CHAPTER 1

INTRODUCTION

As we all know our society is facing a lot of problems, one of which is road accidents. One serious road accident in the country occurs every minute and 16 lives are lost on Indian roads every hour. Worldwide, the number of people killed in road traffic crashes each year is estimated at almost 1.2 million, while the number injured could be as high as 50 million. The statistic says that 1214 road crashes occur every day in India. With rapid urbanization, India has seen an unprecedented growth of motor vehicles. Safety studies have found that a majority of accidents occur either due to the driver's error or due to the negligence of the safety norms.

The statistics show that a large number of road accidents take place at blind road corners (hairpin curves) where the drivers are not able to see the incoming vehicle. An example is NH-22 from Solan to Shimla, which consists of sharp bends and turns. More than 3000 accidents, 780 deaths and 5500 serious injuries were reported in this region from 2003-2013. These reasons warrant an efficient reliable system to alert drivers to approaching vehicles along a curve.

1.1 OVERVIEW

The main idea of this proposal is to ensure safe travel in hilly areas and to avoid any possibility of accidents in hill/street curves. These curves provide either partial or no-visibility of the oncoming traffic to the drivers. To address this problem, a system is to be developed to warn drivers about the approaching traffic in hill curves. So, by combining web cameras and warning signals like LEDs and buzzers, with thorough knowledge of IoT and Computer vision, we have designed a model not only to avoid road accidents but also to resolve confusion about who should be the first to move.

1.2 PROBLEM STATEMENT

At the hairpin curves/blind curves drivers won't be aware of approaching vehicles from the other side of the curve and most of the accidents occur at these curves. So, to avoid collisions, a system is designed which ensures the safety of the people travelling in these regions with blind curves. With respect to this, the work can be considered as a lifesaving model, which provides a peaceful journey.

1.3 STUDY AREAS

Our area of study includes IoT, machine learning and computer vision.

- IoT - The internet of things (IoT) is a computing concept that describes the idea of everyday physical objects being connected to the internet and being able to identify themselves to other devices.

- Machine learning - Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

- Computer vision – Computer vision is a field of computer science that works on enabling computers to see, identify and process images in the same way that human vision does, and then provide appropriate output. It is like imparting human intelligence and instincts to a computer to recognize images of different objects.

1.4 OBJECTIVE

- To develop a technology which can reduce the accidents in hairpin bends, blind curves and U turns. Through this project, we will be able to reduce this type of problem.

- To alert the drivers by warning them about the approaching vehicle using red LEDs (to alert and stop), green LEDs (to cross safely) and buzzers to alert the drivers because the theory says that humans instinctively react to sound.

- It also reduces the confusion among the drivers so as to which vehicle should move first based on the programmed preference.

1.5 METHODOLOGY

- Vehicles speeding along a curve are not aware of the presence of vehicles coming from the other direction. Here, a system is proposed to alert drivers going around a blind curve to the presence of oncoming vehicles.

- Cameras, LED warning lights and piezoelectric buzzers are mounted on a pole which is then connected to a Raspberry Pi and video from the cameras is processed to detect oncoming vehicles.

- XML Classifiers for cars, trucks and bikes are generated using LBP Cascade training. The live video feed from two cameras placed on either side of the curve is given to the OpenCV algorithm to detect presence of vehicles. If vehicles approaching on both the sides of the curve are detected, the buzzers and lights are activated, thus alerting the drivers to be aware of the incoming vehicle. Once they are alerted with red LEDs, one of the two green LEDs will glow indicating that particular vehicle to move.

- The red LEDs and buzzers will be switched on for specified amount of time indicating the sign of warning the driver and then green LEDs will glow to let vehicles to cross depending on the program.

1.6 ORGANISATION OF REPORT

This report is divided up into chapters, each dealing with different aspects of the project. Each chapter has a short introduction, explaining the subject of each chapter, and then the details of each module is explained separately. The following is a short overview of each of the next 8 chapters:

Chapter 2: This chapter particularly deals with literature survey. It outlines some of the research made on the project in the beginning. More research was made

as this project report was being developed, as new areas had to be investigated. It includes the details about the existing system as well as the proposed system.

Chapter 3: Specifies the various requirements specification such as system analysis, software requirements and hardware requirements for the proposed system.

Chapter 4: Specifies the hardware approach used in the project.

Chapter 5: Specifies the software approach used in the project.

Chapter 6: Specifies about the system design.

Chapter 7: Specifies the implementation details of how the model works.

Chapter 8: It deals with the results and discussion about the project.

Chapter 9: The conclusion and the future scope of the project is discussed in this chapter.

CHAPTER 2

LITERATURE SURVEY

This chapter provides an introduction to the areas of research. It describes the work which has already been done in direct-show and states the new scope. The scope has been clearly explained and the technology used to obtain this has been mentioned in this chapter.

2.1 EXISTING SYSTEM

- Vehicle Horn - The drivers on both sides judge the distance of one another based on the intensities of sound from their respective horns. This method although being the simplest poses to be highly inefficient also causing a lot of confusion between the drivers. [1]

- Headlights - Flashing headlights during the night time works similar to the vehicle horn making it yet another inefficient method to alert the vehicle. Also, this method is completely ineffective during day time due to sunlight. [1]

- Convex Mirrors- This setup is most widely used nowadays to give a glimpse of any vehicle approaching the hairpin bend from the opposite end. But, these have their shortcomings such as the mirror needs to be kept clean at all times which is difficult in hilly areas as its always cold and misty, thereby reducing its visibility. [1]

- Infrared proximity sensors -The proposed system contains a set of proximity sensors, warning lights combined with a convex mirror is installed by the

side of the road. It uses four IR sensors, which are placed on either side of the hairpin bend. Based on the output of sensors, position of vehicles on either side of the bend is detected which is provided as an input to the microcontroller. The priority algorithm intelligently controls the movement of the vehicles at the hairpin bend based on the sensor values giving appropriate warnings. IR proximity sensors are incapable of distinguishing between objects that irradiate similar thermal energy levels. So, they may lead to false detections. Infrared detectors are also rather expensive, so they are not as widely used as they could be. [1]

- RF transmitter attached to vehicles GPS technology – It takes care of the location of the vehicles with respect to the hairpin bend to decide the priority in which vehicles have to move. Information exchange between vehicles through RF transmitter and receiver, regarding speed, distance, direction and vehicle type is captured by the system and the decision based on algorithms is passed on to the vehicle by voice and visual display. Speed of the vehicle is automatically controlled using GPS information regarding the nature of curve. In case of a vehicle breakdown in the control zone, the information is sent to other vehicle for suitable action [2]

- Mishap prevention system using Ultrasonic transceiver and Arduino - This system makes use of ultrasonic sensor and other embedded systems. The 'heart' of the Unit is Arduino microcontroller which performs all the vital tasks of the system. This system will receive information from the Ultrasonic transceiver, and accordingly transmit the data via the Wi-Fi router to the controller. When the vehicle at any of the hairpin bends is detected by the Ultrasonic transceiver it sends the signal to the microcontroller via wireless medium Wi-Fi router. The router transmits the signal to the android phone through Wi-Fi hotspot. The IP address of the mobile phone has to be

configured with the Wi-Fi router's IP address with the help of a microcontroller. When the vehicle driver has installed this android application, the traffic information will be delivered as voice recognition. Changes in nature, for example, temperature, weight, mugginess, air turbulence, and airborne particles influence ultrasonic reaction. Sensor reaction times are ordinarily slower. [3]

- Real time approaching vehicle detection using cameras- This paper presents an image-based method to detect approaching objects in blind-spot area and proposes a verification method by using the recorded video database from real traffic environment. Cameras are installed on two sides of a vehicle to capture the images of blind spots, and the driver is alerted to watch the approaching vehicles as driver would like to change the lane. By taking video frames and converting the images into one dimensional information, the image entropy of the road scene in the near lane are estimated. Thus, by analysis the lane information, an object will be detected and located in a constant time. 2 cameras must be attached to all vehicles, which will be difficult to implement for all vehicles. [4]

2.2 PROPOSED SYSTEM

As we all know, a large number of road accidents take place at blind road corners (hairpin curves) where the drivers are not able to see the incoming vehicle. Vehicles taking a turn assuming no other vehicle is at the opposite end and end up crashing. So, a system is proposed to alert drivers going around a blind curve to the presence of oncoming vehicles.

