CAPSTONE PROJECT REPORT

(Project Term January - April, 2018)

Self-Driving RC Car

Submitted by

Deepak Kajla **Preetam**

Course Code: CSE445

Under the Guidance of

Preeti Gupta, Assistant Professor

School of Computer Science and Engineering

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

In 1769, first steam-powered automobile capable of human transportation was built. Since then, a lot have changed in the way automobiles work. Today, almost all motor vehicles work on fossil fuels. This is changing too, with the emergence of electric vehicles that are cheaper and don't pollute the environment. However, what hasn't changed since the horse drawn carts to these modern electric vehicles today, is that they all have a human driver in control of their actions. Only military applications see autonomous navigation capabilities, but they are not willing to make the technology public. But now big corporations have begun investing vast amounts of money to make driverless cars commercially available. In this project, we try to make our own self driving car using an RC Car, Raspberry Pi and camera module, Arduino, ultrasonic sensor and machine learning algorithms (artificial neural networks) in order to figure out how viable they are in real life scenarios.

1. INTRODUCTION

1.1 BACKGROUND

Driverless or autonomous transportation is a field of study to make vehicles capable of navigating roads to reach the target location in a safe manner, i.e., by following traffic rules and avoiding collisions with obstacles. This approach to handover control from humans to machines is not only a matter of comfort and luxury, but its primary motive is safety. As we take away the human factor from calculation, the human error is also removed. Now, it is only a matter of mechanical maintenance, which is still required in manually driven vehicles. According to Association for Safe International Road Travel (ASIRT), nearly 1.3 million people die in road accidents and approximately 20-50 million people get injured every year. What this means is that road travel is currently the most dangerous form of transportation. Removing the human factor would not only improve safety of road travel, but greatly improve the efficiency. Computers are very good at following instructions and solving mathematical problems with a high degree of accuracy and speed. Thus, autonomous vehicles will be more effective in navigating junctions and crossroads without stopping or huge changes in speed. This will save fuel and time. It will also make traffic and traffic lights a thing of the past. As these vehicles will either make decisions independently or via a hive mind, they eventually won’t need traffic signals to act on the road.

1.2 PROFILE OF THE PROBLEM

Through this project we try to tackle the problem of creating a commercially viable selfdriving car. To figure this out, we will need to convert a commonly available car into an autonomous vehicle. But first we must be able to answer these questions.

- What makes a vehicle autonomous?
- Is it possible to convert a regular vehicle into an autonomous vehicle? If yes, then what changes are required?
- How viable is the approach to make these autonomous vehicles?

1.3 SCOPE

A few points regarding the limitations that this project may suffer from.

- The aim here is not to develop a completely autonomous vehicle but to only test some fundamental cases.
- The project does not take place in a perfectly controlled environment.
- An RC Car does not closely model a commercial-grade car, but it is much easier to work with and sufficiently resembles the original vehicle in terms of our end goal.
- The RC Car being light and made of plastic suffers from slippage on tile floors such as the one used in this test.
- Resources are limited to free and open source versions of software which may have limited functionality compared to the commercial versions.

2. LITERATURE REVIEW

2.1 AUTONOMOUS DRIVING

2.1.1 LEVELS OF DRIVING AUTOMATION

A classification system published in 2014 by the Society of Automotive Engineers (SEA International), Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems, states 6 levels of automation ranging from fully manual to fully automated.

- Level 0: Automated warnings but no vehicle control
- Level 1 (hands on): Driver and automated system share control. Ex: Adaptive Cruise Control
- Level 2 (hands off): The driver monitors the driving and may need to intervene while automated system has full control of vehicle including acceleration, braking and steering.
- Level 3 (eyes off): The driver may perform some other task but must be prepared to intervene when alerted in advance. The automated system meanwhile has full control including responses like emergency braking.
- Level 4 (mind off): Similar to level 3 but driver is not required to be attentive and is not required for any intervention inside a limited region, like a traffic jam. When the vehicle leaves this region, it is able to safely abort the trip by parking itself but cannot drive to location on its own.
- Level 5 (steering wheel optional): No human intervention required

2.1.2 EXISTING METHODS

Autonomous driving can be implemented in a number of ways, but they can be grouped into 2 categories based on what determines the vehicles output. One of the approach is to separately perform the sensing, interpretation and decision-making tasks. This approach was widely used in the DARPA Grand Challenge (2007). Radar, lidar, camera and GPS were the most used sensors used to scan environment, avoid obstacles and provide localization. Camera was used for edge detection and locate road and lane markings.
Based on this input, a decision was made by the system.

A different approach is to directly map sensor input to vehicle control using machine learning algorithms. DA. Pomerleau worked on a similar approach which took an input image and a laser range finder as input to the three to four hidden layer neural networks and had steering as output. The picture was 30x32 pixels and the car's top speed was 90 km/h. The training data came from recording a human driver and pairing an image of the road ahead with the drivers steering at the time. This system seemed to function well under the specific conditions it was trained for but also faced multiple issues. Homogeneous training data made the system less versatile for different scenarios. System was also not very fault tolerant if the driver input was mistake. This approach is also known as end-to-end approach.

2.2 ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANNs) are interconnected computing systems that try to mimic biological neural networks present in animal brains. Machine learning (ML) implements feed-forward artificial neural networks or multi-layer perceptrons (MLP), the most commonly used type of neural networks. A general model of an ANN is shown in Figure 2.1. Commonly, the connection between artificial neurons have a real number value and the output of each neuron is calculated by a non-linear function of their inputs. These neurons and their connections have a weight which improves with the learning process.

The ANNs are organized in layers where the signals pass from input layer to the output layer.

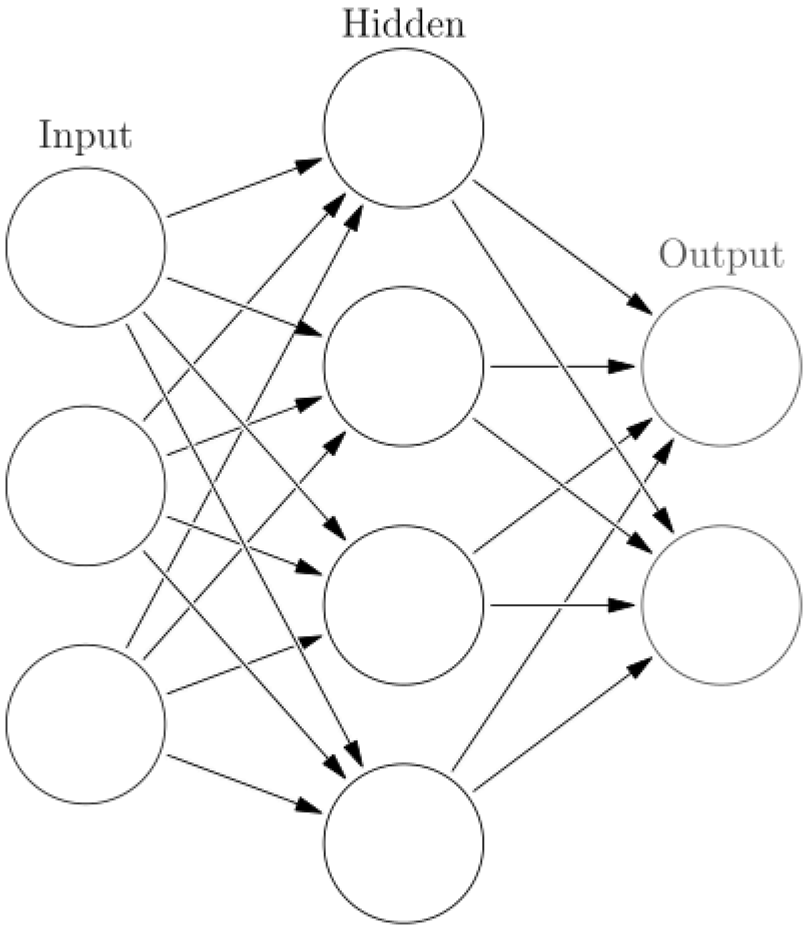


Figure 2.1: Structure of Artificial Neural Network with one hidden layer

To train our ANN, we used a machine learning technique called Backpropagation. Backpropagation requires a known, desired output for each input value, therefore it is called a supervised learning algorithm. The goal of any supervised learning technique is to find a function that best maps a set of inputs to their target output.

2.3 DISTANCE DETECTION USING MONOCULAR VISION

A Raspberry Pi supports only one camera module effectively. So, to find the distance of objects detected by the camera, this project made use of a study by Chu Jiangwei, Ji Lisheng, Guo Lie, Libibing and Wang Rongben to calculate distances in monocular image.

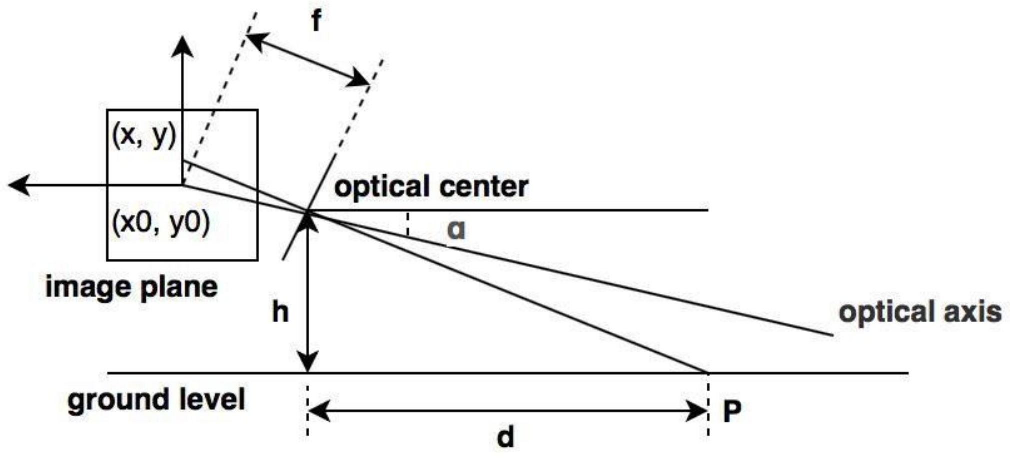


Figure 2.2: The geometry model of detecting distance in monocular image

In Figure 2.2, *P* represents any point on the object, *h* is the height of optical center of the camera, *α* is the tilt angle of camera, *f* is the focal length of camera, *d* is the distance between optical center and target object, (x0, y0) refers to the intersection point of image plane and optical axis and (x, y) refers to projection of point P on the image plane. Suppose O’ (u0, v0) is the point of intersection of optical axis and image plane and the dimensions of a pixel on the image plane are dx and dy. The study gives the following formulas using this model.

$$d = h/\tan(\alpha + \arctan((y - y_0)/f)) \qquad (1)$$

$$u = (x/dx) + u_0, \qquad v = (y/dy) + v_0 \qquad (2)$$

Let $x_0 = y_0 = 0$, using (1) and (2):

$$d = h/\tan(\alpha + \arctan((v - v_0)/a_y)), \qquad (a_y = f/dy) \qquad (3)$$

In formula (3), *h* and *α* are known parameters from when camera is installed. The *dx*, *dy*, *u0* and *v0* are the interior parameters of the camera. OpenCV provides functionality for camera calibration which returns the camera matrix containing these values.

$$\begin{bmatrix} a_x=316.31800735 & 0 & u_0=161.87488041 \\ 0 & a_y=317.82640456 & v_0=113.59339963 \\ 0 & 0 & 1 \end{bmatrix}$$

The matrix returns values in pixels and *h* is measured in centimeters. By applying formula (3), the physical distance *d* is calculated in centimeters. Camera calibration process will be explained in detail later in the report.

