LIVE DIGITAL TRAFFIC MANAGEMENT SYSTEM

by

RISHABH VERMA 16BCE1394
KUNAL DILIP CHANDIRAMANI 16BCE1396

A project report submitted to

Dr. Jagadeesh Kannan R

in partial fulfilment of the requirements for the course of

CSE3999 – Technical Answers for Real World Problems

in

B.Tech. (Computer Science and Engineering)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

VIT UNIVERSITY, CHENNAI

Vandalur – Kelambakkam Road

Chennai – 600127

APRIL, 2019

BONAFIDE CERTIFICATE

Certified that this project report entitled "**LIVE TRAFFIC MANAGEMENT SYSTEM"** is a bonafide work of RISHABH VERMA (16BCE1394), KUNAL DILIP CHANDIRAMANI (16BCE1396) who carried out the "J"-Project work under my supervision and guidance for CSE3999 – Technical Answers for Real World Problems.

Place: Chennai

Date: 03rd April, 2019

Dr. Jagadeesh Kannan R

School of Computing Science and Engineering (SCSE)

VIT University, Chennai

Chennai – 600 127.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Jagadeesh Kannan R**, School of Electronics Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work. We express our thanks to our Programme Chair **Dr.Rajesh Kanna (for B.Tech-CSE)** for his/her support throughout the course of this project. We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course. We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

TABLE OF CONTENTS

1. ABSTRACT

Congestion in traffic is a major problem these days. Despite the fact that it appears to pervade all over, urban cities are the ones most influenced by it. And it's ever increasing nature makes it imperative to know the road traffic density in real time for better signal control and effective traffic management. There can be different causes of congestion in traffic like insufficient capacity, unrestrained demand, large Red Light delays etc. While insufficient capacity and unrestrained demand are somewhere interrelated, the delay of respective light is hard coded and not dependent on traffic.

This project presents the method to use live images feed from the cameras at traffic junctions for real time traffic density calculation using image processing. It also focuses on the algorithm for switching the traffic lights according to vehicle density on road, thereby aiming at reducing the traffic congestion on roads which will help lower the number of accidents. In turn it will provide safe transit to people and reduce fuel consumption and waiting time. It will also provide significant data which will help in future road planning and analysis. In further stages multiple traffic lights can be synchronized with each other with an aim of even less traffic congestion and free flow of traffic.

2. INTRODUCTION

Over the last decade, the adoption and use of technologies like Mobility, Cloud and Social Platforms has made it possible for common, middle class users to use small, focused applications for making their life easier and comfortable. Whether it is simply paying your utility bills using mobile banking or getting that favorite movie ticket by just clicking couple of buttons, use of technology has really changed the way we live, play and work. Though we have been referring to Smart Cities and communities for some time now, let us look at how use of Information and data available to us can be used to really create some smart services, which in a true sense provide us with better living. We shall look at a key case, which impacts us almost daily: traffic management. Use of technology and real time analysis can actually lead to a smooth traffic management.

The common reason for traffic congestion is due to poor traffic prioritization, where there are such situations some lane has less traffic than the other. Vehicular congestion is increasing at an exponential rate. Let us take the case study of Chandigarh, one of the Union Territories of India. Chandigarh has the largest number of vehicles per capita in India. According to Chandigarh Transport Undertaking, more than 45,000 vehicles were registered this year in Chandigarh making the total count of more than 8 lakhs vehicles on the road. While the number of vehicles are increasing at a fast pace, the infrastructure in the city is not being able to match this growth. Traffic jams during rush hours are becoming a routine affair, especially in the internal sectors where long queues of vehicles can be seen stranded. Therefore, we have tried to address the problem with the help of our project wherein the focus would be to minimize the vehicular congestion. We have achieved this with the help of image processing that can be obtained from surveillance cameras and eventually to deploy a feedback mechanism in the working of the traffic lights where the density of the traffic would also be factored in the decision making process.

3. OBJECTIVES AND GOALS

We propose a technique that can be used for traffic control using image processing. In which we present the method to use live images feed from the cameras at traffic junctions for real time traffic density calculation, deciding priorities when emergency vehicles, traffic violation detection, Ability to read vehicle number using image processing.

It also focuses on the algorithm for switching the traffic lights according to vehicle density on road, thereby aiming at reducing the traffic congestion on roads which will help lower the number of accidents. In turn it will provide safe transit to people and reduce fuel consumption and waiting time. In further stages multiple traffic lights can be synchronized with each other with an aim of even less traffic congestion and free flow of traffic. The vehicles are detected by the system through images instead of using electronic sensors embedded in the pavement.

3.1 BENEFITS

The project will help reduce the time spent at the traffic signals and will therefore enable faster commute within the city traffic. Apart from saving time, since the signal synchronisation is based on the traffic density, it helps in managing the traffic at high density and chaotic hours.

The feature to detect wanted vehicles will help to enforce law and will help in taking timely actions against the criminals. The facial detection feature will enable the law enforcement bodies of the nation to strengthen the law and catch the culprit.

