Lab 3 – Machine to Machine Communication

Online Links

This lab is available on my personal website at: http://AaronNelson95.com/IT441Lab3.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 Arduino devices can be configured to communicate with each other. This is done by having one Wemos board act as a web server to broadcast information, while another Wemos board acts as a client to respond to the posted values. This lab also introduces one to the HC-SR04 Ultrasonic Sensor, how to code with one, and how to use this device to detect how far away an object is. This will create a device that can be used, for example, to tell how close a vehicle is and provide a warning stoplight if they have more room to back up or not. This lab will help one learn:

- How to develop an Arduino program that works as a server that will continually post information
- How to develop an Arduino program that can act as a client and read data from a server
- How the Ultrasonic Sensor works and how this can be used to detect distances
- How basic state diagrams and flow charts help in developing and understanding a project

Materials

To complete this lab, you will need:

- Two Wemos D1 mini microcontrollers
- Two Micro USB power cords
- Two breadboards
- A traffic light LED display module
- An Ultrasonic Sensor (HC-SR04)
- 8 Male-Male Jumper Wires
- A computer with the Arduino IDE

The HC-SR04 Ultrasonic Sensor

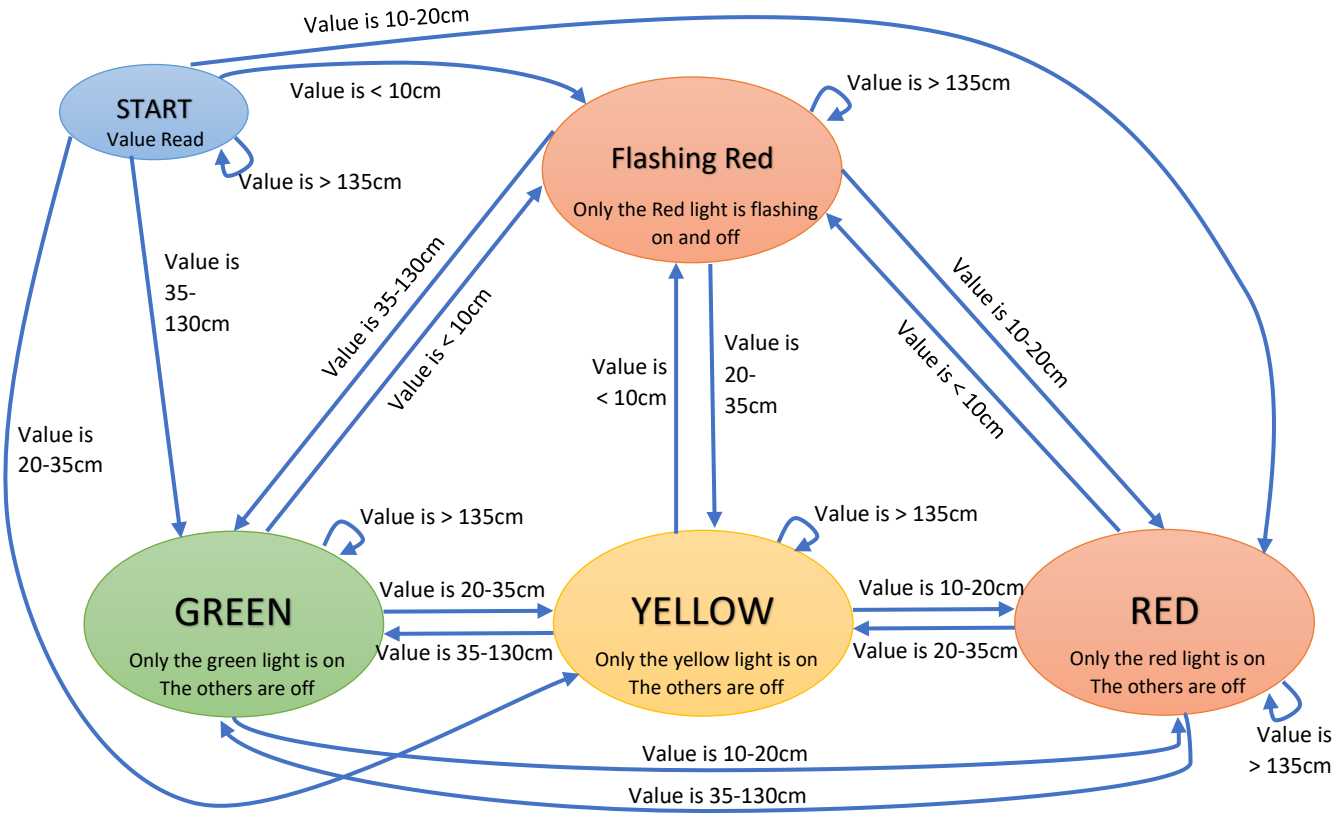
References

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/esp8266-dht11dht22-temperature-and-humidity-web-server-with-arduino-ide/ - A tutorial on setting up an Arduino temperature and humidity web server. We are hosting distance values in a similar way.
- http://www.esp8266learning.com/wemos-webserver-example.php - A basic tutorial for connecting the microcontroller to the internet and broadcasting a simple server. This is also available in the IDE through File -> Examples -> ESP8266WiFi -> WiFiManualWebServer.
- The IDE example for creating a simple web client found through File -> Examples -> ESP8266WiFi -> WiFiClient.
- 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.

