstream. As the self-driving car is a collection of networked computers wirelessly connected to the outside world, the most daunting challenge is keeping the systems safe from intruders i.e. Cyber Security. Similarly disputes concerning liability, driving safely despite unclear lane markings or recognize traffic lights that are not working. Other obstacles could be missing driver experience

in potentially dangerous situations, ethical problems in situations where an autonomous car's software is forced during an unavoidable crash to choose between multiple harmful courses of action, and possibly insufficient adaptation to respond to spoken commands or hand signals from pedestrians or highway safety employees. Autonomous cars today are intelligent enough to drive around the city with very low chances of crashing, that is great but there are some limitation which they are bounded with, is that they are limited to GPS and ground truth of the Earth i.e. Google Maps or similar. If we removed that which is common in mountainous, forest or new area then car is zero and then you have to drive yourself.

In this project, we are focusing on the learning process of a self-driving car. We have specified a road environment and will train the car using deep learning by extracting road features with some real world obstacle and constraints and let the car learn the whole state by it self from zero to pro.

1.1 Background

The history of autonomous cars goes beyond early 1920s when experiments were started being conducting to turn the fantasy of self-driving cars concept into reality. Promising trials took place in 1950s; first self-sufficient and truly autonomous car appeared in 1980s. There has been a strong uprising tendency in the autonomous driving technologies since in 2004 DARPA introduced its first grand challenge. Currently, there are many projects on autonomous car are going on industry level that include Google self-driving car (now Waymo), Tesla, General Motors, Ford, Volkswagen, Audi and Volvo etc., who have begun testing autonomous, or self-driving, vehicles.

There are significant number of projects going on self-driving car. For instance, Udacity is building open source self-driving car which provides with learning models for example many different neural networks are trained to predict steering angle and Robot OS is used as a middle-ware to interact with these models. Theyve also collected over 10 hour of driving data from different real-time traffic scenarios and labeled it to make annotated datasets for training of autonomous vehicles. For detecting cars, pedestrians and traffic lights Yolo models are trained against the annotated Udacity datasets.

1.2 History

This journey of transmutation, fantasy to reality was not as simple as it sounds. The GM Futurama Exhibit, General Motors exhibited the show of self-driving cars. By the 1950s, the company had begun experiments with autonomous cars that drove by the use of receivers that could detect special circuit installed on roadways.[16]

In 1970s introduced the world with Stanford Cart, a buggy equipped with video camera and remote control, which was imbued with greater intelligence overtime.[5]

In 1980s, VaMoR made its autonomous debut and drove on a public highway at speed of about 60mph, by Ernst Dickmanns. The VaMoRs was the world's first real-deal au-

tonomous car. In 1990s, Project Promethues, VaMP and VITA-2 could recognize road markings, its relative position in the lane and presence on other vehicles.[15]

From early 2000s, DARPA has been organizing long distance competitions for autonomous vehicles in different regions which encourage new waves of research and development. [3]

1.3 System Application and Environment

Training the car in the real-time environment like motorway or a road with heavy traffic is a very risky and cost-driven task. So we have created a small Raspberry Pi based car, and a small custom environment of our own so we can do the training without any fear. The environment consist of a map track. We have two kinds of road track, one simply rounded road and one with the forks. We have direction markers, traffic lights and other signs which contributes to the decision of the action which car takes.

This Raspberry Pi based small car and defined specific environment makes our own system. Once the car is trained on this environment, it can easily be scaled to any other bigger environment with an expensive car.

1.4 Purpose of an Autonomous Car

A vehicle must be able to navigate, to certify as autonomous, without human input to a pre-established destination over a path that have not been adapted for its use. It makes its own driving decisions, able to cope with all situations and continuously keep learning from its environment and decisions on that environment to maximize its output over time. A fully automated car is able to handle all driving tasks in all driving mode and under all environmental conditions, just like human drivers.

1.5 Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse.

It have 40 GPIO pins which allow user to include modules such as motors, ultrasonic sensors and cameras.

It have a quiet good specs such as 1.3 Ghz quad-core processor and 1 GB of RAM. This spec allowed us to easily handle camera stream while *Arduino* lack camera feature due the fact of low end spec; 10 Mhz CPU.

Camera's was the main reason due to which we selected Raspberry Pi for the project. It can support up-to 5 cameras: 4 on USB port and one Pi camera.

Chapter 2

Hardware: Engineering the Car

The structure of DrivingMatter is based on different hardware components each performing it's specific task and contributing to the skeleton of the car.

2.1 Equipment

For the construction of the car we need different modules which summed up to get us with a working car. Those modules and hardware parts are discuss in coming section.

2.1.1 General Purpose I/O Pin

GPIO pins on Raspberry Pi enable us to interface physical devices like Motor Drivers, Sensors, and Cameras etc with the Linux processor and allow us to control their input through Python program in Raspbian operating system. We have used RPi.GPIO library to handle interfacing with pins.

There are two different numbering systems for GPIO pins. The BOARD, which uses the pins exactly as they are laid out on Raspberry Pi, and the BCM (Broadcom SoC numbering) which differs for different versions of Raspberry Pi. We have set pins mode to BCM.

2.1.2 Motor Driver

Two **Dual H-Bridge L298** motor drivers are used to control motors. Motors are controlling wheels of car which are connected to Raspberry Pi through motor drivers.(See Appendix B) Motor driver controls the motor by two input pin, one to move forward and another for backward for a single tyre/moto. The abstract representation of the connection of motor driver and motor is as shown in Figure 2.1.

We have used PWM pin of motor driver to control the speed of car. PWM process is used to control the voltage frequency of the source to vary the speed by controlling the enable

pin of the motor driver (Figure 2.2).

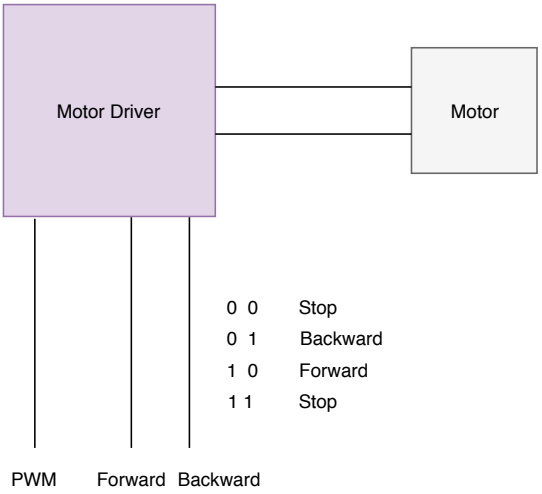


Figure 2.1: Motor Driver connection with Motor

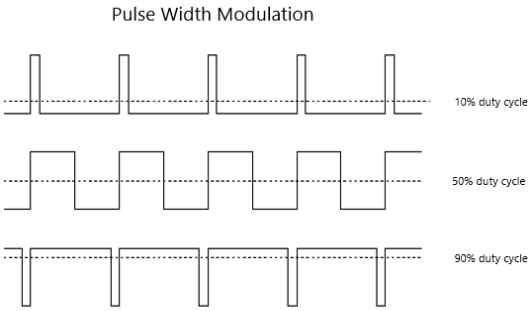


Figure 2.2: Voltage Control Using PWM

2.1.3 Ultrasonic Distance Sensor

Collision detection and distance measurement of upcoming collision is a requisite for an autonomous car. For that, three **HC-SR04** ultrasonic distance sensors (sensorL, sensorC, sensorR) are used in Driving Matter for collision detection in left, center and right direction respectively.

There are four pins on HC-SR04 ultrasonic sensor that are connected to Raspberry Pi GPIO pins. VCC and GND are connected to voltage and ground respectively. An input signal is sent to TRIG pin from one of the GPIO pins, which triggers to send an ultrasonic pulse. The pulse waves bounce off objects in their path and some are bounce back to the sensor. If the sensor detects these return waves, a 5V signal is sent to the ECHO pin. The duration of the pulse is measured and thus distance is calculated from it. (Appendix B.2 for detail)

2.1.4 Camera Module

The eyes of our car, one Pi Camera module and two USB cameras (webcams) are connected to Raspberry Pi's camera and USB ports. With *picamera* library, we can manipulate its settings. It provides a number of effects and other configurations that can be applied.

2.2 Power Management

A 12V (1.2A) battery along with a power bank is attached as a power source which is charged by 5V 2.5A Micro USB charge power adapter. Power is given to motor drivers and Raspberry Pi separately. For testing purpose, when battery is low, 12V transformer is used to supply power to the components.

