THE CITY COLLEGE OF NEW YORK

Department of Electrical Engineering

DYNAMIC TRAFFIC CONTROL WITH HIGH DEFINITION IP CAMERA

Senior Design Project by:

Shadman Kazi
Michal Kropiewnicki
Saad Arshad
Haresh Mohabir
Gerlin Hernandez

Abstract:

Dynamic Traffic Light Control with High-Definition IP Camera is a novel Intelligent Traffic System (ITS) technology developed to optimize cycle lengths, green/red light times, or phasing sequences for traffic signals based on the changing traffic volumes collected from advanced detectors. This new ITS is considered to improve traffic safety and reduce congestion, it also has the potential to reduce crashes. This paper is concerned with the development and implementation of Dynamic traffic light control with an HD IP camera.

Our project "Dynamic Traffic Control With High Definition IP Camera" will be constructed on top of an existing "smart street light" infrastructure. In practical terms, the microcontroller will be placed in the same cabinet as the smart streetlight where the decision for the time cycles will be made from the received camera images. The HD IP camera is mounted on top of the traffic light pole. Ideally, the camera must take frequent pictures throughout the entire time cycle. This will enable the camera to send updated data to the cloud in real-time to provide a better estimation of traffic conditions. Through digital image processing, the microcontroller will count the number of cars and dictate the green or red light signal time. The next section will explain how the ideas of the previous projects were used to build our infrastructure.

Key Terms: HD IP camera, Raspberry Pi, Wifi, traffic signal, DOA (Direction of Arrival), DIP (Digital Image Processing), microcontroller, IDE (Integrated Development Environment), & IoT (Internet of Things).

Introduction:

"New York City is the third most traffic-congested city in the world" (Walker, 2018). With the growing population and number of vehicles in cities, traffic congestion is becoming a

major problem. “Per the 2017 analysis, New York drivers averaged 91 peak hours stuck in traffic last year” (Walker, 2018). Traffic jams not only cause additional delays and stress for drivers, but they also increase fuel consumption, transportation costs, and carbon dioxide (CO2) emissions in the air. CO2 emissions ultimately cause air pollution. Many factors can cause traffic congestion, e.g. environment, mechanical, human, and infrastructure. Infrastructure can be further divided into different categories. However, in this project, we will address traffic congestion due to large red light delays. One of the most important factors influencing traffic flow is traffic lights. To overcome this issue, we are proposing a Dynamic Traffic Light Control with a High-Definition IP Camera system. Traffic flow will be monitored through DIP. For instance, an intersection has two directions, north to south and east to west, and each direction has only one lane, so traffic can only go forward. We will neglect right or left turns. A traffic light is used to provide three signals: yellow to slow down, red to stop, and green to go in the lane. To detect traffic flow, HD IP cameras will be placed in a total of two lanes. The camera will take pictures and send them to the base station, which is located locally. After collecting the real-time traffic information, the local controller at the base station will count the number of cars through DIP and compute the green light sequence and duration accordingly.

Need for Design:

To support our increasingly urbanized society, we must engage with smart technology and improve operations across the city. Smart dynamic traffic control with HD IP cameras design is needed in the market because we must cater to rapidly growing demands for public services and infrastructure. According to the Smart Cities Community, IEEE, “the total population living in cities is forecast to increase by 75% by 2050” (IEEE, 2020). This creates the need for a larger, sustainable traffic light control infrastructure that can improve traffic flow as

the number of vehicles on the road increases. There are several smart city pathways such as smart energy, smart mobility, smart living, and overall transformation to digital cities.

The main goals of our design are to improve traffic flow and reduce total travel time and wait time at the red light. The HD IP camera will acknowledge the instantaneous traffic conditions and relay that data to the traffic light to control the time it will be red, yellow or green on the four sides. We will be using the existing infrastructure of smart LED streetlights to mount the camera which establishes a reliable communication network that can be used for other smart devices since it is connected to the cloud. According to the EPA, "producing and using electricity more efficiently reduces both the amount of fuel needed to generate electricity and the number of greenhouse gases and other air pollution emitted as a result" (EPA, 2019). The feature of the streetlight dimming when no motion is detected saves energy which ultimately decreases our contribution to global warming or climate change. LEDs in our streetlight further contribute to energy efficiency, along with other advantages such as long life, reliability, and high brightness intensity (Levison, 2020).

The upgrade of traffic light control infrastructure is needed in the market because of public safety and reliability. For quality control and assurance purposes, we must ensure that our infrastructure does not fail because it compromises the safety of civilians. For instance, there must be sufficient time allotted to civilians crossing the street and for vehicles passing the intersections. According to Kingson, LED streetlights have already been implemented in cities such as New York, Chicago, Atlanta, Los Angeles, Philadelphia, and Cleveland (Kingson, 2021). Therefore, it is sensible to implement our design in these cities because the infrastructure is already there and it is the next step forward in our urban development. With the advancement in our communications and technology, our traffic light control network supports a more digital city. "Smart street lighting has grown at a robust CAGR [compound

annual growth rate] of 52% since 2012 and will maintain steady growth through the 2020s" (Northeast Group, 2020). The growth of smart street lighting will simultaneously allow our design to be implemented and invoke further development in our lighting and traffic control network.

The main competitor to our design is using a smart antenna system in a telecommunication context as opposed to using an HD IP camera. The smart antenna system estimates the Direction of Arrival (DOA) of incoming signals, including the interfering signals and the multipath signals, using the DOA algorithms (R. Yaqub, A. Joyo, K. Heidary, and N. Madamopoulos, 2018). In "Green-initiative for Realizing Adaptive Traffic-signals by Exploiting Smart-antenna Technology (GREATEST)," it states that smart antennas are complex, expensive, and are large. Furthermore, "another limitation is the aggregation of the traffic statistics from diverse mobile service providers, as different mobile operators have their Base Stations" (Yaqub, A. Joyo, K. Heidary, and N. Madamopoulos, 2018). Our smart dynamic traffic control with HD IP cameras design will use the same basics from the smart LED street light, therefore, it would not be difficult to implement into the pre-existing design. Also, since the cameras will be mounted on the streetlights, we can estimate the arrival time of vehicles at the intersection based on motion and speed.

The Problem we are Solving:

Street lights play an essential part in keeping drivers safe while they drive during the night. Streets and roads that do not have street lighting account for nearly one-third of all fatal crashes at night (Bhagavathula, 2021). The current solution to this problem would be to install more sodium vapor lamps to keep cities lit during the night. However, this solution is problematic. These street lights are not capable of detecting if light service is needed and are

not designed to be dimmed. The light pollution that these lamps produce affects the local ecosystems. For example, crops do not mature in the natural time frame and animals such as sea turtles confuse the artificial light with the light of the moon which can decrease their probability of surviving to adulthood (Bhagavathula, 2021). LED technology can help resolve these issues. Smart city lights which are adapted with LED technology can be dimmed or controlled intelligently (Bhagavathula, 2021). The smart LED street lights can be used only when needed and can be controlled to be dimmed in areas where light pollution is harming the local ecosystems. Our project will build on this already existing infrastructure.

According to the environmental defense fund, an idling car uses between 1/5 to 7/10 of a gallon of fuel an hour (EDF, 2009). This is problematic as the emission of greenhouse gasses is contributing to the warming of the climate. Cars that are directed to stop at an intersection by traffic lights are forced to idle until they have the right of way to pass. Traditional fixed cycle timing traffic lights contribute to this problem. They are responsible for keeping vehicles from passing no matter if no other cars are present at the intersection. Once the timed cycle comes to a close, the vehicle will have a specific amount of time to pass until the light changes once more. A smarter application is required to help reduce the idling time of vehicles on intersections.

According to the department of transportation of the state of New York, the typical cycle lengths of a traffic light vary between 45 and 120 seconds (DOT, 2021). Even with traffic lights that are programmed based on traffic volume and traffic history, cars still produce unnecessary greenhouse gases by idling. To create an environment that changes the lights based on real-time demand, an HD IP camera can be implemented. The camera will detect vehicles that want to pass an intersection. The camera's image will be processed locally to determine how long it will take for the light to change so that cars can pass. This new approach will produce the most efficient time for the vehicles at the intersection to wait to reduce the idle time and reduce the

emission of greenhouse gasses. Implementing this new approach with the already existing infrastructure of smart street lights will produce an environmentally and user-friendly environment.

Public safety is the primary concern in our project. With the support of intelligent video analytics and recognition technologies, HD IP cameras can quickly transmit real-time and relevant data back to the control center to immediately respond to any events. This ITS service will support automatic fault detection and alarm mechanisms. Once a fault is detected in the system, an alarm message will automatically be sent to the relevant maintenance personnel. Based on electrical parameter values of traffic lights, an ITS service will be able to analyze these data, provide predictive information, and notify whether traffic lights are approaching the end of life. Then replacements can be scheduled proactively. Our system will be able to withstand severe weather like a snowstorm and hot climate. HD IP cameras have better night vision, which will ensure the safety of pedestrians and drivers.

Company Requirements:

While the main goal of the smart system is to be more user-friendly and eco-friendly, we will have to meet the requirements to successfully maintain and implement the traffic. Although multiple metropolitan cities have 5G towers for cellular use, we may require either a modification of the tower to be versatile to enable multiple uses or a designated 5G base station used to meet the data rate requirements. Also, it is best for all components to be connected to a single network, so that the data/conditions can be consistent depending on real-time location services.

For the devices to be able to communicate with each other, Bluetooth mesh networking will be the most convenient way to do so. Bluetooth mesh networking allows a large number of devices to be connected. This will result in a large-scale device network, which is necessary for our traffic control system. To successfully utilize the concept of a smart device, an algorithm will be required for regulating the time traffic lights are red, green, etc., and the duration of these lights.

