Final Lab

Garage-Full-O-Debt

Branden Bearden
Jordan Stafford
Final Lab
ECE 631

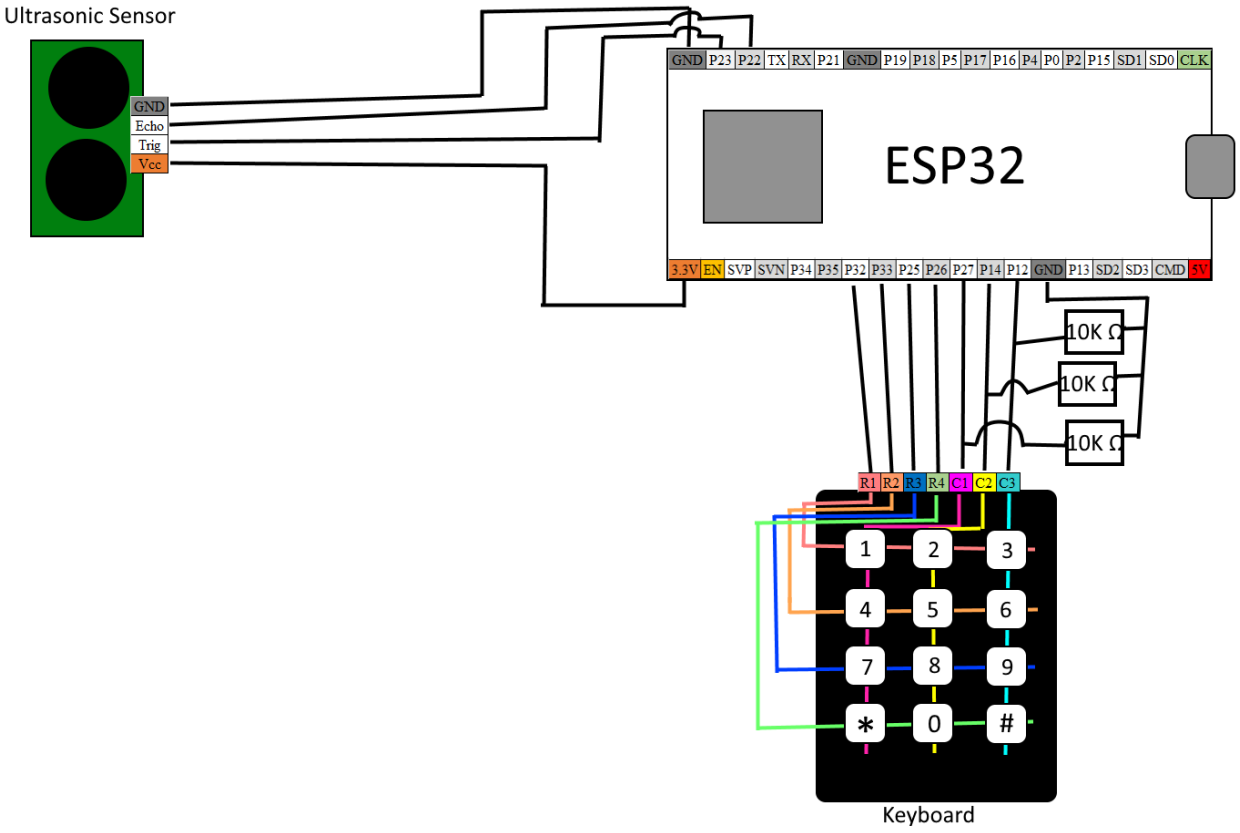
Introduction

In our final lab, our group decided to create and code a working garage door network. Our garage door would have the capability to sense if the car is in the garage or not, could be accessed by using keycards, or could also be accessed by a keypad. To achieve these features we used 2 ESP32s, a Raspberry pi 4, Ultrasonic sensor, a NFC, and a 4x3 keypad. The software we used to connect all of our hardware was HiveMQTT, Arduino IDE, and Freeboard.

Design Partitioning

In our lab, we assigned each part to do a certain role in the garage door. For our hardware, we separated all the devices into 3 separate seconds: Input, Relay, and Driver. One of the ESP32s was strictly designed for input. It would drive both the Ultrasonic Sensor and the keypad. It would then relay this information to the Raspberry pi 4, which was our network hub and wiki relay. The Raspberry pi would take all input from the first ESP32 and relay it to the MQTT server and the other ESP32. The second ESP32 would be our garage door driver. This would hold all password and UID keycard codes and drive the mechanics of the garage door.

Hardware



Our input design structure consisted of an ESP32, an Ultrasonic Sensor, and a 3x4 keypad. The diagram above shows all the design elements and how each device was connected. A picture of the setup is also given in appendix B.

The ultrasonic sensor uses a sound wave to determine the distance the object is from the sensor. This sound wave is triggered by the ESP32 during its main loop of the code where the trigger pin (pin 37 on the ESP32) goes from low to high. When the sound wave is sent from the sensor, it hits the object and bounces back, being caught by the sensor. Once the sound wave returns to the sensor, an echo is sent back to the ESP32 that triggers an interrupt subroutine that changes the state of the trigger pin from high to low. The pin setup is as followed:

ESP 32 Pins	**Ultrasonic Sensor Pins**
Pin 1: 3.3V	Pin 1: VCC
Pin 37: GPIO 23	Pin 2: Trigger
Pin 36: GPIO 22	Pin 3: Echo
Pin 38: GND	Pin 4: GND

On the ESP32, pin 37 was set to output and pin 36 was set to input_pullup. This enabled the signal from the ESP32 to be sent via pin 37 to the ultrasonic sensor. Then a group of pulses are sent from the ultrasonic sensor to the ESP32 that help determine the distance of the object that the ultrasonic sensor picked up.