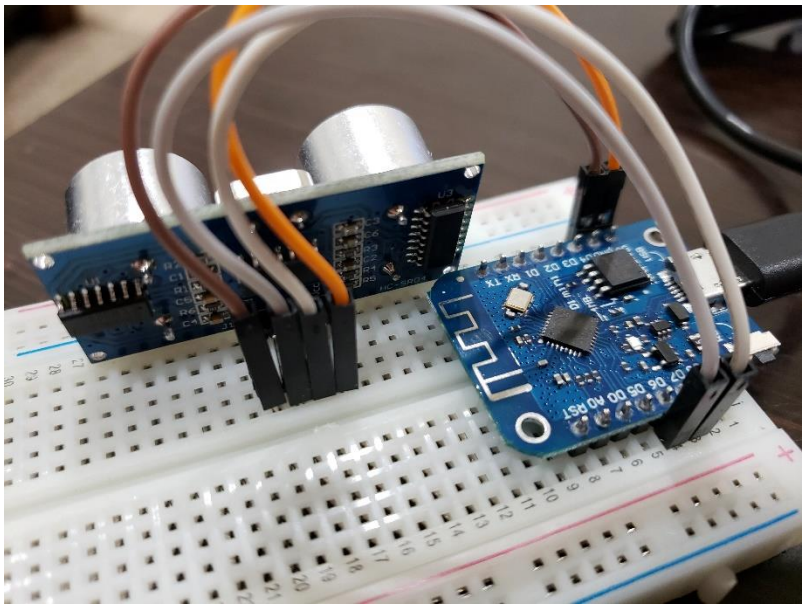
Procedures

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 4 light states: when the green light is on, when the yellow light is on, when the red light is on, and when the red light is flashing. The states are shown through the stoplight client. The transitions between the states are determined by the sensor server.
        ii. When the board is turned on, the device will receive a distance value from the ultrasonic sensor. Depending on this value, it will go to one of these four states (or ignore the reading if it is invalid and over 135cm).
        iii. When a value is received from the sensor periodically, that value is used to determine what state the light should turn. If it is between 35-130cm, it goes to green. 20-35cm goes to yellow. 10-20cm goes to red. 0-10cm blinks the red light. Any value over 135 is rejected, as these are often error values (such as when half of the sensor is blocked).

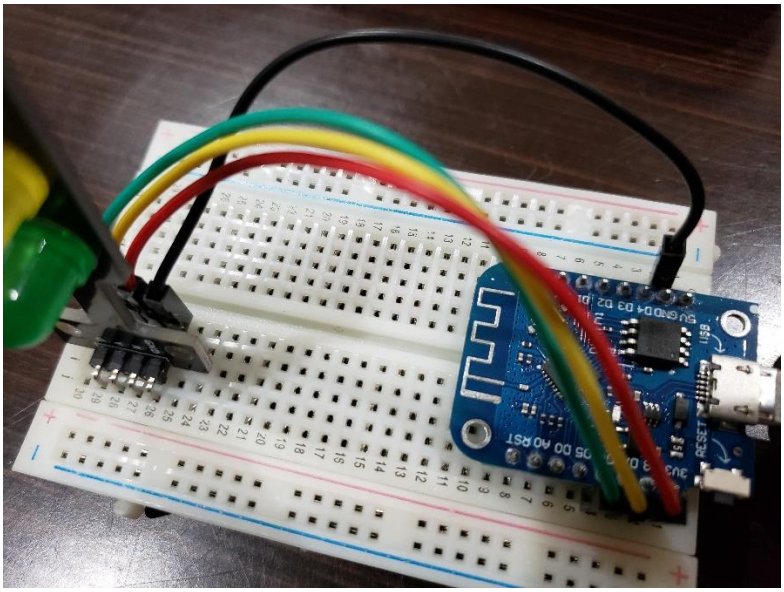


b. Develop a system flowchart
    i. This should show an idea of how to program the microcontroller. Arduino has two main functions: the startup() function, which runs once as soon as the board gains power, and the loop() function which continually runs after the startup is finished. Try to consider how you can implement the state diagram using these two functions. Realize that there will be two parts to this flow chart- one that reads information from the sensor and posts it to a server, and one that retrieves this data as a client and lights up a stoplight. This diagram will be modified as the code progresses.
    ii. An example of a finished system flowchart for this lab is available in the appendix.

4. Connect the server 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 boards GND. A circuit diagram of this setup is available in the appendix.



5. Also connect the client 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. Implement the system flowchart and state diagram designed earlier. Begin by first working on the server Wemos board. 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. Save the sketch to the board by selecting **Sketch** -> **Upload**. The code will be compiled and written to the board.
7. Next, include a server functionality for this script. An example server can be found in **File** -> **Examples** -> **ESP8266WiFi** -> **WiFiManualWebServer**. You can set your port (such as 17) to communicate the data and pass the sensor's distance value to this location each time the program loops.
8. Now, with the other board, build a wifi client that will read data from this web server. An example client can be found in **File** -> **Examples** -> **ESP8266WiFi** -> **WiFiClient**. See if you can receive the value and print it to the Serial Monitor. Next, work with those values to adjust the stoplight's color. This can be done with turning different GPIO pins on and off. For help with this part, see my lab 2 example (http://aaronnelson95.com/IT441Lab2.php). Be careful what COM port you upload this code to, as it may be likely that both devices are plugged into your computer. Be sure you are writing to the client board and you are not overwriting the server board.
9. When both parts are complete, you should be able to power on your server board which will connect to the internet and begin broadcasting data. Read from the server's IP address (this might change over time so you may need to reupload your client code when you return to this project later) and when certain distance values are detected, the stoplight should respond and behave like how the state diagram was designed.

Thought Questions

1. Think of the interaction between your devices. How well would this scale to multiple devices? Is it easy to add another sensor? Another actuator?