3.2 FEATURES

- Traffic density counter
- Number plate detection
- Face detection
- Dynamic traffic light synchronisation
- Preference given to emergency vehicles
- Tracking wanted vehicles

4. PROPOSED SYSTEM

We propose a technique that can be used for traffic control using image processing. Traffic density of lanes is calculated using image processing which is done of images of lanes that are captured using digital camera. According to the traffic densities on all roads, our model will allocate smartly the time period of green light for each road. We have chosen image processing for calculation of traffic density as cameras are very much cheaper than other devises such as sensors.

5. CONCEPTUAL MODEL

Introduction to Image Processing

Image Processing is a technique to enhance raw images received from cameras/sensors placed on space probes, aircrafts and satellites or pictures taken in normal day-today life for various applications. An Image is rectangular graphical object. Image processing involves issues related to image representation, compression techniques and various complex operations, which can be carried out on the image data. The operations that come under image processing are image enhancement operations such as sharpening, blurring, brightening, edge enhancement etc. Image processing is any form of signal processing for which the input is an image, such as 9 photographs or frames of video; the output of image processing can be either an image or a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it. Image processing usually refers to digital image processing, but optical and analog image processing are also possible.

Image Acquisition

Generally an image is a two-dimensional function f(x,y)(here x and y are plane coordinates).The amplitude of image at any point say f is called intensity of the image. It is also called the grey level of image at that point. We need to convert these x and y values to finite discrete values to form a digital image. The input image is a fundus taken from stare data base and drive data base. The image of the retina is taken for processing and to check the condition of the person. We need to convert the analog image to digital image to process it through digital computer. Each digital image composed of a finite elements and each finite element is called a pixel.

Formation of Image

We have some conditions for forming an image f(x,y) as values of image are proportional to energy radiated by a physical source. So f(x,y) must be nonzero and finite. i.e. $0< f(x,y) < \infty$.

Image Pre-Processing

- Image Resizing/Scaling Image scaling occurs in all digital photos at some stage It happens anytime you resize your image from one pixel grid to another. Image resizing is necessary when you need to increase or decrease the total number of pixels. Even if the same image resize is performed, the result can vary significantly depending on the algorithm.
- RGB to GRAY Con 10 able to generate almost any detectable color. This is the reason behind why color images are often stored as three separate image matrices; one storing the amount of red (R) in each pixel, one the amount of green (G) and one the amount of blue (B). We call such color images as stored in an RGB format. In grayscale images, however, we do not differentiate how much we emit of different colors, we emit the same amount in every channel. We will be able to differentiate the total amount of emitted light for each pixel; little light gives dark pixels and much light is perceived as bright pixels. When converting an RGB image to grayscale, we have to consider the RGB values for each pixel and make as output a single value reflecting the brightness of that pixel. One of the approaches is to take the average of the contribution from each channel: (R+B+C)/3. However, since the perceived brightness is often dominated by the green component, a different, more "human-oriented", method is to consider a weighted average, e.g.: 0.3R + 0.59G + 0.11B.
- Image Enhancement Image enhancement is the process of adjusting digital images so that the results are more suitable for display or further analysis. For example, we can eliminate noise, which will make it easier to identify the key characteristics. In poor contrast images, the adjacent characters merge during binarization. We have to reduce the spread of the characters before applying a threshold to the word image. Hence, we introduce “Power- Law Transformation” which increases the contrast of the characters and helps in better segmentation. The basic form of power-law transformation is

$$\mathbf{s = cr^{\gamma}}$$

where r and s are the input and output intensities, respectively; c and γ are positive constants. A variety of devices used for image capture, printing, and display respond according to a power law. By convention, the exponent in the power-law equation is referred to as gamma. Hence, the process used to correct these power-law response phenomena is called gamma correction. Gamma correction is important, if displaying an image accurately on a computer screen is of concern. In our experimentation, γ is varied in 11 the range of 1 to 5. If c is not equal to ’1’, then the dynamic range of the pixel values will be significantly affected by scaling. Thus, to avoid another stage of rescaling after powerlaw transformation, we fix the value of c = 1. With γ = 1, if the power-law transformed image is passed through binarization, there will be no change in the result compared to simple binarization. When γ > 1, there will be a change in the

histogram plot, since there is an increase of samples in the bins towards the gray value of zero. Gamma correction is important if displaying an image accurately on computer screen is of concern.

- Edge Detection
  Edge detection is the name for a set of mathematical methods which aim at identifying points in a digital image at which the image brightness changes sharply or, more technically, has discontinuities or noise. The points at which image brightness alters sharply are typically organized into a set of curved line segments termed edges. Edge detection is a basic tool in image processing, machine vision and computer envisage, particularly in the areas of feature reveal and feature extraction. Following are list of various edge-detection methods: -

  Sobel Edge Detection Technique
  Perwitt Edge Detection
  Roberts Edge Detection Technique
  Zerocross Threshold Edge Detection Technique
  Canny Edge Detection Technique

  In our project we use "Canny Edge Detection Technique" because of its various advantages over other edge detection techniques.

- Canny Edge Detection
  The Canny Edge Detector is one of the most commonly used image processing tools detecting edges in a very robust manner. It is a multi-step process, which can be implemented on the GPU as a sequence of filters. Canny edge detection technique is based on three basic objectives.