A microcontroller, such as Raspberry Pi in our project, should also be used to execute the algorithms mentioned before. Essentially, the microcontroller is the central part of our smart traffic control system, since many sources of data will come from various components and the microcontroller will provide feedback to the destination component. For instance, the microcontroller will be able to process images taken by the camera, which will then enable a certain algorithm based on the data provided by the processed image.

According to the New Jersey Department of Transportation, the casing HD IP camera that will be integrated into the smart control system will have to comply with the following standards:

a. NEMA (National Electrical Manufacturers Association) Type 4X

(Enclosures constructed for either indoor or outdoor use to provide a degree of protection to personnel against incidental contact with the enclosed equipment; to provide a degree of protection against falling dirt, rain, sleet, snow, windblown dust, splashing water, hose-directed water, and corrosion; and that will be undamaged by the external formation of ice on the enclosure)

b. As a minimum, IP 66 Environmental Rating:

In this case, the camera will be protected from dust ingress, as well as high-pressure water jets regardless of direction, allowing the camera to be at least water-resistant (not waterproof because limited ingress is permitted) and weatherproof.

c. ONVIF Profile S (Profile S is designed for IP-based video systems. A Profile S device (e.g., an IP network camera or video encoder) can send video data over an IP network to a Profile S client. A Profile S client (e.g., a video management software) can configure, request, and control video streaming over an IP network from a Profile S device. Profile S also covers ONVIF specifications for PTZ control, audio in, multicasting, and relay outputs for conformant devices and clients that support such features)

d. UL (Underwriter Laboratories) Listed for outdoor use

e. NEC (National Exhibition Centre)

Despite that it's only the standard for one department, it provides a rough outline of what specifications we should set for the camera to comply with at least the majority of the government standards.

Because we are trying to integrate an HD IP camera into a traffic system, we would need a database to rely on that will provide an accurate location using the Cartesian coordinate system. Google Maps, Latlong.net, or even Waze can provide coordinates of the location of a certain camera if there's an ongoing traffic issue. From there, we can use the 5G base towers that were mentioned earlier to allow the camera to utilize the database, which can then analyze the area and select the appropriate algorithm depending on the traffic condition.

Previous Projects:

The Dynamic Traffic Light System with HD IP camera is built based on the foundations of the smart street light systems proposed in previous semesters. Like our model, the smart street traffic system requires a microcontroller that will execute code and make decisions depending on where the parking spot is empty. As the light is turned on for the empty parking spot, the microcontroller will also send data to the cloud via MQTT Protocol, which will then be published on a web application where end users can see where the empty parking spots are at. However, the key difference is that the traffic system will require an algorithm that consists of a traffic cycle, which will dictate the traffic flow of the intersection. The street light system, on the contrary, simply detects empty parking spots and notifies the web application through the cloud.

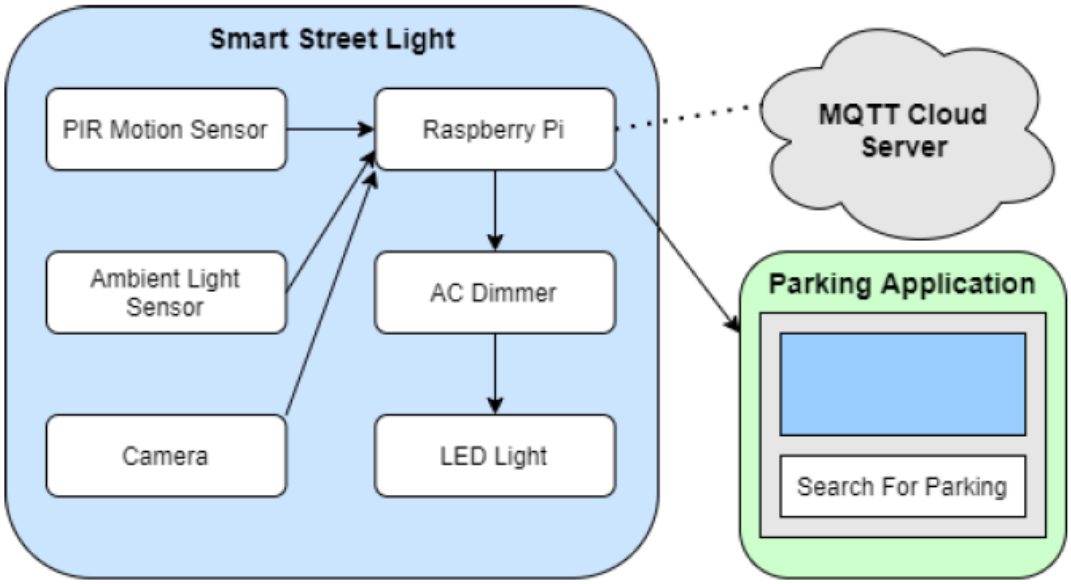

Figure 1: Smart Street Light Flowchart

We were able to implement our dynamic traffic system using a similar method used above for the smart street lights. Since the street lights are based on the detection of the empty parking spot, a motion and light sensor is used to determine if a car is detected and is occupying the parking spot. If any motion is detected, the Raspberry Pi will send a command to the dimmer to allow the LED light to slowly turn on. From there, the light sensor will detect if there is a presence of a car

leaving or entering the parking space. Finally, the microcontroller will send data to the cloud, notifying that a new empty parking spot is available. The cloud will then send the new parking spot location to the web application, which will notify end users that a new parking spot has been detected.

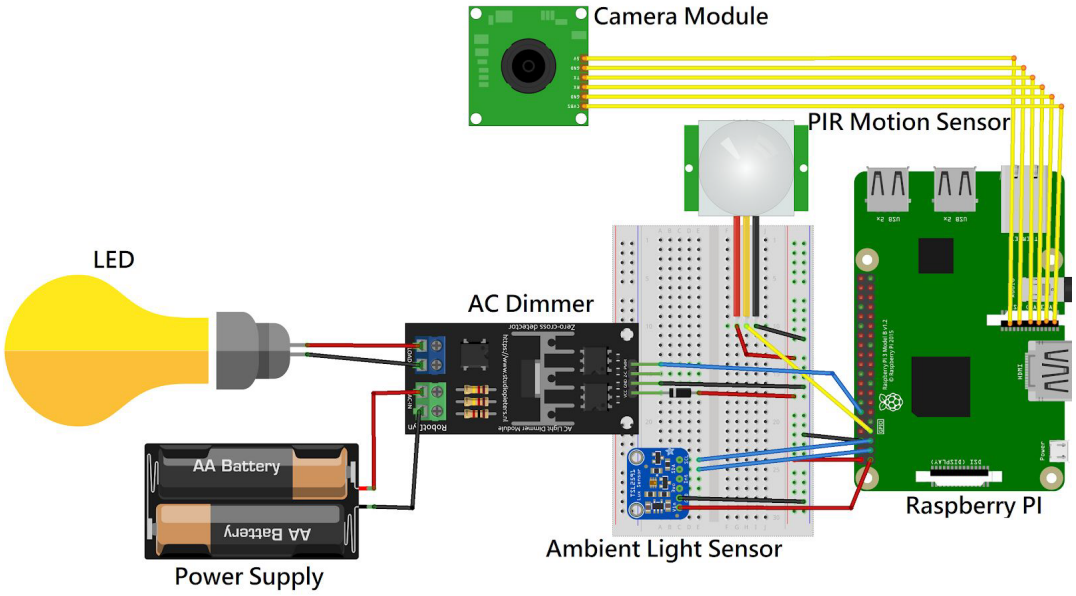

Figure 2: Smart Street Light Schematic

Based on the above circuit diagram, the Raspberry Pi will be the central computer to communicate with all components. The camera module will be placed directly on the Raspberry Pi, as well as the PIR motion and ambient light sensors, but both sensors will be grounded. The AC dimmer will be connected to the power supply since it will dictate the amount of power sent to the LED light. Our traffic light circuit diagram will be similar, but we will be replacing an AC dimmer and LED lights with a traffic light module, and we will neglect the use of sensors since the camera module will act as one using digital signal processing.

The group who worked on the street light system placed their microcontroller on a flat surface where the camera can detect cars. In their particular case, they used the floor and often tested using different lighting conditions. In our case, we will be using a flat surface. However, since the

lighting is just constant, we decided to set up a table with clear lighting, so that we can get an accurate amount of cars, so we can get more accurate calculations.

Three Main Components for IoT:

Since our traffic system will be an Internet of Things device, three components will be required for the IoT device to be operational, which are sensors, measurement and decision, and the connectivity from the server to the cloud. The sensors in this case will be the Raspberry Pi Camera Module, which will take pictures of the intersection at a certain frequency, and will send the picture to the microcontroller, which will make the measurements and decisions via digital signal processing. Once the traffic cycle is adjusted, the Raspberry Pi that acts as the traffic system, the traffic controller, will send data to the other Raspberry Pi, which will play the role of a server, to publish the number of cars at the intersection. Similarly, if we wanted to incorporate the microcontroller on an existing system, we can assign each pin of the Raspberry Pi to the corresponding fluorescent lights, and have the camera be connected to the microcontroller with a wire to successfully implement digital signal processing.

Proposed Model:

Unlike other traffic control systems, we plan on using the High Definition IP camera as our source of data. The IP camera will frequently take pictures of the intersection. Next, we will use image processing, as well as digital signal processing, to render and analyze the image. From there, we will utilize an algorithm that will detect objects, which will count these objects from there.

