Faculty of Engineering & Applied Science

OntarioTech UNIVERSITY

ENGR 4941U
Capstone System Design II

Design of Smart Traffic Signals for Autonomous Transportation and Connected Communities in Smart Cities

R1: Detail Design and Integration Testing Report

Group#: 8
Section CRN#:73459
Advisor: Dr. Hossam Gaber
Coordinator: Dr. Vijay Sood
Due Date: February 18th, 2023

First Name	Last Name	Student Number
Tirth	Patel	100751761
Vatsal	Patel	100728903
Saro	Karimi	100755079

Abstract Executive Summary

There are a number of things included in this report, first we started our report with the introduction about what our project is about and followed by that we had diagrams about the main components we used for this project. Then we did analysis about our project, how it is going to look like and how it will work in the real-life. After having our analysis, we came up with the algorithms that would support our analysis we have described. After the detailed design was described, then we started the integration part. In the integration section, we have described our server, the module, how it will work, how it will support our system. Following that we have described how the Edge server will take the information from the car. Then we have our updated project plan, and our contribution matrix table and describe in detail what each group member did. At last, we have our references.

To conclude, by doing this project and working on it physically. It gave us a better understanding about the overall project, and we also learned about how this stuff works in real-life from day-to-day life.

Acknowledgement

We want to appreciate and sincerely thank our advisor Dr. Hossam Gaber for making this work possible. His guidance accompanies us through every phase of this report-writing. We also want to express our gratitude to our coordinator Dr. Vijay Sood for providing us with some inputs regarding the reports and the overall project.

Table of content

List of Tables

List of figures

Abbreviations

GPIO	General Purpose Input Output
GLONASS	Global Navigation Satellite System
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
I2C	Inter-Integrated Circuit
IP	Internet Protocol
LAT	Latitude
LNG	Longitude
MQTT	Message Queuing Telemetry Transport
NMEA	National Marine Electronics Association
SPI	Serial Peripheral Interface
STS	Smart Traffic System
TTL	Time-To-Live
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver Transmitter

1. Detail Design

1.1 Introduction

The conceptual design that we chose to move forward is related to distributed network-based Smart Traffic System (STS). In this design there will be many sensors and communication devices placed across the traffic network as part of a distributed network-based smart traffic system architecture in order to gather real-time data and interact with one another to coordinate traffic flow. The sensors can include cameras, infrared sensors, GPS, wifi module or other tools for seeing moving objects like cars and people on foot. In order to identify patterns and forecast traffic patterns and congestion, the data gathered from these sensors would be sent to a central server or a cloud-based system.

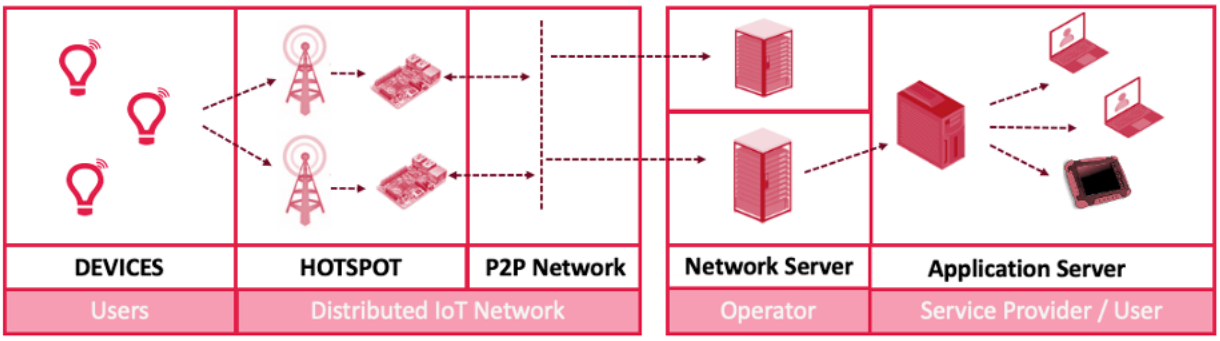

Figure 1.1: Distributed Network

The technology could also facilitate communication between individual sensors through a distributed network architecture, enabling them to exchange data and plan their operations. As an illustration, a sensor detecting traffic congestion on a certain road could transmit this information to other sensors nearby, which could subsequently modify traffic signals or other systems to reroute traffic to different routes.

Ultimately, a distributed network-based smart traffic system would be a highly advanced and dynamic system that could adjust to shifting traffic conditions and improve traffic flow in real-time.

1.2 Components/diagrams

As you can see in the diagram below, that is ESP32 WROOM Module. The ESP32 WROOM is a well-liked and reasonably priced WiFi module that is utilized to construct IoT applications. Because of its low price, low power usage, and inbuilt WiFi capability, it has grown in demand. The module can connect with other electronics using industry-standard communication protocols including HTTP, MQTT, and TCP/IP and may be programmed using the Arduino IDE or other programming environments. But for our project we have used raspberry pi to programme the ESP32 module. Its support for various WiFi protocols, a variety of GPIO pins, and integrated components like timers, SPI, I2C, and UART interfaces are some of the characteristics of the ESP32 module. It can be used for duties like data recording and analytics, real-time surveillance, and virtual access of devices. It also has a variety of on-board sensors. Intelligent technologies, house automated systems, remote surveillance and control systems, sensor nodes, switches, faucets and other IoT apps frequently use the ESP32 module, as well as to collect and send sensor data from a distance. Due to its portability and minimal power requirements, it is a preferred platform for battery-operated products and is a widely used module.