Another part of the input hardware is the keypad. In our project, the keypad was used to send a passcode to the input ESP32. This passcode would either be used to verify the user and open the garage door, or be recorded to use as a new password. The keypad has 3 columns and 4 rows. Each pin is connected to certain columns and rows, that when you push down one of the buttons, the button causes a short between the column and row it is associated with. For example, if button 1 was pressed, a short would occur between column 1 and row 1. For a keypad like this to work, one pin setup is to make the row pins as outputs, while the column pins as inputs. When a row pin is set to high, and a button that is associated with that pin is pushed, the column that the button is also associated with becomes high as well. Knowing the exact 2 pin combination tells the ESP32 which button on the keypad is pressed. There also needs to be resistors attached to the input pins and ground. These resistors are there for once the button is released, the voltage on the input pin also goes away, telling the ESP32 that the button was released and normal operation can resume. The pin setup between the keypad and the ESP32 is as followed:

Keypad	ESP32
Pin 1: Row 1	Pin 32: GPIO 32
Pin 2 : Row 2	Pin 33: GPIO 33
Pin 3: Row 3	Pin 25: GPIO 25
Pin 4: Row 4	Pin 26: GPIO 26
Pin 5: Column 1	Pin 27: GPIO 27
Pin 6: Column 2	Pin 14: GPIO 14
Pin 7: Column 3	Pin 15: GPIO 15

The last of our hardware is the Elechouse Near Field Communication V4 (NFC). This device is used to read UID codes from a keycard and send them directly to the Raspberry Pi 4. The UID code was an alternate way to open the garage door without having to use the keypad. The pin setup is as followed:

NFC	RPi4
Pin 1: Serial Clock (SCLK)	Pin 23: SCLK
Pin 2: Master In Slave Out (MISO)	Pin 21: MISO
Pin 3: Master Out Slave In (MOSI)	Pin 19: MOSI
Pin 4: SS	Pin 24: Chip Select (CE0)
Pin 5: Vcc	Pin 17: 3.3V
Pin 6: GND	Pin 20: GND
Pin 8: Reset (RST0)	Pin 15: GPIOI 22 Reset

Software

Our project's software consisted of three separate groups of code. The input code, the relay code, and the driver code. These three code groups were used with the input, relay, and driver hardware respectively.

The input code drives the input ESP32 with gathering and sending data from the keypad and the Ultrasonic Sensor. The code itself is in C++ and utilizes the Arduino programming application. This prebuild code takes integers representing a pulses frequency, a channel number, and a bit resolution and sets up and operates a variable repeating pulse that can be used to send to the ultrasonic sensor. This pulse is then used to send a sound wave to the object, which determines the object's distance. Example code can be seen listed under Tech Explorations in references. When the sound wave returns, an interrupt that is attached to the echo pin (pin 36) is triggered. When the interrupt is triggered, that state goes from high to low. The amount of time that the state is high, determines the distance that the object is from the sensor. This is determined by this formula:

$$\text{Distance} = \frac{(\text{Time.End} - \text{Time.Start}) * (\text{Speed of Sound})}{2}$$

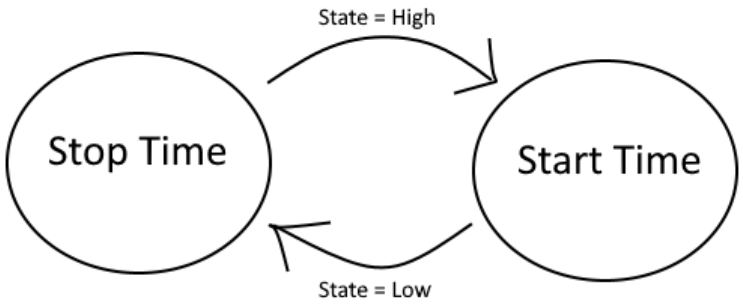
In the code, we first determined the width of the pulse in time in milliseconds. We then converted it to seconds and times it by the speed of light divided by 2. We divided the equation by 2 because the sound wave had to travel double the distance of the object to get back to the sensor and trip the interrupt.

Once the distance is calculated, the code loops through again. With each iteration, the distance that is calculated is added together until the number of iterations reaches a certain sample size. Once the sample size is reached, We use a moving average filter to take the average of the sample sizes to determine the actual distance from the object. The formula to do this is:

$$y[n] = \frac{1}{L} \sum_{k=0}^{L-1} x[n-k]$$

A moving average filter is necessary here to adjust for the movement of the sensor or the object. If the object were to move slightly while calculating the distance, this will be recorded and presented into the printed value. We implemented a type of moving average filter into the code. Instead of using a buffer, We added the variables up into a single distance value, divided the value by the sample size, and then calculated the distance, resetting the variables value to zero after. This wasn't as accurate as a true moving average filter, but it worked for the lab.

