**CHAPTER 1**

**INTRODUCTION:**

Imagine a bus carrying passengers on its own, driving better than any bus driver could do. Imagine a taxi, which can be called through an app installed in your smartphone, which carries you to your destination as fast and economically for you as possible. self-driving car is a ground vehicle that is capable of driving without user interference. Automobile manufacturers like General Motors, Ford, Mercedes Benz, Volkswagen, Audi, Nissan, Toyota, BMW, Volvo, and other companies like Google, NVIDIA, Waymo are testing driverless cars. Governments of countries in Europe and of the USA are creating regulations for self-driving cars, as a result of the latest advances of the industry.

An autonomous vehicle is a system capable of sensing its environment and navigate without the need of human action. Some of the sensors an autonomous car may use are: laser for measuring distance (LIDAR), radar, camera (or cameras, since two cameras allow to have three-dimensional information through the use of stereo vision), inertial sensors and GPS.

The need of so many different sensors is, to have a trustful knowledge of the position of the vehicle. While the GPS and the inertial sensor will approximately tell the position of the vehicle, the laser, radar and camera will give information about the environment. First of all, the system’s “brain” has to filter the signals to remove the sound and fuse them to have trustful information. This information will be used to drive the car by taking decisions based on it and by doing control operations so, the better the quality of the information about its state and environment, the safer the vehicle will be.

Then, as said, the car will have to take logical decisions for the real life based on its map of the zone and the obstacles it detects while following the fastest of the possible paths that will lead it to its desired destination. Once best path is decided, it will send commands to the actuators that will perform the task of the driving wheel, the throttle and the brake.

The beneﬁts of fully autonomous vehicles can go way beyond removing the need of a human driver. Transportation services such as Uber will start using self-driving cars instead of human drivers, and might become a cheaper and better alternative for end consumers than owning a car. This will represent a shift on the way cities are planned, as fewer parking places will be needed, and most importantly, in a smart city with most of its vehicles being connected and autonomous, trafﬁc optimization will be able to be heavily applied by coordinating movement. This will result in a major decrease on travel time, and it will also save lives as emergency services will be able to reach their destinations faster. The autonomous cars are not all equal and they don’t use the exact same sensors or algorithms, but they all need to sense accurately and be able to take fast decisions if they want to compete with human beings, so they need powerful computers to be mounted onboard to manage to do it apart from the said sensors, since the computing load to carry this task is very high, independently on the methods used

The existing self-driving cars use propriety software and hardware, hence are expensive. We intend to reduce the cost of system, use CNN (Convolution Neural Network) for learning and control and also provide virtual remote control of car by human if necessary. We use a single camera as input for navigation and obstacle avoidance.

**Objectives:**

* To build a self-driving car based on IoT and RF using entirely open source software (OpenCV, TensorFlow, Python) and hardware (Arduino) technology.
* To reduce the cost of system, use CNN (Convolution Neural Network) for learning and control and also provide virtual remote control of car by human if necessary.
* To use a single camera as input for navigation and obstacle avoidance.

**CHAPTER 2**

**LITERATURE SURVEY:**

In order to know the existing methods for building self-driving car based on IoT and CNN for control and learning we have gone through the below mentioned published papers.

**Aditya Kumar** [1] et al., have discussed the evolution of Artificial Intelligence has served as the catalyst in the field of technology. We can now develop things which was once just an imagination. One of such creation is the birth of self-driving car. Days have come where one can do their work or even sleep in the car and without even touching the steering wheel, accelerator you will still be able to reach your target destination safely. This paper proposes a working model of self-driving car which is capable of driving from one location to the other or to say on different types of tracks such as curved tracks, straight tracks and straight followed by curved tracks. A camera module is mounted over the top of the car along with Raspberry Pi sends the images from real world to the Convolutional Neural Network which then predicts one of the following directions .i.e. right, left, forward or stop which is then followed by sending a signal from the Arduino to the controller of the remote controlled car and as a result of it the car moves in the desired direction without any human intervention.

**Michael G. Bechtel** [2]et al., have developed a low-cost deep neural network based autonomous car platform. DeepPicar is a small-scale replication of a real self-driving car called DAVE-2 by NVIDIA. DAVE-2 uses a deep convolutional neural network (CNN), which takes images from a front-facing camera as input and produces car steering angles as output. DeepPicar uses the same network architecture—9 layers, 27 million connections and 250K parameters—and can drive itself in real-time using a web camera and a Raspberry Pi 3 quad-core platform. Using DeepPicar, they analyze the Pi 3’s computing capabilities to support end-to-end deep learning based real-time control of autonomous vehicles. They also systematically compare other contemporary embedded computing platforms using the DeepPicar’s CNN-based real-time control workload. They found that all tested platforms, including the Pi 3, are capable of supporting the CNN-based real-time control, from 20 Hz up to 100 Hz, depending on hardware platform. However, also they found that shared resource contention remains an important issue that must be considered in applying CNN models on shared memory based embedded computing platforms; we observe up to 11.6X execution time increase in the CNN based control loop due to shared resource contention. To protect the CNN workload, we also evaluate state-of-the-art cache partitioning and memory bandwidth throttling techniques on the Pi 3. Also, that cache partitioning is ineffective, while memory bandwidth throttling is an effective solution.

