Thermal Bath

Introduction:

A thermal bath is an instrument used to maintain a solution at a constant temperature. It is a thermodynamic system with a heat capacity so large that the temperature of the reservoir does not change when a reasonable amount of heat is added or extracted [1].

The methodology used for this implementation consists of a PID controller which operates a submerged heater in a casing. The thermal bath implemented makes use of a water pump to move the water from the casing to the thermal bath, where the thermal system is to be placed. The PID controller is implemented through an Arduino microcontroller that gets the input through a DS18B20 submersible temperature sensor placed in the heated casing. The thermal bath itself also features a light box, which serves the purpose of illuminating the thermal system, for when photo and/or video capture is needed.

Although the implementation is controlled through the Arduino microcontroller, it can be controlled through a Raspberry PI. The code to control the thermal bath through the Raspberry PI is also provided.

Materials:

1. 1x Resistor Heater
2. 1x Max6675 Module
3. 1x MOC3020 Optocoupler
4. 1x PC817 Optocoupler
5. 1x Full Wave Rectifier
6. 1x TIP120 Transistor
7. 1x Res. 39Ohm 1/2W
8. 2x Res. 1kOhm 1/2W
9. 1x Res. 4.7kOhm 1/2W
10. 1x Res. 10kOhm 1/2W
11. 2x Res. 27kOhm 1W
12. 1x Res. 330Ohm 1/2W
13. 1x Res. 270Ohm 1/2W
14. 1x Cap. 0.01uF 250V
15. 1x Triac Q4015L5
16. 1x 1N4007 Diode
17. 1x LED Strip
18. 1x Stainless Steel Casing
19. 1x Non-submersible Water Pump 6-12V
20. 1x DS18B20 Temperature Sensor

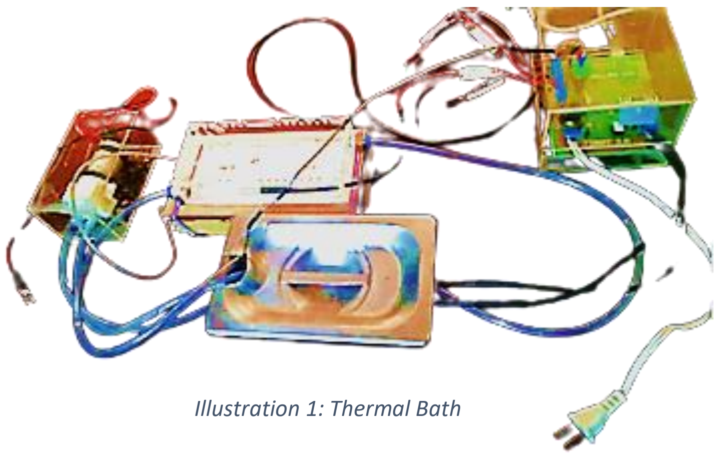


Illustration 1: Thermal Bath

Component Explanation:

MAX6675:

The MAX6675 module serves to compensate and linearize the output of type K thermocouples. The module offers a 0.25°C resolution, with an accuracy of ±1.5°C. Since it has an operation voltage of 3-5.5V, it can be easily used with Arduino microcontrollers [2]. This module is used because the thermal bed is monitored using a type K thermocouple, and the module can output the data obtained in 12-bit resolution and it is SPI compatible.

Optocouplers:

The MOC3020 optocoupler is a device which emits and receives light and serves as a light-activated-switch. It is often used in implementations which require to trigger an isolated triac through a digital signal [3]. The MOC3020 has an isolation voltage of about 5.3kV. In comparison with the MOC3020, the PC817 is a general purpose optocoupler.

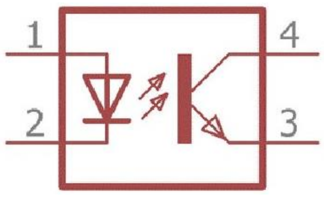


Illustration 2: General Optocoupler Diagram [4]

This component is most often used in implementations which require the isolation of currents, in this case the isolation of the DC part of the circuit with the AC part of the circuit, while still being able to have interactions between the two.

DS18B20:

It is a submersible probe which allows for the measurement of temperatures in the range of -55°C to 125°C [5]. This temperature sensor operates using the 1-Wire protocol for data transmission. This allows for the use of only 1 pin to operate various DS18B20 temperature sensors. This sensor was taken into account due to its uncertainty values. For temperatures in the range of -10°C to 85°C, the sensor measures with an accuracy of ±0.5°C, while for temperatures in the ranges of -55°C to -10°C and 85°C to 125°C, it measures with an accuracy of ±2°C. Since the maximum temperature threshold for this implementation is 60°C, this sensor provides the required characteristics.

Water Pump:

The R385 non-submersible was chosen for this implementation due to its low operation voltage as well as its large flux rate. The R385 has an inner and outer diameter of 6mm and 8.5mm respectively and can pump from 1.5 to 2.1 liters per minute. Its operation voltage can vary between 6-12V. It should be noted that the R385 water pump can withstand water up to 75°C. For this implementation, the water pump is powered by the Arduino microcontroller.

It should be noted that this water pump is non-submersible, as such it should be placed safely outside of the thermal bath and the heated reservoir. A case to hold the water pump and keep it dry should also be considered if the components are to be integrated.

Illustration 3: R385 Water Pump

Thermal Bed:
The thermal bed implemented is created using 3mm acrylic. The entry and exit points for the water flow are located diagonally on opposite walls, to make sure that the water flow covers most of the thermal bed and thus allows for the desired temperature throughout the thermal system. The other openings for the temperature sensor and the probes are located above the water pump's entry and exit points to minimize leakage. The probe openings are 1.5mm in diameter, and the water pump's entry and exit points are 1cm in diameter. Another feature of the thermal bed is an embedded case in the bottom of the thermal bed which can hold a microfluidic system. This implementation is 3mm deep and 25mm wide. The following image shows the thermal bed implementation in 3mm acrylic. It should be noted that the thermal bed is not thermally isolated

