ELEN30013 Temperature Controller Project Report

Group 4 Friday 2:30pm
Didi Chi (683820)
Shiyu Zhang(683557)

0. Abstract

Temperature controller has a wide range of applications in both daily life and laboratories. This report studies the methodologies behind the temperature controller. Single-stage thermoelectric cooler (TEC) has been used in this project to change the temperature on a small surface due to its wide working temperature range and its property of allowing bidirectional current to either heat up or cool down a surface. The whole system consists of five parts: controller, signal-conditioning circuit, actuator, user interface and the TEC. Controller is in charge of making decisions depending on the user input and the reading from the sensor. Actuator circuit generates current to drive the TEC. Signal-conditioning circuitry provides temperature reading to the controller. User interface is a generalisation of both the user input circuitry and the display module. The design allows users to set a temperature between 5 celsius degree and 45 celsius degree; the controller collects data from the sensor in the signal conditioning circuit and sends signals to the actuator to either heat up or cool down the aluminium slab by TEC, depending on the set temperature.

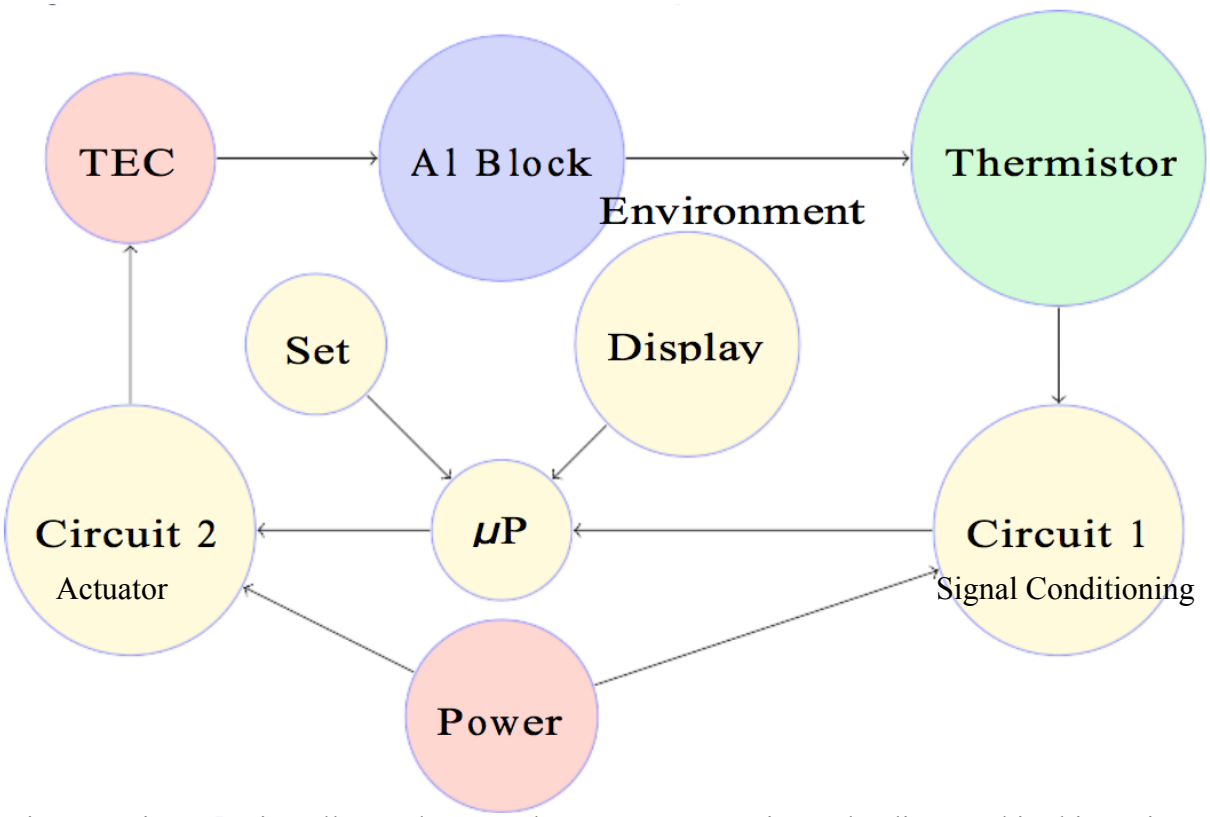


Figure 0 Five parts in yellow colour are the components going to be discussed in this project

Table of Contents

1. Introduction

1.1 Motivation

The temperature controller is a device that changes the temperature and holds that to a desired value which is set by user. In household, temperature controller can be used to control the temperature of water (drinks) such as maintaining a cup of drink at a certain commanded temperature ("Temperature Controller Basics Handbook | Instrumart", 2016). Also, it has many applications in industries and laboratories. For example, in biochemistry experiment, polymerase chain reaction (PCR) requires multiple heating and cooling cycles, and our temperature controller, which monitors the temperature of the experiment settings in realtime and hold the temperature at the desired temperature by the users, will be well applied to this scenario. In industries, temperature controller are also widely used in food packaging, food processing, thermo-forming machines, etc ("Temperature Controller Basics Handbook | Instrumart", 2016).

1.2 Background

Thermoelectric cooler (TEC) has two sides. As the DC current flows through the device, heat wis brought from one side to the other, so that the temperature of one side decreases while that of the other side increases. The heating side is attached to a heat sink so that the temperature is maintained. Due to safety concerns, the maximum current in both directions is set to 1.5 A in this project, and the maximum voltage allowed is set 2.5V.

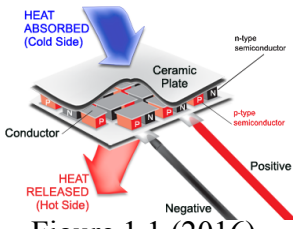


Figure 1.1 (2016)

The actuator drives TEC should be able to generate current in two directions. This circuit turns control signal into large current to raise or lower the temperature of the AI block.

Signal conditioning circuit maps temperature to a range of values that can be read by the controller. The accuracy of the system is largely dependent on this component. It is desired that sufficient sampling is done so that the temperature reading is accurate enough to be applied in the scenarios mentioned above. Because NE555 generates stable square waves that its frequency depends on the resistor cascaded with it, signal conditioning circuit is able to convert temperature to frequency so that it can be understood by Arduino, which can read frequency up to 8 MHz by using third party libraries.

A nice user interface is also significant. A well implemented interface will bring much convenience to the users. Two ways of displaying readings will be used: LCD display and a remote interface feature.

1.3 Objectives

Range: 5 celsius degree to 45 celsius degree

Accuracy: 0.01 celsius degree.

Speed (reaction time): heating up from 5 celsius degree to 45 celsius degree within 2 minutes; cooling down from 45 celsius to 5 celsius degree within 2 minutes.