Two poles are erected on either side of the curve, bearing cameras mounted over the road. Piezoelectric buzzers are used to alert the driver with sound and LED

warning lights are used to warn the vehicles and co-ordinate their movement. All these components are connected to a Raspberry Pi 3 B through jumper wires.

The live video feed from the cameras is processed using image processing algorithms to detect the presence of vehicles. If vehicles are approaching on both sides of the curve at the same time, both the buzzers and red LED lights are activated indicating the presence of the vehicle, thus alerting the drivers of the vehicle coming from the other side. If the driver fails to notice the lights, the buzzer will alert him.

This system gives 2-layer security. This system also guides the drivers using green LEDs so as to allow one vehicle to move past the curve first and then let the second vehicle pass, preventing collision.

CHAPTER 3

SYSTEM ANALYSIS AND REQUIREMENTS

Analysis is the process of breaking a complex topic into smaller subtopics understand it better. Here, analysis had been done based on three aspects: System analysis, Requirements analysis and Functional requirements. System analysis comprises of relevance of the platform and relevance of the programming language. Requirements analysis involves specifying the scope, boundary and assumptions. Functional requirements specify about hardware and software requirements.

3.1 SYSTEM ANALYSIS

Here, the analysis of the system is made with respect to relevance of platform, programming languages.

3.1.1 Relevance of platform

This system can work on Raspberry Pi with Raspbian Stretch operating system installed. OpenCV 3.3.0 is installed with Python 3 bindings on the Raspberry Pi, running Raspbian Stretch. The Raspberry Pi itself doesn't come with an operating system. Installation of the OS image onto the microSD card was done using Win32DiskImager.

3.1.2 Relevance of Programming Language

Python is a very relevant force in the modern software engineering landscape. Python is one of the most popular languages measured on GitHub and has a huge amount of support as a popular in-house language at Google. Python is approachable enough that educators use it as an introductory programming language, and experienced developers use it to build side projects for fun. User community and forums for Python provide broad support, making it easy to debug errors.

The broad variety of powerful functions in Python make it suitable for use with our project. GPIO control is also very easy in Python.

3.2 REQUIREMENTS ANALYSIS

Requirement analysis explains the scope, boundary and assumptions of the project.

3.2.1 Scope and Boundary

The users will need cars to use the system. The system is meant to be implemented on curved roads.

3.2.2 Assumptions and Dependencies

It is assumed that a stable power supply is available, to provide power for the Raspberry Pi and the power hub. We assume that the road under consideration poses a high risk for accidents.

3.3 FUNCTIONAL REQUIREMENTS

Functional requirements are of two types: software and hardware requirements.

3.3.1 Software Requirements

- Python 3
- Raspbian Stretch OS
- OpenCV 3.3.0

3.3.2 Hardware Requirements

- Raspberry Pi 3 B

  The Raspberry Pi 3 Model B is the earliest model of the third-generation Raspberry Pi. Its features include:

  - Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
  - 1GB RAM
  - BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
  - 40-pin extended GPIO
  - 4 USB 2 ports
  - 4 Pole stereo output and composite video port
  - Full size HDMI
  - CSI camera port for connecting a Raspberry Pi camera
  - DSI display port for connecting a Raspberry Pi touchscreen display
  - Micro SD port for loading your operating system and storing data
  - Upgraded switched Micro USB power source up to 2.5A

- Micro SD card

  A class 10 16 GB SanDisk microSD card is used, to hold the Raspbian OS.

- Jumper Wires

  A jumper wire is an electrical wire with a connector or pin at each end, which are normally used to interconnect the components of a breadboard internally or with other equipment.

- Powered USB hub

  A USB hub is a device that expands a single Universal Serial Bus (USB) port into several so that there are more ports available to connect devices to a host system, similar to a power strip. Powered USB hubs have a power adapter which powers the devices connected to it.

- Monitor

  Monitor is a display screen used to provide visual output from a computer, cable box, video camera, VCR or other video generating device. Computer monitors use CRT and LCD technology. GUI of the Raspbian OS is displayed on the monitor.

- Keyboard and Mouse

These are used to interact with the GUI of the Raspbian OS.

- LEDs - 4

A light-emitting diode (LED) is a semiconductor device that emits visible light when electric current passes through it. The light is not particularly bright, but in most LEDs, it is monochromatic, occurring at a single wavelength. LEDs must be used in series with resistors.

- Buzzers - 2

A buzzer or beeper is an audio signalling device, which may be mechanical, electromechanical, or piezoelectric. Here, we have used a piezoelectric buzzer. Piezo buzzers produce sound based on reverse of the piezoelectric effect.

- 330Ω Resistors - 2

330Ω resistors make excellent LED current limiters. Not using resistors with LEDs can cause damage to the GPIO board of the Pi.

CHAPTER 4

HARDWARE APPROACH

4.1 PHYSICAL SETUP OF RASPBERRY PI AND PERIPHERALS

The system is centered around a Raspberry Pi 3 B. A monitor is connected via VGA-to-HDMI converter to the HDMI input port of the Pi.

A powered USB hub is used to connect all USB peripherals to the Pi, to prevent draining of power from the Pi and to ensure constant power supply to peripherals. A USB keyboard, USB mouse, and 2 USB cameras are connected to the 4 ports of the hub. Power adapter of the hub is connected to the external power source. A USB 3.0 Data Cable is used to connect the hub to the USB port of the Raspberry Pi.

The Pi is powered using a 5V 2.1A power source
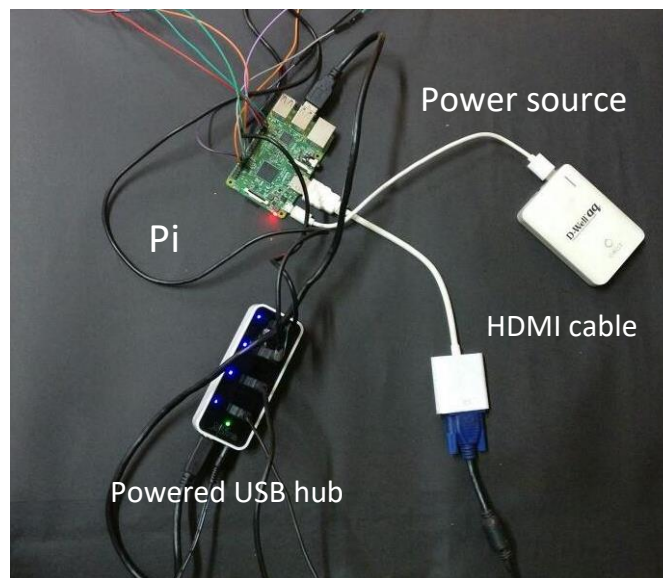
Figure 4.1 Connection setup of Pi

4.2 GPIO ALARM SUB-SYSTEM CIRCUITRY
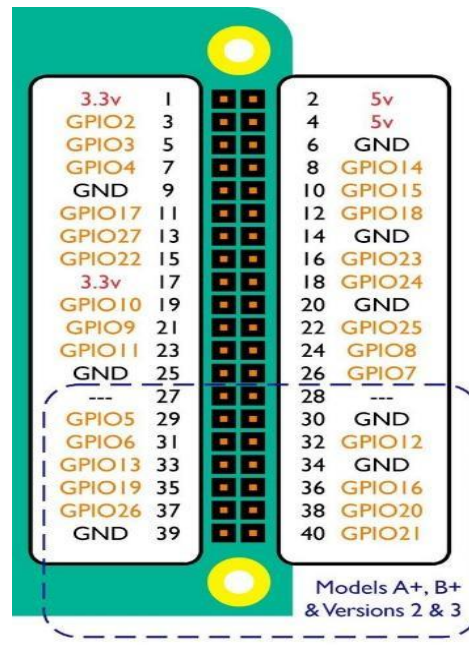
Figure 4.2 Pin diagram of Raspberry Pi, with USB ports facing downwards

The Raspberry Pi has a GPIO (General Purpose Input Output) board with 40 pins. The pins are numbered in 2 ways — physical numbering and Broadcom numbering (BCM). In the former, the pins are numbered sequentially from 1 to 40, as shown in Figure 4.1.

The Broadcom numbering system is the default option for the SoC (System-on-Chip). Raspberry Pi uses a SoC developed by Broadcom Limited. Raspberry Pi 2 and Zero use BCM2836 and BCM2835, while the Pi 2 version 1.2 and 3 use BCM2837. This is also known as GPIO numbering and is shown as GPIO# in Figure 4.1.

