Lab 4 – MQTT Event Hub – Garage Door Sensor

Online Links

This lab is available on my personal website at: http://AaronNelson95.com/IT441Lab4.php
The code is also available on my GitHub repository at https://github.com/AaronNelson95/IT441

Objective

The purpose of this lab is to learn how to implement an event hub to communicate notifications between devices. This lab incorporates a MQTT event hub known as Mosquitto. By managing messages sent from Arduino boards, other boards will interpret those messages and behave depending on the information that was received. This lab also incorporates a magnetic reed switch, which will behave like a door, and will detect if a magnet is close to the base. This lab will help one learn:

- How state diagrams and flow charts can help in developing and understanding a project
- How to incorporate three devices to communicate together
- How to run a MQTT event hub to manage device notifications
- How to have a device publish and subscribe to a MQTT topic
- How to use a magnetic reed switch

Materials

To complete this lab, you will need:

- Three Wemos D1 mini microcontrollers
- Three Micro USB power cords
- Three breadboards
- A traffic light LED display module
- An Ultrasonic Sensor (HC-SR04)
- A Magnetic Reed Switch (such as the Aleph DC-1561)
- 8 Male-Male Jumper Wires
- A computer with the Arduino IDE and a MQTT Broker

The Aleph DC-1561 Magnetic Reed Switch

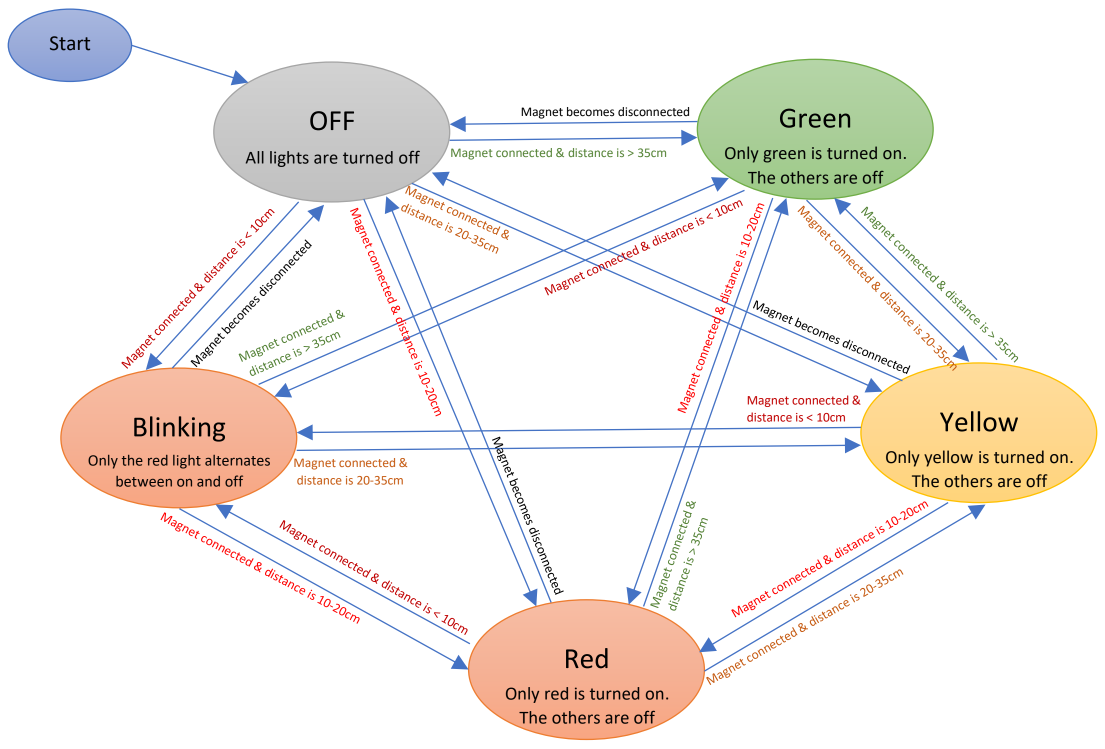
References

The following links may be helpful throughout this lab:

- https://www.arduino.cc/en/main/software - Installing the Arduino IDE (I did it from the Windows 10 app store)
- https://www.teachmemicro.com/getting-started-wemos-d1-mini/ - The exact steps to take to initially setup Arduino IDE to work with the Wemos D1 mini, including the link where the esp8266 boards are located at for the boards manager.
- https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/ - An excellent guide all about the ultrasonic sensor, how it works, and how to implement it in code. My sensor code comes from here.
- https://mosquitto.org/ - Information about the Mosquitto MQTT broker and how to use it.
- https://arduinodiy.wordpress.com/2017/11/24/mqtt-for-beginners/ - A great guide teaching about the basics of MQTT and how to incorporate this with Arduino. I based my MQTT code from this example.
- https://arduino.stackexchange.com/questions/13064/digital-pin-input-stays-high-after-receiving-a-high-signal - Information about the INPUT_PULLUP issue I encountered, and why the pin behaves that way.
- The PubSubClient library by Nick O'Leary (found in Arduino Tools -> Manage Libraries) - Used for managing MQTT connections with Arduino boards.

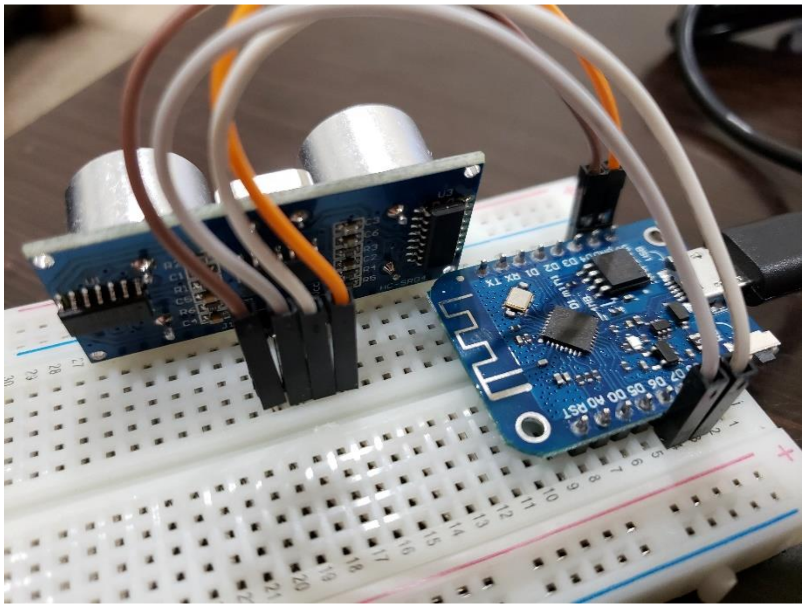
Procedures

1. Install the Arduino IDE either from the Windows Store or from the Arduino website.
2. Setup the IDE to work with the Wemos D1 mini model
    a. Click **File** -> **Preferences**. Under **Additional Boards Manager**, input https://arduino.esp8266.com/stable/package_esp8266com_index.json. Click OK to exit out.
    b. Click **Tools** -> **Board:** -> **Boards Manager**. Here, search for ESP8266 and one result should appear. Click the **Install** button.
    c. Now select this board to work with. Go to **Tools** -> **Board:** -> **"LOLIN(WEMOS) D1 R2 & mini"**.
    d. Plug one of your Wemos boards into the computer. Windows should automatically install necessary drivers for the microcontroller.
    e. Select the port the board is plugged into by going to **Tools** -> **Port:**. Next select the COM port your chip is in. If multiple ones are recognized, consult the Windows Device Manager to find the correct port.
3. Design documents to help you understand the logical flow of the system.
    a. Develop a state diagram.
        i. There should be 5 light states: when the green light is on, when the yellow light is on, when the red light is on, when the red light is blinking, and finally when the lights are all off. The states are shown through the stoplight device. The transitions between the states are determined by the echo sensor (to determine the color it should be) or the magnetic garage sensor switch (to determine if the lights should all be off).
        ii. Initially, the stoplight is in the off state. It listens to the MQTT topic that is published from the echo sensor. When the sensor detects a distance between 35-200cm (values higher are assumed to be errors and are discarded), it goes to green. 20-35cm goes to yellow. 10-20cm goes to red. 0-10cm blinks the red light. In addition, this sensor is only active when the magnetic reed switch has their magnets close together. If the magnets are apart, then the echo sensor tells the stoplight to turn all lights off.
        iii. Because we program the MQTT topic to only update when a color actually changes, a state will never have the chance to try to return to itself.

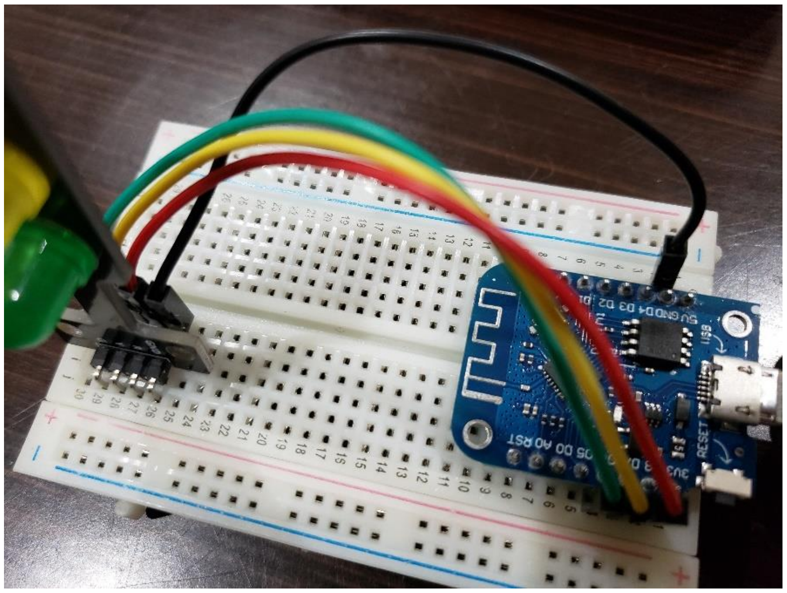

Start
OFF
All lights are turned off
Magnet becomes disconnected
Magnet connected & distance is > 35cm
Green
Only green is turned on.
The others are off
Magnet connected & distance is 20-35cm
Magnet connected & distance is < 10cm
Magnet becomes disconnected
Magnet connected & distance is > 35cm
Magnet connected & distance is 10-20cm
Magnet connected & distance is 10-20cm
Magnet connected & distance is 20-35cm
Magnet connected & distance is > 35cm
Magnet becomes disconnected
Magnet connected & distance is 10-20cm
Magnet connected & distance is < 10cm
Magnet connected & distance is > 35cm
Blinking
Only the red light alternates between on and off
Magnet connected & distance is < 10cm
Magnet connected & distance is 20-35cm
Yellow
Only yellow is turned on.
The others are off
Magnet becomes disconnected
Magnet connected & distance is > 35cm
Magnet connected & distance is 10-20cm
Magnet connected & distance is < 10cm
Magnet connected & distance is 10-20cm
Magnet connected & distance is 20-35cm
Red
Only red is turned on.
The others are off

b. Develop a system flowchart
   i. This should show an idea of how to program the microcontrollers. Arduino has two main functions: the startup() function, which runs once as soon as the board gains power, and the loop() function, which continually runs after the startup is finished. Try to consider how you can implement the state diagram using these two functions. With a MQTT broker, you must also consider that your boards can *publish* information to the server, and they can subscribe (using a callback function) to *listen* to a server topic. Chart out what information each device should be sending to and receiving from the other devices, and how they will respond to notifications.
   ii. An example of a finished system flowchart for this lab is available in the appendix.