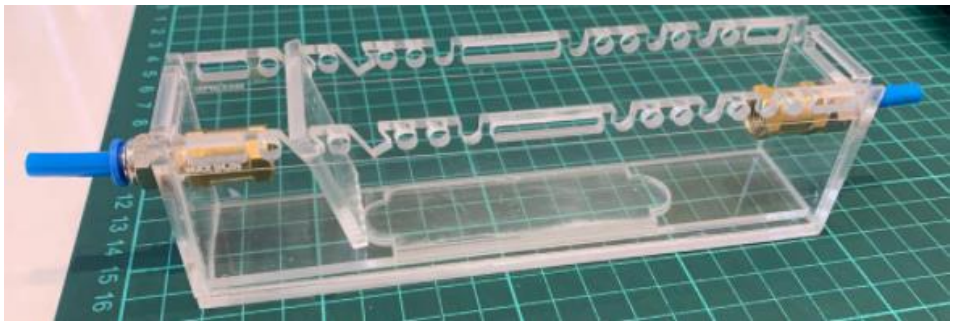


Illustration 4: Thermal Bed Implementation

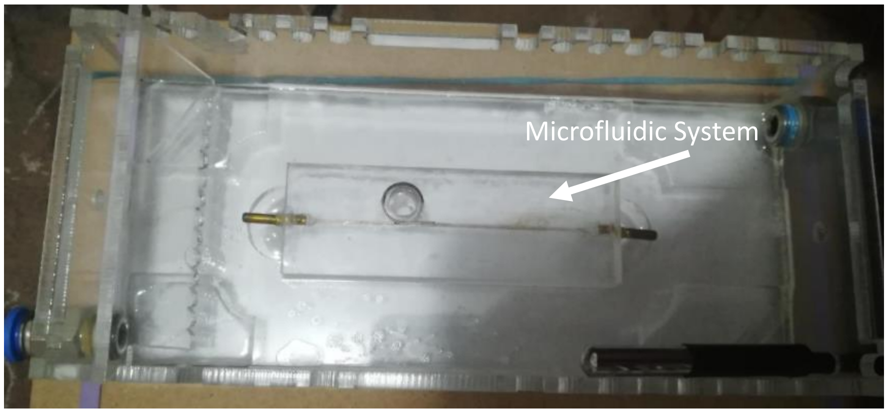


Illustration 5: Thermal Bed with Microfluidic System

Light Box:
Another important element of this implementation is the light box, which allows for the proper illumination of the thermal bed and the microfluidic system when photo and/or video capture is needed. The light box consists of an MDF casing with an LED strip glued to the inside walls. The operation of the light box is done through the same implementation which controls the thermal bed. That's to say, a command is sent from either the Arduino microcontroller or the Raspberry PI. The following illustrations show the LED strip as well as the light box MDF implementation.

The light box serves as a base for the thermal bed, which is to be placed on the outline seen in illustration 7. This properly illuminates the acrylic thermal bed.

Illustration 6: LED Strip

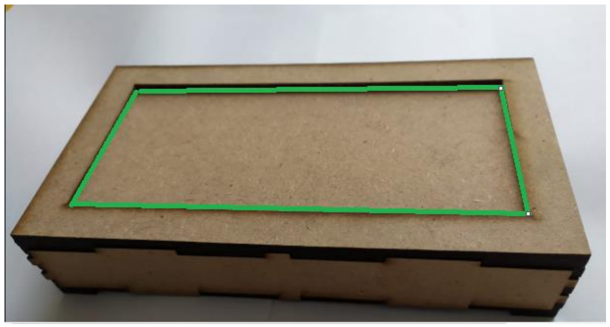


Illustration 7: Light Box Implementation

Arduino:
Arduino is a microcontroller board based on the ATmega328P. It has an unregulated power source in the VIN pin which supports 7-12V, and a regulated 5V power source. For this implantation the Arduino is the device which controls the thermal bath's temperature as well as the state of the light box. The control of this implementation is done either through the Arduino's IDE monitor or through the Raspberry PI, if the Arduino is connected to it through USB.

Raspberry PI:
The Raspberry Pi is a low-cost, small-size computer which allows the user to run code [6]. For this implementation, the Arduino controller is implemented through python code, which communicates through serial communication with the microcontroller. In this implementation, the job of the Raspberry PI is that of the Arduino's IDE monitor. The Raspberry PI uses the Serial communication library to emulate the Arduino's monitor.

Diagram:

Water Pump:
Although the water pump has a maximum flux rate of 2.1 liters per minute, it was found that this flux rate could cause disturbances within the microfluidic system. Thus, the electric current to the water pump was limited. Although some implementations using Darlington pairs were taken into account, the final implementation can be observed below [7].

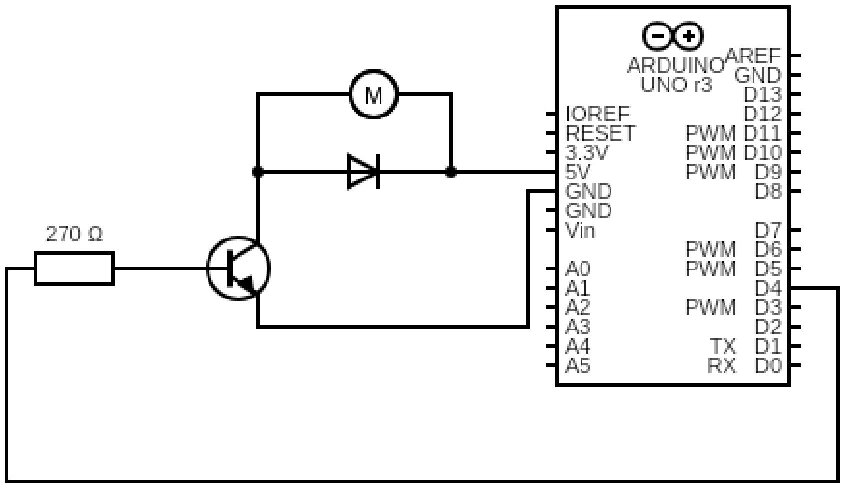


Illustration 8: Water Pump Circuit

It should be noted that to protect the electronic components connected to the water pump, a 1N4007 rectifier diode is placed in parallel with the motor. This is done, because once the water pump is turned off, a magnetic field generated by back EMF, can cause problems within the circuit. The motor is controlled through the D4 pin in the Arduino, using an NPN transistor as a switch to close the circuit.

Light Box and DS18B20:

The light box's implementation requires an input voltage of 12V. As with the water pump diagram, an NPN transistor is used as a switch to close the circuit between the LED strip and the 12V input. The LED strip is controlled by the Arduino through the D5 pin. The following illustration shows the circuit diagram for the LED strip implementation in the light box.

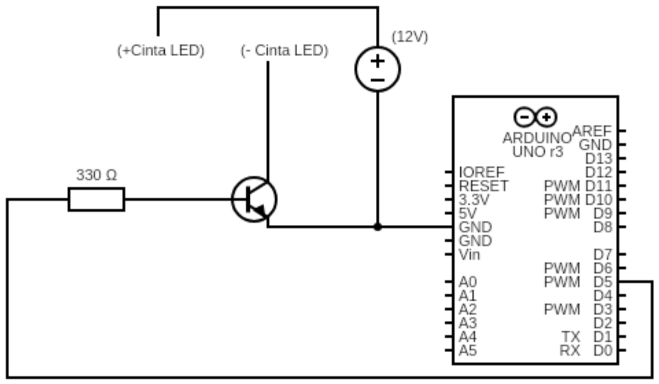


Illustration 9: LED Strip Circuit

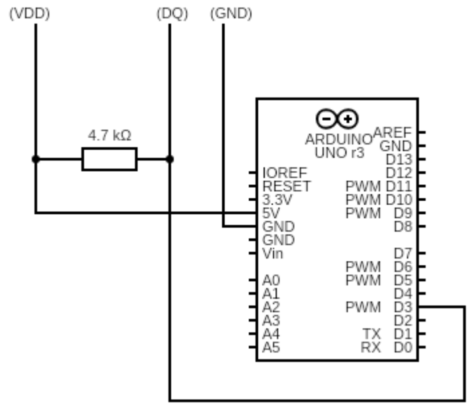


Illustration 10: DS18B20 Diagram

Illustration 10 shows the temperature sensor diagram.

Heater Diagram:

The heater circuit schematic can be observed in the following illustration. It should be noted that the circuit has both AC as well as DC voltages. Thus, it was necessary to isolate both tensions using optocouplers. The optocouplers used are the MOC3020 as well as the PC817. That's to say, all previous diagrams are not shown in this one. As such, the final implementation would have all diagrams in tandem.

