ASHESI UNIVERSITY

AN AFFORDABLE DENSITY-BASED TRAFFIC MANAGEMENT SYSTEM

CAPSTONE PROJECT

B.Sc. Electrical and Electronics Engineering

Odero Margaret Anyango

2019

ASHESI UNIVERSITY

AN AFFORDABLE DENSITY-BASED TRAFFIC MANAGEMENT SYSTEM

CAPSTONE PROJECT

Capstone Project submitted to the Department of Engineering, Ashesi University in partial fulfilment of the requirements for the award of Bachelor of Science degree in Electrical and Electronics Engineering.

Odero Margaret Anyango

2019

Declaration

I hereby declare that this capstone is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date: ……

I hereby declare that preparation and presentation of this capstone were supervised in accordance with the guidelines on supervision of capstone laid down by Ashesi University.

Supervisor's Signature:

……

Supervisor's Name:

……

Date: ……

Acknowledgement

I would like to express my gratitude to my capstone supervisor, Dr. Nathan Amanquah whose encouragement, as well as professional and academic advice, helped me successfully undertake this project while gaining a lot of new knowledge. I am deeply indebted.

Special thanks to Salley K. N.(2019), whose constant inspiration encouraged me to always keep moving, and who impacted me with a lot of resilience as I undertook this project.

My sincere gratitude goes to the engineering faculty and staff members who have been ready to assist throughout the project period.

I would also like to acknowledge my family and friends, who have continuously encouraged me to be a better version of myself throughout my four years of study, and especially as I undertook this project

Finally, I thank the Almighty God, whose sufficient grace kept me strong throughout this project.

Abstract

Traffic management systems at road intersections of most African cities is based on fixed timings where equal length of green light is assigned to each lane at the intersection. This method of traffic management is inefficient because it causes unnecessarily long waiting times for vehicles at the intersections especially during periods when some lanes have few or no vehicles passing through them. In this project, an affordable density based traffic management system is designed and implemented. This is achieved by selecting components that are low cost, consume low amounts of power and do not require digging onto the road pavements. A pi camera with night vision capability is used as the vehicle detection device for every lane. Image processing is then done using an algorithm running on a raspberry pi to determine the number of vehicles on each lane. This information on the number of vehicles is sent to a central raspberry pi via a radio transceiver which then uses an algorithm to determine the next lane to receive green light, as well as the green light duration. In testing the system, the best position of the camera relative to the lane is investigated in an experiment. The furthest distance the camera can be mounted is determined to be 30m. The night vision capability of the camera is also tested. Despite the advantages that come with this low-cost system, it has some limitations. The camera quality is low hence the image processing algorithm may not be completely accurate in distinguishing between objects in the image. The camera's field of view is also small, necessitating the use of an extra lens that will increase its field of view.

Key Terms: Intelligent Transport Systems; vehicle detection; pi to pi communication; green light sequence and length determination.

Table of Contents

List of Tables

List of Figures

Chapter 1 : Introduction

1.1. Background

The concept of smart cities evokes the idea of using improved technologies to increase efficiency in urban areas. A smart city is a technologically advanced city capable of attaining competitiveness and sustainability by integrating different dimensions of development, thus is self-sufficient [1]. A smart city "monitors and integrates conditions of all its critical infrastructure, hence can better optimize its resources, plan its preventive maintenance activities, and monitor security aspects while maximizing services to its citizens" [2]. This concept emphasizes various aspects of a city's functionality such as smart economy, smart environment, smart governance, smart living, smart people, and smart mobility [3]. One important aspect is smart mobility [3] in which traffic management is core. For efficient traffic management in smart cities, the existing transportation networks are integrated with digital communication and other advanced technologies to achieve intelligent transport systems (ITS). ITS is an area that has received much attention from scholars, and many countries have made attempts at implementing some ITS aspects.

Several countries worldwide have made great progress in the application of ITS technologies to traffic management. Advanced technologies of traffic management have been employed in nations such as Australia, UK, and the USA. In Sydney (Australia), an adaptive traffic control systems known as Sydney Co-ordinates Adaptive Traffic System (SCATS) and InSync are commonly used to reduce stop time of vehicles hence reducing delays [4]. In the UK, the ITS technology used is called Urban Traffic Optimization by Integrated Network Automation (UTOPIA) while in the USA, Adaptive Control Software (ACS) Lite is used [5].

These technologies have effectively reduced time delays by vehicles on roads, reduced fuel consumption, as well as air pollution from exhaust fumes. For instance, a simulation to test the effectiveness of UTOPIA was performed using real-world data from an intersection in Stockholm, Sweden. The results show that with UTOPIA, five out of the seven intersections tested operate at acceptable vehicle waiting times [6]. Similarly, another performance evaluation was conducted on ACS Lite whose results show a reduction in fuel usage by 4% and 7% in two different locations, and a reduction in delay time by 35% and 27% in the two locations [7]. However, such technologies are hardly found in many African cities.

1.2. Problem Definition

For most cities in Africa, there is an increasing number of automobiles without a corresponding improvement in urban planning to manage the traffic congestion. Hours on end are spent on the road each day as motorists try to maneuver their way through the nerve-wracking traffic in the city. Nairobi residents, for instance, spend an average of 62.44 minutes a day in traffic to commute from residential areas to the Central Business District. According to a Bloomberg Business issue of 2014, traffic costs Nairobi $570,000 a day [8], a worrying statistic on the country's economy. According to the Kenyan government in 2016, the time wasted in traffic jams represents a cost of $ 578,000 (Sh58.4 million) a day in lost productivity which translates to more than Sh17.3 billion a year. Kenya's urban population is increasing by around 4.3 percent a year [9], hence this calls for construction of more roads if traffic congestion is to be minimized. Nairobi's road network covers only about 12% of the land [10], which is less than half of the 30% recommended by UN-Habitat for a modern capital city. However, the

cost of increasing the road network from 12% land cover to 30% is high, hence a need for a more affordable way of dealing with the traffic congestion.

One major area of improvement for traffic congestion in smart cities is the traffic management system at road intersections. Currently, the traffic signal system that is mostly used in road intersections of most African cities is based on scheduled timings. Equal 'GO' times are allocated for all lanes meeting at an intersection irrespective of the number of vehicles on these lanes [11], and the traffic lights do not respond to the changing traffic conditions. In most cases there are unnecessarily large time delays between traffic lights, causing vehicles to spend a large amount of time on the roads, waste fuel and emit more fumes to the environment if other lanes with the 'GO' time have no vehicles. This means there is a need for a more efficient system that will cut down on time consumed by vehicles on roads, reduce on the usage of fuel by vehicles, as well as cut down on exhaust fume emissions.

1.3. Objectives

This project aims at devising an affordable and efficient traffic management system with special attention to road intersections in African cities. Traffic flow is regulated by monitoring traffic density on the streets merging at an intersection and using that to influence the traffic light durations. For each lane at the intersection, there needs to be an effective vehicle detection technique that will accurately determine the number of vehicles on that lane. There also needs to be communication amongst the different lanes to compare the traffic densities on different lanes and this information is fed into an algorithm run by a microcomputer to determine the optimal wait times and green light times for different lanes. In doing this, the following objectives are achieved:

- Cutting down on time consumed by vehicles on roads by reducing unnecessary wait times at intersections
- Reducing fuel wastage as well as exhaust fume emissions
- Improving the quality of life of road users by ensuring smooth flow of traffic

1.4. Expected Outcome

The system to be designed and implemented consists of three major components:

a) A vehicle detection technique with:
    a. ability to detect both moving and stationary vehicles;
    b. accuracy in detection;
    c. low power consumption;
    d. a non-intrusive technology that does not impact negatively on the pavement of existing roads; and
    e. ability to detect vehicles at different lighting conditions.

b) A reliable and efficient communication system to enable communication between devices on the different lanes meeting at the intersection. This will require timely data transmission.

c) Traffic signal control algorithm whose features include:
    a. ability to reduce time delays
    b. an intelligent system to determine the least time-consuming traffic flow; and
    c. a system with extra features such as detecting an ambulance and giving it priority over the other regular vehicles.

1.5. Research Methodology

Throughout this project, secondary research is conducted from academic papers, journals, and relevant articles to inform the decisions taken in terms of the technologies chosen to achieve the objectives of the project. Experiments are conducted during testing to come up with empirical results on some desirable properties of the system.

Chapter 2 : Literature Review

Density-based traffic control at intersections is an area that is widely researched. Several concepts have been suggested by academics as the appropriate methods for managing the ever-growing traffic congestion. Different vehicle detection techniques have been suggested by scholars, and some of them have been implemented in different countries, mostly in developed nations. Furthermore, algorithms for determining wait-time durations have been developed all in an attempt to deal with the problem of traffic congestion at intersections. Also, different data communication techniques have been researched that could enable communication between one lane to another at an intersection.

2.1. Vehicle Detection (Sensors)

Scholars have suggested both invasive and non-invasive techniques for vehicle detection. Invasive techniques refer to those that require drilling of the runways to install the sensors underground while non-invasive techniques are those that do not require road drilling for installation. A survey conducted by The Vehicle Detector Clearinghouse, a project with the US Department of Transportation outlines some trends in vehicle detection and surveillance technologies. In general, most states tend to stick to invasive technologies as they are considered traditional and mature [12]. They are well understood and so mostly employed. This suggests that there has been more research on these traditional methods as compared to newer less invasive technologies.

Specifically, inductive loops are the most commonly used technology. In an inductive loop, the inductive element comprises a wire loop excited with signals of a certain frequency. This is buried under the tarmac. When a vehicle with metallic parts passes over the inductive loop, the

inductance drops and the frequency increases, indicating the presence of a vehicle [12]. However, the wires of the loops are subjected to stresses due to traffic as well as extreme temperatures.

Zarnescu et al suggest the use of magnetic sensors as a relatively new method for vehicle detection [13]. These sensors are buried 10-15 cm below the ground on roads and detect vehicles based on measurement of the disturbance of the earth's magnetic field by metallic parts of vehicles [13]. As compared to the traditional inductive loops, the method of magnetic sensors is presented to be more accurate, cost-effective, and is resistant to traffic stress. The downside with these two methods is that they cannot detect stopped vehicles [12], rendering them less effective for the proposed application. Furthermore, these two techniques obstruct traffic flow during installation, repair, and maintenance [14].

The ultrasonic sensor has also been proposed as a vehicle detection technique. This device emits sound waves at a frequency beyond the human audible range. When these waves are reflected back, the distance between the sensor and the obstacle is determined. For vehicle detection, this distance between the sensor and the obstacle is predetermined, hence if a vehicle becomes an obstacle, the distance obtained is different from the predetermined one, indicating the presence of a vehicle. However, when using the ultrasonic sensors, there is a need to install several along one street, hence inflating the cost of using ultrasonic sensors. In this project, one of the most important factors is to keep the technology low cost hence ultrasonic sensors are not desirable as they are expensive.

A method that is able to detect both moving and stationary vehicles is the use of cameras. Video detection cameras, which have been employed in Texas, have a wide field of view, can be used for counting vehicles, detect vehicle presence, speed, and occupancy and have the

ability to recognize license plates and track vehicles [5]. A review of current traffic congestion management in Sydney, Australia, shows that the use of the InSync technology, which employs internet protocol cameras, has a better performance than inductive loops, and provides the possibility of visual monitoring of runways [4].

Although the use of cameras has often been dismissed as expensive, the study by Fernando et al reveals that cameras cost less when used for several road intersections than induction loops [4]. For a large road network with several intersections, therefore, the use of cameras can be economically justified. Additionally, the use of a pi camera with a raspberry pi is a more economical option since this combination is less expensive than the mainstream surveillance cameras. Vidhya & Banu [15] explain the use of raspberry pi as the image processor for video frames captured from a pi camera. One main advantage of using a pi camera is that it uses the raspberry pi as its image processor. This is advantageous because the raspberry pi directly integrates the open source computer vision (OpenCV) library [15]. OpenCV is an open source image processing library with functions for real-time computer vision and image processing. It is easy to install, quick, open-source, and can be used in real time applications. These features make the raspberry pi and pi camera a good fit for this project.

In choosing a pi camera for this project, an important feature is to have night vision. Datondji et al [16] explain that visible light cameras are only suitable for daylight operations. However, in this project, the camera is to be used for both daylight and night-time operations. The use of infrared cameras [16] is suggested as a solution. Hence, the infrared camera module v2 (Pi NoIR) is selected for this application. This version of the pi camera is capable of daytime vision as well as night vision using infrared illumination.

2.2. Image Processing

Image processing has been done using different techniques. Tiwari & Singh [17] use the OpenCV libraries in Python. They suggest that OpenCV is more efficient in image processing than MATLAB. [17]. Its usefulness in real-time computer vision makes it a better fit for real-time processing of images taken of streets in this project. The authors also suggest the use of a raspberry pi together with the pi camera for image detection and processing, as they are more budget friendly. This corresponds to the aim of this project which is to make traffic control affordable, hence the relevance of using raspberry pi and pi camera.

Image processing is essential in vehicle detection and tracking in any computer vision application. In video processing applications, image processing techniques are applied to individual video frames then the motion of vehicles in the video is realized by comparing sequential video frames [18]. In image processing, images are processed to obtain a wide range of data such as vehicle count, speed, and vehicle type, depending on the relevant data for a specific application. To obtain this data, the images captured by the camera undergo image processing techniques to achieve vehicle detection, segmentation, and tracking.

i) Vehicle detection

In vehicle detection, Justin & Kumar [18] propose a step-by-step technique. This involves image preprocessing steps (resizing, RGB to grayscale conversion, image enhancement using power-law transformation/gamma correction) and Canny edge detection. The captured image is resized because it could be so large that it consumes too much disk space. It is then converted from RGB to grayscale. This reduces on disk space occupied by the image and in addition, Justin & Kumar suggest that grayscale images produce more acceptable results compared to their corresponding RGB images [18]. Image

enhancement sharpens the image features such as contrast and edges hence increasing image quality. Choudekar et al [19] use the power law transformation (gamma correction) as the image enhancement technique where the user chooses the appropriate value of gamma to make fine details of the picture identifiable.