There are 8 ground pins and two +5V pins and three +3.3V pins, which cannot be programmed. The Python package used for Raspberry Pi GPIO programming is *RPi.GPIO.*

Our alarm system consists of 2 breadboards, each bearing connections to one red LED, green LED and one piezoelectric buzzer, which are mounted on a pole on either side of the curve. The negative ends of all the LEDs are connected in series with 330Ω resistors. The positive ends are connected to programmable pins. Buzzers are connected in a similar fashion but without a resistor. Each breadboard is placed on either side of the curve.

Figure 4.3 Circuit diagram for GPIO alarm subsystem

On breadboard 1,

- The negative pin of the red LED is connected in series with R_1 to pin 6 (GND).
- The positive pin of the red LED is connected to pin 12.
- The negative pin of the green LED is connected along with R_2 parallelly to the GND line of the red LED.
- The positive pin of the green LED is connected to pin 38.
- The negative pin of the buzzer is connected parallelly to the ground connection of the red LED.
- The positive pin of the buzzer is connected to pin 13.

On breadboard 2,

- The negative pin of the red LED is connected in series with R_3 to pin 9 (GND).
- The positive pin of the red LED is connected to pin 16.
- The negative pin of the green LED is connected along with R_4 parallelly to the GND line of the red LED.
- The positive pin of the green LED is connected to pin 40.
- The negative pin of the buzzer is connected parallelly to the ground connection of the red LED.
- The positive pin of the buzzer is connected to pin 11.
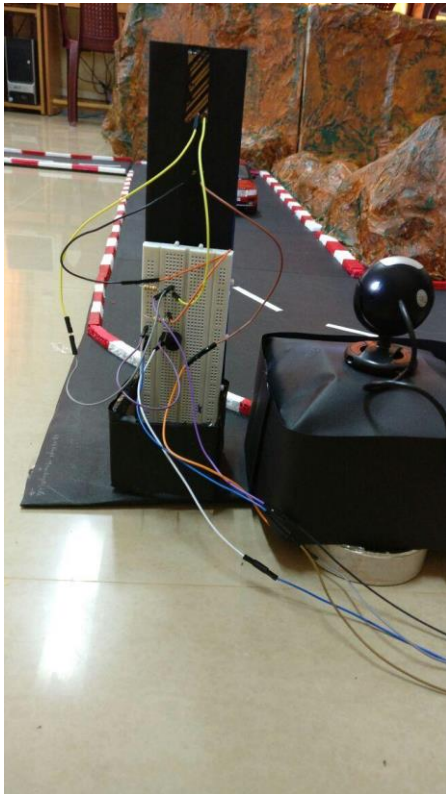
Figure 4.4 Breadboard 1
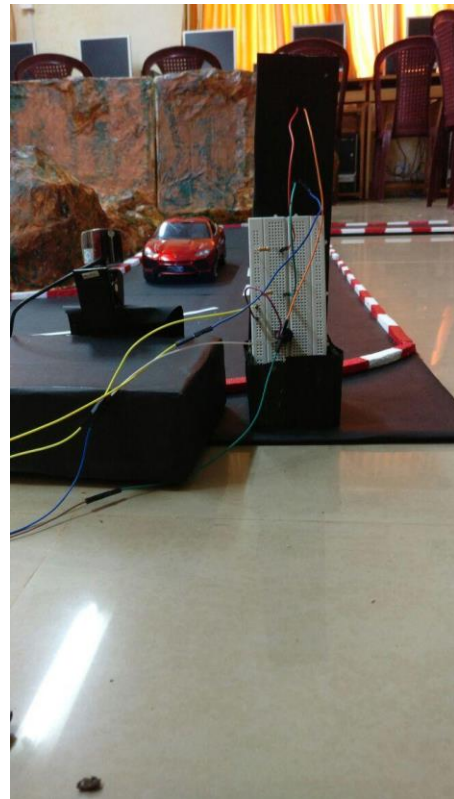
Figure 4.5 Breadboard 2

CHAPTER 5

SOFTWARE APPROACH

Software approach is a description of the techniques used in solving an engineering problem.

5.1 OPENCV

Computer vision deals with making computers analyze and understand images or videos, as humans do. It is intended as a replacement for human surveillance of videos/images.

OpenCV is an open-source library of functions for real-time computer vision applications, thus making it the most suitable for real time situations such as ours. OpenCV is written in C++ and its main interface is in C++. It utilizes multi-core processing to speed up performance. It provides interfaces in Python, Java and MATLAB. We are using the Python API, OpenCV-Python in our system. The version of OpenCV being used is 3.3.1

The functionalities of OpenCV include:

- Image/video I/O processing, display
- Feature detection
- Geometry-based monocular or stereo computer vision
- Computational photography
- Machine learning library including artificial neural networks, clustering, boosting, Support Vector Machines(SVM) etc.

5.2 CASCADE OF BOOSTED CLASSIFIERS

Cascading is based on concatenating many classifiers. The output of one classifier serves as additional information for the next classifier in the cascade. Unlike voting or stacking, which are multi expert systems, cascading is a multistage one. The first implementation of cascade classifiers was the Viola-Jones object detection framework (2001), mainly geared towards face detection using Haar-like features.

Cascading classifiers are trained with hundreds of positive sample views of a particular object, all scaled to the same size and random negative images, which don't contain the object to be detected. After the classifier is trained, it can be applied to a region of an image (of same size as the training model size) and detect the object. The classifier outputs a "1" if the region is likely to show the object (i.e., car), and "0" otherwise. To search for the object in the entire frame, the search window can be moved across the image and check every part for the classifier. A cascade of gradually more complex classifiers achieves better detection rates, since the first few stages serve as filter, to weed out the negatives. The goal is to reject negative samples as early as possible. So, if a sample fails the first stage, subsequent stages are not checked. A sample is classified as an object only if it passes all the stages in the cascade.

The classifier can be easily resized in order to be able to find the objects of interest at different sizes, instead of resizing the image itself. So, to detect an object of unknown size in the image, the scan procedure should be repeated at different scales. This level of scaling can be adjusted in the scaleFactor parameter of **detectMultiScale()**.

Here, we have used a cascade of boosted classifiers. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques.

Currently, Discrete AdaBoost, Real AdaBoost, Gentle AdaBoost and LogitBoost are supported. The basic classifiers are decision-tree classifiers with at least 2 leaves.

The type of boosted classifier we have generated is **Gentle AdaBoost**.

Gentle AdaBoost (Adaptive Boost) initialize a strong learner (usually a decision tree) and iteratively create a weak learner that is added to the strong learner. At each iteration, adaptive boosting increases the weights of the wrongly classified instances and decreases the ones of the correctly predicted instances. The weak learner thus focuses more on the difficult instances. After being trained, the weak learner is added to the strong one according to his performance. The better it performs, the more it contributes to the strong learner. The process is repeated until required error rate is achieved.

5.3 LBP FEATURES

The input to the basic classifiers (stages) are LBP (Local Binary Pattern) features. This is specified during training by setting the featureType parameter to LBP. The training algorithm selects features with minimum error rate, which classify the object most accurately. LBP features compute local texture information, by comparing each pixel with its surrounding neighborhood of pixels.

The first step in constructing the LBP texture descriptor is to convert the image to grayscale. For each pixel in the grayscale image, we select a neighborhood of size r surrounding the center pixel. A LBP value is then calculated for this center pixel and stored in the output 2D array with the same width and height as the input image.

For example, let's take a look at the original LBP descriptor which operates on a fixed 3x3 neighborhood of pixels.
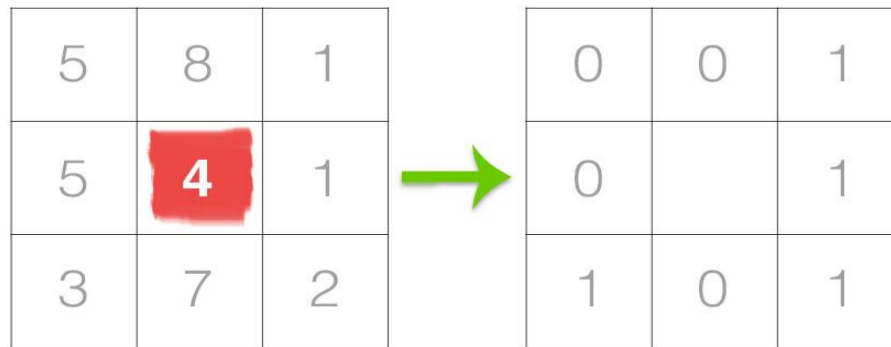
Figure 5.1 Thresholding of 8-pixel neighborhood around center pixel

In the above figure we take the center pixel (highlighted in red) and threshold it against its neighborhood of 8 pixels. If the intensity of a neighborhood pixel is greater-than-or-equal to the central pixel, then we set the value to 1; otherwise, we set it to 0.

