Electronics A+D

Matthieu Croci

March 22, 2024

Abstract

In this lab, some of the functionalities of the micro-controller Arduino Uno are explored within multiple small tasks which added together led to the development, both hardware and software-wise, of a temperature controller.

Contents

1 Introduction

In this experiment, some functionalities of Arduino Uno board, the Grove Starter kit and a 4-pin fan were analysed. Namely, the construction of a temperature control system. This was first regulated by a classic 2-point controller and ultimately by a PID controller (both definition are in the last two paragraphs of this section 1), but initially more basic circuits and tasks were built to help reach the final goal.

The Arduino Uno is an electronic board equipped with a micro-controller that can be connected directly via an USB cable to a computer from where its operations can be programmed. The Arduino was founded as an open-source hardware and software company where it has found many applications in education and also in the professional field. With those devices and a programming language, it is possible to create various automated controller and measurement systems. In order to simplify the connections, I also used a printed circuit board ,so called shields, which is connected directly above the Arduino and makes it easier to connect all the sensors and devices.

Programming the micro-controller is done via a dedicated IDE (integrated development environment) and using the Arduino language, which resembles the C++. One has to define the initial conditions of the Arduino pins in the **setup**() function and then assign a job that will be executed continuously in the **loop**() function. From here, all the board's connections and functionality can be controlled.

For most measurements a temperature sensor supplied in the kit was used (Grove - Temperature Sensor V1.2), additionally another thermal sensor was built starting from the basics, with the following electrical components: an operational amplifier, a voltage divider and a NTC (Negative Temperature Coefficient) thermistor. The first component mentioned takes two input voltages and amplifies a single output, in this case it has been used the op-amp as a voltage follower, i.e. a non-inverting amplifier with gain 1. In this case the output voltage is calculated using the following formula.

$$V_{out} = V_{in}(1 + \frac{R_2}{R_1}) \tag{1}$$

Where in this equation V_{out} is the voltage output of the op-amp, V_{in} is the positive input voltage supply and R_1, R_2 are the resistance connected to the negative input. While a voltage divider is a passive linear circuit which starts from an input voltage and returns a fraction of it at the output, the shape of the circuit can be seen on the left in the diagram 1. The purpose of this system is to create a reference voltage and reduce the intensity of the voltage so that it can be measured. Finally, the most important component is the NTC thermistor. This component has an internal resistance that varies depending on the temperature it is subjected to. By knowing the specifics of this variation and calibrating it, it is possible to derive a very accurate temperature measurement. In my case, the formula that links temperature and average input voltage is as follows:

$$\frac{1}{T} = \frac{\ln\left(\frac{1028}{V_{means}} - 1\right)}{B} + \frac{1}{T_0} \tag{2}$$

Where T stands for the actual temperature, T_0 is the room temperature (298.15 K), B is a constant of the sensor and V_{means} is the mean voltage that arrives to the Arduino in a given time interval. This relationship is derived from the measurement properties of the hardware used and the circuit configuration.

