MIT World Peace University

INTERNET OF THINGS
Second Year B.Tech, Semester 2

Smarter Signals for Better Roads

Project Report

Project Members

PA15. Parth Zarekar
PA20. Krishnaraj Thadesar
PA25. Nandana Nambiar
PA26. Anuj Choudhary
PA31. Rajdeep Chauhan

Batch A1

May 8, 2023

Contents

1 Introduction

The objective of this project is to develop an IoT-based system called "Smart Signals" that detects emergency vehicles and manipulates traffic lights to facilitate their safe and quick passage. The system uses computer vision techniques and a Raspberry Pi to detect emergency vehicles, and manipulate traffic lights accordingly. This project can be a useful addition to smart city infrastructure, as it helps to reduce traffic congestion and improve emergency response times.

2 Literature Survey

Several research studies have been conducted on IoT-based traffic management systems. Leekha et al. (2017) proposed an IoT-based traffic management system that uses sensors to monitor traffic flow, and provides real-time information to drivers to help them avoid congested areas. Kumar et al. (2018) developed an RFID-based smart traffic control system that allows emergency vehicles to bypass traffic signals, thereby reducing response times.
In this project, we have used computer vision techniques and a Raspberry Pi to detect emergency vehicles and manipulate traffic lights accordingly. We have used the OpenCV library to process images captured by a camera module, and Python programming to manipulate traffic lights.

3 Implementation

3.1 Components used with detailed specifications

- **Raspberry Pi 4:** The Raspberry Pi 4 is a powerful single-board computer that is ideal for IoT applications. It has a 1.5GHz quad-core ARM Cortex-A72 CPU, 2GB of RAM, and supports multiple operating systems.
- **Jumper wires:** Jumper wires are used to connect the components of the system together.
- **Wires:** Wires are used to connect the Raspberry Pi to the camera module and other components.
- **Breadboard:** A breadboard is used to make temporary connections between the components of the system.
- **Camera module:** The camera module is used to capture images of the road.

3.2 Architecture

The system architecture can be divided into three main components:

1. **Image Capture:** The Camera module captures images of the road.
2. **Image Processing:** The images captured by the camera module are processed using the OpenCV library.
3. **Traffic Light Control:** The Raspberry Pi manipulates traffic lights based on the results of image processing.

3.3 Connection Details

The camera module is connected to the Raspberry Pi using a ribbon cable. The Raspberry Pi is connected to a breadboard using jumper wires, and the traffic lights are connected to the breadboard using wires. The connections between the components are as follows:

- **Camera Module to Raspberry Pi:** The ribbon cable is connected to the camera module and the Raspberry Pi.
- **Raspberry Pi to Breadboard:** The Raspberry Pi is connected to the breadboard using jumper wires.
- **Traffic lights tot Breadboard:** The traffic lights are connected to the breadboard using wires.

4 Codes

4.1 Code for Image Processing

```
import cv2
import numpy as np

def has_ambulance(image_path):
    # Load the image
    image = cv2.imread(image_path)
    image = cv2.resize(image, (640, 480))
    # Convert the image to HSV color space
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Define the range of red color in HSV
    lower_red = np.array([0, 50, 50])
    upper_red = np.array([10, 255, 255])
    mask1 = cv2.inRange(hsv, lower_red, upper_red)

    lower_red = np.array([170, 50, 50])
    upper_red = np.array([180, 255, 255])
    mask2 = cv2.inRange(hsv, lower_red, upper_red)

    # Combine the two masks for red color detection
    mask = cv2.bitwise_or(mask1, mask2)

    # Check if there are any non-zero pixels in the mask
    has_red = np.any(mask)

    return has_ambulance
```

4.2 Code for Traffic Light Control

```
import RPi.GPIO as GPIO
import time

    # Set up GPIO pin 12 (which corresponds to board pin 32) as output
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(32, GPIO.OUT)

    # Turn on the LED for 5 seconds
    GPIO.output(32, GPIO.HIGH)
    time.sleep(5)

    # Turn off the LED
    GPIO.output(32, GPIO.LOW)

    # Clean up the GPIO settings
    GPIO.cleanup()
```

4.3 Code for Server

```
import os
import socket
import cv2
import numpy
import base64
import glob
import sys
import time
import threading
import objectrecognition as objrec
from datetime import datetime

class ServerSocket:

```

```
    def __init__(self, ip, port):
        self.TCP_IP = ip
        self.TCP_PORT = port
        self.createImageDir()
        self.folder_num = 0
        self.socketOpen()

        self.receiveThread = threading.Thread(target=self.receiveImages)
        self.receiveThread.start()
        self.process_images_and_send_thread = threading.Thread(target=self.
    process_images_and_send)
        self.process_images_and_send_thread.start()

    def process_images_and_send(self):
        while True:
            try:
                obj = objrec.has_red_color('./8080_images0/img.jpg')
                time.sleep(0.203)
                if obj == True:
                    # time.sleep(1)
                    self.conn.send(bytes('1', "utf-32"))
                    print('Detected')
                else:
                    # time.sleep(1)
                    self.conn.send(bytes('0', "utf-32"))
                    print('Not Detected')
            except Exception as e:
                print(e)
    def socketClose(self):
        self.sock.close()
        print(u'Server socket [ TCP_IP: ' + self.TCP_IP + ', TCP_PORT: ' + str(self.TCP_PORT) +
     ' ] is close')

    def socketOpen(self):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.bind((self.TCP_IP, self.TCP_PORT))
        self.sock.listen(1)
        print(u'Server socket [ TCP_IP: ' + self.TCP_IP + ', TCP_PORT: ' + str(self.TCP_PORT) +
     ' ] is open')
        self.conn, self.addr = self.sock.accept()
        print(u'Server socket [ TCP_IP: ' + self.TCP_IP + ', TCP_PORT: ' + str(self.TCP_PORT) +
     ' ] is connected with client')

    def receiveImages(self):
        cnt_str = ''
        cnt = 0

        try:
            while True:
                if (cnt < 10):
                    cnt_str = '000' + str(cnt)
                elif (cnt < 100):
                    cnt_str = '00' + str(cnt)
                elif (cnt < 1000):
                    cnt_str = '0' + str(cnt)
                else:
                    cnt_str = str(cnt)
                if cnt == 0: startTime = time.localtime()
                cnt += 1

                length = self.recvall(self.conn, 64)
                length1 = length.decode('utf-8')
                stringData = self.recvall(self.conn, int(length1))
                # stime = self.recvall(self.conn, 64)
                # print('send time: ' + stime.decode('utf-8'))
                # now = time.localtime()
                # print('receive time: ' + datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.%f'))
                data = numpy.frombuffer(base64.b64decode(stringData), numpy.uint8)
                decimg = cv2.imdecode(data, 1)
                cv2.imshow("image", decimg)
```

```
81                cv2.imwrite('./' + str(self.TCP_PORT) + '_images' + str(self.folder_num) + '/
    img' + '.jpg', decimg)
82                print(cnt_str + ' image received')
83                cv2.waitKey(1)
84                # if (cnt == 60 * 10):
85                #     cnt = 0
86                #     convertThread = threading.Thread(target=self.convertImage(str(self.
    folder_num), 600, startTime))
87                #     convertThread.start()
88                #     self.folder_num = (self.folder_num + 1) % 2
89            except Exception as e:
90                print(e)
91                self.convertImage(str(self.folder_num), cnt, startTime)
92                self.socketClose()
93                cv2.destroyAllWindows()
94                self.socketOpen()
95                self.receiveThread = threading.Thread(target=self.receiveImages)
96                self.receiveThread.start()
97
98        def createImageDir(self):
99
100           folder_name = str(self.TCP_PORT) + "_images0"
101           try:
102               if not os.path.exists(folder_name):
103                   os.makedirs(os.path.join(folder_name))
104           except OSError as e:
105               if e.errno != errno.EEXIST:
106                   print("Failed to create " + folder_name +  " directory")
107                   raise
108
109           # folder_name = str(self.TCP_PORT) + "_images1"
110           # try:
111           #     if not os.path.exists(folder_name):
112           #         os.makedirs(os.path.join(folder_name))
113           # except OSError as e:
114           #     if e.errno != errno.EEXIST:
115           #         print("Failed to create " + folder_name + " directory")
116           #         raise
117
118           folder_name = "videos"
119           try:
120               if not os.path.exists(folder_name):
121                   os.makedirs(os.path.join(folder_name))
122           except OSError as e:
123               if e.errno != errno.EEXIST:
124                   print("Failed to create " + folder_name + " directory")
125                   raise
126
127       def recvall(self, sock, count):
128           buf = b''
129           while count:
130               newbuf = sock.recv(count)
131               if not newbuf: return None
132               buf += newbuf
133               count -= len(newbuf)
134           return buf
135
136       def convertImage(self, fnum, count, now):
137           img_array = []
138           cnt = 0
139           for filename in glob.glob('./' + str(self.TCP_PORT) + '_images' + fnum + '/*.jpg'):
140               if (cnt == count):
141                   break
142               cnt = cnt + 1
143               img = cv2.imread(filename)
144               height, width, layers = img.shape
145               size = (width, height)
146               img_array.append(img)
147
148           file_date = self.getDate(now)
```

```
        file_time = self.getTime(now)
        name = 'video(' + file_date + ' ' + file_time + ').mp4'
        file_path = './videos/' + name
        out = cv2.VideoWriter(file_path, cv2.VideoWriter_fourcc(*'.mp4'), 20, size)

        for i in range(len(img_array)):
            out.write(img_array[i])
        out.release()
        print(u'complete')

    def getDate(self, now):
        year = str(now.tm_year)
        month = str(now.tm_mon)
        day = str(now.tm_mday)

        if len(month) == 1:
            month = '0' + month
        if len(day) == 1:
            day = '0' + day
        return (year + '-' + month + '-' + day)

    def getTime(self, now):
        file_time = (str(now.tm_hour) + '_' + str(now.tm_min) + '_' + str(now.tm_sec))
        return file_time

def main():
    server = ServerSocket('', 8080)

if __name__ == "__main__":
    main()
```

