Self-driving Autonomous Car Implementing Maps and V2V

Khyati Mehta[1], Mehzabeen Najmi[2], Deepthi. V[3], K.N. Hemalatha[4]

[1,2,3]UG Students, [4]Assistant Professor

Dept. of ECE, Dr. Ambedkar Institute of Technology, Bengaluru, Karnataka, India

***Abstract*—This paper introduces an autonomous car that drives itself built using Raspberry Pi. It is trained using an artificial neural network to drive like the trainer and follows Google Map directions along with V2V for blind spot detection.**

Keywords—*RaspberryPi, artificial neural network, google maps, V2V, blind spot detection*

I. Introduction

Over 1,000,000 accidents happen every year due to bad human driving. The major cause is drunk driving. Other causes include diversion of the driver from road, due to falling asleep, or texting while driving. These incidences require a revolutionary idea that can implement road safety. Here's where a driverless car can be helpful. This paper introduces a 1/18th prototype which is state of the art autonomous car that drives without any human intervention, by being trained using artificial neural networks and by following google maps directions along with implementing V2V protocol for blind spot detection.

A. Block diagram

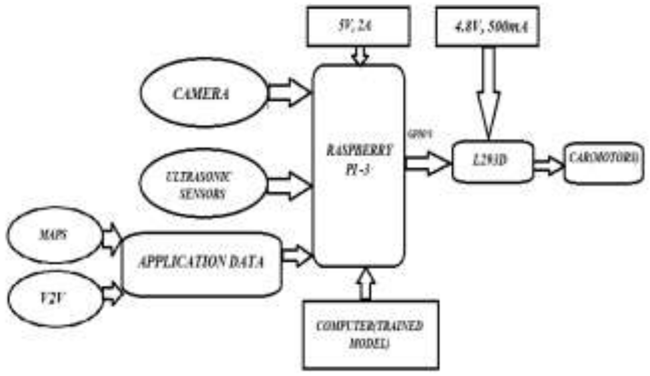


Fig 1.1 Functional Architecture

B. Hardware

- Raspberry pi 3 model B,
- Pi camera from Raspberry pi,
- HC-SR04 Ultrasonic Sensor,
- L239d motor driver,
- 9V batteries for the motors, along with 5V 1A power bank for Raspberry Pi,
- Solar panels to charge above batteries.

C. Software

- Raspbian OS – for raspberry pi.
- RPi_Cam_Web_Interface [1] – for recording Pi camera video stream to the client.
- Application written using python for v2v and Google Maps directions.

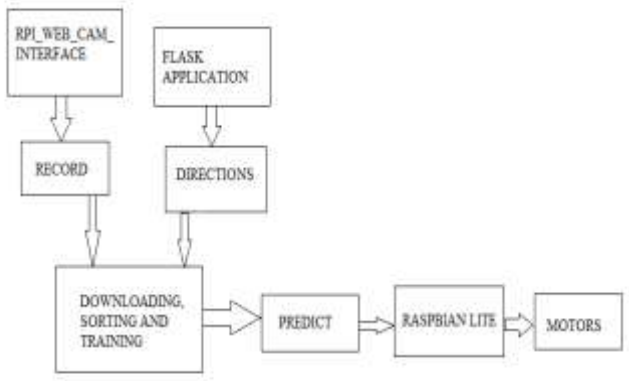


Fig 1.2 Working Block Diagram

D. Working

The raspberry pi launches the RPi_Cam_Web_Interface cloned from Github, which in turn launches a lighttpd or an Apache server. A client on the same network can get a live stream of the Pi camera. This interface also has the option for video recording, which is used for recording the video of initial driving of the car. The car is driven by the user itself, and the recorded video is downloaded to a computer. This video is then converted into pictures with 10 frames per second into original colored images. These images are then converted into OpenCV array [2] which can then be used as data for artificial neural network to train on. Training is done by labeling these images as left, right, forward and reverse directional images, which is done manually. After training, the car is now capable of driving itself in a manner similar to the initial driver.

On Google maps, a new route is created which is the travel path for the autonomous car within campus. This route is then downloaded in the form of kml file which contains the route split up into many geographic co-ordinates. This information is of importance. Alternately, the google maps Roads API can also be used for generalized roads. The above hack is done for the prototype only.

The python application uses the car's present location, and utilizes Haversine formula [3] to calculate the direction of movement towards the next location to reach. This direction is converted into degrees. The autonomous driving is done by taking the prediction and multiplying that with the direction. For example, suppose the predicted direction is left and direction output in degrees is somewhere over 270 then the car goes left. In this way, the car drives on its own, avoiding obstacles with ultrasonic sensor, while following maps.