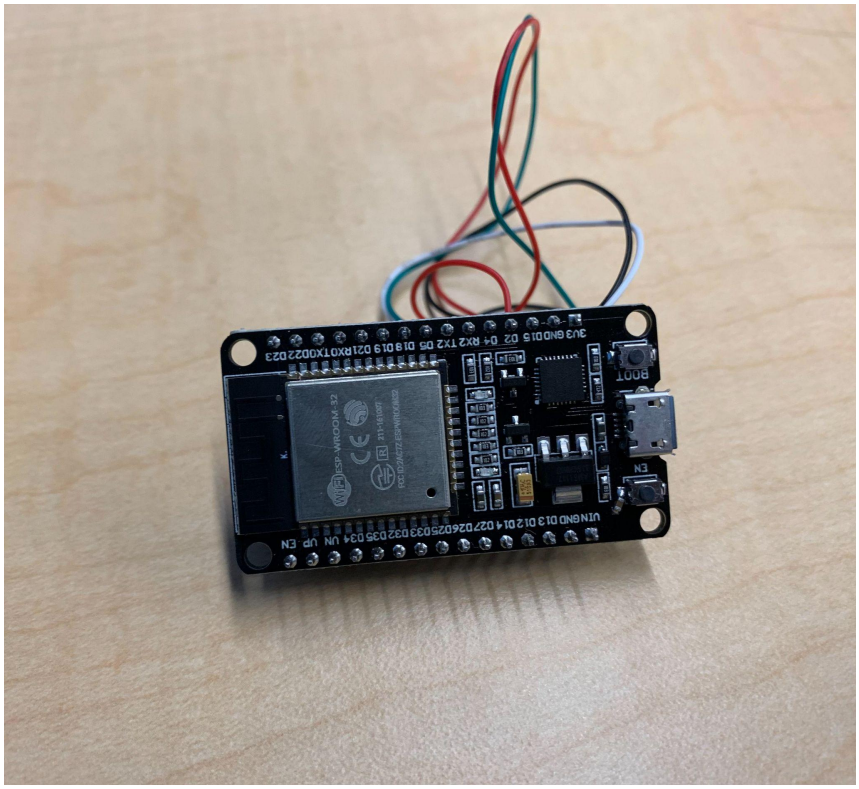

Figure 1.2: ESP32 WROOM Module

As you can see in the diagram below, that is Beitian BN-220 GPS Module TTL Dual GPS. The module is a GPS receiver device that is tiny, reasonably priced, and simple to use. It is intended for use in applications and projects. The module's dual-mode support for GPS and GLONASS satellite navigation systems enables it to deliver precise and dependable location data. The gadget has a built-in ceramic transmitter and reserve battery that enable satellite monitoring even in the event of a power outage. It is also well suited for a variety of uses, including geocaching, monitoring vehicles, drone tracking, and other location-based activities, and it consumes little battery, in situations where precise location information is crucial. This module can be tightly integrated with a range of embedded systems, including Arduino and Raspberry Pi, and it employs TTL-level serial communication to transmit and receive data. The module can output GPS and GLONASS position data in the common NMEA format, making it simple for embedded systems boards and other devices to handle. Because of its small dimensions and minimal power requirements, the module is perfect to be used in battery-powered apps. Additionally, it has a reserve battery and an integrated ceramic antenna that enable satellite monitoring even in the event of an electricity outage. To conclude, we can say that the Beitian BN-220 GPS module is an all-around adaptable and cost-effective GPS transceiver module that is perfect for use in a variety of DIY projects and apps that demand precise location data.

