Lab 4 Report
IoT System of Garage Door and Parking Distance Indicator

Online Link:
This lab is available as part of my online portfolio at: https://github.com/DemonPiranha/ITC-441-labs.git

Objective

The purposes of this lab are to:

- Implement an Event Hub for publish/subscribe notifications between devices.
- Develop a communications protocol for devices across the event bus.
- Establish more complex conditions for the actuator involving multiple sensors.

Materials
I used the following materials to accomplish this lab:

- 3 x Arduino Wemos D1 Mini with ESP8266
- 1 x Raspberry Pi 3b+
- 4 x MicroUSB Cable with power source
- Wifi device, this lab uses a smartphone with a wifi hotspot
- 1 x SR04 Ultrasonic sensor
- 3 x breadboards
- 1 x Reed switch
- 1 x Green LED
- 1 x Yellow LED
- 1 x Red LED
- 3 x 100 Ω Resistor (BrBlBrGl)
- 8 x Jumper Wires

References
I used the following resources in this lab:

- This lab assumes the hardware setup of the stoplight and ultrasonic sensor from the previous lab
- https://www.instructables.com/How-to-Use-MQTT-With-the-Raspberry-Pi-and-ESP8266/ - mosquitto installation
- https://stackoverflow.com/questions/30207649/address-already-in-use-error-in-mosquitto - mosquitto service controlling
- https://docs.arduino.cc/tutorials/generic/digital-input-pullup - Correct digital input code
- https://pubsubclient.knolleary.net/api - PubSubClient API Documentation
- https://github.com/luisllamasbinaburo/Arduino-MedianFilter - Median Filter Library Documentation
- https://1drv.ms/w/s!Ais81h0TsK5NhqEUuevEs8zMavzLag?e=hdaL2m - Todd Berrett's example write-up. Used for formatting.

Procedures:

Raspberry Pi setup

1. This will setup a RaspberryPi as an MQTT borker using Mosquitto
2. Get a RaspberryPi with fresh Raspian install preferably (This will avoid any port restrictions)
3. Run normal software updates using `sudo apt-get update` followed by `sudo apt-get upgrade`
4. Install mosquitto with `sudo apt-get install mosquitto -y` and `sudo apt-get install mosquitto-clients -y`
5. Edit the mosquitto configuration file using `sudo nano /etc/mosquitto/mosquitto.conf`
6. Comment out this line: `include_dir /etc/mosquitto/conf.d`
7. Add the following to the end of the file this order: `allow_anonymous true` `listener 1883`
8. Exit the file saving it.
9. Reboot the Pi to apply all changes with `sudo reboot`
10. Mosquitto is now installed and will run automatically in the background.
11. `systemctl status mosquitto.service` can be used to see the status of the service.

Ardunio Setup

12. Configure the Arduino IDE to interpret ESP8266 boards. Follow this guide to do that: https://arduino-esp8266.readthedocs.io/en/latest/installing.html
13. For the first Ardunio with the stoplight components use Lab4-1.ino code. For the second Arduino with the ultrasonic sensor use Lab4-2.ino code. For the second Arduino with the reed switch use Lab4-3.ino code Then follow the steps below for each.
14. Copy the Ardunio code in this project into an Arduino sketch. Adjust the global variables for SSID and password to match the wifi network you will be using. Also adjust the vale of the MQTT server variable to match the IP address of the Raspberry Pi setup earlier.
15. Make sure the Arduino is properly connected to the computer with the computer via a USB cable.
16. Upload the code. Once the upload has completed the built-in Arduino LED should now be rapidly flashing. (This indicates awaiting connection to wifi)
17. Once a connection is established the built-in LED will stop blinking
18. Distance from the sensor will now be translated into color LED status while the reed sensor is open.

How are they talking, how is the system setup, what are the relevant points, what automation did you set up (Answer this)

System view

This project simulates an environment of a garage with a door, and parking spot. Two sensors and an actuator are used which communicate to a central event hub for intercommuicaiton.

Stoplight module

This part includes the Ardunio and the three LEDs to model a stop light. The user is able to receive visual feedback from these lights determined by the ultrasonic distance module while the garage door is open. Once the door is closed all lights are turned off until a new light command is received.

Ultrasonic distance module

This part includes the second Ardunio and an SR04 ultrasonic distance sensor. The module as a whole receives distance readings from the sensor and sends MQTT messages for the various stoplight colors depending on the distance from the sensor. With green being beyond 55cm, yellow between 50cm and 22cm, red between 20cm and 12cm, and flashing yellow and red warning below 10cm. The module has logic included to stop sending commands if it has received the communication that the garage door is closed. This simulates the car bing fully in or fully out of the garage where the stoplight system is no longer needed for it's distance sensing function.

Reed switch module

This simple module contains a reed switch that similates a garage door being open or closed. The sensor is hypothetically located at the bottom of the garage door. This allows the sensor to show when the door is definitely secured in the closed position however any location away from the sensor will read as open simulating the door is open to some capacity.

Raspberry Pi module

This device is used to serve as the central event up using mosquitto MQTT broker. MQTT communication is directed to this device and sent back out on specified topic channels. In this lab a single topic is used for communication between all devices to simplify arunio programming.

All together

All together this system allows for a user to park a vehicle into a garage with the perfect distance to the wall of the garage using the stoplight indicator to find track the distance. This system will not provide feedback when the garage is closed and not in use.