To detect the objects on the image, edge detection is applied to the preprocessed image. Edges are defined as points in a digital image at which brightness or gray levels change suddenly [18]. Different edge detection methods have been proposed, and these are based on different mathematical principles. Gradient-based edge detection [19] is one method proposed, which detects edges by determining the maximum and minimum in the first derivative of the image. One such gradient based technique is the Canny algorithm which is suggested by most scholars, [14], [15], [18], [20]. Different edge detection algorithms have been compared to determine their performance when the resulting pictures from these algorithms are subjected to object counting algorithms [14]. In the test, 5 edge detection algorithms (Boolean, Marr Hildreth, Sobel, Prewitt, and Canny) were used on ten different image samples during a process of counting objects on each image. The result, as shown in table 2.1, shows that Canny Edge detector was the most accurate at 93.47% accuracy, making it very suitable for accurate detection of edges of vehicles in an image.

Canny edge detection method has also been proven to have better advantages than other edge detection methods when other parameters other than accuracy are compared. Therefore, Choukekar & Bhosale [21] have shown that Canny edge detection has better overall performance than other techniques.

Table 2.1: The accuracy of different edge detection techniques

Image no.	Actual no. of objects	Boolean	Marr Hildreth	Sobel	Prewitt	Canny
1	4	2	6	2	2	4
2	3	0	4	1	1	2
3	4	2	3	2	3	4
4	5	2	3	2	3	6
5	5	2	3	3	3	5
6	7	3	5	3	2	6
7	4	1	5	1	1	4
8	5	2	5	3	2	5
9	3	0	3	0	1	2
10	6	4	3	2	3	6
Accuracy %		39.13	84.74	41.30	45.65	93.74

ii) Background/Foreground Segmentation

This involves segmentation of the image scene into individual objects in preparation for tracking. Dangi et. al. suggest static background subtraction as the traditional method for real-time segmentation of images in a video-based system [14]. Background subtraction is a technique used to separate foreground objects from the background. The background subtraction methods that have been suggested are BackgroundSubtractorMOG [22], BackgroundSubtractorMOG2 [23] [24], and BackgroundSubtractorGMG [25]. All of these three methods have been implemented in OpenCV. They model the background pixels by a mixture of a certain number of Gaussian distributions. However, BackgroundSubtractorMOG2 algorithm is preferable because it selects an appropriate number of Gaussian distribution for each pixel as compared to BackgroundSubtractorMOG which assigns a certain number of Gaussian distributions for all pixels in the picture. Hence

BackgroundSubtractorMOG2 is chosen for this application as it provides better adaptability to varying scenes for each pixel in the picture [26].

iii) Tracking

Tracking each individual vehicle helps to update the position of the vehicle in the image. From literature, a number of tracking techniques have been proposed. These include 3D model-based tracking, region-based tracking, active contour-based tracking, and feature-based tracking [27]. 3D-model based tracking involves tracking of objects in an image based on knowledge of its 3D features and geometric trajectories. This is inefficient since it is very difficult to have detailed models of all types of vehicles on the road. Active contour-based tracking relies on the idea of active contours. Active contours are contours that are dynamically changing to find the boundaries of the object. Active contour-based tracking hence retains an approximate boundary of the object and changes this boundary as the object moves within the image. However, this method is unable to segment vehicles that are partially occluded by others. Region-based tracking involves identifying a connected region on an image and tracking it over time. This technique is considered less effective especially under congested traffic conditions as vehicles partially occlude one another instead of being isolated hence making segmentation of individual vehicles difficult [27]. It can, therefore, be deduced that a major problem with most of these techniques is their inability to track objects accurately in the event of partial or full occlusion. Feature-based technique solves this problem by tracking features of the vehicle rather than the whole vehicle, hence even when the vehicle is partially occluded, some of its features remain visible [27]. Justin & Kumar [18] suggest Kalman filter as a feature-based technique for tracking detected

vehicles in images. Kalman filter is an estimation algorithm that predicts the next state of a continuously changing phenomenon using uncertain information. In the case of vehicle tracking, only one Kalman filter is applied to each vehicle [18] hence the number of Kalman filters applied to each video frame depends on the number of detected vehicles.

2.3. Pi to Pi Communication

The image processing steps described in the preceding sections are done for each lane in an intersection. There is, therefore, need for a communication system that will enable collation of data from the different lanes, comparison, and use of this data to control traffic lights.

Kotwal et al [5] explore the various communication technologies that could be employed in data transfer between two devices. First, they mention the use of serial communication over copper twisted pair cables. The downside of this is that as the distance of transmission increases, the signal strength is reduced and there is limited bandwidth. To improve on bandwidth, they suggest the use of Ethernet communication over fiber optical cable [5]. However, optical fiber is quite expensive hence not suitable for an application that seeks to reduce costs as in this project. Also, optical fiber and twisted pair copper cables require digging into the road pavement hence it is an invasive technology.

Wireless technologies (unguided media) are advantageous because they can have a wider coverage area, allow for remote signal monitoring and reduced operation and maintenance cost [5]. However, they only allow for limited transmission distance and bandwidth. The distance for transmission in this project is about 10m between two devices, and the data being transmitted is only the number and speed of vehicles on a street, hence wireless

communication can be used. Furthermore, wireless technologies do not require digging through the road during installation hence are very suitable for this application.

Yanbing [28] proposes the use of nRF2401 transceiver as the low power wireless communication system that helps achieve real-time data transfer at low cost, low power, and dependable performance [29]. nRF2401 is a radio transceiver that requires very low power during transmission. In fact, the input current is less than that required by one LED. Bluetooth is also another common wireless communication method. It is a protocol which operates in the 2.4GHz ISM license-free band and uses the UART interface protocol. It has a 10m range. Wi-Fi also uses the 2.4GHz band, has a maximum range of 100m and can transfer data at very high speeds. However, compared to the other methods, Wi-Fi is more expensive. Based on the comparisons, the nRF24L01 transceiver is chosen for its low power consumption, affordability, and acceptable range of transmission.

In this chapter, different vehicle detection technologies have been discussed and their performance assessed based on information from some related work. From literature, the pi camera has been determined to be the most suitable since it does not require drilling through the road surface during installation or maintenance. Image processing steps have also been discussed and techniques suitable for this application decided. Finally, the means of communication between two lanes at an intersection is examined.

Chapter 3 : Design and Implementation

In this chapter, the design and implementation of the project are discussed. First, the design decisions are explained by using Pugh matrices to compare different technologies. The system architecture is then explained and components and software used are outlined. Finally, the implementation of the components of the system is then explained. In this regard, affordability is the major factor considered when designing this system, and in addition, each of the different components of this system has requirement specifications.

The system constitutes the following components:

1. The vehicle detection component
2. The image processing component
3. The pi-to-pi communication component
4. The traffic signal scheduling algorithm section
5. The traffic light section

3.1. Design Decisions and Pugh Matrices

In choosing the technologies to use for the different components of the system, some design decisions were considered for the different components of the system.

3.1.1. The Vehicle Detection Component

There are various options for sensors that could be used for vehicle detection. These include the inductive loops, ultrasonic sensors, magnetic sensors, surveillance cameras, and pi cameras. The Pugh matrix in table 3.1 shows the process of arriving at the design decision. The most

important factors for choosing a vehicle detection technique are cost, accuracy, ability to detect both moving and stationary vehicles and ease of maintenance.

Table 3.1: Pugh matrix to choose the best vehicle detection technique

#	Criteria	Weight	Inductive loops(Baseline)	Ultrasonic sensor	Magnetic sensor	Surveillance camera	Pi camera
1	Cost	3	0	-1	-1	-1	1
2	Accuracy	2	0	1	1	1	1
3	Ability to detect both moving and stationary vehicles	3	0	1	1	1	1
4	Ease of maintenance	2	0	1	0	1	1
Total			0	4	2	4	10

From the Pugh matrix, the best sensor to use is the pi camera, mainly due to its cost-effectiveness as compared to other sensors.

The camera to be used should satisfy the following criteria:

i. Versatility- the ability to record outstanding images in various lighting conditions and weather conditions such as extreme fog
ii. Night vision and ability to take quality pictures under low light conditions
iii. High still resolution to ensure high quality of images.
iv. Wide field of view (both horizontal and vertical)

The Raspberry Pi camera v2 has a field of view of 62.2 degrees

The raspberry pi camera options available are:

i. Pi camera module v1

ii. Pi camera module v2

The pi camera version 2 is an upgrade of version 1, having a wider field of view, higher resolution as well as night vision capabilities. Table 3.2 shows the differences in specifications.

Table 3.2: The differences in characteristics of the two pi camera modules

Characteristics	**Camera module v1**	**Camera Module v2**
Horizontal field of view	53.50 +/- 0.13 degrees	62.2 degrees
Vertical field of view	41.41 +/- 0.11 degrees	48.8 degrees
Net price	$25	$25
Still resolution	5 Megapixels	8 Megapixels
Night vision	no	yes

Table 3.3 shows the Pugh matrix used to choose the best pi camera to be used.

Table 3.3: Pugh matrix for evaluating the best pi camera to be used.

Criteria #	**Criteria**	**Weight**	**Pi camera module v1**	**Pi camera module v2**
1	Sensor resolution	2	0	1
2	Night vision	3	0	1
3	Field of view(vertical and horizontal)	3	0	1
4	Still resolution	2	0	1
Total			0	10

From the Pugh matrix, the module v2 is picked for its desirable characteristics of night vision, the wide field of view, and its resolution. The Raspberry pi NoIR camera board v2 was therefore used in this project.

3.1.2. The Image Processing Component

The choice of the image processing component depends on the choice of the vehicle detection component, hence the raspberry pi is chosen as it is compatible for use with the pi camera. The image processing compartment consists of an algorithm running on a raspberry pi attached to every camera for image processing to extract important information (number of vehicles on the lane). The raspberry pi is a single board microcomputer that uses a Linux-based operating system and external peripherals such as keyboard, mouse, and monitor. It can also be accessed remotely using another computer's monitor via SSH or VNC.

3.1.3. The Pi to Pi Communication Component

A communication system to enable data communication between the raspberry pi's on the roads at the intersection. Table 3.4 shows a comparison of the different possible technologies for pi-to-pi data communication.

Table 3.4: A comparison among the different possibilities for pi-to-pi data communication.

Characteristics	**Wifi**	**Bluetooth**	**NRF24L01**	**433 MHz Tranceiver**
Working frequency	2.4GHz	2.4GHz	2.4GHz	433.4MHz- 473.0MHz
Communication distance/range	400m	10m	10m	1000m
Interface protocol	UART,SPI	UART	SPI	UART

In this project, the nRF24L01 transceiver was chosen to be used for pi to pi communication because it is cheaper than the Bluetooth, Wi-Fi, and the 433MHz transceiver.

3.2. System Architecture

In the system, there is a pi camera per lane on the intersection, and each camera is attached to a raspberry pi. The cameras record an image of the lane traffic, and this image is transmitted to the raspberry pi and processed. The raspberry pi performs image processing on each image, periodically determining the number of vehicles at the intersection on each street. Each raspberry pi transmits its information about the number of vehicles to a central raspberry pi using nLRF2401 transceivers connected to the raspberry pi's. The receiving raspberry pi has the algorithm that compares the number of vehicles on the lanes, and based on this, assigns the amount of green light time for the lanes. The raspberry pi then sends the instruction to the traffic lights which respond accordingly, ensuring an optimum amount of wait time for vehicles at the intersection. This process is summarized in figure 3.1.

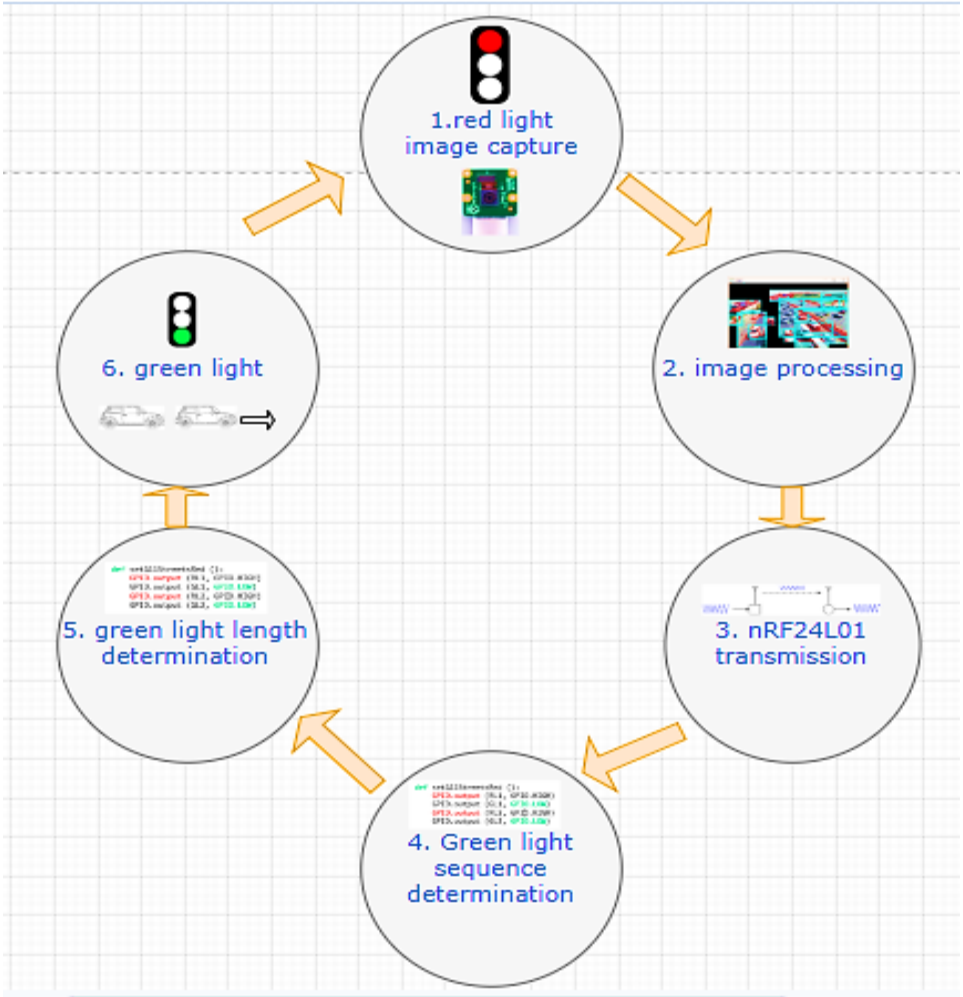

Figure 3.1: The cycle of what happens for every lane in the intersection

Figure 3.2 shows the working of the system with two lanes. The description of the arrows are as follows:

A- Camera transmitting the image to the raspberry pi for processing.

