Project Report

on

"DENSITY BASED TRAFFIC SURVEILLANCE"

Submitted in partial fulfillment of the

T.Y.B.Tech.

in

INFORMATION TECHNOLOGY

Submitted by

Shreya S. Kapgate (16141125)

Akshay V. Jadhav (16141207)

Under the Guidance of

Prof. R.S. Mawale

Government College Of Engineering, Karad

(An Autonomous Institute of Government of Maharashtra)

Academic Year 2018-2019

Government College Of Engineering, Karad

(An Autonomous Institute of Government of Maharashtra)

Department Of Information Technology

CERTIFICATE

This is to certify that the project entitled **"DENSITY BASED TRAFFIC SURVEILLANCE"** has been carried out by team: Shreya S. Kapgate (16141125)
Akshay V. Jadhav (16141207)

of T.Y.B.Tech. IT class under the guidance of Prof. R.S. Mawale during the academic year 2018-19 (Sem-VI).

Prof. R. S. Mawale	**Dr. S. J. Wagh**
Project Guide	**Head**
	Department Of Information Technology

External Examiner

ACKNOWLEDGEMENT

Apart from individual efforts, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental throughout the project work.

It is our privilege to express our sincere gratitude towards our project guide, Prof. R.S. Mawale, for their valuable guidance, encouragement, inspiration and whole-hearted cooperation throughout the project work. We thank him for being a motivation through all our highs and importantly, our lows.

We deeply express our sincere thanks to our Head of Department Dr. S.J. Wagh for encouraging and allowing us to present the project on the topic "DENSITY BASED TRAFFIC SURVEILLANCE" and providing us with the necessary facilities to enable us to fulfill our project requirements as best as possible. We take this opportunity to thank all faculty members and staff of Department of Information Technology, who have directly or indirectly helped our project.

We pay our respects to honorable Principal Dr. A.T. Pise for their encouragement. Our thanks and appreciations also go to our family and friends, who have been a source of encouragement and inspiration throughout the duration of the project.

I. ABSTRACT:

The existing Traffic control system is based on the “time” which is already assigned in the system. According to these times the signals are working in each lane. But in these system condition is occurs as all vehicles in lane(L1) are passed and vehicles in another lane (L2) still in waiting state because time is not over and hence signal is still red. These systems are very inefficient because they are unable to handle various simple situations which are occurs throughout the day. Major drawback is it has unnecessary waiting time and there is no facility to handle emergency vehicles. The project is designed to develop a system which perform execution based on density of vehicles (Vehicle Count). After calculating the number of vehicles we will came to know in which side the density is high based on which signals will be allotted for a particular side. Raspberry pi is used as a microcontroller which provides the signal timing based on the traffic density. And can provide facility to handle emergency vehicles automatically and efficiently.

II. LIST OF FIGURES :

FIGURE NO.	DESCRPTION THE FIGURE	PAGE NO
2.1.1	Existing System	5
4.2.1.1	Use case diagram for Camera	11
4.2.1.2	Use case diagram for Raspberry Pi	11
4.2.1.3	Global view	11
4.2.2.1	Activity Diagram	12
4.2.3.1	Sequence diagram	13
4.2.4.1	Collaburation diagram	13
5.1.1	Raspberry Pi	15
5.1.2	LED	16
6.2.1	Circuit Diagram	23
6.1.1	Project Setup	25
6.2.2	Object detection	25

Table of Contents

Topic **Pg. No**

- Abstract i
- List of Figures i

Chapter 1 **1**
Introduction..
1.1 Background.. 2
1.2 Motivation... 2
1.3 Importance.. 2

Chapter 2 **3**
Literature Survey..
2.1 Introduction…………………………………………………….... 4
2.2 Existing System………………………………………………. 4
2.3 Proposed Architecture……………………………………. 6

Chapter 3 **8**
Module Description…………………......................................
3.1 Related Theory………………………………………………… 9

13

Chapter 4
System Design………...

Chapter 5 **16**
Technical Specifications…………………………………….
5.1 Technical Background………………………………… 17

Chapter 6 **18**
Implementation……...
6.1 Convolutional Neural Network Implementation…………… 19

Chapter 7 **20**
Result……………………...

Chapter 8 **24**
Conclusion and Future Scope………………………………
8.1 Refrences………………………………………………..……… 26

CHAPTER NO. 1

INTRODUCTION

1.1 Background

Traffic congestion is now considered to be one of the biggest problems in the urban environments. Traffic problems will be also much more widely increasing as an expected result of the growing number of transportation means and current low-quality infrastructure of the roads. In addition, many studies and statistics were generated in developing countries that proved that most of the road accidents are because of the very narrow roads and because of the destructive increase in the transportation means. Due to the massive growth in urbanization and traffic congestion, intelligent vision based traffic light controller is needed to reduce the traffic delay and travel time especially in developing countries as the current automatic time based control is not realistic while sensor based traffic light controller is not reliable in developing countries. This idea of controlling the traffic light efficiently in real time has attracted many researchers to work in this field with the goal of creating automatic tool that can estimate the traffic congestion and based on this Variable, the traffic signal can be changed.