From there, we need to calculate the LBP value for the center pixel. We can start from any neighboring pixel and work our way clockwise or counter-clockwise, but our ordering must be kept consistent for all pixels in our image and all images in our dataset. The results of the binary test are stored in an 8-bit array, which we then convert to decimal, like this:
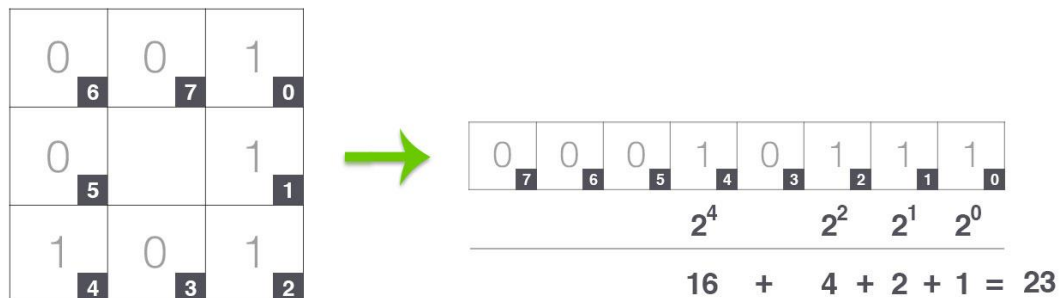
Figure 5.2 Converting binary neighborhood of the center pixel into decimal representation

In this example we start at the top-right point and work our way clockwise accumulating the binary string as we go along. We can then convert this binary string to decimal, yielding a value of 23.

This value is stored in the output LBP 2D array, which we can then visualize below:
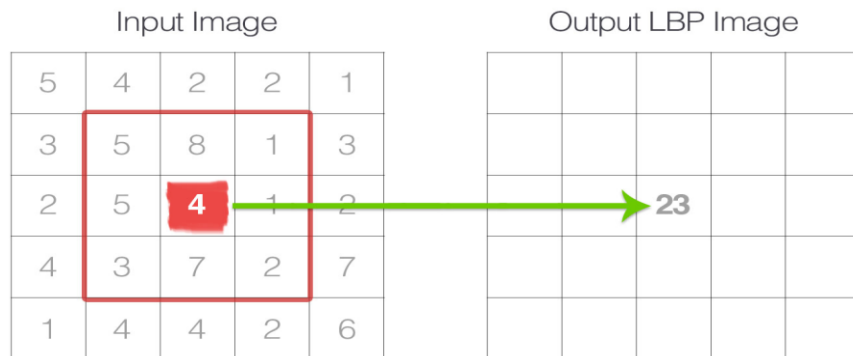
Figure 5.3 The calculated LBP value is then stored in an output array with the same width and height as the original image

This process of thresholding, accumulating binary strings, and storing the output decimal value in the LBP array is then repeated for each pixel in the input image.

LBP is seen to work well in spite of monotonic greyscale conversions.
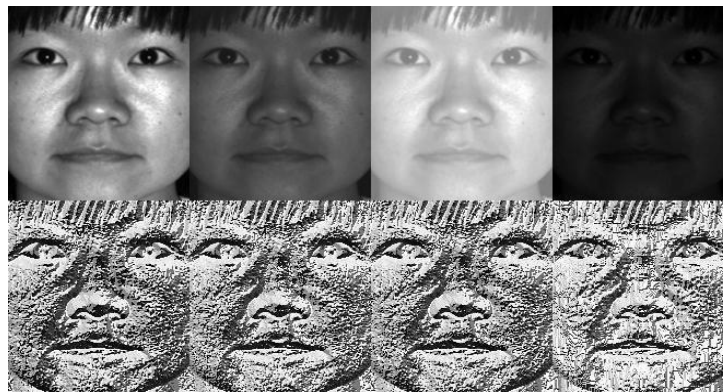
Figure 5.4 LBP representation of an image

CHAPTER 6

SYSTEM DESIGN

Systems design is the process of specifying the architecture, modules, interfaces, and data for a system to satisfy specified user requirements.

Block diagram describes the major components and modules of a system and shows how they are interconnected. The block diagram for our system is given below. The Raspberry Pi serves as the master controller of the components. It takes input from the cameras and processes it using the OpenCV algorithm to detect vehicles, based on which it controls the GPIO alarm systems on both sides of the curve.
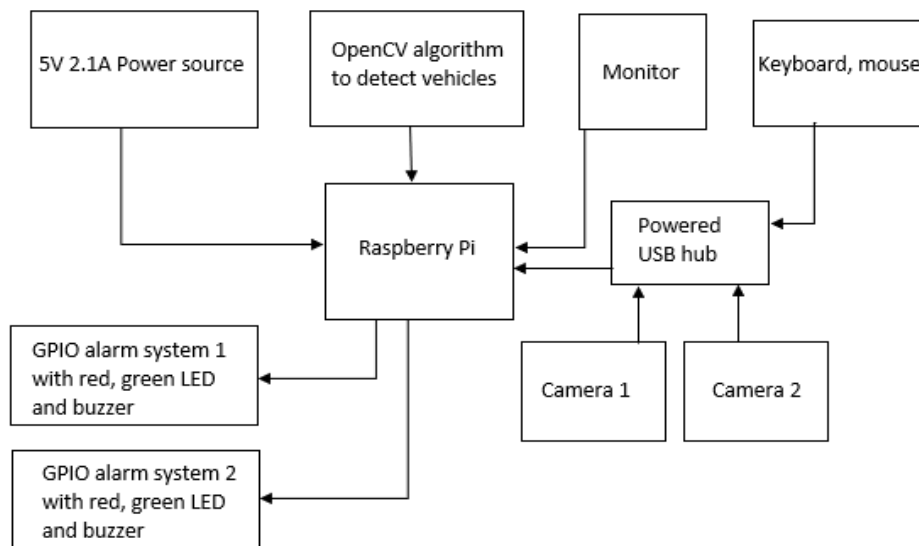
Figure 6.1 Block diagram of the system

Activity diagram describes the flow of control in a system. It consists of activities and links. It describes how the execution of the system occurs. The activity diagram for our system is given below.
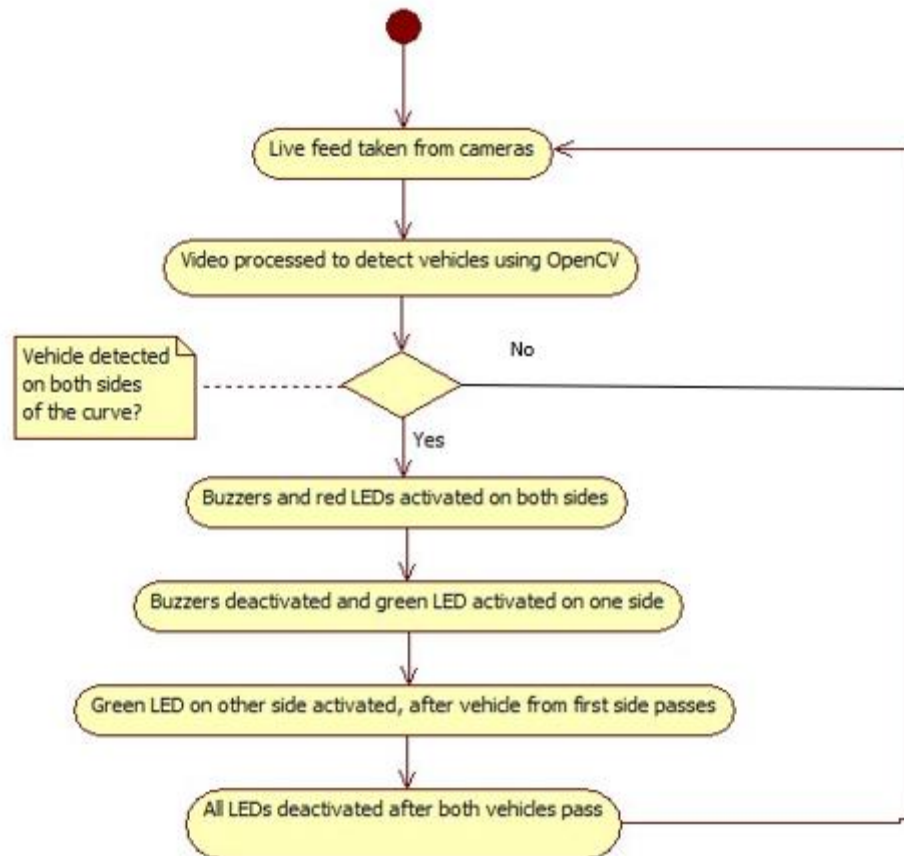
Figure 6.2 Activity diagram of the system