B- The raspberry pi passing on the data on the number of vehicles to the transceiver.

C- Pi-to-pi communication via the transceiver module.

D- Transceiver passing on received data to the raspberry pi.

E- The central raspberry pi sending commands to traffic lights.

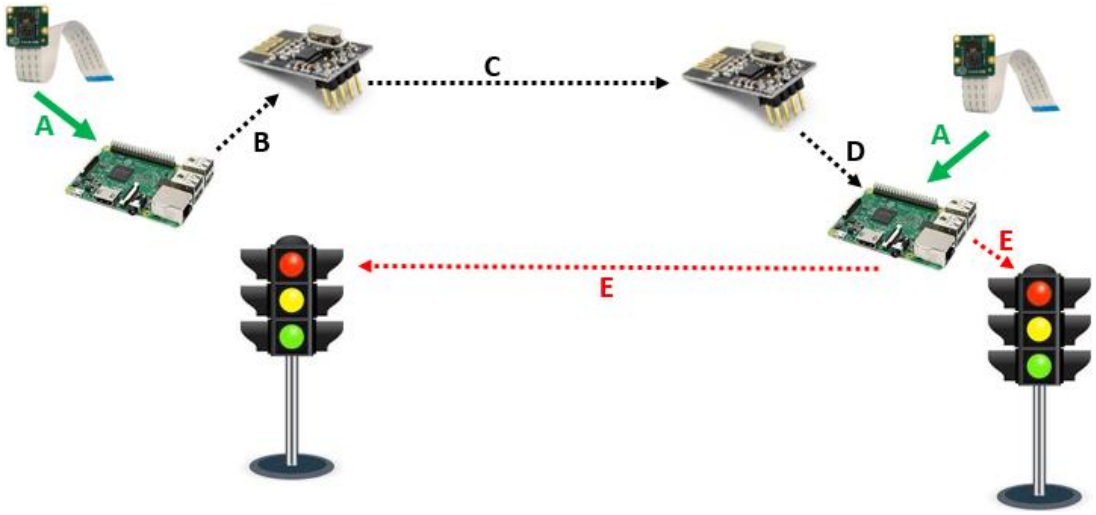

Figure 3.2: Schematic of the system showing major components and communication lines (arrows)

3.3. Description of Components and Software Used

3.3.1. Raspberry Pi 3

Raspberry pi is a small chip-like single board computer. There are various model of Raspberry pi available in the market such as the raspberry pi1 model B, raspberry pi1 model B+, raspberry pi2, raspberry pi3 model B. These differ in memory capacity and hardware features. Raspberry pi3, which is used in this project, has inbuilt Bluetooth and Wi-Fi modules whereas in previous versions these modules were not available on board. It has 1.2 GHz 64-bit quad core ARMv8 CPU with 1 GB of RAM [4]. Additional characteristics include:

- 2.5A power adapter
- 8GB SD card preinstalled with the new out of the box (NOOBS) software. NOOBS' advantage is that it allows you to remove, install and reinstall Linux operating system

(OS) via an easy to use graphical user interface (GUI). The OS used in this project is Raspbian – an open source Debian Linux OS.

- 4-2.0 USB ports- these are used for connecting peripheral devices such as mouse and keyboard to the raspberry pi.
- 40 pin GPIO header- these pins are used to control the external devices connected to the raspberry pi via these pins. In this project, these pins are used to control traffic lights and signals sent to and from the transceiver.
- Camera serial interface (CSI) port- this is the port through which the pi camera is connected to the raspberry pi.
- Display serial interface (DSI) port - for connecting an LCD to the raspberry pi
- Ethernet port- this is used to connect the raspberry pi to the PC to allow network sharing between the PC and the raspberry pi.
- An HDMI output- this is for connecting the raspberry pi to a monitor through a HDMI cable.
- A Micro USB port for powering.
- Built-in chip antenna connecting to built-in Wi-Fi
- SD slot for SD card

Raspberry pi pins have two numbering conventions: pin numbers (BOARD) and Broadcom (BCM) GPIO numbers. Any of these conventions can be used but it is important not to mix them up. For the applications in this project, the BCM numbering system has been used.

3.3.2. Pi Camera

The camera is used to take the pictures of the lanes. These images are sent to the raspberry pi to perform image processing for counting vehicles. The pi camera version 2 is used, which has night vision capabilities.

3.3.3. Python Software

The algorithms and code blocks in this project are written in the Python programming language.

3.3.4. OpenCV

OpenCV stands for Open Source Computer Vision. It is a library of programming function mainly for real-time computer vision. It has over 2500 optimized algorithms [8] which can be used for facial recognition, object detection, and other computer vision applications. OpenCV has interfaces for Python, C++, Java, and MATLAB, and supports different operating systems- Windows, Linux, Mac.

3.3.5. nRF24L01 Transceiver

This is a single chip 2.4GHz transceiver from the Nordic company in Norway. It operates in the 2.4GHz frequency band. This is the Industrial, Scientific, and Medical (ISM) frequency band reserved for unlicensed low power devices. The data transfer rate (DTR) can either be 250kbps, 1Mbps, or 2Mbps, and it uses Gaussian Frequency Shift Keying (GFSK) modulation. The transceiver module's operating voltage is between 1.9V to 3.6V and the nominal current is 50mA, the maximum operating current is 250mA. The version of the transceiver module with the on-board antenna can communicate over a maximum distance of 100meters.

This transceiver module uses the Serial Peripheral Interface (SPI) communication protocol. The output power and communication channel and data rate are configured by software through

the SPI interface. It is cheaper than Bluetooth, has no complicated communication agreement, free to communicate within products of a type [28].

3.4. Implementation

The implementation of the proposed design was done at three levels: algorithm implementation, image capture and processing, and data communication implementation

3.4.1. Algorithm Implementation.

To solve the problem of unnecessarily long wait times of vehicles at traffic intersections, an algorithm was implemented based on an intelligent system proposed by Zhou et al in [30]. They proposed the factors that determine the green light determination. These are: traffic volume (number of vehicles on a lane), waiting time (amount of time vehicles wait on a lane before getting green light time), blank cases(length of spaces between vehicles on a lane), special circumstances (such as presence of an ambulance or fire trucks), and hunger level(the extent to which a lane has not received green light.). The first step in designing the algorithm is assuming an intersection as shown in figure 3.3. The arrows indicate the direction of flow of traffic, and the lanes are numbered 1-8.

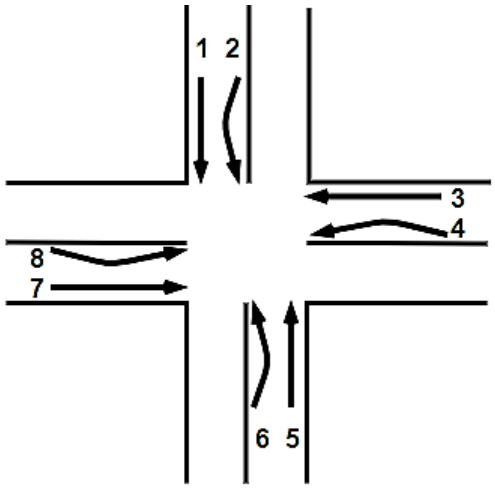

Figure 3.3: An outline of the intersection being considered

The intersection consists of four two-way streets, each way having two lanes. Based on this outline of lanes, there are 12 possible cases of green lights, as shown in figure 3.4.

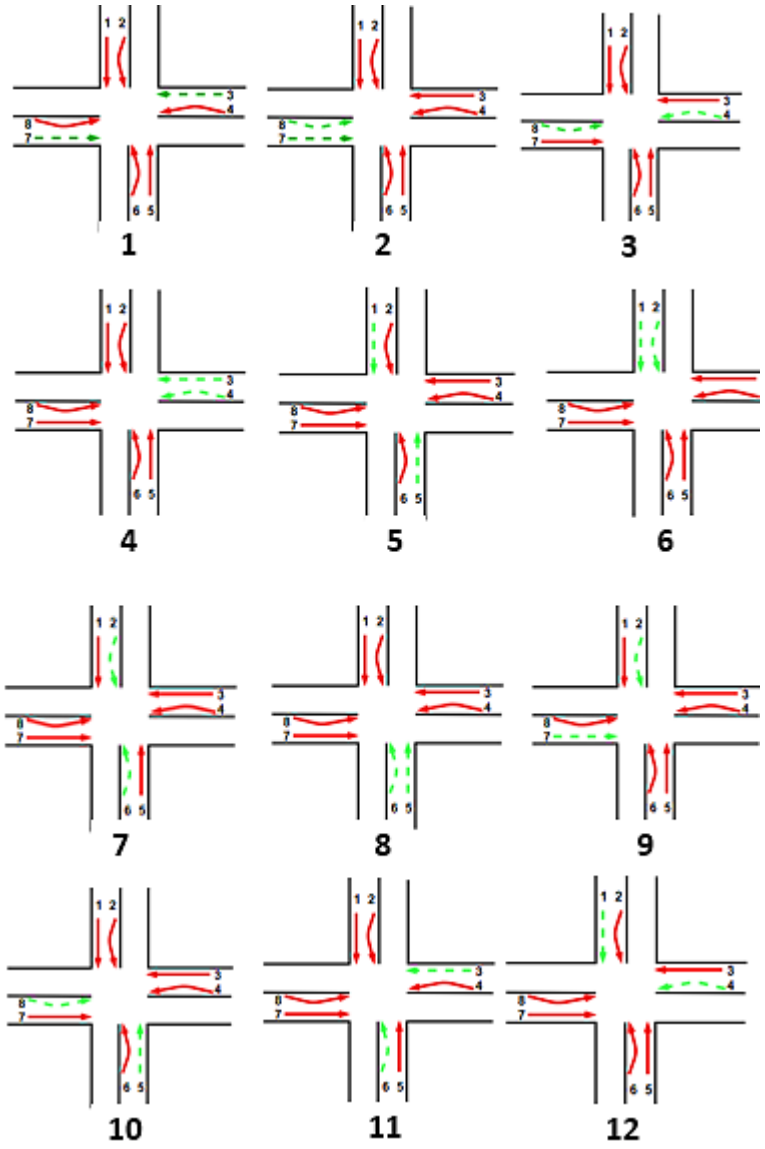

Figure 3.4: The twelve possible cases of green light on an 8 lane intersection

In the figure 3.4, the dotted arrows show the lanes with green light, and the continuous arrows show the lanes where the traffic light is red and vehicles are not moving.

To formulate the algorithm, the following assumptions were made:

a. The intersection has eight lanes
b. The road operates on a right-drive system
c. All the vehicle move at a constant average speed

First, the algorithm determines the green light sequence by determining the next case to have green light, then it determines the green light length for that case. The next case to have the green light is determined by several factors: traffic volume, hunger level, and blank cases. The different factors are explained below:

3.4.1.1.Traffic Volume

Traffic volume of a case is computed as the fraction of the total number of vehicles on all lanes that is the number of vehicles on the lanes in that case.

$$TV_{case\ i} = \frac{TV_{lane\ a} + TV_{lane\ b}}{\sum_{j=1}^{8} TV_{lane\ j}}\ , \quad 1 < i < 12$$

Where $TV_{lane\ a}\ and\ TV_{lane\ b}$ are the number of vehicles in the two lanes associated with case i. The value of i is between 1 and 12 because there are 12 possible cases as in figure 3.4. The values of $TV_{lane\ a}\ and\ TV_{lane\ b}$ are determined by image processing.

3.4.1.2.Hunger Level

The hunger level of a case refers to the extent to which that case has not received green light time. This is calculated to ensure fairness in the allocation of green light periods. This is because, for a case where there are only very few vehicles on certain lanes, they wait for so long without getting green light if only traffic volume is considered.

$$HL_{case\ i} = 1 - \frac{N_i}{\sum_{i=1}^{12} N_i}$$

Where N_i is the number of times $case\ i$ has received green light and $\sum_{i=1}^{12} N_i$ is the total number of times all the cases have received green light.

3.4.1.3.Blank Cases

The term blank cases refers to the occurrence of spacing between vehicles on a lane. The importance of this is to ensure that as traffic flows on a lane with green light, the light turns red when there is a blank case to ensure that wait time for vehicles on other lanes is minimized. In so doing, there is no situation where a lane has green light yet there are no vehicles passing. This parameter is however not applicable in this design because pictures are taken of lanes at the time when the vehicles are at rest. Hence there is a minimal possibility that a vehicle waiting on a lane will stop a long distance from the one ahead of it.

Although [30] suggests that waiting time and special circumstances are also important factors, the actual implementation of this work does not include those.

