POLITECNICO DI TORINO

Faculty of Engineering
Master of Science in Electronic engineering

Master Thesis

Temperature control system development in microfluidic environment for biological purposes

Supervisor
Prof. Danilo Demarchi

Candidate
Enrico Picco

University tutor
Pázmány Péter Catholic University - Faculty of Information Technology and Bionics
Dr. Laki András József

July 2020

Abstract

Microfluidic devices are rising in importance during the last decades. Biomedical science is one of the most important fields of application of these devices, since they are one of the main innovative tools that can help life and medical scientist to solve biological and biomedical problems through the application of engineering tools. For example, the applications at the interface of microfluidics and biomedical sciences have given birth to countless devices and research fields: single cell analysis, circulating tumour cells (CTC) detection, food safety diagnostic, lab-on-chip (LOC), blood analysis, and many others.

In particular, there are some kind of microfluidic devices (i.e. foodborne parasitology chips) that requires precise working condition to perform in a correct way.

The system developed and presented in this thesis is an embedded system that can assure an accurate temperature control of the microfluidic device , and it is also compatible with the instruments of the Microfluidic Laboratories of the Pázmány Péter Catholic University in terms of size , instruments needed, and power consumptions. The main application for which the system was needed by the staff of the Laboratory is for detection of foodborne parasites by filtration microfluidic devices. The parasite detected in this devices is Trichinella spiralis larvae, which are organisms that can be maintained alive only in a temperature range close to the human body, since they live and grow in the digestive human system and then spread further in other tissues (blood, muscles, lymph).

The correct functioning of the embedded system has been tested and verified. It is capable of maintain the temperature of the microfluidic device fixed to a precise temperature for a specific amount of time: both these parameters are decided by the user. This enhances the flexibility, making this embedded system able to work with any microfluidic device that requires a temperature-controlled environment.

Acknowledgements

If I had to thank all the people that supported me through these years of hard work, this paragraph would occupy more space than the explanation of the work done.

In primis, I would like to thank my family for having supported my studies and taught me the importance of hard working.

As a student, I would like to thank all the people that played an important role in my academic career: Prof. Danilo Demarchi for giving me the opportunity to have such an achieving experience abroad, Dr. Laki Andràs for his great support and availability during the time that I spent in Budapest, and all the students that shared with me this journey.

As a person, I would like thank all my friends from Savigliano which were close to me during these years, for all the amazing moments and laughs shared that are the fuel to keep moving forward.

Last, but absolutely not least, I would like to thank from the bottom of my heart all the people that I met during my stay in Budapest, especially my flatmates: thanks for having filled me with enthusiasm toward whatever challenge I will face in my life , and for showing me that it is possible to have a family even when you are far from home.

Contents

List of Figures

List of Tables

Chapter 1

Introduction, Motivations and Goals

This thesis project has been carried out during a mobility period of five months, in the spring semester of the academic year 2018/2019 at the Pázmány Péter Catholic University of Budapest. The work has been done in the Microfluidic Laboratory situated in the facilities of the Faculty of Information Technology and Bionics, under the supervision of Dr. Laki András József.
The project is a **Temperature control system for microfluidic devices employed in biological applications** which was required by the Microfluidic Laboratory in order to perform studies on microfluidic chips for the study of the parasite *Trichinella Spiralis*, but the system was adapted to cover a broader range of applications. In section 1.1 will be given an overview about the kind of microfluidic devices taken into account in the design of the system while section 1.2 gives an example of application, which is the study of *T. Spiralis*.

1.1 Microfluidics

Nowadays microfluidic devices are rising in importance in many fields, and one of the main applications is the development of lab-on-chip (LOC) devices as point-of-care (POC) diagnostic tools. Usually LOC devices includes several features and modules, such as sample transport, sample preparation, separation, detection and analysis module [2].
One of the key points in these devices is the miniaturization of reactions and assays, which confer many advantages over "macro scale" techniques: lower volumes, higher surface to volume ratio, shorter diffusion distances, smaller heat capacities, faster heat exchange, shorter assay times, improved parallelization and better process control. Fort these reasons, POC diagnostic test devices provide fast results on a wide

range of medical tests [3].
POC devices have usually to deal with biological samples (blood,serum, etc.) or with pathogen detection (bacteria, parasites, etc.) hence one of the vital parameters to be controlled for this kind of applications is the temperature: and for this reason a temperature control system for microfluidic devices plays a fundamental role.
Since microfluidic technologies are an efficient alternative of serologic separations of blood borne parasites, one possible application is the detection of food-borne parasites by the use of filtration microfluidic devices: these techniques offer an opportunity for separating and detecting different parasites, for example the *Trichinella spiralis* (*T. Spiralis*). This last is the main application for which the temperature control system has been created, but the project has been adapted to cover a broader range of applications in which the temperature of a microfluidic chip has to be kept under control.

1.2 Application: Trichinella Spiralis

T. Spiralis is not only a threat to human health, but also represents an economic problem. Infection of humans occur when Trichinella larvae are ingested, and they are found in muscle tissues of animal's meat, typically under cooked pig or horse meat [4]. The lowest infection dose causing diseases in not clearly defined but is approximately between 100 and 300 larvae ingested [5].*T. Spiralis* is a small nematode with size of around 1 mm, with differences between genders (females are smaller than males) [6]. The infection begins when an individual (a human or an animal) ingests contaminated meat containing larvae. The stomach acid melts the capsule-like cyst and the larvae is released and enters in the small intestine [7] and its migratory phase begins, as shown in figure 1.1. During its lifetime, a single worm of *T. Spiralis* can generate around 500-1500 newborn parasites, that spread in the muscle fibers using the circulatory system. Once in the muscle fibers, they become infective in 15 days and remain for months or even years [8] [9].

The early diagnosis of human trichinellosis is difficult because the observed symptoms are not specific enough [10], the standard tests available to diagnose it are not sensitive to light infections and to the early stage of infection [11].
The technique for the detection of *T. Spiralis* for which the temperature control system is needed is called *Microprecipitation (MP)*: when an antigen and an antibody come in contact, the reaction manifests with visible precipitate. In our application the *T. Spiralis* are incubated in a serum containing specific antibody, and if they are

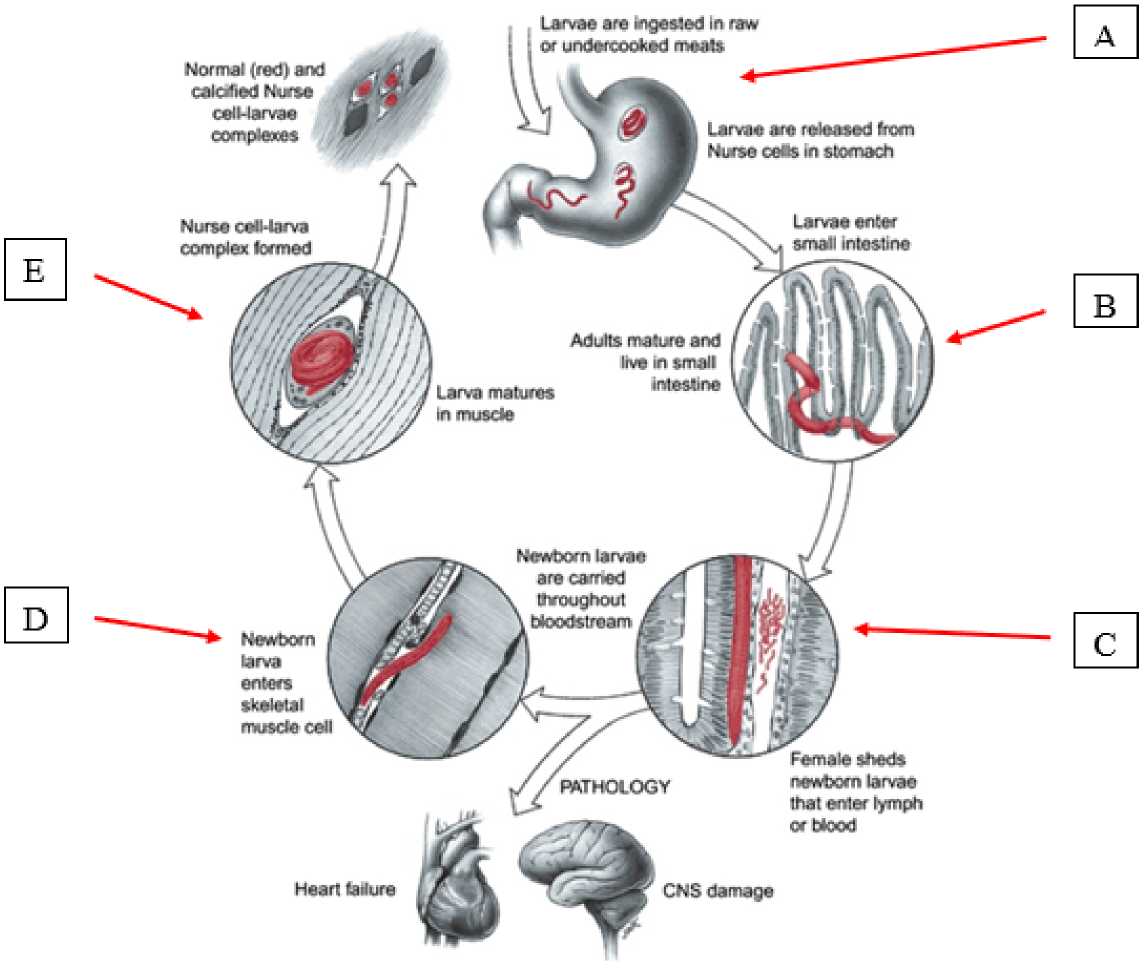

Figure 1.1: T. spiralis life-cycle: A) Larvae are ingested in raw or under cooked meat, B) Live in small intestine, C) Newborn larvae are carried throught bloodstream, D) Newborn larva enters skeletal muscle cell, E) Larva matures in muscle [9]

incubated at 37 °C for 24 -36 hours they produce microscopically visible precipitate, thus making possible to detect their presence [12].

1.3 Final goal

The temperature control system that has to be designed has to meet different requirements. First of all, it has to be compatible with the tools of the Microfluidic Laboratory of the University, since it will be used there: this means that it has to fit in the inverted microscope of the Laboratory, and also that it has to be designed so that it can be used with the power supply and the other electronic equipment. Therefore, the system should be capable of keep the temperature constant to 37 °C for at least 24 hours as required in the application of *T. Spiralis*. However, in order

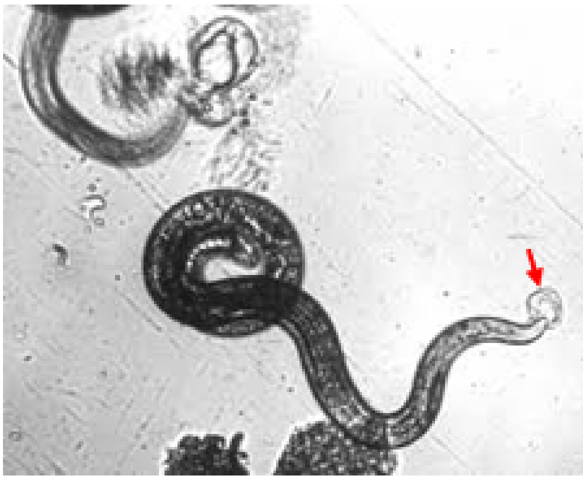

Figure 1.2: Trichinella larvae surrounded by bubble-shaped precipitates after 24 hours of incubation at 37 °C [12]

to make it more useful and flexible, the system is designed so that the user can select both the desired **temperature** of the microfluidic device and the amount of **time** in which the temperature of the chip has to be controlled.
Finally, the system is designed to be portable, stand-alone and robust.
All the design choices are explained in the following chapters: first of all it is given an **overview** of the control system, then a Simulink **mathematical model** is created in order to perform simulations and choose the best components; next, the **hardware** and **software** employed are described in detail. After that there is a description of the **3D-printed structures** used to build the system and finally the description of the project ends with the **Testing and final results**.

Chapter 2

System Overview

The previous chapter is focused on the *main motivations* and *goals* of the work, while from now on the discussion will be focused on the *realization* of the system. Since at the beginning the system had to be built from scratches, a design process flow has been followed: this flow is exploited in figure 2.1.

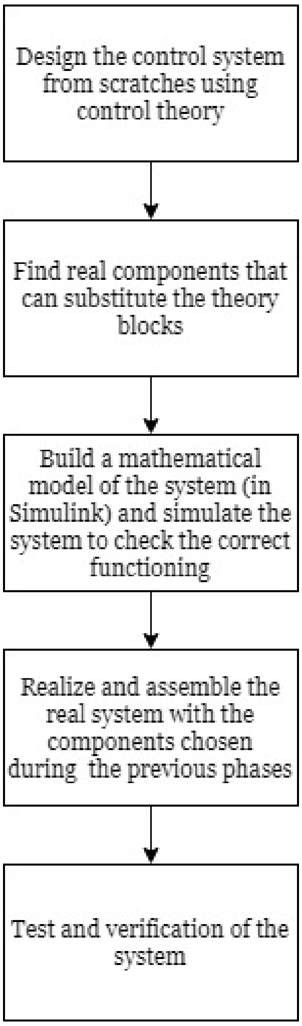

Figure 2.1: Design flow of the overall system

The basic idea is to employ a **closed loop control system**. All the specifications of the system (blocks used and signals) and the theory are explained in the next sections.

2.1 Closed loop control system

A **closed loop control system** is a set of mechanical or electronic devices that automatically regulates a process variable to a desired state or set point without human interaction [13, 14]. The main advantages of choosing a close loop systems over an open loop systems are namely the sensitivity to disturbances and the inability to correct for these disturbances. [15]
The basic idea of the closed loop system employed in this project in exploited in picture 2.2.

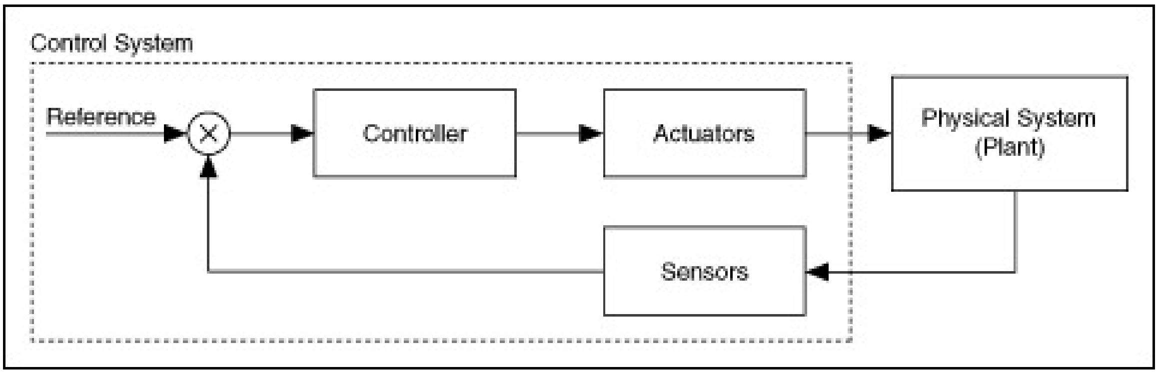

Figure 2.2: Closed-loop control system

A brief explanation of the blocks :

- the **reference signal** , which is the input variable of the system: in this case the reference signal is the temperature selected by the user;
- the **controller**; its input is the error signal (i.e. the difference between the reference and the feedback signals) and gives as output the control signal;
- the **actuator**; receives the control signal and transform it into an actuating signal which drives the plant;
- the **physical system** (plant) is our system that we want to control;
- the **sensor** reads the output from the plant and transform it into a feedback signal which is fed back to the beginning of the system, subtracted to the reference so that the error signal can be constantly updated.

In the next sections all the blocks and signals will be explained in detail.

2.2 Set point (reference)

The **set point** or **reference signal** is the input of the system. In this temperature control system the *set point* is the desired *input temperature* of the microfluidic device, which can be selected by the user.

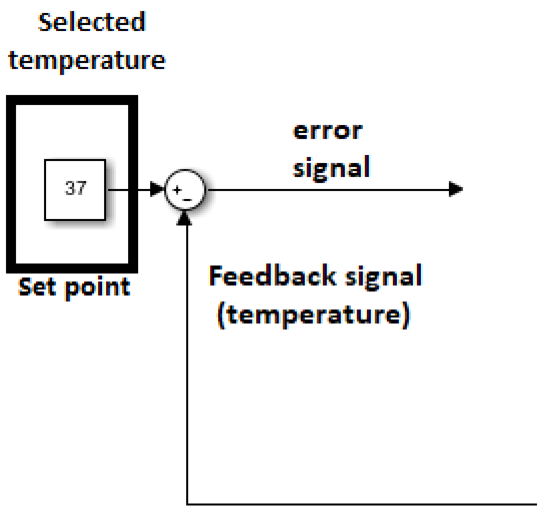

Figure 2.3: Set point block with basic signals highlighted

The *input temperature* is thus constantly subtracted from the *feedback temperature* coming from the temperature sensor, obtaining as a result the *error signal* which drives the controller; ideally, after a certain time the *input temperature and the feedback temperature* should become the same, leading to an error signal equal to zero. In reality, the error signal will never be exactly zero because of many effects like oscillations, bias errors, etc.

2.3 PID controller

The **controller** is one of the most important elements of the closed loop system and the choice of this block is of great relevance for the correct behaviour of the system.

The controller chosen for this system is a **PID controller** , suitable for the temperature control application [16] being a good trade-off between performance and complexity [17, 18] . If tuned correctly, it can improve the stability of the system,

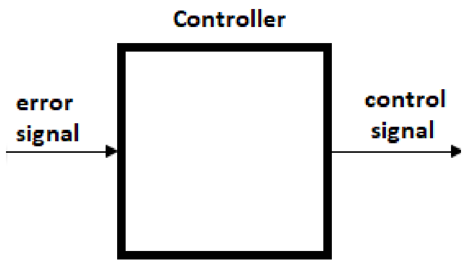

Figure 2.4: Controller block with its input and output signals

reduce the steady state error to zero and reduce the response time needed to reach the input (reference) value.
PID controllers have to be tuned using the three variables $\boldsymbol{K_p}$, $\boldsymbol{K_i}$ and $\boldsymbol{K_d}$, whence the name; the structure of these controllers is shown in figure 2.5 and explained below.

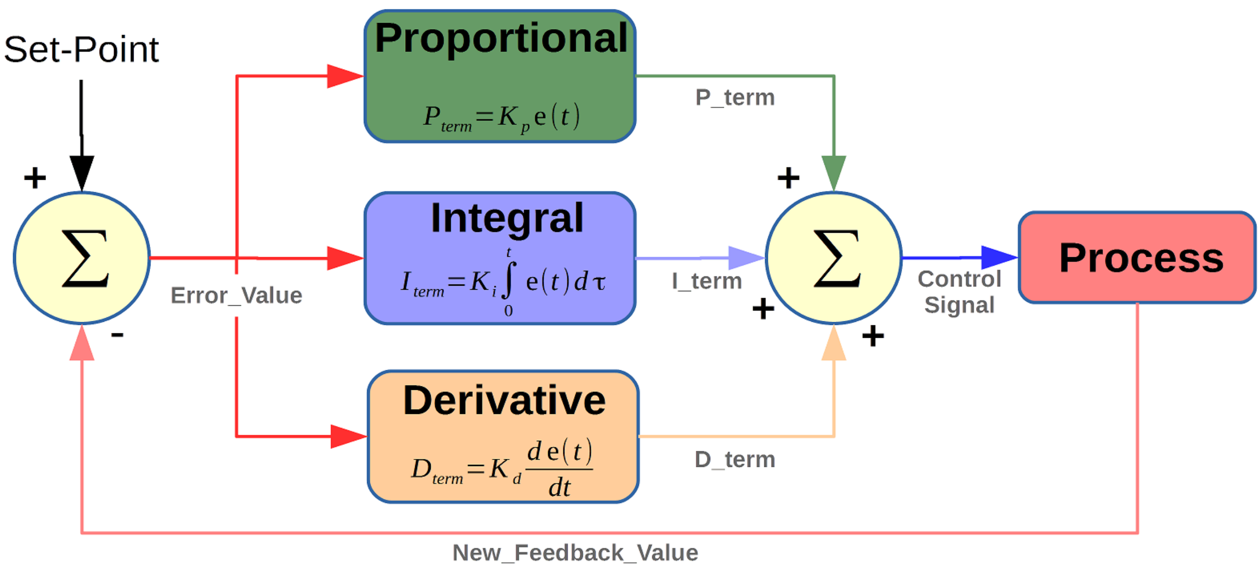

Figure 2.5: PID controller structure

The control signal (i.e. the output of the PID controller) $u(t)$ is a function of the error value $e(t)$ according to the equation 2.1

$$u(t) = K_{\mathrm{p}}e(t) + K_{\mathrm{i}} \int e(t)dt + K_{\mathrm{d}}\frac{\partial e(t)}{\partial t} \tag{2.1}$$

where the first contribution is called *proportional*, the second *integral*, the third *derivative*. More in detail:

- the *proportional contribution* compensates for the present error, it is a multiplication of $e(t)$ multiplied by the proportional gain K_p; in general a large value of K_p increases the speed of the system but can also generate an oscillation if too large;

- the *integral contribution* compensates for the errors in the past, it is obtained by multiplying the integration of $e(t)$ and the integral gain K_i; it helps the system to reach the reference value faster as well as decreasing the steady state error;
- the *derivative* contribution is a prediction of errors in the future, based on the present rate of errors , and it is a multiplication between the differentiation of $e(t)$ and the derivative gain K_d; it improves the stability of the system.

