# Major Project Report

On

# SELF DRIVING RC CAR

submitted in partial fulfillment of the
requirements for the award of the degree of

## Bachelor of Technology
in
## Electrical and Electronics Engineering

by

156724 15EE142 **RAVEESH SINHA**
156147 15EE151 **VILOHIT SARMA KAZA**
156135 15EE239 **RUPASHI SANGAL**

Under the Guidance of

**DR. DEBASHISHA JENA**
Dept. of Electrical and Electronics Engineering
National Institute of Technology Karnataka


**Department of Electrical and Electronics Engineering**
**National Institute of Technology Karnataka, Surathkal**
**2018-2019**

# Abstract

Self-driving cars are a state of the art technology in the field of Artificial Intelligence, which has gained a lot of popularity for its usefulness in real life situations. High level implementation of such projects uses the steering angle, GPS installed and many other car sensor related data to accurately predict the direction of motion of the car. A low level implementation of such an idea would be a self-driving RC car wherein the images are captured and the corresponding remote control data is used for training the car and thereby, testing it on various tracks. The current project focuses on a comparative study of the use of Artificial Neural Networks and Convolutional Neural Networks on the RC Car, using minimal hardware while obtaining the best results.

# Declaration

We hereby declare that the Project Work Report entitled **SELF DRIVING RC CAR** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** for the award of the degree of **Bachelor of Technology** in **Electrical and Electronics Engineering** is a *bonafide report of the work carried out by us.* The material contained in this Thesis has not been submitted to any University or Institution for the award of any degree.

*Register Number, Name and Signature of the Student:*
(1) 156724 15EE142 Raveesh Sinha
(2) 156147 15EE151 Vilohit Sarma Kaza
(3) 156135 15EE239 Rupashi Sangal

Department of Electrical and Electronics Engineering
PLACE : NITK, Surathkal
DATE : 29 April 2019

# Certificate

This is to *certify* that the Undergraduate Thesis entitled **SELF DRIVING RC CAR** submitted by *Raveesh Sinha* 156724 15EE142, *Vilohit Sarma Kaza* 156147 15EE151, *Rupashi Sangal* 156135 15EE239 as a record of the work carried out by them is *accepted as the B.Tech Thesis submission* in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Electrical and Electronics Engineering**.

Dr. Debashisha Jena
Associate Professor
Dept. of Electrical and Electronics Engineering
National Institute of Technology Karnataka

Dr. B Venkatasa Perumal
Head of Department

## Acknowledgments

We would like to thank the Department of Electrical and Electronics Engineering, National Institute of Technology Karnataka for providing us the opportunity and funding for the project. We would like to express our sincere gratitude to Dr. Debashisha Jena for his invaluable guidance and suggestions throughout the course of this project. We would also like to thank Mr. K.M. Naik for helping us acquire the electronics components.

# Contents

# List of Figures

# 1. Introduction

With the advent of Machine Learning and Artificial Intelligence, what seemed like a far off thought is now a reality: *Self-driving cars*. Various machine learning algorithms are used extensively in the development of self-driving cars. The number of sensors/systems used in any modern day car is unimaginable: accelerometer, GPS system, sonar, lidar, Electronic Control Unit(ECU) as a whole; are some of the few systems used in a car. Logging the sensor data provides a huge amount of data which proves useful in the implementation of autonomous cars. An AI based voice guide while constantly guiding the driver to the desired destination, is also helpful in suggesting nearby locations of interest to the driver. This reduces the trouble of navigating and searching for the destination, once the mapping system is setup. Also, with the rapid rise in the population, power of purchasing, automobile industry and consequently, traffic on a daily basis, autonomous cars are powerful as they aim to minimize the human reaction time error, road rage, inability to adhere to the traffic rules, which lead to road accidents. To sum it up, autonomous cars reduce the human effort and error by ensuring proper driving of the cars.

On the downside, the models are not completely accurate and could predict a wrong direction once in a while. Added to that is the fact that if the streets are too narrow, the car might have trouble navigating itself. The sensor data is vulnerable to changes based on the weather or internal damage. Since, the cars are now autonomous, the employment opportunities of drivers could get affected and the government needs to find other means to accommodate jobs for them. There have been reports of failed autonomous cars on the run leading to accidents. At the same time there have been reports regarding the usefulness of the cars when the driver is not in a position to drive.

The current work suggests a small scale implementation of a self driving car in the form of a *self-driving RC car*. The RC car is tested on both Artificial Neural Network and Convolutional Neural Networks. A hardware model consisting of an RPi and Arduino is discussed first and the final model after eliminating the Arduino is presented. The RPi is used to tap into the controller module of the RC car along with a PiCamera to capture the live-feed of the track and the images are sent to the Neural Network through a TCP server-client based program and the predicted direction is sent back to the RPi which commands the direction to the car. A training accuracy of *97%* and a testing accuracy of *92%* was obtained using the final CNN.

# 2. Related literature

## 2.1 Fundamentals of Artificial Neural Network

*Artificial neural networks (ANNs)* are information processing systems that have been developed as generalizations of mathematical models of human cognition or neural biology. A neuron is the basic unit of operation of ANN. The block diagram given below shows the model of a neuron. The three basic elements of the neuronal model are:

- A set of synapses or connection links which is characterized by a weight of its own. The input signal $x_j$ is passed between neurons via these connection links after being multiplied by the synaptic weight $w_{kj}$.
- An adder for summing the input signals, weighted by the respective synapses of the neuron also known as a linear combiner.
- An activation function for limiting the amplitude of the output of a neuron.

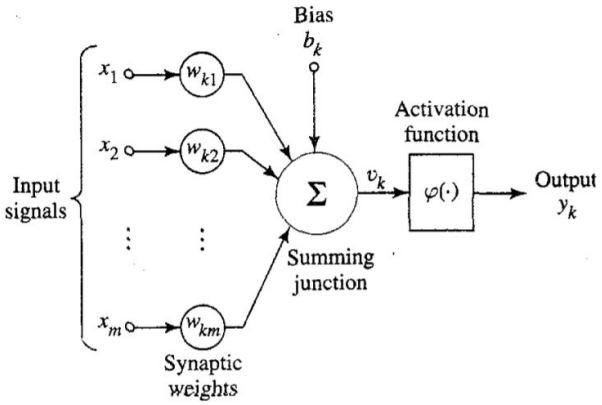


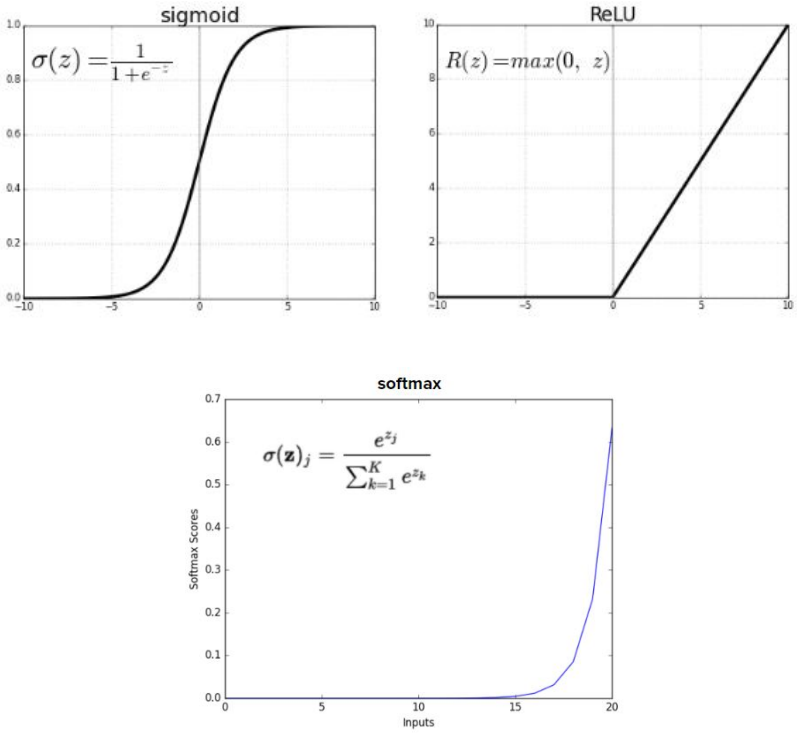

**Fig. 2.1.** Nonlinear model of a neuron (from [1])

The neuronal model also includes an externally applied bias denoted by $b_k$. The bias has the effect of increasing or lowering the net input of the activation function. It acts like a negative of a threshold. The mathematical representation of ANN is as follows:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

$$y_k = \phi(u_k + b_k) \tag{2.2}$$

**Fig. 2.2.** Some important activation functions




*Sigmoid function:* The sigmoid function, whose graph is s-shaped, is the most common form of activation function used in neural networks. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior.

*ReLU function:* The rectified linear unit activation function returns 0 if it receives any negative input, but for any positive value x it returns the same value back. One major benefit of ReLU activation function is the reduced likelihood of the gradient to vanish. This arises when the value of activation $> 0$. The constant gradient of ReLUs results in faster learning. The other benefit of ReLUs is sparsity. Sparsity arises when activation $\leq 0$.

*Softmax function:* The softmax function, also known as the normalized exponential function, is used in various multiclass classification methods, such as multinomial logistic regression, multi class linear discriminant analysis, naive Bayes classifiers, and artificial neural networks. In ANNs, it is often used in the last layer to represent the probabilities of every output node.

**Multilayer perceptrons (MLPs):** Typically, a network consists of a set of sensory units (source nodes) that constitute the input layer, one or more hidden layers, and an output layer. The input signal propagates through the network in a forward direction, on a layer-by-layer basis. These

neural networks are commonly known as multilayer perceptrons which represent a generalization of single-layer perceptrons (Fig. 2.1). MLPs have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with a highly popular algorithm known as error back-propagation algorithm.

**Back-propagation algorithm:** This algorithm consists of two passes through the different layers of the network, a forward pass and a backward pass. In the forward pass, an input vector is applied to the sensory nodes of the network, and its effect propagates through the network layer by layer. Finally, a set of outputs is produced as the actual response of the network. During forward pass, the synaptic weights of the networks are all fixed. During the backward pass, the synaptic weights are all adjusted in accordance with an error-correction rule. Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal. This error signal is then propagated backward through the network against the direction of synaptic connections - hence it is called "error back-propagation". The synaptic weights are adjusted to make the actual response of the network move closer to the target response.

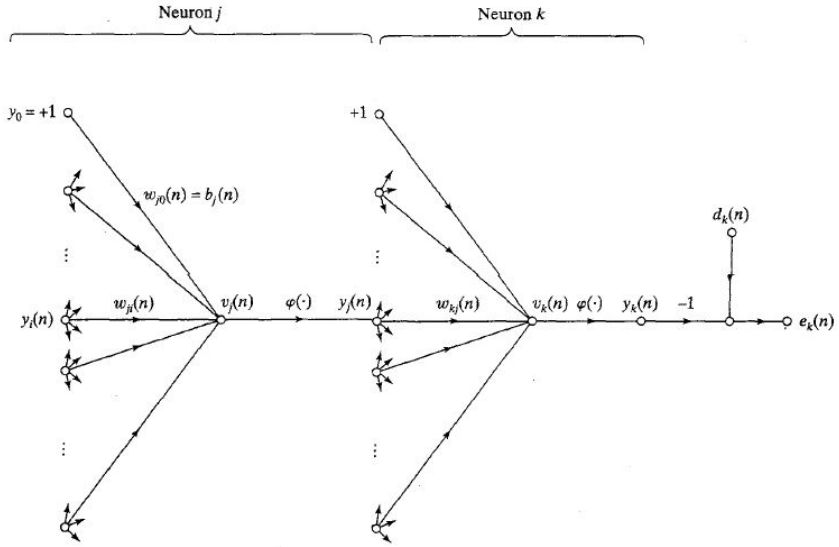



**Fig. 2.3.** Signal-flow graph of forward pass (from [1])

*Mathematical model of back-propagation algorithm:*
The error signal at output of node j (an output node) at the nth iteration is defined by

$$e_j(n) = d_j(n) - y_j(n) \tag{2.3}$$

The instantaneous value E(n) of the total error, when number of output nodes is C, is given as:

$$E(n) = \frac{1}{2}\sum_{j=1}^{C} e_j^2(n) \tag{2.4}$$

The local error gradient (from chain rule) is given by:

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = e_j(n)\phi_j'(v_j(n)) \tag{2.5}$$

Case 1: When neuron $j$ is an output node, it is supplied with a desired response of its own. The error signal $e_j$(n) can be computed using Eq.(2.3). This value can be plugged into the Eq. (2.5) directly to compute the local gradient.