    This sensor server and stoplight client system works fairly well. They communicate with each other by recording and reading a simple value across devices. I imagine it would work to add a couple more client-based devices as these are simply reading data that the server is providing. However, port 17 may not be the best method for doing more large scale projects. It may work better to show this information on HTTP port 80 instead and have a longer delay time between cycles. I noticed when I was using my server and client Arduino boards, and I *also* tried accessing the server's port directly on my computer, it would often freeze up the server and I would have to reset my board (which seemed very strange). I believe it would stretch farther if it was additional simple Arduino boards acting as clients instead. It would be possible to add another sensor server but this would take longer for the client to run through its loop. If one of the servers the client is connected to is having a problem or is timing out, then this device may continually run into errors or could return inaccurate results. It would be possible to implement multiple devices together, but a programmer would need to be careful to keep all of the different devices and information straight and handle disconnection issues.

    Overall, this setup could be useful with reporting and receiving data across multiple devices. They can be used for simple home automation. However, they are very heavily dependent that both devices are constantly connected to the internet. If a server has an issue, the clients will all just timeout possibly unexpectedly. Also,

another large challenge for this interaction is their overall network connectivity. If these devices are used in a different location, then they would all need to be manually updated with the proper wifi information (or updated with the wifi manager). Having static IPs set would be important as well, because if a server changes its IP, then all of its clients will need to be hard coded and reuploaded with that new address. I encountered a problem with these boards and trying to set static IPs on my wifi hotspot (they still were connected and only worked through the address assigned by the computer even though the serial monitor and the board's "WiFi.localIP()" reflected the IP *I* assigned it, and I did try multiple methods suggested on the Arduino site). It would require busy work with reupdating the code and devices, but if Arduino device static IPs properly work on a home's network and they are likely to stay in that location, then this system could be more scalable.

2. What are the strengths and weaknesses of the tennis-ball-on-a-string system that Don had originally? What are strengths and weaknesses of the IoT system that he developed? What enhancements would you suggest?

   A major advantage that the traditional tennis-ball-on-a-string system has over this device is that it would work even with the power out or if the wifi wasn't working properly. This method requires no technology, so it is not susceptible to any electronic issues. However, like the case study mentioned, children may play around with the ball or it could become disconnected. This system also needs to be carefully measured out with the specific car that will be parked in that spot, and it may only work for that one car. With the Arduino system, any car can park in the spot depending on where the sensor is placed. Also, this device will work very well in the dark because it uses sound to determine distance. A glowing stoplight would be easy to see in these conditions, while the device would still work accurately. However, this system may occasionally receive sporadic values, especially if half of the device is covered by something. You would need good clearance for it to work properly. Also, this device will need to remain connected to the home's network, because if one part of the system goes down, you would be out of luck with parking your car. Also, this sensor would be constantly sending and receiving waves and writing these distances to the server, which is pointless considering you would only need it to work for a minute when parking your car (it shouldn't need to use this power while you're sleeping).

   To improve this device, it would be helpful to incorporate more accurate error detection. I noticed a lot of incorrect values were very large (like in the thousands) so I simply reject any value over the typical size of my room/testing area. Unfortunately, this method doesn't help if the sensor ever jumps from large to extremely small values. To improve this, it could be possible to remember the last 3 or so values that were returned and compare the average of those numbers to the next read value. If that value is off (whether too large or too small of a jump), you can reject that value but still keep it in the accounting for the next average. This method would require some careful planning and testing. This device can also be improved by incorporating a lights out feature such as when the garage door is closed, or when the sensor is picking up readings that reach the other end of your garage (the car isn't there). It would work best to activate and turn this system on when you are *approaching* your house (but there may be not enough time to connect to the network and other devices if this is only triggered once the garage door is opened). Finally, this device will also be improved with fixed IP addresses. Often, when they are reconnected to the internet, they receive a different IP address. This new address will then need to be hardcoded and reuploaded to the client device. Having static IPs will help these devices connect to the internet faster and find each other.

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

   The largest challenge I had with this lab was with my server disconnecting. Often it would work just fine and post information every quarter of a second like I wanted. However, one day after this was working great for long periods of time, when I returned to work on my code, I noticed that my server would stop broadcasting and the sensor stopped reading data after about a minute. This continued to happen when I reset the board and changed the waiting times before the code would loop again. This was very frustrating, but I eventually

narrowed down part of the problem. First of all, I reused some of the client connection code from the first lab (such as “client.available()”), I realized that I didn’t need to incorporate this information at all, because the server isn’t directly responding or listening to a client in any way. I also noticed that when I checked the server’s port in my computer’s web browser, the server would stop working after a few more cycles. When I only connected to my server with my Arduino client after making these changes, I did not receive any more timeout issues. I also had an odd issue when trying to test my device at school. Although they are connected to my computer’s hotspot, I noticed they behaved very strangely and extremely slowly at school. Simply turning off my internet connection from eduroam (so I wasn’t “sharing” internet to these devices, only providing a network), caused them to work quickly and correctly again.

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

   Luckily, I was able to work quickly through this lab. It only took me around 4 hours total to learn about the ultrasonic sensor, write the server and client scripts, comment the code, and deal with any problems I encountered. After learning about Arduino from the last lab, I was more comfortable with programming on the boards and I came into this project with an idea of how to accomplish it right away. The lab report took me a while longer, around 7 hours, for a total of 11 hours.

