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

R2: Acceptance Test Demonstration Report

Group#: 8
Section CRN#:73459
Advisor: Dr. Hossam Gaber
Coordinator: Dr. Vijay Sood
Due Date: March 8th, 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 acceptance and performance testing, we defined them what it is. Followed by that we have written about five to six test cases. So in each test case, we have described the test cases first, then we provided some steps that we need to take for each test case. After taking some steps we have provided details about what will be the expected outcomes. And to support our expected outcomes we have attached our code and the results we got after running the codes. But in text cases three and four, we have attached the flowchart and in the flowchart we have described our test case in detail, outlining what this test case is and how it will work if we execute it properly. Then in the end we have our contribution matrix, defining the contribution that has been made by each team members. To conclude, by doing this project and working on it physically. It gave us a better understanding of the overall project.

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 input regarding the reports and the overall project.

Table of content

List of Tables

List of Figures

Abbreviations

GPS	Global Positioning System
IP	Internet Protocol Address
STS	Smart Traffic System

1. Acceptance and/or Performance Testing

1.1 Introduction

An essential part of an efficient transportation system are smart traffic systems (STS). Real-time data is used by these signals to modify signal timings and enhance traffic movement. To make sure that they are functioning correctly and up to performance standards, acceptance and performance testing must be done on smart traffic systems prior to deployment.

A smart traffic system is put through acceptance testing to see if it satisfies the criteria and is prepared for implementation. To make sure it is operating as anticipated and satisfies the performance requirements, the smart traffic system is put through approval testing in a controlled setting.

- Requirements and specifications: As part of the acceptance testing process, it is first checked to see if the smart traffic system satisfies the performance criteria. To guarantee that the signal satisfies the intended functionality, accuracy, and reliability, it should be evaluated against the performance criteria.
- Simulation testing: The smart traffic signal is usually tested during acceptance testing in a regulated environment, like a testing facility or a closed intersection. This makes it possible for evaluators to model different traffic situations and check that the indicator is operating as intended.
- Integration testing: Modern traffic lights can be combined with other traffic management tools, like central control systems or traffic sensors. The smart traffic signal's ability to interact and cooperate with other systems is confirmed through integration testing.
- User acceptance testing: User approval testing, which is the final step, entails confirming that the smart traffic signal satisfies the requirements of end users, including motorists, walkers, and transit organizations.

Performance testing is a sort of testing that evaluates how well a smart traffic system actually performs in actual use. During this testing, any potential problems, such as traffic gridlock or unforeseen traffic patterns, are found before the signal is actually implemented.

- Real-world testing: smart traffic system performance testing entails putting the signal through its paces in a real-world setting, like a functioning intersection. This makes it possible for users to watch how the signal operates in actual traffic and evaluate how well it improves traffic movement.
- Data collection and analysis: Data is gathered on crucial parameters like cycle length, green light, red light, intersection delay, queue length, traffic flow, and pedestrian safety in order to assess the efficacy of the smart traffic system. The efficacy of the signal is assessed using this data, and potential growth areas are noted.

- Continuous testing and optimization: The efficacy of smart traffic signals can be constantly assessed and improved. To enhance data gathering and processing, this may entail modifying signal timings or installing extra sensors.

To ensure that smart traffic systems are operating efficiently and enhancing traffic movement, acceptance and performance testing are crucial. This could lead to a more effective transit system that is safer and less congested.

1.2 Test Cases:

1.2.1 Test Case 1

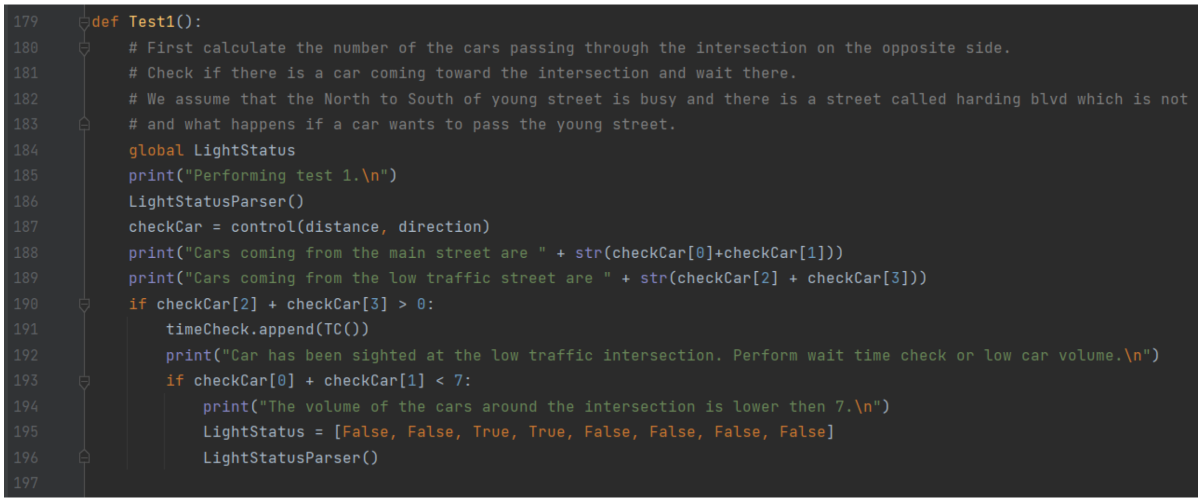
The system needs to be able to effectively control traffic flow. To determine whether this works, we compare the actual time to the usual amount of time it takes for the light to turn green upon arriving at an intersection where the light is red. In this test, we are also testing if our algorithm is successful or not. The test is deemed successful if the system meets or beats the anticipated wait time.

Test Steps:

- Send vehicles equipped with GPS and wifi modules through a simulated traffic intersection where the light is red.
- Observe the STS's response to the incoming vehicles.

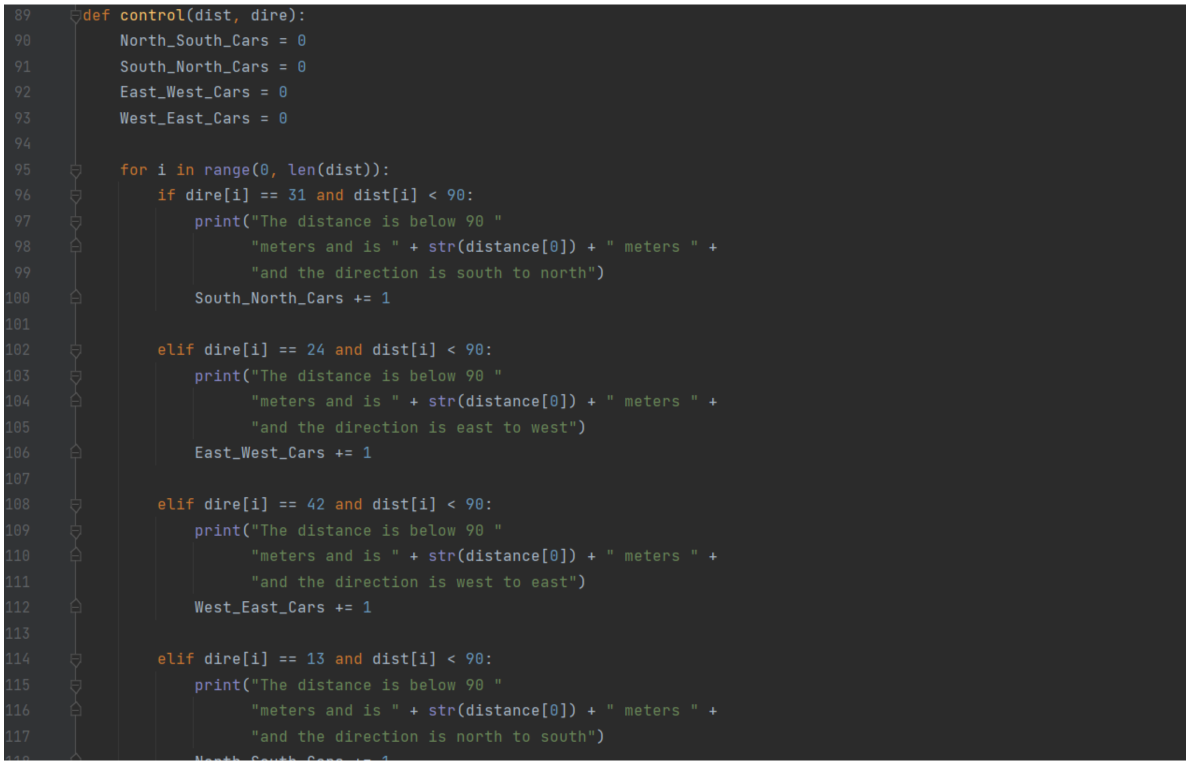
Expected Outcome:

- The STS should be able to detect the vehicle arriving at the red light and depending on the traffic adjust the timing of the signals.