Case 2: When the neuron $j$ is hidden, the back-propagation formula for the local gradient is derived as:

$$\delta_j(n) = \phi_j'(v_j(n))\sum_k \delta_k(n) w_{kj}(n) \tag{2.6}$$

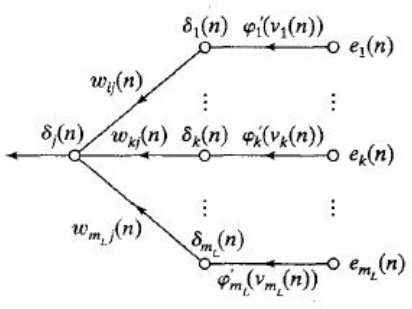



**Fig. 2.4.** Signal-flow graph of backward pass (from [1])

Generalized equation:

$$\begin{pmatrix} \textit{Weight} \\ \textit{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \textit{learning-} \\ \textit{rate parameter} \\ \eta \end{pmatrix} \cdot \begin{pmatrix} \textit{local} \\ \textit{gradient} \\ \delta_j(n) \end{pmatrix} \cdot \begin{pmatrix} \textit{input signal} \\ \textit{of neuron } j \\ y_i(n) \end{pmatrix}$$

## 2.2 Fundamentals of Convolutional Neural Network

Convolutional Neural Network is a very popular Deep learning technique used in the field of Artificial Intelligence and machine learning. The *deep* in deep learning isn't a reference to any kind of deeper understanding achieved by the approach; rather, it stands for the idea of successive layers of representations. The number of layers that contribute to a model of the data is called the depth of the model. A typical convnet is described by the following layers:

- *The convolutional layer:* Convolutions operate over 3D tensors, called feature maps, with two spatial axes (height and width) as well as a depth axis (also called the channels axis). For an RGB image, the dimension of the depth axis is 3, because the image has three color channels: red, green, and blue. For a black-and-white picture, the depth is 1 (levels of gray). The convolution operation extracts patches from its input feature map and applies the same transformation to all of these patches, producing an output feature map. This output feature map is still a 3D tensor: it has a width and a height. Its depth can be arbitrary, because the output depth is a parameter of the layer, and the different channels in that depth axis no longer stand for specific colors as in RGB input; rather, they stand for filters. Convolution layers learn local patterns, for example, in the case of images, patterns found in small 2D windows of the inputs.

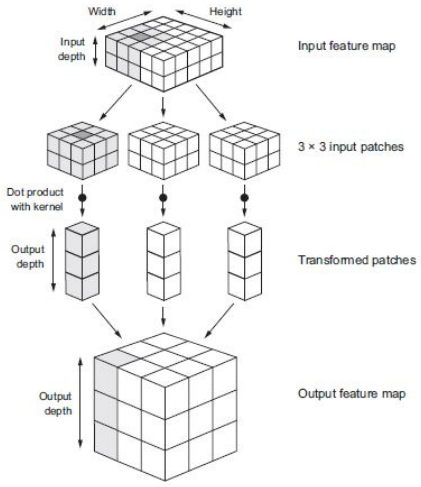



**Fig. 2.5.** Convolution operation (from [3])

- *The max-pooling layer:* The role of max pooling layer is to aggressively down sample feature maps in order to reduce the number of feature-map coefficients to process, as well as to induce spatial-filter hierarchies. It consists of extracting windows from the input feature maps and

outputting the max value of each channel. It is usually done with 2 x 2 windows and stride 2 (Fig. 2.6).

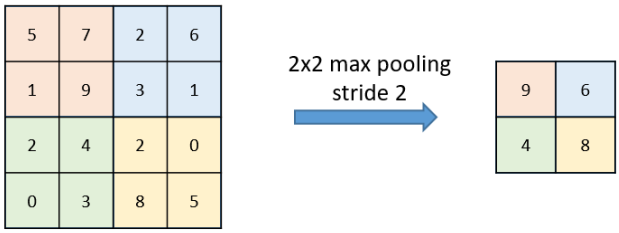



**Fig. 2.6.** Max pooling

- *Dense layer:* The input tensor goes through a series of convolutional and max pool layers post which the 3D outputs are flattened to 1D, and fed into a densely connected classifier network: a stack of Dense layers. The number of nodes in the output layer of this stack corresponds to the number of classes to be predicted in the problem statement.

## 2.3 Previous work on self driving cars

*iDriver - Human Machine Interface for Autonomous Cars [4]:* An efficient human machine interface is essential for autonomous cars to monitor car motion and path which is used in the decision making process. In this paper, a software called "iDriver" is described which is designed for an iPad and works as a remote control system. Two research cars namely 'Spirit of Berlin' and 'Made in Germany' are used for testing this software. Various sensors are mounted on the top of these cars which provide input to the software. These include GPS antennas for car position and direction, video cameras for road surface markings and laser scanners for object detection. iDriver has a client-server architecture where the software acts as a client and receives the processed sensor data from the server (connected to car using Ethernet) over Wi-Fi. iDriver then displays a 3D simulation of the car scene along with the current car state such as speed, gear state, revolutions per minute, heading and turn signals etc. Using this information, commands are transmitted to the server over TCP for autonomous control.

*Design and Implementation of Autonomous Car using Raspberry Pi [5]*: The paper discusses a way to drive autonomously using Raspberry Pi, PiCamera and ultrasonic sensor. Raspberry Pi does the processing of the feed from camera for lane detection using openCV Python and controls the motion of car using motor controller circuit made using L293D. The ultrasonic sensor is used for collision avoidance. RPi is connected to laptop through Wi-Fi. The prediction algorithm used

is a hybrid of two lane detection techniques, namely, feature based and model based. The feature based technique combines low level features, such as painted lines, or lane edges etc. The feature based lane detection algorithm fails if the road is not well painted or lacks bold edges. The model based technique works on the shapes assumed by roads , could be straight lines or parabolic curves. The algorithm used:

- Extract the color range for road.
- Define a region of interest.
- Convert the region of interest into a simple shape.
- Determine the shape of road.
- Filter the noise.
- Make it robust.
- Determine the turns and move the car.

The car is controlled by a mobile app. Obstacle avoidance feature is added using ultrasonic sensor. Algorithm for lane detection and movement does not use Neural Networks but can be enhanced with neural network. Results only show road detection with no details about accuracy and working in real-time situation.

*Neural controller of autonomous driving mobile robot by an embedded camera [6]:* This paper proposes an autonomous driving mobile robot system that uses Artificial Neural Network (ANN) for control. The hardware structure consists of 3 subsystems:

- Input unit: A Raspberry Pi camera module used for data collection. This camera can acquire resolution images up to 3280 x 2464 and supports 1080p30, 720p60 and 640 x 480p90 videos with 15 fps frame rate. This camera is connected to RPi via CSI port.
- Processing unit: Raspberry Pi 3 version B is used which contains the neural network algorithm that makes prediction based on the input image. The control signals are sent to the control unit of the mobile robot based on this prediction.
- Power unit circuit: This consists of the motor unit having 2 DC motors which make up the mobile robot and a power interface system based on L298N motor driver. The GPIO pins from RPi are connected to the rear and front motors via L298n. This enables the user to control the forward/reverse direction and speed of motor (by adjusting frequency and duty cycle of PWM output on the enable pin).

Neural network Architecture: Input images are converted to gray scale and lower half of image size (320 x 120) is used. Number of input nodes is 38400 (corresponding to pixel values), number of

nodes in single hidden layer is 20 and number of output nodes is 4 (forward, reverse, right and left). Activation function used is sigmoid function as it represents values between 0.05 and 0.95 which is better suited for nonlinear systems. Logistic regression cost function is used as it is a convex function which makes it easily reducible. Learning algorithm is back-propagation.
Result: 96% accuracy with 2000 training images.

*Working model of self-driving car using CNN, RPi and Arduino [7]:* Instead of using just GPS and image processing, the paper focuses on using Machine Learning and Image Processing wherein the camera image is fed into the ANN, which helps in deciding the direction, and the signal is sent through the controller to the car. The computer and RPi3 are hosted on the same network. The image captured on the PiCamera (5MP) is converted into grayscale and scaled, based on the size required, before feeding it into the NN through the RPi. The images are captured after driving the car through the path and recording the videos (24 total), which later are placed in folders based on their label(L, R, F, Stop). 320*240 size of images was used with 128 input nodes, 32 nodes in the 2 hidden layers and 4 output nodes(direction). The activation functions used are: ReLU(input and hidden layers) and softmax function (hidden and output layers). Number of epochs and the batch size used were 3 and 10 respectively. Also, to avoid overfitting a dropout of 0.5 was set.

*Autonomous Driving Car Using Convolutional Neural Networks [8]:* In this paper, an open world game called Grand Theft Auto 5 is used for data extraction, an Inception V3 model which uses CNN is utilized for training and the final output is simulated.

- Input Data: GTA V contains 250+ different types of vehicles, many roads and bridges, traffic signals etc. There is an in-game AI which drives the car by itself to the way point set by the user on the map. This in-game AI enables extraction of dataset which includes current frame, throttle, brake, steering angle, location and driving mode.
- Neural Network: CNN Contains large number of hidden layers thus avoiding extra processing. Stochastic gradient descent (SGD) algorithm along with Nesterov momentum is used for optimization to accelerate SGD.
- Training: The network is given frame as input and steering angle (ranging from -1 to 1 which is up scaled to 0 - 999) as target, keeping the other parameters constant. Based on training data, the network predicts the steering angle. This output is sent to the in-game AI to simulate final result. Data imbalance reduces accuracy. Image processing helped reduce training time.

*NVIDIA - End to End Learning for Self Driving Cars [9]:* Due to advancement of neural network and related fields, CNN has proved to be a powerful tool for training a network with as little

human intervention as possible. In this paper, the network does not use human specific features like lane markings or other cars, but learns everything internally with optimization techniques also embedded into the network, thereby almost eliminating human based inputs. The main aim is to use the steering commands and the pixel values captured by the camera mounted on the car and map them both to obtain meaningful results. The images are fed into the CNN and a steering command is obtained, which is then compared with the required output based on the training data and weights are adjusted accordingly. Torch 7 machine learning package is used for back-propagation. During training, 3 cameras are used Left, Right and Centre and images are adjusted accordingly for a single centre camera mount for the period after training. Other than human driven data, additional data is augmented to help it learn from the mistakes thereby preventing it going off road. 9 layers are used in the network architecture: 5 convolutional, 1 normalization and 3 fully-connected layers. A frame rate of 10fps is selected and the training data is input based on the lane driving function of the car.

*Obstacle Detection and Object Size Measurement for Autonomous Mobile Robot Using Sensor [10]:* Ultrasonic sensor and Camera used to feed input to RPi which in turn outputs control signals to motor driver circuit. The measurement of object size is done as follows:

$$Distance = speed * timefieldofview(\theta) \tag{2.7}$$

$$h = 2 * x * tan(\theta/2) \tag{2.8}$$

where h is horizontal viewing length at distance x. If m pixels denote length h, and object takes q pixels in image, object length horizontally becomes:

$$p = (h * q)/m \tag{2.9}$$

Camera image is converted to grayscale, thresholded to distinct the object, opened and closed to eliminate noises. Only drawback is distortion of lens and error in calculating distance due to quantization.

*Obstacle Avoidance with Ultrasonic Sensors, IEEE Journal of Robotics and Automation [11]:* This paper discusses the obstacle avoidance algorithm used for a mobile robot. Since the algorithm depends heavily on the performance of the ultrasonic range finders, these sensors and the effect of their limitations on the obstacle avoidance algorithm were discussed as well. Paper shows how orientation of obstacles may give false results and makes a algorithm to tackle this condition. But it uses lights in room for location as the mobile robot is designed to be used in closed room which isn't possible for us. The obstacle sensing and avoidance algorithm can be used to improve performance of self driving car.

# 3. Hardware

The hardware consists of an RC car, Raspberry Pi 3 Model B+ as computing unit, PiCamera Version 2 for capturing track images, and power supplies for car and Raspberry Pi. The Raspberry Pi is mounted on top of the car along with the PiCamera module to capture the front view of the car. A compact portable charger is used to power the Raspberry Pi.