Component View

Model of the functionality, logical flow, and components of the system.

a. Stoplight Module
    i. Setup:
        1. The system configures built-in outputs.
        2. Wifi system is enabled and the connection process begins. This sub-process will loop until a connection is established. The program is then able to continue.
        3. MQTT process is started and connection to the MQTT broker is established. If connection fails the built-in indicator light stays solid represntintg a fault.
        4. The primary loop then begins
    ii. Primary Loop:
        1. This loops primary job is to listen to web requests and MQTT messages and perform actions based on this. For this view only MQTT will be explained.
        2. If "green" message is heard:
            a. Cycle is turned off.
            b. tooClose is turned off.
            c. greenlight function is called which switches the green GPIO pin to HIGH and the rest LOW
        3. If "yellow" message is heard:
            a. Cycle is turned off.
            b. tooClose is turned off.
            c. yellowlight function is called which switches the yellow GPIO pin to HIGH and the rest LOW
        4. If "red" message is heard:
            a. Cycle is turned off.

b. tooClose is turned off.
c. redLight function is called which switches the red GPIO pin to HIGH and the rest LOW

5. If "tooClose" message is heard:
    a. Cycle is turned off.
    b. tooClose is turned on.
    c. This causes a loop to run that flashes the red and yellow lights.
6. If "allOff" or "closed" message is heard:
    a. Cycle is turned off.
    b. tooClose is turned off.
    c. allLightsOff function is called which switches all GPIO pins to LOW

b. UltraSonic Module
    i. Setup:
        1. The system configures built-in outputs.
        2. Wifi system is enabled and the connection process begins. This sub-process will loop until a connection is established. The program is then able to continue.
        3. MQTT process is started and connection to the MQTT broker is established. If connection fails the built-in indicator light stays solid represntintg a fault.
        4. The primary loop then begins
    ii. Primary Loop:
        1. This loops primary job is to listen to the distance from the sensor and perform actions based on this.
        2. If distance from sensor is beyond 55cm:
            a. Publish MQTT message of "green" to the MQTT broker on topic "garage_system"
            b. If the light has already been green then no message is sent
        3. If distance from sensor is between 50cm and 22cm:
            a. Publish MQTT message of "yellow" to the MQTT broker on topic "garage_system"
            b. If the light has already been yellow then no message is sent
        4. If distance from sensor is between 20cm and 12cm:
            a. Publish MQTT message of "red" to the MQTT broker on topic "garage_system"
            b. If the light has already been red then no message is sent
        5. If distance from sensor is below 10cm:
            a. Publish MQTT message of "tooClose" to the MQTT broker on topic "garage_system"
            b. This triggers the stoplight to start flashing red and yellow lights
            c. If the light has already been tooClose then no message is sent

c. Reed Switch Module
    i. Setup:
        1. The system configures built-in outputs.
        2. Wifi system is enabled and the connection process begins. This sub-process will loop until a connection is established. The program is then able to continue.
        3. MQTT process is started and connection to the MQTT broker is established. If connection fails the built-in indicator light stays solid represntintg a fault.
        4. The primary loop then begins
    ii. Primary Loop:
        1. This loops primary job is to read the reed switch and perform actions based on this.
        2. If the reed switch is open:

   a. Publish MQTT message of “open” to the MQTT broker on topic “garage_system”
   b. If the “open” message has already been sent no message will be sent.
3. If the reed switch is closed:
   a. Publish MQTT message of “closed” to the MQTT broker on topic “garage_system”
   b. If the “closed” message has already been sent no message will be sent.

Distance Checking Imporvments
This time around I spent sometime figuring out a better solution for data filtering of the ultrasonic sensor. I decided to use a rolling median filter that would take 5 samples, find the median, and return the median value. This would roll along with new raw data being put in fron the sensor. This helped to significantly reduce the outliers experienced in the previous version. It also did not appear to introduce any significant lag to the system. I could probably raise the sample size but I kept it at the default for now.

Stoplight Logic
On the stoplight module some additional logic was made to handle the new open and closed commands that were available. This allowed the stoplight to turn off all lights when the closed command was received. This simulates a garage door being closed and all lights shutting off due to them no longer being needed. Manual commands to the stoplight are still available however.

MQTT Communication
The server is then able to read these requests like any other and performs it’s set actions depending on those requests. Another feature I built is that of redundant http requests not being sent. If a color has already been sent then it will no longer send any more requests while the distance remains within the set parameter. Once the distance goes past this parameter a new http request for a different color or command may be sent and then the process repeats. This reduces unnecessary http requests being sent unless a legitimate update occurs.
EXPLAIN MORE

Observations

I really enjoyed this lab once I got pass the difficulties I had. Learning MQTT was relatively easy and made me wish we used it sooner. It is cool to finally see a few devices all talking to eachother and reacting to the various inputs. It was fun making little adjustments to the code and then testing to to see if the behaviour matched expectation. I struggled in the beginning to get the MQTT broker up and running as my Raspberry Pi was most likely blocking communications on the 1883 port. This was only able to be solved with a complete reinstall of the operating system. All of the trouble burned a lot of my available time which made it more difficult to have the time desired to create quality code for the ardunios. I have denefetily started to see how building on top of previous labs is starting to take it’s toll on my code with previous systems becoming clunky in some party. I hope to have more time next time to do some house keeping on my code.

