EECS 113

PROCESSOR HARDWARE/SOFTWARE INTERFACES

Alexa 5.0

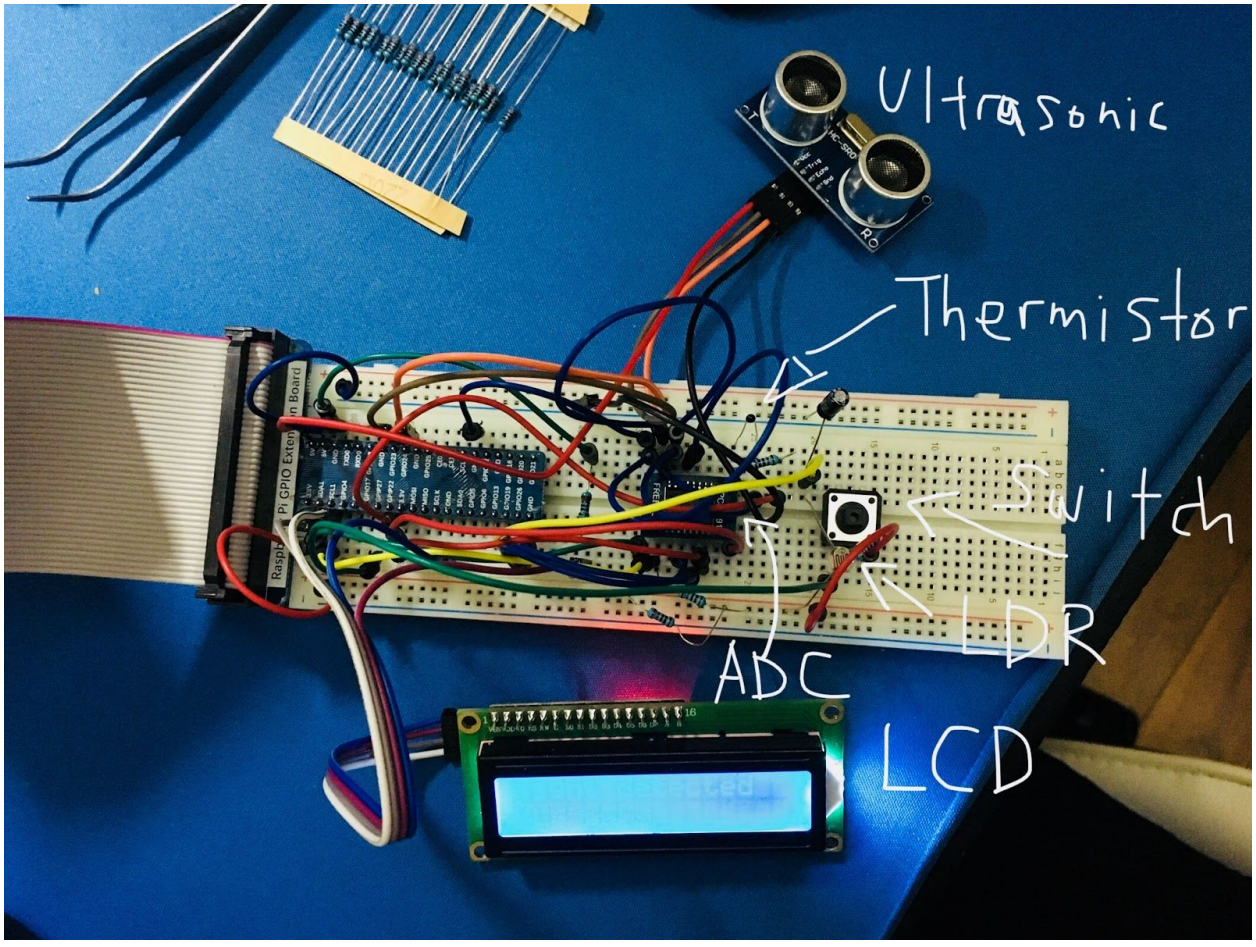
Brandon Ngo, #13786834

OBJECTIVE

The objective of this project is to create a IoT device using the Raspberry Pi 3 B Plus that has two modes. The main purpose of this device ss to act as a simple smart home controller. It utilizes the LCD, ADC with an input from the thermistor, a Light Dependent Resistor (LDR), ultrasonic sensor, a switches, and the internet.

In the normal mode also known as the in-home mode, the Raspberry Pi measures the temperature and displays it on the LCD. It also displays other information such as the clock and the date.

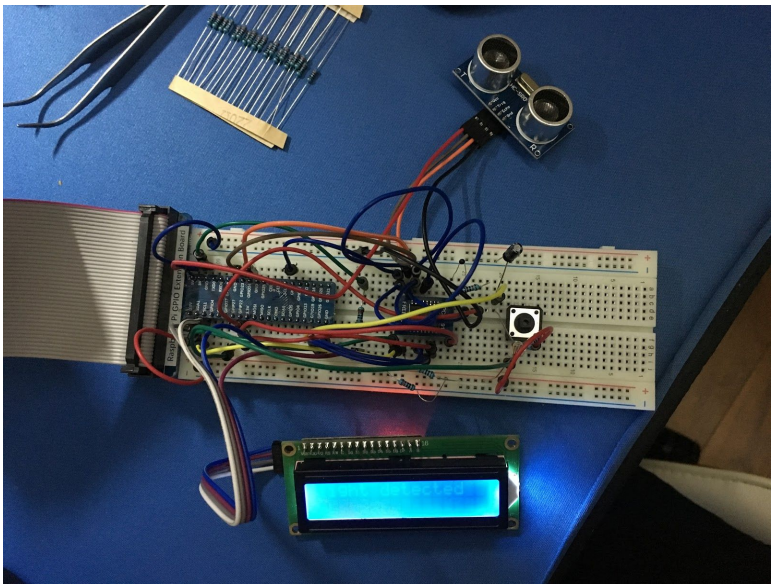
In the security mode also known as the out-of-home mode, Raspberry Pi constantly reads data from the LDR and the ultrasonic sensor to detect any suspicious events such as someone crossing the ultrasonic sensor or a light is turned on. If any suspicious event is detected, the Raspberry Pi sends an email reporting this suspicious event to the user.

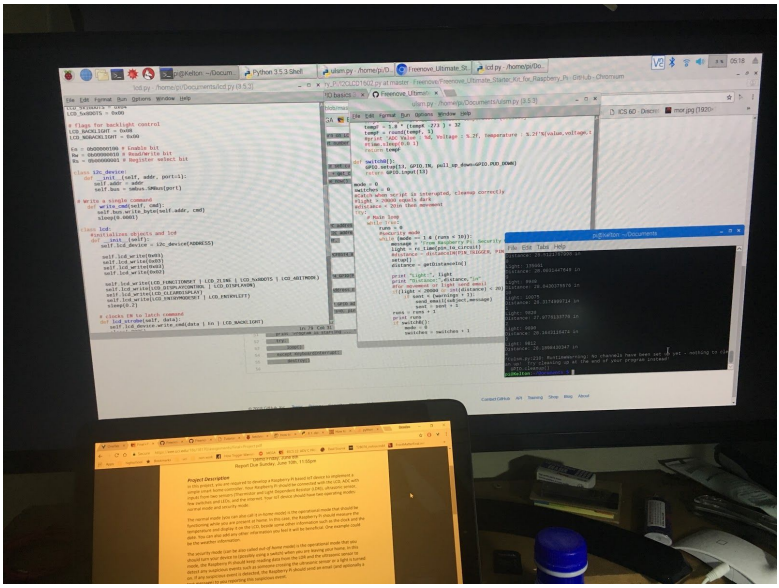


METHODOLOGY

// Explain in short how project was realized.
// Support with images and circuit diagrams (photo is ok).
// Add the brief explanations of the code.
The frenove tutorial pdf was referenced in my project

The projected was created with GPIO extension and the ribbon cable as well as a long bread board. The Raspberry Pi as connected to a monitor.