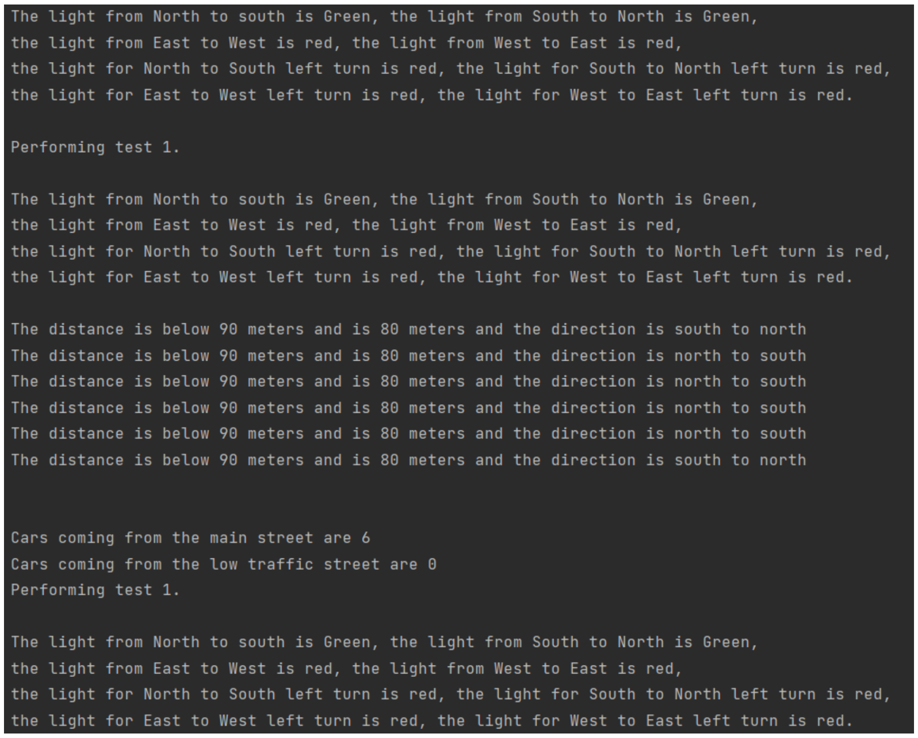
```python
def Test1():
    # First calculate the number of the cars passing through the intersection on the opposite side.
    # Check if there is a car coming toward the intersection and wait there.
    # We assume that the North to South of young street is busy and there is a street called harding blvd which is not
    # and what happens if a car wants to pass the young street.
    global LightStatus
    print("Performing test 1.\n")
    LightStatusParser()
    checkCar = control(distance, direction)
    print("Cars coming from the main street are " + str(checkCar[0]+checkCar[1]))
    print("Cars coming from the low traffic street are " + str(checkCar[2] + checkCar[3]))
    if checkCar[2] + checkCar[3] > 0:
        timeCheck.append(TC())
        print("Car has been sighted at the low traffic intersection. Perform wait time check or low car volume.\n")
        if checkCar[0] + checkCar[1] < 7:
            print("The volume of the cars around the intersection is lower then 7.\n")
            LightStatus = [False, False, True, True, False, False, False, False]
            LightStatusParser()

```

Figure 1.1: Code for testing process of the algorithm and scenario 1



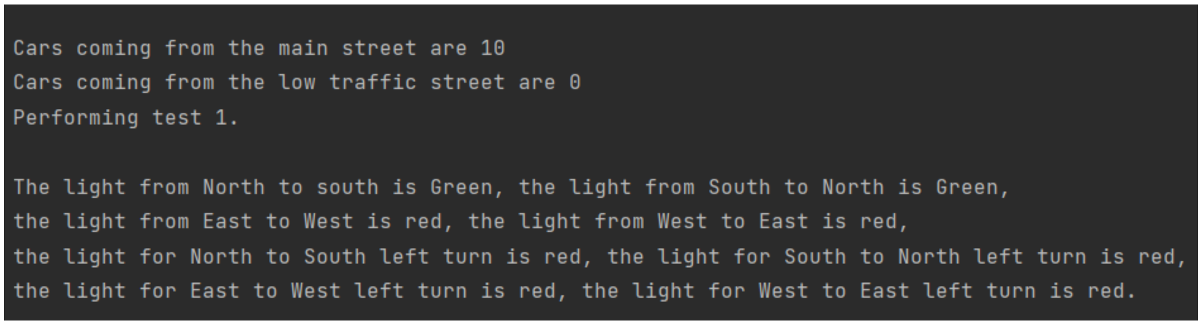
```python
def control(dist, dire):
    North_South_Cars = 0
    South_North_Cars = 0
    East_West_Cars = 0
    West_East_Cars = 0

    for i in range(0, len(dist)):
        if dire[i] == 31 and dist[i] < 90:
            print("The distance is below 90 "
                  "meters and is " + str(distance[0]) + " meters " +
                  "and the direction is south to north")
            South_North_Cars += 1

        elif dire[i] == 24 and dist[i] < 90:
            print("The distance is below 90 "
                  "meters and is " + str(distance[0]) + " meters " +
                  "and the direction is east to west")
            East_West_Cars += 1

        elif dire[i] == 42 and dist[i] < 90:
            print("The distance is below 90 "
                  "meters and is " + str(distance[0]) + " meters " +
                  "and the direction is west to east")
            West_East_Cars += 1

        elif dire[i] == 13 and dist[i] < 90:
            print("The distance is below 90 "
                  "meters and is " + str(distance[0]) + " meters " +
                  "and the direction is north to south")
```

Figure 1.2: Analyzing the received geo coordinates from the vehicles and getting directions and distance of them related to the intersection



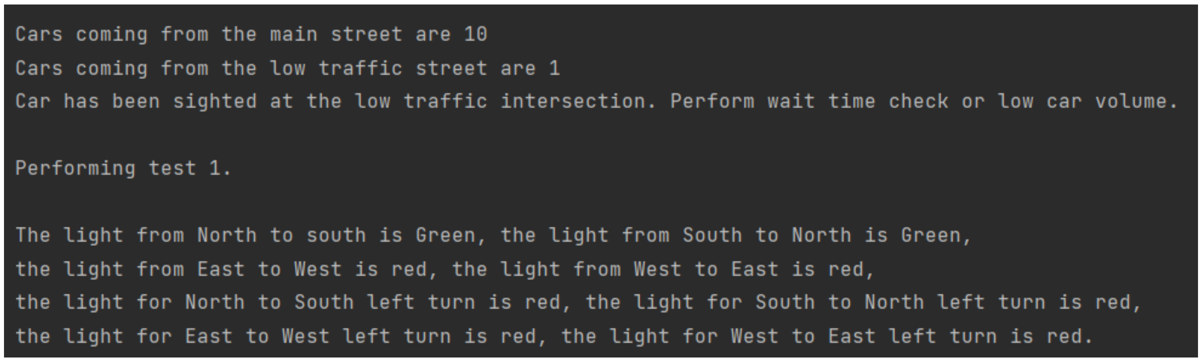
```
The light from North to south is Green, the light from South to North is Green,
the light from East to West is red, the light from West to East is red,
the light for North to South left turn is red, the light for South to North left turn is red,
the light for East to West left turn is red, the light for West to East left turn is red.

Performing test 1.

The light from North to south is Green, the light from South to North is Green,
the light from East to West is red, the light from West to East is red,
the light for North to South left turn is red, the light for South to North left turn is red,
the light for East to West left turn is red, the light for West to East left turn is red.

The distance is below 90 meters and is 80 meters and the direction is south to north
The distance is below 90 meters and is 80 meters and the direction is north to south
The distance is below 90 meters and is 80 meters and the direction is north to south
The distance is below 90 meters and is 80 meters and the direction is north to south
The distance is below 90 meters and is 80 meters and the direction is north to south
The distance is below 90 meters and is 80 meters and the direction is south to north


Cars coming from the main street are 6
Cars coming from the low traffic street are 0
Performing test 1.

The light from North to south is Green, the light from South to North is Green,
the light from East to West is red, the light from West to East is red,
the light for North to South left turn is red, the light for South to North left turn is red,
the light for East to West left turn is red, the light for West to East left turn is red.
```