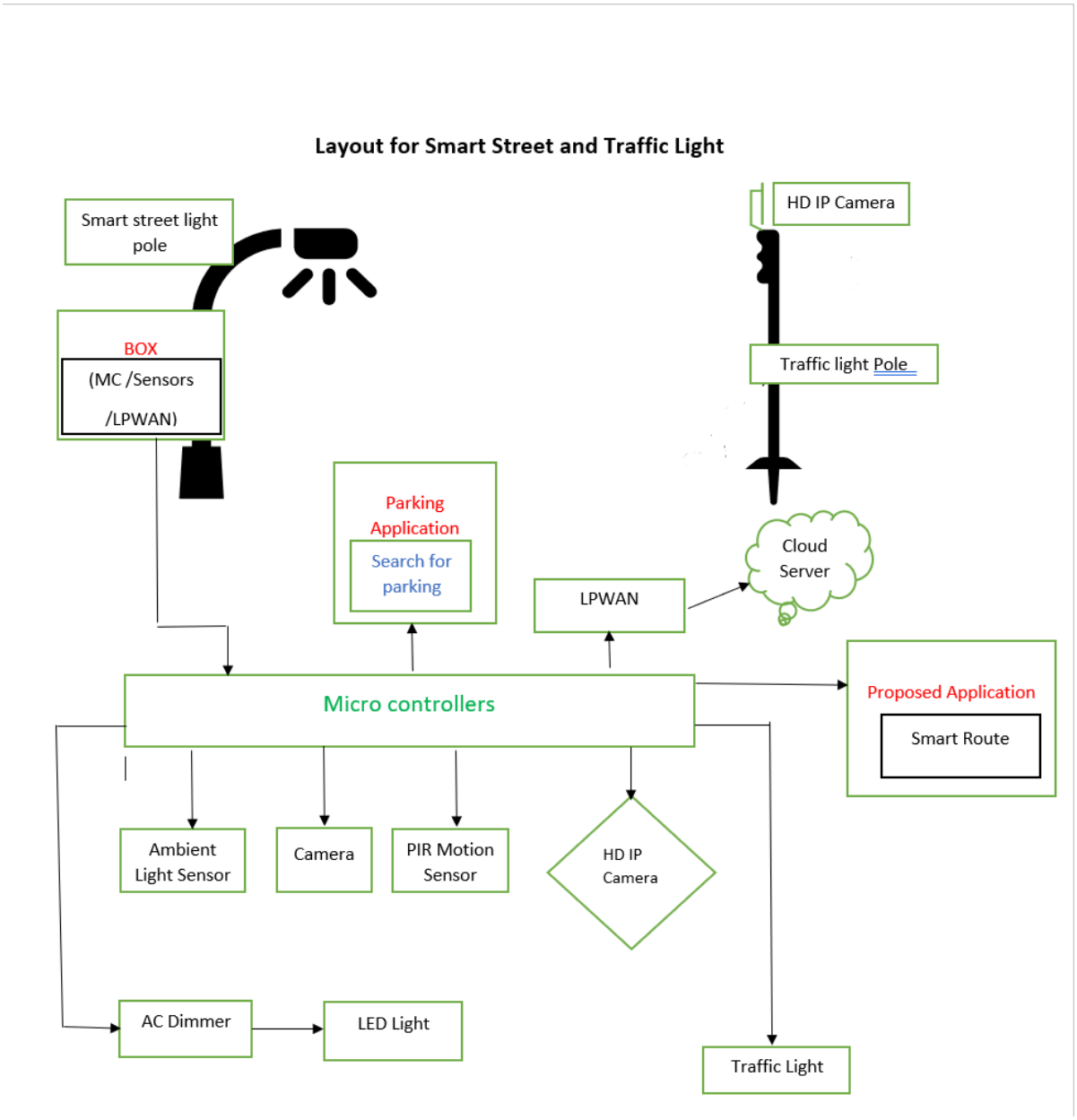

Figure 3: Proposed Model Concept Diagram

The flowchart further specifies the conditions of each step of the model concept diagram.

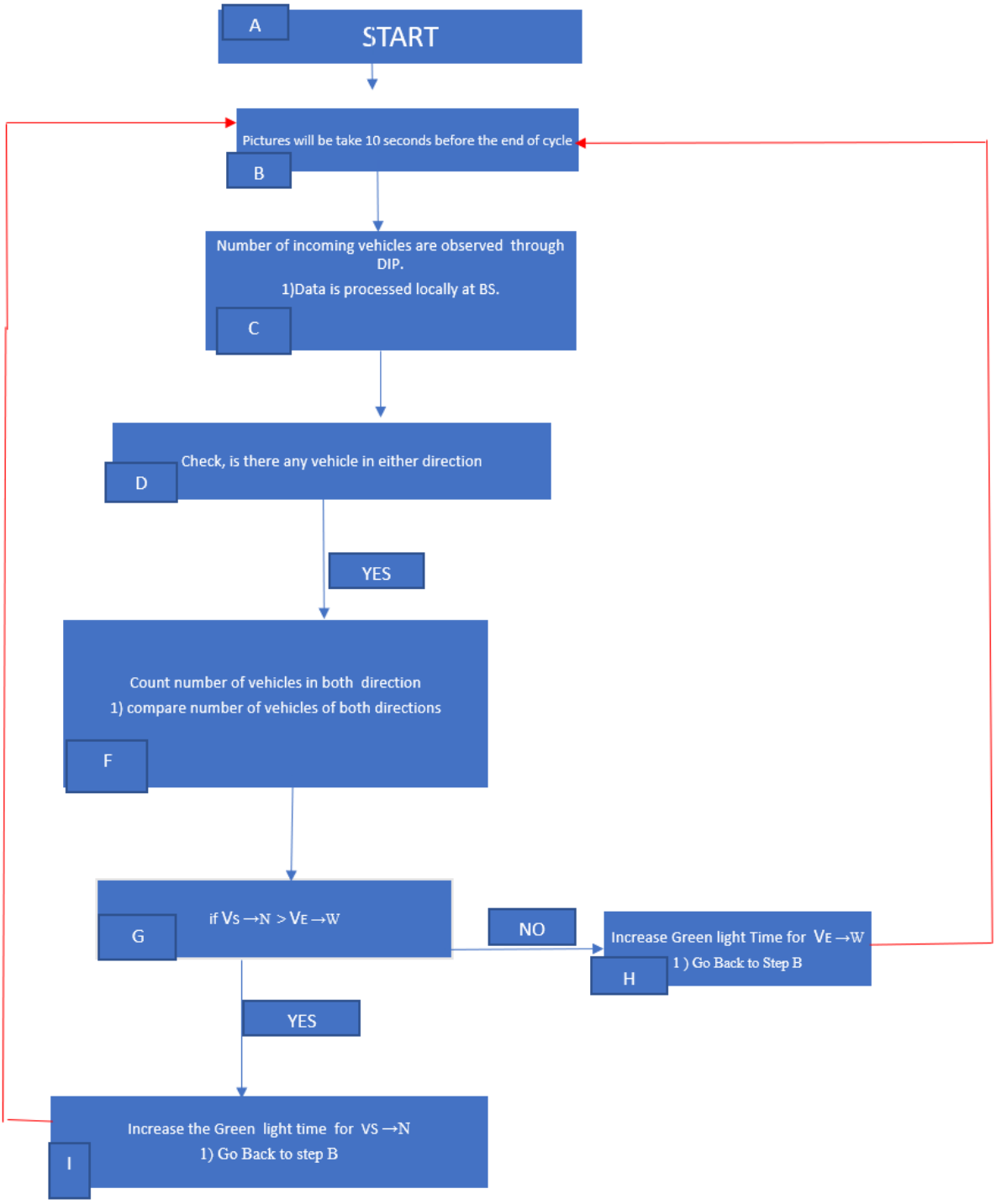

Figure 4: Flowchart for Dynamic Traffic Control with HD IP Camera

Required Materials:

1. Microcontroller - Raspberry Pi

The microcontroller is the most important component of our experiment because it will be performing numerous tasks. From digital signal processing to image processing, the microcontroller will be our central unit for the traffic control system. A microcontroller is wifi-compatible, so we can configure it to collect real-life traffic statistics and allow all components to be connected to one network. For our design, we will be using the Raspberry Pi 4 kit. While the Arduino microcontroller is cheaper, it requires an external computer to retrieve and simulate the code. The Raspberry Pi microcontroller, on the other hand, can operate as a computer and can process the code. We will require two Raspberry Pi microcontrollers for both coordinate planes (x and y-plane). Note that the other components used, except the toy cars, will ensure that it is compatible with this particular Raspberry Pi model.

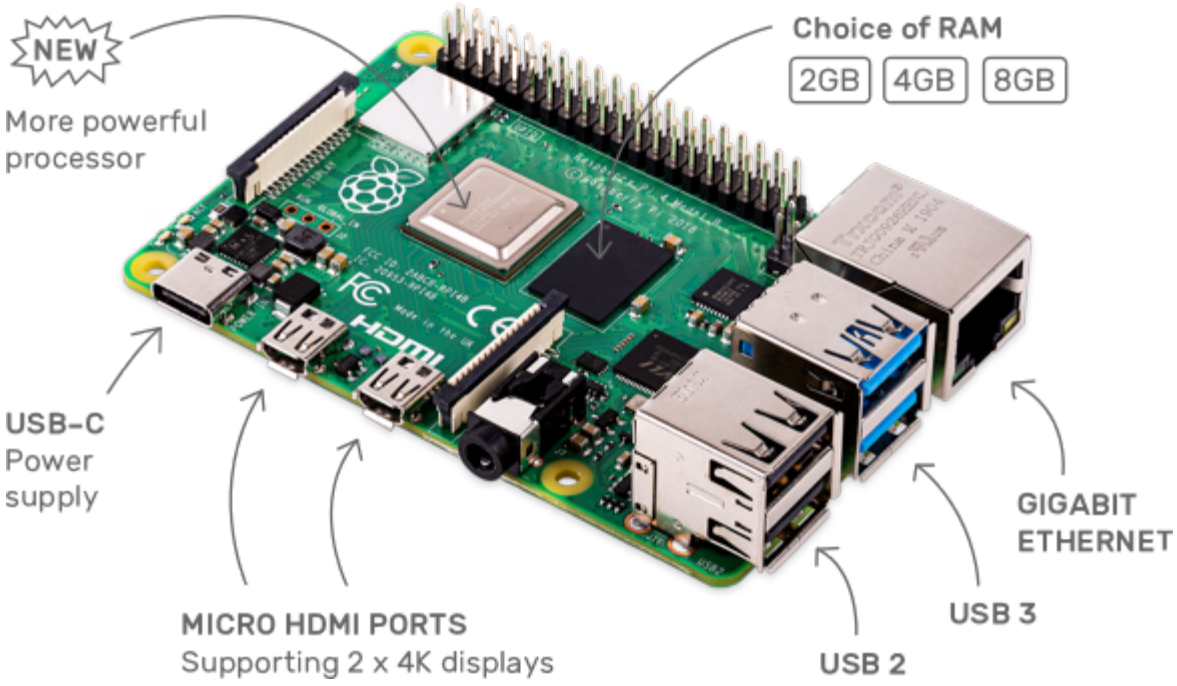

Figure 5: Raspberry Pi Module

2. LED Traffic Lights

For any traffic system, traffic lights are required to dictate traffic flow for the cars. The traffic light is responsible for signaling cars to either go, slow, or stop (for green, yellow, and red lights, respectively). Instead of the fluorescent lights used for traffic lights today, we believe that the LED lights will require significantly less power. More importantly, the traffic lights used are compatible with the Raspberry Pi. We will only need a quantity of one set of five traffic lights.