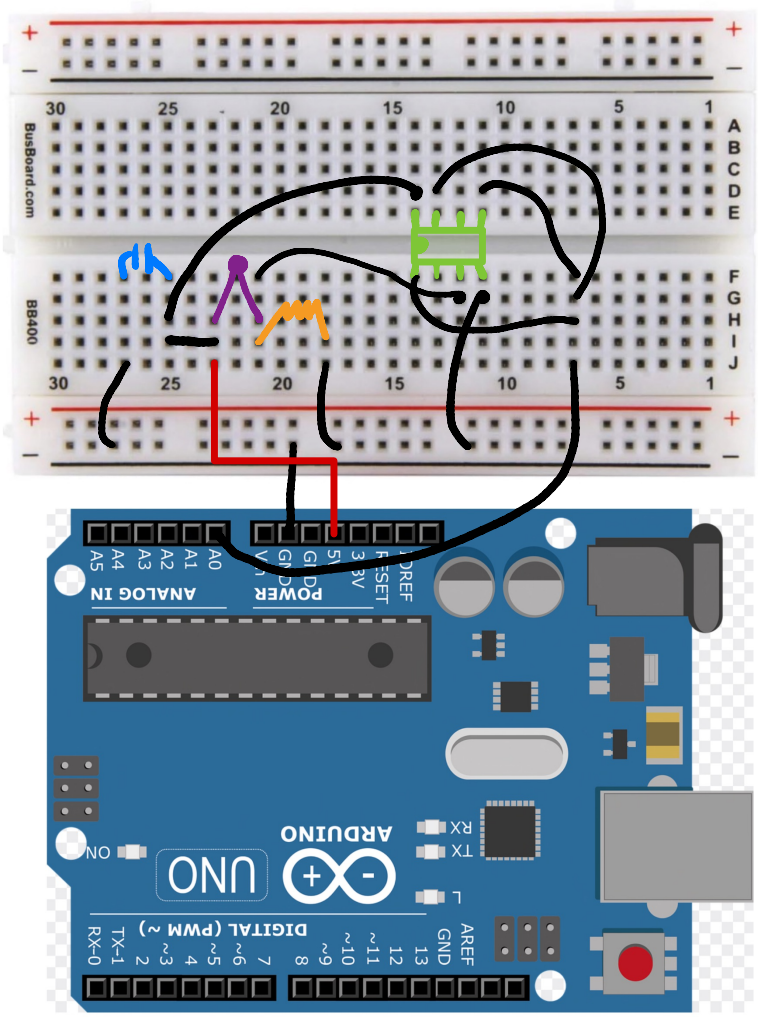

Figure 1: Circuit diagram, NTC thermistor in violet, op-amp in green, a resistance in orange and a capacitor in blue

Here follow a description of what is meant by a 2-point controller; we start by defining an

interval (2-point) in which the quantity we want to control operates, then we program what lowers the quantity so that if it exceeds the highest value (second point) it has maximum duty and only shuts down completely once the quantity has fallen below the previously selected minimum level (first point). The same mechanism inversely applies to the device that raises the quantity. This type of control has the advantage of easy programming, but it is very likely that the system oscillates around the equilibrium quantity and thus suffers from overshooting and undershooting (meaning that the actual quantity goes over or under the desired value).

From the point of view of the speed and precision with which the system approaches the desired value, PID (proportional, integral and derivative) control is significantly better. As the acronym says this algorithm is based on three principles, proportional, integral and derivative; the first takes into account the error between the desired value and the actual measured value, the second is the sum of all errors in the previous steps, while the last parameter considers the difference between the last error and the current error. By adjusting these three parameters, it is possible to have a direct and precise convergence towards the desired value without suffering from overshooting and undershooting. Thus, this controller is based on continuous monitoring the distance of the measured value to the desired one and calibrates the output accordingly. In this experiment, it was analysed the PID controller by increasing the system temperature by a few degrees and by means of a fan, controlled with this algorithm, the system is cooled down to equilibrium temperature.

2 Experiment

The main tools used in this experiment are, an Arduino one, the Grove Kit, a 4-pin fan and the classic electronic components needed to build a circuit (meaning a bread board, resistors, wires, a capacitor and some LED light). At the beginning, the main task was to install the Arduino IDE and start familiarising with the programming language. Therefore, the first circuits have the purpose to better understand the mechanism and helped to build later a temperature controller from the basics.
The upcoming text will be presented the short projects that were undertaken to achieve the ultimate objective. It is also important to know that all code can be found on Git with this link, and some of that are also printed in the appendix A. In particular, the first code 8 controls the blinking frequencies of two LED and also the electrical circuit diagram 11 is shown, then the code 9 is for monitoring the temperature sensor, a potentiometer and an LCD screen and at the end we have the 2-point controller code 10.

In the first project with Arduino, I built/programmed a classic blinking LED light and subsequently two light that blink at different frequencies. In this case I used the digital pins of the Arduino as output and alternately set HIGH and LOW, i.e give current and remove it.

Next, via the Grove shield it was connected the temperature sensor, the potentiometer and the LCD display supplied in the kit. In this step, the goal was first to transform the voltage inputs of the temperature sensor into actual temperature values with the above formula 2. Then it was printed these values on an external LCD screen and via a potentiometer set a threshold temperature where the screen would change colour (as if the threshold is the desired temperature). Specifically, the program that controls this circuit sets the background colour of the screen red if the temperature is above the threshold and blue when it is below. Both the potentiometer and the temperature sensor are connected to the Arduino via an analogue port capable of reading values from 0 V to 5 V with a resolution of 1023 spaces, which means that has a sensibility of ≈ 4.9 mV (value found by dividing the maximum input voltage with the resolution). The screen, is connected to a digital port where, after installing the library, it was possible to configure it via the Arduino language . In this case, the temperature is printed on the first line and the threshold value controlled by the potentiometer is printed on the second. This hardware configuration will be called Setup 1' in what follows.