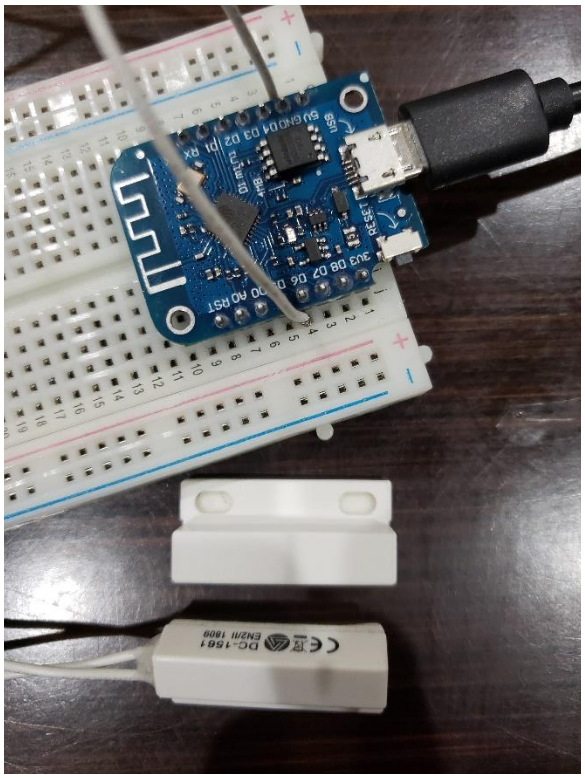
4. Connect one Wemos board to the ultrasonic sensor using a breadboard and jumper wires. In this project, the Vcc on the sensor will be connected to 5V on the board, Trig on the sensor is pin 13 (D7) on the board, Echo is connected to pin 12 (D6), and the Gnd is connected to the board's GND. A circuit diagram of this setup is available in the appendix.



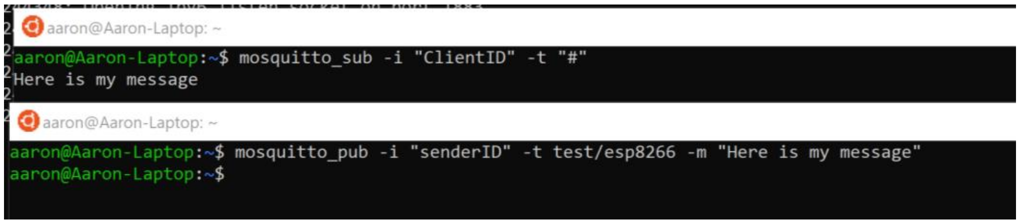
5. Connect another Wemos board to the stoplight using a breadboard and jumper wires. In this project, the green light will be connected to pin 12 (D6), the yellow light is on pin 13 (D7), the red light is on pin 15 (D8), and the stoplights ground is on the board's GND. A circuit diagram of this setup is also available in the appendix.



6. Connect a third board to the magnetic reed switch. One wire will be connected to pin 12 (D6) and the other will be connected to GND. It does not matter which wire is connected to which pin. This circuit diagram is also shown in the appendix.



7. Setup the Mosquitto MQTT Broker
   a. On the computer you wish to install the broker on (this can be a Raspberry Pi, Linux machine, or the Windows Subsystem for Linux), run the terminal command "$ sudo apt-get install mosquitto". Then run "$ sudo apt-get install mosquitto-clients" to be able to test that your MQTT server is working. This machine will need to be on for your Arduino boards to communicate through MQTT. The IP address of this machine (that your boards can see) will be used when setting up the Arduino boards.
   b. In the terminal, run the command "$ mosquitto -v" to start up the server. Your devices can now connect to it.
   c. To test that Mosquitto is working, open up another terminal and run the command "$ mosquitto_sub -i "<ClientID>" -t "#"", where # can be replaced with the topic you wish to listen to. Now, in another terminal, you can post to the server with "$ mosquitto_pub -i "<ClientID>" -t my/topic -m "my message"". The -i flag establishes the client's ID (each should be unique. -t sets the topic to post or listen to ('#' acts as a wildcard). -m sets your message.



```
aaron@Aaron-Laptop: ~
aaron@Aaron-Laptop:~$ mosquitto_sub -i "ClientID" -t "#"
Here is my message
```

```
aaron@Aaron-Laptop: ~
aaron@Aaron-Laptop:~$ mosquitto_pub -i "senderID" -t test/esp8266 -m "Here is my message"
aaron@Aaron-Laptop:~$
```

   d. Now that you understand how MQTT works, plan out how you would like to incorporate topics with this lab. For my project, I use the following topics and messages to communicate between my devices:

Topic	Message	Description	Published By	Subscribed By
device/doorSwitch	Activated	Device came online	Door Opener	
device/echoSensor	Activated	Device came Online	Echo Sensor	
device/stoplight	Activated	Device came online	Stoplight	
garage/doorSwitch	0	Magnets are disconnected	Door Opener	Echo Sensor
	1	Magnets are connected		
Garage/echoSwitch/color	g	Distance read is > 35cm	Echo Sensor	Stoplight
	y	Distance read is 20-35cm		
	r	Distance read is 10-20cm		
	b	Distance read is 0-10cm		
	o	Received “0” from garage/doorSwitch		

8. Implement the system flowchart and state diagram designed earlier. You can begin with trying to connect any Wemos board to the MQTT broker. The “PubSubClient.h” library can help with managing the necessary communications. In the Arduino IDE, select **Tools** -> **Manage Libraries**, and search for the “PubSubClient” by “Nick O’Leary”. Install this library and import it into your code. After establishing the connection to the server, see if you can post a message using “*client.publish(“my/topic”, “message”);*” This should appear in the terminal window you opened earlier. Also, try to implement the callback function (examples of this are available online). In the setup, you can choose which topics to listen to with “*client.subscribe(“my/topic”);*”. The callback function will listen to any changes in the topic and you can display the output to the serial monitor for testing.
    a. For more information on how to do this, you can refer to the tutorial at https://arduinodiy.wordpress.com/2017/11/24/mqtt-for-beginners/.
    b. You can save the sketch to the board by selecting **Sketch** -> **Upload**. The code will be compiled and recorded to your device. Be sure you specify the right COM port the board is connected to, as it may become confusing when working with three different devices.
9. Once you have become comfortable with connecting to your MQTT server with Arduino, you can begin working with your sensors. One sensor is the magnetic reed switch. Try to program your board so when the switch becomes connected or disconnected, it sends a value to the MQTT server. This will not need a callback or subscribe function, as it will only be used to post data. Make sure you set the switch’s pin to “INPUT_PULLUP” in the setup function to work properly (see Thought Questions #3).
10. Next, work with your echo sensor. It may help to take this a step at a time and start off by working with the Trigger and Echo pins on the ultrasonic sensor. A helpful tutorial on how to do this is provided at https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/. Basically, with the trigger pin working as an output, a wave will be sent out. This wave will bounce off a surface and will be received by the echo pin (working as an input). The time it takes this wave to travel a round trip can help you find the distance value. Try to build a program that finds distances on a repeated basis with the sensor, and report their results to the serial monitor. Examine how accurate the values are and adjust your code as needed.
    a. You can view my lab 3 information (http://aaronnelson95.com/IT441Lab3.php) for help in setting up the echo sensor.
11. After your echo sensor is working and obtaining accurate data, implement the MQTT subscribe/callback function. You should only run the sensor when the garage door is connected (or disconnected). This can be done by setting a variable, such as “startSensor”, to a certain value and then in the loop function place the sensor code within an if statement (to check if this variable is set to that value). Also, you can implement the color ranges directly in this portion of the code. Depending on the color the range is within (for example, if it is between 20-35cm the light should be yellow), you should post that color in a MQTT topic (different from the one used by the magnetic garage sensor).

12. Finally, implement the stoplight device. If you posted colors (by calculating their ranges with the echo sensor) inside of its own topic, this will be easy to implement. You can simply subscribe to the MQTT topic and when a certain color value comes through, you can turn stoplight lights on or off.
13. When all three devices are complete, you should turn on your computer's Mosquitto server. When the devices are powered on, they should connect to this server. When the magnetic reed switch is put together, a message should appear in its topic. This will instruct the echo sensor to begin gathering distance values. Depending on how close something is to the sensor, the stoplight should change colors. When the magnetic switch becomes disconnected, the lights should all turn off. This should respond and behave the way the state diagram and flow chart were designed.

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?

   An event hub like MQTT allows multiple client devices communicate with one centralized server. These devices can publish and put information to this server, and they can subscribe and receive information when data changes. MQTT allows this to happen very quickly, and devices can easily perform both roles at the same time. For example, our echo sensor is able to listen to information passed from the door opener, and it is able to give information out to the stoplight. With the client/server system we were using before, this information transfer would be slower and the client would need to actively be checking a server that was constantly updated (unless a technology like web sockets can be implemented). Our client wouldn't know if anything was pushed until it checked, so often, it may obtain the same information as before without being aware of any change. This is like having phone notifications check periodically (such as every minute). MQTT in our case behaves more like push notifications, where once data needs to be sent to the client, the client responds and knows it is coming (with the callback function). In addition, this method makes it much easier to subscribe to multiple channels (vs. having to check multiple servers throughout the loop). With this, MQTT is much more scalable and multiple devices can be publishing data.

