UNIVERSIDAD DE LOS ANDES

THESIS

Infrared Spectrophotometry using Thermal Cameras for Microfluidic Systems

THESIS Author:

Juan Sebastian Velasquez Montoya

THESIS Advisor:

Johann Faccelo Osma Ph.D.

For the Title of Bachelor in
Electronic Engineering

Engineering Faculty
Department of Electric and Electronic Engineering

Introduction

Spectrophotometry in microfluidic systems has become an important tool to observe and obtain valuable information about small scale reactions, which tend to be conducted in fields such as physics, molecular biology, chemistry, and biochemistry. Spectrophotometry tends to focus on measuring the transmission and reflection of different wavelengths of light through a medium; in the scope of this thesis, the medium used is a fluid in a small-scale controlled system [1].

Some uses of spectrophotometry focus on using the visible light spectrum to find frequencies that are absorbed by the medium, to correlate the contents of the medium itself. However, in fields such as biology and chemistry, it is often useful to find the heat radiated by a reaction in a controlled system. Understanding if a reaction is exothermic or endothermic can provide valuable information about the contents of the fluid and how it should behave under different conditions [2]. Furthermore, with lithographic techniques it is possible to manipulate small quantities of fluids, which allow to correctly observe the chemical reactions in a controlled environment [3].

The proposed work, consists of the implementation of a device that facilitates the observation of thermal reactions, using the near-infrared spectrum (NIR: 780nm – 3000nm), in a microfluidic system using infrared cameras. This thesis' purpose focuses on the characterization of the thermal sensor to accurately observe thermal changes that can be seen in the proposed microfluidic systems. For this, it is also important to create a data management system that allows for proper data manipulation and visualization. Thus, a device capable of recording video samples and capturing images is needed. The last scope of this thesis is to create a methodology for the proper calibration and replication of the thermal sensors implemented in the device, to be used with the proposed microfluidic systems.

Although, spectrophotometers that allow for a frequency sweep from visible to infrared light are already available in the market, they are not quite as affordable. The proposed work focuses on a design that couples well with the laboratory equipment already developed for the evaluation and understanding of reactions in microfluidic systems in the BioMicroSystems research group at Universidad de Los Andes.

Objectives:

General Objective:

- Calibrate and implement a device capable of taking temperature measurements through time using video-like characteristics and infrared cameras for microfluidic systems.

Specific Objectives:

- Analyze and characterize the thermal camera for microfluidic systems taking into account thermal variations.
- Design a data acquisition system that allows for proper data manipulation of the temperature measurements.
- Implement a calibration system that allows the user to observe correct measurements in the proposed microfluidic systems.

Table of contents

Theoretical-Framework

Spectrophotometry

A method used to measure the transmission or reflection of visible light to find out the absorbance of a fluid [1]. This is done using Beer-Lambert's Law, $A = \mathcal{E}lc$, where A is the absorbance of the fluid [%], $\mathcal{E}$ is the molar extinction coefficient, l is the length of the path light travels to the sample [cm] and c the concentration of the fluid [mol] [4]. It should be noted that 2mm acrylic has light transmission of approximately 90% up to 1600nm in the near-infrared spectrum [7].

The infrared spectrum

A type of electromagnetic waves not visible to human eyes, that can be expressed as heat. These waves have frequencies between 300 GHz and 430THz, which makes their wavelengths between 700 nanometers and about 1mm [5]. It should be noted that near-infrared ranges from 780nm to 3µm.

Thermal imaging

The process through which infrared radiation (the infrared spectrum can be partially used, such as NIR) is modified as visible images that show the temperature distribution [6]. The type of thermal imaging done in this thesis is done by converting the temperature values given by the sensor into a gradient color matrix.

Microfluidic systems

Systems that implement fluid manipulation in channels with diameters of tens of micrometers. Often used to simulate environments from fields such as chemistry, physics, biology, and molecular biology [3]. For the development, and calibration of this thesis a standard model of the microfluidic system was used. (Annex 1: Proposed micro-fluidic system)

Seebeck Effect

This is the process through which a material with at least 2 different conductors, with different temperatures at different junctions creates a difference in electromotive force. This, in turn, creates an electric current through the material, proportional to the temperature difference at the junctions. This phenomenon can be modeled by the equation $V = \alpha\Delta T$, where α is the Seebeck coefficient of the material and, ΔT is the temperature difference at the junctions [9].

Previous Works

Some of the popular methods used for thermal sensing in microfluidic systems have included processes like the Seebeck Effect, either using thermocouples, or temperature sensors with submersible probes. Thus, once calibrated and keeping one junction as a control, it allows to measure the temperature of the other junction located in the microfluidic system [9].

Other ways of obtaining precise temperature readings also include the creation and implementation of temperature-controlled baths for the microfluidic systems, which in turn allow for the observation and replication of chemical reactions at different temperatures. [11] uses Chromium/Platin micro-thermocouples and heaters from Pt films, as well as resistive temperature sensors to not only control the temperature of the microfluidic system, but to sense temperature changes induced by the chemical reactions taking place in the system itself. However, methods for

obtaining thermal measurements in microfluidic systems often use thermal imaging, or frequency sweeps in the IR or UV spectrum to find the transmittance of the fluid.

An example of this method can be observed in experiments such as [12], where the tissue specificity of natural dyes such as beetroot is evaluated. The process is done by doing a spectrum sweep close to UV light to find the frequencies that are most absorbed by the dye. This allowed the researchers to find information about the pigments the dye has and its stains in different tissues. This same process can be done through the IR spectrum to find the type of heat that a fluid or reaction can emit.

This thesis is a continuation of the project: Thermal Cameras for Microfluidic Systems [8]. The main objective of the project was to design and implement a device capable of doing thermal measurements with the ability to display the information in a 2-dimensional gradient color grid. Thus, it would allow for thermal sensing while also performing a visible spectrum sweep, when paired with the laboratory's spectrophotometer. Devices capable of doing the complete visible to IR frequency sweep can be found in companies such as LabCompare [10], which possess spectrometers that have wavelength ranges from 1/7000 cm to 1/350 cm using FITR (Fourier-Transform Infrared Spectroscopy). However, the seamless integration of such systems, with the equipment previously developed as one device, would pose a greater challenge.

Scope

The scope of this thesis is a device that serves as a hub for electronic equipment for microfluidic systems, as well as a thermal sensor for NIR spectrophotometry. The implemented device is to have video processing and data manipulation capabilities as well as serial communication with Arduino microcontrollers and TCP communication through local networks using a Client-Server model.

The scope of this thesis also covers a methodology that allows for proper calibration and characterization of thermal cameras for use with the standard microfluidic systems. The following diagram shows a simple architecture of the scope of the thesis.

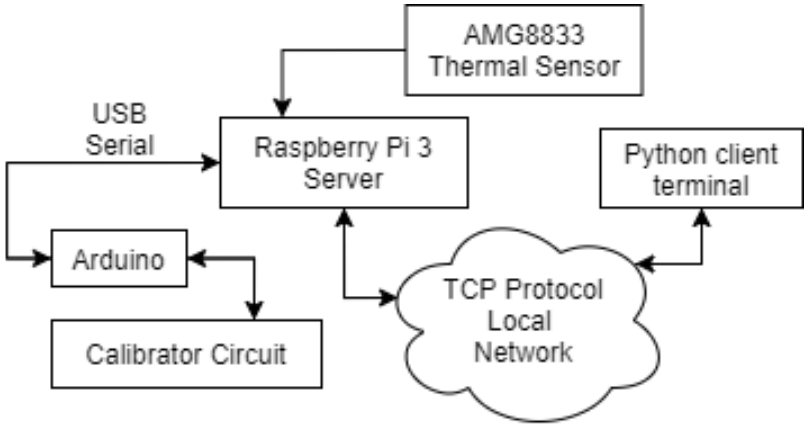


Figure 1: Simplified device diagram

Methodology

The proposed methodology of this thesis is as follows.

ACTIVITY	STARTING WEEK	DURATION
System Analysis and Familiarization	1	2
Evaluation of Software Implementations	2	2
Software Implementations	3	3
Software Evaluation	4	3
Analysis of Experiments for Calibration	5	2
Device Calibration	6	3
Evaluation of the Calibration of the Device	7	4
Tests after Calibration	9	2
Device Characterization	10	2
Final Changes	11	2
Final Tests	12	2
Final Observations and Conclusions	14	1

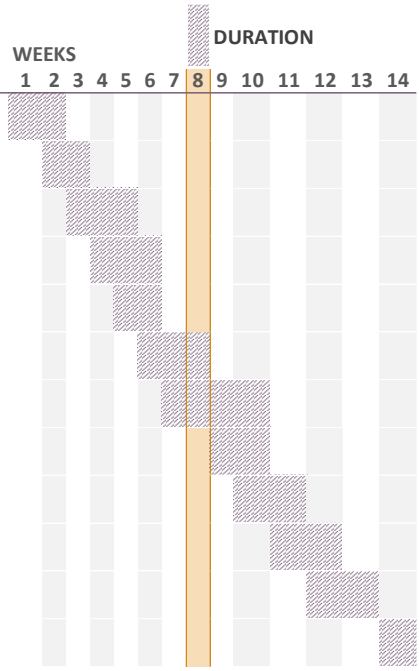


Figure 2: Methodology Schedule

System analysis and familiarization

As previously stated, the methodology starts with the analysis, familiarization and understanding of the project: Thermal Cameras for Microfluidic Systems [8]. The main component to analyze is the software developed and how the thermal imaging is shown in the interface. The way the data is collected and shown in real time will also be considered.

After understanding the scope of the project and the functions of the implemented device, the software will be modified to allow for proper data manipulation as well as integration with the existing spectroscopy equipment in the laboratory.

Software

After understanding the device and all its components, the software changes will be implemented. This phase focuses on the changes that need to be made for the embedded system [8] to be used as a hub that allows for remote access and serial communication with Arduino microcontrollers over a local network. This will be done using TCP protocol over a Client-Server model, that in turn communicates with the microcontrollers using a USB connection.

The video characteristics and the extensions for data manipulation will also be implemented, as well as the possibility for electronic improvements. However, the project's major electronic components are the AMG8833 thermal camera and the RaspBerry Pi 3, which should have enough RAM to process the thermal imaging and data manipulation needed [13].

Calibration

Then, the calibration process of the thermal sensor will take place. This step will consider the different variables that affect the thermal sensor's measurements, such as the sensor's distance to the measured surface, the material that encases the microfluidic systems as well as a linear regression curve that calibrates the sensor for proper thermal sensing.

This process will be done by first adjusting the calibration curve of the thermal sensor using linear regression. This would in turn allow to have a control distance for the temperature vs. distance calibration. This calibration will also be done using a linear regression curve, since the distance changes for which the device will be used should not exceed 10cm. Once the device has been calibrated using both methods, the final factor to consider, is the calibration using the standard acrylic microfluidic system that encases the fluid.

Characterization

Once the thermal sensor is properly calibrated, the characterization of the device will take place using the standard microfluidic system. During this phase, the calibration process will be repeated several times, to compare and analyze the calibration itself as well as the devices functionality with the standard microfluidic system (Annex 1: Proposed micro-fluidic system).

Once enough data to form a correlation from the calibrations has been recorded, the process of characterization will be done. After which, the procedure for thermal sensor characterization for the standard microfluidic system will be concluded.

Materials

The main components used from the project, Thermal Cameras for Microfluidic Systems [8] are as follows:

- Raspberry Pi 3 Model B
  The most important specifications that the earliest model of third generation Raspberry Pi has, are 1GB of RAM, a Quad Core 1.2GHz CPU, as well as a BCM43438 wireless chip.
- Thermal Sensor AMG8833
  A thermal camera that features an 8x8 grid of sensors, that provides a 64-number temperature measurement array through I2C protocol.
- 2mm Acrylic Film
  The standard microfluidic system as well as the casing of the thermal sensor and final design are laser cut in this material.

System analysis and familiarization

The embedded system designed in [8] features a case with the Raspberry Pi 3, the AMG8833 thermal camera, as well as a 240x320 ILITEK touch screen, that serves to visualize the gradient color matrix and control the device. The device is to be completely utilized through the touch screen; therefore, the interface is created using a 240x320 layout, and a default fullscreen mode which cannot be exited through the touchscreen. The initial screen features an 8x8 grid with buttons that allow for fullscreen video recording as .MKV with the 240x320 color grid in the middle and data recollection as .CSV. It also features a settings button and a slider that turns the 8x8 into a 32x32 interpolated

gradient color matrix that helps visualize the thermal imaging (Annex 2: 240x320 thermal imaging interface).

The settings screen features five buttons, which configure: lower temperature limit, upper temperature limit, point calibration, exit, and shutdown, respectively (Annex 3: 240x320 settings interface). The temperature limits modify the colors observed in the gradient matrix; the lower limit defines the temperature from which blue is set, while the upper limit defines the color red. The point calibration button helps adjust the temperature sensed by the central 4 pixels of the 8x8 thermal grid. The exit button, then returns to the 8x8 gradient thermal map, while the shutdown button completely stops all processes and turns off the raspberry Pi 3 by calling an exit subprocess.

The overall design is very useful as a thermal sensing camera; however, it is not well adapted for use as a hub with the electronic equipment for microfluidic systems. Data-wise, although it already possesses video capabilities, it is not well suited for use in a desktop environment, as it is shown as fullscreen per default and its interface size is rather small, especially when used through the HDMI cable with a computer screen. On the other hand, the data recollection through the .CSV extension works well and does not require significant changes. However, since the system itself is embedded and designed to be used separated from a desktop (screen, keyboard, mouse, etc.) the root directory for file management is set for a USB drive and cannot be changed. Also, when the USB is not connected, or its name is modified, the program generates an error and exits. Another error that causes the current program to exit, is in the subprocess used for the video capture. If an out of memory error happens during the video recording due to the creation of a new thread, the raspberry Pi logs out the user, and thus the program exits, and the data/video being collected/recorded is lost. The last modification that should be considered is the shutdown button, which once the software is modified to work as a hub for the electronic equipment for microfluidic systems, will be a liability that could cause problems in the long run.

Software

Evaluation of software implementations

Since the device is to be operated from a desktop configuration, the first thing is to modify the size and default initialization of the program. The program was created using the PyGame library, which allows for a lot of versatility creating the user interface, however, it lacks a lot of components and relies on the continuous updating of the interface screen to function properly. The library, as its name indicates, is often used for simple game development in python and for the development of simple GUIs. Thus, when a lot of buttons and images refresh constantly it can cause disruptions or visual glitches in the interface. However, since the general system architecture, as well as the interface updating system and the event mouse handling are already well developed and both serve as the backbone of the software, it was decided that the software modifications would be continued using the PyGame library. If the user interface was to be started from scratch, it is recommended to use tools such as Qt, tKinter, or Gtk, which allow to use versatile yet optimized components for GUIs in python [14].

Once the size of the interface has been modified, it would also be necessary to increase the interpolation factor. This change would allow for a less pixelated and smoother way to visualize the data; as well as a more detailed and real approximation of what the heat map from thermal sensor looks like. With regards to the data collection processes, the program should also allow the user to

choose between different file extensions for both data collection and video recording, such as .TXT, or .MP4. The video recording modifications should take into account the existing process, which captures the video of the full screen using a python terminal subprocess. It should add a way to capture the video of the gradient color matrix, without recording the rest of the user interface.

In the settings screen, the shutdown button should be removed since it could create situations where data is compromised. Moreover, since the program is to be used in a desktop configuration, the user should be able to properly shut down the raspberry Pi 3 from the Raspbian drop down menu. Another modification is that the existing program does not allow to change the root folder where the data and videos are stored. Thus, there should be a directory explorer that lets the user know and choose the current directory, whether it be in a USB drive or in a folder inside the raspberry Pi's memory.

The program should also have the option to start a server for local networks, based on a client-server model, which would allow the user to remotely control the interface, collect data, record videos and even export the files to the client, through TCP protocol. Lastly, the raspberry Pi, should have the ability to control Arduino microcontrollers through serial communication via the USB protocol. This would in turn allow to control other devices connected to the arduino using the raspberry Pi as a hub.

Software implementation and evaluation

The size of the user interface was modified to fit a larger portion of the screen. The size chosen was 480x640, since sizes bigger than that caused stability issues in the gradient color matrix once the interpolation was applied. The user interface size was also kept constant, due to the tool used for the creation of the GUI. PyGame does not have optimized interface layouts that allow for resize of the buttons once the initial interface has been created. Rather, if the size were to be changed, the overall layout would have to be updated. It should be noted that by using a more powerful microcomputer, the size of the interface could be increased keeping the integrity of the system.

Following the size change in the interface, the interpolation done by the library scipy.interpolation was also doubled. The type of interpolation used is cubic interpolation, since the results in [8], showed that it is the best interpolation method in the scipy.interpolation library for thermal imaging. The resultant matrix was modified from a 32x32 (1024 squares) gradient color matrix, to a new 64x64 (4096 squares) gradient color matrix that details the thermal imaging done by the sensor by a factor of 4. The resulting interpolation can be observed in the following image.

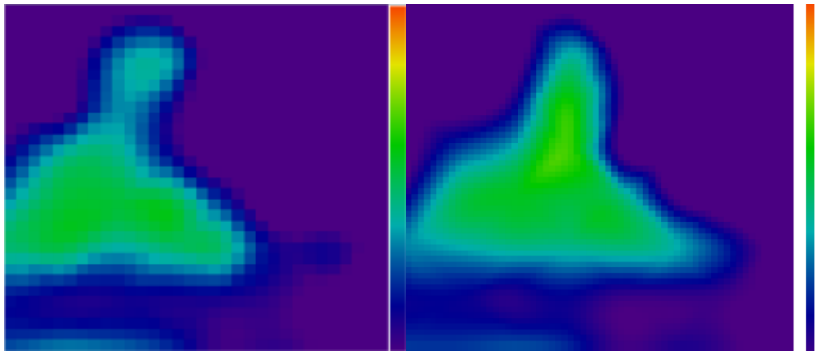


Figure 3: Thermal imaging interpolation (before - after)*

*It should be noted that the images, before/after the interpolation, are different; since the program works in real time, it is not possible to apply the size changes and the different interpolation factors to a single image. However, it can be observed that the after image is both sharper and more detailed.

The settings interface was modified to allow the user to choose the extension of the data collection and the video recording. The data collection was only previously implemented using a .CSV extension, it was modified to support both .TXT and .PNG extensions. The .TXT data collecting, works in the same way, which takes between 8 to 10 thermal readings of the 8x8 matrix per second and shows it as an array of temperatures per pixel. The .PNG extension, however, takes a screenshot of the matrix grid in the thermal imaging interface. This screenshot is dependent on the type of information shown; it can capture the temperature array or the interpolated image.

The video characteristics have been modified to support both .MKV and .MP4 extensions. The user also has an option to choose the type of video to be recorded; the existing video could capture fullscreen format, which showed all the menus and Raspbian components. It is now possible to choose only the thermal map showed in the 8x8 grid of the thermal imaging interface. As implemented with the .PNG extension, the video capture can record the temperature array, or the interpolated image. Figure 4 shows part of the settings interface that allows the user to select the file extensions. It should be noted that the text above the corresponding images changes depending on the extension currently active. The selection method is implemented through buttons, since the PyGame library does not contain a dropdown list component, and it would have to be manually implemented. It should be noted that the thread creation for the video capture was optimized as to decrease the memory being taken up. It should also be noted that the video capture is associated to a python terminal, thus, if the program is to be executed as a standalone script (without being called by the terminal), the video recording will most likely log out the user. To call the program, the terminal commands are shown in figure 5. Furthermore, an example of the video files recording for both MAP and FULLSCREEN can be observed in [15] and [16] respectively.

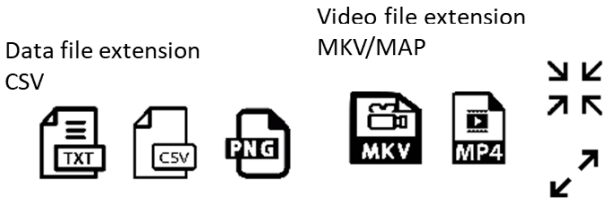


Figure 4: File extension selection in the settings interface

```
pi@raspberrypi:~ $ cd Camara/
pi@raspberrypi:~/Camara $ python2 thermal_cam.py
```

Figure 5: Python terminal commands

Once the video and data manipulation has been correctly implemented, a directory explorer is added to browse and save the files generated. Since PyGame has no components for directory explorer, or file explorer, it had to be done either using an OS process or another GUI window. The file explorer was implemented by creating a tKinter file directory window once the folder button is pressed. The file directory allows to select the root folder to save all the files generated through

video recording and data collection. It should also be noted that the root folder for the data creates folders for both data and video files, respectively. The current root folder for all the data was set up as the Raspbian desktop as to make the data visualization and file manipulation for the user as fast as possible.