Thought Questions

1. How does the communication between devices change with the shift to using an event hub? How does this facilitate greater scalability? What things did you do to learn to use MQTT?

With the shift to an event hub all communication is routed through a single device. With a central communication location many other devices can be connected to this hub allowing for large expandability. This also means that there is a single point of failure for the communication. This isn't a big deal though as there are several single points of failure in any IoT system. With the hub it makes it possible for devices to be added on with only a connection to the hub neded. This simplifies the networking thought needed to do multiple devices that are directly connected. To learn MQTT I reviewed the class slides as well as watched a few youtube videos on the technology. Then using various gudies I learned the commands that could be used.

2. What are strengths and weaknesses of the direct communication between the sensor and actuator? What are strengths and weaknesses of the event hub? Which do you feel is better for this application?

Some benefits of direct communication is that it simplifies device setup as one can act as host and the other can send to it. This doesn't make it easy to do communication between the two however. This is where MQTT really shines. The ability to do back and forth communication allows logic to be spread out through the devices and allows for easier expandability with out much changes to any host devices. This ability however comes with the addition of a broker device which can be separate from the sensors however is another device to operate within a IoT system. Having a central communication point however makes sensor programing easier as there is only one host connection. Loosing this broker device spells disaster for the devices communicating via MQTT where was direct communication could be designed to still work without different devices. MQTT is definitely better for this application as managing multiple devices with direct connection could get complicated. A direct MQTT communication method that could work like a mesh without needed a broker would be a cool technology.

3. What was the biggest challenge you overcame in this lab?

The biggest challenge I had with this lab was issues configuring the Mosquitto MQTT broker. Initially I attempted to setup a username and password to allow for a more secure system however this did not work out so to simplify I removed it. Even after this however I was still having issues getting MQTT to be access from outside the pi. I spent hours fiddling with settings trying to find the problem to no avail. Out of desperation I decided to do a clean install of raspian on the pi and then configure everything. This resulted in positive results and I was able to communicate to the broker from outside the pi. After this was accomplished ardunio code was much smoother.

4. Please estimate the total time you spent on this lab and report?

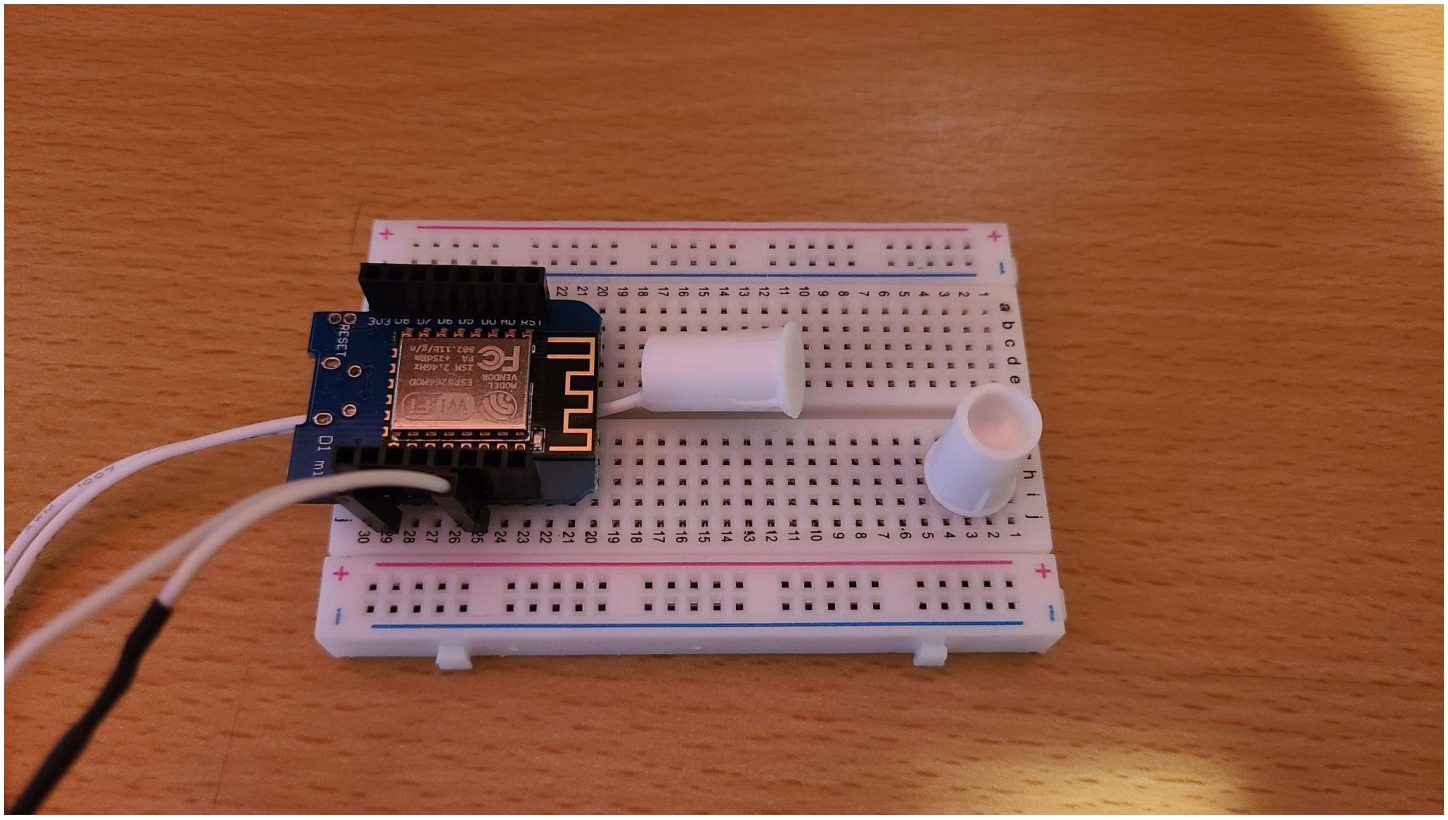
I spent about 12 hours on the project with 6 hours on this lab report.