These are all the libraries that I imported. I needed the basic GPIO, time/datetime (for date, time, and sleep(), smtplib for sending emails, math, smbus for the LCD, the LCD file, and the imaplib which read emails so that the Pi could take commands.

```
import RPi.GPIO as GPIO
import time
import smtplib
import math
import smbus
from datetime import datetime
from lcd import lcd
import imaplib
```

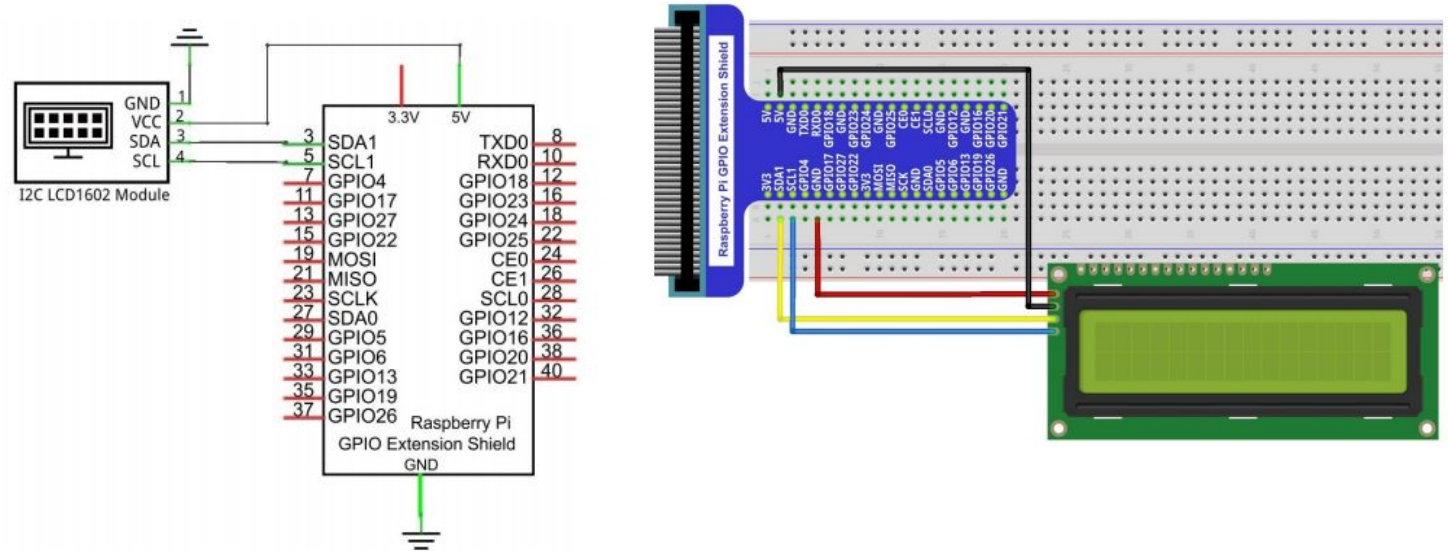
The main function has an infinite while loop with two nested while loops. The nest loops determine the mode it is in an executes the body accordingly. Mode = 0 is the in home mode and mode = 1 is the out of home mode. In the in home mode I call on the temp() (using the thermistor) and datetime.now() function and display the temp and date/time on the LCD. There is a switch/button to

change modes. If the button is held down at the end of the cycle it will change mode to the opposite and exit the nested loop to enter to other nested loop.

In the security mode I display ‘Security mode’ on the lcd and then check for the LDR values. If the values are over 20000 cycles (explained later) it is considered dark. If it is under then there is light and thus the Pi will send an email. If the ultrasonic sense anything closer than 20 inches it will also send an email saying that security is compromised. These messages were also displayed on the LCD. I also was able to use imaplib to get the first email in the inbox and get all the contents. Code below:

```
try:
    # Main loop
    while True:
        runs = 0
        sent = 1
        #security mode
        while (mode == 1 & (runs < 10)):
            lcd.lcd_display_string(lcd(), 'Security Mode',1)
            message = 'From Raspberry Pi: Security Compromised ' + str(sent)
            light = rc_time(pin_to_circuit)
            if light > 20000:
                lightprint = 'It is dark'
            else:
                lightprint = 'Light has been shown'
            #distance = distanceIN(PIN_TRIGGER, PIN_ECHO)
            setup()
            distance = getDistanceIn()

            print "Light:", light, ' ',lightprint
            print "Distance:",distance,"in"
            #for movement or light send email
            if(light < 20000 or int(distance) < 20):
                if light < 20000:
                    print 'light detected'
                    lcd.lcd_display_string(lcd(), 'light detected',1)
                    time.sleep(.5)
                    lcd.lcd_display_string(lcd(), 'Email Sent',1)
                elif int(distance) < 20:
                    print 'movement detected'
                    lcd.lcd_display_string(lcd(), 'movement detected',1)
                    time.sleep(.5)
                    lcd.lcd_display_string(lcd(), 'Email Sent',1)
                if sent < (warnings + 1):
                    send_email(subject,message)
                    lcd.lcd_display_string(lcd(), 'Email Sent',1)
                    sent = sent + 1
            runs = runs + 1
            print runs, '\n'
            if switchB():
                mode = 0
                switches = switches + 1

        #normal mode
        while (mode == 0 & (runs < 10)):
            print 'Temp: ', temp(), ' F'
            print datetime.now()
            lcd.lcd_display_string(lcd(),str(datetime.now()),1)
            time.sleep(1)
            lcd.lcd_display_string(lcd(),'Temp: ' + str(temp()) + ' F',2)
            time.sleep(1)
            runs = runs + 1
            print runs
            if switchB():
                mode = 1
                switches = switches + 1


except KeyboardInterrupt:
    GPIO.cleanup()
finally:
    GPIO.cleanup()
```

I first started working on one mode at a time. My partner decided that he could not work on the project a week before the deadline so it was quite a challenge. There are two files in my project, the main code that includes all the functions for the components and a seperate one for the LCD because the LCD was complicated. I started with the in home mode first (particularly the LCD) because I saw that the LCD would be a crucial component for both modes. I referenced the Frenove pdf for the diagrams and connections. The LCD1602 module was included in the kit and it consolidated the 16 pins into 4 pins. The 4 pins (VCC -> 5V) were connected to the matching labels on the board.