Certification of Work

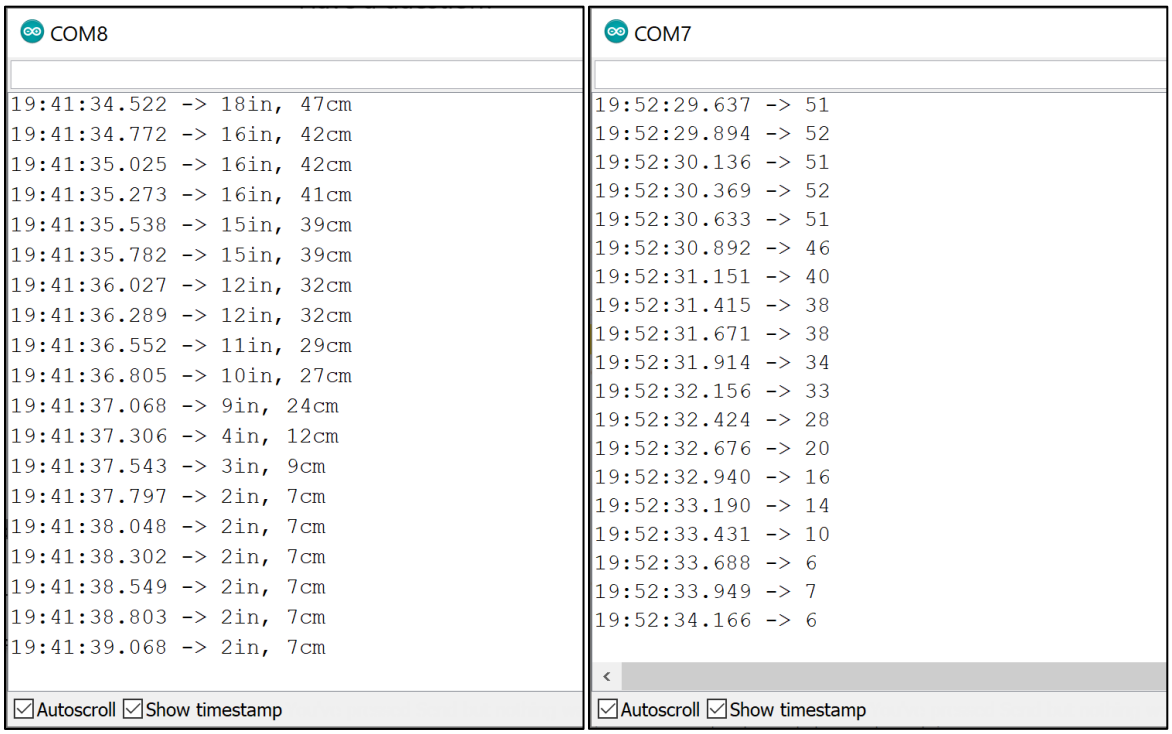
I certify that this lab represents my own work. I used various web resources to help get me started with this Arduino project though, such as the basis for obtaining and converting readings form the ultrasonic sensor. a simple web server example, and a simple web client, but I did not copy anything that was not basic knowledge unless I specified it inside of my code comments.