Figure 6: LED Traffic Lights

3. Raspberry Pi Camera Module

The role of our High Definition IP Camera is to take pictures of the intersection so that the image can be processed and obtain data from the picture. Important information such as the number of cars in each direction of the intersection can be deducted from that data. The camera module we will be using provides an image resolution of 8 megapixels, as well as a video resolution of 1080p. This is the best camera module in terms of technical specifications that is compatible with our microcontroller. We will require one camera per coordinate plane, resulting in a total of 2 camera modules.

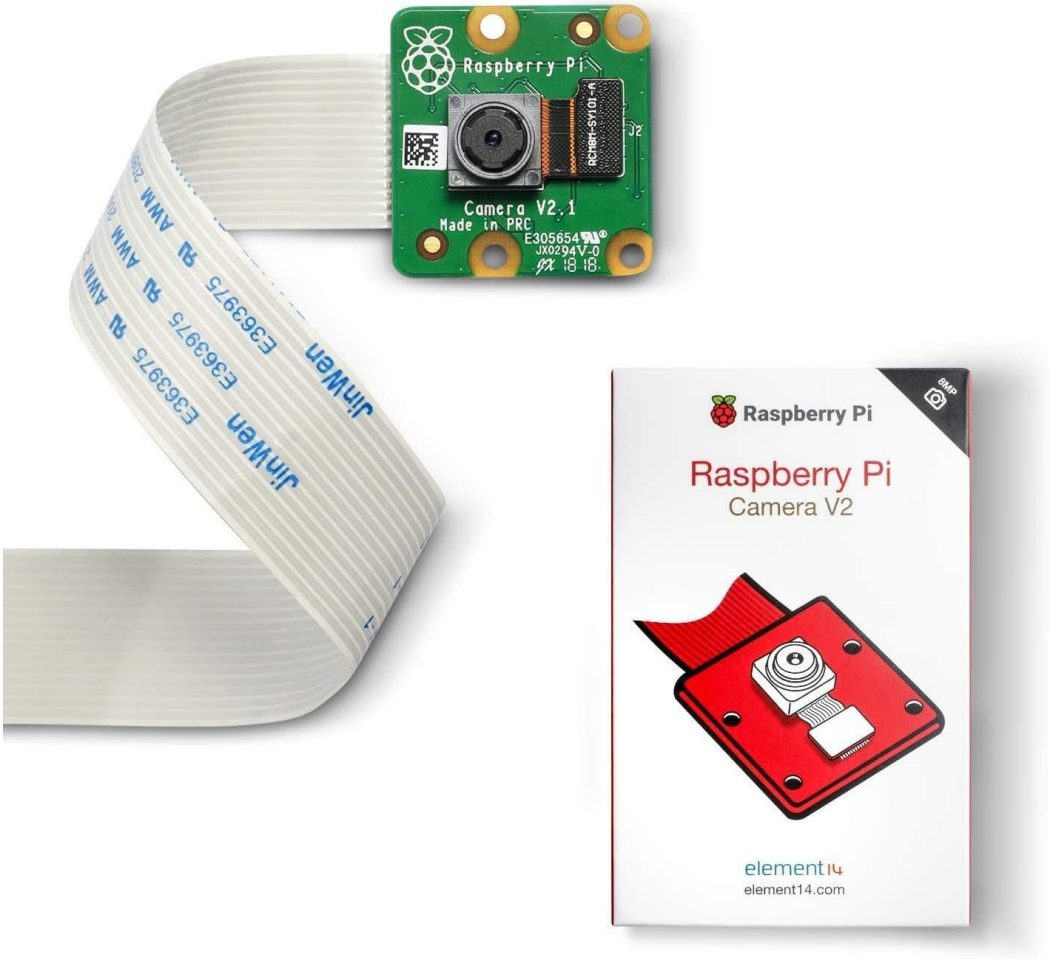

Figure 7: Raspberry Pi Camera Module

4. Toy Car (6-Pack)

The toy cars will be used as test subjects for our traffic control system. The main object of a traffic system is to control car traffic movement. Therefore, we need to simulate a real-life traffic control system, so we will be controlling the traffic of these toy cars.

Figure 8: Toy Cars

5. Raspberry Pi Power Supply.

The microcontroller kit previously mentioned will include a power supply to turn on not just the microcontroller but also its components.

Figure 9: Raspberry Pi Power Supply

Assembly:

Our traffic control system needs to be set up for our components to be able to communicate with each other in the following manner:

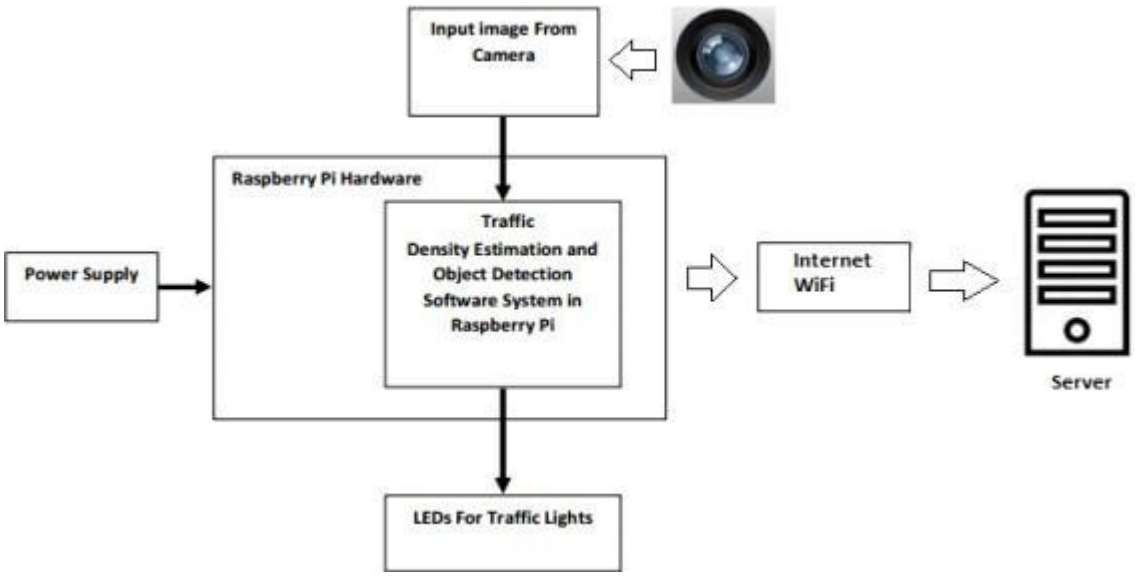

Figure 10: System Architecture for Traffic Control System Applications

The camera will first take a picture of the intersection and send it to the Raspberry Pi to detect the number of cars at an intersection. Depending on the number of cars, the Raspberry Pi will send a signal to the appropriate GPIO pins of the traffic light module, whether it is green or red. The Raspberry Pi will then send the data for the number of cars at an intersection to a server Raspberry Pi in our infrastructure. We can confirm if the server works if it shows the number of cars as an output on the terminal.

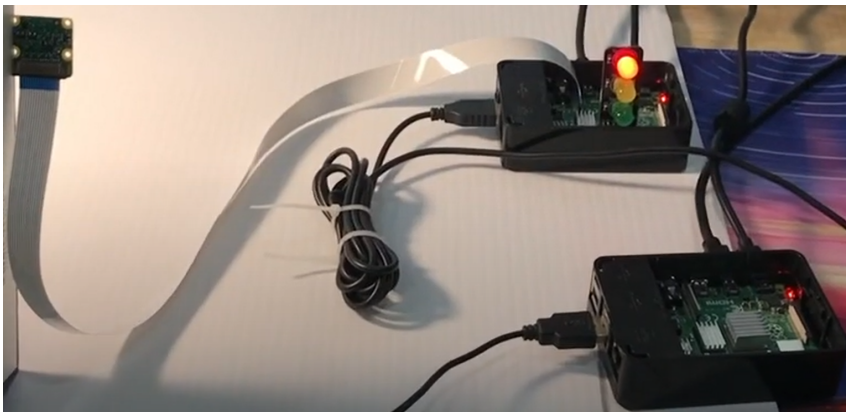

Figure 11: Traffic Controller and Server Raspberry Pi setup

Figure 12: All Required Components for Project (Power Supply not Shown)

IDE:

An IDE, Integrated Development Environment, is where we will implement and run our code. We will be using the built-in IDLE IDE to run and simulate our algorithms.