The execution begins with opening the cameras to begin taking in the live feed. The frames from both cameras are processed using OpenCV to detect any vehicles. If no vehicles are detected on both sides, next frames are taken in from the cameras and processed. If vehicles are detected on both sides at the same time, the buzzers and red LEDs are activated on both sides to alert the drivers to slow down. Then the

green LED is activated on one side to allow one vehicle to move forward. After it passes, the other vehicle is allowed to move by activating the green LED on the other side. Red LED is reactivated on the previous side, to stop any vehicles that were behind the first vehicle. After all vehicles pass, all LEDs are deactivated and the algorithm proceeds with fresh frames from the cameras

CHAPTER 7

IMPLEMENTATION

The training of the LBP cascade classifier, detecting vehicles and the display of the output is done using OpenCV and Python. This chapter gives the implementation details of the entire system. The major modules in our system are:

- Training of the cascade classifier for different types of vehicles
- OpenCV code to detect vehicles from live feed
- GPIO alarm module

7.1 TRAINING OF CASCADE CLASSIFIER

The major steps in creating the classifier are:

- Collecting negative and positive images and pre-processing
- Creating background description file
- Creating training samples and annotations file
- Generating feature vector file
- Cascade training using opencv_traincascade utiltity

We installed OpenCV 3.3.1 in our Windows system. First, we created a folder called workspace, to keep all images and training files.

mkdir opencv_workspace

cd opencv_workspace

For the training of the LBP classifier, we need positive and negative images. Positive images are images of vehicles and negative images are images without any

vehicles. We gathered negative images from image-net.org's APIs using the following code:

```python
import numpy as np
import os

def store_raw_images():
    neg_image_link='//image-net.org/api/text/im
    agenet.synset.geturls?wnid=n00523513'
    neg_image_urls = urllib.request.urlopen(neg_images_link).read().decode()
    pic_num = 1

    if not os.path.exists('neg'):
        os.makedirs('neg')

    for i in neg_image_urls.split('\n'):
        try:
            print(i)
            urllib.request.urlretrieve(i, "neg/"+str(pic_num)+".jpg")
            img = cv2.imread("neg/"+str(pic_num)+".jpg",cv2.IMREAD_GRAYSCALE)
            resized_image = cv2.resize(img, (100, 100))
            cv2.imwrite("neg/"+str(pic_num)+".jpg",resized_image)
            pic_num += 1

        except Exception as e:
            print(str(e))
```

This script will visit the links, grab the URLs, and proceed to the images at the URLs. From here, we grab the images, convert to grayscale, resize to 100X100, and then save it in the folder called "neg". We executed the same code for gathering different negative images from different URLs.

Figure 7.1 "neg" folder with all negative images

To delete all the ugly negative images (blank images downloaded from the link), we first copied one of the downloaded blank images into a folder called uglies and then we used the following code to delete them. This function runs through all the images in the "neg" folder and deletes all images that match the blank image in uglies folder.

```python
def find_uglies():
    match = False
    for file_type in ['neg']:
        for img in os.listdir(file_type):
            for ugly in os.listdir('uglies'):
                try:
                    current_image_path = str(file_type)+'/'+str(img)
```

```python
            ugly = cv2.imread('uglies/'+str(ugly))
            question = cv2.imread(current_image_path)
            if ugly.shape == question.shape and
            not(np.bitwise_xor(ugly,question).any()):
                print('Deleting!')
                print(current_image_path)
                os.remove(current_image_path)
        except Exception as e:
            print(str(e))
```

This left us with a total of 6340 negative images. In the next step, we created the descriptor file for all the negative images. To create this descriptor file, we used the following function:

```python
def create_neg():
    for file_type in ['neg']:
        for img in os.listdir(file_type):
            line = file_type+'/'+img+'\n'
            with open('bg.txt','a') as f:
                f.write(line)
```

After running this code, we got "bg.txt" file, which is the descriptor file that contains the path to all the negative images. The figure given below shows a part of bg.txt file.
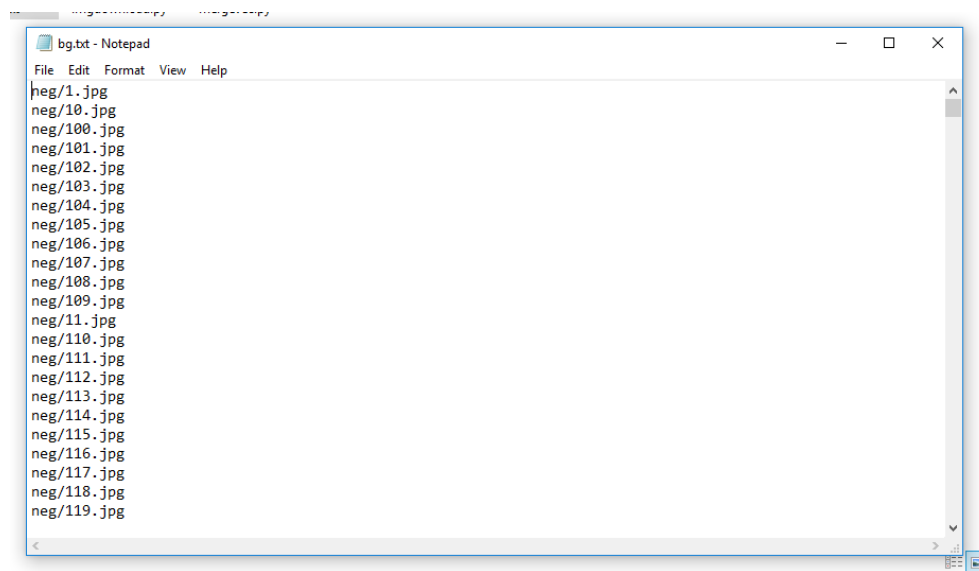
Figure 7.2 bg.txt

We created a directory called "pos" to store the positive images. Every positive image is cropped to include only the vehicle. A positive image is shown below.
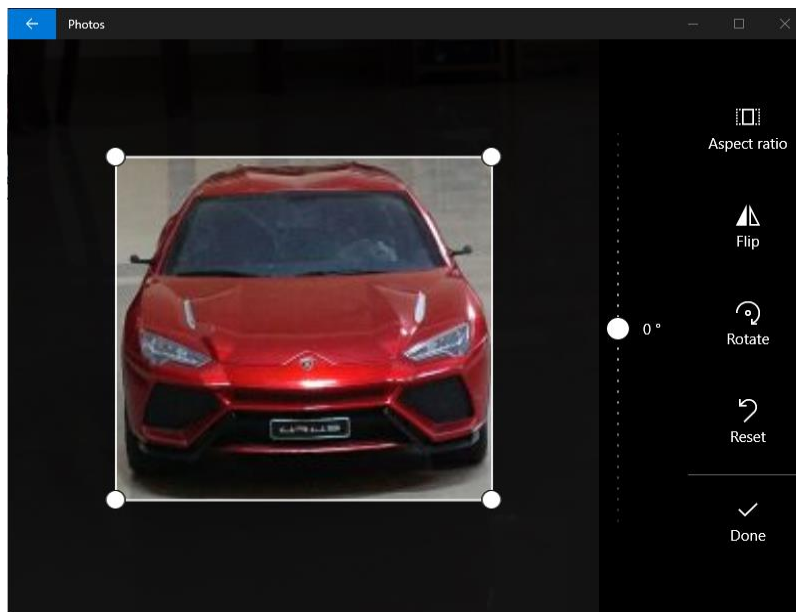
Figure 7.3 Cropping a positive image

We have collected 5 positive images, converted to grayscale and resized to 70x70.