3. PROBLEM ANALYSIS

The objective of problem analysis is to get a deeper understanding of problems before development process begins. There are five stages to problem analysis:
Stage 1: Agreeing on the problems and define them
Stage 2: Identifying underlying faults that cause the problem
Stage 3: Identifying the target demographic and other stakeholders
Stage 4: Defining the system boundaries and edge cases
Stage 5: Identifying system constraints

3.1 PRODUCT DEFINITION

Our project on 'Self-Driving RC Car' aims to convert a regular remote-controlled car into an intelligent system capable of navigating dynamic tracks in a slightly controlled environment. We are using sensors like Pi Camera and HC-SR04 Ultrasonic Range sensor interfaced through the Raspberry Pi Zero W mounted on the car to relay information about environment surrounding the car. This information is sent to the Laptop (processing unit) via Wi-Fi network. The laptop processes the image inputs through already trained neural networks, sends an output to the Arduino Uno board via the serial port. Arduino board signals the remote controller so that the car can move in the most suitable manner.

3.2 FEASIBILITY ANALYSIS

Parts required for our project are remote-controlled car with a controller, Arduino Uno unit, Raspberry Pi (any version is suitable as we used the smallest unit, Zero W) with Camera module and Ultrasonic sensor (HC-SR04). All these components are not only widely available globally but also very cheap. Working knowledge is available widely through the open source documentations and several tutorials online.
The research methodology used was:

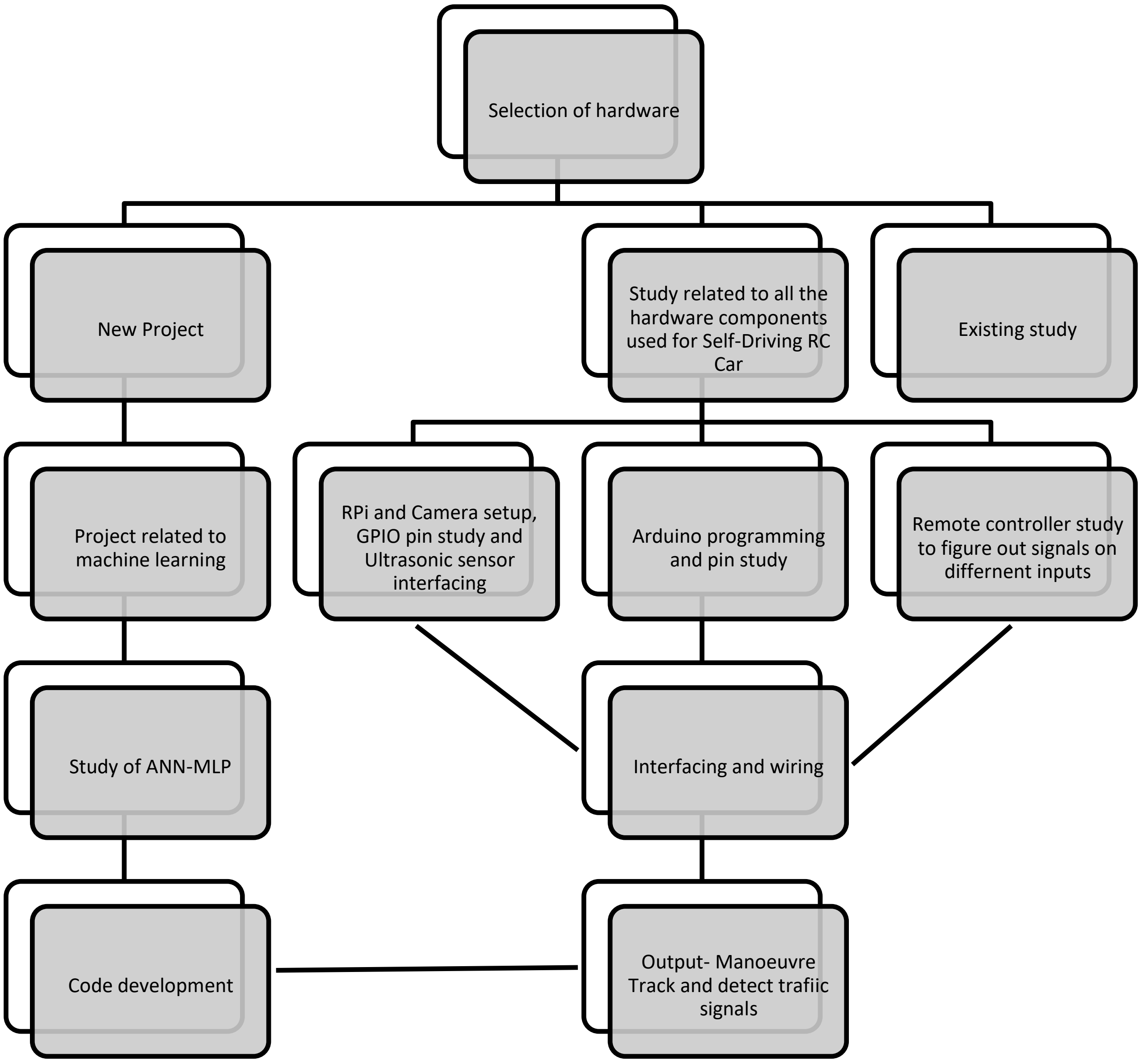
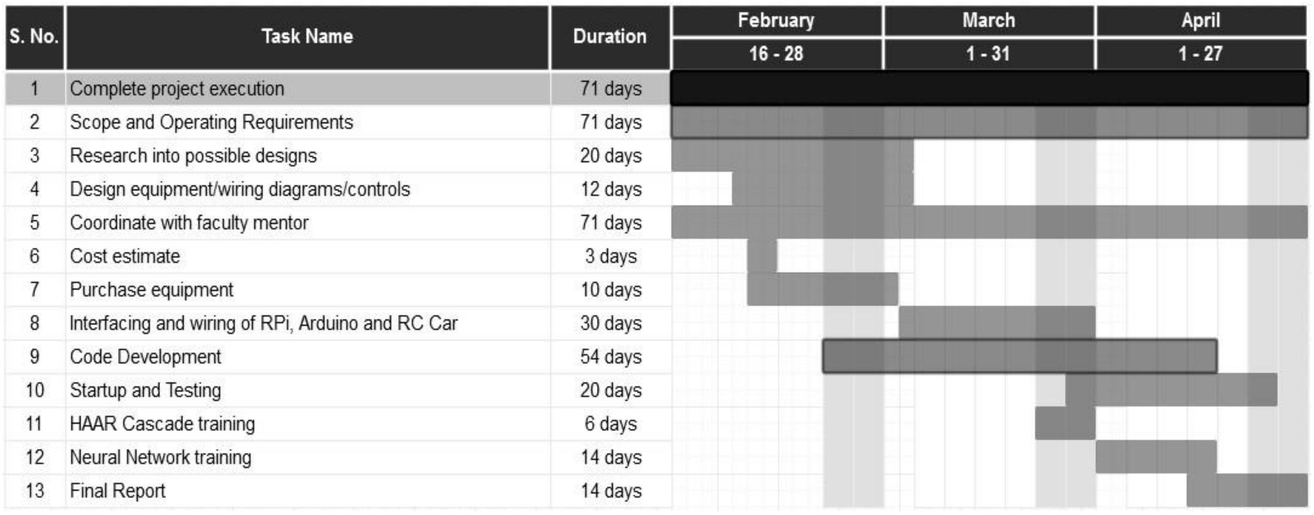

Selection of hardware
New Project
Study related to all the hardware components used for Self-Driving RC Car
Existing study
Project related to machine learning
RPi and Camera setup, GPIO pin study and Ultrasonic sensor interfacing
Arduino programming and pin study
Remote controller study to figure out signals on differnent inputs
Study of ANN-MLP
Interfacing and wiring
Code development
Output- Manoeuvre Track and detect trafiic signals

Figure 3.1: Research Methodology

3.3 PROJECT PLAN

Time and task management were kept in high priority so that the project can be finished within given deadline. Gantt charts are a standard to represent the progress of any project over the total duration of time. They are easy to understand and provide critical data at quick glance. The following Gantt chart displays the progress of over project from 16 February 2018 to 27 April 2018.



S. No.	Task Name	Duration	February	March	April
			16 - 28	1 - 31	1 - 27
1	Complete project execution	71 days	▨	▨	▨
2	Scope and Operating Requirements	71 days	▨	▨	▨
3	Research into possible designs	20 days	▨	▨	
4	Design equipment/wiring diagrams/controls	12 days	▨	▨	
5	Coordinate with faculty mentor	71 days	▨	▨	▨
6	Cost estimate	3 days	▨		
7	Purchase equipment	10 days	▨	▨	
8	Interfacing and wiring of RPi, Arduino and RC Car	30 days		▨	
9	Code Development	54 days	▨	▨	▨
10	Startup and Testing	20 days		▨	▨
11	HAAR Cascade training	6 days		▨	
12	Neural Network training	14 days			▨
13	Final Report	14 days			▨

Figure 3.2: Gantt Chart showing project progress over 71 days

4. SOFTWARE REQUIREMENT ANALYSIS

Requirement analysis in software engineering consists of tasks to determine the needs and conditions of a project. These needs are usually characterized by consumer demands and expectations from the project. Requirement analysis also considers, any conflicts between requirements of different stakeholders. All this information is then analyzed, documented and managed for reference throughout the development process.
This analysis is critical to the development process and may be the difference between the success and failure of any project.

4.1 GENERAL DESCRIPTION

All the code working on Laptop and Raspberry Pi are written in Python 2.7 for compatibility reasons with Raspbian Lite operating system and bundled libraries. Arduino Uno is burned with an '.ino' script using Arduino utility. This script transfers laptop commands to remote controller. Real VNC server was setup between laptop and Pi to transfer files such as codes and images captured for training Haar Cascades.

4.1.1 OPEN CV LIBRARY

We are using the latest OpenCV library with version 3.4.1. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. We used OpenCV as it is bundled with powerful image processing utilities and statistical machine learning library for artificial neural networks. OpenCV was used to generate positive samples and then train our Haar cascades for Stop sign and Traffic light. OpenCV was also used for image processing from the Raspberry Pi camera stream, calibrate the Pi Camera and then finally train our Backpropagation neural network to drive the car. It is a huge open source library with extensive documentation present online.

4.1.2 NUMPY LIBRARY

We are using the latest NumPy library version 1.14.2. The official website states that NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. Since we are dealing with large amount for images and these images are converted to 2-Dimensional arrays to process them in code, we use NumPy to handle these large arrays.

4.1.3 SKLEARN LIBRARY (SCIKIT-LEARN)

We are using the latest sklearn library version 0.19.1. It's a machine learning library for python with simple and efficient tools for data mining and data analysis. It is built on top of NumPy, SciPy and matplotlib python libraries. We used this to after data collection in our project to split the entire dataset into 2 parts: 80% training data and 20% test data. This was done so that the effectiveness of training can be determined after completion.

4.2 SPECIFIC REQUIREMENTS

This section contains all the functional and quality requirements of the system. It gives a detailed description of the system and all its features.

4.2.1 FUNCTIONAL REQUIREMENTS

Controlling Car from PC

Input: User sends input to PC by pressing arrow keys (up, down, left, right) Processing: PC sends signals to Arduino through serial connections.