Figure 1.3: First Output of the program testing with having the North to South and vise versa green



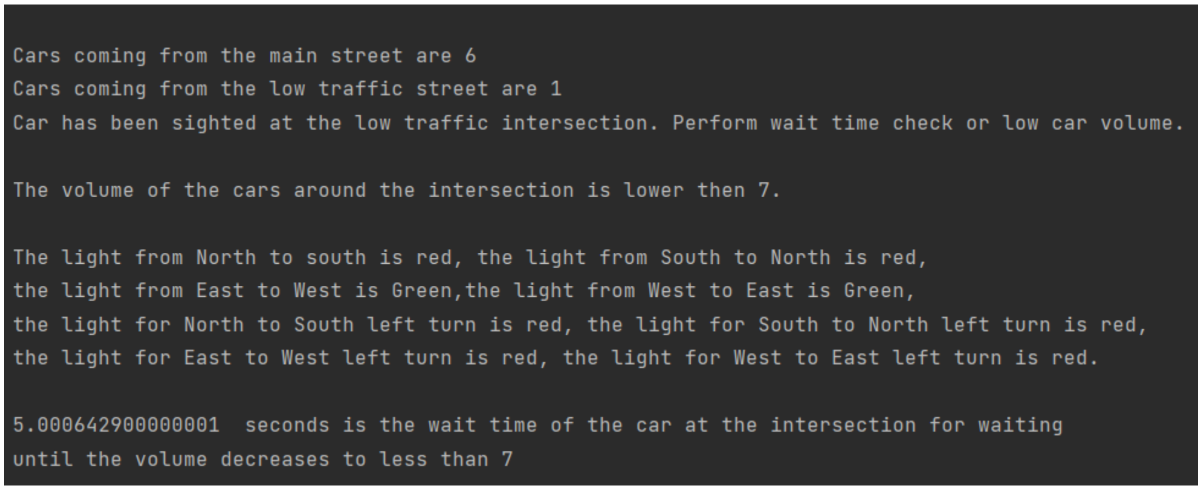
```
Cars coming from the main street are 10
Cars coming from the low traffic street are 0
Performing test 1.

The light from North to south is Green, the light from South to North is Green,
the light from East to West is red, the light from West to East is red,
the light for North to South left turn is red, the light for South to North left turn is red,
the light for East to West left turn is red, the light for West to East left turn is red.
```

Figure 1.4: Output of getting more vehicles at the intersection by calculating the connected IP addresses to the Edge Server



```
Cars coming from the main street are 10
Cars coming from the low traffic street are 1
Car has been sighted at the low traffic intersection. Perform wait time check or low car volume.

Performing test 1.

The light from North to south is Green, the light from South to North is Green,
the light from East to West is red, the light from West to East is red,
the light for North to South left turn is red, the light for South to North left turn is red,
the light for East to West left turn is red, the light for West to East left turn is red.
```

Figure 1.5: Output shows that a vehicle has been sighted at the intersection's low traffic side but can't perform the light status change since there are more than 7 vehicles appeared at the intersection. Timer has been set to determine the change of the light if the vehicles in the main street remain more than 7 for around 90 seconds.



```
Cars coming from the main street are 6
Cars coming from the low traffic street are 1
Car has been sighted at the low traffic intersection. Perform wait time check or low car volume.

The volume of the cars around the intersection is lower then 7.

The light from North to south is red, the light from South to North is red,
the light from East to West is Green,the light from West to East is Green,
the light for North to South left turn is red, the light for South to North left turn is red,
the light for East to West left turn is red, the light for West to East left turn is red.

5.000642900000001  seconds is the wait time of the car at the intersection for waiting
until the volume decreases to less than 7
```

Figure 1.6: The number of vehicles on the main street decreased to less than 7 in 5 seconds. The process of the traffic light to turn green starts.

1.2.2 Test Case 2

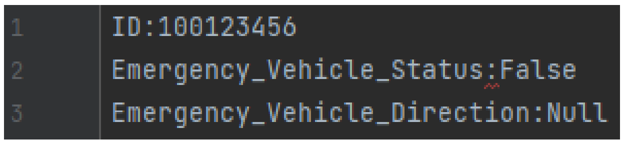
In this test, we want to ensure that the smart traffic system can detect emergency vehicles, such as ambulances and fire trucks, and prioritize their movement through traffic. The test is deemed successful if the STS is successfully able to recognize an emergency vehicle and prioritize its movement through the intersection.

Test Steps:

- Send an emergency vehicle (e.g., an ambulance) equipped with GPS and a wifi module through a simulated traffic intersection.
- Observe the STS's response to the emergency vehicle.
- Verify that the emergency vehicle's location is detected by the smart traffic system.
- Verify that the STS adjusts the traffic signals to prioritize the emergency vehicle's movement through the intersection.
- Verify that the emergency vehicle is able to move through the intersection with minimal delay.

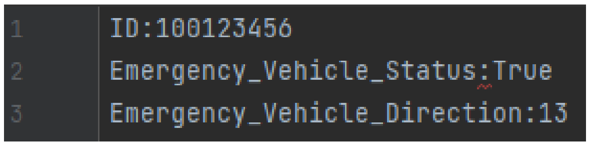
Expected Outcome:

- The STS should detect the emergency vehicle's location and prioritize its movement through the intersection.
- The system should adjust the traffic signals to clear the emergency vehicle's path through the intersection.
- The emergency vehicle should be able to move through the intersection with minimal delay.



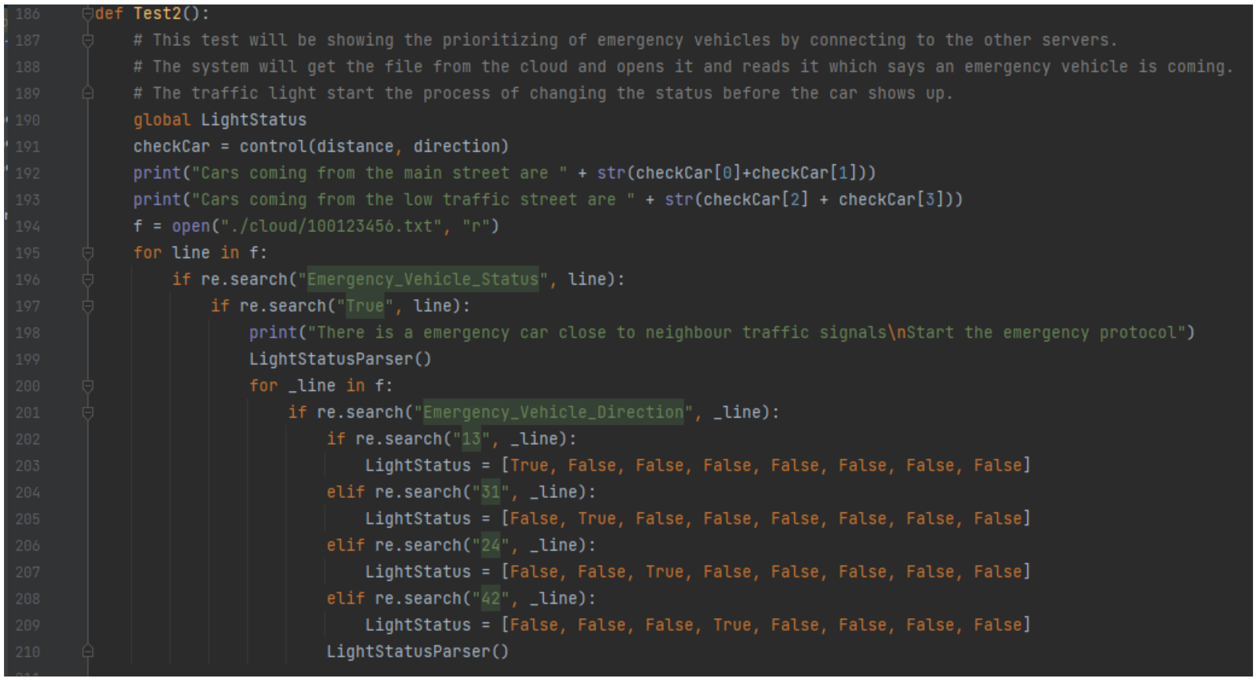
```
ID:100123456
Emergency_Vehicle_Status:False
Emergency_Vehicle_Direction:Null
```

Figure 1.7: Other traffic light output on the server before the emergency vehicle is sighted