Figure 7.4 Positive images

We created the training samples from the positive images by superimposing them on the negative images at different angles and positions using the **opencv_createsamples** utility. Samples created using the first positive image are stored in the directory info/info1.

opencv_createsamples -img vehicle1.jpg -bg bg.txt -info info/info1/info.lst -pngoutput info/info1 -maxxangle 0.5 -maxyangle 0.5 -maxzangle 0.5 -num 6300

The parameters are

- img, which specifies the positive image from which to create samples i.e. vehicle1.jpg
- bg, which specifies the negative descriptor file
- info, which specifies the location of the positive annotation file generated by the command
- pngoutput, which is where we want to place the newly generated samples
- maxxangle, maxyangle and maxzangle specify the maximum angles by which the positive image may be rotated while superimposing
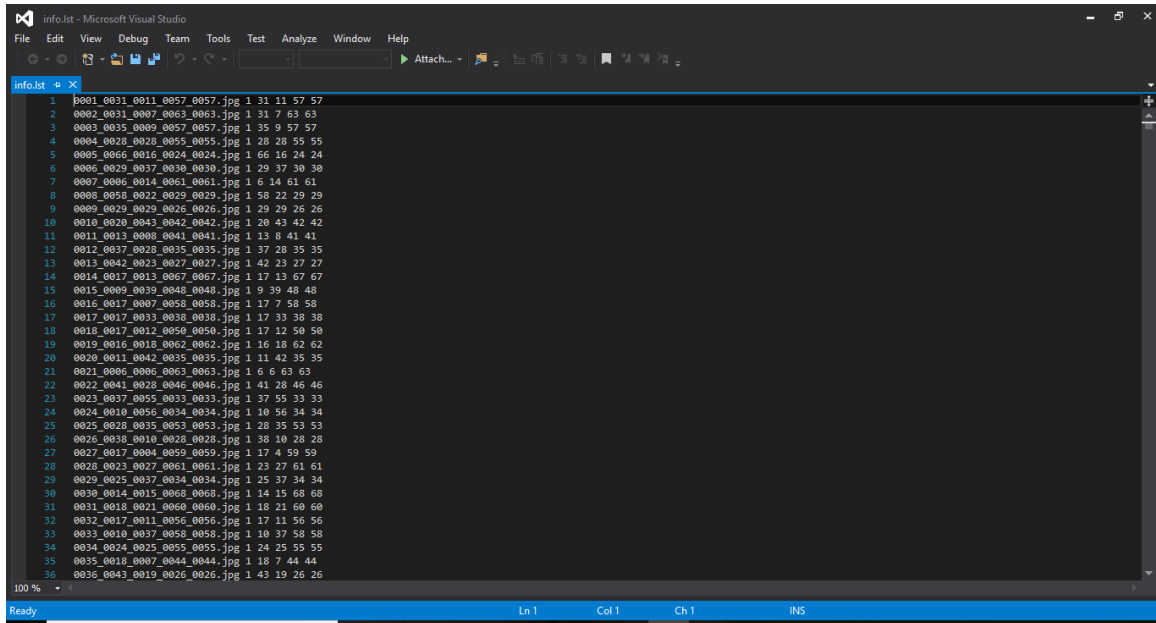- num is the number of samples which will be created.

Figure 7.5 Sample creation command

Figure 7.6 Samples and annotation file generated

Similarly, we executed this command five times for the five positive images and got five info directories, info1 through info5.

Figure 7.7 info.lst contents

Now you should have ~6,000 images in each sample directory, and a file called info.lst. This file is our positive annotation file. Consider a line from the file:

0001_0014_0045_0028_0028.jpg 1 14 45 28 28

First, we have the sample file name, then the number of positive objects in the image, followed by the bounding box position i.e. the (x,y) coordinates of the bottom left corner width, and height of the bounding box.

At this stage our file system looks like this:

opencv_workspace

   --neg

      ---negimages.jpg

   --info

      ---info1

      ---info2

      ---info3

      ---info4

      ---info5

   --pos

      ---posimage.jpg

   --bg.txt

Then, we created a vector file of all features of the positives. For this, we used the following command.

opencv_createsamples -info info/info1/info.lst -num 6300 -w 70 -h 70 -vec positives1.vec

The parameters are:

- info, which is path to the annotation file of the samples created from the first positive image
- num, which is the number of samples
- w and h specify the model dimensions
- vec, which is the path to the vector file to be generated

```
C:\Users\Rithika Chowta\Desktop\NOTES\Final yr project\finalnewcars>opencv_createsamples -info info/info1/info.lst
6300 -w 70 -h 70 -vec vecs/positives1.vec
Info file name: info/info1/info.lst
Img file name: (NULL)
Vec file name: vecs/positives1.vec
BG  file name: (NULL)
Num: 6300
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 1.1
Max y angle: 1.1
Max z angle: 0.5
Show samples: FALSE
Width: 70
Height: 70
Max Scale: -1
Create training samples from images collection...
Done. Created 6300 samples
```

Figure 7.8 Creating vector file from samples in info1

We repeated the command for the samples created from the other positive images as well i.e. info2 through info5. This results in the creation of 5 vector files. Now these files must be merged to a single vector file. This was done using a python program available on GitHub. The final vector file was **finalpositive.vec**

At this stage our workspace looked like this:

opencv_workspace

   --neg

      ---negimages.jpg

   --info

      ---info1

      ---info2

      ---info3

      ---info4

      ---info5

--pos

   ---posimage.jpg

--bg.txt

--vecs

   ---positives1.vec

   ---positives2.vec

   ---positives3.vec

   ---positives4.vec

   ---positives5.vec

 --finalpos.vec

Then we started cascade training using the following command:

opencv_traincascade -data data -vec finalpositive.vec -bg bg.txt -numPos 4000 -numNeg 6300 -numStages 16 -w 70 -h 70 -featureType LBP
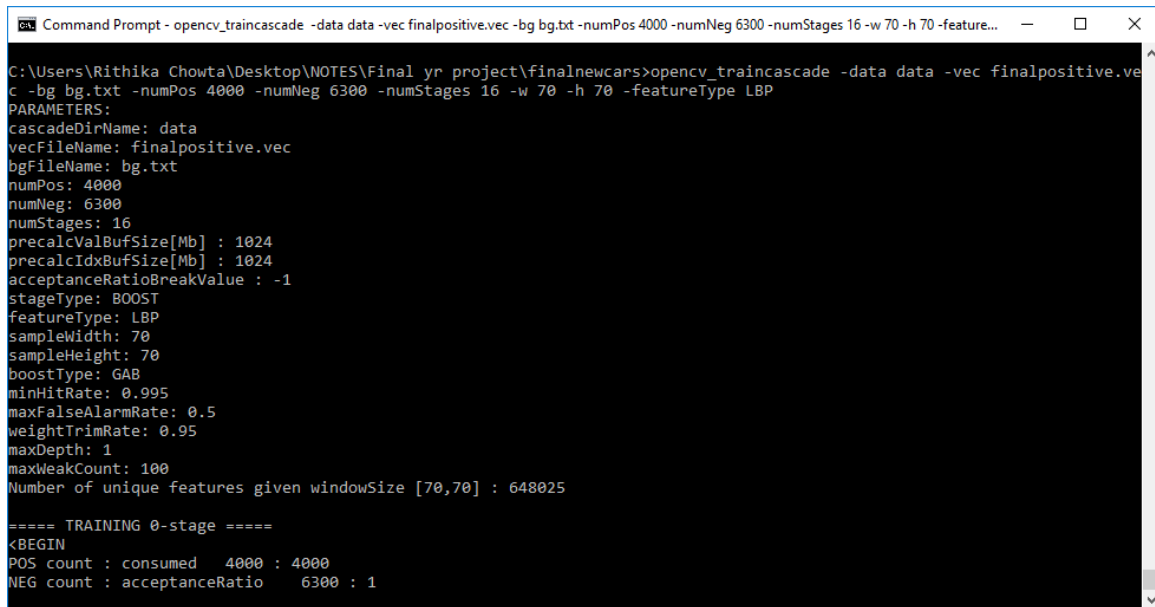
The parameters are:

- data, where each classifier file will be stored. This file must be manually created as it won't be created by the command

- vec, which specifies the path to the positives vector file

- bg, which is the path to the background description file

- numPos, which specifies the number of positive samples used in training for every classifier stage.