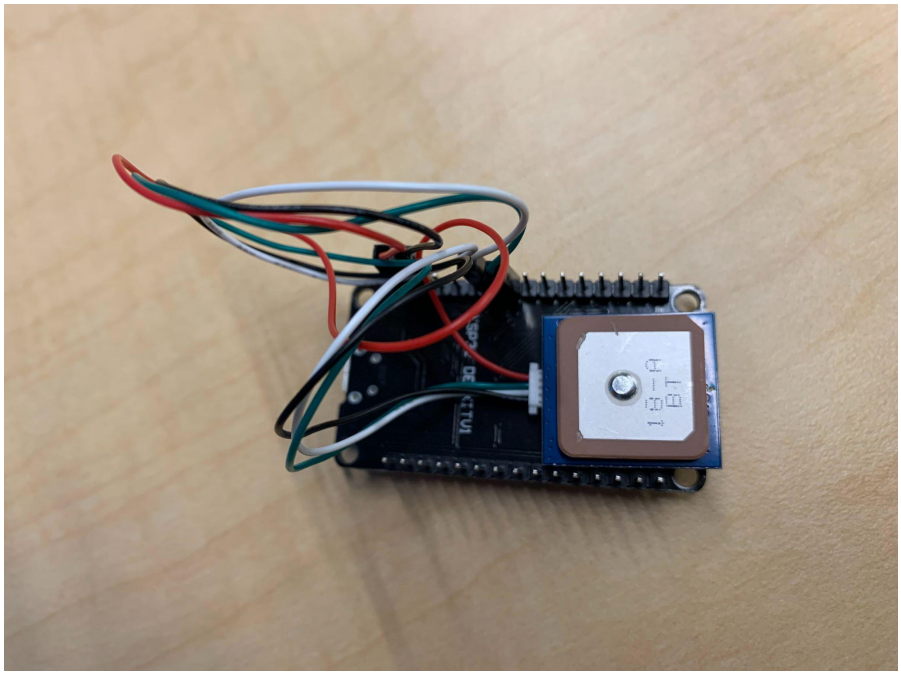

Figure 1.3: Beitian BN-220 GPS Module

As you can see in the diagram below, that is the 4GB Raspberry pi 4, we have the whole Canakit of Raspberry pi 4. The well-known Raspberry Pi 4 single-board computer and all the necessary tools are included in the 4GB Raspberry Pi 4 Canakit, a comprehensive kit. The Raspberry Pi 4 in this iteration is made to handle more demanding tasks like managing big databases or operating numerous programmes simultaneously. The newest model, which is model B of the well-known Raspberry Pi collection, the Raspberry Pi 4, is made for use in a variety of DIY applications. The Raspberry Pi 4 model included in the package has 4GB RAM, the most memory it is possible to get for the Raspberry Pi 4. The Canakit comes with a large battery pack, a microSD card with NOOBS preinstalled (the programme that makes it simple to configure the operating platform), a case, heat sinks, a fan, and a set of cords to link up the Raspberry Pi to a screen and other devices. In addition, the Raspberry Pi 4 has integrated WiFi and Bluetooth, two micro HDMI connections that can handle displays with up to 4K quality, and a number of other features, such as four USB 2.0 and two USB 3.0 ports, a Gigabit Ethernet port, a microSD card reader, and a 40-pin GPIO (General Purpose Input/Output) header for attaching additional sensors and gadgets. As our project is build on edge-server and we need to work with the bigger datasets and more complicated task, as we are building smart traffic, the 4GB Raspberry Pi 4 is the perfect choice.

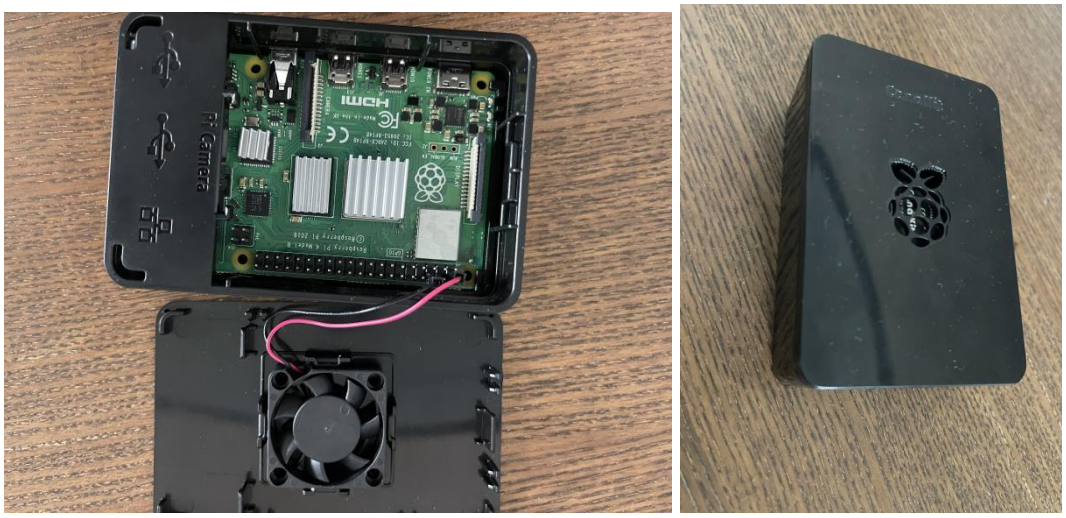

Figure 1.4 & 1.5: Show the 4GB Raspberry pi 4 Module

1.3 Analysis

Test Case Scenarios:

Scenario1:

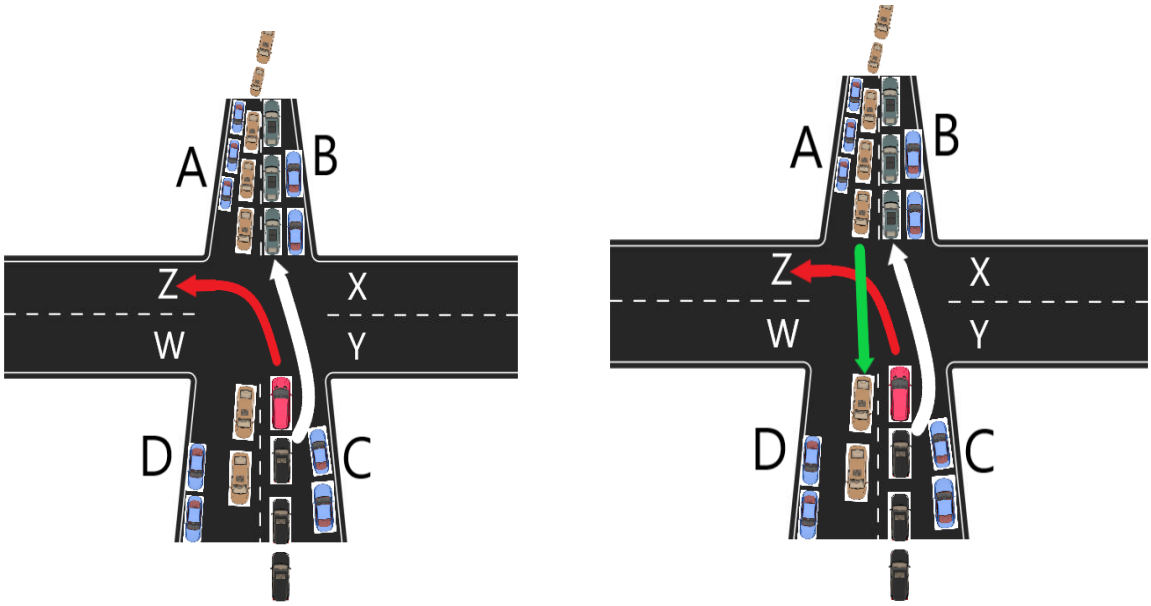

Figure 1.6 & 1.7: Test Case Scenario 1

In this scenario, there is traffic congestions because of the red car that is waiting for the opposing traffic to make the left turn. This is a very common case of traffic congestion that we can see at the intersection where there is no designated left turning lane. The Raspberry PI will recognize the red car is trying to turn left because it has been stopped for a certain period of time. In response, it will turn all the light at road A to red for a few seconds allowing the red car to safely turn left and the signal at road A will return to green.

Scenario 2:

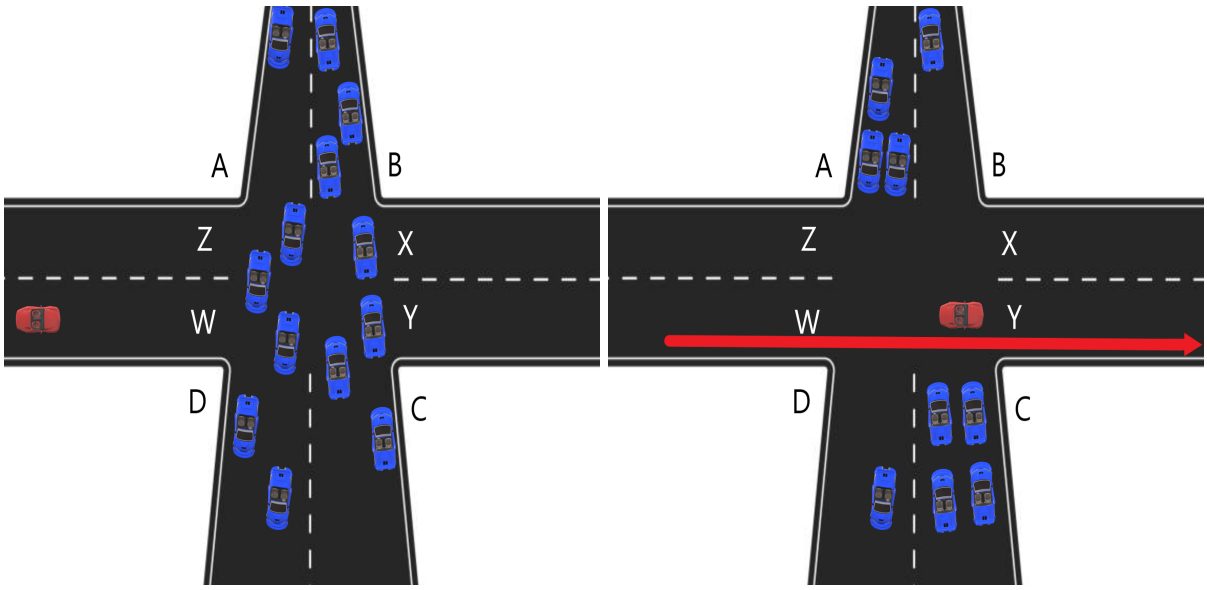

Figure 1.8 & 1.9: Test Case Scenario 2

In this scenario, the roads W,X,Y,Z are not busy whereas roads A,B,C,D are busy as it is the main road. Our system is able to update the traffic lights based on the information gathered in readl time. What it means is that if there is no traffic roads W,X,Y,Z the light will remain red. If our system detects an incoming car the signal at roads A,B,C,D will turn red and signal at roads W,X,Y,Z will turn green. Furthermore, if our system detects that there is no incoming traffic on roads W,X,Y,Z then it will turn the signal to red earlier than the usual timer.