While in the main loop, We use a simple state machine to help record the time that the pulse is high. When the trigger pin is written to, the state is written to high. When this happens, the time when the state turns high is recorded. Once the trigger pin is written to again, state toggles to low. This advances the state machine and the time that the state goes low is recorded. The difference between the stop and start time is the width of the pulse that was initiated. This is then used in the distance formula to get the distance of the object. An illustration of the state machine is provided below.



The other aspect of the input code is the keypad. The keypad is a 4x3 matrix with each button on the keypad being able to be represented by a combination of 1 row and 1 column pin. All pins associated with rows are set to digital outputs, while the rows are set to digital inputs. For the code to tell if a button is pressed, all of the row pins are first set to high. If one of the column pins reads high, then a button is pressed. The code then goes into a separate method and checks each individual row and column combination to see which button is pressed. Once the button that is pressed is determined in the code, a character associated with that button is returned out of the method back into the main loop and recorded into a dynamic character array. If for some reason, the method can't determine which button is actually pressed, an empty character is returned, which tells the main loop not to record the entry. If a "*" or a "#" is returned from the method, this signifies that the characters that have already been recorded in the character array need to be erased and to go into one of two modes. If a "*" is returned, this signifies the code needs to go into default settings. If a "#" is returned, then this signifies that the code needs to go into "New Passcode" recording mode and recorde the next entries into the character array. Once 6

characters are recorded into the character array, the main loop goes into sending the password to the MQTT server on the Raspberry Pi. If the code is in default mode, this will send the new passcode under the topic “Password”. If the code is in new password mode, the passcode will be sent under the topic “New Password”. The code for the input hardware is included under Appendix D.

The Raspberry Pi was the hub of all of our input information and processed code. The Raspberry Pi hosts a MQTT server that allows us to take the information from the NFC, keypad, and ultrasonic sensor, and send it to our garage door driver where the input from these devices are processed. All the information from our input devices are sent to a MQTT server topic named “/ece631/Final/Input”. The information processed in our garage door driver code is read from that input topic and when there is an action to the garage door or if there is an error with the input, that information is sent to another topic in our MQTT server named “/ece631/Final/Garage”. With our MQTT server having these to two topics, the Raspberry Pi was also used to create our dashboard in Freeboard. The dashboard consisted of widgets showing the most recent password used, “Incorrect Password” is displayed when the password was typed in wrong, the most recent UID card scanned, “Incorrect UID” is displayed if someone scanned an invalid card, a light widget that, when on, reads “Car is not clear of the garage door” and “Car is clear” when off, and a gauge showed and stated when the garage was open, closed, opening, and closing.

The ESP used as the garage door driver is where the input code is processed and determines the action of the garage door. The code is set up to read the input from our MQTT server topic “/ece631/Final/Input” where we set the payload of the UID, password, and ultrasonic sensor topics to a variable. These variables are then processed in the main loop to open/close the garage when the car is clear of an arbitrary distance, representing the car is all the way in or all the way out of the garage, and the input UID or password matched with a valid UID or password. To represent the opening and closing of the garage door, we flashed the blue LED on the ESP each second the garage door was moving. When the garage was open, the LED stayed on, and when the garage stayed closed, the LED was off. We published the state of the garage door, the distance the ultrasonic sensor was reading, and the UID and password to our MQTT server topic “/ece631/Final/Garage” which was used to display information in our dashboard as explained above.

A link to the repository for our code is included in Appendix A.

Design Issues

Most of our design issues we encountered were from setting up the keypad, both in hardware and in software. Other design complications were stopping the garage to constantly open and close when the UID was read in multiple times, and keeping Freeboard from crashing.

With the keypad, the biggest problem we came across was that the pins we were using weren't reading right. This came from two problems, once with how the keypad was wired up to the ESP32, and also how the code was set up. With wiring the keypad, at first we used pins 34, 35, 32, and 33 for the rows, and pins 25, 26, and 27 for the columns. This was a problem due to pins 34 - 39 have no internal configurable pullup/pulldown. This means that if I were to set one of these pins to high, then back to low, since the pin has no reference to ground in the ESP32, then it would stay high. This problem was solved after moving the keypad to pins 32, 33, 25, 26, 27, 14, and 15.

With the code portion of the keypad, the problem with this was that, if a button was pressed, regardless of what column it belonged to, the return value would be on column one. This happened because if the method to find the pressed button was called, and it failed to actually find which button was pressed, it would return a random character from the first column. This problem was solved by encasing the main body of the method in an if statement that if a button wasn't determined, then a space character was returned from the method. In the main loop, this would force the returned value to be ignored and for the main loop to loop around again.

Getting the garage to not open and close constantly when the UID was read multiple times came when we combined our hardware and software code. We thought the issue could be the garage door driver reading in the input while the code was in the opening/closing loop, so we tried to get the garage door driver to only read inputs when the garage door was stationary. That did not work because MQTT has a sort of queue for topics and payloads to be processed when the process in front of it has finished. Instead, we took a look at the hardware code and read in the NFC card in a larger interval than one second.

The last problem that we have is that Freeboard sometimes crashes when reading in information from MQTT. This is not a problem we can fix, but we were able to demonstrate that all of Freeboard works.