   To learn and understand how MQTT works, we had a small class demo. This helped me to understand a lot about how you can start Mosquitto, listen to a topic, and then push data to it. Before I started using MQTT with Arduino, I played around with it using only my Windows Linux terminals. I opened one terminal to start the server, one to listen in for data, and one to push out data. I experimented with different flags to see how changing the id and topic would affect the data going through. Once I felt that I got a good feel for how to publish a message, I only subscribed to the specific topic I was expecting to receive from, and I tested and checked that I was using topics correctly. Then, I began implementing this into Arduino, with first seeing if I can send a message to those topics, and finally if I could use the callback function to listen to a specific topic.

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?

   An event hub such as MQTT makes it very simple for multiple devices to communicate with each other. You are able to publish information to topics and also listen to only the topics you need information from. This method uses a single device to act as the MQTT server. If any of the clients go offline, the rest of the clients will still be able to communicate. However, if the server goes down, all devices will stop working, which can be very problematic. This method is also handy because the structure of topics will also devices to ignore all extra traffic. When a change is made to a topic it is subscribed to, it almost acts as if the change is *pushed* to the clients involved. This allows for very quick communication and responses. MQTT is also great if your overall internet connection is unreliable and if your data is sent at irregular intervals.

A disadvantage MQTT has is that it is a very light messaging protocol. It cannot support large, heavy amounts of data. Using the sensor/actuator method, you can transfer or broadcast lots of information, such as an entire webpage. Unfortunately, unless a more advanced technology is utilized, a sensor/actuator system would need to be checked at regular intervals to see if any information has changed. Another disadvantage of MQTT is that each connected client needs to have a unique ID. If a client with the same ID comes online, then the old one is kicked off. Nothing is subscribed to a sensor/actuator system so the names of connected devices do not matter as they are pulling information from the server (rather than the server pushing information to them). In this application, I think the MQTT system makes much more sense to use for this project. It is simple to setup and it performs very quickly because topics can be subscribed to and will receive data as it comes. We do not need to transfer large amounts of data between devices, in fact, I only send a single character as the topic message generally, and so we do not need the large data support the sensor/actuator system provides. In a simple application, where quick data needs to be transferred, an event hub is a great choice to use.

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

   The largest challenge I faced actually didn't have anything to do with the MQTT server. That was pretty straightforward, and I was able to figure out how to see and push information to it. Instead, the part that took me the longest with this lab was working with the magnetic reed switch (used as the garage door opener). First, I was a little confused by which pins this switch was supposed to be plugged into. What I heard from class wasn't correct and this specific model of switch we used isn't documented very well online. I tried looking up other reed switches, and I learned pretty quickly that this one doesn't work the same and it doesn't use any voltage pins. Ultimately, I discovered that just one wire went to ground and the other went to a digital pin. Now I was able to work with my switch, but then I noticed a new issue where once I disconnected my magnet, my switch wouldn't notice the change digitally and it would be unusable unless I reset my board. The fix for this was simple, though difficult to find. I just had to set my switch to "INPUT_PULLUP" and it worked correctly. Aside from these small issues, I didn't really have any other problems with the lab. It was just a matter of taking some time to learn the new MQTT system.

4. Estimate the time you spent on this lab and report.

   The lab portion took me around 6 hours. A good portion of this time was spent trying to understand how MQTT works and how to implement it in my project. I also spent quite a bit of time trying to solve the challenges I faced with the magnetic reed switch (mentioned in question 3). The report took me about 8 hours, for a total project time of about 14 hours.

Certification of Work

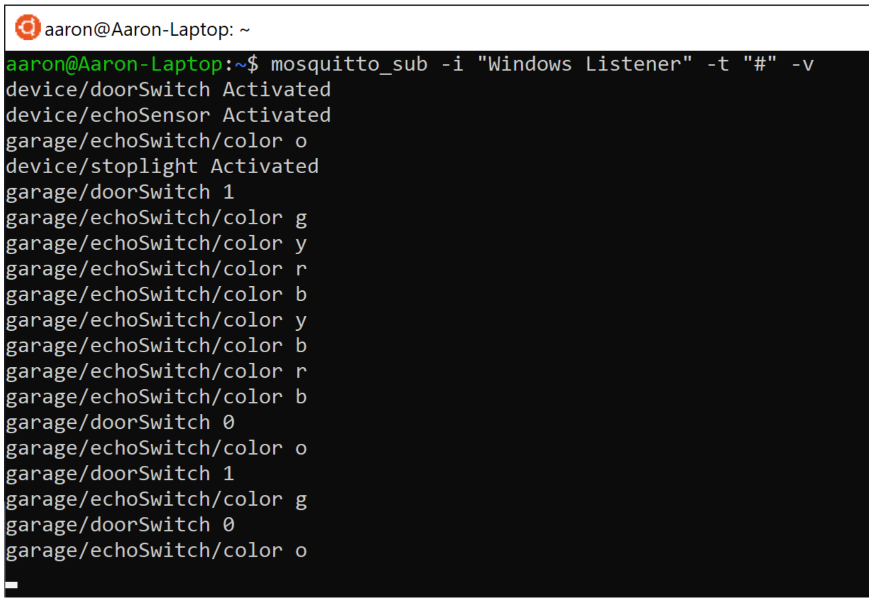
I certify that this lab represents my own work. I did use various resources on the internet to help get me started with parts of this project through, such as how to use Mosquitto, incorporating it with Arduino boards, and how to gain a distance reading from the ultrasonic sensor. However, I did not copy anything that was not basic knowledge and I referred to the work I received help from inside of my code comments.

– Aaron Nelson

Appendix

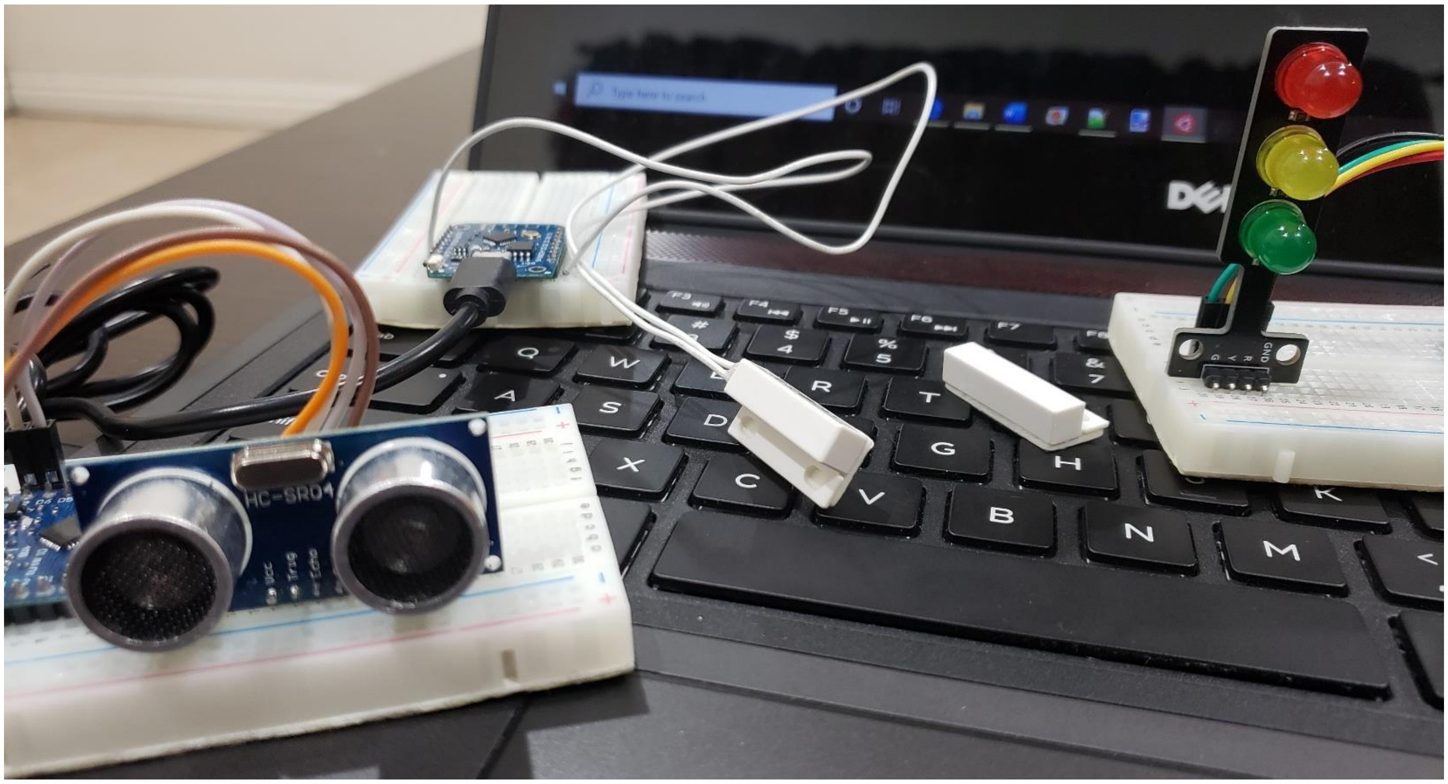
Images of Final Product

As can be seen below, when the magnet is not connected to the reed switch, the stoplight should be off and the ultrasonic sensor should not be gathering data. When the magnet is connected, the distance an object is away from the sensor will determine the color of the stoplight. As seen in the terminal, once the devices are connected, they send an "Activated" message to the server. A change in the magnet status immediately allows the echo sensor to begin working (as seen how once the garage/doorSwitch becomes 0, the garage/echoSwitch/color immediately becomes 'o' (for off)).