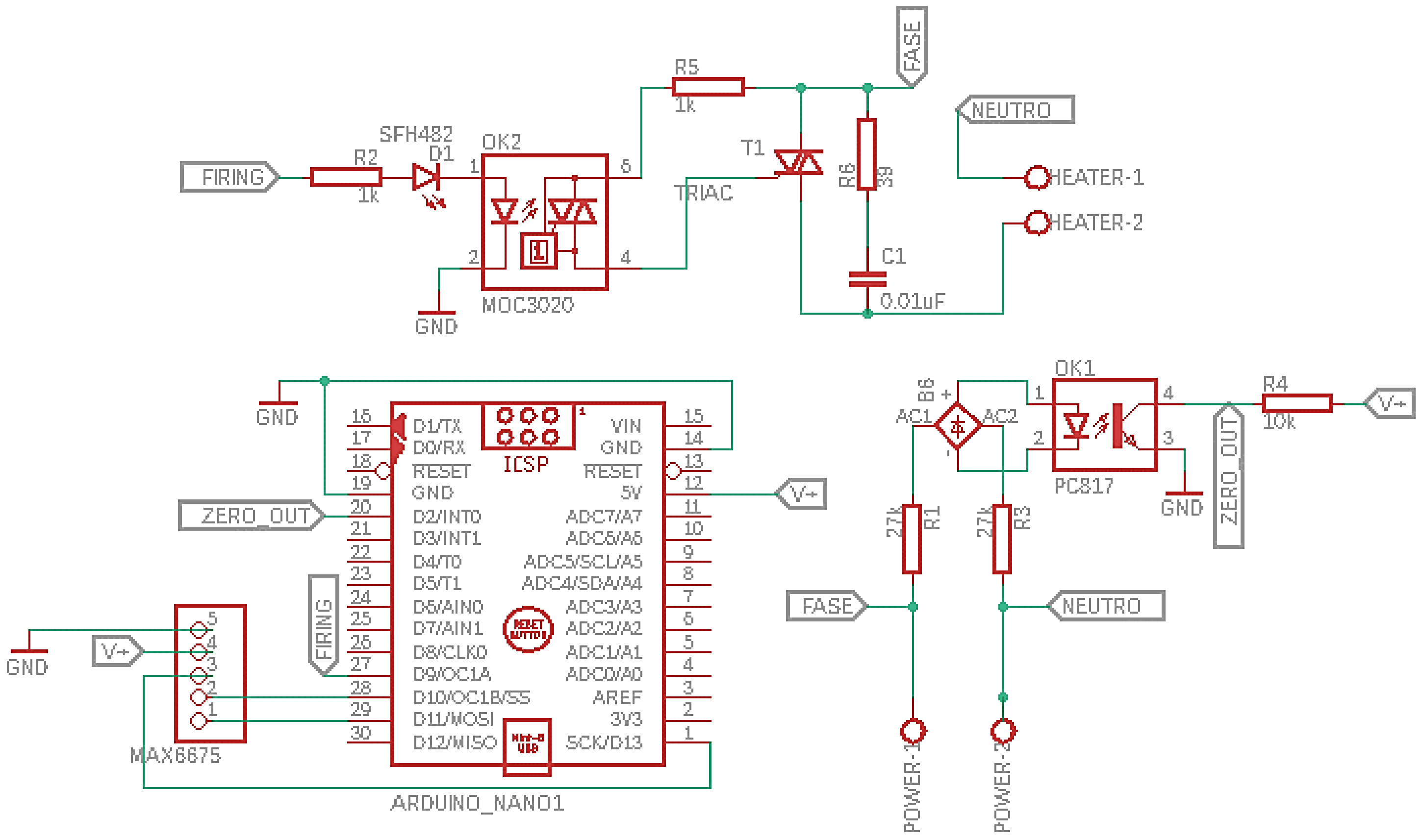


Illustration 11: Heater Circuit Diagram

The heater diagram consists of the Arduino microcontroller (bottom left) as well as an AC zero detection circuit (bottom right) and a heater controller (top). The Arduino is responsible for taking the thermocouple's measurements through the MAX6675 module and detecting the zero intercept of the AC voltage through the AC zero detection circuit's ZERO_OUT. The Arduino microcontroller is also responsible for triggering the heater controller circuit.

The AC zero detection circuit uses the rectified AC current as an input for the optocoupler PC817. This optocoupler then allows current to flow through its phototransistor grounding the 5V voltage. This grounding of V+ allows the microcontroller to detect the zero intercept of the AC voltage.

Lastly, once the Arduino detects the zero intercept, it can trigger the heater controller using the PID implemented through the PID_v1 library. The heater controller also makes use of a triac, which allows for the proper flow of AC current through the circuit, heating the thermal resistor. The triac is needed in this implementation, as the optocoupler is not able to properly handle AC current through the phototransistor; the phototransistor would be damaged which would render the signal from the Arduino microcontroller to the thermal resistor inaccurate.

Code:

Arduino:

The Arduino code implemented makes use of the PID_v1, MAX6675 and DallasTemperature libraries. These libraries allow for the correct operation for the thermal bath, as they make it easier to read the temperature information from the thermocouple as well as calculate the temperature response that the thermal bath should have, through the PID controller implemented in the Arduino microcontroller. The Arduino code can be observed in Annex 1. It should be noted that the user communication is done through Serial communication, as such if the Arduino microcontroller is not connected to the Raspberry PI, the default will be the Arduino's IDE.

Raspberry PI:

The Raspberry PI code consists of a serial communication log between the Raspberry PI and the device located in the /dev/ttyACM0 address, which for this implementation, is the Arduino Microcontroller. The code runs a loop that asks the user for commands, which are sent to the Arduino to control the thermal bath, as well as the light box. The Raspberry PI code can be observed in Annex 2. It should be noted that since the Arduino microcontroller reads the commands through serial communication, the same implementation can be controlled just using the IDE's monitor.

Implementation:

Casings:

The reservoir casing is made from stainless steel, which holds the heated water and the electric heater itself. The water pump's hoses are connected to the reservoir. The following illustration shows the reservoir's implementation.

Illustration 12: Reservoir Casing

The 3mm acrylic thermal bath casing is placed on top of the MDF light box, to properly light the microfluidic system held within. The following illustration shows the implementation. The light box template can be observed in Annex 3 at 1:3 ratio; and the thermal bed can be observed in Annex 4 at 1:2 ratio.

Illustration 13: Thermal Bath Casing

Calibration:

To observe how water circulation affects the thermal bed's temperature, four different tests are conducted. The tests consist of powering the water pump when the reservoir's desired temperature has been reached. This is done using the desired temperature as scale: for the first test the desired temperature is as is. Next, is the temperature minus 0.25°C; then 0.5°C and finally 1°C. Using this information, and analyzing the standard deviation, as well as the EST variable, the test with the best behavior can be identified. Using the information from the identified test, a delta between the average temperature of the reservoir as well as the average temperature of the thermal bed with water circulation can be obtained.

The standard experimental uncertainty is obtained by observing the device's operation under repeatability conditions. For the analysis, the following expression was employed [8].

$$s(q) = \sqrt{\frac{1}{n-1} \cdot \sum_{j=1}^{n} (q_j - \bar{q})^2} \qquad (1)$$

Where n is the number of samples, q es the sample and $\bar{q}$ is the average of the samples.