Output: Arduino activates remote by sending LOW/HIGH signals.

Streaming camera input to PC

Input: Camera signals are taken as input which is sent to Raspberry Pi.

Processing: Raspberry Pi converts these signals to image stream which is send to PC at 10 frames per second.

Output: PC starts showing videos of the input received by camera.

Streaming ultrasonic input to PC

Input: Ultrasonic sensor sends the time taken by the reflected waves to Raspberry Pi.

Processing: Speed of sound and time taken by signals are used to calculate the distance by Raspberry Pi.

Output: PC will start showing the calculated distance sent by Raspberry Pi over Wi-Fi.

Object detection by Haar cascade

Input: Image stream from Raspberry Pi is given to PC.

Processing: Haar cascade is used for object detection in the images.

Output: Boundary will be shown on the image around the object.

Responding to traffic signals

Input: Image stream from Raspberry Pi is given to PC.

Processing: Haar cascade is used for processing of the images.

Output: Car is controlled by PC according to the object detected. It will wait for 5 seconds for stop signal, will move or keep moving for green light and finally if it detects red light it will wait until green light.

Maneuver a track autonomously

Input: Image stream from Raspberry Pi is given to PC.

Processing: Neural networks are used for processing of the images.

Output: Based on predictions given by PC, Arduino signals the car to move.

5. DESIGN

5.1 SYSTEM ARCHITECTURE

Our project consists of a Remote-Controlled car and three different computing nodes: A Raspberry Pi Zero W, Arduino Uno board and a Personal Computer.

We are using sensors like Pi Camera and HC-SR04 Ultrasonic Range sensor interfaced through the Raspberry Pi Zero W mounted on the car to relay information about environment surrounding the car. This information is sent to the Laptop (processing unit) via Wi-Fi network. The laptop transforms the incoming sensor data and after processing the inputs through already trained neural networks, sends an output to the Arduino Uno board via the serial port. Arduino board signals the remote controller so that the car can move in the most suitable manner.

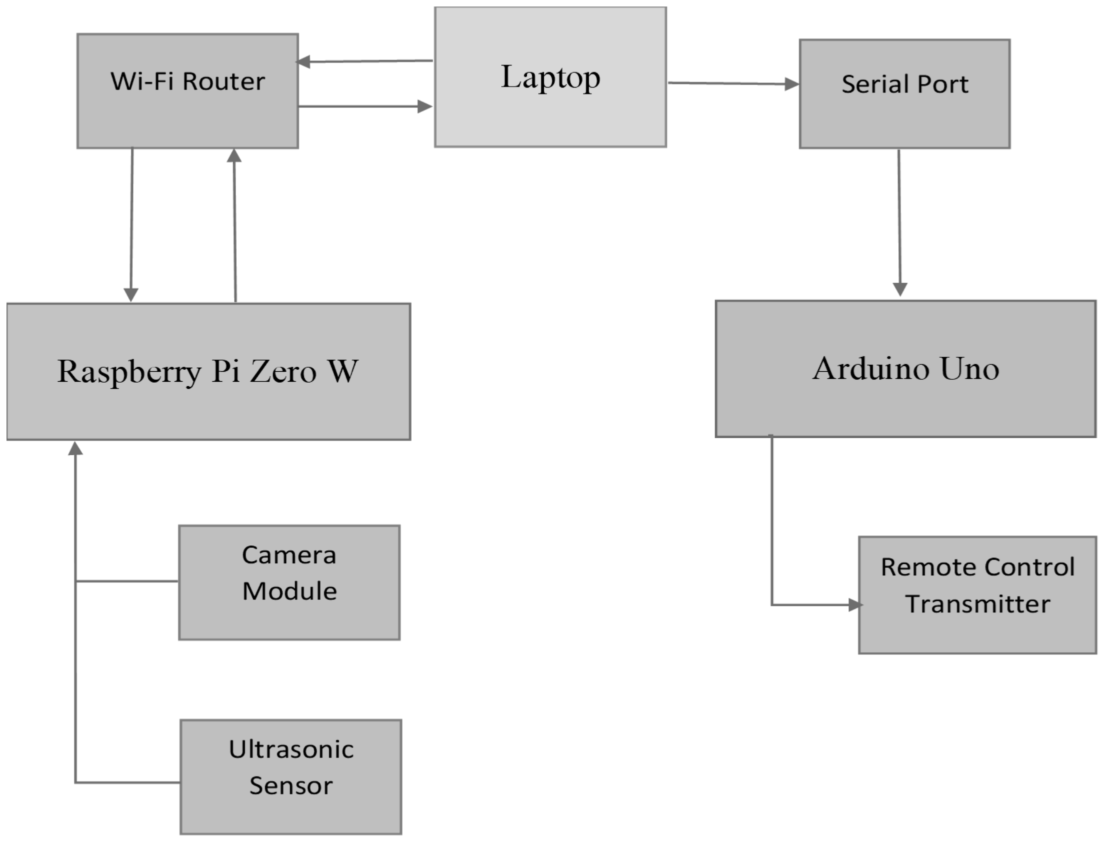


Figure 5.1: Self-driving RC Car architecture

5.2 DETAILED DESIGN

Our project consists of a Remote-Controlled car and three different computing nodes: A Raspberry Pi Zero W, Arduino Uno board and a Personal Computer.

5.2.1 RC CAR

We use a toy remote-controlled car from a small name manufacturer. It is a rear-wheel drive system using a DC motor and front wheel navigation using a servo. The car is powered by four AA 1.2V Ni-Cd rechargeable batteries. The batteries take around 5 hours to fully charge and are depleted on 40 minutes of use. We added a cardboard platform on the car to secure a Raspberry Pi Zero with camera module and ultrasonic sensor on top. On the back, a power bank is held loosely to power the Pi unit.

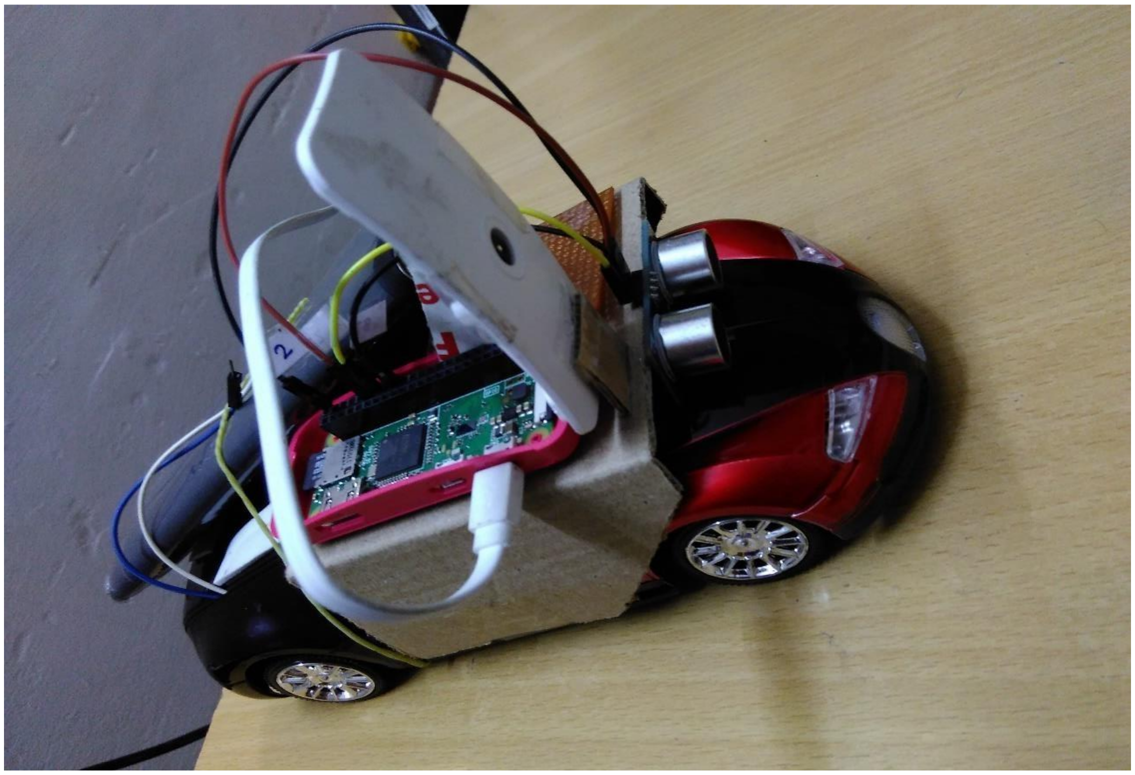


Figure 5.2: RC Car with RPi and sensors

5.2.2 PERSONAL COMPUTER

Training of neural networks and their usage requires extensive processing power and memory. Thus, a portable laptop with medium range specifications was chosen for this task.

The PC receives the track image bundles with the car state from Raspberry Pi as inputs. This is processed by the neural networks to generate a steering prediction. The output is transferred to the Arduino Uno board which signals the remote control accordingly. A video stream shows the camera input during the process.

5.2.3 RASPBERRY PI ZERO W

Raspberry Pi Zero is the smallest of the Raspberry Pi series. The Zero W version includes wireless communication capabilities like Wi-Fi and Bluetooth. It has very low power consumption but still provides multiple GPIO ports and a dedicated CSI (camera serial interface) port. The complete specification of Pi Zero W are:

- 1GHz, single-core CPU
- 512MB RAM
- Mini HDMI and USB On-The-Go ports
- Micro USB power
- HAT-compatible 40-pin header
- Composite video and reset headers
- CSI camera connector
- 802.11 b/g/n wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

It is mainly used to stream sensors inputs from camera module and ultrasonic sensor to the PC.

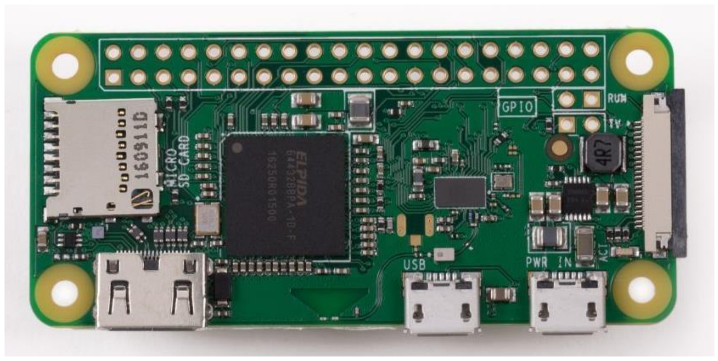
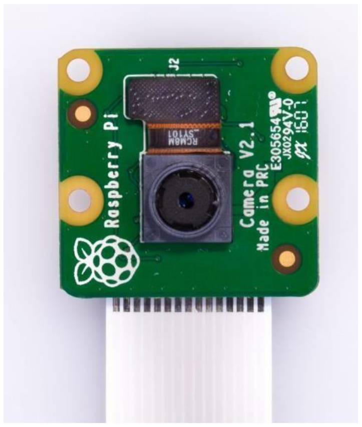



Figure 5.3 (a): Pi Zero W unit and (b): Pi Camera Module

5.2.4 ARDUINO UNO

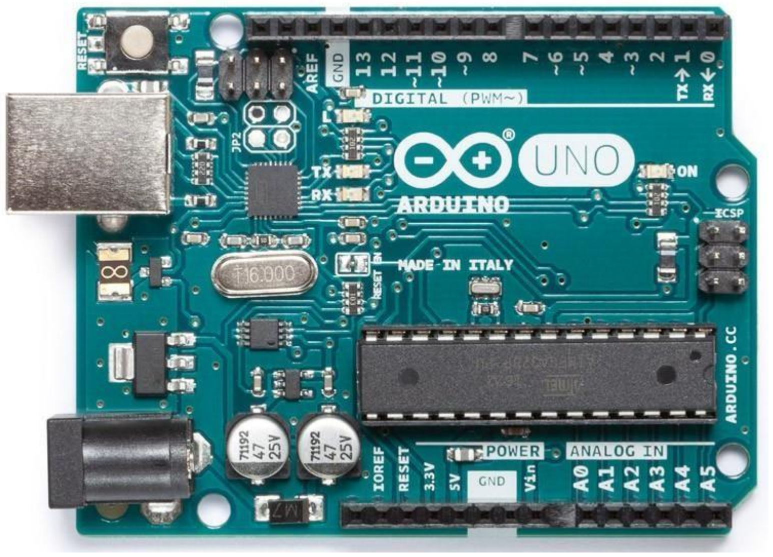


Figure 5.4: Arduino Uno microcontroller

An Arduino is microcontroller board based on Atmega328P, which is open source and made for rapid prototyping and learning. It has extensive online documentation and preferred for ‘plug and play’ development and user friendliness. It is capable of reading and controlling hardware components. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.