**Abdur R. Fayjie** [3] et al., say thatDeep Reinforcement Learning has led us to newer possibilities in solving complex control and navigation related tasks. This paper presents Deep Reinforcement Learning autonomous navigation and obstacle avoidance of self-driving cars, applied with Deep Q Network to a simulated car an urban environment. The approach uses two types of sensor data as input: camera sensor and laser sensor in front of the car. It also designs a cost-efﬁcient high-speed car prototype capable of running the same algorithm in real-time. The design uses a camera and a Hokuyo Lidar sensor in the car front. It uses embedded GPU (Nvidia-TX2) for running deep-learning algorithms based on sensor inputs.

**Truong-Dong Do** [4] et al., in this paper, a monocular vision-based self-driving car prototype using Deep Neural Network on Raspberry Pi is proposed. Self-driving cars are one of the most increasing interests in recent years as the definitely developing relevant hardware and software technologies toward fully autonomous driving capability with no human intervention. Level-3/4 autonomous vehicles are potentially turning into a reality in near future. Convolutional Neural Networks (CNNs) have been shown to achieve significant performance in various perception and control tasks in comparison to other techniques in the latest years. The key factors behind these impressive results are their ability to learn millions of parameters using a large amount of labeled data. In this work, we concentrate on finding a model that directly maps raw input images to a predicted steering angle as output using a deep neural network. The technical contributions of this work are two-fold. First, the CNN model parameters were trained by using data collected from vehicle platform built with a 1/10 scale RC car, Raspberry Pi 3 Model B computer and front facing camera. The training data were road images paired with the time-synchronized steering angle generated by manually driving. Second, road tests the model on Raspberry to drive itself in the outdoor environment around oval-shaped and 8-shaped with traffic sign lined track. The experimental results demonstrate the effectiveness and robustness of autopilot model in lane keeping task.

**Wen-Yen Li** [5]et al., developed an agent that can imitate the behavior of humans driving a car. When human beings driving a car, he/she majorly uses vision system to recognize the states of the car, including the position, velocity, and the surrounding environments. In this paper, we implemented a self-driving car which can drive itself on the track of a simulator. The self-driving car uses deep neural network as a computational framework to “learn” what is the position of the car related to the road. While the car understands the position of itself related to the track, it can use the information as a basis for feedback control.

**B Padmaja** [6] et al., say that an autonomous car is a ground vehicle that is capable of driving without user interference. Traffic congestion and number of collisions are major issues in road traffic control due to rapid increase day-by-day. Autonomous cars provide a solution to this problem in an efficient and economical way. Their proposed system utilizes mathematical models like neural networks and image processing techniques to sense the environment. This is implemented as three major components: curved road detection (steering), road sign and signal detection and obstacle detection (collision avoidance). Back Propagation is used for steering control with detection of curved roads; Haar features are used for road signal, sign detection and a distance sensor for collision avoidance. Data collected from the sensors is sent to a server for processing. Based on the result, a command is sent to the car. A GPS module attached to the car identifies the location of the car and with the help of a 3rd party location service, route to destination is identified and directions are sent to the car. Wireless networks are used to transmit data between sensors and the server. Python scripts are used to control and integrate all the units together. The designed system can attain high accuracy with real – time constraints.

**Masmoudi Hajer Omrane** [7] et al., proposed to build an autonomous RC Car that uses Artificial Neural Network (ANN) for control. It describes the theory behind the neural network and autonomous vehicles, and how a prototype with a camera as its only input can be designed to test and evaluate the algorithm capabilities. The ANN is a good algorithm that could help recognize patterns in an image, it can with a training set, containing 2000 images, classify an image with 96% of accuracy rate. The main contribution of this paper consists in using a single camera for navigation, possibly for obstacle avoidance.

**Summary of literature review**

This section summarizes the work done by various authors in the field of autonomous/self-driving cars. This presents an overview of previous work on related topics that provide the necessary background for the purpose of this research. The literature review concentrates on building a car based on Artificial Neural Networks (AAN) and Convolution Neural Networks (CNN) for control, RaspberryPi as the target hardware, using image processing techniques like Haar cascade. They also cite the sensing the environment using visual input and send control signals accordingly.

All these topics require further research, as it is essential for self-driving cars. Thesis and research articles were studied thoroughly and referred. The idea behind doing literature review is to collect data and have understanding on different methods and approaches that can be used, to clear understand the software requirement of the project.

**CHAPTER 3**

**THEORITICAL BACKGROUND:**

**3.1 SOFTWARE**

**3.1.1 Python** is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale projects. Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

**3.1.2 OpenCV**

OpenCV is an image and video processing library with bindings in C++, C, Python, and Java. OpenCV is used for all sorts of image and video analysis, like facial recognition and detection, license plate reading, photo editing, advanced robotic vision, optical character recognition, and a whole lot more.

**3.1.3 pySerial**

pySerial is a library that allows Python to send and receive data from Arduino much like the Serial Monitor does.