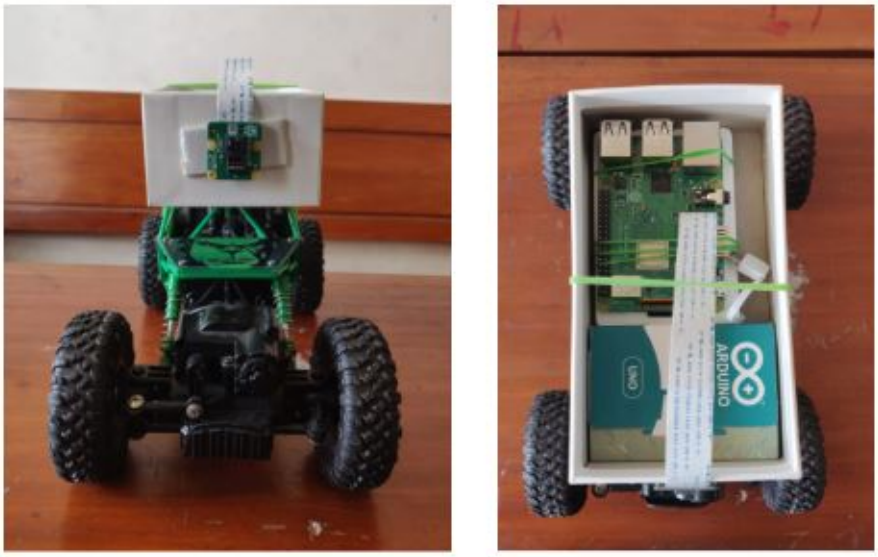



**Fig. 3.1.** RC Car Setup: Front view and Top View

Two models for driving the RC car were proposed in this project. A detailed description of the components used, models and a comparison for the selection between the two models has been discussed in this section.

## 3.1 Components

### 3.1.1 Raspberry Pi 3 Model B+

This is one of the newest models of RPi with 1.4GHZ 64-bit quad-core processor and 1GB RAM. It uses an SD Card for loading the operating system and for storage purposes. Working on a 5V, 2.5A power supply (DC), it supports remote access over internet, SSH, SFTP, SCP, SSHFS, rsync and FTP. It also boasts a GUI based operating system which has to be setup during the first run and

then the GUI can be viewed on laptop with VNC viewer. It also offers 40 GPIO (General-Purpose Input/Output) pins which gives it micro-controller capabilities with computation capabilities. A power bank of small size was used to power up the RPi.

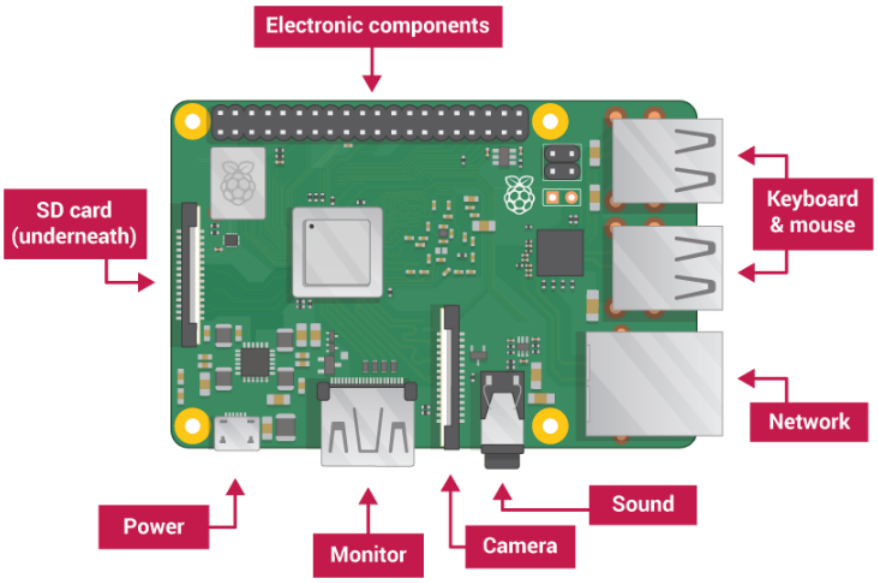



**Fig. 3.2.** Raspberry Pi 3 B+

### Setup

To setup the RPi we require: power supply, SD card, keyboard, monitor and a mouse. The following steps need to be followed to setup the RPi:

- Download NOOBS (zip file) from the Raspberry Pi website, extract the zip file, insert SD card in laptop and format the SD card to ensure full space and paste all the files from zip to the SD card. Now eject the SD card and insert it in the Raspberry Pi.
- Connect mouse, keyboard and monitor to the Raspberry Pi. Now connect the power supply (5V DC) and a red light turns on indicating power supply is connected.
- After the RPi boots up, the NOOBS installer window is displayed on screen. Install Raspbian following which the Raspbian desktop can be accesssed.
- Configure the RPi to set date, time, location.

**Softwares Installed**

- **VNC Viewer:** to access the Graphical User Interface (GUI) remotely.
- **WinSCP:** to transfer files between laptop and RPi using Secure File Transfer Protocol (SFTP).
- **Putty:** to SSH into RPi from laptop.

### 3.1.2 PiCamera

This Raspberry Pi camera module, launched in 2016, contains Sony IMX219 8-megapixel sensor and can be used to take videos up to 1080p at 30fps as well as can be used to capture still images. A CSI port is provided on RPi to directly interface this camera module with a ribbon cable. Python provides an in-built library for PiCamera which is used in this project. The PiCamera is used at a resolution of 640 x 480 at 10fps and was attached to the box which is mounted on top of the RC Car.

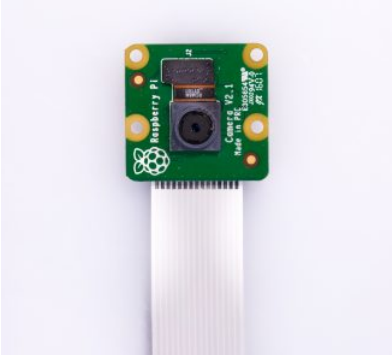


**Fig. 3.3.** PiCamera V2

### 3.1.3 RC Car

Remote Controlled car uses RF transmitter and receiver for communication. The remote controller was opened and the RF transmitter was tapped to control direction signals. A box is mounted over the car to contain RPi and fix the PiCamera in position. The RF transmitter circuit is also interfaced with RPi and contained in this box. RC Car used is a rock crawler with remote controller range of more than 20m and rechargeable battery.

**Fig. 3.4.** Remote Controlled Car

### 3.1.4 Arduino UNO

Arduino UNO is based on ATMega 328P [13] microcontroller which provides digital and analog pins for input/output. It operates at 5V and is used to interface the RF transmitter in the early stages of the project, but was discontinued later to decrease communication delay. Since, the RF transmitter works at 3.3V, a voltage divider circuit was used to bridge the gap between the two. Arduino is powered and controlled (via serial communication) by laptop.

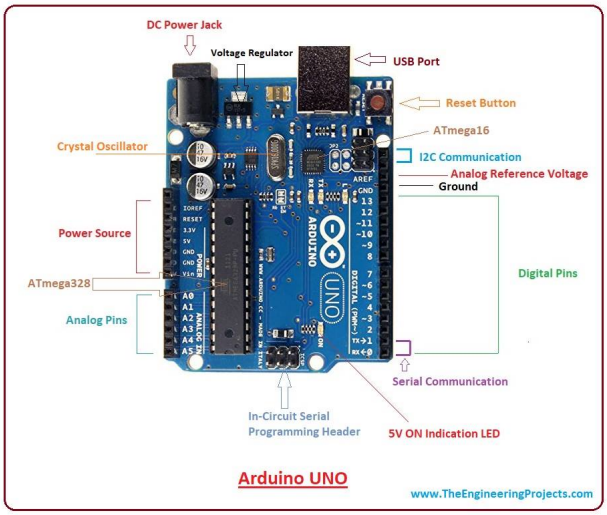



**Fig. 3.5.** Arduino Uno

### 3.1.5 Voltage Divider Circuit

A voltage divider circuit is used as the Arduino works on 5V while RF transmitter at 3.3V. To interface the two circuits, a voltage divider circuit to reduce 5V to 3.3V is used. Resistors used are 1kΩ (connected to arduino digital pin) and 2.2kΩ (connected to ground) and output is taken from the connecting point of these two resistors.

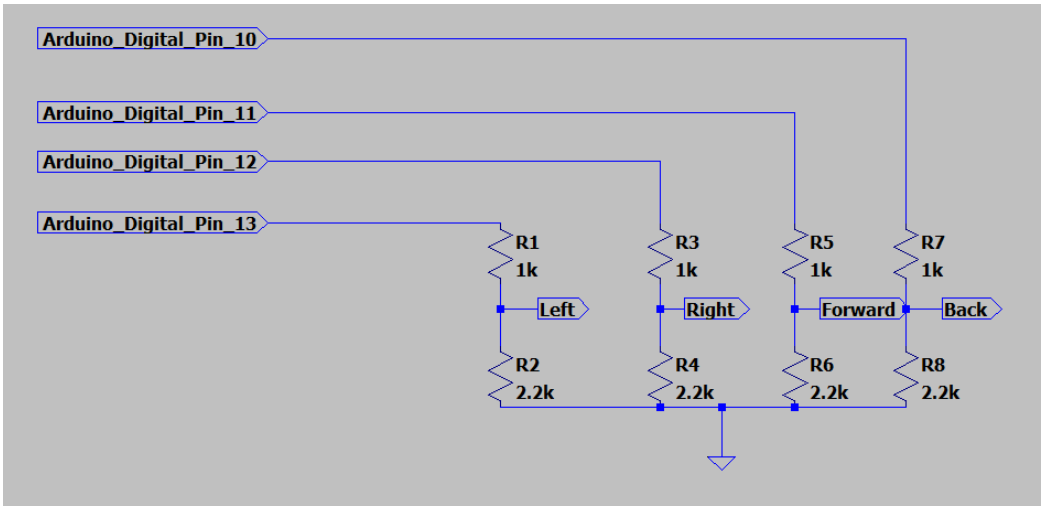



**Fig. 3.6.** Schematic Diagram

## 3.2 Model 1 - Arduino based controller

In this model, Arduino is used to control the car. The arduino was interfaced with the RF transmitter, taken from the remote controller of the RC car, by tapping the direction buttons with help of the voltage divider circuit. In the controller, 4 push buttons were given to control the four directions. When the push buttons are pressed, they pull down the connected circuit causing current to flow through the circuit. The RF transmitter circuit was tapped using Arduino and a voltage divider circuit, and to control the direction, the digital pin in arduino corresponding to that direction was made low. The signal is received by RF receiver on the car and the motors are controlled internally for moving the car. The direction input is taken from laptop keys using *Pygame* library available in Python. This input signal is sent to arduino and through RF transmitter-receiver the direction control is implemented.

Raspberry Pi was installed on the car for running PiCamera which captures the images to be fed to the neural network. A small low-weight power bank is used as supply power to RPi.

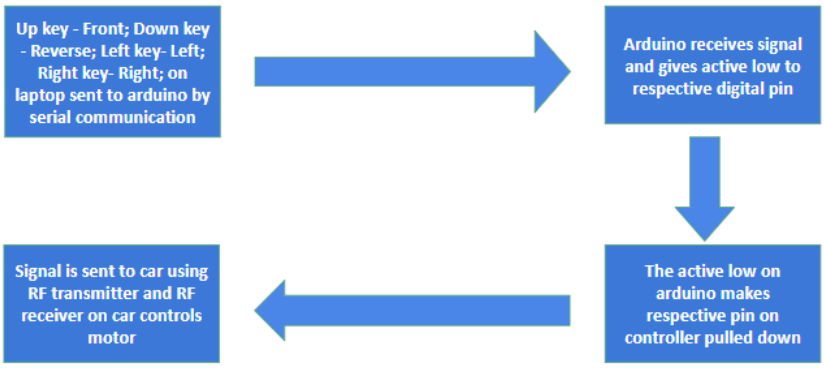



**Fig. 3.7.** Arduino Based Remote Controller

## 3.3 Model 2 - Raspberry Pi based controller

In this model, the RF transmitter circuit has been directly interfaced with the RPi. The RF transmitter is mounted on the car itself. When a direction input is given using the pygame library, the corresponding GPIO pin on the RPi is turned low, resulting in activating the circuit and running the car. The pygame input module for direction is also running on RPi itself for faster response from the circuitry. A battery pack is used to power the RF transmitter circuit taken from the original controller while power bank is used as supply for RPi.