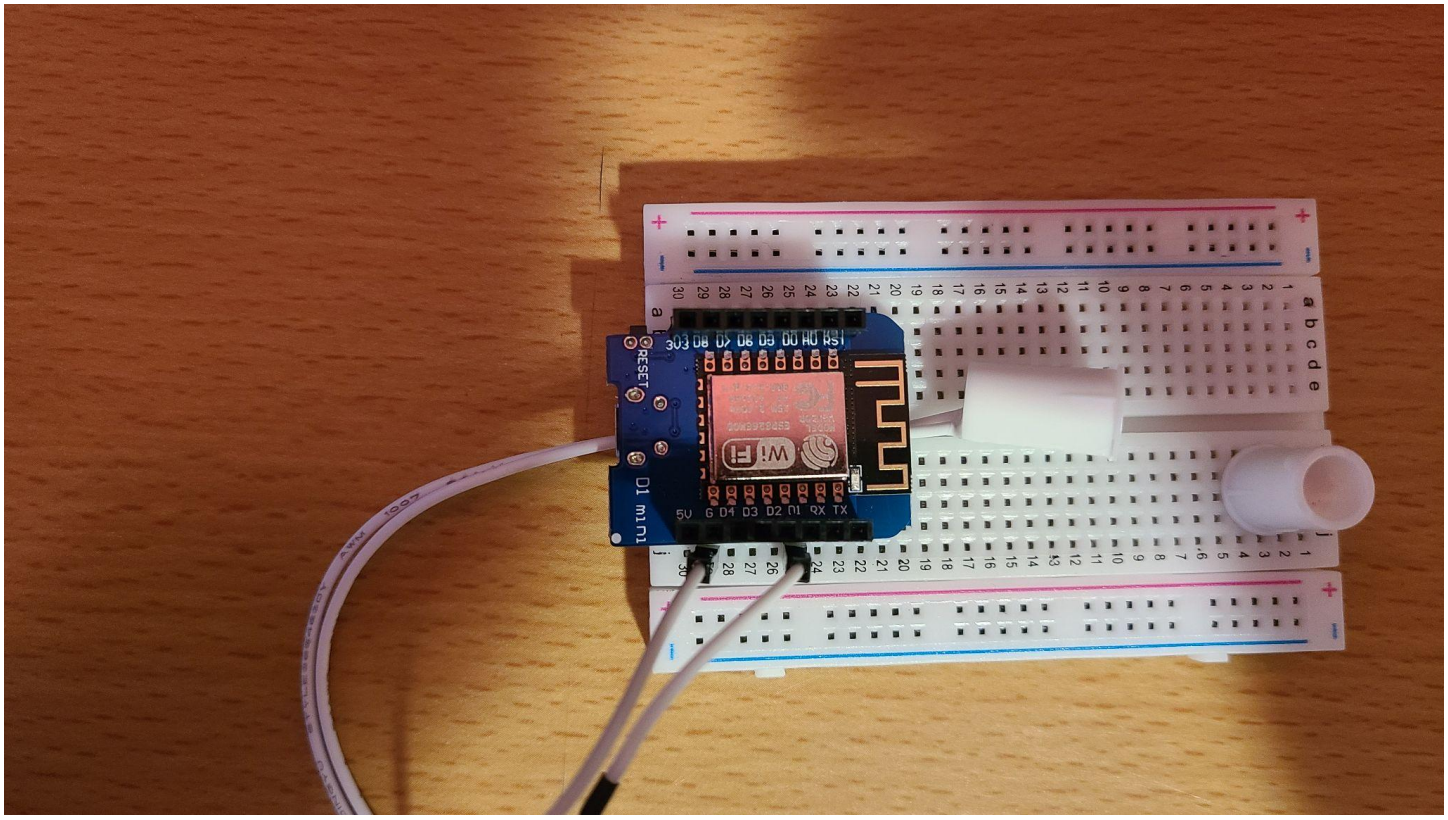
Certification of Work

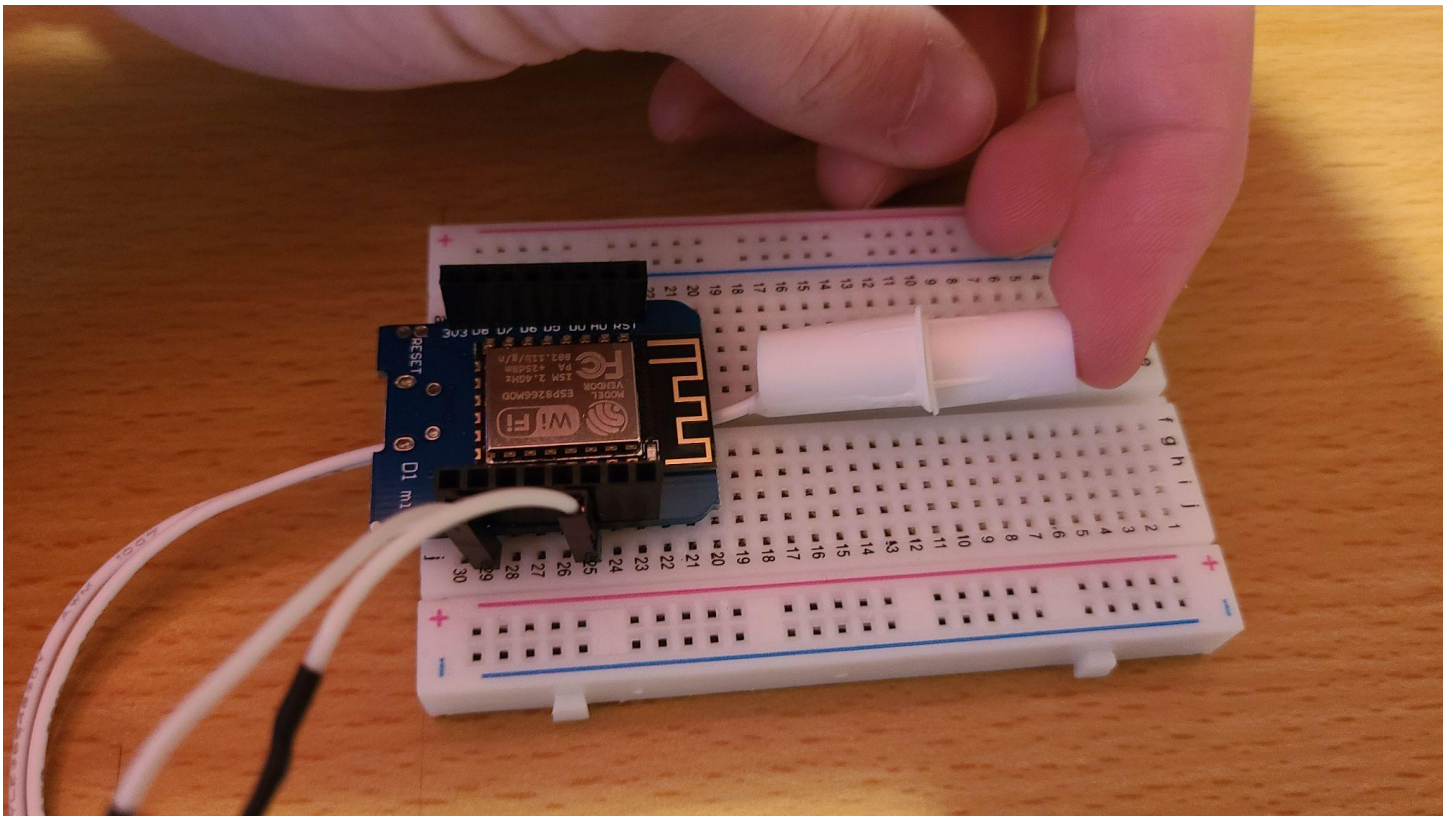
I certify that the solution presented in this lab represents my own work. In the case where I have borrowed code or ideas from another person, I have provided a link to the author's work in the references and included a citation in the comments of my code.

--Blake Porter

Appendix 1: Physical Components Layout of Ultrasonic sensor assembly






RESET
D1 m1
Wi Fi

Appendix 2: Ardunio Code

(available at https://github.com/DemonPiranha/ITC-441-labs/tree/main/Lab%204)

Due to extensive code revision and additions full code will be included.