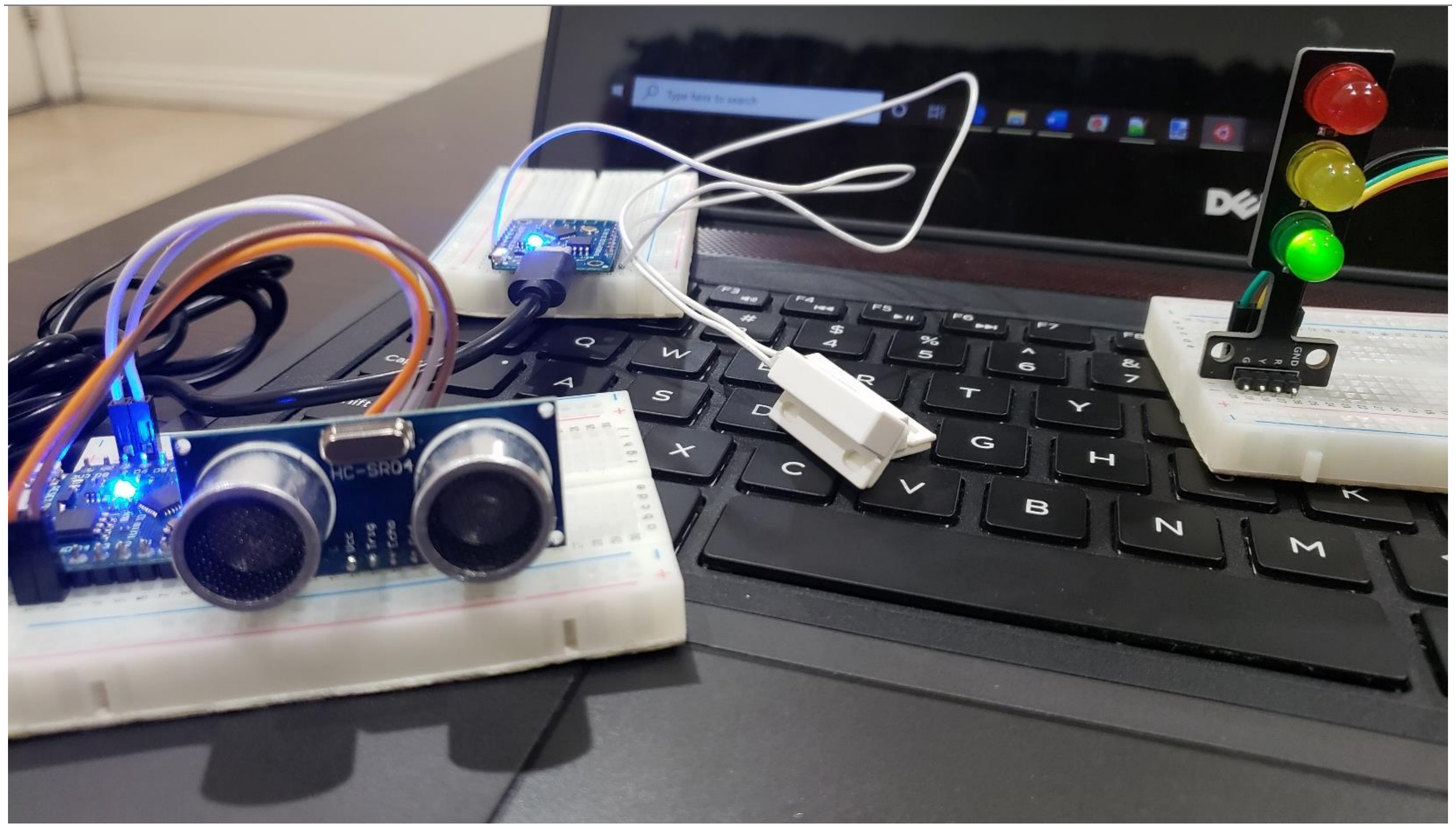


```
aaron@Aaron-Laptop:~$ mosquitto_sub -i "Windows Listener" -t "#" -v
device/doorSwitch Activated
device/echoSensor Activated
garage/echoSwitch/color o
device/stoplight Activated
garage/doorSwitch 1
garage/echoSwitch/color g
garage/echoSwitch/color y
garage/echoSwitch/color r
garage/echoSwitch/color b
garage/echoSwitch/color y
garage/echoSwitch/color b
garage/echoSwitch/color r
garage/echoSwitch/color b
garage/doorSwitch 0
garage/echoSwitch/color o
garage/doorSwitch 1
garage/echoSwitch/color g
garage/doorSwitch 0
garage/echoSwitch/color o
```

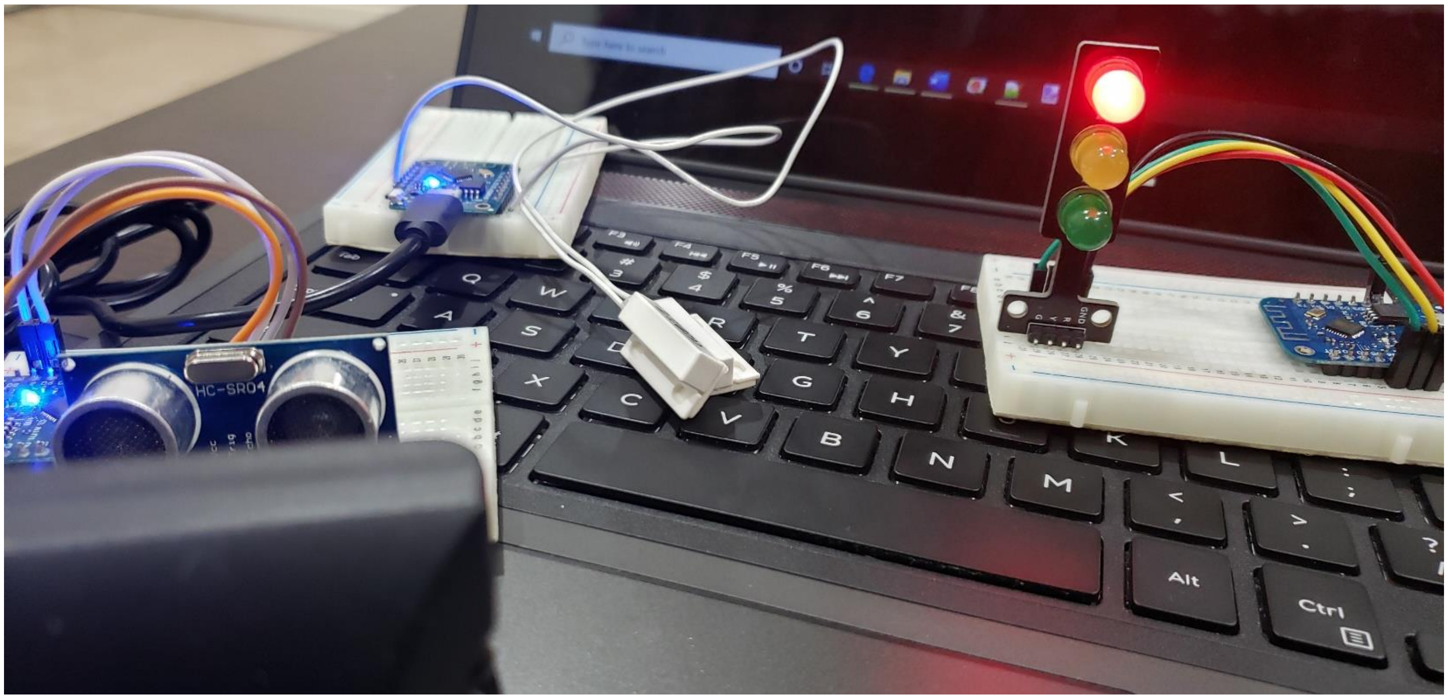
A terminal window listening to all information being exchanged on the server.



When the magnet is disconnected, the stoplight is turned off

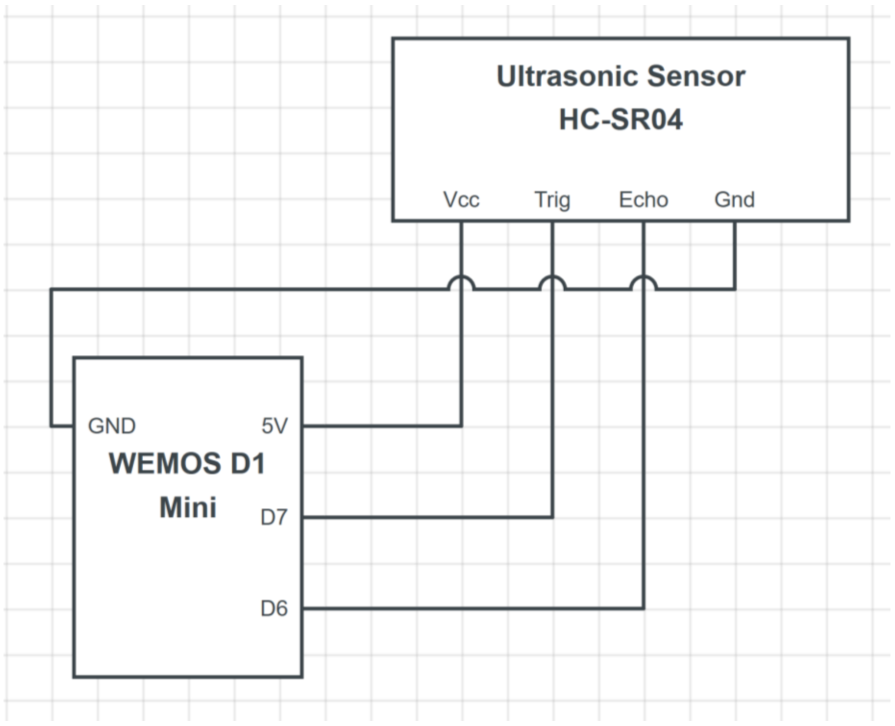


When the magnet is together, the sensor begins detecting distances. If an object is far away, the stoplight turns green

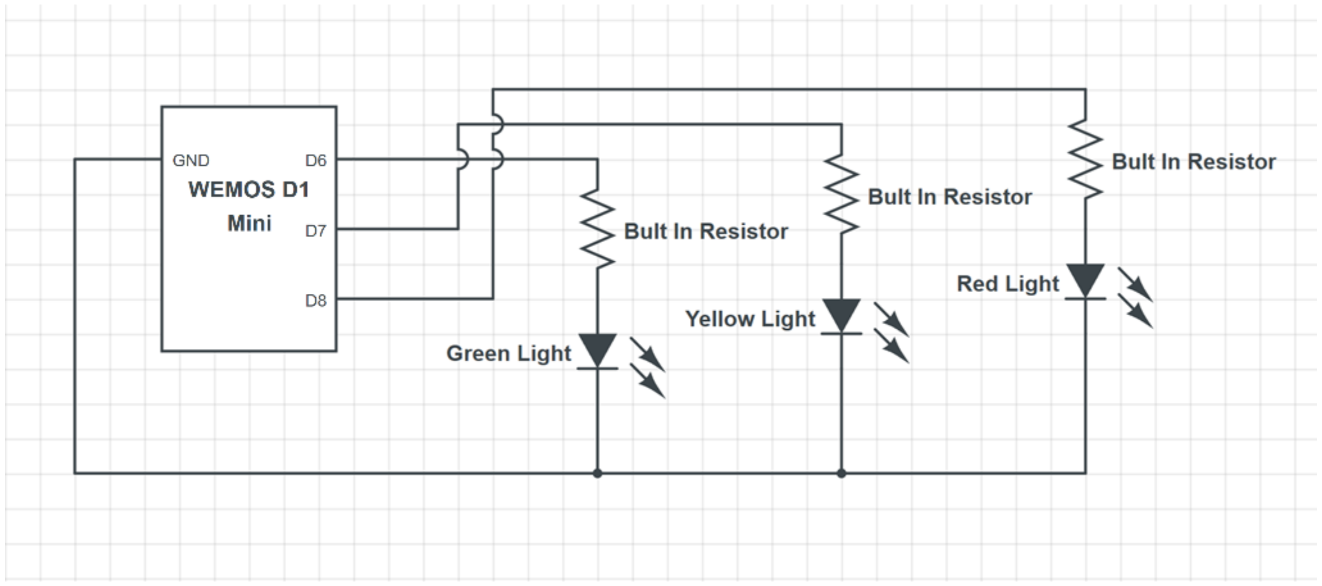


When the magnet is together and the sensor is detecting distances, if an object is close, then the stoplight will turn red