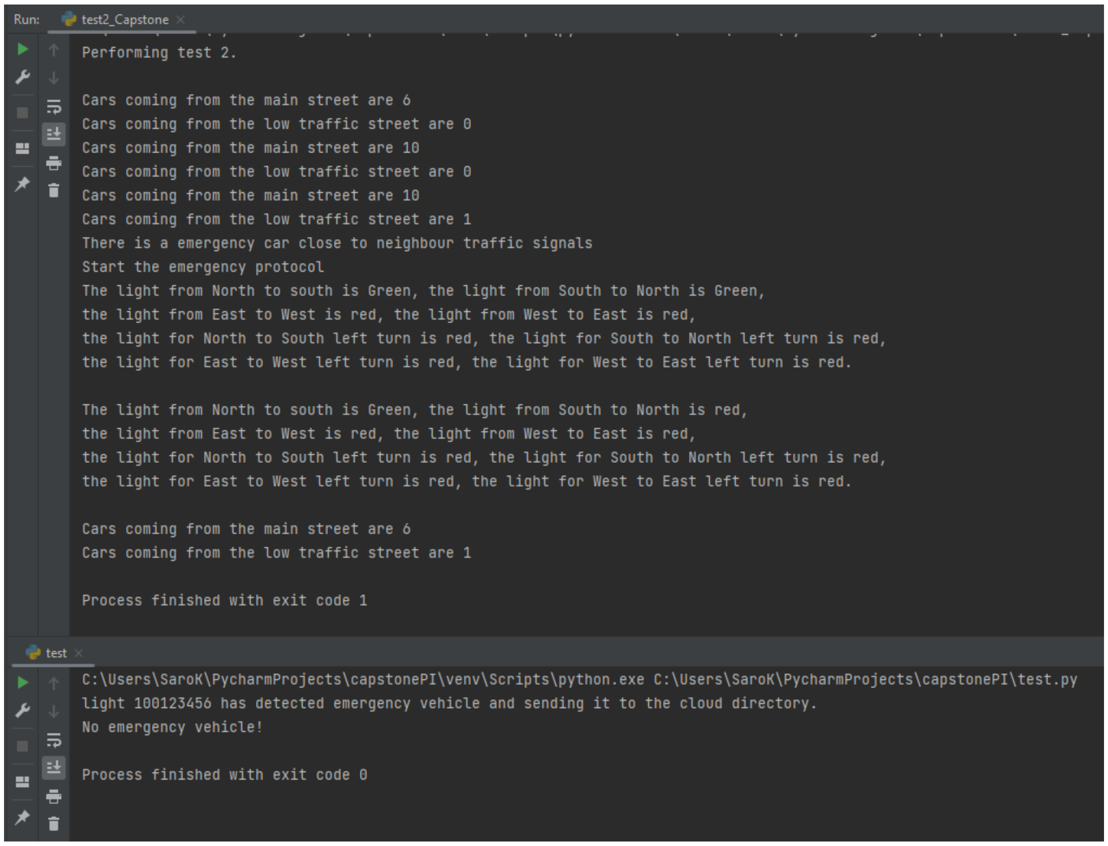
```
ID:100123456
Emergency_Vehicle_Status:True
Emergency_Vehicle_Direction:13
```

Figure 1.8: Other traffic light output on the server after the emergency vehicle is sighted



```python
def Test2():
    # This test will be showing the prioritizing of emergency vehicles by connecting to the other servers.
    # The system will get the file from the cloud and opens it and reads it which says an emergency vehicle is coming.
    # The traffic light start the process of changing the status before the car shows up.
    global LightStatus
    checkCar = control(distance, direction)
    print("Cars coming from the main street are " + str(checkCar[0]+checkCar[1]))
    print("Cars coming from the low traffic street are " + str(checkCar[2] + checkCar[3]))
    f = open("./cloud/100123456.txt", "r")
    for line in f:
        if re.search("Emergency_Vehicle_Status", line):
            if re.search("True", line):
                print("There is a emergency car close to neighbour traffic signals\nStart the emergency protocol")
                LightStatusParser()
                for _line in f:
                    if re.search("Emergency_Vehicle_Direction", _line):
                        if re.search("13", _line):
                            LightStatus = [True, False, False, False, False, False, False, False]
                        elif re.search("31", _line):
                            LightStatus = [False, True, False, False, False, False, False, False]
                        elif re.search("24", _line):
                            LightStatus = [False, False, True, False, False, False, False, False]
                        elif re.search("42", _line):
                            LightStatus = [False, False, False, True, False, False, False, False]
                        LightStatusParser()
```

Figure 1.9: Code for testing process of the emergency algorithm and scenario



```
Performing test 2.

Cars coming from the main street are 6
Cars coming from the low traffic street are 0
Cars coming from the main street are 10
Cars coming from the low traffic street are 0
Cars coming from the main street are 10
Cars coming from the low traffic street are 1
There is a emergency car close to neighbour traffic signals
Start the emergency protocol
The light from North to south is Green, the light from South to North is Green,
the light from East to West is red, the light from West to East is red,
the light for North to South left turn is red, the light for South to North left turn is red,
the light for East to West left turn is red, the light for West to East left turn is red.

The light from North to south is Green, the light from South to North is red,
the light from East to West is red, the light from West to East is red,
the light for North to South left turn is red, the light for South to North left turn is red,
the light for East to West left turn is red, the light for West to East left turn is red.

Cars coming from the main street are 6
Cars coming from the low traffic street are 1

Process finished with exit code 1
```

```
C:\Users\SaroK\PycharmProjects\capstonePI\venv\Scripts\python.exe C:\Users\SaroK\PycharmProjects\capstonePI\test.py
light 100123456 has detected emergency vehicle and sending it to the cloud directory.
No emergency vehicle!

Process finished with exit code 0
```

Figure 1.10: Output of two traffic lights, that one sends the information to the cloud of sighting an emergency vehicle and the other one is receiving the information and starts the process of turning the light to green or make it stay green before reaching the light.

1.2.3 Test Case 3

In this test, we would like to test that the STS can detect a vehicle trying to turn left for more than 15 seconds and adjust the signal so that it can turn left. This is a very common scenario testing that occurs when there is no designated left lane.

Test Steps:

- Send a vehicle (e.g., a car) equipped with GPS and Wifi devices through a simulated traffic intersection.
- Instruct the vehicle to make a left turn.
- Observe the smart traffic system's response to the left-turning vehicle.
- Verify that the vehicle's location and direction of travel are detected by the smart traffic system.
- Verify that the smart traffic system adjusts the traffic signals to recognize the vehicle turning left and waiting for more than 15 seconds
- Verify that the left-turning vehicle is able to complete its turn with minimal delay.

Expected Outcome:

- The STS should detect the left-turning vehicle's location and direction of travel.
- The STS should adjust the traffic signals to prioritize the left-turning vehicle's movement through the intersection.
- The left-turning vehicle should be able to complete its turn with minimal delay.