1.4 System Setup

Controller: Arduino Uno is a microcontroller board based on the ATmega328P with 14 digital input/ output pins, 6 analog inputs, a 16 MHz clock ("Arduino - ArduinoBoardUno", 2016).

Actuator: H-bridge: Four 10k resistors, four TIP41C npn transistors and four BC548B npn transistors.

Signal-conditioning: one NE555 oscillator, one NTC resistive thermistor, capacitors and resistors.

User interface: Display method: Liquid Crystal Display Module and a remote interface feature (connected by Iduino Yun Wifi shield).

User input: 10k Potentiometer, a push button.

1.5 Content and Organisation

This report discusses the design of temperature controller in following sections: user interface, signal conditioning, actuator, control system and conclusion. Each section will discuss the methodologies and result of the tests done with these components.

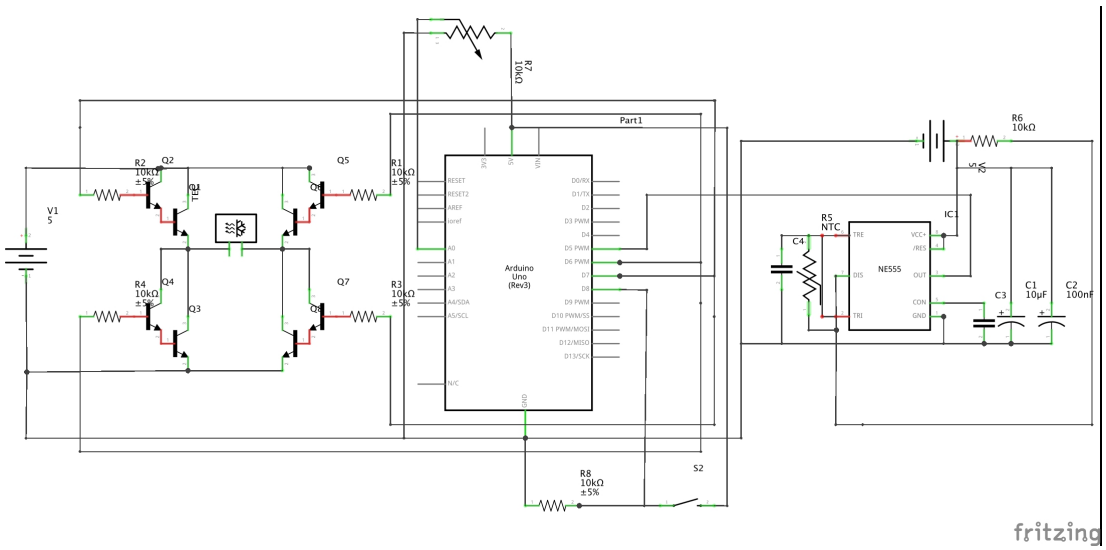


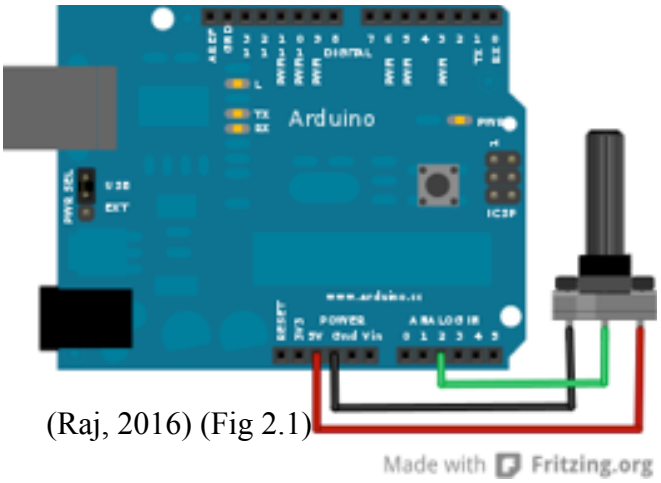
Figure 0. Overall Schematics (without LCD module, which is shown in the following part of this report separately)

2. User Interface

2.1A Methodology: Liquid Crystal Display and Potentiometer

Local user input: potentiometer.

Hardware setup:



(Raj, 2016) (Fig 2.1)

The potentiometer was connected to the central processing unit of the system in the manner of the schematic above shows. The signal pin of the potentiometer was connected to an Arduino analog input pin and the sides terminals were connected to +5V and GND.

Software setup:

The function anaologRead() was used to read the analog value that outputted by the potentiometer. Recall that analogRead() return a raw value of the analog pin ranging from 0 to 1024, the function map() was used to remap this output value to the user desired input range 5 to 45°C.

```
/*USER IO SESSION*/

analogVal = analogRead(analogIn);
int set = map(analogVal,0,1024,5,45);
```

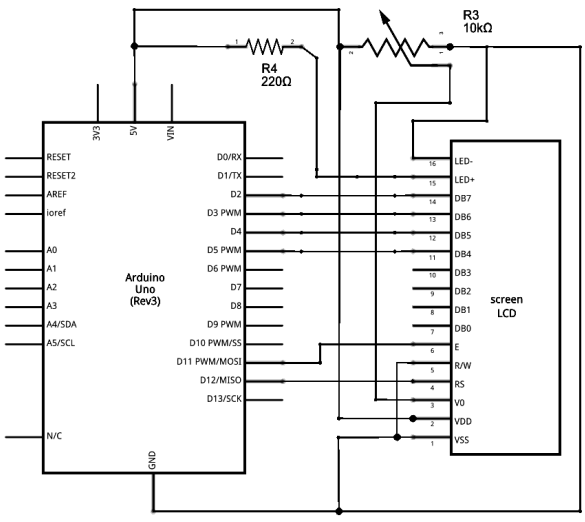
Figure 2.2

2.1.B Local user output: Liquid Crystal Display

A liquid crystal display was used to display status of the system including: real time system temperature(integer), user set temperature(integer), and the mode of the system(heat/cool).

Hardware setup:

Figure 2.3 was the schematic that has been referred to during the design it requires connections to 3 of



(Figure 2.3) ("Arduino - HelloWorld", 2016)

the PWM pin from the Arduino, in our case, pin 3,9 and 11. A 10K ohm potentiometer has

```
void LCD_SETUP(){
  lcd.begin(16, 2);
  // Print a message to the LCD if setup successfully.
  lcd.print("REAL_TIME  USER");
}

void LCD_DISPLAY(int temp_real_time, double USER_input, int mode){
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  String USER_input_string = String(USER_input);
  String temp_real_time_string = String(temp_real_time);
  String mode_string = String(mode);
  String total = temp_real_time_string + "    " + USER_input_string +" "+ mode_string;
  lcd.print(total);
}
```

Figure 2.4

been used to connect the VSS VDD and V0 pin on the LCD screen in order to adjust the contrast of the text on the screen.