2.3 Configuration

The abstract representation of the hardware structure of the car is as shown below:

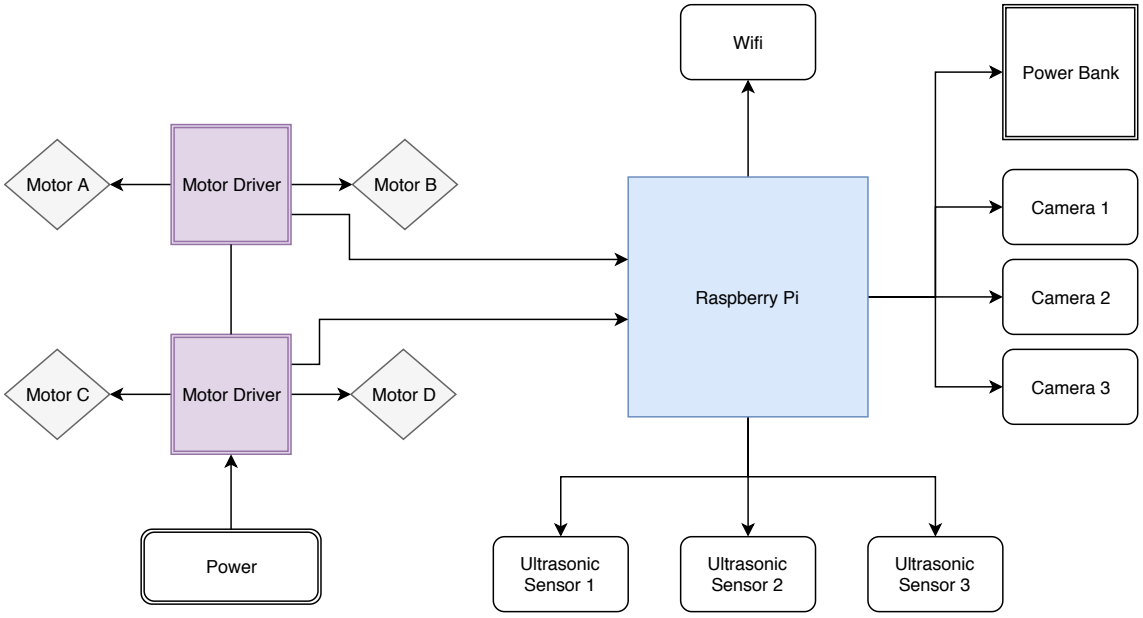


Figure 2.3: Abstract diagram of the hardware structure of the car.

Each of the above device has specific pin configuration.These Pins inputs are controlled through python code in placed in Raspbian using `RPi.GPIO`, a module that controls Raspberry Pi GPIO channels.

For Ultrasonic sensors, Table 2.1 shows pins that are set in GPIO pins:

Sensors	Echo Pin	Trigger Pin
ultrasonic center	21	20
ultrasonic left	6	5
ultrasonic right	16	26

Table 2.1: Pin configuration for Sensors

Similarly, for tyres, controls are given in Table 2.2 :

Tyre	Forward Pin	Backward Pin	PWM Pin
front right	24	25	19
front left	11	9	13
back right	15	14	12
back left	23	17	18

Table 2.2: Pin configuration for Motor Drivers

2.4 Setting up Raspberry Pi

There are few step that were needed to set up Raspberry Pi before we can start working with it. The first step is to install the operating system in Raspberry Pi. We've installed Raspbian through loading NOOBS in microSD card. NOOBS (New Out Of the Box Software) is OS install manager for the Raspberry Pi. (See Appendix A)

Initially, we connected Raspberry Pi to monitor through HDMI cable and other devices through USB ports to complete all configuration. Later, we have introduced automatic Service Discovery, which is explained in later chapter.

Finally, once everything is connected, since the Raspberry Pi does not have a power switch, on plugging the power adapter, it starts up by itself.

When Raspbian boot process is completed, Raspbian home screen appears. Raspberry Pi's configuration is set from Menu- Preferences- Raspberry Pi configuration and then rebooted. (See Appendix A for detail)

2.5 Assembling Car

Two smart car chassis (Acrylic) are used to for the assembling of car. The chassis is accurately designed to fit various devices and compatible boards like Raspberry Pi, Motor Drivers, Bread Board etc. and contains holes to easily mount various sensors and cameras etc. It's capable of mounting 4 motors which are controlling 4 tyres of the car (Figure 2.4).

Sensors are attached on the bottom front of one chassis with motors and motor drivers behind while right above on top front, there are three cameras with Raspberry Pi and bread board behind it (Figure 2.5).

The second chassis is mounted above, two keep all the mess of wires hidden, with the help of copper pillars whose height can be differentiate to change the height between two chassis (Figure 2.6). On the top of second chassis, power bank and batteries are attached for power supply to Raspberry Pi and motor drivers (Figure 4.1).

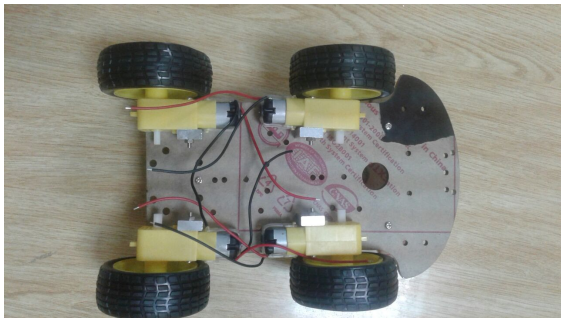


Figure 2.4: Tyres and Motors on bottom chassis

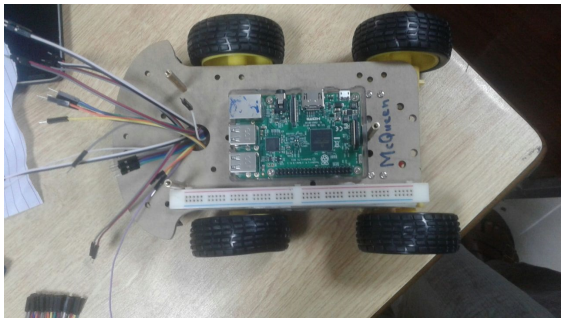


Figure 2.5: Top of bottom chassis

Figure 2.6: Height between chassis

2.6 Possible Issue

We faced many hardware related issues while structuring the car, some of them are very common that could happen very often which are discussed below.

2.6.1 Check which pin configuration is used?

The main thing to remember while connecting all modules to Raspberry Pi is which numbering system of pin are we using. As, for our case, we are using BCM pin configuration, we need to set the GPIO mode to BCM, and all pins then must be assign according to that.

2.6.2 PWM pins in Raspberry Pi

There are already pins set for PWM in Raspberry Pi. Any other general purpose pins cannot be assigned for PWM. PWM are those pin which have higher frequency then other for example we want to switch GPIO pin 100 time in a seconds which 100 Hz this will slow down the whole Raspberry Pi that is why PWM are made available.

Chapter 3

System Analysis and Design

Whenever concepts become complicated and confusing, they need to be disintegrated into smaller problems. Also not only the product, but the development activity needs to be structured to make it feasible when the development of product is complex. In the areas of software development and systems engineering, many of such processes exists. The UML Design of software engineering process has been presents in using Figure 3.1 to 3.4 that help in building a rigorous design of software architecture.

3.1 Activity Diagram

From the Figure 3.1, it can be seen that the car will initiate its sensors and camera after automatic discovery of network (which will be discussed in next chapter) and will take input from PC about action (i.e. forward/backward/forwardRight/forwardLeft etc).The current state of car will be generated after executing the collision avoidance function. If the car detects the collision then the stop action will be executed , otherwise, car will move according to input action and will keep processing the information using its sensors. In either case, state of the car, which will include sensor's values, cameras frames, time span,actions etc, along will be saved in database.

3.2 Use Case Diagram

The Use Case diagram (Figure 3.2) is depicting the main use cases that the DrivingMatter car will perform to execute the proposed concept.

3.3 Sequence Diagram

The sequence diagram (Figure 3.3) helps in designing the verified logical sequence between the different components of the proposed system of car.