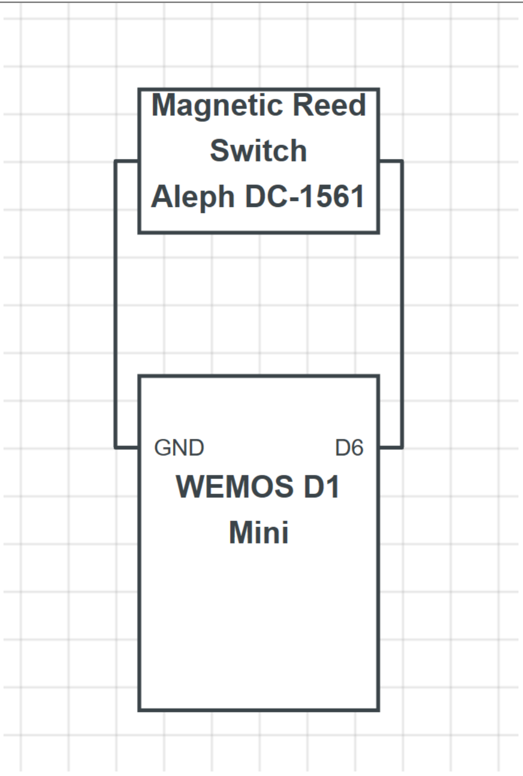
Circuit Diagrams



The Ultrasonic Sensor



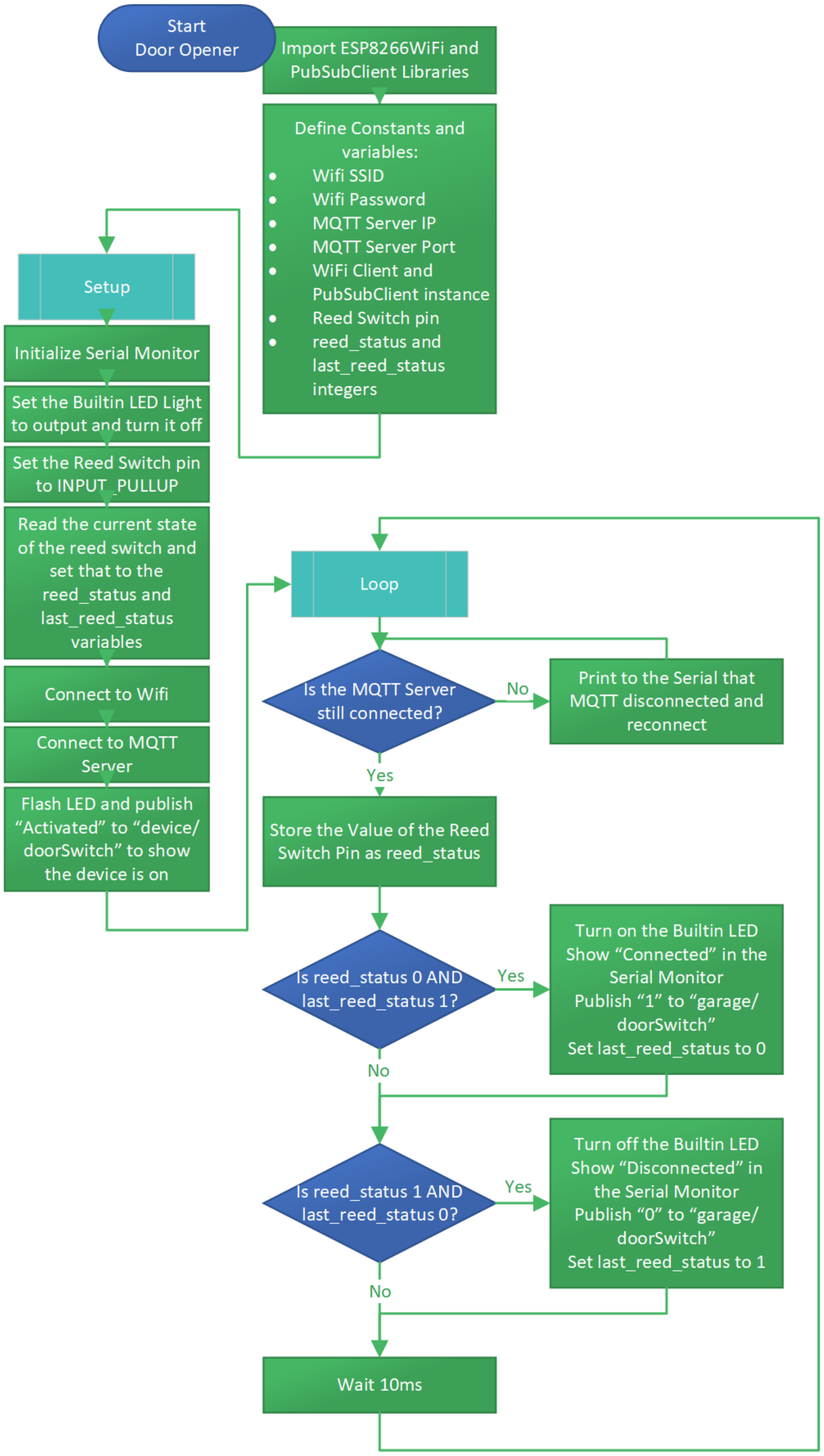
The Stoplight

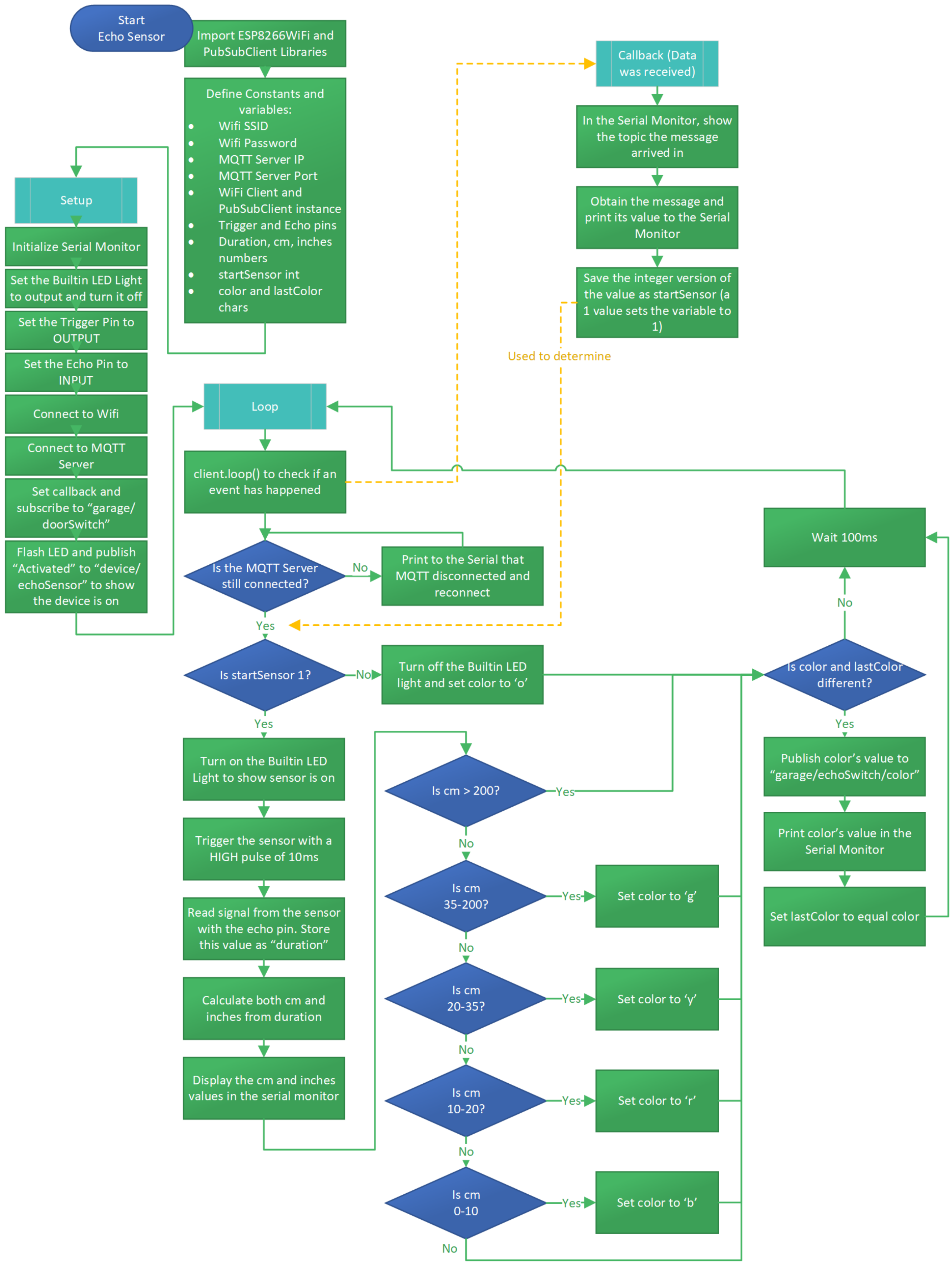


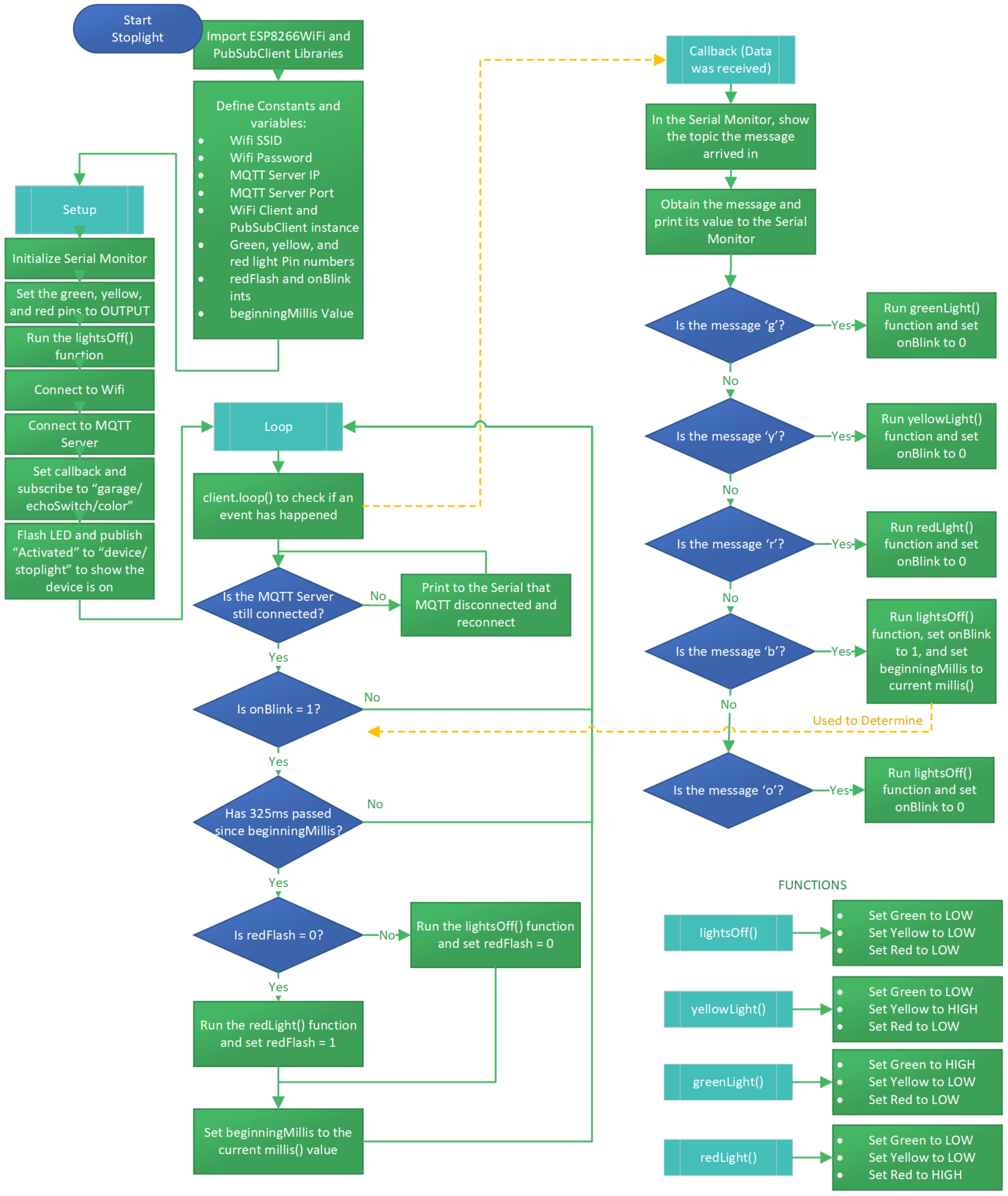
The Magnetic Reed Switch

System Flowchart

NOTE: A larger version of my flowcharts are available on my website.


Start
Door Opener
Import ESP8266WiFi and PubSubClient Libraries
Define Constants and variables:
• Wifi SSID
• Wifi Password
• MQTT Server IP
• MQTT Server Port
• WiFi Client and PubSubClient instance
• Reed Switch pin
• reed_status and last_reed_status integers
Setup
Initialize Serial Monitor
Set the Builtin LED Light to output and turn it off
Set the Reed Switch pin to INPUT_PULLUP
Read the current state of the reed switch and set that to the reed_status and last_reed_status variables
Connect to Wifi
Connect to MQTT Server
Flash LED and publish “Activated” to “device/doorSwitch” to show the device is on
Loop
Is the MQTT Server still connected?
No
Print to the Serial that MQTT disconnected and reconnect
Yes
Store the Value of the Reed Switch Pin as reed_status
Is reed_status 0 AND last_reed_status 1?
Yes
Turn on the Builtin LED
Show “Connected” in the Serial Monitor
Publish “1” to “garage/doorSwitch”
Set last_reed_status to 0
No
Is reed_status 1 AND last_reed_status 0?
Yes
Turn off the Builtin LED
Show “Disconnected” in the Serial Monitor
Publish “0” to “garage/doorSwitch”
Set last_reed_status to 1
No
Wait 10ms


Start Echo Sensor
Import ESP8266WiFi and PubSubClient Libraries
Define Constants and variables:
• Wifi SSID
• Wifi Password
• MQTT Server IP
• MQTT Server Port
• WiFi Client and PubSubClient instance
• Trigger and Echo pins
• Duration, cm, inches numbers
• startSensor int
• color and lastColor chars
Setup
Initialize Serial Monitor
Set the Builtin LED Light to output and turn it off
Set the Trigger Pin to OUTPUT
Set the Echo Pin to INPUT
Connect to Wifi
Connect to MQTT Server
Set callback and subscribe to “garage/doorSwitch”
Flash LED and publish “Activated” to “device/echoSensor” to show the device is on
Callback (Data was received)
In the Serial Monitor, show the topic the message arrived in
Obtain the message and print its value to the Serial Monitor
Save the integer version of the value as startSensor (a 1 value sets the variable to 1)
Used to determine
Loop
client.loop() to check if an event has happened
Is the MQTT Server still connected?
No
Print to the Serial that MQTT disconnected and reconnect
Yes
Is startSensor 1?
No