  1. Low error rate: All edges should be found, and there should be no spurious responses. That is, the edges must be as close as possible to the true edges.
  2. Edge point should be well localized: 12 The edges located must be as close as possible to the true edges. That is, the distance between a point marked as an edge by the detector and the center of the true edge should be minimum.
  3. Single edge point response: The detector should return only one point for each true edge point. That is, the number of local maxima around the true edge should be minimum. This means that the detector should not identify multiple edge pixels where only a single edge point exists.

The Canny edge detection algorithm consist of the following basic steps;

i. Smooth the input image with Gaussian filter.
ii. Compute the gradient magnitude and angle images.
iii. Apply non-maxima suppression to the gradient magnitude image.
iv. Use double threshold and connectivity analysis to detect and link edges.

6. HARDWARE ARCHITECTURE

The proposed model is constructed as follows:

We have a Raspberry Pi that is connected to 3 sets of LEDs that represent the traffic lights. The captured images and the reference images are fed manually to the Raspberry Pi currently but we have in mind an automated way to do this via a CCTV camera.

1. **Raspberry Pi:** In this model, we have made use of the Raspberry Pi model 3. The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside of its target market for uses such as robotics. Peripherals (including keyboards, mice and cases) are not included with the Raspberry Pi. The Raspberry is the primary controller of the entire system. It makes use of a python service that runs on start-up automatically to 14 control the lights. It takes the captured image and compares it to the reference image. Depending on the percentage match with the reference image, the Pi determines which signal gets first priority for a green signal. If the traffic is higher on a particular road compared to the other roads at the junction, it gets first priority and the longest green signal. Additionally, the Pi transmits data to the cloud for analytics purposes. The data is sent in a JSON format which includes the following- the location of the signal, the percentage matches of each image and the time when the images were captured. The Pi also receives data from the cloud occasionally about the statistics of the particular junction.

The Pi is connected in the following manner to the traffic lights

Traffic Light 1: R- Pin 11, G- Pin 12

Traffic Light 2: R- Pin 15, G- Pin 13

Traffic Light 3: R- Pin 18, G- Pin 22

Ground: Pin 31

7. SOFTWARE IMPLEMENTATION

7.1 Software Components

The proposed model makes use of following software components:

1. OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human 15 actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies. Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan. It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A fully featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

2. Respbian 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.[1] Raspbian was created by Mike Thompson and Peter Green as an independent project.[4] The initial build was completed in June 2012.[5] The operating system is still under active

development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs.

Raspbian uses PIXEL, Pi Improved X-Window Environment, Lightweight as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Openbox stacking window manager with a new theme and few other changes. The distribution is shipped with a copy of computer algebra program Mathematica and a version of Minecraft called Minecraft Pi[6] as well as a lightweight version of Chromium as of the latest version