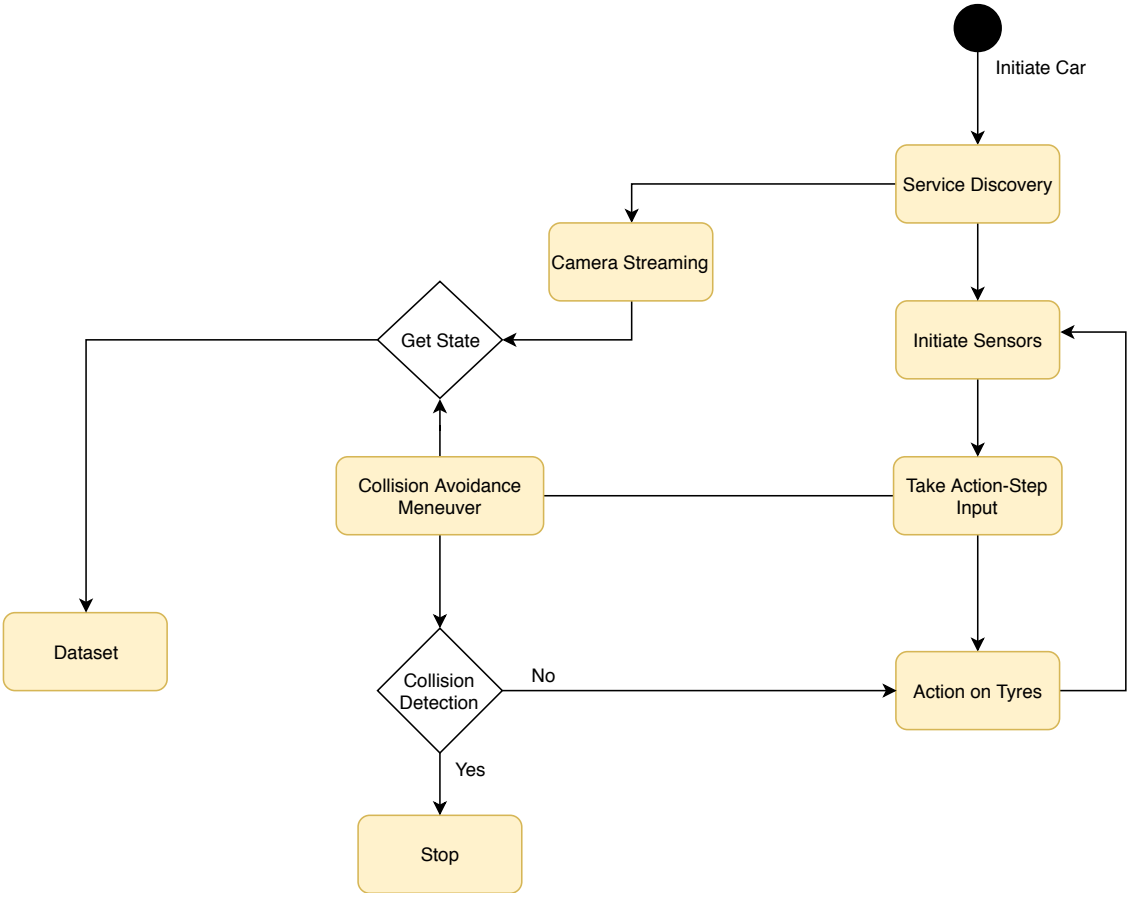


Figure 3.1: Activity diagram depicting software scheme of driving matter

3.4 Class Diagram

The class diagram in Figure 3.4 helps in defining the properties, variables in this case, and functions of the proposed system.

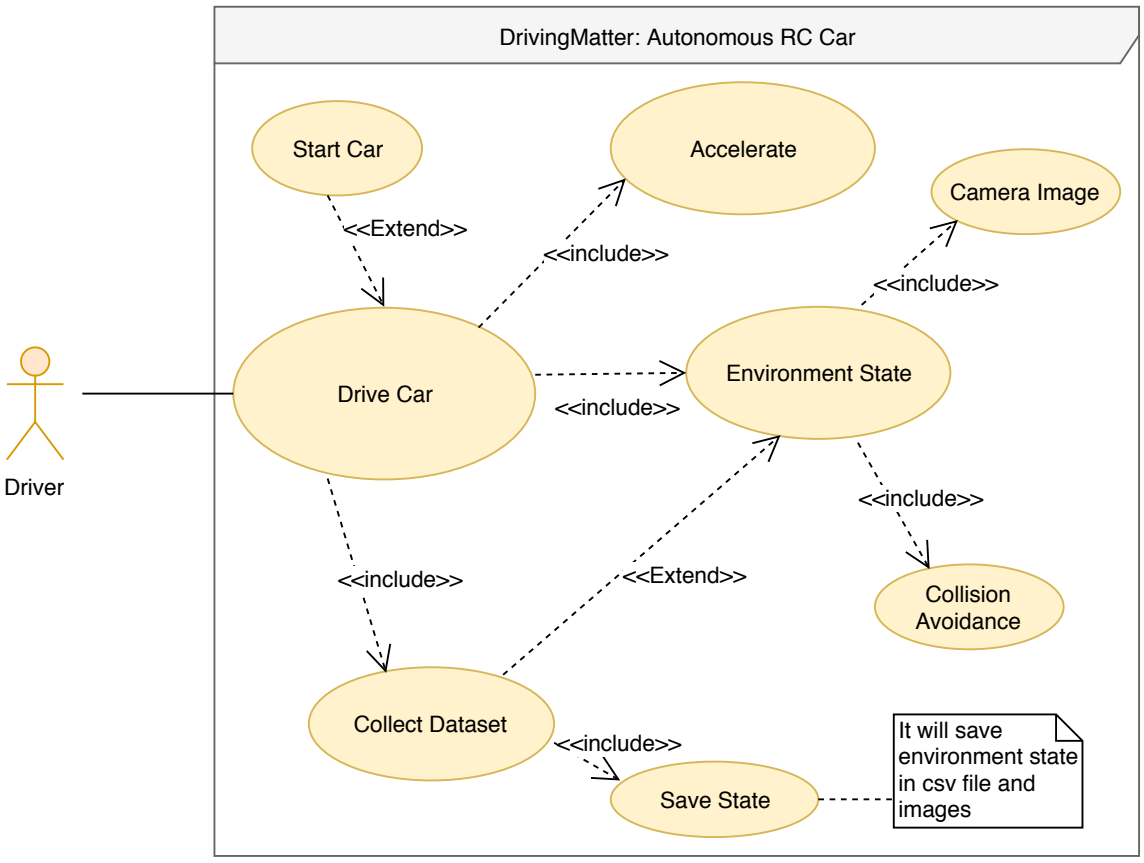


Figure 3.2: Use Case diagram of proposed scheme.

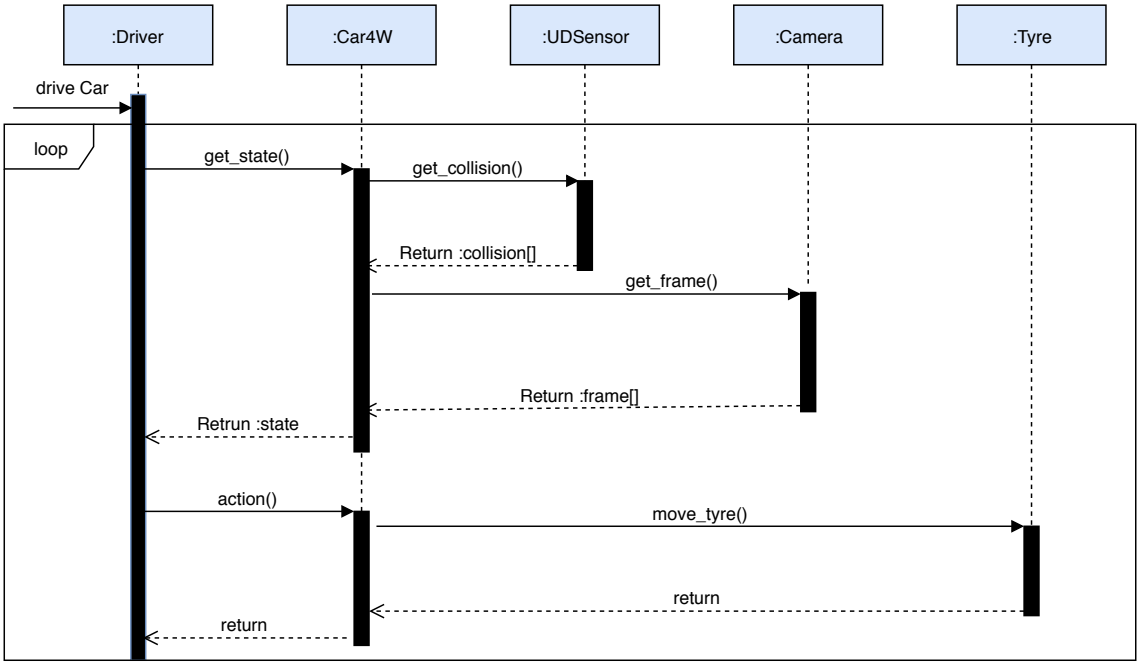


Figure 3.3: Sequence diagram of the car for collision avoidance scheme.

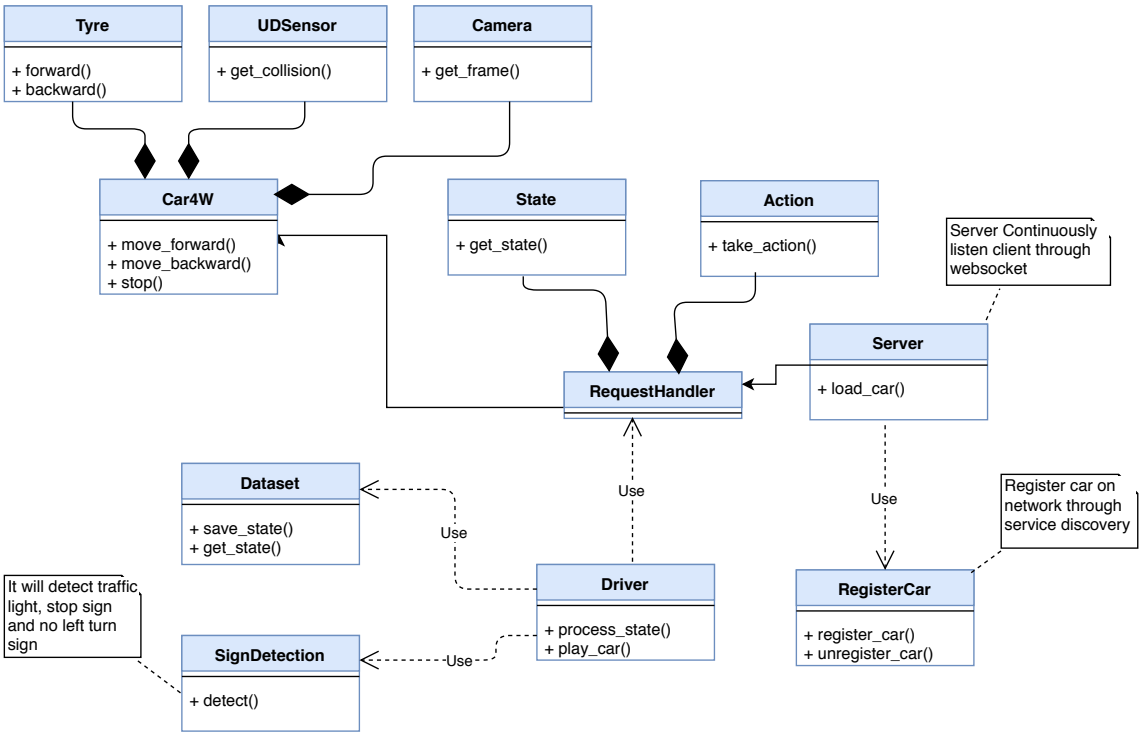


Figure 3.4: Class diagram of the car

Chapter 4

Software: Interfacing

"I think it's fair to say that personal computers have become the most empowering tool we've ever created. They're tools of communication, they're tools of creativity, and they can be shaped by their user."