1.2 Motivation

The traffic lights that are in widespread use today do not do much intricate reasoning when deciding when to change the lights for the various road users waiting in different lanes. How long the signal stays green in one lane and red in another is most often determined by simple timing that is calculated when the crossing is designed. Even though today's methods are robust and work well when the traffic load is distributed evenly across the lanes in the intersection, the systems are very inefficient because they are unable to handle various simple situations that arise throughout the day. Unnecessary waiting time in the signal can be avoided by determining in which side the green signal should be large during the traffic. In Case the structure of the traffic[1].

1.3 Scope

The present system uses a single camera for monitoring traffic at an intersection. By using a separate camera for each road at an intersection can improve the system efficiency further. The vehicle objects can also be categorized into various classes depending upon the geometrical shape of vehicle for blocking the passage of large vehicles e.g. trucks during day time. The emergency mode can be refined further by installing a GPS receiver in ambulance so that the base station will keep track of the ambulance location on a continuous basis and clear the road whenever will be required.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing System

In existing system the traffic signals are based on fixed time system for each lane. Due to this if there are no vehicle in any lane then also time is allotted to that lane. This increases unnecessary waiting time for the vehicles in other lanes. This system was first developed using sensors, but since sensors have a complicated hardware and implementation, the project was developed using OpenCV, which made the project comparatively easy to implement and understand, also there were changes in the hardware such as the microcontroller used was Raspberry Pi. We can see here green signal is allotted to the lane where there are no vehicle and other lanes where there is heavy traffic density are still in waiting state. So this was the drawback of existing system.

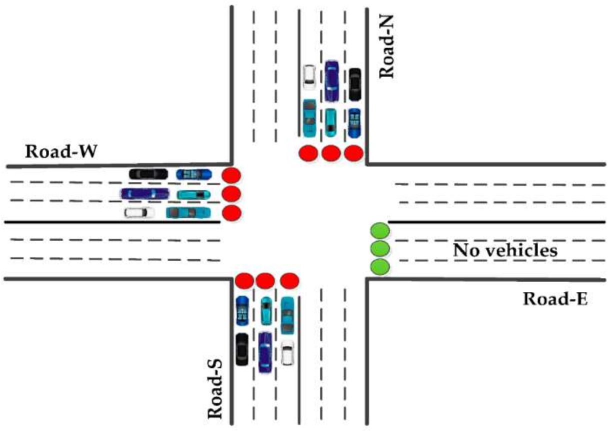


Fig. 2.1.1:Existing System

2.2 Proposed System

In this system we are allotting time based on density of traffic in each lane. If number of vehicles in a lane is more then time allotted will be more. This reduces time and also avoids traffic jams. This system uses OpenCV as a software, and uses the concept of Image Processing. The language that will be used is python. Instead of depending on information generated by costly sensors, economic situation calls for using available video cameras in an efficient way for effective traffic congestion estimation. Speaking about the feasibility, since we are using OpenCV as the software, the entire cost of the project is minimized.

CHAPTER NO. 3

MODULE DESCRIPTION

3.1 Related Theory

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps:

- Importing the image via image acquisition tools;
- Analyzing and manipulating the image;
- Output in which result can be altered image or report that is based on image analysis.

3.2 Step 1: Setting Up the Raspberry Pi

If you have a brand-new Raspberry Pi and are looking for instructions to load the OS into the SD card, use installing the OS for Raspberry Pi. The default OS used with the Pi is Raspbian. Insert memory card into Raspberry Pi and connect it to pc or laptop using LAN[2]. Then open putty and insert IP of Raspberry Pi which was obtained earlier from network and sharing setting. Then enable X11 forwarding option in putty in ssh option. Save and open this session.

A terminal window will appear insert pi as user name and raspberry as password. Once you login type the following command –

```
sudo raspi-config
```

Then select interfacing options and enable SSH and VNC .

Then enter command – vncserver .

After this you will get a IP, note down this IP. The open VNC-Viewer software and connect using this IP. Enter the user name and password. Click ok. A desktop window of Raspbian OS will appear.Install Open-CV libraries[3].

3.3 Step 2 : Image Capturing

The first step towards image processing for smart traffic surveillance is live footage capturing. This can be done by using either USB webcam or picamera. In this project I have used the USB webam.

Steps to Install USB webcam on Raspberry Pi

1. Attach your USB webcam to Raspberry Pi through the USB port on Pi and run the lsusb command in the terminal. This command lists all the USB devices connected to the computer.
2. Install the fswebcam utility by running the following command:
   ```
   sudo apt-get install fswebcam
   ```
   The fswebcam is a simple command-line utility that captures images with webcams for Linux computers.
3. Use following code to get a continue video from USB webcam.
   ```
   cam=cv2.VideoCapture(0)
   ret,frame = cam.read()
     frame = np.array(frame)
   cv2.imshow('frame', frame)
   ```

3.4 Step 3: Processing

The output from camera is processed. First we need to locate calibration points in our model. Here we have considered calibration points as the points as those if crossed by vehicle, that vehicle is not counted. Then we apply Dilate and Erode methods to the stream. Thresholding operation is used to detect the object by considering their RGB values. Findcountours method is applied to the object of threshold. Then count of vehicles is calculated[4][5][6].