7.1 Sample Program Code For Density based traffic light management.

```
# import the necessary packages
from __future__ import print_function
from imutils.video import VideoStream
import numpy as np
import datetime
import imutils
import time
import cv2
import RPi.GPIO as GPIO

from picamera.array import PiRGBArray
from picamera import PiCamera

# initialize the camera and grab a reference to the raw camera capture
webcam1 = VideoStream(src=0).start()
webcam2 = VideoStream(src=1).start()
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))

#use trained cars XML classifiers
car_cascade = cv2.CascadeClassifier('cars.xml')

#setting up LEDs
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)

#initalizing timmers
timmer1 = 30
timmer2 = 30
timmer = 0
counter = 0
counter1 = 0
counter2 = 0

#setting up LED pins
pinr = 18
```

```
ping = 22
pin1r = 12
pin1g = 11
pin2r = 13
pin2g = 15
GPIO.setup(pinr, GPIO.OUT)
GPIO.setup(ping, GPIO.OUT)
GPIO.setup(pin1r, GPIO.OUT)
GPIO.setup(pin1g, GPIO.OUT)
GPIO.setup(pin2r, GPIO.OUT)
GPIO.setup(pin2g, GPIO.OUT)
flag = 0

start = time.time()
# capture frames from the camera
for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):
# grab the raw NumPy array representing the image, then initialize the timestamp
# and occupied/unoccupied text
count = 0
count1 = 0
        count2 = 0
#capture frame by frame
        frame1 = webcam1.read()
        frame2 = webcam2.read()
image = frame.array
#resize your screen
frame1 = imutils.resize(frame1, width=400)
        frame2 = imutils.resize(frame2, width=400)
#convert video into gray scale of each frames
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        gray1 = frame1
        gray2 = frame2
#detect cars in the video
        cars = car_cascade.detectMultiScale(gray, 1.1, 3)
        cars1 = car_cascade.detectMultiScale(gray1, 1.1, 3)
        cars2 = car_cascade.detectMultiScale(gray2, 1.1, 3)

        #to draw arectangle in each cars
        for (x,y,w,h) in cars:
            cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
            count=count+1
        cv2.putText(image, str(count),(10,100), cv2.FONT_ITALIC, 2,(255,255,255),2)

        #to draw arectangle in each cars1
        for (x,y,w,h) in cars1:
            cv2.rectangle(frame1,(x,y),(x+w,y+h),(0,255,0),2)
            count1=count1+1
        cv2.putText(frame1, str(count1),(10,100), cv2.FONT_ITALIC, 2,(255,255,255),2)
```

```
#to draw arectangle in each cars2
for (x,y,w,h) in cars2:
    cv2.rectangle(frame2,(x,y),(x+w,y+h),(0,255,0),2)
    count2=count2+1
cv2.putText(frame2, str(count2),(10,100), cv2.FONT_ITALIC, 2,(255,255,255),2)

if(count != counter or count1 != counter1 or count2 != counter2):
    #setting the timmer
    #for frame
    if(count<=1):
        timmer = 5
    elif(count==2):
        timmer = 10
    else:
        timmer = 15

    #for frame1
    if(count1<2):
        timmer1 = 5
    elif(count1==2):
        timmer1 = 10
    else:
        timmer1 = 15

    #for frame2
    if(count2<2):
        timmer2 = 5
    elif(count2==2):
        timmer2 = 10
    else:
        timmer2 = 15
    print("camera-direction    timmer")
    print("     picam:          ",timmer,"s")
    print("    webcam1:         ",timmer1,"s")
    print("    webcam2:         ",timmer2,"s")
    counter = count
    counter1 = count1
    counter2 = count2

if (flag == 0):
    tim = timmer1
    if(time.time() - start <= tim):
        GPIO.output(pin1g, GPIO.HIGH)
        GPIO.output(pin2r, GPIO.HIGH)
        GPIO.output(pinr, GPIO.HIGH)
    else:
        GPIO.output(pin1g, GPIO.LOW)
        GPIO.output(pin2r, GPIO.LOW)
        GPIO.output(pinr, GPIO.LOW)
```

```
            start = time.time()
            flag = 1

        if (flag == 1):
            tim = timmer2
            if(time.time() - start <= tim):
                GPIO.output(pin2g, GPIO.HIGH)
                GPIO.output(pin1r, GPIO.HIGH)
                GPIO.output(pinr, GPIO.HIGH)
            else:
                GPIO.output(pin2g, GPIO.LOW)
                GPIO.output(pin1r, GPIO.LOW)
                GPIO.output(pinr, GPIO.LOW)
                start = time.time()
                flag = 2

        if (flag == 2):
            tim = timmer
            if(time.time() - start <= tim):
                GPIO.output(ping, GPIO.HIGH)
                GPIO.output(pin1r, GPIO.HIGH)
                GPIO.output(pin2r, GPIO.HIGH)
            else:
                GPIO.output(ping, GPIO.LOW)
                GPIO.output(pin1r, GPIO.LOW)
                GPIO.output(pin2r, GPIO.LOW)
                start = time.time()
                flag = 0


# show the frame
#cv2.imshow('video1', frame1)
        #cv2.imshow('video2', frame2)
        #cv2.imshow("Picam Frame", image)
        key = cv2.waitKey(1) & 0xFF

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

# if the `q` key was pressed, break from the loop
if key == ord("q"):
break

#close all the frames
cv2.destroyAllWindows()
```