The next step is to create the serial communication with the Arduino microcontroller and the Client-Server model for local networks. The serial communication is implemented using the Serial library which communicates through a USB serial port given the /dev location on the Raspberry Pi 3 system. The implementation is done by having the Arduino /dev location as a constant to encourage the Arduino microcontroller to not be removed from the device. However, it can be modified by simply changing the microcontroller's location in the program's code. To send the data to the Arduino, both sides of the serial communication use the Serial.println() method while also using the Serial.readUntil() to read up to the \n character of the array received. The communication protocol consists of a request-response or request-reply model; once the raspberry communicates with the microcontroller, a reply needs to be sent to the raspberry Pi within a few milliseconds. Thus, if there must be persistence in the data collected through the arduino microcontroller, it is encouraged to use the non-volatile (electrically erasable programmable read-only memory) EEPROM included in the arduino. One major complication in the implementation of the interface to communicate with the Arduino microcontroller, is the fact that the PyGame library has no textbox component that allows the user to type or enter characters, and it has to be manually implemented.

A work around this problem is the implementation of a simple rectangle which updates every few milliseconds as a thread of its own and hears when a keyboard key event is registered. Then, with the same updating method, a text is modified to show what the user has previously typed. An enter key event sends the data to the arduino and listens to the arduino's response. Thus, whenever the user types any character, the rectangle will show the string typed. It should be noted that the implementation of the text input box is directly connected to the serial communication of the arduino's location. This way, if more arduino microcontrollers wish to be integrated into the system, all that is needed to do, is add a new text input box that has the new microcontroller's /dev location. A simple demonstration of the serial interface can be observed in [17], where the LED_BUILTIN of the microcontroller is activated (Annex 4e: serial communication interface). A consequence of this type of implementation, is that the arduino microcontroller is the system that interprets the commands sent. The updating rectangle serves the same purpose as the serial communication input box which can be found in the arduino IDE.

The last modification that must be implemented, is the Client-Server model to allow for the remote control of the program over local networks. This is done using the socket library, which allows to create a TCP connection between two devices using the local IP address assigned. A model for remote control over global IP was considered; however, due to a lack of direct access to the router's configuration during the creation of the project, as well as having many different devices connected to the same internet hub, meant that the configuration of communication through a TCP port could not be properly implemented.

The Client-Server model (CSM) features a button that enables the remote access connection (it can also be configured to start with the initialization of the program) and shows the local IP address of the Raspberry Pi 3. Since all the methods for the correct recollection of data and video recording have been implemented, as well as the methods for the serial communication with the microcontrollers, all that is needed is to call the corresponding method, once the TCP connection

has been established. The CSM also allows the user to modify folder directories, temperature limits, as well as file extensions. The major implementation of the CSM is the export function which allows to browse video and data files in the supported extensions and send them to the client over the TCP connection. The client implementation is fully operated through the terminal and features a help menu to assist the user. A simple demonstration of the menu as well as the export functions can be observed in Annex 5 and [18] respectively. The general architecture of the system can be observed in Figure 1.

Calibration

Theory for calibration

Before the calibration can be performed, it is necessary to understand how infrared thermometers are able to accurately measure the temperature of an object. If an object has a temperature above 0 K, it will emit electromagnetic waves which are proportional in magnitude to its temperature because of the motion of its particles. The relationship between these two variables can be observed from the following linearized formula:

$$A = \alpha\beta(T_2^4 - T_1^4) \qquad (1)$$

$$T_2 = \frac{T_0}{\sqrt[4]{\beta(T_1)}} \qquad (2)$$

Where T_2 is the absolute temperature of the object, T_1 is the ambient temperature, T_0 is the radiant temperature, β is the radiation rate, α is the Stephen Pozman constant and A is the radiation degree [19]. Thus, through a comparison between the magnitude of the electromagnetic waves emitted and received, the infrared thermometer can obtain the value for the absolute temperature of the object. However, it should be noted that the linearized equations consider a factor (A), for variations in the medium; this in turn, factors a control distance at which the measurements should be conducted. Thus, to correctly calibrate the device, a temperature vs. distance calibration must be implemented. The following image shows the radiant flux as a function of the wavelength for different temperature curves. It can be observed that for surfaces over 300 K, the radiation flux is in the range of the NIR spectrum [23].

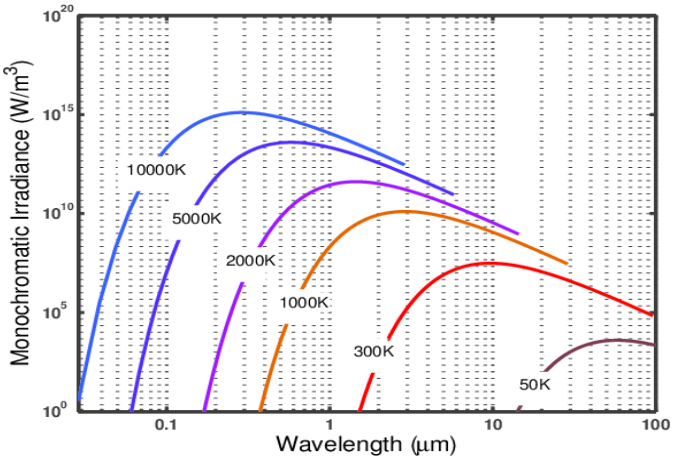


Figure 6: Plank function

One more important characteristic that must be addressed, is the transmittance of light of 2mm Acrylic or Plexiglas, which is the main component of the microfluidic systems used and the material encasing the fluid. Colorless acrylic or plexiglass allows the NIR spectrum to pass through, up to wavelengths of 1.1µm [20]. In [21] and [22] the specific transmission percentage of 2mm and 2.5mm acrylic substrate is shown as a function of the wavelength; this shows that up to wavelengths of approximately 1.6µm, the factor needed to calibrate for the infrared measurements is close to one, this is due to the transmission percentage being approximately 85%. The following image shows the transmission for 2mm acrylic substrate [21].

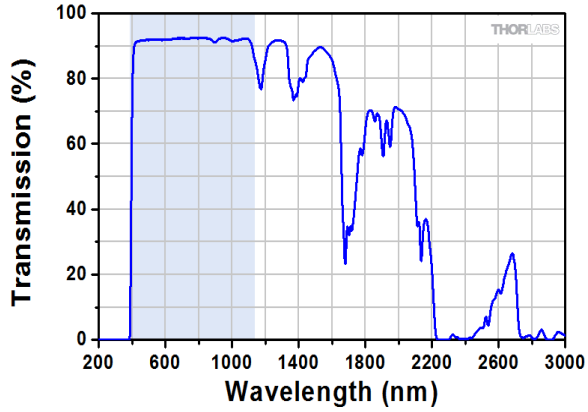


Figure 7: Transmission of 2mm Acrylic Substrate

However, for the remaining 1.4µm (to complete the NIR spectrum up to 3µm), the necessary calibration must consider a transmission factor to compensate for the low transmission percentages of the material.

Analysis of experiments for calibration

For the correct implementation of the thermal sensor in the new environment, it is necessary to conduct three calibrations. The first calibration serves to adjust the default temperature curve of the sensor to one that is previously known. This calibration provides a baseline for the distance vs. temperature curve since the temperature adjustment is conducted at a control distance. The second calibration conducted is the distance vs. temperature adjustment. This is done to create a linear regression equation that allows the user to know the temperature of the measured surface knowing the distance to the thermal sensor. This adjustment also serves to factor the variations that the medium can pose on the measured electromagnetic waves read by the thermal sensor. The final calibration to be conducted is done once the thermal sensor is correctly adjusted and the distance vs. temperature equation has been proven effective. This calibration places the sensor directly on top of the standard microfluidic system to factor the transmission percentage from the 2mm acrylic film.

To start the calibration, a heating plate is needed to create a heated surface that allows the sensor to correctly measure thermal variations. The use of thermometers, both submersible as well as infrared, is also needed to verify the calibration of any equipment that needs to be used for the calibration of the thermal sensor. Using the heated surface and placing the thermal sensor at a known distance, the sensor's default calibration is adjusted to the known values to create a linear regression curve. With the previous calibration as a baseline, a contraption designed to hold the

thermal camera at different heights is used to measure the heating plate. The contraption is to allow at least four different heights, with the highest one not being more than 15cm, as most infrared thermometers (including the infrared thermometer used for the calibration) state this as the longest distance for an accurate reading. This way, the distance vs temperature equation can be obtained from a linear regression approximation. Using this information, a calibrator circuit can be designed to replicate the process and allow for a fast and accurate adjustment of the thermal imaging done by the sensor, considering the variations in the medium. Finally, the contraption, which serves as the final casing, which will hold the thermal camera, as well as the microfluidic system and the calibrator circuit, is used to factor the transmission of the acrylic film. The methodology for the calibration process can be observed below:

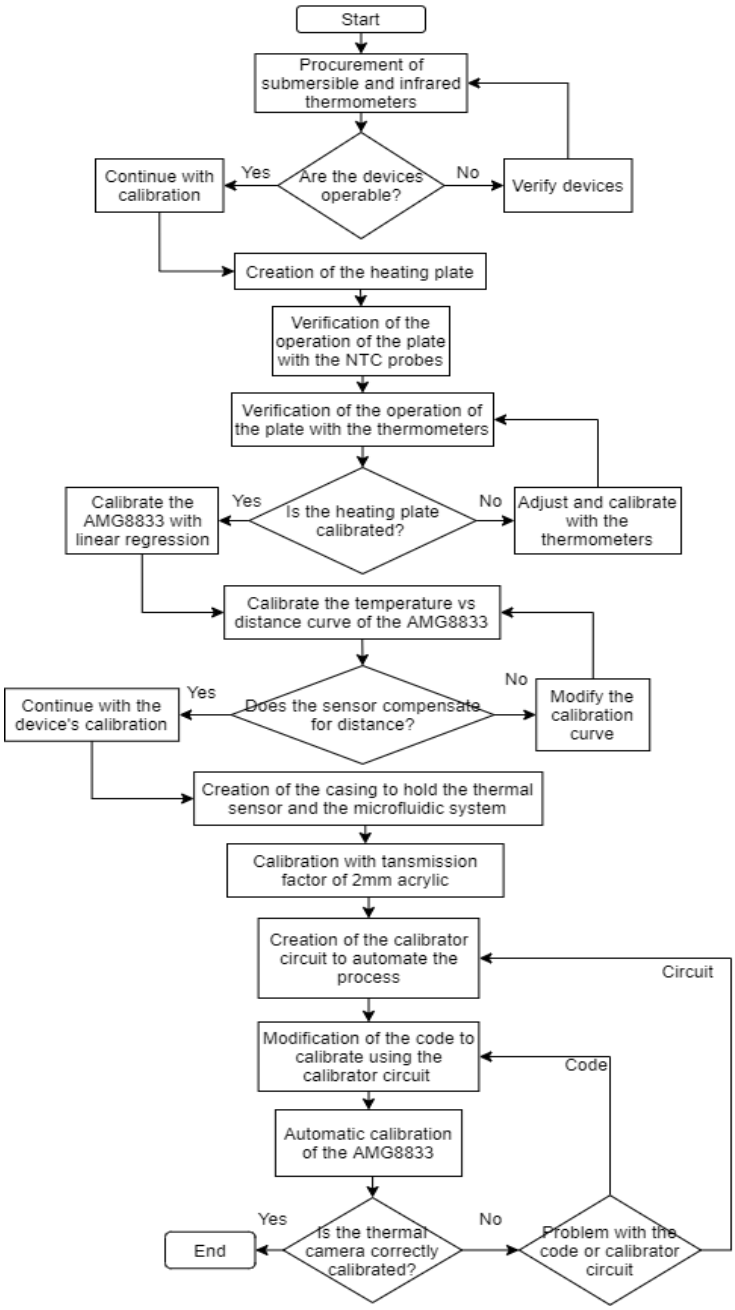


Figure 8: Flow diagram for the calibration methodology

Heating plate for the calibration of thermal cameras for microfluidic systems

Due to the conditions in which the calibration and implementation of this thesis is conducted, it is necessary to create a heating plate that can accurately create a heated surface. It should be noted that a commercial heating plate can be used to replicate the calibration process of the thermal camera.

Heating plate methodology

The methodology used for the development of the heating plate consists of an ON-OFF controller with feedback that allows to adjust the temperature of an electric stove. This implementation makes use of two digital thermostats that control the input signal of two NE555 astable oscillators, which in turn, control the duty cycle of the electric stove. The implementation also possesses an AC voltage regulator to control the power dissipated by the resistance in the stove.

The first implementation was done using a single XH-W1209 digital thermostat. However, it was observed that the temperature curve had an exponential behavior that easily surpassed the desired temperature. Thus, an astable oscillator controlled by the XH-W1209 was integrated to flatten the heating curve. This process improved the heating curve; however, significant temperature changes could still be observed. This final implementation makes use of a second digital thermostat which creates a feedback loop taking into account the temperature of the thermal bath. The device's architecture can be observed in Annex 6: Heating plate diagram.

Materials

The main components for this implementation are as follows:

- Electric stove
- Digital submersible thermometer
- XH-W1209 digital thermostat (2)
- 10K NTC 0.5% (2, included with the XH-W1209 digital thermostat)
- AC-DC 12V converter

Components

XH-W1209

The digital thermostat XH-W1209 is a programable controller which allows to activate a relay when the temperature detected is higher or lower than a desired temperature. This is done through a 10K NTC, which is included with the programable controller; the thermostat also needs an operating voltage of 12V.

When turned on, it shows the current temperature detected by the 10K NTC. When the SET button (left button) is pressed , the LCD display starts flickering and shows the desired temperature. Using the + and – buttons (center and right, respectively), it is possible to change the temperature. By holding the SET button, the controller allows to cycle through the different configurations using the + and – buttons. Pressing SET on one of those configurations allows to access the value and modify

it using the + and – buttons. Waiting two seconds on the settings screen, cycles back to the current temperature being sensed by the thermostat; [24] gives a good overview of the programable controller. The list of the configurations is as follows:

> P0 H/C (C default, allows to activate/deactivate the relay when the sensed temperature is below the desired one)
> P1 Backlash 0.1 15 (2 default, allows to adjust the relay trigger to: desired temperature + Backlash)
> P2 Upper limit (110 default, Adjusts the maximum sensing temperature)
> P3 Lower limit (-50 default, Adjusts the minimum sensing temperature)
> P4 Correction -7.0 – 7.0 (0.0 default, allows to correct the temperature measurements)
> P5 Delay trigger 0 – 10 (0 default, adjusts the time for the relay trigger [minutes])
> P6 Alarm (OFF default, sets a maximum temperature for the operation of the controller)
> *Pressing + and -, for a few seconds sets the values back to default.

One of the XH-W1209 must be adjusted to P0 H, and the other to P0 C. Through this, one relay is triggered below the desired temperature, while the other one serves as a feedback loop when the temperature is reached. The values for P4 are adjusted using the submersible thermometer (+0.6°C).

H-oscillator implementation

This oscillator is an astable NE555 timer, which regulates the flow of current to the electric stove using relays. It should be noted that this oscillator works with the LOW duty cycle of the timer, meaning that for resistance and capacitor values that provide a 70% duty cycle, the duty cycle used for the flow of current is 30%. The H-oscillator configuration and schematic can be found in Annex 7: H-Oscillator schematic.

In the configuration implemented, the controller has a frequency of 0.153Hz through the K1 relay with a duty cycle of 71.43%, which allows the current to flow to the electric stove for 1.87 seconds. It is recommended that the value of the Period trimmer is not modified, as a duty cycle of less than 500 milliseconds could damage the electric stove. Modifying the duty cycle trimmer, allows to change the controller's duty cycle without modifying the LOW time of the controller. Thus, the stove always receives current for 1.87 seconds. The K2 relay is active whenever the H thermostat triggers its relay.

C-oscillator implementation

This oscillator's implementation is virtually the same as that of the H-oscillator. However, the design values provide a frequency of 0.534Hz, or 1.87 seconds, which is the LOW time for the H-oscillator. The duty cycle is 54%, which helps to cut the current flow to the stove by about 50%. The C-oscillator configuration and schematic can be found in Annex 9: C-oscillator schematic (the trimmer values are to be 100%).

If the period trimmer in the H-oscillator is to be modified, the operation frequency of the C-oscillator must be adjusted using the period trimmer. Doing this, the duty cycle of the oscillator stays at approximately 50%, with a maximum change of about 10%.

The system's feedback is done by the C-oscillator, which modifies the duty cycle of the H-oscillator once the temperature has reached the desired value. This is done to flatten the heating curve of the electric stove.

AC voltage regulator

The AC regulator makes use of a BT136 triac as well as a linear potentiometer to clip the output voltage. When the potentiometer is at 100% of its value, the stove receives all the current from the output of the oscillators. It should be noted that having the potentiometer at approximately 75% of its value provides a 50% decrease in the output voltage. The AC voltage regulator schematic can be observed in Annex 12: AC voltage regulator schematic.

Schematic

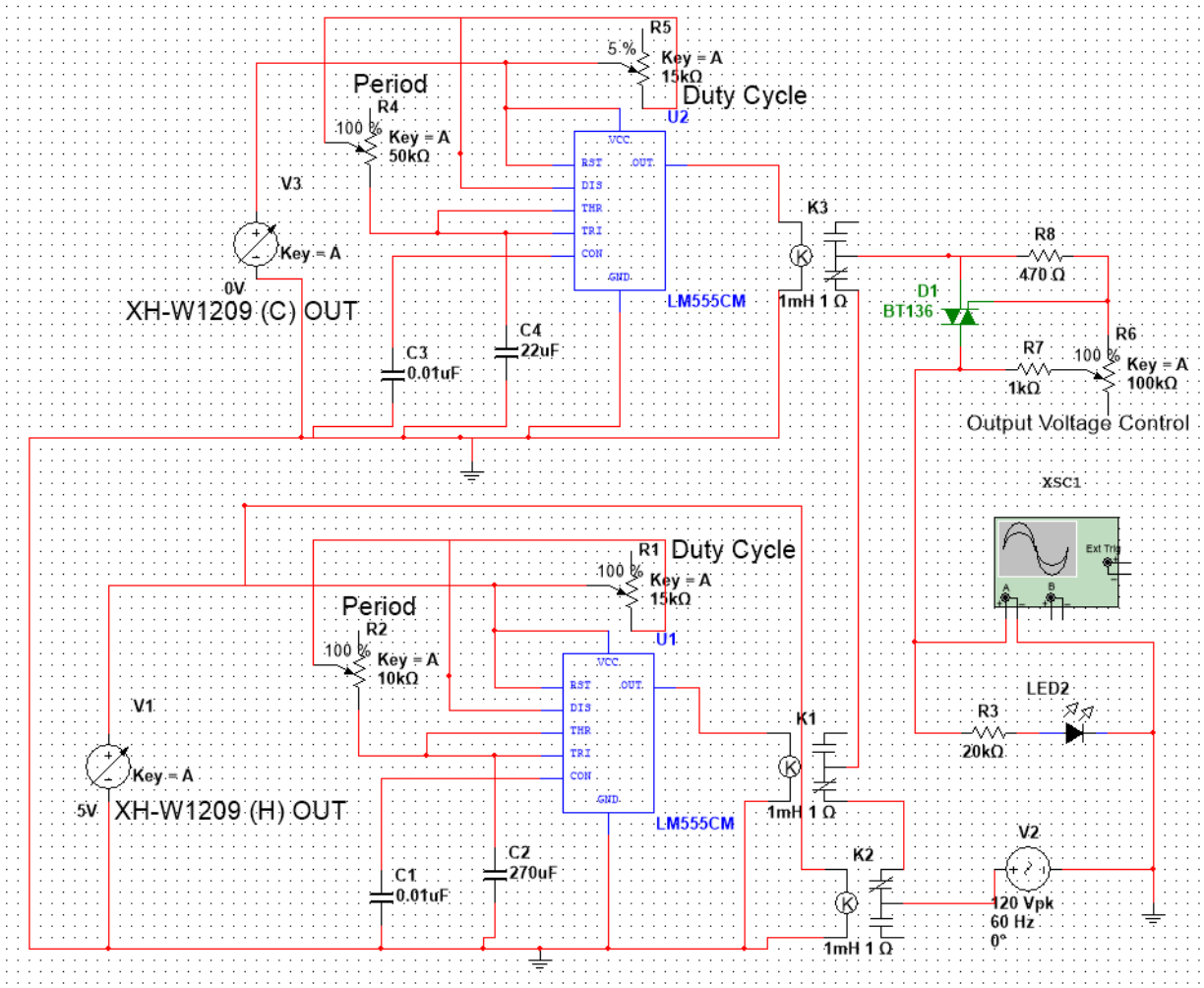


Figure 9: Heating plate diagram

The above schematic shows the integration of all the designed components to create an ON-OFF controller for the electric plate.