**3.1.4 NumPy**

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

**3.1.5 Canny Edge Detection**

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It consists mainly Noise Reduction Stage and Hysteresis Thresholding Stages. In this stage decides which are all edges are really edges and which are not. For this, we need two threshold values, *minVal* and *maxVal*. Any edges with intensity gradient more than *maxVal* are sure to be edges and those below *minVal* are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to “sure-edge” pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:

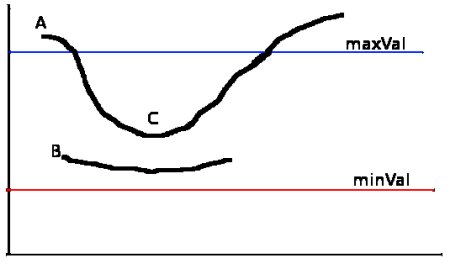


Figure 3.1 Canny Edge detection hysteresis threshold curve

The edge A is above the *maxVal*, so considered as “sure-edge”. Although edge C is below *maxVal*, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above *minVal* and is in same region as that of edge C, it is not connected to any “sure-edge”, so that is discarded. So, it is very important that we have to select *minVal* and *maxVal* accordingly to get the correct result. This stage also removes small pixels noises on the assumption that edges are long lines, we finally get is strong edges in the image.

OpenCV puts all the above in single function, **cv2.Canny()**. We will see how to use it. First argument is our input image. Second and third arguments are our *minVal* and *maxVal* respectively. Third argument is *aperture size*. It is the size of Sobel kernel used for find image gradients. By default, it is 3. Last argument is *L2gradient* which specifies the equation for finding gradient magnitude

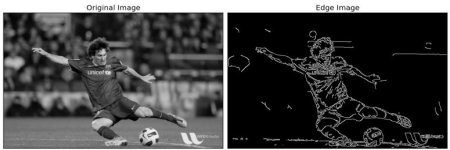


Figure 3.2 Edge detection example

**3.1.6 Gaussian Blurring:**

In this, instead of box filter, gaussian kernel is used. It is done with the function, **cv2.GaussianBlur()**. We should specify the width and height of kernel which should be positive and odd. We also should specify the standard deviation in X and Y direction, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as same as sigmaX. If both are given as zeros, they are calculated from kernel size. Gaussian blurring is highly effective in removing gaussian noise from the image. Given below is example code

blur = cv2.GaussianBlur(img,(5,5),0)

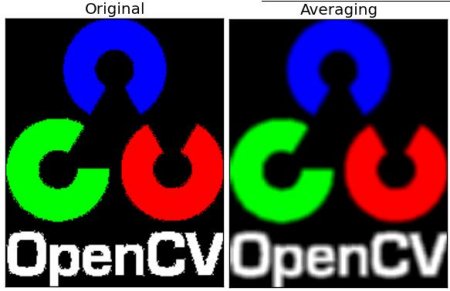


Figure 3.3 Example of Gaussian Blur

**3.1.7 The Hough Transform**

It is a method that is used in image processing to detect any shape, if that shape can be represented in mathematical form. It can detect the shape even if it is broken or distorted a little bit. Hough transform works for line detection using the Houghline transform method. To apply the Houghline method, first an edge detection of the specific image is desirable.

In the **Basics of Houghline Method** a line can be represented as y = mx + c or in parametric form, as r = xcosθ + ysinθ where r is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise ( That direction varies on how you represent the coordinate system. This representation is used in OpenCV).

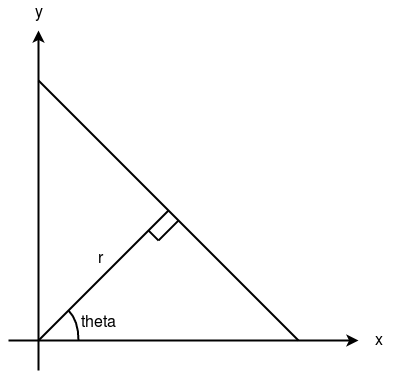
[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/line-detection-1.png)

Figure 3.4 Houghline geometry for straight line

Any line can be represented in these two terms, (r, θ).

**Elaboration of function (cv2.HoughLines (edges,1,np.pi/180, 200)):**

1. First parameter, Input image should be a binary image, so apply threshold edge detection before finding applying Hough transform.
2. Second and third parameters are r and θ(theta) accuracies respectively.
3. Fourth argument is the threshold, which means minimum vote it should get for it to be considered as a line. Number of votes depend upon number of points on the line. So, it represents the minimum length of line that should be detected.

The figure shows Houghline detection in an image of the building



Figure 3.5 Houghline detection example

**3.1.8 TensorFlow**

TensorFlow is an end-to-end open source platform for machine learning by Google. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

TensorFlow for python is a library for fast numerical computing created and released by Google. It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow. The API is nominally for the Python programming language, although there is access to the underlying C++ API.

Unlike other numerical libraries intended for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems. It can run on single CPU systems, GPUs as well as mobile devices and large-scale distributed systems of hundreds of machines.

**3.1.9 TensorFlow Object Detection API**

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models with bounding boxes, objects in images and/or video through models you can train on your own (which the API also makes easier).

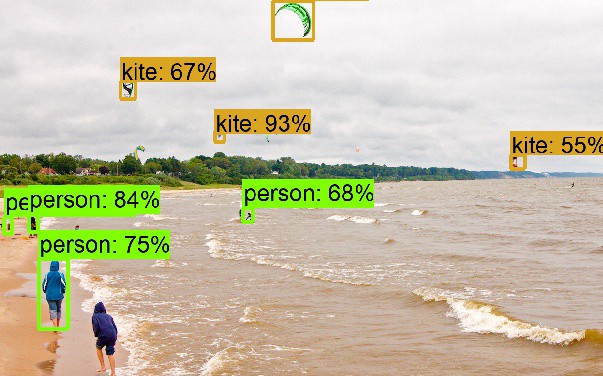


Figure 3.6 TensorFlow object detection example

**3.1.10 CUDA®**

It is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs. The CUDA Toolkit from NVIDIA provides everything you need to develop GPU-accelerated applications. The CUDA Toolkit includes GPU-accelerated libraries, a compiler, development tools and the CUDA runtime. TensorFlow runs up to 50% faster on the latest Pascal GPUs and scales well across GPUs. Now we can train the models in hours instead of days.