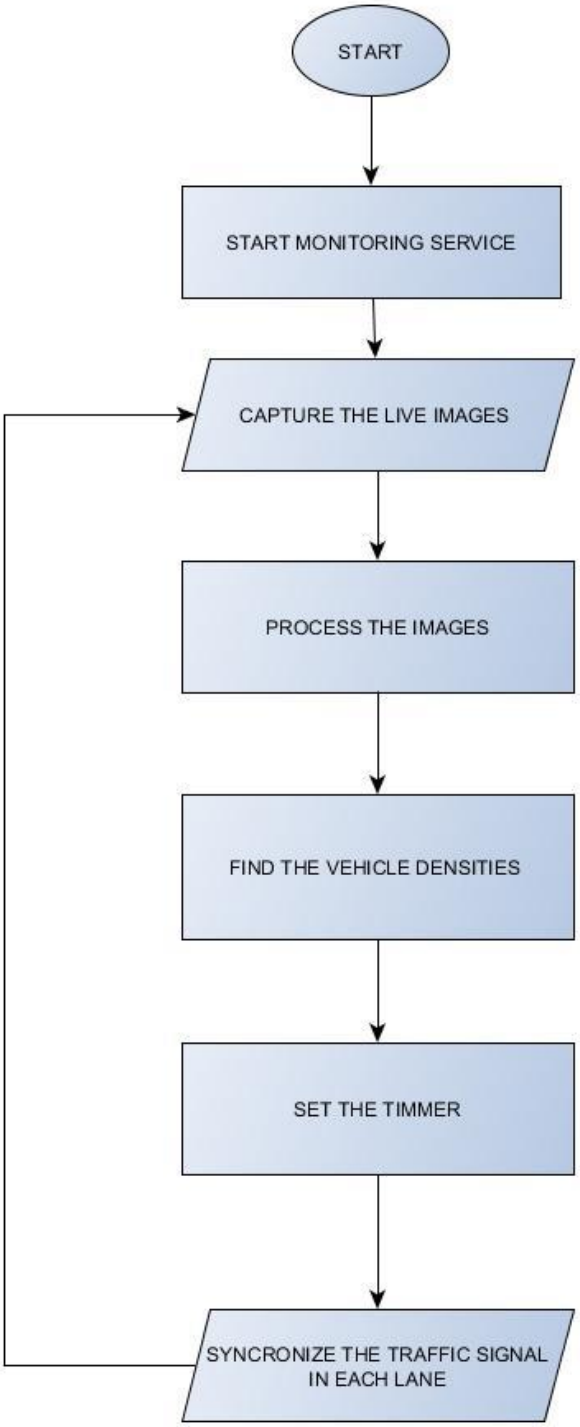
7.2 Sample Program Code for Number plate detection

```
cap = cv2.VideoCapture(0)
df = pd.DataFrame(columns = ['date', 'v_number'])

results = []
with open("input.csv") as csvfile:
   reader = csv.reader(csvfile) # change contents to floats
   for row in reader: # each row is a list
      results.append(row)

while True:
        ret,image = cap.read()

        image = imutils.resize(image, width=500)

        cv2.imshow("Original Image", image)

        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        #cv2.imshow("1 - Grayscale Conversion", gray)

        gray = cv2.bilateralFilter(gray, 11, 17, 17)
        #cv2.imshow("2 - Bilateral Filter", gray)

        edged = cv2.Canny(gray, 170, 200)
        cv2.imshow("4 - Canny Edges", edged)

        (new, cnts, _) = cv2.findContours(edged.copy(), cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
        cnts=sorted(cnts, key = cv2.contourArea, reverse = True)[:30]
        NumberPlateCnt = None
        flag = 0

        count = 0
        for c in cnts:
                        peri = cv2.arcLength(c, True)
                        approx = cv2.approxPolyDP(c, 0.02 * peri, True)
                        if len(approx) == 4:
                                flag = 1
                                NumberPlateCnt = approx
                                break

        if flag == 1:
                # Masking the part other than the number plate
                mask = np.zeros(gray.shape,np.uint8)
                new_image = cv2.drawContours(mask,[NumberPlateCnt],0,255,-1)

                new_image = cv2.bitwise_and(image,image,mask=mask)
                cv2.namedWindow("Final_image",cv2.WINDOW_NORMAL)
                cv2.imshow("Final_image",new_image)
```

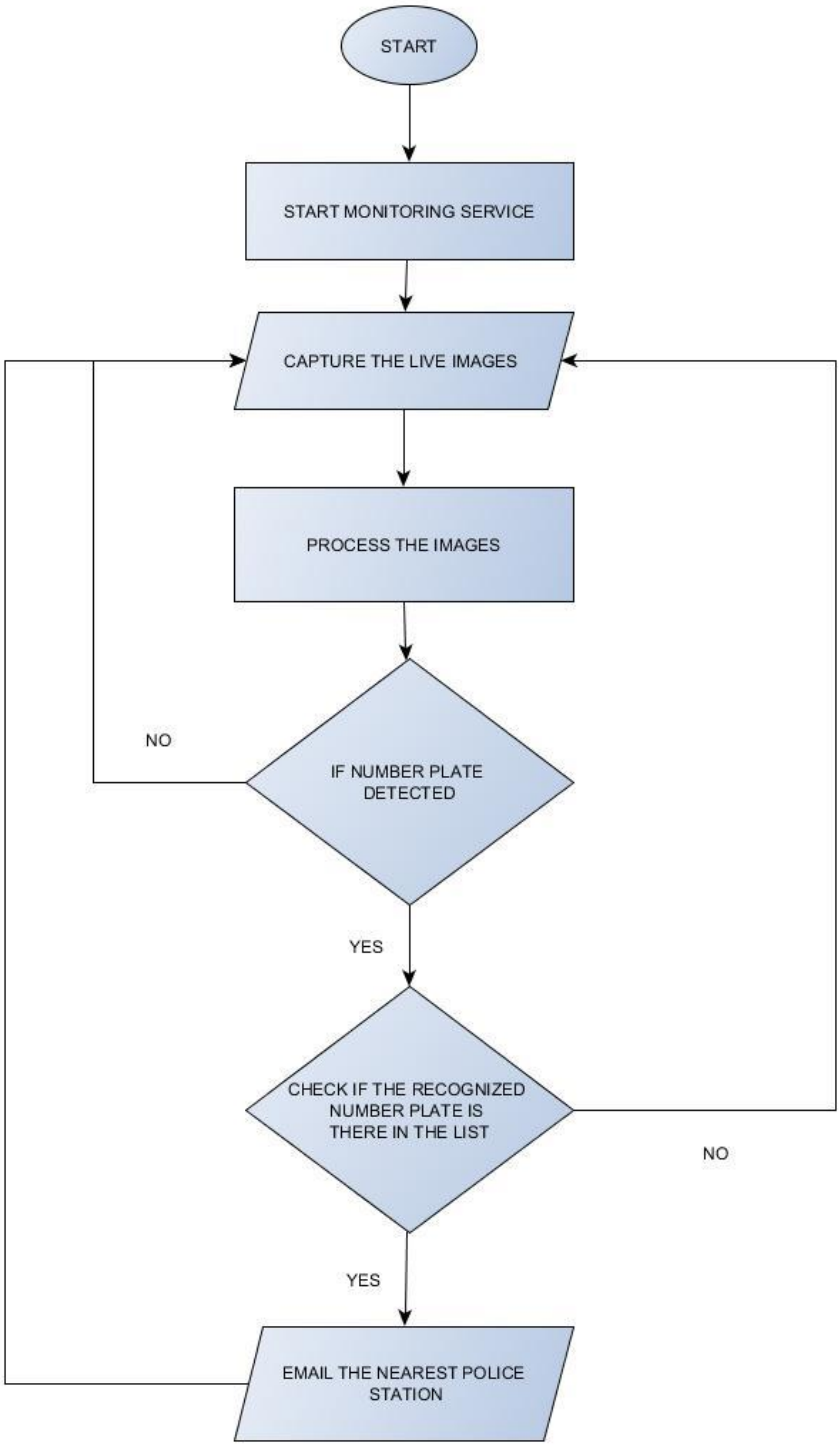
```
                # Configuration for tesseract
                config = ('-l eng --oem 1 --psm 3')

                # Run tesseract OCR on image
                text = pytesseract.image_to_string(new_image, config=config)

                for i in range(len(results)):
                        if(results[i][0] == text):
                                print("Wanted Number plate found: ",text)
                                now = datetime.datetime.now()
                                # creates SMTP session
                                s = smtplib.SMTP('smtp.gmail.com', 587)

                                # start TLS for security
                                s.starttls()

                                # Authentication
                                s.login("kunal.kc.chandiramani@gmail.com",
"7666561900")

                                # message to be sent
                                SUBJECT = "Wanted vehicle found "
                                TEXT = text +" "+ results[i][1] +" car was detected near the
Kelambakkam traffic signal at " + str(now)
                                message = 'Subject: {}\n\n{}'.format(SUBJECT, TEXT)

                                # sending the mail
                                s.sendmail("kunal.kc.chandiramani@gmail.com",
"kunal.kc.chandiramani@gmail.com", message)

                                # terminating the session
                                s.quit()
                                break

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

df.to_csv('data.csv')
cap.release()
cv2.destroyAllWindows()
```