Testing

To validate that everything was working properly, we tested our project goals of making sure that all inputs could be read in, which we confirmed through MQTT, validated that the functions of the LED, and freeboard was working when the ultrasonic sensor distance read in a valid distance

and the correct UID and password were entered. Then we tested an invalid distance with a valid UID and password, a valid distance with invalid UID and password. Through doing all of this, we tested each function of the keypad by clearing and setting a new password.

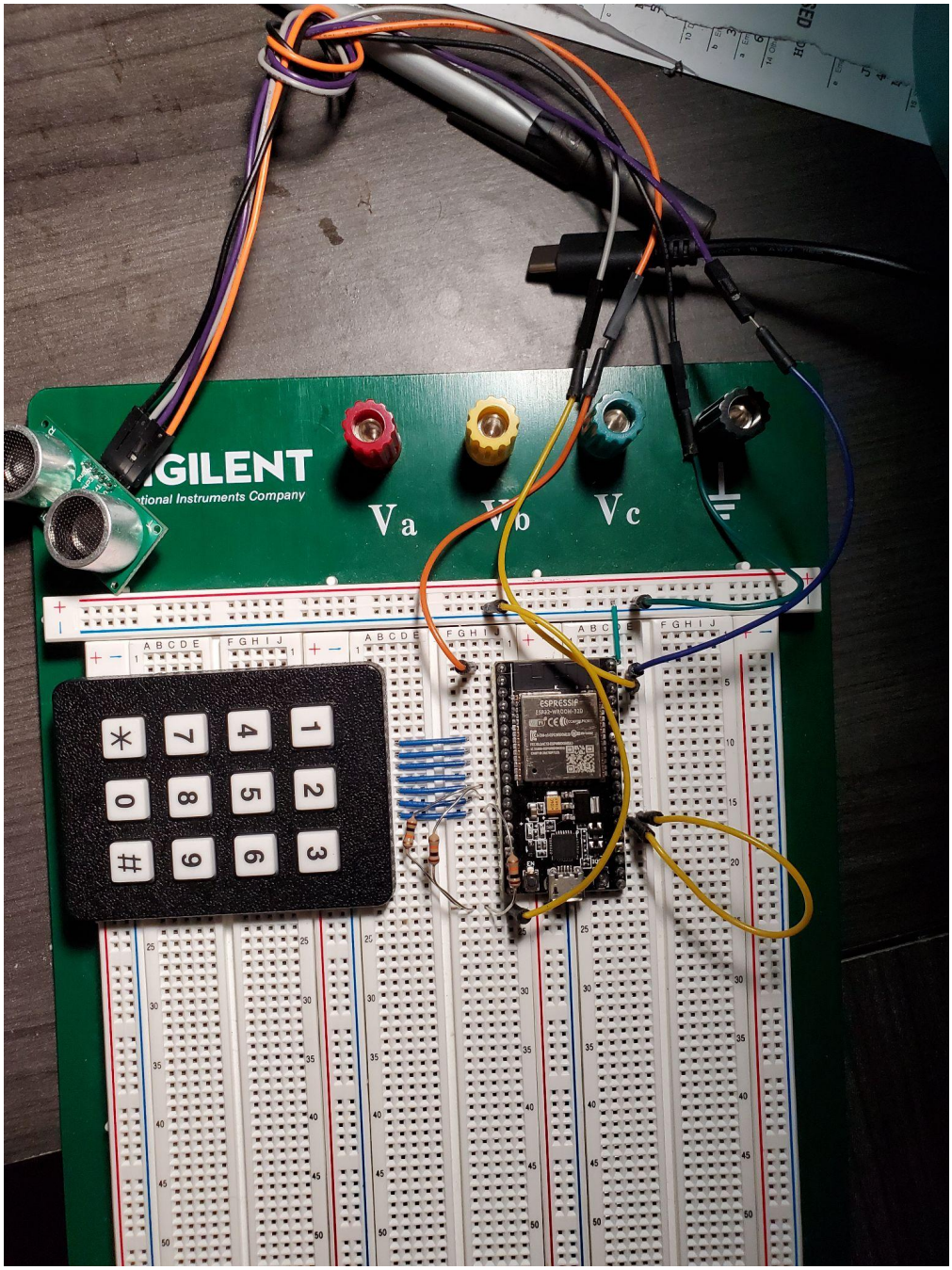
Conclusion

In conclusion, we successfully completed our project goals and made a working product. To improve this we could have made it more secure by having each UID have its correlated password that also had to be put in, at least when the garage was closed. Also, we could have used an app that was hosted by our Raspberry Pi that allowed us to open and close the garage from our phone, without using UID or password, if someone needed in the garage, but forgot their password and UID or if someone wanted to close the garage from the couch.