– Aaron Nelson

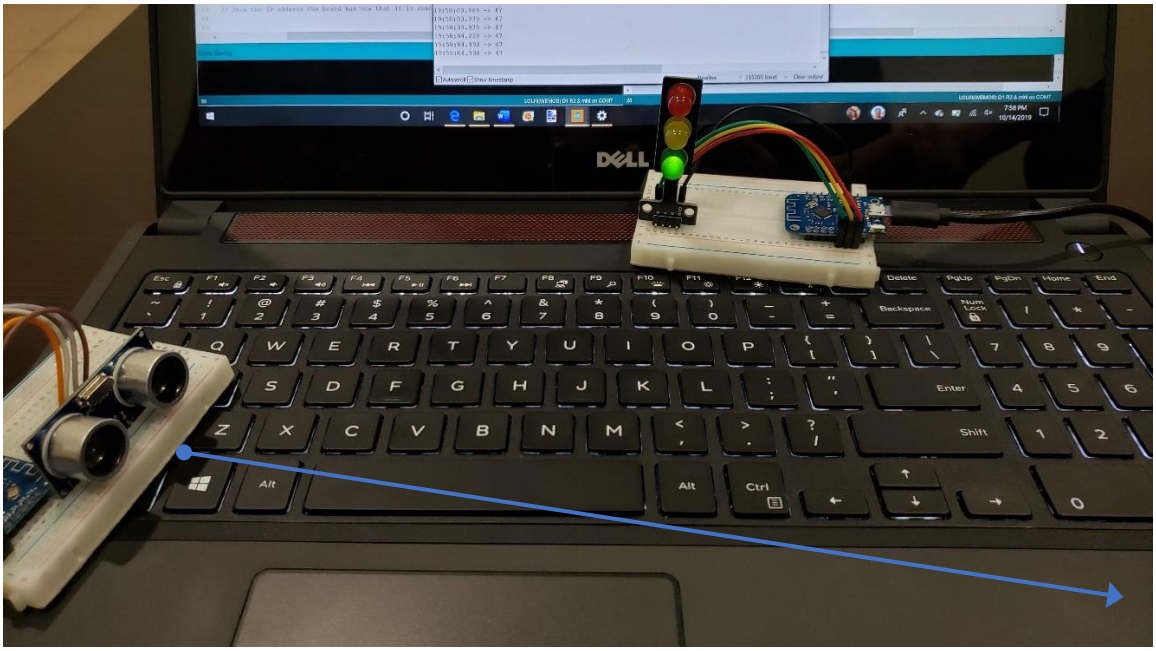
Appendix

Images of Final Product

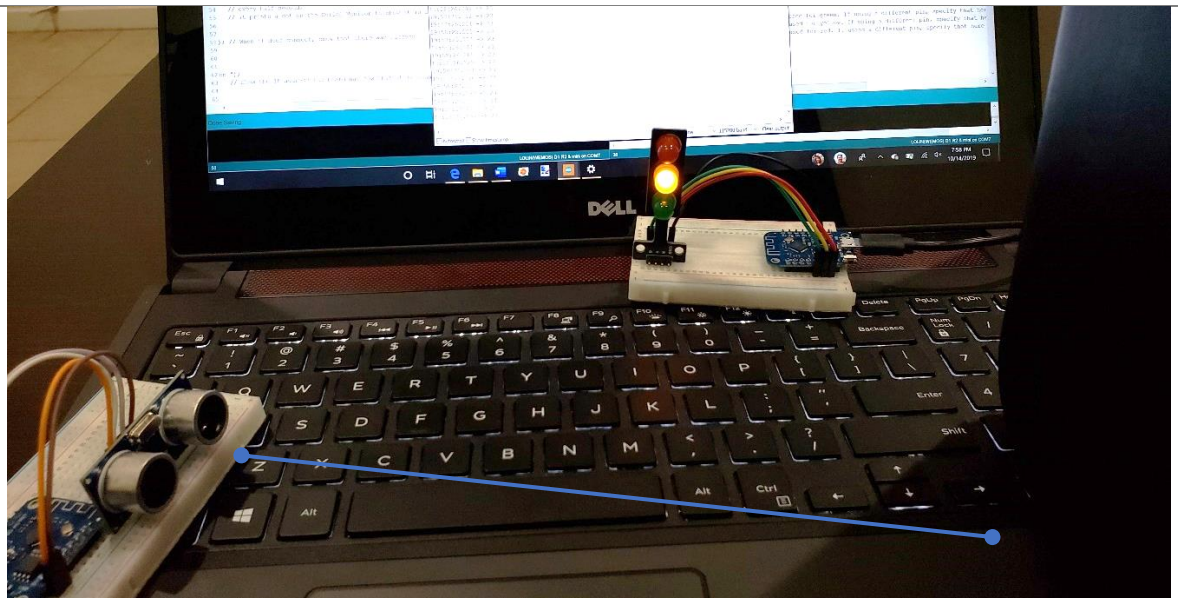
As can be seen below, depending on the distance an object is away from the sensor, a different stoplight color will light up. The Serial Monitor can be used for debugging and for informing the developer of certain information, such as the board's IP address and the current readings that are being processed from a sensor or from a server.



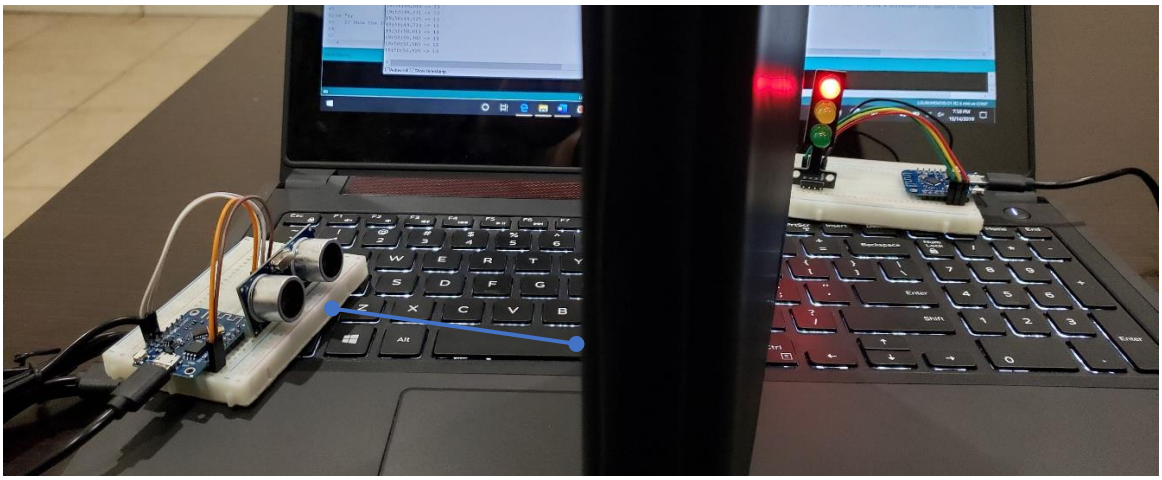
The Serial Monitor output of the Sensor Server connected to COM8 (left) which is showing sensor readings, and the Stoplight Client on COM7 (right) which shows what is being read from the server.



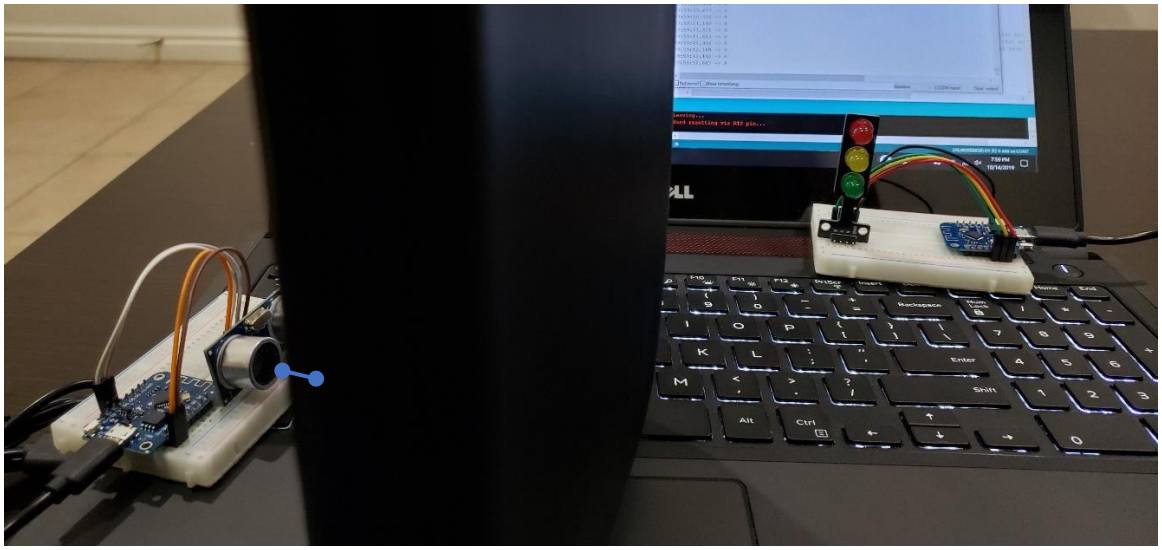
With large open distances (over 35cm), the stoplight turns green.