Implementation

For the device's implementation, the AC-DC 12V converter is integrated into the AC_IN of the electric stove, to simplify the integration of the different components. Although, the AC-DC 12V converter can be connected separately, the designed PCB has the connections for an integrated system.

Controller test

Using the following configuration, a simple test can be conducted to verify the operation of the device (a more detailed PCB schematic can be observed in Annex 13: Heating plate PCB):

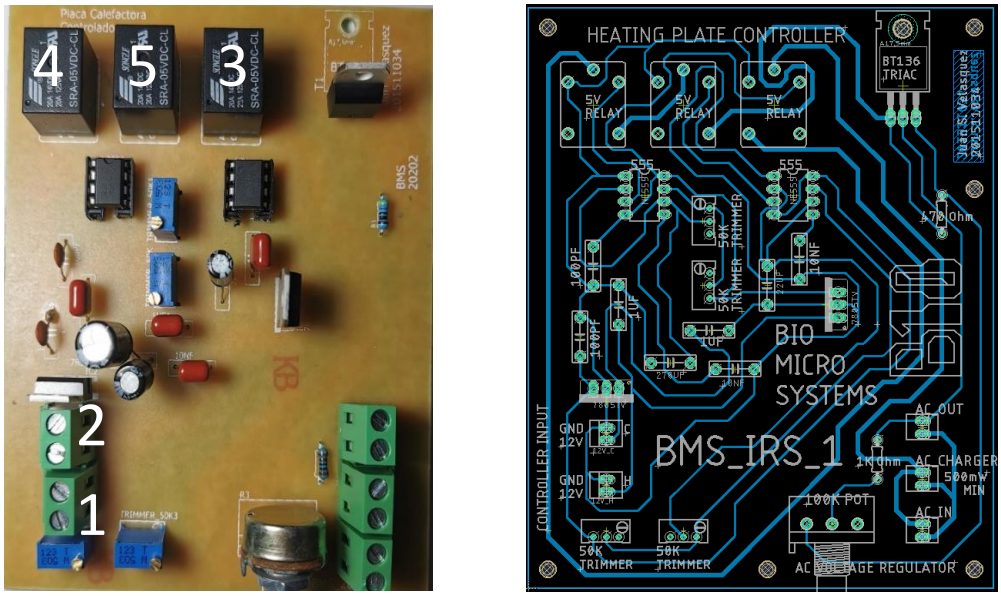


Figure 10: (a) Heating plate operation test*, (b) Heating plate PCB

*The screen print was added after the first PCB was printed.

The relays' correct operation can be observed in [25], in which the input (1) is responsible for the trigger of the relays (3) and (4). The first relay of the schematic (3) is powered when the input (1) is connected, while the second relay (4) is triggered every 1.87s – 4.7s. The input (2) is responsible for the operation of relay (5) [45]. This relay must be triggered every 900ms – 1s. [46] shows the controller's operation with both inputs at 12V.

Temperature measurement

The following image shows the configuration needed to activate the controller circuit with the signals from the digital thermostats.

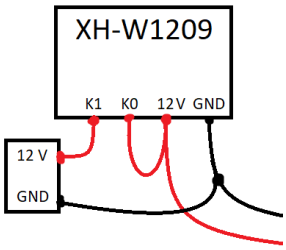


Figure 11: Simplified XH-W1209 connection diagram

For the feedback loop to function properly using the XH-W1209, the 10K NTC of the H thermostat Is placed on the electric stove, while the other NTC must be placed in the thermal bath. This configuration allows to begin the feedback process only when the thermal bath's temperature is close to the desired temperature. The temperature of the H thermostat is set to the desired temperature of the thermal bath, while the set temperature of the C thermostat is set 5°C lower to that of the H thermostat. The AC voltage regulator is kept at 100%.

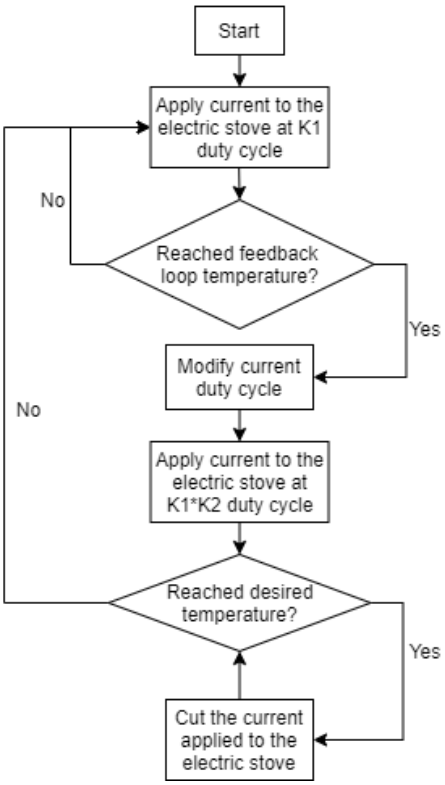


Figure 12: Heating plate operation methodology

Calibration

For the device's calibration, the device is connected, and the thermostats are set to the desired temperature (in this case, the H -thermostat was set to 35°C, while the C-thermostat to 30°C for the feedback loop). The setting P0 for both thermostats should be verified. The 10K NTC of the H-thermostat is placed on the stove, while the NTC of the C-thermostat is placed in the thermal bath.

Once the temperature of the electric stove has stabilized, the submersible thermometer can be used to adjust the correction values of the digital C-thermostat. It is recommended to set the

temperature of the H-thermostat between 35-40°C because most commercial thermometers have the lowest measurement uncertainties at this range; thus, the calibration will be more precise. Making use of the P4 setting in the thermostat, the temperature adjustment can be used to calibrate the device. This process can be repeated for a few values inside the stated range, to accurately calibrate the device (In this case, the P4 setting had to be adjusted 0.6°C).

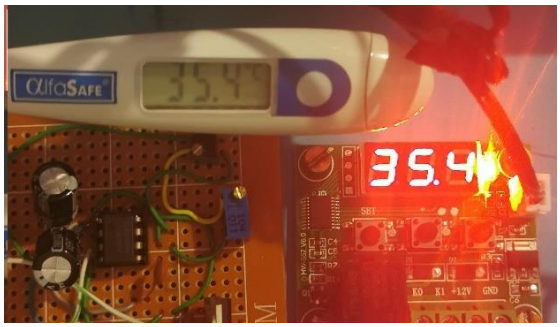


Figure 13: Heating plate calibration

In case any calibration is necessary for the H-thermostat, the infrared thermometer should be used. The thermometer should be placed at the maximum distance recommended (15cm for most thermometers) for an accurate reading at an angle of approximately 45°. The verification of the calibration of the thermometers used can be observed below:

Thermometer Data

SUNPHOR BZ-R6		
	Min	Max
Measurement range for minimum uncertainty °C	36	39
uncertainty °C	0,2	0,2
Measurement distance (cm)	5	15

AlfaSafe DT-11F		
	Min	Max
Measurement range for minimum uncertainty °C	35,5	42
uncertainty °C	0,1	0,1

Measurement Data °C

	BZ-R6*	DF-11F**
Subject1	36	36,3
Subject2	36,5	36,6
Subject3	36,4	36,4

*Measurements done at 5cm
**All measurements done in the armpit

Table 1: Thermometer information [26][27]

Once the calibration of the XH-W1209 thermostats is implemented for 35°C, the value for the duty cycle trimmer in the H-controller is modified to observe the behavior of the heating curve at different duty cycle percentages. The value for the AC voltage regulator has been kept at 100%. The different temperature curves for the adjusted duty cycle percentages can be observed in the image below.

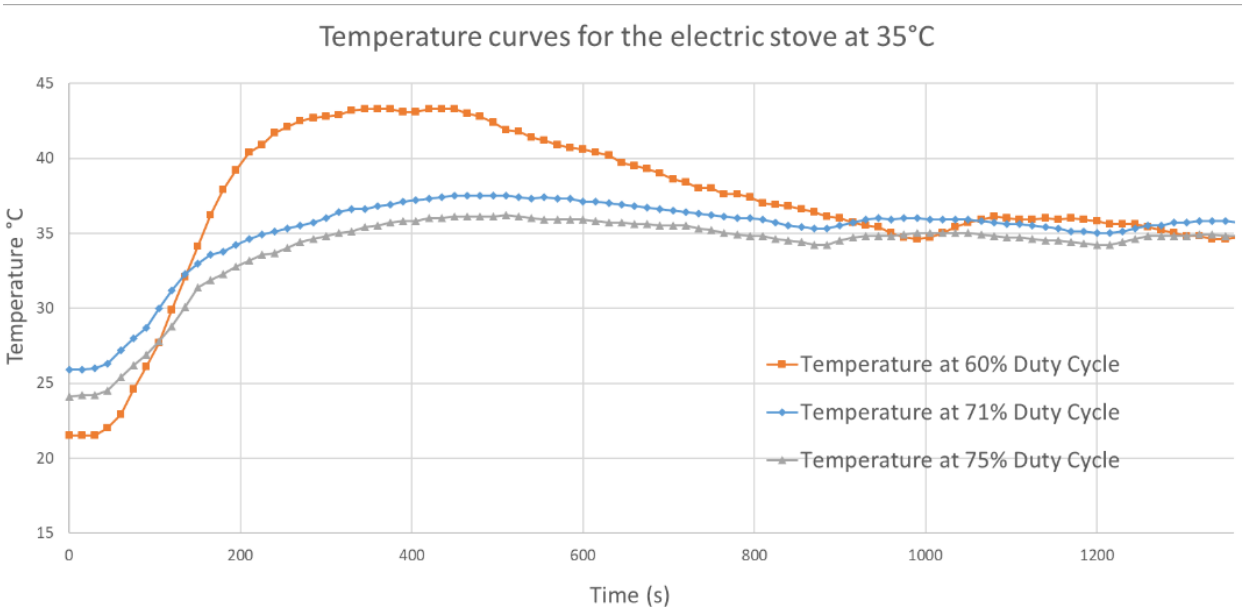


Figure 14: Temperature curves for 35°C*

*The values for the duty cycle percentages are obtained from measuring the trigger time from the controller's relays. The temperature curves for 60%, 71% and 75% duty cycle can be found in [28], [29] and [30], respectively.

After adjusting the duty cycle trimmer to a value of 75%, the curves for different temperatures are analyzed to observe how the specific values for the 35°C calibration affect the device at different temperatures. This is done to create a calibration curve and obtain a larger range of temperatures from the 35°C calibration. Using the 35°C temperature curve at a duty cycle of 75%, the device's base uncertainty can be obtained (±1.2°C, standard deviation = 0.588°C); this value is then used in the images 14-16.

Temperature curves

Using the configuration and values for 35°C, the device's heating curves for temperatures for 38°C, 44°C and 50°C are obtained. It should be noted that the H-thermostat temperature is the one stated, while the C-thermostat temperature is 5°C lower as for the operation of the feedback loop. The temperature curves obtained are as follow.

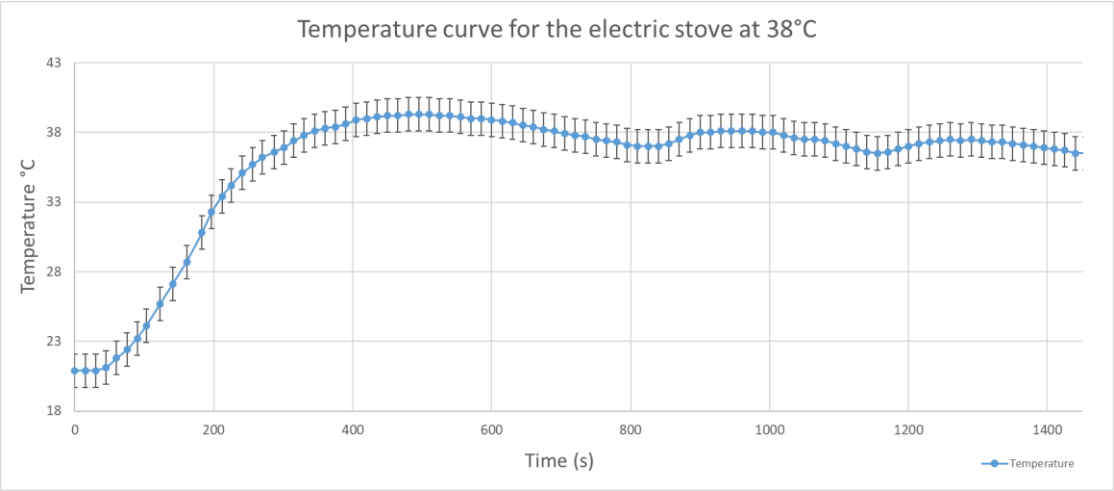


Figure 15: Temperature curve for 38°C [31]

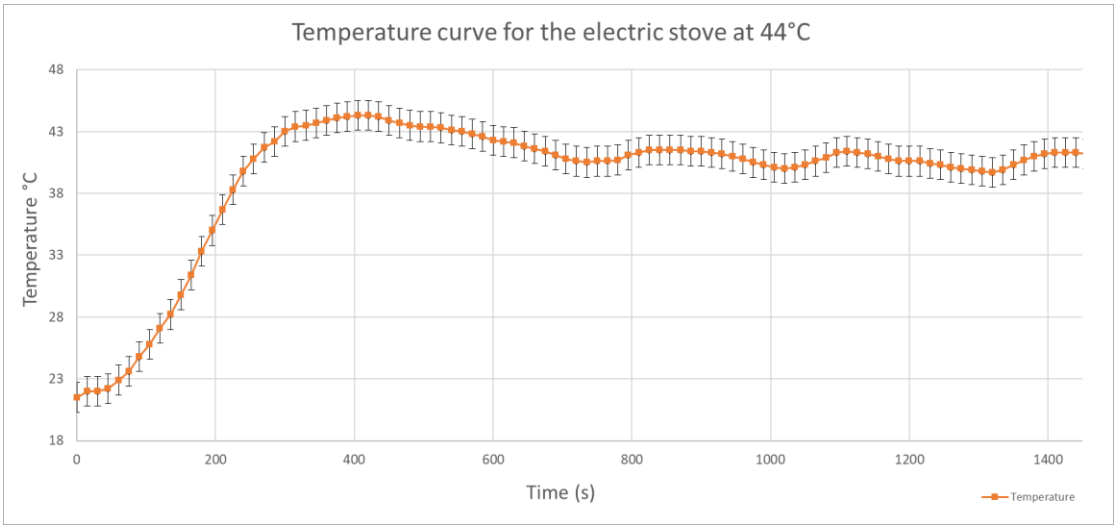


Figure 16: Temperature curve for 44°C [32]

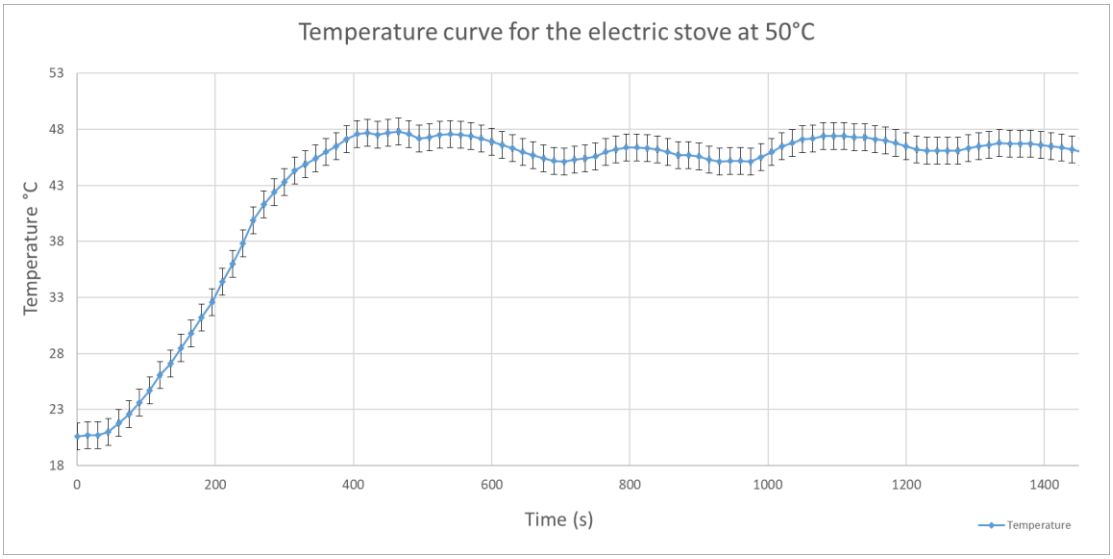


Figure 17: Temperature curve for 50°C [33]

Linear regression

Working out an average of the temperature values from minute 5 to the end of the measurement time, the real temperature from the heating plate can be obtained. The relevant information from the curves is shown in the table below.

Desired temperature (H-controller)	Feedback temperature (C-controller)	Real temperature	Standard deviation
35°C	30°C	35,089°C	0.588°C
38°C	33°C	37,809°C	0.820°C
44°C	39°C	41,528°C	1.287°C
50°C	45°C	46,389°C	0.777°C

Table 2: Temperature curve data

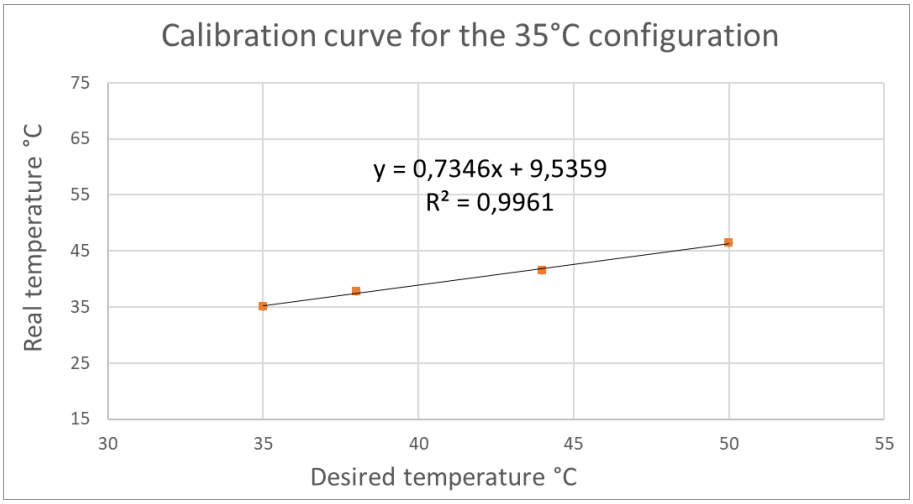


Figure 18: Heating plate linear regression

To verify the linear regression equation obtained through the temperature curves, the thermostats are set to a different temperature. The test temperature is set to 50°C, which is then applied to the linear regression to obtain the temperatures for both thermostats.

$$\frac{50°C \ - \ 9.5359}{0.7346} = H\ thermostat\ temperature;\ HTT = \ 55.083°C \ \simeq 55.1°C$$

This in turn gives a temperature of 50.1°C for the C-thermostat, for the feedback loop to function properly. Once the temperatures are set, the temperature curve of the heating plate is recorded; it can be observed in the image below.

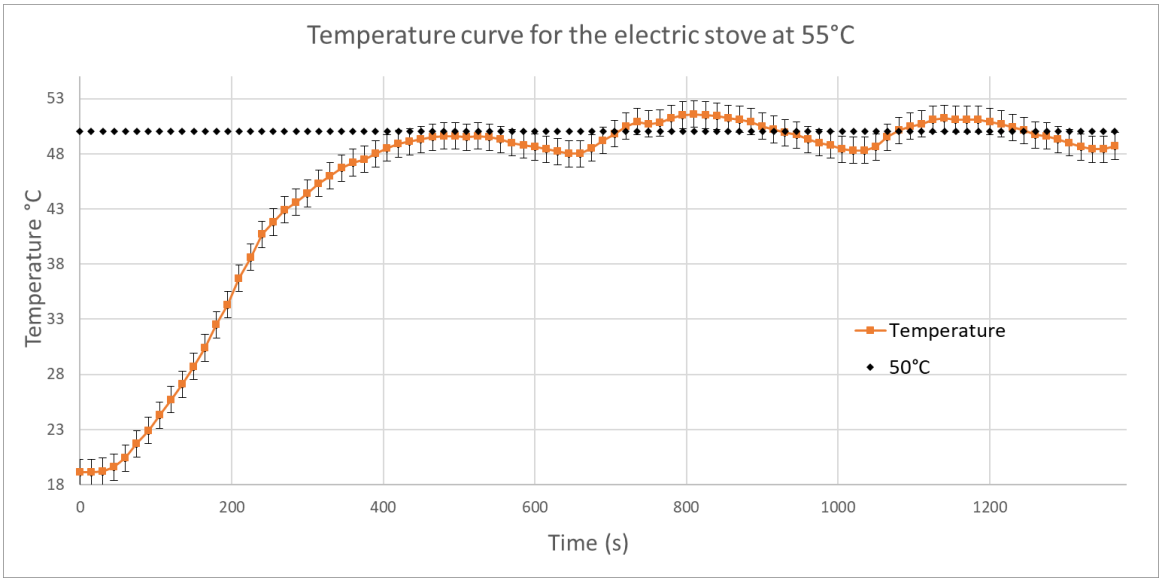


Figure 19: Verification of the linear regression

It can be observed that the linear regression does in fact provide the desired thermal bath temperature for the set thermostat values. The heating curve recorded for 55°C can be seen in [34]. The curves also have the 1.2°C error bars obtained from the standard deviation, from the configuration for 35°C. Using the same methodology for table 2, the real temperature from the 55°C curve is 49.494°C, which is in fact within the uncertainty values previously calculated. The standard deviation for figure 19 is also calculated, and comes out to be 1.33°C.