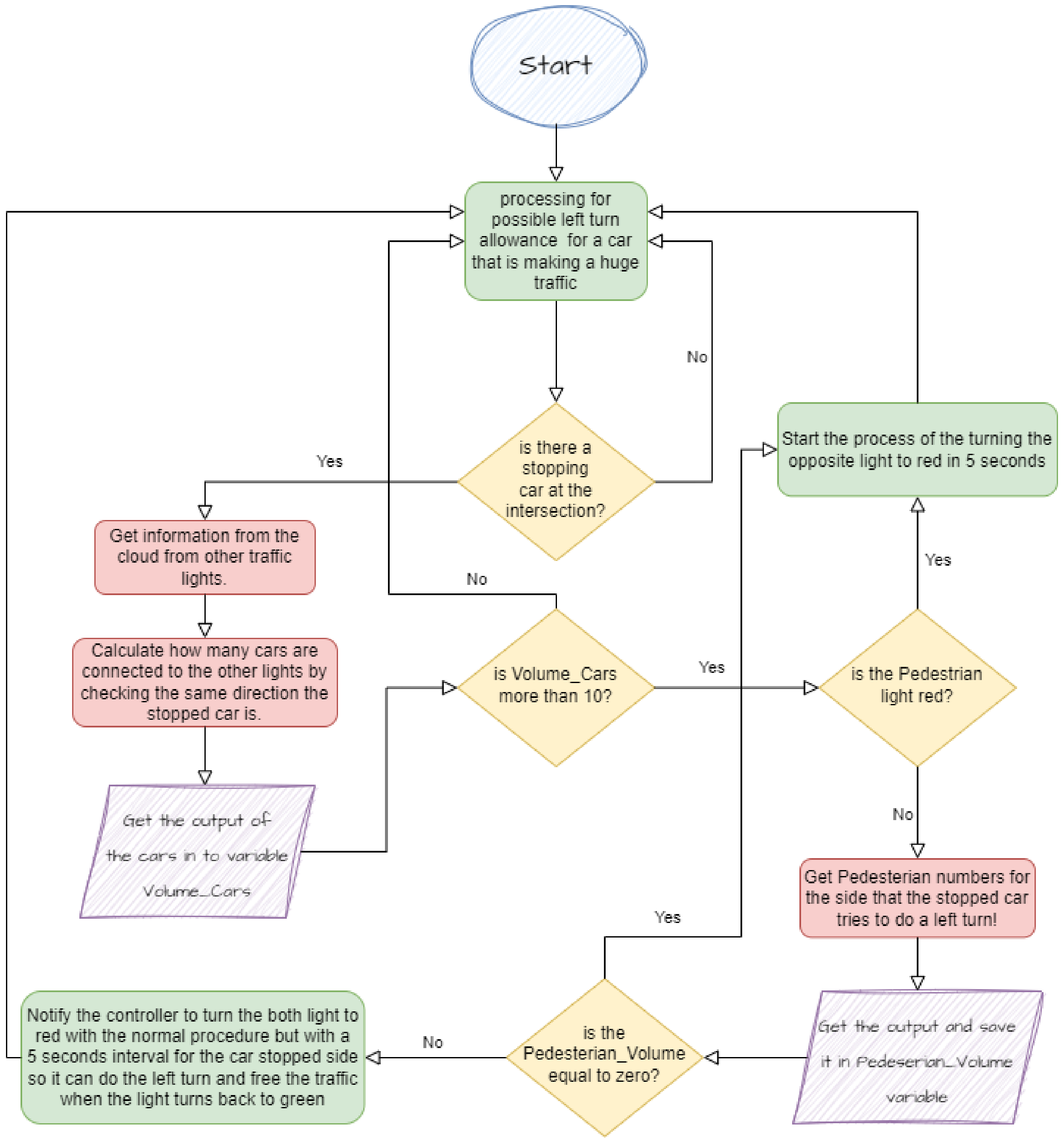

Start
processing for possible left turn allowance for a car that is making a huge traffic
is there a stopping car at the intersection?
No
Yes
Get information from the cloud from other traffic lights.
Calculate how many cars are connected to the other lights by checking the same direction the stopped car is.
Get the output of the cars in to variable Volume_Cars
No
is Volume_Cars more than 10?
Yes
is the Pedestrian light red?
Yes
Start the process of the turning the opposite light to red in 5 seconds
No
Get Pedesterian numbers for the side that the stopped car tries to do a left turn!
Get the output and save it in Pedeserian_Volume variable
is the Pedestrian_Volume equal to zero?
Yes
No
Notify the controller to turn the both light to red with the normal procedure but with a 5 seconds interval for the car stopped side so it can do the left turn and free the traffic when the light turns back to green

Figure 1.11: Flowchart of the Scenario 2, that explains the process of sighting a stopped car at an intersection and wants to do a left turn and it is making a big traffic behind it.

1.2.4 Test Case 4

The STS should be able to correctly detect incoming vehicles and change the caution light from red to green depending on the incoming vehicles. The test is deemed successful if the STS is successfully able to detect the vehicle coming from the opposite lane and change the caution light.

Test Steps:

- Send a vehicle (e.g., car) equipped with GPS and Wifi devices through a simulated traffic intersection.
- Instruct the vehicle to make a left turn.
- Send another vehicle in the opposite direction of the left-turning vehicle
- Verify STS is able to detect the incoming vehicle and change the caution light.

Expected Outcome:

- The STS should detect the left-turning vehicle's location and direction of travel.
- The STS should detect the incoming vehicle in the opposite lane and change the caution light.

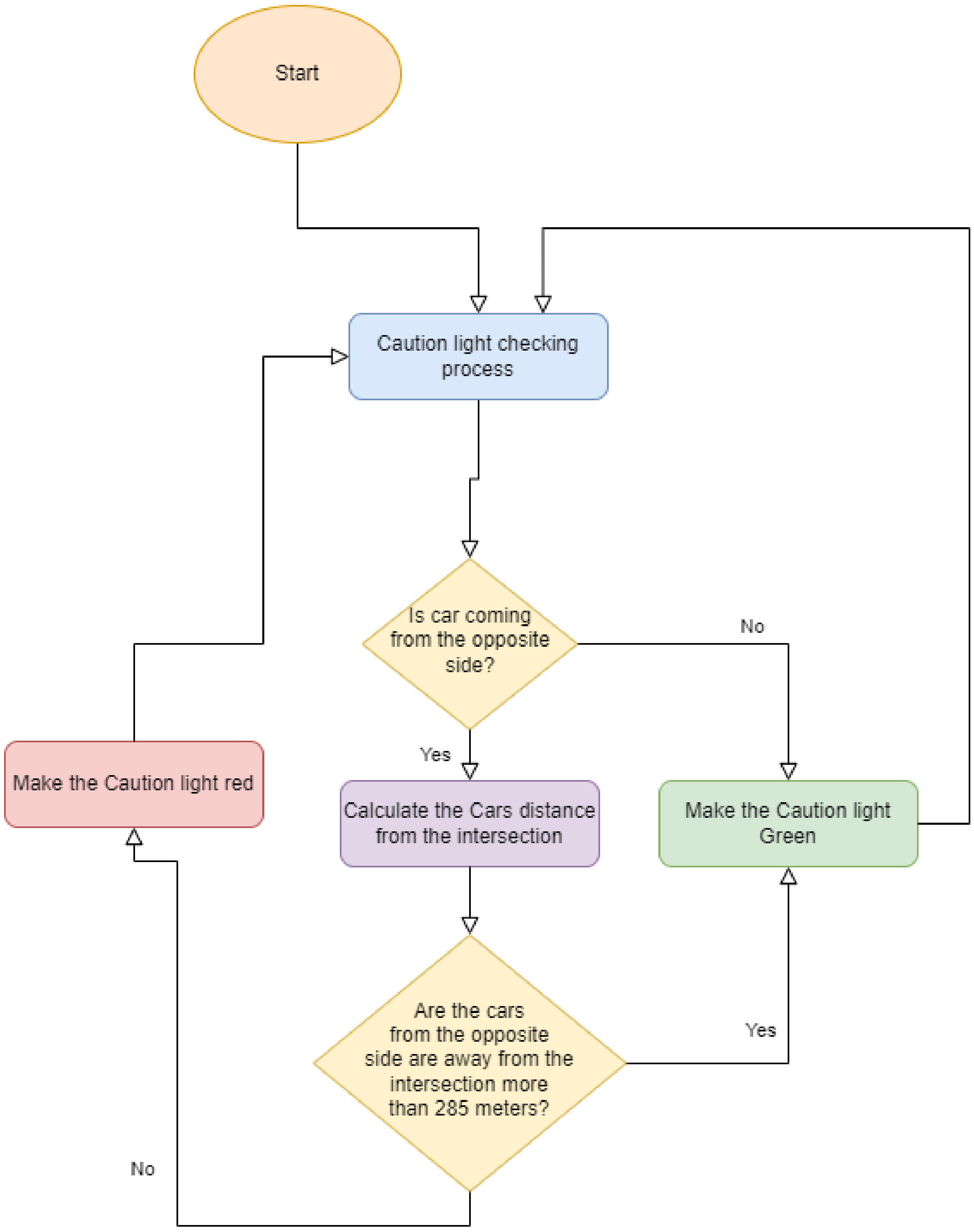

Start
Caution light checking process
Is car coming from the opposite side?
No
Yes
Make the Caution light red
Calculate the Cars distance from the intersection
Make the Caution light Green
Are the cars from the opposite side are away from the intersection more than 285 meters?
Yes
No

Figure 1.12: Flowchart and algorithm of scenario 3 which light will indicate that a left turn is safe or not