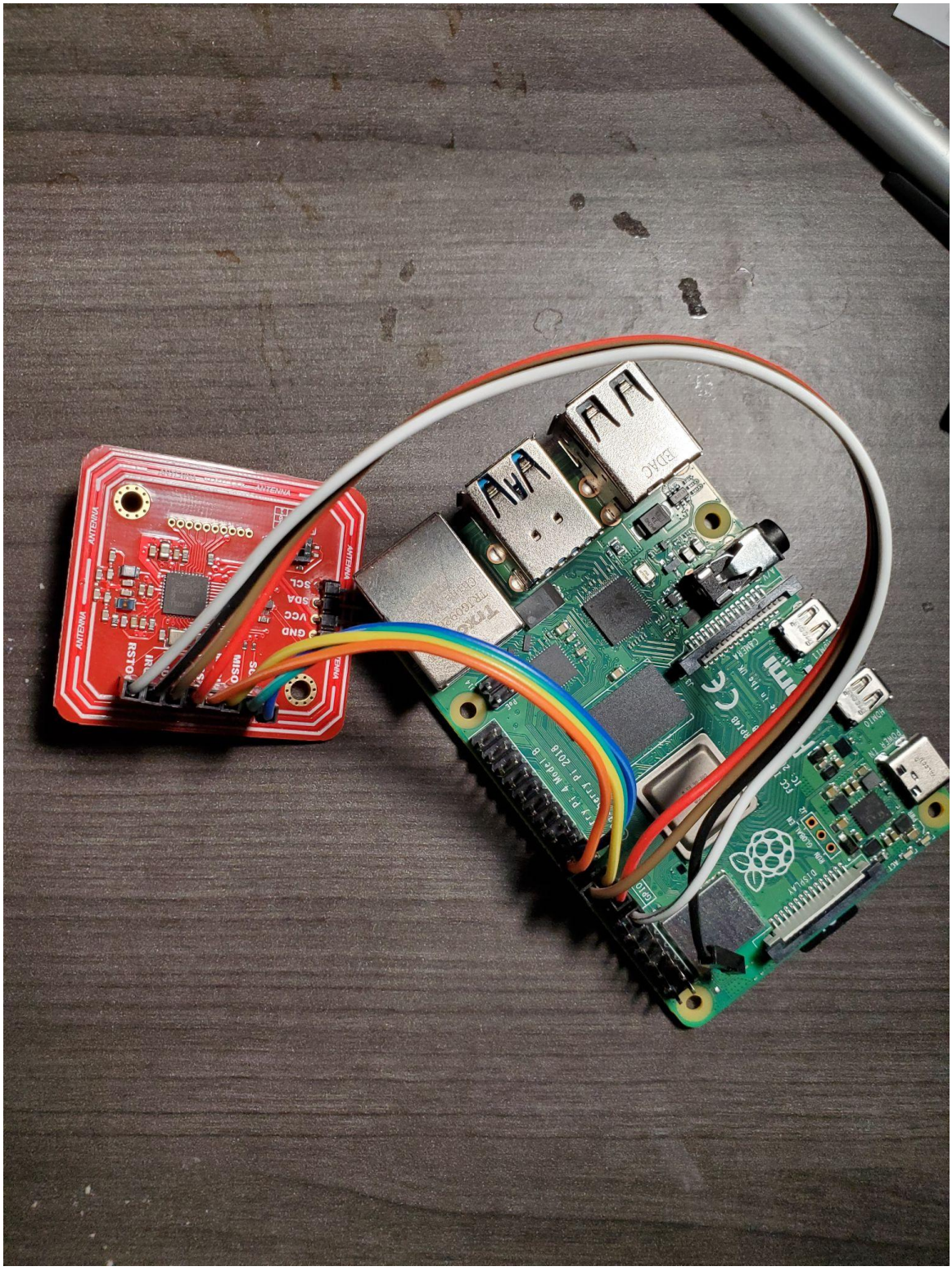
Appendix

Appendix A: Github Repository
https://gitlab.beocat.ksu.edu/s21.garage-full-o-debt/finallab

Appendix B: Input Hardware Setup - ESP32 / Ultrasonic Sensor / Keypad



Appendix C: Raspberry Pi 4 / NFC Board



Appendix D: Input Code - ESP32

```
#include <ArduinoJson.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <CircularBuffer.h>

/* Define pins */
#define trigger (23)
#define echo (22)
#define r1 (32)
#define r2 (33)
#define r3 (25)
#define r4 (26)
#define c1 (27)
#define c2 (14)
#define c3 (12)

/* Set up wifi constants */
const char* ssid = "ece631Lab";
const char* password = "esp32IOT!";
const char* mqtt_server = "192.168.1.126";

//const char* ssid = "ATTsEpqNuI";
//const char* password = "sps?awqe4udy";
//const char* mqtt_server = "192.168.1.126";

/* Initiate wifi clients */
WiFiClient espClient;
PubSubClient client(espClient);
DynamicJsonDocument doc(1024);
DynamicJsonDocument pass(1024);
DynamicJsonDocument newPass(1024);
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE  (50)
char msg[MSG_BUFFER_SIZE];

int PWM_FREQUENCY = 16; // Defines the frequency
int PWM_CHANNEL = 0;    // Selects the channel number
int PWM_RESOLUTION = 8; // Defines the resolution of the signal
int dutyCycle = 1;      // Defines the width of the signal
volatile byte state = LOW;
volatile byte prevState = LOW;
float distance = 0;
float totalDistance = 0;
void ISR() { state = !state;}

/* Moving Average Filter Initiate */
const int sampleSize = 11;
CircularBuffer<float, sampleSize> Buffer;

float start;
float finish;


unsigned long printStart = millis();
unsigned long printFinish;
unsigned long buttonStart = millis();
unsigned long buttonFinish;
unsigned long buffStart = millis();
unsigned long buffFinish = 0;
```

```
unsigned long inputStart = millis();
unsigned long inputFinish = 0;

int keyIndex = 0;

bool buttonPressed = false;

String passcode;
String newPasscode;
bool isNewPass = false;

void setup_wifi()
{

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect()
{
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP32Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str()))
    {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("outTopic", "hello world");
      // ... and resubscribe
      client.subscribe("/ece631/Lab4Python/NFC/UID");
    }

    else
    {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