The Python code for the determination of these factors is included in Appendix IV

3.4.1.4.Determining the Green Light Priority

The green light priority is proportional to each of the factors mentioned above. The equation below shows a relation between the green light priority and the different factors.

$$GLD_{case\ i} = a_1 BC_{case\ i} + a_2 HL_{case\ i} + a_3 TV_{case\ i}$$

Where $a_1,\ a_2,\ and\ \ a_3$ are coefficients of the factors that determine how important the factors are in determining the green light priority of a case. According to [30], the factors in order of importance are: blank cases, hunger level, and traffic volume. This same order of preference is applied in this design, and the higher the order of preference, the higher the value of the coefficient. Hence the values of the coefficients are:

$$a_1 = 5; \quad BC - blank\ cases$$
$$a_2 = 4; \quad HL - hunger\ level$$
$$a_3 = 3; \quad TV - traffic\ volume$$

The Python code for the determination of green light priority is included in Appendix IV

3.4.1.5.Determining the Green Light Length

Having determined the case to get the next green light, the length of time the green light will be on for this case is then computed. The determination of green light length depends on the number of vehicles on the lanes associated with that case.

The amount of time one car takes to move across an intersection is calculated as:

$$T_{next\ 1} = \frac{L_{intersection}}{car\ speed}$$

Hence the total time taken for all cars in a particular case to move across an intersection is:

$$T_{next} = T_{next\ 1} * \max(TV_{lane\ a}, TV_{lane\ b})$$

The green light is then on for this amount of time. However, there is a set maximum time T_{max} during which green light can be assigned to a case. If the amount of time T_{next} is greater than T_{max}, then that case is assigned T_{max}. This is because a case might have a very large number of vehicles such that the vehicles in other lanes wait for too long before getting green light, hence defeating the purpose of this project which is to minimize wait times of vehicles.

The actual implementation of this algorithm is done using the Python programming language and the code is provided in Appendix IV. The high-level process of the algorithm of green light sequence and green light length determination is shown in the flow chart in figure 3.5.

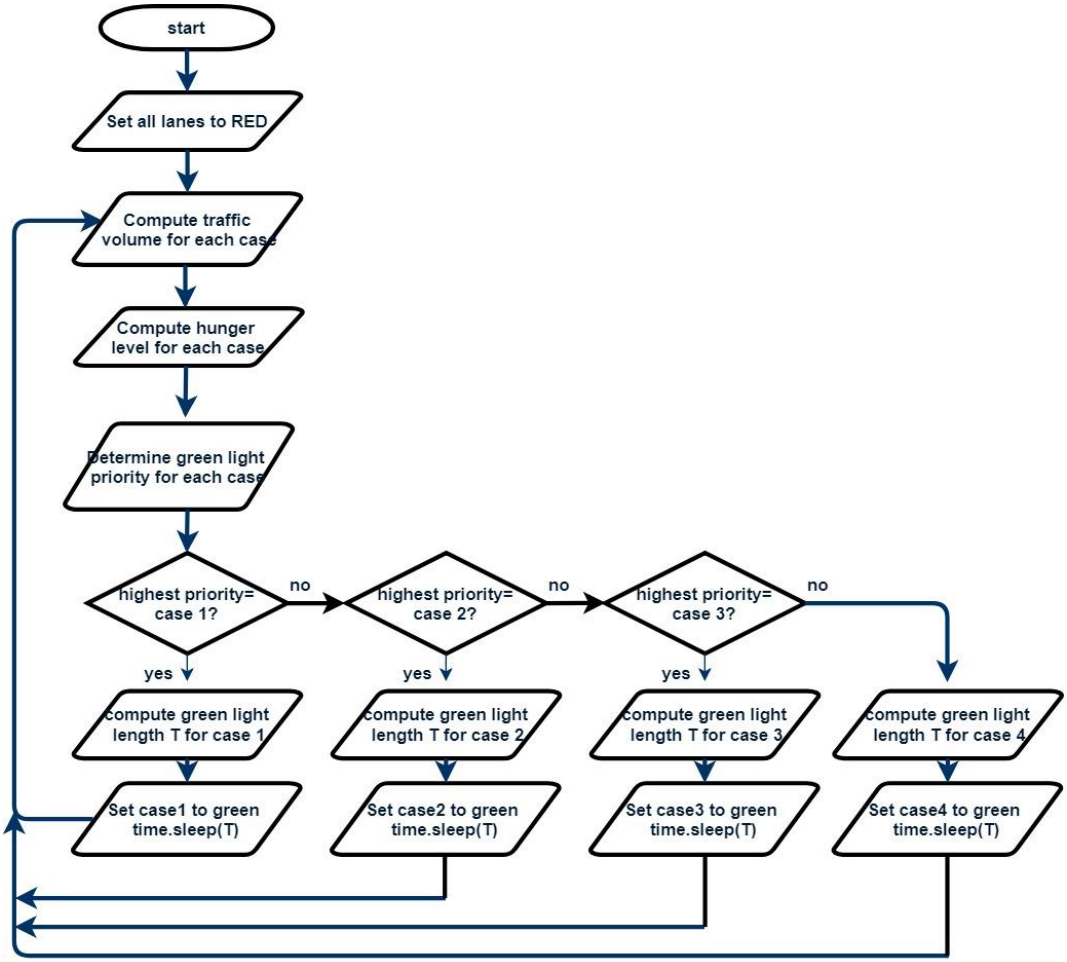

Figure 3.5: A flow chart showing the high-level steps in green light sequence and green light length determination

3.4.2. Image Capture

A raspberry pi camera was used to capture the images that will then be processed to obtain the inputs required. The camera chosen for this purpose has an infrared capability for night vision. The factors to be considered when positioning the camera for image capture include the camera's field of view. In this case, the horizontal field of view is 62^{o} which will inform the positioning of the camera during testing. The image is captured at the point when vehicles in most of the lanes are at rest, and only vehicles in the lanes associated with the case with green light are moving.

To take pictures using the pi camera, the camera module was first connected to the raspberry pi camera serial interface. The camera was mounted onto the interface such that the silver stripes of the ribbon face the side the HDMI port, as in figure 3.6.

Figure 3.6: Positioning the camera module on the raspberry pi

The raspberry pi was then configured by ensuring the ‘camera’ option in the ‘interfaces’ tab of the ‘Raspberry Pi Configuration’ is enabled, as shown in figure 3.7. The ‘Raspberry Pi Configuration’ can be accessed from the preference section of the raspberry pi. The raspberry pi was then rebooted to ensure the new configuration is active. To take a picture, the python code in Appendix V was run.

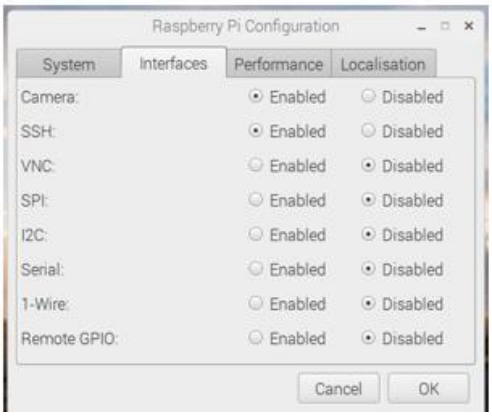

Figure 3.7: The raspberry pi desktop steps to enabling the camera option

Due to the night vision requirement, the camera was fitted with infrared LEDs to emit infrared which helps the camera achieve night vision. The setup of the camera with infrared LEDs mounted on the raspberry pi is shown in figure 3.8.

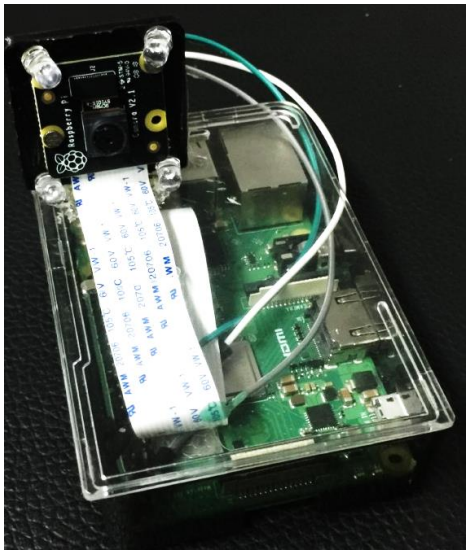

Figure 3.8: The setup of the pi camera fitted with infrared LEDs

3.4.3. Image Processing

The captured image was taken through a series of image processing stages to determine the parameters of interest such as the number of vehicles. The step-by-step process is outlined in the flow chart in figure 3.9.

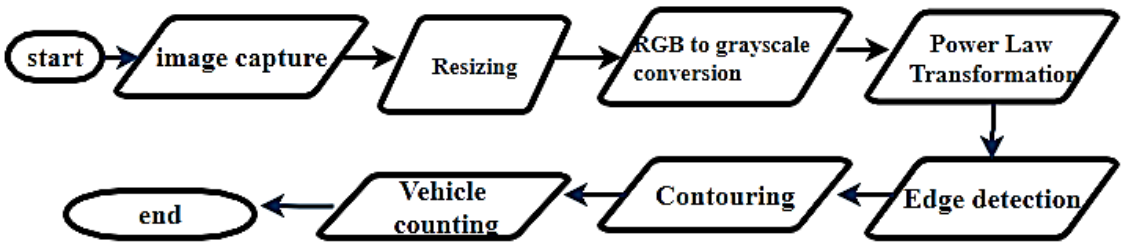

Figure 3.9: A flow chart summarizing the image processing steps

i. Resizing

Resizing was done for two reasons: the image size could be too large to fit in the viewing window, and the image size could be consuming too large disk space. In resizing, new height and width are defined by scaling the original image to a certain scaling factor. Listing 1 shows the code used in resizing.

```
height, width, depth = oriimg.shape
imgScale = 5
newX,newY = oriimg.shape[1]*imgScale,
oriimg.shape[0]*imgScale
sizedimg = cv2.resize(oriimg,(int(newX),int(newY)))
```

Listing 1

ii. RGB to Grayscale Conversion

The use of grayscale images was important to obtain more acceptable image processing results. The cv2. cvtColor method is used as in the line of code in listing 2.

```
gray_image = cv2.cvtColor(sizedimg, cv2.COLOR_BGR2GRAY)
```

Listing 2

iii. Image Enhancement: Power Law Transformation

Image enhancement was done to sharpen the image features, hence increasing quality. The power law transformation is used where a gamma value is chosen. Adjusting the gamma value adjusts the sharpness of the image. For this case, a gamma value of 1.2 is used which provides sufficient image sharpness. Listing 3 shows the code used.

```
im_power_law_transformation = cv2.pow(gray_image1,1.2)
```

Listing 3

iv. Canny Edge Detection

Edge detection was done to find points in the image where the color changes significantly. The Canny edge detection uses hysteresis thresholding in which a minimum threshold and maximum threshold are defined. The pixel values above the maximum threshold are definite edges, those below the threshold are not part of the edge, and those in between can be part of the edge or not depending on whether they are connected to an edge pixel or not. In the code in listing 4, the first argument is the image, the second is the minimum threshold, and the third is the maximum threshold.

```
edges = cv2.Canny(img,100,200)
```

Listing 4

v. Contours

Contours are continuous pixels that are used to crop objects out of the image. This is used to crop the vehicles out of an image before counting them.

vi. Vehicle Counting

The longest continuous contours are counted as these are the contours outlining the vehicles in the image. Rectangles were then traced around these vehicles to show their position. The Python code for this implementation is shown in Appendix III.

3.4.4. Pi to Pi Communication

Communication between two raspberry pi's is established using nRF24L01 transceiver whose pin-out is shown in figure 3.10. Table 3.5 explains the functions of the pins.

Figure 3.10: The pin-out and top view of the nRF24L01 transceiver module

Table 3.5: nRF24L01 pin functions and connections to the raspberry pi

nRF24L01			**Pi pin (BCM notation)**
Pin	**Abbreviation**	**Function**	
Power	VCC	For powering the transceiver module using 3.3V	3.3V
Ground	GND	For connecting to the ground of the system	Ground
Chip Select Not	CSN	To be kept high always to ensure the SPI is not disabled	GPIO 8
Chip Enable	CE	For enabling SPI communication	GPIO 17
Master Out Slave In	MOSI	Connected to the MOSI pin of the microcontroller unit(MCU) for the module to receive data from the MCU to the nrf24L01	GPIO 10
Master In Slave Out	MISO	Connected to the MISO pin of the MCU for the module to send data to the MCU from the nrf24L01	GPIO 9
Serial Clock	SCK	Provides the clock pulse for SPI functionality	GPIO 11

The sending and receiving raspberry pi's are each connected to a transceiver module, both with the same pin-out as outlined on the table. Figure 3.11 shows a schematic of the connection.