The transfer function of the PID controller is obtained using Laplace transform to equation 2.1, obtaining equation 2.2:

$$U(s) = K_\mathrm{p} + \frac{K_\mathrm{i}}{s} + K_\mathrm{d} s \tag{2.2}$$

The tuning of the PID controller, i.e. the choice of the three constants K_p, K_i and K_d, has been has been carried out with the Ziegler – Nichols method; the explanation, application and results of this method are exploited in section 3.3

2.4 Actuator

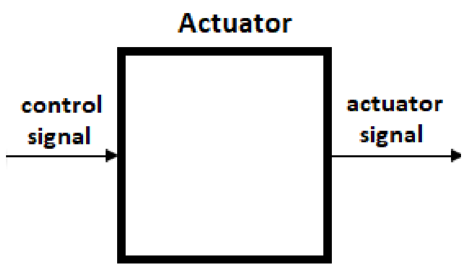

Figure 2.6: Actuator block with its input and output signals

The next block is the actuator: as exploited in picture 2.6, it has as input the control signal coming from the controller and as output the actuator signal that will be fed straight into the plant .
In the real word, the function of this block is to convert the input electric control signal into a precise heat flux which is responsible of the temperature of the microfluidic device under test.
How to implement this block in the real word as been one of the most important

decision in the design of the system, since there is a huge choice of solutions and components that can be used in a temperature control system. The main choices, that are explained in section 2.4.1 and 2.4.2 are related to the **heating element** and the **driver**.

2.4.1 Heating element : Thermoelectric cooler/heater (TEC)

The heating element is the device responsible of converting the electrical energy into heat.
For this system an external heating approach is used, which requires obviously an external heater. Two categories of these devices have been taken into account for the choice; their differences is the physical process employed to generate heat:

- *resistive heaters*: they use the Joule heating (also known as Ohmic heating or resistive heating) , i.e. the process by which the passage of an electric current through a conductor produces heat;
- *thermoelectric heaters*: they convert directly the temperature difference in electric voltage and vice versa via termocouple;.

Taking into account both several examples in technical literature [19] [20] [21] and a trade-off between performances and costs, the device chosen is the **Peltier cell**, also known as **Thermoelectric Cooler (TEC)** .
It is a thermoelectric device built with many Peltier - effect junctions in series. It is basically a solid - state heat pump with the aspect of a thin square plate: one of the two sides absorbs heat, while the other one emits heat. The direction of the heat transfer depends on the sense of the DC current applied at the nodes of the device.

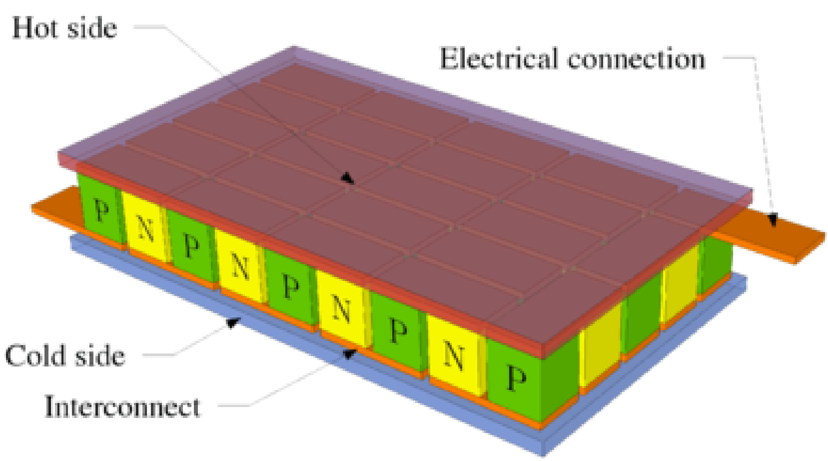

Figure 2.7: Structure of a Peltier cell

A common Peltier cell is made by two doped semiconductor materials (n-type and p-type), connected by a copper slat. If a voltage drop is applied between the two semiconductors, one of the two slats is heated while the other one cools down; if the voltage is reversed, also the flux of thermal energy is reversed.
The main problem of the cell is the control of the current intensity and the correspondent increase of heat; if the thermal source changes value of heating emission, also the increase made by the cell have to change accordingly. This change has to be made with the help of temperature sensors , so that with an appropriate close-loop feedback circuit , the current intensity supplied to the cell can maintain the temperature in the admitted range.
A more rigorous discussion about the equations that model the functioning of the cell (i.e. the equations that relate heat flux, voltages and currents) is presented in section 3.2.2: in this section the behaviour of the Peltier cell has been described from a mathematical point of view, in order to be implemented in the Simulink model.

2.4.2 Driver : PWM and H-Bridge

The next component to be selected is the driver of the TEC: the selection scheme of the TEC driver depends on many requirements:

- *mode of operation*: if both cooling and heating modes of operation are needed, the driver should be able to change the output polarity;
- *power supply*: driver with single power supply is better suited with the instrumentation of the Microfluidic laboratory;
- *maximum output voltage and current* of the driver, which must not exceed the maximum admitted input values of the TEC in order to avoid the breakdown of the Peltier cell;
- *efficiency* and *noise compliance*.

Traditionally, depending on these requirements, many different techniques are available for the control of amplitude and direction of the TEC current [22]: ***linear driver***, ***switch mode power supply (SMPS)*** and ***H-Bridge driver***.
In *linear drivers* the switches operate continuously in the active region, thus with continuous heat dissipation; furthermore their efficiency is quite low (it can not be increased more than 60%) , and for providing bidirectional current they need two different power supplies. For these many reasons, they were discarded.
In *SMPS* the switches operate in cutoff and saturation region, thus the power losses are lower. But also for this kind of drivers two different power supplies are needed

to provide bidirectional current in output, so they were also discarded.
Besides the traditional techniques, many different solutions were provided by the technical literature such as dedicated integrated circuitry [23, 24] or use of current pulses [25], or a customized temperature controller [26] : unfortunately the main drawback of this solutions is the cost, since all of them employ components or devices which are not available in the Microfluidic laboratory and are too expensive to purchase .

Taking into account all these considerations, the driver chosen is a **H-Bridge** driven with **Pulse Width Modulation (PWM)**, which is a traditional techniques used also in many temperature control systems employed in modern applications [27, 28].
H-Bridge is a MOSFET based bridge , and its configuration allows the control of both the amplitude and the direction of the current flowing in the TEC. Due to its great flexibility, it is used in countless applications, such as motor controllers , AC/AC converters, DC-to-AC converters and many other kind of power electronic components. One of the main advantages is that with this type of driver only one power supply is needed for bidirectional output current; furthermore it also ensures good regulation, good precision and even fewer power losses.

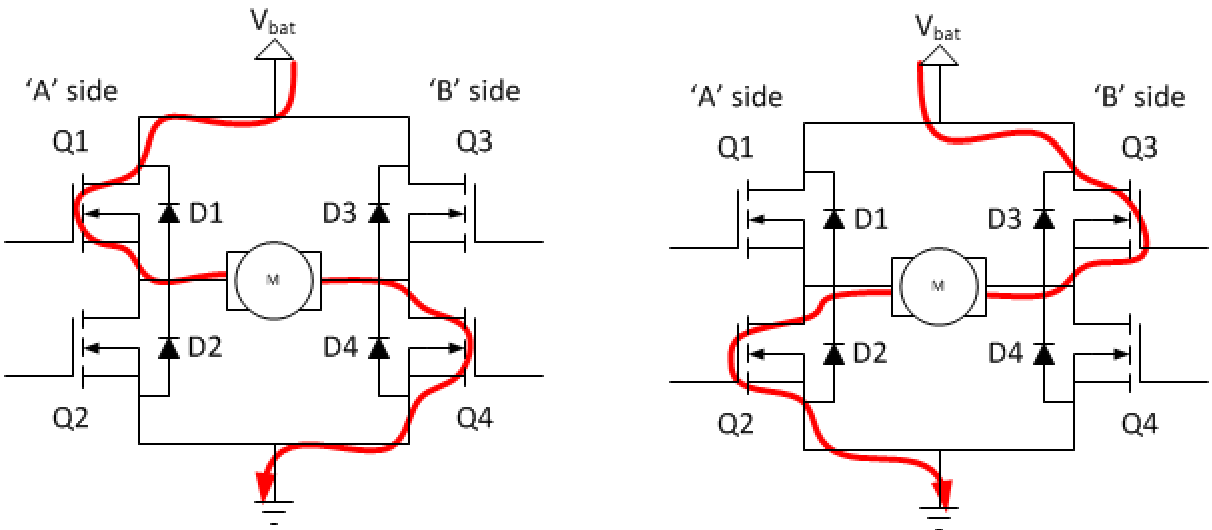

Figure 2.8: H-Bridge circuit; its bidirectionality is highlighted by showing (red lines) the two opposite current directions possible.

In order to reduce even more the power consumption of the driver, **Pulse-width modulation (PWM)** is employed to drive the H-Bridge.
PWM is a digital modulation of an electric signal which allows to obtain an analog signal using a digital source: the resulting analog signal depends on the duty cycle

(i.e. the ratio between the period of high signal and the total period of the signal) of the digital square wave. It is used to change the voltage, i.e. the power, transmitted to a generic load: it is used in several applications such as Dc motors, LEDs, and power supplies. The main advantage of this technique is the smaller power consumption when it is used with transistors or MOSFET : the power is the product between the voltage difference across the device and the current that flows into it, and since when a transistor conducts the voltage difference across it is almost zero and when it is not conducting the current is zero, the power consumption in both cases is almost zero.

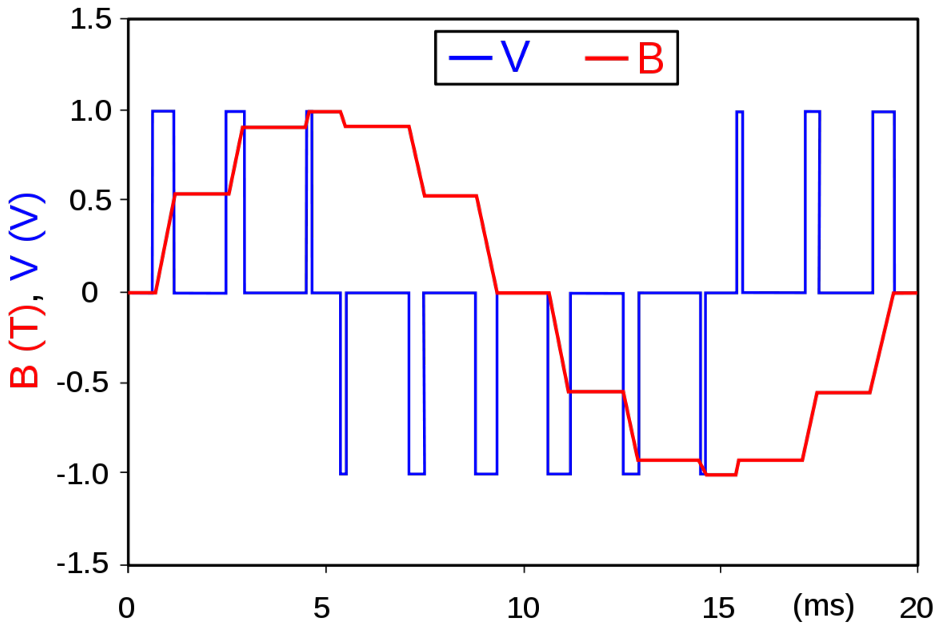

Figure 2.9: PWM example: the blue waveform is the PWM signal while the red one is the resulting waveform

The use of PWM in this system is related to the concept of average voltage. When the PWM generator sends a square wave, if the frequency of the signal is high enough, the H-Bridge does not detect a digital signal but an analog voltage whose value is directly proportional to the duty cycle of the digital signal : the greater the duty cycle, the greater the analog voltage. This is displayed in figure 2.10.

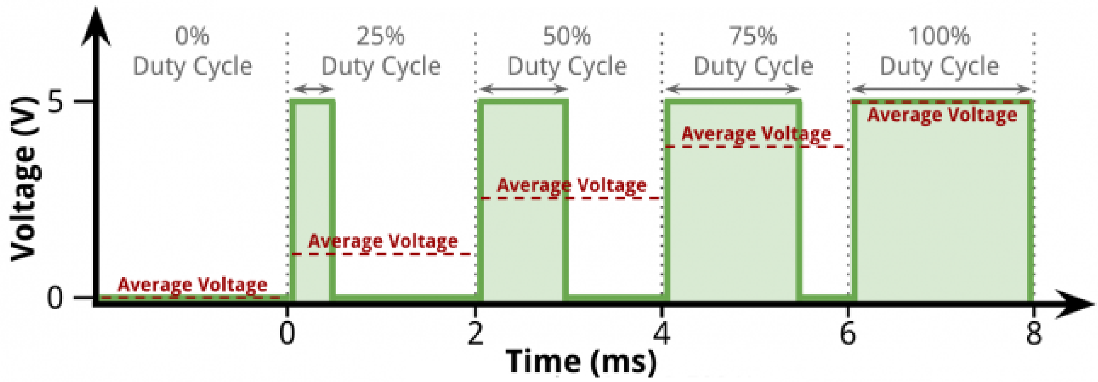

Figure 2.10: Average voltage detected in input by the H-Bridge depends on the duty cycle of the PWM signal: the greater the duty cycle, the greater the average voltage.

How do the PWM signal drives the H-Bridge? The PWM generator receives in input the *control signal* from the controller and transform it into a square wave : the duty cycle of the square wave is proportional to the control signal. In other words, the greater the value of the control signal, the greater the duty cycle of the PWM signal ; a greater value of duty cycle means also a greater average value of the voltage (i.e. the power) transmitted to the H-bridge and so a bigger current flowing into the TEC.

An example of real world PWM signal is displayed in figures 2.11 and 2.12. These two waveforms were obtained and during the testing of the system (described in detail in section 7) respectively in two different time instants: it is clear to see the difference between the two, since the signal in figure 2.11 has a greater duty cycle with respect to the signal in figure 2.12.

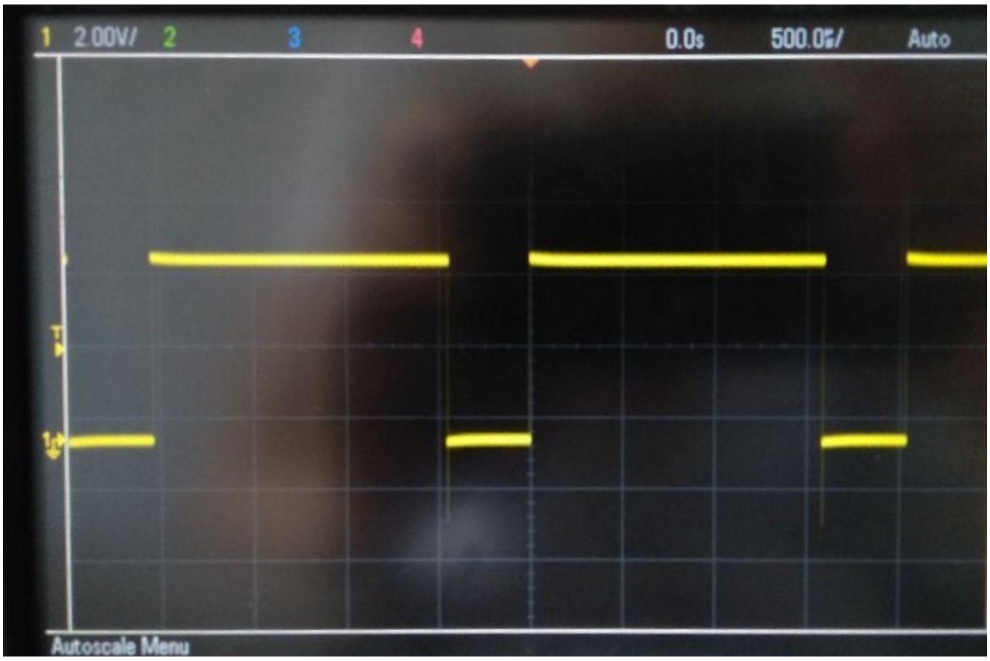

Figure 2.11: PWM control signal with high duty cycle measured with an oscilloscope during the testing of the system

.

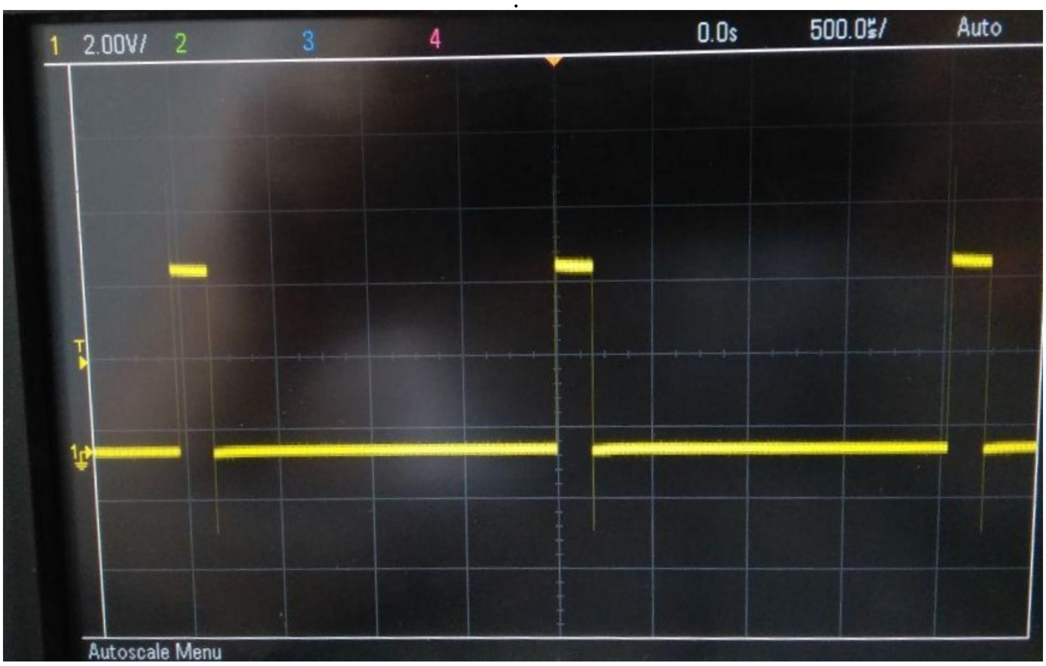

Figure 2.12: PWM control signal with low duty cycle measured with an oscilloscope during the testing of the system

.

Now all the elements that make the Actuator have been described: **PWM generator**, **H-Bridge** and **TEC**. The correct order of connection, together with the input control signal and the output actuator signal, is displayed in figure 2.13.

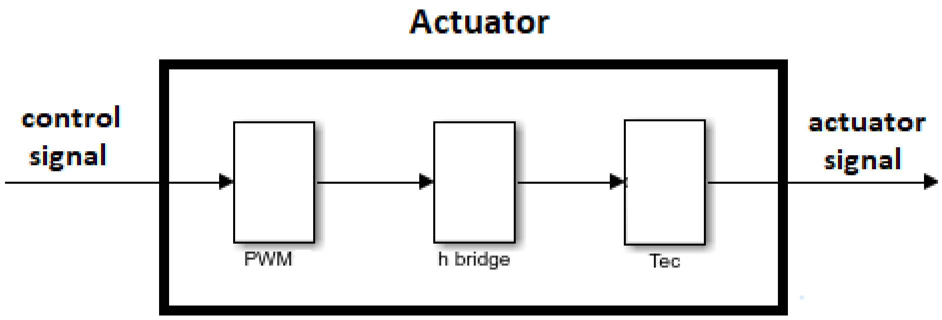

Figure 2.13: Actuator block with its sub-blocks: *PWM generator*, *H-Bridge* and *TEC*

.

2.5 Plant

The **plant** in control theory is the physical system under test, i.e the system from which we are sampling the output variable that we are interested in. The plant is driven by the *actuator signal* coming from the Actuator block , and its output is the variable observed, which in our case is the *temperature*. A plant is often described mathematically with its transfer function (obtained with Laplace transform) which gives the relation between input and output signal of the system basing on its physical properties.

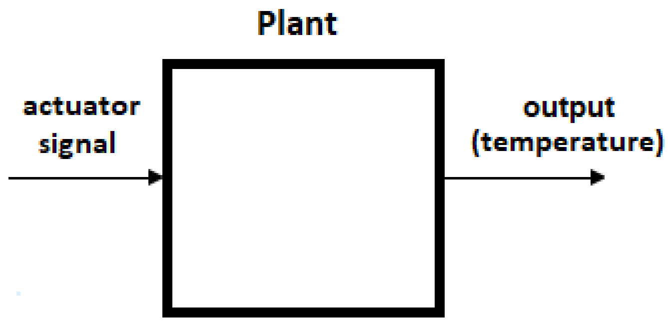

Figure 2.14: Plant block with its input and output signals: *actuator signal* and *output* (i.e. the *temperature*)

.

In the temperature control system designed the plant is the Petri dish containing

the microfluidic device, which form the heating chamber of the temperature control system : the heat exchange between the chamber and the environment is managed by the TECs, and the temperature is sampled and fed back to the input of the system through the feedback loop.
In order to perform simulations of the system, a mathematical description (i.e. the transfer fucntion) of the plant has been made, as described in section 3.2.2

2.6 Sensor

The final block of our control system is the sensor: its role is to convert the *output signal* (i.e. the temperature) in a electrical signal , the *feedback signal*, which is fed back to the input of the system where it is subtracted from the reference input value in order to obtain the *error signal*.