The EST variable is the contribution of the measured uncertainty by the temperature sensor in the thermal bath. It is represented by the following expression [9].

$$\frac{(Lsup - Linf)}{\sqrt{12}} \qquad (2)$$

Where Lsup is the average of the maximum reservoir temperature and Linf is the average of the temperature in the thermal bed with water circulation. Using this methodology, the following data is obtained.

Maximum Reservoir Temperature	**Water Pump HIGH**	**Standard Experimental Uncertainty (Thermal Bed)**	***Est***
25 ºC	Setpoint – 0.25 ºC	±1.58 ºC	0.685ºC
25 ºC	Setpoint – 0.5 ºC	±1.65 ºC	0.71 ºC
25 ºC	Setpoint – 1 ºC	±0.98 ºC	0.6 ºC
25 ºC	Setpoint ºC	±0.643 ºC	0.596 ºC

30 ºC	Setpoint – 0.25 ºC	±1.3 ºC	0.87ºC
30 ºC	Setpoint – 0.5 ºC	±2.23ºC	1.42ºC
30 ºC	Setpoint – 1 ºC	±1.76ºC	1.17ºC
30 ºC	Setpoint ºC	±1.2168ºC	1.2063ºC
35 ºC	Setpoint – 0.5 ºC	±0.683 ºC	0.62ºC
35 ºC	Setpoint – 0.25 ºC	±3.5ºC	1.44ºC
35 ºC	Setpoint – 1 ºC	±1.45ºC	1.13ºC
40 ºC	Setpoint – 0.25 ºC	±4.5 ºC	1.6ºC
40 ºC	Setpoint – 0.5 ºC	±2.3ºC	1.44ºC
40 ºC	Setpoint – 1 ºC	±3.46ºC	1.96ºC
40 ºC	Setpoint	±4.5ºC	2.37ºC
45 ºC	Setpoint – 0.25 ºC	±6.85 ºC	3.03ºC
45 ºC	Setpoint – 0.5 ºC	±0.5ºC	1.46ºC
45 ºC	Setpoint – 1 ºC	±5ºC	2.1ºC
45 ºC	Setpoint	±2.76ºC	2.3ºC
50 ºC	Setpoint – 0.25 ºC	±4.94 ºC	3.3ºC
50 ºC	Setpoint – 1 ºC	±4.6ºC	2.6ºC
55 ºC	Setpoint – 0.5 ºC	±6 ºC	3.9ºC
55 ºC	Setpoint – 1 ºC	±5.4ºC	3.4ºC

Table 1: Calibration Data

Using the information from the table, the best data set for each reservoir temperature is identified to be implemented in the Arduino code. This is done to obtain the best response from the thermal bed once the user sends a temperature command to the thermal bed.

Results:

PCB:

The PCB implemented as well as the implementation is shown in the following illustration.

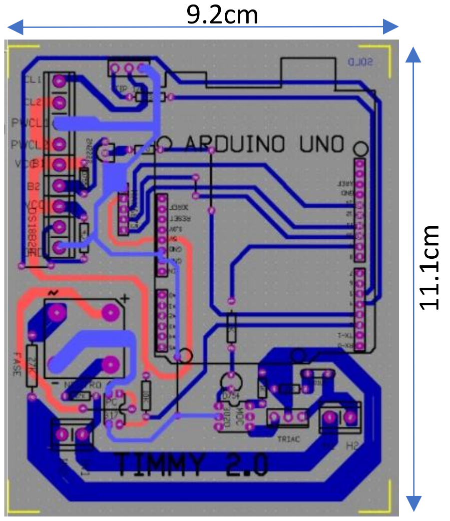


Illustration 14: PCB Diagram

Illustration 15: PCB implementation

A higher resolution image can be observed in Annex 5.

Regression:

For the regression, the data was taken into account from the time the water starts circulating; the heating process is then analyzed. The analysis is done using Sigmoid functions, since the behavior of these closely resembles the behavior of the heating process of the thermal bath. In the beginning, the water heats up rapidly; the heating decreases over time until it stabilizes at the desired temperature. The Sigmoid function constants are obtained using the python scipy.optimize library. The constants obtained are from the following expression.

$$\frac{a}{b+e^{-tc}} \tag{3}$$

Temperature Range	**Water Pump HIGH**	**Regression ***	ΔT *
Temp objective <= 25 ºC	Setpoint - 1 ºC	$y(t) = \frac{1.243e{+}02}{5.386{+}e^{-t*7.051e-03}}$	2 ºC
25 ºC < Temp obj <= 30ºC	Setpoint – 0.25 ºC	$y(t) = \frac{8.903e{+}01}{3.205{+}e^{-t*1.0915e-02}}$	3 ºC
30 ºC < Temp obj <= 35ºC	Setpoint – 0.5 ºC	$y(t) = \frac{5.491e{+}01}{1.701{+}e^{-t*6.376e-03}}$	2,8 ºC
35 ºC < Temp obj <= 40ºC	Setpoint – 0.5 ºC	$y(t) = \frac{3.773e{+}01}{1.032{+}e^{-t*6.625e-03}}$	5 ºC
40 ºC < Temp obj <= 45ºC	Setpoint – 0.5 ºC	$y(t) = \frac{6.547e{+}01}{1.634{+}e^{-t*6.608e-03}}$	5 ºC
45 ºC < Temp obj <= 50ºC	Setpoint – 1 ºC	$y(t) = \frac{3.567e{+}01}{7.825e{-}01{+}e^{3.262e-03}}$	5 ºC

50 ºC < Temp obj <= 60ºC	Setpoint – 1 ºC	$y(t) = \frac{4.0856e+01}{8.681e-01+e^{2.775e-03}}$	5 ºC

Table 2: Results Equations

The regressions are implemented in the Arduino code, after which the device characterization is done to evaluate the data obtained and make sure that the Sigmoid equations closely match the temperatures of the thermal bed.

Characterization:
The characterization is performed once the Sigmoid equations are implemented to observe the how they affect the performance of the thermal bath. The data can be observed below.

Desired Temperature	Water Pump HIGH	Sample Time	Circulation Time	Maximum Reservoir Temperature	Average Thermal Bath Temperature
23.6 ºC	Setpoint – 1 ºC	13.5 min	8.3 min	25.8 ºC	23.07 ºC
27.5 ºC	Setpoint – 0.25 ºC	20 min	15 min	29.4 ºC	27.4 ºC
46.5 ºC	Setpoint – 1 ºC	39 min	22 min	52.2 ºC	46.7 ºC
32.7 ºC	Setpoint – 0.5 ºC	22 min	15min	35.4 ºC	32.4 ºC
38.3 ºC	Setpoint – 0.5 ºC	32 min	19min	43.4C	38.9 ºC
43 ºC	Setpoint – 0.5 ºC	39 min	22min	46.9 ºC	43.45 ºC
57.5ºC	Setpoint – 1 ºC	65min	42min	61.43 ºC	57.54 ºC

Table 3: Thermal Bath Characterization

The graphs obtained by the thermal bath's characterization can be observed in Annex 6-12.

Observations:

it can be observed that the average temperature of the thermal bath matches the desired temperature controlled by the Arduino. It should be noted that for temperatures in the range below 35ºC the testing time is about 25 minutes, while for temperatures above 35ºC, the testing time can reach up to 1 hour. For replicability, it should also be noted that the thermal bed is located at a height of 24.7cm above the heated reservoir.