3.5 Step 4: Controlling LED’s

Once the count of vehicle in horizontal and vertical lane is obtained, we can use that count to control traffic light. If horizontal lane has more vehicles than vertical lane, the green led of horizontal lane turns on and red led of vertical lane turns on.

CHAPTER NO. 4

DESIGN METHOLOGY

4.2 UML diagram

4.2.1 Use Case Diagram

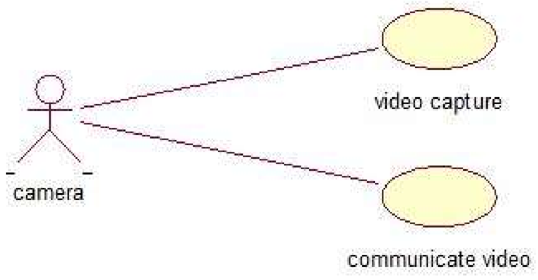


Fig. 4.2.1.1:Use case diagram for Camera

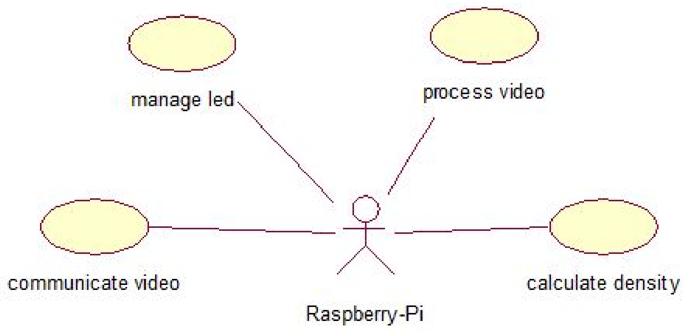


Fig. 4.2.1.2:Use case diagram for Raspberry Pi

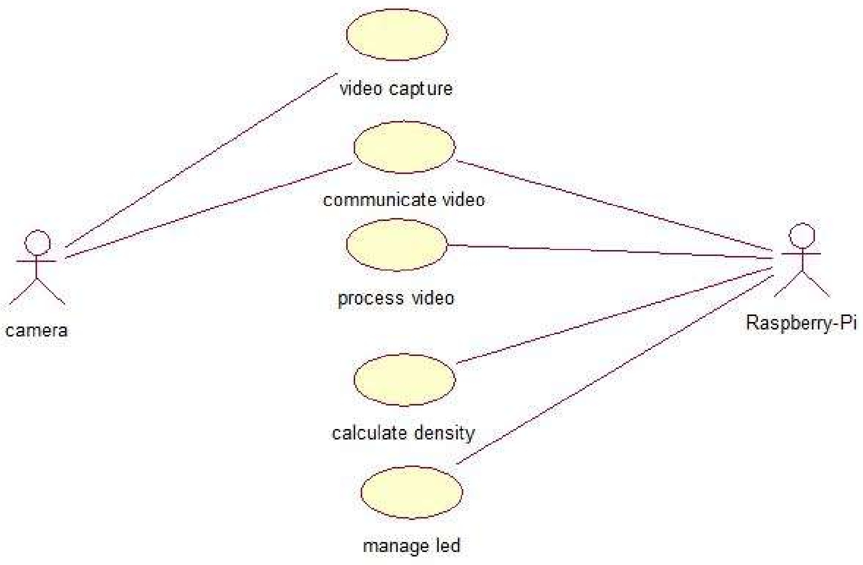


Fig. 4.2.1.3: Global view

4.2.2 Activity diagram:

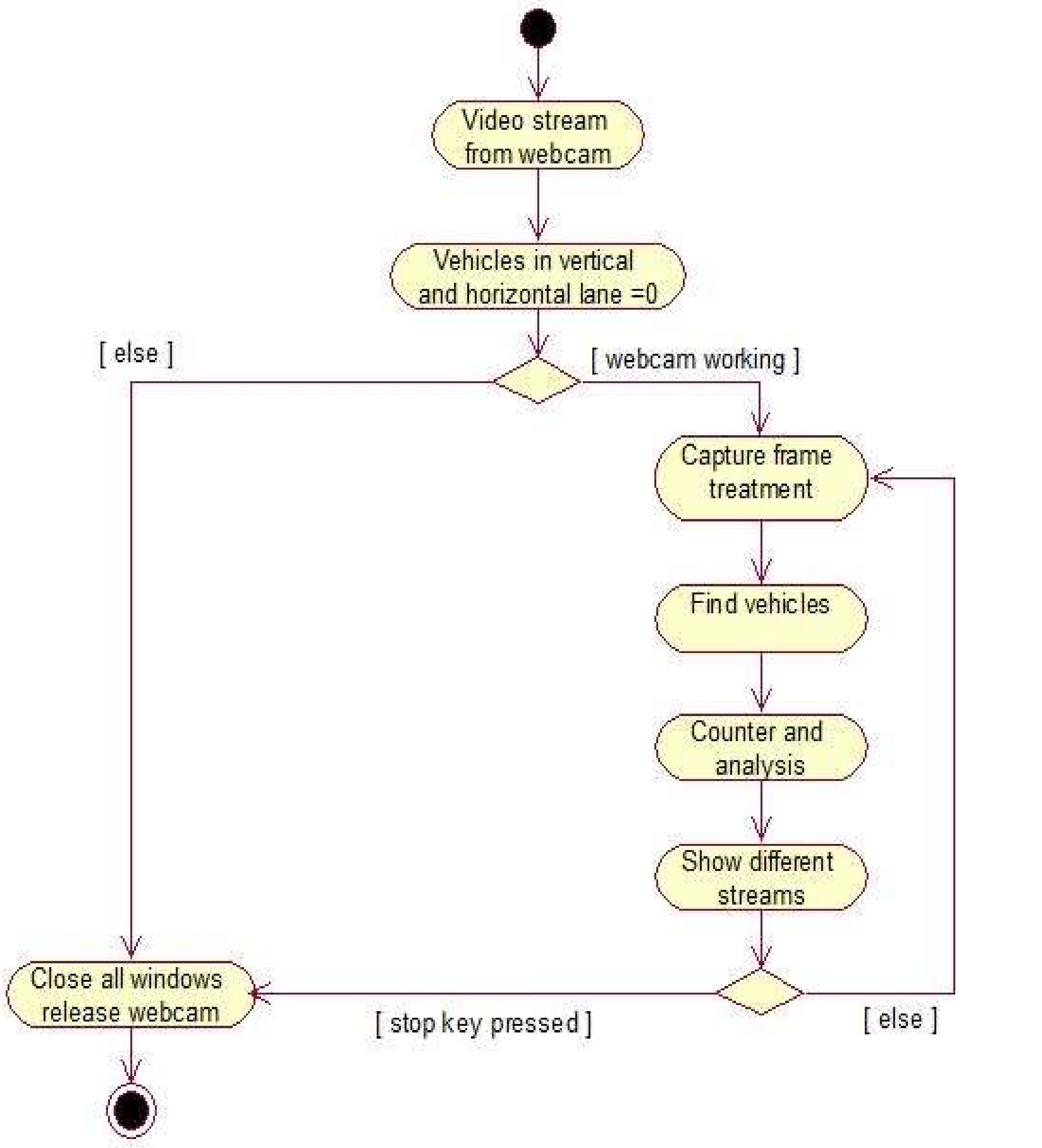


Fig. 4.2.2.1: Activity Diagram

4.2.3 Sequence diagram:

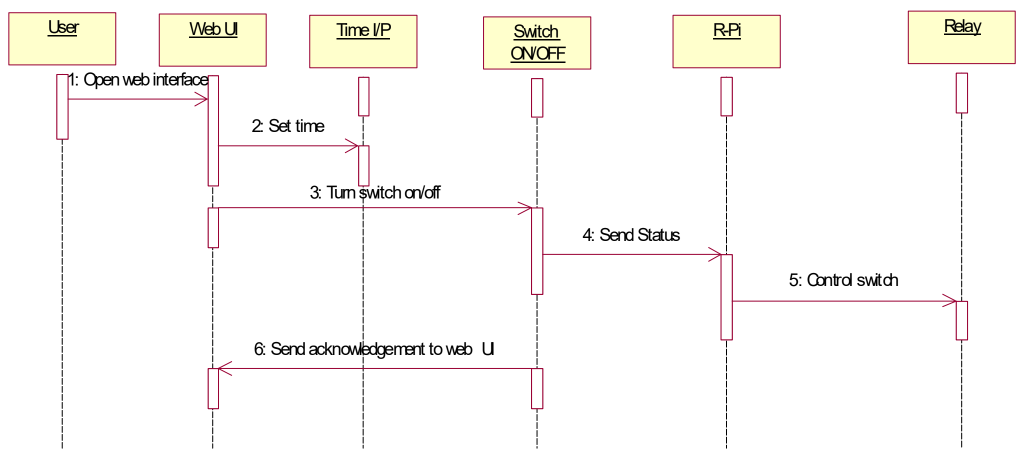


Fig 4.2.3.1: Sequence diagram

CHAPTER NO. 5
TECHNICAL SPECIFICATION

5.1 Hardware specification:

- **Raspberry Pi**: The Raspberry Pi 3 is the third-generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016. It has Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAMBCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board40-pin extended GPIO4 USB 2 ports,4 Pole stereo output and composite video port, Full size HDMICSI, camera port for connecting a Raspberry Pi camera DSI display port for connecting a Raspberry Pi, a touchscreen display Micro SD port for loading your operating system as well as storing data and Upgraded switched Micro USB power source up to 2.5A