**3.1.11 AlexNet**

AlexNet CNN was the winning entry in ILSVRC 2012. It solves the problem of image classification where the input is an image of one of 1000 different classes (e.g. cats, dogs etc.) and the output is a vector of 1000 numbers. The ith element of the output vector is interpreted as the probability that the input image belongs to the ith class. Therefore, the sum of all elements of the output vector is 1.

The input to AlexNet is an RGB image of size 256×256. This means all images in the training set and all test images need to be of size 256×256.

If the input image is not 256×256, it needs to be converted to 256×256 before using it for training the network. To achieve this, the smaller dimension is resized to 256 and then the resulting image is cropped to obtain a 256×256 image.

Multiple Convolutional Kernels (a.k.a filters) extract interesting features in an image. In a single convolutional layer, there are usually many kernels of the same size. For example, the first Conv Layer of AlexNet contains 96 kernels of size 11x11x3. Note the width and height of the kernel are usually the same and the depth is the same as the number of channels.

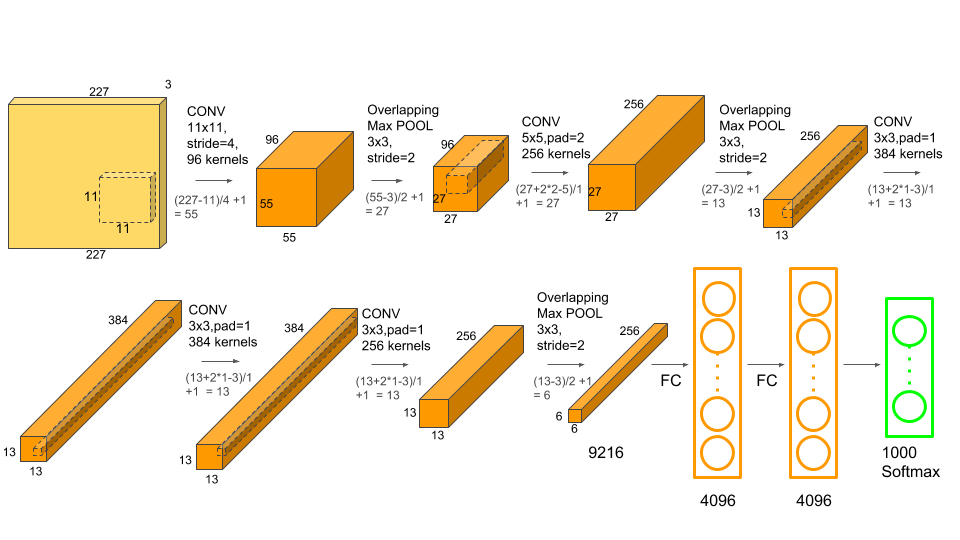
The first two Convolutional layers are followed by the Overlapping Max Pooling layers that we describe next. The third, fourth and fifth convolutional layers are connected directly. The fifth convolutional layer is followed by an Overlapping Max Pooling layer, the output of which goes into a series of two fully connected 

Figure 3.7 AlexNet neural network layers

layers. The second fully connected layer feeds into a softmax classifier with 1000 class labels.

ReLU nonlinearity is applied after all the convolution and fully connected layers. The ReLU nonlinearity of the first and second convolution layers are followed by a local normalization step before doing pooling.

**3.2 HARDWARE**

**3.2.1 Arduino**

Arduino is an open-source hardware and software company, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices and interactive objects that can sense and control both physically and digitally. Its products are licensed under the GNU Lesser General Public License (LGPL) or the GNU General Public License (GPL), permitting the manufacture of Arduino boards and software distribution by anyone. Arduino boards are available commercially in preassembled form or as do-it-yourself (DIY) kits. The figure shows the Arduino Uno R3 board layout.



Figure 3.8 Arduino Uno R3 layout

**Features of the Arduino UNO:**

* Microcontroller: ATmega328
* Operating Voltage: 5V
* Input Voltage (recommended): 7-12V
* Input Voltage (limits): 6-20V
* Digital I/O Pins: 14 (of which 6 provide PWM output)
* Analog Input Pins: 6
* DC Current per I/O Pin: 40 mA
* DC Current for 3.3V Pin: 50 mA
* Flash Memory: 32 KB of which 0.5 KB used by bootloader
* SRAM: 2 KB (ATmega328)
* EEPROM: 1 KB (ATmega328)
* Clock Speed: 16 MHz

Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards or breadboards (*shields*) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers. The microcontrollers are typically programmed using a dialect of features from the programming languages C and C++. In addition to using traditional compiler toolchains, the Arduino project provides an integrated development environment (IDE) based on the Processing language project.

**3.2.2 The NRF24L01**

It is transceiver module. It uses the 2.4 GHz band and it can operate with baud rates from 250 kbps up to 2 Mbps. If used in open space and with lower baud rate its range can reach up to 100 meters.