It should also be noted that the water pump should only be used for about 10 minutes at a time. Intervals of operation of more than 10 minutes can cause damage within the water pump and as such render it unusable. Another restriction that this implementation poses is that it is not thermically isolated from the environment. This is because the user should be able to take both photo and video captures of the microfluidic system. As such, the heating process takes longer than if it were thermically isolated. Once the water starts circulating from the heated reservoir to the thermal bath, there is a temperature difference between that in the reservoir and the water which arrives at the reservoir after circulating. For a range of temperature between 20ºC to 35ºC there is a delta of 2.6ºC. While for temperatures in the range of 36ºC to 60ºC, the temperature difference can be as much as 10ºC.

References:

1. *Baos termostaticos para laboratorio*. FRICAVAL. (n.d.). http://fricaval89.com/productos/aparatos-instrumentos-materiales-equipos/banos/banos-termostaticos-laboratorio.html.
2. Vistronica. (n.d.). *Módulo para Termocupla MAX6675*. VISTRONICA S.A.S. https://www.vistronica.com/sensores/temperatura/modulo-para-termocupla-max6675-detail.html.
3. *Optocoupler MOC3020*. AV Electronics. (2021, February 8). https://avelectronics.cc/producto/optoacoplador-moc3020/.
4. *What is Optocoupler and how it works?* Components101. (n.d.). https://components101.com/articles/what-is-optocoupler-and-how-it-works.
5. *SEN DS18B20 2M*. Sigma Electrónica. (2021, July 14). https://www.sigmaelectronica.net/producto/sen-ds18b20-2m/.
6. Luis, E. R. de. (2018, September 18). *De cero a maker: Todo lo necesario para empezar con raspberry pi*. Raspberry Pi: todo lo necesario para iniciarse como maker desde cero. https://www.xataka.com/makers/cero-maker-todo-necesario-para-empezar-raspberry-pi.
7. Motors 1. (2020) [Online]. Available: http://www.thebox.myzen.co.uk/Workshop/Motors_1.html [Last Accessed: November 2020].
8. Wolfgang A. Schmid y Ruben J. Lazos Martínez. (2020). GUÍA PARA ESTIMAR LA INCERTIDUMBRE DE LA MEDICIÓN [PDF]. Available: http://depa.fquim.unam.mx/amyd/archivero/EstimaciondelaIncertidumbre_32954.pdf
9. Guía técnica sobre trazabilidad e incertidumbre de las mediciones en la caracterización térmica de baños y hornos de temperatura controlada, CENAM, 2008.

Annexes:

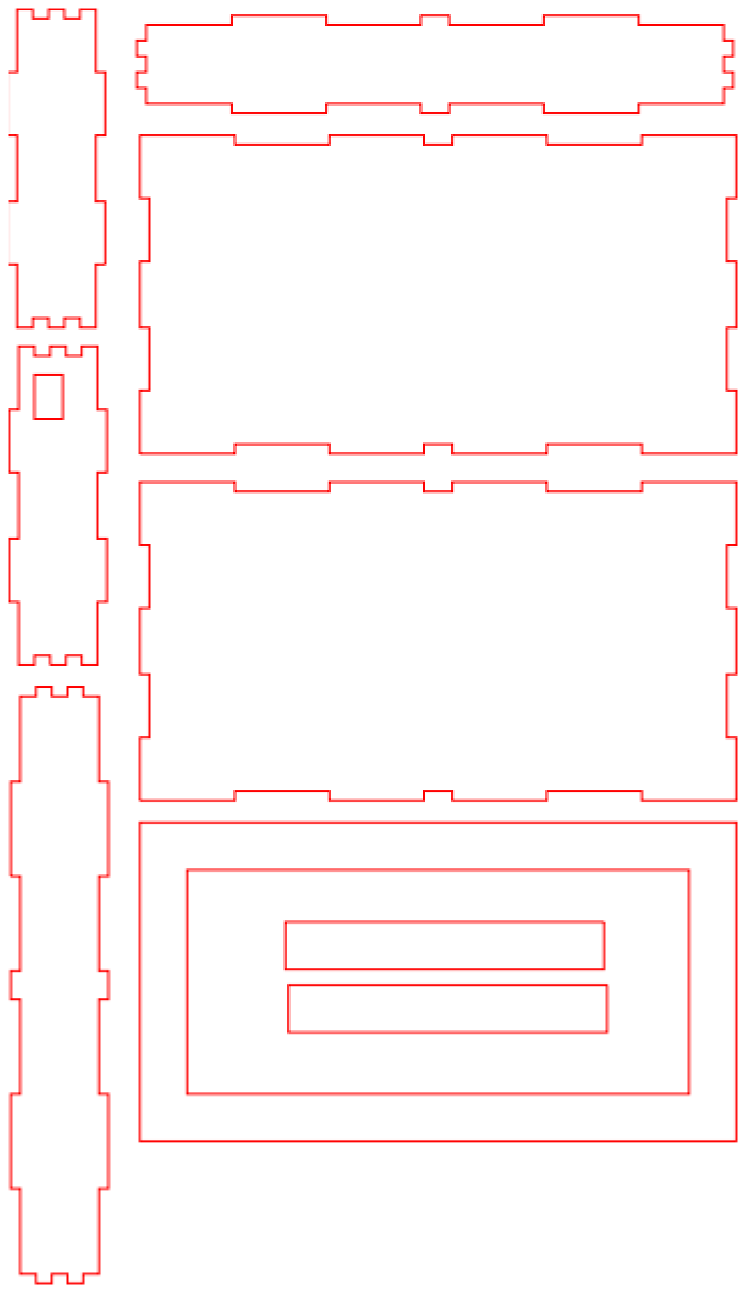
Arduino Code

```
#include <PID_v1.h>
#include <max6675.h>
#include <Wire.h>
#include <OneWire.h>
#include <DallasTemperature.h>

int thermoDO = 11;
int thermoCS = 10;
int thermoCLK = 13;
float tempe;
float tol = 0.5;
const int firing = 9;
const int zc = 2;
int BombaAgua = 3;

OneWire ourWire(4); //Se establece el pin4 como bus de salida
DallasTemperature sensor(&ourWire); //Se declara la variable, en este caso sera sensor

String dataLabel1 = "Temperatura baño";
String dataLabel2 = "Temperatura camas";
bool label = true;

MAX6675 thermocouple(thermoCLK, thermoCS, thermoDO);

double kp = 900;   double ki = 460;   double kd = 460;
double Setpoint,Setpoint_ini , Input, Output;
PID myPID(&Input, &Output, &Setpoint, kp, ki, kd, REVERSE);

String temp;
double temp0;
int pinLuz=5;

void setup() {
  Setpoint = 0;
Setpoint_ini=0;


  myPID.SetMode(AUTOMATIC);
  myPID.SetOutputLimits(0, 6500);

  Serial.begin(115200);
  sensor.begin(); //Se inicia el sensor
  pinMode(firing, OUTPUT);
  pinMode(zc, INPUT);
  attachInterrupt(digitalPinToInterrupt(zc), angle, FALLING);
  pinMode(BombaAgua, OUTPUT); // initialize the digital pin as an output.
  Serial.println("TempBaño TempCamas ");
  pinMode(pinLuz,OUTPUT);
 }

void loop() {
  if (Serial.available()) {
    temp = Serial.readStringUntil('\n');

    if (temp.startsWith("luz")) {
      String valor1="null";
      //Serial.println ("Received: " + temp );
      valor1 = temp.substring(4);

      if(valor1 == "on")   digitalWrite(pinLuz,HIGH);
      else if(valor1 == "off") digitalWrite(pinLuz,LOW);
//      Serial.println(valor);
    }

else if (temp.startsWith("temp")) {
      String valor="null";
     // Serial.println ("Received: " + temp );
      valor= temp.substring(5);
      Setpoint_ini = valor.toDouble();
     // Serial.println ("Received: " + temp );
      //Serial.println (Setpoint );
}

else {
      temp = "Received: " + temp + ", command not found; hlp for list";
      Serial.println(temp);
    }
  }

    tempe = thermocouple.readCelsius(); delay(178);
    Input = tempe; // IMPORTANTE!: Definir la entrada del sistema
    myPID.Compute(); // Calcular acción de control
    digitalWrite(firing, LOW);

 sensor.requestTemperatures(); //Se solicita leer la temperatura
 float temp1= sensor.getTempCByIndex(0); //Se obtiene la temperatura en ºC
 digitalWrite(firing, LOW);

if (Setpoint_ini <= 25){  //// Temp 1
  Setpoint = Setpoint_ini + 2;

  if ((tempe >= Setpoint - 1) && (Setpoint_ini!=0) ) {
    digitalWrite(BombaAgua, HIGH); // turn the motor on by making the voltage HIGH
    }
 else {
   digitalWrite(BombaAgua, LOW);  // turn the motor off by making the voltge LOW
 }
  }

else if (Setpoint_ini<=30 && Setpoint_ini >25){ ////Temp 2
  Setpoint = Setpoint_ini + 3;

  if ((tempe >= Setpoint - 0.25) && (Setpoint_ini!=0) ) {
    digitalWrite(BombaAgua, HIGH); // turn the motor on by making the voltage HIGH
    }
 else {
    digitalWrite(BombaAgua, LOW);  // turn the motor off by making the voltge LOW
  }

}
else if (Setpoint_ini<=35 && Setpoint_ini >30){ ////Temp 3
  Setpoint = Setpoint_ini + 2.8;

  if ((tempe >= Setpoint - 0.5) && (Setpoint_ini!=0) ) {
    digitalWrite(BombaAgua, HIGH); // turn the motor on by making the voltage HIGH
    }
 else {
    digitalWrite(BombaAgua, LOW);  // turn the motor off by making the voltge LOW
 }
  }

  else if (Setpoint_ini<=40 && Setpoint_ini >35){ /// Temp 4
  Setpoint = Setpoint_ini + 5;

  if ((tempe >= Setpoint - 0.5) && (Setpoint_ini!=0) ) {
    digitalWrite(BombaAgua, HIGH); // turn the motor on by making the voltage HIGH
    }
 else {
    digitalWrite(BombaAgua, LOW);  // turn the motor off by making the voltge LOW
 }
  }

  else if (Setpoint_ini<=45 && Setpoint_ini >40){ ///Temp 5
  Setpoint = Setpoint_ini + 5;

  if ((tempe >= Setpoint - 0.5) && (Setpoint_ini!=0) ) {
    digitalWrite(BombaAgua, HIGH); // turn the motor on by making the voltage HIGH
    }
 else {
    digitalWrite(BombaAgua, LOW);  // turn the motor off by making the voltge LOW
 }
  }
  else if (Setpoint_ini<=50 && Setpoint_ini >45){ ///// Temp 6
  Setpoint = Setpoint_ini + 5;

  if ((tempe >= Setpoint - 1) && (Setpoint_ini!=0) ) {
    digitalWrite(BombaAgua, HIGH); // turn the motor on by making the voltage HIGH
    }
 else {
    digitalWrite(BombaAgua, LOW);  // turn the motor off by making the voltge LOW
  }
  }
  else if (Setpoint_ini<=65 && Setpoint_ini >50){ /// Temp 7
  Setpoint = Setpoint_ini + 5;

  if ((tempe >= Setpoint - 1) && (Setpoint_ini!=0) ) {
    digitalWrite(BombaAgua, HIGH); // turn the motor on by making the voltage HIGH
    }
 else {
    digitalWrite(BombaAgua, LOW);  // turn the motor off by making the voltge LOW
 }
  }

 if (Setpoint_ini!=0){
 Serial.print(tempe);
 Serial.print(" ");
 Serial.println(temp1);
 delayMicroseconds(10);
}
}

void angle() {
  if (tempe > Setpoint && tempe - Setpoint >= 0.25) {
    digitalWrite(firing, LOW);
  }

    else if (Setpoint>tempe){
      delayMicroseconds(Output);
      digitalWrite(firing, HIGH);
      delayMicroseconds(500);
      digitalWrite(firing, LOW);
    }
  }
```