—Bill Gates

In previous chapter we looked at different hardware components of car chassis of the car, Raspberry Pi, cameras, ultrasonic distance sensors, and battery etc. At the end we assembled them and get the car body. Some images of car body are given in (Figure 4.1).

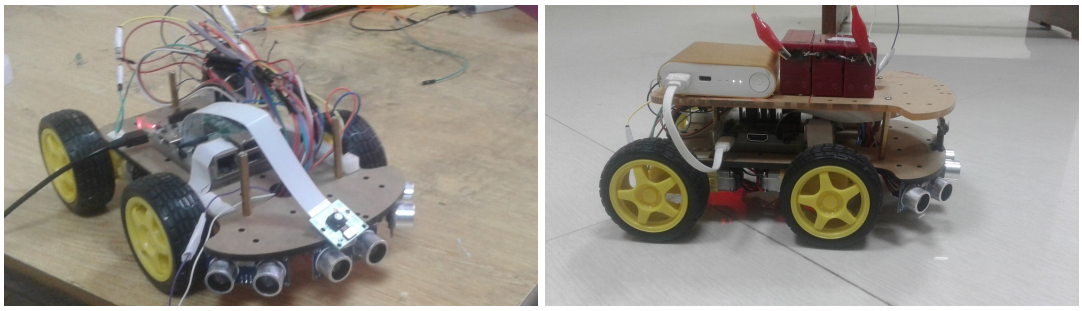


Figure 4.1: Car Body

Raspberry Pi is like a mind of car that is controlling all other components. It give instructions to other components what to do or what not to do. Car can see the real world with its cameras and can sense obstacles with ultrasonic distance sensors. Motor drivers control the movement of car Tyres according to the instructions that raspberry pi gives.

Now there is a question lies how the communication is done between components? The answer is car soul that is actually a software lies in Raspberry Pi. Now lets talk about how we add soul to the car.

4.1 What is Service Discovery?

Service discovery is the automatic detection of devices and services offered by these devices on a computer network. A service discovery protocol (SDP) is a network protocol that helps accomplish service discovery. Service discovery requires a common language to allow software agents to make use of one another's services without the need for continuous user intervention.

There are many service discovery protocols but we are using **Multicast Domain Name System Service Discovery (mDNS-SD)**, a component of Zero Configuration Networking. The **(mDNS)** resolves host names to IP addresses within small networks that do not include a local name server.

4.2 What is WebSocket?

WebSockets provides a persistent, bi-directional, full duplex connection between a client and server that both parties can use to start sending data at any time. A WebSocket connection is established by sending a handshake request from client to a server.

Why we need WebSockets in DrivingMatter: Autonomous RC Car? Answer is: Car is controlled by PC and communication between car and PC is done via Wi-Fi using WebSocket We are using tornado websocket framework for establishing this communication.

```
class EchoWebSocket(WebSocketHandler):
  def open(self):
    print("WebSocket opened")

  def on_message(self, message):
    self.write_message(u"You said: " + message)

  def on_close(self):
    print("WebSocket closed")
```

Listing 1: Server Sample Code (Python)

```
var ws = new WebSocket("ws://localhost/websocket");

ws.onopen = function() {
  ws.send("Hello, world");
};

ws.onmessage = function (e) {
  alert(e.data);
};
```

Listing 2: Client Side Sample Code (JavaScript)

4.3 DrivingMatter Package (Classes)

In object oriented programming (OOP), we tried to find out objects that relates to real world to make software code easy to understand and read. In car there are different components that we have to manage with code i.e tyre, camera, sensor etc so we made class for individual component that handles all functionalities of that component. Here is the big picture of objects interaction in the car

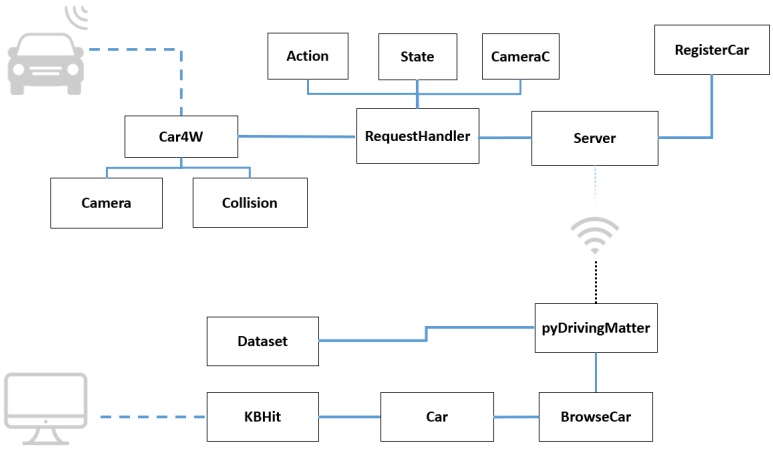


Figure 4.2: Component Interaction Diagram

4.3.1 Camera

Camera class control the streaming of cameras. There is queue which hold the frames extracted from the camera which can be access through Picamera (Pi camera) or OpenCV (for webcam). These frames can be access later on without any delay from this class which is the constraint for the dataset vector.

4.3.2 Ultrasonic Distance Sensor

Ultrasonic Distance Sensor (UDSensor) class is to control sensors. It has function get_distance() that returns the distance of obstacle in front of sensor and we have set a threshold value for collision that tells either sensor have collision or not.

4.3.3 Collision

There are three ultrasonic distance sensors in front of car i.e center, left, right. Collision class control all sensors together. get_collision() function return a list of sensors with collision information.

4.3.4 Sign Detection

This class take the state information of the car and detect the sign (stop, no left and traffic light) and show the information.

4.3.5 Driver

This class take car and object and handle all method related to car driving.

4.3.6 KBHit

This is class is used when controlling car using keyboard. It help getting the keycode of the pressed key which can be linked with driver action.

4.3.7 Tyre

This configure a tyre setting, I have method to move th tyre forward or backward, this is the core functionality which provide us with advance action in car class.

4.3.8 Car4W

This class take four tyre instances, cameras, sensors and provide with some advance action like move left, right, backward and stop.

4.3.9 LoadCar

This class provided with method to automatically load tyre, camera, sensor from config.json and return a car instance.

4.3.10 RegisterCar

RegisterCar class is using Service Discovery protocol to register and unregister car on network. To register car there is a function register_car() and to unregister car there is a function unregister_car().

4.3.11 BrowseCar

On the client side there is class BrowseCar that browse all available cars on the network. For that purpose there is function browse_car() that get information of all available classes.

4.4 Inside Car - Server

Car server is connected with PC via web socket, so it continuously listening on a defined port using IOloop. Whatever message comes from PC it passes to request handler that take actions according to given message.

4.5 Controlling Car from Raspberry Pi - Directly

Car is controlled by PC. Client(PC) send actions to car server using web socket via network like move forward, backward, right and left.

4.5.1 Configuration File

It get confusing when you have to configure code against your hardware requirement. Changes can be spread in to the deep portion of the code. For this we have introduce config.json file which do the changes and follow. We can specify camera information (pi camera or usb), ultrasonic sensors and tyre detail in the file and it will be loaded automatically while execution of the code.
rps_ms: Dataset should be send every 100ms.
timeframe: Collect dataset after every 100ms (0.1s).
port: Websocket server port.
car_speed: Car speed, controlled via PWM pin of motor driver.
sensor_settings: Include echo and trigger pin array of different sensors
sensor_settings: Include echo and trigger pin array of different sensors
tyre_settings: Includes pin settings for four tyres.
camera_settings: Includes camera settings, *camera_type* can have two values; 0 for Pi Camera and 1 for Webcam. *camera_num* is the camera index value when using Webcam.

```
{
    "rps_ms": 100,
    "timeframe": 0.1,
    "port": 8000,
    "car_speed": 30,
    "sensor_settings": {
    "ultrasonic_c": {
        "echo": 27,
        "trigger": 22
    },
    "ultrasonic_l": {
        "echo": 16,
        "trigger": 26
    },
    "ultrasonic_r": {
        "echo": 6,
        "trigger": 5
    }
    },
    "tyre_settings": {
        "front_right": {
            "forwardPin": 24,
            "backwardPin": 25,
            "pwmPin": 19
        },
        "front_left": {
            "forwardPin": 11,
            "backwardPin": 9,
            "pwmPin": 13
        },
```

```
        "back_left": {
            "forwardPin": 15,
            "backwardPin": 14,
            "pwmPin": 12
        },
        "back_right": {
            "forwardPin": 23,
            "backwardPin": 17,
            "pwmPin": 18
        }
    },
    "camera_settings": {
        "camera_c": {
            "camera_type": 1,
            "camera_num": 0,
            "resolution": [160, 120],
            "framerate": 15,
            "rotation": 0
        },
        "camera_l": {
            "camera_type": 1,
            "camera_num": 2,
            "resolution": [160, 120],
            "framerate": 15,
            "rotation": 0
        },
        "camera_r": {
            "camera_type": 1,
            "camera_num": 1,
            "resolution": [160, 120],
            "framerate": 15,
            "rotation": 0
        }
    }
}
```

Listing 3: Configuration file

4.5.2 Code

For controlling the car we have written different classes which we have discussed in earlier session.
In the code below we did the follow steps

1. Load car by setting required settings taken from "config.json".
2. Created Driver instance by provided it with car instance.