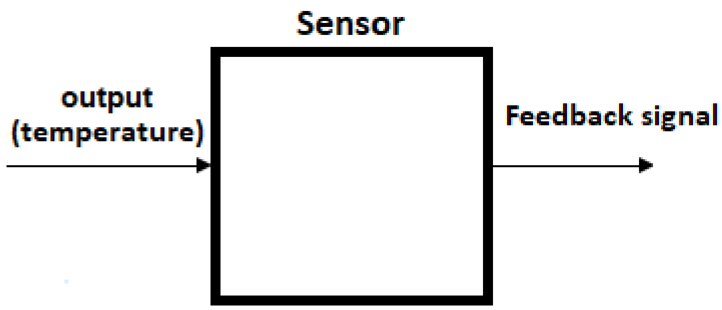

Figure 2.15: Sensor block with its input and output signals: *output* and *feedback signal* (i.e. the *temperature*)

.

The sensor in this system is obviously a temperature sensor: it is capable to detect the temperature of the microfluidic chamber and generate an electric signal proportional to the sensed temperature. All the decisions related to the choice of the sensor are described in section 4.1.4.

Now all the blocks have been described: the complete closed-loop control system is shown in figure 2.16.

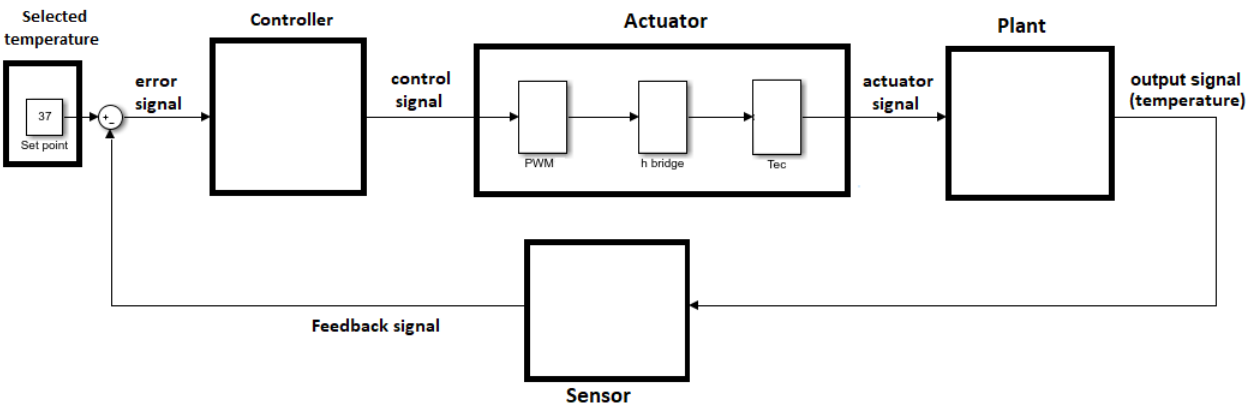

Figure 2.16: Block scheme of the temperature control system

Chapter 3

Mathematical Model

3.1 Motivations and goals of the mathematical model

Build a correct mathematical model is the first main step for the design of the overall system. The main goal of this mathematical model is to simulate the functioning of the temperature control system with precision and accuracy as close as possible to the real ones. This aspect is fundamental in the design process since by simulating the system is possible to choose the correct hardware to be employed in the real system; this was especially useful since at the beginning the hardware needed for the system was not available in the Microfluidic Laboratory of the University, and it had to be purchased. But since the shipment for some components can take from up to three months and my stay in Budapest was of five months, I had to be sure that the purchased components were the correct ones for the control system in terms of performances: i.e. the best way to choose the correct components was to run the simulations with the parameters of different components and see the best ones that could be suited for the application.
The model has been implemented using the software MATLAB with its tools Simulink and Simscape.

3.2 Steps for the building of the system

Based on the reasons explained in the previous section, the design flow followed for the building of the system followed few but important steps, as shown in the following flowchart.

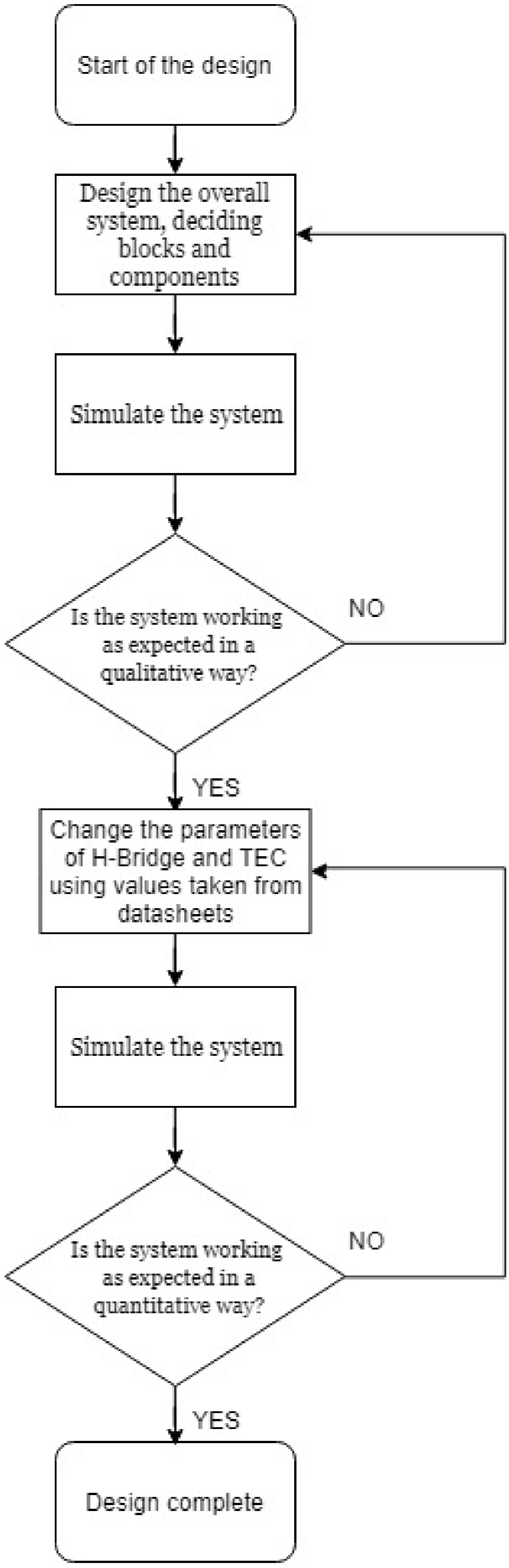

Figure 3.1: Design flow of the mathematical model

As exploited in the flowchart, there are two main steps of the design:

- design of the overall system, i.e. building all the blocks needed and connecting them properly so that the system behaves as a temperature controller;
- after the design is complete , several simulations are performed using different models of H-bridges and TECs until the best ones are selected for the purchase.

In the next sections, these two steps will be analyzed in detail.

3.2.1 Overall system

The original idea of the system is a closed-loop control system, which consists The picture 3.2 display the system developed in Simulink and Simscape environment, with the addition of some highlights which refer to the standard closed-loop block .

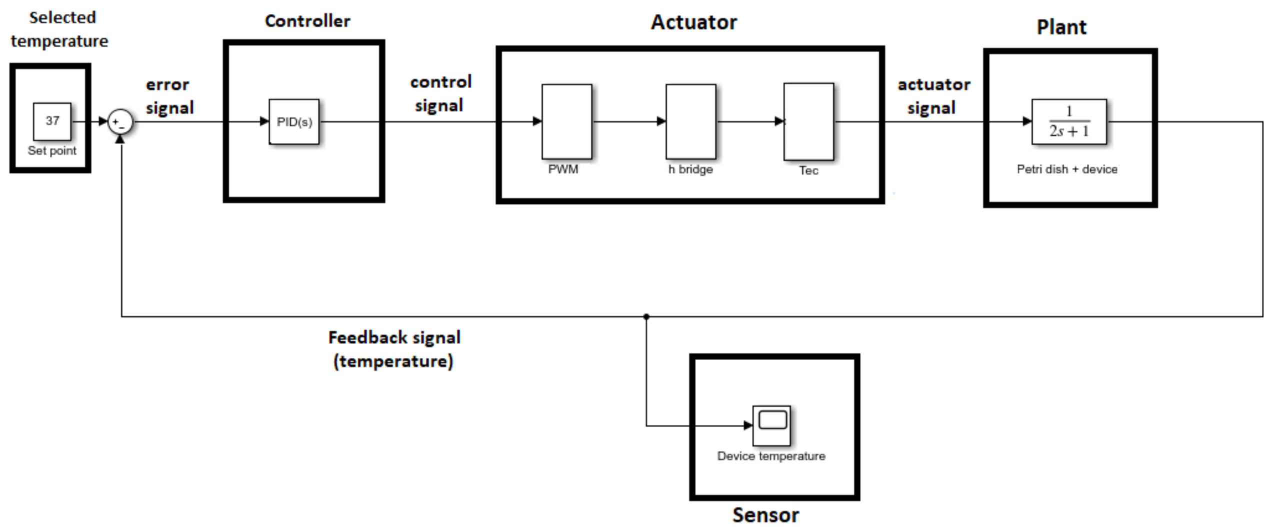

Figure 3.2: Mathematical model of the system

3.2.2 Simulink Blocks

In this section every block is analyzed in detail in terms of its Simulink implementation, internal working principles and choice of parameters.
Some blocks are already available in the Simulink environment and employed, while some others are not used since they are not compatible with the other blocks, so they have been built using other Simulink objects as basic blocks.

PID controller

The *PID controller* block is already implemented in Simulink so there is no need to built it from scratches (for the *Actuator* and *Plant* blocks it's the opposite, as it will be described in the next sections).
As it can be seen in figure 3.3 there are many design choices available in the PID block provided by Simulink, such as output saturation, tracking mode, data type and many others, but in this case only the main function of the block is going to be used, i.e. the tuning of the controller parameters: K_i, K_p and K_d.

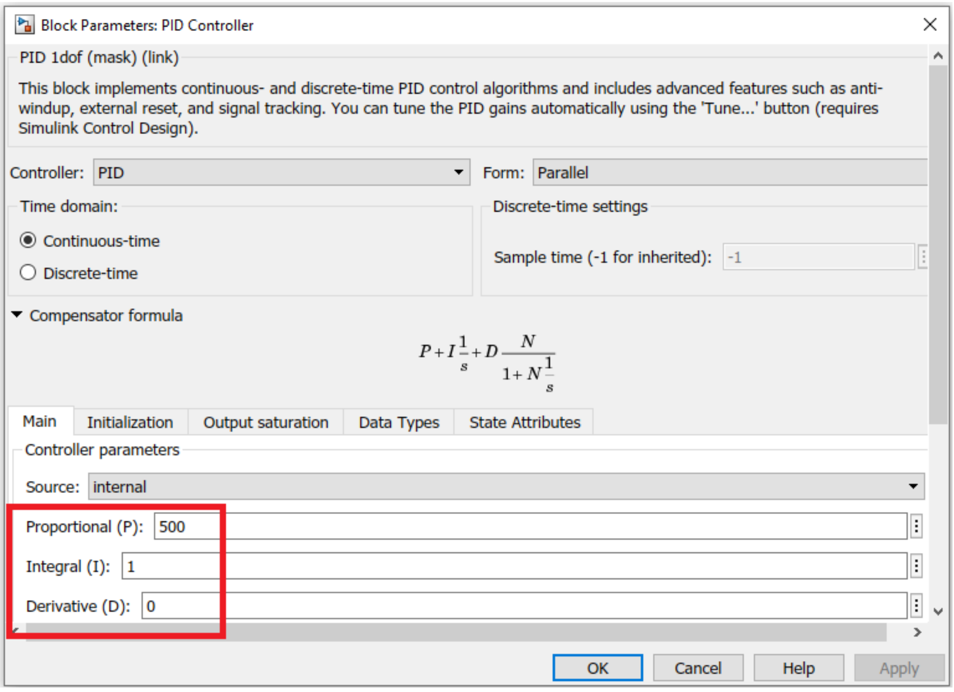

Figure 3.3: Simulink block of the PID controller: the tuning of the parameters can be done in the section highlighted in red

The tuning of the PID controller is described in section 3.3

PWM generator

The *PWM generator* block has been built from scratches using basic blocks from the Simulink library, since a "PWM generator " block is not available in the library. The basic idea of the PWM generator is a pulse generator which changes the duration of the pulse (the duty cycle) with respect to the analog input, that here is the control signal coming from the PID controller. The block is built following some previous works [22][29]and adapting them to this particular system, and it is shown in figure 3.4.

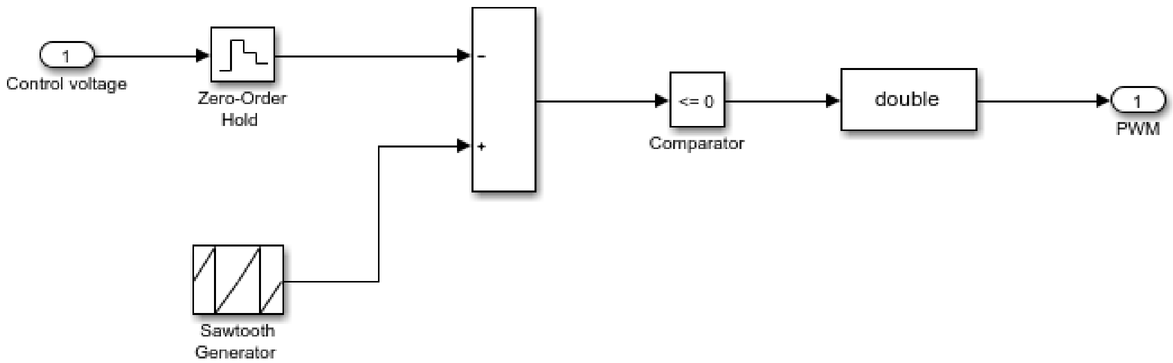

Figure 3.4: Simulink block of the PWM generator

The starting idea is that the analog control signal is compared trough a *comparator* to a sawtooth wave: depending if the control signal is higher or lower than the sawtooth wave, the output will be high or low. In this way a voltage-controlled square wave generator with variable duty cycle is implemented. The frequency of the square wave is the same as the frequency of the sawtooth wave.
The model is modified with two blocks: the *Zero-Order Hold* keeps the analog control signal constant for a short period of time (1 ms in this case), in order to model the sampling time of the hardware that will be chosen [29].
The block named *double* in figure 3.4 is a *Data Type Conversion* block which convert the output signal (the square wave) from boolean type to analog type, this is necessary for coherence with the *H-Bridge* block which requires an analog signal as input, but has no meaning in the real-world system.
The final result is a square wave at the output with duty cycle that changes according to the control voltage and compatible with the other Simulink blocks, as expected.

H-Bridge

The H-Bridge is driven by the PWM signal and provides bidirectional current to the load, which is the TEC in this system; the amplitude of the current depends on the duty cycle of the PWM signal.
In order to implement the *H-Bridge* block, also the tool box **Simscape** has been used. It is a tool box in Simulink's library which is necessary for the simulation of physical systems, allowing a bigger variety of signals and blocks to be used in Simulink simulations.
The basic schematic of an H-Bridge is shown in figure 2.8. However some additional parameters must be introduced for a complete modelization of the H-Bridge:

- $\boldsymbol{L_1}$, $\boldsymbol{L_2}$, $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$: the parasitic inductances and capacitances values that are needed to model the frequency behaviour;
- $\boldsymbol{R_L}$: the load resistance, here the load is the TEC so its value is taken from the datasheet of the chosen TEC;
- $\boldsymbol{V_d}$: the supply voltage.
- $\boldsymbol{I_{LB}}$: the boundary current, i.e. the maximum value of current that can be provided by the H-Bridge.

The values of L_1, L_2, C_1 and C_2 can be found in the datasheet of the chosen H-Bridge; for this system the H-Bridge chosen is the *L298N* as described in section 3.4 and the relative values can be found in table 3.1. R_L is the internal resistance of the TEC and is taken from the datasheet of the TEC. The value of V_d can be chosen taking into account the minimum values required by a typical TEC, which is never less than 12 V [30] for a correct working range. Since in this design step the chosen TEC is still unknown, a reasonable value of **24 V** is chosen.
Instead, I_{LB} is selected taking into account the maximum current which can be provided by the power supply available in the Microfluidic Laboratory, which is **1,05 A**

V_d	I_{LB}	R_L	L_1	L_2	C_1	C_2
24 V	1,05 A	1 Ω	150 μL	150 μL	15 μF	15 μF

Table 3.1: Values for the mathematical modelization of the H-Bridge

Now that all the values are available, it is possible to build the Simulink model of the H-Bridge, which is shown in figure 3.5. The blue elements and connections are the part of the circuit built with Simscape.

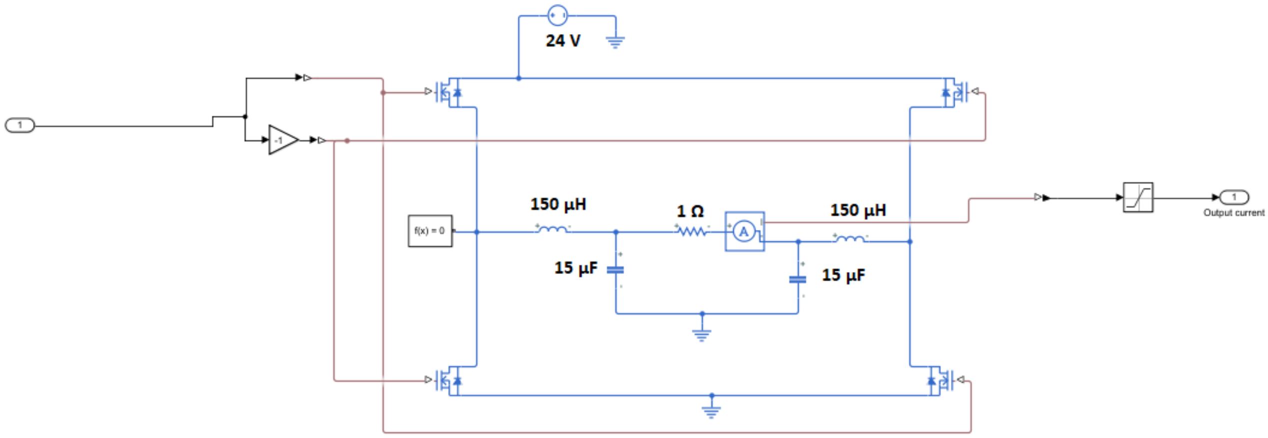

Figure 3.5: Simulink block of the H-Bridge, implemented using Simscape

TEC

TEC working principle is based on Peltier effect and it is described in section 2.4.1. This is the most complex block to implement in Simulink, since the block has as input the PWM current signal coming from the H-Bridge and must convert it into a heat flux which causes a change in the temperature of the plant: this means that the equations that describe the TEC will involve at least the parameters regarding current, heat, and temperature.
The equations regarding the **heat** can be written by taking into account 3 contributions:

- first of all the contribution related to the **Peltier effect**, the basic principle of the TEC: the greater the current flowing, the greater the heat flux between the plates of the Peltier cell, thus the greater the temperature difference between the plates. This behaviour is simply described by:

$$Q = \pi I \tag{3.1}$$

  where Q is the heat transferred, π is the Peltier coefficient and I is the current flowing in the TEC. The Peltier coefficient measures the heat pumping capability of the cell: the greater this coefficient, the greater the heat flux between the two surfaces of the TEC.
  This relation can also be written exploiting the *Seebeck coefficient*, which is related to the Peltier coefficient π by:

$$\pi = ST \tag{3.2}$$

  where S is the Seebeck coefficient (expressed VK^{-1}) and T is the temperature of the device. The Seebeck coefficient can be defined as:

$$S = \frac{V_{\mathrm{max}}}{T_{\mathrm{h}}} \tag{3.3}$$

Where V_{max} is the *maximum voltage* admitted by the TEC and can be found in the datasheet of the Peltier device.
The result is that $\boldsymbol{Q_{pa}}$ and $\boldsymbol{Q_{pe}}$, which are respectively the heat emitted from the cold surface and the heat absorbed by the hot surface are given by:

$$Q_{\mathrm{pa}} = SIT_{\mathrm{c}} \tag{3.4}$$

$$Q_{\mathrm{pe}} = SIT_{\mathrm{h}} \tag{3.5}$$

where T_{c} is the temperature of the cold plate and T_{h} is the temperature of the hot plate.

- the second contribution is related to the **thermal conductivity** of the material. It influences the heat transfer with a term $\boldsymbol{Q_t}$ which is given by:

$$Q_{\mathrm{t}} = K_{\mathrm{th}} \Delta T \tag{3.6}$$

Where ΔT is the temperature difference between the hot and cold plate

$$\Delta T = T_{\mathrm{h}} - T_{\mathrm{c}} \tag{3.7}$$

and K_{th} is the *thermal conductivity* of the material and can be found using the relation:

$$K_{\mathrm{th}} = \frac{V_{\mathrm{max}} I_{\mathrm{max}} (T_{\mathrm{h}} - \Delta T_{\mathrm{max}})}{2 T_{\mathrm{h}} \Delta T_{\mathrm{max}}} \tag{3.8}$$

where the parameters V_{max}, I_{max} and ΔV_{max} are respectively the *maximum voltage*, *maximum current* and *maximum temperature difference* admitted by the TEC and can be found on the datasheet of any Peltier device.