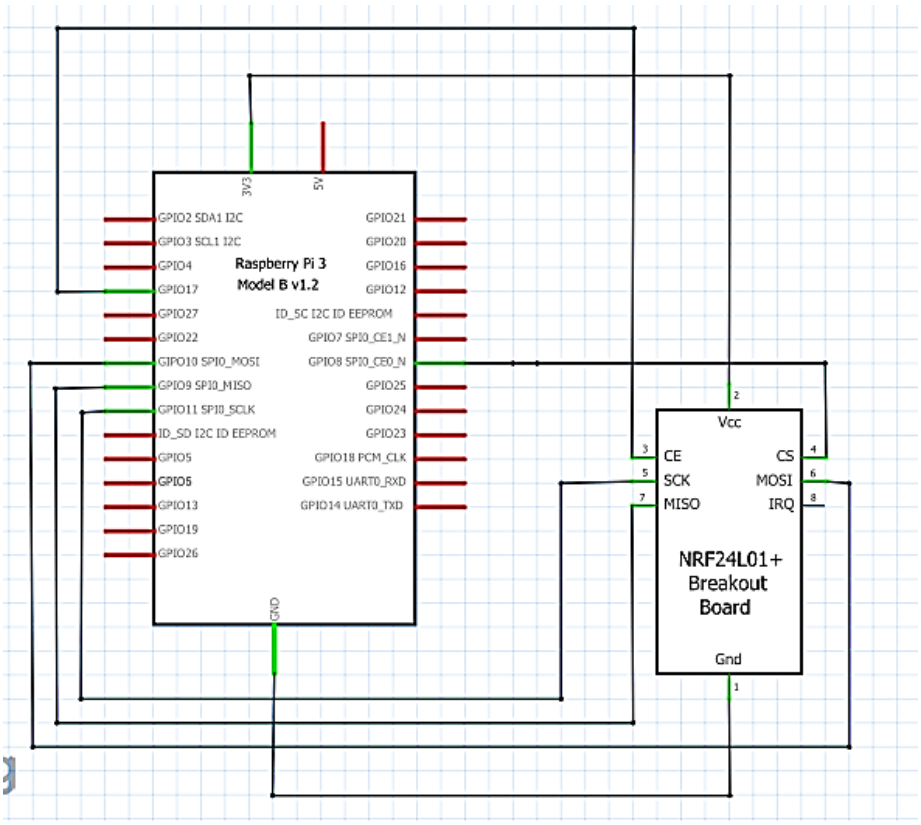

Figure 3.11: Schematic of an nRF24L01 transceiver connection to the raspberry pi

Chapter 4 : Testing, Results, and Analysis

In this chapter, the testing of the various components of the system is done, and the results analyzed.

4.1. Green Light Sequence and Length Determination

The green light sequence and length determination algorithm was tested using only four lanes out of eight. This is because of the limited number of raspberry pi digital pins, each of which is connected to a specific color of traffic lamp on a lane. For two colors (red and green; two digital pins of the raspberry pi were required per lane). For scalability, the use of GPIO port expander chips can be employed. The four lanes used are highlighted in figure 4.1.

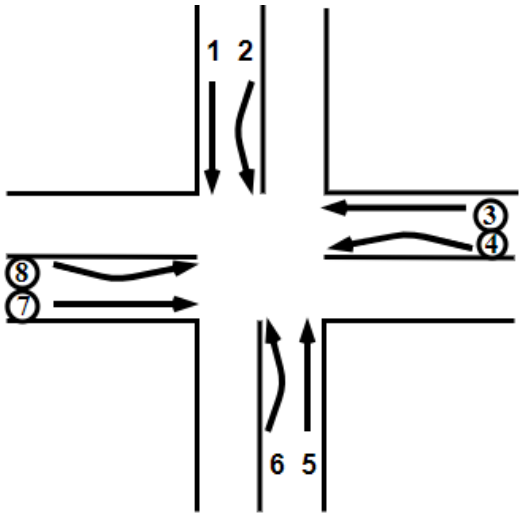

Figure 4.1: Lanes whose numbers are circled are used for testing

These lanes are sufficient for testing case 1 to 4 from the cases shown in figure 3.4. A case is an instance of green light, where two lanes out of the eight receive green light while the rest of the lanes' light is red. There are twelve possible cases as illustrated in the previous chapter in figure 3.4. The cases used and their corresponding lanes are shown in table 4.1.

Table 4.1: The lanes related to cases 1 to 4

Case	Lanes
1	3&7
2	7&8
3	4&8
4	3&4

During this test, the values of traffic volume for each lane were generated using a randomizer in python. These values were fed into the green light sequence and length determination algorithm to determine the next case to receive green light and the length of green light. The instruction was then sent to the traffic lights (represented by LEDs in figure 4.2.) to turn the green lights of the appropriate lane on for a specific amount of time. The circuit diagram used for testing is also shown in figure 4.2 and 4.3.

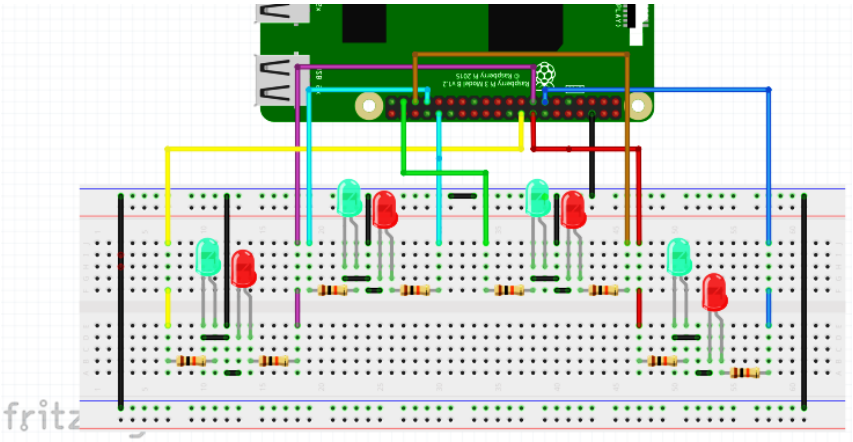

Figure 4.2: Circuit diagram for testing the green light sequence and length determination algorithm

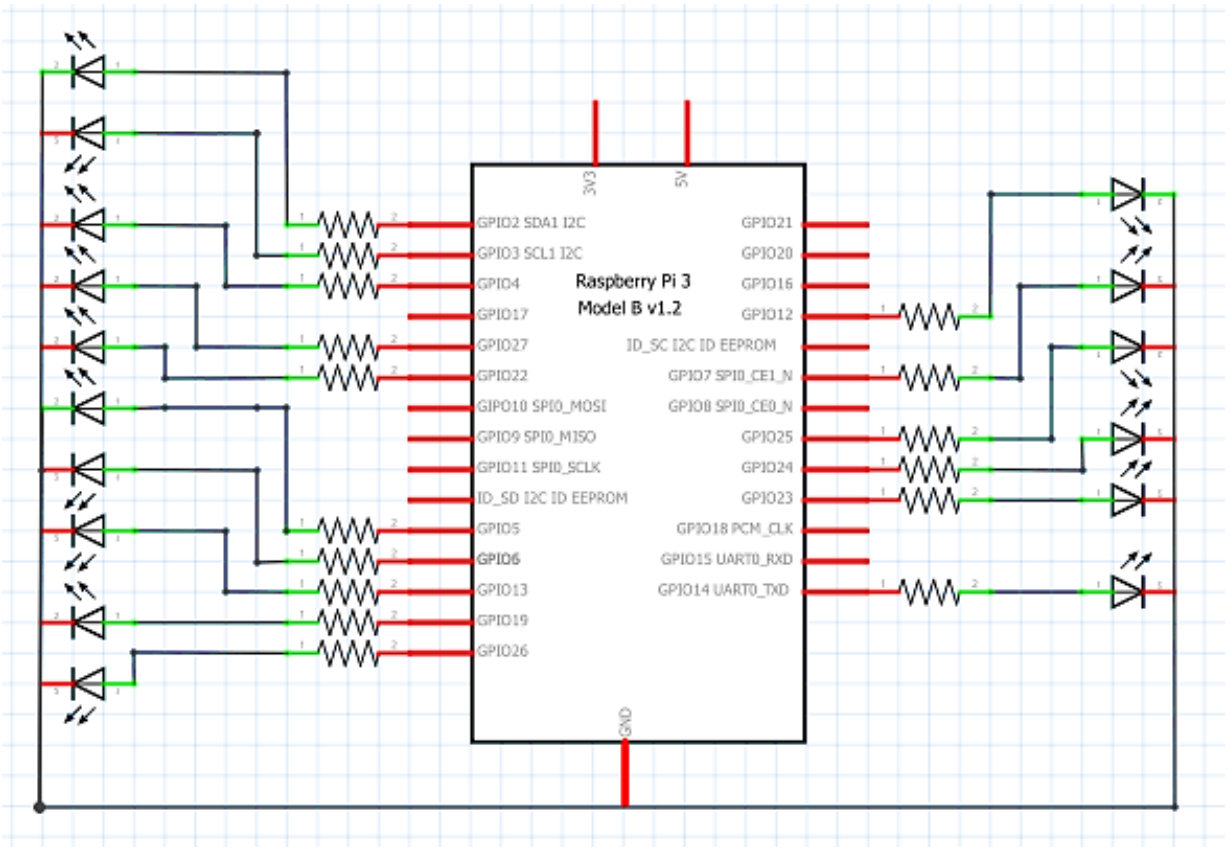

Figure 4.3: Schematic for testing the green light sequence and length determination algorithm

From this circuit diagram, the setup in figure 4.4 was built to test the working of the algorithm.

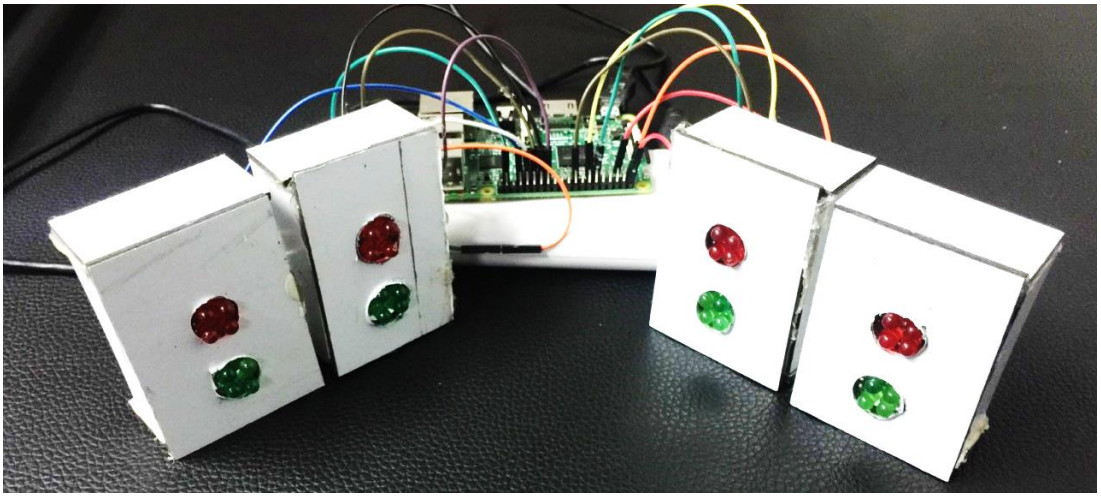

Figure 4.4: The setup of the traffic light system of four lanes at an intersection

The test was run 5 times and the results obtained from the first two test runs are as shown in the Python output in table 4.2 and 4.3.

Table 4.2: Results of a test run on the green light sequence and length determination algorithm

Test run 1				
	Case 1	**Case 2**	**Case 3**	**Case 4**
TV	0.2549	0.1765	0.2549	0.3333
HL	0.6	0.9	0.9	0.9
GLD	6.4279	6.1600	6.5722	8.1419
Maximum GLD is 8.14192403258522 Maximum GLD is of case4 Max traffic volume of case4 is 10 Green light length is 25.0				

On the table, traffic volume (TV), hunger level (HL), and green light determination (GLD) are shown for case 1 to 4. From the results, case 4 has the highest traffic volume and is among the cases with the highest hunger level. Hence, it has the highest priority for green light. The results also shows that the lane with the maximum number of vehicles in case 4 has 10 vehicles, hence green light length is calculated based on this to be 25.

Table 4.3: Results of a second test run on the green light sequence and length determination algorithm

Test run 2				
	Case 1	**Case 2**	**Case 3**	**Case 4**
TV	0.2	0.2	0.16	0.16
HL	0.623	0.769	0.769	0.769
GLD	5.591	7.379	6.354	6.755
Maximum GLD is 7.379 Maximum GLD is of case2 Max traffic volume of case4 is 6 Green light length is 15.0				

In test run 2, the case with the highest priority is again determined from the traffic volume and hunger level data. This time, case 2 has the highest priority. However, the lane with

the highest traffic volume in case 2 has 6 cars as compared to the 10 in test run 1 (table 4.2), hence the green light length assigned is 15 which is less than that assigned in test run1.

The traffic lights on the lanes are also seen to turn green for the determined period of time, showing that the algorithm works as desired.

4.2. Test Results on Vehicle Counting

In the vehicle counting algorithm, the image taken by the pi camera goes through a series of steps before the vehicles in the image are finally identified. These include resizing, RGB- grayscale conversion, edge detection, contouring and finally vehicle counting where rectangles are drawn around objects that are determined by the algorithm to be vehicles.

When the images taken by the pi camera are passed through the algorithm, the effectiveness of the algorithm in giving the correct number of vehicles is low. Even from observation, pictures taken using the pi camera are blurry. The image of a lane taken by a pi camera is less clear than that taken by a phone camera from the same distance.

(a)

(b)

Figure 4.5: Pictures taken from the same distance (10m) from the lane using a pi camera (a) and phone camera (b).

Objects in image (a) are indistinguishable by the algorithm. The image processing algorithm uses the contouring concept to distinguish objects in the image. Contours are

neighboring pixels where the color intensity in the image changes sharply. In the case of a blurry image as in the pi camera image (a) in figure 4.5, the color intensities do not change as sharply as they do in a clearer image as in the phone camera image (b).

The blurry effect of an image can either be caused by low resolution or low shutter speed of the camera. In the case of the pi camera, the camera resolution is 8megapixels, and the shutter speed is 2 seconds. The two seconds is long enough to cause a blurry effect on the image.

