TEST DOCUMENTATION

Temperature Sensors for Veterans

Email: sdmay23-07@gmail.com Website: https://sdmay23-07.sd.ece.iastate.edu/

Members: Michael McDonough, Maxwell Berthold, Jamie Andersen, Bridget Schmitt, Caleb Arnold, George Makhali, Jared Cox

TABLE OF CONTENTS

INTERFACE TESTING

ARDUINO / TEMPERATURE SENSOR

OVERVIEW

This testing procedure will be used to determine the accuracy of the Arduino MCU and temperature sensor interface. Criteria used in testing will be used to ensure that the temperature sensor is accurate within ±2° Fahrenheit as specified by the Client and Potential Users.

PROCEDURE

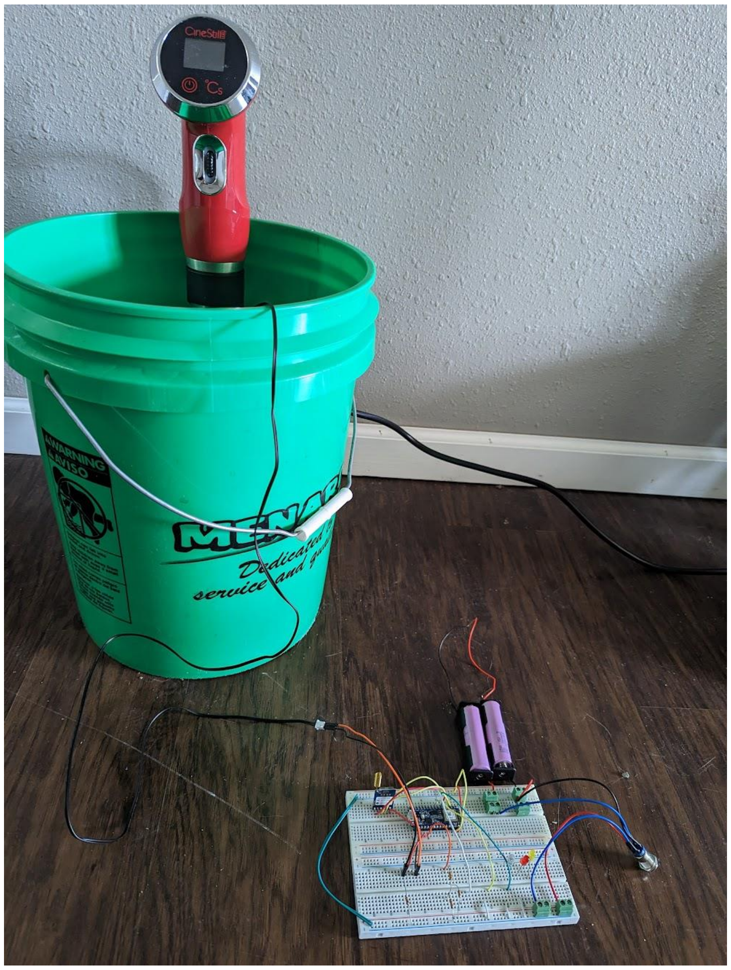
The temperature sensor, once connected to the Arduino MCU, will be subjected to controlled temperatures using an immersion circular thermostat. This device is similar to a kitchen precision cooker, but with a greater degree of precision: ±0.1° Fahrenheit. The procedure has two steps: initial calibration of the thermistor and calibration testing.

Initial calibration is performed by heating water to three points, obtaining the resistance of the thermistor at those three points, and calculating the Steinhart-Hart coefficients of the thermistor to add in software. The thermistor will be submerged in the water for ten minutes at each temperature point of 65, 85, and 110° F.

Testing will be performed by using the same procedure as calibration, but this time reading the temperature output of the Arduino rather than the resistance of the thermistor. The temperature reading will be compared to reference values between 60 and 100° F at 10° intervals.

HARDWARE SETUP

A picture of the setup for this testing procedure can be found below:

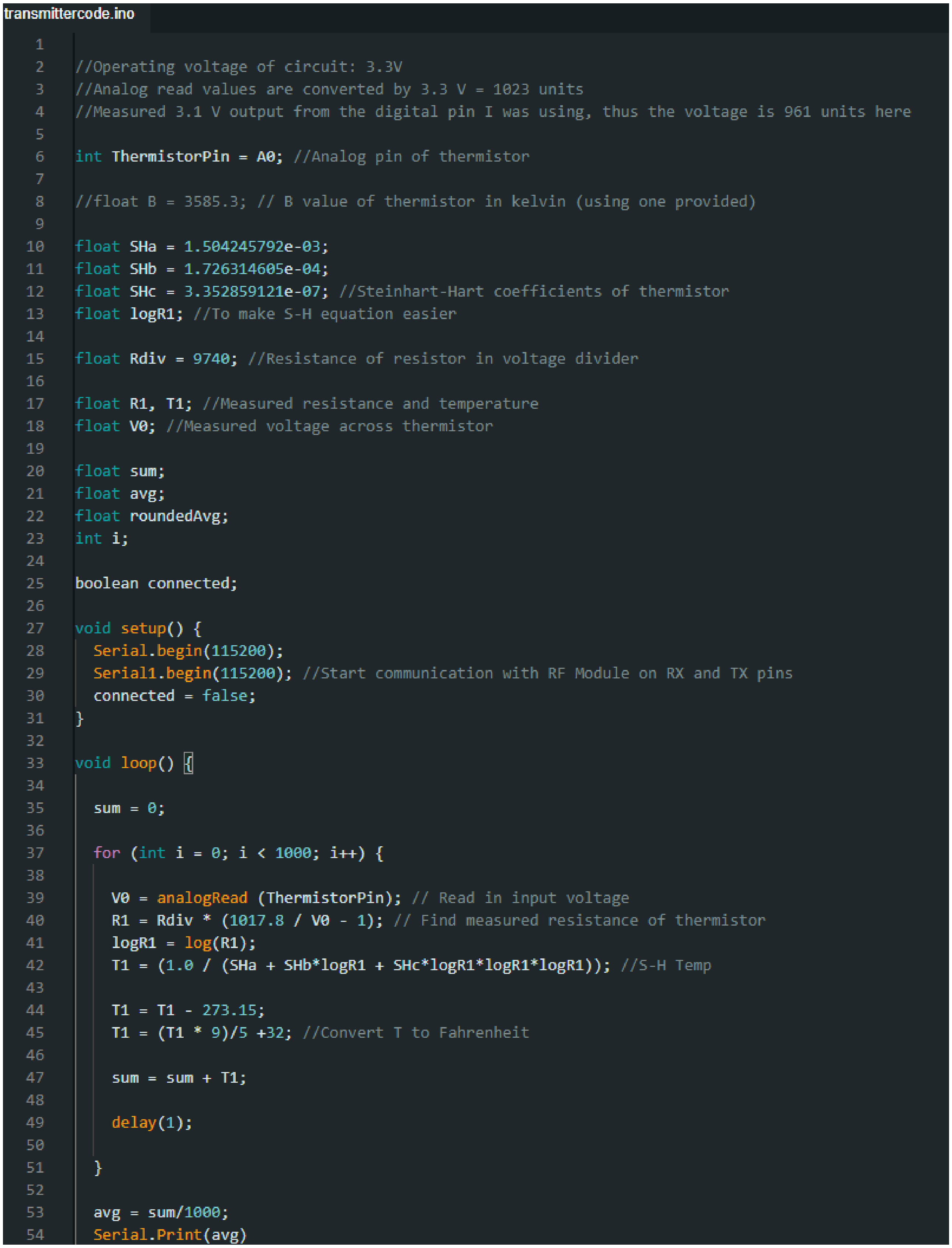


Description of testing setup: Our group utilized the following materials in order to create this setup:

- Arduino Nano MCU
- 2M NTC 10K 3950 1% Temperature Waterproof Probe
- (1) 10K Ohm Resistor
- Jumper Wires
- 5 gallon bucket
- Cinestill °CS Temperature Control System

The temperature sensing circuit in the prototype device shown above includes a voltage divider circuit including the NTC thermistor and oowered by the Arduino Nano's 3.3V output. An analog pin, A0 in this case, is used to monitor the voltage drop across the thermistor so that it can be converted to temperature. The Cinestill °CS is used to heat water in a 5 gallon bucket to a uniform reference temperature that the Arduino output can be compared to.

SOFTWARE SETUP


transmittercode.ino

```

//Operating voltage of circuit: 3.3V
//Analog read values are converted by 3.3 V = 1023 units
//Measured 3.1 V output from the digital pin I was using, thus the voltage is 961 units here

int ThermistorPin = A0; //Analog pin of thermistor

//float B = 3585.3; // B value of thermistor in kelvin (using one provided)

float SHa = 1.504245792e-03;
float SHb = 1.726314605e-04;
float SHc = 3.352859121e-07; //Steinhart-Hart coefficients of thermistor
float logR1; //To make S-H equation easier

float Rdiv = 9740; //Resistance of resistor in voltage divider

float R1, T1; //Measured resistance and temperature
float V0; //Measured voltage across thermistor

float sum;
float avg;
float roundedAvg;
int i;

boolean connected;

void setup() {
  Serial.begin(115200);
  Serial1.begin(115200); //Start communication with RF Module on RX and TX pins
  connected = false;
}

void loop() {

  sum = 0;

  for (int i = 0; i < 1000; i++) {

    V0 = analogRead (ThermistorPin); // Read in input voltage
    R1 = Rdiv * (1017.8 / V0 - 1); // Find measured resistance of thermistor
    logR1 = log(R1);
    T1 = (1.0 / (SHa + SHb*logR1 + SHc*logR1*logR1*logR1)); //S-H Temp

    T1 = T1 - 273.15;
    T1 = (T1 * 9)/5 +32; //Convert T to Fahrenheit

    sum = sum + T1;

    delay(1);

  }

  avg = sum/1000;
  Serial.Print(avg)
```

Above is the code used to process and display temperature inputs to the Arduino. Floats ShA, ShB, and ShC represent the Steinhart-Hart coefficients of the NTC thermistor. Other variables declared include measured resistance of our nominal 10k resistor and variables for input and processing data. The main loop is used to take 1000 measurements of the voltage drop across the thermistor over one second, convert this voltage reading to temperature using Steinhart-Hart and voltage divider equations, and average these readings. The result is then printed to the serial monitor.

RESULTS

Results of the thermistor calibration as well as testing at multiple points are shown below:

Thermistor Calibration			**Calibration Testing**	
Temperature (°F)	Resistance (Ω)		Reported Temp (°F)	Actual Temp (°F)
65	13310		59.5	60
85	8180		69.8	70
110	3570		80.0	80
			90.1	90
			100.6	100
			110.4	110

Results are very promising, with very little variation between the device's readings and the reference temperature. The temperature delta at our testing values is much less than the two degrees required by the client, therefore this test is successful.

ARDUINO / TEMPERATURE SENSOR / ALERT LEDs

OVERVIEW

This testing procedure will be used to determine the functionality of the alert system in collaboration with the temperature sensor and Arduino MCU interface. Criteria used in testing will be used to ensure that the alert systems are being activated as intended, i.e., temperature out of range, battery level low, and loss of connection between the application and the device.

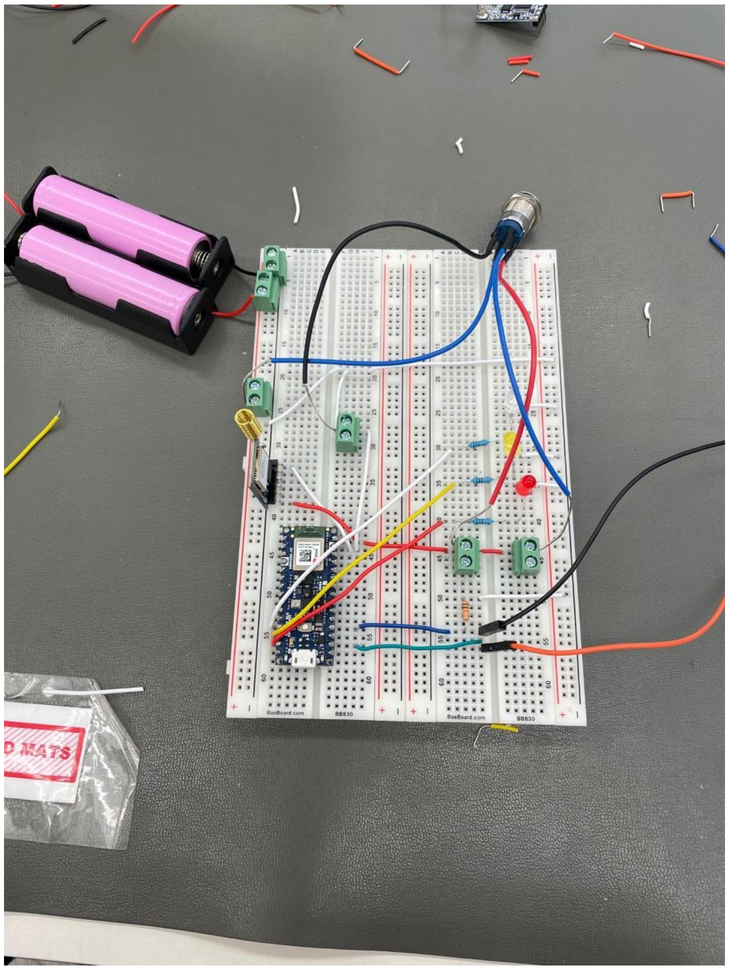
PROCEDURE

Three LEDs will be connected to the Arduino and temperature sensor interface (reference ARDUINO / TEMPERATURE SENSOR TESTING for setup). The Arduino will then be programmed to have one LED activate when temperatures measured are out of the range specified, one LED activate when the battery supply is below 30%, and one LED that will activate when the Bluetooth connection between the device and phone application has been broken. Testing will then be completed as follows:

1. Load program onto Arduino MCU (see Software Setup for code used)
2. Turn on Arduino MCU / Initialize program
3. Subject Temperature Sensor to various temperatures both in and out of range
4. Observe LEDs for activation under designed circumstances

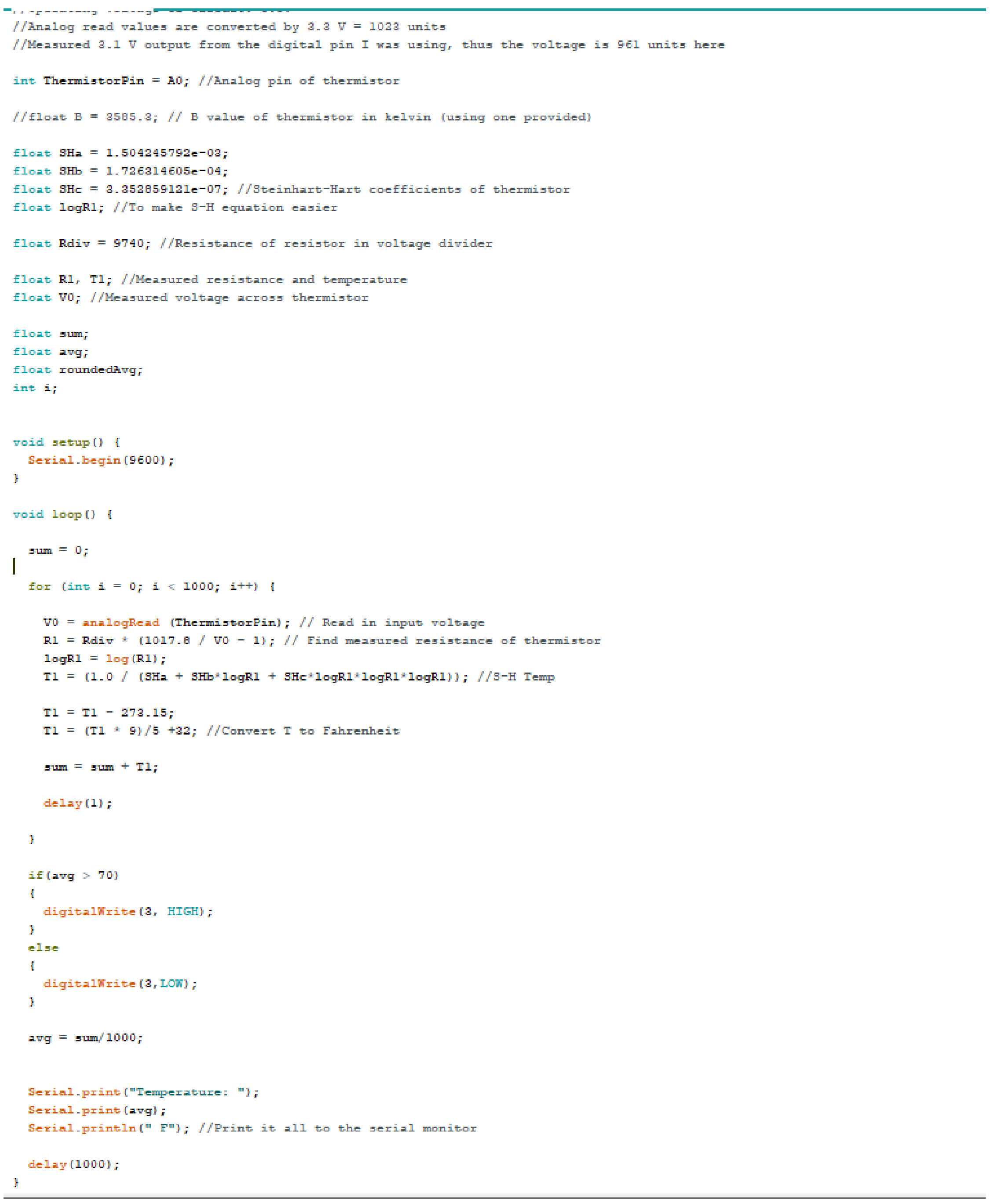
HARDWARE SETUP

A picture of the hardware setup for this procedure can be found below:



SOFTWARE SETUP

Below is the code used to program the Arduino MCU for this testing procedure:



```
//Analog read values are converted by 3.3 V = 1023 units
//Measured 3.1 V output from the digital pin I was using, thus the voltage is 961 units here

int ThermistorPin = A0; //Analog pin of thermistor

//float B = 3585.3; // B value of thermistor in kelvin (using one provided)

float SHa = 1.504245792e-03;
float SHb = 1.726314605e-04;
float SHc = 3.352859121e-07; //Steinhart-Hart coefficients of thermistor
float logR1; //To make S-H equation easier

float Rdiv = 9740; //Resistance of resistor in voltage divider

float R1, T1; //Measured resistance and temperature
float V0; //Measured voltage across thermistor

float sum;
float avg;
float roundedAvg;
int i;


void setup() {
  Serial.begin(9600);
}

void loop() {

  sum = 0;

  for (int i = 0; i < 1000; i++) {

    V0 = analogRead (ThermistorPin); // Read in input voltage
    R1 = Rdiv * (1017.8 / V0 - 1); // Find measured resistance of thermistor
    logR1 = log(R1);
    T1 = (1.0 / (SHa + SHb*logR1 + SHc*logR1*logR1*logR1)); //S-H Temp

    T1 = T1 - 273.15;
    T1 = (T1 * 9)/5 +32; //Convert T to Fahrenheit

    sum = sum + T1;

    delay(1);

  }

  if(avg > 70)
  {
    digitalWrite(3, HIGH);
  }
  else
  {
    digitalWrite(3,LOW);
  }

  avg = sum/1000;


  Serial.print("Temperature: ");
  Serial.print(avg);
  Serial.println(" F"); //Print it all to the serial monitor

  delay(1000);
}
```

RESULTS

Using this test procedure, we were able to confirm that we could activate LEDs based on different temperature ranges read by the temperature sensor and Arduino board.

ARDUINO / LORA MODULE

OVERVIEW

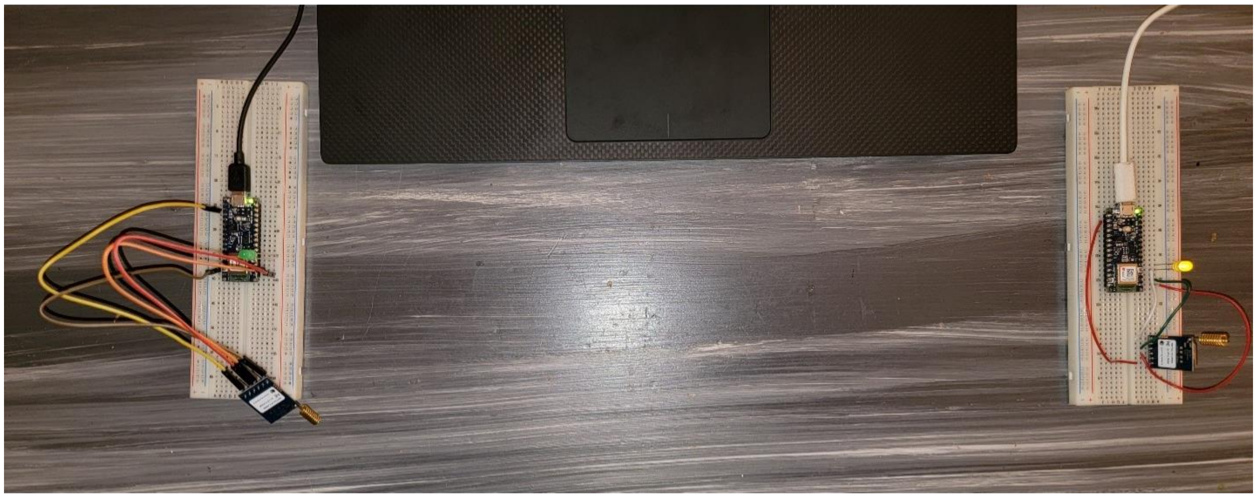
This basic test was performed to test how the Arduino MCU interacts with the RYLR890 LoRa module.

PROCEDURE

The goal of this testing was to send data from one Arduino and have the other Arduino turn on/off an LED based on what the transmitted data was. If the data read by the receiver Arduino was "HI" then the LED would turn on, and if the data was "BYE" the LED would turn off.

HARDWARE SETUP

A picture of the setup for this testing procedure can be found below:



Description of Testing Setup: Our group utilized the following materials in order to create this setup:

- Two (2) Arduino Nano 33 BLE
- Two (2) Relax RYLR890 LoRa Modules
- Jumper Wires
- Power Supply from Computer via USB cable
- One (1) LED Indicator

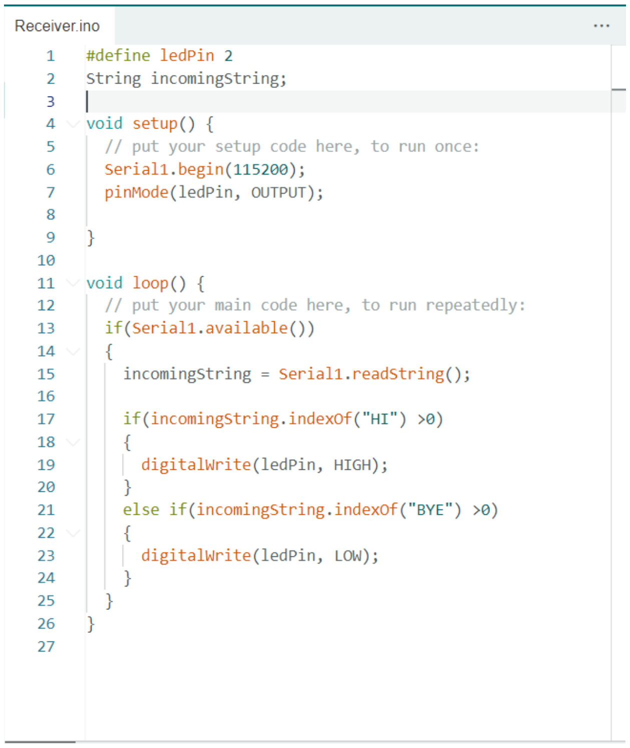
The two Arduino Nano 33 BLE circuits were set up the exact same way allowing both LoRa modules to be both a receiver and a transmitter. The only difference was the code that each Arduino was running.

Arduino Nano 33 BLE to RYLR890 Connection:

- Arduino 3V3 pin to RYLR890 VDD pin
- Arduino RX pin to RYLR890 TX pin
- Arduino TX pin to RYLR890 RX pin
- Arduino GND to RYLR890 GND

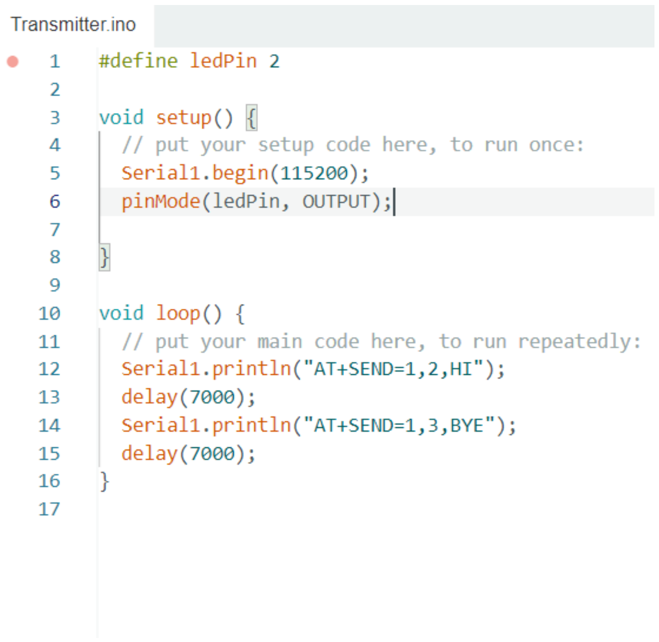
SOFTWARE SETUP

Below is the code utilized for the transmitter Arduino MCU and receiver Arduino MCU. To begin, I needed to initialize the RX and TX pins on the Arduino using the Serial1.begin() function. I then used Serial1.println(AT+SEND=1,2,HI) to have the transmitter send "HI" to the receiver, and Serial1.readString() on the receiver to read the incoming data.



Receiver.ino

```
#define ledPin 2
String incomingString;

void setup() {
  // put your setup code here, to run once:
  Serial1.begin(115200);
  pinMode(ledPin, OUTPUT);

}

void loop() {
  // put your main code here, to run repeatedly:
  if(Serial1.available())
  {
    incomingString = Serial1.readString();

    if(incomingString.indexOf("HI") >0)
    {
      digitalWrite(ledPin, HIGH);
    }
    else if(incomingString.indexOf("BYE") >0)
    {
      digitalWrite(ledPin, LOW);
    }
  }
}
```



Transmitter.ino

```
#define ledPin 2

void setup() {
  // put your setup code here, to run once:
  Serial1.begin(115200);
  pinMode(ledPin, OUTPUT);

}

void loop() {
  // put your main code here, to run repeatedly:
  Serial1.println("AT+SEND=1,2,HI");
  delay(7000);
  Serial1.println("AT+SEND=1,3,BYE");
  delay(7000);
}
```

RESULTS

We were successfully able to send data between the two Arduinos and LoRa radio frequency modules a distance of 60 meters.

ARDUINO / TEMPERATURE SENSOR / LORA MODULE

OVERVIEW

This test was used to confirm that we could accurately send the temperature readings from one Arduino board and LoRa RF module to the other Arduino board and LoRa RF module.

PROCEDURE

1. Setup two Arduino 33 BLE boards with LoRa modules
2. Setup one of the Arduino boards with the temperature sensing circuit
3. Power both boards on and run the transmitter code on the one with the temperature sensor and the receiver code on the other board
4. Check the serial monitor of the receiver board to confirm data is being received
5. Cross check the received data with the actual temperature readings from the transmitter board

HARDWARE SETUP

Transmitter board:

RYLR890 LoRa Module Connections:

- Arduino 3V3 pin to RYLR890 VDD pin
- Arduino RX pin to RYLR890 TX pin
- Arduino TX pin to RYLR890 RX pin
- Arduino GND to RYLR890 GND

Temperature Sensor Connection:

The NTC Thermistor is used in the 10K voltage divider circuit in order to obtain the voltage that is then translated into a temperature reading by the Arduino MCU. The temperature sensor pins are connected to A0 (voltage reading), 3.3V (power supply), and GND.

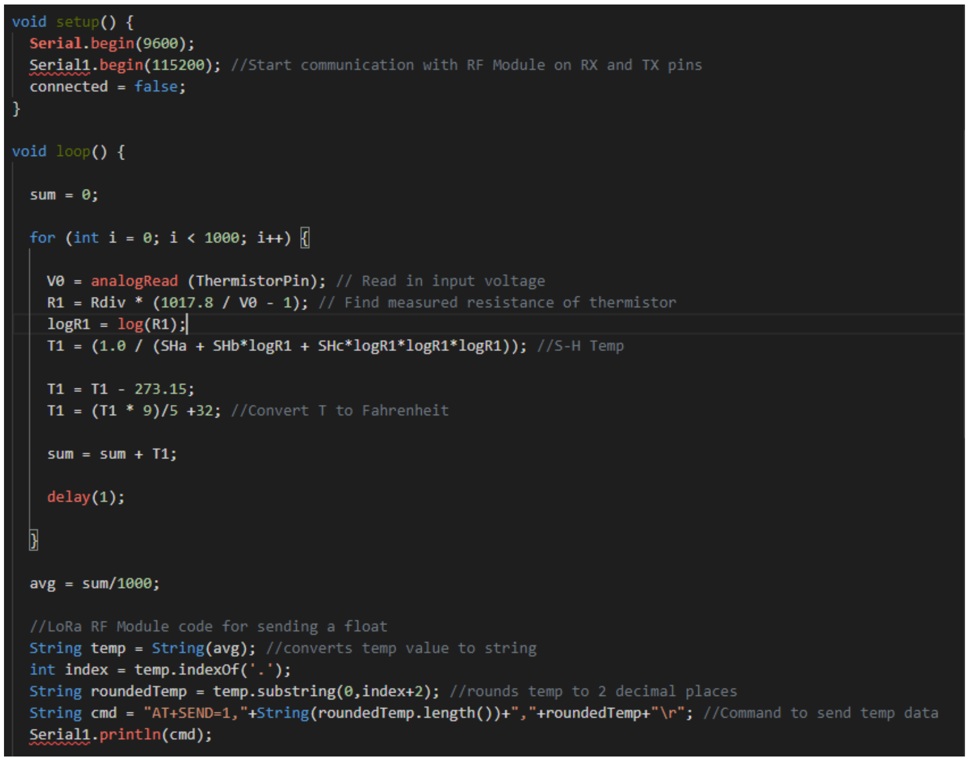
Receiver Board:

RYLR890 LoRa Module Connections:

- Arduino 3V3 pin to RYLR890 VDD pin
- Arduino RX pin to RYLR890 TX pin
- Arduino TX pin to RYLR890 RX pin
- Arduino GND to RYLR890 GND

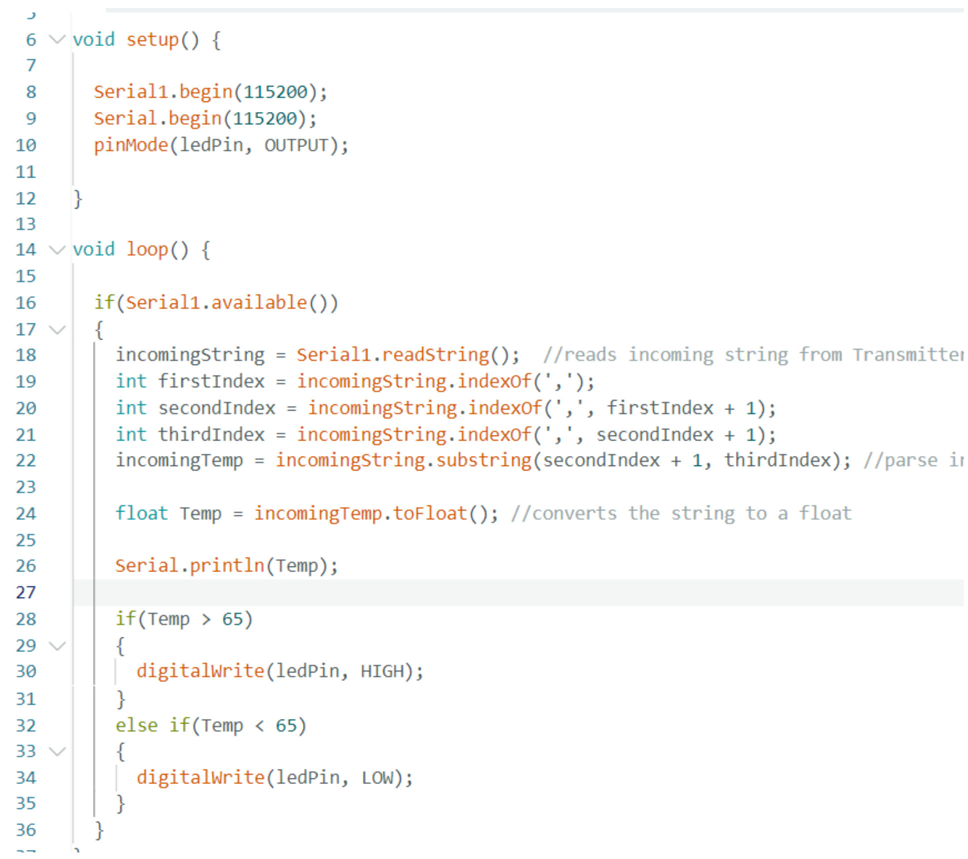
SOFTWARE SETUP

Transmitter Code:



```
void setup() {
  Serial.begin(9600);
  Serial1.begin(115200); //Start communication with RF Module on RX and TX pins
  connected = false;
}

void loop() {

  sum = 0;

  for (int i = 0; i < 1000; i++) {

    V0 = analogRead (ThermistorPin); // Read in input voltage
    R1 = Rdiv * (1017.8 / V0 - 1); // Find measured resistance of thermistor
    logR1 = log(R1);
    T1 = (1.0 / (SHa + SHb*logR1 + SHc*logR1*logR1*logR1)); //S-H Temp

    T1 = T1 - 273.15;
    T1 = (T1 * 9)/5 +32; //Convert T to Fahrenheit

    sum = sum + T1;

    delay(1);

  }

  avg = sum/1000;

  //LoRa RF Module code for sending a float
  String temp = String(avg); //converts temp value to string
  int index = temp.indexOf('.');
  String roundedTemp = temp.substring(0,index+2); //rounds temp to 2 decimal places
  String cmd = "AT+SEND=1,"+String(roundedTemp.length())+","+roundedTemp+"\r"; //Command to send temp data
  Serial1.println(cmd);
```

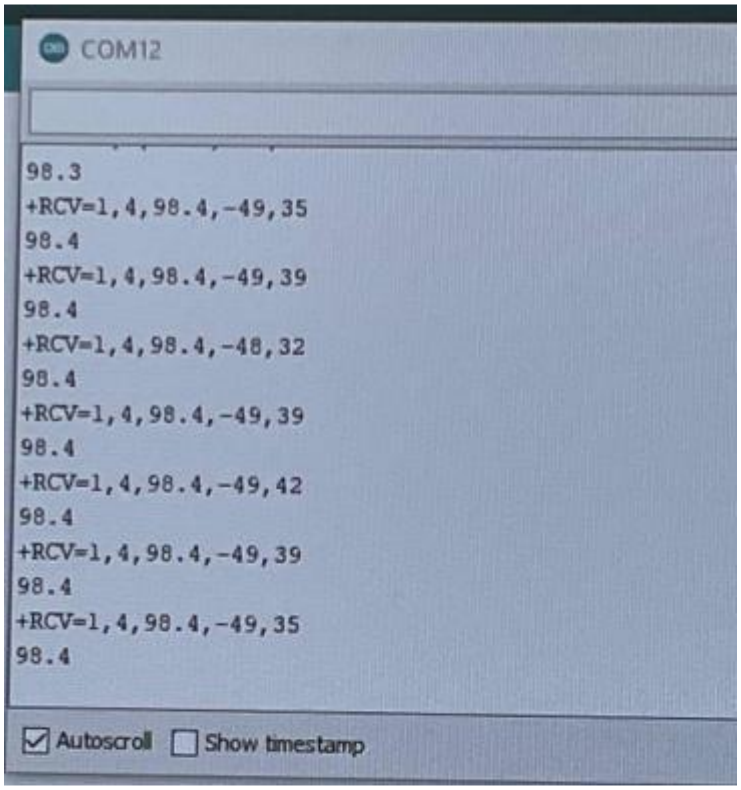
Receiver Code:



```
void setup() {

  Serial1.begin(115200);
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);

}

void loop() {

  if(Serial1.available())
  {
    incomingString = Serial1.readString();  //reads incoming string from Transmitte
    int firstIndex = incomingString.indexOf(',');
    int secondIndex = incomingString.indexOf(',', firstIndex + 1);
    int thirdIndex = incomingString.indexOf(',', secondIndex + 1);
    incomingTemp = incomingString.substring(secondIndex + 1, thirdIndex); //parse i

    float Temp = incomingTemp.toFloat(); //converts the string to a float

    Serial.println(Temp);

    if(Temp > 65)
    {
      digitalWrite(ledPin, HIGH);
    }
    else if(Temp < 65)
    {
      digitalWrite(ledPin, LOW);
    }
  }
```

RESULTS

Below is a screenshot of the receiver’s serial monitor:


COM12

```
98.3
+RCV=1,4,98.4,-49,35
98.4
+RCV=1,4,98.4,-49,39
98.4
+RCV=1,4,98.4,-48,32
98.4
+RCV=1,4,98.4,-49,39
98.4
+RCV=1,4,98.4,-49,42
98.4
+RCV=1,4,98.4,-49,39
98.4
+RCV=1,4,98.4,-49,35
98.4
```

As can be seen from the image above, the receiver was receiving the data and then accurately parsing the data for the temperature reading. The “+RCV=1,98.4,-49,39” is how the data is sent using the LoRa modules and the line below is the data parsed for the temperature reading.

ARDUINO / PUSH BUTTON SWITCH

OVERVIEW

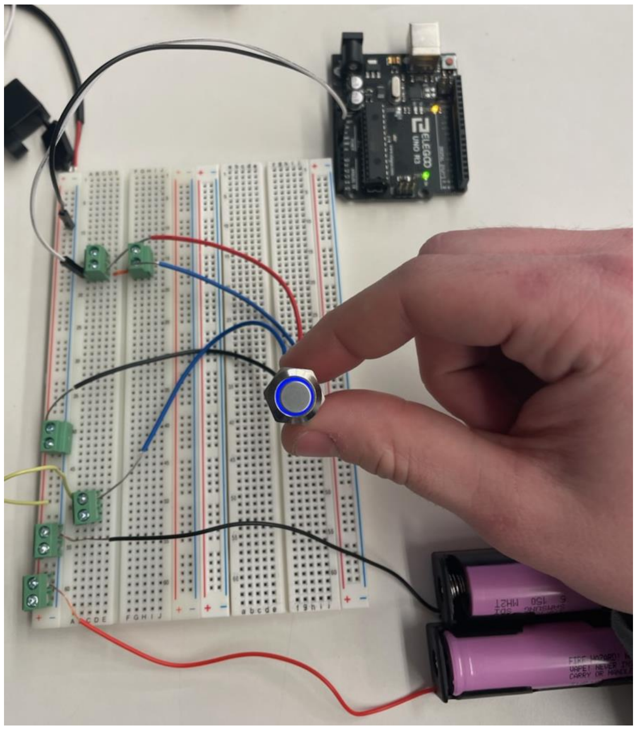
This testing procedure was used to confirm functionality of the push button. The push button is used to control the power to the Arduino board and also serves as an indication that the device is powered on because there is an LED around the push button that is powered when in the “on” position.

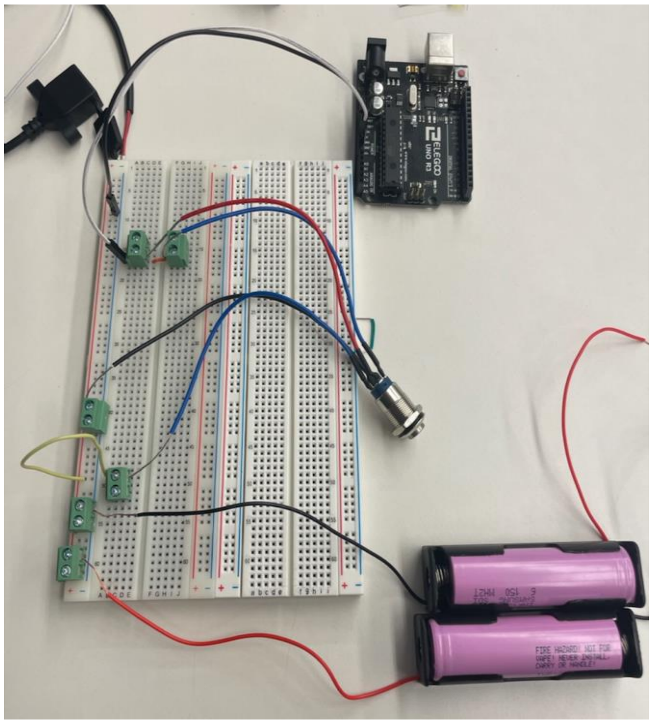
PROCEDURE

1. Connect push button to Arduino board
2. Connect push button to battery pack
3. Confirm Arduino is powered by checking if the LED onboard is lit

HARDWARE SETUP

Images showing the hardware setup can be seen below.





RESULTS

As you can see from the above images, both the LED on the push button and the LED on the Arduino board are powered on. This test confirmed functionality of the push button power control.

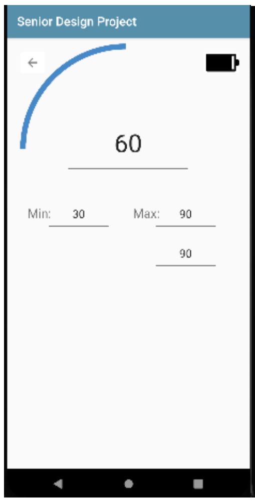
APPLICATION / TEMPERATURE SENSOR PROGRESS BAR OVERVIEW

This testing procedure will be used to determine the temperature sensor progress bar represents an accurate visual to depict how close the user's skin temperature is to the top or the bottom the user set range.

PROCEDURE

1. Open application in Android Studio
2. Log into application with user credentials, not instructor credentials
3. When on page with "Welcome to Adaptive Adventures", press the triple line symbol on the top corner of the page to reach the menu.
4. When at the menu, press "Temperature Sensor"
5. Initially, if not connected to hardware, the minimum value is 30, current value is 60, and high value is 90.
6. Check that the half circle progress bar ends at the top of the screen making approximately a quarter of a circle.
7. Change the minimum value to 59, check that you can only see a very small fraction of progress bar on the left side of the screen. With these values, the progress bar should be a dot, the same width as length.
8. Change the minimum value to 30 and change the maximum value to 61. Check that the progress bar is almost a full half circle. The progress bar should be missing a portion about as big as the dot we saw in step 7.
9. Change the maximum value to 70. The half circle should be 3/4 complete or a half circle missing 1/8 of a full circle.
10. Change the maximum value to 90 and the minimum value to 50. The half circle progress bar should be ¼ complete or 1/8 of a full circle.

RESULT

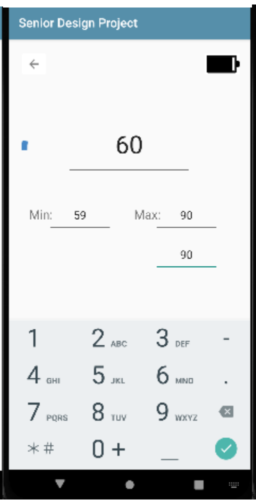


60

Min: 30 Max: 90

90

Step 6

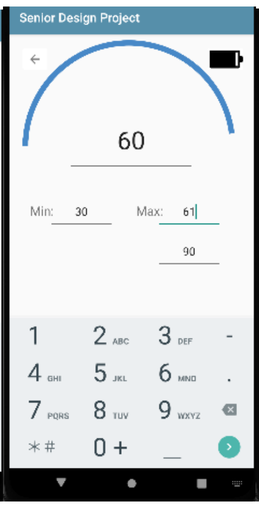


60

Min: 59 Max: 90

90

Step 7

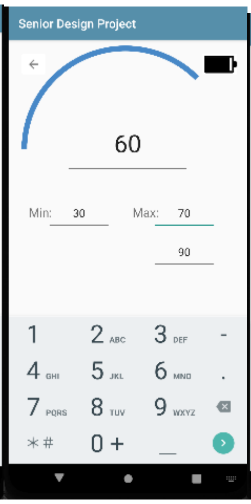


60

Min: 30 Max: 61

90

Step 8

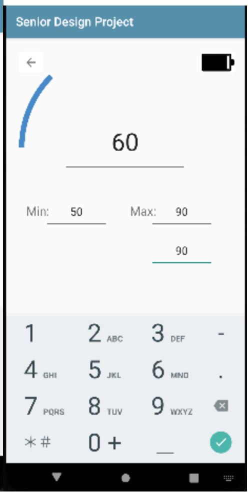


60

Min: 30 Max: 70

90

Step 9



60

Min: 50 Max: 90

90

Step 10

☒Step 6: Half circle sensor progress bar half full

☒Step 7: Half circle sensor progress bar almost empty

☒Step 8: Half circle sensor progress bar almost full

☒Step 9: Half circle sensor progress bar ¾ full

☒Step 10: Half circle sensor progress bar ¼ full

After going through the testing procedure, all of the tests worked as expected. Based on this procedure, the application is working correctly in respect to the sensor progress bar.

APPLICATION / BATTERY USAGE

OVERVIEW

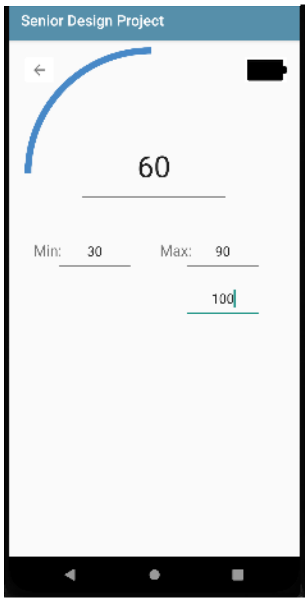
This testing procedure will be used to determine if the battery usage of the temperature device is accurately shown in the battery shaped icon on the Temperature Sensor page.

PROCEDURE

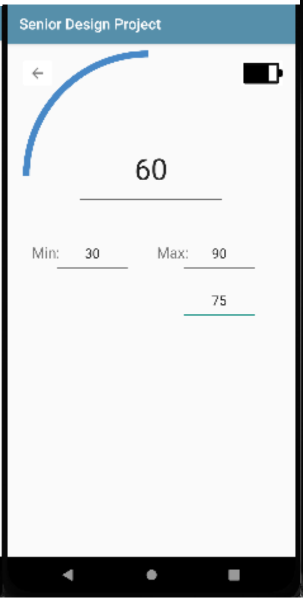
1. Open application in Android Studio
2. Log into application with user credentials, not instructor credentials

3. When on page with "Welcome to Adaptive Adventures", press the triple line symbol on the top corner of the page to reach the menu.
4. When at the menu, press "Temperature Sensor"
5. Initially, if not connected to hardware, the battery test line is located directly below maximum value line. To test the battery indicator, type a new value between 0-100 onto the line located directly below the maximum value line.
6. Change the battery value to 100 on the battery test line. Check that the battery indicator in the top right corner of the application is completely filled in with black.
7. Change the battery value to 75 on the battery test line. Check that the battery indicator in the top right corner of the application is around 75% filled with black.
8. Change the battery value to 50 on the battery test line. Check that the battery indicator in the top right corner of the application is around 50% filled with black.
9. Change the battery value to 25 on the battery test line. Check that the battery indicator in the top right corner of the application is around 25% filled with black.

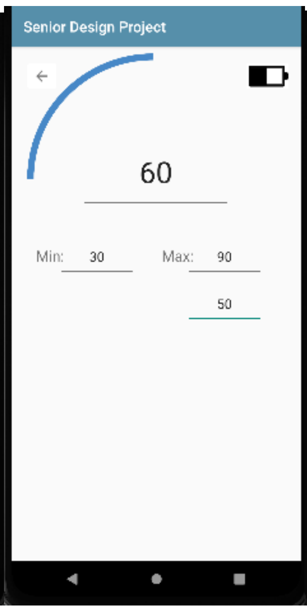
RESULT



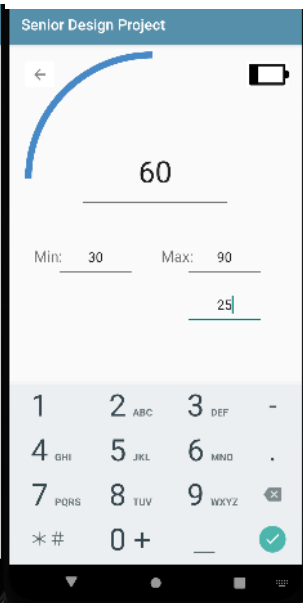
Step 6



Step 7



Step 8



Step 9

☒Step 6: Battery indicator bar full

☒Step 7: Battery indicator bar ¾ full

☒Step 8: Battery indicator bar ½ full

☒Step 9: Battery indicator bar ¼ full

After going through the testing procedure, all of the tests worked as expected. Based on this procedure, the application is working correctly in respect to the battery indicator.

APPLICATION / TEMPERATURE SENSOR PROGRESS BAR NOTIFICATIONS

OVERVIEW

This testing procedure will be used to determine the temperature sensor page alerts the user when skin temperature is at an unsafe level.

PROCEDURE

1. Enable vibration, sound, and notifications on testing device
2. Open application in Android Studio
3. Log into application with user credentials, not instructor credentials
4. When on page with "Welcome to Adaptive Adventures", press the triple line symbol on the top corner of the page to reach the menu.
5. When at the menu, press "Temperature Sensor"
6. Initially, if not connected to hardware, the minimum value is 30, current value is 60, and high value is 90.
7. Change the minimum value to 61. Listen for sound warning, feel for vibration, look for a single notification popup within the temperature sensor page, and pull-down phone notifications to see that there is a low temperature alert. Exit out of the popup to see a red low temperature warning.
8. Change the minimum value to 30. Check that the half circle sensor progress bar goes to its initial state of half full. Check that the red warning text disappears. Repeat step 7 and 8 once to see that the result remains the same.
9. Change the maximum value to 59. Listen for sound warning, feel for vibration, look for a single notification popup within the temperature sensor page, and pull-down phone notifications to see that there is a high temperature alert. Exit out of the popup to see a red high temperature warning.
10. Change the minimum value to 90. Check that the half circle sensor progress bar goes to its initial state of half full. Check that the red warning text disappears.

RESULT

☐Step 7: Temperature too low

☐Program alerts using audio ping

☐Program alerts using vibration

☐Popup appears on Temperature Sensor page stating “Skin Temperature too Low”

☐After popup is closed, there is red text stating “Skin Temperature too Low”

☐Phone notification on top banner stating “Skin Temperature too Low”

☐Step 8: Temperature back to normal

☐Half circle sensor progress bar half full

☐Notifications and warnings disappear

☐Step 7 again: Temperature too low

☐Program alerts using audio ping

☐Program alerts using vibration

☐Popup appears on Temperature Sensor page stating “Skin Temperature too Low”

☐After popup is closed, there is red text stating “Skin Temperature too Low”

☐Phone notification on top banner stating “Skin Temperature too Low”

☐Step 8 again: Temperature back to normal

☐Half circle sensor progress bar half full

☐Red warning disappears

☐Step 9: Temperature too high

☐Program alerts using audio ping

☐Program alerts using vibration

☐Popup appears on Temperature Sensor page stating “Skin Temperature too High”

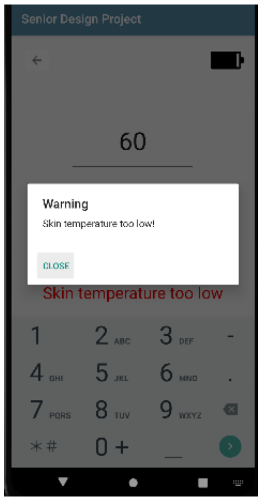
☐After popup is closed, can see red text stating “Skin Temperature too High”

☐Phone notification on top banner stating “Skin Temperature too High”

☒Step 10: Temperature back to normal

☐Half circle sensor progress bar half full

☐Red warning disappears


Senior Design Project

60

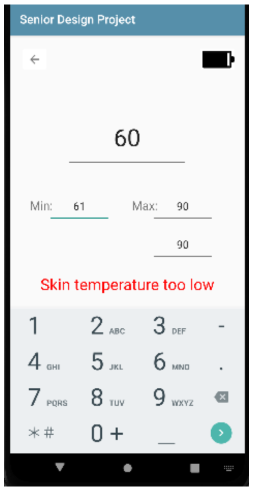
Warning

Skin temperature too low!

CLOSE

Skin temperature too low

Example popup


Senior Design Project

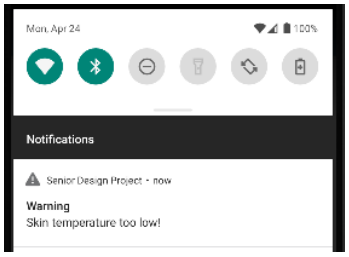
60

Min: 61 Max: 90

90

Skin temperature too low

Example warning


Mon, Apr 24 100%

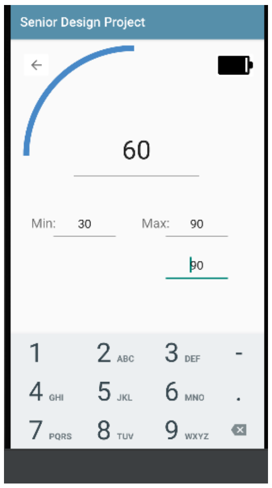
Notifications

Senior Design Project • now

Warning

Skin temperature too low!

Example notification


Senior Design Project

60

Min: 30 Max: 90

90

Example after warning back to normal

After going through the testing procedure, the tests worked as expected. Based on this procedure, the application is working correctly in respect to the sensor progress bar.

APPLICATION / BATTERY USAGE NOTIFICATIONS

OVERVIEW

This testing procedure will be used to determine that the temperature sensor page alerts the user when device battery level is low or out.

PROCEDURE

1. Enable vibration, sound, and notifications on testing device
2. Open application in Android Studio
3. Log into application with user credentials, not instructor credentials

4. When on page with "Welcome to Adaptive Adventures", press the triple line symbol on the top corner of the page to reach the menu.
5. When at the menu, press "Temperature Sensor"
6. Initially, if not connected to hardware, the battery test line is located directly below maximum value line. To test the battery indicator, type a new value between 0-100 onto the line located directly below the maximum value line.
7. Change the battery value to 15. Listen for sound warning, feel for vibration, look for a single notification popup within the temperature sensor page, and pull-down phone notifications to see that there is a low battery alert.
8. Change the battery value to 75. Check that the battery indicator shows quite a bit of battery left.
9. Change the battery value to 0. Listen for sound warning, feel for vibration, look for a single notification popup within the temperature sensor page, and pull-down phone notifications to see that there is a out of battery warning.
10. Change the battery value to 100. Check that the battery indicator shows that the battery is completely full.

RESULT

☐Step 7: Battery Low

☐Program alerts using audio ping

☐Program alerts using vibration

☐Popup appears on Temperature Sensor page stating "Battery Low"

☐Phone notification on top banner stating "Battery Low"

☐Step 8: Battery 75

☐Battery indicator 75% full

☐Step 9: Battery out

☐Program alerts using audio ping

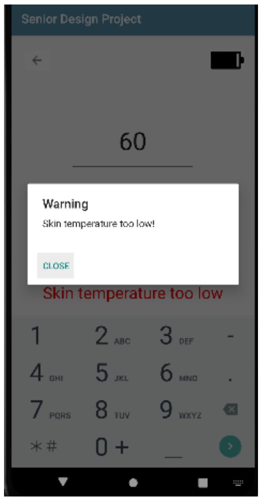
☐Program alerts using vibration

☐Popup appears on Temperature Sensor page stating "Device out of power"

☐Phone notification on top banner stating “Device out of power”

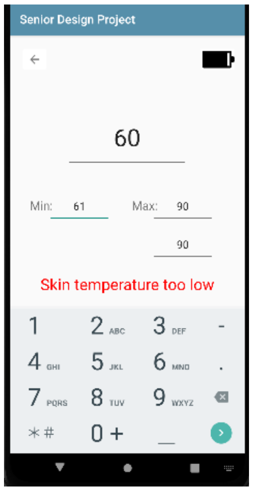
☒Step 10: Battery 100

☐Battery indicator 100% full



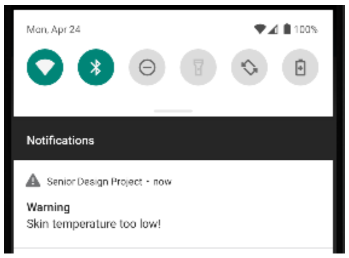
Warning
Skin temperature too low!
Skin temperature too low

Example popup



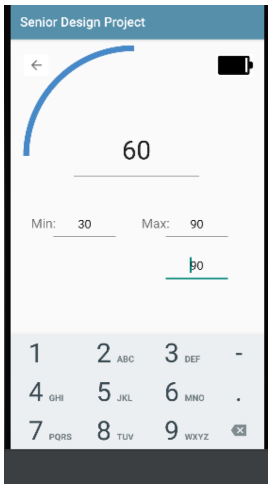
Skin temperature too low

Example warning



Senior Design Project • now
Warning
Skin temperature too low!

Example notification



Example after warning back to normal

After going through the testing procedure, the tests worked as expected. Based on this procedure, the application is working correctly in respect to the sensor progress bar.

APPLICATION / ACCESSIBILITY USER OVERVIEW

This testing procedure will be used to determine that all the pages of the application can be accessed.

PROCEDURE

1. Open application in Android Studio
2. Log into application with user credentials, not instructor credentials
3. When on page with “Welcome to Adaptive Adventures”, press the triple line symbol on the top corner of the page to reach the menu.
4. When at the menu, press “Temperature Sensor”. Check that a half circle progress bar can be seen. Press the arrow in the top left corner and leave to main screen.

5. Open the menu page with the triple lines on the top left corner of the screen and press “Instructor View”. Check that a blank page opens. Press the arrow in the bottom left corner and leave to main screen.
6. Open the menu page with the triple lines on the top left corner of the screen and press “Profile”. Check that a profile page opens. Press the arrow in the bottom left corner and leave to main screen.
7. Open the menu page with the triple lines on the top left corner of the screen and press “Settings”. Check that a settings page opens. Press the arrow in the top left corner and leave to main screen.
8. Open the menu page with the triple lines on the top left corner of the screen and press “Bluetooth Connection”. Check that a blank page opens. Press the arrow in the bottom left corner and leave to main screen.
9. Open the menu page with the triple lines on the top left corner of the screen and press “log out”. Check that the account is logged out and you are taken to the login screen.

RESULT

☐Step 4: Temperature Sensor page opens

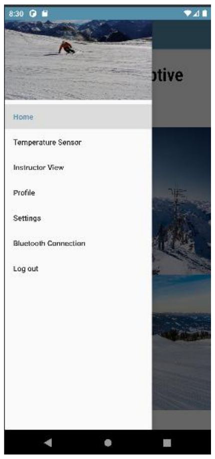
☐Step 5: Instructor View page opens

☐Step 6: Profile page opens

☐Step 7: Settings page opens

☐Step 8: Bluetooth Connection page opens

☐Step 9: Log out button logs user out



Menu page

After going through the testing procedure, the tests worked as expected. Based on this procedure, the application is working correctly in respect to the sensor progress bar.

APPLICATION / USER ACCOUNT CREATION TESTING

OVERVIEW

This testing procedure is used to determine the ability of the AWS backend to facilitate user account creation for users from the front-end application.

PROCEDURE

1. Open application, either on a physical device or from the pre-built emulator on android studio platforms.
2. Click the "Create Account" button at the bottom of the screen.
3. Fill out the information required by screen
    a. Mini tests and results
        i. Not inputting email address: pop-up, refuses to continue until rectified
        ii. Not inputting password at first or second box: pop-up, refuses to continue until rectified
        iii. Passwords that are not 7+ characters in length: pop-up, refuses to continue until rectified
        iv. Mismatched passwords: pop-up, refuses to continue until rectified
4. Memorize secret code from verification email and input into screen.

   a. Mini tests and results
      i. Attempting to continue without code: pop-up, refuses to continue until rectified
      ii. Attempting to continue with wrong code: pop-up, refuses to continue until rectified
5. Fill out data on new screen
   a. Mini tests and results
      i. Attempting to continue without filling out any data: allowed, user profile left blank in these aspects. User account is defaulted to "student" classification.
6. Check AWS Cognito for user in data pool and DynamoDB for user in correct classification database. Ensure data within DynamoDB is accurate to input (email, name, classification, etc.)
7. Delete newly created users from both Cognito and DynamoDB.
8. Repeat with different scenarios (different emulation method, different versions of android, different emails, etc).

Results

User account creation is working as intended, with no issues committing user information to backend services.

APPLICATION / RECIEVING EMAIL VERIFICATIONS

OVERVIEW

Testing procedure to determine the backend ability to send important emails to verified user emails

PROCEDURE

1. Proceed with "Account Creation Testing" until the email verification step. Use variations of emails, including non-accessible and non-existing accounts.
   a. Results for non-accessible/non-existing accounts: account creation is locked, cannot continue without access to the code within the email. Account is seen within AWS Cognito, but as "Unverified". User information is not seen in DynamoDB, as no user profile information has been necessary. Attempts at

creating a different account with the same email address are locked, as a previous account already is registered with that email.

2. Proceed with mini-tests from step 4 in "Account creation".
3. Have multiple "resend verification email" messages sent; check each code for usability.
    a. Results: a code that was previously sent to an account that has requested another becomes defunct, no longer useable. Only the most recently created code is functional, regardless of how soon after any previous ones were created.
4. Attempt to re-verify an account
    a. Results: failure. Pop-up informs user that the account has already been verified, and they may login from the original application window.
5. Attempt to verify an account after 48 hours have passed from the code being sent.
    a. Results: failure. Pop-up informs user that too much time has passed, and they must have a new code sent to verify their account.
6. Repeat with different scenarios (different emulation method, different versions of android, etc.)

RESULTS

User account verification is working as intended. More could be done in user accessibility/readability in regard to error pop-ups, but this is extraneous.

OVERVIEW

Test the ability of AWS to authenticate users and allow them access to the rest of the application.

PROCEDURE

Unlike other testing methods explored within the software aspect of this project, the testing for authentication is built around "penetrative testing" procedures, similar to hacking accounts is done. The following were recorded:

1. Attempts to get into an account with a non-registered email
2. Attempts to log into an account with a registered email without a password
3. Attempts to log into an account with an incorrect password

4. Attempts to “guess”/dictionary-attack the account in question

Results

All attempts to break authentication failed, causing attacking individuals to remain on the login screen and unable to access the rest of the application. The only issues in regard to authentication within accounts is user password/account security, which can be prevented in the future with more strict password requirements (longer password length, inclusion of numbers/special characters, etc.)

OVERALL SYSTEM TESTING

Overiew

This testing was completed to ensure the compatibility of all system component integration. Testing criteria utilized ensures device meets required runtime, alerts users for temperature exceedance, alerts users for low battery life, alerts users for loss of connection between receiver and transmitter, and successfully transmits temperature data between the transmitter and receiver.

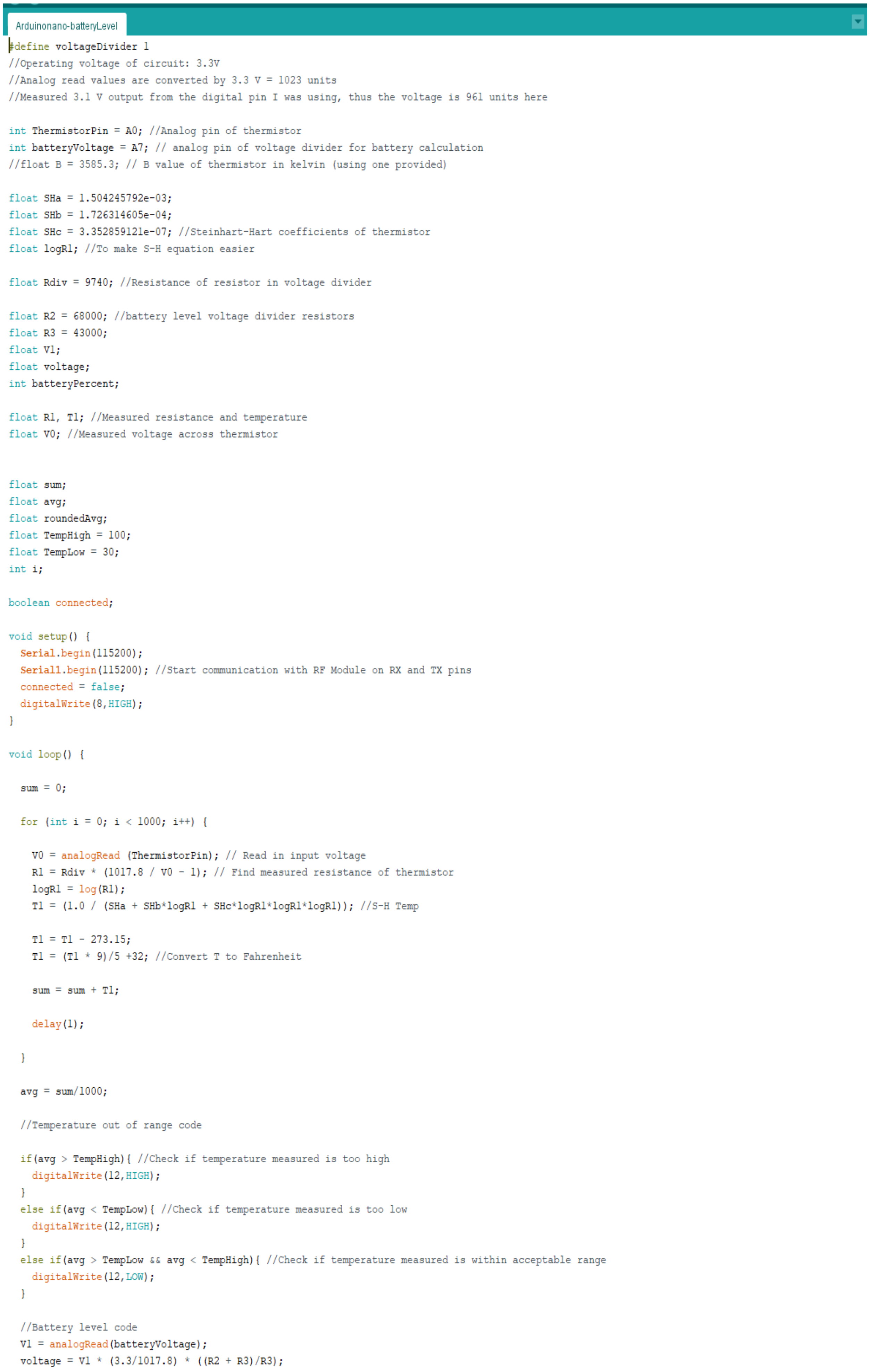
Procedure

The procedure we used for this testing is as follows:

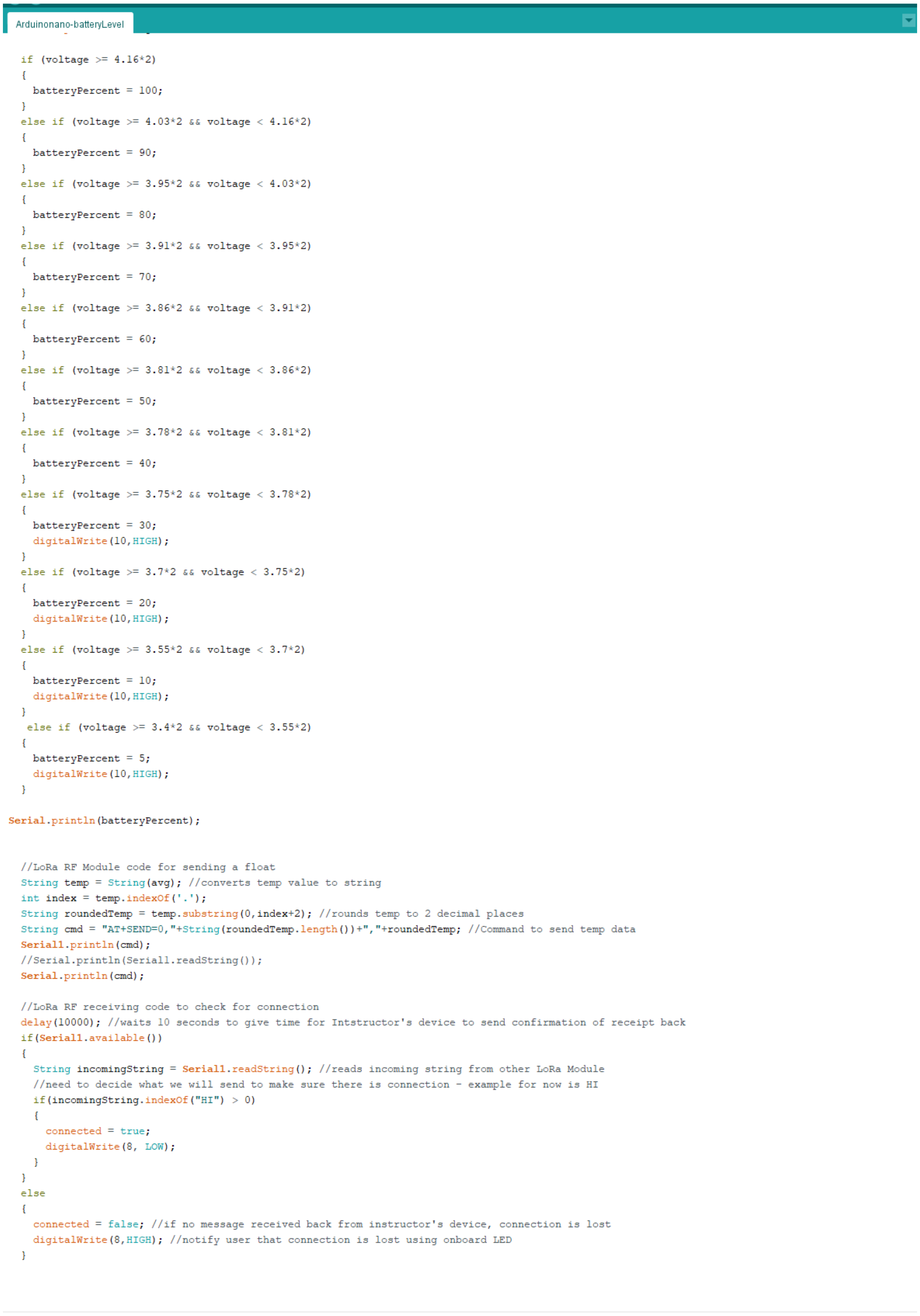
- Have the user put on and power up the transmitting device
- Have the supervising individual connect the receiver to the Arduino IDE serial monitor
- Observe the serial monitor and confirm temperature data communication
- Observe on-board LEDs to ensure alert activation when expected

Software Setup

User Program (Transmitter):



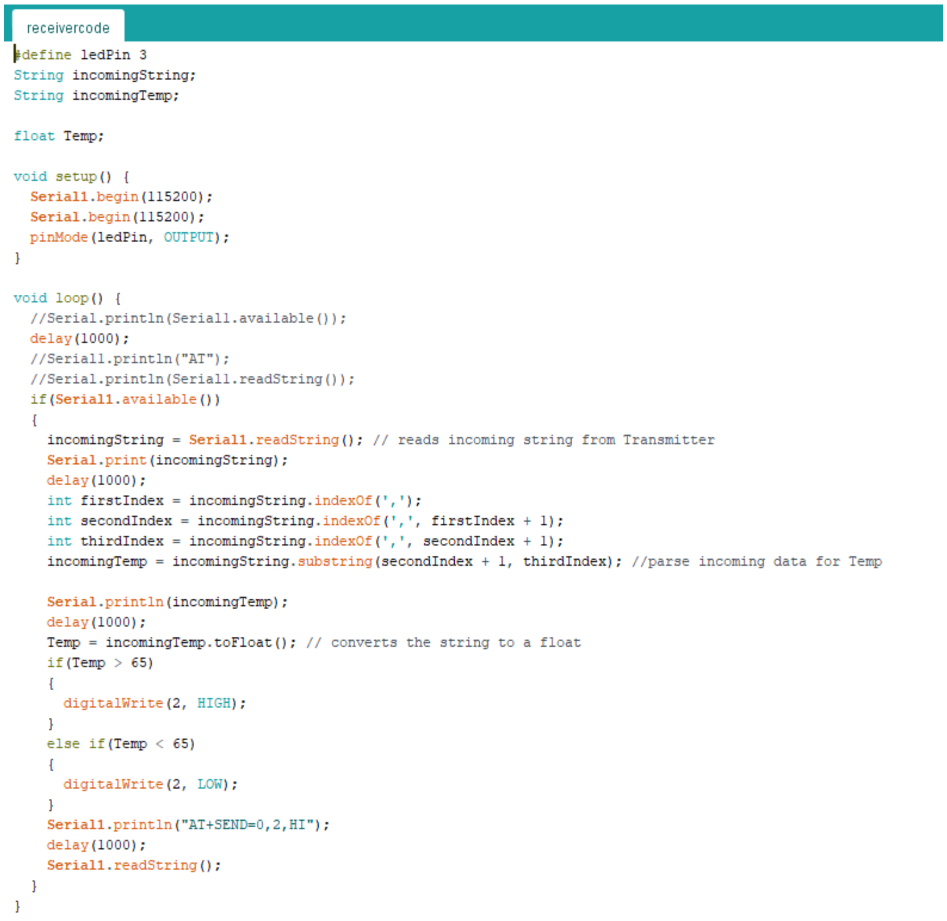
```
#define voltageDivider 1
//Operating voltage of circuit: 3.3V
//Analog read values are converted by 3.3 V = 1023 units
//Measured 3.1 V output from the digital pin I was using, thus the voltage is 961 units here

int ThermistorPin = A0; //Analog pin of thermistor
int batteryVoltage = A7; // analog pin of voltage divider for battery calculation
//float B = 3585.3; // B value of thermistor in kelvin (using one provided)

float SHa = 1.504245792e-03;
float SHb = 1.726314605e-04;
float SHc = 3.352859121e-07; //Steinhart-Hart coefficients of thermistor
float logRl; //To make S-H equation easier

float Rdiv = 9740; //Resistance of resistor in voltage divider

float R2 = 68000; //battery level voltage divider resistors
float R3 = 43000;
float Vl;
float voltage;
int batteryPercent;

float Rl, Tl; //Measured resistance and temperature
float V0; //Measured voltage across thermistor


float sum;
float avg;
float roundedAvg;
float TempHigh = 100;
float TempLow = 30;
int i;

boolean connected;

void setup() {
  Serial.begin(115200);
  Serial1.begin(115200); //Start communication with RF Module on RX and TX pins
  connected = false;
  digitalWrite(8,HIGH);
}

void loop() {

  sum = 0;

  for (int i = 0; i < 1000; i++) {

    V0 = analogRead (ThermistorPin); // Read in input voltage
    Rl = Rdiv * (1017.8 / V0 - 1); // Find measured resistance of thermistor
    logRl = log(Rl);
    Tl = (1.0 / (SHa + SHb*logRl + SHc*logRl*logRl*logRl)); //S-H Temp

    Tl = Tl - 273.15;
    Tl = (Tl * 9)/5 +32; //Convert T to Fahrenheit

    sum = sum + Tl;

    delay(1);

  }

  avg = sum/1000;

  //Temperature out of range code

  if(avg > TempHigh){ //Check if temperature measured is too high
    digitalWrite(12,HIGH);
  }
  else if(avg < TempLow){ //Check if temperature measured is too low
    digitalWrite(12,HIGH);
  }
  else if(avg > TempLow && avg < TempHigh){ //Check if temperature measured is within acceptable range
    digitalWrite(12,LOW);
  }

  //Battery level code
  Vl = analogRead(batteryVoltage);
  voltage = Vl * (3.3/1017.8) * ((R2 + R3)/R3);
```


Arduinonano-batteryLevel

```
  if (voltage >= 4.16*2)
  {
    batteryPercent = 100;
  }
  else if (voltage >= 4.03*2 && voltage < 4.16*2)
  {
    batteryPercent = 90;
  }
  else if (voltage >= 3.95*2 && voltage < 4.03*2)
  {
    batteryPercent = 80;
  }
  else if (voltage >= 3.91*2 && voltage < 3.95*2)
  {
    batteryPercent = 70;
  }
  else if (voltage >= 3.86*2 && voltage < 3.91*2)
  {
    batteryPercent = 60;
  }
  else if (voltage >= 3.81*2 && voltage < 3.86*2)
  {
    batteryPercent = 50;
  }
  else if (voltage >= 3.78*2 && voltage < 3.81*2)
  {
    batteryPercent = 40;
  }
  else if (voltage >= 3.75*2 && voltage < 3.78*2)
  {
    batteryPercent = 30;
    digitalWrite(10,HIGH);
  }
  else if (voltage >= 3.7*2 && voltage < 3.75*2)
  {
    batteryPercent = 20;
    digitalWrite(10,HIGH);
  }
  else if (voltage >= 3.55*2 && voltage < 3.7*2)
  {
    batteryPercent = 10;
    digitalWrite(10,HIGH);
  }
   else if (voltage >= 3.4*2 && voltage < 3.55*2)
  {
    batteryPercent = 5;
    digitalWrite(10,HIGH);
  }

Serial.println(batteryPercent);


  //LoRa RF Module code for sending a float
  String temp = String(avg); //converts temp value to string
  int index = temp.indexOf('.');
  String roundedTemp = temp.substring(0,index+2); //rounds temp to 2 decimal places
  String cmd = "AT+SEND=0,"+String(roundedTemp.length())+","+roundedTemp; //Command to send temp data
  Serial1.println(cmd);
  //Serial.println(Serial1.readString());
  Serial.println(cmd);

  //LoRa RF receiving code to check for connection
  delay(10000); //waits 10 seconds to give time for Intstructor's device to send confirmation of receipt back
  if(Serial1.available())
  {
    String incomingString = Serial1.readString(); //reads incoming string from other LoRa Module
    //need to decide what we will send to make sure there is connection - example for now is HI
    if(incomingString.indexOf("HI") > 0)
    {
      connected = true;
      digitalWrite(8, LOW);
    }
  }
  else
  {
    connected = false; //if no message received back from instructor's device, connection is lost
    digitalWrite(8,HIGH); //notify user that connection is lost using onboard LED
  }
```

The User/Transmitter code above first declares all of the global variables that will be used in the main section of code. Then, it takes measurements from the temperature sensing element over one second and averages that data for a more accurate reading. After getting the temperature reading, the code goes through a series of “if” statements to decide if the temperature reading is within acceptable ranges or not and powers on an LED if the reading is outside of the acceptable range. The next series of “if” statements takes the voltage reading of the batteries to calculate a battery percentage and turns on an LED if low battery is detected. Finally, the last section of code transmits the data to the receiver / supervisor using the LoRa RF module and checks for connection by waiting for a response back from the receiver device.

Supervisor Program (Receiver):



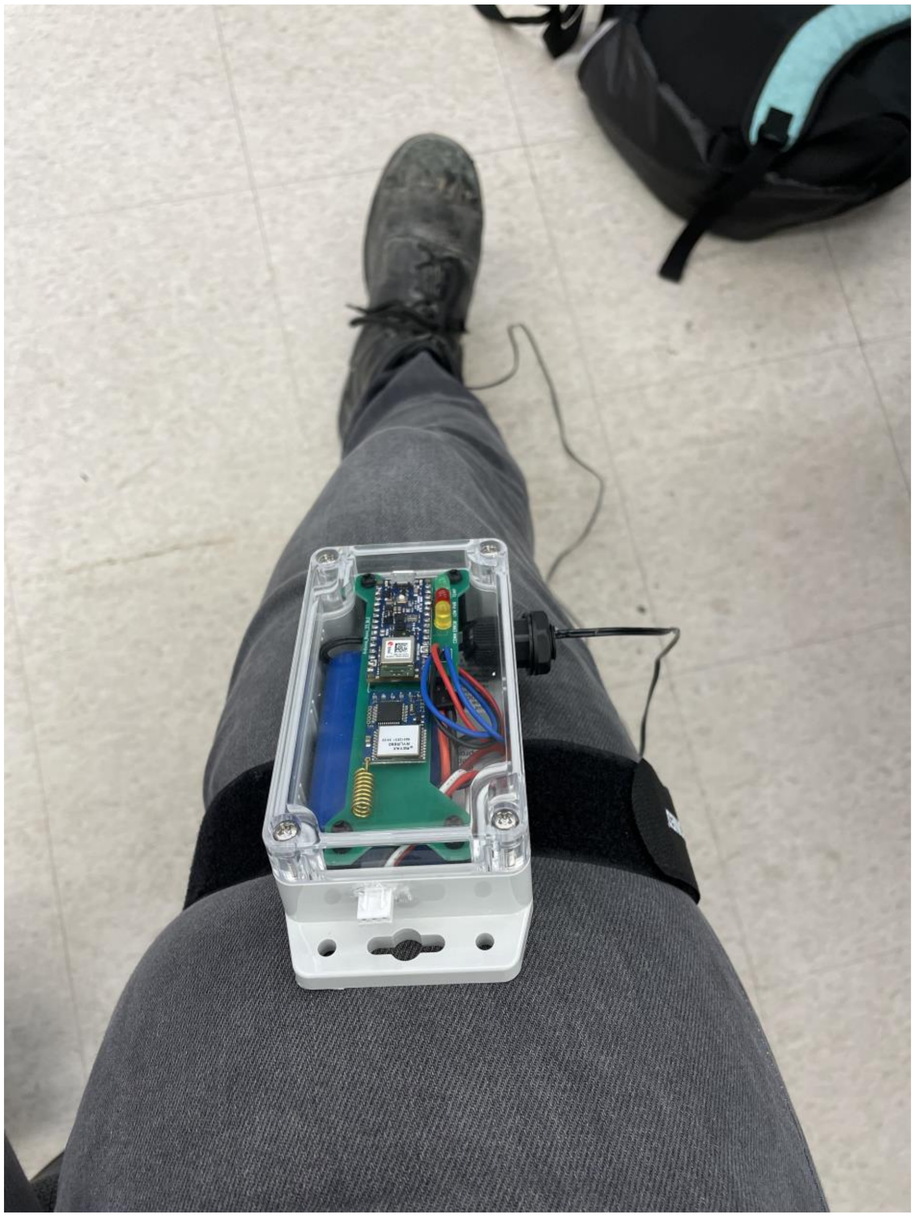
receivercode

```
#define ledPin 3
String incomingString;
String incomingTemp;

float Temp;

void setup() {
  Serial1.begin(115200);
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  //Serial.println(Serial1.available());
  delay(1000);
  //Serial1.println("AT");
  //Serial.println(Serial1.readString());
  if(Serial1.available())
  {
    incomingString = Serial1.readString(); // reads incoming string from Transmitter
    Serial.print(incomingString);
    delay(1000);
    int firstIndex = incomingString.indexOf(',');
    int secondIndex = incomingString.indexOf(',', firstIndex + 1);
    int thirdIndex = incomingString.indexOf(',', secondIndex + 1);
    incomingTemp = incomingString.substring(secondIndex + 1, thirdIndex); //parse incoming data for Temp

    Serial.println(incomingTemp);
    delay(1000);
    Temp = incomingTemp.toFloat(); // converts the string to a float
    if(Temp > 65)
    {
      digitalWrite(2, HIGH);
    }
    else if(Temp < 65)
    {
      digitalWrite(2, LOW);
    }
    Serial1.println("AT+SEND=0,2,HI");
    delay(1000);
    Serial1.readString();
  }
}
```

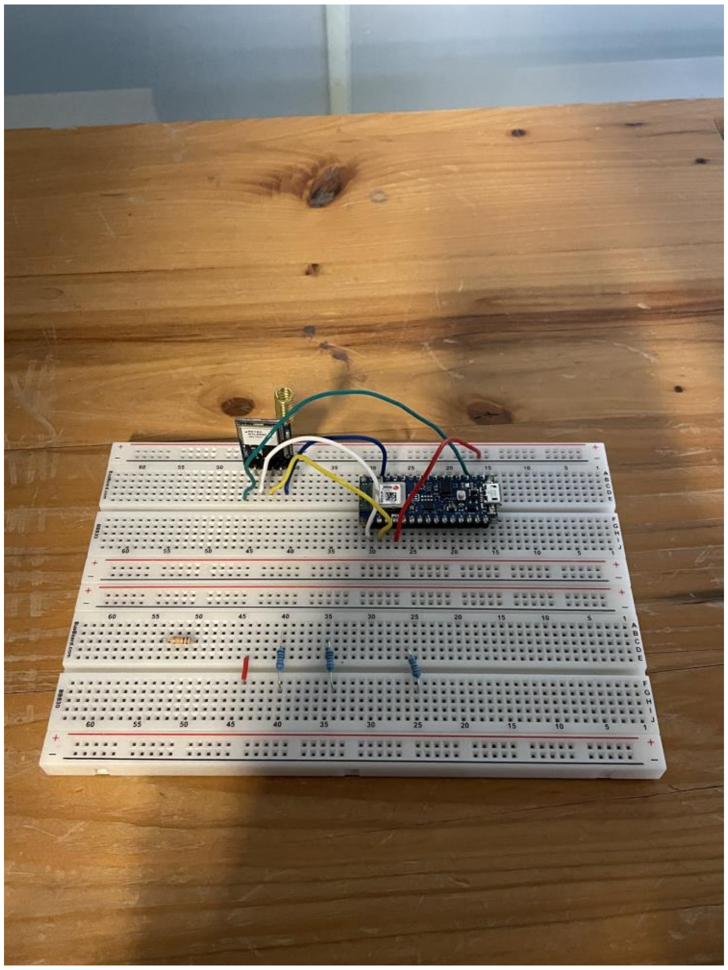
The code performs the following actions: initializes the receiver, checks to see if data is being transmitted, reads the incoming data, parses the data to obtain the actual temperature data measurement, formats the temperature data accordingly, and visualizes the temperature data on the Arduino IDE serial monitor.

Hardware Setup

The hardware setup for the overall system testing can be seen below:



The user straps the device to the desired part of the body to either obtain a foot or hand temperature data measurement. The user then places the probe on an adhesive patch and applies the patch to the desired location. Finally, the device is powered on using the push button.



The receiver is plugs into the computer or laptop. The supervising individual then opens Arduino IDE, connects the device using the proper port and board selection (Arduino Nano 33 BLE), and opens the serial monitor.

Results

This testing was extremely successful as the device was able to transmit data over a range of at least 60m. Below are pictures showing the distance range used in testing as well as the visualization of temperature data on the Arduino IDE.