8. OPERATIONAL FLOWCHARTS

8.1 Vehicle density detection

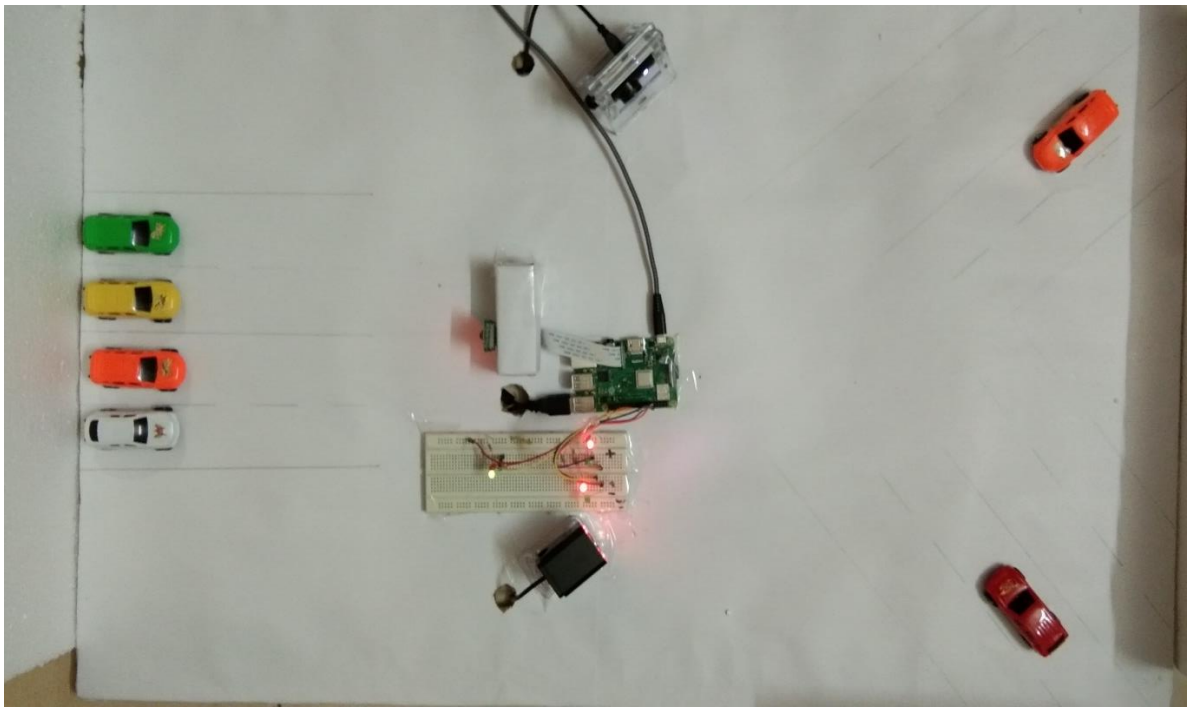


8.2 Vehicle number plate detection

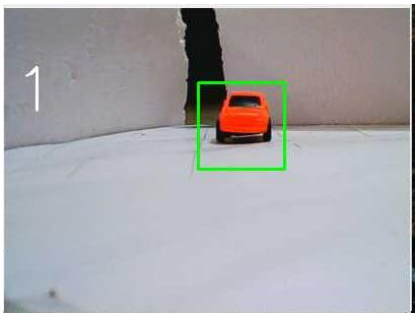


9. TEST CASES

Test Case 1:

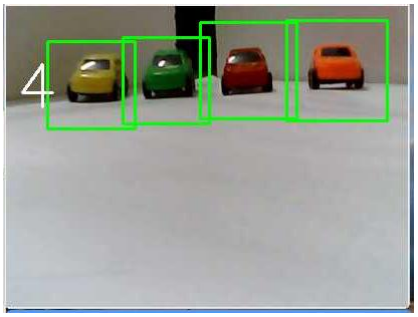


Test Case

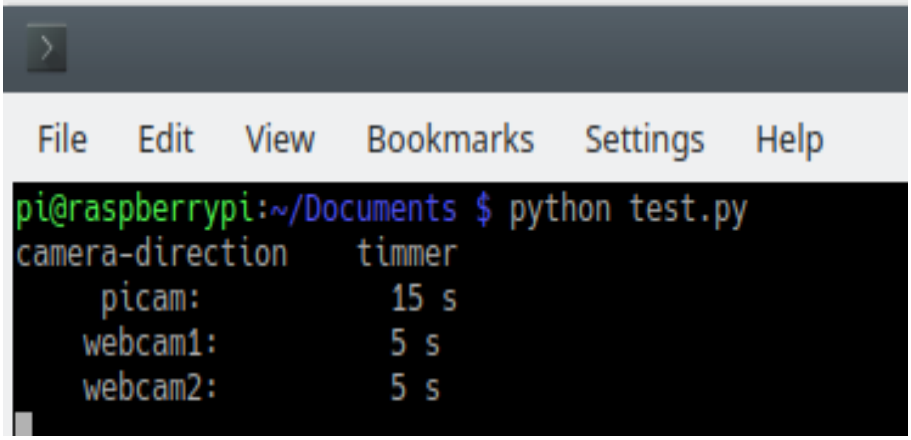


Webcam 1

Webcam 2

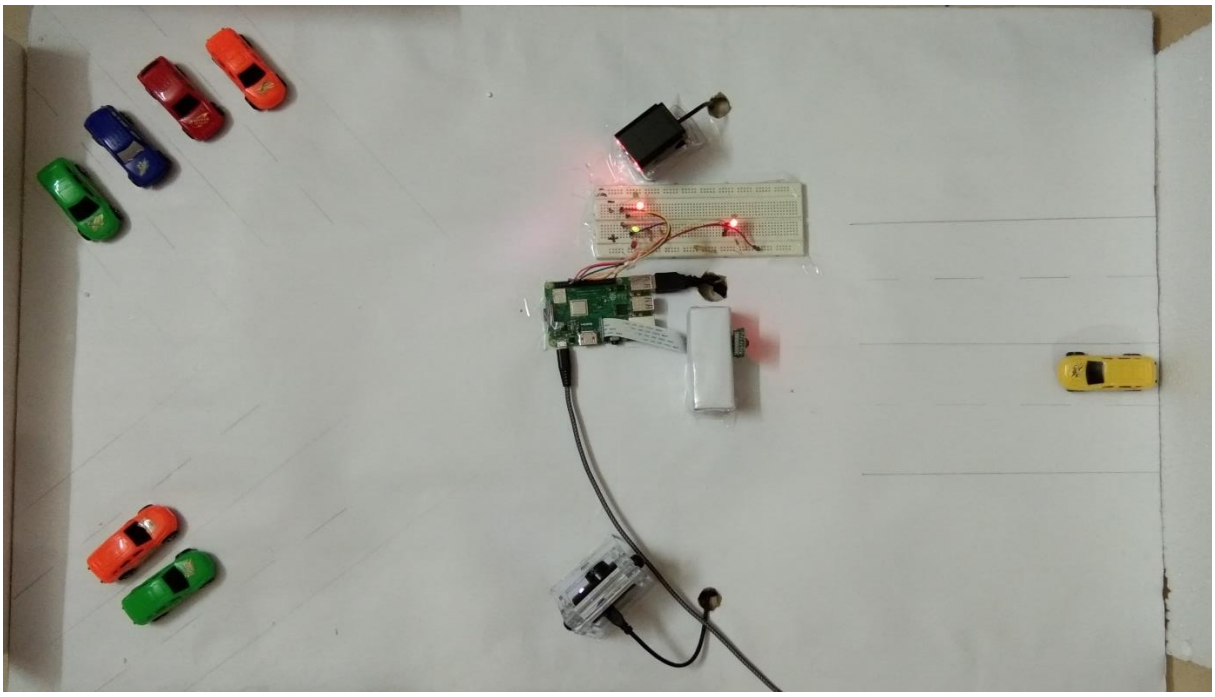


piCam

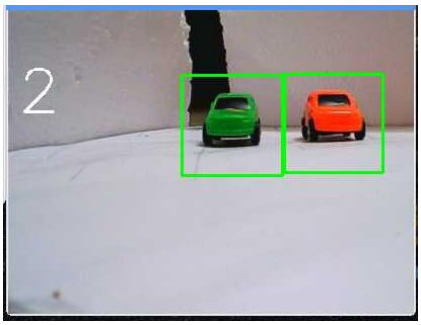


Traffic Light Timers

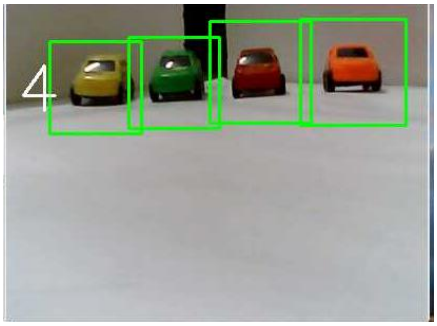
Test Case 2:



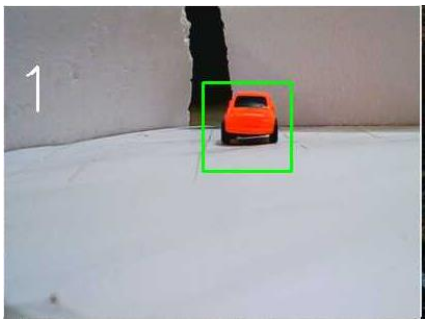
Test Case



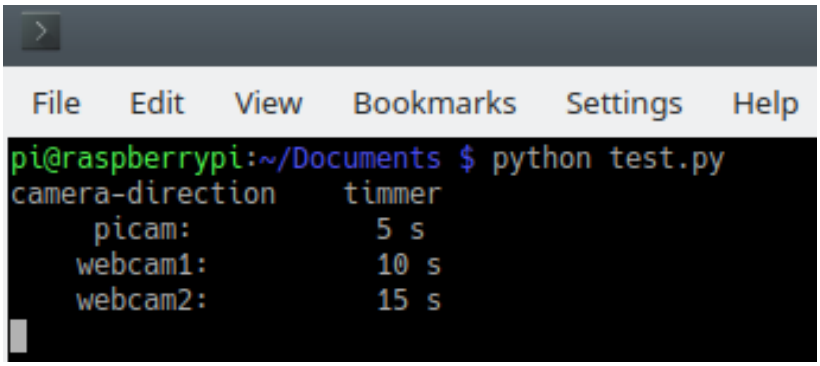
Webcam 1



Webcam 2



piCam



Traffic Light Timers

10. CONCLUSION AND FUTURE WORK

10.1 CONCLUSION

Our model provides a solution to reduce traffic congestion on roads overriding the older system of hard coded lights which cause unwanted delays. Reducing congestion and waiting time will lessen the number of accidents and also reduces fuel consumption which in turn will help in controlling the air pollution. Moreover, the purview of our project can be augmented for Coordination Control which places traffic signals on a coordinated system so that drivers encounter long strings of green lights. This will also provide data for future road design and construction or where improvements are required and which are urgent like which junction has higher waiting times.

10.2 FUTURE WORK

In future, some enhancements can be done on the project. Some tasks that should be done in the future and would develop the system to a more mature state are as follows:

1. We can use infrared cameras and accordingly apply some suitable image processing methods which in turn will enable the system to control traffic effectively in low lights and night time also.