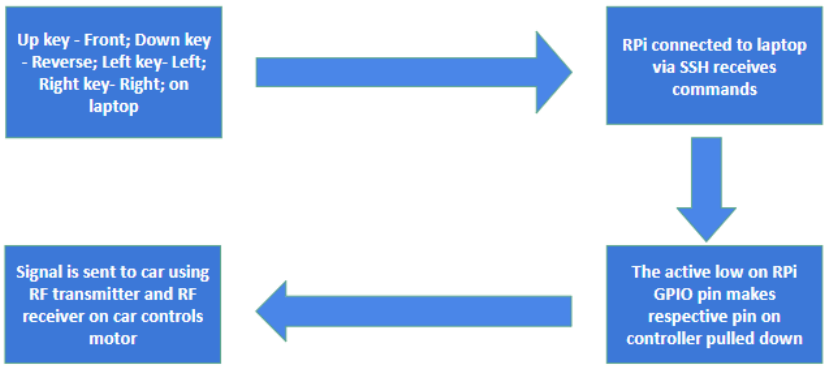



**Fig. 3.8.** RPi Based Remote Controller

## 3.4 Model 1 vs Model 2

Raspberry Pi based controller model (Model 2) gives us several advantages over the Arduino based model and thus, has been used finally for the self driving RC car model. Firstly, the communication delays in the controller have been reduced significantly. In Arduino based model, the communication channel being RPi to Laptop (via TCP/IP), to Arduino (via serial communication), to RF transmitter and then to RF receiver while the RPi based model sends signal to RF module. This reduces the communication channel and hence, the communication delays. Secondly, it reduces the complexity of the circuit as RPi provides 3.3V on its GPIO pins and thus, the voltage divider circuit and Arduino have been removed from the circuit. Since, the extra electronic circuitry is reduced, the car is tidier as a final product. Removing excess circuitry also helps in reducing the noise in the direction signal. Also, the use of an RPi based controller can make the model self-sufficient. If a microcomputer with sufficient RAM can be used to load the CNN, the model will become self sufficient and would not need a laptop for processing of direction prediction through CNN. The above mentioned reasons play a major role in the final output and thus Raspberry Pi based controller model has been used.

# 4. Software

## 4.1 Dataset

MAC0460 Self driving dataset available on *Kaggle* [15] is used for building the model. The data was collected by driving the robot car along a circular track. It consists of 1,02,496 track images in RGB colorspace. Each image has a dimension of 45 x 80 x 3 and a corresponding label ('0', '1' and '2') representing a command for the robot car ('0' = forward, '1' = left, '2' = right).

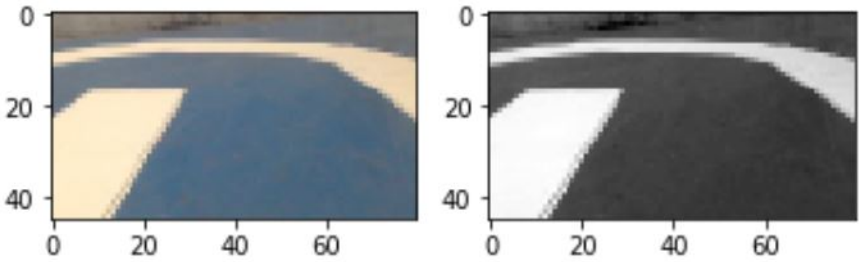



**Fig. 4.1.** Image of a track and its grayscale version

Each image is converted to grayscale and then fed to the network as a colored image does not play an important role in this problem statement. This reduces the input dimension and makes the training process computationally efficient. So, the overall processing time is decreased.

The dataset is divided in 80:20 ratio into training and validation sets respectively. A separate dataset consisting of 1000 images is available for testing the trained model. The exact distribution of data is given in the table below.

| Dataset Distribution | |
|---|---|
| **Training Set** | 81996 |
| **Validation Set** | 20500 |
| **Testing Set** | 1000 |

## 4.2 Artificial Neural Network Model

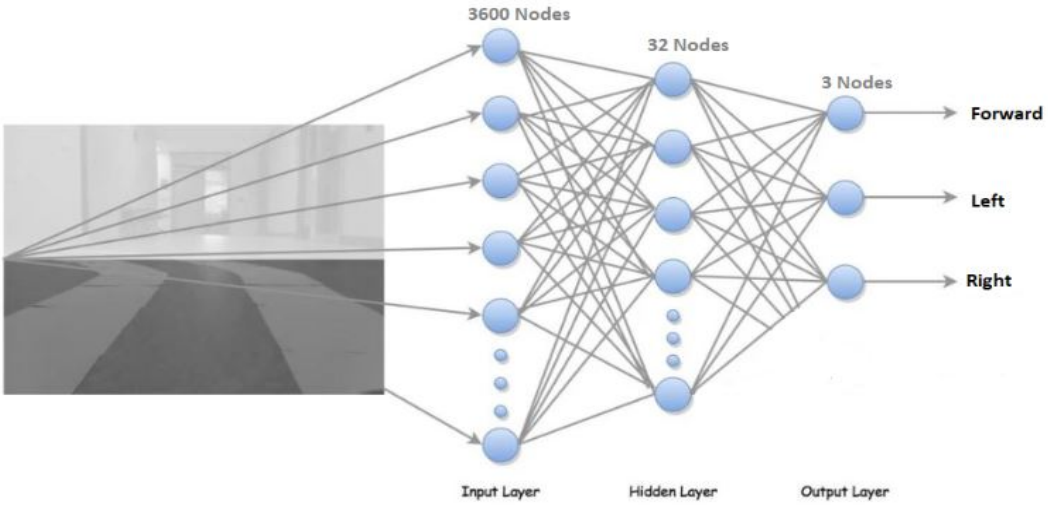



**Fig. 4.2.** ANN Architecture

The neural network has one input, one hidden and one output layer. The input layer has 3600 nodes. The output layer has 3 nodes corresponding to the direction predicted. Many experiments were conducted by varying the the number of nodes in the hidden layer (32, 64, 128 ..) and the best results were obtained with 64 nodes.

The model was trained using backpropagation algorithm and sigmoid activation function. The learning rate was taken as 0.05 and batch size as 5. Optimizer used was *stochastic gradient descent*. The trained model was tested on the set consisting of 1000 unseen images. An accuracy of 82.4% was obtained. The learned weights were saved in a csv file. Since the accuracy obtained was not high, a CNN model was designed to get better results.

## 4.3 Convolutional Neural Network Model

The CNN Model designed takes a 45 x 80 grayscale image as input. It has 3 convolutional layers with 32, 64 and 64 kernels respectively. The kernel size is 3 x 3. The third convolutional layer is flattened and fed to the Fully Connected Layer. The final output layer uses softmax activation function and has 3 nodes corresponding to the direction predicted (forward, left or right). The model was trained using backpropagation algorithm and ReLU activation function with a batch size of 5. Optimizer used was *adam*. Pooling layers are avoided as the input is small in size.

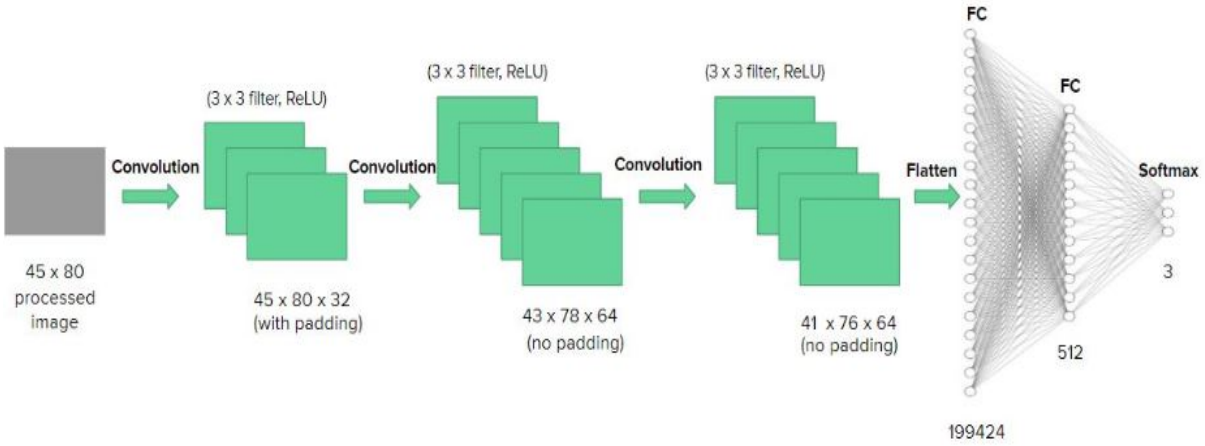



**Fig. 4.3.** CNN Architecture

## 4.4 Training Results and Comparison

| Model | Epochs | Training Accuracy | Validation Accuracy | Testing Accuracy |
|---|---|---|---|---|
| **ANN** | 25 | 82.39% | 81.68% | 82.6% |
| **CNN** | 5 | 91.9% | 89.37% | 89.3% |
| **CNN** | 10 | 97.64% | 91.09% | 92% |

*The ANN model* starts training with an initial loss of 0.5348 which reduces to 0.4433 over the course of 25 epochs. In this period, the accuracy increases from 78.56% in the first epoch to **82.39%** in the final epoch. Hence, it can be concluded that the learning is very slow and saturates quickly.

*The CNN model* begins training with an initial loss of 0.4874 which dips to 0.0685 in 10 epochs. Correspondingly, the accuracy increases from 80.76% to **97.64%**. Hence, the learning is significant as the loss is drastically reduced resulting in high accuracy. The validation loss decreases till 8 epochs and then starts increasing which indicates that the model is beginning to overfit and training must be stopped. The total training time is 4.5 hours on Tesla GPU. The trained model is saved as a .h5 file.

The confusion matrix, in general, depicts that the images labelled as 'forward' have a low accuracy of being correctly predicted in comparison to other two directions. This could possibly be because our dataset is collected using only circular tracks which has a majority of 'left' and 'right' directions.

The CNN model trained with 10 epochs gives the best result and hence, this model is integrated with the hardware for self driving.