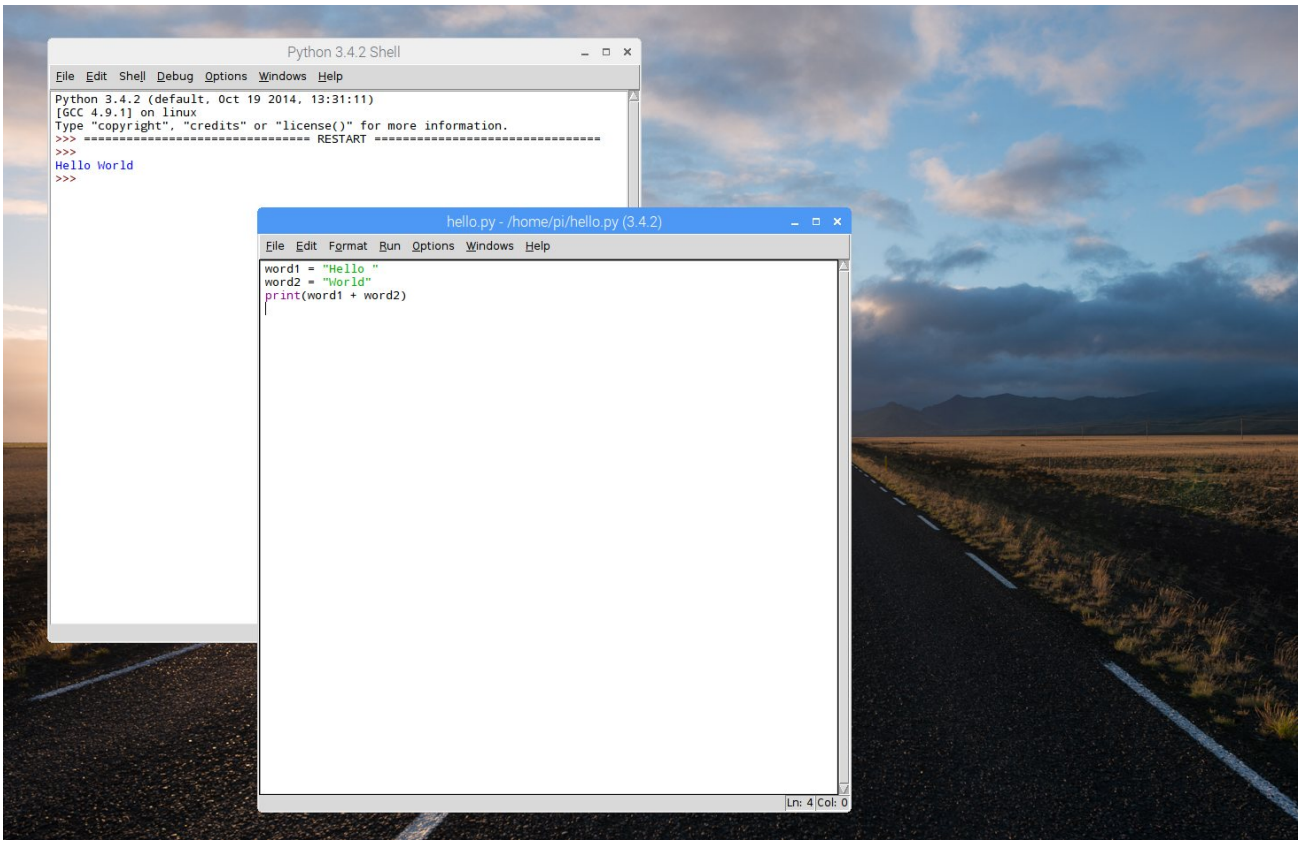

Figure 13: Raspberry Pi OS with IDLE python IDE Running

Specifications:

Microcontroller

Specifications

Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
Gigabit Ethernet
2 USB 3.0 ports; 2 USB 2.0 ports.
Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
2 × micro-HDMI ports (up to 4kp60 supported)
2-lane MIPI DSI display port
2-lane MIPI CSI camera port
4-pole stereo audio and composite video port
H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
OpenGL ES 3.1, Vulkan 1.0
Micro-SD card slot for loading operating system and data storage
5V DC via USB-C connector (minimum 3A*)
5V DC via GPIO header (minimum 3A*)
Power over Ethernet (PoE) enabled (requires separate PoE HAT)
Operating temperature: 0 – 50 degrees C ambient

* A good quality 2.5A power supply can be used if downstream USB peripherals consume less than 500mA in total.

Figure 14: Raspberry Pi 4 Specifications

The operating system that we will be working with on the Raspberry Pi is the Raspberry Pi OS, formerly known as Raspibian. We used the latest version of the operating system, which is Debian 11, known as Bullseye. The IDE application used to execute the python file is the IDLE application. Most libraries were included in IDLE since they were fundamental libraries such as time and enabling the GPIO pins. To install the OpenCV library, which we use for objection detection, we used the following command on the terminal: pip install OpenCV-python.

Camera Module

Product Name	**Raspberry Pi Camera Module**
Product Description	High Definition camera module compatible with all Raspberry Pi models. Provides high sensitivity, low crosstalk and low noise image capture in an ultra small and lightweight design. The camera module connects to the Raspberry Pi board via the CSI connector designed specifically for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the processor.
RS Part Numer	**913-2664**
Specifications	
Image Sensor	Sony IMX 219 PQ CMOS image sensor in a fixed-focus module.
Resolution	8-megapixel
Still picture resolution	3280 x 2464
Max image transfer rate	1080p: 30fps (encode and decode) 720p: 60fps
Connection to Raspberry Pi	15-pin ribbon cable, to the dedicated 15-pin MIPI Camera Serial Interface (CSI-2).
Image control functions	Automatic exposure control Automatic white balance Automatic band filter Automatic 50/60 Hz luminance detection Automatic black level calibration
Temp range	Operating: -20° to 60° Stable image: -20° to 60°
Lens size	1/4"
Dimensions	23.86 x 25 x 9mm
Weight	3g

Figure 15: Raspberry Pi Camera Module 2 Specs

Server

We will use LPWAN as a way to transport data to the cloud. LPWAN (Low Power Wide Area Networks) technology enhances the battery life of devices and connects devices that have previously been hard to reach. They are both available today, standardized, and built on the 4G network which means they are future-proof, have global network coverage, and are backed up by GSMA and telecom standards (Sternhoff, J. 2021).

The type of LPWAN we will be using is LTE-M. NB-IoT is another type of LPWAN, but LTE is more feasible. Operators typically deployed LTE-M first in countries that had LTE coverage

already such as the US. It's relatively easier to upgrade an existing LTE tower to support LTE-M than to add NB-IoT support. LTE-M has the advantage of having cellular roaming agreements already in place, while NB-IoT is far behind. Therefore, roaming for NB-IoT is comparatively limited (Bosson, M. 2021).

To use this MQTT library over an XBee 3 Cellular LTE-M/NB-IoT, you need a basic proxy that transfers a payload received via the MQTT client's socket to the serial or COM port that the XBee 3 Cellular LTE-M/NB-IoT is active on, as well as the reverse; transfer of a payload received on the XBee 3 Cellular LTE-M/NB-IoT's serial or COM port to the socket of the MQTT client. This is simplest with the XBee 3 Cellular LTE-M/NB-IoT in Transparent mode, as it does not require code to parse or create API frames, and not using API frames means there is no need for them to be queued for processing.

1. To put the XBee Cellular Modem in Transparent mode, set **AP** to **0**.
2. Set **DL** to the IP address of the broker you want to use.
3. Set **DE** to the port to use, the default is 1883 (0x75B). This sets the XBee 3 Cellular LTE-M/NB-IoT to communicate directly with the broker, and can be performed in XCTU as described in Example: MQTT connect.
4. You can make the proxy with a dual-threaded Python script, a simple version follows:

```
import threading
import serial
import socket

def setup():
    """
    This function sets up the variables needed, including the serial port,
    and its speed/port settings, listening socket, and localhost address.
    """
    global clisock, cliaddr, svrsock, ser
    # Change this to the COM port your XBee Cellular module is using.  On
    # Linux, this will be /dev/ttyUSB#
    comport = 'COM44'
    # This is the default serial communication speed of the XBee Cellular
```

```
    # module
    comspeed = 115200
    buffer_size = 4096  # Default receive size in bytes
    debug_on = 0  # Enables printing of debug messages
    toval = None  # Timeout value for serial port below
    # Serial port object for XBCell modem
    ser = serial.Serial(comport,comspeed,timeout=toval)
    # Listening socket (accepts incoming connection)
    svrsock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    # Allow address reuse on socket (eliminates some restart errors)
    svrsock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    clisock = None
    cliaddr = None  # These are first defined before thread creation
    addrtuple = ('127.0.0.1', 17300)  # Address tuple for localhost
    # Binds server socket to localhost (allows client program connection)
    svrsock.bind(addrtuple)
    svrsock.listen(1)  # Allow (1) connection

def ComReaderThread():
    """
    This thread listens on the defined serial port object ('ser') for data
    from the modem, and upon receipt, sends it out to the client over the
    client socket ('clisock').
    """
    global clisock
    while (1):
        resp = ser.read()  ## Read any available data from serial port
        print("Received {} bytes from modem.".format(len(resp)))

        clisock.sendall(resp)  # Send RXd data out on client socket
        print("Sent {} byte payload out socket to client.".format(len(resp)))

def SockReaderThread():
    """
    This thread listens to the MQTT client's socket and upon receiving a
    payload, it sends this data out on the defined serial port ('ser') to the
    modem for transmission.
    """

    global clisock
    while (1):
        data = clisock.recv(4096)  # RX data from client socket
        # If the RECV call returns 0 bytes, the socket has closed
        if (len(data) == 0):
            print("ERROR - socket has closed.  Exiting socket reader thread.")
            return 1  # Exit the thread to avoid a loop of 0-byte receptions
        else:
            print("Received {} bytes from client via socket.".format(len(data)))
            print("Sending payload to modem...")
```

```
            bytes_wr = ser.write(data)  # Write payload to modem via UART/serial
            print("Wrote {} bytes to modem".format(bytes_wr))

def main():
    setup()  # Setup the serial port and socket
    global clisock, svrsock
    if (not clisock):  # Accept a connection on 'svrsock' to open 'clisock'
        print("Awaiting ACCEPT on server sock...")
        (clisock,cliaddr) = svrsock.accept()  # Accept an incoming connection
        print("Connection accepted on socket")
    # Make thread for ComReader
    comthread = threading.Thread(target=ComReaderThread)
    comthread.start()  # Start the thread
    # Make thread for SockReader
    sockthread = threading.Thread(target=SockReaderThread)
    sockthread.start()  # Start the thread

main()
```

This proxy script waits for an incoming connection on localhost (**127.0.0.1**), on port **17300**. After accepting a connection, and creating a socket for that connection (**clisock**), it creates two threads, one that reads the serial or COM port that the XBee 3 Cellular LTE-M/NB-IoT is connected to, and one that reads the socket (**clisock**), that the MQTT client is connected to.