The images with higher resolution from the phone camera are therefore used for testing the image processing algorithm. From the vehicle counting algorithm, images (a) to (d) in figure 4.6 show the outputs of counting vehicles in an image**.**

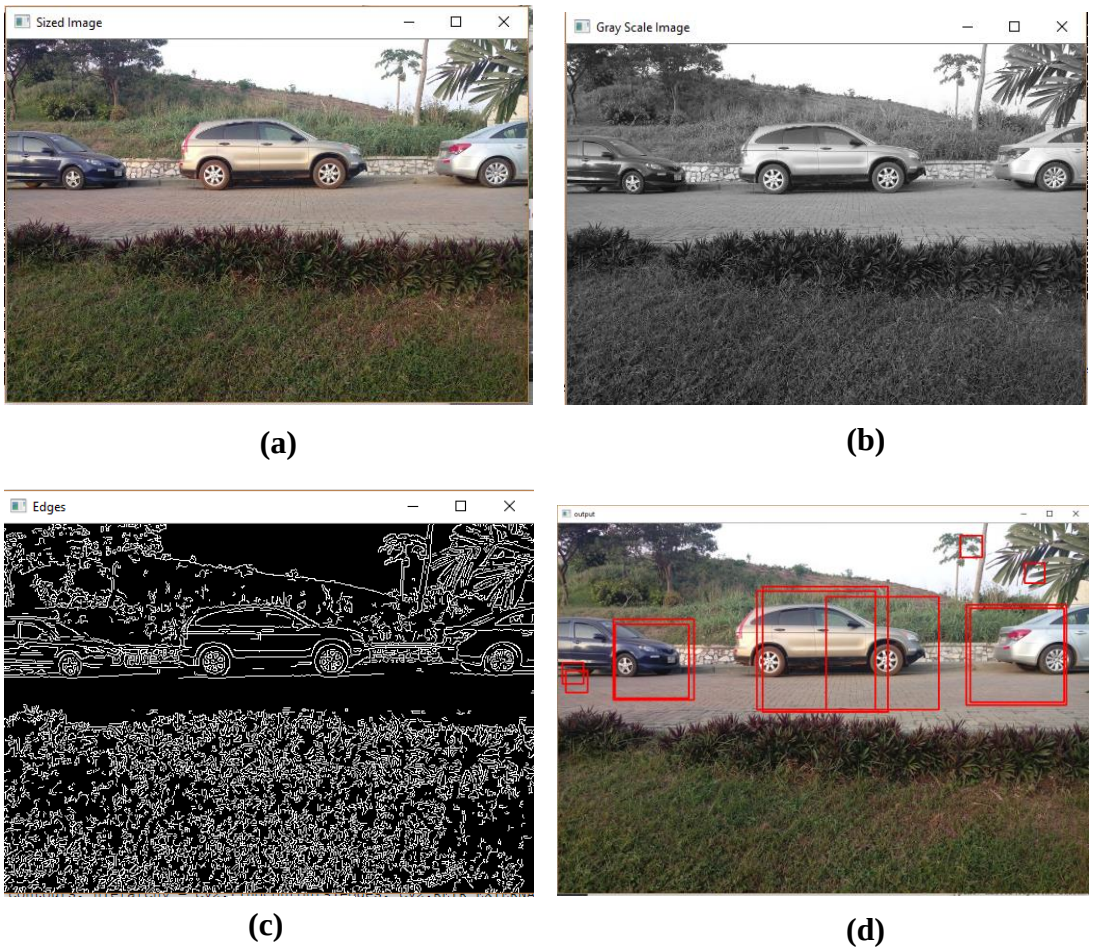

Figure 4.6: Output stages of images taken through image processing during vehicle counting. In (a), the image output after it has been resized is shown. Image (b) is the grayscale image of the resized image is shown. Image (c) shows the results of edge detection and (d) shows the rectangles superimposed on the vehicles on the resized image.

4.3. Camera Calibration

Camera calibration is the process of determining the best intrinsic and extrinsic parameters of the camera. Knowing the intrinsic characteristic of the camera (field of view), it is possible to estimate the extrinsic characteristics (positioning) of the camera through a series of computations and experiments.

The field of view of the noir pi camera is 62.2^{o} which limits the positioning of the camera relative to the intersection and affects the accuracy of the vehicle count on a lane. If the camera is placed close to the intersection, it underestimates the traffic volume to just that within the field of view when there could be a longer queue of cars waiting beyond the field of view of the camera. If the camera is placed too far from the lane, the images could be too blurred hence cannot be processed efficiently by the image processing algorithm.

An experiment was then conducted to determine the behavior of the camera as the perpendicular distance L from the lane changes. This experiment was carried out at a constant lighting condition in the afternoon. The height h of the camera from the ground kept constant at 1.5m. This height h=1.5m is chosen by finding the approximate average height of vehicles common on the road. The whole vertical height of the vehicle should be seen within the field of view of the camera.

Max height of any vehicle on the road = 3m

Hence the camera should be at least 1.5m above the ground. For the max height to be captured within the field of view, the camera should be a certain minimum distance away from the lane. The vertical field of view of the pi camera is 48.8^{o} hence the setup is as shown in the diagram in figure 4.7.

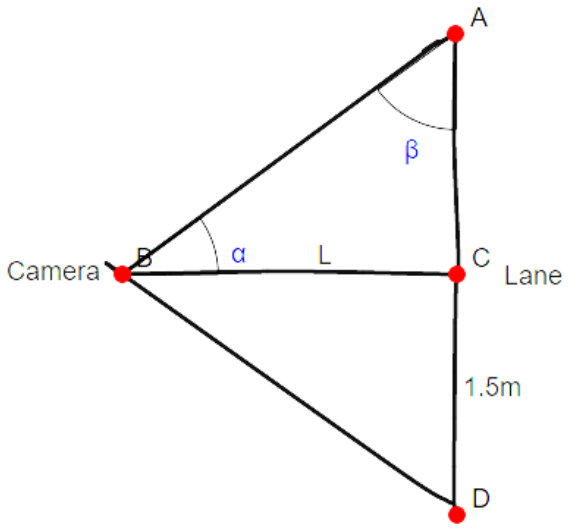

Figure 4.7: Sketch of the lane and the relative position of the camera

$$\alpha = 24.4°$$
$$\beta = 65.6°$$
$$vertical\ height\ of\ the\ camera\ from\ the\ ground = 1.5m$$
$$L = horizontal\ distance\ of\ the\ camera\ from\ the\ lane = 1.5m$$
$$AD = vertical\ distance\ within\ the\ field\ of\ viw\ of\ the\ camera$$

From sine rule:

$$\frac{1.5}{\sin 24.4°} = \frac{L}{\sin 65.6°}$$

$$L = 3.3m$$

Hence the camera has to be at least 3.3m from the lane. Pictures are taken from varying distances L from the lane as in the figure below. The pictures are taken using the raspberry pi and a phone camera for comparison. Taking the pictures from the side view, the height of the camera from the ground is maintained at 1.5m.

An experiment is further carried out to determine the maximum horizontal distance the camera can be mounted away from the lane. Pictures are taken with the pi camera from various horizontal distances of the camera from the lane. The setup is as shown in figure 4.8.

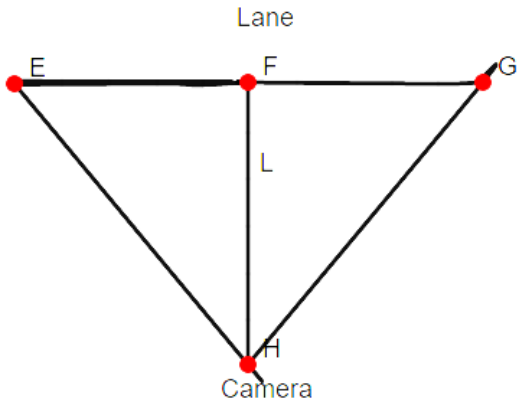

Figure 4.8: Sketch of the lane and horizontal distance L of the camera from the lane

$$EG - horizontal\ distance\ within\ the\ field\ of\ view\ of\ the\ camera$$

The results of this experiment are recorded in table 4.4.

Table 4.4: Number of vehicles detected within the field of view of a pi camera and phone camera

Distance L (m)	No of cars within the field of view (pi camera)	No of cars within the field of view (phone camera)
5	1.5	1.5
10	3	3
15	3.5	4
20	4.5	5
25	5	5.5
30	6	6.5

From the table, the longer the length L, the more vehicles can be seen within the field of view of both cameras. It is desirable to have as much view as possible to ensure an informed judgment of the traffic volume. However, during the experiment, pictures taken from a distance of 30m or more had so many obscurities making it difficult for the image processing algorithm to distinguish between objects in the image. Other limitations to how far the camera can be mounted from the lane are:

- Limited right-of-way for construction of traffic infrastructure

- The maximum distance through which the noir pi camera can ‘see’ using infrared illumination.

A further experiment was carried out to determine the maximum length L through which the pi camera can see. This experiment was carried out in low light intensity at night with only the street lights and the infrared emitters illuminating the street. Infrared LEDs were used as the source of infrared.

Table 4.5: Number of cars within the field of view of the camera during low light conditions

Distance L (m)	No of cars within the field of view
5	1
10	2
15	0

From table 4.5, the number of vehicles within the field of view of the camera increases as the camera position is moved further away from the lane. However, at a distance of 15m, the image taken of the lane is so indistinguishable that the results of vehicle counting shows that there are no cars at all within the field of view.

4.4. Results on Pi to Pi Communication

The pi to pi communication component is essential in ensuring constant communication between lanes at an intersection. The use of the nRF24L01 radio transceiver was selected as the means of communication. This transmission was not successful despite the fact that fundamental steps for using this transceiver were followed. The transmitter and receiver were both set to be on the same channel, with equal data rate, payload size, and CRC (cyclic redundancy check) length.

Communication was therefore done over Wi-Fi where one raspberry pi is set as a server while the other is a client, sending requests to the server. The flask web micro-framework is used to set one raspberry pi as a server, while the urllib3 python library is used to set the other raspberry pi as an HTTP client that sends HTTP requests. The payload to be sent is embedded within the request. The code for this process is found in Appendix VII.

Chapter 5 : Conclusion, Limitations and Future Work

5.1.Conclusion

The successful design and implementation of this affordable density based traffic management system depends on appropriate camera positioning, an accurate image processing algorithm, an efficient and low power pi to pi communication system, as well as an efficient algorithm for green light sequence and length determination.

The image processing for vehicle detection done in this project can certainly be improved for more accurate results and better working of the system as a whole. The light intensity at an intersection is dynamically changing. The vehicle detection technique and image processing algorithm should take into account these changing variables so as to obtain an accurate reading of the traffic density. From the experiments, the longer the distance of the camera from the lane, the more the vehicles within the field of view of the camera. However, at a point 30m, the vehicles are indistinguishable in the image, hence there is a limit to how far the camera can be mounted from the lane.

In determining the green light sequence, vehicle count for the lanes at the intersection is important. This data is obtained from the image processing section. To ensure fairness, the hunger level parameter is introduced. This takes into account the number of times a lane has not received green light and ensures that wait times of lanes with few vehicles are not overly prolonged. Blank cases of lanes are noted to ensure green light is not assigned to a lane when there are no vehicles passing through that lane. All these parameters put together reduce the wait time of the vehicles at the intersections.

The use of raspberry pi and pi camera for this application lower the cost of the system as these are affordable devices. The use of nRF24L01 transceiver is also advantageous since it is low cost, consumes very low power during use and operates within the license-free 2.4GHz band. However, in this project, the use of this radio transceiver was unsuccessful hence communication was done over Wi-Fi.

5.2.Limitations

The results obtained by the image processing algorithm are limited by the angle of field of view of the camera. The camera can only view a few cars within its field of view when placed at a distance from the lane. This number increases as the camera position moves further away from the lane but is limited by certain factors. If the camera is too far from the lane, the images of cars are not so clear hence the image processing algorithm is unable to distinguish between objects in the image. The maximum position the camera can be mounted for the images to remain distinguishable is 30m. The distance of the camera from the lane is also limited by the right-of-way available for use for transport infrastructure. The camera can only be mounted within the land that is allocated for this purpose.

The use of Wi-Fi or the proposed nRF24L01 utilizes the 2.4GHz license-free ISM bandwidth which is one of the most widely used bandwidths. This bandwidth is usually overcrowded, which may limit the speed of data transfer between the raspberry pi's.

The implementation of the green light sequence and length determination algorithm in this project does not take into account the special circumstances such as presence ambulances and fire trucks. This limits its functionality in case such emergency vehicles need to be given priority.

5.3.Future Work

The green light determination depends on several traffic factors. Incorporating other factors such as special circumstances are also important. For instance, the incorporation of a system that will detect special vehicles such as fire brigade trucks, ambulances is important. This factor should be included to give priority to the lane with such vehicles over the rest of the lanes. To include this parameter, it would be necessary to incorporate methods for detecting siren sounds such as the use of neural networks as explored by Tran et al in [31].

For a more accurate image processing algorithm, the use of machine learning will increase the accuracy of the vehicle detection algorithm. Using a trained algorithm that will learn the specific features of a vehicle will greatly improve the accuracy of vehicle detection.

The field of view of the camera can be increased using special lenses to ensure that even when there are limitations to how far the camera can be mounted from the lane, a maximum field of view is maintained.

References

[1] I. Greco and A. Cresta, "A Smart Planning for Smart City: The Concept of Smart," *ICCSA 2015, Part II, LNCS 9156*, p. 563–576, 2015.

[2] M. Eremia, L. Toma and M. Sanduleac, "The Smart City Concept in the 21st Century," in *10th International Conference Interdisciplinarity in Engineering*, 2017.

[3] D. Sikora-Fernandez and D. Stawasz, "The Concept of Smart City in the Theory and Practice of Urban Development Management," *The Journal of the Romanian Regiaonal Science Association,* vol. 10, no. 1, 2016.

[4] B. Fernando, E. Gray and J. Kellner, "A Review of Current Traffic Congestion Management in the City of Sydney," Sydney, 2013.

[5] A. R. Kotwal, S. J. Lee and Y. J. Kim, "Traffic Signal Systems: A Review of Current Technology in the United States," *Science and Technology* , vol. 3, no. 1, pp. 33-41, 2013.