Software Setup:

The official LiquidCrystal library is used in order to display string on the screen. Two functions was abstracted in the Arduino code: void LCD_SETUP(void) and void LCD_DISPLAY(int temp_real_time, double USER_input, int mode). LCD_SETUP() was placed in the setup() function of the Arduino sketch to print the first line on the LCD to indicate what values are being displayed on the screen, and LCD_DISPLAY was placed under the loop() function of the Arduino sketch to update real time values mentioned before, on the second row of the LCD.

2.1.C Methodology: The Arduino Yun wireless local area network shield

In order to implement the remote user interface feature, an Arduino Yun shield was used in our system.

Hardware setup:

The wifi shield was placed right on top of the Arduino Uno board, with all the pins connected to on another, then, an external 12V power supply was required to power the Arduino Uno board for the system to run.

The wifi shield was configured by connecting to its OpenWRT sub system's web page for wireless network connection etc.

```
    HttpClient client;
    client.get("http://www.dweet.io/dweet/for/YUN_ANALOG_IN_DWEETING_2?UserInput="
    +String(set)+"&Frequency="+String(frequency)+"&temperature="
    +String(temperature)+"&mode="+String(mode));
```

Figure 2.5

Software setup:

The official Bridge and Httpclient library was used in the Arduino code. The Bridge library was used for the information exchange between the shield and Arduino board. The HttpClient library was used in order to send http request to the Dweet.io API, our remote user monitor platform. ("Arduino - HttpClient", 2016)

2.2 Results

As expected, the LCD was displaying the information mentioned before, and the user was able to adjust system's temperature by using the potentiometer. What is more, the website in Dweet.io was updating at the rate of approximately once every second.

2.3 Discussion

By using these three components, the user of this temperature control system will be able to monitor and interact with the system efficiently.

However, it has also been observed that the LCD connection was not very stable since jumper wires and breadboard were used for the connection. This can be improved by soldering the LCD onto a Vero board or integrate it on to a PCB to increase its reliability.

Moreover, since internet connectivity has been implemented as one of the features to the system, a remote input functionality could also be added as an improvement. To obtain that, the Amazon Web Service and its SDK for Arduino Yun could be used. By accessing AWS's IOT API, the Arduino board will have access to precreated database like MySQL and a web application can be created to modify the database data in order to control the system. ("aws-iot-device-sdk-arduino-yun", 2016)

3. Temperature sensing and signal conditioning

3.1 Methodology

Thermistor

A NTC thermistor was used as the temperature sensor of the system. It has been discovered that the resistance of the thermistor will change according to the temperature that it exposed to, in the relationship of the higher the temperature, the lower the resistance, as figure 3.2 shows.

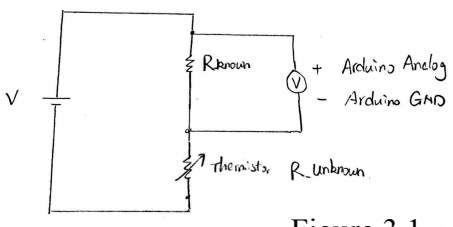


Figure 3.1

This measurement resistance of the thermistor was taken by the method of voltage divider (Figure 3.1), which is putting the thermistor in series of a known resistor, and power source, by using the following calculation process, the relationship between temperature and the resistance of the thermistor was recorded.

However, this straight forward method has two major downsides. First of all, the range of the reading is limited to the Arduino measure capacity, secondly, the accuracy of the

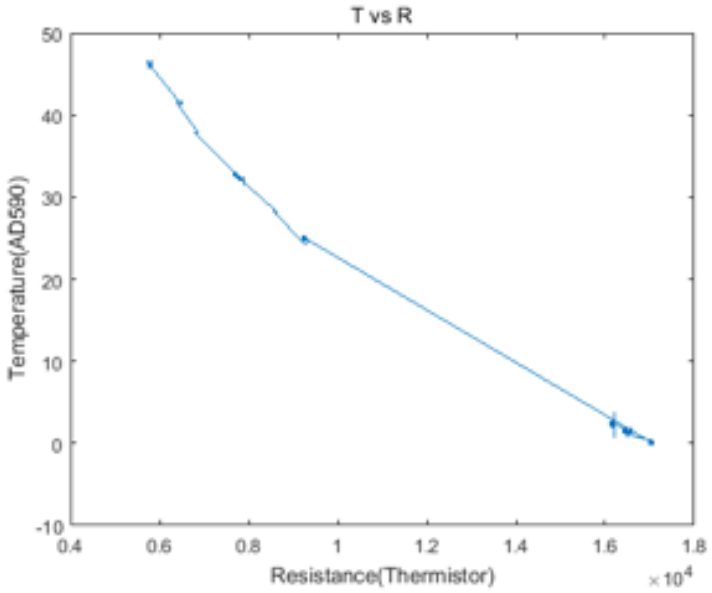


Figure 3.2

$$I_{R_{known}} = \frac{V_{Rknown}}{Rknown}$$

$$V_{thermistor} = V - V_{Rknown} \quad \text{(Cal 3.1)}$$

$$R_{thermistor} = \frac{V_{thermistor}}{I_{Rknown}}$$

calculation is depending on the resistance value of the known resistor. Therefore, a NE555 timer was adopted into the system for signal conditioning purpose in order to obtain more accurate temperature information.

NE555 timer hardware setup

It is known that under Astable configuration, the frequency of the outpit waveform is depending on the resistor and capacitor in the system. The relationship between them was given by the official specification(reference).

$$f = \frac{1}{T} = \frac{1.44}{(R_A + 2R_B)C}$$

(Cal.3.2) ("NE555 Timer", 2016)

To obtain a more accurate relationship between the temperature and the thermistor resistance value, an Arduino digital pin will be used to read the frequency output of the NE555 timer, which connected to the thermistor in the configuration.

Expected output frequency

As measured in the previous session, the typical resistance value of the thermistor in the system's temperature range is from 6 to 18K Ohm. To get a wide range reading of frequency, Ra was designed to be 10kOhm and the thermistor was placed in Rb's position, C was designed to be 10nF. This will result in a frequency range from f(low)= 1.44/(10*10e3 + 2*18*10e3)*10*10e-9 = 3130Hz to f(high)= 1.44/(10*10e3 + 2*6*10e3)*10*10e-9 = 6545Hz.

Software setup

According to research, a third party Arduino frequency reading library was used to read the frequency from the NE555's output pin. The library invoked a hardware 16 bit counter in Arduino for accurate frequency measuring and it has been proved that the function provided by the library is capable to reading frequency up to 8MHz, which is suitable for our application ("Arduino Frequency Counter Library", 2016).

The use of the library in the Arduino code is attached in the Fig 3.2 below.

```
98      FreqCounter::f_comp = 8;
99      FreqCounter::start(1000);
100     while(FreqCounter::f_ready == 0);
101     unsigned long frequency = FreqCounter::f_freq;
```

Figure 3.3

Record the relationship (calibration)

In order to record the relationship between the frequency and the temperature, the thermistor was tied down to the surface of the TEC and AD590. By using a current source, the TEC was heated up to 50 Celsius degrees and cooled down to 3 degrees, while using the Arduino to measure the frequency of the output pin of NE555 and temperature from AD590, as the diagram shown above (Figure 3.4).

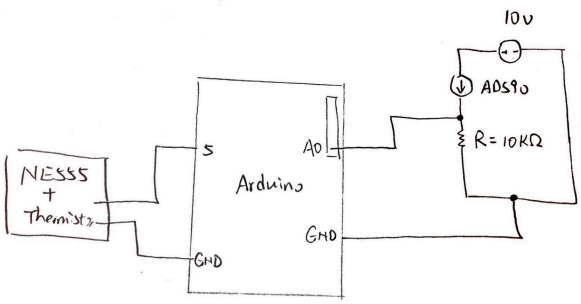


Figure 3.4

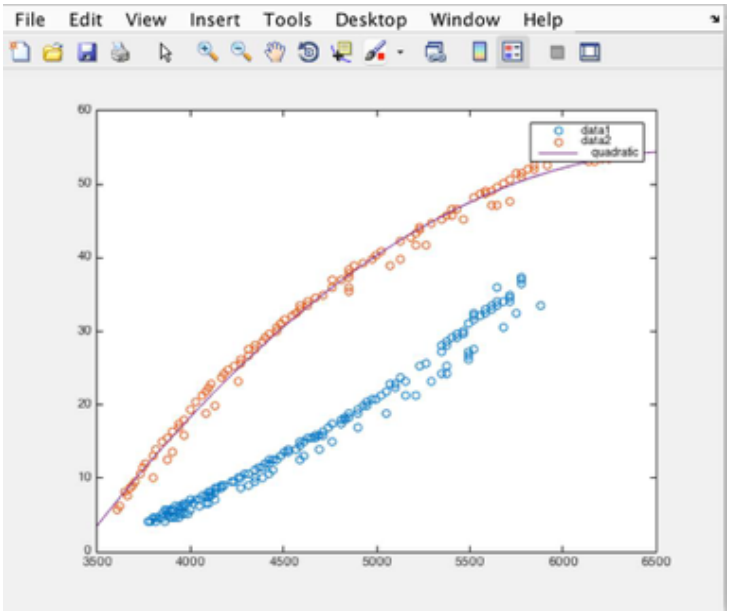


Figure 3.5

After importing the collected data into Matlab, it can be seen the curve for heating up does not coincide with the one for cooling even when the temperature is the same. This might due to the dynamic response of the thermistor to the temperature change. This problem can be solved by using different equations for the relationship for heating up and cooling down in the control system, and this will be discussed in later sessions.

The curve fitting tool in Matlab was used to generate a numerical relationship between the temperature and the output frequency. As shown in the Fig 2.3, the output frequency was what has been expected.

3.2 Result

Here is the result of the calibration. Matlab curve fitting curve was used to generate function that fit the curves the most, and the best fit has been found to be using a fourth order polynomial function.

Curve for heating up

p1 = -1.243e-12; p2 = 2.528e-08; p3 = -0.0001957; p4 = 0.6987; p5 = -944.8 ;

heat_temperature = p1 * pow(frequency,4) + p2 * pow(frequency,3) + p3 * pow(frequency, 2) + p4 *frequency + p5;

Curve for cooling down

q1 = 4.906e-13; q2 = -9.443e-09; q3 = 6.947e-05; q4 = -0.2192; q5 = 250.9;

cool_temperature = q1 * pow(frequency,4) + q2 * pow(frequency,3) + q3 * pow(frequency, 2) + q4 *frequency + q5;

These has been implemented into the Arduino code to calculate the temperature from its detected frequency.

3.3 Discussion

As mentioned before, the voltage divider method could have been used for temperature detection, but it wasn't accurate enough due to the following reasons. A. the voltage range is too narrow. It has been researched that Arduino has a 10-bit ADC on board ("Arduino - ArduinoBoardUno", 2016), which means it has 2^10 -1 = 1023 quantization levels. This also indicates that the voltage reading from an Arduino analog pin can only be divided up to 1023 levels. Comparing to the NE555's frequency range, which is from 3130Hz to 6545Hz, 1023 levels is not accurate enough. B. the accuracy of resistance calculated by the voltage divider method depends on the tolerance of resistor. It has been demonstrated in Cal.2.1 that the resistance is calculated based on a known resistor. Nevertheless, there tolerance of a resistor from the market can range from 2% to 20%, which will introduce unwanted new

error to the system's measurement. Since the NE555 method doesn't calculate the resistance directly, this problem was avoided.

4. Actuator

4.1 Methodology: H-Bridge

H-Bridge is used in the actuator circuit in the system. An H-Bridge can be used to drive TEC, which is represented as RL in Figure 3.1, bidirectionally. When A, D are closed, the current will flow from left to right across TEC, while C and B are closed, the current will flow from right to left across TEC. Based on the testings on TEC in week 3, it is known that changing the direction of the current through TEC allows to achieve either heating up or cooling down it.

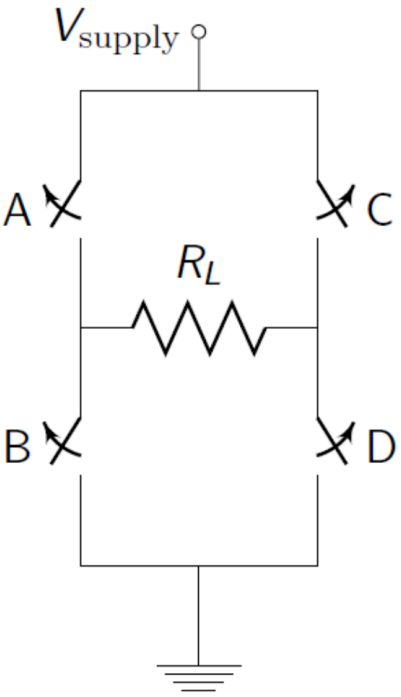


Figure 4.1 Basic structure of a H-bridge

At the beginning stage of designing H-Bridge circuit, four npn transistors were used as A,B,C,D switches in Figure 3.1. We were provided by three kinds of npn transistors: TIP41C, MJE200 and BC548B. The collectors (C) of transistors of A and C are connected to the power supply and their emitters (E) are connected to collectors (C) of B and D. The emitters (E) of B and D are connected to a common ground. One input voltage controlling the status of A and D and the other one controlling the status of C and B. They are connected to resistors, which are concatenated with the bases of these transistors (as what is shown in Figure 4.2). Based on previous workshops, it is known we need around 1A to heat up or cooling down TEC fluently. According to specifications of these npn transistors ("BC548B", 2016), BC548B is not powerful enough to drive TEC because the maximum collector current it can have is 100 mA. The other two should work fine for the circuit.

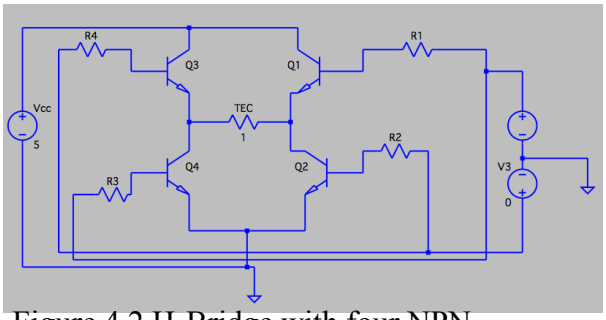


Figure 4.2 H-Bridge with four NPN transistors

Therefore, A,B,C and D are replaced by TIP41Cs. Varying the resistance of the four resistors connected to the bases of the transistors from 500 Ohm to 100 Ohm, with one input voltage turned on with 5V, we got the following result from the simulation.

Resistance (Ohms)	I (TEC)	I (Arduino)
100	1.499A	65.4mA
200	1.081A	35.5mA
300	871.5mA	24.58mA
400	736.9mA	18.87mA
500	641.7mA	15.342mA

Table 4.1 A summary of the current across TEC and Arduino with different size of resistors connecting to transistors bases.

According to the simulation result, when resistance is 200 ohms the current across TEC is around 1A. However, the current through Arduino is 35.5mA, which is larger than the Arduino's current limit 20mA. Therefore, the resistance should be between 300 ohms and 400 ohms. 380 ohm resistor is chosen as the resistor in this circuit. In this case, only 760mA current will running through TEC. After connecting the circuit to a breadboard, and using a dummy load (1 ohm resistor) as the TEC, the measured voltage across the dummy load (which is equal to the current across the load) varied from 0.5V to 0.7V. Consider we might use PWM (analogWrite() function from Arduino), it would be better that the circuit generates current larger than this value so that we can have a wider range of current available for driving TEC. Therefore Darlington structure is introduced to further amplify the current so that we can have enough current and not exceeding the current limit of

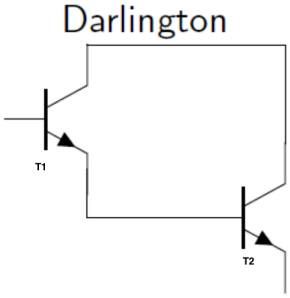


Figure 4.3 Darlington Structure

Resistance(Ohm)	I(TEC)	I(Arduino)
10k	1.3478776A	458.55777μA
15k	1.1843177A	321.26353μA
18k	1.110014A	273.64725μA
22k	1.0284261A	229.24428μA

Table 4.2 A summary of the current across TEC and Arduino with different size of resistors connecting to transistors bases.

Arduino. BC548B is chosen for T1 (in Figure 4.3) and

TIP41C is chosen for T2. This time, we do not need to worry about that BC548B is not powerful enough since TIP41C following it will amplify the current with a forward current gain around 75 (refer to the specification of TIP41C). The resistor need to be increased or the current across TEC will be too large and damage TEC. Therefore, varying the resistance value among the resistors that are available to us (10k, 15k, 18k, 22k), simulation result are shown in the Table 4.2. Based on the data achieved from previous testing on H-bridge without Darlington transistors, there was always a small discrepancy between the actual result and that from simulation.Therefore we decided to use 10k resistors connecting to the bases of the npn transistors in the circuit.

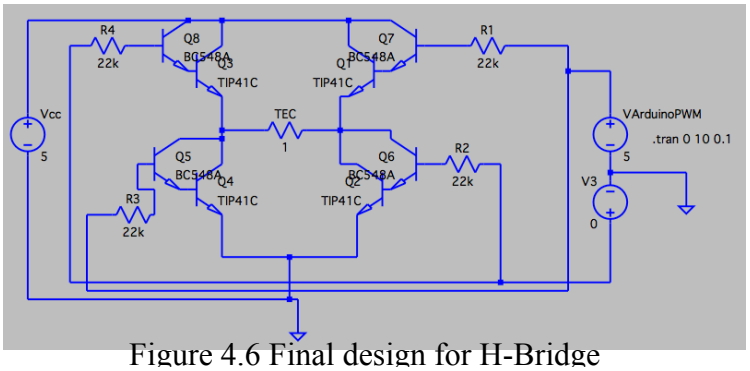


Figure 4.6 Final design for H-Bridge

4.2 Result

Two sets of tests were done: one run of testing was done on a breadboard and the other sets of testing was done after all the components were soldered to a veroboard. Same as the testing done before, a 1Ohm dummy load was used as a substitution of TEC.

The results from these testing were consistent with the result from the LTSpice simulation. There was 1.3V voltage drop across the dummy load when one input voltage was turned on with 5V (from the desk bench DC power supply). By alternatively turning on the two input voltages, a change in direction of the voltage drop could be observed.

```
#define HEAT 6
#define COOL 7
void setup() {
  // put your setup code here, to run once:
  pinMode(HEAT,OUTPUT);
  pinMode(COOL,OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(HEAT,HIGH);
  digitalWrite(COOL,LOW);
}
```

Figure 4.7 A simple program for heating up TEC

After tests on breadboard and veroboard were done, we connected the H-Bridge to TEC and let Arduino's digital output become the voltage inputs. A simple test program for powering the H-Bridge was written. It took around 30 seconds to heat up the TEC from room temperature (25 degrees) to 45 degrees while it took around 2 minutes to cool down from 45 degrees to 5 degrees. In general, heating up TEC took less time than cooling down it. The voltage across two ports of TEC varies from 1.4V to 1.7V, which is larger than the simulation result (across the dummy load). This might be due to the internal resistance of TEC is greater than 1Ohm.

The power dissipated in the system is estimated to be 5V * 1.3A = 6.5W

3.3 Discussion

Limitations of this H-Bridge circuit:

Since there are two inputs to this H-Bridge, there will be totally four combinations of inputs.

A	B	C	D
1	1	1	1
0	1	1	0
1	0	0	1
0	1	1	0

Table 4.3 All four possible combinations of inputs to H-Bridge

The first combination will be dangerous because it will create a short circuit and damage the components. Our way of preventing this short circuit happening is to program the controller so that only one input will be turned on each time (as what is shown on the left). For example, when current temperature is lower than the set temperature, the input for heating TEC is turned on, and the other one is off. When temperature goes beyond the target temperature, instead of turning on the cooling input, both inputs are turned off, considering the time between Arduino executing the if statement and turning on the other input will result in a short period of short circuit result in malfunction of the H-Bridge

```
if(abs(temperature - set) < 1){
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
}
else if(temperature < set){
  digitalWrite(7,HIGH);
  digitalWrite(6,LOW);
} else {
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
}

if(abs(temperature - set) < 1){
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
}
else if(temperature > set){
  digitalWrite(7,LOW);
  digitalWrite(6,HIGH);
} else {
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
}
```

Figure 4.8

Comparison with other choice of actuator circuit (Push-Pull amplifier):

Another common method for driving a load bidirectionally is the push-pull amplifier.

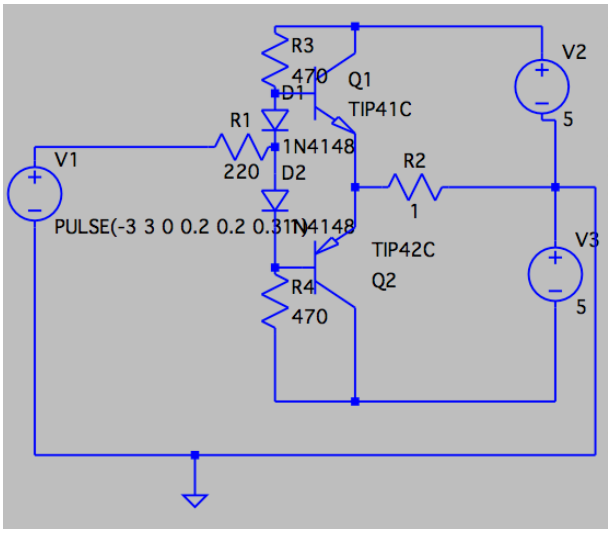


Figure 4.9 Push-pull amplifier

In this design, Q1(NPN) “pushes” current from V2 through TEC to GND, while Q2(PNP) “pulls” current from GND through TEC to V3. V1 is the control signal from Arduino. R2 is TEC. The diodes were added to cancel the crossover distortion.

Comparing an H-Bridge actuator and a push-pull amplifier one, there are a few reasons for us to

choose H-Bridge as our driving circuit.

1. Arduino can only generate 0V or 5V. Even with PWM pins and analogRead() function, it can only generate positive voltages. Bipolar control signal can be achieved by adding an op-amp to shift the range of the signal. However, H-Bridge does not have this problem.
2. H-Bridge can be easily driven by PWM pins while a LPF (low pass filter) circuit is probably required to convert the square wave to a DC voltage.

In conclusion, H-Bridge uses less components than push-pull amplifier and it is more easily driven by a micro-controller (Arduino).

5. Control Signal

5.1 Methodology: Bang-Bang Control

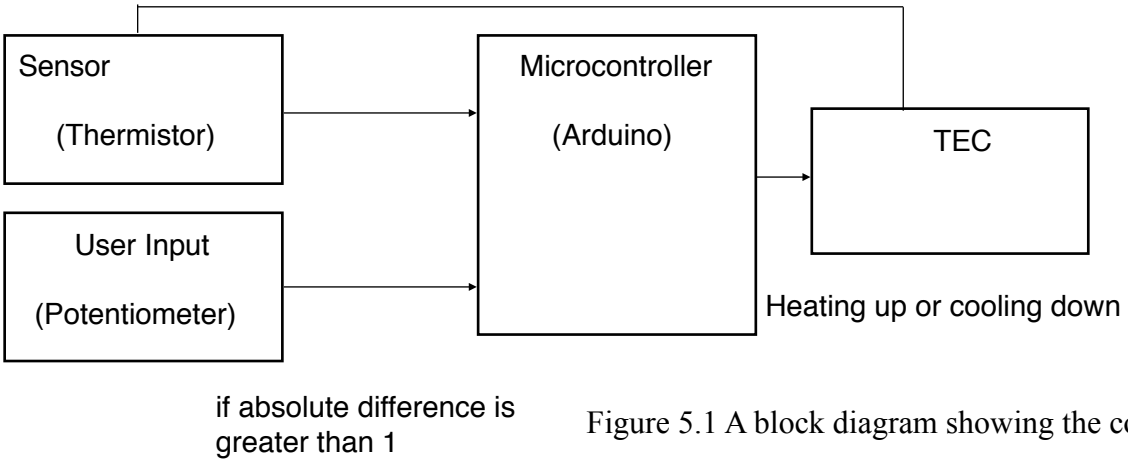


Figure 5.1 A block diagram showing the control system

Bang Bang control was used in our system. The principle behind this control is really simple. when the difference between the set value and the real measured temperature is greater than zero, one of the output pins will be turned on depending on which mode (heating or cooling) the controller is in. Since we have two different calibration curves for cooling TEC and heating TEC, a push button was introduced in the circuit for the user to set the mode of the controller. When a temperature higher than the real temperature was set, the “HEAT” mode should be selected as well so that controller can follow the heating curve to heat up TEC.

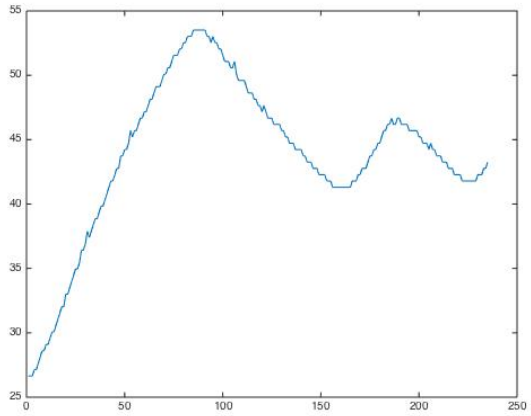


Figure 5.3 time vs temperature heating from 26 degree to 45 degree

5.2 Result

Setting the target temperature to be 45 degree at room temperature (26 degree), the result is shown in the graph below.

This graph is based on the temperature measure every half second. Therefore the time step of the x-axis is half second. According to the graph, it took around 25 seconds (50 * 0.5 second) to heat the TEC from 26 degree to 50 degree. However, it overshoots to 53 degree afterwards. Because bang bang control always drives the circuit with full power, it is easy for the system to overshoot the target temperature. It took more than two minutes (250 * 0.5 second = 125 seconds) for the system to settle down to the target temperature.

(The code for the control system is attached in Appendix C.)

5.3 Discussion

Limitations of Bang Bang control:

As Bang Bang control always drive the circuit with full power, it consumes much power than using PWM with PID control. Also, this issue brought us another problem that the system always overshoots the target and takes longer time to settle down to the target temperature than expected. These problem can be solved by using PID control. Since we don't have enough PWM pins on Arduino UNO (LCD display and frequency reading functions use all these PWM pins), we are not able to apply PID control on our system. In Appendix C, there is a piece of code (code 2) which is suppose to do PID control. By using millis() function to record the time elapsed and calculating the integration of errors, as well as the derivative of the error, the output's value will be dependent on the amount of the error, the integration of errors and the derivative of the errors. Also, a balanced weighting of three parameters kp,ki,kd is required. By doing this, the circuit will not always driven by full power so that energy can be saved. Also, it can solve the problem of overshooting the target.

6. Conclusion

The summary of this project includes the following four aspects: power consumption, accuracy, speed, reliability and the cost.

Regarding to power, the two components consuming the majority of the power are wifi shield and H-bridge. An extra 12V DC power is required to be plugged in to the 2.5 mm DC socket to drive the

WiFi shield. According to the official instruction of the Iduino Yun Shield ("Iduino Yun Shield - Geeetech Wiki", 2016), it requires 200mA current when in full load, so it is powered by the Arduino VIN pins to avoid overheated in the Arduino onboard 5V LD0. Therefore the estimated power consumption of the wifi board is P = I * V = 12V * 0.2A = 2.4W. The power consumption of the H-Bridge has been discussed in previous section, which is around 6.5W. The oscillator NE555 did not require much power; according to its specifications, its power dissipation is 1180mW ("NE555 Timer", 2016). According to the datasheet of the LCD display module, its maximum supply current is 2.5mA ("Specifications of LCD Module", 2016); therefore, the power it consumes is around 12.5mW(5V * 2.5mA). For improvements, power can be saved by using a more sophisticated control than the naive Bang Bang control.

About the accuracy of the system, by the time we did the presentation, it was still not optimal. It ended up having an offset of 4-5 degrees. However, in the future, this can be greatly improved by doing more calibration and curve fitting. Since the frequency read by NE555 was very stable, a smooth curve is expected. Also, the range of the reading from our signal conditioning circuit is large so that the sampling of our mapping from frequency to temperature is quite large as well.

The speed of our system heavily depends on H-bridge and control system. The H-bridge has provided a relatively high current to drive TEC. As what has been discussed in section 4, it only took a short time to hit the target temperature, but the total time for the system stabling down was a little bit longer. The limitation can also be improved by using PID control in the control system rather than the current approach. Also, smaller resistors could be cascaded to npn transistors in H-bridge so that less current would flow through TEC to slow down the reaction and refine the control.

The cost of the whole project as a product will be around $90 to $105, depending on which kind of Arduino board being used (more detailed in the specification in Appendix B).

Finally, this project has successfully realised basic functions of a temperature controller: the system is able to maintain at the temperature set by its user. Some basic knowledge of electronics (transistors, oscillators), programming, and techniques of soldering have been applied in this project. In future, there are many aspects of this project can be improved on: the system can be integrated to a network of IoT (Internet of Things), which is a increasingly growing topic of research nowadays.

Appendix A Reference

Arduino - ArduinoBoardUno. (2016). Arduino.cc. Retrieved 29 October 2016, from https://www.arduino.cc/en/Main/ArduinoBoardUno

Arduino - HelloWorld. (2016). Arduino.cc. Retrieved 29 October 2016, from https://www.arduino.cc/en/Tutorial/HelloWorld

Arduino - HttpClient. (2016). Arduino.cc. Retrieved 29 October 2016, from https://www.arduino.cc/en/Tutorial/HttpClient

Arduino Frequency Counter Library. (2016). Interface.khm.de. Retrieved 29 October 2016, from http://interface.khm.de/index.php/lab/interfaces-advanced/arduino-frequency-counter-library/

aws/aws-iot-device-sdk-arduino-yun. (2016). GitHub. Retrieved 29 October 2016, from https://github.com/aws/aws-iot-device-sdk-arduino-yun

BC548B. (2016). Retrieved 30 October 2016, from http://www.philohome.com/sensors/gp2d12/gp2d12-datasheets/bc548.pdf

Iduino Yun Shield - Geeetech Wiki. (2016). Geeetech.com. Retrieved 30 October 2016, from http://www.geeetech.com/wiki/index.php/Iduino_Yun_Shield

NE555 Timer. (2016). Texas Instruments. Retrieved 29 October 2016, from http://www.ti.com/lit/ds/symlink/ne555.pdf

Raj, A. (2016). Retrieved from https://qph.ec.quoracdn.net/main-qimg-3b294f2e74073525723aacf5e7587b2b?convert_to_webp=true

Specifications of LCD Module. (2016). Retrieved 30 October 2016, from https://www.sparkfun.com/datasheets/LCD/GDM1602K-Extended.pdf

Temperature Controller Basics Handbook | Instrumart. (2016). Instrumart.com. Retrieved 30 October 2016, from https://www.instrumart.com/pages/283/temperature-controller-basics-handbook

(2016). Retrieved from https://s-media-cache-ak0.pinimg.com/originals/e9/77/52/e97752cc08d4d6c0b70c769860aa8685.jpg

Appendix B Specification of the Temperature Controller

Range of operation: 5 celsius degree ~ 45 celsius degree

Power consumption: <= 9W

Accuracy: 0.01 celsius degree

Speed: from room temperature to maximum degree: 40 seconds

from maximum degree to minimum degree : 2 minutes

Maximum Current flowing through in the system: 1.5 A

DC power supply to the system: 5 V with 2 A current limit.

Cost: Thermoelectric Cooler $20

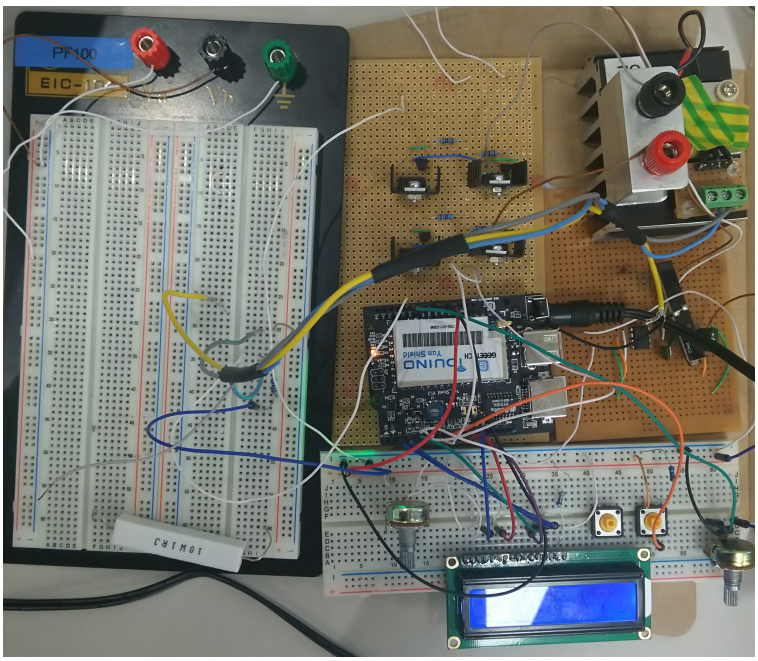
Arduino Uno $15 ~ $30

Iduino WiFi Shield $30

H-Bridge + Signal Conditioning Circuit $6

LCD module $4

(These information is based on the price found on the internet.)



Appendix C Code

Code 1: the program for the whole system

```
#include <FreqCounter.h>
#include <LiquidCrystal.h>
#include <Bridge.h>
#include <HttpClient.h>
#include <Console.h>
#include <math.h>
#define HEAT 1
#define COOL 0
LiquidCrystal lcd(12, 11, 9, 4, 3, 2);
long double temperature = 0;
const int buttonPin = 8;     // the number of the pushbutton pin
int mode = HEAT;
int buttonState = 0; // variable for reading the pushbutton status
int lastButtonState = buttonState;
const long double p1 =  -1.243e-12;//  (-1.985e-12, -5.015e-13)
const long double p2 =   2.528e-08;//  (1.067e-08, 3.989e-08)
const long double p3 =  -0.0001957;//  (-0.0003026, -8.88e-05)
const long double p4 =      0.6987;//  (0.3545, 1.043)
const long double p5 =      -944.8 ;// (-1356, -533.4)
const long double q1 =   7.912e-12;//  (6.001e-12, 9.824e-12)
const long double q2 =  -1.456e-07 ;// (-1.826e-07, -1.087e-07)
const long double q3 =    0.001004 ;// (0.0007378, 0.001271)
const long double q4 =      -3.062 ;// (-3.911, -2.213)
const long double q5 =        3481 ;// (2471, 4490)
/* AD590: FOR CALIBRATION */
int AD590_PIN = A2;
int AD590RAW = 0;
int mappedValue = 0;

int analogIn = A0;
int analogVal = 0;

void LCD_SETUP(){
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("REAL_TIME  USER");
}

void LCD_DISPLAY(int temp_real_time, double USER_input, int mode){
    // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  String USER_input_string = String(USER_input);
  String temp_real_time_string = String(temp_real_time);
  String mode_string = String(mode);
  String total = temp_real_time_string + "    " + USER_input_string +" "+ mode_string;
  lcd.print(total);
}
void setup() {
  // Bridge takes about two seconds to start up
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  LCD_SETUP();
  Bridge.begin();
  Console.begin();
    // Wait for Console port to connect
 while (!Console);
  Console.println("CONSOLE ON");//Data flow: Arduino --> Yun Shield --> Arduino IDE
}

void loop() {
  buttonState = digitalRead(buttonPin);
  // compare the buttonState to its previous state
  if (buttonState != lastButtonState) {
    // if the state has changed, increment the counter
    if (buttonState == HIGH) {
      if (mode == HIGH)
          mode = LOW;
      else
          mode = HIGH;
```

```
    }
  }
  /* FOR CALIBRATION
   * AD590RAW = analogRead(AD590_PIN);
   * float voltageAD = (AD590RAW*1.0)/1024 * 4.0;
   * float AD590_temperature = (voltageAD) * 100.0 - 273.15;
   * Console.print(AD590_temperature);
   */
  lastButtonState = buttonState;
  /* COUNT THE FREQUENCY */
  FreqCounter::f_comp = 8;
  FreqCounter::start(1000);
  while(FreqCounter::f_ready == 0);
  unsigned long frequency = FreqCounter::f_freq;
  /* READ THE USER INPUT */
  analogVal = analogRead(analogIn);
  int set = map(analogVal,0,1024,0,60);
  set = constrain(set, 5, 45);
   LCD_DISPLAY((int)temperature,set,mode);
  /* INITIALIZE HTTPCILENT */
  HttpClient client;
  // Make a HTTP request:To send analog input values of A0 and A1
  //interfacing with dweet
  //access by https://dweet.io/follow/YUN_ANALOG_IN_DWEETING_2
  client.get("http://www.dweet.io/dweet/for/YUN_ANALOG_IN_DWEETING_2?UserInput="+String(set)
  +"&Frequency="+String(frequency)+"&temperature="+String((double)temperature)+"&mode="+String(mode));
  /* BANG BANG CONTROL */
  if(mode == HEAT){
    temperature =  p1 * pow(frequency,4) + p2 * pow(frequency,3) + p3 * pow(frequency,2) + p4 *frequency + p5;
    if(abs(temperature - set) >= 1){
      digitalWrite(6,LOW);
      digitalWrite(7,HIGH);
    }
    else {
      digitalWrite(6,LOW);
      digitalWrite(7,LOW);
    }

  } else if( mode == COOL){
    temperature = q1 * pow(frequency,4) + q2 * pow(frequency,3) + q3 * pow(frequency,2) + q4 *frequency + q5;
    if(abs(temperature - set) >= 1){
      digitalWrite(6,HIGH);
      digitalWrite(7,LOW);
    }
    else {
      digitalWrite(6,LOW);
      digitalWrite(7,LOW);
    }

  }

}
```

Code 2: A segment of code using PID Control

```
unsigned long now = millis();
double timeChange = (double)(now - lastTime);
if(mode == HEAT){
  temperature =  p1 * pow(frequency,4) + p2 * pow(frequency,3) + p3 * pow(frequency,2) + p4 *frequency + p5;
  double error = set - temperature;
  errSum+=(error * timeChange);
  double dErr = (error - lastErr)/timeChange;
  Output = kp * error + ki*errSum + kd *dErr;
  lastErr = error;
  lastTime = now;
  if(Output > 255){
    Output = 255;
  }
  if(Output < 0){
    Output = 0;
  }
  analogWrite(6,Output);
  digitalWrite(7,LOW);
} else if( mode == COOL){
  temperature = q1 * pow(frequency,4) + q2 * pow(frequency,3) + q3 * pow(frequency,2) + q4 *frequency + q5;
  double error = temperature - set;
  errSum+=(error * timeChange);
  double dErr = (error - lastErr)/timeChange;
  Output = kp * error + ki*errSum + kd *dErr;
  lastErr = error;
  lastTime = now;
  if(Output > 255){
    Output = 255;
  }
  if(Output < 0){
    Output = 0;
  }
  analogWrite(7,Output);
  digitalWrite(6,LOW);
}
```