Turn off the Builtin LED light and set color to ‘o’
Yes
Turn on the Builtin LED Light to show sensor is on
Trigger the sensor with a HIGH pulse of 10ms
Read signal from the sensor with the echo pin. Store this value as “duration”
Calculate both cm and inches from duration
Display the cm and inches values in the serial monitor
Is cm > 200?
Yes
No
Is cm 35-200?
Yes
Set color to ‘g’
No
Is cm 20-35?
Yes
Set color to ‘y’
No
Is cm 10-20?
Yes
Set color to ‘r’
No
Is cm 0-10
Yes
Set color to ‘b’
No
Wait 100ms
No
Is color and lastColor different?
Yes
Publish color’s value to “garage/echoSwitch/color”
Print color’s value in the Serial Monitor
Set lastColor to equal color


Start Stoplight
Import ESP8266WiFi and PubSubClient Libraries
Define Constants and variables:
• Wifi SSID
• Wifi Password
• MQTT Server IP
• MQTT Server Port
• WiFi Client and PubSubClient instance
• Green, yellow, and red light Pin numbers
• redFlash and onBlink ints
• beginningMillis Value
Setup
Initialize Serial Monitor
Set the green, yellow, and red pins to OUTPUT
Run the lightsOff() function
Connect to Wifi
Connect to MQTT Server
Set callback and subscribe to “garage/echoSwitch/color”
Flash LED and publish “Activated” to “device/stoplight” to show the device is on
Loop
client.loop() to check if an event has happened
Is the MQTT Server still connected?
No
Print to the Serial that MQTT disconnected and reconnect
Yes
Is onBlink = 1?
No
Yes
Has 325ms passed since beginningMillis?
No
Yes
Is redFlash = 0?
No
Run the lightsOff() function and set redFlash = 0
Yes
Run the redLight() function and set redFlash = 1
Set beginningMillis to the current millis() value
Callback (Data was received)
In the Serial Monitor, show the topic the message arrived in
Obtain the message and print its value to the Serial Monitor
Is the message ‘g’?
Yes
Run greenLight() function and set onBlink to 0
No
Is the message ‘y’?
Yes
Run yellowLight() function and set onBlink to 0
No
Is the message ‘r’?
Yes
Run redLIght() function and set onBlink to 0
No
Is the message ‘b’?
Yes
Run lightsOff() function, set onBlink to 1, and set beginningMillis to current millis()
No
Used to Determine
Is the message ‘o’?
Yes
Run lightsOff() function and set onBlink to 0
FUNCTIONS
lightsOff()
• Set Green to LOW
• Set Yellow to LOW
• Set Red to LOW
yellowLight()
• Set Green to LOW
• Set Yellow to HIGH
• Set Red to LOW
greenLight()
• Set Green to HIGH
• Set Yellow to LOW
• Set Red to LOW
redLight()
• Set Green to LOW
• Set Yellow to LOW
• Set Red to HIGH

Code (available at https://github.com/AaronNelson95/IT441)