A standard deviation curve is also implemented. However, due to the controller being an ON-OFF system, there are variations in the temperatures' curves, which create significant uncertainty in the standard deviation curve. The curve can be observed in Annex 14: Heating plate standard deviation curve.

$$Standard\ deviation = 0{,}0353x - 0{,}5228;\ R^2 = 0.4032$$

Recommendations

For the first operation, the C-thermostat's temperature should be set 6°C, instead of 5°C, below that of the desired temperature. This is due to the time it takes the capacitors to be charged, which can impact the response time of the feedback loop and thus, the heating curve of the electric stove. For better measurements, the NTCs should be fixed in place to always measure the same spot throughout the electric stove's operation. It should be noted that all the heating curved were done using a thermal bath of approximately 100mL.

Final observations

The implemented device is then compared to some of the commercial heating plates available to find if the device is capable of being operated in a laboratory environment. The following shows a general comparison between the device and some commercial heating plates.

Brand/product	Uncertainty	Temperature range
CorningPC-400D	±5°C	5°C – 550°C
SCILOGEX SCI550-Pro	±1°C	20°C – 550°C
ONILAB 340C	±0.5°C	20°C – 280°C
FOUR E'S MI0102003	±1°C	20°C – 280°C
Device at 35°C*	±1.2°C	N/A
Device at 55°C*	±2.6°C	N/A

Table 3: Heating plate comparison

It should be noted that the temperature range for the implemented device is not available as it is intended to be used at very specific temperatures for the calibration of the thermal sensor. At the device's calibration point, it can be observed that the uncertainty is on par with those from commercial brands, which allows it to be used in a laboratory environment. However, for different temperatures, the uncertainty might increase, in which case a different calibration point should be implemented.

Although both controllers possess trimmers to control the duty cycle as well as their frequency, only the duty cycle trimmer in the H-controller was modified for the configuration at 35°C. It should be noted that modifying the period trimmers in conjunction with the duty cycle trimmers, could provide a better heating curve that allows for lower uncertainty values when the linear regression curve is obtained. Moreover, although the device possesses an AC voltage regulator, it was always kept at 100% output. Modifying this component could also allow for different heating curves for different applications.

Device calibration

Once the heating plate has been calibrated, the linear regression curve for the thermal camera can be implemented. This is done using the manual point calibration previously implemented, in which the four center pixels of the thermal camera are used to obtain an average temperature (if the temperature calibration is kept as is, the equivalent curve would be y = 0.25x; an average value of 25°C appears as a 100 to take into account the 4 pixels of the thermal sensor). The temperature is then corrected by having a matrix with two arrays: the sensor measured temperature and the actual temperature, which is later used to create a linear regression.

It should be noted that the linear regression method is done by calling the function linregress() from the scipy library, which takes both arrays and gives the m and b values from the y = mx + b equation. It should also be noted that if more than 2 X values have the same Y value, the linregress function is not able to properly calculate the parameters of the equation and returns values of NaN (At which point, the file should be deleted and the calibration should be done again). The flow diagram of the calibration used is as follows.

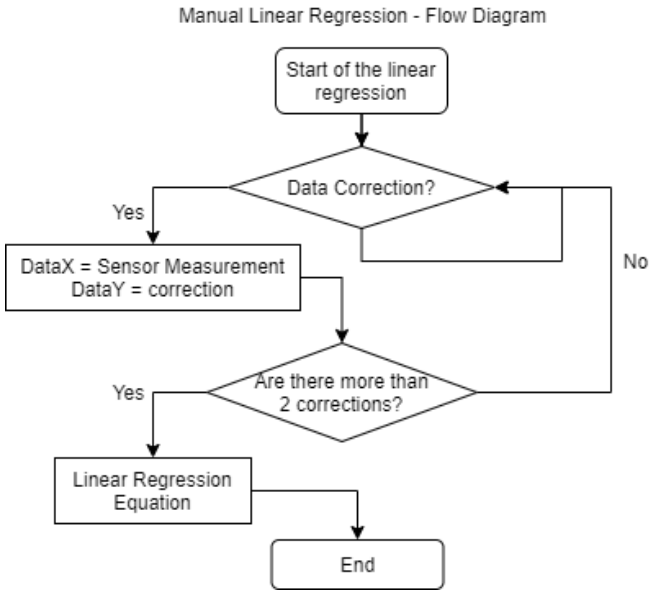


Figure 20: Software calibration flow diagram

Thermal bath calibration

Using the manual point calibration previously implemented, the linear regression curve is then implemented at 19mm from the surface of the thermal bath. The values were corrected for the 30°C to 50°C range using the heating plate as a reference, after which the equation was obtained using the linregress function. The thermal camera was kept at the following position for the whole calibration process; that is while the heating plate was operational and covering the range of 30°C to 50°C.

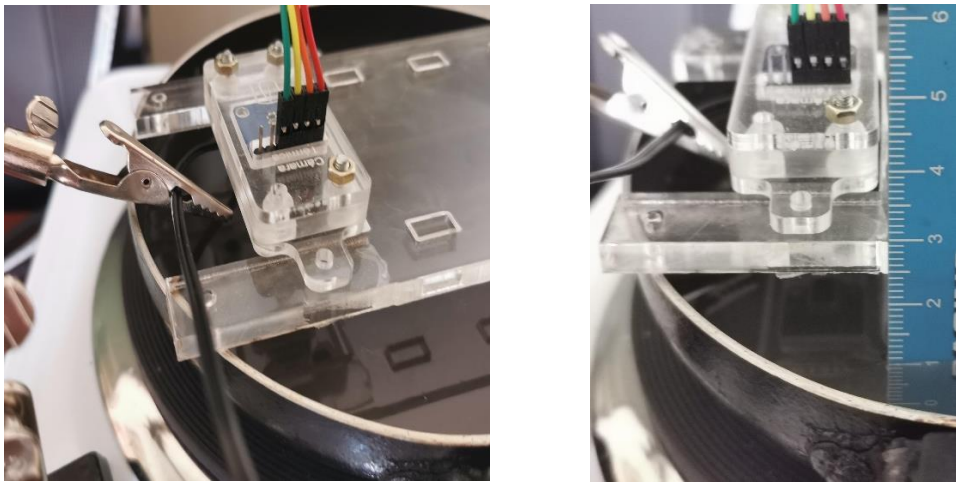


Figure 21: Thermal bath calibration 19mm

This process was repeated ten times to obtain an average calibration curve that could eliminate outliers in the calibration process. Using the linregress function and dividing the X values by 4 to account for the average temperature measurement method previously stated, the graph obtained is as follows. Each of the linear regression were extrapolated to a range from 25°C to 55°C; the standard deviation of the data can also be observed using the secondary axis. After which, the

average calibration equation is obtained and is shown in the following curve. The average calibration curve with the sensor's averaging method can be observed in Annex 15: Thermal bath 19mm.

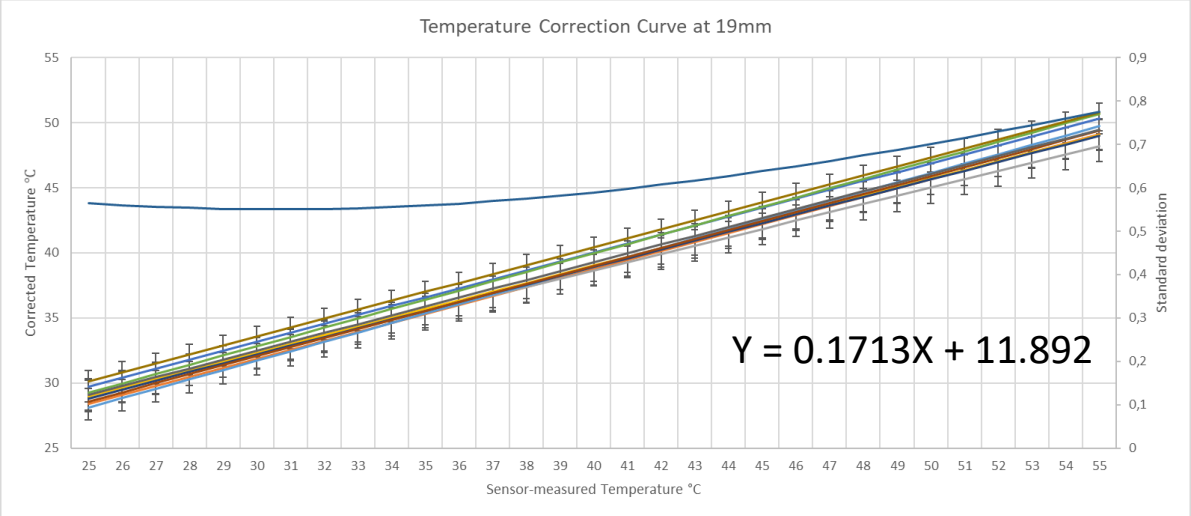


Figure 22: Temperature correction curve at 19mm

The same process was also done at a second distance to observe how the thermal sensor's measurements change as a function of its distance to the measured surface. This information is needed since the sensor gets the temperature readings using the refraction of its lasers, thus, variations in the environment can drastically change the temperature readings obtained. The reading was done at 25mm, using a contraption made of 2mm acrylic which would later allow the sensor to be placed at different heights and thus make the calibration process replicable. The sensor's placement is shown below.

Figure 23: Thermal bath calibration 25mm

The process implemented for the 19mm thermal bath calibration was repeated, and the following curve was obtained. The average calibration curve with the sensor's averaging method can be observed in Annex 16: Thermal bath 25mm.

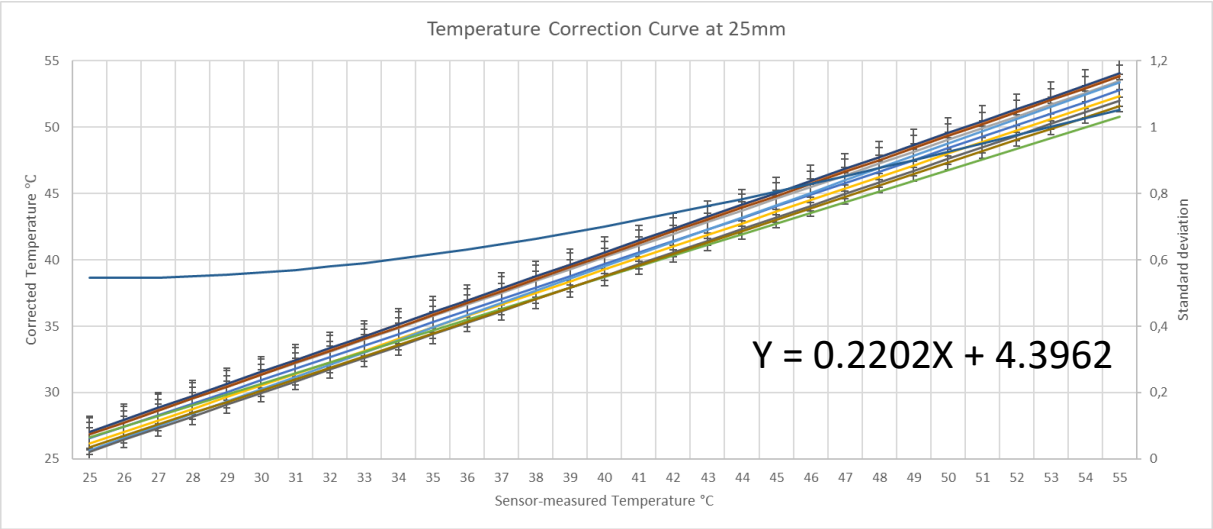


Figure 24: Temperature correction curve at 25mm

From the standard deviation of the data, it can be observed that there are more variations as the distance increases. Also, the variations in the m coefficient of the mx + b equation show that the temperature read by the sensor was much higher than the real temperature the closer it got to the heated surface. Thus, the distance – temperature calibration is carried out to make sure the camera can properly measure the heated surface. The calibration process is repeated for 57mm, 92mm and 129mm. It should be noted that the methodology set a maximum distance of 150mm as that is the maximum measurement distance for infrared thermometers. This can also be observed in Table 1: thermometer information. It should also be noted that since the four pixels used for the average temperature readings are the ones in the center, for the distance used it is not necessary to consider the geometry of the lens.

The distance vs temperature curve serves to find the calibration needed for a smaller distance, such as the 2mm acrylic film, which is hard to place along the surface of the thermal bath. It also helps find the temperature offset as a function of the environmental variations for the proper temperature measurement. The curve obtained is as follows.

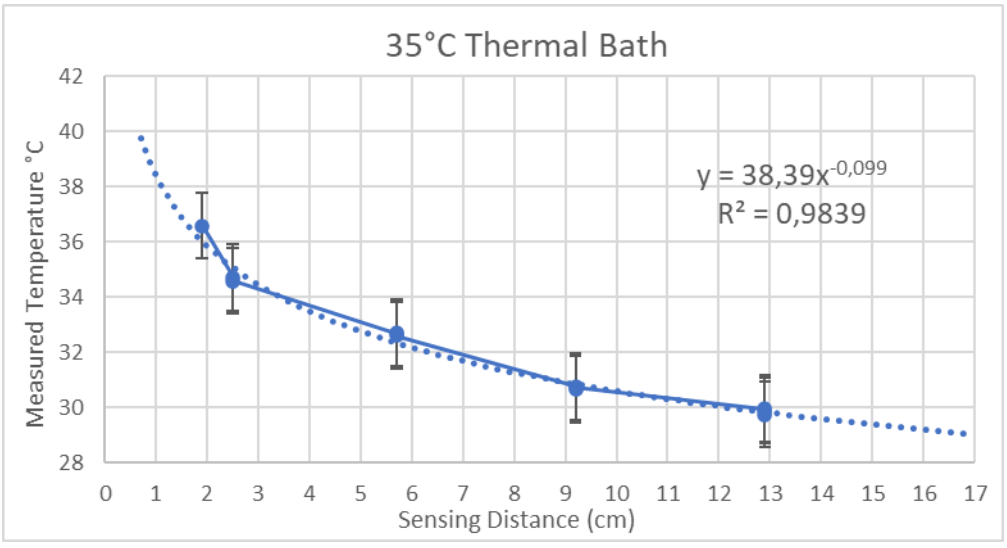


Figure 25: 35°C distance calibration

The curve shows that for 2mm acrylic, the temperature measured by the sensor would be approximately 45°C, which is 10°C above the real temperature of the heated surface. Also, the parameters of the linear regression equations obtained for the calibration process for 19mm and 25mm have a large degree of variability, which shows that the calibration process conducted creates a lot of variations in the environment which in turn affect the thermal camera's thermal measurements.

In [35], it is stated that infrared thermometers are often susceptible to significant changes in the temperature readings, in the presence of frost, moisture, dust, fog, smoke, etc. Thus, the use of the thermal bath for the calibration process does not allow for an accurate reading and a proper distance – temperature calibration. Therefore, the calibration process is then repeated using the same method, however, the heated surface used is to be modified from the thermal bath, to the heating plate itself. Although the variations in the heating plate will now be larger, due to the new configuration of the feedback loop, the heating plate always shows the current temperature reading, thus making it a viable option for the calibration of the thermal camera.

It should be noted that the calibration performed is valid for the AMG8833 thermal sensor in environments in which the camera is to be placed for long periods of time in high humidity conditions, or for the temperature readings of liquid surfaces.

Heating plate calibration

For the new calibration process, the heating plate is used without the thermal bath. The NTC 10K probes are placed on the heating plate to keep a good measurement of the current temperature. It should also be noted that the contraption made of 2mm acrylic bent during the thermal bath calibration previously conducted. Thus, it was modified to a more sturdy 5mm acrylic contraption.

The previous calibration process was replicated for 5mm, placing the acrylic base directly on top of the electric stove. It should also be noted that to avoid further variations in the environment caused by the fluctuation of hot air under the thermal camera, the thermal sensor was not kept in the shown position for the length of the calibration process. Rather, once the temperatures of the heating plate had stabilized at the desired values, the thermal camera was placed for no more than two seconds in the desired position, at which time the thermal reading was measured (this process was replicated for the different distances). The purpose of this calibration process is to make sure that under the right circumstances, the thermal camera does not need a linear regression curve to show the correct temperature readings.

Figure 26: Heating plate calibration 5mm

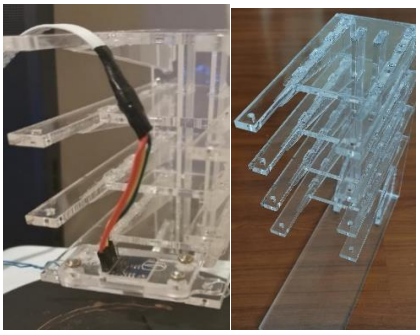


Figure 27: Heating plate calibration contraption

The data obtained from this calibration is shown below. The average calibration curve with the sensor's averaging method can be observed in Annex 17: Heating plate 5mm.

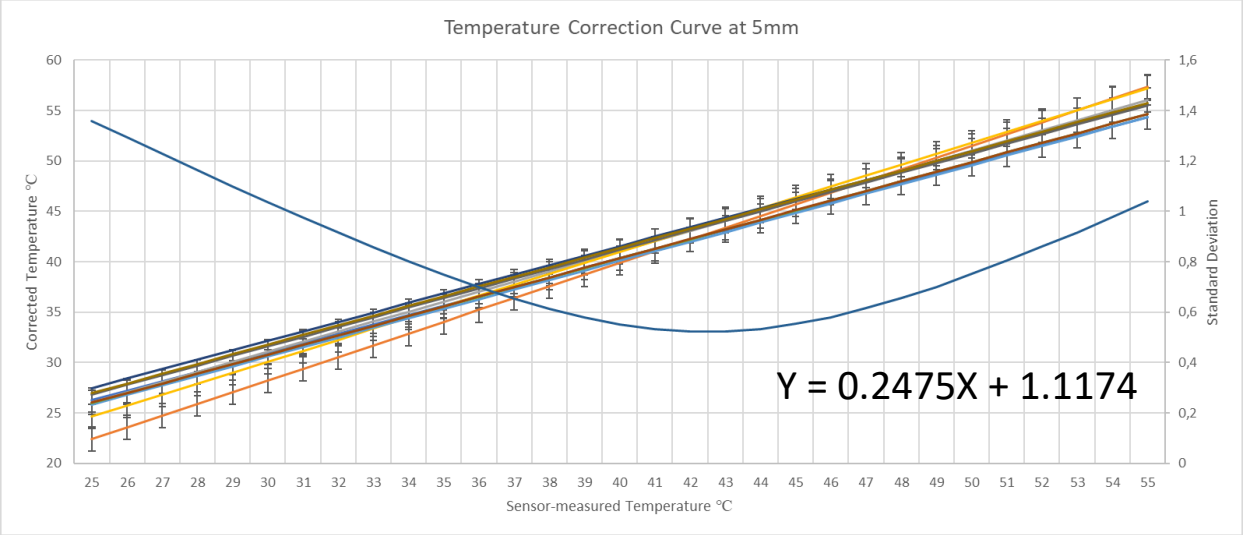


Figure 28: Temperature correction curve at 5mm

It can be observed that by not keeping the sensor in the calibration distance during the process, the readings are much clearer, as the linear regression equation resembles more the sensor's default readings. It should also be noted that the measurement distance is about 25% of the first thermal bath reading, which showed an increase of about 1.5°C above the control temperature. Since the new calibration method seems to have taken care of a significant amount of the environmental variations, the same procedure is then repeated for distance of 10mm and 33mm. To make sure that all the thermal measurements were taken in the same way, the new 5mm acrylic contraption was used to fix the thermal sensor to the desired distances. The following curves were obtained. The average calibration curves with the sensor's averaging method can be observed in Annex 18: Heating plate 10mm and Annex 19: Heating plate 33mm, respectively.