```
char keypad()
{
  char keyVal;
  bool valCheck = false;

  // Row 1
  digitalWrite(r1, HIGH);
  if (digitalRead(c1) == HIGH)
  {
    keyVal = '1';
    valCheck = true;
  }
  else if (digitalRead(c2) == HIGH)
  {
    keyVal = '2';
    valCheck = true;
  }
  else if (digitalRead(c3) == HIGH)
  {
    keyVal = '3';
    valCheck = true;
  }

//  Serial.print("r1-> ");
//  Serial.print(digitalRead(c1));
//  Serial.print(digitalRead(c2));
//  Serial.println(digitalRead(c3));
  digitalWrite(r1, LOW);

  // Row 2
  digitalWrite(r2, HIGH);
  if (digitalRead(c1) == HIGH)
  {
    keyVal = '4';
    valCheck = true;
  }
  else if (digitalRead(c2) == HIGH)
  {
    keyVal = '5';
    valCheck = true;
  }
  else if (digitalRead(c3) == HIGH)
  {
    keyVal = '6';
    valCheck = true;
  }
//  Serial.print("r2-> ");
//  Serial.print(digitalRead(c1));
//  Serial.print(digitalRead(c2));
//  Serial.println(digitalRead(c3));
  digitalWrite(r2, LOW);

  // Row 3
  digitalWrite(r3, HIGH);
  if (digitalRead(c1) == HIGH)
  {
    keyVal = '7';
    valCheck = true;
  }
  else if (digitalRead(c2) == HIGH)
  {
```

```
    keyVal = '8';
    valCheck = true;
  }
  else if (digitalRead(c3) == HIGH)
  {
    keyVal = '9';
    valCheck = true;
  }
//  Serial.print("r3-> ");
//  Serial.print(digitalRead(c1));
//  Serial.print(digitalRead(c2));
//  Serial.println(digitalRead(c3));
  digitalWrite(r3, LOW);

  // Row 4
  digitalWrite(r4, HIGH);
  if (digitalRead(c1) == HIGH)
  {
    keyVal = '*';
    valCheck = true;
  }
  else if (digitalRead(c2) == HIGH)
  {
    keyVal = '0';
    valCheck = true;
  }
  else if (digitalRead(c3) == HIGH)
  {
    keyVal = '#';
    valCheck = true;
  }
//  Serial.print("r4-> ");
//  Serial.print(digitalRead(c1));
//  Serial.print(digitalRead(c2));
//  Serial.println(digitalRead(c3));
  digitalWrite(r4, LOW);

  if (!valCheck)
    keyVal = ' ';

  return keyVal;
}

void setup()
{
  Serial.begin(9600);
  delay(100);
  pinMode(trigger, OUTPUT);
  pinMode(echo, INPUT_PULLUP);
  pinMode(r1, OUTPUT);
  pinMode(r2, OUTPUT);
  pinMode(r3, OUTPUT);
  pinMode(r4, OUTPUT);
  pinMode(c1, INPUT);
  pinMode(c2, INPUT);
  pinMode(c3, INPUT);
  digitalWrite(r1, LOW);
  digitalWrite(r2, LOW);
  digitalWrite(r3, LOW);
  digitalWrite(r4, LOW);
  attachInterrupt(digitalPinToInterrupt(echo), ISR, CHANGE);
  ledcSetup(PWM_CHANNEL, PWM_FREQUENCY, PWM_RESOLUTION);
```

```
  ledcAttachPin(trigger, PWM_CHANNEL );
  ledcWrite(PWM_CHANNEL, dutyCycle);

  Serial.setTimeout(100);
  Serial.println("ECE631 Final Lab");

  for (int i = 0; i < sampleSize; i++)
    Buffer.unshift(0);

  setup_wifi();
  client.setServer(mqtt_server, 1883);
}

void loop()
{

  if (!client.connected())
  {
    reconnect();
  }

  client.loop();

  printFinish = millis();

  digitalWrite(trigger, state);

  if (state == HIGH && prevState == LOW)
  {
    buffStart = millis();
    prevState = HIGH;

  }


  else if (state == LOW && prevState == HIGH)
  {
    buffFinish = millis() - buffStart;
    prevState = LOW;

    distance = (buffFinish / 1000.0) * (13503.9 / 2.0)/(float)sampleSize;  // distance in
inches
    Buffer.unshift(distance);

    totalDistance = 0;

    for (int i = 0; i < sampleSize; i++)
      totalDistance = Buffer[i] + totalDistance;