1) Distance Calculation Using Haversine Formula [3]

Haversine formula is used to calculate the great-circle distance between two points – that is, the shortest distance over the earth's surface – giving an 'as the crow flies' distance between the points. The Haversine formula is shown below:

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

Haversine formula:

$$d = R \cdot c$$

Where φ *is latitude,* λ *is longitude,* R *is earth's radius (mean radius = 6,371km); note that angles need to be in radians to pass to trig functions!*

Direction Formula:

$$\theta = \operatorname{atan2}(\sin\Delta\lambda \cdot \cos\varphi 2 , \cos\varphi 1 \cdot \sin\varphi 2 - \sin\varphi 1 \cdot \cos\varphi 2 \cdot \cos\Delta\lambda)$$

where $\varphi 1,\lambda 1$ is the start point, $\varphi 2,\lambda 2$ the end point ($\Delta\lambda$ is the difference in longitude)

The distance d is in km and the bearing is in radians.

Specifically in this case, the formula is used to calculate distance and direction between two consecutive latitude-longitude points. These points belong to the route plotted on the maps that the car must follow. Once the car reaches that point, the destination location is set to the next point and so on till the end of the route.

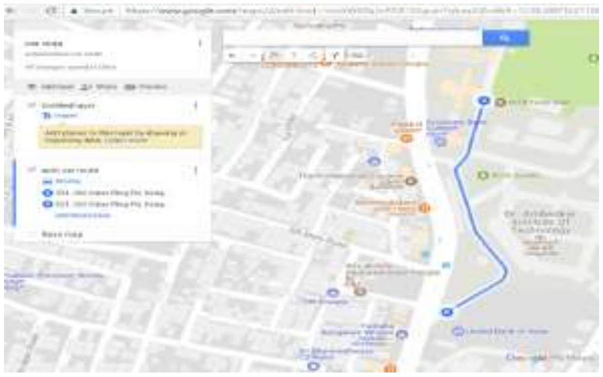
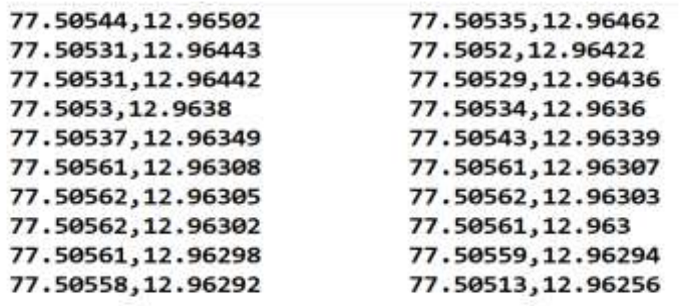


Fig 2.1 Car's Route on Google Maps

Figure 2.1 shows a snapshot of the car's route plotted in Google maps.

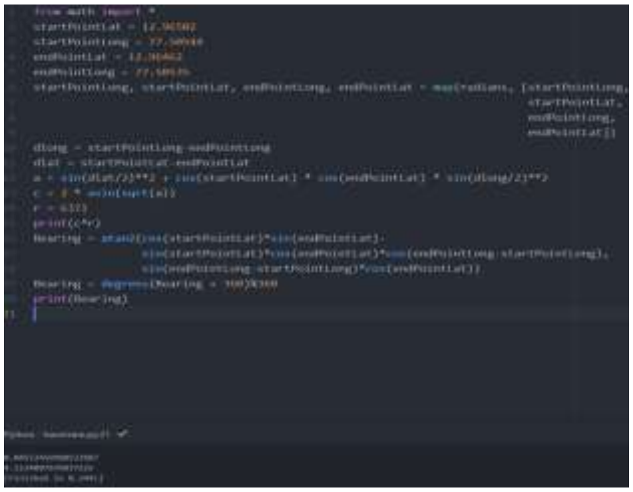
Figure 2.2 shows a snapshot of the longitude-latitude points collected from this route using Google Maps kml file.



```
77.50544,12.96502          77.50535,12.96462
77.50531,12.96443          77.5052,12.96422
77.50531,12.96442          77.50529,12.96436
77.5053,12.9638            77.50534,12.9636
77.50537,12.96349          77.50543,12.96339
77.50561,12.96308          77.50561,12.96307
77.50562,12.96305          77.50562,12.96303
77.50562,12.96302          77.50561,12.963
77.50561,12.96298          77.50559,12.96294
77.50558,12.96292          77.50513,12.96256
```