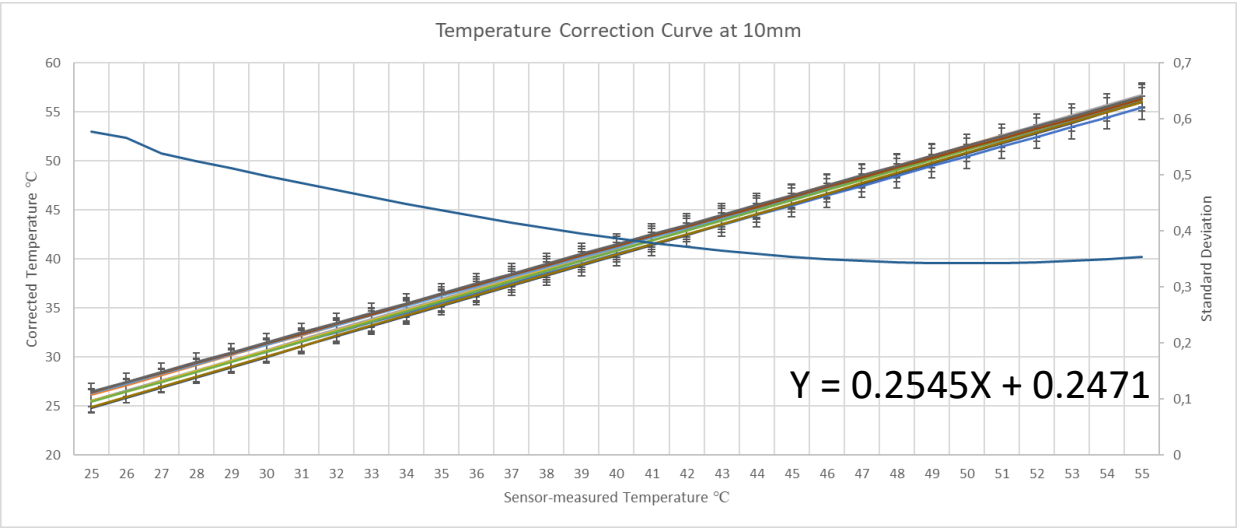


Figure 29: Temperature correction curve at 10mm

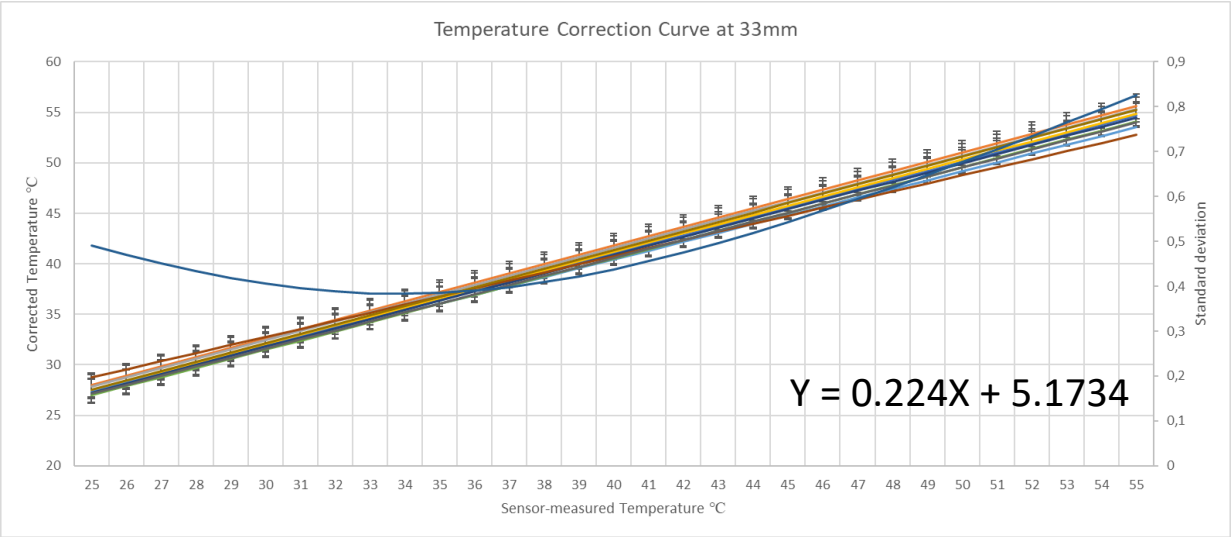


Figure 30: Temperature correction curve at 33mm

Although some variations may be observed, the data is more coherent than that obtained using the thermal bath as the heated surface. Below is a comparison and summary of the data obtained for the calibration method implemented. It should be noted that although the thermal camera was not kept in the calibration position for the duration of the process, the heating plate was not moved. Thus, the heated air could in fact affect the temperature readings of the thermal sensor.

Temperature	5mm	10mm	33mm
30°C	30.8174°C	30.7871°C	32.0534°C
35°C	35.7674°C	35.8771°C	36.5334°C
40°C	40.7174°C	40.9671°C	41.0134°C

Table 4: Linear regression temperature comparison

It can be observed that the maximum variation between the data measured and the control temperature is ±2.05°C, which shows that even taking measures to decrease the amount of temperature variations in the thermal camera's environment, a distance vs temperature calibration is needed. However, for the distances used in the thermal readings of the standard microfluidic system, it can be observed that the linear regression equations are very close to that of the default calibration of the device. Which is to say, that if the environment in which the camera is used allows for the proper manipulation of the thermal sensor to decrease the temperature variations and the distance to the heated surface is in the 5mm to 10mm range, no further calibration will be needed.

However, for longer distances, the thermal camera does need a calibration to have precise and accurate readings regarding the heated surface. Thus, this type of calibration is useful if the camera can be removed and there can be minimal temperature variations in the thermal camera's environment. It should be noted that in the final casing the camera is not to be continuously removed from the casing before a thermal reading is performed, thus there has to be a correction which will take into account some degree of thermal variations. Also, since the final casing of the thermal sensor and the microfluidic system is at 2mm, there needs to be a distance vs. temperature calibration to make sure that the thermal variations of keeping the camera still does not significantly affect the thermal readings. To create the new distance vs. temperature calibration curve, the camera is kept at the calibrating position during the calibration process, although the heating plate

is kept as the thermal surface for the readings. Since there can be some temperature variations, the calibration process is to be done at different temperatures and repeated at the different heights to make sure that an average calibration curve can be obtained. Although there can be variations, the overall temperature change based on distance should be reliable enough as to have a distance coefficient for the final casing of the thermal sensor.

Heating plate distance calibration

For the distance vs temperature calibration using the heating plate as the surface, each thermal reading was performed five times for each of the distances, as to have an average curve that is reliable. The purpose of this calibration is to later create a prediction curve that should allow to have an idea of the temperatures that the thermal sensor will measure when placed in the final casing. The distance curves are repeated for different temperatures to observe how the thermal sensor's readings change as a function of the heated surface's temperature and its distance to it. The design of the 5mm acrylic contraption can be found in Annex 20: 5mm acrylic distance contraption design, in 18:25 ratio.

The first calibration curves are set up having the temperature of the heating plate at 37°C. The average distance curve obtained is as follows. The graph with all the values can be observed in Annex 21: 37°C distance calibration curve.

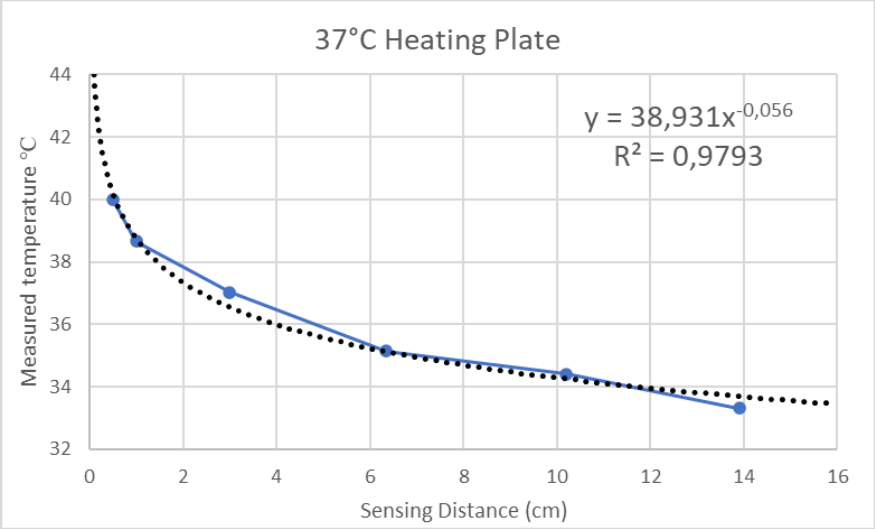


Figure 31: 37°C average distance calibration curve

Although the overall tendency of the data obtained resembles a linear regression, the chosen equation to model the behavior of the calibration was set as a power regression. This is due to the behavior of the data obtained using the thermal bath as the heated surface. A second distance curve must be implemented using the heated surface at a different temperature to figure out how the thermal variations as a function of distance change. Taking into account equation (2), to have an increase in the absolute temperature of the surface, both its radiant temperature and radiation rate into the environment must increase. Thus, an increase in the heated surface's temperature, signifies an increase in thermal variations which are then measured by the sensor. The second distance calibration curve was produced using a heated surface at 52°C and can be observed below. The same procedure as that of figure 31 was implemented. The graph with all the values can be observed in Annex 22: 52°C distance calibration curve.

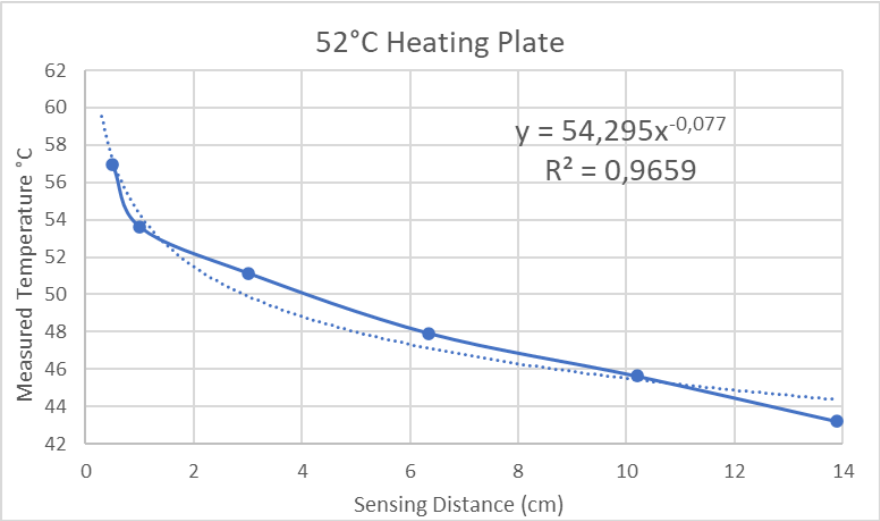


Figure 32: 52°C average distance calibration curve

As previously stated, the temperature of the heated surface has an impact on the thermal variations, which can be observed by looking at the 5mm data points in the previous curves. For the 37°C distance curve, there is an increment of 3°C in the sensor's measurement with respect to the control temperature. However, for the second curve, there is an increase of approximately 5°C. which shows that equation (2) does reflect the changes in the environment. Percentagewise, the curves show an increment of 8.108%, and 9.481%, respectively.

After the distance calibration curves have been created, it is possible to start to predict how to the thermal sensor's thermal imaging works in the final casing. It should be noted that the prediction curve is only used as reference to make sure that the calibration methodology used and implemented is reliable and can be replicated (regardless of the thermal camera's environment) or further expanded if need be. It should also be noted that the results found so far should only be applied for uses of the thermal camera which take place in similar environments as the ones in which the calibration was conducted.

Distance vs temperature prediction curves

Using the data obtained from the thermal camera's heating plate calibration, it is possible to predict the thermal imaging values measured at 2mm from the heated surface. This is done using a linear regression for each of the temperatures obtained at the measured distances. It should be noted that this prediction does not take into account the thermal variations in the sensor's environment which as seen on the distance calibration, significantly affect the thermal readings.

The prediction is done by making sure that the measured points coincide with the linear regression curves. This process is done for both 2mm and 0.2mm (as later a comparison will be made to make sure that the predictions are true, with caliber 8 acetate), the obtained data points are found in Annex 23: prediction curves. After which, the linear regression calibration will be implemented using the values obtained and the linregress function from the scipy library. The equations found are as follows.

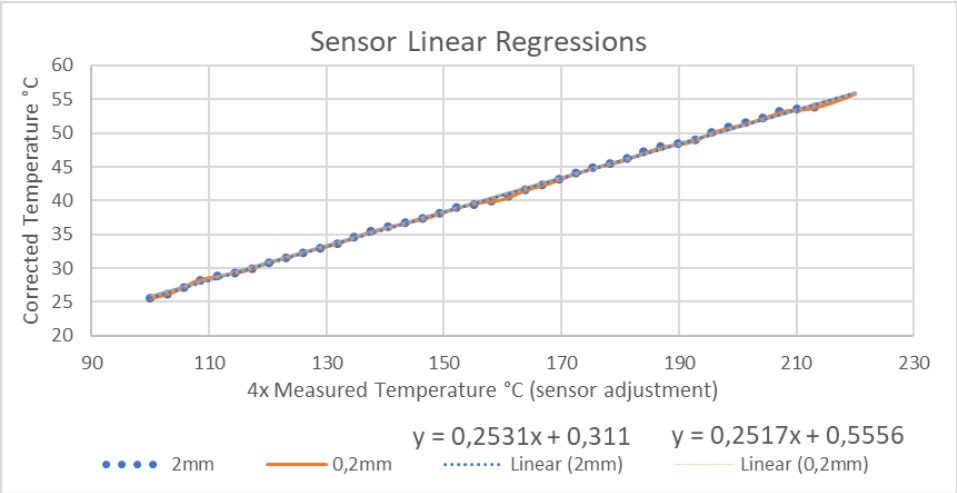


Figure 33: Linear regression predictions

As expected, the calibration curves resemble those of the heating plate calibration for 5mm and 10mm. However, these equations do not compensate for thermal variations. Using the data obtained from the heating plate distance calibration, it is possible to get an offset temperature that could make the prediction curve equations viable for 2mm while keeping the sensor in the calibration position for the length of the process.

The offset is obtained by using the power equations obtained from the 37°C and 52°C curves for a given distance of 2mm. This allows to extrapolate the increase in temperature that the thermal camera might measure for a given temperature. In this case, the temperature for the calibration is 35°C. The data obtained is as follows.

Distance correction			
Distance	Temperature	Equation (figure 30, 31)	Correction
0.2 cm	52°C	$Y = \left(54.295 * (0.2^{-0.077})\right) - 52$	9.4583°C
0.2 cm	37°C	$Y = \left(38.931 * (0.2^{-0.056})\right) - 37$	5.6028°C

Table 5: Distance correction prediction

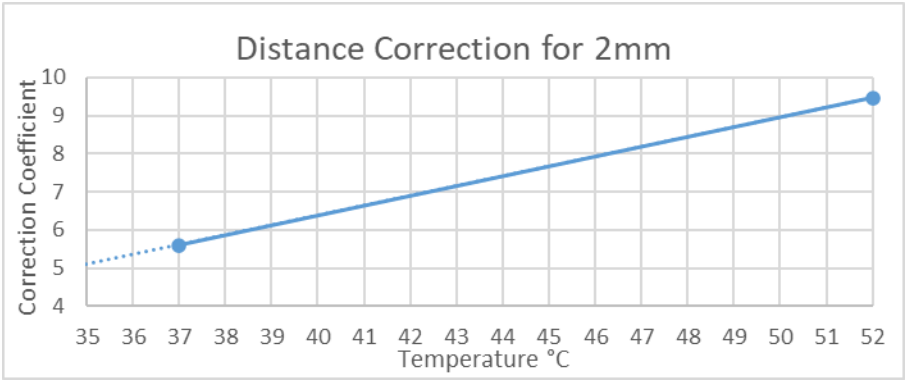


Figure 34: Temperature offset

The data obtained from figure 33 shows that for 2mm, the linear regression is $Y = 0.2531x + 0.311$. Taking into account the correction coefficient from figure 34, the new calibration formula

becomes (3). The same procedure is done for 0.2mm, which gives equation (4). Thus, to verify the calibration obtained, the next step is to apply the m and b coefficients of the formula into the thermal camera's calibration file to take readings of the heating plate through the 2mm acrylic film. It should be noted that for the calibration previously conducted, the geometry of the lens was not needed as the heated surface is large enough to always have accurate data from the sensor's lasers. However, now that the thermal camera is being calibrated using 2mm acrylic for use in the microfluidic system, the size of the microfluidic system's canal needs to be observed, as well as the sensor's distance to the fluid.

$$Y = 0.2531x - 4.7765 \quad (3)$$
$$Y = 0.2517x - 10.12 \quad (4)$$

Thermal sensor lens geometry

The sensor used is the AMG8833 which offers an 8x8 grid of thermal readings. Looking at [36], the geometry of the sensor's laser can be found as a function of their angle. The distribution can also be observed in Annex 24: AMG8833 laser distribution. A script was implemented in python2 to observe the distances of each of the lasers as a function of the thermal camera's distance to a heated surface. This was done using the $Tan^{-1}()$ function in the math library. The script can be observed in Annex 25: thermal sensor geometry code. The results obtained are as follows.

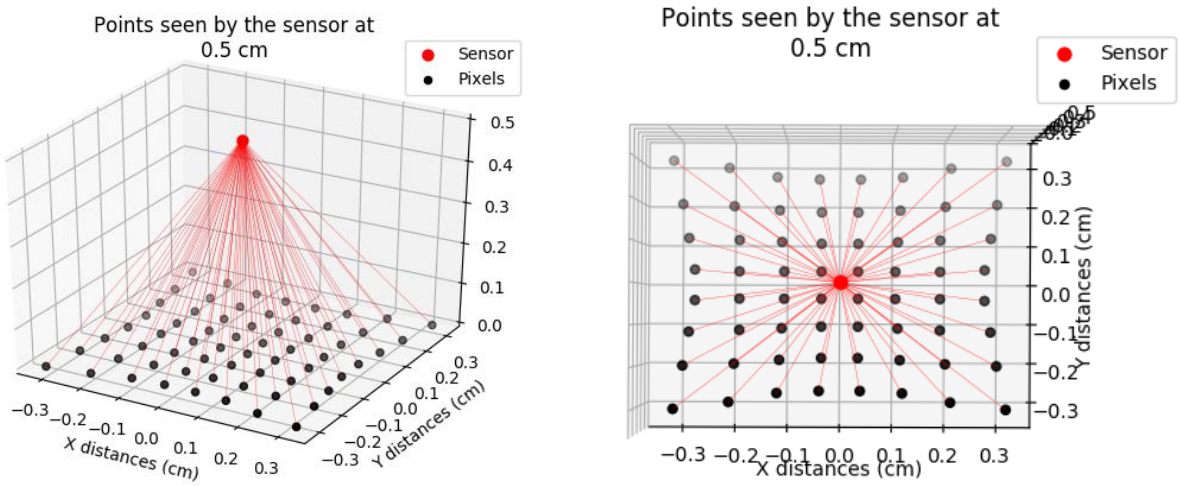


Figure 35: (a) Thermal camera laser geometry 1, (b) Thermal camera laser geometry 2

It can be observed that for the middle four pixels, the X and Y distances are always less than 10% of the distance to the surface. Thus, for the 5mm acrylic contraption created for the distance calibration, the pixels always measure the heated surface. However, the length of the canal in the microfluidic system is 2mm, thus there is a need to find the optimal distance for the thermal camera. The standard microfluidic system can be found in Annex 1: Proposed micro-fluidic system. The films used in the creation of the system are 2mm acrylic, which makes the distance from the bottom of the canal to the sensor, 4mm. Using the python script, it is possible to create a model of the standard microfluidic system as well as the projection of the lasers in the stated configuration. The model of the thermal camera's readings into the microfluidic system can be observed below. It should be noted that not all the length of the microfluidic system is modeled, as only a section is needed to observe the path of the lasers.