Arduino Code

Lab_4_Door_Opener.ino

```
/*
 * Created by Aaron Nelson
 * Lab 4 Event Bus
 * 10-26-19
 * This script is used with the reed switch. It will post to the MQTT server when the
     magnet becomes connected or disconnected.
 *
 * The MQTT portion of this script was obtained from
     https://arduinodiy.wordpress.com/2017/11/24/mqtt-for-beginners/
 *
    Reed Switch Pins:
        One Wire: Pin12 (D6)
        The Other Wire: GND
*/

#include <ESP8266WiFi.h>            // This contains the libraries that allows the board
                                          to connect to wifi
#include <PubSubClient.h>           // This is used for communication with the MQTT
                                          Server

const char* ssid = "YOUR WIFI HERE";       // Specify the name of your wifi
const char* password = "YOUR PASSWORD HERE";      // Specify the password for your wifi

const char* mqttServer = "192.168.137.1"; // The location of your MQTT Server
const int mqttPort = 1883;                // The port number your MQTT Server is running
                                                on (1883 is the default for Mosquitto)

WiFiClient doorSwitchClient;        // Name for our MQTT connection client
PubSubClient client(doorSwitchClient);    // Creates a partially initialized client
                                                instance

int reed_switch = 12;               // Pin D6 is connected to one of the switch wires.
                                          If using a different pin, specify that here
int reed_status;                    // When this is 0, it means it is connected to the
                                    magnet. A 1 means it is disconnected and separated
int last_reed_status;               // Keeps the last value of the reed magnet (to only
                                          publish to the sensor when a CHANGE is made)


void setup() {                      // This runs when the board is first turned on
  Serial.begin(115200);             // This allows serial information when connected to
                                          a computer (this will tell when the magnet is
                                          connected)
  pinMode(LED_BUILTIN, OUTPUT);     // Prepares the builtin light pin for output (which
                                          turns on when the reed magnet is connected)
  digitalWrite(LED_BUILTIN, HIGH);  // Initially, turn off the builtin LED light
  pinMode(reed_switch, INPUT_PULLUP);     // Initialize the reed switch pin for input.
                                                Pullup is necessary for when the magnet
                                                becomes reconnected
  reed_status = digitalRead(reed_switch); // Read the current value of the magnet switch
  last_reed_status = reed_status;   // Store the initial reading of the magnet switch to
                                          detect state changes

  /* Connect to WiFi */
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");    // This is shown on the serial if connected to a
                                        computer
  Serial.println(ssid);              // It displays the wifi it is trying to connect to

  WiFi.mode(WIFI_STA);               // It sets the wifis mode to "Station" (rather than
                                        "AP" which is the Access Point mode)
  WiFi.hostname("DoorSwitch");       // Hostname to uniquely identify our device
  WiFi.begin(ssid, password);        // Attempts to connect to wifi using provided
                                        credentials

  while (WiFi.status() != WL_CONNECTED) {      // While the wifi is not connected yet:
    delay(500);                      // every half second,
    Serial.print(".");               // it prints a dot on the Serial Monitor to show it
                                        is still trying to connect
  }
  Serial.println("");
  Serial.println("WiFi connected"); // When it does connect, show that there was success

  connectMQTT();                     // Runs the function to set up our MQTT connection

  // Blink LED quickly to show that it is connected to wifi and the MQTT server
  digitalWrite(LED_BUILTIN, LOW);    // light turns on
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);   // light turns off
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);    // light turns on
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);   // light turns off

  client.publish("device/doorSwitch", "Activated");  // Post in the device channel that
                                                         this device is on
}




void loop() {                        // This constantly cycles through, detecting a
                                        change in the reed switch and posting to the
                                        MQTT server if there is one
  if (!client.connected()){          // Occasionally the client drops connection randomly
                                        and stops working
    Serial.println("MQTT was Unexpectedly Disconnected");
    connectMQTT();                   // If the connection dropped, try to reconnect to
                                        our server
  }

  reed_status = digitalRead(reed_switch);      // On each loop through, check if the
                                                  magnet is connected or not

  // Respond to state changes
  if (reed_status == 0 && last_reed_status == 1) {     // If the magnet just barely
                                                           became connected
      digitalWrite(LED_BUILTIN, LOW);          // The LED light turns on
      Serial.println("connected");             // Displays it is connected in the serial
                                                  monitor
      client.publish("garage/doorSwitch", "1");        // Posts a value to the MQTT
                                                           server
      last_reed_status = 0;          // Remembers the last state was connected (to only
                                           work off state changes)
   }
   if (reed_status == 1 && last_reed_status == 0) {     // If the magnet just barely
                                                         became disconnected
       digitalWrite(LED_BUILTIN, HIGH);        // The LED light turns off
       Serial.println("disconnected");         // Displays it is disconnected in the
                                                       serial monitor
       client.publish("garage/doorSwitch", "0");       // Posts a value to the MQTT
                                                         server
       last_reed_status = 1;          // Remembers the last state was connected (to only
                                           work off state changes)
   }
   delay(10);    // Delay is used so that little random (dis)connections aren't detected
                   (like if the magnet is barely on the verge of connection)
}




void connectMQTT() {               // When first connecting to MQTT or when it becomes
                                           disconnected, run this script
  client.setServer(mqttServer, mqttPort);      // Setup the server connection with
                                                 information the user provided
  while(!client.connected()) {     // If the MQTT server is not connected or becomes
                                     disconnected, try again
    Serial.println("connecting to MQTT...");
    if (client.connect("DoorSwitch")) {         // Try to connect with this name
      Serial.println("Connected to MQTT");
    } else {
      Serial.print("failed... Re");            // Repeatedly says "failed...
                                           Reconnecting" as this loop cycles through
      // Serial.print(client.state());            // Show in the Serial Monitor what the
                                           current state of the server connection is
      delay(2000);               // Wait two seconds before trying to connect to the
                                           server again
    }
  }
}
```

Lab_4_Echo_Sensor.ino

```
/*
 * Created by Aaron Nelson
 * Lab 4 Event Bus
 * 10-26-19
 * This script is used for the ultrasonic sensor (HC-SR04). It first listens to the MQTT
      server (whether or not the reed switch is connected) and it posts sensor reading
      groupings to the server (to be red by the stoplight)
 *
 * The MQTT portion of this script was obtained from
      https://arduinodiy.wordpress.com/2017/11/24/mqtt-for-beginners/
 *
 * Original Script for the sensor functionality was created by Rui Santos, provided at:
 *      https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/
 *
```

```
    Ultrasonic Sensor Pins:
        VCC: +5VDC
        Trig: Trigger (INPUT) - Pin13 (D7)
        Echo: Echo (OUTPUT) - Pin12 (D6)
        GND: GND
 */

#include <ESP8266WiFi.h>            // This contains the libraries that allows the board
                                        to connect to wifi
#include <PubSubClient.h>           // This is used for communication with the MQTT
                                        Server

const char* ssid = "YOUR WIFI HERE";        // Specify the name of your wifi
const char* password = "YOUR PASSWORD HERE";      // Specify the password for your wifi

const char* mqttServer = "192.168.137.1"; // The location of your MQTT Server
const int mqttPort = 1883;                // The port number your MQTT Server is running
                                              on (1883 is the default for Mosquitto)

WiFiClient echoClient;              // Name for our MQTT connection client
PubSubClient client(echoClient);    // Creates a partially initialized client instance

int trigPin = 13;                   // The trigger is on pin D7. If using a different
                                        pin, specify that here
int echoPin = 12;                   // The echo is on pin D6. If using a different pin,
                                        specify that here
long duration, cm, inches;          // This code can read distances in both cm and
                                        inches. They will both be printed to the
                                        Serial Monitor
int startSensor = 0;                // When this is a 1, it means the door magnet is
                                        connected and that the sensor should obtain
                                        distance readings

char color = 'o';                   // Contains the current color the stoplight should
                                        turn (to be posted to the MQTT server)
char lastColor = 'g';               // Contains the last color the stoplight should be
                                        (to only update on changed states)

void setup() {                      // This runs when the board is first turned on
  Serial.begin (115200);            // This allows serial information when connected to
                                        a computer (in here, this shows a constant
                                        stream of sensor readings)

  pinMode(trigPin, OUTPUT);         // Trigger acts as an output (it sends off a wave)
  pinMode(echoPin, INPUT);          // Echo acts as in input (it listens for the wave to
                                        come back)
  pinMode(LED_BUILTIN, OUTPUT);     // The LED light will work as an output and show
                                        when the sensor is currently collecting data
  digitalWrite(LED_BUILTIN, HIGH);  // Initially turn the built in LED light off

  /* Connect to WiFi */
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");   // This is shown on the serial if connected to a
                                        computer
  Serial.print(ssid);               // It displays the wifi it is trying to connect to

  WiFi.mode(WIFI_STA);              // It sets the wifis mode to "Station" (rather than
                                        "AP" which is the Access Point mode)
  WiFi.hostname("EchoSensor");       // Hostname to uniquely identify our device
  WiFi.begin(ssid, password);        // Attempts to connect to wifi using provided
                                           credentials

  while (WiFi.status() != WL_CONNECTED) {      // While the wifi is not connected yet:
    delay(500);                      // every half second,
    Serial.print(".");               // it prints a dot on the Serial Monitor to show it
                                           is still trying to connect
  }
  Serial.println();
  Serial.println("WiFi connected"); // When it does connect, show that there was success

  connectMQTT();                     // Runs the function to set up our MQTT connection

  // Blink LED quickly to show that it is connected
  digitalWrite(LED_BUILTIN, LOW);    // light turns on
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);   // light turns off
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);    // light turns on
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);   // light turns off

  client.publish("device/echoSensor", "Activated");  // Post in the device channel that
                                                           this device is on

}




void loop() {                        // This constantly cycles through, performing MQTT
                                           tasks and if the door switch is connected,
                                           finding sensor distances and if a range change
                                           occurs, posting this to the server
  client.loop();                     // Runs through the MQTT functions to process
                                           incoming messages and send publish data

  if (!client.connected()){          // Occasionally the client drops connection randomly
                                           and stops working
    Serial.println("MQTT was Unexpectedly Disconnected");
    connectMQTT();                   // If the connection dropped, try to reconnect to
                                           our server
  }

  if (startSensor == 1){             // If the door sensor is connected, we can begin to
                                           collect echo sensor readings
    digitalWrite(LED_BUILTIN, LOW); // Turn on the LED light to show we are allowed to
                                           gather data

    // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
    // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
    digitalWrite(trigPin, LOW);
    delayMicroseconds(5);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    // Read the signal from the sensor: a HIGH pulse whose
    // duration is the time (in ms) from the sending
    // of the ping to the reception of its echo off of an object.
    pinMode(echoPin, INPUT);
    duration = pulseIn(echoPin, HIGH);

    // Convert the time into a distance (because the sound is bounced, the total
            duration is divided by two to gain the distance
    cm = (duration/2) / 29.1;     // Divide by 29.1 or multiply by 0.0343 to obtain cm
                                          from ms
    inches = (duration/2) / 74;   // Divide by 74 or multiply by 0.0135 to obtain in
                                          from ms

    // Show the reading of inches and centimeters in the Serial Monitor.
    Serial.print(inches);
    Serial.print("in, ");
    Serial.print(cm);
    Serial.print("cm");
    Serial.println();


    // Respond to the readings from the echo sensor. Change the color variable to
            represent the range of the distance
    if (cm > 200) {
      // This is likely an odd reading, do nothing... (if a room is bigger, expand this
            number)
    } else if (cm > 35){
      // Distance is over 35cm, a fair bit away. It is safe to keep driving
      color = 'g';
    } else if (cm > 20) {
      // Distance is between 20-35cm, we are getting closer so show a warning light
      color = 'y';
    } else if (cm > 10) {
      // Distance is between 10-20cm, the car should stop now
      color = 'r';
    } else if (cm > 0) {
      // Distance is less than 10ms, show that the driver is too close and there is no
            walkway and they should back up
      color = 'b';
    }
  } else {                        // The echo sensor is not authorized to work (the
                                          magnet switch is disconnected)
    digitalWrite(LED_BUILTIN, HIGH); // The LED light turns off
    color = 'o';                  // The stoplight state should be off because the door
                                          is closed
  }

  // If there is a state change (the color range now is different than the last), post
            the new value to the MQTT server
  if (!(color == lastColor)) {    // The color is different
    if (color == 'g') {
      client.publish("garage/echoSwitch/color", "g");
    } else if (color == 'y') {
      client.publish("garage/echoSwitch/color", "y");
    } else if (color == 'r') {
      client.publish("garage/echoSwitch/color", "r");
    } else if (color == 'b') {
      client.publish("garage/echoSwitch/color", "b");
    } else if (color == 'o') {
      client.publish("garage/echoSwitch/color", "o");
    }
    Serial.println(color);        // Show the new color value in the serial monitor
    lastColor = color;            // Recognize the new color as the last color
  }

  // Wait a tenth of a second before taking another sensor reading (this number is
            variablle, smaller amounts will occur faster)
  delay(100);
}





void connectMQTT() {              // When first connecting to MQTT or when it becomes
                                          disconnected, run this script
  client.setServer(mqttServer, mqttPort);      // Setup the server connection with
                                                      information the user provided
  client.setCallback(callback);   // Setup the callback to listen to changes made on the
                                          MQTT server
  while(!client.connected()) {    // If the MQTT server is not connected or becomes
                                          disconnected, try again
    Serial.println("connecting to MQTT...");
    if (client.connect("EchoSensor")) {        // Try to connect with this name
      Serial.println("Connected to MQTT");
    } else {
      Serial.print("failed... Re");            // Repeatedly says "failed...
                                          Reconnecting" as this loop cycles through
      // Serial.print(client.state());            // Show in the Serial Monitor what the
                                          current state of the server connection is
      delay(2000);                // Wait two seconds before trying to connect to the
                                          server again
    }
  }
  client.subscribe("garage/doorSwitch");      // Listen to this path on the server and
                                                      when a change is noticed, run the
                                                      callback function
}





void callback(char* topic, byte* payload, unsigned int length)
{
  // The callback listens to the MQTT server and checks for changes in the
            "garage/doorSwitch"
  // This function occurs after a message was received
  Serial.print("Message arrived in topic: ");
  Serial.println(topic);          // Notify which subscribed path received a change
  Serial.print("Message:");

  String value = "";              // To contain the value from the MQTT path
  for(int i = 0; i < length; i++) {
    char ch = ((char)payload[i]); // Only one character is read at a time
    value.concat(ch);             // So create "value" based on the string of these
                                          characters
  }
  Serial.print(value);            // Show the value that came through in the serial
                                          monitor
  startSensor = (value.toInt());  // If a one came through, then the loop() function can
                                          begin collecting sensor data. If a 0 came
                                          through, it will stop
  Serial.println();
}
```