With:

- The proxy script running
- The MQTT client connected to the proxy script via localhost (**127.0.0.1**)
- The XBee 3 Cellular LTE-M/NB-IoT is connected to the machine via USB and properly powered
- **AP**, **DL**, and **DE** set correctly

The proxy acts as an intermediary between the MQTT client and the XBee 3 Cellular LTE-M/NB-IoT, allowing the MQTT client to use the data connection provided by the device. Think of the proxy script as a translator between the MQTT client and the XBee 3 Cellular LTE-M/NB-IoT. The following figure shows the basic operation.

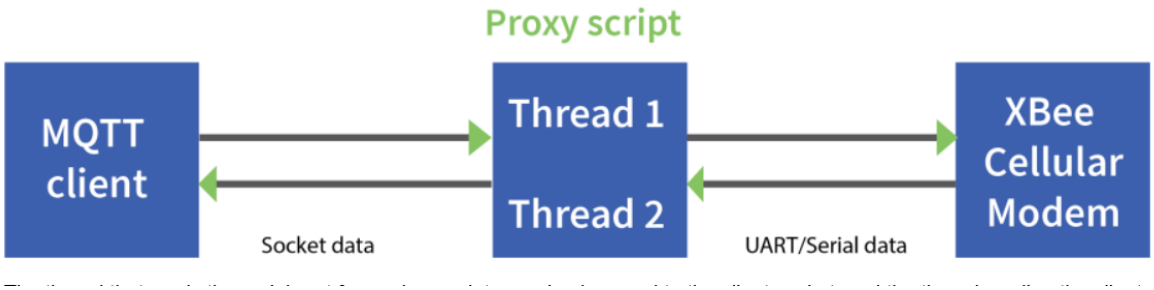

Figure 16: Communication between MQTT, Proxy Script, and Cellular Modem

The thread that reads the serial port forwards any data received onward to the client socket, and the thread reading the client socket forwards any data received onward to the serial port. This is represented in the figure above.

The proxy script needs to be running before running an MQTT publish or subscribe script.

1. With the proxy script running, run the subscribe example from Example: receive messages (subscribe) with MQTT, but change the connect line from **client.connect("m2m.eclipse.org", 1883, 60)** to **client.connect("127.0.0.1", port=17300, keepalive=20**). This connects the MQTT client to the proxy script, which in turn connects to a broker via the XBee 3 Cellular LTE-M/NB-IoT's internet connection.
2. Run the publish example from Example: send messages (publish) with MQTT in a third Python instance (while the publish script is running you will have three Python scripts running at the same time). (Digi International Inc, 2020)

Proposed Application (Smart Route):

Through this method, the MQTT server can receive the data generated by the Raspberry Pi. The Raspberry Pi processes the images seen on the HD definition camera and generates a ratio for the green light time and the number of cars at the intersection. These 2 sets of information will be sent to the cloud server via the process explained above with LTE-M, Xbee 3 cellular modem, and

MQTT server. The server will be able to receive information on specific intersections and their adjacent intersections.

The number of cars will be used to calculate the congestion within the lanes. The number of cars in the lanes will display a color on the street to signify the level of congestion. The color spectrum will consist of blue, green, yellow, orange, and red. The lane will light up blue at 1-3 cars, green at 3-5, yellow at 6-8, orange at 9-11, and red at 12+. The colors will have their intensity of color based on the number of cars. For example, each color will have a light, solid, or dark tone specific to the number of cars. If 1 car is at the intersection the street will be light blue, 2 cars will show a solid blue, and 3 will show a dark blue.

The server will receive a new set of data every 10 seconds and change the colors in a smooth spectrum in real-time. When the server registers 3 cars and then 10 seconds later it registers 8 cars, the server will take the difference between the number of cars and divide it by 10. The resulting value will be incremented or decremented until it reaches the consecutive registered number of cars and colors. This spectrum within the lanes will be sent to users via mobile devices.

The green light ratio information will be used to estimate the amount of time spent within a route. The estimated time will be calculated using the amount of time the traffic light will be red or green and also use the change of the color spectrum within connection lanes at intersections. The color spectrum will not only help show the congestion seen in specific locations but also show a flow of congested traffic. The green light ratio and the flow of color congestion can estimate how much of the congested road will make it past the intersection before the light changes to red. This helps to predict if traffic congestion will increase or decrease at a specific route depending on the time of arrival for the color spectrum at an intersection. The driver will then be redirected to a route that has less flow of congestion to arrive at their destination faster.

Wireless Communication:

In our case, we used one Raspberry Pi to operate as the traffic controller, and another as the server. The two microcontrollers will be connected using the same wireless network to transfer data. Therefore, a wireless communication protocol will not be required between a server and one microcontroller. However, for the smart traffic system to be successful over multiple intersections, we'd need to implement the MQTT protocol, which allows for multi-node connections. In other words, the protocol would allow the capability for wireless communication between the server and multiple traffic controllers. One of the libraries we can use is the Paho MQTT Python client. The following command should be entered:

```
pip install paho-MQTT
```

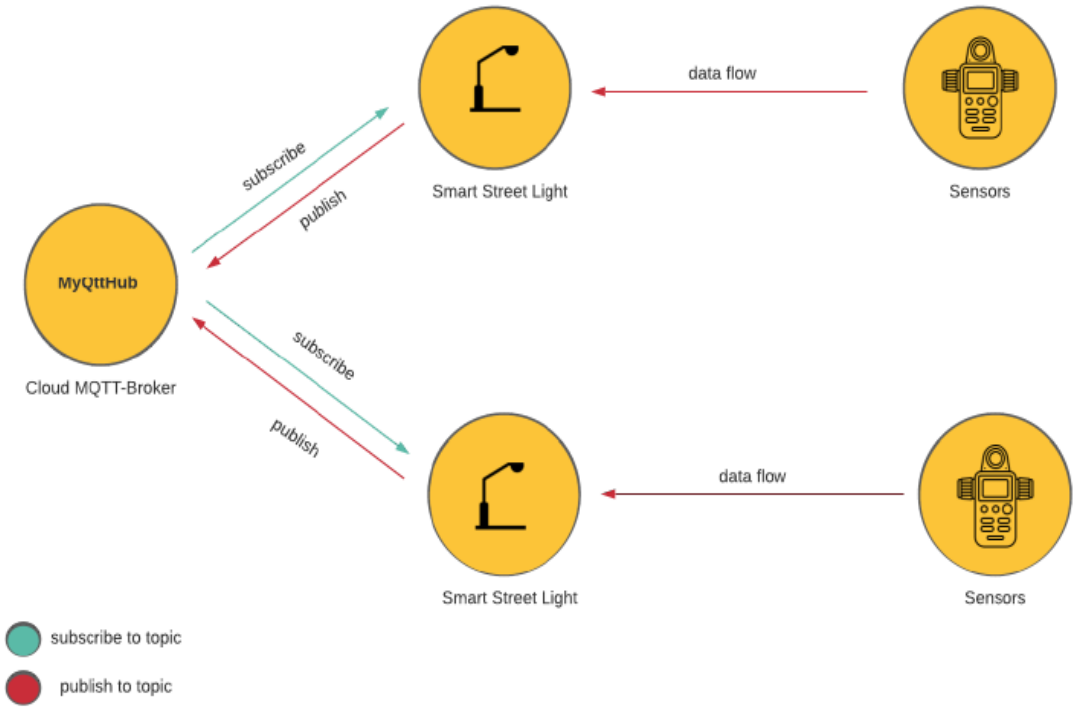

Figure 17: Conceptual Diagram of MQTT Protocol Communicating with Multiple Clients

The MQTT protocol is the optimal form of wireless communication between multiple clients, in this case, would be the smart street lights, as well as the dynamic traffic lights. In this

case, the sensors send data to the smart street lights through the microcontroller, which then publishes the same data to the cloud. A different type of data will be sent back to the microcontroller so that they can adjust its components if necessary. For example, in our smart traffic system, we propose that the microcontroller will send the ratio to the cloud, and the cloud can send back the traffic cycle of that traffic light to the neighboring traffic lights.

Limitations:

Our project is only able to detect cars from one direction using one camera and its associated microcontroller. It is possible to take into account the other three directions of the intersection with more cameras and microcontrollers. In this case, one Raspberry Pi is still being used as the server for the four-way intersection. Another limitation is the field of view of the camera. We have explored the specifications of the Raspberry Pi Camera Module and did not find a field of view or range for the component. This prevented us from determining the number of cars that the camera can detect in a wide range such as at an actual intersection. However, for demonstration purposes, we fit 4 cars into the view and successfully determine the number of cars.

Python Libraries and Car Detection:

Import OpenCV: open-source software that helps solve real-time computer vision and image processing problems.

Import time: Allows various operations regarding time, its conversions, and representations

Import RPI.GPIO: module to control the GPIO interface on the Raspberry Pi.

Import math: standard to use mathematical functions.

Import socket: This allows the connection of two nodes on a network to communicate with each other.

ClassNames = []: Directory that has all the different types of objects which can be detected.