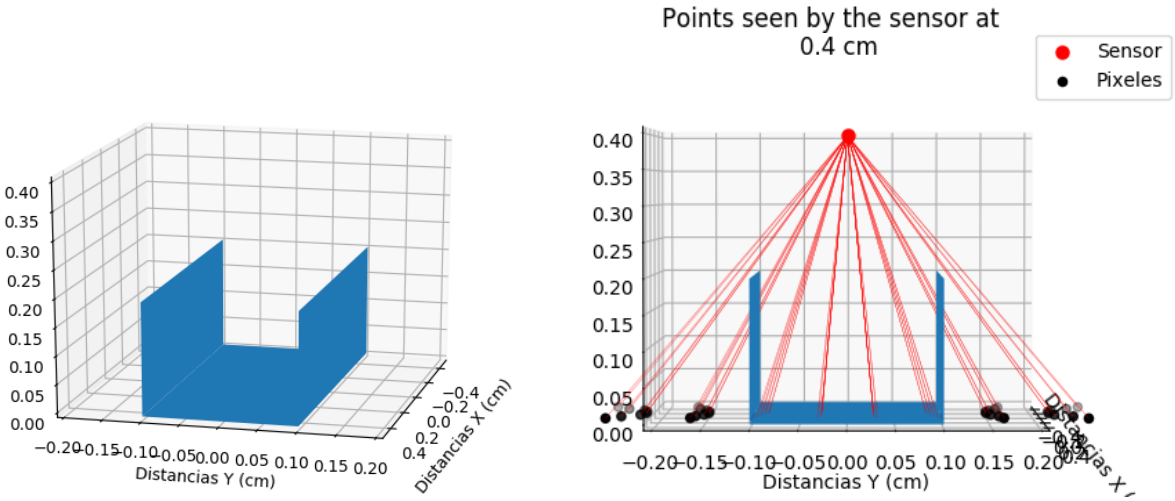


Figure 36: (a) Microfluidic system canal model, (b) Laser geometry at 4mm

At 4mm from the bottom of the canal (specific distances for the proposed microfluidic system), it can be observed that ¼ of the thermal sensor's laser are not directly focused on the fluid. Although, this does not necessarily mean that the thermal reading will be off, it does mean that there might be an increase in thermal variations. Using the script, the maximum distance to be implemented, as to make sure that all the lasers read the fluid's temperature, is 1.5mm. It should be noted that distances below that are also reliable since the lasers read the temperature of the canal's walls as well as the temperature of the bottom of the canal. This makes the previous prediction curves viable for caliber 8 acetate, which offers a thickness of 0.2mm. It should also be noted that using materials which have smaller thicknesses is more viable for the thermal readings, as the material will be more sensitive to thermal changes in the fluid.

2mm acrylic film calibration

To account for the sensitivity of the material to thermal changes, a time vs temperature curve needs to be implemented. For this, the sensor is place on top a 2mm acrylic film which rests on the heating plate. All the time vs temperature calibration curves were executed at a temperature of 35°C. This process is repeated various times, as significant variations were observed while measuring the temperature through the 2mm acrylic film. The time vs temperature curves are found below.

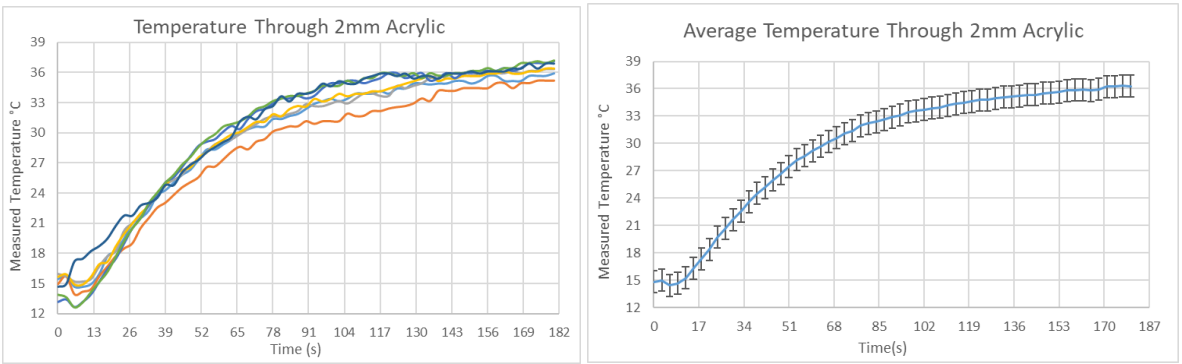


Figure 37: (a) Time responses of 2mm acrylic, (b) Average time response of 2mm acrylic

In [8], a time vs temperature curve which compares the acrylic's temperature response to the fluid's response can be found. However, it does not state the temperature of the fluid which passes through the acrylic canal; thus, it does not allow for replication. It should be noted, however, that

the temperature response time of the acrylic seems to match that which is shown in [8]. It should also be noted that the heated surface used is the heating plate previously implemented. For the previous calibrations, it serves very well as the temperature is always known. For this calibration, however, the temperature of the heating plate does fluctuate, as can be seen in figure 14, for 75% duty cycle.

In the average time response curve (figure 37b), the maximum temperature obtained is 36.33°C. The error bars shown correspond to the ±1.2°C variation for the heating plate at 35°C. Considering this, equation (3) seems to correctly measure the temperature of the heated surface through the acrylic, taking into account thermal variations for keeping the camera in the calibrating position throughout the process [37]. To make sure that the calibration is correct, the same process is repeated using caliber 8 acetate (thickness 0.2mm) and using equation (4) as the thermal sensor's calibration. The obtained curves are as follows.

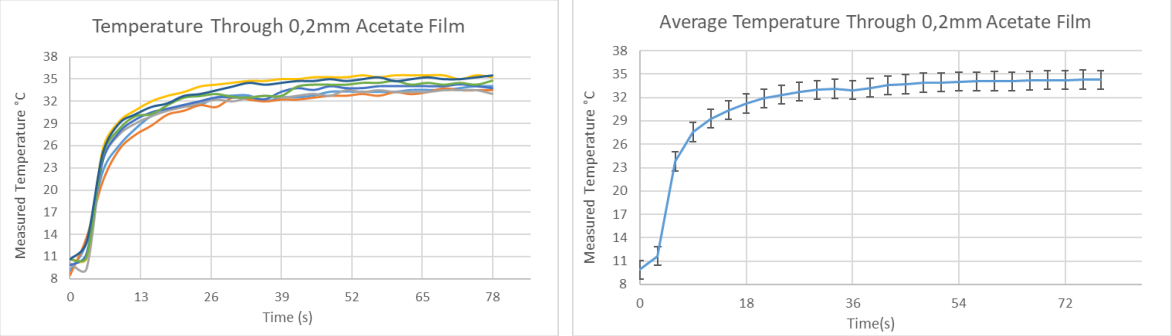


Figure 38: (a) Time responses of 0.2mm acetate, (b) Average time response of 0.2mm acetate

Observing the acetate temperature curve, the material responds in approximately ¼ of the time to heat changes when compared with the 2mm acrylic. Although both materials are viable when compared to the time it takes to adjust the temperature of the fluid [8], acetate provides a better way to measure thermal variations in the system. The highest temperature in the acetate curve is 34.28°C, which when taking into account the temperature fluctuations caused by the heating plate, makes the calibration done by equation (4), viable [38].

Now that the calibration has been proven correct, the parameters of either (3) or (4) are valid for the uses of the thermal sensor at 2mm or 0.2mm, respectively; this, taking into account thermal variations by not moving the sensor. Thus, and to not repeat the whole calibration process, a prototype for automatic or semi-automatic calibration is to be developed.

Characterization

Microfluidic system verification

A case was created to hold the thermal sensor directly on top of the standard microfluidic system. The distribution of the thermal camera's lasers can be observed in figure 36. It should be noted that a microfluidic system using the caliber 8 acetate film should provide a better heat measurement, as well as put up with the liquid's pressure. However, this type of microfluidic system has not been manufactured by BioMicroSystems. It should also be noted that the casing used (figure 39) was

modified for the incorporation of a calibrator circuit prototype. The design used at the time, can be observed in Annex 27: Initial casing, at a 18:25 ratio. The annex also contains schematics for the new microfluidic system dimensions to be correctly mounted on the casings. The contraption used is as follows. Annex 28: microfluidic system pump, shows the architecture used to measure the heated liquid through the microfluidic system.

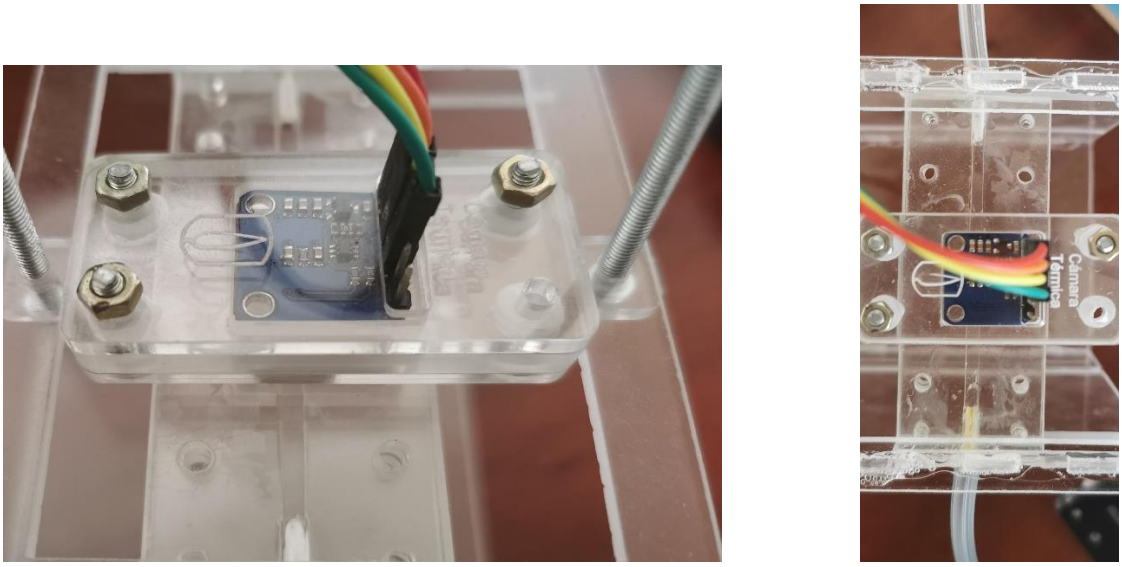


Figure 39: Initial casing for the microfluidic system

Using the stated architecture, the thermal camera's calibration was set using equation (3). The liquid used for the verification of the thermal sensor is tap water, which was heated using the heating plate developed previously and set to a temperature of 35°C. The water was pumped using an arduino 3V-6V submersible water pump, running at 5V. The measurements were taken ten times to obtain accurate data about the verification of the thermal camera's calibration. The curves obtained can be observed below [43].

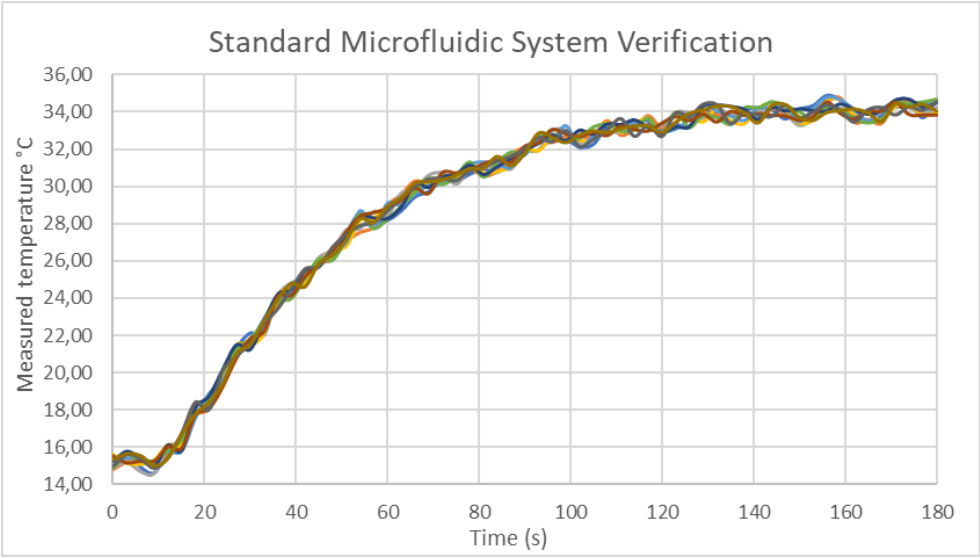
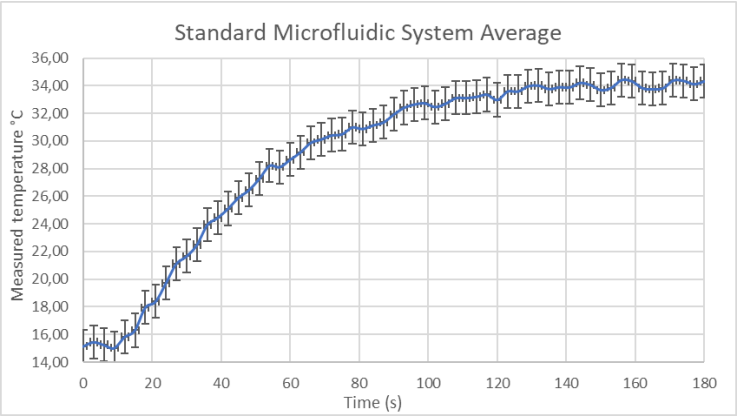


Figure 40: Thermal camera system verification



33.29	33.29	33.86	33.0	33.29	33.0	32.72	32.72
33.57	33.86	34.14	34.14	34.14	33.86	33.0	32.43
33.86	34.43	34.71	34.71	34.14	34.14	34.14	33.0
34.71	35.0	35.0	35.0	35.0	35.0	34.14	33.0
34.14	34.71	35.0	35.0	34.43	34.71	34.14	33.57
34.14	34.71	34.71	35.0	34.71	34.71	34.71	32.72
34.43	34.71	35.0	35.28	34.43	34.14	34.14	33.86
35.57	35.0	34.71	35.57	35.0	34.43	35.0	34.14

Figure 41: (a) Thermal camera average system verification, (b) thermal camera readings

It can be observed that there is less than a 2°C difference between the calibration verification of the microfluidic system and the 2mm acrylic film calibration (which can be observed in figure 37). It can be observed that the variations in figure 40 are much less than those in the previous calibrations; since the heated liquid moves through the microfluidic system, the thermal variations are drastically decreased. This behavior could also be observed during the heating plate calibration of the device, in which the thermal camera was positioned for a maximum of two seconds above the heated surface. Thus, the thermal camera using the 2mm acrylic film calibration obtains correct measurements from the microfluidic system. The heating plate temperature oscillations should also be considered, which are represented by the error bars in figure 41. If need be, the thermal camera's temperature measurements can be modified to read 1°C lower than in equation (3), by adding a coefficient of 1 to the 'b' parameter.

Calibrator circuit

For the creation of a calibrator circuit prototype, the main idea is a device that allows to obtain a temperature gradient at a known distance from the thermal camera. Thus, using information from the previous calibrations, as well as the variations from the temperature gradient, the curves can be modified to take into account the thermal variations in the environment.

Theory

The prototype is to focus on mapping the thermal variations present in the thermal camera's environment to the already known equations (3) or (4), depending on the distance at which the sensor will be used. For the creation of the temperature gradient, [39] uses a current mirror configuration to heat up a resistor and create a voltage vs temperature curve that stabilizes at certain values. Thus, keeping the voltage the same, but implementing different resistor values, would allow to dissipate different amounts of power, which should have the same effect. To make sure that the resistors get to a specific temperature value every time, it is necessary to pass current through them for a known period. This way, the variations of the resistors' heating curve should be minimized.

For the current mirror configuration to properly work, all the transistors should have the same β values as well as the same internal characteristics. Thus, the device would require transistor arrays, which possess the same characteristics, such as the CA3046 NPN transistor array or the CA3081 in

emitter configuration. These, however, were not available for the time frame needed for the completion of the project. Thus, a different alternative was needed, in the form of a motor driver. The motor drivers taken into account are the ULN2803 and the ULN2003, which consist of 7 and 8 Darlington pairs, respectively. Although the configuration of the Darlington pairs is not the same as the one used in [39] (as can be observed below), the input impedance and the feedback impedance between the common emitter and the base, should allow for a more stable operation [42].

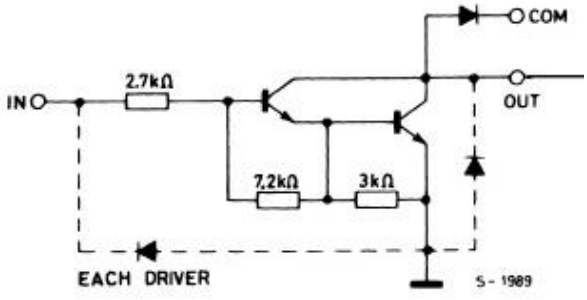


Figure 42: ULN2003A Darlington pair [42]

To make sure that the current always passes for the same amount of time, a relay is powered through a microcontroller signal. It should be noted that each of the resistors should be measured, to know if the temperature curves of the resistors are consistent.

Simulation

Different values were selected in the simulation, however, once implemented it was difficult to either obtain a visible temperature gradient through the thermal camera, or the resistors did not dissipate enough power as to properly heat up. Using the following combination of resistors, it was possible to obtain a visible temperature gradient. The power dissipated through the resistors in the simulation is as follows. It should be noted that since the values selected dissipate more than 1W, the resistors used are 1% tolerance with 2W capacity. It should also be noted that although some resistor values such as 50 Ω do not cross the 2W threshold, the implementation of these resistors was not possible as the real power dissipated seemed to be greater than that of the simulation. This caused some damage in the resistors which had values below 100 Ω.

Power dissipated by current mirror configuration	
Resistor value	Power dissipated
470 Ω	301.352mW
270 Ω	519.676mW
200 Ω	696.494mW
150 Ω	920.289mW
120 Ω	1.140W
100 Ω	1.356W

Table 6: Current mirror data

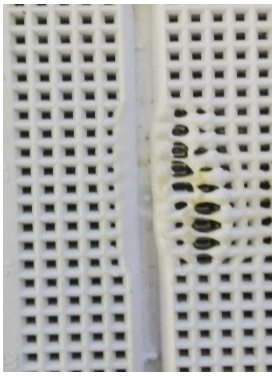


Figure 43: Breadboard damage from ULN2803

The overall schematic of the device, as well as the microcontroller signals which control the device's functionality can be found in Annex 26: Calibrator circuit schematic. The input voltage is set to 12V

as for inputs of less magnitude, the power dissipated was not as great, and the temperature gradient was not visible through the thermal sensor. It should be noted that the temperature sensing of each of the resistors is done using LM35 sensors, which in turn send the readings to the microcontroller which controls the device's relay.

Implementation

The device was implemented using both the ULN2803 and the ULN2003 in a breadboard to verify their functionality. However, it was found that the ULN2803 reaches higher collector currents for the same saturation voltage and input current as the ULN2003 [39][40]. Thus, although the functionality of both the motor drivers was consistent, the ULN2803 heated up to the point which caused damage to the breadboard. It should be noted, that due to the distances between the motor drivers and the current mirror resistors, the reading was thrown off due to the ULN2803's temperature. Thus, the device is to be implemented strictly using the ULN2003 motor driver (a more detailed PCB schematic can be observed in Annex 29: Calibrator circuit PCB).

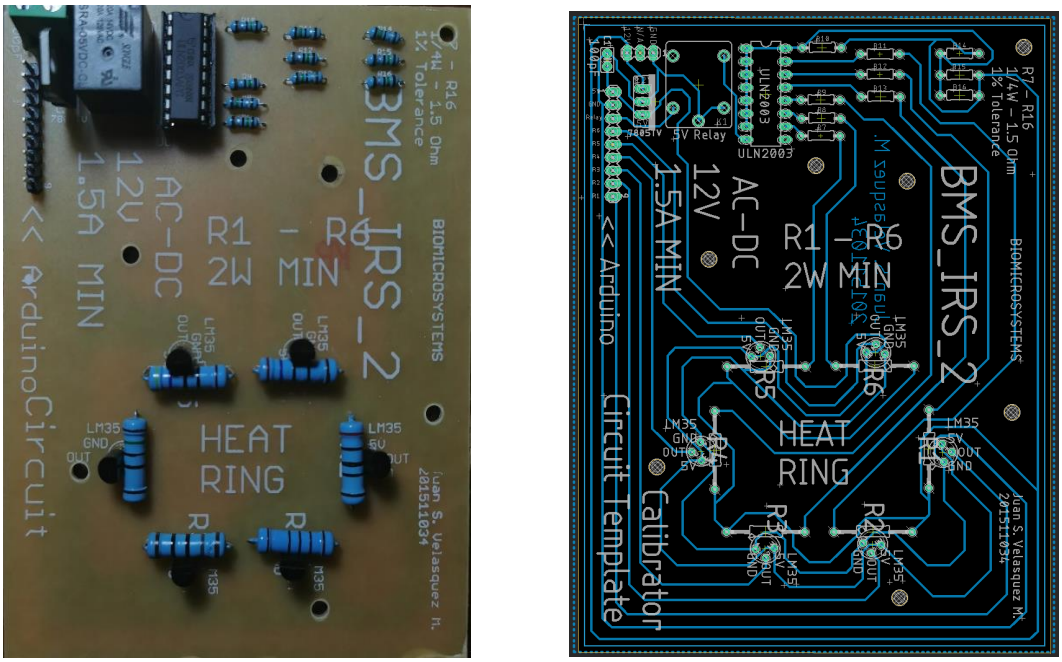


Figure 44: (a) Calibrator circuit implementation, (b) calibrator circuit PCB

To make sure that the LM35 sensors are correctly measuring the resistor's temperature, they are calibrated using the heating plate and the infrared thermometer. The calibration curves can be observed in Annex 30: LM35s' calibration. This is needed to later correlate the circuit's temperature range with the temperature range measured by the thermal camera. A demonstration of how the thermal camera observes the heat ring in the circuit can be observed below. It should be noted that these images are not taken in a specific set up, rather the camera was just positioned above the heat ring at a distance which allowed to properly view the temperature gradient.