- the third and last contribution is related to the **Joule effect**, which is heat production due to an electric current flowing in a conductor.
$\boldsymbol{Q_j}$ is the heat produced by Joule heating and is given by:

$$Q_{\mathrm{j}} = I^2 R \tag{3.9}$$

Where I is the current flowing in the TEC and R is the resistance of the TEC device, which, as K_{th}, can be found as a function of the basic parameters V_{max},

$I_{\max}$ and $\Delta T_{\max}$:

$$R = \frac{V_{\max}(T_{\mathrm{h}} - \Delta T_{\max})}{I_{\max} T_{\mathrm{h}}} \tag{3.10}$$

The sum of these three contributions just described gives as a result the equations that define $\boldsymbol{Q_c}$ and $\boldsymbol{Q_h}$, respectively the *heat absorbed from the cold surface* and the *heat emitted from the hot surface*:

$$Q_{\mathrm{c}} = SIT_{\mathrm{c}} - \frac{I^2 R}{2} - K_{\mathrm{th}} \Delta T \tag{3.11}$$

$$Q_{\mathrm{h}} = SIT_{\mathrm{h}} + \frac{I^2 R}{2} - K_{\mathrm{th}} \Delta T \tag{3.12}$$

From equation 3.11, it is clear that in order to have a high (efficient) heat remove from the cold surface, the value of S should be high while the value of K_{th} should be low.

Since the typical application field of temperature controllers for microfluidic devices involves typically organisms which must be kept at 37 °C , our TEC is used mainly for heating and not for cooling, so it can be assumed that the hot surface heats the microfluidic device while the cold surface temperature is always kept constant using an heat sink. The equation 3.11 can be manipulated in order to write the *cold side temperature* $\boldsymbol{T_c}$ as a function of the other variables:

$$T_{\mathrm{c}} = \frac{0.5 I^2 R + K_{\mathrm{th}} T_{\mathrm{h}} + Q_{\mathrm{c}}}{SI + K_{\mathrm{th}}} \tag{3.13}$$

With this , we have all the equations which are necessary for a mathematical description of the TEC block in Simulink, that is implemented with a simple *function* Simulink block, in which the variable is the current I flowing in the TEC from the H-Bridge while the parameters R and K_{th} are calculated in a separate MATLAB script using the values of $V_{\max}$, $I_{\max}$ and $\Delta T_{\max}$ in the equations 3.8 and 3.10.

The values of $V_{\max}$, $I_{\max}$ and $\Delta T_{\max}$ and Q_c are taken from the datasheet of the Peltier cell.

As an example, the Peltier cell *TES1-12704* has this values taken from the datasheet

Q_{max}	34,0 W
Δ T_{max}	66 ° C
I_{max}	3,3 A
V_{max}	14,0 V

Table 3.2: Values of the peltier module *TES1-12704*

The Simulink representation of the TEC is simple since there is the *function* block that receives the current I as input and gives in output the temperature of the hot side, which is subtracted from the cold side temperature in order to obtain the *temperature difference* which can be monitored with a *scope*: the hot side temperature is then sent as an output to the plant.

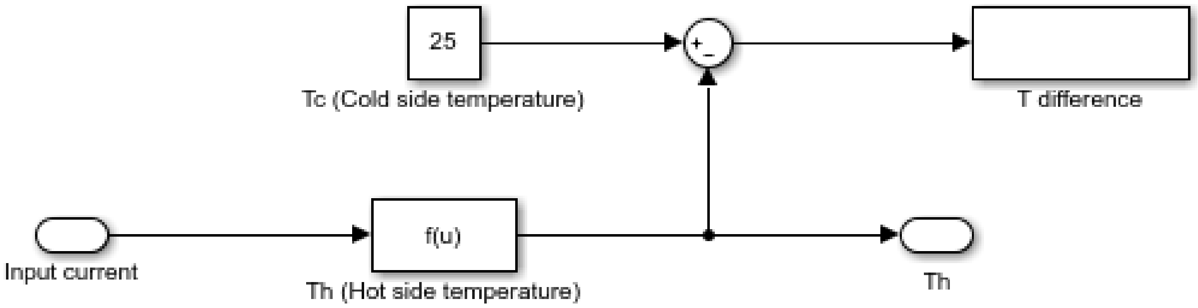

Figure 3.6: Simulink block of the Thermoelectric Cooler

Set point, Plant, Temperature sensor

The last three blocks are the **Set point**, the **Plant** and the **Temperature sensor**. They will be briefly described since they are quite simple: however, some aspects have to be highlighted for a better comprehension of the system.

- **Set point** is the temperature selected by the user, in Celsius degree: in the real word the user can select the desired temperature with a dedicated interface, that will be described later. The block is already described in section 2.2 and further explanations would be unnecessary. In Simulink the block is a simple *constant value* block.

- the **Plant** is the physical system which has to be heated, in this case the Petri dish containing the microfluidic chip under observation. Since when the mathematical model was being built there were not heating elements in the Microfluidic Laboratory since they still had to be chosen and purchased, it was not possible to experimentally obtain the transfer function of the plant through the most common methods and so a lot of literature has been reviewed in order to find a good transfer function for this plant; in many papers and research works the plant of temperature control system is usually described with a generic first order transfer function, and this approach is followed also in this work.

- the **Temperature sensor** is not considered as a block in the mathematical model: since the output of the plant is already the current temperature of the microfluidic devices, in the mathematical description there is no need to add another block for the sensor since it is like it is already included in the plant description. It is important to consider that this design choice implies that the real device for the temperature sensing will not be chosen among the ones which provide an analog signal, since in this case the signal would need to be transduced from an analog to a digital signal trough a DAC converter , which would not benefit the system both in terms of complexity and costs; instead, a simpler approach with a digital temperature sensor is chosen. The last aspect to highlight is that in this mathematical model the uncertainty of the sensor is not taken into account.

3.3 Tuning the PID controller

The method used to tuner the PID controller is the Ziegler–Nichols method. It is a popular heuristic method which consists in different steps:

- first of all, the derivative and integral gains $\boldsymbol{K_d}$ and $\boldsymbol{K_i}$ are set to zero. Also the proportional gain $\boldsymbol{K_p}$ is set initially to zero, and then it is increased until it reaches a value for which the output of the control loop has stable and consistent oscillations.
- This value of K_p is called ***ultimate gain*** $\boldsymbol{K_u}$, while the period of the oscillation is called T_u. For this Simulink model it was found that $K_u = 130$; the output waveform for $K_p = K_u = 130$ is shown in picture 3.7. The value of T_u can be extracted from the same figure, and it is found to be $T_u = 19{,}90$ s.
- from the values of K_u and T_u it is possible to obtain the parameters $\boldsymbol{K_p}$, $\boldsymbol{K_i}$, $\boldsymbol{K_d}$, $\boldsymbol{T_i}$, $\boldsymbol{T_d}$ of the PID controller using the values provided in the table 3.3.

Control type	$\mathbf{K_p}$	$\mathbf{T_i}$	$\mathbf{T_d}$	$\mathbf{K_i}$	$\mathbf{K_d}$
classic PID	$0{,}6\ \mathrm{K_u}$	$\mathrm{T_u}/2$	$\mathrm{T_u}/\ 8$	$1{,}2\ \mathrm{K_u}/\ \mathrm{T_u}$	$3\ \mathrm{K_u}\ \mathrm{T_u}/40$
some overshoot	$\mathrm{K_u}/3$	$\mathrm{T_u}/2$	$\mathrm{T_u}/\ 3$	$0{,}666\ \mathrm{K_u}/\ \mathrm{T_u}$	$\mathrm{K_u}\ \mathrm{T_u}/9$
minimize overshoot	$\mathrm{K_u}/5$	$\mathrm{T_u}/2$	$\mathrm{T_u}/\ 3$	$0{,}4\ \mathrm{K_u}/\ \mathrm{T_u}$	$\mathrm{K_u}\ \mathrm{T_u}/15$

Table 3.3: Values of the Ziegler-Nichols tuning method

For this application of temperature control for biological purposes, the main parameter that has to be monitored and minimized is the **overshoot**, which in this case is the occurrence of the temperature exceeding its target. In other words, in this application it is extremely important to prevent the temperature to exceed significantly the target temperature since this could cause irreversible damage to the biological material confined in the microfluidic device. For this reason, the PID parameters are chosen so that the maximum overshoot is limited as much as possible, even if it means to extend the time needed for the system to reach the desired temperature. In other words, the parameters

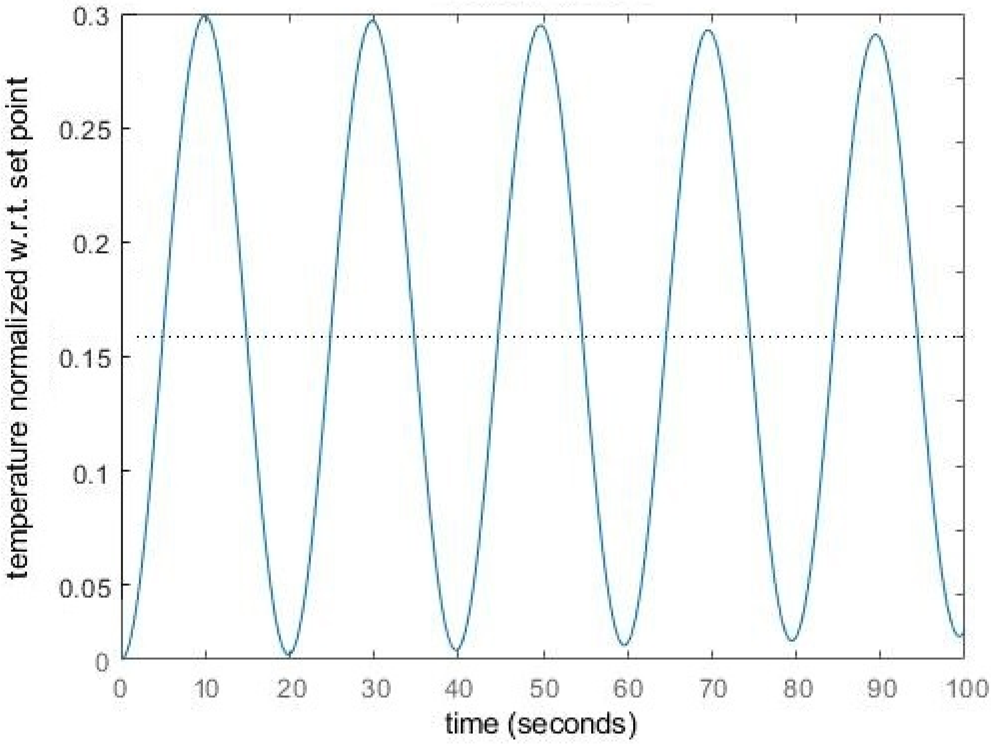

Figure 3.7: Output of the Simulink model for with $K_p = K_u$

are evaluated following the last line of table 3.3 as following (the values are rounded up to the second decimal digit):

$$K_{\text{p}} = K_{\text{u}}/5 = 26 \tag{3.14}$$

$$T_{\text{i}} = K_{\text{u}}/2 = 9,95s \tag{3.15}$$

$$T_{\text{d}} = K_{\text{u}}/3 = 6,63s \tag{3.16}$$

$$K_{\text{i}} = 0.4 \cdot K_{\text{u}}/T_{\text{u}} = 2,61 \tag{3.17}$$

$$K_{\text{i}} = K_{\text{u}} \cdot T_{\text{u}}/15 = 1,72 \tag{3.18}$$

3.4 Simulation results

The simulations were performed in two steps, according to the design flow described in section 3.2.
First of all, the Simulink models was built. In order to perform a qualitative simulation and evaluate if the model was working in a proper way, test simulations were performed using the values of a standard H-Bridge and a standard Peltier cell. The variables of the H-Bridge L_1, L_2, C_1, C_2 and R, while the variables of the Peltier cell are V_{max}, I_{max}, ΔT_{max} and Q_c. For this initial quantitative analysis,,this values were taken from a standard *H-Bridge* and *TEC*, namely the *SN754410* and *ETC-200-14-06E*.
Simulation results are provided in figure 3.8.

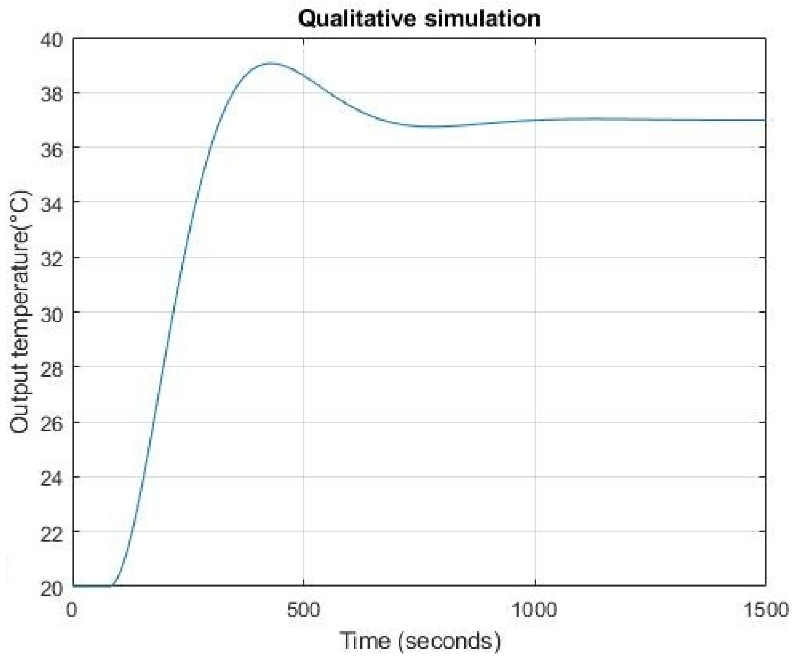

Figure 3.8: Simulation of the Simulink model, with an input temperature of 37 °C

The system is working as expected: the temperature control system is increasing the temperature of the plant from the initial temperature (20 °C as the typical laboratory temperature) up to the desired temperature (37 °C in this case), and it is trying to maintain the temperature equal to it, with a significative overshoot and almost negligible oscillations. Anyway, in this part of the design these two latter aspects are not relevant, since the goal here is to check if the model is working as expected from a qualitative point of view.
The model is working properly, so now it is time for the second step of the design: in order to choose the correct components to purchase (H-Bridge and TEC), many simulations were performed using different values of L_1, L_2, C_1, C_2, R, V_{max}, I_{max}, ΔT_{max} and Q_c corresponding to different devices.

Besides, the choice of the component was performed following two guidelines:

- **performances**: the components with the better behaviour in terms of *rise time*, *maximum overshoot* and *ringing*;
- **costs**: a trade-off between performances and costs has been taken into account in the choice;

Following some relevant guidelines in the available literature [31] [32] [33] three choices were taken into account. Their respective results are provided in figures 3.9, 3.10, 3.11 and a brief comment about the comparison is provided here:

- *Simulation 1* is performed with the **cheapest** devices available, the H-Bridge is a *TC648* and the Peltier cell is a *TEC1-12706*. As expected from this really cheap setup, the performances are the worst between the three Simulations.
- *Simulation 2* is performed with an approach of **trade-off** between costs and performances, using the H-Bridge *L298N* and the Peltier cell *TES1-12704*.
- *Simulation 3* is performed choosing good quality devices in terms of **performances**: the H-Bridge chosen is a *ifx9201* and the TEC is a *ETC-200-14-06E*. The simulations results are the best in terms of rise time and accuracy, but the cost is way higher than the other two cases.

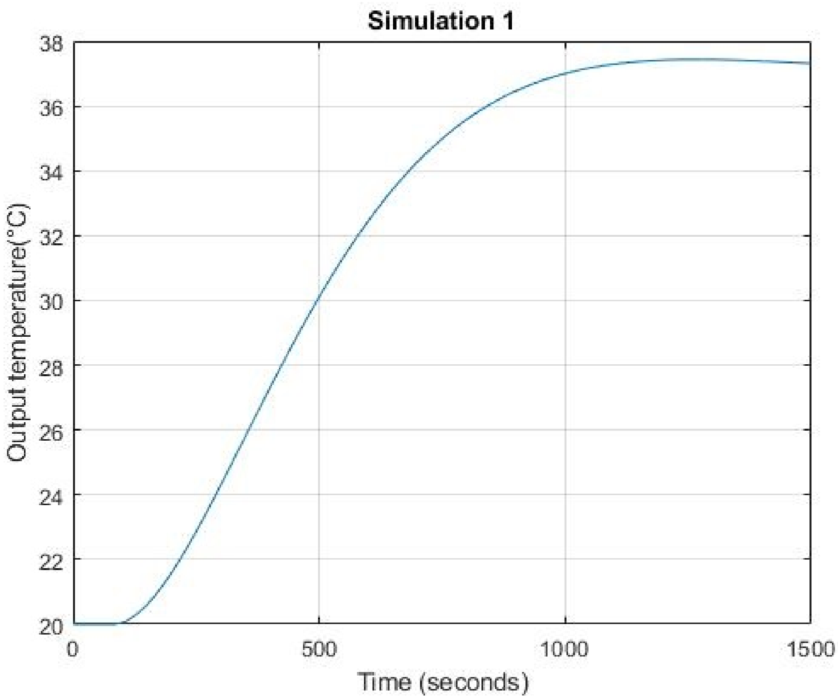

Figure 3.9: Simulation 1

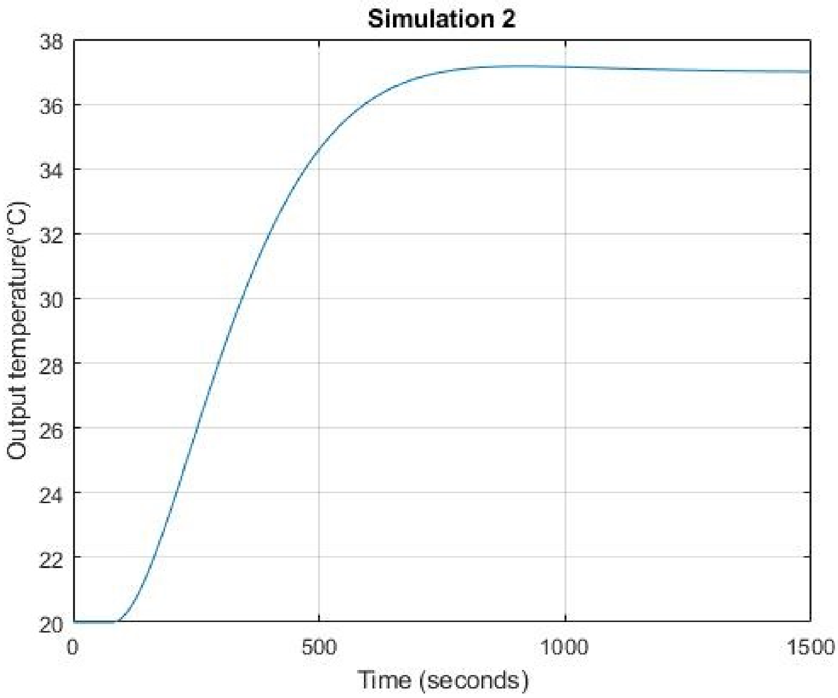

Figure 3.10: Simulation 2

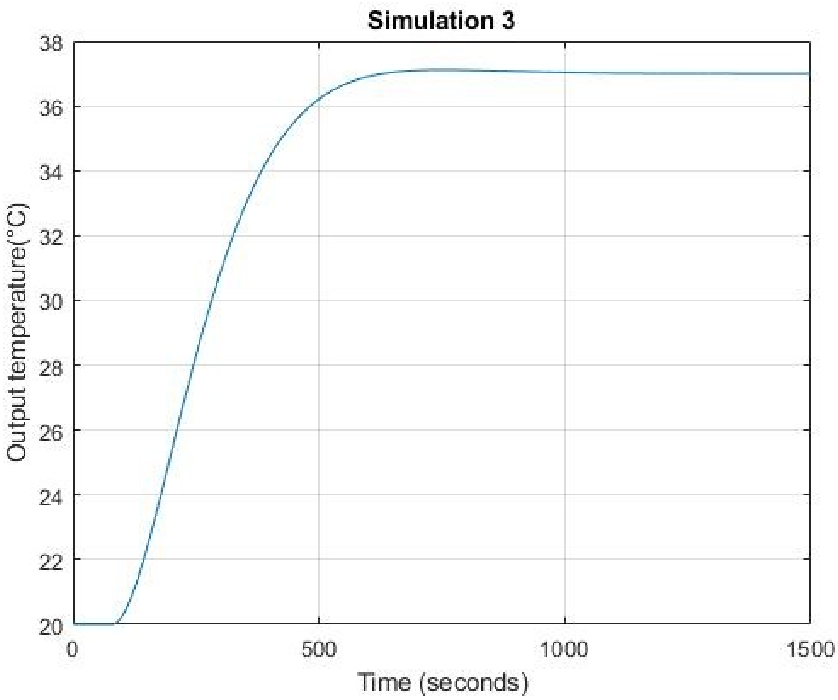

Figure 3.11: Simulation 3

The better results in terms of performances were given by the most expensive devices (*Simulation 3*), while the cheaper ones (*Simulation 1*) had worse results: a trade-off approach between costs and performances was chosen (*Simulation 2*). In table 3.4 are provided the significative results for the three different simulations:

Simulation	Rise time	Overshoot	Bias error	Total cost (estimate)
Simulation 1	16 min 34 s	0,80° C	0,62 °	3,5 €
Simulation 2	12 min 29 s	0,36° C	0,29 °	9 €
Simulation 3	9 min 50 s	0,32° C	0,24°	120 €

Table 3.4: Results for three different simulations of the Simulink model

At the end, the chosen components are the H-Bridge **L298N** and the Peltier cell **TES1-12704**. The reason of this choice is that, with respect to the most expensive case (*Simulation 3*), even though the rise time is consistently longer 26,9% more), the decrease in temperature accuracy is small (12,5% for the maximum overshoot and 20,1% for the bias error) and it is not worth to spend thirteen times more for the budget of this components for a so small improvement . As a matter of fact, for the application of this temperature control system, i.e. the study of parasites in a microfluidic environment, the rise time is not so important, while the accuracy is of vital importance especially because an oscillation too large in temperature could lead to a damage to the organisms in the device. On the other hand, the most cheap setup (*Simulation 1*) performs significantly worse (the rise time, maximum overshoot and bias error increase respectively of 68,5%, 150,0% and 158,3% with respect to *Simulation 3*) so it is not worth considering.