Afterwards, it was built a temperature sensor from an NTC thermistor and an op-amp on a breadboard and then connected it to the Arduino. The circuit I am referring to is shown in Fig. 1. On the left-hand side there is a voltage divider that converts the resistance of the thermistor into a measurable voltage and on the right side an op-amp with gain 1 which amplifies the signal. These voltage are interpreted by the Arduino and the program written for this purpose. Since the thermistor is not calibrated, it will be analysed in the next section how the uncertainties are calculated and the constant values modified so that the system is correctly calibrated.

Setup 1 is extended and multiple measurements are carried out. The first consisted of producing heat with a resistor and bringing it closer to the sensor. By measuring the temperature every 0.1 second (this time interval was chosen in order to have sufficient data collection but still allow time for the sensor to adapt to the new temperature), it was possible to build and analyse the heating curve of the sensor near the resistor. All plots and considerations of these results will be presented in the next section 3.1. To achieve a relevant temperature difference, a basic circuit was built with a power supply with 7.2 V and a 100 Ω resistor, this produced via $P = \frac{V^2}{R}$ a power dissipation of approximately 0.51 W. After implementing a heat source inside the system it was also connected a fan to cool the components. The fan used has four output wires with the following properties; the first is just connected to the ground, the second to the Arduino's 5 V output, the third is the PWM (Pulse Width Modulation) controller and is connected to a digital pin, and the tachometer via a pull-up resistor (used to maintain a more constant level of voltage) is connected to an analogue output. At this point, the circuit is complete, and all that remains is to write a program to control everything.

Before getting into the experiment, it was necessary to understand how to calculate the fan rotation and how to adjust it accordingly. Starting from the fact that the tachometer returns two pulses every revolution of the fan and that the maximum rotation is 1900 RPM,

using the Nyquist-Shannon theorem, it is possible to calculate the minimum frequency at which the measurements must be taken in order to get correct value, in the sense of avoiding distortions i.e. aliasing. In this case we have to measure at least 64 time per second, that is a much lower than what **analogRead**() can do, in fact this feature goes up to 10'000 measurements per second. To ensure that the measurements are successful, it was used a measurement frequency much higher than the limit frequency calculated in accordance with Nyquist-Shannon's theorem. Again it is possible to analyse the code written for this purpose in the Git link.

Through the fan controller it is also possible to adjust the duty cycle of the fan, i.e. using the code from before and varying the duty cycle by 5.8 % (value chosen so that it has a sufficient amount of data) at a time it was measured the respective rotation speeds. Here again, a graph was built which illustrates the results.

The first program that explores this functionality is a 2-point controller, after deciding on a temperature range for example 24-25 C°, the fan is programmed to switch on when the temperature sensor indicates a value greater than 25 C° and to only switch off when the temperature is below 24 C°. In reverse, the heater turns on when the temperature is below 24 C° and turns off when it is greater than 25 C°. When it is said turns on and off, the reader must intend that the device is at its maximum 'performance' and respectively at its minimum. Even if this method of control is not the most efficient one, it still achieves a temperature that fluctuates around an equilibrium. To clearly observe what happens in the process of this system the temperature was measured, the fan rotations and when the heater turned on and the result are illustrated in the those plots Fig. 3 and Fig. 4.

Finally, as a last task, a PID controller that regulates the speed of rotation of the fan was programmed and it has been tested in the following way, the sensor is heated with the resistor and when the temperature raised by 3,4 C°, I started the program by setting the desired temperature equal to the room temperature, in this way the fan cools the sensor and the closer it gets to the set temperature the lower its speed becomes, proceeding in this way the system avoids undershooting and there is a much more precise and fast convergence to the desired temperature. Again, the plot of temperature as a function of time and the fan rotation is shown in the next sections and this result plays a central role in this lab.

3 Results and data analysis

3.1 First programs and circuits with Arduino

As explained in the previous section, the first part of the lab was dedicated to familiarise with Arduino hardware and software. In particular, one of the first codes and circuits created concerns a system that regulates the blinking frequency of two LED light. The code and the circuit diagram are illustrated in the appendix A Code 8), Fig. 11. In order to make this possible, instead of using the **delay**() function, which stops the whole code for an amount of time, it is used the **millis**() function, which returns the elapsed time from the start of the code. Using this function, it is possible to individually adjust the switch-on and switch-off frequencies of each LED.