- numNeg, which specifies the number of negative samples used in training for every classifier stage.

- numStages, which specifies the number of cascade stages to be trained. As this value is increased, level of accuracy increases. Care should be taken to avoid overtraining.

Taking a 1:2 ratio of numPos:numNeg yielded good results

Figure 7.9 Training began

The acceptanceRatio value is an indicator of the accuracy of the detector. Ideally, it should be in the range 0.0001-0.0004. Finally, we got the classifier stages and final cascade as xml files, describing the features of the cars, in the data folder. We trained up to 12 stages and attained an acceptanceRatio of 0.0003. Similarly, we repeated the entire process to get the cascade classifier files for trucks as well as for bikes.

7.2 DETECTING THE VEHICLES

The processing of vehicle detection is done on the raspberry Pi. We first installed OpenCV 3.3.0 on the Raspberry Pi and we have written a python code to detect vehicles using our cascade classifier.

We first import the cv2 module which has all the OpenCV functions. We use the function, **VideoCapture([cameraIndex])**, which takes in the video stream from the camera, having index cameraIndex, and store it in a VideoCapture object.

To load the cascade classifier file, we use **CascadeClassifier([filename])**. This function takes the xml file as the argument and returns a CasacdeClassifier object. Since we have made three classifiers, describing the features of the cars, bikes and trucks, we have used the CascadeClassifier function thrice, for each type of vehicle separately.

Now we enter the main loop of the algorithm. We have used **read()** function on the 2 VideoCapture objects of each camera, cap1 and cap2, to grab the next frame from the video stream and we store the returned frames in frame1 and frame2 respectively.

cvtColor([frame],COLOR_BGR2GRAY) converts the frame given in the parameter to grayscale for the faster processing.

The key function is **detectMultiScale3(),** which is used to detect objects of different sizes from the input image. The detected objects are returned as a vector of rectangles, where each rectangle is the bounding box of the detected object. It's syntax is as given below,

CascadeClassifierObject.detectMultiScale3(image[,scaleFactor[, minNeighbors[, flags[, minSize[, maxSize[, outputRejectLevels]]]]]])

scaleFactor - This specifies how much the image size is reduced at each image scale, which helps to decide what size of objects to detect.

minNeighbors - Parameter specifying how many neighbors each candidate rectangle should have to retain it. It affects the quality of the detected objects; higher value results in less detections but with higher quality.

minSize - Minimum possible object size. Objects smaller than that are ignored.

maxSize - Maximum possible object size. Objects larger than that are ignored.

outputRejectLevels - Boolean value. If True, it returns the rejectLevels and levelWeights. Default value is False. We set this value to True since the levelWeights, which is a measure of the confidence of the detection, can be used to filter out false positives.

We have used this function to detect cars, bikes and truck using the frames captured from both the cameras.

For each object in the vectors returned, we first check the levelWeights of the detection. If it is above the **levelWeights** observed for true positives, we consider it as an accurate detection and we draw the bounding box on the current frame using the **rectangle()** which takes the frame and bounding box coordinates as arguments. After drawing, the frames are displayed using **imshow().**

We use the flags f1 and f2 to check if detections were made in both camera frames at the same time. Initially, both are set to false. If both become true, then the beep() function is called. beep() is described in the next section. After the beep() is executed, both flags are reset to false.

The while loop keeps continuously processing frames until Esc is pressed.

7.3 GPIO ALARM MODULE

The alarm operations are defined in the beep() function. The global variable last_beep, which is set to 0 at the beginning of the code, is used to save the time when the most recent beep occurred.

First, we use an if condition to check if there is a gap of at least 5 seconds between consecutive beeps. We do this so that there aren't repeated beeps for the same pair of vehicles. If this condition is satisfied, we activate the alarms.

setMode() is used to choose the numbering system of the GPIO board.

setup(pin_number, OUT) is used to setup the pins as either input or output pins. We have set the positive pins to two buzzers and four LEDs as output pins.

output(pin_number, HIGH) is used to enable or disable the output to the output pins as required.

When the vehicles are detected, output is enabled for 5 seconds for two red LEDs and two buzzers. After 5 seconds, on the right side, red LED is disabled and green LED is enabled. After 5 more seconds, on the left side, red LED is disabled and the green LED is enabled. Along with that the right side red LED is enabled, to prevent collisions with vehicles behind the vehicle detected on right side. After 5 seconds, all the LEDs are disabled. The delays are done using the **sleep(sec)**. The last beep time is updated to the current time.

Final OpenCV code:

```python
from cv2 import *
from RPi.GPIO import *
from time import *

f1, f2, last_beep = False, False, 0

def beep():
    global last_beep
    if int(time()-last_beep) > 5:
        setwarnings(False)
        setup(11, OUT)
        setup(13, OUT)
        setup(16, OUT)
        setup(12, OUT)
        setup(38, OUT)
        setup(40, OUT)
        print("Switching LEDs on")
        print("Switching buzzers on for 5 seconds")
        output(16, HIGH)
        output(12, HIGH)
        output(11, HIGH)
        output(13, HIGH)
        sleep(5)
        output(11, LOW)
        output(13, LOW)
        print("Turning right red off and right green on")
```

```
        output(16, LOW)
        output(40, HIGH)
        sleep(5)
        print("Turning left red off and left green on")
        output(12, LOW)
        output(40, LOW)
        output(16, HIGH)
        output(38, HIGH)
        sleep(5)
        print("Turning all off")
        output(38, LOW)
        output(40, LOW)
        last_beep = time()
        cleanup()


cap1 = VideoCapture(0)
cap2 = VideoCapture(1)


car_cascade = CascadeClassifier('cars.xml')
truck_cascade = CascadeClassifier('trucks.xml')
bike_cascade = CascadeClassifier('bikes.xml')
while True:
    ret1, frame1 = cap1.read()
    ret2, frame2 = cap2.read()

    gray1 = cvtColor(frame1, COLOR_BGR2GRAY)
    gray2 = cvtColor(frame2, COLOR_BGR2GRAY)
```

```
    cars1, rejectlevels1, weights1 =  car_cascade.detectMultiScale3(gray1,
scaleFactor=1.1, minNeighbors=70, minSize=(90,90), maxSize=(270,270),
outputRejectLevels=True)
    cars2, rejectlevels2, weights2= car_cascade.detectMultiScale3(gray2,
scaleFactor=1.2, minNeighbors=70, minSize=(120,120), maxSize=(300,300),
outputRejectLevels=True)
    trucks1, rejectlevels3, weights3=truck_cascade.detectMultiScale3(gray1,
scaleFactor=1.5, minNeighbors=700,minSize=(90,110), maxSize=(320,360),
outputRejectLevels=True)
    trucks2, rejectlevels4, weights4= truck_cascade.detectMultiScale3(gray2,
scaleFactor=1.2, minNeighbors=760, minSize=(90,110), maxSize=(320,360),
outputRejectLevels=True)


    bike1, rejectlevels5, weights5 = bike_cascade.detectMultiScale3(gray1,
scaleFactor=1.7, minNeighbors=790, minSize=(90,120), maxSize=(200,270),
outputRejectLevels=True)
    bike2, rejectlevels6, weights6= bike_cascade.detectMultiScale3(gray2,
scaleFactor=1.5, minNeighbors=740, minSize=(90,120), maxSize=(200,270),
outputRejectLevels=True)


    if cars1 is not () or trucks1 is not () or bike1 is not () :
        print("Camera 1",str(cars1),str(weights1))
        for i in range(len(cars1)):
            if weights1[i][0] > 1.937:
                x,y,w,h = cars1[i][0], cars1[i][1], cars1[i][2], cars1[i][3]
                rectangle(frame1, (x, y), (x+w, y+h), (0, 255, 255), 3)
```

```
            f1 = True


    for i in range(len(trucks1)):
        if weights3[i][0] > 3.0:
            x,y,w,h = trucks1[i][0], trucks1[i][1], trucks1[i][2], trucks1[i][3]
            rectangle(frame1, (x, y), (x+w, y+h), (0, 255, 255), 3)
            f1 = True


    for i in range(len(bike1)):
        if weights5[i][0] > 3.2:
            x,y,w,h = bike1[i][0], bike1[i][1], bike1[i][2], bike1[i][3]
            rectangle(frame1, (x, y), (x+w, y+h), (0, 255, 255), 3)
            f1 = True



if cars2 is not () or trucks2 is not () or bike2 is not ():
    print("Camera 2",str(cars2),str(weights2))
    for i in range(len(cars2)):
        if weights2[i][0] > 1.937:
            x,y,w,h = cars2[i][0], cars2[i][1], cars2[i][2], cars2[i][3]
            rectangle(frame2, (x, y), (x+w, y+h), (255, 255, 0), 3)
            f2 = True


    for i in range(len(trucks2)):
        if weights4[i][0] > 3.0:
            x,y,w,h = trucks2[i][0], trucks2[i][1], trucks2[i][2], trucks2[i][3]
            rectangle(frame2, (x, y), (x+w, y+h), (0, 255, 255), 3)
```

```
                    f2 = True


        for i in range(len(bike2)):
              if weights6[i][0] > 3.2:
                      x,y,w,h = bike2[i][0], bike2[i][1], bike2[i][2], bike2[i][3]
                      rectangle(frame2, (x, y), (x+w, y+h), (0, 255, 255), 3)
                      f2= True


    print("updating frames")
    namedWindow('Camera 1 - Quantum', WINDOW_NORMAL)
    resizeWindow('Camera 1 - Quantum', 25, 25)
    imshow('Camera 1 - Quantum', frame2)


    namedWindow('Camera 2 - Itek', WINDOW_NORMAL)
    resizeWindow('Camera 2 - Itek', 25, 25)
    imshow('Camera 2 - Itek', frame1)


    if f1 and f2:            # If car detected in both frames
        print("Calling beep")
        beep()
        f1, f2 = False, False    # Resetting flags


    if waitKey(33) == 27:   # Wait for Esc key to stop
        break

destroyAllWindows()   # Closes all windows
```