Stoplight Ardunio code:

```
// This is code for a Stoplight Module


#include <ESP8266WiFi.h>

#include <PubSubClient.h> // Allows us to connect to, and publish to the MQTT broker


//Sets some global variables

const char* ssid = "Blake-Hotspot";

const char* password = "blakepublic";

WiFiServer server(80);

IPAddress local_IP(192, 168, 157, 184);

IPAddress dns(192, 168, 91, 254);

IPAddress gateway(192, 168, 91, 254);

IPAddress subnet(255, 255, 0, 0);

unsigned long startMillis;

bool cycle = false;

bool tooClose = false;


// MQTT configs

const char* mqtt_server = "192.168.157.117";

const char* mqtt_topic = "garage_system";

const char* clientID = "stoplight";


// Sets up the Wifi and MQTT client

WiFiClient wifiClient;
```

```
PubSubClient MQTTclient(wifiClient);


//Setup for pins, clears all of them, starts MQTT connection.
void setup() {


  //Setup serial and all LED outputs then clears the LEDs off
  Serial.begin(9600);
  pinMode(D1, OUTPUT);
  pinMode(D2, OUTPUT);
  pinMode(D3, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  allLightsOff();


  //Sets Static IP for this device.
  if (!WiFi.config(local_IP, dns, gateway, subnet)) {
    Serial.println("STA Failed to configure");
  }


  //Using the Wifi library begin connecting to the set network and provide status updates.
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.disconnect();
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  redLight();
  while (WiFi.status() != WL_CONNECTED) {
    digitalWrite(LED_BUILTIN, HIGH);
```

```
    delay(100);
    Serial.print(".");
    digitalWrite(LED_BUILTIN, LOW);
    delay(100);
  }
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  yellowLight();
  delay(100);
  Serial.println();
  Serial.println("(( WiFi connected ))");
  Serial.println(WiFi.localIP());
  Serial.println(WiFi.gatewayIP());
  Serial.println(WiFi.dnsIP());
  Serial.print("RRSI: ");
  Serial.println(WiFi.RSSI());


  //Starts the http server
  server.begin();
  Serial.println("((( Server Started )))");


  // Connect to MQTT Broker
  MQTTclient.setServer(mqtt_server, 1883);
  MQTTclient.setCallback(ReceivedMessage);
  if (MQTTclient.connect(clientID)) {
    MQTTclient.subscribe(mqtt_topic);
```

```
    greenLight();
    delay(500);
    allLightsOff();
  }
  else {
    redLight();
  }
  Serial.print("MQTT State: ");
  Serial.print(MQTTclient.state());
}

void loop() {

  //Sets up a client object for connections to the webpage. Constantly checks if a request is made
  WiFiClient wifiClient = server.available();
  if (wifiClient) {
    digitalWrite(LED_BUILTIN, LOW);
    delay(10);
    digitalWrite(LED_BUILTIN, HIGH);
    while (!wifiClient.available()) {
      delay(10);
    }
  }
  wifiClient.setTimeout(100);

  // Tells the MQTT client code to do what it needs to do
  MQTTclient.loop();
```

```
  //This section reads for inbound requests made by the web client. If any one of the endpoints is hit the corresponding code will trigger.
  String req = wifiClient.readStringUntil('\r');
  if (req.indexOf("/off") != -1) {
    cycle = false;
    tooClose = false;
    allLightsOff();
  }
  if (req.indexOf("/green") != -1) {
    cycle = false;
    tooClose = false;
    greenLight();
  }
  if (req.indexOf("/yellow") != -1) {
    cycle = false;
    tooClose = false;
    yellowLight();
  }
  if (req.indexOf("/red") != -1) {
    cycle = false;
    tooClose = false;
    redLight();
  }
  if (req.indexOf("/cycle") != -1) {
    cycle = true;
    tooClose = false;
    startMillis = millis();
```

```
}
if (req.indexOf("/tooClose") != -1) {
  cycle = false;
  tooClose = true;
}

if (tooClose == true){
  yellowLight();
  delay(150);
  redLight();
  delay(150);
}

//Checks if the program is in cycle mode and plays the appropriate light pattern.
if (cycle == true){
  if ((millis() - startMillis) < 5000 ){
    greenLight();
  }
  if ((millis() - startMillis) > 5000 && (millis() - startMillis) < 8000 ){
    yellowLight();
  }
  if ((millis() - startMillis) > 8000 && (millis() - startMillis) < 13000 ){
    redLight();
  }
  if ((millis() - startMillis) > 13000){
    startMillis = millis();
  }
```

```
  }