1.2.5 Test Case 5

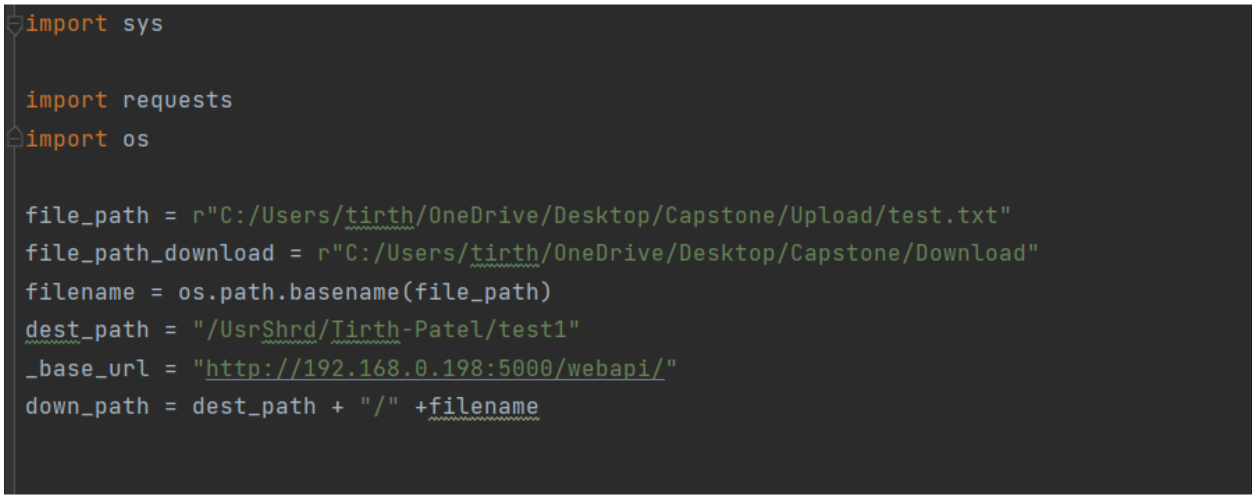
In this test, we would like to check the connectivity of the edge server to the SESL cloud. The test is deemed successful if the edge server is successfully able to connect to the SESL cloud.

Test Steps:

- Attempt to establish a connection between the edge server and the cloud platform.
- Verify that the connection is established successfully.
- Send the GPS coordinates and direction of the vehicles from the edge server to the cloud platform.
- Verify that the cloud platform receives the test data.
- Send a larger amount of test data from the edge server to the cloud platform.

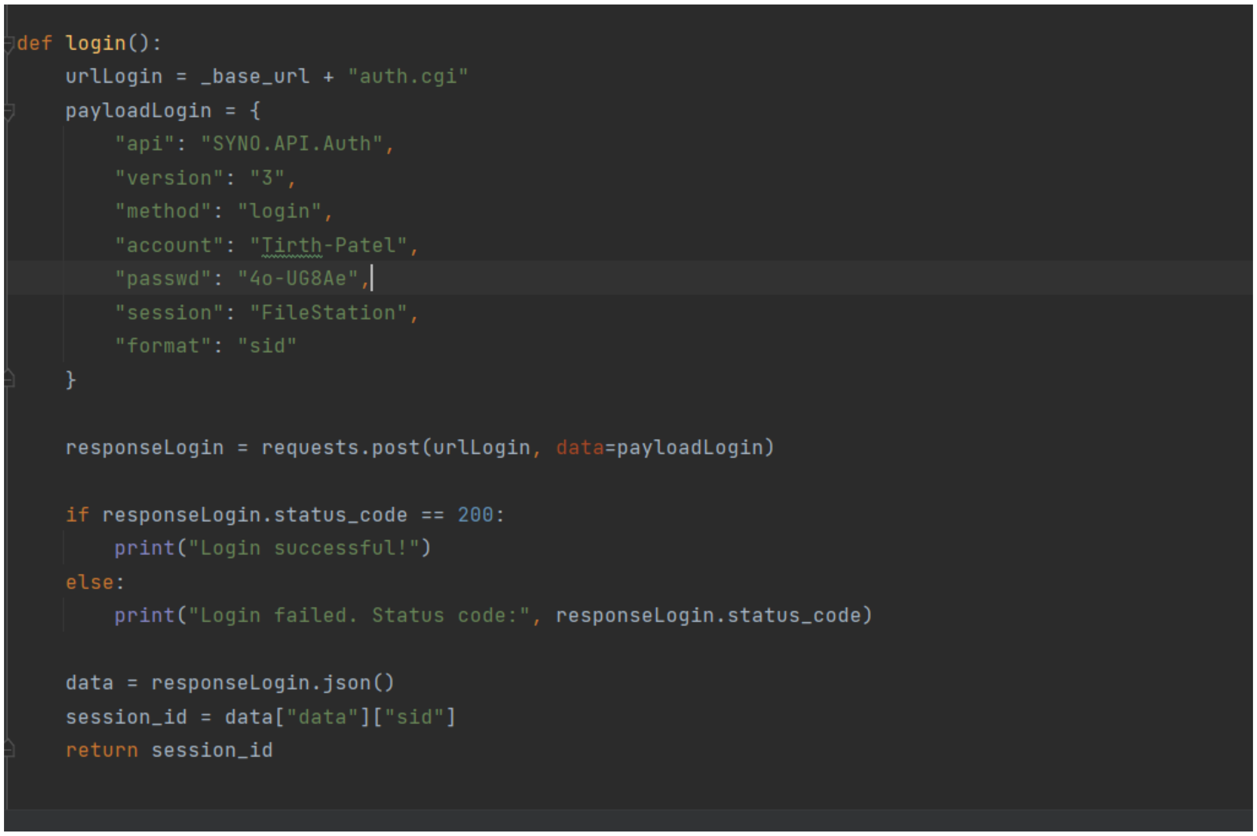
Expected Outcome:

- The edge server should be able to establish a connection with the cloud platform.
- The connection should be stable and reliable.
- The cloud platform should receive GPS coordinates and directions of the vehicles sent from the edge server.
- The data sent from the edge server should be received by the cloud platform in a timely manner.



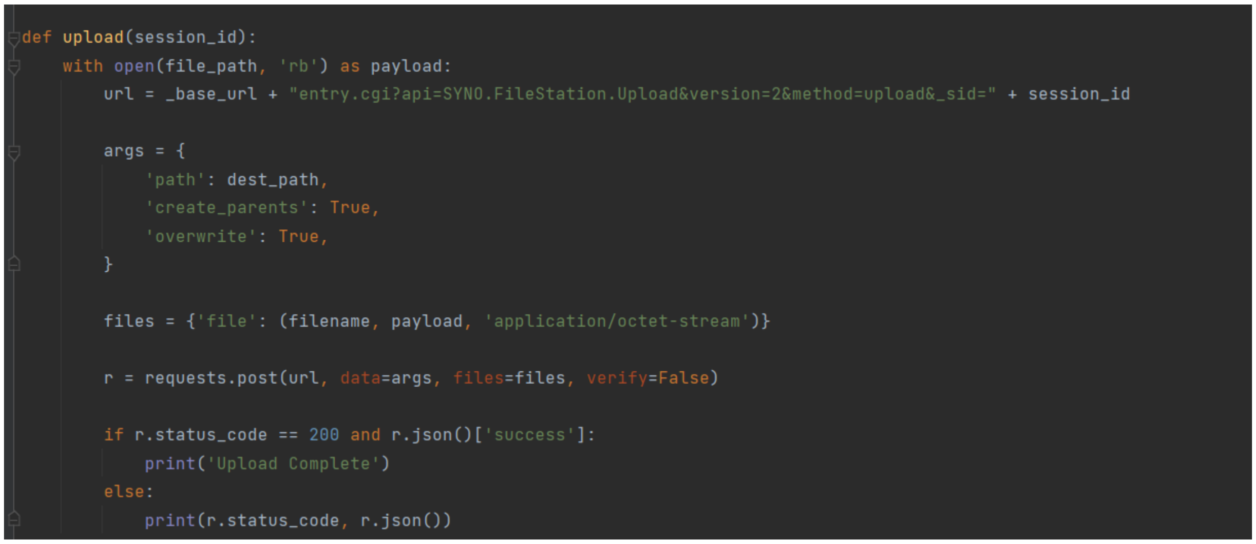
```python
import sys

import requests
import os

file_path = r"C:/Users/tirth/OneDrive/Desktop/Capstone/Upload/test.txt"
file_path_download = r"C:/Users/tirth/OneDrive/Desktop/Capstone/Download"
filename = os.path.basename(file_path)
dest_path = "/UsrShrd/Tirth-Patel/test1"
_base_url = "http://192.168.0.198:5000/webapi/"
down_path = dest_path + "/" +filename
```