Next, it is described the code that regulates the temperature sensor, the potentiometer and the LCD display. First the various constants and pin input ports are defined, which in this case are four, two analogue for the potentiometer and temperature sensor, and two digital for an LED and the display. Then in the loop function some tasks are executed every second, these are the calculation of the temperature, the remapping of the potentiometer value, and then an if statement that sets the screen background red and turns on the external LED if the measured temperature is higher than the threshold value. The threshold value is determined by the user via the same potentiometer. And moreover the display background is blue when the temperature is below the threshold. The code is shown in the appendix A 9.

Then a thermal sensor was constructed from a NTC thermistor and an operational amplifier, as a first measurement, i.e. with no calibrated systemm, I obtained $22.54 \pm 0.07 C^{\circ}$. Where the value is calculated with B = 4600 K and the uncertainties taken into account are for B $\pm$138 K, R_0 $\pm$5 kΩ and the resolution of the ADC $\pm$0.005 V . By adjusting the B-value and comparing the measured temperature with the reference temperature, that was 22.48 $C^{\cdot}$, it was possible to calibrate the device. The new B-value is $4492 \pm 18 K$. In this situation, we obtained a clear improvement in the uncertainty of the value of B.

In order to measure a non-constant temperature, a simple heater consisting of a resistor and a power supply has been constructed. As I said in the previous section (section 2.) the resistor used was 100 Ω and the potential difference was 7.2 V, resulting in a power dissipation of 0.51 W . This increase in temperature was measured and the results are illustrated in the plot in Fig. 2. The code used for these measurements is still the one mentioned before, in addition since the Arduino software used does not allow saving data a python code was written such that it could be integrated into this mechanism and data could be saved.

From the heating curve Fig. 2 below it can be observed that it is measured an increase of more than 12 C° in less than 10 s, moreover the increase has a rather linear trend, although there are many local small oscillations.

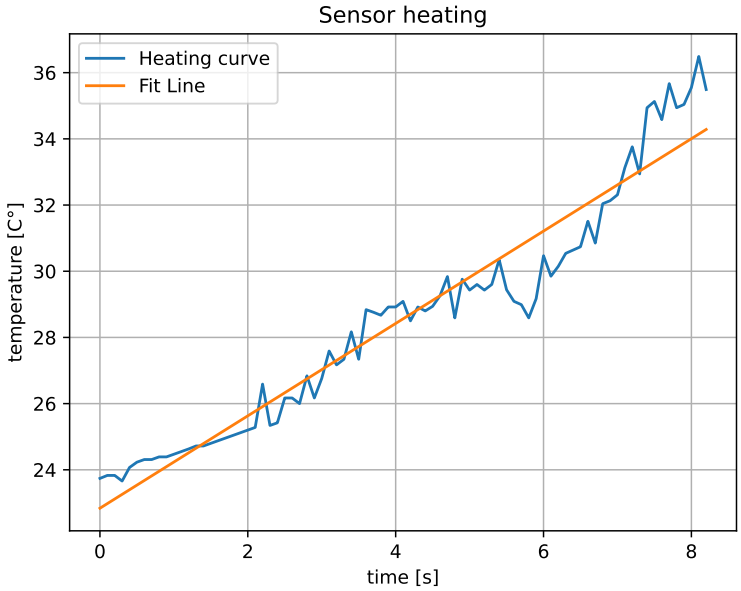

Figure 2: Heating curve of the Grove thermal sensor in blue, and in orange a linear fitting

3.2 Temperature controller

Two measurements of the 2-point controller system will be shown, the first setting the range from 23-24 $C°$and the second from 24-25 $C°$ (since both the fan and the heater work optimally i.e., act effectively on the system these intervals were chosen). In the plots one can see three measured values, the temperature, with it's scale on the left, then the fan rotations and finally the switching on and off of the heater. In Fig. 3 and 4, it can be noted the oscillatory nature of the temperature around the mean value and how this control mechanism does not guarantee stability in the system. In the second case it can also be seen that the fan cools much faster than the resistance heats the system, whereas in the first case the fan is often called upon to do its maximum work. When the fan is assigned to 0 work naturally it continues to run for a few more seconds, this cool down the system even more. In both graphs this phenomenon can be seen. To summarise, in both cases when the temperature enters the chosen range, a thermal oscillation is formed in the system which rarely finds long-term stability.

In order to better control the fan, a program has been written to read and manage the speed of the fan; the following Figure 5 shows the revolutions per minute as a function of the duty cycle undergone. It can be seen that the trend initially follows a straight line and flattens out as the maximum speed is reached. In addition, it was also measured the minimum work to start the fan rotation, i.e. the minimum amount of voltage with respect to the total 5V needed to start the fan. that is 6 %.