In this way, the mathematical model has performed its aim: provide essential help in the design of the system and the choice of the components. Now, the next step to build the system is the purchase of these chosen devices and of the rest of the components (temperature sensor, hardware for the user interface, heatsinks,etc.).

Chapter 4

Hardware

In this chapter all the design choices regarding the hardware are explained.
Thanks to the successful simulation of the Simulink model, it was possible to choose the H-Bridge and the Peltier device, as explained in section 3.4. But all the others components were still to choose.
The starting idea was to build the system using a **Raspberry PI 3 B** board: it is s single-board computer, with an operative system based on a Linux kernel. The Raspberry was supposed to manage the interface through a *Graphical User Interface (GUI)*, in which the user could set the temperature and time from the GUI using a keyboard and a monitor connected to the Raspberry, and the board would have manage the correct functioning of the automatic controlled system via software written in *Python* code, and driving the hardware components sending the correct control signals.

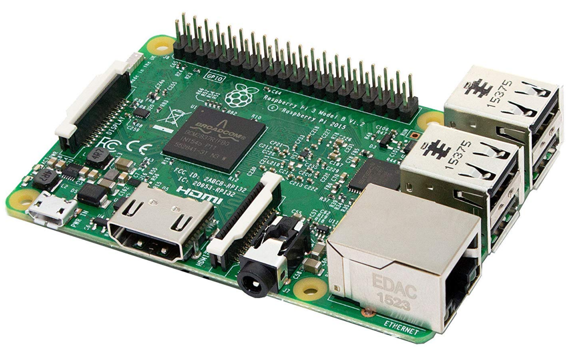

Figure 4.1: Raspberry 3 model B board

However the use of the Raspberry was discarded because of two main reasons:

- the us of a Raspberry board would have been an ”overkill” for this application: the board would have need to be connected (at least) to a monitor and a keyboard, and many tools available on the board would have not been used.

- the system was meant to be designed to be portable and easy to use, so a more stand-alone approach was preferred.

For this reasons, the final choice for the system was an **Arduino Uno** board.

4.1 Components List

4.1.1 Arduino

Arduino is an open-source electronics platform, which consists of both a physical programmable circuit board (microcontroller) and a piece of software (Integrated Development Environment, IDE). The reasons for which the Arduino board has been chosen instead of the Raspberry is because it fits better into the project: it is possible to work without being connected to a PC (given that the code is previously uploaded on the board from a PC through an USB cable), which makes it more portable and flexible for the usage; furthermore, it is capable of drive all the pieces of hardware that are needed for the temperature control system.

Figure 4.2: Arduino Uno board

Specifically, the Arduino board in this project is responsible of many tasks:

- it receives from the user interface (which will be composed of a LCD screen and two pushbuttons) the desired temperature selected from the user and the desired amount of time for which the temperature has to be regulated;

- it implements, via software, the *PID controller*;
- it generates the *PWM signals* which drive the H-Bridge;
- it reads the data sent from the *temperature sensor*, i.e. the current temperature of the microfluidic device.

The functions managed by the Arduino board are explained more in detail in section 5.

4.1.2 H-Bridge

All the design considerations regarding the H-Bridge have been explained in detail in section 3.4.
The chosen H-Bridge is the **L298N**: it is a dual H-Bridge controller, designed mainly for driving DC motors but perfectly fit for driving two Peltier cells at the same time. It is cheap, compact, and reliable thanks to the overvoltage protections.

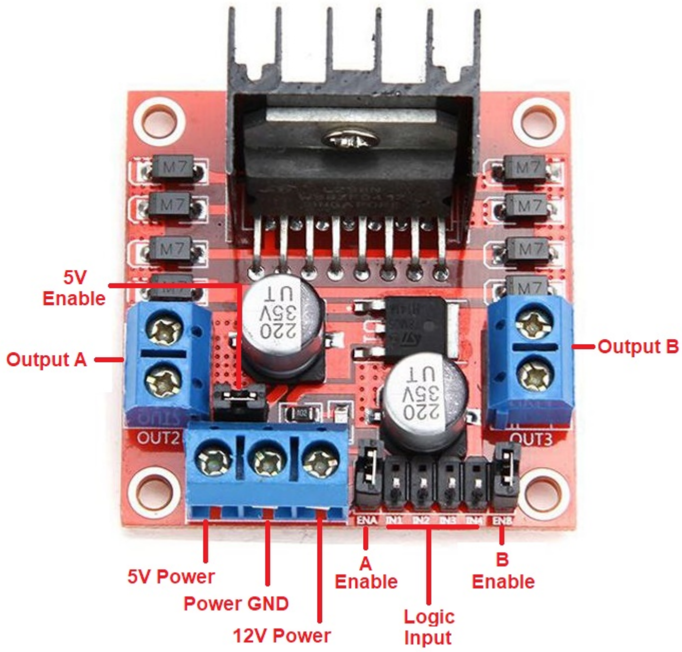

Figure 4.3: The *L298N* H-Bridge

As explained in the previous chapters, the H-Bridge is driven by the Arduino board with a PWM signal, which is sent to the *Enable A* and *Enable B* from the *digital pin* **9** of Arduino. The four pins *Logic input* set the flowing direction of the current depending on their values, and are driven by the *digital pins* **7** and **8** of Arduino. Each one of the pins *Output A* and *Output B* is connected to a Peltier cell.

Figure 4.4: The power supply *Keithley 2410*

In this project, *l298N* is powered by the power supply **Keithley 2410**, which is the one available in the Microfluidic Laboratory, through the pin *Power GND* and *12V power*.

4.1.3 Thermoelectric cooler (TEC)

The chosen Peltier devices are the **TES1-12704**, as explained in section 3.4. They are 30 mm x 30 mm x 3,2 mm Peltier cells, made of ceramic, with two power supply cables of 15 cm each.

The Peltier plates are places below the Petri dish containing the microfluidic device, in this way the heat can be diffused in an optimal way; in order to hold the Peltier cells in the correct position, a customized 3D-printed structure has been designed and printed, as explained in section 6.1.
The typical area of a Peltier cell is 40 mm x 40 mm, but for this project have been used cells of size 30 mm x 30 mm, because the first ones could not fit in the space below the Petri dish on the reverse microscope of the laboratory.
In order to improve an uniform diffusion of the temperature, two *TES1-12704* are used.
So in this configuration the top plate of the Peltier cell is heating / cooling (depending on the selected temperature) while the bottom plate has the opposite behaviour; besides, in order to dissipate the heat/cold on the bottom surface, aluminium fins are used.

Figure 4.5: The Peltier cell *TES1*

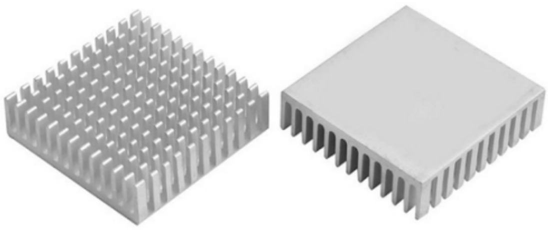

Figure 4.6: The aluminium heat sink used

As with the Peltier devices, the typical surface of the Aluminium fins is 40 mm x 40 mm. Fins of this size have been purchased and then, in order to obtain fins with area of 30 mm x 30 mm, they have been shaped with a metal saw.

4.1.4 Temperature sensor

It exists a huge variety of temperature sensors on the market, which differs for temperature range, accuracy, price, application and so on.
After the decision of using Arduino as the "brain" of the system, the temperature sensor has been chosen to among the ones that are designed to be used with Arduino.
Furthermore, some sensors like the *thermocouples* require additional hardware to work, thus increasing the complexity and cost of the project.
The chosen temperature sensor is the **DS18B20**: it is cheap, fairly precise and it is perfectly suited for the integration with the Arduino board since it requires only one digital pin for the bidirectional communication with the microcontroller, besides the analog output voltage of the Arduino pins is enough to power it without need of

external power supplies. The complete specifications of the sensor are in table 4.1

Figure 4.7: Th temperature sensor *DS18B20*

Power supply	3 V to 5,5 V
Current consumption	1 mA
Temperature range	-55 °C to 125 °C
Accuracy	± 0,1 °C
Resolution	9 to 12 bit (selectable)

Table 4.1: Specifications of the *DS18B20* temperature sensor

The sensor has three pins: V_{cc}, *Data* and *GND*. The pins V_{cc} and *GND* are connected, respectively, to the 5 V pin and the ground of the Arduino, for the powering of the sensor. The pin *Data* is connected to the *digital pin* **10** of Arduino, and is also connected to the 5 V pin through a $4,7k\Omega$ resistor as recommended in its datasheet.

4.1.5 User friendly interface: LCD display and pushbuttons

The user interface must be capable of allowing the user to select the temperature of the microfluidic device and the desired amount of time for which the temperature must be kept constant.
One option for the implementation of the interface is through a GUI implemented via software and accessible from a PC; however to employ this solution means that every time that the system has to be used, the Arduino need to be connected to a PC, which makes the controller less flexible and portable.
So the decision was to create a customized interface for the project, consisting into a LCD screen and two pushbuttons, that can allow the user to select the parameters: both the LCD and the pushbuttons are powered and communicate directly with the Arduino board so there is no need to connect the board to a PC. The working

principle of the interface is explained in section 5.2

Figure 4.8: Pushbutton

Each **pushbutton** has two pins: one pin is connected to the 5 V pin of the Arduino, while the other one is connected to ground through a 10 kΩ pull-down resistor. When the pushbutton is pressed, an *high* signal is sent to the corresponding *digital pin* on the Arduino. The dedicated pins on the board for the two pushbuttons are the *digital pins* **12** and **13**.

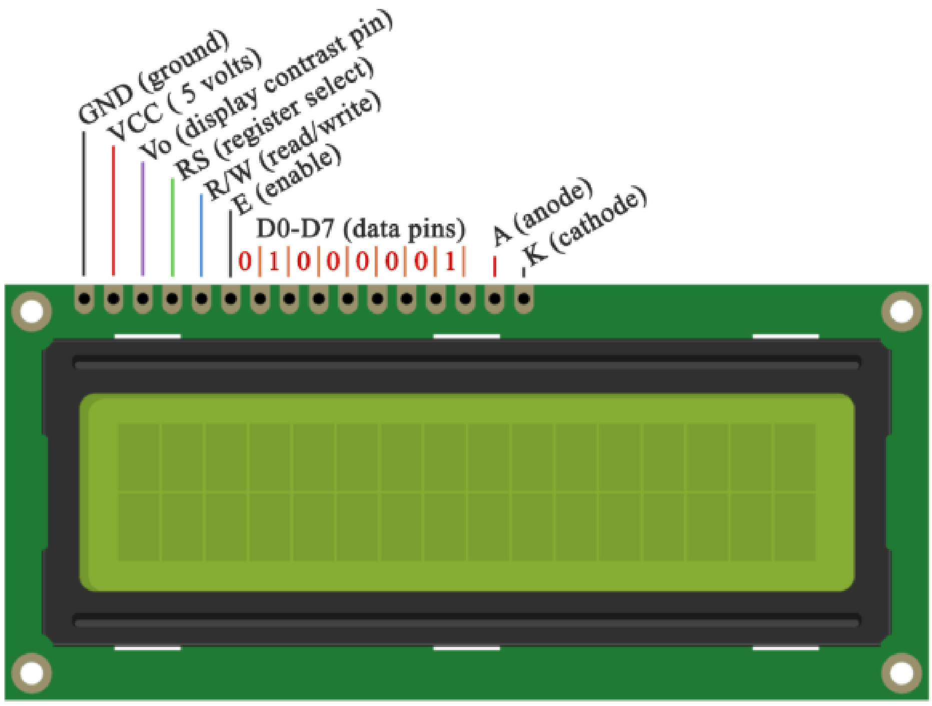

Figure 4.9: The *LCD 1602*

The **LCD** screen has 16 pins. The first one is V_{ss}, which is the ground pin, connected to the ground of Arduino. The second pin is V_{dd} which is connected to the 5 V pin of the board. Next is V_0, this pin is connected to a 10 kΩ potentiometer for the control of the contrast of the display. Next one is the *RS* pin: it is set *low* when data have to be sent to the screen, while it is set *high* when commands have to be sent, like turn on, turn off or set the cursor in a specific point of the display.It is connected to the *digital pin* **1** of Arduino. The pin $R\backslash W$ is set high or low depending if the data of LCD must be written or read: in this case, is connected to ground. Next pin is the enable pin *E*, which enables the writing to the registers of the LCD: it is driven by the *digital pin* **2** of Arduino. The 8 data pins from *D0* to *D7* send the data that have to be displayed in ASCII code; 8-bit or 4-bit configuration can be used: in this system 4-bit is used, thus the pins *D0, D1, D2, D3* are not connected while the pins *D4, D5, D6, D7* are connected to the Arduino *digital pins* **3, 4, 5 , 6**. The last two pins *A* and *K* are the anode and cathode and are connected respectively to the 5 V pin and to ground, since they are needed for the LED back light.

4.2 Assembling of the system

The final schematic of the temperature control system is shown in figure 4.11.
In order to integrate all the hardware in a compact way, a **stripboard** has been used. A pictures of the system completed and assembled is shown in figure 6.13 together with the 3D-printed case which contains all the hardware.

Figure 4.10: The *Weller WSD81* soldering set

The soldering has been performed with a **Weller WSD81** provided by the Microfluidic Laboratory, shown in fig 4.10

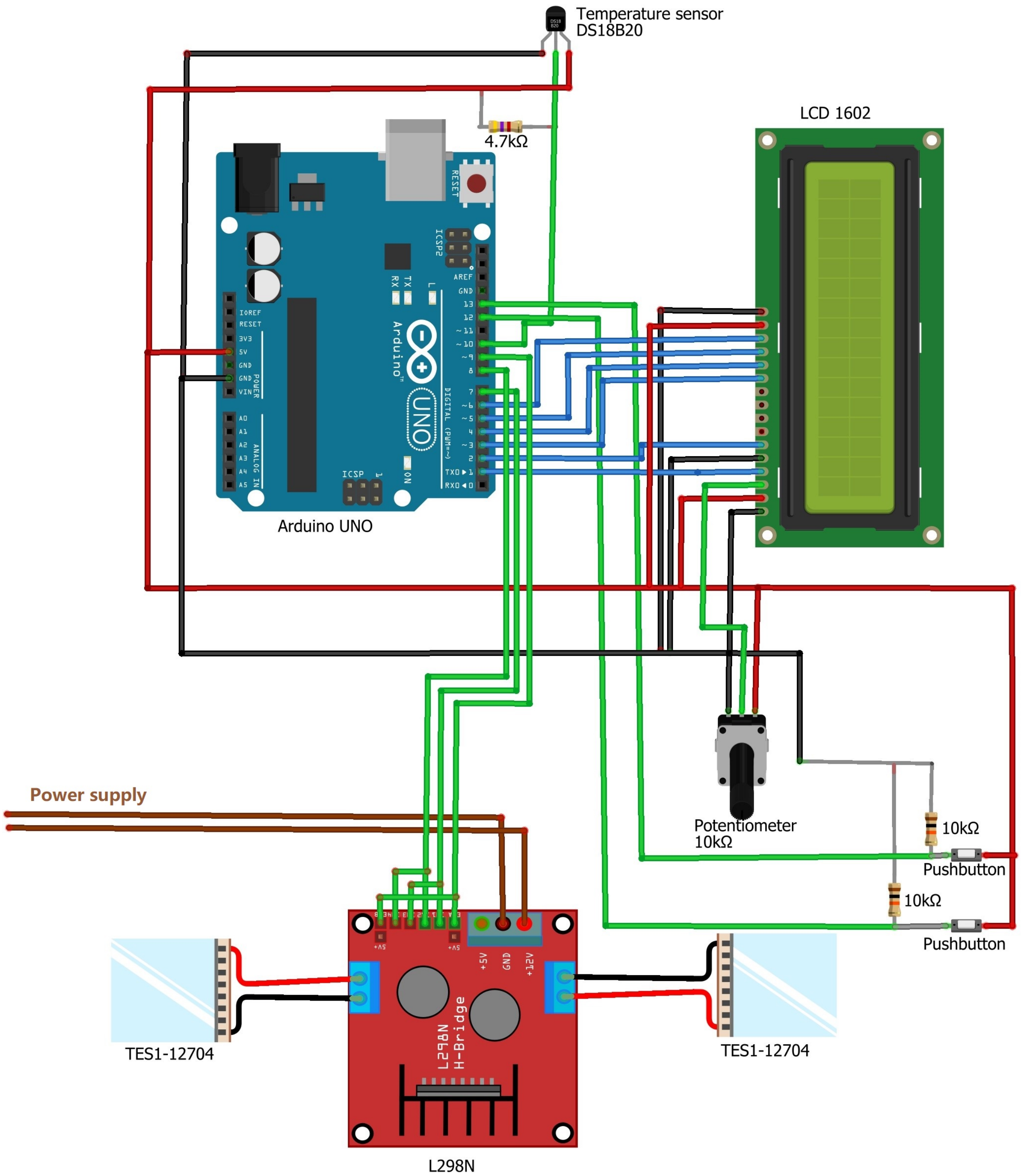

Figure 4.11: Schematic of the complete system: in *blue* the connections between Arduino and the display, in *green* all the other general connections. In *red* the connections to the 5 V pin of Arduino while in *black* the ones to the Ground pin of Arduino. Finally, in *brown* the connections with the external power supply.

Chapter 5

Software

The software is written in **C++** code and implemented using the **Arduino IDE**, which is an open-source software that is optimized to write code and implement it on the board. The environment is written in Java and based on Processing and other open sources software.
The code is uploaded to the Arduino through a USB cable; it is sufficient to upload it only one time because the code remains saved in the non-volatile memory of the board. This feature allows the system to be more portable and flexible, since once the code has been uploaded, it will be necessary to connect the Arduino only to its power supply and the system will work in a stand-alone way.
The main task of the code is to control the Arduino board which will drive the other components; specifically, the main functions of the software are:

- manage the user interface, ensuring communication between Arduino, the LCD display and the pushbuttons;
- implement the PID controller, which is managed entirely via software
- read the temperature from the DS18B20 sensor and acting accordingly;
- generate the PWM signal to control properly the H-Bridge and thus the current in the heating elements.

The following sections explain in details the code.

5.1 Initialization

The first step to do is to declare the **libraries** that are going to be used in the code:

```
#include LiquidCrystal.h
#include OneWire.h
#include DallasTemperature.h
#include PID_v1.h
```

These four libraries have different purposes:

- *LiquidCrystal* is needed to drive correctly the LCD display, it contains a lot of functions that are different than a simple "writing on the screen" and will be useful later;
- *OneWire* contains function for the management of One-Wire communication protocol, which in this project is used by the temperature sensor DS18B20;
- *DallasTemperature* is also needed to drive the sensor DS18B20;
- *PID_v1* is a library which allows to implement an effective and precise PID control via software.

The next step is to **define the pin** of Arduino board, associating to each variable the number of the pin: in this way, if the pin number needs to be changed, it has to be done only one time in those lines and not multiple times in the main code. These lines define the pins which drive the H-Bridge, namely the pin **7** for the enable signal and pin **8** and **9** for the input signals.

```
#define en1 7
#define in1 8
#define in2 9
```

Instead, the following line defines the pin for the communication with the temperature sensor, pin **13**:

```
#define ONE_WIRE_BUS 13
```

While the following two lines declare that the pin **10** and **11** will be used to communicate withe the two pushbuttons: *buttonPin10* is related to the button which increases temperature_time, while *buttonPinOk* is related to the button for the confirmation of the selection.

```
#define buttonPin 10
#define buttonPinOK 11
```

Next step is to **initialize the hardware**: the temperature sensor and the LCD are initialized using dedicated functions of their libraries. In particular, the command *LiquidCrystal lcd(1, 2, 3, 4, 5, 6)* creates an LCD object and the Arduino pins **1, 2, 3, 4, 5, 6** are associated to the LCD pins *RS, E, D4, D5, D6, D7.*

```
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
LiquidCrystal lcd(1, 2, 3, 4, 5, 6);
```

Next step is to **define the variables related to the closed-loop system**: *Celsius* is the temperature (in Celsius degree) detected by the sensor, *out_perc* is the duty cycle of the PWM signal, *Setpoint* is the desired temperature selected by the user, while *Input* and *Output* will be used later for the implementation of the PID controller.

```
double Celsius, out_perc, Setpoint, Input, Output;
```

After that, it is the turn of the **PID controller initialization**, which is performed by declaring the variables *Kp, Ki, Kd* and then associating the variables *Input, Output, Setpoint, Kp, Ki, Kd* to the correspondent signals of the virtual PID controller through the function *myPID* taken from the PID library:

```
double Kp=26, Ki=2.61, Kd=1.72;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
```

The values of the PID constants K_p, K_i, K_d are the same evaluated in section 3.3. The reason why these values obtained from the mathematical model are used instead of evaluating new constants for the real system is related to some practical issues appeared during the tuning process with the real system. During the first step of the Ziegler-Nichols method, in which the derivative and integral gains K_d and K_i are set to zero while the proportional gain K_p is increased until stable and consistent oscillations are reached, the temperature oscillations are so high in magnitude that they overcome the melting point of the plastic material of the Petri dish containing the microfluidic device, causing its melting. For this reason the PID tuning using the real system was impracticable and the values obtained from the mathematical model were used instead.