  }

  unsigned long printTime = printFinish - printStart;

  if (printTime >= 1000)
  {
    Serial.print("Distance: ");
    Serial.print(totalDistance);
    Serial.println(" in");

    doc["Distance"] = totalDistance;
    doc["units"] = "inches";
```

```
    String payload;
    serializeJson(doc, payload);

    client.publish("/ece631/Final/Input", payload.c_str());

    printStart = millis();
  }

  digitalWrite(r1, HIGH);
  digitalWrite(r2, HIGH);
  digitalWrite(r3, HIGH);
  digitalWrite(r4, HIGH);

  buttonFinish = millis();

  if ((digitalRead(c1) == HIGH || digitalRead(c2) == HIGH || digitalRead(c3) == HIGH) &&
buttonPressed == false)
  {

    digitalWrite(r1, LOW);
    digitalWrite(r2, LOW);
    digitalWrite(r3, LOW);
    digitalWrite(r4, LOW);

    buttonPressed = true;
    char keyVal = keypad();

    if (keyVal != ' ')
    {
      Serial.print("KeyVal = ");
      Serial.println(keyVal);
    }

    if (keyVal == ' ')
      buttonPressed = false;

    else if (keyVal == '*')
    {
      passcode = "";
      newPasscode = "";
      keyIndex = 0;
      isNewPass = false;

      Serial.println("-----Reset To Default Input-----");
    }

    else if (keyVal == '#')
    {
      isNewPass = true;
      keyIndex = 0;
      passcode = "";
      newPasscode = "";
      Serial.println("-----Reset To New Password Input-----");
    }

    else if (isNewPass)
    {
      newPasscode += keyVal;
      keyIndex++;
    }

    else if (keyIndex < 6)
```

```
    {
      passcode += keyVal;
      keyIndex++;
    }

    else if(keyIndex >= 6)
      keyIndex = 6;
  }

  else if (digitalRead(c1) == LOW && digitalRead(c2) == LOW && digitalRead(c3) == LOW &&
buttonPressed == true)
  {
    buttonStart = millis();
    buttonPressed = false;
  }

  inputFinish = millis();

  if (keyIndex >= 6 && inputFinish >= inputStart + 5000)
  {
    keyIndex = 0;

    if (!isNewPass)
    {
      pass["Password"] = passcode;

      String payload;
      serializeJson(pass, payload);
      client.publish("/ece631/Final/Input", payload.c_str());
      Serial.print("Password: ");
      Serial.println(passcode);

      passcode = "";
    }

    else
    {
      newPass["New Password"] = newPasscode;

      String payload;
      serializeJson(newPass, payload);
      client.publish("/ece631/Final/Input", payload.c_str());
      Serial.print("New Password: ");
      Serial.println(newPasscode);

      newPasscode = "";
      isNewPass = false;
    }

    inputStart = millis();
  }

  else if (keyIndex > 0 && keyIndex <= 6 && inputFinish <= inputStart + 5000)
  {
    keyIndex = 0;
    passcode = "";
    newPasscode = "";

    Serial.println("Invaid Input. Garage door opening or closing.");
  }
}
```

Appendix E: Python MQTT Code - Raspberry Pi 4

```
#! /usr/bin/env python3
# ECE 631 Spring 2021
# Author:
# Demo of NFC UID get.
#
"""
This example shows connecting to the PN532 with I2C (requires clock
stretching support), SPI, or UART. SPI is best, it uses the most pins but
is the most reliable and universally supported.
After initialization, try waving various 13.56MHz RFID cards over it!
"""

import time
import datetime
import binascii
import RPi.GPIO as GPIO
import pn532
import json
import MQTTClients

MQTTPub = MQTTClients.MQTTPusher('127.0.0.1', 1883)
curTime = time.time()
actionTime = time.time()

if __name__ == '__main__':
    try:
        PN532 = pn532.PN532_SPI(debug=False, reset=22, cs=8)

        ic, ver, rev, support = PN532.get_firmware_version()
        print('Found PN532 with firmware version: {0}.{1}'.format(ver, rev))

        # Configure PN532 to communicate with MiFare cards
        PN532.SAM_configuration()

        print('Waiting for RFID/NFC card...')
        while True: #Main Loop
            try:
                # Check if a card is available to read with 100ms timeout
                uid = PN532.read_passive_target(timeout=0.1)
                elapTime = time.time()

                if elapTime >= curTime + 1:
                    curTime = time.time()
                    if uid is None:
                        pass
                    elif curTime >= actionTime + 5:
                        topic2 = "/ece631/Final/Input"
                        payload2 = {"UID":binascii.hexlify(uid).decode('utf-8')}
                        MQTTPub.PushData(topic2, json.dumps(payload2))
                        actionTime = time.time()
                    else:
                        pass