Finally, the code and plots for the PID controller are shown below Fig. 6 and 7. Both

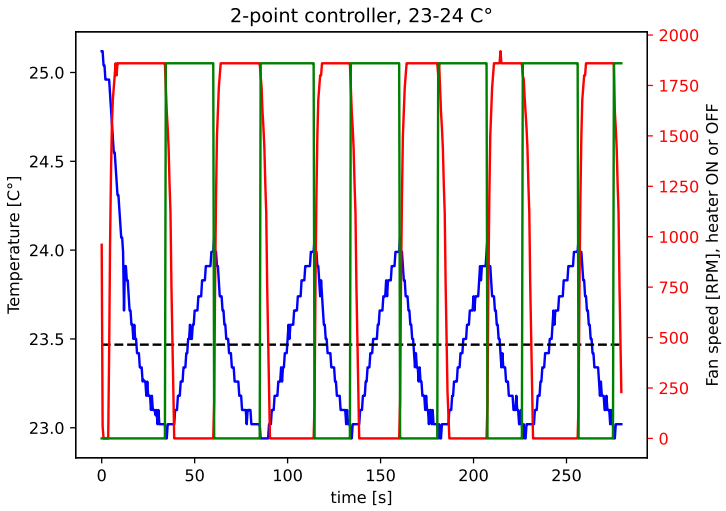

Figure 3: 2-point controller, 23-24 $C°$, the blue curve is the temperature, the red is the fan speed, the green line is the switching on of the heater and the dashed line is the mean temperature

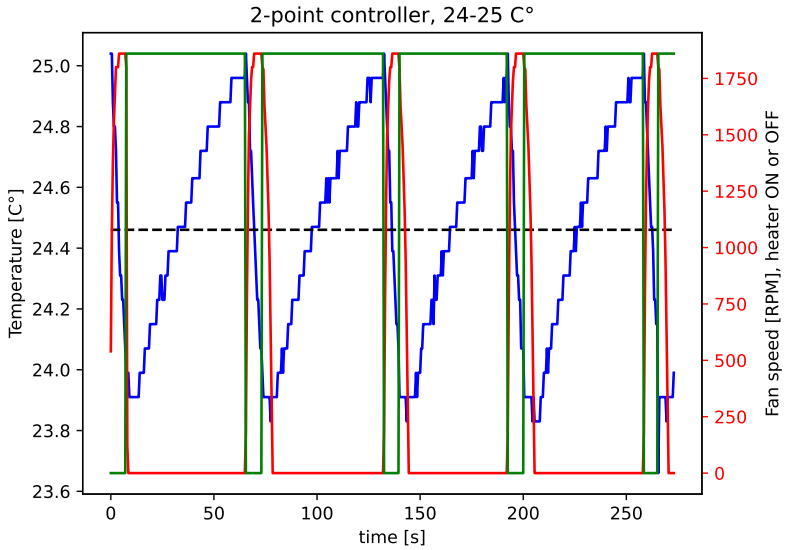

Figure 4: 2-point controller, 24-25 $C°$, the blue curve is the temperature, the red is the fan speed, the green line is the switching on of the heater and the dashed line is the mean temperature

with the fastest convergence parameters it could been found and also with some not too precise in reaching the desired temperature. In the first case it can be seen very well that the system reaches the desired point quickly and does not suffer from undershooting, while

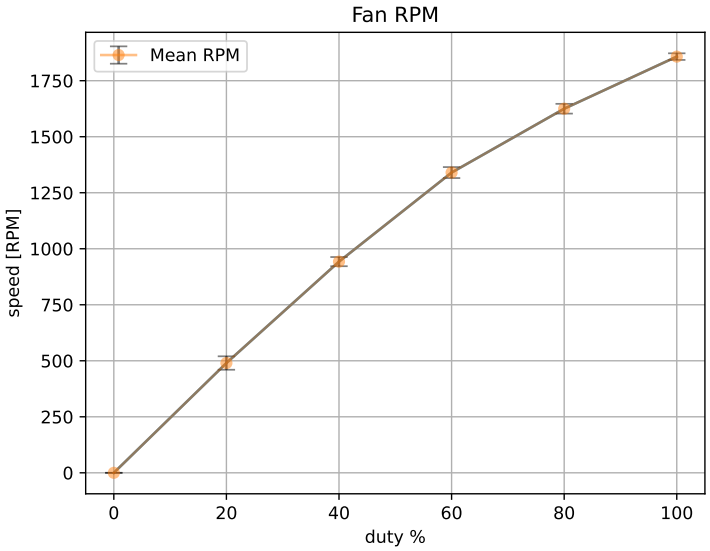

Figure 5: RPM of the fan as a function of duty cycle. Error bars are the standard deviation of the collected values.

in the second case the temperature is lowered more than it should be and oscillating up and down again it returns to the desired temperature. This method is the most efficient in performing the task set at the beginning of the experiment.