Lab_4_Stoplight.ino

```
/*
 * Created by Aaron Nelson
 * Lab 4 Event Bus
 * 10-26-19
 * This script is used with the stoplight module. It will light up different colors
      based on what is passed to the MQTT server (posted from an ultrasonic sensor).
 *
 * The MQTT portion of this script was obtained from
      https://arduinodiy.wordpress.com/2017/11/24/mqtt-for-beginners/
 *
    Stoplight Pins:
        Green: Pin12 (D6)
        Yellow: Pin13 (D7)
        Red: Pin15 (D8)
        GND: GND
*/

#include <ESP8266WiFi.h>            // This contains the libraries that allows the board
                                          to connect to wifi
#include <PubSubClient.h>           // This is used for communication with the MQTT
                                          Server

const char* ssid = "YOUR WIFI HERE";        // Specify the name of your wifi
const char* password = "YOUR PASSWORD HERE";       // Specify the password for your wifi

const char* mqttServer = "192.168.137.1"; // The location of your MQTT Server
const int mqttPort = 1883;                // The port number your MQTT Server is running
                                                on (1883 is the default for Mosquitto)

unsigned long beginningMillis;      // A variable to hold our current time (used for
                                          auto mode to keep track of how much time has
                                          passed)
int onBlink;                        // Set if the server reading sent 'b'. Will change
                                          the red light on and off

WiFiClient stoplightClient;         // Name for our MQTT connection client
PubSubClient client(stoplightClient);     // Creates a partially initialized client
                                                instance

int green = 12;                     // Pin D6 is used for green. If using a different
                                          pin, specify that here
int yellow = 13;                    // Pin D7 is used for yellow. If using a different
                                          pin, specify that here
int red = 15;                       // Pin D8 is used for red. If using a different pin,
                                          specify that here
```

```
int redFlash = 0;                    // While this value is 1, it specifies the light
                                           should be red, while 0, it is off during the
                                           flashing stage


void setup() {                       // This runs when the board is first turned on
  Serial.begin(115200);              // This allows serial information when connected to
                                           a computer (this will tell what echo sensor
                                           message was sent to the MQTT server)
  pinMode(green, OUTPUT);            // Prepares the pin connected to the green light for
                                           output
  pinMode(yellow, OUTPUT);           // Prepares the pin connected to the yellow light
                                           for output
  pinMode(red, OUTPUT);              // Prepares the pin connected to the red light for
                                           output
  lightsOff();                       // Initialize the device by turning all of the
                                           stoplight lights off

  /* Connect to WiFi */
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");    // This is shown on the serial if connected to a
                                           computer
  Serial.println(ssid);              // It displays the wifi it is trying to connect to

  WiFi.mode(WIFI_STA);               // It sets the wifis mode to "Station" (rather than
                                           "AP" which is the Access Point mode)
  WiFi.hostname("Stoplight");        // Hostname to uniquely identify our device
  WiFi.begin(ssid, password);        // Attempts to connect to wifi using provided
                                           credentials

  while (WiFi.status() != WL_CONNECTED) {     // While the wifi is not connected yet:
    delay(500);                      // every half second,
    Serial.print(".");               // it prints a dot on the Serial Monitor to show it
                                           is still trying to connect
  }
  Serial.println("");
  Serial.println("WiFi connected"); // When it does connect, show that there was success

  connectMQTT();                     // Runs the function to set up our MQTT connection

  // Blink LED quickly to show that it is connected to wifi and the MQTT server
  digitalWrite(LED_BUILTIN, LOW);    // light turns on
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);   // light turns off
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);    // light turns on
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);   // light turns off

  client.publish("device/stoplight", "Activated");  // Post in the device channel that
                                                          this device is on
}
```

```
void loop() {                    // This constantly cycles through, checking the MQTT
                                       server for a message, reconnecting if it
                                       becomes disconnected, and blinking the red
                                       light if that setting is on
  client.loop();                 // Runs through the MQTT functions to process
                                       incoming messages and send publish data

  if (!client.connected()){          // Occasionally the client drops connection randomly
                                       and stops working
    Serial.println("MQTT was Unexpectedly Disconnected");
    connectMQTT();               // If the connection dropped, try to reconnect to
                                       our server
  }

  if (onBlink == 1){             // If the red light is supposed to be blinking
    if ((325 + beginningMillis) < millis()) {        // 325ms after the light last
                                                   blinked
      if (redFlash == 0){           // If the red light was off,
          redLight();               // turn the red light on
          redFlash = 1;             // and change the variable to have it turn off next
                                       time
      } else if (redFlash == 1) {   // If the red light was on,
          lightsOff();              // turn the red light off
          redFlash = 0;             // and change the variable to ahve it turn on next
                                       time
      }
      beginningMillis = millis();   // Restart the counter for when the loop will run
                                       again
      delay(5);
    }
  }
}




void connectMQTT() {               // When first connecting to MQTT or when it becomes
                                       disconnected, run this script
  client.setServer(mqttServer, mqttPort);     // Setup the server connection with
                                          information the user provided
  client.setCallback(callback);    // Setup the callback to listen to changes made on the
                                       MQTT server
  while(!client.connected()) {     // If the MQTT server is not connected or becomes
                                       disconnected, try again
    Serial.println("connecting to MQTT...");
    if (client.connect("Stoplight")) {         // Try to connect with this name
      Serial.println("Connected to MQTT");
    } else {
      Serial.print("failed... Re");            // Repeatedly says "failed...
                                       Reconnecting" as this loop cycles through
      // Serial.print(client.state());           // Show in the Serial Monitor what the
                                       current state of the server connection is
      delay(2000);                 // Wait two seconds before trying to connect to the
                                       server again
    }
  }
  client.subscribe("garage/echoSwitch/color");  // Listen to this path on the server and
                                 when a change is noticed, run the callback function
}




void callback(char* topic, byte* payload, unsigned int length)
{
  // The callback listens to the MQTT server and checks for changes in the
          "garage/echoSwitch/color"
  // This function occurs after a message was received
  Serial.print("Message arrived in topic: ");
  Serial.println(topic);          // Notify which subscribed path received a change
  Serial.print("Message:");

  String value = "";              // To contain the value from the MQTT path
  for(int i = 0; i < length; i++) {
    char ch = ((char)payload[i]); // Only one character is read at a time
    value.concat(ch);             // So create "value" based on the string of these
                                          characters
  }
  Serial.print(value);            // Show the value that came through in the serial
                                          monitor
  Serial.println();

  // Respond to the color the Echo Sensor sent to tell which color the light should turn
  if (value == "g") {
    // Echo sensor is over 35 cm
    greenLight();
    onBlink = 0;
  } else if (value == "y") {
    // Echo sensor is between 20-35 cm
    yellowLight();
    onBlink = 0;
  } else if (value == "r") {
    // Echo sensor is between 10-20 cm
    redLight();
    onBlink = 0;
  } else if (value == "b") {
    // Echo sensor is between 0-10 cm
    lightsOff();
    // Because this is too close, we wish to flash the red light (functionality for this
          is in loop())
    onBlink = 1;
    beginningMillis = millis();
  } else if (value == "o") {
    // The door switch became disconnected and the echo sensor is no longer reading
          values
    lightsOff();
    onBlink = 0;
  }
}




/* These are the functions that are called depending on the read MQTT value.
```

```
    They only turn on the light they are supposed to according to the pin number
      specified at the beginning of the script */
void lightsOff() {
  digitalWrite(green, LOW);
  digitalWrite(yellow, LOW);
  digitalWrite(red, LOW);
}

void greenLight() {
  digitalWrite(green, HIGH);
  digitalWrite(yellow, LOW);
  digitalWrite(red, LOW);
}

void yellowLight() {
  digitalWrite(green, LOW);
  digitalWrite(yellow, HIGH);
  digitalWrite(red, LOW);
}

void redLight() {
  digitalWrite(green, LOW);
  digitalWrite(yellow, LOW);
  digitalWrite(red, HIGH);
}
```