When an object comes closer, between 20-35cm, the stoplight turns yellow.

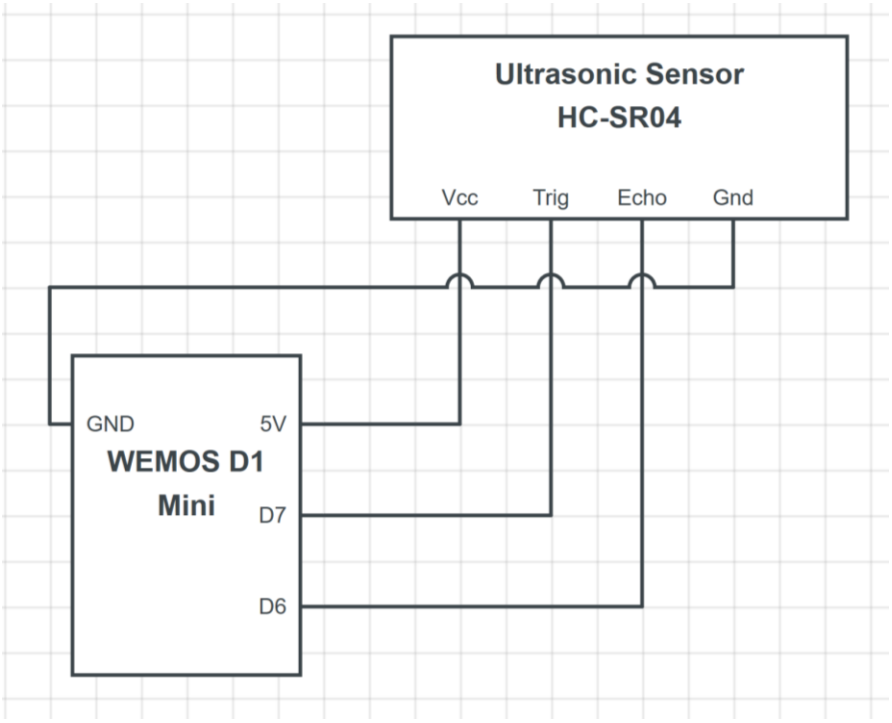


An even closer object, between 10-20cm turns the light red.

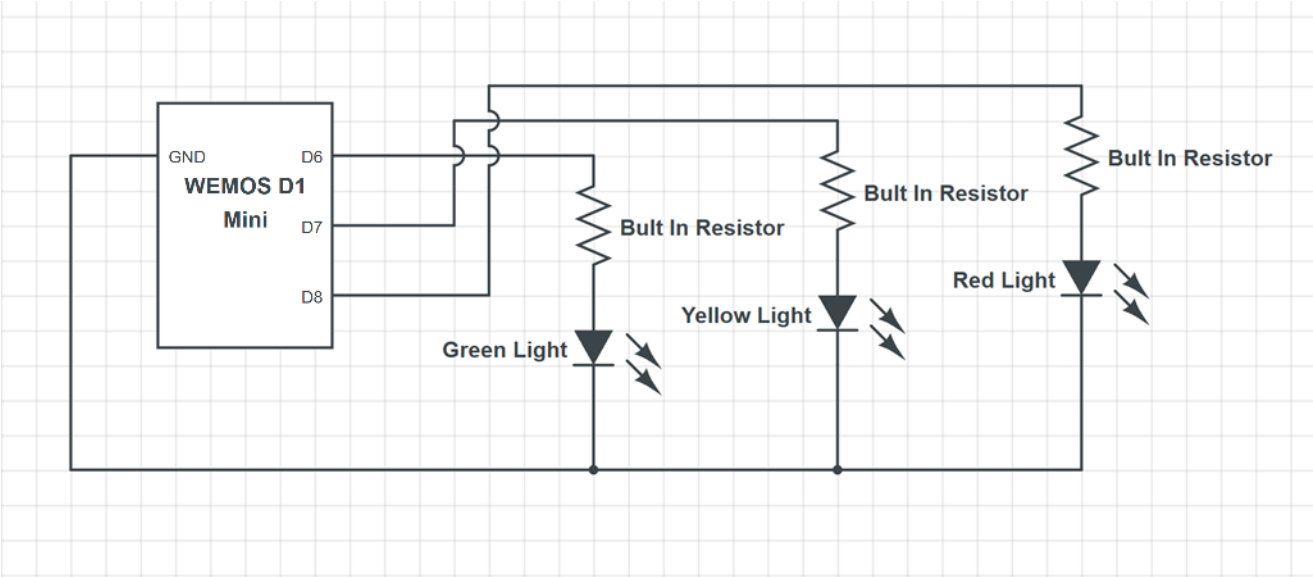


When an object is too close (< 10cm), the light will blink (it is now in the dark stage).