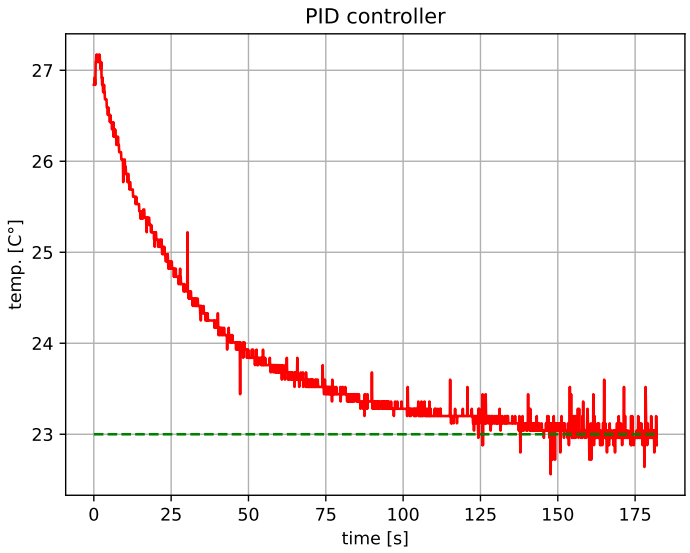

Figure 6: PID controller, best convergence where P = 9, I = 20 and D = 6, in green is the equilibrium temperature

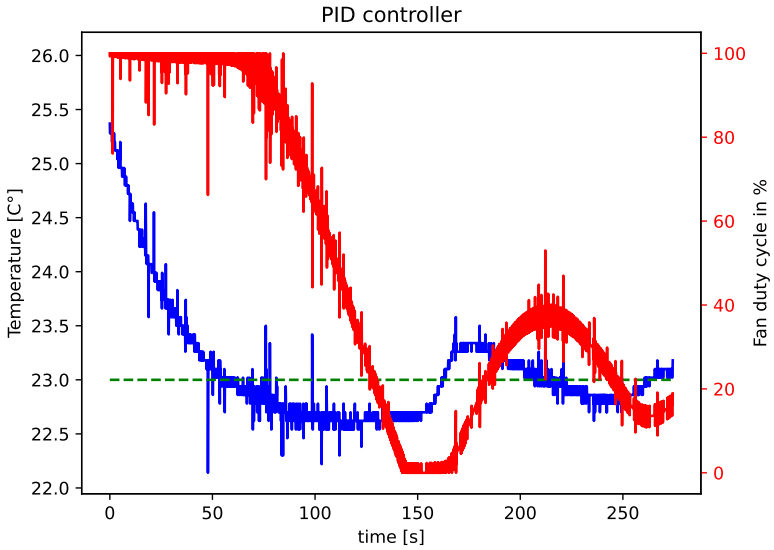

Figure 7: PID controller, the temperature (in blue) has small oscillation on the equilibrium temperature due to bad PID parameter (P = 5, I = 10 and D = 7) and in red is shown also the duty cycle of the fun

The uncertainty of the temperature measurement in Fig. 3, 4, 6 and 7 derives from the minimum resolution of the Arduino's analogue port. As mentioned earlier in the Arduino Uno, there is an uncertainty of 0.0048 V in the input due to the minimum resolution. This error is then propagated in the calculation of the temperature. For example, this result in an uncertainty of 0.05 C° for each temperature value around 23 C°. Therefore it is important for the reader to note that although there are no error bars in the plots (so that the are more readable) there is an uncertainty for each temperature point.

4 Discussion and conclusion

In general, the results obtained are in great agreement with those expected at the beginning of the lab. PID control, with the right parameters, is definitely the best choice for thermal regulation of a system even though perfect tuning is difficult to reach. Nevertheless, it is always possible to improve the results obtained and to proceed by working in a environment that is more isolated from external phenomena which may influence measurements.