Next step is the **definition of the variables related to the user interface**: *temp_in* is the temperature which is displayed at the beginning on the LCD, can be increased through the related pushbutton and it is initialized to 20 Celsius degrees;

time_selected is the time (in minutes) displayed on the LCD, can be chosen by increasing it with the related pushbutton and is initialized to 0 minutes; *temp_selected* and *time_selected* are flag variables used to point if the temperature and the time have been selected by the user; *buttonState* and *buttonStateOK* are flag variables used to detect if the two pushbuttons have been pressed, they are initialized with low (0) values and assume high values (1) when the associated buttons are pressed; *time_min* and *ds_counter* are used for the decreasing time counting operation needed to calculate how much time is left for the system to work, according to the time set by the user; *finished* is a flag variable used to detect when the decreasing time counting operation reaches the zero, i.e. when the system has to stop.

```
int temp_in = 20, temp_selected = 0, time_hours = 0,
    time_selected = 0, buttonState = 0, buttonStateOK = 0,
    time_min = 0, finished = 0, ds_counter=0;
```

Once this first initialization has been performed, the function *setup* embedded in the Arduino IDE is used.
First of all, the pins **7**, **8** and **9** related to the H-Bridge (the enable and the input pins) are initialized as output pins. Besides, one of the two inputs is set *high* while the other is set *low*: in this way the initial current flow direction, which determines the initial heat flow direction, is set.

```
void setup() {

  //H BRIDGE
  pinMode(en1, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);

  // Set initial heat flow direction ;
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
```

Next step is the turn of the temperature sensor, which is initialized and start to aquires data with the function:

```
  sensors.begin();
```

Next step is the initialization of the interface of the LCD screen with the function *lcd.begins(16,2)*; 16 and 2 refer to the width and height dimensions of the display:

```
lcd.begin(16,2);
```

The last initialization involves the buttons, which are set as input pins. The last curly bracket close the loop function *setup* opened previously:

```
pinMode(buttonPin, INPUT);
pinMode(buttonPinOK, INPUT);
```

5.2 Working code

The previous section described all the necessary initialization, while in this section the main part of the code is explained.
This second part of code is entirely contained in the function *loop*, i.e. it runs continuously until the Arduino board is powered down.

The first piece of code manage the initial part of the user interface, i.e. the **temperature selection**:

```
void loop() {

buttonStateOK = digitalRead(buttonPinOK);

  //TEMPERATURE SELECTION
  while (temp_selected == 0)
  {
  lcd.clear();
  lcd.print(Selected desired );
  lcd.setCursor(0,1);
  lcd.print(temperature );
  lcd.setCursor(13,1);
  lcd.print(temp_in);

   //read the state of the pushbutton value
  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {
     //increase temperature
    temp_in++;}
```

```
  if (temp_in == 50) {
    //reset temperature to 20
    temp_in=20;}

  buttonStateOK = digitalRead(buttonPinOK);
  if (buttonStateOK == HIGH) {
  temp_selected=1; }
```

When the system is powered on, the LCD screen displays the message *Select desired temperature: 20*, as shown in picture 5.1. The number *20* represent the temperature in Celsius degrees, and can be changed using the proper pushbutton.

Figure 5.1: Initial state of the system when it is powered on

Each time the green pushbutton is pressed, the temperature increases of one Celsius degree, until the limit temperature of 50 °C is reached and the counting starts from 20 °C. In the example shown in figure 5.2, the selected temperature is 37 °C in this case.

Once the desired temperature is selected, the black button has to be pressed to confirm.

The next piece of code is related to the **time selection**:

```
// TIME SELECTION
  while (time_selected==0 and temp_selected ==1){
     lcd.clear();
     lcd.print(Select desired);
```

Figure 5.2: Desired temperature selected

```
    lcd.setCursor(0,1);
    lcd.print(time (hours) );
    lcd.setCursor(14,1);
    lcd.print(time_hours);

     //read the state of the pushbutton value
    buttonState = digitalRead(buttonPin);

    if (buttonState == HIGH) {
    // increase min
    time_hours++; }

    if (time_hours == 99) {
    reset temperature to 99
    time_hours=0; }

    buttonStateOK = digitalRead(buttonPinOK);
    if (buttonStateOK == HIGH) {
    time_selected=1; }
    delay(200); }
```

This part of the interface works in the same way as the temperature selection, but here the time (in hours) is selected, and it can ranges from 0 to 99 hours. As before, the green button is used to increase the value and the black button is used to confirm. An example is shown in figure 5.3.

Figure 5.3: Desired time selected: in this example, 5 hours

Next, the **PID controller is turned on** and set on *automatic* mode. Therefore, the temperature selected by the user and stored in the variable *temp_in* is set as *Set point* of the PID controller and so of the overall closed-loop system:

```
//Set the desired temperature
Setpoint = temp_in;
//Turn the PID on
myPID.SetMode(AUTOMATIC);
//Adjust the values of PID
myPID.SetTunings(Kp, Ki, Kd);
```

From now on, the most important part of the code starts: in few words, this is the part in which the closed-loop control system begins to work and the desired temperature of the microfluidic device is reached and maintained constant.
Meanwhile, the screen display the current temperature of the device and the remaining time, in hours and minutes, as shown in picture 5.4 . When the countdown reaches zero, all the process can be started all over again by pressing any button.

This piece of code starts with a *if* and a *while* loops that employ the flag variables *temp_selected*, *time_selected* and *finished*. In few words, the meaning of this loops is to keep the temperature of the microfluidic constant as soon as the desired temperature and time have been selected, and until the countdown has finished.
First of all, the temperature of the microfluidic device is sampled and stored in the variable *Celsius*, which is subtracted from the *Set point* in order to obtain the error

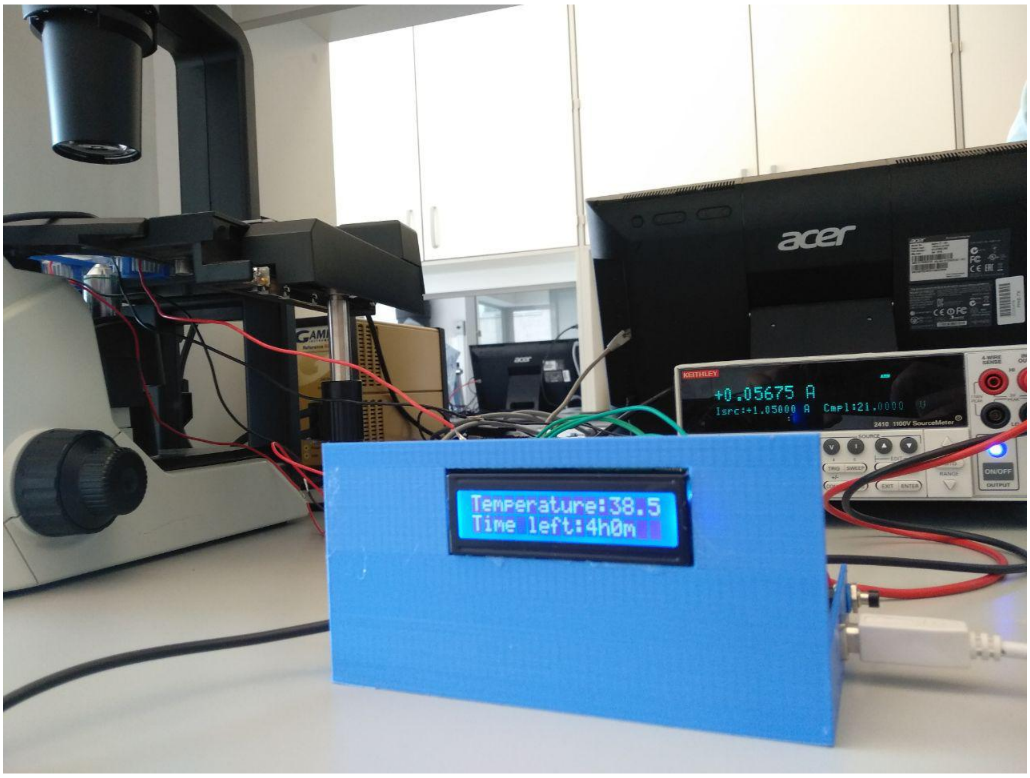

Figure 5.4: Display when the system is working, with the current temperature and the remaining time displayed

signal which is the *Input* of the PID controller and it is stored in the variable of the same name.
Next, the *Output* of the PID controller is computed: it is an analog value with value between 0 and 1023. As explained in the previous chapters, this output signal is called *control signal* and its value drives a PWM generator which generates a square wave with duty cycle proportional to this *control signal*. In this way we are implementing via software both the PID controller and the PWM generator. In order to convert this *control signal* into a PWM signal, the Arduino needs it to to be mapped into a value between 0 and 255: this is performed with the function *map()*. So the variable *Output*, whose value ranges from 0 to 1023, is linearly mapped into the variable *pwmOutput*, which ranges from 0 to 255.
The PWM signal is sent to the enables of the H-Bridge L298N through the pin **en**, which is the pin 7 of the Arduino board.

```
if (temp_selected == 1 and time_selected == 1) {

      while (finished == 0)
     {
       //SENSOR
```

```
        sensors.requestTemperatures();
        Celsius=sensors.getTempCByIndex(0);
        Input = Celsius;
        //PID calculation
        myPID.Compute();
         int pwmOutput = map(Output, 0, 1023, 0 , 255);  // Map the output PID value from 0 to 255
         analogWrite(en1,Output);} //PWM to L298N Enable pin
```

The next lines of codes, in which the function *Serial.print* is employed, are used for two reasons: data acquisition and debug. This function prints on the serial monitor of the Arduino IDE, which is visible only by PC, the current temperature and the duty cycle of the PWM signal, every time that the loop is repeated. This part of the code has been useful to collect data during the testing of the system. Since the system is designed to be used without PC, this section is not present in the final code but it is shown here for completeness.

```
         // Send data by serial for plotting
         Serial.print(INPUT );
         Serial.println(Input);

         out_perc = Output / 2.55;

         Serial.print(OUTPUT );
         Serial.print(out_perc);
         Serial.println( %);
```

The next part of code is responsible of printing the current temperature and remaining time on the LCD screen, until the countdown reaches zero.
First of all, the temperature and the remaining time at the current instant are displayed on the screen (as shown in figure 5.4) with the function *lcd.print*.
The following lines of code is responsible for the countdown. The *while loop* that contains this part of code is repeated every 100 ms, and each time the variable *ds_counter* is increased by one. So when the value of *ds_counter* reaches 599 it means that one minute is elapsed since 100 ms $\times$ 600 = 60 s = 1 min. In this way, every minute the countdown shown in the display goes down of one minute.
In a similar way, the variable *time_min* is used to perform the counting operation of the minutes elapsed: it starts from 59 and decrease each minute. When it reaches zero, also the variable *time_hours*, which represents the remaining hours, decreases by one.

Finally, when both *time_hours* and *time_min* reach zero, the counting stops and the flag variable *finished* changes value, allowing the code to exit the *while loop* and reach the final step.

```
        //displaying TEMPERATURE
        lcd.clear();
        lcd.print(Temperature);
        lcd.print(Celsius);
        lcd.print( C);

        //displaying TIME
        lcd.setCursor(0,1);
        lcd.print(Time left);
        lcd.print(time_hours);
        lcd.print(h);
        lcd.print(time_min);
        lcd.print(m);

        //wait one minute and decrease minutes
        if (ds_counter == 600)  {
          time_min--;
          ds_counter = 0;   }

        //when min reaches 0, decreases hours
        if(time_min  0) {
         time_min = 59;
         time_hours--;   }

        //when reaching 00h00m, STOP
       if(time_hours < 1 and time_min == 0)   {
          finished = 1;   }
      delay(100);   }
```

The last part of code manages the system when the desired time is elapsed and the countdown reaches zero. The closed-loop feedback system stops working and the display shows *"FINISH. Press to restart"*. If one of the pushbuttons is pressed, the system is reset, the variables are set again to the original values and the display shows again the first screen for the temperature selection, as in figure 5.1.

```
if (finished == 1)
 {
   while (temp_selected == 1 and time_selected == 1)
   {
   lcd.clear();
   lcd.print(FINISH);
   lcd.setCursor(0,1);
   lcd.print(Press to restart);

   buttonState = digitalRead(buttonPin);
   buttonStateOK = digitalRead(buttonPinOK);

   if (buttonState == HIGH or buttonStateOK == HIGH) {
     finished = 0;
     temp_selected=0;
     time_selected=0;
     lcd.clear();  }
   delay(100);  }
```

5.3 Complete code

For completeness, the full version of the code is reported here.

```
#include LiquidCrystal.h
#include OneWire.h
#include DallasTemperature.h
#include PID_v1.h

//H BRIDGE
#define en1 7 //Enable of H-Bridge
#define in1 8 //Input of H-Bridge
#define in2 9 //Input of H-Bridge

//SENSOR
#define ONE_WIRE_BUS 13   // sensor pin

//LCD
#define buttonPin 10    // increase pushbutton pin
```

```
#define buttonPinOK 11  // confirm pushbutton pin

//SENSOR
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

//LCD
LiquidCrystal lcd(1, 2, 3, 4, 5, 6); //Creates LCD object

double Celsius, out_perc, Setpoint, Input, Output

//PID parameters
double Kp=26, Ki=2.61, Kd=1.72;
//create PID instance
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

//Initialization
int temp_in = 20, temp_selected = 0;

int time_hours = 0, time_selected = 0;

int buttonState = 0, buttonStateOK = 0;

int time_min = 0, finished = 0;

int ds_counter=0;

void setup() {

  //H BRIDGE
  pinMode(en1, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);

  // Set initial heat flow direction ;
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);

  //SENSOR
```

```
  sensors.begin();

  //LCD
  lcd.begin(16,2);  // Initializes LCD interface

  //PUSHBUTTONS
  pinMode(buttonPin, INPUT);
  pinMode(buttonPinOK, INPUT);  }

void loop() {

  buttonStateOK = digitalRead(buttonPinOK);

  //TEMPERATURE SETTING
  while (temp_selected == 0)
  {
  lcd.clear();
  lcd.print(Selected desired );
  lcd.setCursor(0,1);
  lcd.print(temperature );
  lcd.setCursor(13,1);
  lcd.print(temp_in);

   //read the state of the pushbutton value
  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {
    // increase temperature
    temp_in++;  }

  if (temp_in == 50) {
    //reset temperature to 20
    temp_in=20;  }

  buttonStateOK = digitalRead(buttonPinOK);
  if (buttonStateOK == HIGH) {
  temp_selected=1;

  //for debug purpose, to avoid to skip the time selection
```

```
    delay(300);   }

  delay(200);
  }

  // TIME SETTING
  while (time_selected==0 and temp_selected ==1)
  {
     lcd.clear();
     lcd.print(Select desired);
     lcd.setCursor(0,1);
     lcd.print(time (hours) );
     lcd.setCursor(14,1);
     lcd.print(time_hours);

      //read the state of the pushbutton value
     buttonState = digitalRead(buttonPin);

     if (buttonState == HIGH) {
     // increase min
     time_hours++;   }

     if (time_hours == 99) {
     reset temperature to 99
     time_hours=0;   }

     buttonStateOK = digitalRead(buttonPinOK);
     if (buttonStateOK == HIGH) {
     time_selected=1;   }

     delay(200);
  }

 //PID
  //Set the desired temperature
  Setpoint = temp_in;
  //Turn the PID on
  myPID.SetMode(AUTOMATIC);
  //Adjust the values of PID
```

```
myPID.SetTunings(Kp, Ki, Kd);

if (temp_selected == 1 and time_selected == 1) {

    while (finished == 0) {

     //SENSOR
     sensors.requestTemperatures();
     Celsius=sensors.getTempCByIndex(0);

     //Read the value from the light sensor
     Input = Celsius;
     //PID calculation
     myPID.Compute();

      //Write the output as calculated by the PID function
      int pwmOutput = map(Output, 0, 1023, 0 , 255);  Map the output PID value from 0 to 255
      analogWrite(en1,Output);  Send PWM signal to L298N Enable pin

     // Send data by serial for plotting
      Serial.print(INPUT );
      Serial.println(Input);

      out_perc = Output2.55;

      Serial.print(OUTPUT );
      Serial.print(out_perc);
      Serial.println( %);

      //displaying TEMPERATURE
      lcd.clear();
      lcd.print(Temperature);
      lcd.print(Celsius);
      lcd.print( C);

      //displaying TIME
```

```
        lcd.setCursor(0,1);
        lcd.print(Time left);
        lcd.print(time_hours);
        lcd.print(h);
        lcd.print(time_min);
        lcd.print(m);

        //wait one minute and decrease minutes
        if (ds_counter == 600)  {
          time_min--;
          ds_counter = 0;   }

        //when min reaches 0, decreases hours
        if(time_min  0)  {
         time_min = 59;
         time_hours--;   }

        //when reaching 00h00m, end everything
        if(time_hours < 1 and time_min == 0) {
         finished = 1; }

       delay(100);
       }
     }
  if (finished == 1)
  {
    while (temp_selected == 1 and time_selected == 1)
    {
    lcd.clear();
    lcd.print(FINISH);
    lcd.setCursor(0,1);
    lcd.print(Press to restart);

    buttonState = digitalRead(buttonPin);
    buttonStateOK = digitalRead(buttonPinOK);

    if (buttonState == HIGH or buttonStateOK == HIGH) {
      finished = 0;
      temp_selected=0;
```

```
      time_selected=0;
      lcd.clear();  }
    delay(100);
    }
  }
}
```

Chapter 6

AutoCad design and 3D printing

At this point of the design, the system is almost complete. The last step to be done is to pack and integrate the hardware together, and the choice was to use some 3D-printed structures to hold everything together allowing the system to be more compact and robust.
The 3d-printed structures realized are two:

- the first structure has been named **Heating chamber** and it has to hold in the right positions the hardware on the inverted microscope (i.e. the TEC modules and the temperature sensor) together with the Petri dish containing the microfluidic device;
- the second structure is a **Case** for the rest of the hardware: it integrates together the Arduino board, the L298N H-Bridge, the pushbuttons, the screen and all the interconnections. It is basically the interface with which the user interacts with.

Both the structures have been first designed with the software **AutoCAD**, and the AutoCAD models have been printed with 3D printer available in the laboratories of the University. The next sections explain in detail all the design aspects that has been taken into account during the realization of the models.

6.1 Heating chamber

The design aspects taken into account in the realization of the **Heating chamber** are:

- it has to perfectly fit in the reverse microscope.
- it has to keep the TEC modules (i.e. the Peltier cells along with the aluminium heat sinks) in contact with the Petri dish, thus allowing the microfluidic device to be heated/cooled;
- it has to hold the temperature sensor *inside* the Petri dish and *in contact* with the microfluidic chip, so that the temperature can be sampled in the most precise way;

The inverted microscope used in the Microfluidic laboratory is a **Olympus CKX53** and it is shown in picture 6.1.

Figure 6.1: Inverted microscope Olympus CKX53

The main aspect to consider in the design is that the heating chamber has to fit perfectly in the microscope observation chamber: the two main limitations are the size of the "hole" in which the chamber has to fit and the space below which has to be enough for the microscope lenses. As a matter of fact, as shown in picture 6.2, the microscope has a system of four different rotating lenses, and enough space must be provided for these lenses to rotate without touching the structure.

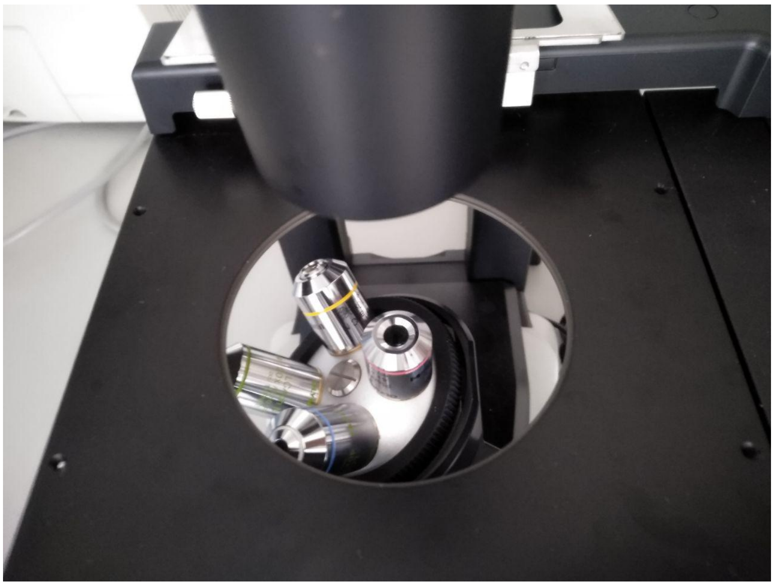

Figure 6.2: Close-up on the lenses of the microscope

The structure has been designed taking into account all these considerations: the figures 6.3 and 6.4 show, respectively, the AutoCAD model and its realization with the 3D Printer.