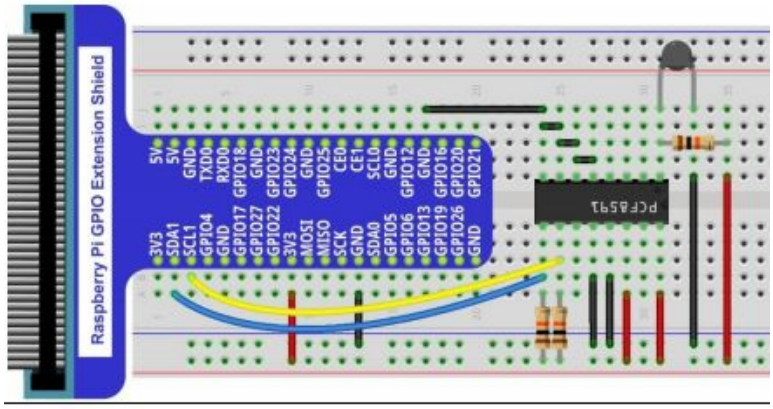


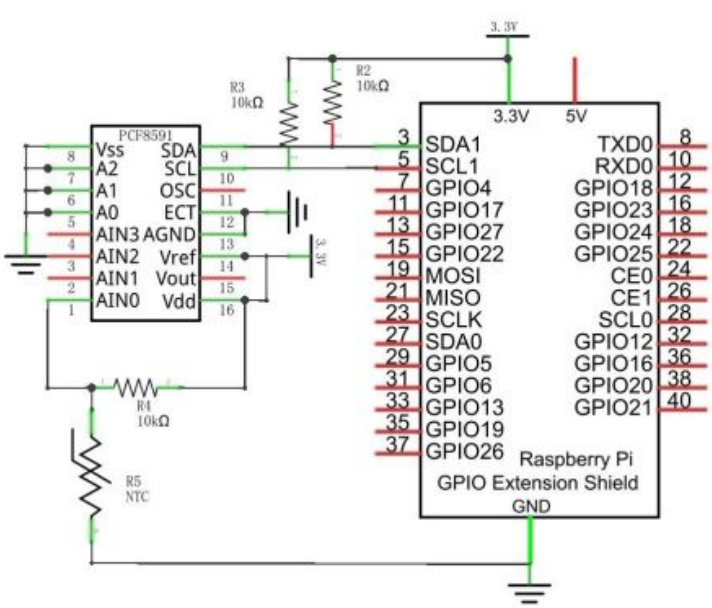
I made a separate file for the LCD as it was complicated. Here SMbus was import to use the I2C functions. Also I2C is enabled on the Pi. The address of the LCD is x3F and there are flags that reference the address of for each condition. First the i2c device and LCD is initialized. The i2c device writes the byte of the cmd and the lcd writes the data that is inputted as an argument. The main function of the LCD is the lcd_display_string which takes in a string and a line number. For the purpose of the project used line 1 and 2 and was able to display and string that fit in the 16 character display.
Code below:

```
import smbus
from time import *

# LCD Address
ADDRESS = 0x3f

# commands
LCD_CLEARDISPLAY = 0x01
LCD_RETURNHOME = 0x02
LCD_ENTRYMODESET = 0x04
LCD_DISPLAYCONTROL = 0x08
LCD_CURSORSHIFT = 0x10
LCD_FUNCTIONSET = 0x20
LCD_SETCGRAMADDR = 0x40
LCD_SETDDRAMADDR = 0x80

# flags for display entry mode
LCD_ENTRYRIGHT = 0x00
LCD_ENTRYLEFT = 0x02
LCD_ENTRYSHIFTINCREMENT = 0x01
LCD_ENTRYSHIFTDECREMENT = 0x00

# flags for display on/off control
LCD_DISPLAYON = 0x04
LCD_DISPLAYOFF = 0x00
LCD_CURSORON = 0x02
LCD_CURSOROFF = 0x00
LCD_BLINKON = 0x01
LCD_BLINKOFF = 0x00

# flags for display/cursor shift
LCD_DISPLAYMOVE = 0x08
LCD_CURSORMOVE = 0x00
LCD_MOVERIGHT = 0x04
LCD_MOVELEFT = 0x00

# flags for function set
LCD_8BITMODE = 0x10
LCD_4BITMODE = 0x00
LCD_2LINE = 0x08
LCD_1LINE = 0x00
LCD_5x10DOTS = 0x04
LCD_5x8DOTS = 0x00

# flags for backlight control
LCD_BACKLIGHT = 0x08
LCD_NOBACKLIGHT = 0x00

En = 0b00000100 # Enable bit
Rw = 0b00000010 # Read/Write bit
Rs = 0b00000001 # Register select bit
```

```
class i2c_device:
    def __init__(self, addr, port=1):
        self.addr = addr
        self.bus = smbus.SMBus(port)

 # Write a single command
    def write_cmd(self, cmd):
        self.bus.write_byte(self.addr, cmd)
        sleep(0.0001)

class lcd:
    #initializes objects and lcd
    def __init__(self):
        self.lcd_device = i2c_device(ADDRESS)

        self.lcd_write(0x03)
        self.lcd_write(0x03)
        self.lcd_write(0x03)
        self.lcd_write(0x02)

        self.lcd_write(LCD_FUNCTIONSET | LCD_2LINE | LCD_5x8DOTS | LCD_4BITMODE)
        self.lcd_write(LCD_DISPLAYCONTROL | LCD_DISPLAYON)
        self.lcd_write(LCD_CLEARDISPLAY)
        self.lcd_write(LCD_ENTRYMODESET | LCD_ENTRYLEFT)
        sleep(0.2)

    # clocks EN to latch command
    def lcd_strobe(self, data):
        self.lcd_device.write_cmd(data | En | LCD_BACKLIGHT)
        sleep(.0005)
        self.lcd_device.write_cmd(((data & ~En) | LCD_BACKLIGHT))
        sleep(.0001)

    def lcd_write_four_bits(self, data):
        self.lcd_device.write_cmd(data | LCD_BACKLIGHT)
        self.lcd_strobe(data)

    # write a command to lcd
    def lcd_write(self, cmd, mode=0):
        self.lcd_write_four_bits(mode | (cmd & 0xF0))
        self.lcd_write_four_bits(mode | ((cmd << 4) & 0xF0))

    # put string function
    def lcd_display_string(self, string, line):
        if line == 1:
            self.lcd_write(0x80)
        if line == 2:
            self.lcd_write(0xC0)
        if line == 3:
            self.lcd_write(0x94)
        if line == 4:
            self.lcd_write(0xD4)

        for char in string:
            self.lcd_write(ord(char), Rs)

    # clear lcd and set to home
    def lcd_clear(self):
        self.lcd_write(LCD_CLEARDISPLAY)
        self.lcd_write(LCD_RETURNHOME)
```

For the thermistor I referenced the PDF. I simply read the data from the bus and got the ADC(PCF8591) address. Voltage is the value / 255 * 3.3. Resistance is voltage * 10 / (3.3V - voltage). The temperature Kelvins is 1/(1/(273.15 + 25)) + math.log(Resistance/10)/3950). The relation of Fahrenheit is 1.8 * Kelvin + 32.