Circuit Diagrams

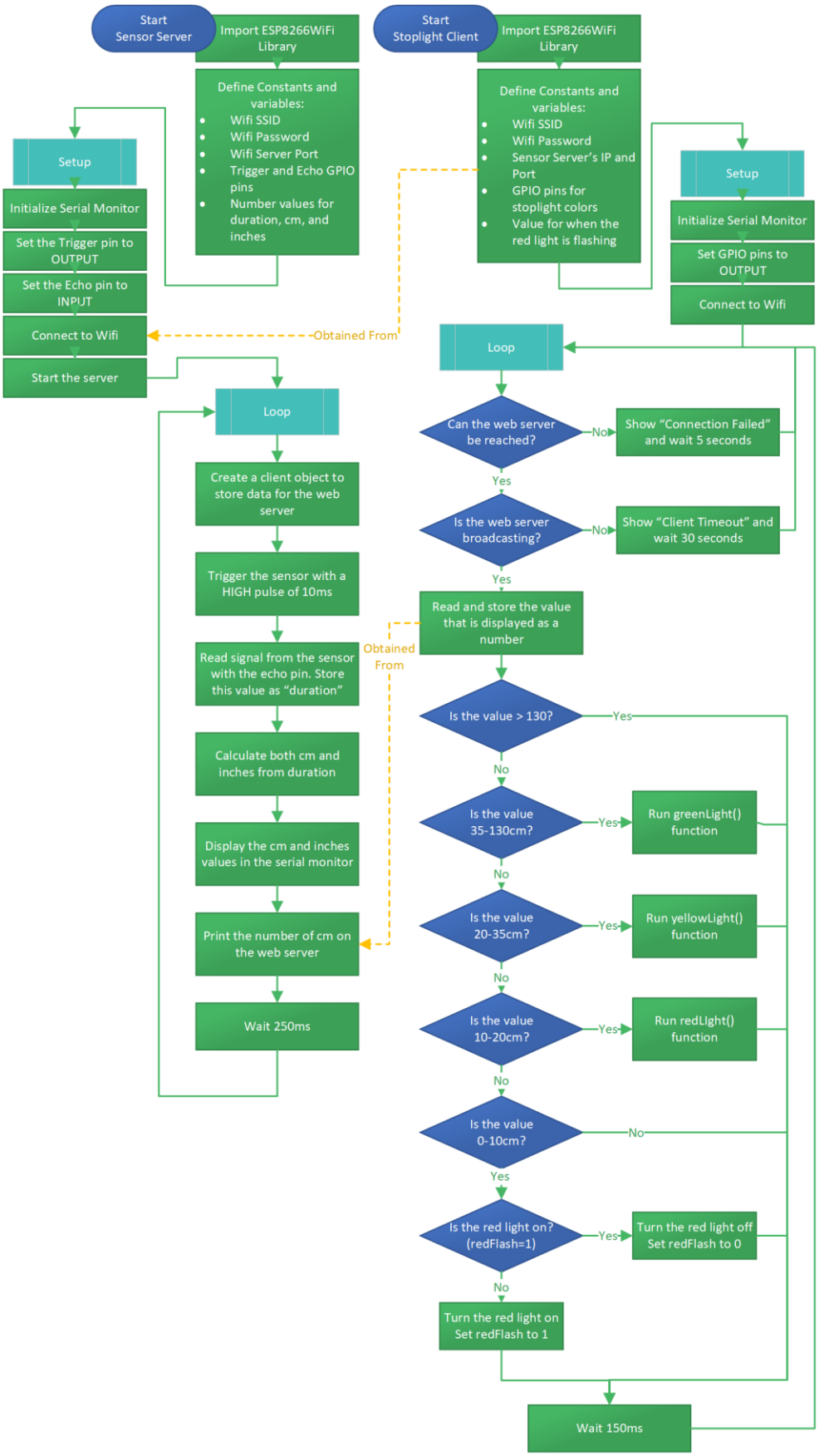


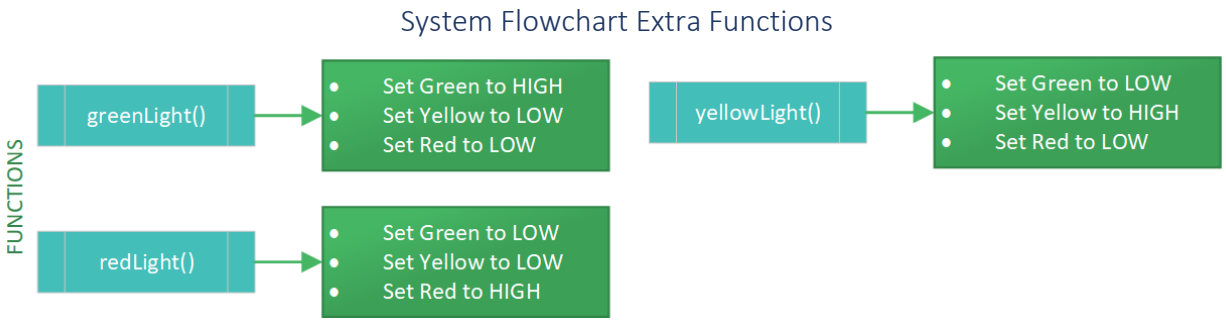
The Sensor Server



The Client Stoplight

System Flowchart





NOTE: A larger version of my flowchart is available on my website.