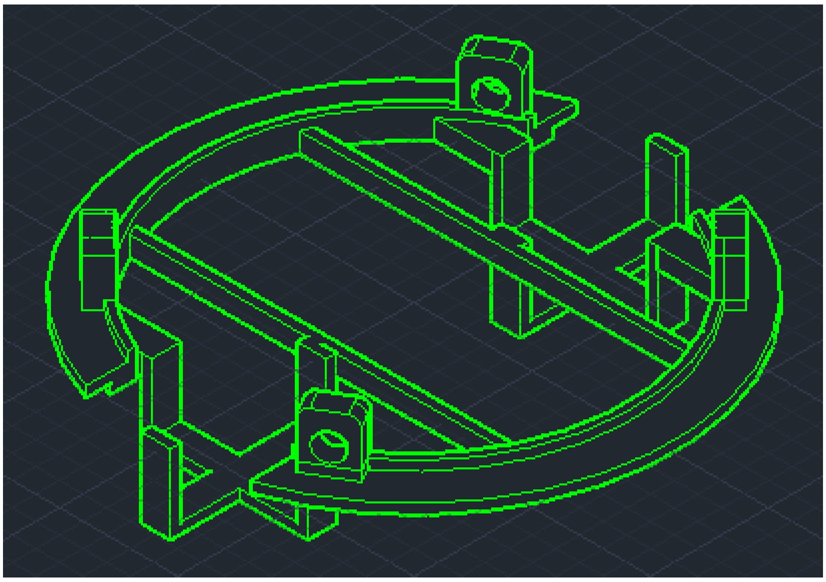

Figure 6.3: AutoCAD model of the *Heating chamber*

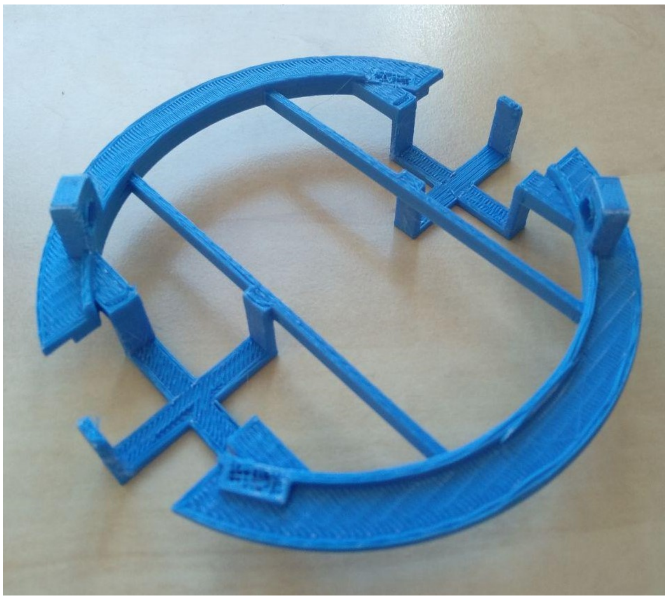

Figure 6.4: The *Heating chamber* obtained with 3D printing

The two cross-shaped structures are designed so that the TEC modules can perfectly fit inside, as shown in figure 6.5. The TEC modules are kept below and in contact with the Petri dish, as shown in figure 6.6. The same figure shows that the *Heating chamber* dimensions are correct and it fit in the observation chamber of the inverted microscope.

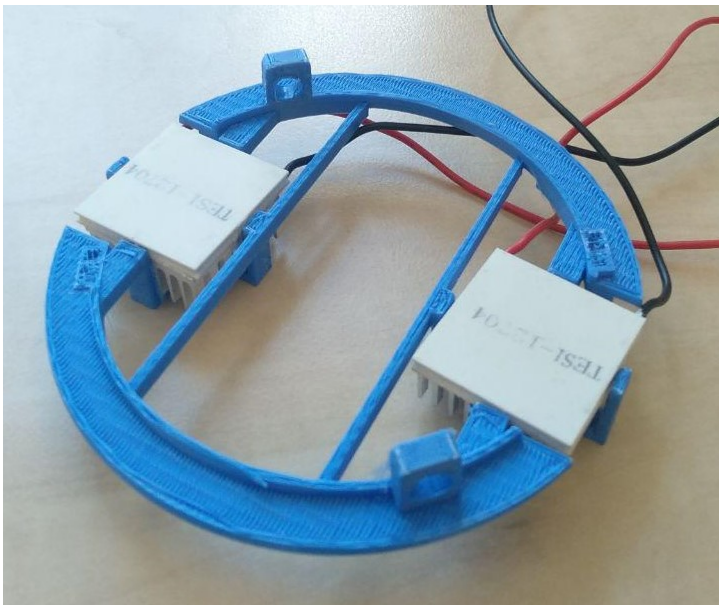

Figure 6.5: The *Heating chamber* with the TEC modules

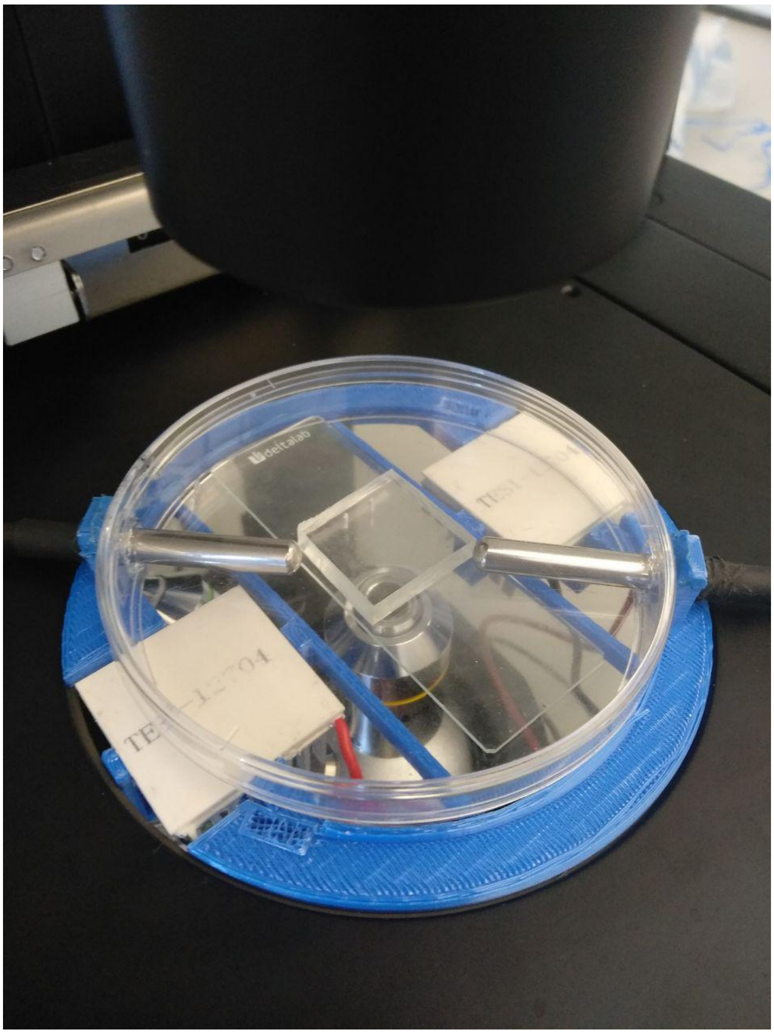

Figure 6.6: The *Heating chamber* on the inverted microscope with the TEC modules, the temperature sensor (shown in two different positions) and the Petri dish containing the microfluidic device

The two small structures on the top with an hole are responsible of keeping the temperature sensor in a steady position and in contact with the microfluidic device, as shown in figure 6.6 (this figure shows the two different positions in which the sensor can be inserted, but the system is designed to work with only one sensor. The two-sensors configuration doesn't bring any significative advantage to the system in terms of accuracy).
The tip of the sensor can access inside the Peltri dish through a hole suitably created and touch the glass support plate of the microfluidic device. This particular askew position is the best one for the sensing of the temperature in the chamber, since in other positions the sensor would be straight above the Peltier cells and the temperature sensed would be much higher than the temperature of the microfluidic device.
As planned, the structure leaves enough space for the lenses, they can be rotated

without touching the structure, as shown in figure 6.7

Figure 6.7: View from the bottom of the *Heating chamber* on the inverted microscope: the lenses have enough space to rotate without touching the structure

6.2 Case

The second structure needed for the project is a **Case** which can contain the Arduino board, the H-Bridge L298N, the pushbuttons, the LCD, and the strip board with all the interconnections of the electronic system.

The starting idea was to design the *Case* as a box, as shown in figure 6.8. This first design was discarded due to the excessive material needed by the 3D printer: more than 100 m of plastic wire and a process time of 40 hours.

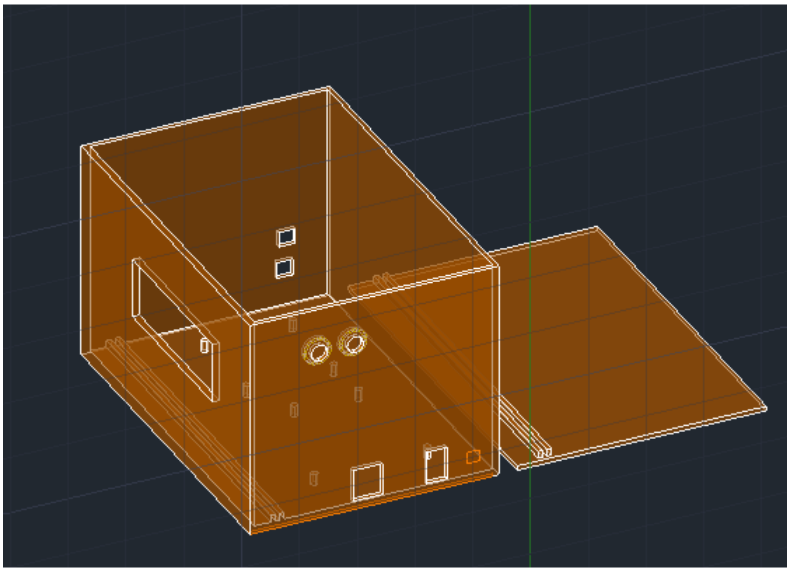

Figure 6.8: First version of the *Case*. The structure on the left is the box in which all the hardware can be contained: it presents the apertures for the LCD and the pushbuttons, and several other holes suitably sized for the cables of the Arduino, the power cables for the H-Bridge and the temperature sensor. The small channel on the bottom left part is sized to hold in a vertical position the stripboard.
The structure on the right is the cover of the box.

Taking into account this aspect, another version of the case was designed. In this second and final version, about the 73% if the material has been removed with respect to the first version, thus allowing an achievable structure in terms of material and time needed. This structure presents the same features of the previous version: apertures for LCD, pushbuttons, cables, contact pins for the fit of the H-Bridge and Arduino,and a channel to hold the stripboard in a vertical position. The drawback of this version with respect to the first one is that it is not closed like a box and thus can not protect the hardware as the other one, but it is not a big issue since this system is going to be used in laboratories or other non-hazardous environments, so a complete protection is not fundamental.
The AutoCAD model is shown in 6.9 while its realization in 6.10.

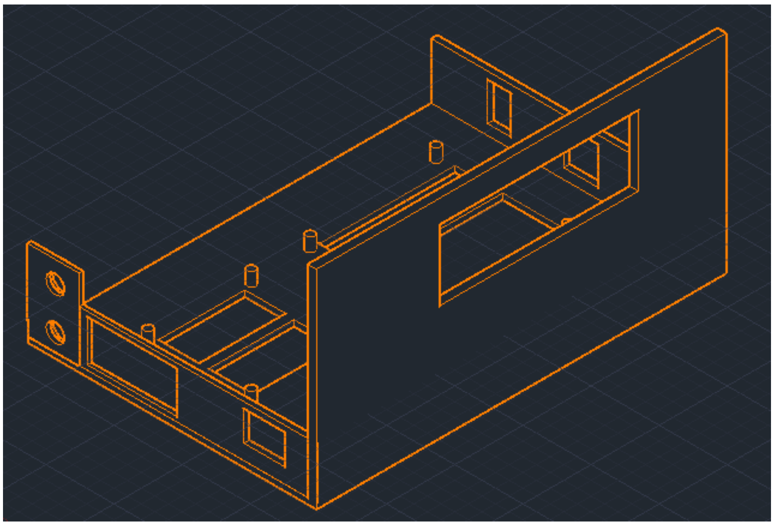

Figure 6.9: Final version of the *Case* in AutoCAD

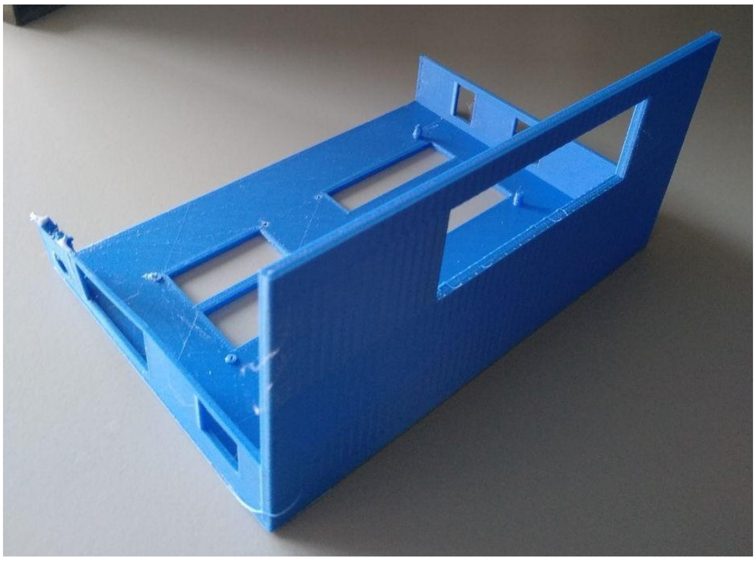

Figure 6.10: Final version of the *Case* obtained with 3D printing

The first components to be mounted are the Arduino board and the H-Bridge, as shown in fig 6.11.

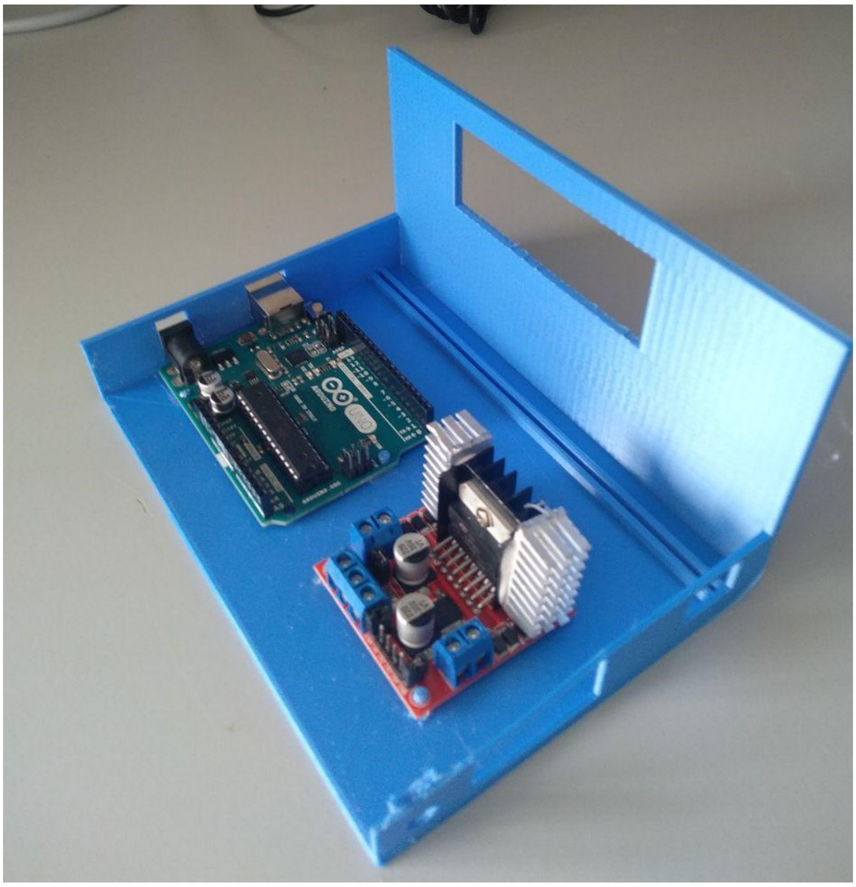

Figure 6.11: *Case* with the Arduino and the L298N mounted.

Next step is the mounting all the other components: first of all, the interconnections have been soldered using the tools available in the microfluidic components, i.e. the soldering station shown in figure 4.10 and a soldering helping hand, as shown in figure 6.12. The schematic followed is the one presented in figure 4.11 and most of the interconnections have been soldered on the stripboard, which is visible in figure 6.12. In particular, the LCD screen is directly mounted on the stripboard and the case is designed so that they fit perfectly.

The final result of is shown in figure 6.13, in which the case is seen from behind and all the cables and components are visible.

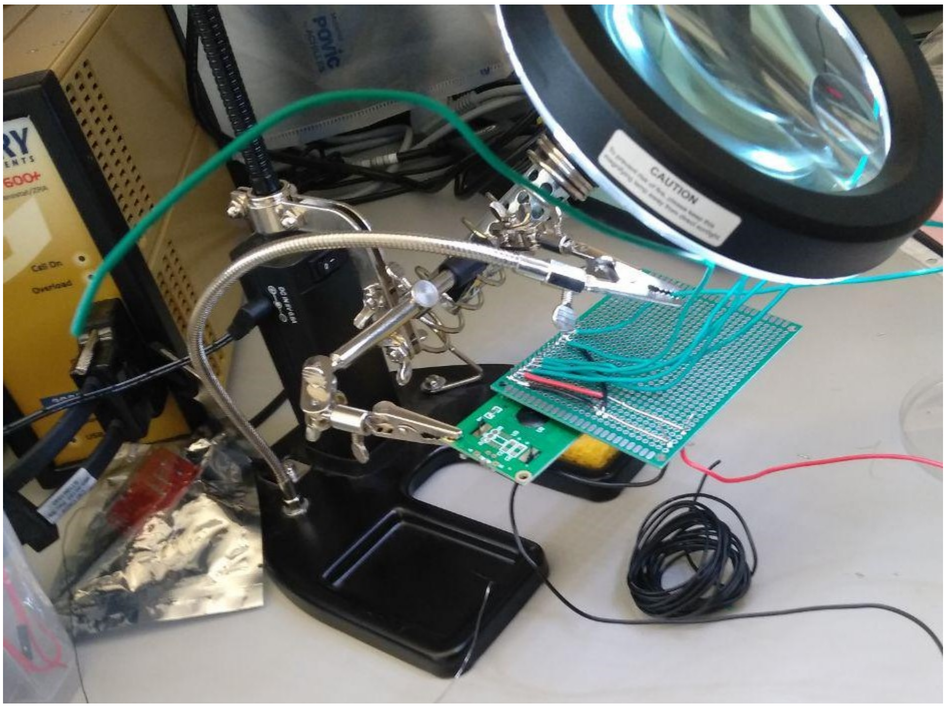

Figure 6.12: Stripboard on the soldering helping hand during the soldering process

On the lateral sides there are the two pushbuttons (one for each side) and also the apertures for the cables needed to enter or exit from the case: the sensor cable, the wires of the Peltier cells and the USB and power cables of the Arduino, shown in figure 6.14. The wires which connect the power supply to the H-Bridge L298N come out straight from behind the case and have to be connected to the pins of the power supply.
A front view of the case can be seen in the figures of the previous chapter, for example 5.4. The user can interact with the system directly from the case, since with the pushbuttons and the screen the desired temperature and amount of time can be selected and also monitored afterwards.
In this way the system has all the desired features: it is compact, portable, simple to use and stand-alone , since it does not require to be connected to a PC after the code has been uploaded once.

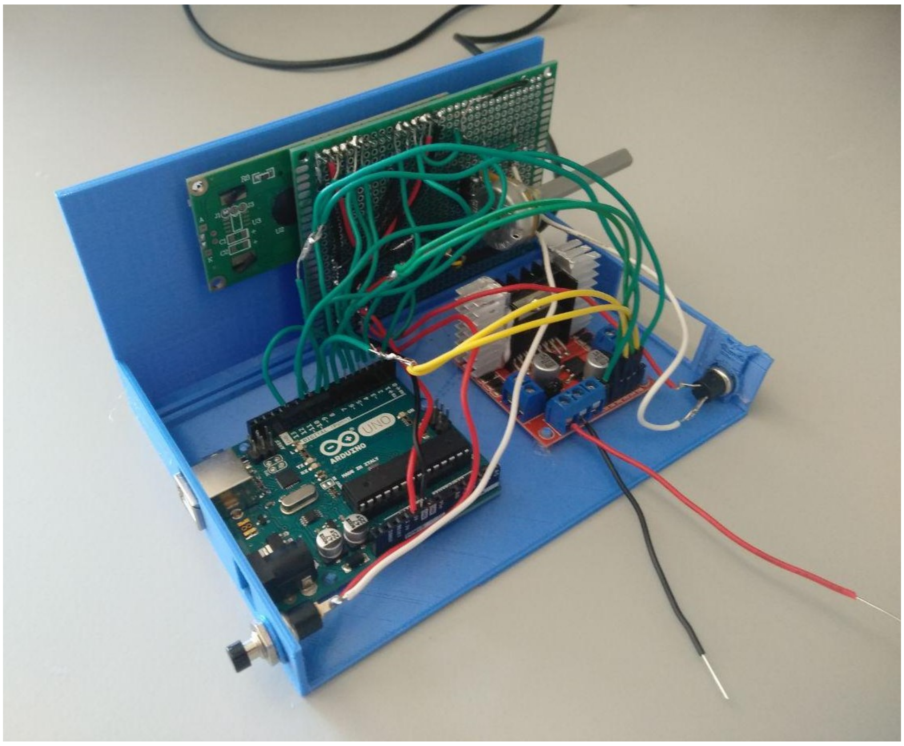

Figure 6.13: Complete *Case* with all the hardware mounted on it

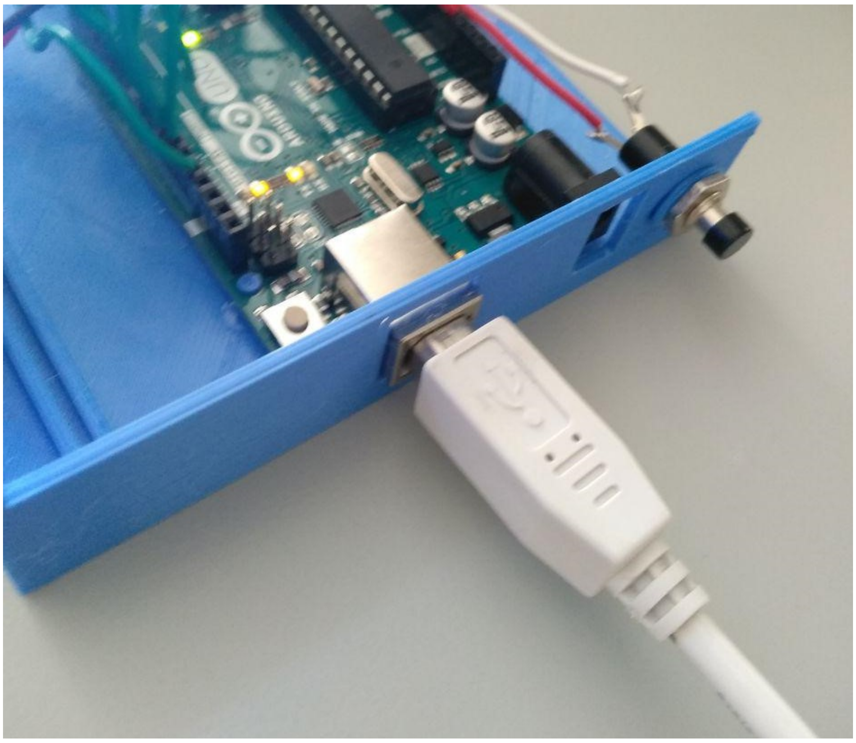

Figure 6.14: Close-up on the left side of the *Case*: the apertures for the power and USB cables of Arduino, as well as the black pushbutton, are visible.