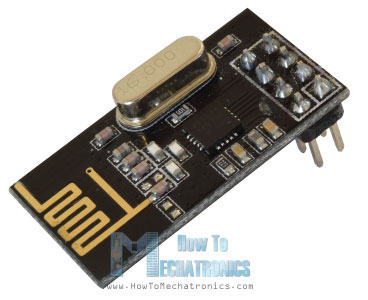


Figure 3.9 NRF24L01 module

The module can use 125 different channels which gives a possibility to have a network of 125 independently working modems in one place. Each channel can have up to 6 addresses, or each unit can communicate with up to 6 other units at the same time.

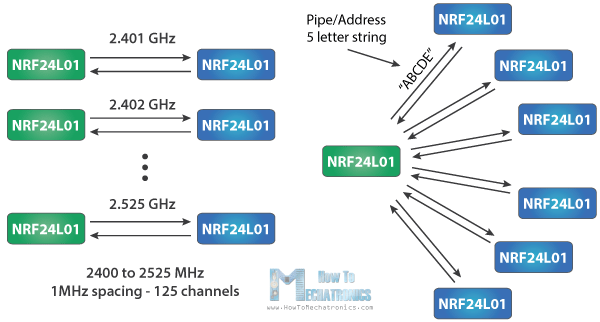


Figure 3.10 NRF24L01 channels and spacing

The power consumption of this module is just around 12mA during transmission, which is even lower than a single LED. The operating voltage of the module is from 1.9 to 3.6V, but the good thing is that the other pins tolerate 5V logic, so we can easily connect it to an Arduino without using any logic level converters.

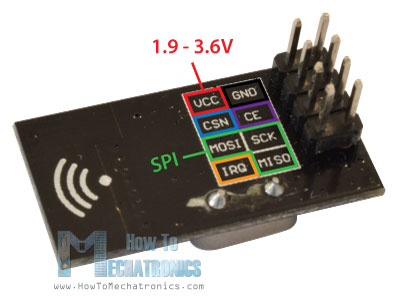


Figure 11 NRF24L01 pin breakout

Three of these pins are for the SPI communication and they need to be connected to the SPI pins of the Arduino, but note that each Arduino board have different SPI pins. The pins CSN and CE can be connected to any digital pin of the Arduino board and they are used for setting the module in standby or active mode, as well as for switching between transmit or command mode. The last pin is an interrupt pin which doesn’t have to be used.

**3.2.3 DC Motor**

A DC motor shown in figure 3.12 is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor.

The DC motor works on the principle of passing a current in a conductor inside a magnetic field. A force is developed on the conductor. When a group of these conductors are fixed on a rotating armature, a resultant torque is produced from all the forces on the individual conductors.



Figure 3.12 A 12 V DC geared motor

A coil of wire with a current running through it generates an electromagnetic field aligned with the centre of the coil. The direction and magnitude of the magnetic field produced by the coil can be changed with the direction and magnitude of the current flowing through it.

A simple DC motor shown in figure 3.13 has a stationary set of magnets in the stator and an armature with one or more windings of insulated wire wrapped around a soft iron core that concentrates the magnetic field. The windings usually have multiple turns around the core, and in large motors there can be several parallel current paths. The ends of the wire winding are connected to a commutator. The commutator allows each armature coil to be energized in turn and connects the rotating coils with the external power supply through brushes. (Brushless DC motors have electronics that switch the DC current to each coil on and off and have no brushes.)

The total amount of current sent to the coil, the coil's size and what it's wrapped around dictate the strength of the electromagnetic field created.

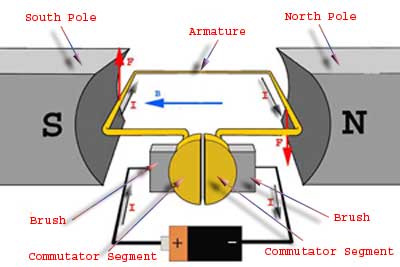


Figure 3.13 Motor construction

The left-hand rule can be used to find the direction of the resultant force on the conductor. The left-hand rule states that if the index of the left-hand points in the direction of the magnetic flux and the middle finger points in the direction of the electric current, the thumb point in the direction of the resultant force.

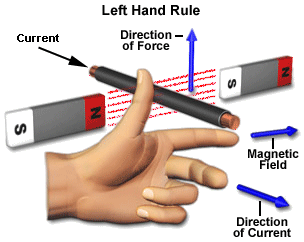


Figure 14 Fleming’s left-hand rule

**3.2.4 Relay Module**

Relays are switches that open and close circuits electromechanically or electronically. Relays control one electrical circuit by opening and closing contacts in another circuit. As relay diagrams show, when a relay contact is normally open (NO), there is an open contact when the relay is not energized.

A relay is an electromagnetic switch operated by a relatively small electric current that can turn on or off a much larger electric current. The heart of a relay is an electromagnet (a coil of wire that becomes a temporary magnet when electricity flows through it). You can think of a relay as a kind of electric lever: switch it on with a tiny current and it switches on ("leverages") another appliance using a much bigger current.



Figure 3.15 8 Channel relay module

.

1.The input circuit (black loop) is switched off and no current flows through it until something (either a sensor or a switch closing) turns it on. The output circuit (blue loop) is also switched off.

2.When small current flows in the input circuit, it activates the electromagnet (shown here as a red coil), which produces a magnetic field all around it.

3.The energized electromagnet pulls the metal bar in the output circuit toward it, closing the switch and allowing a much bigger current to flow through the output circuit.