  //These client.print commands print out the html for the webpage and ultimately generates the page. This allows for the page to be served and interacted with in real time.
  wifiClient.println("<!doctype html>");
  wifiClient.println("<html>");
  wifiClient.println("<head>");
  wifiClient.println("<link rel='stylesheet' href='https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css' integrity='sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh' crossorigin='anonymous'>");
  wifiClient.println("<title>Aurdunio Stoplight</title>");
  wifiClient.println("</head>");


  wifiClient.println("<center>");
  wifiClient.println("<body>");
  wifiClient.println("<header>");
  wifiClient.println("<h1>Arduino Stoplight Lab 2</h1>");
  wifiClient.println("</header>");


  wifiClient.print("<h2>Red Light</h2>");
  wifiClient.print("<a href='http://");
  wifiClient.print(WiFi.localIP());
  wifiClient.print("/red' class='btn btn-block btn-lg btn-primary' role='button'><br>Turn on<br><br></a><br>");


  wifiClient.print("<h2>Yellow Light</h2>");
  wifiClient.print("<a href='http://");
  wifiClient.print(WiFi.localIP());
```

```
  wifiClient.print("/yellow' class='btn btn-block btn-lg btn-primary' role='button'><br>Turn
on<br><br></a><br>");


  wifiClient.print("<h2>Green Light</h2>");
  wifiClient.print("<a href='http://");
  wifiClient.print(WiFi.localIP());
  wifiClient.print("/green' class='btn btn-block btn-lg btn-primary' role='button'><br>Turn
on<br><br></a><br>");


  wifiClient.print("<h2>All Lights Off</h2>");
  wifiClient.print("<a href='http://");
  wifiClient.print(WiFi.localIP());
  wifiClient.print("/off' class='btn btn-block btn-lg btn-primary' role='button'><br>Turn
on<br><br></a><br>");


  wifiClient.print("<h2>Cycle Light</h2>");
  wifiClient.print("<a href='http://");
  wifiClient.print(WiFi.localIP());
  wifiClient.print("/cycle' class='btn btn-block btn-lg btn-primary' role='button'><br>Turn
on<br><br></a><br>");




  wifiClient.println("<span>Created by Blake Porter</span>");
  wifiClient.println("</body>");
  wifiClient.println("</center>");
  wifiClient.println("</html>");
}
```

```
//These functions directly contol the lights so that the loop can call these at anytime to change the light that is on.
void allLightsOff() {
  digitalWrite(D1, LOW);
  digitalWrite(D2, LOW);
  digitalWrite(D3, LOW);
}

void greenLight() {
  digitalWrite(D1, HIGH);
  digitalWrite(D2, LOW);
  digitalWrite(D3, LOW);
}

void yellowLight() {
  digitalWrite(D1, LOW);
  digitalWrite(D2, HIGH);
  digitalWrite(D3, LOW);
}

void redLight() {
  digitalWrite(D1, LOW);
  digitalWrite(D2, LOW);
  digitalWrite(D3, HIGH);
}

//MQTT message handler. Used by the callback function to receive messages from MQTT
void ReceivedMessage(char* topic, byte* payload, unsigned int payloadLength) {
```

```
  // Convert the char* topic to a String
  String topicFull = (String)topic;
  String message = "";
  // // Convert the payload to a usable String by iterating through each character to create the full message string
  for (int i = 0; i < payloadLength; i++) {
    message += (char)payload[i];
  }

  //Checks if the topic received matches the one originally subscribed to.
  //Then it parses the inputs of the message and triggers the events required that match the command.
  if (topicFull == (String)mqtt_topic) {
    if (message == "green") {
      Serial.println("green");
      cycle = false;
      tooClose = false;
      greenLight();
    } else if (message == "yellow") {
      Serial.println("yellow");
      cycle = false;
      tooClose = false;
      yellowLight();
    } else if (message == "red") {
      Serial.println("red");
      cycle = false;
      tooClose = false;
```

```
      redLight();
    } else if (message == "cycle") {
      Serial.println("cycle");
      cycle = true;
      tooClose = false;
      startMillis = millis();
    }
    else if (message == "tooClose") {
      Serial.println("tooClose");
      cycle = false;
      tooClose = true;
    }
    else if (message == "allOff" || message == "closed") {
      delay(200);
      Serial.println("allOff");
      cycle = false;
      tooClose = false;
      allLightsOff();
    }
  }
}
```