Scenario 3:

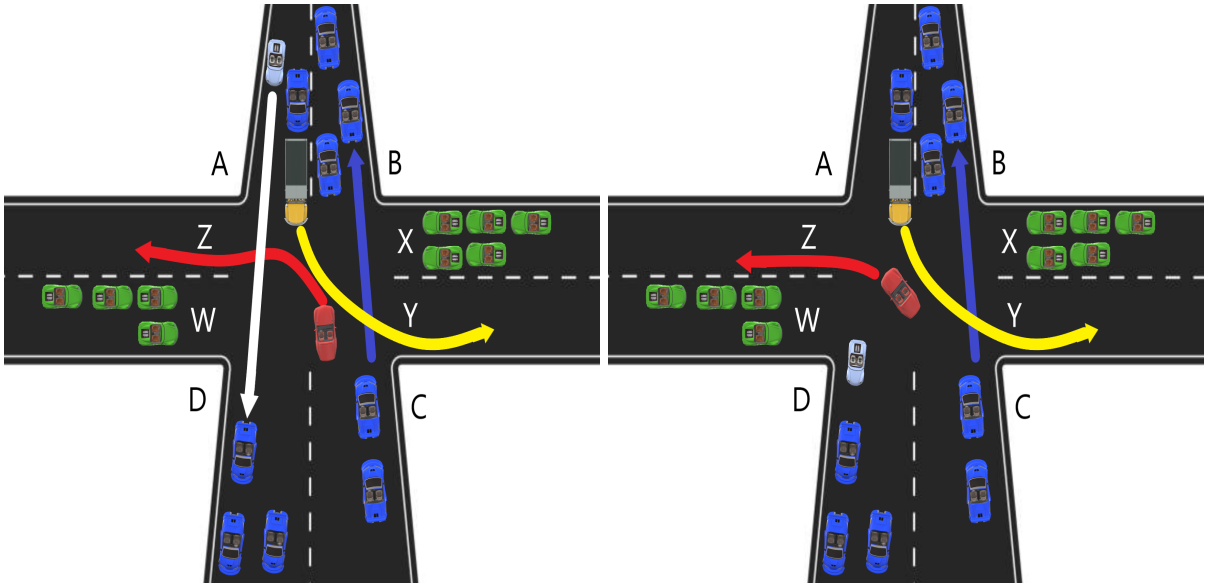

Figure 1.10 & 1.11: Test Case Scenario 3

In this scenario, we have a red car that is turning left on to road Z and yellow truck that is turning left on to road Y. The red car is having difficult seeing the incoming traffic because the yellow truck is blocking the line of sight. Our system will have a caution light which will light up if the system detects the the incoming traffic from road A so that the red car can turn left in a safe manner.

1.4 Algorithms

Our system will be using an Edge Server which is a server that run the processing at an edge location[1]. Our edge location will be traffic light intersection and the cars will be the nodes connected to it. The cars will be using an wifi module connected to a gps which transmit the data via the wifi connection to the edge server. The server then get the information from the wifi modules which is the GPS coordinates.

After receiving the coordination of the car, the server try to process the data using its stationary coordinates. So, if the car is in the 50 meter range of the traffic light, the edge server start processing the data using the coordinates again. What it will do is to determine where the car is coming from. There are four different state as follow by calculating the radians of latitude and longtitudes:

1. If (edge server lat - car lat) > 0 and (edge server lng - car lng) > 0
2. If (edge server lat - car lat) < 0 and (edge server lng - car lng) > 0
3. If (edge server lat - car lat) > 0 and (edge server lng - car lng) < 0
4. If (edge server lat - car lat) < 0 and (edge server lng - car lng) < 0

For each state there will be different motions assigned to it as follow:

1. North to South which we assign the number 13
2. South to North which we assign the number 31
3. East to West which we assign the number 24
4. West to East which we assign the number 42

The number is to get the direction of the car easier for the computer to store and proces which the first number it is where the car is and the second number is where the car is going to.

After finding the direction of each car and where it is going and using the scenarios above and integrating it into our system using the information processed from the cars, we can now change the state of the light based on the car location and direction.

2. Integration and/or Unit Testing

2.1. Edge Server configuration

By using the raspberry pi and the GPS module we made with ESP32 board, we made an network with wifi connection between the devices to transfer the location of the cars to the raspberry pi. First we worked on changing the raspberry pi network configuration using the hostapd and dnsmasq libraries. By configuring the raspberry pi into a router or Edge Server, now we can connect any device to it using the username and password we assigned to the raspberry pi. In the figure 2.1, you can see our progress in configuring the raspberry pi[2].