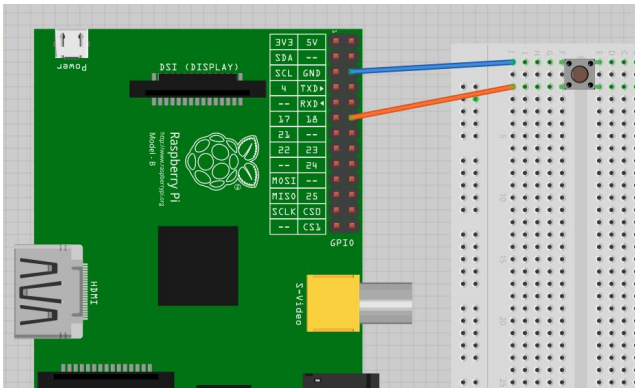





```
def temp():
    value = bus.read_byte_data(address,cmd+chn)
    voltage = value / 255.0 * 3.3 #calculate voltage
    Rt = 10 * voltage / (3.3 - voltage) #calculate resistance value of thermistor
    tempK = 1/(1/(273.15 + 25) + math.log(Rt/10)/3950.0) #calculate temperature (Kelvin)
    tempC = tempK -273.15 #calculate temperature (Celsius)
    tempF = 1.8 * (tempK -273 ) + 32
    tempF = round(tempF, 1)
    #print 'ADC Value : %d, Voltage : %.2f, Temperature : %.2f'%(value,voltage,tempC)
    #time.sleep(0.01)
    return tempF
```

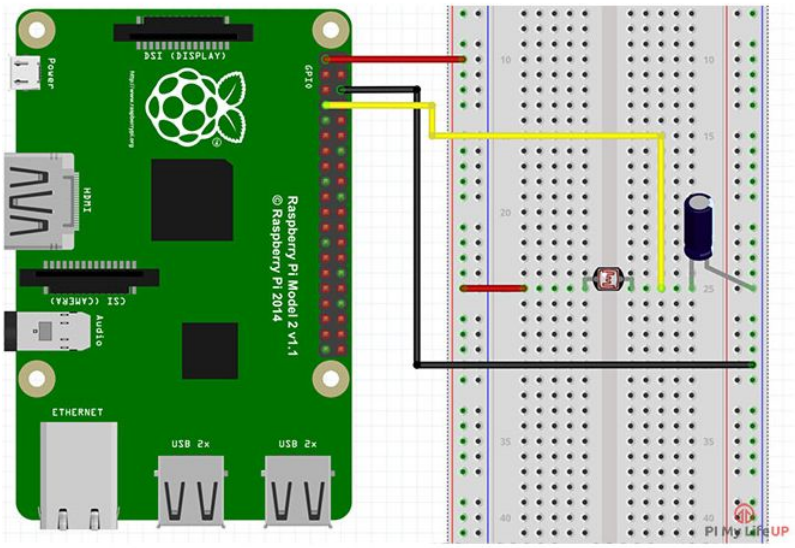
To switch modes I made a pull down circuit and reading it from a pin. Pushing down completed the circuit and using the value, the mode switched. In this case I assigned it to 13 and also put a resistor in series

```
def switchB():
    GPIO.setup(13, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    return GPIO.input(13)
```



Now I talk about security mode.

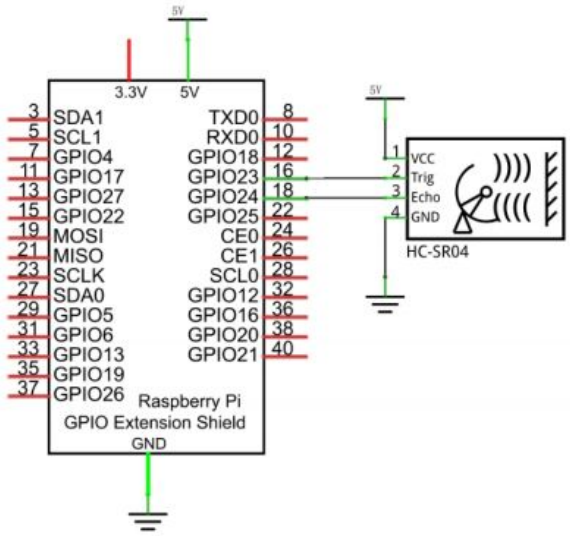
I tackled the LDR and light sensing function. I opted to not use the ADC and instead used a method where I would calculate if it was light or dark using the time is takes to charge to capacitor. This is the analog way of doing this because I had already finished this part before finding out I could have just used the ADC. The thermistor will have a resistance of only a few hundred ohms in the light while it can have a resistance of several megohms in the dark. Using the capacitor in series with the LDR, I was able to determine how much resistance the LDR is giving out to see it it was light or dark. The function used is the rc_time which takes the GPIO pin number and outputs a count. It goes into a while loop and increments count until the pin goes to high, this is when the capacitor charges to about 3/4. Once it goes high, we return the count value to the main function. Values of over 20000 cycles I considered high which meant that it took a long time for the capacitor to charge so that meant its dark.

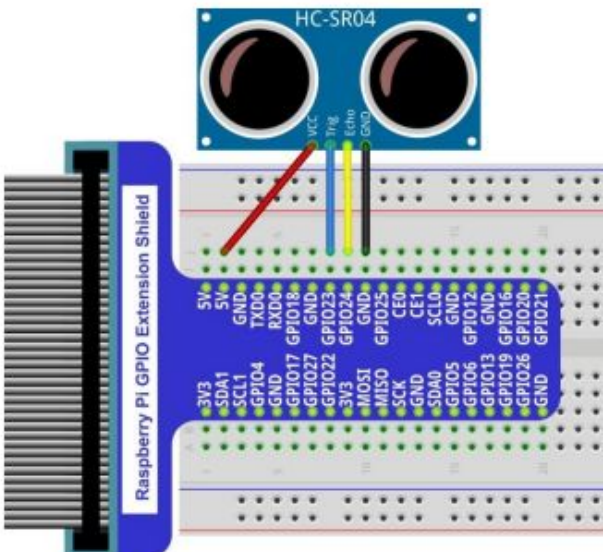



```
def rc_time (pin_to_circuit):
    count = 0

    #Output on the pin for
    GPIO.setup(pin_to_circuit, GPIO.OUT)
    GPIO.output(pin_to_circuit, GPIO.LOW)
    time.sleep(0.1)

    #Change the pin back to input
    GPIO.setup(pin_to_circuit, GPIO.IN)

    #Count until the pin goes high
    while (GPIO.input(pin_to_circuit) == GPIO.LOW):
        count += 1
    return count
```

The ultrasonic was tricky. I had to define a limit at which the sensor could read because if the wave traveled to far it would be inaccurate. Essentially the ultrasonic outputs a wave and the time it takes for the wave to come back determines the distance. The time starts when the pulse is sent out and the time ends when the input is HIGH (pulse comes back in). The pulse time is end time minus start time. Using speed of sound relations (pingTime * 340 / 2/ 1000) I got centimeters and multiplied it by 0.3937 to get inches.

```
trigPin = 16
echoPin = 18
MAX_DISTANCE = 220          #define the maximum measured distance
timeOut = MAX_DISTANCE*60   #calculate timeout according to the maximum measured distance


def pulseIn(pin,level,timeOut): # function pulseIn: obtain pulse time of a pin
    t0 = time.time()
    while(GPIO.input(pin) != level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    t0 = time.time()
    while(GPIO.input(pin) == level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    pulseTime = (time.time() - t0)*1000000
    return pulseTime

def getSonar():     #get the measurement results of ultrasonic module,with unit: cm
    GPIO.output(trigPin,GPIO.HIGH)      #make trigPin send 10us high level
    time.sleep(0.00001)     #10us
    GPIO.output(trigPin,GPIO.LOW)
    pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)   #read plus time of echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0     # the sound speed is 340m/s, and calcul
    return distance

def getDistanceIn():
    GPIO.setup(trigPin, GPIO.OUT)   #
    GPIO.setup(echoPin, GPIO.IN)
    GPIO.setup(11,GPIO.IN)
    distance = getSonar() * 0.3937    #multiply to get inches
    time.sleep(1)
    return distance
```





Lastly I was able to use smtplib to send emails. The function send_email takes in a subject and message. The smtp object instance is created and I log in. Then use the sendmail function.

```
def send_email(subject, message):
    try:
        s = smtplib.SMTP('smtp.gmail.com:587')
        s.ehlo()
        s.starttls()
        s.login('brandon.w.ngo@gmail.com', '[redacted]')
        print 'login success'
        message = 'Subject: {}\n\n{}'.format(subject,message)
        s.sendmail('brandon.w.ngo@gmail.com', 'brandon.w.ngo@gmail.com', message)
        s.quit()
        print 'email sent'
    except:
        print 'email failed to send'
```