4.4 Code for Client

```
    import socket
    import cv2
    import numpy
    import time
    import base64
    import sys
    from datetime import datetime

    class ClientSocket:
        def __init__(self, ip, port):
            self.TCP_SERVER_IP = ip
            self.TCP_SERVER_PORT = port
            self.connectCount = 0
            self.connectServer()

        def connectServer(self):
            try:
                self.sock = socket.socket()
                self.sock.connect((self.TCP_SERVER_IP, self.TCP_SERVER_PORT))
                print(u'Client socket is connected with Server socket [ TCP_SERVER_IP: ' + self
    .TCP_SERVER_IP + ', TCP_SERVER_PORT: ' + str(self.TCP_SERVER_PORT) + ' ]')
                self.connectCount = 0
                self.sendImages()
            except Exception as e:
                print(e)
                self.connectCount += 1
                if self.connectCount == 10:
                    print(u'Connect fail %d times. exit program'%(self.connectCount))
                    sys.exit()
                print(u'%d times try to connect with server'%(self.connectCount))
                self.connectServer()

        def sendImages(self):
            cnt = 0
            capture = cv2.VideoCapture(0)
            capture.set(cv2.CAP_PROP_FRAME_WIDTH, 480)
            capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 315)
```

```
            try:
                while capture.isOpened():
                    ret, frame = capture.read()
                    resize_frame = cv2.resize(frame, dsize=(480, 315), interpolation=cv2.
    INTER_AREA)

                    now = time.localtime()
                    stime = datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.%f')

                    encode_param=[int(cv2.IMWRITE_JPEG_QUALITY),90]
                    result, imgencode = cv2.imencode('.jpg', resize_frame, encode_param)
                    data = numpy.array(imgencode)
                    stringData = base64.b64encode(data)
                    length = str(len(stringData))
                    self.sock.sendall(length.encode('utf-8').ljust(64))
                    self.sock.send(stringData)
                    self.sock.send(stime.encode('utf-8').ljust(64))
                    print(u'send images %d'%(cnt))
                    cnt+=1
                    time.sleep(1)
            except Exception as e:
                print(e)
                self.sock.close()
                time.sleep(1)
                self.connectServer()
                self.sendImages()

    def main():
        TCP_IP = 'localhost'
        TCP_PORT = 8080
        client = ClientSocket(TCP_IP, TCP_PORT)

    if __name__ == "__main__":
        main()
```

5 Output

The output of the system is the manipulation of the traffic lights at the intersection where the emergency vehicle is approaching. When an emergency vehicle is detected by the system, the traffic lights in the direction of the emergency vehicle turn green, and the traffic lights in all other directions turn red. This ensures that the emergency vehicle can navigate through the intersection without any hindrance and reach its destination as quickly as possible.