Chapter 7

Testing and final results

Finally the design and building of the system is complete: the only step left is the **Test** of the system and the **Discussion** of the final results.
The tests have been performed by acquiring the temperature data with the *Serial monitor* of Arduino, transferring them in *Microsoft Excel* and using *MATLAB* in order to plot and analyze them. The system has been tested choosing as set point temperatures between 25 °C and 40 °C with a step of 1 °C.
An example of plot of temperature (in °C) versus time (in minutes) is shown in figure 7.1; in this case the set point is **37 °C**, which is also the temperature needed for the main application of the system, the study of *T. Spiralis*.

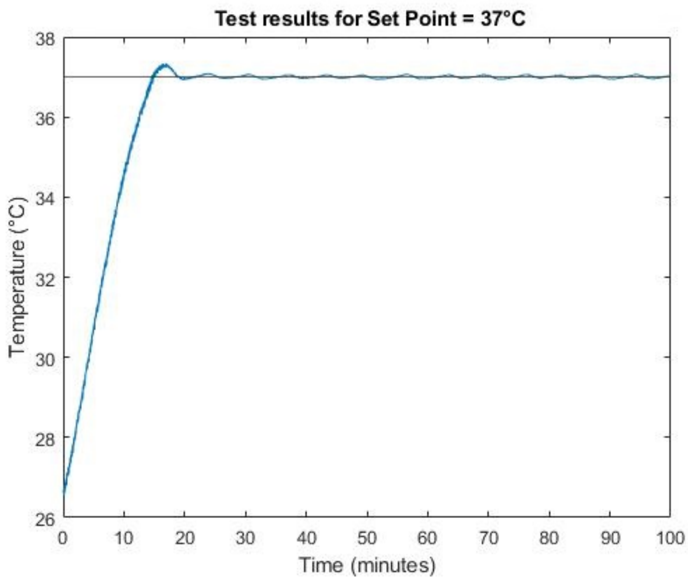

Figure 7.1: Test performed with an input temperature of 37 °C

The initial temperature of the microfluidic device at the beginning is the same as the room temperature of the Microfluidic Laboratory, usually around 25 °C; as soon as the system starts to work, the temperature of the device increases linearly until it reaches the set point and, after an initial small overshoot of 0,25 °C, it starts to have slow and small oscillations around it. For this input temperature, the *rise time* is **15 min 14 s** while the *maximum value of the oscillation* is ± **0,15 °C**.

7.1 final results

Many tests have been performed choosing different input temperatures: all the results are summarized in table 7.1. All the tests have been performed during a time window of **2 hours each**, with a sample frequency of **120 Hz**. For each test is reported the **Average Temperature**, i.e. the average value of all the samples considering the time samples after the overshoot, the **Rising time**, i.e. the time required for the temperature to reach the 90% of the set point value, and the **Overshoot**.

Selected temperature	Average temperature	Rising time	Overshoot
25 °C	25,04 °C	2 min 21 s	± 0,32 °C
26 °C	26,01 °C	3 min 01 s	± 0,26 °C
27 °C	27,09 °C	3 min 56 s	± 0,26 °C
28 °C	28,06 °C	4 min 42 s	± 0,22 °C
29 °C	28,95 °C	5 min 35 s	± 0,26 °C
30 °C	30,11 °C	6 min 50 s	± 0,26 °C
31 °C	30,96 °C	8 min 04 s	± 0,29 °C
32 °C	31,99 °C	9 min 15 s	± 0,32 °C
33 °C	33,08 °C	10 min 21 s	± 0,32 °C
34 °C	34,07 °C	11 min 36 s	± 0,35 °C
35 °C	35,10 °C	12 min 59 s	± 0,29 °C
36 °C	35,92 °C	14 min 09 s	± 0,27 °C
37 °C	37,02 °C	15 min 14 s	± 0,25 °C
38 °C	38,12 °C	16 min 37 s	± 0,21 °C
39 °C	39,03 °C	18 min 32 s	± 0,23 °C
40 °C	40,11 °C	19 min 49 s	± 0,23 °C

Table 7.1: Final results

7.2 Discussion of results

From the results shown in table 7.1 some considerations can be done.
First of all, as expected the *rise time* increases with the selected temperature, and this is straightforward since the chip requires more time to be heated to higher temperatures. This experimental result is exploited in figure 7.2 which plots the rise time versus desired temperature,

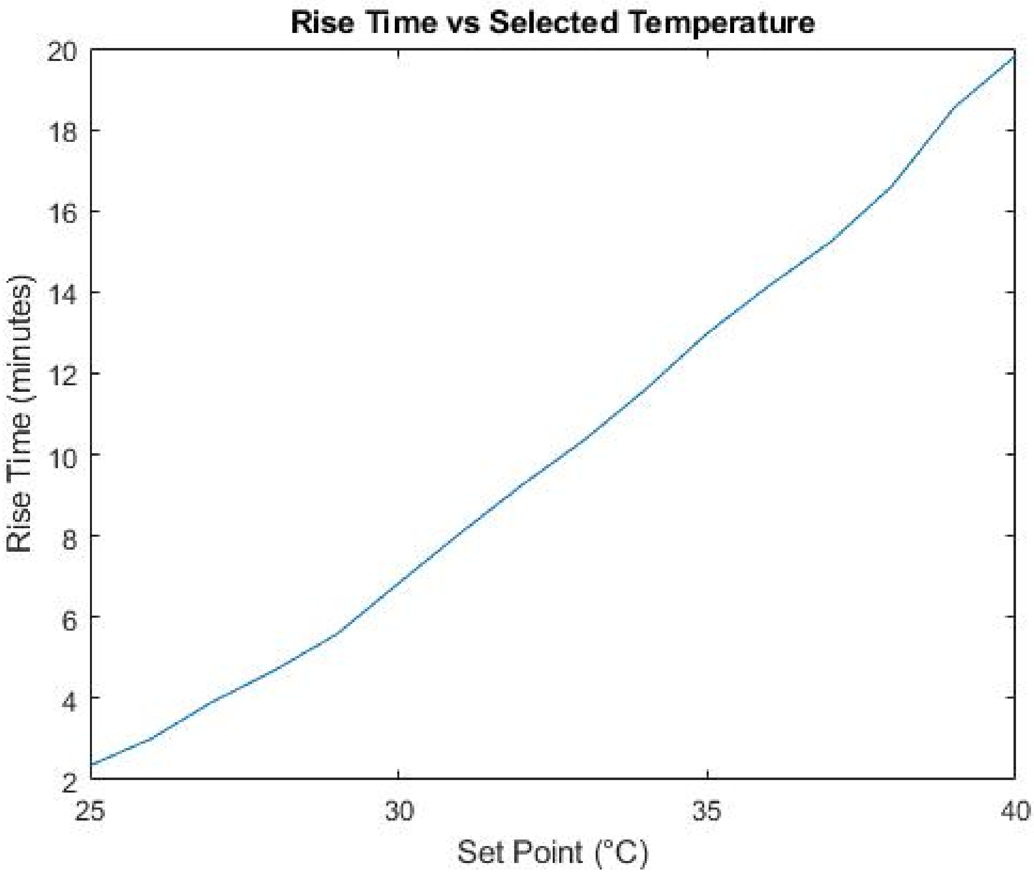

Figure 7.2: Rise Time vs Selected Temperature

Regarding the *Average temperature*, the bigger error in magnitude is of **0,12 °C** in the case where a temperature of *38* °*C* is selected. In all the other cases the error is lower. The *Overshoot* ranges **from ± 0,21°C to ± 0,35 °C** depending on the input temperature; this is not a problem in the study of *T. Spiralis* which is the main application of this project. However, for other applications the overshoot has to be taken into account because it could maybe lead to some issues.

7.3 Conclusion

The temperature controller performs all the tasks asked in the requirements: it allows the user to select the desired temperature of the microfluidic device and also the amount of time in which the temperature needs to be monitored. It is portable, robust and stand-alone: furthermore, it is designed to be mounted and used with the instruments of the Microfluidic Laboratory of the Pázmány Péter Catholic University, which requested this project.
In conclusion, two pictures of the working system are shown in pictures 7.3 and7.4.

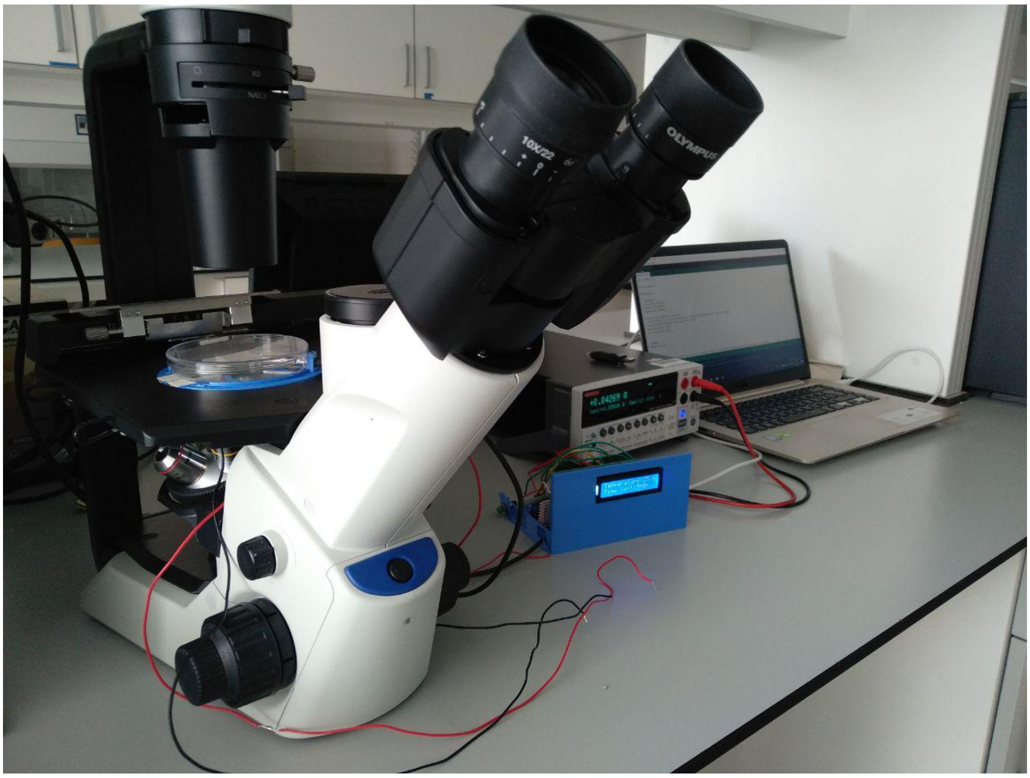

Figure 7.3: Final version of the system

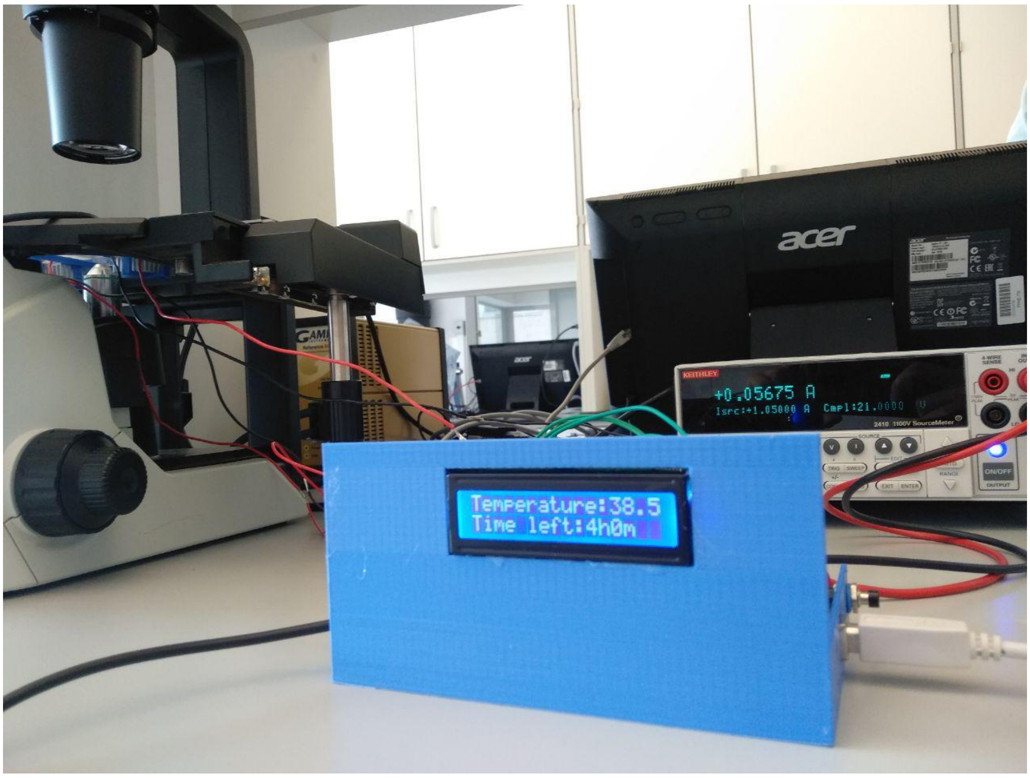

Figure 7.4: Final version of the system

Bibliography

[1] Kwang W. Oh. *Multidisciplinary Role of Microfluidics* . University at Buffalo, State University of New York , 2017.

[2] Auroux P., Iossifidis D., Reyes D., Manz A., *Micro Total Analysis Systems*, Analytical Standard Operations and Applications, Analytical Chemistry, vol. 74, pp. 2637-2652, 2002.

[3] Squires T., Quake S., *Microfluidics: Fluid physics at the nanoliter scale*, Reviews of Modern Physics, vol. 77, pp. 977-1026, 2005.

[4] Gottstein B., Pozio E., Nöckler K., *Epidemiology, Diagnosis, Treatment, and Control of Trichinellosis*, Clin. Microbiol. Rev. vol. 22(1), pp. 127-145, 2009.

[5] Dupouy-Camet J., Bruschi F., *Management and diagnosis of human trichinellosis*, World Organisation of Animal Health Press, vol. 32, pp. 37-68, 2007.

[6] Babos S., *Élősködő fonálférgek III. Nematoda parasitica III. Magyarország Állatvilága*, Fauna Hungariae III. vol. 4, pp. 13-15, 1958.

[7] Katz M., Gwadz R., *Parasitic Diseases*, Springer Verlag, Second edition, New York,pp. 260-265, 1989.

[8] Piekarski G., *Medical Parasitology*, Springer Science & Business Media,New York, pp. 320-324, 2012.

[9] Allen K., *Comparative growth and development of Trichinella spiralis in vitro and in vivo, with a redescription of the life cycle*, Experimental Parasitology, vol. 16, pp. 74-106, 1965.

[10] Wang ZQ., Cui J., *Diagnosis and treatment of trichinellosis*, Chinese Journal of Parasitology & Parasitic Diseases, vol. 26(1), pp. 53-57, 2008.

[11] Murrell K.D., *Foodborne parasites*, International Journal of Environmental Health Research,vol. 5(1), pp. 63-85, 1995.

[12] Roth H., *The in vitro action of trichina larvae in immune serum-a new precipitin test in trichinosis*, Acta Pathologica Microbiologica Scandinavica banner, vol. 18. pp. 160-167, 1940.

[13] N. S. Nise *Control System Engineering*, 5th Edition, Wiley, 2008.

[14] Karl Johan Astrom, Richard M. Murray *Feedback Systems*, Princeton University Press, New Jersey, 2008

[15] Bemjamin C. Kuo *Automatic Control System*, Prentice - Hall, New Jersey, 1975.

[16] F. Ioannidis *Intelligent controller based on Raspberry PI*, School of Computer Science, University of Manchester, 2014.

[17] Jean Claude Mugisha, Bernard Munyazikwiye, Hamid Reza Karími *Design of temperature control system using conventional PID and Intelligent Fuzzy Logic controller*, International Conference on Fuzzy Theory and Its Applications (iFUZZY), 2015.

[18] S. Bennett *Development of the PID controller*, IEEE Control Systems Magazine (Volume: 13 , Issue: 6 , Dec. 1993)

[19] Maltezos, G.; Gomez, A.; Zhong, J.; Gomez, F.A.; Scherer, A. *Microfluidic polymerase chain reaction*, Appl. Phys. Lett. 2008, 93, 243901:1–243901:3.

[20] Maltezos, G.; Johnston, M.; Taganov, K.; Srichantaratsamee, C.; Gorman, J. *Exploring the limits of ultrafast polymerase chain reaction using liquid for thermal heat exchange: A proof of principle*, Appl. Phys. Lett. 2010, 97, 264101:1–264101:3.

[21] Khandurina, J.; McKnight, T.E.; Jacobson, S.C.; Waters, L.C.; Foote, R.S.; Ramsey, J.M. *Integrated system for rapid PCR-based DNA analysis in microfluidic devices.*, Anal. Chem. 2000, 72, 2995–3000.

[22] Chandrabhan Patel . *Simulation of Thermoelectric Cooler and H-Bridge Driver for Precise Temperature Control Applications* , NATIONAL INSTITUTE OF TECHNOLOGY, KURUKSHETRA-136119 JUNE 2017.

[23] Yang Jiagu *Design of Temperature Control System of Laser Diode Based on MAX1968*, YANG Jiagui , Electronic Engineer , 2007

[24] Jim Williams *A Thermoelectric Cooler Temperature Controller for Fiber Optic Lasers* , LINEAR Technology, Application note 89, 2001

[25] G. Jeffrey Snyder, Jean-Pierre Fleurial, and Thierry Caillat *Supercooling of Peltier cooler using a current pulse*, Journal of Applied Physics 92, 1564 (2002); https://doi.org/10.1063/1.1489713

[26] Hoffmann, D, Curtin, M, Moore, E, Loughran, M; *Development of an integrated microsystem for injection, transport and manipulation of encoded microbeads*, Lab on a chip, 7(3): 392-395, 2007.

[27] Zdenek Slanina, Martin Uhlik, Vaclav Sladece. *Cooling device with Peltier element for medical applications*, VSB - Technical Universty of Ostrava , Faculty of Electrical Engineering and Computer Science , Ostrava, Czech Republic, January 2018

[28] Dave Salerno. *Closed-Loop Temperature Regulation Using the UC3638 H-Bridge Motor Controller and a Thermoelectric Cooler*, Application Report SLUA202A - September 2001.

[29] Huan-Liang Tsai, Jium-Ming Lin, *Model Building and Simulation of Thermoelectric Module Using Matlab/Simulink*, Journal of ELECTRONIC MATERIALS, Vol. 39, No. 9, 2010

[30] Ivaylo Belovski, Liliya Staneva, Anatoliy Aleksandrov, Pavlik Rahnev *Mathematical Model of Thermoelectric Peltier Module* , Journal of Communication and Computer 14 (2017) 60-64

[31] Mohit Thakkar *A report on "Peltier (thermoelectric) cooling module"*, Technical report, February 2016

[32] Vincent Miralles, Axel Huerr, Florent Malloggi, Marie-Caroline Jullien *A Review of Heating and Temperature Control in Microfluidic Systems: Techniques and Applications*, Diagnostics 2013, 3, 33-67

[33] Jianing Yang, Yingjie Liu, Cory B. Rauch, Randall L. Stevens, Robin H. Liu, Ralf Lenigk, Piotr Grodzinski *High sensitivity PCR assay in plastic micro reactors*, Physical Sciences Research Laboratories, Motorola Labs, November 2002