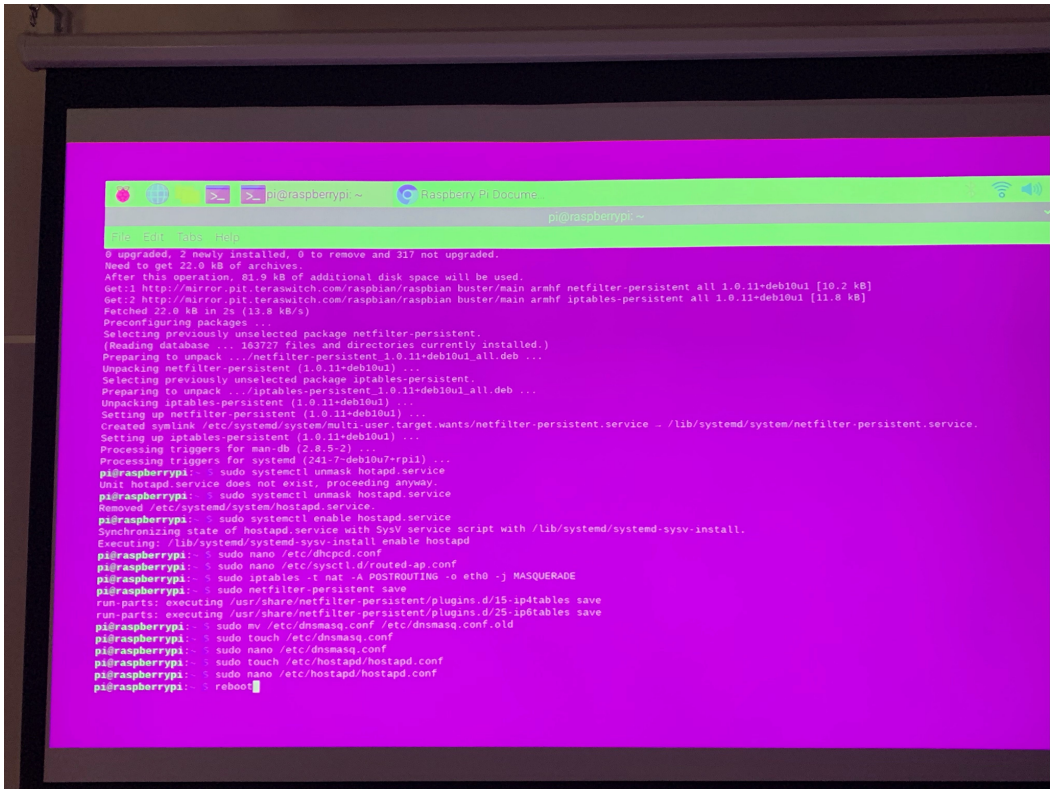

Fig 2.1:Process of turning the Raspberry Pi into a Edge Server

2.2. ESP32 wifi server and GPS Module Data Parsing

In the ESP32 design, first we worked on the connection between the GPS and the board to get the latitude and longitude. We used an online resource for creating our function to parse the data of the GPS which you can see in the figure 2.2. Then we moved on to code a program using the Arduino IDE to make the ESP32 into a wifi server. So, first we programmed the ESP32 to connect to the raspberry pi with the previous credentials we assigned to it. Then we got the IP address of the ESP32 to connect to it and receive the information. As you can see in the figure 2.3, that is the setup for our ESP32 simple server [3][4].

```
99  // Function to return GPS string
100 String displayInfo()
101 {
102   // Define empty string to hold output
103   String gpsdata = "";
104
105   // Get latitude and longitude
106   if (gps.location.isValid())
107   {
108     gpsdata = String(gps.location.lat(), 6);
109     gpsdata += (",");
110     gpsdata += String(gps.location.lng(), 6);
111   }
112   else
113   {
114     return "0";
115   }
116   // Return completed string
117   return gpsdata;
118 }
```

Fig. 2.2: Parsing Data Function

```
#include <TinyGPS++.h>
#include <SPI.h>
#include <WiFi.h>

static const uint32_t GPSBaud = 9600;
// String to hold GPS data
String gpstext;

// GPS write delay counter variables
// Change gpsttlcount as required
int gpscount = 0;
int gpsttlcount = 30;

// TinyGPS++ object
TinyGPSPlus gps;

const char* ssid     = "stsCapstone";
const char* password = "stsCapstone@";

WiFiServer server(80);

void setup()
{
  Serial.begin(115200);
  Serial2.begin(GPSBaud);

  Serial.println("GPS initialized...");
  delay(10);

  // We start by connecting to a WiFi network

  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  server.begin();
}
```

Fig 2.3: Setting up the server and connecting the ESP32 to our Edge Server

Continuing to create the server, figure 2.4, we made a simple HTTP webpage and post it on the given ip address of the server which for one of them was 10.20.1.30. So, after searching the ip address you would see the result of the GPS which you can see in the figure 2.5.

```
52  void loop()
53  {
54    while (WiFi.status() != WL_CONNECTED) {
55      delay(500);
56      Serial.print(".");
57    }
58
59    WiFiClient client = server.available();
60
61  if (client) {
62      Serial.println("New Client.");
63      String currentLine = "";
64      while (client.connected()) {
65        if (client.available()) {
66          char c = client.read();
67          if (c == '\n') {
68            if (currentLine.length() == 0) {
69              // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
70              // and a content-type so the client knows what's coming, then a blank line:
71              client.println("HTTP/1.1 200 OK");
72              client.println("Content-type:text/html");
73              client.println();
74              // the content of the HTTP response follows the header:
75                while (Serial2.available() > 0)
76                  if (gps.encode(Serial2.read()))
77
78                gpscount = gpscount + 1;
79                if (gpscount > gpsttlcount) {
80                  gpscount = 0;
81                }
82                gpstext = displayInfo();
83
84              client.print(gpstext);
85              break;
86            } else {
87              currentLine = "";
88            }
89          } else if (c != '\r') {
90            currentLine += c;
91          }
92        }
93      }
94      client.stop();
95      Serial.println("Client Disconnected.");
96    }
97  }
```

Fig 2.4: Integration of simple webpage and sending the gps lat and lng

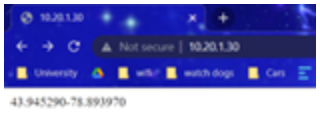

Fig 2.5: Result of searching the ip address by staying connected to the Edge Server which this location is the place we tested our system and it's the university's entrance intersection