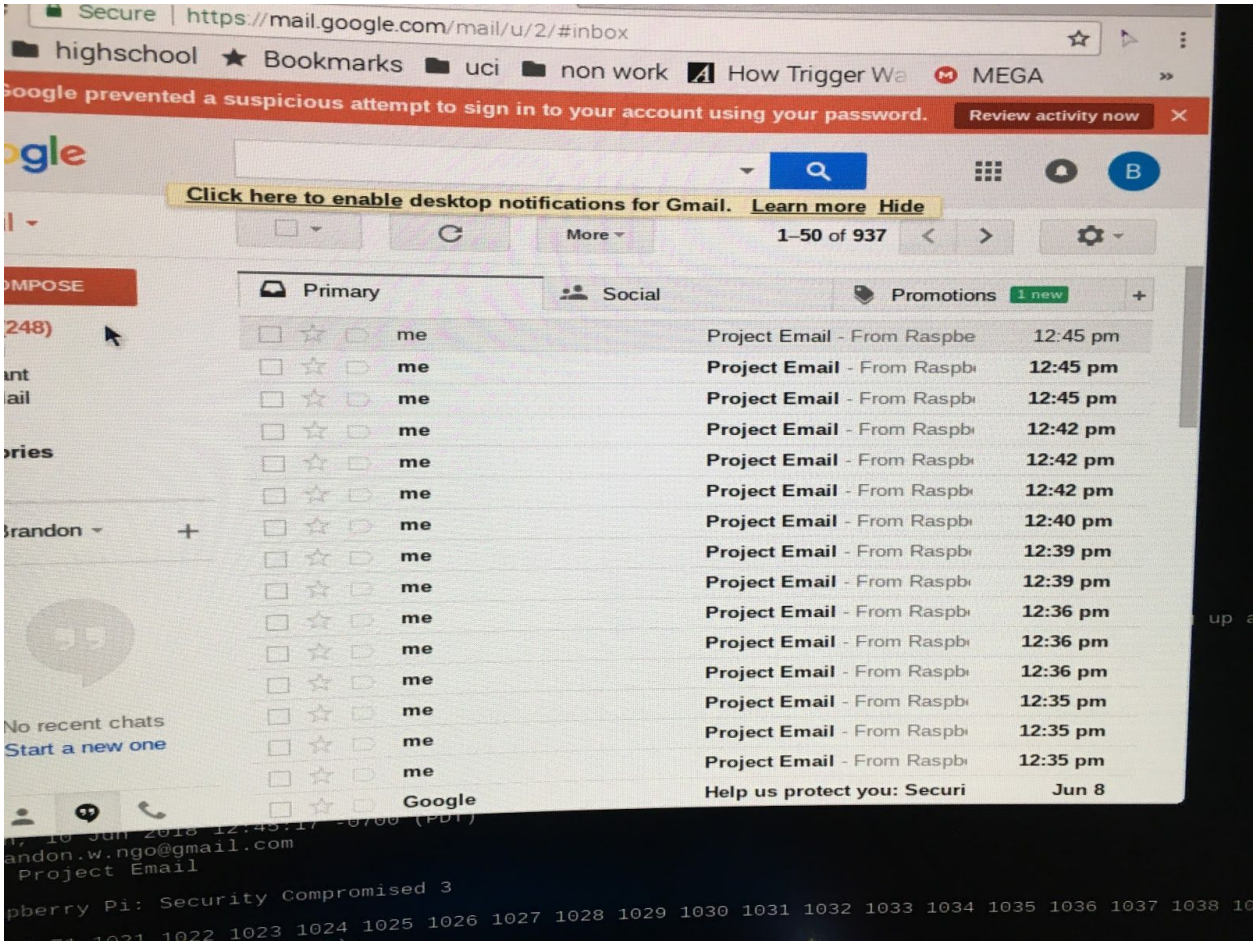
I also used imaplib to read emails.

```
import imaplib

mail = imaplib.IMAP4_SSL('imap.gmail.com')
mail.login('brandon.w.ngo@gmail.com', 'GuySolo721')
mail.list()
# Out: list of "folders" aka labels in gmail.
mail.select("inbox") # connect to inbox.
result, data = mail.search(None, "ALL")

ids = data[0] # data is a list.
id_list = ids.split() # ids is a space separated string
latest_email_id = id_list[-1] # get the latest

# fetch the email body (RFC822) for the given ID
result, data = mail.fetch(latest_email_id, "(RFC822)")

raw_email = data[0][1] # here's the body, which is raw text of the whole email
# including headers and alternate payloads
```

RESULTS

I was able to get both the modes working without a problem. Some struggles had was with accessing gmail. At home I was able to login to gmail but I had to change security option to be able to access it from a less secure device and google still gave me many warnings. I should have kept this in mind because in the demo google locked me out because I was on a new network (school wifi). I should have checked my email and given permission but was too rushed and I took the loss. However at home I can confirm that it works and I know that it would work at school if I gave the necessary permissions.

Every time movement or light was detected an email was sent. Note the security issues that I mentioned.



Here is the terminal to show that everything was working for all cases.