A Appendix, Codes and circuit diagrams

P3/Arduino/Blink2/Blink2.ino

```
// Matthieu Croci
// 2 LED blinking with different frequency

# define led_pin1 12
# define led_pin2 8

unsigned long privTime1 = 0;
unsigned long delay1 = 1300;
byte state1 = LOW;

unsigned long privTime2 = 0;
unsigned long delay2 = 400;
byte state2 = LOW;

void setup() {
  pinMode(led_pin1, OUTPUT);
  pinMode(led_pin2, OUTPUT);

}

void loop() {
  // Blinking LED 1
  unsigned long now1 = millis();
  if(now1-privTime1>delay1){
    privTime1+=delay1;
    if(state1==LOW){
      state1 = HIGH;
    }
    else{
      state1 = LOW;
    }
    digitalWrite(led_pin1, state1);
  }
 // Blinking LED 2
  unsigned long now2 = millis();
  if(now2-privTime2>delay2){
    privTime2+=delay2;
    if(state2==LOW){
      state2 = HIGH;
    }
    else{
      state2 = LOW;
    }
    digitalWrite(led_pin2, state2);
  }
}
```

Figure 8: 2 Blinking LED Arduino code

P3/Arduino/Temp_LCD_Pot/Temp_LCD_Pot.ino

```
1  // Matthieu Croci
2
3  // Temp. Sensor printed on the LCD screen and the cut is regulated with
   a Potentiometer
4  // Also a LED is turned on when the temperature is above the cut
5
6  #include <math.h>
7  #include "rgb_lcd.h"
8  rgb_lcd lcd;
9
10 // Screen
11
12 const int colorR = 0;
13 const int colorG = 0;
14 int colorB = 100;
15
16 // Temp. Sensor
17
18 const int B = 4275;
19 const int R0 = 100000;
20 const int pinTempSensor = A0;
21
22 // Potentiometer
23
24 const int pinPot = A1;
25
26 // LED
27
28 # define LED 3
29
30 void setup(){
31
32     Serial.begin(9600);
33
34     lcd.begin(16, 2);
35     lcd.setRGB(colorR, colorG, colorB);
36
37     pinMode(LED,OUTPUT);
38     delay(1000);
39 }
40
41 void loop()
42 {
43     int a = analogRead(pinTempSensor);
44
45     float R = 1023.0/a-1.0;
46     R = R0*R;
```

Figure 9: Arduino Code that control the temperature sensore, the potentiometer and the LCD display

P3/Arduino/2_point_Heat_fan_copy_20231009090246/2_point_Heat_fan_copy_20231009090246.ino

```
1  // Matthieu Croci
2  // 2-point controller :
3
4  #include <math.h>
5  #include "rgb_lcd.h"
6  rgb_lcd lcd;
7
8  const int colorR = 0;
9  const int colorG = 0;
10 int colorB = 100;
11
12 const int B = 4275;
13 const int R0 = 100000;
14 const int pinTempSensor = A0;
15
16 const int pinPot = A1;
17
18 const int fancontr = 9;
19
20 int count=0;
21 int rpm;
22 int pwn;
23 byte heating = HIGH;
24 int indiy = 0;
25
26 #define heater 8
27
28 void setup(){
29     Serial.begin(9600);
30     lcd.begin(16, 2);
31     lcd.setRGB(colorR, colorG, colorB);
32     pinMode(fancontr,OUTPUT);
33     pinMode(heater,OUTPUT);
34 }
35
36 void loop()
37 {
38     int a = analogRead(pinTempSensor);
39
40     float R = 1023.0/a-1.0;
41     R = R0*R;
42     float temp = 1.0/(log(R/R0)/B+1/298.15)-273.15; // convert to temperature via
   datasheet
43
44     int pot = analogRead(pinPot);
45     int cut = map(pot,0,1023,13,33);
46
47     lcd.setCursor(0, 0);
48     lcd.print(temp);
49     lcd.setCursor(0, 1);
50     lcd.print(cut-1);
51     lcd.print(" , ");
52     lcd.print(cut);
```

??

```
53
54      // 2-point controller:
55
56      if(temp>=cut){
57        lcd.setRGB(255, 0, 0);
58        pwn = 255;
59        heating = LOW;
60        indiy = 0;
61        }
62      else if(temp<cut-1){
63        lcd.setRGB(0, 0, 100);
64        pwn = 0;
65        heating = HIGH;
66        indiy = 1;
67        }
68      else{
69        lcd.setRGB(0, 100, 0);
70      }
71      analogWrite(fancontr, pwn);
72      digitalWrite(heater,heating);
73
74
75      // Fan speed calculation
76      count = 0;
77      attachInterrupt(digitalPinToInterrupt(2), counter, RISING);
78      delay(1000);
79      detachInterrupt(digitalPinToInterrupt(2));
80      rpm = (count / 2) * 60;
81      Serial.print(temp);
82      Serial.print(", ");
83      Serial.print(rpm);
84      Serial.print(", ");
85      Serial.println(indiy);
86
87      delay(500);
88
89  }
90  void counter() {
91    count++;
92  }
```