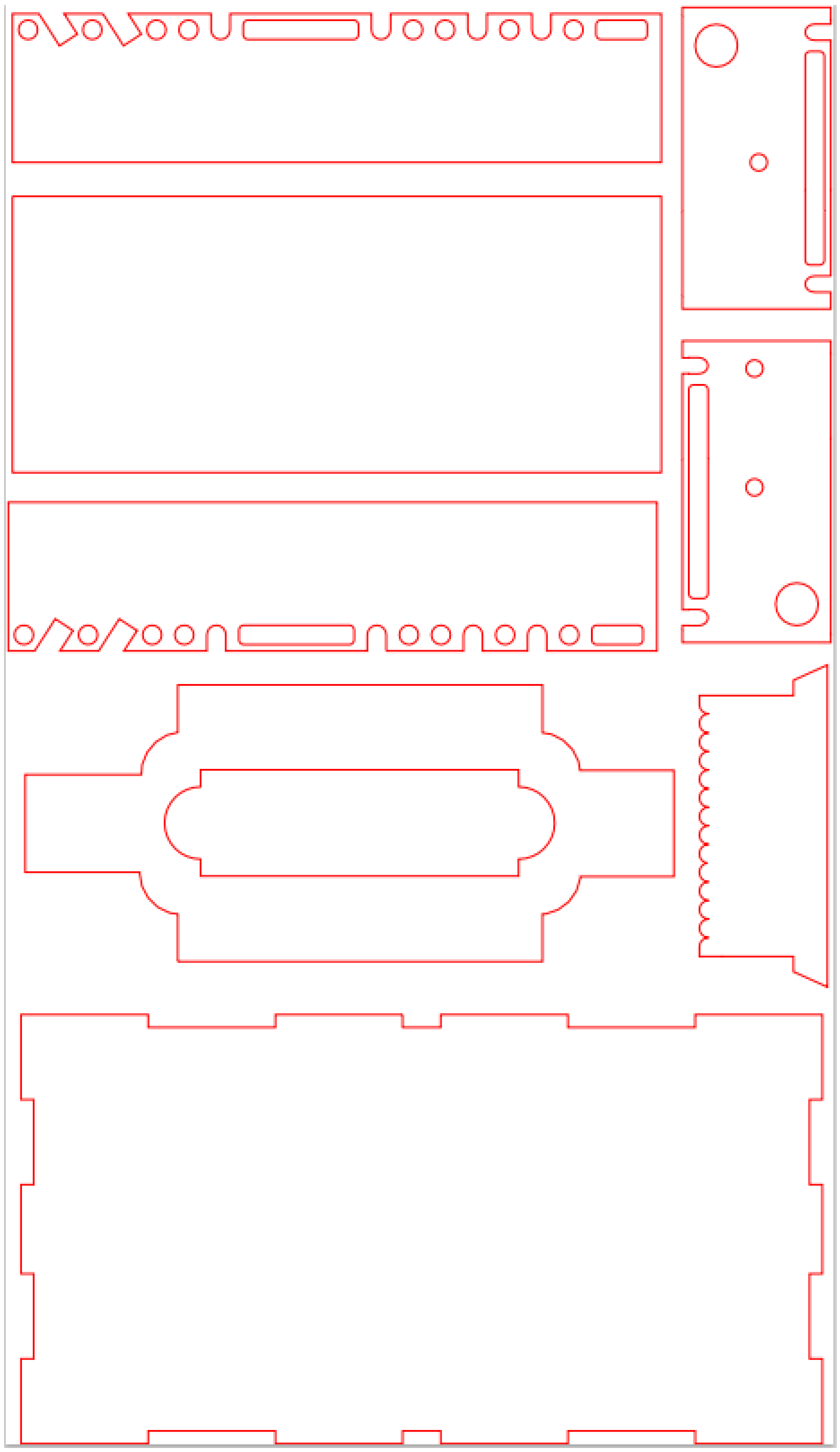
Raspberry PI code

```
import serial
import time

arduino=serial.Serial('/dev/ttyACM0',baudrate=115200)
arduino.open()
txt=''

while True:
        var = raw_input("Introducir un Comando: ")
        arduino.write(var)
        time.sleep(0.1)
        while arduino.inWaiting() > 0:
        txt += arduino.read(1)
        print txt
        txt = ''
arduino.close()
```