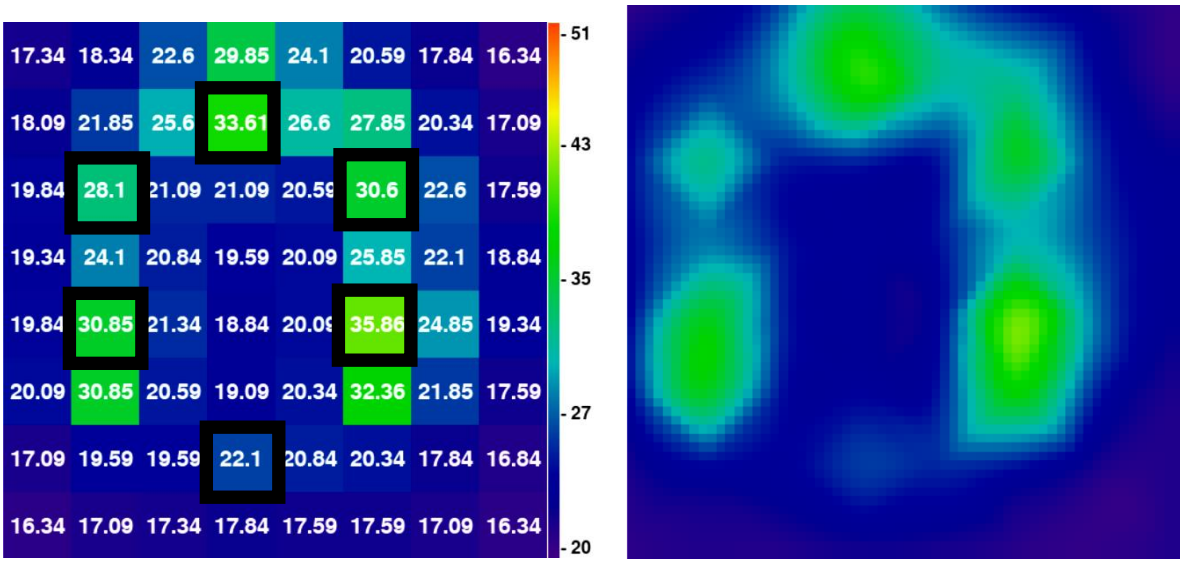


Figure 45: Heat ring temperature gradient

It can be observed that the calibrator circuit prototype creates a temperature gradient. It should be noted that during the implementation of the calibrator circuit in the breadboard as well as in the soldered board, the LM7805 voltage regulator as well as the ULN2003 motor driver did not present any heating issues.

In the PCB, however, the LM7805 voltage regulator overheated. Thus, a heat sink was implemented in the PCB; to not throw off the thermal readings, for the calibrator circuit prototype implemented, the circuit was kept unpowered from the AC-DC converter, until the thermal calibration was needed. The circuit is then implemented in the final casing, as to have the same laser distribution for the temperature gradient.

The system calibration was implemented using the serial communication with the arduino microcontroller. The calibrator circuit prototype has pins which send the data from the LM35 sensors to the microcontroller, then, the microcontroller sends the data to the raspberry Pi through the serial connection previously implemented. In the serial communication interface, by sending the command ‘cal ‘, the arduino microcontroller activates the relay in the calibrator circuit, then takes the temperature measurements and keeps them in the EEPROM. It should be noted that the serial connection expects a response within a few milliseconds, thus, the data needs to be stored in the microcontroller. When the command ‘get’ is used, the arduino sends the data to the raspberry Pi, which then creates the calibration curve. It should be noted that the data is stored in the arduino in a non-volatile memory, thus, if the calibration file is compromised, it is only necessary to call the command ‘get’. The implementation also possesses a command ‘clc’, which allows to erase the EEPROM data stored. It is not necessary to call ‘clc’ before the calibration command, as ‘cal’ overwrites the data. However, for the calibration to be correct, the calibration.txt file needs to be removed from the root directory.

The calibration curve is created by taking a thermal map when the ‘get’ command is sent. This thermal map is then correlated with the data obtained from the arduino microcontroller, by knowing the pixels which correspond to the resistors. The thermal camera’s readings are previously calibrated taking into account the calibration process for a heated surface with thermal variations

at 72mm. This, due to the distance from the thermal sensor, to the calibrator circuit prototype. The calibration equation obtained is:

$$Y = 0.251x + 2.4431 \qquad (5)$$

In Annex 32: calibrator circuit pixels, it can be observed which connections are made with the arduino, and the pixels that should be selected for the calibration curve. These pixels are the ones selected when the calibrator circuit is in the final casing. The temperature measurements from the thermal camera are corrected using equation (5), and then with equation (3), to have a good idea of the values in terms of the 2mm acrylic film calibration. Then, a linear regression equation is done using the values obtained with those of the LM35 sensors with the linregress function.

Results

First, using the LM35s touching the resistors, the readings obtained were not accurate and presented significant variations. Thus, it was necessary to improve the thermal conductivity between the resistors and the LM35s. For this, the resistors were connected to the sensors using HY510 thermal paste. In [44] the calibrator circuit's operation can be observed, when used with the thermal paste.

AMG Temperatures °C				
X1	X2	X3	X4	Average
31,42	34,8	33,15	32,82	33,0475
25,6	29,18	27,64	27,56	27,495
43,57	43,03	43,36	43,35	43,3275
37,24	36,58	36,7	36,85	36,8425
45,84	43,68	45,13	44,86	44,8775
29,39	31,54	30,04	30,02	30,2475

Table 7: AMG readings

LM35 with thermal paste °C				
X1	X2	X3	X4	Average
38,48	37,67	38,75	37,82	38,18
32,26	31,84	31,76	33,51	32,3425
49,85	49,09	50,71	51,74	50,3475
44,18	41,74	40,3	42,47	42,1725
50,07	52,35	52,81	51,48	51,6775
35,64	37,87	38,37	36,47	37,0875

Table 8: LM35 readings

Figure 46: Thermal paste

For the values obtained using the thermal camera and doing the corrections with equations (5) and (3), the linear regressions created with the LM35 sensors' values are as follows.

Linear regression curves				
X1	X2	X3	X4	Average
$Y = 0.2314x + 2.5458$	$Y = 0.3204x - 13.73$	$Y = 0.2787x - 5.6385$	$Y = 0.2761x - 4.956$	$Y = 0.274x - 4.955$

Table 9: Calibrator circuit linear regressions

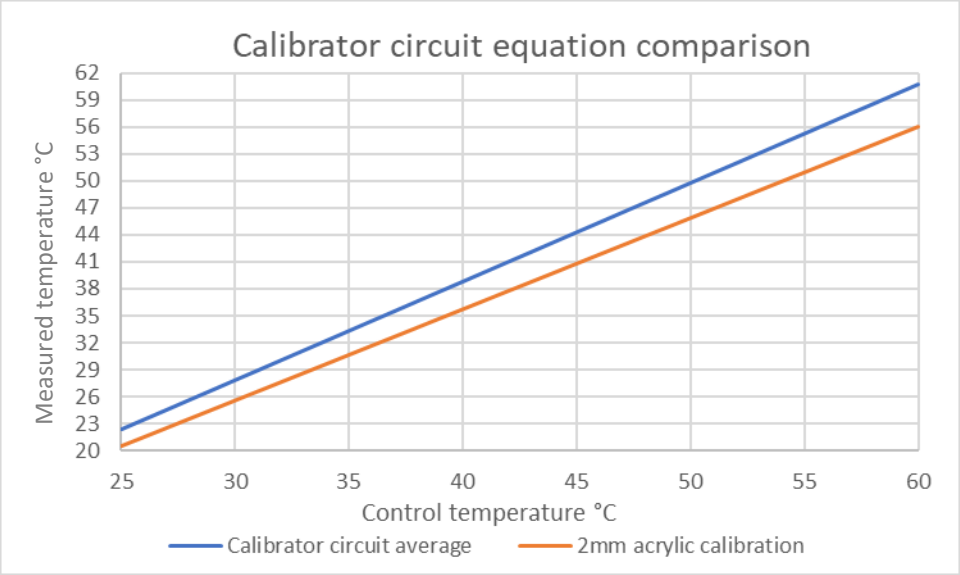


Figure 47: Calibration comparisons

Although there are variations between the average linear regression from the calibrator circuit and the 2mm acrylic calibration, the overall process seems to behave as expected. The values appear to be mapped into the 2mm acrylic calibration range, while taking into account some degree of thermal variations. It should be noted that the 2mm acrylic calibration was done for a temperature of 35°C, while the calibrator circuit's temperature creates a range of approximately 20°C.

The microcontroller's code can be observed in Annex 33: Arduino code, the client code can be observed in Annex 34: Client code, and the raspberry Pi's code can be observed in Annex 35: Raspberry Pi code.

Device implementation

Casing

The final casing for the device is implemented in 5mm acrylic, taking into account that the dimensions for the new standard microfluidic system should be 30mmx100mm, with a 2mm canal. This is due to the fact, that the new casing also has the space to set up the 6-channel color sensor, used in visible light spectrophotometry. It should be noted that the casing also has a bedding, which allows to use the current microfluidic system, which possesses dimensions of 30mmx75mm, to be used. The casing also has a bedding for the calibrator circuit, which has holes that match those of the calibrator circuit's PCB. Thus, it ensures that the laser distribution is kept consistent, and replicable. The final casing is shown below; the final casing design can also be observed in Annex 31: Final casing design.

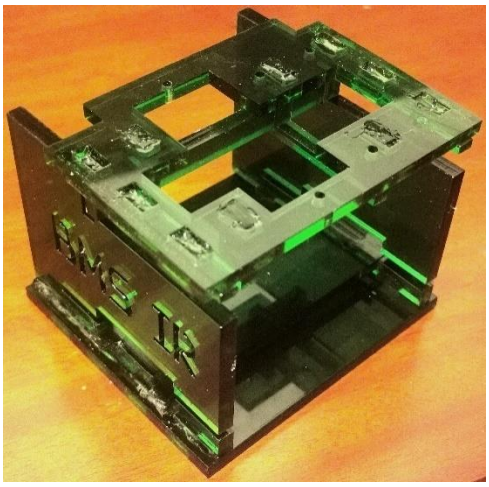


Figure 48: Final casing

Circuitry

The final implementation of the device consists of the raspberry Pi, an arduino microcontroller, the AMG8833 thermal camera, as well as a current mirror calibrator circuit. It should be noted that for proper operation, the raspberry Pi should be supplied with 2.5A; this, to properly control the arduino (it should be noted, as previously stated, that the implementation can operate multiple arduino microcontrollers). The calibrator circuit should be supplied 1.5A to function properly. It should also be noted that the calibrator circuit also possesses an LM7805 voltage regulator which can serve as a secondary power for the arduino, since the calibrator circuit only uses 1.3A.

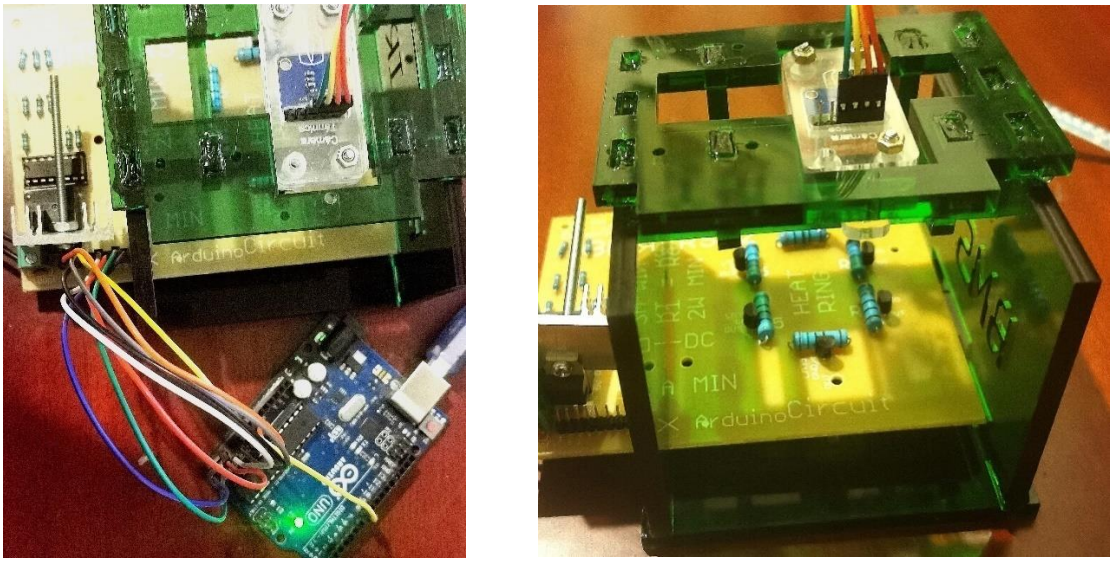


Figure 49: Device implementation

The calibrator circuit prototype itself uses all the arduino's analog pins. For proper data communication between the device and the arduino, the 5V and GND should be connected, although just the GND pin suffices. It should be noted the arduino oversees the power time for the current mirror, thus, a pin should be allocated for the circuit's relay.

Discussion

The previous software was modified to serve as a hub for the various electronic devices in the BioMicroSystems laboratory. Thus, the major implementations were communication interfaces using serial ports as well as TCP protocols, to be able to control the various devices.

The serial communication interface was implemented in such a way that allows to increase the amount of arduino microcontrollers without complications. Since in the serial interface, the input text box is directly connected to the arduino's /dev location, creating input box variables with new system locations, allows a direct communication with the microcontrollers. Thus, the arduino microcontrollers' code can be implemented using the serial terminal integrated in the arduino IDE, which works as the implemented input boxes in the serial interface. One major complication of the implementation is the library in which the previous software was created. Although PyGame is very versatile, it does not possess important components which allow for the proper functionality of the interface. The serial interface and input box's updating must be called manually inside their respective loops. Therefore, in some instances the keyboard events are not registered properly.

If any more substantial implementations are to take place, it is recommended to rewrite the interface using a different library, which possesses optimized components, as to save memory and avoid further creation of threads. Moreover, the previous implementation showed some errors while saving data, which logged out the user; the thread was modified to be tied to a terminal, thus, the user won't be logged out as long as the program is run using a python terminal. This implementation also allows for video and data collection simultaneously.

Regarding the calibration process, it should be noted that access to the laboratory's equipment was limited. The heating plate used as the measurement surface for the thermal camera was always kept below 50°C, as such the temperature ranges for which all the calibration were conducted, did not surpass this threshold. This is due to the nature of the heating plate ON-OFF controller, which presents greater temperature variations, the higher the temperature set. However, the AMG8833, can measure temperatures up to approximately 80°C. As such, the calibration conducted may not provide the best results for such temperatures, as the variations of the linear regression equations at these temperatures may be significant.

If the liquid to be used inside the microfluidic systems possesses a higher temperature than the ones in the calibration, the calibration process was reliable enough for it to be replicable using a commercial heating plate that could provide a larger temperature range. It should be noted that the higher the temperatures used, the more thorough the calibration process should be, this was observed with how the thermal variations affected the distance vs temperature curves. If the calibration process is to be repeated at much higher temperatures, the thermal sensor's readings could be thrown off by the heated air above the heating plate (this can be observed with an infrared thermometer measuring a subject's mouth: by exhaling slowly, the thermal readings can be modified), thus, a mechanism to minimize these variations should be developed. Although the results obtained were correct, another distance vs calibration curve between 35°C and 52°C could have provided a more accurate outcome.

During the characterization process, the results obtained were on par with those from the calibration process, which shows that the methodology implemented was able to accurately compensate for thermal variations in the environment. Thus, although the scope of the thesis did

not include a system that could replicate the calibration process in a few minutes, a prototype was proposed. Although the theory behind the calibrator circuit was planned out, the implementation required a lot of changes, since the simulation does not account for heat dissipation in the resistors. Rather, the simulation was done to observe how the resistor values altered the power dissipated. After which, the circuit was implemented to observe if a temperature gradient had been created. This process was repeated until a range of temperatures of about 20°C was achieved. It should also be noted that the prototype itself, also takes into account equation (3) to map the values, thus it is not an independent device, but rather a tool to take into account thermal variations.

Conclusions

Regarding the device's software, the program can properly record and save video files using both .mkv and .mp4 extensions as well as let the user choose between full screen recording and thermal map recording. The difficulties which stemmed from the root folder (and a lack of a directory explorer) and the video threads which logged out the user were also addressed. Furthermore, the creation of serial interface as well a Client-Server model, allows the program a large degree of versatility as an electronic hub for the laboratory equipment. Lastly, although the specifications of the raspberry Pi 3 model B suffice, a newer model with more processing power could address some of the issues created by the manual updating of the PyGame library.

For the device's calibration, the methodology applied for the calibration process proved correct, as can be observed with the data obtained in the beginning of the device's characterization. The thermal camera was able to be properly calibrated to account for the thermal variations in the environment; this also includes the heat which emanates from the heated surfaces. Although the device's calibration proved correct, a more thorough analysis as well as more calibration curves at different temperatures could have provided a better understanding of how the thermal camera observes the variations in the environment.

Furthermore, when comparing the data shown in [8] for the temperature response time of the thermal bath, with the response time of the 2mm acrylic, it can be observed that the material responds faster and thus serves as a good casing for the microfluidic system. Moreover, an analysis of the laser distribution of the thermal camera was implemented to observe if the data obtained corresponded to that from the fluid in the canal. Although ¼ of the lasers are not directly focused on it, the thermal variations seemed to be minimal; although this could also be considered as a factor which could impact the thermal readings (a minor part of the variations between the calibration for 2mm acrylic and the characterization with the microfluidic system could perhaps be attributed to this). In [8], it is proposed that an alternate material with faster thermal changes could prove more useful in observing the thermal maps inside the canal. Caliber 8 acetate possesses the same characteristics as the 2mm acrylic film. However, it is much thinner, which not only allows the laser distribution to be solely focused on the canal but makes the thermal changes in the material happen in about ¼ of the time. This would serve as a better casing as it would allow to observe more detailed thermal reactions inside the canal.

Regarding the laboratory equipment needed for the replication of the calibration process, commercial heating plates could serve as a good alternative and provide large temperature ranges, which could provide better outcomes. However, as access to the BioMicroSystems' laboratory was restricted, the methodology required the creation of laboratory equipment. The heating plate implemented proved effective and accurate for the temperatures used, during the calibration of the thermal sensor as well as provides added value to the methodology implemented. It is also important to state that the heating curves of the heating plate were adjusted to values of 35°C, adjusting the voltage regulator in conjunction with the trimmers could provide very accurate heating curves for values well above the 80°C threshold.

The thesis conducted was based on [8], which provided the backbone for the software as well as the understanding of the thermal sensor employed. The objectives of the thesis conducted were met regarding the analysis of the thermal variations as well as the implementation of a system that accurately measures the heat of the fluid in the microfluidic system. Taking into account the video/data characteristics of the device, paired with the implementations of the communication interfaces in the device's program, the objectives previously stated were exceeded.

Acknowledgements

Como comenzar los agradecimientos cuando hay demasiadas personas que me han ayudado. Quiero agradecer a mi mamá y a mis papás por siempre apoyarme en mis decisiones y en las oportunidades que se han presentado. Muchas gracias por escucharme cuando me pongo insoportable hablando de la tesis, y básicamente de cualquier otra cosa que se me cruce la mente. También a Brayan Ariza por el trabajo realizado anteriormente, y por ayudarme a tener una idea para la tesis.