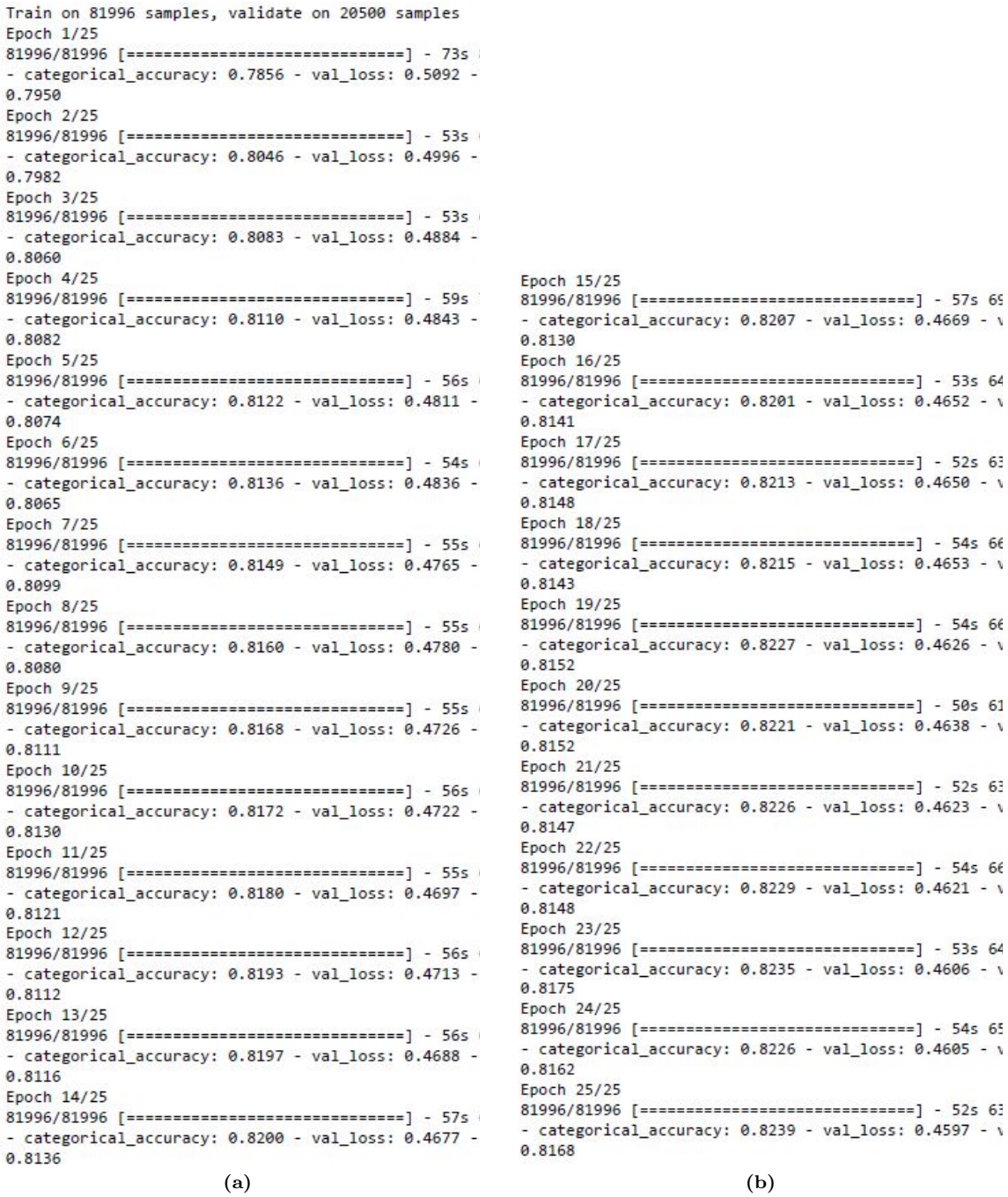



**Fig. 4.4.** Screenshot of per epoch training summary of ANN model

```
o.10.0 locally
81996/81996 [==============================] - 1557s 19ms/step - loss: 0.4874 - acc: 0.8076 - val_loss: 0.4429 - val_acc
: 0.8291
Epoch 2/10
81996/81996 [==============================] - 1544s 19ms/step - loss: 0.3710 - acc: 0.8529 - val_loss: 0.3628 - val_acc
: 0.8559
Epoch 3/10
81996/81996 [==============================] - 1541s 19ms/step - loss: 0.2922 - acc: 0.8868 - val_loss: 0.3325 - val_acc
: 0.8723
Epoch 4/10
81996/81996 [==============================] - 1542s 19ms/step - loss: 0.2271 - acc: 0.9141 - val_loss: 0.3076 - val_acc
: 0.8939
Epoch 5/10
81996/81996 [==============================] - 1526s 19ms/step - loss: 0.1798 - acc: 0.9319 - val_loss: 0.2810 - val_acc
: 0.8985
Epoch 6/10
81996/81996 [==============================] - 1501s 18ms/step - loss: 0.1417 - acc: 0.9481 - val_loss: 0.3241 - val_acc
: 0.9037
Epoch 7/10
81996/81996 [==============================] - 1502s 18ms/step - loss: 0.1141 - acc: 0.9593 - val_loss: 0.3458 - val_acc
: 0.9101
Epoch 8/10
81996/81996 [==============================] - 1501s 18ms/step - loss: 0.0945 - acc: 0.9665 - val_loss: 0.3216 - val_acc
: 0.9056
Epoch 9/10
81996/81996 [==============================] - 1501s 18ms/step - loss: 0.0782 - acc: 0.9728 - val_loss: 0.4018 - val_acc
: 0.9104
Epoch 10/10
81996/81996 [==============================] - 1501s 18ms/step - loss: 0.0685 - acc: 0.9764 - val_loss: 0.4380 - val_acc
: 0.9109
```

**Fig. 4.5.** Screenshot of per epoch training summary of CNN model

```
loss: 0.39%
acc: 92.00%
(1000, 3)
 [[7.2144385e-10 1.0000000e+00 5.8864131e-17]
 [7.1350137e-10 1.0000000e+00 6.1368431e-33]
 [9.9998689e-01 1.0501236e-05 2.6382270e-06]
 ...
 [6.0087984e-04 4.6072057e-12 9.9939907e-01]
 [9.9846566e-01 1.5343562e-03 8.7193941e-12]
 [1.9188301e-03 9.9808121e-01 1.6726571e-12]]
[[299  30  14]
 [ 14 331   7]
 [ 13   2 290]]

Confusion Matrix :

[87.17  8.75  4.08]
[ 3.98 94.03  1.99]
[ 4.26  0.66 95.08]

Accuracy = 92.0%

(venv) viranchi@Tesla-Server:~/rupashi$
```

**Fig. 4.6.** Screenshot of training results of CNN model

| Confusion Matrix (ANN) | | | |
|---|---|---|---|
| | **Forward** | **Left** | **Right** |
| **Forward** | 76.97% | 13.12% | 9.91% |
| **Left** | 11.36% | 86.36% | 2.27% |
| **Right** | 14.1% | 1.31% | 84.59% |

| Confusion Matrix (CNN) | | | |
|---|---|---|---|
| | **Forward** | **Left** | **Right** |
| **Forward** | 87.17% | 8.75% | 4.08% |
| **Left** | 3.98% | 94.03% | 1.99% |
| **Right** | 4.26% | 0.66% | 95.08% |

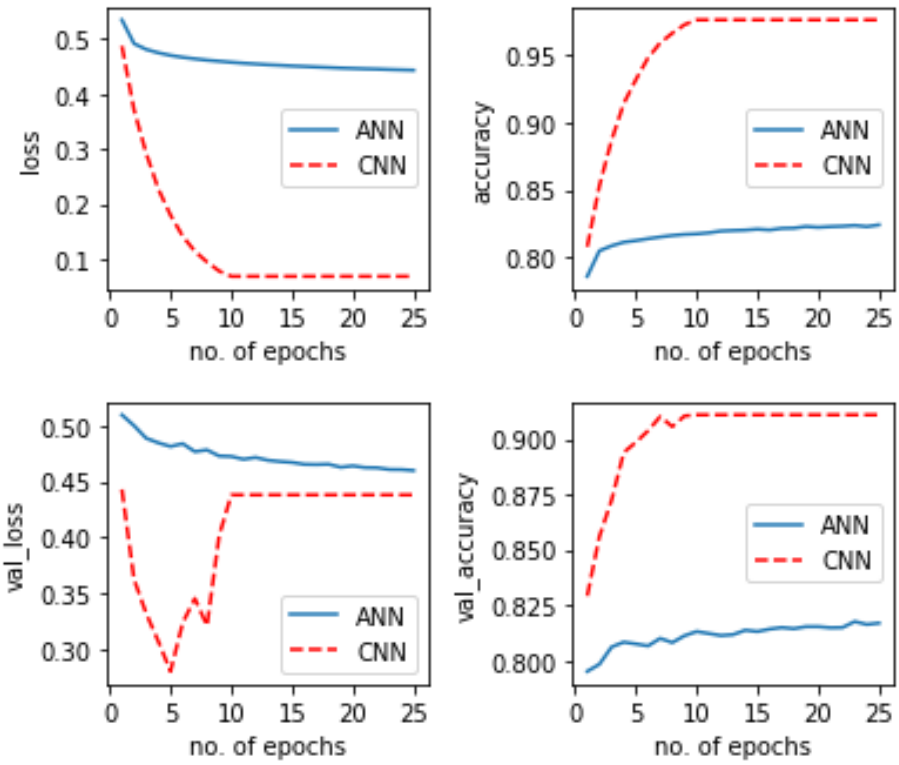



**Fig. 4.7.** Variation in *loss, accuracy, validation loss and validation accuracy* with respect to the number of epochs for ANN and CNN model during training phase.

# 5. Integration of Hardware and Software

## 5.1 TCP/IP Connection

TCP/IP, or the Transmission Control Protocol/Internet Protocol, is a suite of communication protocols used to interconnect network devices on the internet. It uses a client-server architecture in which a user or machine (a client) is provided a service by another computer (a server) in the network. In this project, the laptop acts as a server and Raspberry Pi acts as a client. A connection is established between the server and the client. Frames captured by the Raspberry Pi are transferred to the laptop in the form of a stream of bytes at the rate of 10 fps. The data received by the laptop is decoded as a colored image and the livefeed of the car view is displayed on the laptop screen. The direction predicted by the neural network is sent back to the Raspberry Pi via the established connection in a string format. Hence, there is a two way transfer of information between the client and the server. The transfer delay is negligible. This connection enables us to use the laptop as the main processing unit and Raspberry Pi as input/output unit. This is beneficial as trained neural network models are generally very heavy and cannot be processed using Raspberry Pi.

## 5.2 Overall Architecture

Firstly, a CNN model is trained on GPU by using the code RC_CNN.py using the acquired online dataset and the model is saved for loading later on. The code server.py which is the laptop server code, is then run on laptop. This code includes loading the pre-trained CNN model and then initializes the socket and waits for client to connect. Now, the code Rpi-client.py is run on RPi, which connects to the server through TCP/IP based network. Then, using PiCamera, a colored image is captured with resolution of 640 x 480 and at 10 frames per second and is sent to laptop through the TCP/IP network. The laptop receives the image and preprocesses it. The preprocessing includes converting it into grayscale image, cropping the track part of image and rescaling it into a 45 x 80 image. This preprocessed image is fed to CNN model for prediction of direction. The prediction is then converted into a string and sent back to the RPi through the same socket connection. This direction control is received by RPi and fed to the code drive2.py which

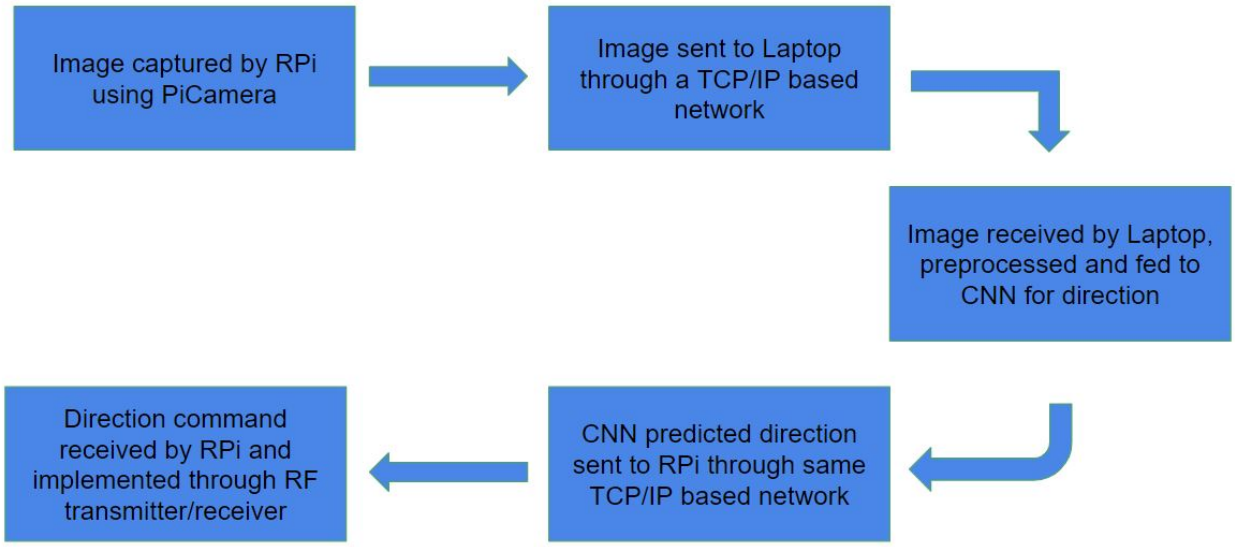



**Fig. 5.1.** Final Architecture

controls the GPIO pins. Whichever direction is received, corresponding pins are made low and the car moves. The process of capturing images and so on repeats till 'q' is pressed on the window showing images.

# 6. Testing and Results

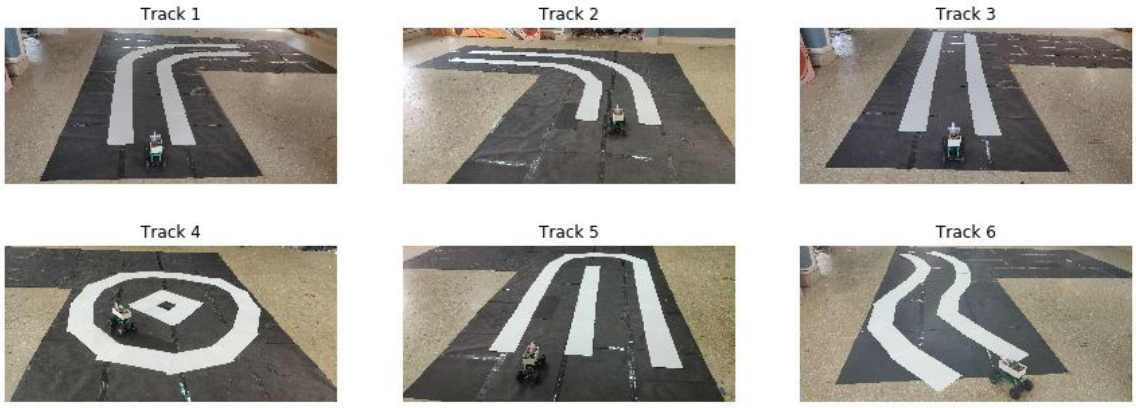



**Fig. 6.1.** Various tracks used for experimentation (from Track 1 to 6): curved-right, curved-left, straight, circular, U-turn and zigzag.