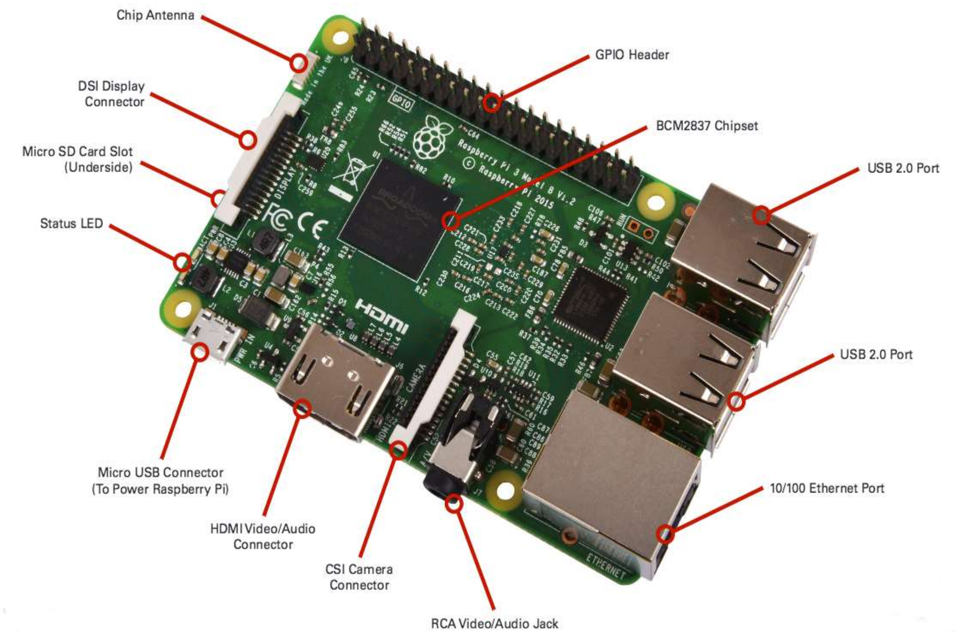


Fig 5.1.1: Raspberry Pi

- Memory Card : 16 GB SanDisk
- Camera : 25 Mega Pixels USB webcam (Quantum – QHM495LM-3937)

- LED lights: A **light-emitting diode** (**LED**) is a two-lead semiconductor light source. It is a p–n junction diode that emits light when activated.

Fig. 5.1.2: LED

- Laptop – Acer A515

5.2 Software specification:

- **Raspbian OS :** Raspbian is a Debian-based computer operating system for Raspberry Pi. There are several versions of Raspbian including Raspbian Stretch and Raspbian Jessie. Since 2015 it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the family of Raspberry Pi single-board computers.
- **SD Card Formatter :** SD Card Formatter is a program that provides quick and easy access to all memory card formats like SD, SDHC and SCXC, and has been designed so that you can get rid of all the content stored on your SD card in one go.

- **PuTTY :** PuTTy is a software terminal emulator for Windows and Linux. It provides a text user interface to remote computers running any of its supported protocols, including SSH and Telnet. Pictured here is an example of a PuTTY SSH session.(Insert Image)
- **Win32 Disk Imager :** This program is designed to write a raw disk image to a removable device or backup a removable device to a raw image file. It is very useful for embedded development, namely Arm development projects (Android, Ubuntu on Arm, etc).
- **VNC Viewer :** , Virtual Network Computing (VNC) is a graphical desktop-sharing system that uses the Remote Frame Buffer protocol (RFB) to remotely control another computer. It transmits the keyboard and mouse events from one computer to another, relaying the graphical-screen updates back in the other direction, over a network.
- **OpenCV library :** OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications.