[6] P. Daniel, K.-N. Daniela and I. Edouard, "Evaluation of Adaptive Traffic Control System," in *59th International Symposium ELMAR*, Zadar-Croatia, 2017.

[7] S. G. Shelby, D. M. Bullock, D. Gettman, R. S. Ghaman, Z. A. Sabra and N. Soyke, "An Overview and Performance Evaluation of ACS Lite – A Low Cost Adaptive Signal Control System," in *87th TRB Annual Meeting*, Washington DC, 2008.

[8] S. McGregor and D. M. Doya, "Traffic Costs Nairobi $570,000 a Day as No. 2 Africa Hub," 25 March 2014. [Online]. Available:

https://www.bloomberg.com/news/articles/2014-03-25/nairobi-traffic-loses-570-000-a-day-as-no-2-africa-hub.

[9] "African cities with worst traffic," May 2018. [Online]. Available: http://www.mediamaxnetwork.co.ke/435409/african-cities-with-worst-traffic/.

[10] A. Dzikus, "Fighting traffic congestion in Nairobi," 28 April 2017. [Online]. Available: https://www.the-star.co.ke/news/2017/04/28/fighting-traffic-congestion-in-nairobi_c1547763.

[11] Y. N. Udoakah and L. Okure, "Design and Implementation of a Density-ased Traffic Light Control with Surveillance System," *Nigerian Journal of Technology,* vol. 36, no. 4, pp. 1239-1248, October 2017.

[12] L. E. Y. Mimbela, L. A. Klein and P. Kent, "A Summary of Vehicle Detection and Surveillance Technologies used in Intelligent Transportation Systems," 2000.

[13] A. Zarnescu, R. Ungurelu, A. G. Iordache, M. Secere and M. Spoiala, "Crossroad Traffic Monitoring Using Magnetic," *Institute of Electrical and Electronics Engineering,* pp. 413-418, 2017.

[14] V. Dangi, A. Parab, K. Pawar and S. Rath, "Image Processing Based Intelligent Traffic Controller," *Undergraduate Academic Research Journal,* vol. 1, no. 1, pp. 13-17, 2012.

[15] K. Vidhya and A. B. Banu, "Density Based Traffic Signal System," in *2014 International Conference on Innovations in Engineering and Technology (ICIET'14)*, 2014.

[16] R. Datondji, Y. Dupuis, P. Subirats and P. Vasseur, "A Survey Of Vision-Based Traffic Monitoring Of Road Intersections," *IEEE Transactions on Intelligent Transport Systems,* pp. 1-19, 2018.

[17] R. Tiwari and D. K. Singh, "Vehicle control using raspberry pi and image processing," *Innovative Systems Design and Engineering,* vol. 8, no. 2, pp. 45-49, 2017.

[18] R. Justin and R. Kumar, "Vehicle Detection and Counting Method Based on Digital Image Processing in Python," *International Journal of Electrical Electronics & Computer Science Engineering,* pp. 141-147, 2018.

[19] P. Choudekar, S. Banerjee and M. K. Muju, "Real Time Traffic Light Control Using Image Processing," *Indian Journal of Computer Sciencee and Engineering,* vol. 2, no. 1, pp. 6-10.

[20] A. Mordvintsev and A.K., "Canny Edge Detection," OpenCV-Python Tutorials, 2013. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html.

[21] G. R. Choukekar and A. G. Bhosale, "Density Based Smart Traffic Light Control System and Emergency Vehicle Detection Based On Image Processing," *International Research Journal of Engineering and Technology (IRJET),* vol. 5, no. 4, pp. 2441-2446, 2018.

[22] P. KadewTraKuPong and R. Bowden, *An improved background mixture model for real-time tracking with shadow detection,* 2001.

[23] Z. Zivkovic, *Improved adaptive Gausian mixture model for background subtraction,* 2004.

[24] Z. Zivkovic, *Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction,* 2006.

[25] A. B. Godbehere, A. Matsukawa and K. Goldberg, *Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation,* 2012.

[26] "Background Subtraction," OpenCV Python Tutorials, [Online]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html.

[27] D. Beymer, P. McLauchlain, B. Coifman and J. Malik, "A Real-time Computer Vision System for Measuring Traffic Parameters," *IEEE,* pp. 495-501, 1997.

[28] Z. Yanbing, "Design of Low-power Wireless Communication System Based on MSP430 and nRF2401," in *2010 International Conference on Measuring Technology and Mechatronics Automation*, Shanxi, 2010.

[29] Z. Yanbing, "Design of Low Power Wireless Communication System Based on MSP430 and nRF2401," in *2010 International Conference on Measuring Technology and Mechatronics Automation*, Shanxi, 2010.

[30] B. Zhou, J. Cao, X. Zeng and H. Wu, "Adaptive Traffic Light Control in Wireless Sensor Network-based Intelligent Transportation System," *IEEE,* 2010.

[31] V.-T. Tran, Y.-C. Yan and W.-H. Tsai, "Detection of Ambulance and Fire Truck Siren Sounds Using Neural Networks," in *Proceedinggs of 51st Research World International Conference*, Hanoi-Vietnam, 2018.

[32] T. Gordon, Z. Bareket, L. Kostyniuk, M. Barnes, M. Hagan, Z. Kim, D. Cody, A. Skabardonis and A. Vayda, "Image processing and feature extraction," in *Site-based Video Sytem Design and development*, The National Academies Press, 2012, pp. 50-55.

[33] N. Patrascoiu and C. Rus, "A mobile system for data acquisition," *Institute of Electrical and Electronics Engineering,* pp. 318-321, 2018.

Appendix

Appendix I: Transmit Code for nRF24L01 Transceiver

```
Title: Raspberry Pi 3 Tutorial 13-Wireless Pi to Pi Communication with
nRF24L01
Author: Sushant Narang
Date: 2016
Availability: http://invent.module143.com/daskal_tutorial/raspberry-pi-3-
wireless-pi-to-pi-python-communication-with-nrf24l01/

import RPi.GPIO as GPIO

from lib_nrf24 import NRF24
import time
import spidev

GPIO.setmode(GPIO.BCM)

pipes= [[0xe7, 0xe7, 0xe7, 0xe7, 0xe7], [0xc2, 0xc2, 0xc2, 0xc2, 0xc2]]

radio = NRF24(GPIO, spidev.SpiDev())
radio.begin(0,17)
radio.setPayloadSize(32)
radio.setChannel(0x60)

radio.setDataRate(NRF24.BR_2MBPS)
radio.setPALevel(NRF24.PA_MAX)
radio.setAutoAck(True)
radio.enableDynamicPayloads()
radio.enableAckPayload()
radio.setCRCLength(16)

radio.openWritingPipe(pipes[1])
radio.printDetails()

while(1):
    message = list("Number of vehicles on lane i")
    radio.write(message)
    print("We sent the message of {}".format(message))

    #check if it returned ackPL
    if radio.isAckPayloadAvailable():
        returnedPL = []
        radio.read(returnedPL, radio.getDynamicPayloadSize())
        print("Our returned payload was {}".format(returnedPL))
    else:
        print("No payload received")
    time.sleep(1)
```

Appendix II: Receive Code for nRF24L01 Transceiver

```
Title: Raspberry Pi 3 Tutorial 13-Wireless Pi to Pi Communication with
nRF24L01
Author: Sushant Narang
Date: 2016
Availability: http://invent.module143.com/daskal_tutorial/raspberry-pi-3-
wireless-pi-to-pi-python-communication-with-nrf24l01/

import RPi.GPIO as GPIO

from lib_nrf24 import NRF24
import time
import spidev

GPIO.setmode(GPIO.BCM)

pipes = [[0xe7, 0xe7, 0xe7, 0xe7,0xe7], [0xc2,  0xc2, 0xc2, 0xc2, 0xc2]]

#create an instance of the radio
radio = NRF24(GPIO, spidev.SpiDev())
#CSN and CE turn radio on
radio.begin(0,17)
radio.setPayloadSize(32)
radio.setChannel(0x60)

radio.setDataRate(NRF24.BR_2MBPS)
radio.setPALevel(NRF24.PA_MAX)
radio.setAutoAck(True)
radio.enableDynamicPayloads()
radio.enableAckPayload()
radio.setCRCLength(16)
radio.openReadingPipe(1, pipes[0])
radio.printDetails()

radio.startListening()
while (1):
    ackPL = [1]
    while not radio.available(0):
        time.sleep(1/100)
    receivedMessage = []
    radio.read(receivedMessage, radio.getDynamicPayloadSize())
    time.sleep(3)
    print("Received: {}".format(receivedMessage))

    print("Translating the received message into unicode characters")
    string = ""
    for n in receivedMessage:
        #Decode into standard unicode set
        if(n>=32 and n<=126):
```

```
        string += chr(n)
print(string)
radio.writeAckPayload(1, ackPL, len(ackPL))
print("Loaded payload reply of {}".format(ackPL))
```

Appendix III: Image Processing Code

```
#Image processing for Vehicle Detection
#@author: Odero Margaret Anyango
#Department of Electrical and Electronics Engineering
#Ashesi University
#Final Capstone Project

#importing necessary libraies
from __future__ import print_function
import cv2
import numpy as np
import matplotlib
from matplotlib import pyplot as plt



#A. Image preprocessing
#a) importing the original image

filename = 'imagesC5m3.jpg'
oriimg = cv2.imread(filename,cv2.IMREAD_UNCHANGED)
cv2.imshow('Original Image',oriimg)

#b) resizing the image

height, width, depth = oriimg.shape
print(height, width, depth)
imgScale = 500/width
newX,newY = oriimg.shape[1]*imgScale, oriimg.shape[0]*imgScale
sizedimg = cv2.resize(oriimg,(int(newX),int(newY)))
cv2.imshow("Sized Image",sizedimg)

#c) gray-scale conversion

gray_image = cv2.cvtColor(sizedimg, cv2.COLOR_BGR2GRAY)
cv2.imshow("Gray Scale Image",gray_image)

#d) power law transformation

gray_image1 = gray_image/255.0
im_power_law_transformation = cv2.pow(gray_image1,1.2)
#cv2.imshow('Power Law Transformation',im_power_law_transformation)
cv2.imwrite('im_power_law_transformation.jpg',im_power_law_transformation)

#B. Edge Detection using Canny Algorithm

img= gray_image.astype(np.uint8)
edges = cv2.Canny(img,100,200)
cv2.imshow('Edges',edges)
```

```
#b) Automatic Canny Algorithm- automiatically generates thresholds
import argparse
import glob
def auto_canny(image, sigma=0.33):
      # compute the median of the single channel pixel intensities
      v = np.median(image)

      # apply automatic Canny edge detection using the computed median
      lower = int(max(0, (1.0 - sigma) * v))
      upper = int(min(255, (1.0 + sigma) * v))
      edged = cv2.Canny(image, lower, upper)

      # return the edged image
      return edged
auto = auto_canny(gray_image)
#cv2.imshow('auto_canny',auto)
cv2.imwrite('auto.jpg',auto)

#C. Background subtraction

foregbackg = cv2.bgsegm.createBackgroundSubtractorMOG()
foregmask = foregbackg.apply(gray_image)
#cv2.imshow('Foreground Mask',foregmask)

#Finding contours
imcontours, contours, hierarchy = cv2.findContours(edges,
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
cv2.imshow('imcontours', imcontours)



large_contours = []
#print ("Return type:",type(contours))
for (i, c) in enumerate(contours):
        if len(c)>200:
                large_contours.append(c)
                #print("\tSize of contour %d: %d" % (i, len(c)))

#Number of large object in the image
print("Found %d objects." % len(large_contours))

# draw contours over original image
cv2.drawContours(sizedimg, large_contours, -1, (0, 0, 255), 1)

#diplaying squares
# create all-black mask image
mask = np.zeros(sizedimg.shape, dtype="uint8")

# draw red rectangles for each object's bounding box
```

```
for c in large_contours:
    (x, y, w, h) = cv2.boundingRect(c)
    cv2.rectangle(mask, (x, y), (x + w, y + h), (255, 255, 255), -1)
    cv2.rectangle(sizedimg, (x, y), (x + w, y + h), (255, 0, 0), 2)

# apply mask to the original image
img = cv2.bitwise_and(sizedimg, mask)

# display masked image
cv2.namedWindow("output", cv2.WINDOW_NORMAL)
cv2.imshow("output", img)
cv2.waitKey(0)
```