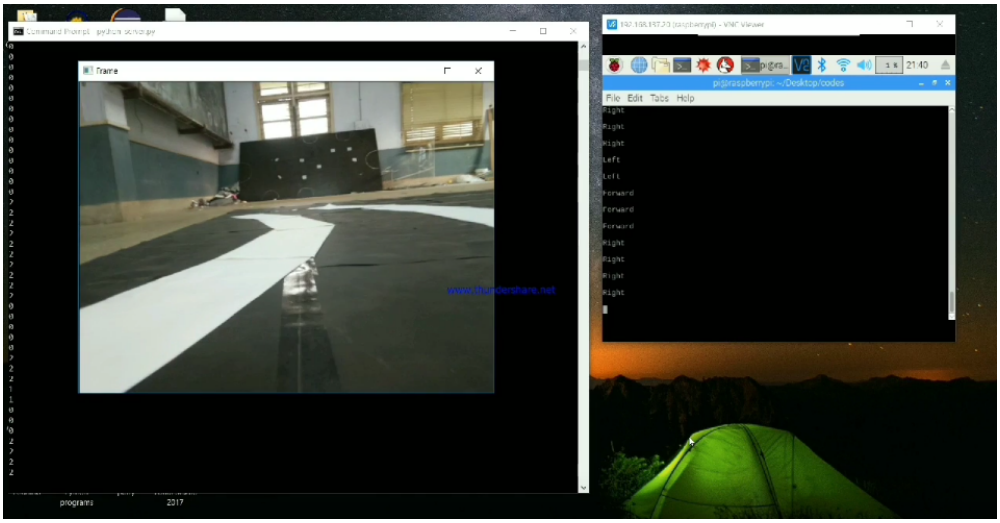


**Fig. 6.2.** Livefeed on server during testing

Tracks of different shapes were used to test the autonomous driving of the RC car as shown in Fig. 6.1. It was observed that the car moved within the boundary of the tracks majority of the

time. At times, the motion of the car was slightly curvy but it stayed in bound. The simulation testing accuracy of 92% worked excellently for our architecture. Whenever the car tends to reach track boundaries or is very close to the boundary, it was observed that our model correctly predicted the direction to avoid the off-track motion of the car. For narrow tracks, the car drove within the bounds but seemed to work in a very zig-zag manner which can be corrected by broadening the track. For the U-turn shown in track 5, the car seems to fail due to its large turning radius. For bigger turns the car works well. The prediction of the model is observed to be affected due to some surrounding conditions. For example, if there is a reflection of light on the track from a window, the CNN model tends to predict wrongly leading to the car going off the track. This was seen in Track 1 and Track 2. The car ran perfectly on Tracks 4 and 6.

The livefeed of the RC Car is streamed on the laptop server as shown in the Fig. 6.2. The feed displays the frames being captured by the PiCamera after being transmitted to the server via TCP/IP based network. The direction predicted for every frame by the trained CNN model is also displayed on screen. This corresponds to '0' for forward, '1' for left and '2' for right. This direction signal is received by RPi in a string format (for example "left") and is displayed on the RPi screen which is being accessed using the vnc player. This entire setup helps us to monitor the real time response of the architecture designed which aids in analysis and debugging.

# References


[1] "Neural Networks: A Comprehensive Foundation (Second Edition)" by Simon Haykin, McMaster University, Hamilton, Ontario, Canada.

[2] "Fundamentals of Neural Networks : Architectures, Algorithms and Applications (First edition)" by Laurene V. Fausett

[3] "Deep learning with Python" by Francois Chollet, 2017, Manning Publications.

[4] Reuschenbach, Arturo, et al. "iDriver-human machine interface for autonomous cars." Information Technology: New Generations (ITNG), 2011 Eighth International Conference on. IEEE, 2011.

[5] Pannu, Gurjashan Singh, Mohammad Dawud Ansari, and Pritha Gupta. "Design and implementation of autonomous car using Raspberry Pi." International Journal of Computer Applications 113.9 (2015).

[6] Omrane, Hajer, Mohamed Slim Masmoudi, and Mohamed Masmoudi. "Neural controller of autonomous driving mobile robot by an embedded camera." Advanced Technologies for Signal and Image Processing (ATSIP), 2018 4th International Conference on. IEEE, 2018.

[7] Jain, Aditya Kumar. "Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino." 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA). IEEE, 2018.

[8] Chaudhari, Raj, et al. "Autonomous Driving Car Using Convolutional Neural Networks." 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT). IEEE, 2018.

[9] Bojarski, Mariusz, et al. "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316 (2016).

[10] Shahdib, Fayaz, et al. "Obstacle detection and object size measurement for autonomous mobile robot using sensor." International Journal of Computer Applications 66.9 (2013).

[11] Borenstein, Johann, and Yoram Koren. "Obstacle avoidance with ultrasonic sensors." IEEE Journal on Robotics and Automation 4.2 (1988): 213-218.

[12] Hwu, Tiffany, et al. “A self-driving robot using deep convolutional neural networks on neuromorphic hardware.” 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017.

[13] Du, Shuyang, Haoli Guo, and Andrew Simpson. “Self-driving car steering angle prediction based on image recognition.” Department of Computer Science, Stanford University, Tech. Rep. CS231-626 (2017).

[14] Rajasekhar, M. V., and Anil Kumar Jaswal. “Autonomous vehicles: the future of automobiles.” 2015 IEEE International Transportation Electrification Conference (ITEC). IEEE, 2015.

[15] Autonomous RC Car driving dataset
`https://www.kaggle.com/c/mac0460-self-driving`

[16] ATMega 328P datasheet
`http://pdf1.alldatasheet.com/datasheet-pdf/view/241077/ATMEL/ATMEGA328P.html`

[17] Socket programming for TCP connection
`https://www.geeksforgeeks.org/socket-programming-python/`

[18] Documentation of Keras library on Tensorflow platform
`https://keras.io/`

[19] Machine learning course by Andrew Ng, Standford University
`https://www.coursera.org/learn/machine-learning`

# Appendix

## A Training and Simulation

### Artificial Neural Network Code from scratch

```
import numpy as np
import scipy.special
import matplotlib.pyplot as plt
import cv2

# Loading dataset
train_X = np.load("data1/train_X.npy")
train_Y = np.load("data1/train_y.npy")
test_X = np.load("data1/test_X.npy")
test_Y = np.load("data1/test_y.npy")

class neuralNetwork:

    # initializing the neural network
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningRate):

        # structure of neural network
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        # set learning rate
        self.lr = learningRate

        # Sigmoid activation function
        self.sigmoid = lambda x: scipy.special.expit(x)

        # link weight matrices, wih and who
        self.wih = (np.random.rand(self.hnodes,self.inodes)- 0.5)
        self.who = (np.random.rand(self.onodes, self.hnodes) - 0.5)

        pass

    # training the neural network
    def train(self, input_list, target_list):

        inputs = input_list
        targets = target_list
```

```
39          inputs = np.reshape(inputs, (len(input_list),1))
40          targets = np.reshape(targets, (len(target_list),1))
41
42          # feedforward algorithm
43          hidden_z = np.dot(self.wih, inputs)
44          hidden_activation = self.sigmoid(hidden_z)
45          output_z = np.dot(self.who,hidden_activation)
46          final_outputs = self.sigmoid(output_z)
47
48          # finding errors
49          output_errors = targets - final_outputs
50          hidden_errors = np.dot(np.transpose(self.who), output_errors)
51
52          # updating weights using Backpropagation
53          self.who = self.who + self.lr * np.dot(output_errors * final_outputs * (1.0-
    final_outputs) , np.transpose(hidden_activation))
54          self.wih = self.wih + self.lr * np.dot((hidden_errors* hidden_activation
    *(1.0-hidden_activation)), np.transpose(inputs))
55
56          pass
57
58      # testing the neural network
59      def  querry(self, inputs):
60
61          inputs = np.reshape(inputs, (len(inputs),1))
62          hidden_z = np.dot(self.wih, inputs)
63          hidden_activation = self.sigmoid(hidden_z)
64          output_z = np.dot(self.who,hidden_activation)
65          final_outputs = self.sigmoid(output_z)
66
67          return final_outputs
68
69  # Preprocessing dataset
70  # traning data
71  train_X_new = np.empty((train_X.shape[0], 45*80))
72  index = 0
73
74  for record in train_X:
75      sample = cv2.cvtColor(np.reshape(record, (45,80, 3)), cv2.COLOR_BGR2GRAY)
76      sample = sample.reshape((1,45*80))
77      sample = (np.asfarray(sample)/ 255.0)
78      if index < train_X.shape[0]:
79          train_X_new[index, :] = sample
80      index = index + 1
81
82  # testing data
83  test_X_new = np.empty((test_X.shape[0], 45*80))
```

```
index = 0

for record in test_X:
    sample = cv2.cvtColor(np.reshape(record, (45,80, 3)), cv2.COLOR_BGR2GRAY)
    sample = sample.reshape((1,45*80))
    sample = (np.asfarray(sample)/ 255.0)
    if index < test_X.shape[0]:
        test_X_new[index, :] = sample
    index = index + 1

# Correcting nomenclature
train_X_old = train_X
test_X_old = test_X
train_X = train_X_new
test_X = test_X_new
print(train_X_old.shape, test_X_old.shape, train_X.shape, test_X.shape)

# Definition of neural network
input_nodes =  row*col
hidden_nodes = 64
output_nodes = 3
epochs = 5
learning_rate = 0.001

# create an instance of the neural network
net = neuralNetwork(input_nodes, hidden_nodes, output_nodes, learning_rate)

# train the network
for e in range(epochs):

    index = -1;
    for record in train_X:
        index = index + 1
        #sample = cv2.cvtColor(np.reshape(record, (45,80, 3)), cv2.COLOR_BGR2GRAY)
        sample = record.reshape((record.size,1))
        input_list = (np.asfarray(sample[:]) / 255.0)
        target_list = np.zeros(output_nodes) + 0.01
        target_list[int(train_Y[index])] = 0.99
        net.train(input_list,target_list)
        pass
    pass

# test the network
correct = 0
total = 0
confusion_matrix = np.zeros((output_nodes, output_nodes))

```

```
for record in test_X:
    target_ans = int(test_Y[total])
    total = total + 1
    #sample = cv2.cvtColor(np.reshape(record, (45,80, 3)), cv2.COLOR_BGR2GRAY)
    sample = record.reshape((record.size,1))
    input_list = (np.asfarray(sample[:]) / 255.0 )
    #input_list = np.resize(input_list, (np.size(input_list),1))

    output_arr = net.querry(input_list)
    predicted_ans = np.argmax(output_arr)

    confusion_matrix[target_ans][predicted_ans] = confusion_matrix[target_ans][
    predicted_ans] + 1

    if target_ans == predicted_ans:
        correct = correct + 1

    img = np.reshape(record, (row,col))
    plt.imshow(img, cmap='gray')
    plt.title("Target = "+ labels[target_ans] + " | Predicted = "+ labels[
    predicted_ans])
    plt.show()

    pass

# Performance accuracy
accuracy = (correct/total) * 100
print("\nAccuracy = "+ str(accuracy) + "%\n")

# Confusion Matrix
for i in range(output_nodes):
    confusion_matrix[i] = np.round((confusion_matrix[i]/np.sum(confusion_matrix[i]))
     * 100, 2)
print("Confusion Matrix : \n")
print(confusion_matrix)

print(np.shape(net.wih), np.shape(net.who))

# Saving learned weights into a file
np.savetxt("wih2.csv", net.wih, delimiter = ",")
np.savetxt("who2.csv", net.who, delimiter = ",")
```

Listing 1: ANN code without libraries

### Artificial Neural Network Code With Keras Library

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import matplotlib as mp
import csv
import pandas as pd
import math
import  cv2

# model
import keras
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers.normalization import BatchNormalization
from keras.optimizers import SGD
# split and other utils
from sklearn.model_selection import train_test_split

# Loading dataset
train_X = np.load("online_dataset/train_X.npy")
train_Y = np.load("online_dataset/train_y.npy")
test_X = np.load("online_dataset/test_X.npy")
test_Y = np.load("online_dataset/test_y.npy")

# Preprocessing dataset
# traning data
train_X_new = np.empty((train_X.shape[0], 45*80))
index = 0

for record in train_X:
    sample = cv2.cvtColor(np.reshape(record, (45,80, 3)), cv2.COLOR_BGR2GRAY)
    sample = sample.reshape((1,45*80))
    sample = (np.asfarray(sample)/ 255.0)
    if index < train_X.shape[0]:
        train_X_new[index, :] = sample
    index = index + 1