COCO Library: configPath = "/home/pi/Desktop/Object_Detection_Files/ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt"
WeightsPath = "/home/pi/Desktop/Object_Detection_Files/frozen_inference_graph.pb"

Our program uses object detection to count the number of cars in the camera's field of view. The Common Object in Context (COCO) Library is large-scale object detection, segmentation, and captioning dataset. Artificial intelligence is used to label objects using per-instance segmentations to aid imprecise object localization (Tim, 2021). With this trained library of objects, the Raspberry Pi can identify 91 unique objects and provide a confidence rating. The included car images allow the microcontroller to make comparisons with what the camera module sees. If the camera sees an image similar to the stored car image data, it will recognize it as a car and draw a rectangular box around the image with a confidence rating.

Code:

For our demonstration, we have set up our traffic control system that consists of a Raspberry Pi that serves as the traffic control system, and a Raspberry Pi that will be a server that receives data from the traffic control system. The two Raspberry Pis will communicate with each other using a wireless network.

Server:

```
import socket

host = '' # Server IP or Hostname
port = 8005 # Pick an open Port (1000+ recommended), must match the client sport
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Socket created")

#managing error exception
try:
    s.bind((host, port))
except socket.error as msg:
    print(msg)
    print("Bind failed ")

s.listen(5)
print("Socket awaiting messages")
(conn, addr) = s.accept()
print("Connected")

# awaiting for message
while True:
    data = conn.recv(1024)
    data = data.decode('utf-8')

    reply = ''
    print(data)

    reply = "Number of cars sent to server"
    conn.send(str.encode(reply))

conn.close() # Close connections
```

Figure 18: Code for Server Raspberry Pi

For the server code, our objective is to make sure data is being sent from the traffic microcontroller. Port 8005 will be enabled for both Raspberry Pis to communicate with each other. If the server has trouble sending data, then it will send a message saying that the connection failed. Otherwise, the socket will allow a certain time to see if they detect any data being received. Once the Raspberry Pis are connected, the server will now wait for data to be received. Once the number of cars sent to the server is received, the code will output that data on the terminal. As long as the two programs are running, the server Raspberry Pi will continue to receive data messages.

Traffic Controller:

```
import cv2
import time
import RPi.GPIO as GPIO
import math

import os

import socket

HOST = '192.168.1.17' # Enter IP or Hostname of your server
PORT = 8005 # Pick an open Port (1000+ recommended), must match the server port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST,PORT))


#These are just the libaries needed for the program to work

#thres = 0.45 # Threshold to detect object


#This right here refers to a specific directory that has all the different types of
#objects which can be detected b this detector. Ex: car, horse, cat...
classNames = []
classFile = "/home/pi/Desktop/Object_Detection_Files/coco.names"
with open(classFile,"rt") as f:
    classNames = f.read().rstrip("\n").split("\n")

configPath = "/home/pi/Desktop/Object_Detection_Files/ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt"
weightsPath = "/home/pi/Desktop/Object_Detection_Files/frozen_inference_graph.pb"



#These specifc parameters werre already set when I go it form online
#These are set for the detection to work
net = cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,320)
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)

#Here you need to write this if you would like to use pins on the raspbery pi
GPIO.setmode(GPIO.BCM)

#Here I assigned the specific numbers to each color tp represetn the appropiate pin
red = 9
yellow = 10
green = 11

#This makes the pins which we are going to use for the LED lights as the outputs with 5V output
GPIO.setup(red,GPIO.OUT)
GPIO.setup(yellow,GPIO.OUT)
GPIO.setup(green,GPIO.OUT)

#This is the detector. It uses the folders mentioned previously and draws a rectangular box around the image
def getObjects(img, thres, nms, draw=True, objects=[]):
    classIds, confs, bbox = net.detect(img,confThreshold=thres,nmsThreshold=nms)
    #print(classIds,bbox)
    if len(objects) == 0: objects = classNames
    objectInfo =[]
    car = "car"
    if len(classIds) != 0:
        for classId, confidence,box in zip(classIds.flatten(),confs.flatten(),bbox):
            className = classNames[classId - 1]
            if className in objects:
                #If a car is detected, this will assign objectInfo to look like this for each car: [[car]]
                objectInfo.append([car])
                #objectInfo.append([box,className])
                if (draw):
                    cv2.rectangle(img,box,color=(0,255,0),thickness=2)
                    cv2.putText(img,classNames[classId-1].upper(),(box[0]+10,box[1]+30),
                    cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
                    cv2.putText(img,str(round(confidence*100,2)),(box[0]+200,box[1]+30),
                    cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
```

```
                    cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
    #This returns the image so that we can see it
    return img,objectInfo

#This is the start of our program
if __name__ == "__main__":
    #This assigns the cycle length in seconds
    #This can be modified according to your liking
    cycle = 5


    #This is our most important loop
    #This will keep looping if the program is running
    #http_server = HTTPServer((host_name, host_port), MyServer)
    #print("Server Starts - %s:%s" % (host_name, host_port))
    #try:
    #    http_server.serve_forever()
    #except KeyboardInterrupt:
    #    http_server.server_close()

    while True:
        #http_server = HTTPServer((host_name, host_port), MyServer)
        #We set the cap to be the camera and to hold all the necessary information
        cap = cv2.VideoCapture(0)
        #This sets the row and height of the image output
        cap.set(3,640)
        cap.set(4,480)
        #This is the expose of the camera (50%)
        cap.set(10,50)
        text = "There are no cars in front of the camera "

        #This will read the image
        success, img = cap.read()
        #This will look at the image and use the function defined before to check if there are any cars
        #In this image. I specifically wrote 'car' and this defined function will only for cars.
        #If the program detects cars, any car detected with a (35%) confidance level will be marked
        result, objectInfo = getObjects(img,0.30,0.4, objects=['car'])

        #This line just tells us if anything was detected.
        print(objectInfo)

        #This line outputs the image so that we can look at it
        cv2.imshow("Output",img)

        #Abything related to the previou simage is deleted. Now on the next loop a fresh image will be used for c
        del cap

        #If 1 car was detected, the specific times for green light and red light are calculated and assigned
        if objectInfo == [['car']]:
            #print("1 CAR")
            text = "There is 1 car in front of the camera"
            R = 1 / 1
            Gama = (math.sqrt(R))/(math.sqrt(R)+1)

            g = Gama*(cycle)
            r = cycle-cycle*Gama
        #If 2 cars were detected, the specific times for green light and red light are calculated and assigned
        elif objectInfo == [['car'], ['car']]:
            text = "There are 2 car in front of the camera"
            R = 2 / 1
            Gama = (math.sqrt(R))/(math.sqrt(R)+1)

            g = Gama*(cycle)
            r = cycle-cycle*Gama
        #If 3 cars were detected, the specific times for green light and red light are calculated and assigned
        elif objectInfo == [['car'], ['car'], ['car']]:
            text = "There are 3 car in front of the camera"
            R = 3 / 1
            Gama = (math.sqrt(R))/(math.sqrt(R)+1)

            g = Gama*(cycle)
            r = cycle-cycle*Gama
        #If 4 cars were detected, the specific times for green light and red light are calculated and assigned
```

```
#If 4 cars were detected, the specific times for green light and red light are calculated and assigned
elif objectInfo == [['car'], ['car'], ['car'], ['car']]:
    text = "There are 4 car in front of the camera"
    R = 4 / 1
    Gama = (math.sqrt(R))/(math.sqrt(R)+1)

    g = Gama*(cycle)
    r = cycle-cycle*Gama
#If no cars are detected, each green and red cycle lengh is just half of the total cycle length
else:
    g = cycle/2
    r = cycle/2

#This turns on the GREEN Light and every other light is turned off
GPIO.output(red,False)
GPIO.output(yellow,False)
GPIO.output(green,True)
#The program waits for the appropiate green wait time
time.sleep(g)
#Just for keeping track, the specific time is displayed
print(g,"seconds as green")


#This turns on the RED Light and every other light is turned off
GPIO.output(red,True)
GPIO.output(yellow,False)
GPIO.output(green,False)
#The program waits for the appropiate RED wait time
time.sleep(r)
#Just for keeping track, the specific time is displayed
print(r,"seconds as red")

while True:
    command = text
    s.send(str.encode(command))
    reply = s.recv(1024)
    reply = reply.decode('utf-8')
    if reply == 'Terminate':
        break
    print(reply)
    break
#This is the end of the loop. Now the loop will start once more.
#A new fres image will be captured and the calculations will be done again
```

Figure 19: Code for Traffic Microcontroller

There are three main sections for the traffic microcontroller code: Parameters, Object Detection, and Traffic Control.

For parameters, the libraries that were installed were the OpenCV library, which is used for detecting cars. The time library is built-in by the IDLE application, and GPIO pins are enabled for signals to be sent to the traffic lights. Then, the traffic lights are assigned to each pin of the microcontroller. Finally, the roles of the GPIO pins will be declared, which is to send 5 V outputs to the appropriate lights. For Object detection, we will use OpenCV to detect cars by marking them

with a green box and solely classifying it as a car and nothing else. The Traffic Control section of the code will dictate the traffic cycle, as well as the proper signals sent to the traffic lights. The traffic control will take data from object detection and calculate the traffic cycle based on the interval set at the beginning of the code, 30 seconds in our case. It will start a new cycle based on the previous cycle's data.