4.The output circuit operates a high-current appliance such as a lamp or an electric motor.

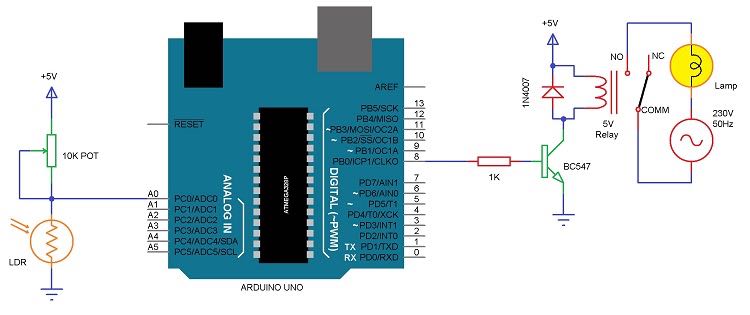


Figure 3.16 Relay circuitry connected to Arduino

**3.2.5 Server Computer**

A server is a computer program or a device that provides functionality for other programs or devices, called "clients". This architecture is called the client–server model, and a single overall computation is distributed across multiple processes or devices. Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients, or performing computation for a client. A single server can serve multiple clients, and a single client can use multiple servers. A client process may run on the same device or may connect over a network to a server on a different device. Typical servers are database servers, file servers, mail servers, print servers, web servers, game servers, and application servers.

**The specification of the computer used for this project are**

* Processor: Intel Core i5-8250U (8th Gen), 4 Cores 8 Threads
* Processor Speed: 1.6 GHz base speed, with Turbo Boost up to 3.4 GHz
* RAM: 8 GB DDR4
* RAM Speed: 2133MHz
* HDD: 1 TB
* Graphics Processor: NVIDIA GeForce 940MX
* Graphics VRAM: 2 GB
* Graphics Memory Type: GDDR5
* Wireless LAN: 802.11 b/g/n
* Power Consumption: 65-Watt AC, 2 Cell Li-ion Battery

**CHAPTER 4**

**PRPOSED WORK**

In this chapter the methodology has been discussed. The work carried out is discussed in two parts. In the first part Houghline Transform is used to find the lanes and send steering commands accordingly. Second part discusses the use of a Convolution Neural Network (CNN) AlexNet to train a model and use it to make steering predictions.

**4.1 Lane detection using Houghline Transform**

The first step of our work is to detect the lanes. This was done by the algorithm given below in the figure 4.1. The algorithm works as follows

Figure 4.1 Flow Chart for Lane Detection Algorithm

1. Grab the incoming video stream from the car and resize it into 800x640 pixels.
2. Convert the RGB frame to grayscale by using inbuilt OpenCV functions.
3. Apply Canny Edge detection function in OpenCV.
4. Apply 5x5 Gaussian Blur to smoothen the image pixels.
5. Crop the region of interest to reduce image size and focus only on useful part.
6. Perform the lane detection by applying Houghline Transform and obtaining two lines(lanes).
7. Calculate the slope of these two lines and provide the output accordingly.

The next step was to use this slope data to make the steering command decision and send the appropriate command to Arduino. This process hierarchy is indicated in Figure 4.2.

Figure 4.2 Send Drive Command to Arduino via Serial Interface

* If the slope of the lines **m1<0 and m2<0** then send **Right Turn** command to Arduino
* If the slope of the lines **m1>0 and m2>0** then send **Left Turn** command to Arduino
* Else send **Go Straight** command to Arduino
* The Arduino communicates with receiver Arduino via RF module link

Then comes the implementation of the TensorFlow Object Detection Algorithm as shown in figure 4.3

Figure 4.3 Object detection using Tensorflow on GPU

**The object detection algorithm works in steps mentioned below**

1. Capture the incoming Stream.
2. Import the object detection module utils.
3. Next, we load the frozen graph of TensorFlow model into memory.
4. After that we load label map indices to categorize name for prediction.
5. Then draw the boxes, classify and visualize the results of detection.
6. Then we calculate approximate distance.

By using the above-mentioned steps, we were able to steer the car logically but the results were not accurate.

**4.2 Training a Neural Network to make driving predictions**

In the next section a convolutional neural network AlexNet applied to train a model and use predictions based on this model to send steering commands for accurate results.

For the convolutional network we need to acquire the data, train a model and test it for its correct working.

The figure 4.4 gives the algorithm for collecting the data.

Figure 4.4 Create training data

The process is as follows

1. Grab the frame from the video stream.
2. Convert the RGB image to Greyscale image and resize it to a more acceptable size for CNN.
3. Next, drive it manually via computer and we log the corresponding key to frames.
4. Then store the image and key data to NumPy file.

The next step is to train the module from the collected data, shown in figure 4.5. The training progress is shown in figure 4.6 The following are the steps for implementation of the algorithm.

1. Set the width, Height and Learning Rate for the CNN.
2. Send these parameters to AlexNet DCNN.
3. Import the balanced training dataset that is already created.
4. Set the training and test values for the data specified.
5. Train the network for specified epochs keeping a watch for loss and accuracy and save it.