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

Arduino Code

Lab_3_Echo_Server.ino

```
/*
 * Created by Aaron Nelson
 * Lab 3 Vehicle Sensor
 * 10-14-19
 * This script is used for the ultrasonic sensor (HC-SR04) to send readings frequently
      to a server. It works in collaboration with the stoplight, which reads from this
      server.
 *
 * Original Script for the sensor functionality was created by Rui Santos, provided at:
 *      https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/
 *
 * The basic server setup and framework was obtained from the Arduino Examples-
 *      File -> Examples -> ESP8266WIFI -> WiFiManualWebServer
 *
    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


#ifndef STASSID
#define STASSID "YOUR WIFI HERE"     // ***Specify the name of your wifi
#define STAPSK  "YOUR PASSWORD HERE" // ***Specify the password for that wifi
#endif
const char* ssid = STASSID;
const char* password = STAPSK;

WiFiServer server(17);              // This sets up the server to listen on port 17

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



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)


  // Connect to WiFi network
  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.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

  // Start the server
  server.begin();
  Serial.println("Server started on ");
  Serial.println(WiFi.localIP());   // Show the IP address the board has now that it is
                                        connected and broadcasting

}




void loop() {                       // This constantly cycles through, reading value
                                        from the sensor and printing that on the
                                        hosted server
  WiFiClient client = server.available();     // A client object is created for
                                                  connections

  // 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 correct 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();

  // Print the number of centimeters on the web server (which is read by the stoplight
      client)
  client.println(cm);

  // Wait half of a second before taking another sensor reading (this number is
      variable, smaller amounts will occur faster but you don't want to bog down the
      server)
  delay(250);
}
```

Lab_3_Stoplight_Client.ino

```
/*
 * Created by Aaron Nelson
 * Lab 3 Vehicle Sensor
 * 10-14-19
 * This script is used with the stoplight module to respond to and act as a client when
     reading data from a server that posts values from an ultrasonic sensor.
 *
 * The basic client setup and framework was obtained from the Arduino Examples-
 *      File -> Examples -> ESP8266WIFI -> WiFiClient
 *
    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

#ifndef STASSID
#define STASSID "YOUR WIFI HERE"     // ***Specify the name of your wifi
#define STAPSK  "YOUR PASSWORD HERE" // ***Specify the password for that wifi
#endif
const char* ssid = STASSID;
const char* password = STAPSK;

const char* host = "192.168.***.***";    // ***The IP address the server is running on
                                            (as specified when the sensor server
                                            connect to internet. Enter that IP here)
const uint16_t port = 17;           // The port that the server is running on (as
                                        specified in the sensor server)

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 (in here, this just tells what cm
                                           reading is seen from the 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

  // We start the client by connecting to a WiFi network
  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.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
}




void loop() {                           // This constantly cycles through, checking the
                                           server and reflecting the proper stoplight
                                           color
  // Use WiFiClient class to create TCP connections
  WiFiClient client;                    // A client object is created for connections
  if (!client.connect(host, port)) {          // If we cannot connect to the server,
    Serial.println("connection failed");      // Show this in the Serial Monitor
    delay(5000);                        // Wait before trying to connect again (after loop
                                           cycles)
    return;
  }

  // wait for data to be available
  unsigned long timeout = millis();           // Used for a non-blocking loop to check
                                                 if the client times out
  while (client.available() == 0) {           // No client is available
```

```
    if (millis() - timeout > 5000) {          // 5 seconds pass before a client is
                                                 picked up again
      Serial.println(">>> Client Timeout !");   // Timeout the client
      client.stop();
      delay(30000);                  // Wait half a minute before trying to reconnect.
                                        There must be a server problem
      return;
    }
  }

  // Read all the lines of the reply from server and print them to the Serial Monitor
  // Information is read from the server one character at a time. This will add up those
      characters to make up the number values (from a string)
  String number = "";                // String to receive the web server characters
  while (client.available()) {       // While the server is connected
    char ch = static_cast<char>(client.read()); // Read each character one at a time
    number.concat(ch);               // Add these values to our string
  }
  Serial.println(number.toInt());    // Show the received value on the Serial Monitor
  int value = (number.toInt());      // Turn the received value to an integer to compare


  // Use the value the sensor sends to change the light color
    if (value > 130) {
      // This is likely an odd reading, do nothing... (if a room is bigger, expand this
            number)
    } else if (value > 35){
      // Distance is over 35cm, a fair bit away. It is safe to keep driving
      greenLight();
    } else if (value > 20) {
      // Distance is between 20-35cm, we are getting closer so show a warning light
      yellowLight();
    } else if (value > 10) {
      // Distance is between 10-20cm, the car should stop now
      redLight();
    } else if (value > 0) {
      // Distance is less than 10ms, show that the driver is too close and there is no
            walkway and they should back up
      digitalWrite(green, LOW);      // Don't show the green light if it bounced to this
                                        state
      digitalWrite(yellow, LOW);     // Don't show the yellow light if it bounced to this
                                        state
      // Bounce between "redFlash" values of 0 or 1 each cycle to flash the red light
      if (redFlash == 0){
        digitalWrite(red, HIGH);     // Turn on the red light
        redFlash = 1;
      } else if (redFlash == 1) {
        digitalWrite(red, LOW);      // Turn off the red (all) light(s)
        redFlash = 0;
      }

    }

  client.stop();                     // Close the connection to the server
  delay(150);                        // Wait 150ms before reading the server for data
                                        again (this number can be variable depending
                                        on how often you'd like to make requests)
}
```

```
/* These are the functions that are called after a sensor reading is read.
    They only turn on the light they are supposed to according to the pin number
    specified at the beginning of the script */
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);
}
```