Installation and Testing:

1. **Installation**
    a. Follow instructions on how to integrate each supplementary module to the main Raspberry Pi microcontroller.
    b. Put casing on necessary devices.
    c. Connect DC Power Supply to the Raspberry Pi.
    d. Verify connections using the following tests.
2. **Testing**
    a. **Camera Test**
        i. Ensure traffic lights are on the same coordinate plane.
        ii. Confirm that traffic lights are operating in all lighting conditions. Keep in mind that the traffic control system will be working at all times.
        iii. Simulate to see if the microcontroller can detect the location of the image taken by the camera.
        iv. If part iii is successful, then the smart traffic control system passes the Camera Test and should be moved on to the Car Test. Otherwise, troubleshoot issues and repeat steps i-ii until it is resolved.
    b. **Car Test**

i. Place one car at an intersection.

ii. Simulate to determine if a car is detected in that direction by the camera. Verify if the traffic light changes from red to green, which would indicate the car to move.

iii. If step ii is successful, repeat steps i-iii in another scenario with at least two cars

iv. Traffic lights should change for the intersection with the greater number of cars. If true, the smart control system has passed the car test, and testing is now complete.

c. Server Test

i. Follow instructions for both Camera and Car Tests.

ii. Change HDMI output to server Raspberry Pi.

iii. See if the number of cars data is received by observing the output of the terminal.

iv. If not successful, troubleshoot any potential issue such as code errors, or wrong port configured, and so on.

Bill of Materials:

Shown below is the breakdown of the prices of each component we plan on using for our smart traffic control system, along with the quantity that we plan on ordering:

Product	Unit Price	Quantity	Total
CanaKit Raspberry Pi 4 4 GB Started PRO Kit - 4GB RAM	$99.99	2	$199.98

Raspberry Pi Camera Module V2 8MP 1080P for Raspberry Pi 4 3 B+ Zero	$29.95	2	$59.90
Low Voltage Labs Pi Traffic Light for The Raspberry Pi (5 pack)	$24.99	1	$24.99
Toy Cars (6 Pack)	$4.89	1	$4.89
Grand Total			$289.76

Table 3: Bill of Materials

Before building the actual design, the components were purchased in the Summer to avoid any delays.

Project Timeline:

Below is our timeline with the anticipated dates and tasks to indicate our progress in building the traffic control system. The tasks highlighted in yellow indicate the tasks done from Winter 2021 to Spring 2021. The green tasks are tasks that will be done during the summer. Lastly, the blue tasks will be done in Fall 2021.

ID	Task Mode	Task Name	Duration	Start	Finish
1		Request mentorship with Professor Ali	2 days	Mon 2/1/21	Tue 2/2/21
2		introductory Meeting with Professor Ali	1 day	Wed 2/3/21	Wed 2/3/21
3		Read papers provided by Professor Ali	7 days	Thu 2/4/21	Fri 2/12/21
4		Research Local/State/Government Standards for smart traffic system	1 day	Mon 3/1/21	Mon 3/1/21
5		Research current smart traffic systems	2 days	Mon 3/1/21	Tue 3/2/21
6		Research on how an IP camera can be integrated into the traffic system	2 days	Mon 3/1/21	Tue 3/2/21
7		Analyze the advantages of having a HD IP Camera in our traffic system	2 days	Thu 2/25/21	Fri 2/26/21
8		Discuss with group about research	1 day	Wed 3/3/21	Wed 3/3/21
9		**Determine Components required for traffic control system**	**2 days?**	**Wed 3/24/21**	**Thu 3/25/21**
10		Meet with Professor Ali to confirm/discuss required components	1 day	Wed 3/24/21	Wed 3/24/21
11		Software	1 day	Thu 3/25/21	Thu 3/25/21
12		Housing/Size	1 day	Thu 3/25/21	Thu 3/25/21
13		Hardware	1 day	Thu 3/25/21	Thu 3/25/21
14		Create blueprint for dynamic traffic control system	2 days	Fri 3/26/21	Sun 3/28/21
15		Start Working on List of Activities	5 days	Mon 4/12/21	Fri 4/16/21
16		**Design**	**3 days**	**Wed 4/14/21**	**Fri 4/16/21**
17		Consult Professor Ali regarding the Activites required to build traffic system	1 day	Mon 4/19/21	Mon 4/19/21
18		**Research components needed for design**	**3 days**	**Tue 4/20/21**	**Thu 4/22/21**
19		Determine quantity of desired component	1 day	Tue 4/20/21	Tue 4/20/21
20		Find best working component for lowest price	1 day	Wed 4/21/21	Wed 4/21/21
21		**Start Bill of Materials**	**2 days**	**Fri 4/23/21**	**Sat 4/24/21**
22		Resarch primary vendors for each component	2 days	Wed 4/21/21	Thu 4/22/21
23		Create table and list all products and prices	1 day	Fri 4/23/21	Fri 4/23/21
		Protocols			
31		Initiate building prototype of dynamic traffic control system	23 days	Wed 9/1/21	Fri 10/1/21
32		Ensure dynamic traffic control system complies with standards, codes and regulations	5 days	Mon 10/4/21	Fri 10/8/21
33		Run simulations	2 days	Mon 10/11/2	Tue 10/12/2
34		Troubleshoot any issues with dynamic traffic control system	24 days	Wed 10/13/21	Mon 11/15/21
35		Run simulations with modified dynamic traffic control system	3 days	Tue 11/16/21	Thu 11/18/21
36		Verify prototype has no flaws/deficiencies	2 days	Fri 11/19/21	Mon 11/22/21
37		Find and analyze observations/data	7 days	Tue 11/23/21	Wed 12/1/21
38		Start writing Final report	4 days	Thu 12/2/21	Tue 12/7/21
39		Revise Final report	6 days	Wed 12/8/21	Wed 12/15/2
40		Submit Report to Professor Ali	1 day	Thu 12/16/2	Thu 12/16/2

Figure 20: Project Schedule

References

“Attention Drivers! Turn off Your Idling Engines.” n.d. Environmental Defense Fund. https://www.edf.org/attention-drivers-turn-your-idling-engines#:~:text=For%20every.

Bosson, M. (2021, September 20). NB-IoT vs LTE-M: A comparison of the two IoT technology standards. https://Blog.Onomondo.Com/Nb-Iot-vs-Lte-m-a-Comparison-of-the-Two-Iot-Technology-Standards

"Bullseye - The New Version Of Raspberry Pi OS - Raspberry Pi". 2021. Raspberry Pi. https://www.raspberrypi.com/news/raspberry-pi-os-debian-bullseye/./

CiteSeerX https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.3186&rep=rep1&type=pdf#:

Craggs, Ian. 2021. "Eclipse Paho | The Eclipse Foundation". Eclipse.Org. https://www.eclipse.org/paho/index.php?page=clients/python/index.php.

Digi International Inc (2020, November 05). Use MQTT over the XBee Cellular Modem with a PC. https://Www.Digi.Com/Resources/Documentation/Digidocs/90002258/Tasks/T_use_mqtt.Htm.

EPA, Environmental Protection Agency, https://www.epa.gov/energy/learn-about-energy-and-its-impact-environment#:~:text=All%20forms%20of%20electricity%20generation,and%20land%2C%20but%20it%20varies.&text=Producing%20and%20using%20electricity%20more,pollution%20emitted%20as%20a%20result.

"IDLE — Python 3.10.1 Documentation". 2021. Docs.Python.Org. https://docs.python.org/3/library/idle.html.

Karr, Steve. “Bluetooth Mesh Networking FAQs.” *Bluetooth® Technology Website*, www.bluetooth.com/learn-about-bluetooth/recent-enhancements/mesh/mesh-faq

Kingson, Jennifer. “The Future of ‘Smart’ Cities Is in Street Lights.” *Axios*, 11 Feb. 2021, www.axios.com/smart-cities-street-lights-859992a6-6931-48e5-81ba-7f0a0b8058d9.html.

Levison, Sarah. “Advantages of LED Lights.” *The Lightbulb Co. UK*, 2 Mar. 2020, www.thelightbulb.co.uk/resources/ultimate-guide-led-lights-advantages-leds.

Ltd, Raspberry Pi. “4 Model B Specifications –.” *Raspberry Pi*, www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications.

Mpl.ch. 2021. IP rating table. https://www.mpl.ch/info/IPratings.html

“NYC DOT - Infrastructure - Traffic Signals.” *NYC DOT*, 2021, https://www1.nyc.gov/html/dot/html/infrastructure/signals

ONVIF. 2021. Profile S - ONVIF. https://www.onvif.org/profiles/profile-s/

R. Yaqub, A. Joyo, K. Heidary, and N. Madamopoulos, "Green-Initiative for Realizing Adaptive Traffic-Signals by Exploiting Smart-Antenna Technology (GREATEST)," in IEEE Intelligent Transportation Systems Magazine, vol. 12, no. 4, pp. 66-77, winter 2020, doi: 10.1109 / MITS .2018.2879229.

Smartcities.ieee.org. 2021. Home. https://smartcities.ieee.org/

“Smart Cities: How IP Surveillance Is Shifting City Safety.” *Vector Security*, 31 Dec. 2015, www.vectorsecurity.com/bizblog/smart-cities-how-ip-surveillance-is-shifting-city-safety.

State.nj.us. 2021. https://www.state.nj.us/transportation/eng/elec/ITS/itsenglishjuly2001/pdf/ipcamera.pdf

Sternhoff, Johan. “LTE-M vs NB-IoT – a Guide and in Depth Comparison.” *Telenor Connexion*, 20

Oct. 2021, www.telenorconnexion.com/iot-insights/lte-m-vs-nb-iot-guide-differences.

Tim. “Object and Animal Recognition with Raspberry Pi and Opencv - Tutorial.” Core Electronics, 25 Nov. 2021, https://core-electronics.com.au/tutorials/object-identify-raspberry-pi.html.

“The Evolution of Street Lighting.” *Zeta Led Lighting Ltd*, 10 June 2021, http://zetaled.co.uk/news-2017/evolution-of-street-lighting

Walker, Ameena. 2018. “New York Drivers Spent an Average of 91 Hours in Traffic Last Year.” Curbed NY. February 6, 2018. https://ny.curbed.com/2018/2/6/16979696/new-york-city-traffic-congestion-second-worst

“What Is Zigbee Wireless Mesh Networking?” *Digi*, 2021, www.digi.com/solutions/by-technology/zigbee-wireless-standard