CHAPTER NO. 6

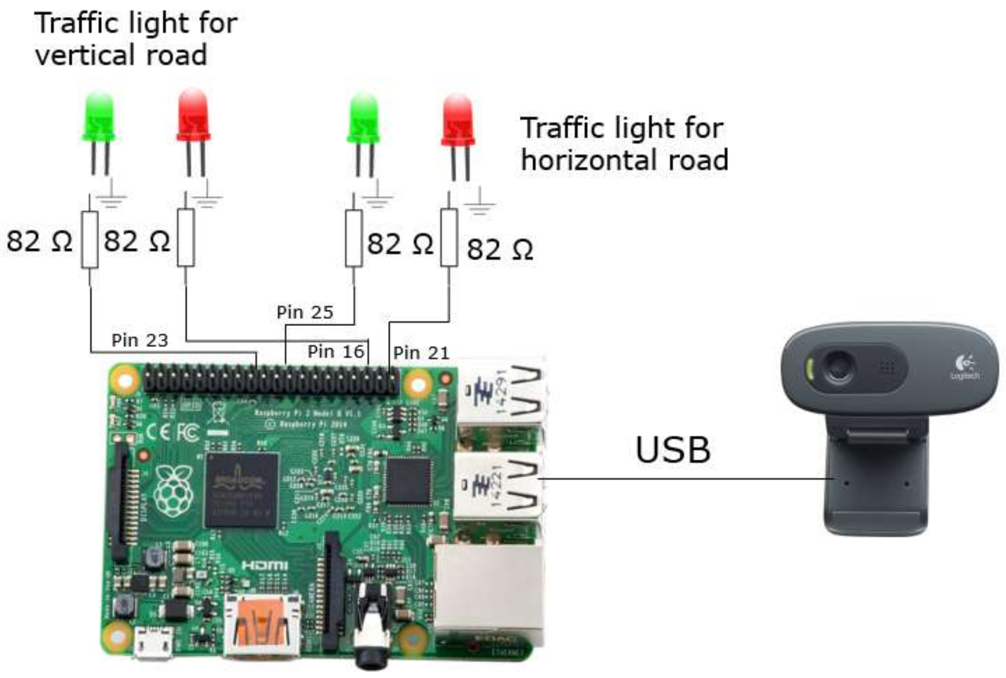
IMPLEMENTATION

6.1 Python Code

```
import numpy as np
import time
import cv2
import RPi.GPIO as GPIO


GPIO.setmode(GPIO.BCM)

for i in (23, 25, 16, 21):
   GPIO.setup(i, GPIO.OUT)


cam=cv2.VideoCapture(0)
cam.set(4,480) #Width=480
cam.set(5,480) #Height=480
cam.set(6,30) #FrameRate = 30

time.sleep(0.1)

colorLower = np.array([0,100,100]) #mid blue
colorUpper = np.array([179,255,255]) #light blue

initvert = 0
inithoriz = 0
counter = 0


xur = 0
yur = 0
xul = 0
yul = 0
xdr = 0
ydr = 0
xdl = 0
ydl = 0

t = 0
t1 = time.time()

while t < 5 and cam.isOpened():
   ret,frame = cam.read()
   frame = np.array(frame) #Transform frame into array
   hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

   mask = cv2.inRange(hsv,colorLower,colorUpper)
   mask = cv2.blur(mask,(3,3))

   mask= cv2.dilate(mask,None,iterations=10)

   mask= cv2.erode(mask,None,iterations=1)

   mask= cv2.dilate(mask,None,iterations=5)

   mask= cv2.erode(mask,None,iterations=1)


   me,thresh = cv2.threshold(mask,127,255,cv2.THRESH_BINARY)

   cnts                                                                        =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)[-2]

   center = None

   print("Centers")

   if len(cnts) > 0:
       for c in cnts:
           (x,y),radius = cv2.minEnclosingCircle(c)
           center = (int(x),int(y))
           print(center)
           radius = int(radius)
           cv2.circle(frame,center,radius,(0,255,0),2)

           x = int(x)
           y = int(y)

           if x > 240: #right
               if y > 240: #up
                   xur = x
                   yur = y

               if y < 240: #down
                   xdr = x
                   ydr = y
           if x < 240: #left
               if y > 240: #up
                   xul = x
                   yul = y

               if y < 240: #down
                   xdl = x
                   ydl = y
    t2 = time.time()
    t = t2-t1


print("upright",xur,yur)
print("downright",xdr,ydr)
print("upleft",xul,yul)
print("downleft",xdl,ydl)
print('\n')
print("Remove calibration objects")

time.sleep(5)


while(cam.isOpened()):
    ret,frame = cam.read()
    frame = np.array(frame) #Transform frame into array
    hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

    mask = cv2.inRange(hsv,colorLower,colorUpper)
    maskhsv = cv2.resize(mask,(250,250))

    mask = cv2.blur(mask,(3,3))
    mask1 = cv2.resize(mask,(250,250))
    #cv2.imshow("mask1",mask)

    mask= cv2.dilate(mask,None,iterations=10)
    mask2=cv2.resize(mask,(250,250))
    #cv2.imshow("mask2",mask)

    mask= cv2.erode(mask,None,iterations=1)
    mask3 = cv2.resize(mask,(250,250))
    #cv2.imshow("mask3",mask)

    mask= cv2.dilate(mask,None,iterations=5)
    mask4=cv2.resize(mask,(250,250))
    #cv2.imshow("mask4",mask)

    imstack = np.hstack((maskhsv,mask1,mask2,mask3,mask4))
    cv2.imshow("masks",imstack)


    me,thresh = cv2.threshold(mask,127,255,cv2.THRESH_BINARY)
    cv2.imshow("thresh",thresh)

    cnts                                                            =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)[-2]
```

```
#print(cnts)

center = None

vert = 0
horiz = 0

if len(cnts) > 0:
    for c in cnts:
        (x,y),radius = cv2.minEnclosingCircle(c)
        center = (int(x),int(y))
        radius = int(radius)

        x = int(x)
        y = int(y)

        if xul < x < xur: #vertical road
            if y > yur:
                vert = vert +1 #up
                cv2.circle(frame,center,radius,(0,255,0),2)
            elif y < ydr:
                vert = vert +1 #down
                cv2.circle(frame,center,radius,(0,255,0),2)
            else:
                vert = vert
        if ydr < y < yur: #horizontal road
            if x > xur:
                horiz = horiz +1 #right
                cv2.circle(frame,center,radius,(0,255,0),2)
            elif x < xul:
                horiz = horiz +1 #left
                cv2.circle(frame,center,radius,(0,255,0),2)
            else:
                horiz = horiz
        if vert != initvert:
            print("Cars in vertical lane: ", str(vert))
            initvert = vert
            print("Cars in horizontal lane: ", str(horiz))
            inithoriz = horiz
            print ("----------------------------")

        if horiz != inithoriz:
            print("Cars in vertical lane: ", str(vert))
            initvert = vert
            print("Cars in horizontal lane: ", str(horiz))
            inithoriz = horiz
            print ("----------------------------")
        if vert < horiz:
            GPIO.output(25,GPIO.HIGH) #Green hor
            GPIO.output(16,GPIO.HIGH) #Red vert
            GPIO.output(21,GPIO.LOW) #Red hor
            GPIO.output(23,GPIO.LOW) #Green vert
        if horiz < vert:
            GPIO.output(21,GPIO.HIGH) #Red hor
            GPIO.output(23,GPIO.HIGH) #Green vert
            GPIO.output(25,GPIO.LOW) #Green hor
            GPIO.output(16,GPIO.LOW) #Red vert

    hsvim = cv2.resize(hsv,(500,500))
    frameim = cv2.resize(frame,(500,500))
    imstack2 = np.hstack((hsvim,frameim))
    cv2.imshow("Frame + hsv",imstack2)


    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cam.release()
cv2.destroyAllWindows()
GPIO.cleanup()
```