3. Once the driver is ready we used *driver.action_nowait(action)* to take action
4. We used *car.get_state_vector* to collect state information
5. We used *driver.display_camera(state)* to display the state information eg. display camera.

Link: https://github.com/drivingmatter/driving-matter-car/blob/master/play/play-fun.py

```
import sys
from os import path
from classes.Driver import Driver
from classes.State import ACTIONS
from classes.Dataset import Dataset
from classes.LoadCar import load_car
from classes.KBhit import KBHit
import logging
from time import sleep

# Setup car and driver
car, rps_ms, port = load_car("../config.json") # Load car data
driver = Driver(car, show_camera = True)

# Take actions
driver.action_nowait("forward")
driver.action_nowait("forwardRight")
driver.action_nowait("backward")
driver.action_nowait("forwardLeft")
driver.action_nowait("stop")

state = car.get_state_vector() # Return camera and sensor data
driver.display_camera(state)

driver.close()
car.close()
kb.set_normal_term()
```

Listing 4: Sample Car Driver Code

4.6 Controlling Car - Network

Car is controlled by PC. Client(PC) send actions to car server using web socket via network like move forward, backward, right and left.
For starting the server following are steps.

1. Clone the "driving-matter-car" (https://github.com/DrivingMatter/driving-matter-car/) repo to the car.

2. Configure the "config.json" according to the car setting.
3. Run main.py file.
4. Server is up and running.

On the PC you will have follow these steps.

1. Clone the "driving-matter" (https://github.com/DrivingMatter/driving-matter/)
2. Open "pyDrivingMatter" directory.
3. Edit main.py and set the IP address.
4. Execute main.py
5. You can now use W (forward), A (left), D (right) and Space (stop) to control the car.

Chapter 5

Environment Learning

5.1 Car Environment

Training the car in the real-time environment like motorway or a road with heavy traffic is a very risky and cost-driven task. So we have created a small Raspberry Pi based car, and a small custom environment of our own so we can do the training without any fear. The environment consist of a map track. We have two kinds of road track, one simply rounded road and one with the forks. We have direction markers, traffic lights and other signs which contributes to the decision of the action which car takes.
This Raspberry Pi based small car and defined specific environment makes our own system. Once the car is trained on this environment, it can easily be scaled to any other bigger environment with an expensive car.
In our case, the car environment is specific that matches the real car environment that contain Road, Traffic Light, Sign Boards and Zebra Crossing. In this specific environment we did experiments to make car autonomous. Some of the images of environment are shown in (Figure 5.1)

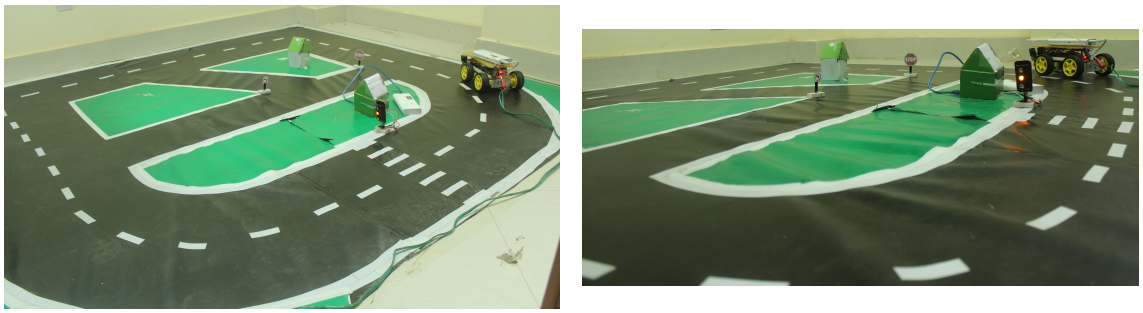


Figure 5.1: Car environment

5.2 Dataset Acquisition

In our case, the external feedback we discussed, is the dataset consist of the knowledge of car's environment state on each action taken by a real driver (cameras images with the action taken on that state), this gives knowledge to the learning function or model to

predict the best action for the new state in future. We have collected dataset by driving car over track. The dataset consists of images, taken from the three cameras of the car which are stored in a folder and dataset.csv file which contains action taken on a state, path to the three images of left, right and center camera stored in images folder, and the time action was taken. So, the basic structure of the dataset is as following:

```
/Dataset-2017-12-01
├── dataset.csv
└── /images
    ├── 1.jpg
    ├── ..
    ├── ..
    └── *.jpg
```

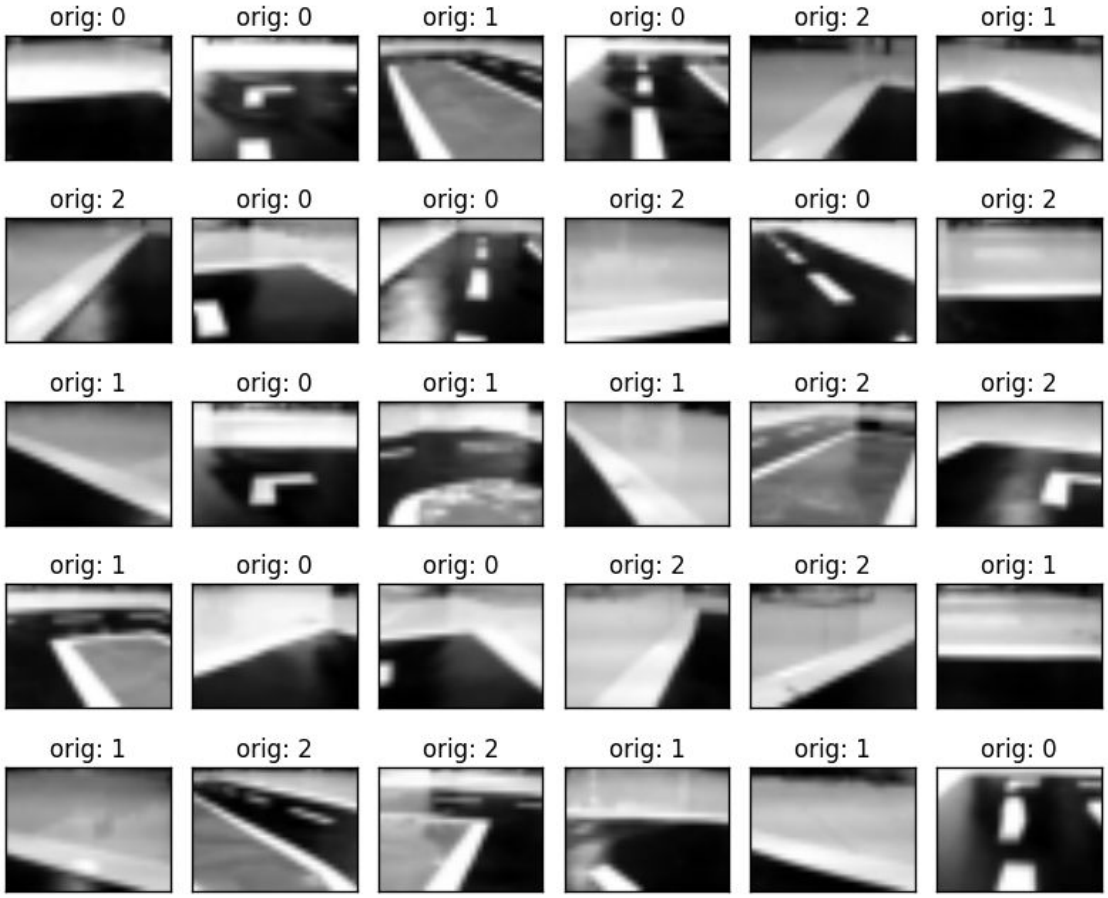


Figure 5.2: Random images from dataset

Fig. 5.2 shows some example of images from our dataset. Actual actions are placed on top of images, 0 is for forward, 1 for left and 2 for right action.
Table 5.1 show the structure of dataset.csv with actions against the images of center camera. The actions, 0 for forward, 1 for left and 2 for right, are stored against images of three cameras (left ,right and center) of each state at certain time along with the images of next state.

state0_camera_c	action	state1_camera_c	time
images/img5.jpg	0	images/img4.jpg	53:43.9
images/img11.jpg	0	images/img10.jpg	53:48.5
images/img17.jpg	2	images/img16.jpg	53:49.6
images/img23.jpg	1	images/img22.jpg	53:51.8
images/img29.jpg	1	images/img28.jpg	53:52.7

Table 5.1: Dataset file (dataset.csv)

5.3 Methodology

To make the car autonomous in environment, Image Processing, Artificial Intelligence and Machine learning techniques are used. There could be different approaches to learn environment either by simple line tracking or model based learning. The two approaches, supervised and reinforcement learning are used for the learning of car to make it autonomous which are discussed in detail in coming section. In both approaches, we focused on deep CNN model. We are interested to know what kind of features CNN extract for decision.