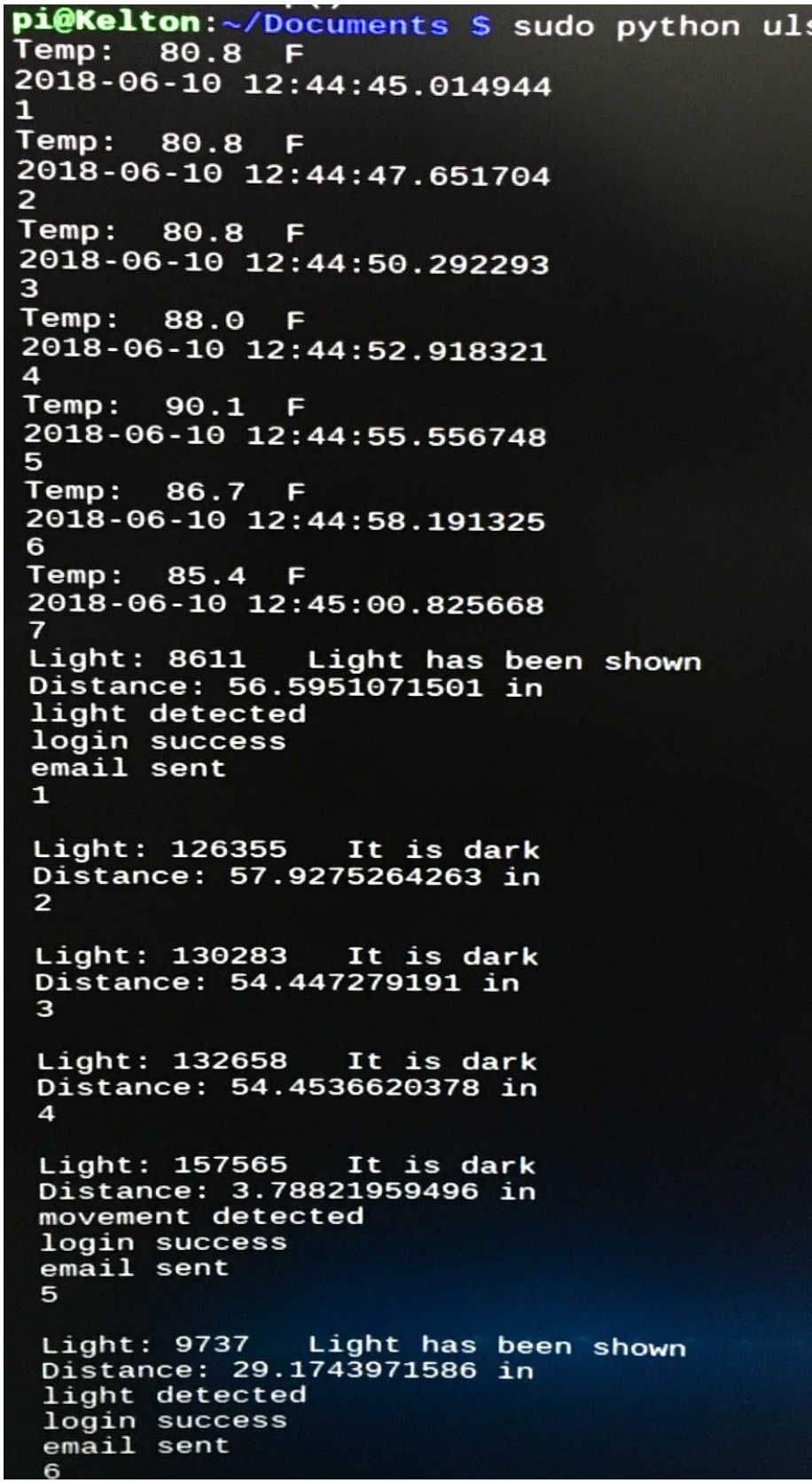
The starting temp and time

Temp rises as I touch the thermistor

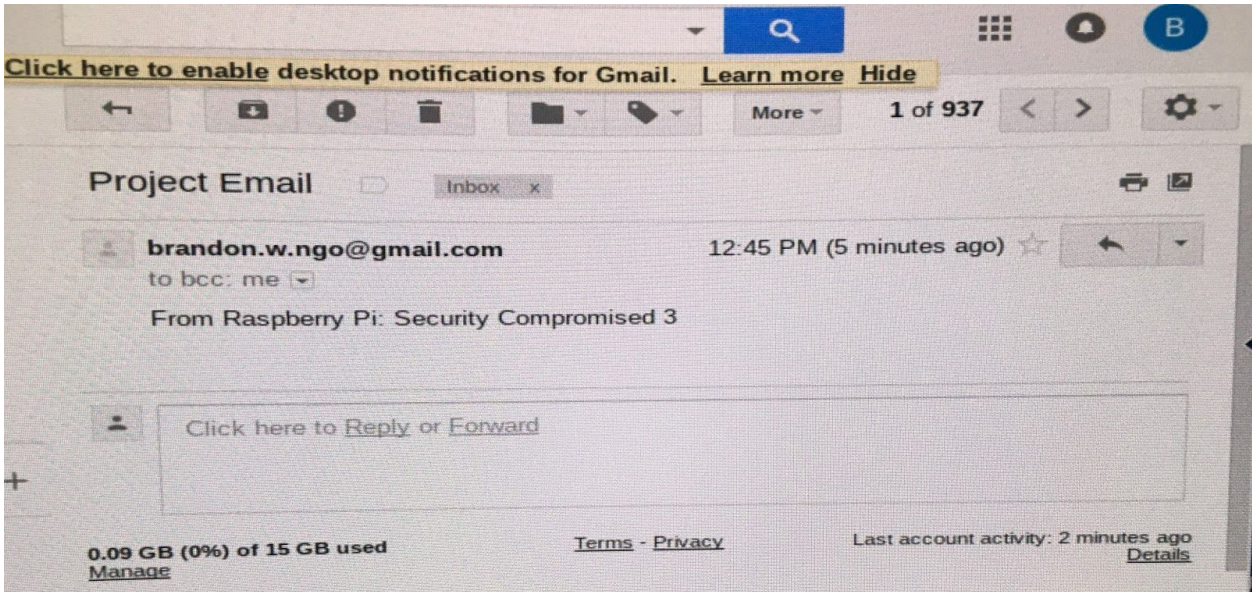
Drops as I let go
Mode switches when button pressed
Light detected (below 20,000)
Email sent

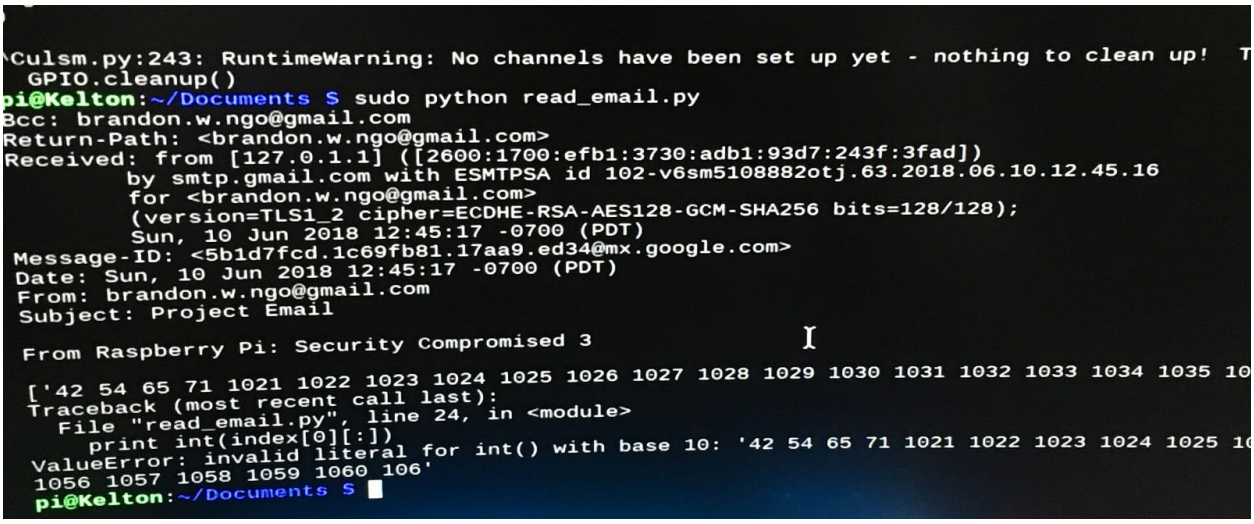
Both (light < 20,000) and (distance > 20 inches) so no activity

Distance is less than 20 inches
Email sent



Here I was able to read the first email in the inbox and get all the text.