# testing data
test_X_new = np.empty((test_X.shape[0], 45*80))
index = 0

for record in test_X:
    sample = cv2.cvtColor(np.reshape(record, (45,80, 3)), cv2.COLOR_BGR2GRAY)
    sample = sample.reshape((1,45*80))
    sample = (np.asfarray(sample)/ 255.0)
    if index < test_X.shape[0]:
        test_X_new[index, :] = sample
    index = index + 1

```

```
# Correcting nomenclature
train_X_old = train_X
test_X_old = test_X
train_X = train_X_new
test_X = test_X_new
print(train_X_old.shape, test_X_old.shape, train_X.shape, test_X.shape)

# Splitting dataset into training, tetsing and validation
print(train_X.shape, test_X.shape, train_Y.shape, test_Y.shape)
train_X, valid_X, train_Y, valid_Y = train_test_split(train_X, train_Y, test_size
    =0.2, random_state=13)
print(train_X.shape, valid_X.shape, train_Y.shape, valid_Y.shape)

train_Y = to_categorical(train_Y, num_classes = 3)
valid_Y = to_categorical(valid_Y, num_classes = 3)

print(train_Y.shape, train_Y)
print(valid_Y.shape, valid_Y)

row = 45
col = 80

model = Sequential()
# Adding the input layer and the first hidden layer
model.add(Dense(output_dim = 64, init = 'uniform', activation = 'sigmoid', input_dim
     = row*col))
# Adding the second hidden layer
model.add(Dense(output_dim = 32, init = 'uniform', activation = 'sigmoid'))
# Adding the output layer
model.add(Dense(output_dim = 3, init = 'uniform', activation = 'softmax'))

epochs = 100
learning_rate = 0.05
decay_rate = learning_rate/epochs
momentum = 0.8
sgd = SGD(lr = learning_rate, momentum = momentum, decay = decay_rate, nesterov=
    False)

# Compiling the ANN
model.compile(optimizer = sgd, loss = 'categorical_crossentropy', metrics = ['
    categorical_accuracy'])

model.summary()

# Fitting the ANN to the Training set
history = model.fit(train_X, train_Y, batch_size = 5, epochs = epochs,
    validation_data=(valid_X, valid_Y))
```

```
92
93 # Plotting training results
94 print(history.history.keys())
95
96 #  "Accuracy"
97 plt.plot(history.history['categorical_accuracy'])
98 plt.plot(history.history['val_categorical_accuracy'])
99 plt.title('model accuracy')
100 plt.ylabel('accuracy')
101 plt.xlabel('epoch')
102 plt.legend(['train', 'validation'], loc='upper left')
103 plt.show()
104
105 # "Loss"
106 plt.plot(history.history['loss'])
107 plt.plot(history.history['val_loss'])
108 plt.title('model loss')
109 plt.ylabel('loss')
110 plt.xlabel('epoch')
111 plt.legend(['train', 'validation'], loc='upper left')
112 plt.show()
113
114 # Evaluate using Test data
115 test_Y_ = to_categorical(test_Y)
116 scores = model.evaluate(test_X, test_Y_, verbose = 0)
117 print("%s: %.2f%%" % (model.metrics_names[0], scores[0]))
118 print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
119
120 # Predicting the Test Results
121 pred_y = model.predict(test_X)
122 print(pred_y.shape,"\n", pred_y)
123
124 i=0
125 pred_Y = np.empty((pred_y.shape[0],1))
126 for rec in pred_y:
127     pred_Y[i] = np.argmax(rec > 0.5)
128     i = i+1
129 print(pred_Y.shape)
130
131 # Confusion Matrix
132 test_Y = test_Y.reshape((test_Y.shape[0],1))  # to make a 2d matrix
133 cm = confusion_matrix(test_Y,pred_Y)
134 print(cm,"\n")
135 print("Confusion Matrix : \n")
136 for i in range(3):
137     print(np.round(cm[i]/np.sum(cm[i])*100,2))
138
```

```
# Performance accuracy
accuracy = ((test_Y == pred_Y)==True).sum() / len(test_Y) * 100
print("\nAccuracy = "+ str(accuracy) + "%\n")

# Save model as a .h5 file
model.save('NN_model_1.h5')
```

Listing 2: ANN with Keras

### Convolutional Neural Network Code with Keras

```
import os
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID";
# The GPU id to use, usually either "0" or "1";
os.environ["CUDA_VISIBLE_DEVICES"]="1";

# importing libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mp
import math
import  cv2
# model
import keras
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers.normalization import BatchNormalization
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from sklearn.model_selection import train_test_split

# Loading dataset
train_X = np.load("online_dataset/train_X.npy")
train_Y = np.load("online_dataset/train_y.npy")
test_X = np.load("online_dataset/test_X.npy")
test_Y = np.load("online_dataset/test_y.npy")

# Preprocessing dataset
# training data
train_X_new = np.empty((train_X.shape[0], 45,80))
index = 0

for record in train_X:
    sample = cv2.cvtColor(np.reshape(record, (45,80, 3)), cv2.COLOR_BGR2GRAY)
    #sample = sample.reshape((1,45*80))
    sample = (np.asfarray(sample)/ 255.0)
    if index < train_X.shape[0]:
```

```
        train_X_new[index, :, :] = sample
    index = index + 1

# testing data
test_X_new = np.empty((test_X.shape[0], 45,80))
index = 0

for record in test_X:
    sample = cv2.cvtColor(np.reshape(record, (45,80, 3)), cv2.COLOR_BGR2GRAY)
    #sample = sample.reshape((1,45*80))
    sample = (np.asfarray(sample)/ 255.0)
    if index < test_X.shape[0]:
        test_X_new[index, :] = sample
    index = index + 1

# Correcting nomenclature
train_X_old = train_X
test_X_old = test_X
train_X = train_X_new
test_X = test_X_new
print("train_X_old.shape, test_X_old.shape, train_X.shape, test_X.shape")
print(train_X_old.shape, test_X_old.shape, train_X.shape, test_X.shape)

# Splitting dataset into training, testing and validation
train_X, valid_X, train_Y, valid_Y = train_test_split(train_X, train_Y, test_size
    =0.2, random_state=13)
print(train_X.shape, valid_X.shape, train_Y.shape, valid_Y.shape)

train_Y = to_categorical(train_Y, num_classes = 3)
valid_Y = to_categorical(valid_Y, num_classes = 3)

# Reshaping input for CNN
train_X = train_X.reshape(train_X.shape[0], 45,80,1)
test_X = test_X.reshape(test_X.shape[0], 45,80,1)
valid_X = valid_X.reshape(valid_X.shape[0], 45,80,1)

#del model (uncomment this for retraining the model)

# Defining our model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(45, 80, 1), padding='same', activation='
    relu'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='valid'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='valid'))

# We flatten our data in order to feed it through the dense(output) layer
model.add(Flatten())
```

```
model.add(Dense(512, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Compiling the ANN
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])

model.summary()

# Fitting the CNN to the Training set
history = model.fit(train_X, train_Y, batch_size = 5, epochs = 5, validation_data=(valid_X, valid_Y))

# Evaluate using Test data
test_Y_ = to_categorical(test_Y)
scores = model.evaluate(test_X, test_Y_, verbose = 0)
print("%s: %.2f%%" % (model.metrics_names[0], scores[0]))
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

# Saving the model as a .h5 file
model.save('CNN_model_1.h5')

# Predicting the Test Results
pred_y = model.predict(test_X)
print(pred_y.shape,"\n", pred_y)

i=0
pred_Y = np.empty((pred_y.shape[0],1))
for rec in pred_y:
    pred_Y[i] = np.argmax(rec > 0.5)
    i = i+1

# Confusion Matrix
from sklearn.metrics import confusion_matrix
test_Y = test_Y.reshape((test_Y.shape[0],1))  # to make a 2d matrix
cm = confusion_matrix(test_Y,pred_Y)
print(cm,"\n")

print("Confusion Matrix : \n")
for i in range(3):
    print(np.round(cm[i]/np.sum(cm[i])*100,2))

# Performance accuracy
accuracy = ((test_Y == pred_Y)==True).sum() / len(test_Y) * 100
print("\nAccuracy = "+ str(accuracy) + "%\n")
```

Listing 3: CNN code

# B RPi Based Controller Model

### Laptop Server Code

```
# importing libraries for Socket programming and Pre processing
import socket
import sys
import cv2
import pickle
import numpy as np
import struct
import io
import matplotlib.pyplot as plt

# Keras Libraries
import keras
from keras.models import load_model
from keras.layers.convolutional import Conv2D

# Loading trained CNN model
model = load_model('CNN_model_2.h5')
print("Model loaded\n")
arr = ["forward", "left", "right"]
i = 1

# Socket intialization
server_socket = socket.socket()
print('Socket created')

server_socket.bind(('192.168.137.1', 64321))
print('Socket bind complete')
print("Socket bound to", server_socket.getsockname()[1])

server_socket.listen(10)
print('Socket now listening')

conn, addr = server_socket.accept()
# print("Got connection from ", addr)
# output =  "Thanks for connection."
# conn.sendall(output.encode('utf-8'))

# Accept a single connection and make a file-like object out of it
#connection = server_socket.accept()[0].makefile('rb')
connection = conn.makefile('rb')
print("Got connection from RPI.\n")

try:
```

```
# Read the length of the image; if length = 0; quit

   while True:
     image_len = struct.unpack('<L', connection.read(struct.calcsize('<L')))[0]
     if not image_len:
       break
     # Construct a stream to hold image data and read image data from the
   connection
     image_stream = io.BytesIO()
     image_stream.write(connection.read(image_len))
     image_stream.seek(0)

     data = np.fromstring(image_stream.getvalue(), dtype = np.uint8)
     image = cv2.imdecode(data, 1)

     rows = image.shape[0]
     cols = image.shape[1]
     M = cv2.getRotationMatrix2D((cols/2,rows/2),180,1)
     image = cv2.warpAffine(image,M,(cols,rows))

     cv2.imshow('Frame', image)
     if cv2.waitKey(1) & 0xFF == ord("q"):
       break

     img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
     # plt.imshow(img, cmap = 'gray')
     # plt.show()

     img = img[150:450,0:img.shape[1]]

     img = cv2.resize(img, (80,45), interpolation = cv2.INTER_AREA)
     #img = cv2.GaussianBlur(img, (3,3), 0)

     img1 = img.reshape((1,45,80,1))

     inp = (np.asfarray(img1)/ 255.0)
     pred_y = model.predict(inp)

     direction = np.argmax(pred_y > 0.5)

     name =  "test_images/" + str(i) + "-" + str(arr[direction]) + ".jpg"
     cv2.imwrite(name, img)

     direction = str(direction)
     print(direction)
     conn.sendall(direction.encode('utf-8'))
     i = i+1
```

```
90
91 finally:
92     connection.close()
93     server_socket.close()
```

Listing 4: Laptop server code

### Raspberry Pi Client Code

```
1  import io
2  import socket
3  import struct
4  import time
5  import picamera
6  from drive2 import RCTest
7
8  drive = RCTest()
9
10 client_socket = socket.socket()
11 client_socket.connect(('192.168.137.1', 64321))
12 #print(client_socket.recv(1024))
13 #client_socket.close()
14
15 # Make a file-like object out of the connection
16 connection = client_socket.makefile('wb')
17 try:
18     with picamera.PiCamera() as camera:
19
20         camera.resolution = (640, 480)
21         camera.framerate = 5
22         # allow the camera to warmup
23         time.sleep(0.1)
24         stream = io.BytesIO()
25         for foo in camera.capture_continuous(stream, 'jpeg'):
26
27             # Write length of capture to stream and flush to ensure it actually gets
       sent
28             connection.write(struct.pack('<L', stream.tell()))
29             connection.flush()
30
31             # Rewind the stream and send image data over the wire
32             stream.seek(0)
33             connection.write(stream.read())
34
35             # Reset the stream for the next capture
36             stream.seek(0)
37             stream.truncate()
38
```

```
            direction = client_socket.recv(1024)
            drive.steer(direction)

    # Write a length of zero to the stream to signal we're done
    connection.write(struct.pack('<L', 0))
finally:
    connection.close()
    client_socket.close()
```

Listing 5: RPi Client Code

**RF transmitter control by RPi code**

```
# Drive via RPi for TESTING
import time
import pytz
import RPi.GPIO as GPIO

class RCTest(object):

  def __init__(self):
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)