We are using it to send signals to remote control of car to control it from the PC. Sending a LOW signal activates the controls and HIGH signals resets it.

5.3 DATA FLOW DIAGRAM

The following figures shows flow of data in the project.

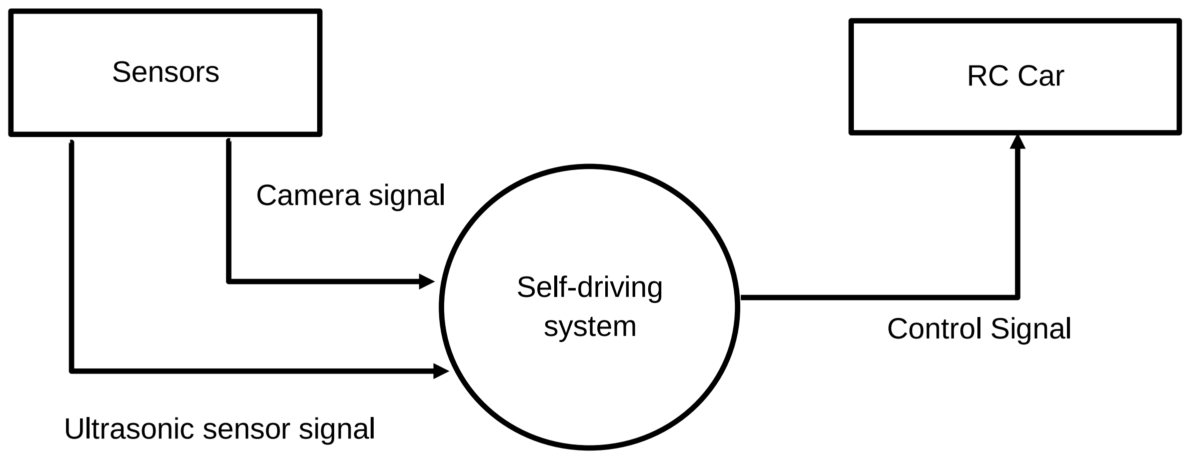


Figure 5.5: Context Level DFD

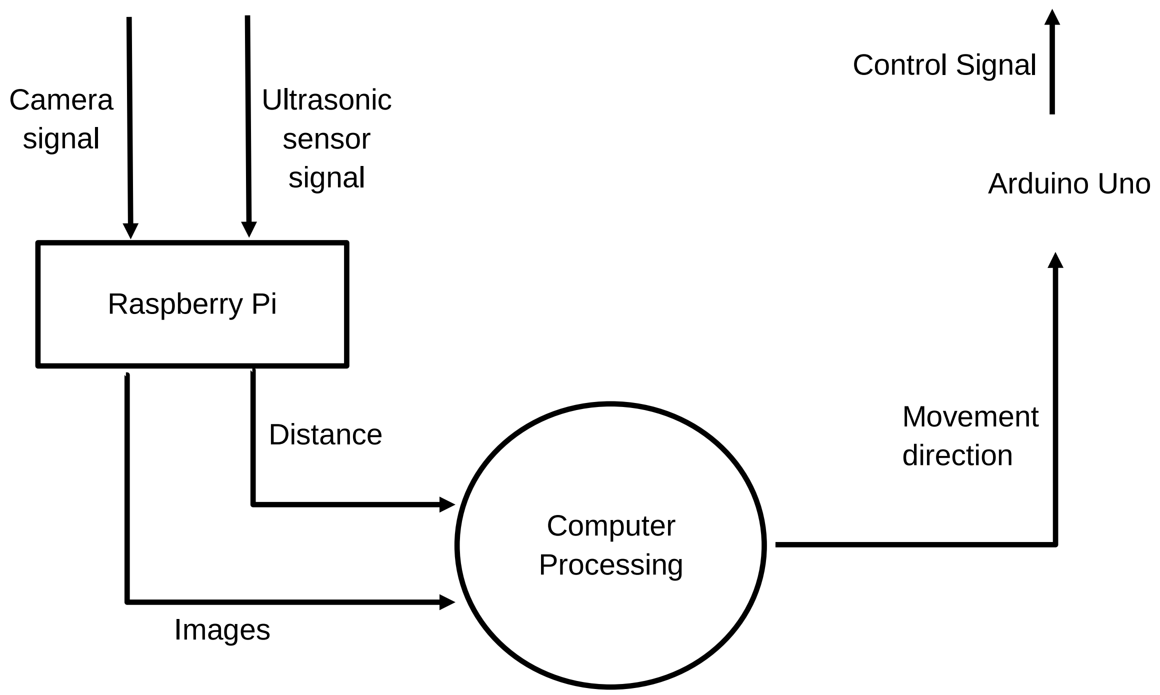


Figure 5.6: Level 1 DFD

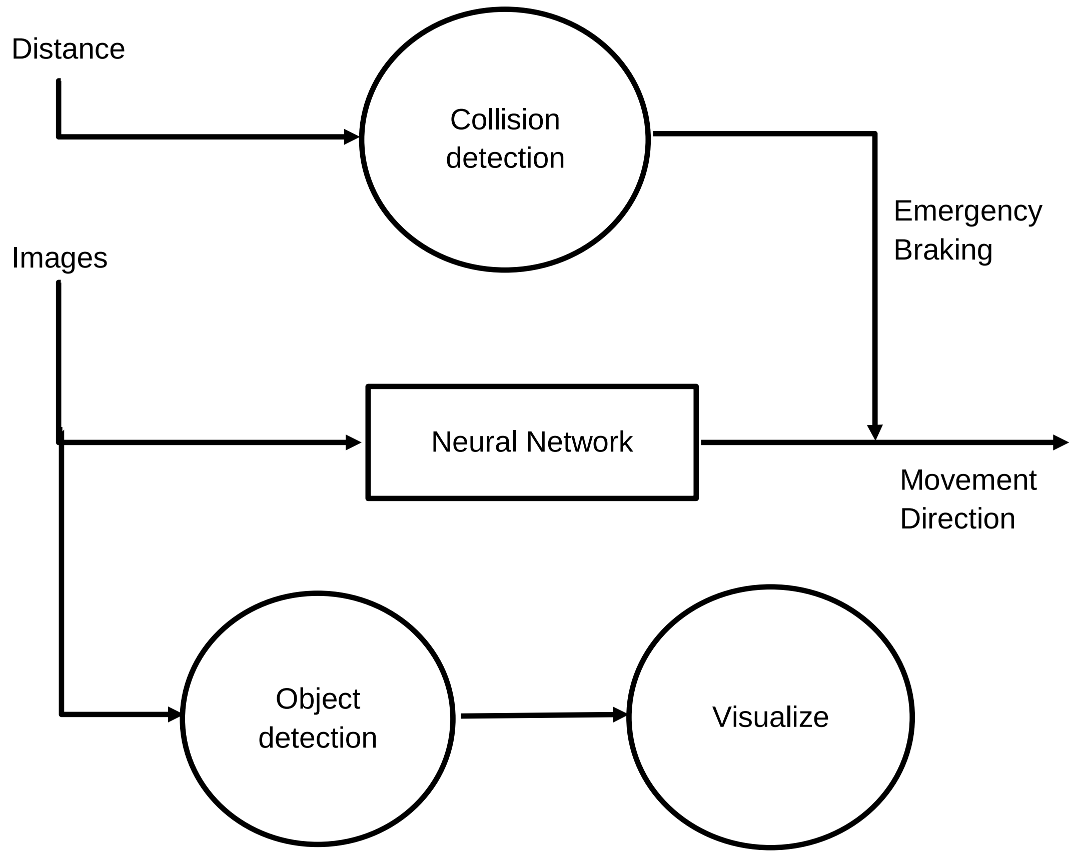


Figure 5.7: Level 2 DFD

6. TESTING

Software testing is the process of investigating the quality of product under stress. Tests are performed to check how the product responds to faults, edge cases and invalid inputs, etc. Depending on the testing methodology being used, testing can be done at any stage of development. Thus, the methodology of tests is governed by the methodology of development.

6.1 FUNCTIONAL TESTING

Functional testing is a type of black box testing and is undertaken for quality assurance reasons. A part of the system is tested by giving it input and checking the output. It is only able to determine what a system does, not how it does something. Steps involved in functional testing are as follows:

- Identifying functions of the product
- Creating input data based on function specification
- Determining output based on function specification
- Checking test cases
- Comparing expected and obtained outputs
- Checking user requirements are satisfied

6.2 STRUCTURAL TESTING

Structural testing is a type of white box testing as we are interested in how functions are implemented and what is accomplished through a code sequence. The testers are required to have a working knowledge of the inner mechanisms of an application, to be able to debug it. Structural testing is done at all levels by developers to maintain bug free code and ensure logical errors don't arise.

6.3 TESTING THE PROJECT

These test cases were checked to successfully implement Self Diving RC Car:

Test Case 1: Video streaming test from Pi Camera to PC through Pi Zero W
Expected Output: Continuous stream of images received on PC at 10fps at resolution 320*240 pixels.
Result: PASS

Test Case 2: Remotely control car from PC keyboard

Expected Output: Vehicle can be controlled directly from PC keyboard via Arduino interfaced with the remote control of car.

Result: PASS (few tries to figure out signals of RC)

Test Case 3: Receive Ultrasonic sensor reading at PC terminal via Pi Zero W

Expected Output: Terminal shows distance readings from ultrasonic sensor at 2 Hz Result: PASS (Distances verified with scale)

Test Case 4: Haar cascade classifiers are tested on video stream

Expected Output: Stream on PC shows the identified objects with name

Result: PASS (clutter and low light hampers detection)

Test Case 5: Run RC Car on white paper instead of between the white papers

Expected Output: Car can’t maneuver as this is not trained

Result: PASS (car stands still/if it detects some familiarity, starts moving in random direction)

7. IMPLEMENTATION

7.1 TRAINING HAAR CASCADES

To identify traffic signals from the camera's stream, we decide to train custom cascade classifiers for our DIY stop sign and traffic light. Since, the OpenCV documentation on training cascades is old and incomplete, we followed a guide by Radamés Ajna and Thiago Hersan who trained similar Haar cascades to detect their cellphones. We used OpenCV utilities for training. To create a decently accurate cascade classifier, we need a lot of both positive and negative image samples. Positive images are those which clearly consist of the object we want to detect. Negative images are those which do not contain any of the objects we want to detect. The following table displays the size of sample we used.