5.3.1 Supervised learning

In supervised learning, the external feedback is used by learning functions to map input to output. This external feedback acts as a teacher for the learning function. We are using Convolutional Neural Network (CNN); one of the deep learning architecture, for the supervised learning. The greatest thing of any deep learning model is extracting feature representations through back-propagation. Unlike other classifiers which need hand-engineered features, CNN knows which information like color or shape is important to do the task. The motivation of using the CNN over other neural network will be discussed later.

5.3.1.1 Image Pre-processing

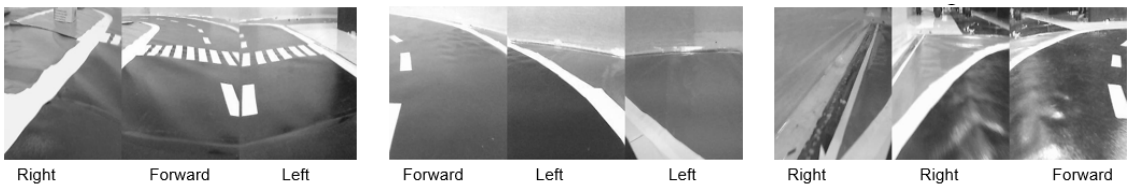
We have total 1025 samples from which training data has shape (820,24,32) because it has 820 training sample images and test data has shape (376,24,32) because it has testing 376 sample images. The length of output class in 3 which are the actions the car can take i.e. include forward, left ad right. The images gathered are in RGB form containing three layer and the size of the images are 320x240.

Resize Each 320x240 image of train and test sets is converted into matrix of size 1x24x32 which is fed to neural network. This resize is done to minimize the learning time of hyper parameters.

Rescaling Data is converted to type float32 before feeding to the network. Also, image pixel values are re-scaled to 0-1 range.

Image equalization Since, we gathered the images by driving the car in real track which could have low contrast places or different illumination conditions which can affect the results. To normalize under different illumination conditions, equalization of image histogram is used. After that, image is smoothed to reduce noise introduced by histogram equalization.

Image labeling The Fig. 5.3 shows the image labeling scheme of for each action over three images. When the car takes the forward action, the center image is labeled as forward, left image is labeled as right and right image is labeled as left (Fig. 5.3a). The described pattern of image labeling is followed because prediction is made on the basis of center camera image, so when a car reaches at a state , where the center camera image is like we have left camera image the appropriate action at that stage would be left. Labeling scheme for left and right actions is shown in (Fig. 5.3b) and (Fig. 5.3c)



(a) forward (b) Left (c) Right

Figure 5.3: Image Labeling

Class labeling Since our data is categorical, we need to convert it to a vector of numbers. Using One-hot encoding, we get a Boolean row vector of 1x3 for each image which consists of all zero values expect for the class it represents. For example, for forward class, it will be [1 0 0].

Class weights For each class, we have different number of samples in dataset e.g. forward class could have 500 while right and left have 400 and 300 etc. In such scenario, we need to assign class_weights to each class and standardize the data so that each class has equal distribution of samples while training.

5.3.1.2 Deep Learning for Autonomous Driving

As we want to predict the best actions, on a given state of the car in environment, we need to classify the state information i.e. cameras images with respect to actions. There are different classification algorithms but we are using Convolutional Neural Networks (CNN) model for our problem.
The motivation of using the CNN over other classification techniques or neural networks is that CNN make efficient use of patterns and structural information in an image. As for instance, in RNN, output dependency on all previous values results very bad for images. Moreover, fewer memory requirements and less parameters. As we are targeting our model to our car with low computing power, CNN is the best bet we can make.

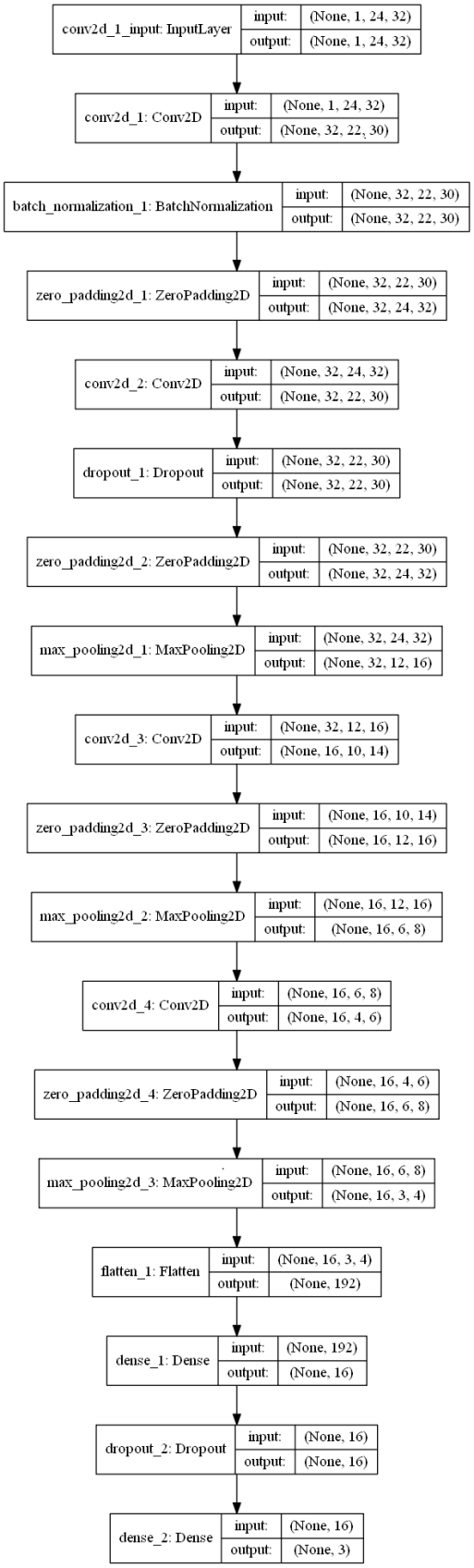


Figure 5.4: CNN Model Architecture

CNN Architecture (DMNet) The best CNN model architecture that fits our data with maximum accuracy is shown in Figure 5.4. We have named it DrivingMatterNet (DMNet). The architecture have 18 layers, of which 4 are convolutional layers. It have around 20,000 parameters. The accuracy graph is shown in fig 5.5. We have got 89% training accuracy and 73% validation accuracy.

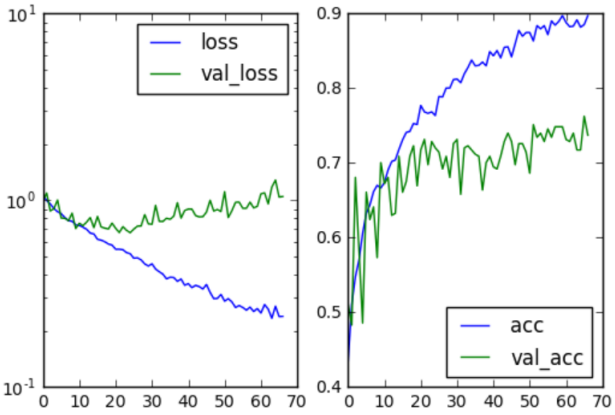


Figure 5.5: Accuracy and Loss on Training and Testing Data Points

5.3.2 Deep Reinforcement learning

Deep reinforcement learning is a mixture of deep learning and reinforcement learning. In reinforcement learning, there is no external feedback or answer key to our problem but agent (the car in our case) still has to decide how to act in a certain situation or task. So, to perform its task, agent learns from its experience. Still, some knowledge is provided about the goodness or badness of an action on a state, known as reward (or punishment in case of negative reward). So unlike supervised learning, reinforcement models employs different dynamics i.e. rewards or punishment to reinforce unprecedented types of knowledge. So the car has a current state s of the environment E, the car can perform an action a which will transition it to next state s'. Each action is evaluated by a reward r. The set of action car takes, define the policy Π and reward it returns, defines the value v. The goal is to select the correct policy Π to maximize the rewards. This just defines our solution in reinforcement learning scenario in Markov Decision Process framework, So we have to maximize

$$E(r_t|\Pi_t, s_t) \tag{5.1}$$

for all possible states s at time t.

5.3.2.1 Deep Q-learning

In Q-Learning Algorithm, a Q-Function $Q(s,a)$ is used to approximate the reward on a state. where Q is a function which calculates the expected future value from state s' and action a. The Q-Function can be described by Bellman Equation as follow:

$$Q(s_t, A) = [r_{t+1} + maxQ(s_{t+1}, A)] \tag{5.2}$$

We are using Q-learning, the policy-based model-free reinforcement learning with a neural network to approximate the reward based on the state, which is also known as Deep Q-Network (DQN).

Neural Network to predict rewards Same CNN model as Fig. 5.4 is used for the simplicity. The training process of model makes the neural net to predict the reward value of each action from a certain state. The action with the maximum reward is performed.
We are setting rewards based on environment information such as specific reward if the car takes appropriate actions on a specific traffic signs, negative reward if car remain stop when theres no stop sign or hurdle, a huge positive award in case of goal state and small positive reward when on each correct step car take. These rewards values will be soon explained in detail.

Exploration - Exploitation The car might find, lets say, a stop sign and continue to take stop action and spend all the time exploiting that discovery by racking up small reward its getting and never go further to find a larger reward of goal state. To overcome this middle exploration-exploitation trade-off is done. This is where epsilon comes in; epsilon is the percent of the time the car will takes a random action rather than the action that is most likely to maximize reward. In the beginning, car takes many random because of higher value of epsilon (0.8) and explores the track. At each step, a decay of 0.99 take place in epsilon which decreases the exploration rate over time and start to exploit the reward information got during exploration.

Discounted Reward We also need to keep the future reward in mind. For example, the prediction of the model could indicate that taking a stop action has the best reward in current state when in fact it can gain more reward if there is a goal state next to stop if car keep moving forward. A loss function that indicates how far our prediction is from the actual target is defined as following:

$$loss = (r + maxQ'(s', a) - Q(s, a))^2 \tag{5.3}$$

where the term $r + maxQ'(s, a')$ represents the target value and $Q(s, a)$ is predicted value. is gamma the defines the discount rate (0.9) which help the model to learn to maximize the discounted future reward based on the given state.
At first, car performs an action a, and get the reward r and resulting new state s'. Based on the result, we calculate the maximum target Q with a discount so that the future reward is worth less than immediate reward. Lastly, we add the current reward to the discounted future reward to get the target value. The difference of current prediction from the target gives the loss.

Memory The memory of the experiences is kept and at each state, model is trained on these previous experiences in memory. These experience contain information of state, action taken, reward and next state.

Traffic Sign Recognition for Rewards A self-driving car should be able to obey the traffic signs so the car needs to recognize the signs. Each traffic sign proposes a different

action to perform with different reward. For example, on a custom stop sign, there should be a positive reward if car takes a stop action and a negative reward if car keeps driving. Traffic signs show a wide range of variations between classes in terms of color, shape, and the presence of pictograms or text. There are thousands types of traffic sign present today, for example even the German Traffic Sign Detection Benchmark (GTSDB) comprises of dataset of over 1200 types of traffic signs. The Fig. 5.6 shows four main categories in GTSDB based on the shape of the sign.

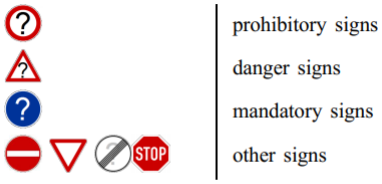


Figure 5.6: GTSDB: Three categories are assigned to the sign on the basis of sign pattern and a fourth category to relevant signs

Sign, detection and recognition of each and every sign is a very long task, we are considering three main sign which are stop, no-left turn and traffic. For detection of these signs, OpenCV provides many efficient and effective methods of detection such as cascade classifiers which we have used. These classifiers are based on voila-jones cascade classifiers based on HAAR features. The main advantage of using cascade classifiers over other OpenCV detection algorithm is it works efficiently in embedded systems with low resources which was very beneficial in our case. Moreover, HAAR feature considers adjacent rectangular features in image instead of computationally expensive image intensities (RGB pixel value at every pixel of image) like in LBP. Since, cascade classifiers are binary classifiers, we have to train three classifier for each class (stop, no-left, traffic-light). For each classifier training, 1800 of positive images (contain the signs in image) are gathered for three classes for positive dataset and a vector file is made from these which contains the path to the image and the coordinates of sign in image. 900 nonsign images are used Negative dataset and *.dat* file is made from it which contains the negative images path. Finally, using information from *.vec* and *.dat* files three cascade classifiers are trained with `opencv_traincascade` function consist of 10 stages with $minHitRatio = 0.998$, $maxFalseAlarm = 0.3$ and sampleheight and samplewidth $(-w, -h)$ of 20. On unseen dataset, `stop_classifier`, `noleft_classsifier` and `trafficlight_classifier` gives 98% , 96% and 91% accuracy respectively.

(a) no-left turn (b) stop sign (c) traffic light

Figure 5.7: Sign detection with OpenCV cascade classifiers

For the recognition of which traffic light in blinking, OpenCV color detection technique is used. Color boundaries for red, green and yellow in BGR are set, the color in boundary is found in image and the light is recognized by the location of the color found. Fig. 5.7 shows a demo of sign detection. The car can detect the sign from a long distance but it must take decision according to sign from a specific distance say 30cm away from sign in our case. So the distance from camera to sign is calculated using Triangular similarity

$$F = P * D/W$$

and

$$D' = F * W/P'$$

where P in sign size in pixel, W is actual width of sign at distance D, F is focal length, P' perceived width and D' is calculated distance from camera to sign.

Reward values Reward for each state is set according to the action taken in presence or absence of different environment factors. For Example, Table 5.2 shows the reward values for action forward and stop in presence or absence of a stop sign. If there is a stop sign detected within a specified distance, and car takes a forward action, it should be punished by negative reward say -0.5 while in case if car stops, it did a good job and should be rewarded with +2 . In a scenario, where there is no stop sign , car should stay put and keep moving forward.

Stop Sign Detected	Action Taken	Reward Value
True	forward	-0.5
	stop	+2
False	forward	+0.05
	stop	-0.5

Table 5.2: Reward Table

Chapter 6

Conclusions and Future Work

"Our imagination is the only limit to what we can hope to have in future."
—Charles F. Kettering

6.1 Conclusion

There are many projects related to self-driving car, in our case we created our own car and environment from scratch which helped in understanding all the back-fall one can face, such as car on map with single light perform different while when 4 light above it i.e. a brighter environment. We also found out that there can not be a single technique to solve the whole problem, its the combination of multiple algorithms together which sums up to give good result.
For our case Supervised learning was done using Convolutional Neural Network and we achieved 73% test and 89% train accuracy. In addition to this we trained car with reinforcement learning for three different sign boards; Stop, No left and Traffic light using deep Q-learning technique with CNN model which we used earlier.

6.2 Future Work

Currently, the training is done in a constrained environment, many factors of real environment can sway the prediction of model. Lightening effects and weather conditions influences the images we get from cameras which can have a great impact on model prediction. We look forward for further improvements so that car can have as good results in real environment as we have in our custom environment. For that decision technique could be done using object and line detection. Moreover we could introduce SLAM for decision making.

Appendix A

Raspberry Pi Setup

A.1 How to get and install NOOBS

Use an SD card with atleast 8GB memory capacity.

1. First thing you need to do for installing NOOBS is to Download NOOBS from the official website. [12]
2. Extract the Zip File and copy files into formatted SD card.
3. Safely remove the SD card and insert it into your Raspberry Pi.

A.2 First Boot

1. Plug in the mouse, keyboard, and monitor cables and USB power cable into Pi.
2. After booting of raspberry pi, A window will appear with a list of different operating systems that you can install.Select Raspbian and click on Install.
3. Installation Process of Raspbian wil began and after completion, Raspberry Pi configuration menu (raspi-config) will load.

A.3 Logging in and accessing the GUI

The default login for Raspbian is username `pi` with the password `raspberry`. To load the GUI, type `startx` and press **Enter**.

Appendix B

Connecting Modules with Raspberry Pi

B.1 Motor Driver

1. Connect IN1 on the L298 to Raspberry Pi pin number 24.
2. Connect IN2 on the L298 to Raspberry Pi pin number 25.
3. Connect the ENA and 12-volt pin to a 12-volt battery.
4. Make sure the -ve and +ve the battery, Raspberry Pi, and L298 are common. This configuration is for a singletyre, repeat same process with pin configuration given in Table 2.1 chapter 2 for other tyres

B.2 Ultrasonic Sensor

There are four pins on the ultrasound sensor module that are connected to the Raspberry, this is the setting for center sensor, repeat the process for other two with given pin configuration in Table 2.2:

1. VCC to Voltage (5v)
2. GND to Ground
3. TRIG to Pin 20
4. connect the 330Ω resistor to ECHO. On its end you connect it to Pin 21 and through a 470Ω resistor you connect it also to Ground.

The GPIO pins only tolerate maximal 3.3V. The connection to GND is to have a obvious signal on Pin 21. If no pulse is sent, the signal is 0, else it will be 1. If there would be no connection to GND, the input would be undefined if no signal is sent (randomly 0 or 1), so ambiguous.

Appendix C

Communicating with Car's Operating System

Once the car is ready considering that circuit is complete and operating system (Raspbian OS) is installed. We know want to control the car from distance. However we have two approaches to communicate with car's OS. We can control car OS with HDMI port by connecting monitor, mouse, and keyboard with the car's Raspberry Pi but this doesn't help us when we want the car to be in moving state.
When we were planning to build the car we took this problem in account and bought Raspberry v3 Model B which come with WiFi modules.
For distance communication we can connect car with a specific WiFi Network (for our case SSID: Research). Doing so, the car automatically connects with network on startup. Once the car is connected we now needs it IP address which can done using following approaches.

1. Get IP address from router's DHCP list

2. Using NMAP or similar Android Application (*Fing* in Figure C.1)

Raspbian come with SSH and VNC server already setup, we will use them for the interaction.

C.1 Secure Shell (SSH)

Using the IP of car we can connect to it but providing that IP, username and password. Default settings are as follow; *User:* pi, *Pass:* raspberry. (Figure C.1)
We can use Bitvise SSH Client (SSH+SFTP), Putty on Windows platform and for Linux platform we can use *ssh* command.
For file transfer when using *ssh* on Linux we can use Gnome File Managers eg. Nemo, Nautilus; SFTP explorer or Bitvise's file transfer for Windows platform. This will allow feasible file transfer of both the platform.

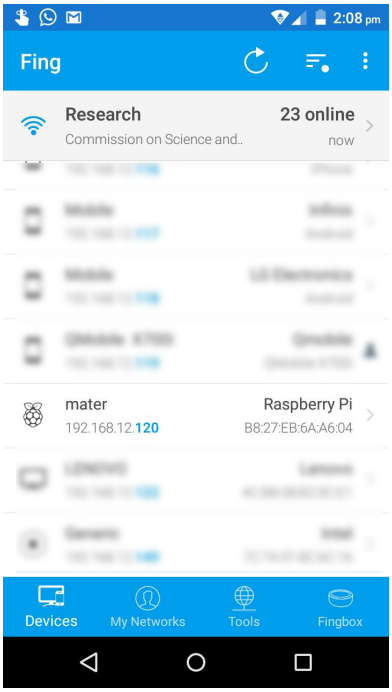


Figure C.1: Fing Android Application - IP Search

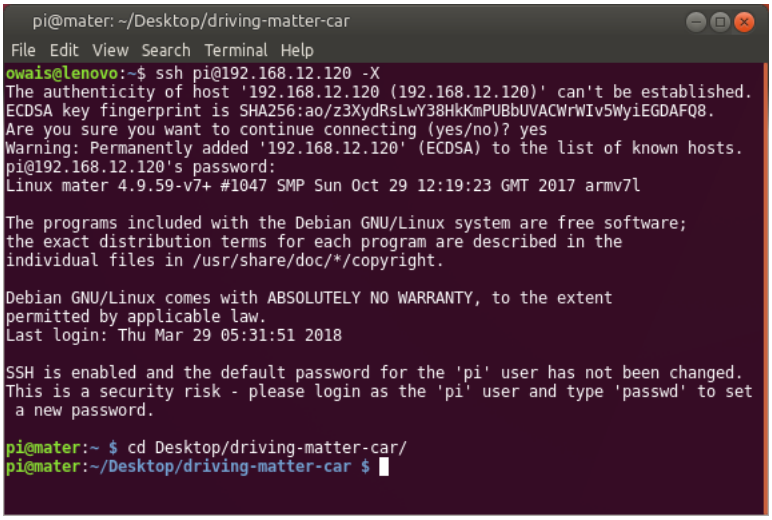


Figure C.2: Connecting with SSH

C.2 VNC Viewer

We can use any VNC viewer to connect to that IP. We suggest RealVNC for the VNC connection. (Figure C.3)

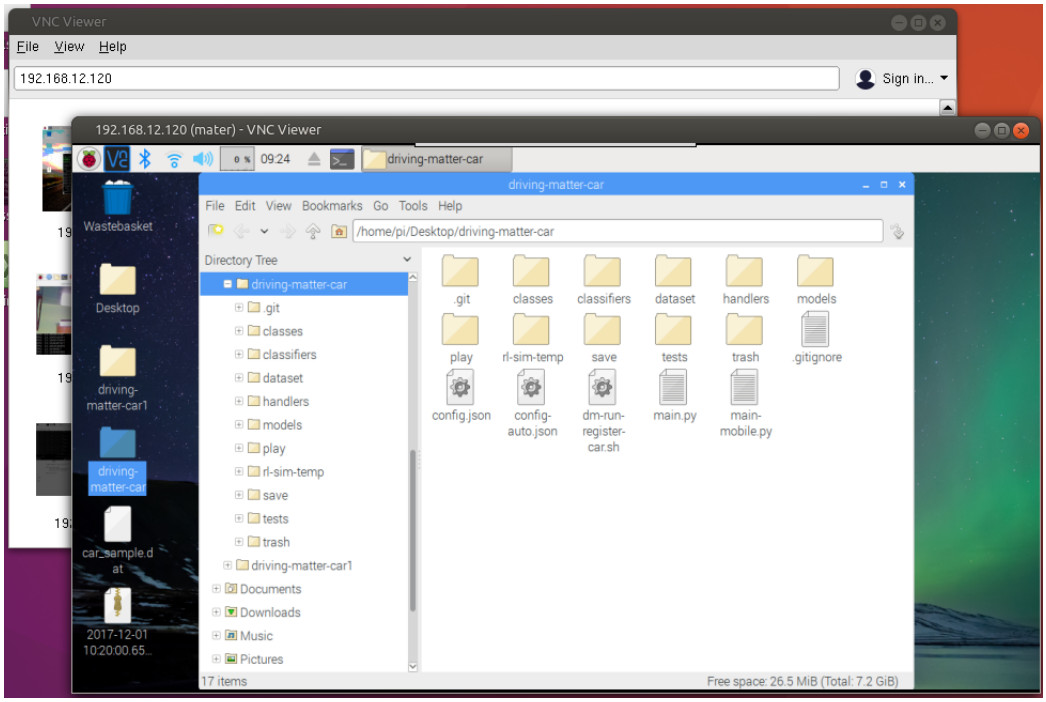


Figure C.3: Connecting with VNC Viewer

Appendix D

Raspberry Pi/Hardware Limitation

While building the car we were caught into issue which are normally the limitation for hardware device. For example if you have 13 mega-pixels camera, but still you can only make video of 2 mega-pixels (1080p). This is due to the fact that camera cannot process 30 frame of size 13 mega-pixels.
Similarly we get stuck with the limitation of the hardware we were using for building the car. Those limitation are as follow.

1. Web-cam and Pi camera's images have different result. We found Pi camera images to have less contrast compared to web-cam
2. Pi camera have dedicated port and can give up-to 30fps.
3. In Raspberry Pi there is *1 bus to handle 4 USB port*, so web-cam can give 7fps at max for resolution of *160x120*. Increasing the resolution impacts the fps.
4. Ultrasonic sensor can give result at rate of 1 per 2 seconds. If we speed this to 1 per second, sensor will become unstable and it will start giving delay of 10+ seconds. This was the reason we didn't consider ultrasonic sensor for our project.
5. Size of dataset is around *900 KB* which is not ideal to transfer over WiFi for real time. To tackle this we kept all the processing on the Raspberry Pi.
6. Using three camera on Raspberry Pi, heat up circuit after 3-5 minutes of streaming. Placing a fan on circuit can increase the streaming time.

References

[1] Autoencoders, "Building autoencoders in keras," 2012. [Online]. Available: https://blog.keras.io/building-autoencoders-in-keras.html

[2] T. Ball, "Train your own opencv haar classifier," 2013. [Online]. Available: http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html

[3] DARPA, "The darpa grand challenge," 2014. [Online]. Available: https://www.darpa.mil/news-events/2014-03-13

[4] A. Deshpande, "The 9 deep learning papers you need to know about," 2016. [Online]. Available: https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

[5] L. Earnest, "Stanford cart," 2012. [Online]. Available: https://web.stanford.edu/~learnest/cart.htm

[6] M. J. Focazio, "Self-driving cars could account for 75% of all vehicles on the roads by 2040," 2012. [Online]. Available: http://co.water.usgs.gov/trace/pubs/wrir-99-4279/

[7] N. GHABOOSI, "Visualizing outputs cnn model training phase," 2017. [Online]. Available: http://www.ir.com/blog/visualizing-outputs-cnn-model-training-phase

[8] S. Guennouni, A. Ahaitouf, and A. Mansouri, "A comparative study of multiple object detection using haar-like feature selection and local binary patterns in several platforms," *Modelling and Simulation in Engineering*, vol. 2015, p. 17, 2015.

[9] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The german traffic sign detection benchmark," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013, pp. 1–8.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[11] S. Mallat, "Understanding deep convolutional networks," *Phil. Trans. R. Soc. A*, vol. 374, no. 2065, p. 20150203, 2016.

[12] Raspberry, "Raspberry pi foundation noobs," 2017. [Online]. Available: https://www.raspberrypi.org/downloads/

[13] R. Ruizendaal, "Deep learning: Convolutional neural networks," 2017. [Online]. Available: https://towardsdatascience.com/deep-learning-2-f81ebe632d5c

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[15] W. Thiel, "The vamors was the world's first real-deal autonomous car," 2017. [Online]. Available: https://www.web2carz.com/autos/car-tech/6396/the-vamors-was-the-worlds-first-real-deal-autono

[16] T. Vanderbilt, "Autonomous cars through the ages," 2012. [Online]. Available: https://www.wired.com/2012/02/autonomous-vehicle-history/

[17] Yosinski, "Deep visualization toolbox," 2015. [Online]. Available: https://github.com/yosinski/deep-visualization-toolbox