Ultrasonic Sensor Arduino code:

```
// This is code for a Ultrasonic sensor Module

#include <ESP8266WiFi.h>
#include <PubSubClient.h> // Allows us to connect to, and publish to the MQTT broker
#include <HCSR04.h> // include the Ultrasonic sensor library "HCSR04 ultrasonic sensor by
gamegine v2.0.3"
#include <MedianFilterLib.h> // Midian Filter Lib by Luis Llamas

//Sets some global variables
const char* ssid = "Blake-Hotspot";
const char* password = "blakepublic";
WiFiServer server(80);
IPAddress local_IP(192, 168, 157, 185);
IPAddress dns(192, 168, 91, 254);
IPAddress gateway(192, 168, 91, 254);
IPAddress subnet(255, 255, 255, 0);
long rawUltraDistance;
int change = 0;
bool state = false;

HCSR04 ultraSonic(D1, D2);  //Class for HCSR04 sets (trig pin , echo pin)
MedianFilter<int> medianFilter(5); // Sets up Median Filter Object

// MQTT configs
const char* mqtt_server = "192.168.157.117";
const char* mqtt_topic = "garage_system";
const char* clientID = "ultrasonic1";

// Sets up the Wifi and MQTT client
WiFiClient wifiClient;
PubSubClient MQTTclient(wifiClient);

//Setup for pins, clears all of them, starts MQTT connection.
void setup() {

  //Setup serial and all LED outputs then clears the LEDs off
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
```

```
  //Sets the static IP to the set variables
  if (!WiFi.config(local_IP, dns, gateway, subnet)) {
    Serial.println("STA Failed to configure");
  }

  //Using the Wifi library, begin connecting to the set network, and provide status updates.
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
    Serial.print(".");
    digitalWrite(LED_BUILTIN, LOW);
    delay(100);
  }
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);
  Serial.println();
  Serial.println("(( WiFi connected ))");
  Serial.println(WiFi.localIP());
  Serial.println(WiFi.gatewayIP());
  Serial.println(WiFi.dnsIP());

  // Connect to MQTT Broker
  MQTTclient.setServer(mqtt_server, 1883);
  MQTTclient.setCallback(ReceivedMessage);
  if (MQTTclient.connect(clientID)) {
    MQTTclient.subscribe(mqtt_topic);
    Serial.println("Connected to MQTT Broker!");
    delay(500);
  }
  else {
    Serial.println("Connection to MQTT Broker failed...");
    digitalWrite(LED_BUILTIN, LOW);
  }
  Serial.print("MQTT State: ");
  Serial.print(MQTTclient.state());
}
```

```
void loop() {

  // Tells the MQTT client code to do what it needs to do
  MQTTclient.loop();

  // Each time the loop repeats ultraDistance will be set to the distance received from the
sensor
  rawUltraDistance = ultraSonic.dist();
  delay(60);

  //Filters the raw distance data to have a smoother data stream
  int filteredUltraDistance = medianFilter.AddValue(rawUltraDistance);

  //If the distance is farther than 55cm and the Garage is open
  if (filteredUltraDistance > 55 && state == true) {
    //If the distance is still within the range then nothing will happen so that http
requests are not sent unnecessarily.
    if(change == 1){
    }
    else {
      Serial.print(filteredUltraDistance);
      Serial.println(" cm");
      MQTTclient.publish(mqtt_topic, "green");
      change = 1;
    }
  }
  //If the distance is between 50cm and 22cm and the Garage is open
  else if (filteredUltraDistance <= 50 && filteredUltraDistance >= 22 && state == true) {
    //If the distance is still within the range then nothing will happen so that http
requests are not sent unnecessarily.
    if(change == 2){
    }
    else {
      Serial.print(filteredUltraDistance);
      Serial.println(" cm");
      MQTTclient.publish(mqtt_topic, "yellow");
      change = 2;
    }
  }
  //If the distance is between 20cm and 12cm and the Garage is open
```

```
  else if (filteredUltraDistance <= 20 && filteredUltraDistance >= 12 && state == true) {
    //If the distance is still within the range then nothing will happen so that http
requests are not sent unnecessarily.
    if(change == 3){
    }
    else {
      Serial.print(filteredUltraDistance);
      Serial.println(" cm");
      MQTTclient.publish(mqtt_topic, "red");
      change = 3;
    }
  }
  //If the distance is closer than 10 cm and the Garage is open
  else if (filteredUltraDistance <= 10 && state == true) {
    //If the distance is still within the range then nothing will happen so that http
requests are not sent unnecessarily.
    if(change == 4){
    }
    else {
      change = 4;
      MQTTclient.publish(mqtt_topic, "tooClose");
    }
  }

}


//MQTT message handler. Used by the callback function to receive messages from MQTT
void ReceivedMessage(char* topic, byte* payload, unsigned int payloadLength) {

  // Convert the char* topic to a String
  String topicFull = (String)topic;
  String message = "";
  // Convert the payload to a usable String by iterating through each character to create the
full message string
  for (int i = 0; i < payloadLength; i++) {
    message += (char)payload[i];
  }
  //Serial.println(message);
```

```
  //Checks if the topic received matches the one originally subscribed to.
  //Then it parses the inputs of the message and triggers the events required that match the
command.
  if (topicFull == (String)mqtt_topic) {
    // If the garage door sensor is open activate the ultrasonic sensor
    if (message == "open") {
      Serial.println("open");
      state = true;
    // If the garage door sensor is closed deactivate the ultrasnoinc sensor and prevent it
from sending commands
    } else if (message == "closed"){
      Serial.println("closed");
      state = false;
    }
  }
}
```