2.3. Getting the information from the cars to the Edge Server

Now that our system was operational and we could get the information from the car and its gps coordinates, we needed to make a program to read the data of the gps on the raspberry pi and parse the information based on what we have. So, as you can see in the figure 2.6, the function getCoordination(IP) gets the information using the ip address of the cars which then using the requests syntax, we get out coordination. For testing if it works or not, we were in one of the group members car and by driving toward the intersection on university campus we tested our system which was successful. In the code we calculated the distance of the car in relation to the coordination of the stationary traffic light then made an if statement that if the car is less than 50 meters close to the traffic light then use the control function to see where is the car coming from. For finding this out, we had to get the radians of the lat and lng for the traffic light and the car, then subtract them from each other. Furthermore by making four if clauses, and comparing them to be greater than zero and less than zero, we found out where our car is headed to which in the figure 2.6 you can see it says from “south to north”.

```
import requests
from time import sleep as t
from math import sin, cos, sqrt, atan2, radians

carNum1 = []
carIP = "http://10.20.1.30"
CarCoord = []
stationLat = 43.94539989984547
stationLng = -78.89245982725625


def getCoordination(IP):

    try:
        carNum = requests.get(IP, timeout=0.5).text
    except Exception:
        return -1
    if carNum == "00":
        return -1
    else:
        return carNum


def parseCoordination(Coord):
    lat = ""
    lng = ""
    for i in range(0, len(Coord)):
        if Coord[i] == ",":
            saveI = i+1
            break
        lat += Coord[i]
    for i in range(saveI, len(Coord)):
        lng += Coord[i]
    return float(lat), float(lng)


def getDestination(coords):...


def greenLight():...


def checkDirection(coord):
    lat1 = radians(coord[0])
    lon1 = radians(coord[1])
    lat2 = radians(stationLat)
    lon2 = radians(stationLng)
    print("Car coordination are " +
          str(coord[0]) + ", " + str(coord[1]))
    print("Light coordination are "
          + str(stationLat) + ", " + str(stationLng))

    if lat2-lat1 < 0 and lon2-lon1 < 0:
        print("north to south")
        return 13
    elif lat2-lat1 > 0 and lon2-lon1 < 0:
        print("east to west")
        return 24
    elif lat2-lat1 > 0 and lon2-lon1 > 0:
        print("Car coming from south to north")
        return 31
    elif lat2-lat1 < 0 and lon2-lon1 > 0:
        print("west to east")
        return 42
    else:
        return -1


def control(coord):
    dir = checkDirection(coord)
    if dir == 31:
        greenLight()


while True:
    if getCoordination(carIP) != -1:
        CarCoord = getCoordination(carIP)
        NewCoord = parseCoordination(CarCoord)
        distance = getDestination(NewCoord)
        while distance < 50:
            print("The distance is below 50 "
                  "meters and is " + str(distance) + "meters")
            control(NewCoord)

    t(0.5)
```

```
C:\Users\SaroK\PycharmProjects\capstonePI\venv\Scripts\python.exe C:\Users\SaroK\PycharmProjects\capstonePI\main.py
The distance is below 50 meters and is 42.6924921605153meters
Car coordination are 43.94509057466029, -78.89277543532853
Light coordination are 43.94539989984547, -78.89245982725625
Car coming from south to north
light is green
```

Fig 2.6: Getting the car coordinates and parse the information using the raspberry pi

2.4. Testing of the project

In this part we had some testing to do to make sure our system is operating correctly before doin anytest on the raspberry pi or the traffic light at university. As figure 2.7 shows we did this correctly and had to move on to make the raspberry pi and ESP32 connection.

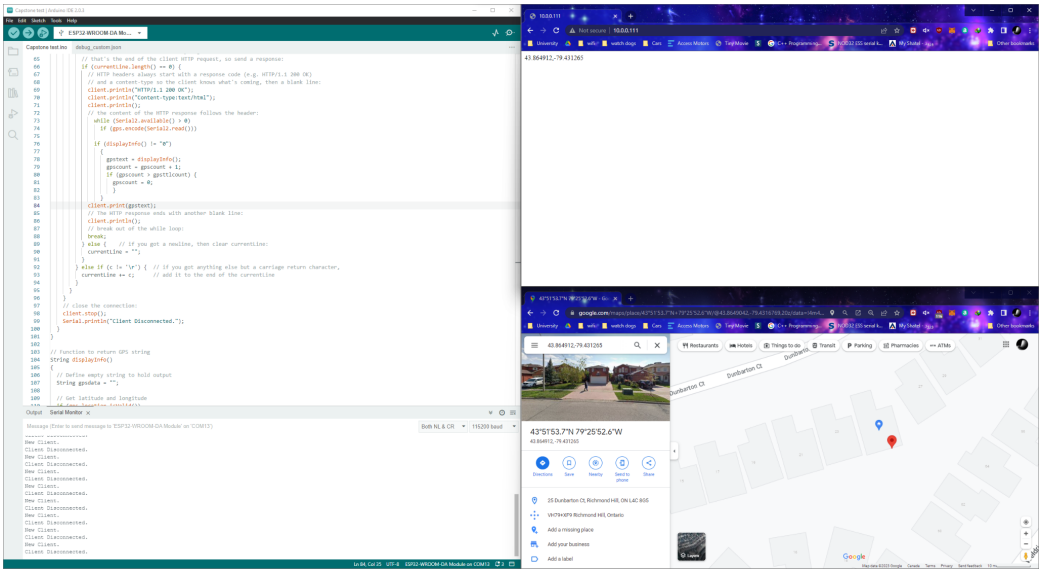

Fig 2.7: Testing the GPS with ESP32 at home

For physical demonstration of the traffic lights, we used some toy cars and 3D printed traffic lights with LED to test some of our scenarios.