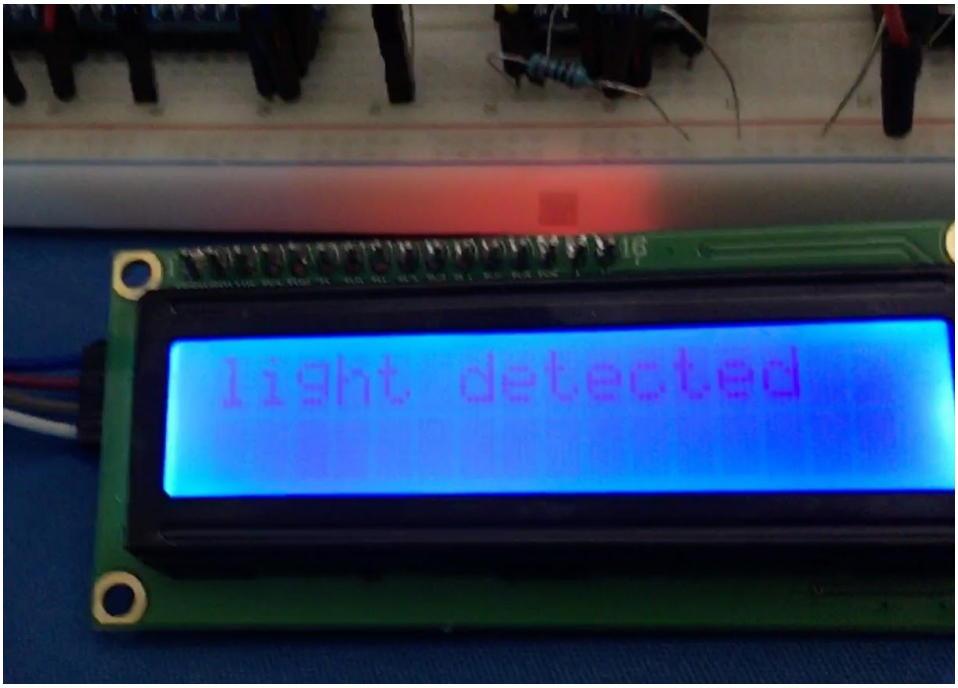


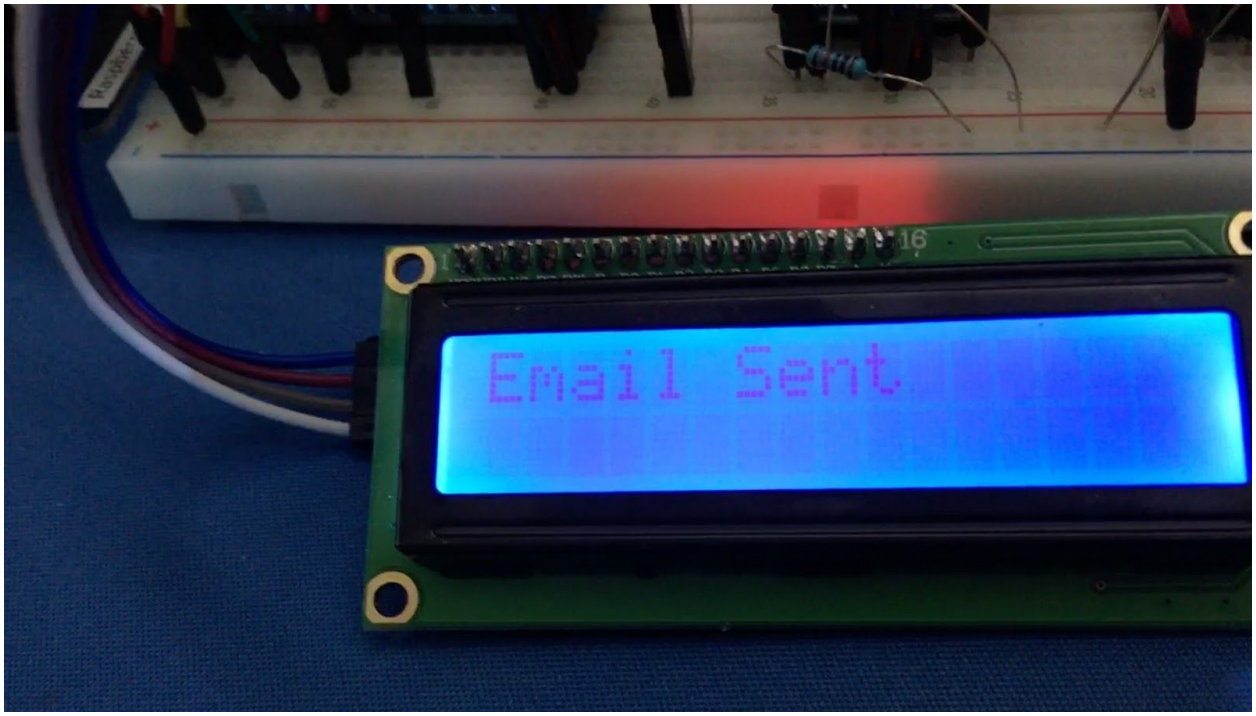
LCD was refreshed every second
Date and time displayed on LCD

Temp displayed on LCD

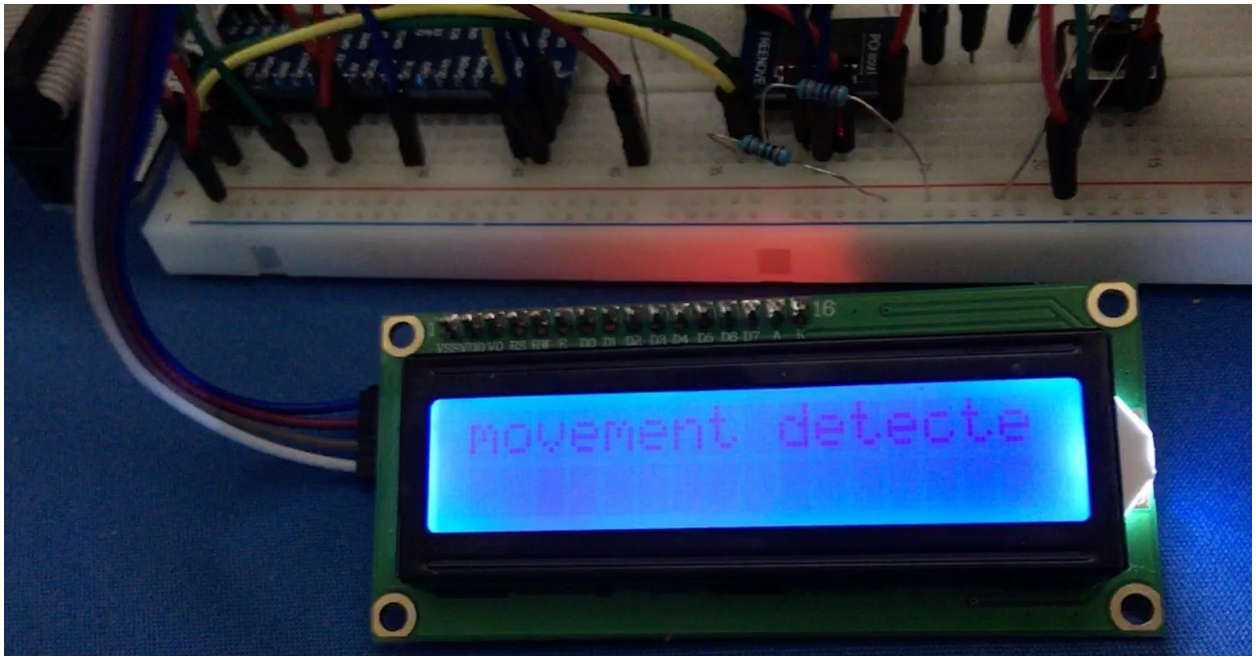
Switched modes, now in security mode and no action

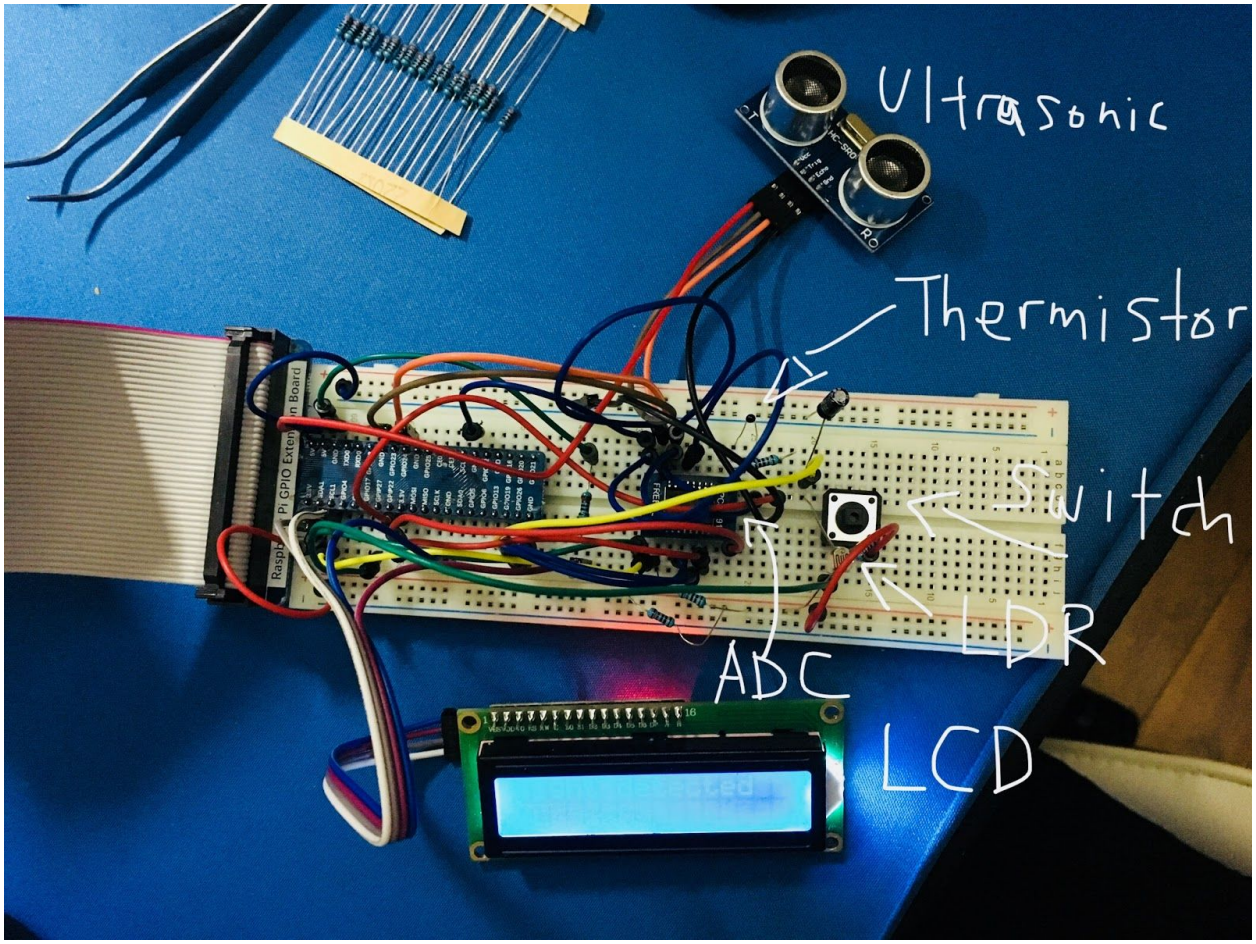
Light detected

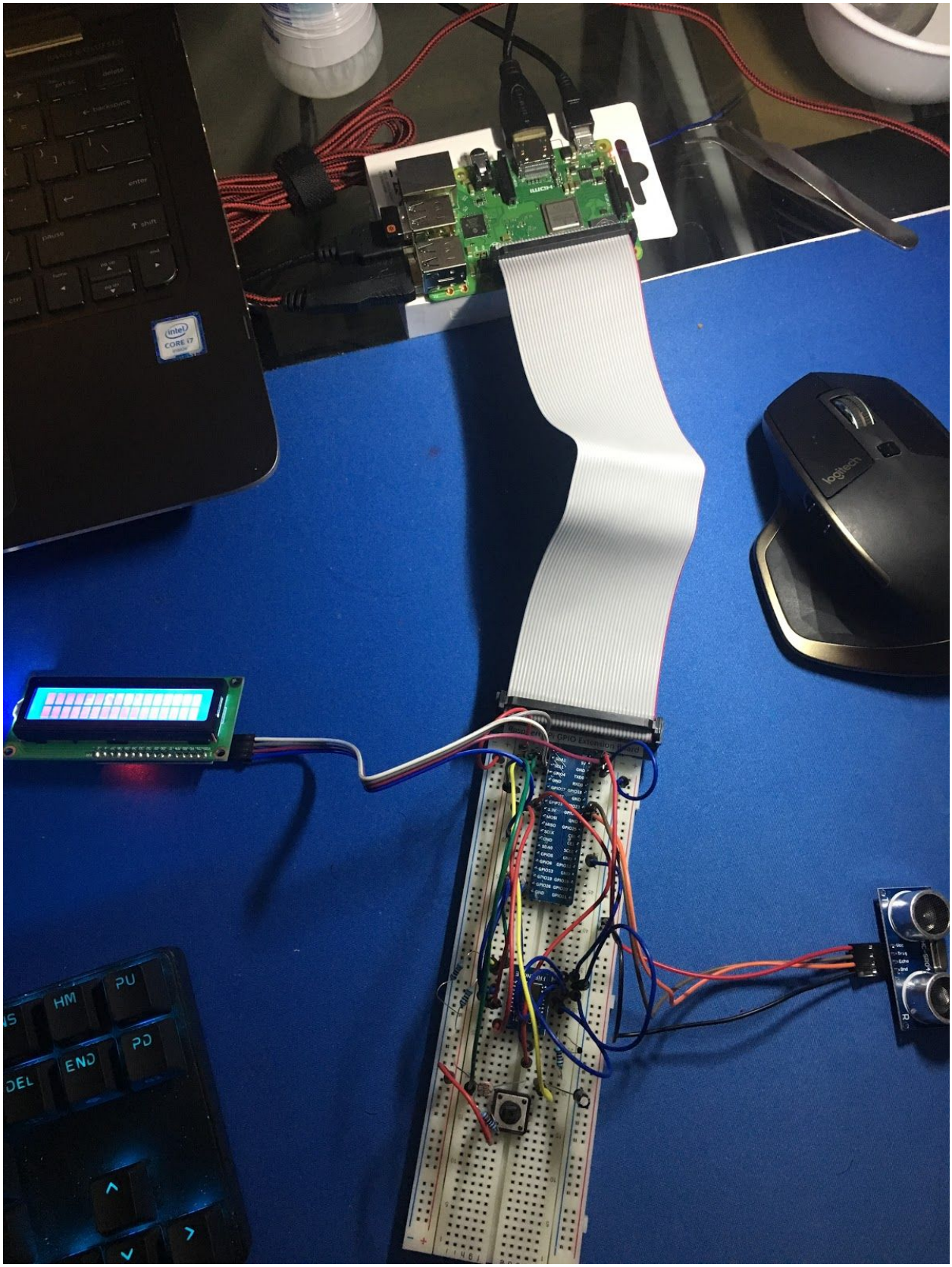




Movement detected




Ultrasonic
Thermistor
Switch
LDR
ADC
LCD


intel
CORE i7
logitech
HM
PU
END
PD

I can definitely expand on this project and make this my go to home controller. I can add a camera, add mechanical moving parts, add LEDs and much more. It was very fun, challenging, and rewarding doing this project and I liked how I was able to apply what I learned in other classes as well.