5.1 Traffic Light Turns Green in the Direction of the Emergency Vehicle

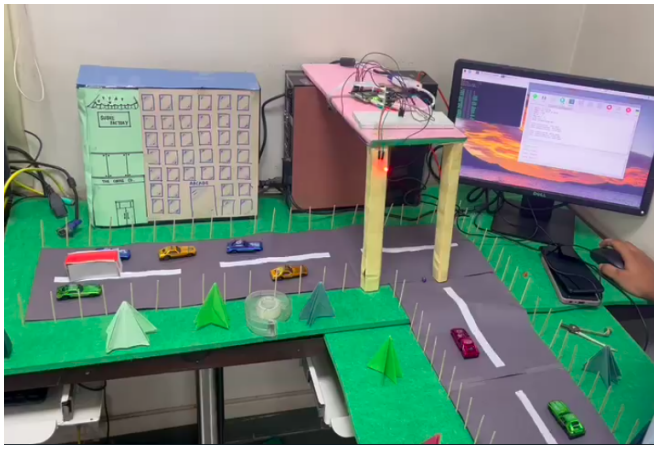

Figure 1: Traffic Light Turns Green in the Direction of the Emergency Vehicle

5.2 Traffic Light Turns Red after the Emergency Vehicle Passes

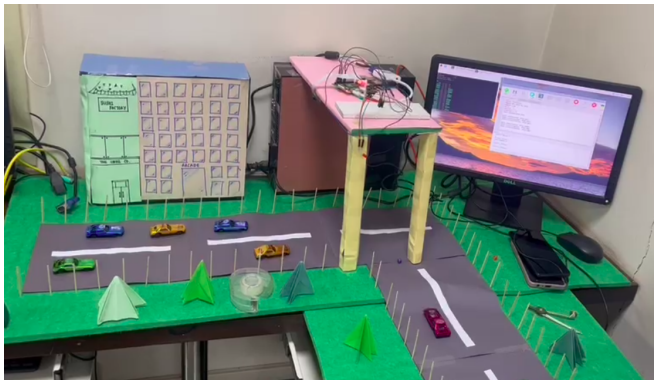

Figure 2: Traffic Light Turns Red after the Emergency Vehicle Passes

6 Future Scope

This project has several potential avenues for future development.

6.1 Integration with other emergency response systems

In its current form, our Smart Signals project can detect and respond to ambulance vehicles. However, in the future, it can be integrated with emergency services such as Fire or police departments, so that the emergency vehicle is detected automatically when it is dispatched. .

6.2 Predict analytics

Our system reacts to an emergency vehicle once it is detected by the camera module. However, it could be enhanced by incorporating predictive analytics to anticipate when an emergency vehicle will arrive. This would allow for the traffic lights to be adjusted in advance, reducing the response time and congestion.

6.3 Integration with Smart City initiatives

Our Smart Signals project can be integrated with other Smart City initiatives, such as intelligent traffic management systems or smart parking systems. This integration would allow for a more comprehensive traffic management solution, improving traffic flow and reducing congestion.

7 Conclusion

In conclusion, our Smart Signals project is a successful proof-of-concept for an intelligent traffic management system. By detecting and responding to emergency vehicles, it can improve response times and reduce congestion. The project is built using readily available hardware components and open-source software, making it affordable and accessible for implementation in real-world scenarios.

8 References

- Huang, Y., Xu, S., Zhou, X., & Li, H. (2019). An intelligent traffic management system based on internet of things and big data. Journal of Ambient Intelligence and Humanized Computing, 10(7), 2791-2802.
- Tiwari, A., & Sen, P. (2020). Smart traffic management system using IoT and AI. International Journal of Computer Sciences and Engineering, 8(1), 11-14.

- Vashistha, A., & Sharma, S. K. (2021). Intelligent traffic management system using IoT and deep learning. International Journal of Advanced Research in Computer Science, 12(2), 1-5.