Figure 1.13: Credentials for our SESL Cloud



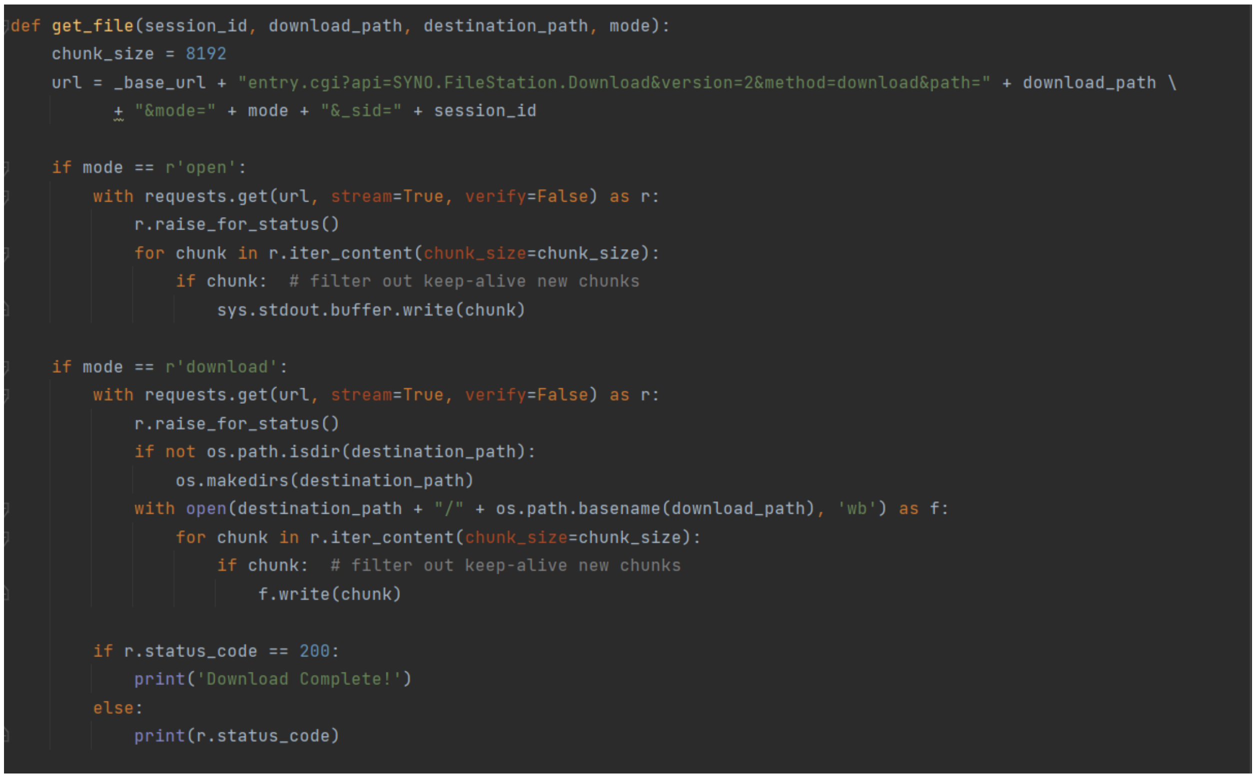
```python
def login():
    urlLogin = _base_url + "auth.cgi"
    payloadLogin = {
        "api": "SYNO.API.Auth",
        "version": "3",
        "method": "login",
        "account": "Tirth-Patel",
        "passwd": "4o-UG8Ae",
        "session": "FileStation",
        "format": "sid"
    }

    responseLogin = requests.post(urlLogin, data=payloadLogin)

    if responseLogin.status_code == 200:
        print("Login successful!")
    else:
        print("Login failed. Status code:", responseLogin.status_code)

    data = responseLogin.json()
    session_id = data["data"]["sid"]
    return session_id
```

Figure 1.14: Login Process



```python
def upload(session_id):
    with open(file_path, 'rb') as payload:
        url = _base_url + "entry.cgi?api=SYNO.FileStation.Upload&version=2&method=upload&_sid=" + session_id

        args = {
            'path': dest_path,
            'create_parents': True,
            'overwrite': True,
        }

        files = {'file': (filename, payload, 'application/octet-stream')}

        r = requests.post(url, data=args, files=files, verify=False)

        if r.status_code == 200 and r.json()['success']:
            print('Upload Complete')
        else:
            print(r.status_code, r.json())
```

Figure 1.15: Uploading information



```python
def get_file(session_id, download_path, destination_path, mode):
    chunk_size = 8192
    url = _base_url + "entry.cgi?api=SYNO.FileStation.Download&version=2&method=download&path=" + download_path \
          + "&mode=" + mode + "&_sid=" + session_id

    if mode == r'open':
        with requests.get(url, stream=True, verify=False) as r:
            r.raise_for_status()
            for chunk in r.iter_content(chunk_size=chunk_size):
                if chunk:  # filter out keep-alive new chunks
                    sys.stdout.buffer.write(chunk)

    if mode == r'download':
        with requests.get(url, stream=True, verify=False) as r:
            r.raise_for_status()
            if not os.path.isdir(destination_path):
                os.makedirs(destination_path)
            with open(destination_path + "/" + os.path.basename(download_path), 'wb') as f:
                for chunk in r.iter_content(chunk_size=chunk_size):
                    if chunk:  # filter out keep-alive new chunks
                        f.write(chunk)

        if r.status_code == 200:
            print('Download Complete!')
        else:
            print(r.status_code)
```

Figure 1.16: Downloading and reading a file from the server

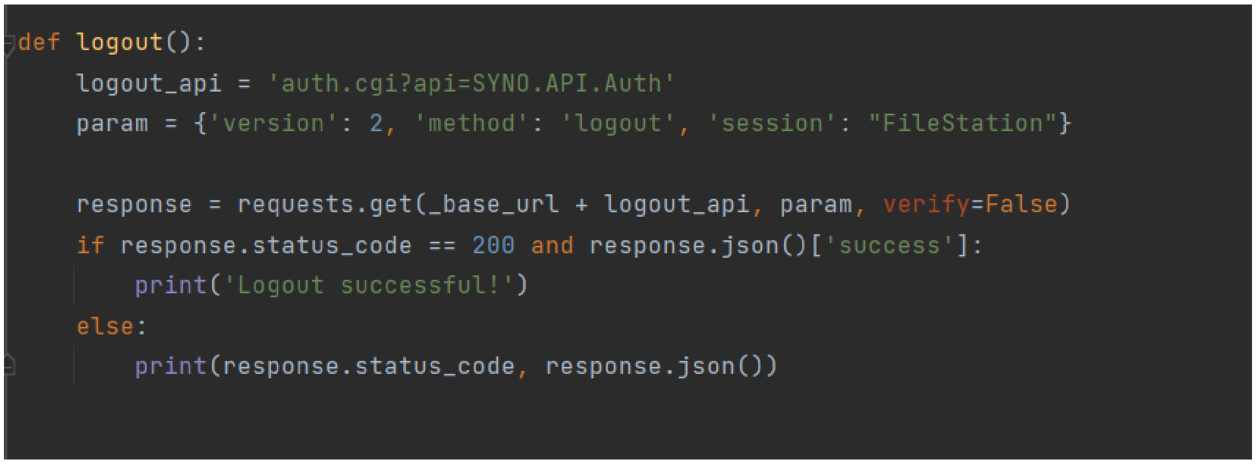

```python
def logout():
    logout_api = 'auth.cgi?api=SYNO.API.Auth'
    param = {'version': 2, 'method': 'logout', 'session': "FileStation"}

    response = requests.get(_base_url + logout_api, param, verify=False)
    if response.status_code == 200 and response.json()['success']:
        print('Logout successful!')
    else:
        print(response.status_code, response.json())
```