Fig 2.8: Testing our scenarios using toy cars

3. Updated Project Plan

Design of Smart Traffic Signals for Autonomous Transportation and Connected Communities in Smart Cities

Project Start Date: September 14, 2022

16

TASK	PROGRESS	START	END
R1 - Fall 2022	100%	September 14, 2022	October 24, 2022
Project Identification	100%	September 14, 2022	September 19, 2022
Background and Research Revi	100%	September 19, 2022	September 26, 2022
Design Process	100%	September 22, 2022	October 2, 2022
Scenarios and Use Cases	100%	September 26, 2022	October 10, 2022
Stakeholder Requirements	100%	October 3, 2022	October 10, 2022
Acceptance Tests	100%	October 10, 2022	October 17, 2022
Project Plan	100%	October 17, 2022	October 22, 2022
R2 - Fall 2022	100%	October 23, 2022	November 13, 2022
Concept Generation and Analys	100%	October 23, 2022	October 30, 2022
Coceptual System Design	100%	October 30, 2022	November 6, 2022
Definition of Integration Tests	100%	November 6, 2022	November 10, 2022
Estimated Cost of Project	100%	November 10, 2022	November 13, 2022
Updated Project Plan	100%	October 23, 2022	October 25, 2022
R1 - Winter 2022	100%	January 9, 2023	February 18, 2023
Detail Design	100%	January 9, 2023	February 8, 2023
Integration and/or Unit Testing	100%	February 8, 2023	February 18, 2023
Updated Project Plan	100%	January 9, 2023	January 11, 2023

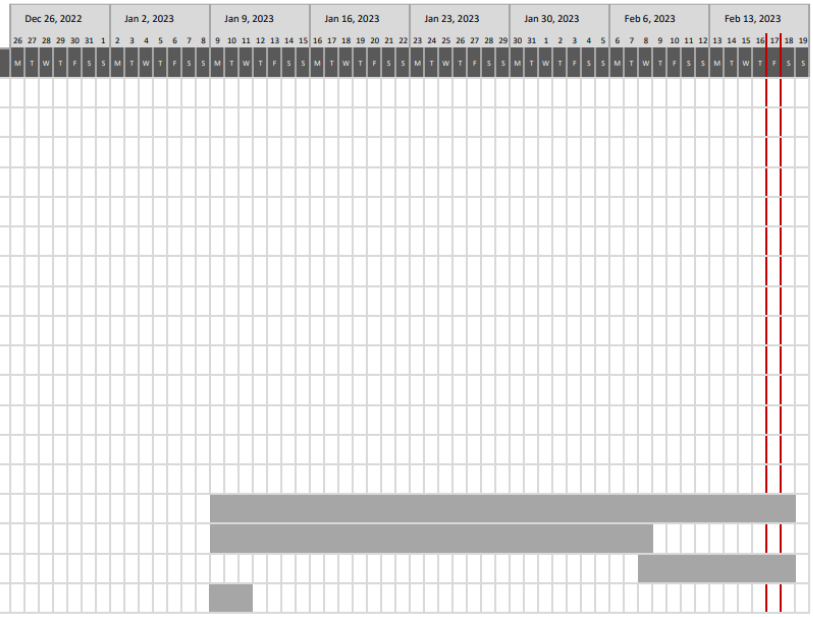

Figure 3.1: Shows our project updated plan

4. Contribution matrix

Table 4.1: Summary of work % contributed by each team member for Report 1

Task	People		
Report 1	Saro Karimi	Tirth Patel	Vatsal Patel
	33.33%	33.33%	33.33%

Each team member worked really hard on the project. Each task was divided between the team mates. Here is a list of who did what:

- Saro Karimi
    1. Designing the algorithm
    2. Creating the edge server and gps connection
    3. Desging and creating the traffic light you can see in the figure 2.8
- Tirth Patel
    1. Defining the scenarios and explaining them thoroughly
    2. Desiging the test cases.
    3. Programming the algorithm in the Edge server
- Vatsal Patel
    1. Researching about the modules we need for the project
    2. Finding some of the modules we could not find online by contacting one of our lab advisors which we were able to get the raspberry pi and the ESP32 modules
    3. Making the electronic connection for the modules

These are just some of the works each team member did and there are more things we did behind the scenes.

5. References

[1] https://stlpartners.com/articles/edge-computing/what-is-an-edge-server/

[2] https://www.raspberrypi.com/documentation/computers/configuration.html

[3] https://dronebotworkshop.com/esp32-intro/

[4] https://dronebotworkshop.com/using-gps-modules/

[5] https://www.espressif.com/en/products/modules/esp8266’

[6] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/