                # No card is available.
                if uid is None:
                    continue

                print('Found card with UID: %s'%(binascii.hexlify(uid).decode('utf-8')))

            except Exception as e:
```

```
            print("Loop Exception: %s"%e)
    except Exception as e:
        print("Main Exception: %s"%e)
    finally:
        GPIO.cleanup()
```

Appendix F: Garage Door Driver

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>

#define LEDBLUE 2 //sets the LED

const char* ssid = "ece631Lab";
const char* password = "esp32IOT!";
const char* mqtt_server = "192.168.1.126";
WiFiClient espClient;
PubSubClient client(espClient);
#define MSG_BUFFER_SIZE  (100)
char msg[MSG_BUFFER_SIZE];
unsigned long timer; //timer controlling the blinking of the LED
String garage = "Closed"; //The original state of the garage is closed
const char* code; //the input code
String correct_code = "000000"; //the correct pin input that is accepted
const char* uid; //the keycard number
String correct_uid1 = "897a9bb3"; //Card 1 that can be accepted
String correct_uid2 = "c94ca5b3"; //Card 2 that can be accepted
bool LEDState; //state of the LED to flash it when opening/closing
int count = 0; //count to open/close garage for 10 seconds
float distance; //distance for Ultrasonic Sensor. DECIDE IF FEET or INCHES
bool read_msg = false; //variable stating if there was a message read in to
make sure that the code does not constantly run something we don't want it to
(i.e. the error checking code)
bool read_uid = false;
bool read_code = false;
bool cread = false;

void setup_wifi() {
```

```
  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  DynamicJsonDocument doc(1024);
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  for (int i = 0; i < length; i++) {
    msg[i] = ((char)payload[i]);
  }
  deserializeJson(doc,msg);

  JsonObject obj = doc.as<JsonObject>();

  //Takes the UID, Code, and Distance from the MQTT topics and sets their
respective payloads
  if(obj.containsKey("Password") && cread == false)
  {
    code = obj["Password"];
    read_code = true;
    cread = true;
  }
  else if(obj.containsKey("UID") && cread == false)
  {
    uid = obj["UID"];
    read_uid = true;
    cread = true;
```

```
  }
  if(obj.containsKey("Distance"))
  {
    distance = (float)obj["Distance"];
  }
  if(obj.containsKey("New Password"))
  {
    correct_code = (int)obj["New Password"];
    client.publish("/ece631/Final/Garage", "Password Successfully Changed");
  }
  read_msg = true;
}

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP32Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      //client.publish("/ece631/Final/Input", "Hello World");
      // ... and resubscribe
      client.subscribe("/ece631/Final/Input");
    }else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {
  pinMode(LEDBLUE, OUTPUT);
  Serial.begin(9600);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  timer = millis();
  digitalWrite(LEDBLUE, LOW);
  LEDState = false;
}

void loop() {
```

```
  // put your main code here, to run repeatedly:
  DynamicJsonDocument doc(1024);
  if (!client.connected()) {
    reconnect();
  }
  client.loop();


  if((((String)uid == correct_uid1 || (String)uid == correct_uid2) ||
(String)code == correct_code) && ((float)distance < 2.0 || (float)distance >
5.0))
  {
    //Publish saying if garage is opening/closing
    if(garage == "Closed")
    {
      //Sends message saying the garage is opening
      doc["Garage"] = "Opening";
      serializeJson(doc, Serial);
      serializeJson(doc, msg);
      client.publish("/ece631/Final/Garage", msg);
      cread = false;

      while(count < 5)
      {
        //Blinks LED every second, the time it takes to open garage.
        if ((millis() - timer + 0xFFFF) % 0xFFFF >= 1000)
        {
          timer = millis();
          LEDState = LEDState ^ HIGH;
          digitalWrite(LEDBLUE, LEDState);
          count++;
        }
      }

      //Sends message saying the garage is now Open
      doc["Garage"] = "Open";
      serializeJson(doc, Serial);
      serializeJson(doc, msg);
      client.publish("/ece631/Final/Garage", msg);
      garage = "Open";
      digitalWrite(LEDBLUE, HIGH);
      count = 0;

    }
    else if(garage == "Open")
    {
      //Sends message saying the garage is closing
      doc["Garage"] = "Closing";
      serializeJson(doc, Serial);
```

```
      serializeJson(doc, msg);
      client.publish("/ece631/Final/Garage", msg);
      cread = false;

      while(count < 5)
      {
        //Blinks LED every second, the time it takes to close garage.
        if ((millis() - timer + 0xFFFF) % 0xFFFF >= 1000)
        {
          timer = millis();
          LEDState = LEDState ^ HIGH;
          digitalWrite(LEDBLUE, LEDState);
          count++;
        }
      }

      //Sends message saying the garage is now closed
      doc["Garage"] = "Closed";
      serializeJson(doc, Serial);
      serializeJson(doc, msg);
      client.publish("/ece631/Final/Garage", msg);
      garage = "Closed";
      digitalWrite(LEDBLUE, LOW);
      count = 0;
    }


  }
  else //One or more of the conditions is not met
  {
    if(read_msg == true)
    {
      if((String)code != "" && (String)code != correct_code)
      {
        doc["Password"] = "Incorrect Password";
      }
      if((String)uid != "" && (String)uid != correct_uid1 && (String)uid !=
correct_uid2)
      {
        doc["UID"] = "Incorrect UID";
      }
      if((float)distance >= 2.0 && (float)distance <= 5.0)
      {
        doc["Distance"] = "Car is not clear of the garage door";
      }
      serializeJson(doc, Serial);
      serializeJson(doc, msg);
      if(read_uid == true || read_code == true)
      {
```

```
        client.publish("/ece631/Final/Garage", msg);
      }
      cread = false;
    }
  }
  read_msg = false;
  uid = "";
  code = "";
  read_uid = false;
  read_code = false;
}
```