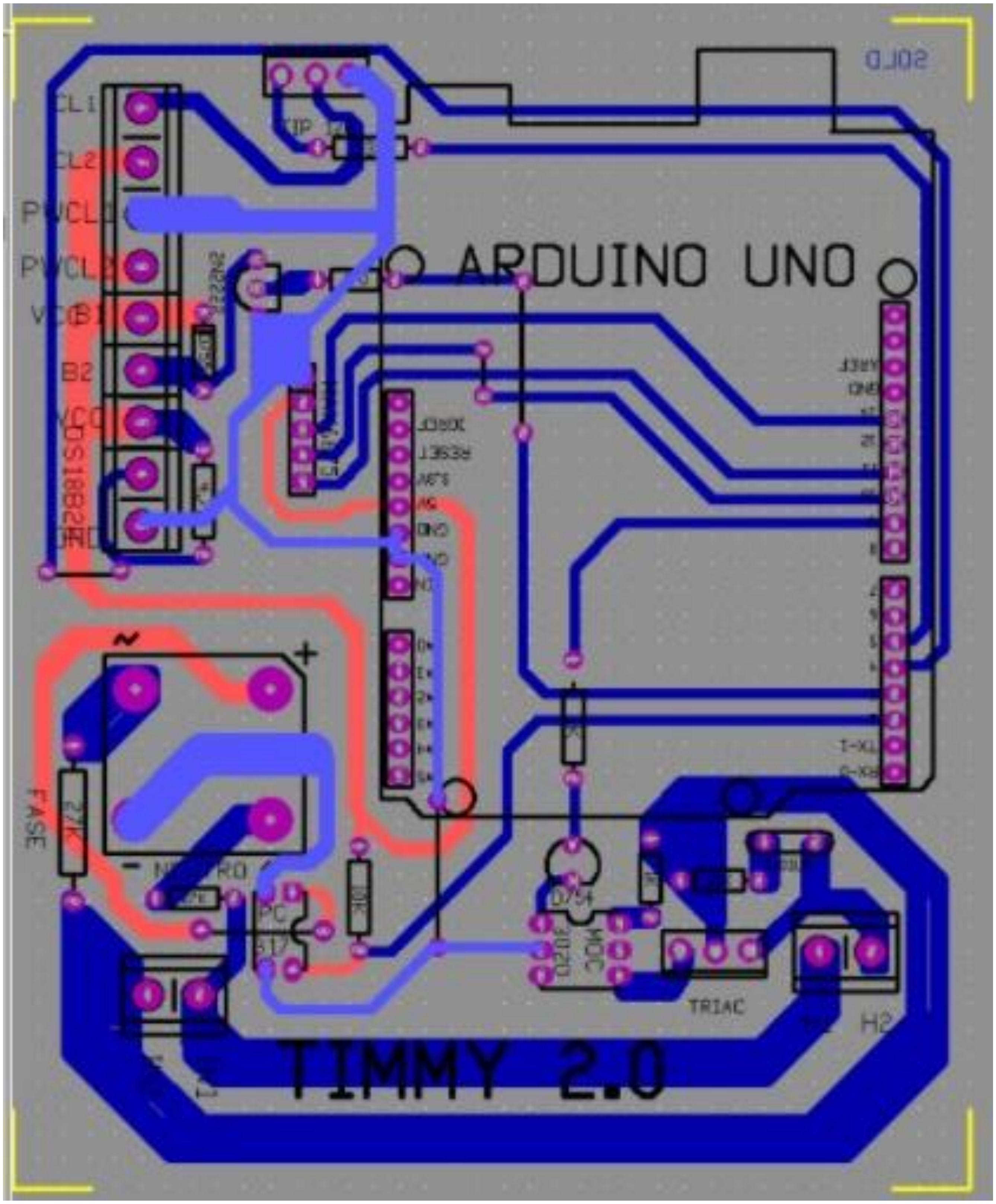
Light Box Template



Thermal Bath Template

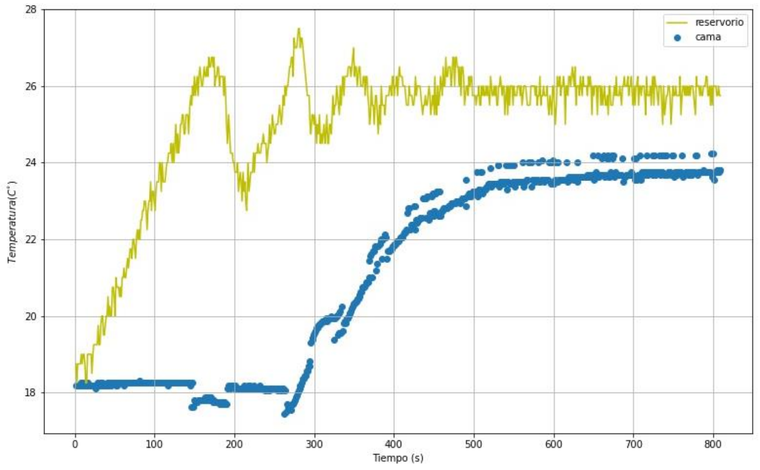


PCB Diagram

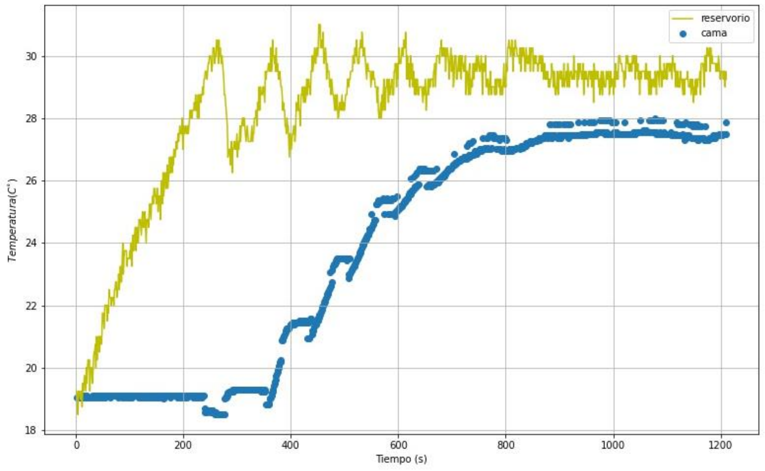


ARDUINO UNO
TIMMY 2.0
CL1
CL2
PWCL1
PWCL2
VCC
B1
B2
VCC
DS18B20
GND
FASE
27K
NEUTRO
PC
817
10K
D754
MOC
3020
TRIAC
H1
H2
TIP
IOREF
RESET
3.3V
5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5
AREF
GND
TX-1
RX-0