Figure 1.17: Logging out of the Server

1.2.6 Test Case 6

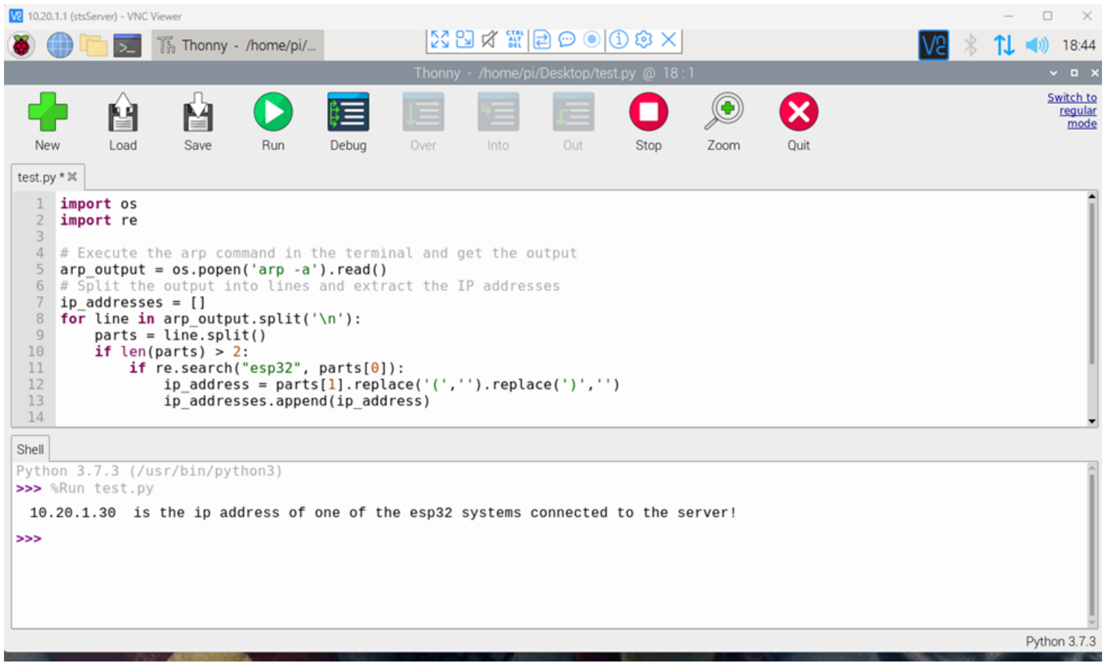
In this test, we would like to verify the information from the vehicles with the edge server and get the correct IP addresses and analyze the data to perform our scenarios. The test is deemed successful if the edge server is successfully able to receive the IP addresses of vehicles arriving at the intersection.

Test Steps:

- Send vehicles (e.g., a car) equipped with GPS and Wifi devices through a simulated traffic intersection vehicles.
- Send the GPS coordinates of the vehicles to the edge server.
- Verify the edge server receives the coordinates.

Expected Outcome:

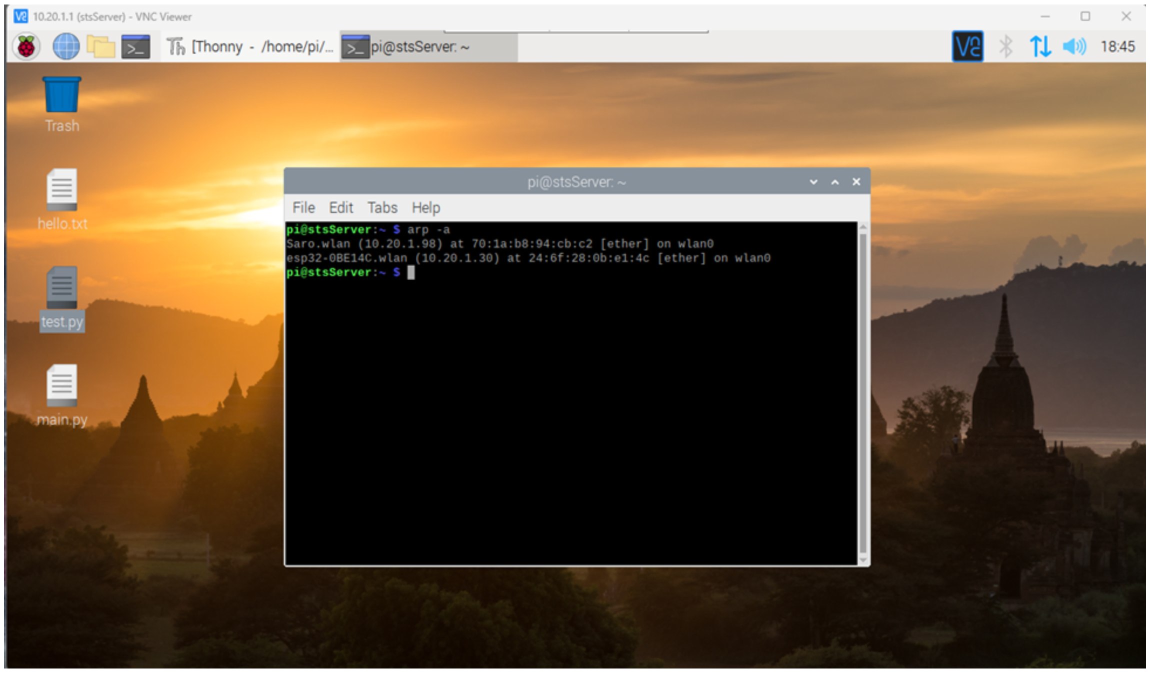
- The edge server should be able to receive GPS coordinates of the incoming vehicles
- The edge server should be able to get the IP addresses of the incoming vehicles
- The edge server should parse the incoming cars coordinates into useful information



```python
import os
import re

# Execute the arp command in the terminal and get the output
arp_output = os.popen('arp -a').read()
# Split the output into lines and extract the IP addresses
ip_addresses = []
for line in arp_output.split('\n'):
    parts = line.split()
    if len(parts) > 2:
        if re.search("esp32", parts[0]):
            ip_address = parts[1].replace('(','').replace(')','')
            ip_addresses.append(ip_address)
```

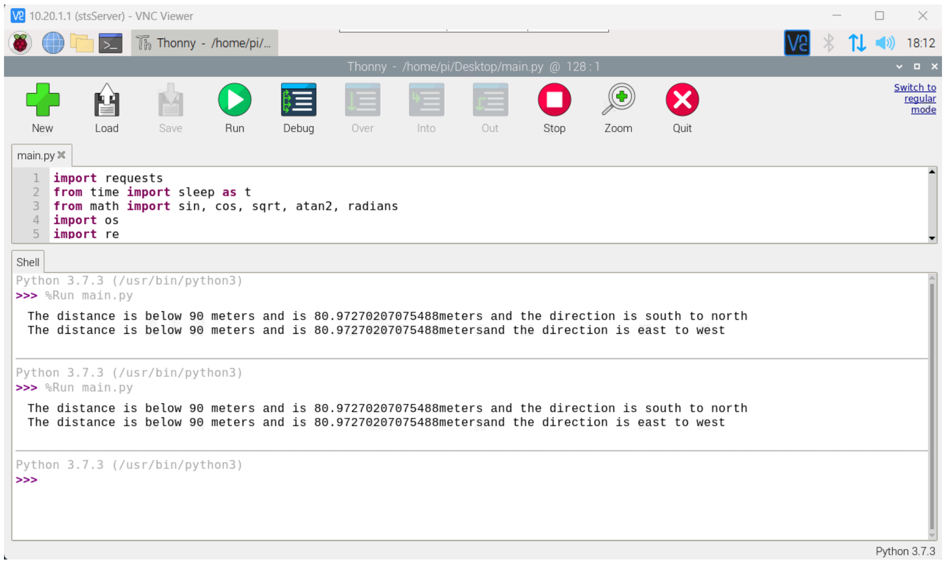
```
Python 3.7.3 (/usr/bin/python3)
>>> %Run test.py
 10.20.1.30  is the ip address of one of the esp32 systems connected to the server!
>>>
```

Figure 1.18 Testing the edge server for getting the IP addresses of the esp32 modules



```
pi@stsServer:~ $ arp -a
Saro.wlan (10.20.1.98) at 70:1a:b8:94:cb:c2 [ether] on wlan0
esp32-0BE14C.wlan (10.20.1.30) at 24:6f:28:0b:e1:4c [ether] on wlan0
pi@stsServer:~ $
```

Figure 1.19 Shows the result of the arp -a command and that Figure 1.18 is parsing the information correctly



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

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run main.py

  The distance is below 90 meters and is 80.97270207075488meters and the direction is south to north
  The distance is below 90 meters and is 80.97270207075488metersand the direction is east to west

Python 3.7.3 (/usr/bin/python3)
>>> %Run main.py

  The distance is below 90 meters and is 80.97270207075488meters and the direction is south to north
  The distance is below 90 meters and is 80.97270207075488metersand the direction is east to west

Python 3.7.3 (/usr/bin/python3)
>>>
```

Figure 1.20 The edge server is getting the information of two ESP32 modules and process the distance away from it and what are the ESP32 modules direction in relation to the server

2. Contribution matrix

Table 2.1: Summary of work % contributed by each team member for Report 2

Task	**Team Members**		
Report 2	Saro Karimi	Tirth Patel	Vatsal Patel
	33.33%	33.33%	33.33%