CHAPTER 8

RESULTS AND DISCUSSION

8.1 Results

The display of the output in our system is done on the monitor and by switching on the Buzzers and LEDs which are all connected to the raspberry pi.

After running the OpenCV code, 2 display windows are shown on the monitor, one for each camera. The road and the vehicles passing through these roads are displayed on both the windows. The rectangular boxes are drawn around the vehicles which got detected.
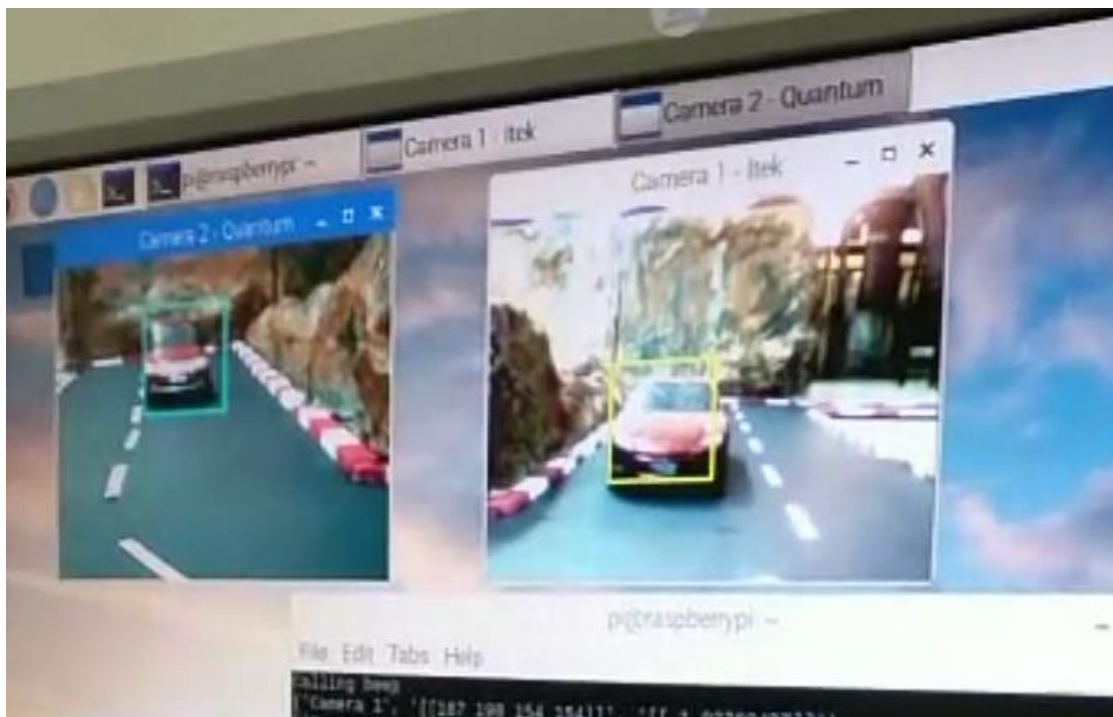
Figure 8.1 Output from the monitor when both the cars got detected

When two vehicles coming from two sides of the blind curve get detected at the same time, the red LEDs for both the sides of the curve switch on.

Figure 8.2 Both the red LEDs switched on upon detection

When both red LEDs are on, both the vehicles slow down and stop, to prevent collision.

Figure 8.3 Both the vehicles stop when both red LEDs are on

After a few seconds, the red LED from the right side of the curve is turned off and the green LED is turned on from the same side. This green LED will indicate the

vehicle coming from the right side of the curve to slowly start going across the curve.

Figure 8.4 Right side green LED on and left red LED on

After few more seconds, that is after the vehicle which was coming from the right side of the curve, passed through the left of the curve, the red LED from the left side of the curve turns off and the green LED turns on, indicating to the vehicle on the left side to start moving. Meanwhile, the right red LED is switched on to prevent collisions with any vehicles which were behind the vehicle which moved first.

Figure 8.5 Left green LED is on and right red LED is on

After both the vehicles go away from the curve, all the LEDs are turned off.

Figure 8.6 LEDs on the both of the sides of the curve are off

8.2 Discussion

LBP cascades had higher hit rate as compared other methods of vehicle detection such as HOG. LBP cascades also had better performance for live video processing on the Pi as compared to Tensor Flow.

We also use the **opencv_visualize** utility to check the features which were detected by the trainer for a test image.

opencv_visualisation --image=testimage.png --model=data/cascade.xml -- data=data/visualization_result/

This command creates an output video of each stage of the classifier showing the detected features for the test image given in the image parameter. It also creates images showing features detected in each stage.

Figure 8.7 LBP features detected for a test image in stage 2 of the cascade classifier

Figure 8.8 LBP features detected for a test image in stage 9 of the cascade classifier

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 CONCLUSION

Our system will alert the drivers going around the blind curve about the presence of the oncoming vehicle. The cameras which are placed in front of the roads will detect the vehicles and then alert the drivers through buzzers and LEDs which are placed on both the sides of the blind curve using a pole. The red and green LED will resolve confusions among the drivers and will ensure a smooth and safe ride, preventing from accidents and deaths.

9.2 FUTURE WORK

The shape of the detection zones can be programmed for specific applications. The system can be adapted to other areas like junctions. We can implement the same project using better cameras to see through rain, fog and mist clearly. In the real-time system, motion sensors and proximity sensors can be integrated with this system to increase accuracy.

REFERENCES

[1] **Chitransh Srivastava, Nikhil Acharya, Fervez Jaffer B.M., Sachin Bhat** (2016) "Implementation of collision avoidance system for hairpin bends in ghats using proximity sensors". *International Journal of Current Engineering and Scientific Research (IJCESR),* Volume 3, Issue 11.

[2] **Jessen Joseph Leo, R.Monisha, B.T.Tharani Sri Sakthi, A. John Clement Sunder** (2014) "Vehicle movement control and accident avoidance in hilly track" *Proceedings of the International Conference on Electronics and Communication System (JCECS),*

[3] **R.S. Rakul, S. Ravia, K. N. Thirukkuralkani** (2016) "Implementation of Vehicle Mishap Averting System using Arduino Microcontroller". *International Journal of Engineering Research & Technology (IJERT),* Volume 5 Issue 4

[4] **C.T. Chen, Y.S. Chen** (2009) "Real-time approaching vehicle detection in blind-spot area". *Proceedings of the 12th International IEEE Conference on Intelligent Transportation Systems*, St. Louis, MO, USA