Fig 2.2 Long-Lat Points of Car's Route

Figure 2.3 shows a code snippet written in python that uses Haversine formula and calculates distance between two points along with direction in degree, to be followed.



```
from math import *
startPointLat = 12.96502
startPointLong = 77.50544
endPointLat = 12.96462
endPointLong = 77.50535
startPointLong, startPointLat, endPointLong, endPointLat = map(radians, [startPointLong,
                                                                          startPointLat,
                                                                          endPointLong,
                                                                          endPointLat])

dlong = startPointLong-endPointLong
dlat = startPointLat-endPointLat
a = sin(dlat/2)**2 + cos(startPointLat) * cos(endPointLat) * sin(dlong/2)**2
c = 2 * asin(sqrt(a))
r = 6371
print(c*r)
Bearing = atan2(cos(startPointLat)*sin(endPointLat)-
                sin(startPointLat)*cos(endPointLat)*cos(endPointLong-startPointLong),
                sin(endPointLong-startPointLong)*cos(endPointLat))
Bearing = degrees(Bearing + 360)%360
print(Bearing)
```

```
[illegible]
[illegible]
[illegible]
```

Fig 2.3 Haversine Formula - Python

II. DATA COLLECTION AND TRAINING THE CAR

As explained above, the car is manually controlled first by launching a server on raspberry pi and running a flask application that contains code to allow the client to control the car by clicking on appropriate arrows, while being able to see what the camera sees. Meanwhile, the RPi_Cam_Web_Interface simultaneously runs when the raspberry pi is turned on. Here, the video recording is begun. The car is manually driven for about 5-10 laps, and that much data is then downloaded into the client computer.

The video recorded and downloaded is then converted into picture frames by extracting 10 frames per second with original color (RGB). Figure 3.1 shows the code snapshot that converts the video into a series of frames

Fig 3.1 Snapshot of the converted video to colored picture frames

A. *Flask application*

Flask is a micro web framework [4] written in python, based on Werkzeug toolkit and Jinja2 template engine. It doesn't require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

In the project, the flask application runs on the raspberry pi. Any device connected to the same network can access this server. The forward, reverse, left, right and stop keys are displayed on the webpage which are clicked to control the bot. Within the app, the code calls appropriate motor controlling functions that run the car.

The car is manually driven around the track for 2-3 laps.

Figure 3.2 shows a screenshot of the flask app running on raspberry pi.

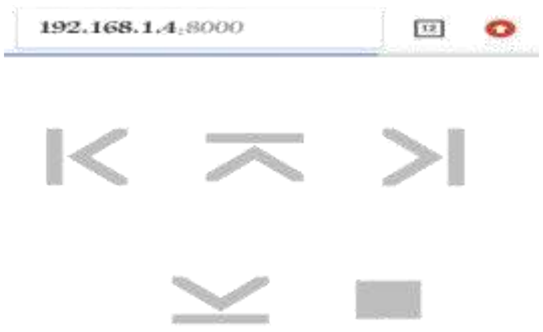


Fig 3.2 flask app screenshot

B. *Streaming video and collecting data*

For streaming real time video, RPi_Cam_Web_Interface is used which is cloned from its Github repository. Unlike many video streaming programs, this particular software allows to stream in real time. There's almost no delay, which is very useful for collecting data for this project.

The video recorded is downloaded into the host computer. The video is then converted to series of images as explained above.

C. *Controlling the motors using Raspberry Pi*

The raspberry Pi controls the motors by controlling the motor driver L293D, which is used to control a maximum of 2 DC motors. This motor driver supports speed control as well by its 'En' pin. The raspberry pi has 40 GPIO pins out of which 6 are used in this project to connect to the L293D. The hardware connection is shown in figure 3.3. Here, to configure the GPIO pins, a python library called RPi.GPIO [5] is used. It allows two methods of configuring the pins –

1. GPIO.PCM – Used to configure the pins as per their GPIO pin names
2. GPIO.BOARD – Used to configure pins as they exist on the board.

In this case, GPIO.BOARD is utilized.