Figure 4.5 Training Dataset using TensorFlow AlexNet on GPU

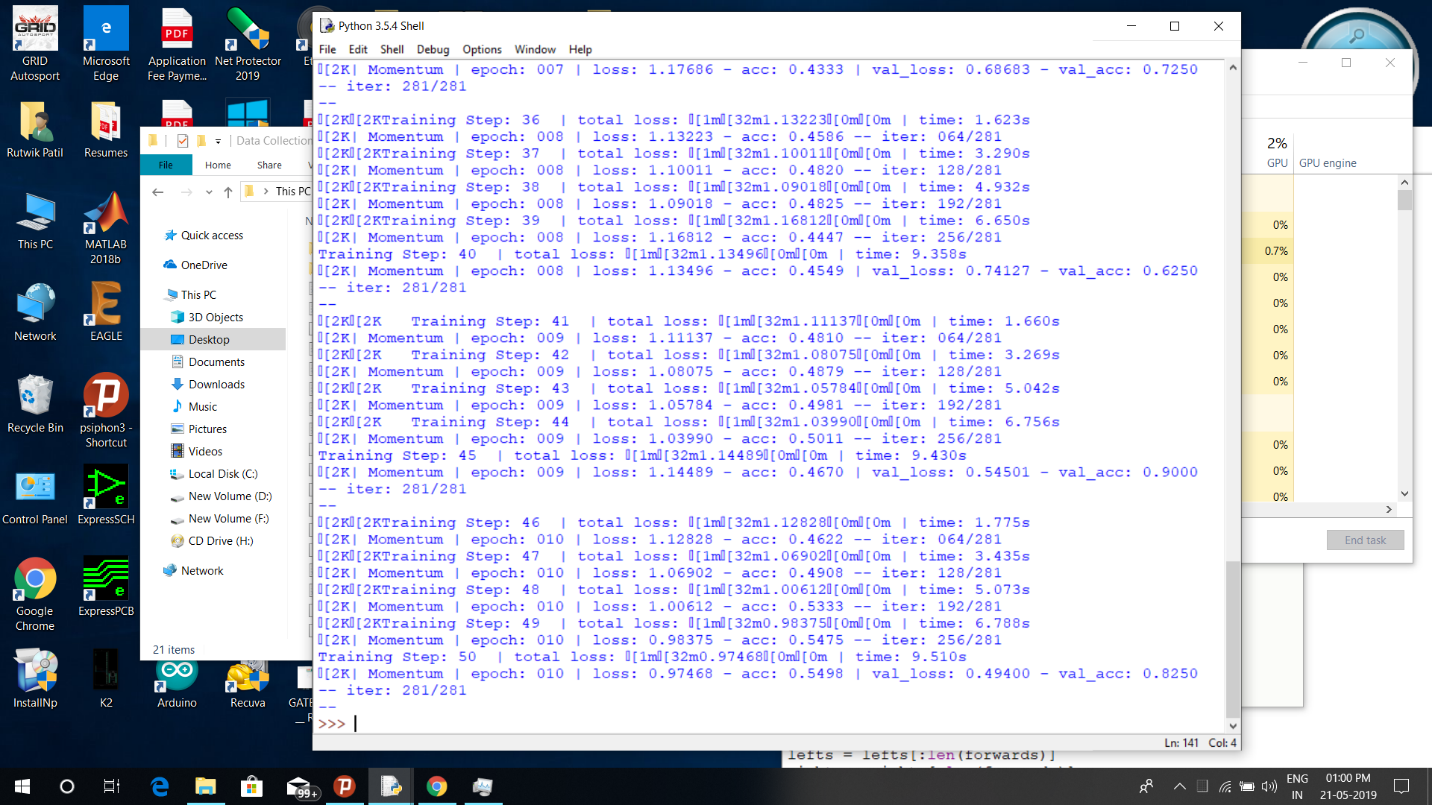


Figure 4.6 Screenshot of the model training

Finally, in the last step use the trained model for real-time prediction to drive the car, shown in figure 4.6.

Figure 4.7 Using Trained model to drive the car

The algorithm works as follows

1. Grab the incoming video stream and convert it to greyscale and resize it to fit the CNN.
2. Load the trained model and CNN with identified parameters.
3. Apply the prediction model.
4. Based on prediction, the program sends drive commands to Arduino via serial interface.

**4.3 Block diagram of the project**

The block diagram of the project is given in figure 4.8.

It consists two parts transmitter and receiver,

* Transmitter contains computer server, Arduino and NRF module.
* Receiver side contains Arduino, NRF module and relay module mounted on car.