Reed Switch Arduino code:

```
// This is code for a Reed sensor Module

#include <ESP8266WiFi.h>
#include <PubSubClient.h> // Allows us to connect to, and publish to the MQTT broker

//Sets some global variables
const char* ssid = "Blake-Hotspot";
const char* password = "blakepublic";
WiFiServer server(80);
IPAddress local_IP(192, 168, 157, 186);
IPAddress dns(192, 168, 91, 254);
IPAddress gateway(192, 168, 91, 254);
IPAddress subnet(255, 255, 255, 0);
int change = 0;

// MQTT configs
const char* mqtt_server = "192.168.157.117";
const char* mqtt_topic = "garage_system";
const char* clientID = "reedSensor";

// Sets up the Wifi and MQTT client
WiFiClient wifiClient;
PubSubClient MQTTclient(wifiClient);


//Setup for pins, starts wifi, and makes MQTT connection.
void setup() {

  //Setup serial and all LED outputs then clears the LEDs
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(D1, INPUT_PULLUP);

  //Sets the static IP to the set variables
  if (!WiFi.config(local_IP, dns, gateway, subnet)) {
    Serial.println("STA Failed to configure");
  }

  //Using the Wifi library, begin connecting to the set network, and provide status updates.
```

```
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
    Serial.print(".");
    digitalWrite(LED_BUILTIN, LOW);
    delay(100);
  }
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);
  Serial.println();
  Serial.println("(( WiFi connected ))");
  Serial.println(WiFi.localIP());
  Serial.println(WiFi.gatewayIP());
  Serial.println(WiFi.dnsIP());

  // Connect to MQTT Broker
  MQTTclient.setServer(mqtt_server, 1883);
  if (MQTTclient.connect(clientID)) {
    MQTTclient.subscribe(mqtt_topic);
    Serial.println("Connected to MQTT Broker!");
    delay(500);
  }
  else {
    Serial.println("Connection to MQTT Broker failed...");
    digitalWrite(LED_BUILTIN, LOW);
  }
  Serial.print("MQTT State: ");
  Serial.print(MQTTclient.state());
}

void loop() {

  // Sets up a client object that sets this device as a client. Let's MQTT do it's loop.
  MQTTclient.loop();

  // Prints the value of the D1 pin for debugging
  Serial.println(digitalRead(D1));
```

```
  // Reads the Reed sensor value and sends an open or closed command.
  // The garage door is open when the circuit open and closed when the circuit is closed.
  if (digitalRead(D1) == 0) {
    if (change == 1) {

    } else {
      // This repetition ensures that all devices receive the command.
      MQTTclient.publish(mqtt_topic, "closed");
      delay(100);
      MQTTclient.publish(mqtt_topic, "closed");
      delay(100);
      MQTTclient.publish(mqtt_topic, "closed");
      delay(100);
      digitalWrite(LED_BUILTIN, HIGH);
      change = 1;
    }
  }
  else {
    if (change == 2) {

    } else {
      // This repetition ensures that all devices receive the command.
      MQTTclient.publish(mqtt_topic, "open");
      delay(100);
      MQTTclient.publish(mqtt_topic, "open");
      delay(100);
      MQTTclient.publish(mqtt_topic, "open");
      delay(100);
      digitalWrite(LED_BUILTIN, LOW);
      change = 2;
    }
  }

}
```