Table 7.1: Sample sizes for different cascades

Type	Number of positive sample	Number of negative sample	Sample size (pixels)
Stop sign	2000	1000	48*48
Green light	2000	1000	48*48
Red light	2000	1000	48*48

Negative samples can be obtained by the online library using the command:

```
cd negativeImageDirectory wget -nd -r -A "neg-
0*.jpg" \
http://haar.thiagohersan.com/haartraining/negatives/
```

We shot only 40 positive samples for our objects, then used OpenCV's create_samples utility to generate more positive samples by superimposing positive samples on negative samples in different angles.

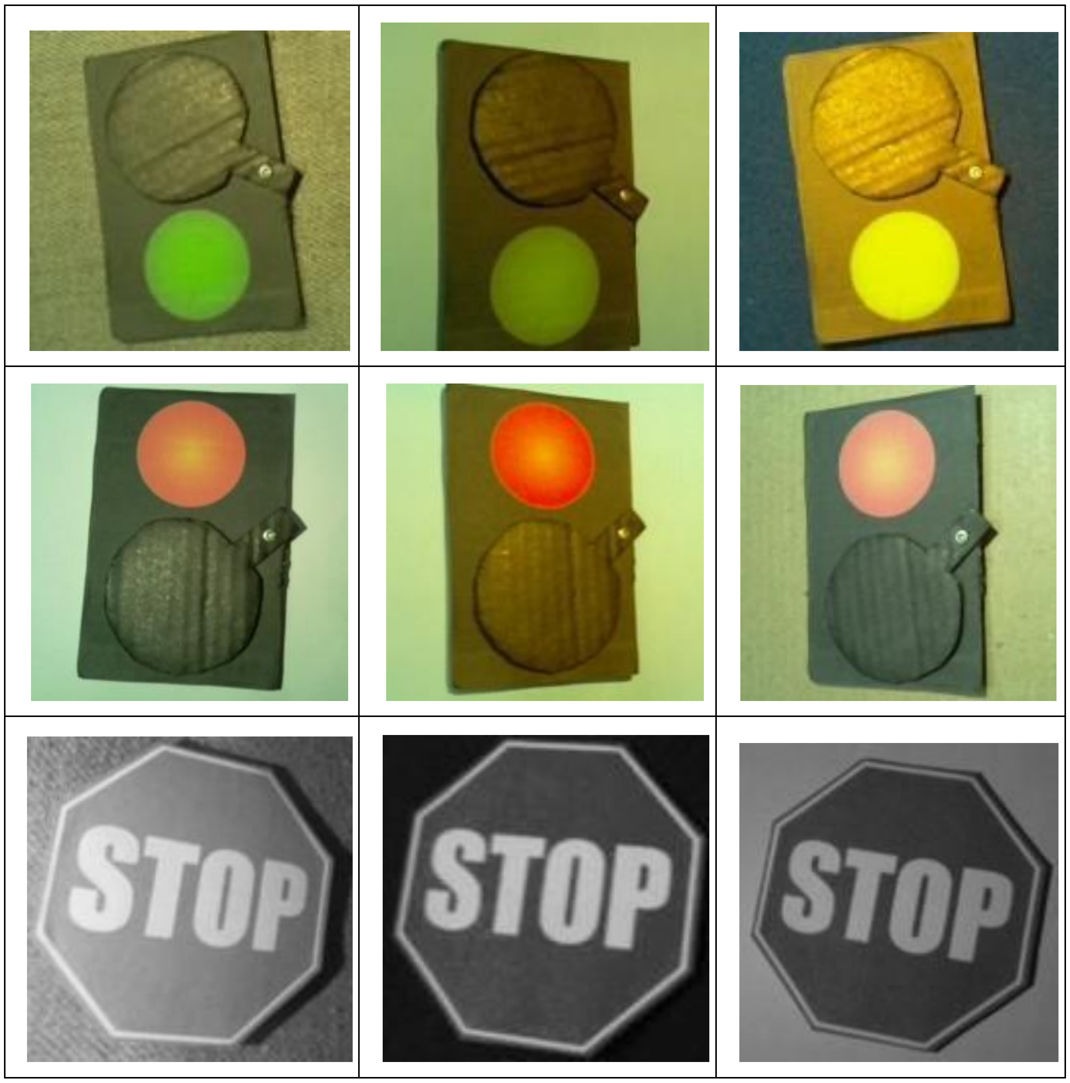


Figure 7.1: Example of positive images used

Then we use OpenCV's train_cascade utility to train our cascade classifiers using this command:

```
opencv_traincascade -data outputDirectory \
-vec cropped.vec \
-bg negativeImageDirectory/negatives.txt \
-numPos 1000 -numNeg 600 -numStages 20 \
-precalcValBufSize 2048 -precalcIdxBufSize 2048 \
-featureType HAAR \
-minHitRate 0.995 -maxFalseAlarmRate 0.5 -w 48 -h 48
```

It took us 48 hours total to completely train all the three cascade classifiers. Then, we tested them on the Raspberry Pi's video stream.

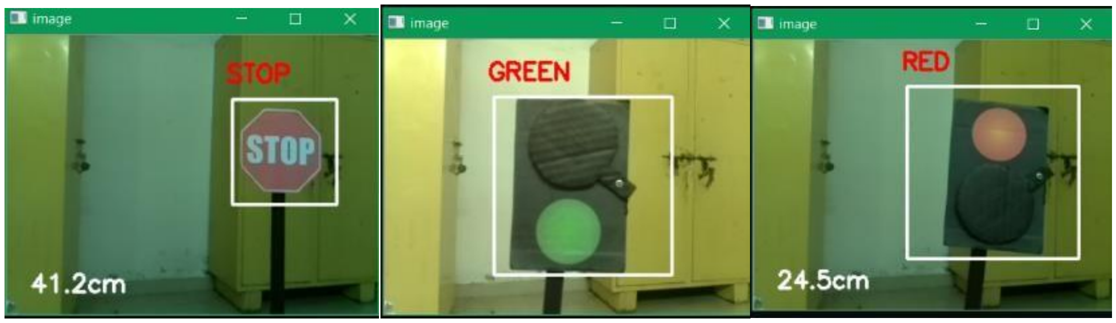


Figure 7.2: Working of Haar Cascades

7.2 ARTIFICIAL NEURAL NETWORKS - MULTI LAYER PERCEPTRON

ANN MLP are a type of supervised neural network. In our project, we are using Backpropagation for training our driverless car. Input is taken form the Pi camera. Input images resolution is 320*240. But we only need the lower half of image to learn the track. So, the input size becomes 320*120 = 38,400. NumPy is used to handle such large input arrays. These 38,400 array elements become the nodes in input layer of our neural network. We arbitrarily choose 40 nodes in the hidden layer. Output layer only has 4 nodes (forward, reverse, left and right), but we don't use reverse. We use sklearn library to split our collected data into 80% training data and 20% test data. This helps as we can immediately check how effective was the training by testing on test data. OpenCV's ANN toolkit is used to train and predict outputs. The resulting neural network diagram is shown below.

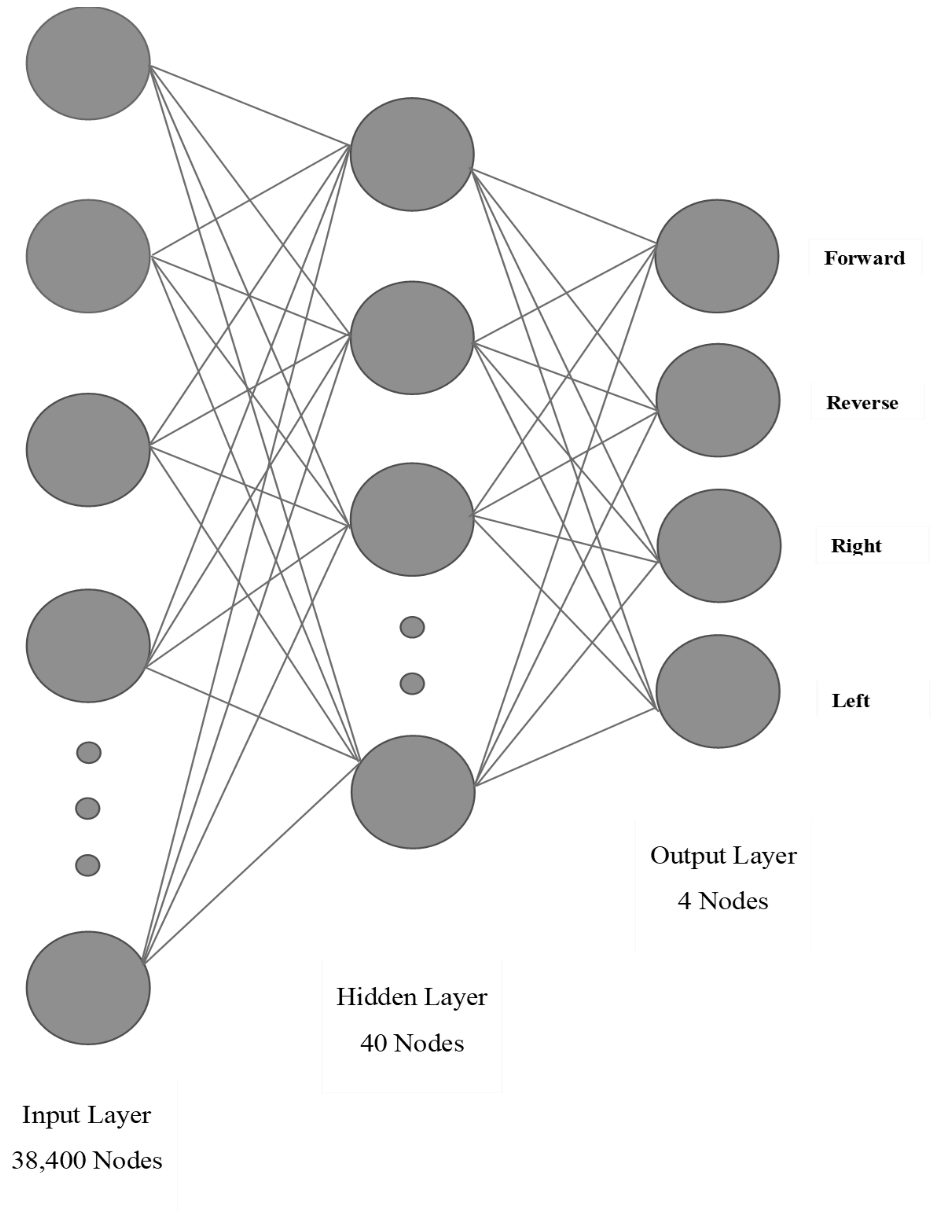
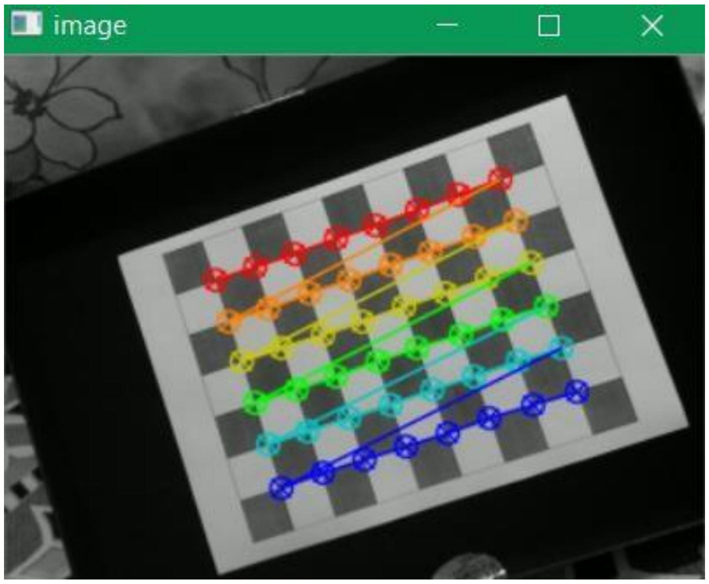
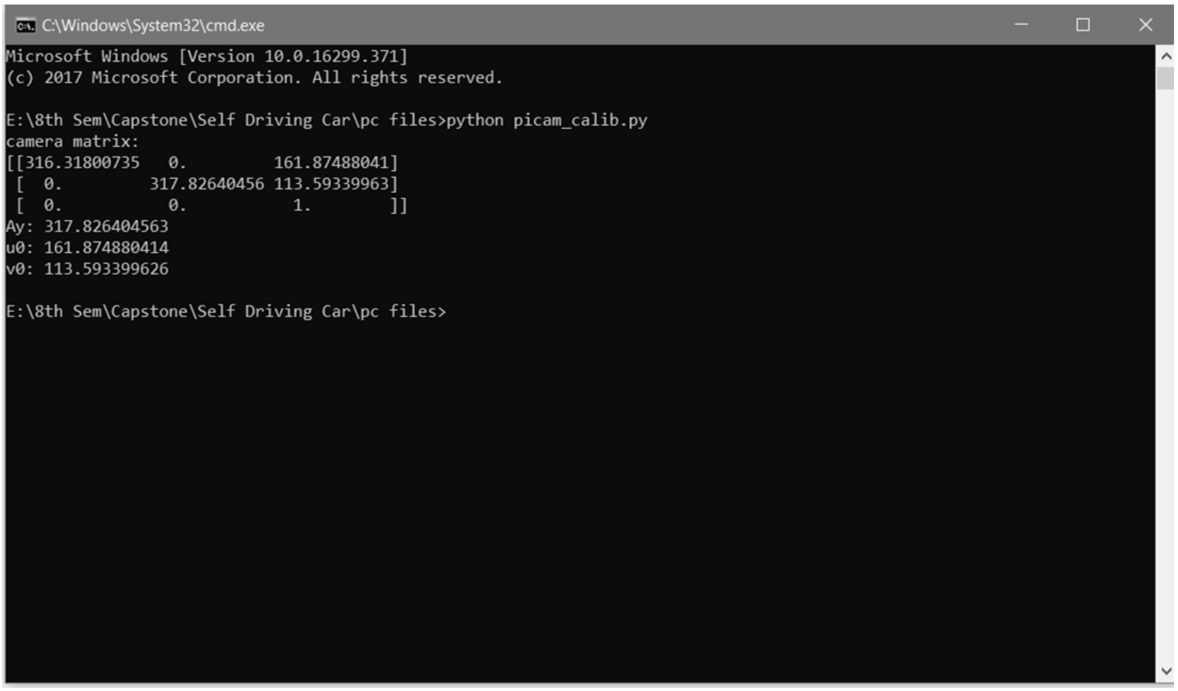


Figure 7.3: Neural network diagram for the project

7.3 PICAM CALIBRATION

In order to get the distance calculation from monocular vision as close to the actual value as possible, we have to account for the imperfections in camera. These are found by obtaining the camera matrix. Code to calibrate Pi Camera is available on the internet which returns the camera matrix. The code works by detecting intersections of black and white squares on a chessboard. This way the straight lines are checked for any curvature, thus determining any imperfections in camera.





```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

E:\8th Sem\Capstone\Self Driving Car\pc files>python picam_calib.py
camera matrix:
[[316.31800735   0.         161.87488041]
 [  0.         317.82640456 113.59339963]
 [  0.           0.           1.        ]]
Ay: 317.826404563
u0: 161.874880414
v0: 113.593399626

E:\8th Sem\Capstone\Self Driving Car\pc files>
```

Figure 7.4 (a): Calibration process detecting chessboard (b): Camera matrix for Pi camera

8. PROJECT LEGACY

8.1 CURRENT STATUS OF PROJECT

The project was successful in converting a regular RC Car into an autonomous vehicle. The car can maneuver track represented by white sheets of paper on both sides. It responds to traffic signs and has collision avoidance. The car checks the fundamental requirements to be called self-driving car.

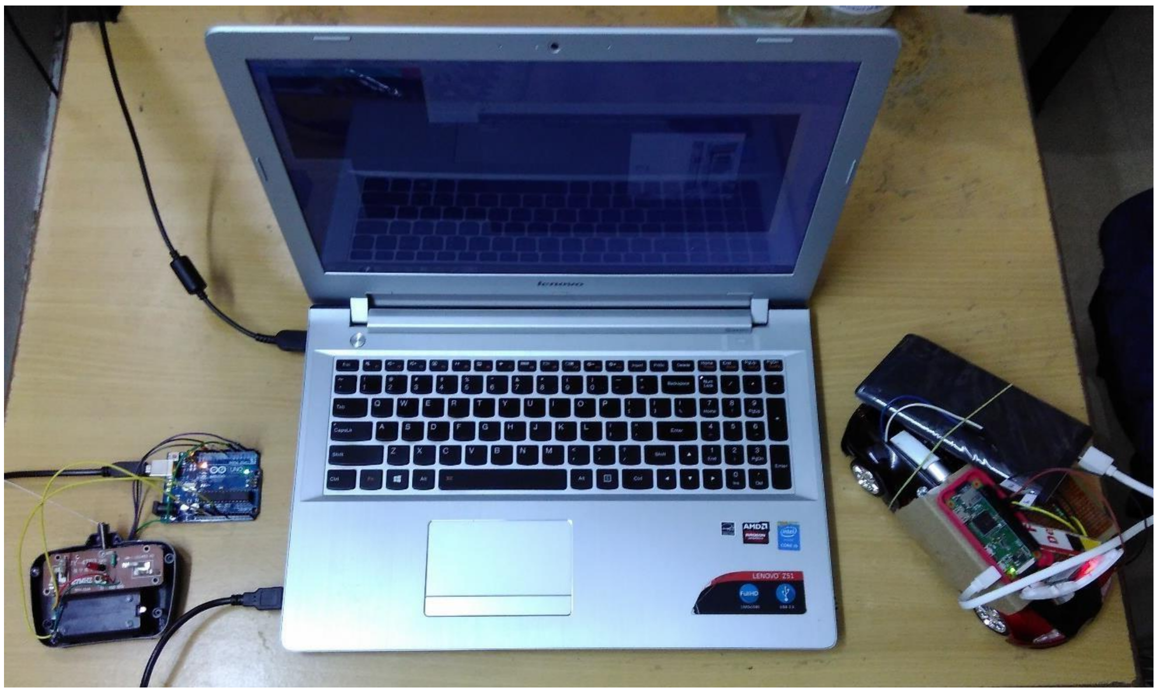


Figure 8.1: All project components shown side by side. Arduino and remote control on left, PC in center and RC car with RPi on the right

8.2 REMAINING AREAS OF CONCERN

This project aims to only touch the basics of what a self-driving vehicle should be like. All the areas can be improved if more time and large samples are provided for training. Our training takes place only on the CPU of PC. But GPUs are better suited for training neural networks. So, using techniques that can utilize GPU instead of CPU will yield better results faster, reducing the training time by a huge margin. The RC Car used here was a very cheap model. It can drive at only one speed and battery backup is very poor. Using an expensive but better built car will provide significantly better results when trained in the same manner. Also, being able to switch between speeds will be helpful for navigating more complex tracks.

8.3 TECHNICAL AND MANAGERIAL LESSONS LEARNT

So, we are finally able to answer the questions we asked when starting this project. An autonomous vehicle is one which does not require human intervention to carry out tasks like navigate roads and follow traffic signals. Yes, any vehicle can be converted into an autonomous vehicle after a certain number of changes are made and sensor and processing units are installed. But, it is not commercially viable to convert all cars this way. However, it’s becoming more and more viable to manufacture autonomous cars commercially as the technology matures.

9. USER MANUAL

- Hardware Used
- Hardware Setup
- Running the Project

HARDWARE USED

Input

1. Raspberry Pi Zero W
2. Raspberry Pi Camera Module
3. Ultrasonic Sensor Output
4. Remote Control Car with Wireless Remote
5. Arduino Uno Processing
6. Laptop

Input:
We are using Raspberry Pi Camera to collect front and to identify objects like traffic lights and stop sign. Ultrasonic sensor to find if an object is present in front of the car and to get the distance of the object. Raspberry Pi Zero W collects this data and transmits to the laptop over a secure wireless connection in form of a video stream and distance as text data which need to be processed by laptop.

Output:
Arduino Uno is used for signaling the remote control of car based on instructions from laptop through serial port. These instructions are to perform tasks for a set time like moving forward or reverse for 50 milliseconds.

Processing:
We are using our laptop as a brain of the whole project. It receives input from Raspberry Pi Camera. From this data, python scripts running on the laptop identify the track, objects

and object's distance from the car. These inputs are processed by ANN to predict movement of car. These are send to Arduino Uno which signals the remote controller of car.

HARDWARE SETUP

Raspberry Pi Zero W have a mini CIS, so we are using adapter cable to connect camera. We also solder GPIO stacking header, to connect the Ultrasonic sensor. To connect ultrasonic sensor, we are using R1 330ohm and R2 470ohm register. Connect Pin 2 (VCC), Pin 6 (GND) and Pin 12 (GPIO 18) of Raspberry PI Zero W to the VCC, GND and TRIG on ultrasonic sensor respectively. Connect R1 to ECHO and another end to GPIO 24 (Pin 18) on Raspberry Pi Zero W and to one end of R2. Register R2's second end to Pin 6 (GND) on Raspberry Pi Zero W. We power the Pi using a power bank. Arduino is powered over USB from the laptop. We connected VCC and GND of Remote Control to the Power 5V and GND on the Arduino Uno. Left, right, forward and reverse of the remote control to the Arduino Uno's pin 2, 5, 9, 12 using some jumper wires.

RUNNING THE PROJECT

We are using Arduino IDE for writing and burning microcontroller code and Python 2.7 for remaining scripts. We are using following python libraries:

-numpy	-glob
-io	-picamera
-cv2	-sys
-pygame	-serial
-os	-math
-SocketServer	-threading
-skylearn	-RPi.GPIO
-time	-socket

Raspberry Pi:

Connect via PuTTY by typing ‘raspberrypi.local’. Enter credentials to complete ssh login. Change to project directory ‘/Desktop/RC Car/’. Execute the two python scripts using command ‘python stream_pi.py & python ultrasonic_pi.py &’.

PC:

Open command prompt in project directory. To collect training data run ‘data_collection.py’ and use arrow keys on keyboard to control RC Car on the track. Press ‘q’ at the end to save capture training data. Now, to train neural network, run ‘mlp_training.py’. If the training accuracy is good enough, run ‘driver.py’ while the car is on the track.

10. SOURCE CODE

PC CODE

picam_calib.py

```
# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# 6x8 chess board, prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*8, 3), np.float32)
objp[:, :2] = np.mgrid[0:8, 0:6].T.reshape(-1, 2)

# 3d point in real world space
objpoints = []
# 2d points in image plane
imgpoints = []
h, w = 0, 0

images = glob.glob('picam/*.jpg')

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    h, w = gray.shape[:2]

    # find chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (8, 6), None)

    # add object points, image points
    if ret:
        objpoints.append(objp)
        cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
        imgpoints.append(corners)

        # draw and display the corners
        cv2.drawChessboardCorners(img, (8, 6), corners, ret)
        cv2.imshow('image', img)
        cv2.waitKey(500)

# calibration
retval, cameraMatrix, distCoeffs, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, (w, h)
, None, None)
```

This program is used for calibrating the Pi camera. It processes 20 chessboard images taken by Pi camera to detect curves in straight lines. Output is the camera matrix which is used for error correction in images.

data_collection.py

```
    def ImgCollection(self):

        saved = 0
        total = 0

        t1 = cv2.getTickCount()
        img_arr = np.zeros((1, 38400))
        label_arr = np.zeros((1, 4), 'float')

        print 'Image collection started...'

        # stream images 1 frame at a time
        try:
            stream_bytes = ' '
            frame = 1
            while self.send_inst:
                stream_bytes += self.connection.read(1024)
                first = stream_bytes.find('\xff\xd8')
                last = stream_bytes.find('\xff\xd9')
                if first != -1 and last != -1:
                    jpg = stream_bytes[first:last + 2]
                    stream_bytes = stream_bytes[last + 2:]
                    image = cv2.imdecode(np.fromstring(jpg, dtype=np.uint8), 0) #grayscale image

                    # extracting bottom half of image as region of interest
                    roi = image[120:240, :]

                    # save streamed images
                    cv2.imwrite('collection_images/img{:>05}.jpg'.format(frame), image)

                    #cv2.imshow('roi_image', roi)
                    cv2.imshow('image', image)

                    # reshape the roi image into one row array
                    temp = roi.reshape(1, 38400).astype(np.float32)

                    frame += 1
                    total += 1

                    # get input from human driver
                    for event in pygame.event.get():
                        if event.type == pygame.KEYDOWN:
                            key_pressed = pygame.key.get_pressed()

                            if key_pressed[pygame.K_UP]:
                                print("Forward")
                                saved += 1
                                img_arr = np.vstack((img_arr, temp))
                                label_arr = np.vstack((label_arr, self.m[0])) #forward as [1 0 0
0]

                                self.ser.write(chr(1))
```

Above snippet shows how images are converted to an array of length of 38,400 and user input for car direction in array of length 4 (for example [1 0 0 0] represents forward direction).

```
            # save images and labels
            train = img_arr[1:, :]
            train_labels = label_arr[1:, :]

            # save data as a numpy file
            file_name = str(int(time.time()))
            directory = "collection_data"
            if not os.path.exists(directory):
                os.makedirs(directory)
            try:
                np.savez(directory + '/' + file_name + '.npz', train=train, train_labels=train_la
bels)
```

On exit, the paired images and inputs are saved as '.npz' file which is used for training the neural network.

mlp_training.py

```
for single_npz in collected_data:
    with np.load(single_npz) as data:
        temp = data['train']
        labels_temp = data['train_labels']
    img_arr = np.vstack((img_arr, temp))
    label_arr = np.vstack((label_arr, labels_temp))

x = img_arr[1:, :]
y = label_arr[1:, :]
print 'Image array shape: ', x.shape
print 'Label array shape: ', y.shape

t2 = cv2.getTickCount()
time0 = (t2 - t1)/ cv2.getTickFrequency()
print 'Images loaded in {0:.2f} secs'.format(time0)

# splitting data into 80% training data and 20% test data
train, test, train_labels, test_labels = train_test_split(x, y, test_size=0.2)

t3 = cv2.getTickCount()

# create MLP
layers = np.int32([38400, 40, 4])
model = cv2.ml.ANN_MLP_create()
model.setLayerSizes(layers)
model.setTrainMethod(cv2.ml.ANN_MLP_BACKPROP)
model.setBackpropMomentumScale(0.0)
model.setBackpropWeightScale(0.001)
model.setTermCriteria((cv2.TERM_CRITERIA_COUNT, 500, 0.0001))
model.setActivationFunction(cv2.ml.ANN_MLP_SIGMOID_SYM, 2, 1)

print 'Training MLP ...'
n_itr = model.train(np.float32(train), cv2.ml.ROW_SAMPLE, np.float32(train_labels))
```

This snippet shows how the collected data is loaded again and split into two parts. 80% training data and 20% test data. In the end, trained data is checked on tested data to find the accuracy of trained network. Trained network is saved as '.xml' file.

```
# training data
ret_0, resp_0 = model.predict(train)
prediction_0 = resp_0.argmax(-1)
true_labels_0 = train_labels.argmax(-1)

train_rate = np.mean(prediction_0 == true_labels_0)
print 'Train accuracy: ', "{0:.2f}%".format(train_rate * 100)

# testing data
ret_1, resp_1 = model.predict(test)
prediction_1 = resp_1.argmax(-1)
true_labels_1 = test_labels.argmax(-1)

test_rate = np.mean(prediction_1 == true_labels_1)
print 'Test accuracy: ', "{0:.2f}%".format(test_rate * 100)

# save the multilayer perceptron model
model.save('mlp_trained/trained_network.xml')
```

Training accuracy is also calculated using test data.

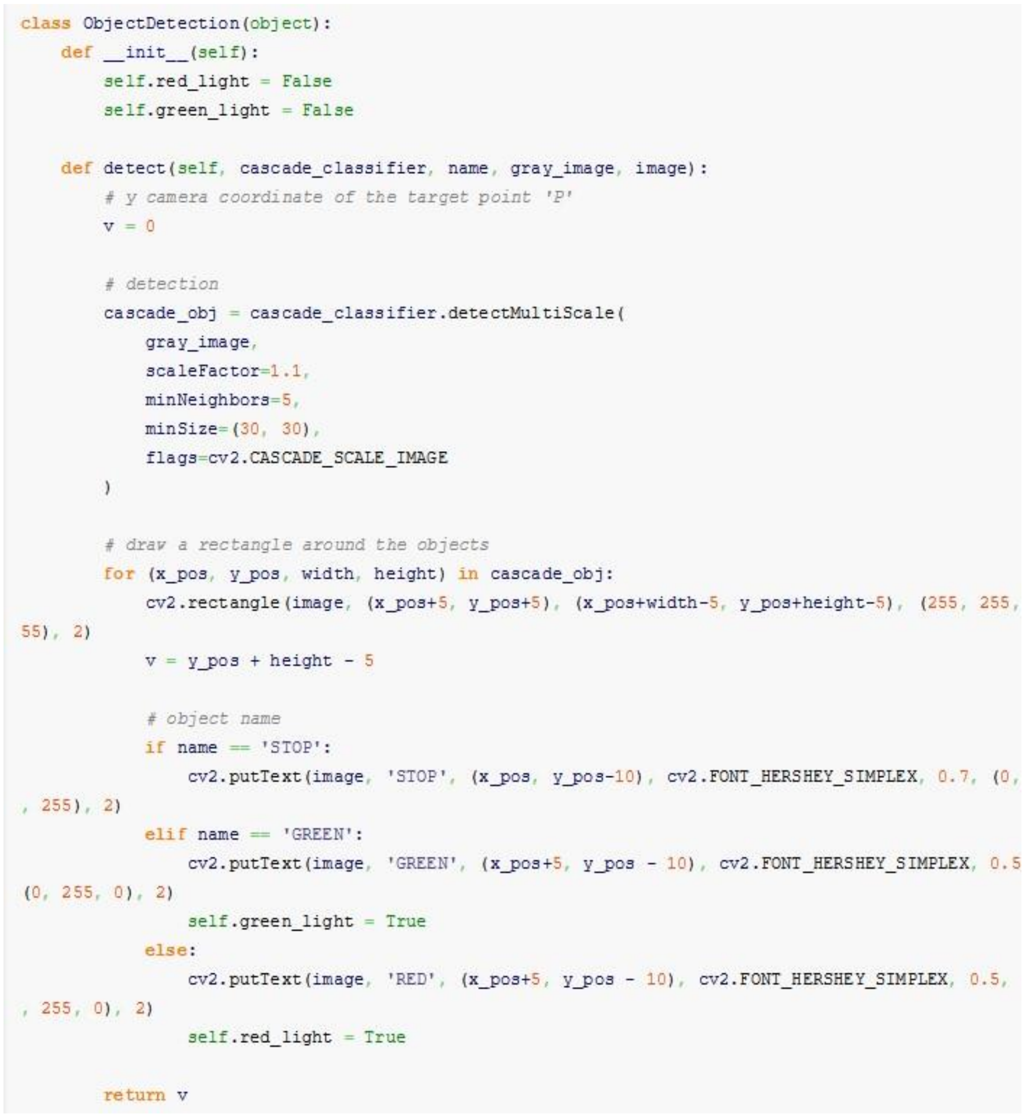
driver.py

```
class NeuralNetwork(object):

    def __init__(self):
        self.model = cv2.ml.ANN_MLP_create()

    def create(self):
        layer = np.int32([38400, 40, 4])
        self.model.setLayerSizes(layer)
        self.model = cv2.ml.ANN_MLP_load('mlp_trained/trained_network.xml')

    def predict(self, samples):
        ret, resp = self.model.predict(samples)
        return resp.argmax(-1)
```

This is the driver program of our autonomous car. This function loads the trained neural network.

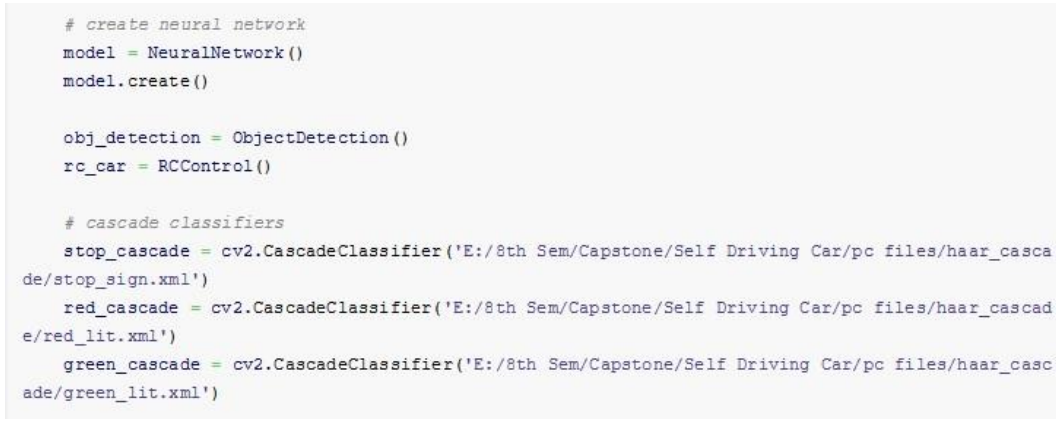
```
class DistanceToCamera(object):

    def __init__(self):
        # picam parameters from calibration
        self.alpha = 8.0 * math.pi / 180 #tilt angle of camera
        self.v0 = 113.593399626
        self.ay = 317.826404563

    def calculate(self, v, h, x_shift, image):
        # compute and return the distance from the target point to the camera
        d = h / math.tan(self.alpha + math.atan((v - self.v0) / self.ay))
        if d > 0:
            cv2.putText(image, "%.1fcm" % d,
                (image.shape[1] - x_shift, image.shape[0] - 20), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (2
55, 255, 255), 2)
        return d
```