Appendix IV: Code for Green Light Sequence and Length Determination

```
#Green Light Sequence and Length Determination
#@author: Odero Margaret Anyango
#Department of Electrical and Electronics Engineering
#Ashesi University
#Final Capstone Project

import RPi.GPIO as GPIO
import random
import time

GPIO.setmode(GPIO.BCM) #specify the naming convention to be used
GPIO.setwarnings(False)

#GPIOs to be used for traffic lights
#RL : RED LIGHT GL: GREEN LIGHT The numbers 1-8 represent the lane numbers
RL1 = 2
GL1 = 3
RL2 = 4
GL2 = 14
RL3 = 27
GL3 = 23
RL4 = 22
GL4 = 24
RL5 = 25
GL5 = 7
RL6 = 5
GL6 = 6
RL7 = 12
GL7 = 13
RL8 = 19
GL8 = 26

#set pins as output
GPIO.setup(RL1, GPIO.OUT)
GPIO.setup(GL1, GPIO.OUT)
GPIO.setup(RL2, GPIO.OUT)
GPIO.setup(GL2, GPIO.OUT)
GPIO.setup(RL3, GPIO.OUT)
GPIO.setup(GL3, GPIO.OUT)
GPIO.setup(RL4, GPIO.OUT)
GPIO.setup(GL4, GPIO.OUT)
GPIO.setup(RL5, GPIO.OUT)
GPIO.setup(GL5, GPIO.OUT)
GPIO.setup(RL6, GPIO.OUT)
GPIO.setup(GL6, GPIO.OUT)
GPIO.setup(RL7, GPIO.OUT)
GPIO.setup(GL7, GPIO.OUT)
GPIO.setup(RL8, GPIO.OUT)
```

```
GPIO.setup(GL8, GPIO.OUT)

#At the start, define all streets red
def setAllStreetsRed():
    GPIO.output(RL1, GPIO.HIGH)
    GPIO.output(GL1, GPIO.LOW)
    GPIO.output(RL2, GPIO.HIGH)
    GPIO.output(GL2, GPIO.LOW)
    GPIO.output(RL3, GPIO.HIGH)
    GPIO.output(GL3, GPIO.LOW)
    GPIO.output(RL4, GPIO.HIGH)
    GPIO.output(GL4, GPIO.LOW)
    GPIO.output(RL5, GPIO.HIGH)
    GPIO.output(GL5, GPIO.LOW)
    GPIO.output(RL6, GPIO.HIGH)
    GPIO.output(GL6, GPIO.LOW)
    GPIO.output(RL7, GPIO.HIGH)
    GPIO.output(GL7, GPIO.LOW)
    GPIO.output(RL8, GPIO.HIGH)
    GPIO.output(GL8, GPIO.LOW)

#Define the different cases(for testing, only case 1-4 out of 12 are
#defined)
def setCase1Active():
    GPIO.output(RL1, GPIO.HIGH)
    GPIO.output(GL1, GPIO.LOW)
    GPIO.output(RL2, GPIO.HIGH)
    GPIO.output(GL2, GPIO.LOW)
    GPIO.output(RL3, GPIO.LOW)
    GPIO.output(GL3, GPIO.HIGH)
    GPIO.output(RL4, GPIO.HIGH)
    GPIO.output(GL4, GPIO.LOW)
    GPIO.output(RL5, GPIO.HIGH)
    GPIO.output(GL5, GPIO.LOW)
    GPIO.output(RL6, GPIO.HIGH)
    GPIO.output(GL6, GPIO.LOW)
    GPIO.output(RL7, GPIO.LOW)
    GPIO.output(GL7, GPIO.HIGH)
    GPIO.output(RL8, GPIO.HIGH)
    GPIO.output(GL8, GPIO.LOW)

def setCase2Active():
    GPIO.output(RL1, GPIO.HIGH)
    GPIO.output(GL1, GPIO.LOW)
    GPIO.output(RL2, GPIO.HIGH)
    GPIO.output(GL2, GPIO.LOW)
    GPIO.output(RL3, GPIO.HIGH)
    GPIO.output(GL3, GPIO.LOW)
    GPIO.output(RL4, GPIO.HIGH)
    GPIO.output(GL4, GPIO.LOW)
```

```
    GPIO.output(RL5, GPIO.HIGH)
    GPIO.output(GL5, GPIO.LOW)
    GPIO.output(RL6, GPIO.HIGH)
    GPIO.output(GL6, GPIO.LOW)
    GPIO.output(RL7, GPIO.LOW)
    GPIO.output(GL7, GPIO.HIGH)
    GPIO.output(RL8, GPIO.LOW)
    GPIO.output(GL8, GPIO.HIGH)

def setCase3Active():
    GPIO.output(RL1, GPIO.HIGH)
    GPIO.output(GL1, GPIO.LOW)
    GPIO.output(RL2, GPIO.HIGH)
    GPIO.output(GL2, GPIO.LOW)
    GPIO.output(RL3, GPIO.HIGH)
    GPIO.output(GL3, GPIO.LOW)
    GPIO.output(RL4, GPIO.LOW)
    GPIO.output(GL4, GPIO.HIGH)
    GPIO.output(RL5, GPIO.HIGH)
    GPIO.output(GL5, GPIO.LOW)
    GPIO.output(RL6, GPIO.HIGH)
    GPIO.output(GL6, GPIO.LOW)
    GPIO.output(RL7, GPIO.HIGH)
    GPIO.output(GL7, GPIO.LOW)
    GPIO.output(RL8, GPIO.LOW)
    GPIO.output(GL8, GPIO.HIGH)

def setCase4Active():
    GPIO.output(RL1, GPIO.HIGH)
    GPIO.output(GL1, GPIO.LOW)
    GPIO.output(RL2, GPIO.HIGH)
    GPIO.output(GL2, GPIO.LOW)
    GPIO.output(RL3, GPIO.LOW)
    GPIO.output(GL3, GPIO.HIGH)
    GPIO.output(RL4, GPIO.LOW)
    GPIO.output(GL4, GPIO.HIGH)
    GPIO.output(RL5, GPIO.HIGH)
    GPIO.output(GL5, GPIO.LOW)
    GPIO.output(RL6, GPIO.HIGH)
    GPIO.output(GL6, GPIO.LOW)
    GPIO.output(RL7, GPIO.HIGH)
    GPIO.output(GL7, GPIO.LOW)
    GPIO.output(RL8, GPIO.HIGH)
    GPIO.output(GL8, GPIO.LOW)

I = ['north', 'south', 'east', 'west']
J = ['forward', 'left']
R = [1,2,3,4,5,6,7,8]
```

```
Cases_dict =
{'case1':['L3','L7'],'case2':['L7','L8'],'case3':['L8','L4'],'case4':['L3',
'L4']}

a1 = 4
a2 = 2
a3 = 6
a4 = 8
a5 = 10

#constant car speed assumed to be 20 while driving across the intersection
car_speed = 20

#transition time between red light and green light
T_transition = 50 #seconds

#Maximum green light time
T_control = 180 #seconds; determined from literature

setAllStreetsRed()
time.sleep(5)
Time = 5 #number of times the testing will be done
while Time > 0:
    No_ofGLD_dict =
    {'case1':random.randint(1,5),'case2':random.randint(1,5),'case3':random
    .randint(1,5),'case4':4}

    #finding the total no. of green light instances
    No_ofGLD_sum = 0
    for val in No_ofGLD_dict.values():
        No_ofGLD_sum = No_ofGLD_sum + val

    #traffic volume for different lanes 1-8, at every time t
        TraVol_dict = {'L1':random.randint(1,10),
      'L2':random.randint(1,10), 'L3':random.randint(1,10),
      'L4':random.randint(1,10), 'L5':random.randint(1,10),
      'L6':random.randint(1,10), 'L7':random.randint(1,10),
      'L8':random.randint(1,10)}

    #finding total TraVol
    TraVol_sum = 0
    for val in TraVol_dict.values():
        TraVol_sum = TraVol_sum + val

      #green light priority determination GLD
    GLD_dict = {}

    #Case 1
    TV_1_t = (TraVol_dict.get('L3')+ TraVol_dict.get('L7'))/TraVol_sum
    WT_1_t = random.uniform(0,1)
```

```
    HL_1_t = 1-(No_ofGLD_dict['case1']/No_ofGLD_sum)
    BC_1_t = 0 #assuming no blank cases
    SC_1_t = 0 #assuming no special circumstances
    GLD_1_t = a1*TV_1_t + a2*WT_1_t + a3*HL_1_t +  a4*BC_1_t + a5*SC_1_t

    GLD_dict['case1'] = GLD_1_t

    #Case 2
    TV_2_t = (TraVol_dict.get('L7')+ TraVol_dict.get('L8'))/TraVol_sum
    WT_2_t = random.uniform(0,1)
    HL_2_t = 1-(No_ofGLD_dict['case2']/No_ofGLD_sum)
    BC_2_t = 0 #assuming no blank cases
    SC_2_t = 0 #assuming no special circumstances
    GLD_2_t = a1*TV_2_t + a2*WT_2_t + a3*HL_2_t +  a4*BC_2_t + a5*SC_2_t

    GLD_dict['case2'] = GLD_2_t

    #Case 3
    TV_3_t = (TraVol_dict.get('L8')+ TraVol_dict.get('L4'))/TraVol_sum
    WT_3_t = random.uniform(0,1)
    HL_3_t = 1-(No_ofGLD_dict['case2']/No_ofGLD_sum)
    BC_3_t = 0 #assuming no blank cases
    SC_3_t = 0 #assuming no special circumstances
    GLD_3_t = a1*TV_3_t + a2*WT_3_t + a3*HL_3_t +  a4*BC_3_t + a5*SC_3_t

    GLD_dict['case3'] = GLD_3_t

    #Case 4
    TV_4_t = (TraVol_dict.get('L3')+ TraVol_dict.get('L4'))/TraVol_sum
    WT_4_t = random.uniform(0,1)
    HL_4_t = 1-(No_ofGLD_dict['case2']/No_ofGLD_sum)
    BC_4_t = 0 #assuming no blank cases
    SC_4_t = 0 #assuming no special circumstances
    GLD_4_t = a1*TV_4_t + a2*WT_4_t + a3*HL_4_t +  a4*BC_4_t + a5*SC_4_t

    GLD_dict['case4'] = GLD_4_t
    TV_dict = {'case1': TV_1_t, 'case2': TV_2_t, 'case3': TV_3_t, 'case4':
TV_4_t}
    WT_dict = {'case1': WT_1_t, 'case2': WT_2_t, 'case3': WT_3_t, 'case4':
WT_4_t}
    HL_dict = {'case1': HL_1_t, 'case2': HL_2_t, 'case3': HL_3_t, 'case4':
HL_4_t}
    print('GLD dictionary: ' + str(GLD_dict))
    print('TV dictionary: ' + str(TV_dict))
    print('WT dictionary: ' + str(WT_dict))
    print('HL dictionary: ' + str(HL_dict))


    GLD_max = max(list(GLD_dict.values()))
    print("Maximum GLD is  "+'{}'.format(GLD_max))
```

```
    #Green Light Duration
    for key, val in GLD_dict.items():
        if val == GLD_max:
            GLD_max_key = key

    print("Maximum GLD is of  "+'{}'.format(GLD_max_key))
    #print("Street 2: "+'{}'.format(street2))

    #m=incrementing the count of green light for case with max GLD by 1
    No_ofGLD_dict[key] = No_ofGLD_dict[key]+ 1

    lanes_maxcase_list = list(Cases_dict[GLD_max_key])
    TraVol1_key = lanes_maxcase_list[0]
    TraVol1 = TraVol_dict[TraVol1_key]

    TraVol2_key = lanes_maxcase_list[1]
    TraVol2 = TraVol_dict[TraVol2_key]

    G_next = ((max(TraVol1,TraVol2))/car_speed) * T_transition
    print("Max traffic volume of {} is
{}".format(GLD_max_key,max(TraVol1,TraVol2)))

    if G_next > T_control:
        G_next = T_control
    print("Green light length is "+'{}'.format(G_next))

    #switching on and off the green light
    GLDkeys_tuple = GLD_dict.keys()
    GLDkeys_list = list(GLDkeys_tuple)
    GLDkeys_list.sort()
    print("The GLD key list is: "+ str(GLDkeys_list))

    if GLD_max_key == GLDkeys_list[0]: #case1
        setCase1Active()
        print ("case1 is active and green")
    elif GLD_max_key == GLDkeys_list[1]: #case2
        setCase2Active()
        print ("case2 is active and green")
    elif GLD_max_key == GLDkeys_list[2]: #case3
        setCase3Active()
        print ("case3 is active and green")
    elif GLD_max_key == GLDkeys_list[3]: #case4
        print ("case4 is active and green")
        setCase4Active()
    print('')
    print('')
    time.sleep(G_next)
    Time-=1
    setAllStreetsRed()
```

Appendix V: Code Block for Image Capture

```
from picamera import PiCamera  #importing the class PiCamera that contains all the methods necessary for use with the pi camera.
from time import sleep
camera =PiCamera() #creating a camera object
camera.start_preview() #start viewing the object within the field of view of the #camera
sleep(10) #allow 10 seconds before taking the picture
camera.capture('/home/pi/Desktop/CapstonePictures/street1.jpg')#capture the picture #and store it in the path indicated with the name street1 of type .jpg
camera.stop_preview()
```

Appendix VI: The Working of the nRF24L01 Transceiver

The nRF24L01 transceiver sends and receives data on a frequency channel. Two modules communicating need to be on the same channel to communicate. The channel occupies a 1MHz bandwidth at 250kbps or 1Mbps data rate, and 2MHz bandwidth at 2Mbps data rate and this bandwidth and could be anywhere between 2.4GHz to 2.525GHz. Hence, there are 125 possible channels to be used. nRF24L01 module has a feature known as multiple transmitters single receiver (multiceiver) in which the RF channel is divided into 6 parallel data pipes, each with a physical address(data pipe address).

The transceiver uses enhanced shockburst protocol with a functionality for payload length specification in each packet. The payload length may vary from 1 to 32bytes. Each packet can also request for acknowledgment to be sent when it is received by another device (stop-and-wait). Packet handling is done automatically by the transceiver, and the MCU is not involved.

It has 4 kinds of working modes, as shown in table 0.1.

Operating mode	Power Up	Pin CE	Pin CS
Transceiver mode	1	1	0
Configure mode	1	0	1
Idle mode	1	0	0
Off mode	0	x	x

Table 0.1: Working modes of the NRF2401 transceiver

Appendix VII: Server-Client Setup for Pi to Pi Communication

1. To install Flask on the server side, this command is used in the command line interface.

```
pip install Flask
```

2. To create a web micro-server

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
      return "hello"

if __name__ == "__main__":
      app.run(host = '0.0.0.0')
```

3. To install urllib3 on the client side

```
pip install urllib3
```

4. To send request from the client side to the server side

```
import urllib3


client = urllib3.PoolManager()
data = 'number of cars lane i'
url = 'http://169.254.92.78:5000/'
#the IP address is of the server raspberry pi
method = 'POST' #to send a post request

response = client.request(method,url,body=data)
#sending a request to the server, with data embedded within
the request
```