2. We can apply some suitable image processing techniques and enable the system to detect the emergency vehicles like ambulances and fire brigades to let them pass as soon as possible i.e. providing priority to such vehicles which will save a lot of lives and property.

11. REFERENCES

1. https://www.jetbrains.com/pycharm/documentation/

2. S. Hamidreza Kasaei, S. Mohammadreza Kasaei, “New Morphology-Based Method for RobustIranian Car Plate Detection and Recognition” International Journal of Computer Theory and Engineering, Vol. 2, No. 2 April, 2010

3. K.V. Arya ; Shailendra Tiwari, “Real-time vehicle detection and tracking” 2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)

4. Luigi Di Stefano, Enrico Viarani, “Vehicle Detection and Tracking Using the Block Matching Algorithm” IEEE International Conference September 16-19 2016

5. R. J. Shalkoff, Digital Image Processing and Computer Vision, Wiley, 1989, pp.137-144.

6. F. Bartolini, V. Cappellini, C. Giani, Motion Estimation and Tracking for Urban Traffic Monitoring, Proceedings 3rd IEEE International Conference on Image Processing ICIP'96,Lausanne, Switzerland, September 16-19 1996, pp. 787-790

7. Khekare, G.S.; Sakhare, A.V., "A smart city framework for intelligent traffic system using VANET," Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013 International Multi-Conference on, vol., no., pp.302,305, 22-23 March 2013

8. http://opencv.org/about.html

9. https://en.wikipedia.org/wiki/Tesseract