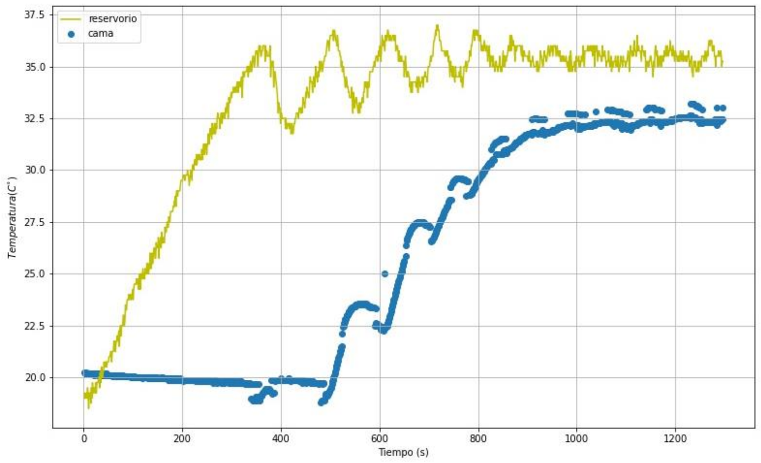
Device characterization at 23.6 ºC



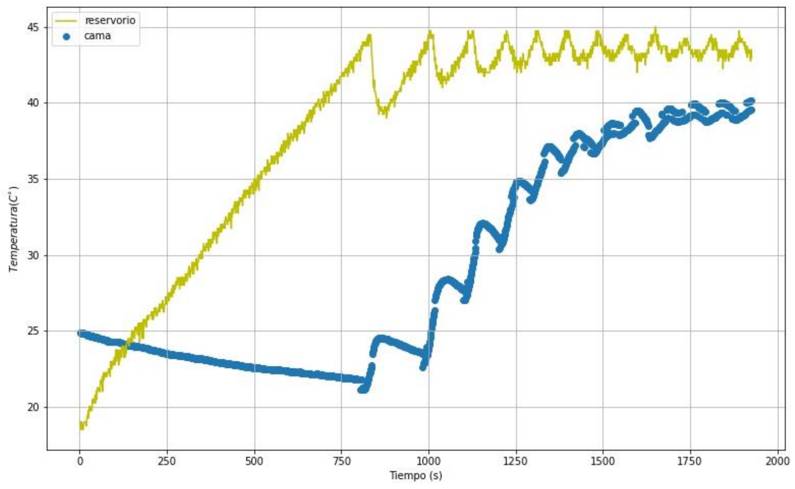
Device characterization at 27.5 ºC



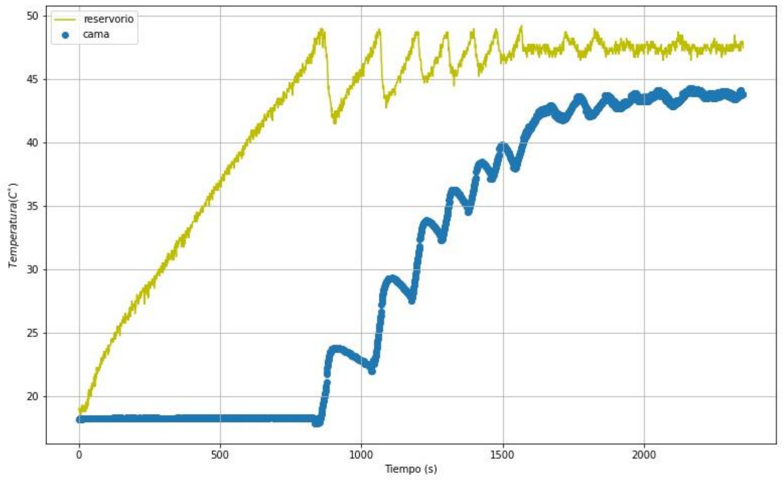
Device characterization at 32.7 ºC



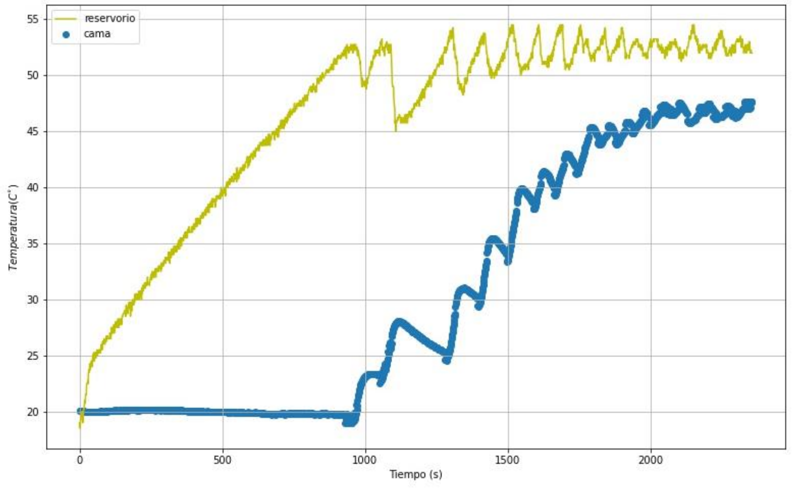
Device characterization at 38.3 ºC



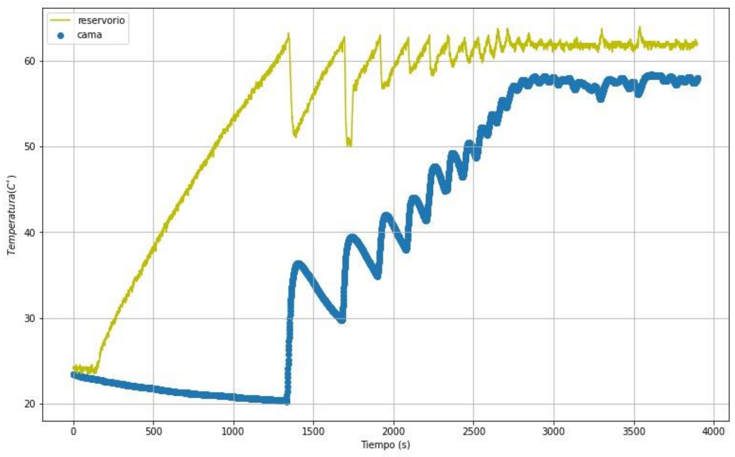
Device characterization at 43.0 ºC



Device characterization at 46.5 ºC



Device characterization at 57.5 ºC


reservorio
cama
Temperatura(C°)
Tiempo (s)