6.2 Circuit



Traffic light for vertical road
Traffic light for horizontal road
82 Ω
82 Ω
82 Ω
82 Ω
Pin 25
Pin 23
Pin 16
Pin 21
USB

Fig 6.2.1: Circuit Diagram

CHAPTER 6

RESULTS

6.1 Project Setup

Webcam is facing towards four lane, it is attached to Raspberry-Pi. The following figure shows final hardware setup .

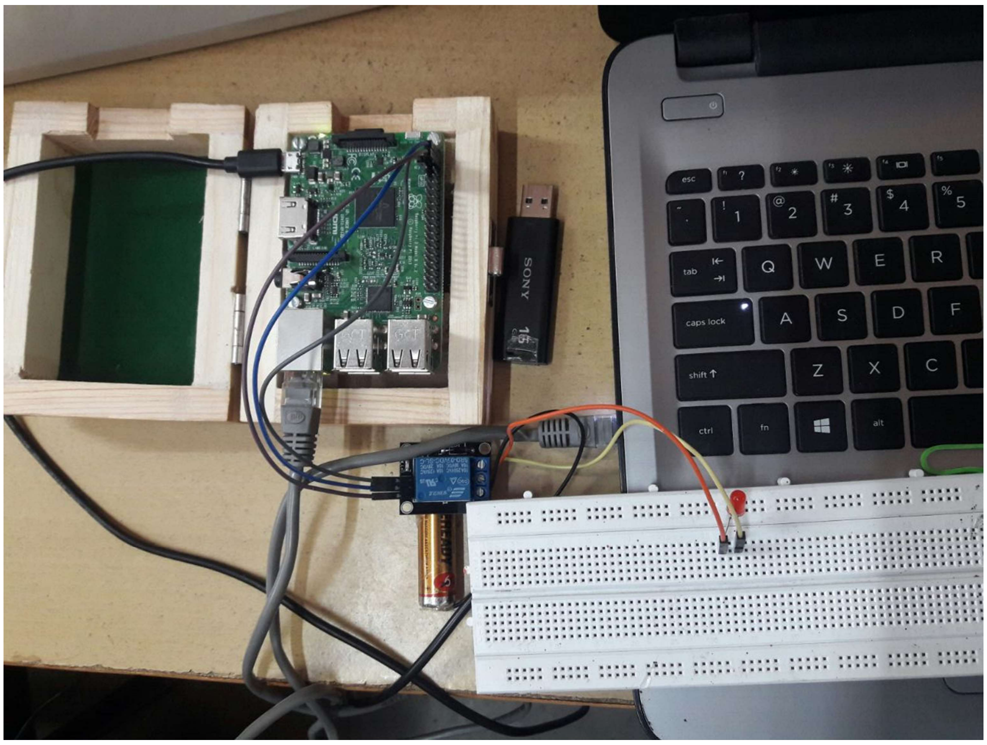


Fig.6.1.1 Project Setup

6.2 Video capturing and detecting objects-

Whenever the program is run video is made and vehicles are detected.