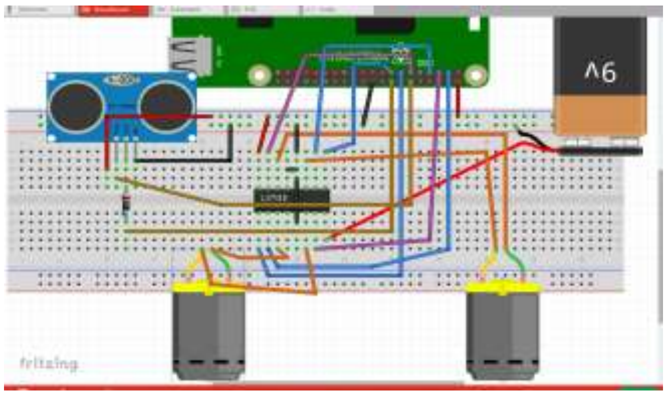
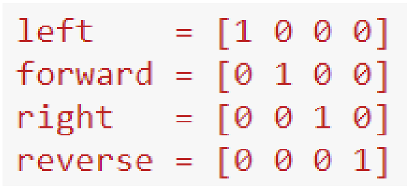


Fig 3.3 connections of the gpio pins

D. *Data sorting and splitting*

Here the collected images are manually sorted out into the left, right, forward and reverse folders respectively.

Once this is done, a program is written that converts images from each of the folders one by one into numpy arrays and labels them accordingly. Figure 3.4 shows the labels defined.



```
left    = [1 0 0 0]
forward = [0 1 0 0]
right   = [0 0 1 0]
reverse = [0 0 0 1]
```

Fig 3.4 direction labels

Suppose, an image is from the folder called 'left', then the Numpy array representing that image is labeled as [1 0 0 0]. In this way, all the images of all the folders are converted in a format mathematically convenient and accordingly labeled.

After the sorting out of data is done, the data has to be split into training data - training labels and validation data – validation labels. It's a good practice as this allows proper testing of the model before actually deploying it. The data is split into 80% training and 20% validation data.

E. Training the car

The car training is done by using OpenCV machine learning library. Here, a fully connected neural network is trained on the data above collected.

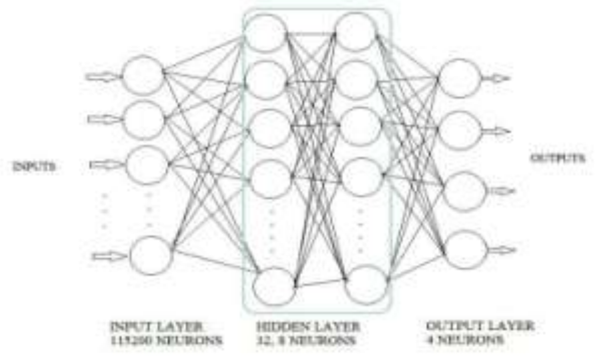


Fig 3.6 model definition

As seen from the figure, the model has 115200 neurons (which is actually the product of the image resolution, in this case 120x320 multiplied by 3 for colored images), in the first layer, followed by 2 hidden layers of sizes 32 and 8 neurons each followed by the final output layer which has 4 neurons; each representing left, right, forward and reverse.

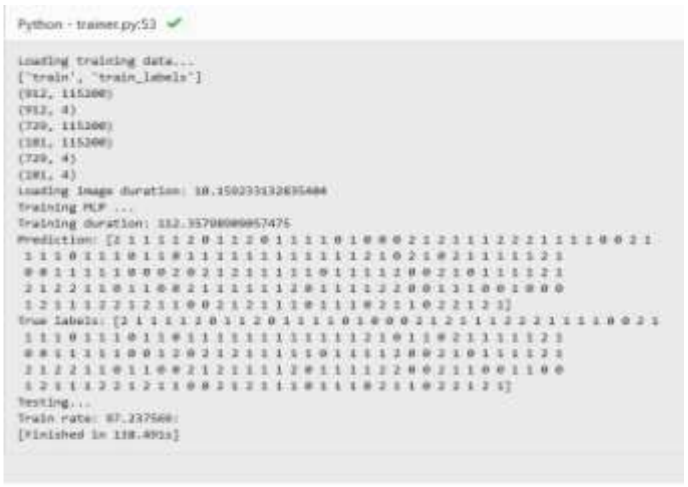
After training the model, it is tested on validation data and the predictions are compared with validation labels, to check its accuracy. With this network model, the project shows an accuracy of 97.54%.

F. Implementing V2V

Vehicle to Vehicle communication is implemented in this prototype only for blind-spot detection. Any time the car is too close to another car, precisely, less than 40cm, the car puts up a pop-up message in the flask app, and slows down a little. The other car connected on the same network, also gets the pop up and can drive safer.