Figure 10: 2-pint controller Arduino code

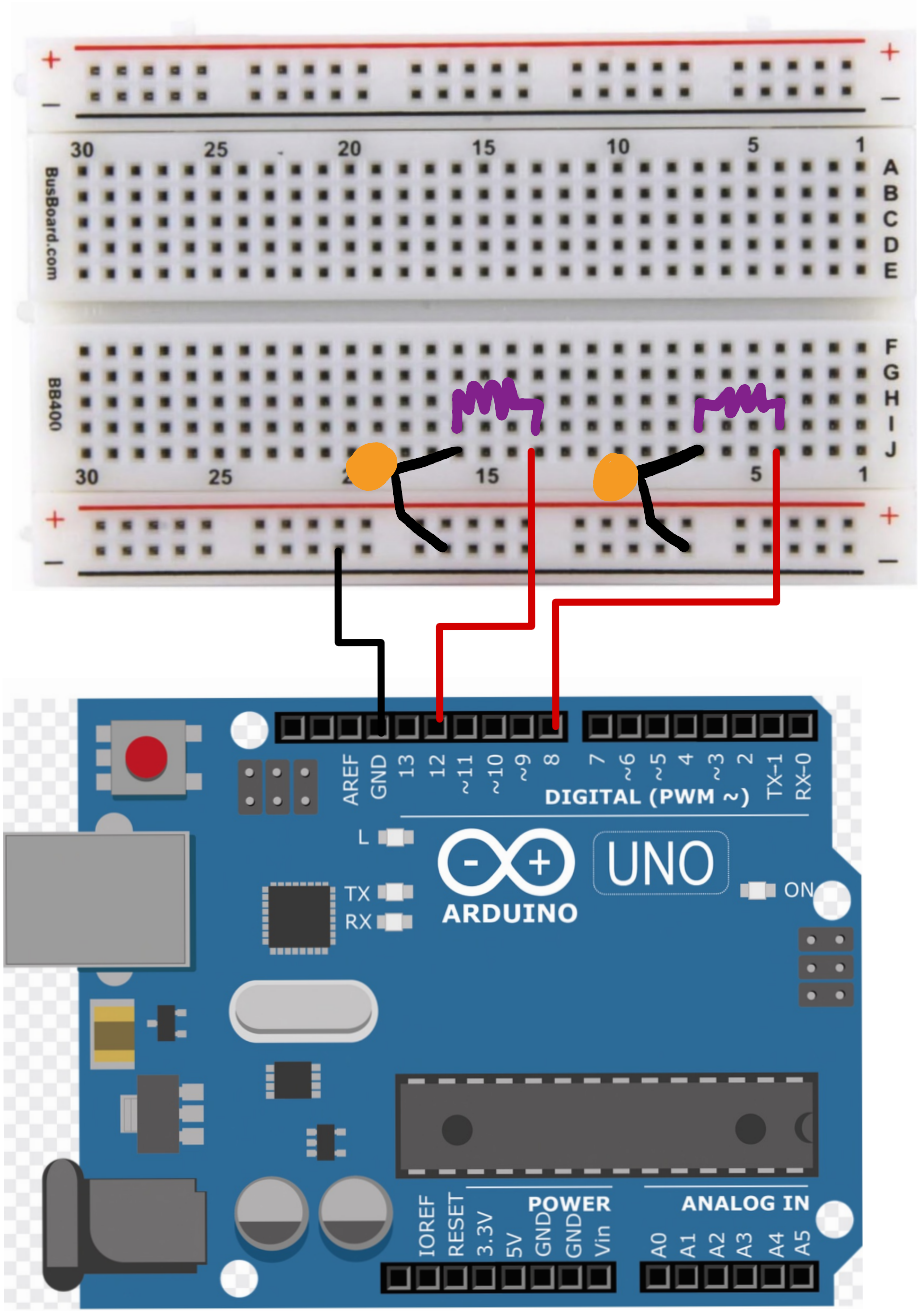

Figure 11: 2 Blinking LED circuit diagram