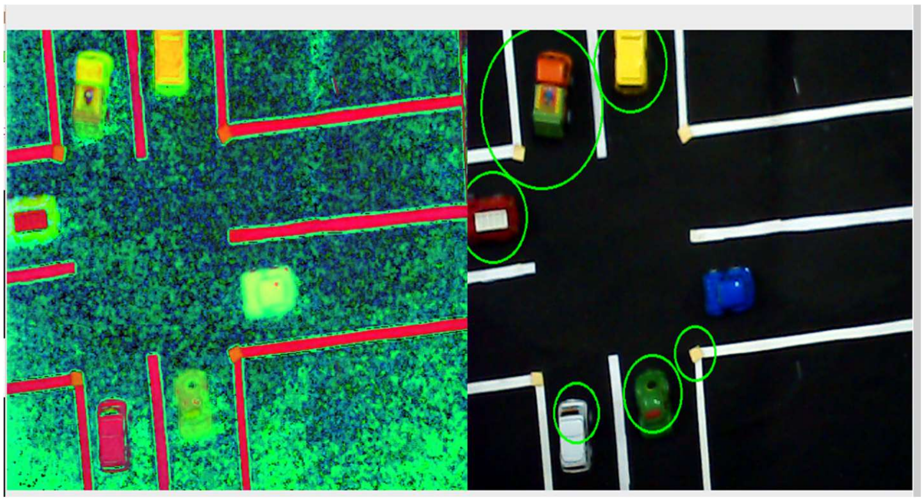


Fig .6.2.2 Object detection

6.3 Web Interface-

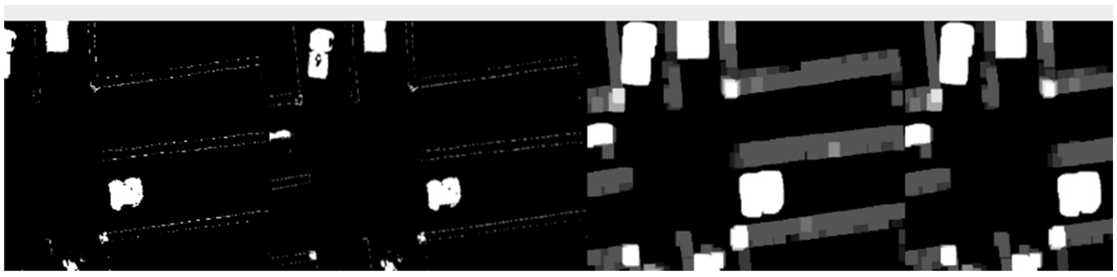


Fig.6.3.3.1: Processing

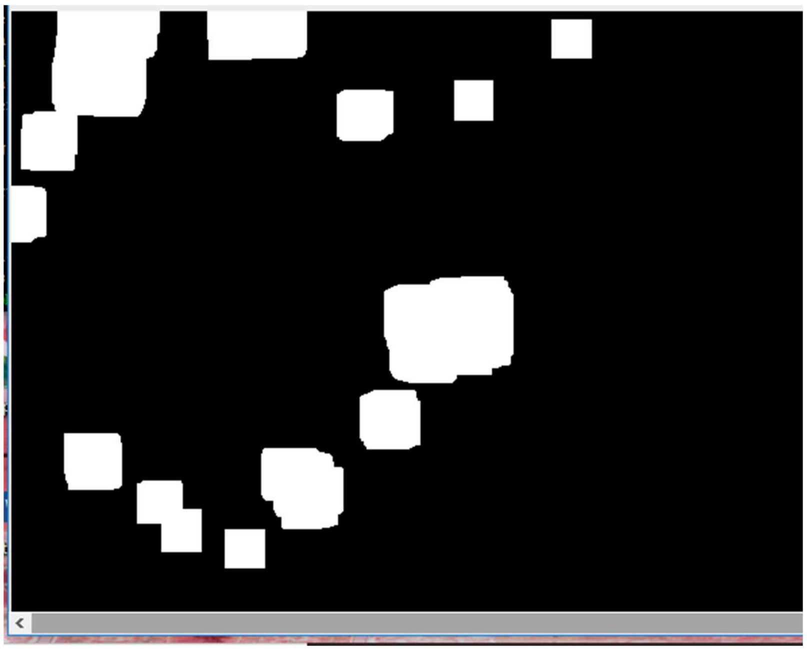


Fig.6.3.3.2: Processing

CHAPTER 7

FUTUTRE SCOPE AND CONCLUSION

7.1 Future Scope

The present system uses a single camera for monitoring traffic at an intersection. By using a separate camera for each road at an intersection can improve the system efficiency further. The vehicle objects can also be categorized into various classes depending upon the geometrical shape of vehicle for blocking the passage of large vehicles e.g. trucks during day time. The emergency mode can be refined further by installing a GPS receiver in ambulance so that the base station will keep track of the ambulance location on a continuous basis and clear the road whenever will be required.

7.2 Conclusion

In this project, a method for estimating the traffic using OpenCV is presented. This is done by using the camera images captured from the road lanes. Each image is processed separately and the number of cars has been counted. This system guarantees that the average waiting time of the vehicle in front of traffic signal will be lesser than present traffic control systems, also the techniques and algorithms used in this project promises to be more effective as compared to the previous system. The advantages of this new method include such benefits as use of OpenCV over sensors, low cost, easy setup and relatively good accuracy and speed. Because this method has been implemented using OpenCV library, production costs are low while achieving high speed and accuracy. The implementation of this system is not possible in rural areas since there are less number of vehicles present due to which there are no traffic issues encountered.

References-

[1] Density Based Traffic Signal System K.Vidhya, A.Bazila Banu Post Graduate Student Dept of Information Technology, Velammal College of Engineering and Technology, Madurai, India

[2] https://www.youtube.com/watch?v=jsi50bCo_W4

[3] https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3- python-on-your-raspberry-pi/

[4] https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

[5] https://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html

[6] https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html