Function to calculate distance from monocular image is defined.



```
class ObjectDetection(object):
    def __init__(self):
        self.red_light = False
        self.green_light = False

    def detect(self, cascade_classifier, name, gray_image, image):
        # y camera coordinate of the target point 'P'
        v = 0

        # detection
        cascade_obj = cascade_classifier.detectMultiScale(
            gray_image,
            scaleFactor=1.1,
            minNeighbors=5,
            minSize=(30, 30),
            flags=cv2.CASCADE_SCALE_IMAGE
        )

        # draw a rectangle around the objects
        for (x_pos, y_pos, width, height) in cascade_obj:
            cv2.rectangle(image, (x_pos+5, y_pos+5), (x_pos+width-5, y_pos+height-5), (255, 255,
55), 2)
            v = y_pos + height - 5

            # object name
            if name == 'STOP':
                cv2.putText(image, 'STOP', (x_pos, y_pos-10), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,
, 255), 2)
            elif name == 'GREEN':
                cv2.putText(image, 'GREEN', (x_pos+5, y_pos - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5
(0, 255, 0), 2)
                self.green_light = True
            else:
                cv2.putText(image, 'RED', (x_pos+5, y_pos - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
, 255, 0), 2)
                self.red_light = True

        return v
```

Object detection function that draws a frame around object and labels it.



```
    # create neural network
    model = NeuralNetwork()
    model.create()

    obj_detection = ObjectDetection()
    rc_car = RCControl()

    # cascade classifiers
    stop_cascade = cv2.CascadeClassifier('E:/8th Sem/Capstone/Self Driving Car/pc files/haar_casca
de/stop_sign.xml')
    red_cascade = cv2.CascadeClassifier('E:/8th Sem/Capstone/Self Driving Car/pc files/haar_cascad
e/red_lit.xml')
    green_cascade = cv2.CascadeClassifier('E:/8th Sem/Capstone/Self Driving Car/pc files/haar_casc
ade/green_lit.xml')
```

Neural network is initialized and Haar cascades are loaded.

```
# lower half of the image
half_gray = gray[120:240, :]

# object detection
v_param1 = self.obj_detection.detect(self.stop_cascade, 'STOP', gray, image)
v_param2 = self.obj_detection.detect(self.red_cascade, 'RED', gray, image)
v_param3 = self.obj_detection.detect(self.green_cascade, 'GREEN', gray, image)

# distance measurement
if v_param1 > 0 or v_param2 > 0 or v_param3 > 0:
    d1 = self.d_to_camera.calculate(v_param1, self.h, 300, image)
    d2 = self.d_to_camera.calculate(v_param2, self.h, 300, image)
    d3 = self.d_to_camera.calculate(v_param2, self.h, 300, image)
    self.d_stop_sign = d1
    self.d_rlight = d2
    self.d_glight = d3
```

Object detection function is called, and monocular distance calculated.

```
# reshape image
img_arr = half_gray.reshape(1, 38400).astype(np.float32)

# neural network makes prediction
prediction = self.model.predict(img_arr)

# stop conditions
if sensor_data is not None and sensor_data < 30:
    print("Stopping, collision imminent")
    self.rc_car.stop()

elif 0 < self.d_stop_sign < 25 and stop_sign_active:
    print("Stop sign ahead")
    self.rc_car.stop()

    # stop for 5 seconds
    if stop_flag is False:
        self.stop_start = cv2.getTickCount()
        stop_flag = True
    self.stop_finish = cv2.getTickCount()

    self.stop_time = (self.stop_finish - self.stop_start)/cv2.getTickFrequency
()

    print "Stop time: {.2f} s".format(self.stop_time)

    # 5 seconds later, continue driving
    if self.stop_time > 5:
        print("Waited for 5 seconds")
        stop_flag = False
        stop_sign_active = False

elif 0 < self.d_glight < 30 and self.obj_detection.green_light:
    print("Green light")
    self.d_glight = 30
    self.obj_detection.green_light = False
    pass

elif 0 < self.d_rlight < 30 and self.obj_detection.red_light:
    print("Red light")
    self.rc_car.stop()
    self.d_rlight = 30
    self.obj_detection.red_light = False
    pass
```

Prediction is made on the input image. Sensor data is checked for collision or if camera detects any traffic signals. RC Car is accordingly stopped or moved.

RASPBERRY PI CODE

stream_pi.py

```python
try:
    with picamera.PiCamera() as camera:
        camera.resolution = (320, 240)      # pi camera resolution
        camera.framerate = 10               # 10 frames/sec
        time.sleep(2)                       # give 2 secs for camera to initilize
        start = time.time()
        stream = io.BytesIO()

        # send jpeg format video stream
        for foo in camera.capture_continuous(stream, 'jpeg', use_video_port = True):
            connection.write(struct.pack('<L', stream.tell()))
            connection.flush()
            stream.seek(0)
            connection.write(stream.read())
            if time.time() - start > 600:
                break
            stream.seek(0)
            stream.truncate()
    connection.write(struct.pack('<L', 0))
```

Above snippet shows how camera images are streamed from Pi to PC.

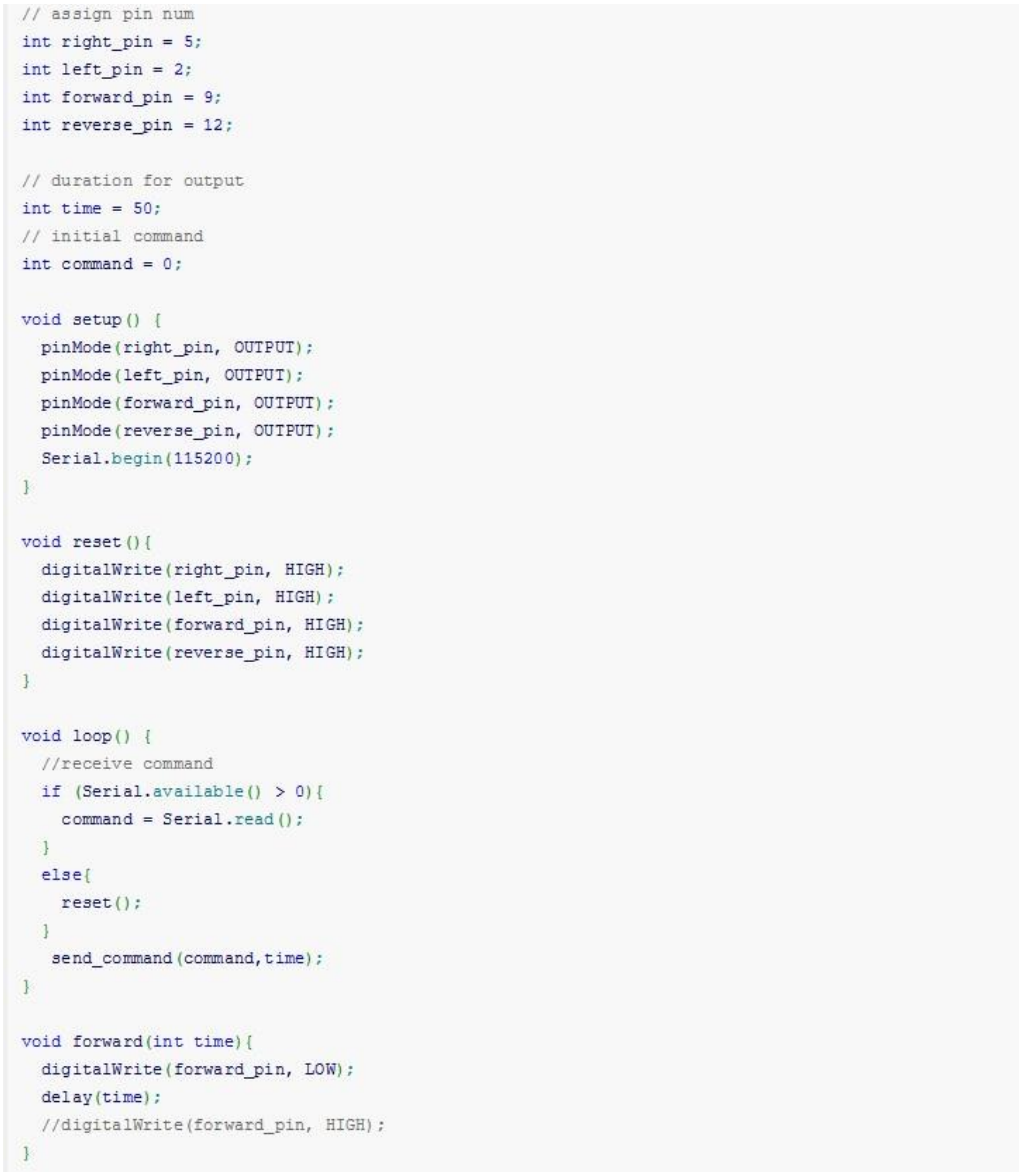
ultrasonic_pi.py

```python
def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance

try:
    while True:
        dist = distance()
        print ("Measured Distance = %.1f cm" % dist)
        client_socket.send(str(dist))
        time.sleep(0.5)
```

Distance is calculated using ultrasonic sensor input. This is then streamed to PC.

ARDUINO CODE

rc_kb.ino



```
// assign pin num
int right_pin = 5;
int left_pin = 2;
int forward_pin = 9;
int reverse_pin = 12;

// duration for output
int time = 50;
// initial command
int command = 0;

void setup() {
  pinMode(right_pin, OUTPUT);
  pinMode(left_pin, OUTPUT);
  pinMode(forward_pin, OUTPUT);
  pinMode(reverse_pin, OUTPUT);
  Serial.begin(115200);
}

void reset(){
  digitalWrite(right_pin, HIGH);
  digitalWrite(left_pin, HIGH);
  digitalWrite(forward_pin, HIGH);
  digitalWrite(reverse_pin, HIGH);
}

void loop() {
  //receive command
  if (Serial.available() > 0){
    command = Serial.read();
  }
  else{
    reset();
  }
   send_command(command,time);
}

void forward(int time){
  digitalWrite(forward_pin, LOW);
  delay(time);
  //digitalWrite(forward_pin, HIGH);
}
```

Above snippet shows how Arduino pins are set and forward fiction is shown. Similarly, other directions can be activated

11. BIBLIOGRAPHY

[1] Study on Method of Detecting Preceding Vehicle Based on Monocular Camera by Chu Jiangwei, Ji Lisheng, Guo Lie, Libibing and Wang Rongben, Jilin university, China

[2] Training Haar Cascades by Radamés Ajna and Thiago Hersan, https://memememememememe.me/post/training-haar-cascades/

[3] OpenCV-Python Tutorials - Camera Calibration and 3D Reconstruction, http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html [4] PiCamera documentation, https://picamera.readthedocs.io/en/release-1.13/recipes2.html [5] Using a Raspberry Pi distance sensor (ultrasonic sensor HC-SR04), https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/

[6] Levels of Driving Automation are Defined in New SAE International Standard J3016 (original link in no longer accessible), https://web.archive.org/web/20170903105244/https://www.sae.org/misc/pdfs/automated_driving.pdf

[7] OpenCV Library, https://opencv.org/

[8] NumPy, http://www.numpy.org/

[9] scikit-learn: machine learning in Python, http://scikit-learn.org/stable/index.html