    #17 - left , 22 - right, 27 - forward
    GPIO.setup(17, GPIO.OUT)
    GPIO.setup(22, GPIO.OUT)
    GPIO.setup(27, GPIO.OUT)

    GPIO.output(17, GPIO.HIGH)
    GPIO.output(22, GPIO.HIGH)
    GPIO.output(27, GPIO.HIGH)

    self.steer()

  def __del__(self):
    GPIO.cleanup()

  def steer(self, dir):
    if dir==2:
      print("Right\n")
      GPIO.output(27, GPIO.LOW)
      GPIO.output(22, GPIO.LOW)
      time.sleep(50.0/1000.0)
      GPIO.output(27, GPIO.HIGH)
      GPIO.output(22, GPIO.HIGH)

    elif dir==1:
```

```
        print("Left\n")
        GPIO.output(27, GPIO.LOW)
        GPIO.output(17, GPIO.LOW)
        time.sleep(50.0/1000.0)
        GPIO.output(27, GPIO.HIGH)
        GPIO.output(17, GPIO.HIGH)

    elif dir==0:
        print("Forward\n")
        GPIO.output(27, GPIO.LOW)
        time.sleep(50.0/1000.0)
        GPIO.output(27, GPIO.HIGH)
```

Listing 6: RF transmitter control by RPi

# C Arduino Based Controller Model

### Laptop Server Code

```
import io
import socket
import sys
import struct
import serial
import time
import pytz
import datetime

# Server socket intialization
server_socket = socket.socket()
server_socket.bind(('192.168.137.1', 59722))
print("Socket created and binded.")
#s.getsockname()
#print("Socket bound to", s.getsockname())
server_socket.listen(1)
print("Socket is listening.")
connection, add = server_socket.accept()
print("Got connection from ", add)
output =  "Thanks for connection."
connection.sendall(output.encode('utf-8'))

# Arduino control
ser = serial.Serial("COM3", 9600, timeout=1)

try:
    while True:
```

```
      f = open('Control_Data.csv', 'w')
      f.write("Command, Timestamp \n")
      output = connection.recv(1024)
      output = output.decode("utf-8")

      if(len(output)==0):
        ser.write(chr(0).encode())
        ser.close()
        break

      print(output)

      if output=="right":
        #print("Forward Right")
        ser.write(chr(6).encode())
        f.write("Forward Right" + ",")

      elif output=="left":
        #print("Forward Left")
        ser.write(chr(7).encode())
        f.write("Forward Left" + ",")

      elif output=="forward":
        #print("Forward")
        ser.write(chr(1).encode())
        f.write("Forward" + ",")

      utc_time = datetime.datetime.now(pytz.utc)
      local_time = (utc_time.astimezone(pytz.timezone('Asia/Calcutta')))
      date = (str(local_time)).split()[0]
      f.write(str(local_time) + "\n")

finally:
  f.close()
  connection.close()
  server_socket.close()
```

Listing 7: Laptop server code

### Raspberry Pi Client Code

```
# For TESTING using keras weights
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
import serial
import datetime
```

```
import pytz
import numpy as np
from drive2 import RCTest
import RPi.GPIO as GPIO
from neural_testing import neuralNetwork
import keras
from keras.models import load_model

model = load_model('online_model.h5')
label = ["forward", "left", "right"]

drive = RCTest()

# initialize the camera
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 5
rawCapture = PiRGBArray(camera, size=(640, 480))

# allow the camera to warmup
time.sleep(0.1)

# capture frames from the camera
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True
    ):

  # grab the raw NumPy array, and rotate image
  image = frame.array
  rows = image.shape[0]
  cols = image.shape[1]
  M = cv2.getRotationMatrix2D((cols/2,rows/2),180,1)
  image = cv2.warpAffine(image,M,(cols,rows))

  # display the frame
  cv2.imshow("Frame", image)

  # Pre process image and send to neural network
  img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  img = img[200:img.shape[0],0:img.shape[1]]
  img = cv2.resize(img, (45,80), interpolation = cv2.INTER_AREA)
  img = cv2.GaussianBlur(img, (5,5), 0)
  #img = cv2.fastNlMeansDenoising(img,None,4,7,21)
  #img = cv2.fastNlMeansDenoisingColored(img,None,10,10,7,21)

  #img = img.reshape((1,img.size))
  input = (np.asfarray(img)/ 255.0)
  pred_y = model.predict(input)
```

```
  pred_Y= np.argmax(pred_y > 0.5)
  drive.steer(pred_Y)

  # clear the stream in preparation for the next frame
  rawCapture.truncate(0)

  # Press q to exit
  if cv2.waitKey(1) & 0xFF == ord("q"):
          break
```

Listing 8: RPi Client Code

### Arduino Based Controller Code

```
// assign pin num
int right_pin = 12;
int left_pin = 13;
int forward_pin = 11;
int reverse_pin = 10;

// duration for output
int time = 5;
// 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(9600);
}

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

void right(int time){
 digitalWrite(right_pin, LOW);
 delay(time);
}
```

```
35
36 void left(int time){
37  digitalWrite(left_pin, LOW);
38  delay(time);
39 }
40
41 void forward(int time){
42  digitalWrite(forward_pin, LOW);
43  delay(time);
44 }
45
46 void reverse(int time){
47  digitalWrite(reverse_pin, LOW);
48  delay(time);
49 }
50
51 void forward_right(int time){
52  digitalWrite(forward_pin, LOW);
53  digitalWrite(right_pin, LOW);
54  delay(time);
55 }
56
57 void reverse_right(int time){
58  digitalWrite(reverse_pin, LOW);
59  digitalWrite(right_pin, LOW);
60  delay(time);
61 }
62
63 void forward_left(int time){
64  digitalWrite(forward_pin, LOW);
65  digitalWrite(left_pin, LOW);
66  delay(time);
67 }
68
69 void reverse_left(int time){
70  digitalWrite(reverse_pin, LOW);
71  digitalWrite(left_pin, LOW);
72  delay(time);
73 }
74
75 void reset(){
76  digitalWrite(right_pin, HIGH);
77  digitalWrite(left_pin, HIGH);
78  digitalWrite(forward_pin, HIGH);
79  digitalWrite(reverse_pin, HIGH);
80 }
81
```

```
void send_command(int command, int time){
 switch (command){

    //reset command
    case 0: reset(); break;

    // single command
    case 1: forward(time); break;
    case 2: reverse(time); break;
    case 3: right(time); break;
    case 4: left(time); break;

    //combination command
    case 6: forward_right(time); break;
    case 7: forward_left(time); break;
    case 8: reverse_right(time); break;
    case 9: reverse_left(time); break;

    default: Serial.print("Invalid Command\n");
   }
}
```

Listing 9: Arduino Based Controller

# D RC Car control for collection of dataset

These set of codes can be utilized by a user to run the RC car through arrow keys on keyboard for collection of own dataset.

**Car control via keyboard and Arduino**

```
import serial
import pygame
from pygame.locals import *
import time
import pytz
import datetime

class RCTest(object):

   def __init__(self):
       pygame.init()
       pygame.display.set_mode((250, 250))
       self.ser = serial.Serial("COM3", 9600, timeout=1)
       self.send_inst = True
       self.steer()
```

```
16
17  def steer(self):
18      f = open('Control_Data.csv', 'w' )
19      f.write("Command, Timestamp \n")
20
21      while self.send_inst:
22          for event in pygame.event.get():
23              if event.type == KEYDOWN:
24                  key_input = pygame.key.get_pressed()
25
26                  # complex orders
27                  #  if key_input[pygame.K_UP] and key_input[pygame.K_RIGHT]:
28                  #      print("Forward Right")
29                  #      self.ser.write(chr(6).encode())
30                  #      f.write("Forward Right" + ",")
31
32                  # elif key_input[pygame.K_UP] and key_input[pygame.K_LEFT]:
33                  #      print("Forward Left")
34                  #      self.ser.write(chr(7).encode())
35                  #      f.write("Forward Left" + ",")
36
37                  # elif key_input[pygame.K_DOWN] and key_input[pygame.K_RIGHT]:
38                  #      print("Reverse Right")
39                  #      self.ser.write(chr(8).encode())
40                  #      f.write("Reverse Right" + ",")
41
42                  # elif key_input[pygame.K_DOWN] and key_input[pygame.K_LEFT]:
43                  #      print("Reverse Left")
44                  #      self.ser.write(chr(9).encode())
45                  #      f.write("Reverse Left" + ",")
46
47                  # simple orders
48                  if key_input[pygame.K_UP]:
49                      print("Forward")
50                      self.ser.write(chr(1).encode())
51                      f.write("Forward" + ",")
52
53                  elif key_input[pygame.K_DOWN]:
54                      print("Reverse")
55                      self.ser.write(chr(2).encode())
56                      f.write("Reverse" + ",")
57
58                  elif key_input[pygame.K_RIGHT]:
59                      print("Right")
60                      self.ser.write(chr(6).encode())
61                      f.write("Right" + ",")
62
```

```
                    elif key_input[pygame.K_LEFT]:
                        print("Left")
                        self.ser.write(chr(7).encode())
                        f.write("Left" + ",")

                    # exit
                    elif key_input[pygame.K_x] or key_input[pygame.K_q]:
                        print("Exit")
                        self.send_inst = False
                        self.ser.write(chr(0).encode())
                        self.ser.close()
                        pygame.quit()
                        break

                    utc_time = datetime.datetime.now(pytz.utc)
                    local_time = (utc_time.astimezone(pytz.timezone('Asia/Calcutta'))
    )
                    date = (str(local_time)).split()[0]

                    f.write(str(local_time) + "\n")


                elif event.type == pygame.KEYUP:
                    self.ser.write(chr(0).encode())
                    f.write("Idle" + ",")
                    utc_time = datetime.datetime.now(pytz.utc)
                    local_time = (utc_time.astimezone(pytz.timezone('Asia/Calcutta'))
    )
                    date = (str(local_time)).split()[0]

                    f.write(str(local_time) + "\n")
        f.close()

if __name__ == '__main__':
    RCTest()
```

Listing 10: Laptop server code

### Car control via keyboard and RPi

```
import serial
import pygame
from pygame.locals import *
import time
import pytz
import datetime
import RPi.GPIO as GPIO

```

```
GPIO.setmode(GPIO.BCM)
#17 - left , 22 - right , 27 - forward
GPIO.setup(17, GPIO.OUT)
GPIO.setup(22, GPIO.OUT)
GPIO.setup(27, GPIO.OUT)
GPIO.output(17, GPIO.HIGH)
GPIO.output(22, GPIO.HIGH)
GPIO.output(27, GPIO.HIGH)

class RCTest(object):

  def __init__(self):
    pygame.init()
    pygame.display.set_mode((250, 250))
    self.steer()

  def steer(self):
    f = open('Direction_Stamps.csv', 'w' )
    f.write("Command, Timestamp \n")

    while 1:
      for event in pygame.event.get():
        if event.type == KEYDOWN:
          key_input = pygame.key.get_pressed()

          utc_time = datetime.datetime.now(pytz.utc)
          local_time = (utc_time.astimezone(pytz.timezone('Asia/Calcutta')))
          date = (str(local_time)).split()[0]
          f.write(str(local_time) + "\n")

          if key_input[pygame.K_RIGHT]:
            f.write("Right" + ",")
            GPIO.output(27, GPIO.LOW)
            GPIO.output(22, GPIO.LOW)
            time.sleep(50.0/1000.0)
            GPIO.output(27, GPIO.HIGH)
            GPIO.output(22, GPIO.HIGH)

          elif key_input[pygame.K_LEFT]:
            f.write("Left" + ",")
            GPIO.output(27, GPIO.LOW)
            GPIO.output(17, GPIO.LOW)
            time.sleep(50.0/1000.0)
            GPIO.output(27, GPIO.HIGH)
            GPIO.output(17, GPIO.HIGH)

          elif key_input[pygame.K_UP]:
```

```
            f.write("Forward" + ",")
            GPIO.output(27, GPIO.LOW)
            time.sleep(50.0/1000.0)
            GPIO.output(27, GPIO.HIGH)

          elif key_input[pygame.K_x] or key_input[pygame.K_q]:
            print("Exit")
            pygame.quit()
            break

        elif event.type == pygame.KEYUP:
          f.write("Idle" + ",")
          utc_time = datetime.datetime.now(pytz.utc)
          local_time = (utc_time.astimezone(pytz.timezone('Asia/Calcutta')))
          date = (str(local_time)).split()[0]
          f.write(str(local_time) + "\n")

    f.close()
    GPIO.cleanup()

if __name__ == '__main__':
   RCTest()
```

Listing 11: Laptop server code