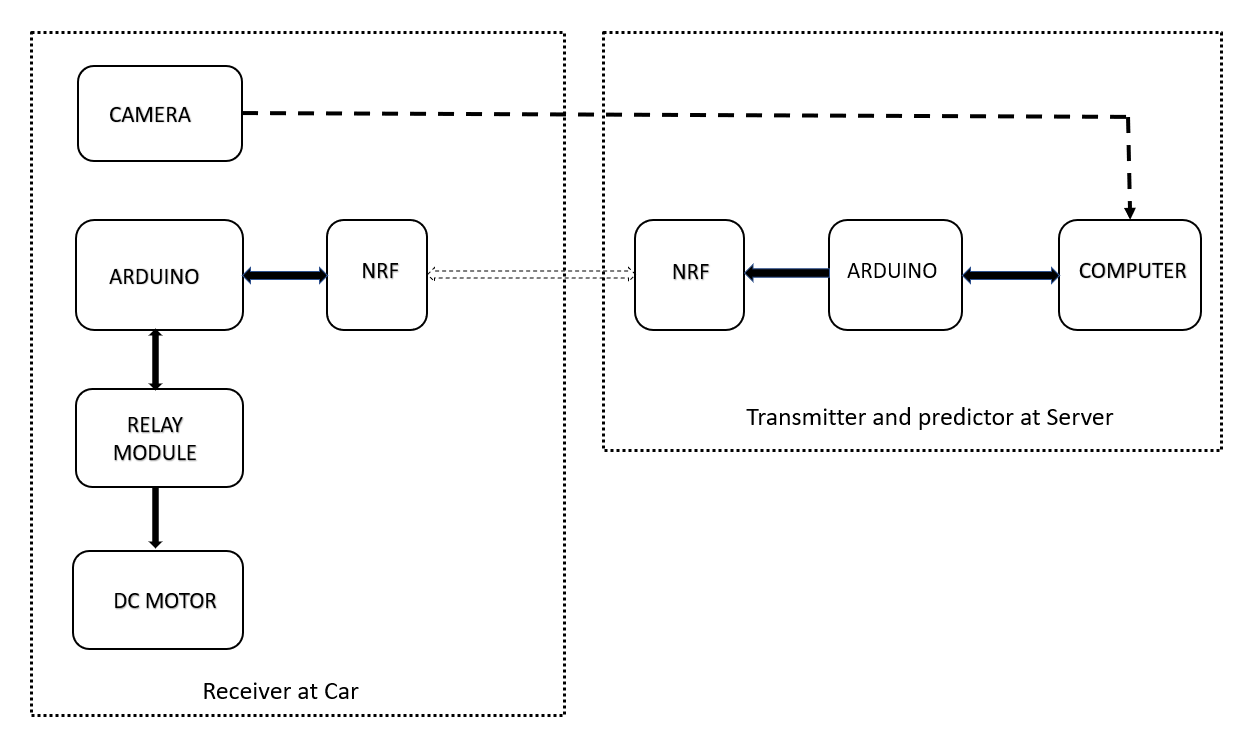
****

Figure 4.8 Block Diagram of the project

**CHAPTER 5**

**RESULTS AND CONCLUSION:**

**5.1 Results**:

* In this project, a self-driving car based on IoT and RF is successfully built. This uses entirely open source software (OpenCV, TensorFlow, Python) and hardware (Arduino) technology
* By using open source technologies, cost of the system has been reduced.
* Successfully used the CNN (Convolution Neural Network) for learning and controlling of car is done and virtual remote control by human is provided if necessary.
* Single camera is used for both navigation and obstacle avoidance.

**5.2 Conclusions**:

In this project, a method to make a model of self-driving car is presented. The different hardware components along with software and neural network configuration are clearly described. With the help of image processing and machine learning a successful model is developed which worked as per expectation. Thus, the model was successfully designed, implemented and tested.

**5.3** **Future Scope**:

Scope and reliability of this model can be improved. As observed, the car slightly moves out of the track which can be a serious issue if it hits nearby objects if a real car is considered. An advanced system can be developed which would take care of this issue, it would not only make the system reliable but at the same time it would make the overall design attractive and risk-free from accidents.

**REFERENCES:**

1. Aditya Kumar Jain “Working model of Self-driving car using Convolutional Neural Network, Raspberry Pi and Arduino”, *Proceedings of the 2nd International conference on Electronics, Communication and Aerospace Technology (ICECA 2018) IEEE Conference Record* # 42487; IEEE Xplore ISBN:978-1-5386-0965-1
2. Michael G. Bechtel, Elise McEllhiney, Minje Kim, Heechul Yun “DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car” *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*
3. Abdur R. Fayjie, Sabir Hossain, Doukhi Oualid, and Deok-Jin Lee “Driverless Car: Autonomous Driving Using Deep Reinforcement Learning in Urban Environment” *2018 15th International Conference on Ubiquitous Robots (UR) Hawaii Convention Center, Hawai'i, USA, June 27-30, 2018*
4. Truong-Dong Do, Minh-Thien Duong, Quoc-Vu Dang and My-Ha Le “Real-Time Self-Driving Car Navigation Using Deep Neural Network” *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*
5. Wen-Yen Li, Wang-Hsin Hsu, Yi-Yuan Chiang “A Combination of Feedback Control and Vision-Based Deep Learning Mechanism for Guiding Self-Driving Car” *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*
6. B Padmaja, P V Narasimha Rao, M Madhu Bala, E Krishna Rao Patro, “A Novel Design of Autonomous Cars using IoT and Visual Features” *Proceedings of the Second International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC 2018) IEEE Xplore Part Number:CFP18OZV-ART; ISBN:978-1-5386-1442-6*
7. Hajer Omrane, Mohamed Slim Masmoudi, Mohamed Masmoudi “Neural controller of autonomous driving mobile robot by an embedded camera” *4th International Conference on Advanced Technologies For Signal and Image Processing - ATSIP' 2018 March 21-24, 2018 - Sousse, Tunisia*
8. <https://www.python.org/>
9. <https://www.arduino.cc/>
10. <https://pypi.org/project/pyserial/>
11. <https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html>
12. <https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html>
13. <https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html>
14. <https://www.tensorflow.org/>
15. <https://github.com/tensorflow/tensorflow>
16. <https://github.com/tensorflow/models/tree/master/research/object_detection>
17. <https://developer.nvidia.com/cuda-zone>
18. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf