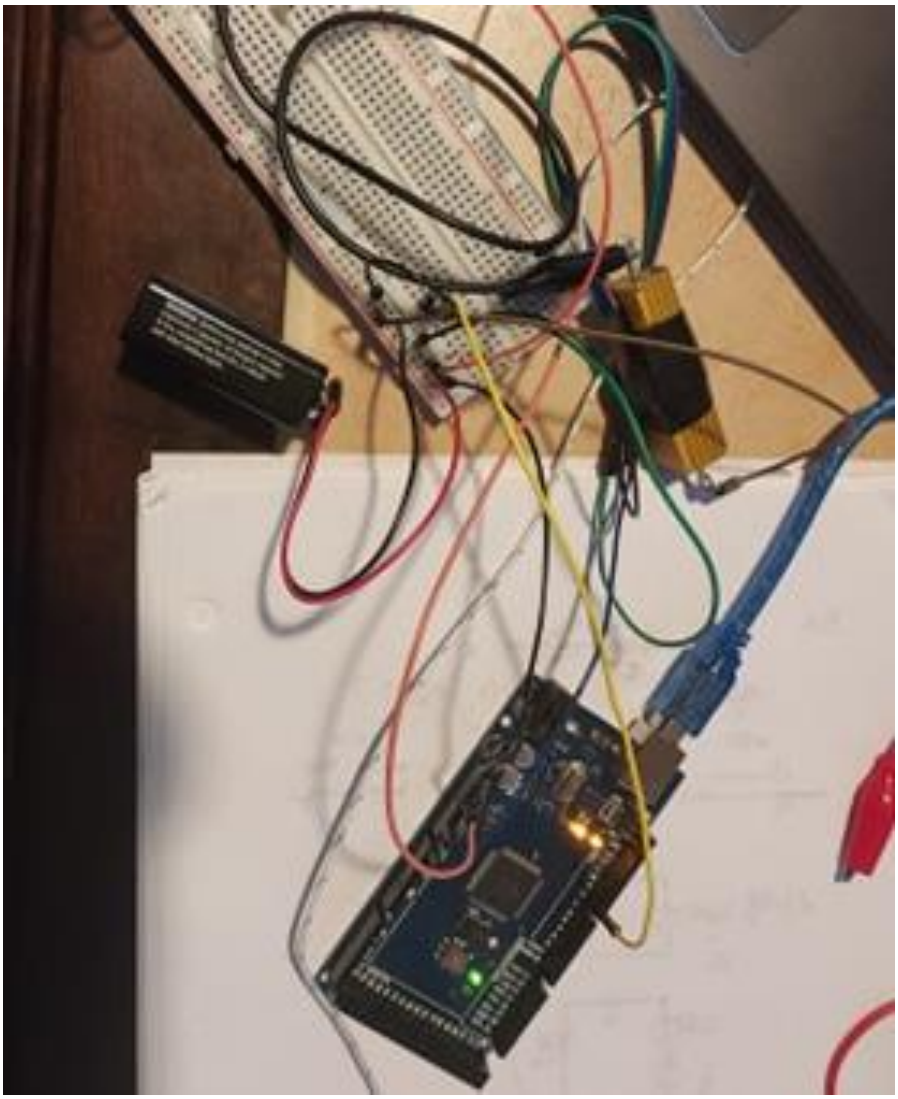


Temperature Control

PID, PI AND P CONTROLL

Shahzoda Staroverov | TCES 455 | 12.11.2020

Table of Contents

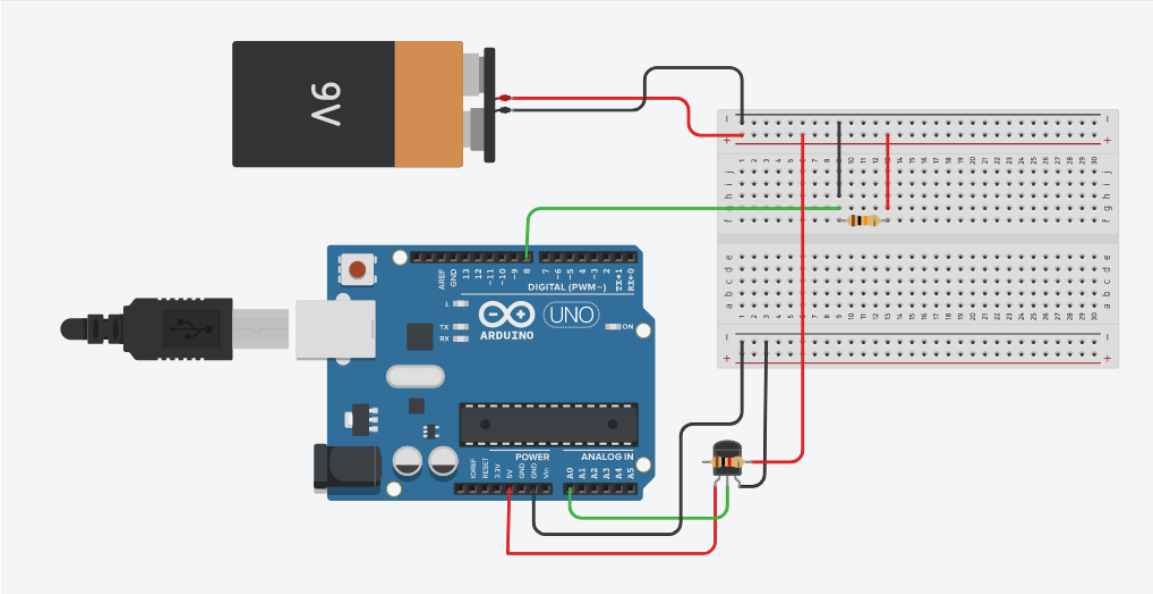
Project Objectives

The goal of this project is to create temperature controller with PID/PI/P design. To achieve this goal, I used

- Arduino MEGA 2560
- Bread Board
- Power Resistor
- Temperature sensor
- Power supply 12V or Battery 9V

NOTE: Keep in mind, 9V battery will only last you 10 minutes.

Hardware connection on bellow: (Software I used to get the following sketch didn't have Arduino Mega or power resistor to display. So, I'm using usual resistor instead. Make sure temperature sensor is pressed against the power resistor. You can use tape for that which will also helps to keep the heat in to get better results from sensor.):



Software used in this project:

- Arduino IDE – to debug and upload the code into the board
- MatLab – to plot the results from txt file.
- CoolTerm – transfer data from serial port to txt file

I created a model that will keep the power resistor at the setpoint(desired) temperature. To do that we need to supply a certain voltage to the power resistor to increase the temperature bellow setpoint and decrease the voltage or turn it off when the temperature goes above the setpoint.

The design a PID controller, we need to know the following:

- Setpoint – temperature we want to set the power resistor to
- Feedback – we will use temperature sensor to get the temperature from power resistor.
- Error – when system makes an error, the difference between the actual temperature and the desired temperature.

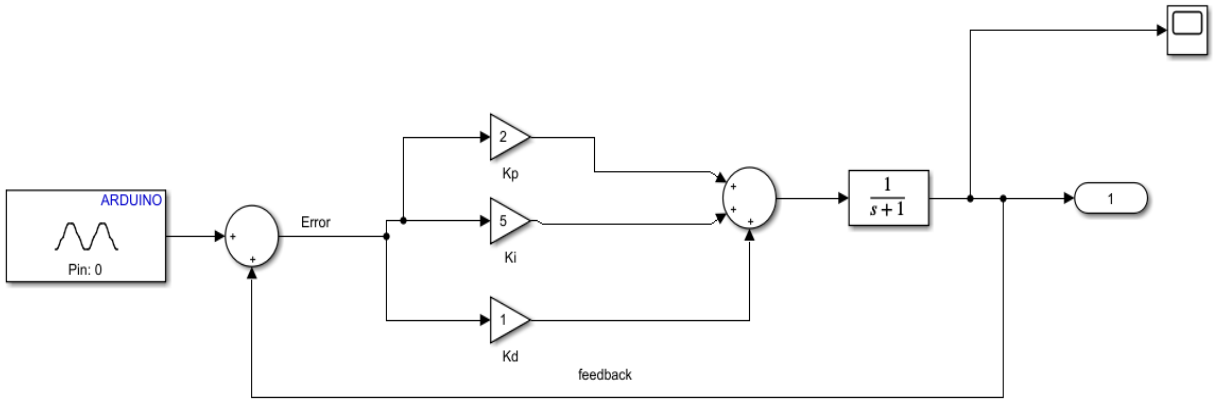


Fig 1.1 PID controller

PI Controller Sketch

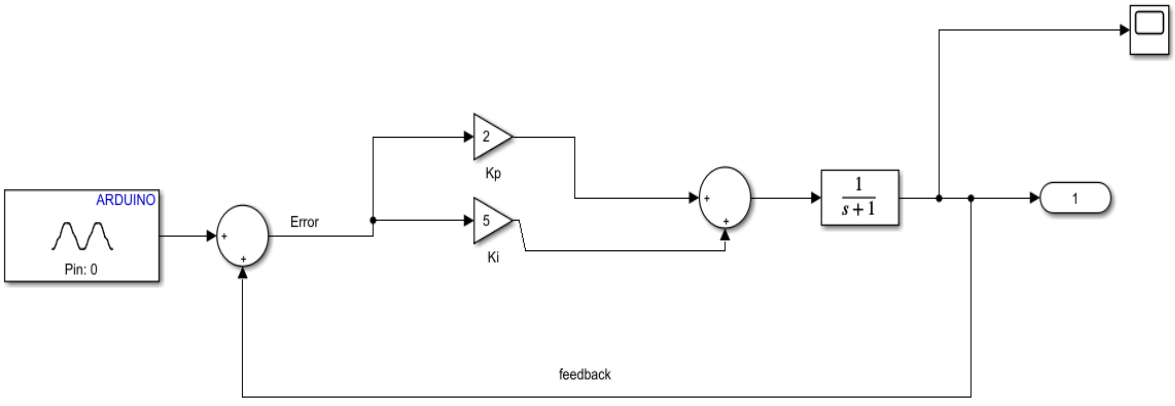


Fig 1.2 PI controller

P Comptroller

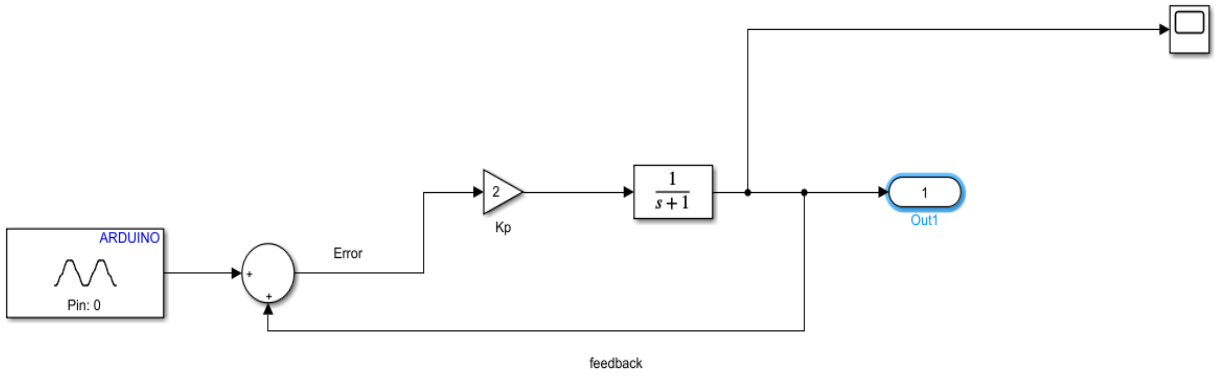


Fig 1.3 P controller

ARDUINO CODE: PID CONTROLLER

For this Arduino code I used the built-in library PID_v1.h which automatically tunes the PID controller as long as you know the following values:

PID::PID(double* Input, double* Output, double* Setpoint, double Kp, double Ki, double Kd, int POn, int ControllerDirection)

INPUT - temperature that you are getting from temperature sensor which is connected to pin Analog Ao (you can choose any other analog pins)

OUTPUT – voltage you are getting from PWM pin 8 (but you can use any other PWN pins)

Setpoint – is the desired temperature

Kp – proportional constant that is based entirely on error value

Ki – integral constant that is based on the rate of change in error

Kd – derivative constant that is based on the changed error over time.

POn – n/a, didn't set this value

ControllerDirection – there are DIRECT and REVERSE option.

- DIRECT – is where + output leads to + input
- REVERSE – is where + output leads to – input

Code is located in the PID_temp folder.

1. Initialize your values

```
#include <PID_v1.h>

#define PIN_INPUT 0 //pin A0 analog input
#define PIN_OUTPUT 8 // pin 8 PWM ouput

//Define Variables we'll be connecting to
double Setpoint, Input, Output, temperature;

//Specify the links and initial tuning parameters
double Kp=2, Ki=5, Kd=1;
PID temp_control(&temperature, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
```

2. Setup() is where you initialize your values. Keep in mind the value you read from the analog is not the value you feed to PID temp_control. You need to first convert that reading to actual temperature. You can initialize your setpoint here as well.

```
void setup()
{
 Serial.begin(9600); //to print values to test and graph
 Input = analogRead(PIN_INPUT);// read the input value
 Setpoint = 40; //in c
 //turn the PID on
 temp_control.SetMode(AUTOMATIC);
}
```

3. Here we are converting the analog volt to actual temperature with the following equations:
   Voltage = sample * reference voltage /1024
   - Sample is the analog value
   - Reference voltage is the voltage that's connected to temperature sensor (+5 V in my case)

   Temperature in C = (voltage – 500mV)*100 where 500mV is an offset

   Once we use this equation to get the actual temperature, it will be used as an input for the PID function.

   Print out your temperature to use it when we start testing.

```
void loop()
{
 Input = analogRead(PIN_INPUT);
 float voltage = Input*5/1024;
 //temperature in C
 temperature=(voltage-0.5)*100;
 temp_control.Compute();
 analogWrite(PIN_OUTPUT, Output);
 Serial.print("\n Temperature: ");
 Serial.print(temperature);
}
```

ARDUINO CODE: PI CONTROLLER

There is no PI controller library that I was able to find to write a code for PI controller. Therefore, we have to come up with our own code as well as doing PI controller tuning by initially setting Kp as 1 and Ki as 0.

INPUT - temperature that you are getting from temperature sensor which is connected to pin Analog A0 (you can choose any other analog pins)

OUTPUT – voltage you are getting from PWM pin 8 (but you can use any other PWN pins)

Setpoint – is the desired temperature. I will set it as 40C. If set higher you will need 12 V power supply instead because 9V won't last as long.

Kp – proportional constant that is based entirely on error value

Ki – integral constant that is based on the rate of change in error

Code is located in the PI_temp folder.

Since we are not able to use PI library, we need to track our own time and error.

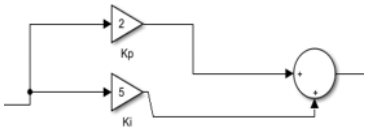
1. Initialize values: we will need to know time elapsed to figure out change in error. To do that we need current time and previous time.

```
#define PIN_INPUT 0
#define PIN_OUTPUT 8

double Kp = 2;
double Ki = 5;

unsigned long currentTime, previusTime;
double elapsedTime;
double error;
double input, output, setPoint, temperature;
double cumError;
```

2. In setup(), initialize your setpoint and serial port connection.

```
void setup(){
  setPoint = 40;
  Serial.begin(9600);
}
```

3. computePI()
   Here we need to get elapsed time and the error which is the difference of the setpoint value and input. The output equation we used below in the code comes from this model:




```
double computePI(double inp){
  currentTime = millis();
  elapsedTime = (double)(currentTime - previusTime);

  error = setPoint - inp;
  cumError += error*elapsedTime;

  double out = Kp*error + Ki*cumError;

  previusTime = currentTime;
  return out;

}
```

4. loop()- this part is self-explanatory and very similar to explanation I gave in above for PID controller.

```
void loop(){
  input = analogRead(PIN_INPUT);
  float voltage = input*5/1024;
  //temperature in C
  temperature=(voltage-0.5)*100;
  output = computePI(temperature);
  delay(10);
  analogWrite(PIN_OUTPUT, output);

 Serial.print("\n Temperature: ");
 Serial.print(temperature);

}
```

ARDUINO CODE: P CONTROLLER

There is no P controller library as well, but we will use the code for PI controller and adjust it to P controller.

INPUT - temperature that you are getting from temperature sensor which is connected to pin Analog Ao (you can choose any other analog pins)

OUTPUT – voltage you are getting from PWM pin 8 (but you can use any other PWN pins)

Setpoint – is the desired temperature

Kp – proportional constant that is based entirely on error value

Code is located in the P_temp folder.

```
void loop(){
  input = analogRead(PIN_INPUT);
  float voltage = input*5/1024;
  //temperature in C
  temperature=(voltage-0.5)*100;
  output = computePI(temperature);
  delay(10);
  analogWrite(PIN_OUTPUT, output);

 Serial.print("\n Temperature: ");
 Serial.print(temperature);
 //Serial.print("\n Power: ");
 //Serial.print(output);

}
double computePI(double inp){
  currentTime = millis();
  elapsedTime = (double)(currentTime - previusTime);

  error = setPoint - inp;
  cumError += error*elapsedTime;

  double out = Kp*error;

  previusTime = currentTime;
  return out;
```

TESTING AND MESURMENT: PID CONTROLLER

To test the PID controller, I used Serial Plotter (fig 2.1) to view the temperature in the graph form to make sure the PID is working properly. You are able to use Serial Monitor as well to view the temperature values instead of graph.

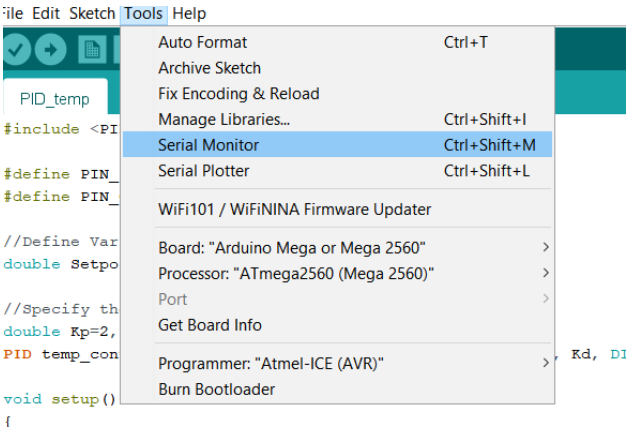


Fig 2.1 serial plotter

If PID controller is working is working, we can use MatLab to graph the data yourself. I used CoolTerm application to read the temperature values from the port and save it in txt file (fig 2.2).

NOTE: Make sure your serial monitors or serial plotter is not open or the cool term will error out when you try to connect.

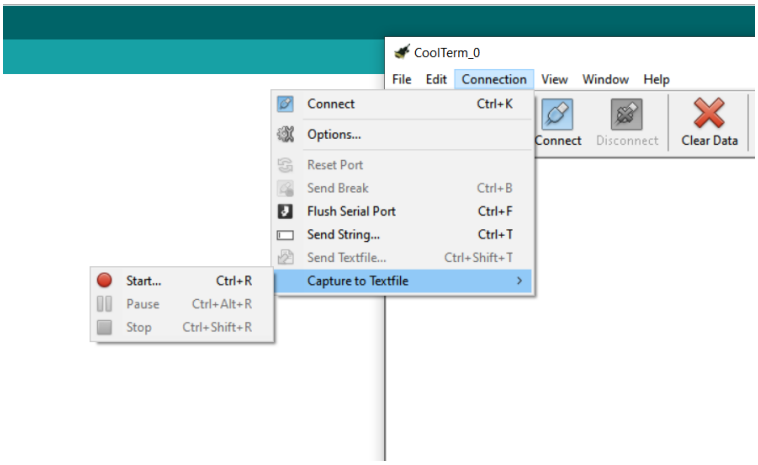


Fig 2.2 CoolTerm

Once, you get sufficient amount of the data, disconnect and graph it, using MatLab to see the result (fig 2.3).

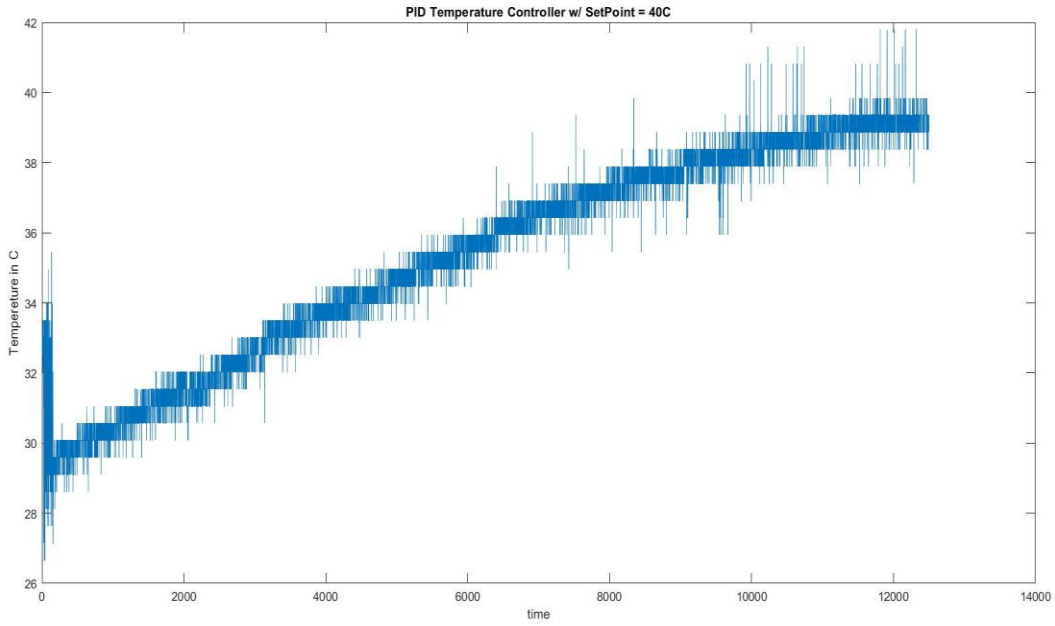


Fig 2.3 MatLab PID result

TESTING AND MESURMENT: PI CONTROLLER

For PI controller, we need to figure out values for Kp and Ki by testing Kp and Ki values as follows:

- set Kp as 1 and Ki as 0
- print error and temperature
- view these values through serial monitor (fig 2.1)
- and adjust the Kp and Ki, based on this value. (this is the tedious part of the project)

Once you get the value that works, grab the temperature values (no longer need to print error values) using CoolTerm(fig 2.2) and plot it using MatLab. The following is my result for PI controller:

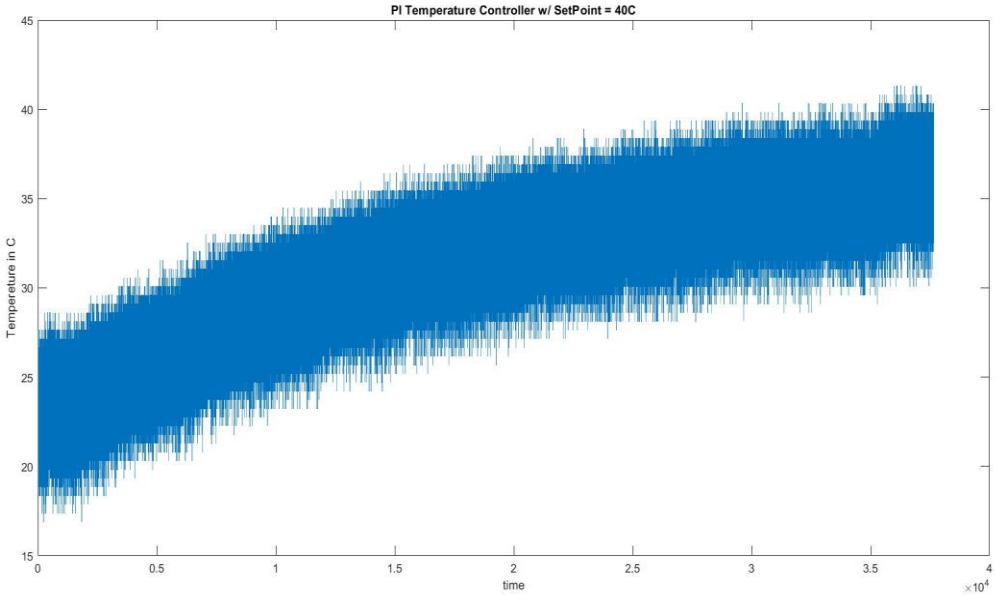


Fig 3.1 MatLab PI controller result

TESTING AND MESURMENT: P CONTROLLER

For P controller, we need to figure out values for Kp by testing Kp value as follows:

- set Kp as 1
- print error and temperature
- view these values through serial monitor (fig 2.1)
- and adjust the Kp, based on this value. (this is the tedious part of the project)

Once you get the value that works, grab the temperature values (no longer need to print error values) using CoolTerm(fig 2.2) and plot it using MatLab. The following is my result for PI controller:

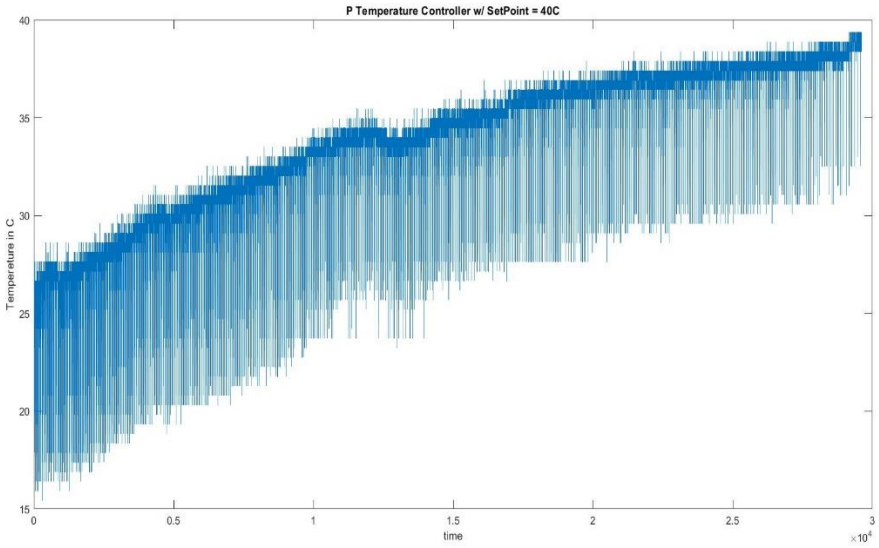


Fig 4.1 MatLab P controller result

ANALYSIS

Compare all three results together by sub-plotting the results for each controller using MaLab (fig 4.2). PID controller has smaller overshoot then PI or P controllers which helps the temperature to be more stable and accurate to the setpoint.

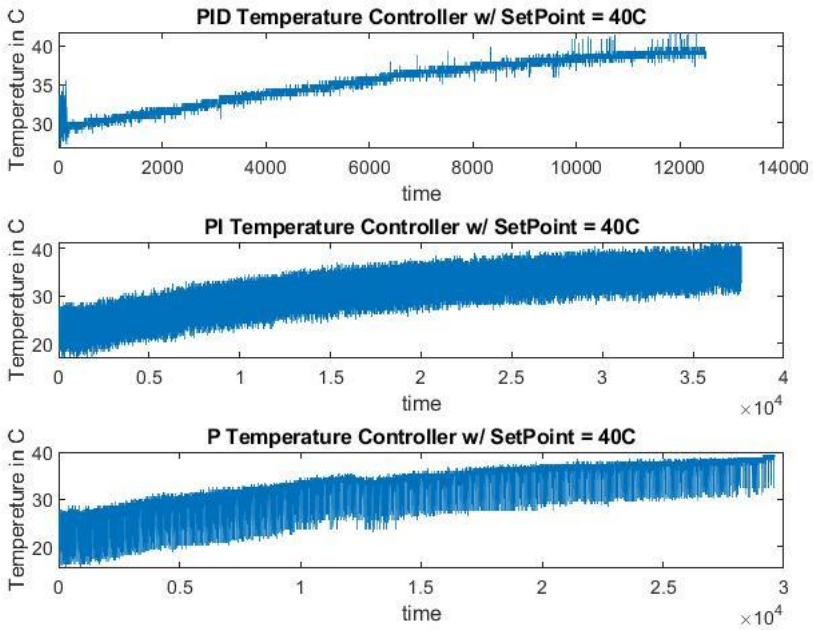


Fig 4.2 PID/PI/P controller

APPENDIX:

Arduino Code for PID controller:

```
#include <PID_v1.h>

#define PIN_INPUT 0 //pin A0 analog input
#define PIN_OUTPUT 8 // pin 8 PWM ouput

//Define Variables we'll be connecting to
double Setpoint, Input, Output, temperature;

//Specify the links and initial tuning parameters
double Kp = 2, Ki = 5, Kd = 1;
PID temp_control(&temperature, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

void setup()
{
       Serial.begin(9600); //to print values to test and graph
       Input = analogRead(PIN_INPUT);// read the input value
       Setpoint = 40; //in c
       //turn the PID on
       temp_control.SetMode(AUTOMATIC);
}

void loop()
{
       Input = analogRead(PIN_INPUT);
       float voltage = Input * 5 / 1024;
       //temperature in C
       temperature = (voltage - 0.5) * 100;
       temp_control.Compute();
       analogWrite(PIN_OUTPUT, Output);
       Serial.print("\n Temperature: ");
       Serial.print(temperature);
}
```

Arduino code for PI controller:

```
#define PIN_INPUT 0
#define PIN_OUTPUT 8

double Kp = 2;
double Ki = 5;

unsigned long currentTime, previusTime;
double elapsedTime;
double error;
double input, output, setPoint, temperature;
double cumError;

void setup() {
       setPoint = 40;
       Serial.begin(9600);
}
```

```
void loop() {
	input = analogRead(PIN_INPUT);
	float voltage = input * 5 / 1024;
	//temperature in C
	temperature = (voltage - 0.5) * 100;
	output = computePI(temperature);
	delay(10);
	analogWrite(PIN_OUTPUT, output);

	Serial.print("\n Temperature: ");
	Serial.print(temperature);

}
double computePI(double inp) {
	currentTime = millis();
	elapsedTime = (double)(currentTime - previusTime);

	error = setPoint - inp;
	cumError += error * elapsedTime;

	double out = Kp * error + Ki * cumError;

	previusTime = currentTime;
	return out;


}
```

Arduino code for P controller:

```
#define PIN_INPUT 0
#define PIN_OUTPUT 8

double Kp = 2;

unsigned long currentTime, previusTime;
double elapsedTime;
double error;
double input, output, setPoint, temperature;
double cumError;

void setup() {
	setPoint = 40;
	Serial.begin(9600);
}

void loop() {
	input = analogRead(PIN_INPUT);
	float voltage = input * 5 / 1024;
	//temperature in C
	temperature = (voltage - 0.5) * 100;
	output = computePI(temperature);
	delay(10);
	analogWrite(PIN_OUTPUT, output);

	Serial.print("\n Temperature: ");
	Serial.print(temperature);
```

```
}
double computePI(double inp) {
	currentTime = millis();
	elapsedTime = (double)(currentTime - previusTime);

	error = setPoint - inp;
	cumError += error * elapsedTime;

	double out = Kp * error;

	previusTime = currentTime;
	return out;

}
```

MatLab Code:

```
pid_txt = 'output.txt';
pi_txt = 'output_PI.txt';
p_txt = 'output_P.txt';

tempereture = read_txt(pid_txt);
tempereture_PI = read_txt(pi_txt);
tempereture_P = read_txt(p_txt);

subplot(3,1,1);
plot(tempereture)
title("PID Temperature Controller w/ SetPoint = 40C")
xlabel("time");
ylabel("Tempereture in C");

subplot(3,1,2);
plot(tempereture_PI)
title("PI Temperature Controller w/ SetPoint = 40C")
xlabel("time");
ylabel("Tempereture in C");

subplot(3,1,3);
plot(tempereture_P)
title("P Temperature Controller w/ SetPoint = 40C")
xlabel("time");
ylabel("Tempereture in C");

function tempereture = read_txt(txt_file)
% read txt file and save the data
file = fopen(txt_file,'r');
tline = fgetl(file);
tempereture = [];
while ischar(tline)
    if contains(tline,"Temperature:")
        get = strsplit(tline,' ');
```

```
        tempereture = [tempereture, str2double(get(3))];
    end
    tline = fgetl(file);
end
fclose(file);

end
```

REFERENCE:

1. PID controller library:
   https://github.com/br3ttb/Arduino-PID-Library/blob/master/PID_v1.cpp
2. Tutorial for PID controller:
   https://www.teachmemicro.com/arduino-pid-control-tutorial/