III. CONCLUSION

The ongoing technology of using Li-dar in self-driving cars is very complicated and expensive. In this paper we present a cost effective technology which implements the concept of a self driving car using maps and v2v. Our approach involves the use of machine learning, specifically, artificial neural networks to replace any/all sensors only with a camera, using precise image processing. The accuracy of this prototype is found to be approximately 97% as shown in the figure 4.1. The use of Li-Dar technology is eliminated here and the concept of Artificial Intelligence in Cars is introduced in this paper. We have open-sourced all the technology used for this prototype [6] and it can easily be implemented. The main purpose of this paper is to introduce artificial intelligence in the car with effective image processing to replace sensors with only cameras and enable it to autonomously drive.



Python - trainer.py:53 ✓

```
Loading training data...
['train', 'train_labels']
(912, 115200)
(912, 4)
(729, 115200)
(181, 115200)
(729, 4)
(181, 4)
Loading image duration: 10.150233132635404
Training MLP ...
Training duration: 112.3570800057475
Prediction: [2 1 1 1 1 2 0 1 1 2 0 1 1 1 1 0 1 0 0 0 2 1 2 1 1 1 2 2 2 1 1 1 1 0 0 2 1
 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 2 1 0 2 1 0 2 1 1 1 1 1 1 2 1
 0 0 1 1 1 1 1 0 0 0 2 0 2 1 2 1 1 1 1 1 0 1 1 1 1 2 0 0 2 1 0 1 1 1 1 2 1
 2 1 2 2 1 1 0 1 1 0 0 2 1 1 1 1 1 1 2 0 1 1 1 1 2 2 0 0 1 1 1 0 0 1 0 0 0
 1 2 1 1 1 2 2 1 2 1 1 0 0 2 1 2 1 1 1 0 1 1 1 0 2 1 1 0 2 2 1 2 1]
True labels: [2 1 1 1 1 2 0 1 1 2 0 1 1 1 1 0 1 0 0 0 2 1 2 1 1 1 2 2 2 1 1 1 1 0 0 2 1
 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 2 1 0 1 1 0 2 1 1 1 1 1 1 2 1
 0 0 1 1 1 1 1 0 0 1 2 0 2 1 2 1 1 1 1 1 0 1 1 1 1 2 0 0 2 1 0 1 1 1 1 2 1
 2 1 2 2 1 1 0 1 1 0 0 2 1 2 1 1 1 1 2 0 1 1 1 1 2 2 0 0 2 1 1 0 0 1 1 0 0
 1 2 1 1 1 2 2 1 2 1 1 0 0 2 1 2 1 1 1 0 1 1 1 0 2 1 1 0 2 2 1 2 1]
Testing...
Train rate: 97.237569:
[Finished in 118.491s]
```

Fig 4.1 Training, Testing and Accuracy of the self-driving car

REFERENCES

[1]. https://github.com/silvanmelchior/RPi_Cam_Web_Interface
[2]. https://github.com/RyanZotti/Self-Driving-Car.
[3]. https://en.wikipedia.org/wiki/Great-circle_distance
[4]. https://en.wikipedia.org/wiki/Flask_(web_framework)
[5]. https://pypi.org/project/RPi.GPIO/
[6]. https://github.com/KhyatiMehta3/AutRcCar
[7]. https://www.nature.com/articles/nature14236 : Human - level control using deep reinforcement learning.
[8]. https://www.citylab.com/life/2014/04/first-look-how-googles-self-driving-car-handles-city-streets/8977/ : First look on how google's self driving car handles city streets.
[9]. https://ai.intel.com/demystifying-deep-reinforcement-learning/ : Demystifying deep reinforcement learning.
[10]. "Build an Autonomous R/C Car with Raspberry Pi" by Adam Conway & William Roscoe.
[11]. https://www.pyimagesearch.com/2017/10/02/deep-learning-on-the-raspberry-pi-with-opencv/
[12]. http://socialledge.com/sjsu/index.php/F16:_Titans
[13]. https://github.com/BoltzmannBrain/self-driving
[14]. "Long-term Planning by Short-term Prediction" By Shai Shalev-Shwartz, Nir Ben-Zrihem, Aviad Cohen & Amnon Shashua
[15]. "End to End Learning for Self-Driving Cars" By Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D., Jackel Mathew, Monfort Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao & Karol Zieba.
[16]. "Programming a self driving car" By Onishim Hasdak.
[17]. https://github.com/multunus/autonomous-rc-car
[18]. https://learn.adafruit.com/adafruit-ultimate-gps-on-the-raspberry-pi/setting-everything-up