I would also like to thank my sister, which just basically makes noises and calls me over occasionally. It somehow does help keep me both entertained and focused, as I have less time to finish my work and consequently, requires me to do it when I have free time. I would like to thank my thesis advisor, Johann Osma for helping me understand the scope of the thesis, as well as the overall functionality of the device. Also, for being incredibly respectful even after I called him on a festive day… I am sorry.

Gostaria de agradecer a Eduin pelas conversas filosóficas muito profundas, que foram uma boa mudança de ritmo.

Наконец, я хотел бы поблагодарить следующих людей: один из моих самых близких друзей, Риккардо Кебак, Фаусто Саламанка, Фелипе Эскаллон, моего друга Джорджа Асеведо, Сантьяго Товара, Даниэля Мекона и моего русского друга, который не является русским, Хосе Рестома. И, конечно же, Ковид за то, что держал меня дома.

References:

[1] “Spectrophotometry,” *Biocompare*. [Online]. Available: https://www.biocompare.com/Lab-Equipment/6531-Spectrophotometry/. [Accessed: Sep-2020]

[2] “Introducción a la Espectroscopia Molecular,” *Lección 14 Química Física*, Nov-2010. [Online]. Available: http://www.uco.es/organiza/departamentos/quimica-fisica/quimica-fisica/QuiFis/L14_QF_10_11.pdf

[3] A. D. Strook, F. Bragheri, and R. Osellame, “Microfluidics,” *Microfluidics - an overview | ScienceDirect Topics*, 2016. [Online]. Available: https://www.sciencedirect.com/topics/materials-science/microfluidics

[4] J. P. Rafferty, “Beer's law,” *Encyclopædia Britannica*. [Online]. Available: https://www.britannica.com/science/Beers-law. [Accessed: Sep-2020]

[5] *Infrared Spectroscopy*, 05-May-2013. [Online]. Available: https://www2.chemistry.msu.edu/faculty/reusch/VirtTxtJml/Spectrpy/InfraRed/infrared.htm. [Accessed: Oct-2020]

[6] K. J. Havens and E. J. Sharp, “Thermal Imaging,” *Thermal Imaging - an overview | ScienceDirect Topics*, 2016. [Online]. Available: https://www.sciencedirect.com/topics/earth-and-planetary-sciences/thermal-imaging. [Accessed: Sep-2020]

[7] T. m. i. society, «Optical Properties» from Characterization and Failure Analysis of PLASTIC, AMS International, 2003, p. 43-44.

[8] B. E. Ariza, “Cámara térmica para sistemas microfluídicos,” *Biblioteca - Universidad de Los Andes*, 2019. [Online]. Available: http://biblioteca.uniandes.edu.co/acepto201699.php?id=20720.pdf. [Accessed: Aug-2020]

[9] “Seebeck effect,” *Encyclopædia Britannica*. [Online]. Available: https://www.britannica.com/science/Seebeck-effect. [Accessed: Oct-2020]

[10] “Infrared Spectrophotometer,” *Labcompare*. [Online]. Available: https://www.labcompare.com/Spectroscopy/116-Infrared-Spectrophotometer-IR-FTIR-Spectrometer/. [Accessed: Sep-2020]

[11] P. A. Petrini, E. A. Patino-Narino, M. V. Puydinger, E. Joanni, R. Savu, S. A. Moshkalev, and J. W. Swart, “Thin Film Heating and Temperature Control System for Microfluidic Devices,” *ResearchGate*, May-2019. [Online]. Available: https://www.researchgate.net/publication/333004296_Thin_Film_Heating_and_Temperature_Control_System_for_Microfluidic_Devices. [Accessed: Sep-2020]

[12] M. F. Udonkang, I. J. Inyang, A. N. Ukorebi, F. Effiong, U. Akpan, and I. E. Bassey, “Spectrophotometry, Physiochemical Properties, and Histological Staining Potential of Aqueous and Ethanol Extracts of Beetroot on Various Tissues of an Albino Rat ,” *ResearchGate*, 25-Oct-2018. [Online]. Available: https://www.researchgate.net/publication/328535691_Spectrophotometry_Physiochemical_Properties_and_Histological_Staining_Potential_of_Aqueous_and_Ethanol_Extracts_of_Beetroot_on_Various_Tissues_of_an_Albino_Rat/fulltext/5bd30d68a6fdcc3a8da6c94f/Spectrophotometry-Physiochemical-Properties-and-Histological-Staining-Potential-of-Aqueous-and-Ethanol-Extracts-of-Beetroot-on-Various-Tissues-of-an-Albino-Rat.pdf. [Accessed: Sep-2020]

[13] L. Hattersley, “Raspberry Pi 4, 3A+, Zero W - specs, benchmarks & thermal tests,” *The MagPi magazine*, 2018. [Online]. Available: https://magpi.raspberrypi.org/articles/raspberry-pi-specs-benchmarks. [Accessed: Oct-2020]

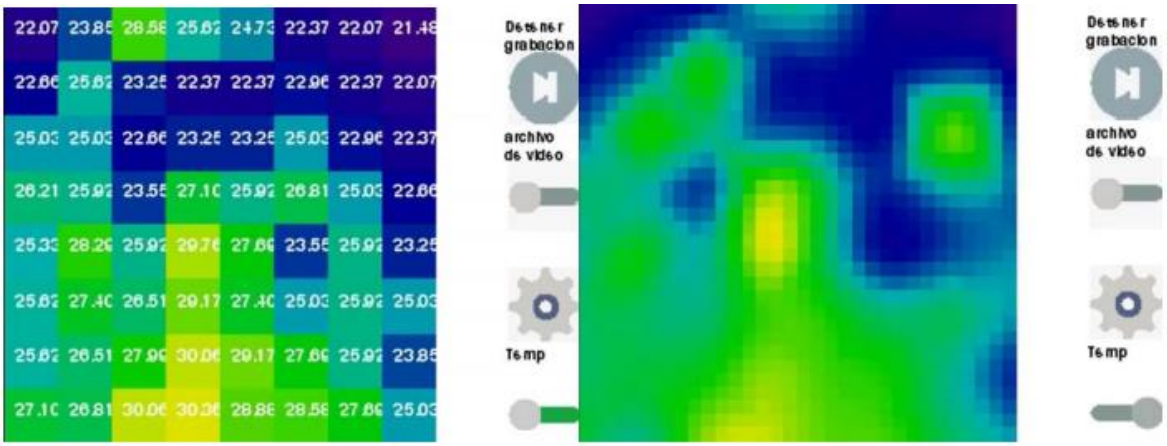
[14] stensootlastensootla and matamata, “pygame vs tkinter.,” Stack Overflow, May-1961. [Online]. Available: https://stackoverflow.com/questions/10668116/pygame-vs-tkinter. [Accessed: Sep-2020]

[15] J. S. Velasquez, “MAP,” Youtube, 13-Nov-2020. [Online]. Available: https://youtu.be/Q3pwlxHuTZE
[16] J. S. Velasquez, “FULLSCREEN,” Youtube, 13-Nov-2020. [Online]. Available: https://youtu.be/dq7FnyysdNo
[17] J. S. Velasquez, “Serial Communication,” Youtube, 04-Sep-2020. [Online]. Available: https://youtu.be/JgEQt4wA8EE
[18] J. S. Velasquez, “Client-Server model,” Youtube, 04-Sep-2020. [Online]. Available: https://youtu.be/me4fzVSy0YM
[19] G. Long, “(PDF) Design of a non-contact infrared thermometer,” *ResearchGate*, Jun-2016. [Online]. Available: https://www.researchgate.net/publication/309262087_Design_of_a_non-contact_infrared_thermometer. [Accessed: Oct-2020]
[20] “Transmission Curves of Optics Materials,” *GS Plastic Optics*. [Online]. Available: https://www.gsoptics.com/transmission-curves/. [Accessed: Oct-2020]
[21] “Molded Plastic Aspheres,” *ThorLabs*. [Online]. Available: https://www.thorlabs.de/newgrouppage9.cfm?objectgroup_id=6138. [Accessed: Oct-2020]
[22] “Acrylic Sheet - Optical and Transmission Characteristics,” *PlexiGlass by Arkema*, 2000. [Online]. Available: https://www.plexiglas.com/export/sites/plexiglas/.content/medias/downloads/sheet-docs/plexiglas-optical-and-transmission-characteristics.pdf. [Accessed: Sep-2020]
[23] M. Kuriakose, “THERMAL INVESTIGATIONS ON POLYMER DISPERSED LIQUID CRYSTAL COMPOSITES & THERMO-ELECTRIC POLYMER COMPOSITES USING PHOTOTHERMAL TECHNIQUES,” *ResearchGate*, Jun-2013. [Online]. Available: https://www.researchgate.net/publication/242012551_PhD_Thesis_Maju-KURIAKOSE_June-2013. [Accessed: Oct-2020]
[24] “W10209 Temperature Control Switch,” *Kelco*. [Online]. Available: http://www.kelco.rs/katalog/images/17670.pdf. [Accessed: Sep-2020]
[25] J. S. Velasquez, “HotRelay,” *Youtube*, 20-Nov-2020. [Online]. Available: https://youtu.be/jHDYbzm5yYI
[26] “SUNPHOR Non-Contact Infrared Thermometer User Manual BZ-R6,” *Samasuka*. [Online]. Available: http://samasuka.my/wp-content/uploads/2020/05/IFU.pdf. [Accessed: Oct-2020]
[27] “DT-11F Flex Tip Digital Thermometer,” Thermoworks. [Online]. Available: https://www.thermoworks.com/pdf/user_manuals/Low_Cost_Digitals/DT_11F_Flex_Tip_DigitalvThermometer_User_Manual.pdf. [Accessed: Oct-2020]
[28] J. S. Velasquez, “CurvaTemperatura 35°C 1,” Youtube, 18-Sep-2020. [Online]. Available: https://youtu.be/cjP7TYS4oLI
[29] J. S. Velasquez, “CurvaTemperatura 35°C 2,” Youtube, 18-Sep-2020. [Online]. Available: https://youtu.be/mK7zi6v_CN8
[30] J. S. Velasquez, “CurvaTemperatura 35°C 3,” Youtube, 18-Sep-2020. [Online]. Available: https://youtu.be/RwB2JGHj_3M
[31] J. S. Velasquez, “CurvaTemperatura 38°C,” Youtube, 17-Sep-2020. [Online]. Available: https://youtu.be/ohRhE2VJytI
[32] J. S. Velasquez, “CurvaTemperatura 44°C,” Youtube, 17-Sep-2020. [Online]. Available: https://youtu.be/vcO8OdUfcsQ
[33] J. S. Velasquez, “CurvaTemperatura 50°C,” Youtube, 17-Sep-2020. [Online]. Available: https://youtu.be/VLtTLCZcf1g
[34] J. S. Velasquez, “CurvaTemperatura 55°C,” Youtube, 19-Sep-2020. [Online]. Available: https://youtu.be/MEbPMuGhH5s
[35] “Limitations of Infrared Thermometers,” Electronic Temperature Instruments, 2020. [Online]. Available: https://thermometer.co.uk/content/101-limitations-of-infrared-thermometers. [Accessed: Sep-2020]

[36] “Infrared Array Sensor Grid-EYE (AMG88),” Panasonic, 02-Apr-2017. [Online]. Available: https://cdn.sparkfun.com/assets/4/1/c/0/1/Grid-EYE_Datasheet.pdf. [Accessed: Oct-2020]
[37] J. S. Velasquez, “Acrílico 2mm,” Youtube, 09-Nov-2020. [Online]. Available: https://youtu.be/1TpfXM1GhIs
[38] J. S. Velasquez, “Acetato 0.2mm,” Youtube, 09-Nov-2020. [Online]. Available: https://youtu.be/VxCsYNCMGPI
[39] S. Tovar, C. A. Hernandez, and J. F. Osma, “Copper-chrome-based glass heater integrated into a 3 PMMA microfluidic system,” MicroMachines, 2020.
[40] Alldatasheet.com, “ULN2003,” ALLDATASHEET. [Online]. Available: https://pdf1.alldatasheet.com/datasheet-pdf/view/125994/ETC1/ULN2003.html. [Accessed: Nov-2020]
[41] Alldatasheet.com, “ULN2803,” ALLDATASHEET. [Online]. Available: https://pdf1.alldatasheet.com/datasheet-pdf/view/12687/ONSEMI/ULN2803.html. [Accessed: Nov-2020]
[42] “ULN2003A Darlington Array 7 NPN 500mA,” ST Microelectronics, 2020. [Online]. Available: https://www.hobbytronics.co.uk/uln2003a-darlington-array. [Accessed: Nov-2020]
[43] J. S. Velasquez, “Microfluidic System Characterization,” Youtube, 19-Nov-2020. [Online]. Available: https://youtu.be/CJkG1_oQL2Q
[44] J. S. Velasquez, “Calibrator Circuit,” Youtube, 19-Nov-2020. [Online]. Available: https://youtu.be/-YCiCg6buSM
[45] J. S. Velasquez, “ColdRelay,” Youtube, 20-Nov-2020. [Online]. Available: https://youtu.be/gNz64srZcGE
[46] J. S. Velasquez, “BothRelays,” Youtube, 20-Nov-2020. [Online]. Available: https://youtu.be/alFb_mDPLmA

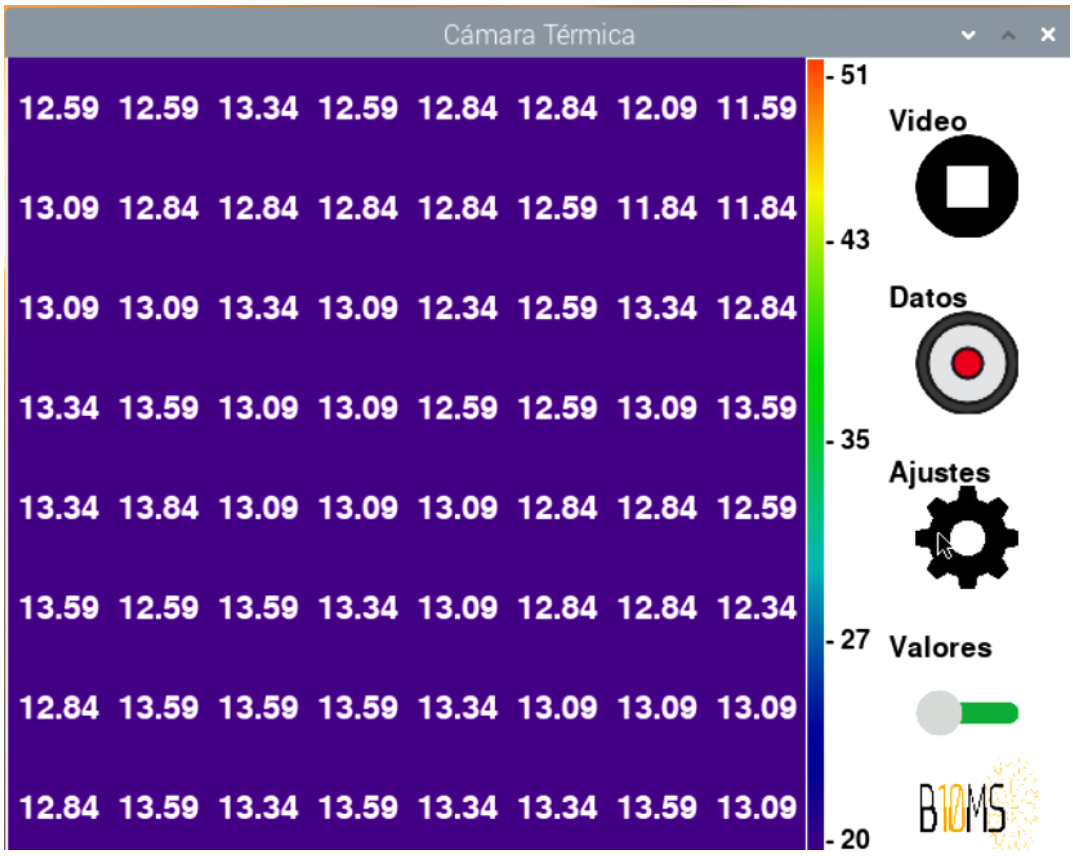
Annexes:

Annex 1: Proposed micro-fluidic system

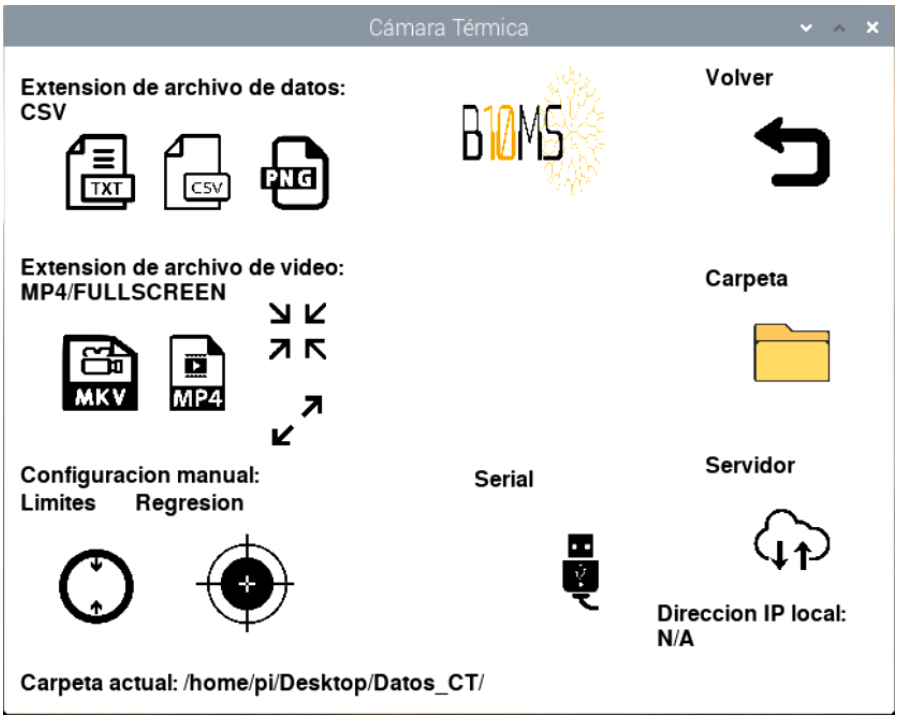


Annex 2: 240x320 thermal imaging interface

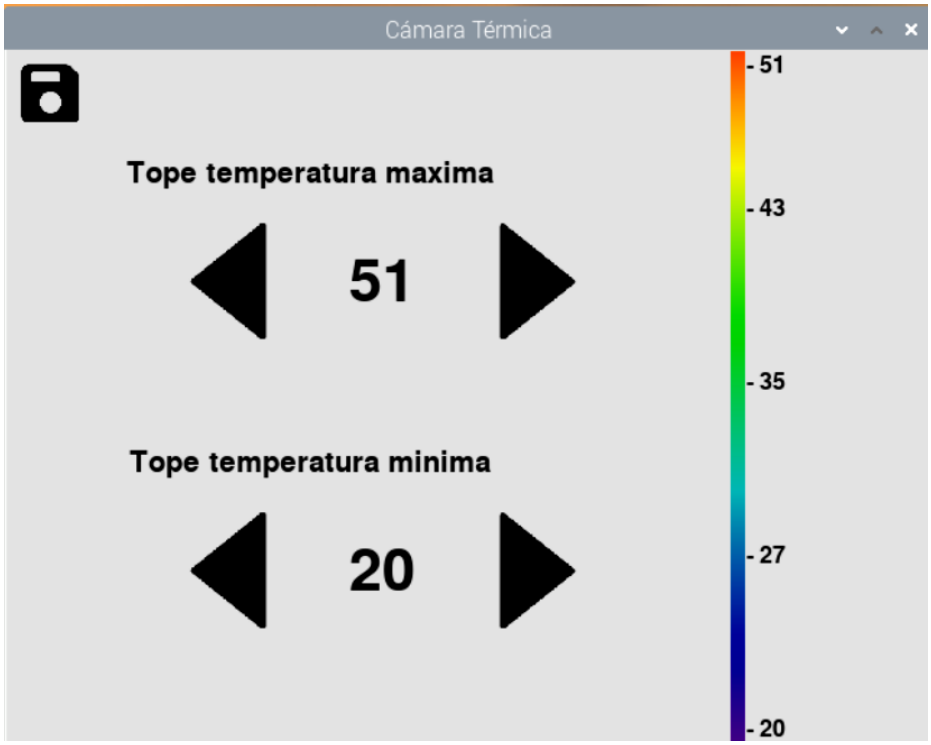
Annex 3: 240x320 settings interface



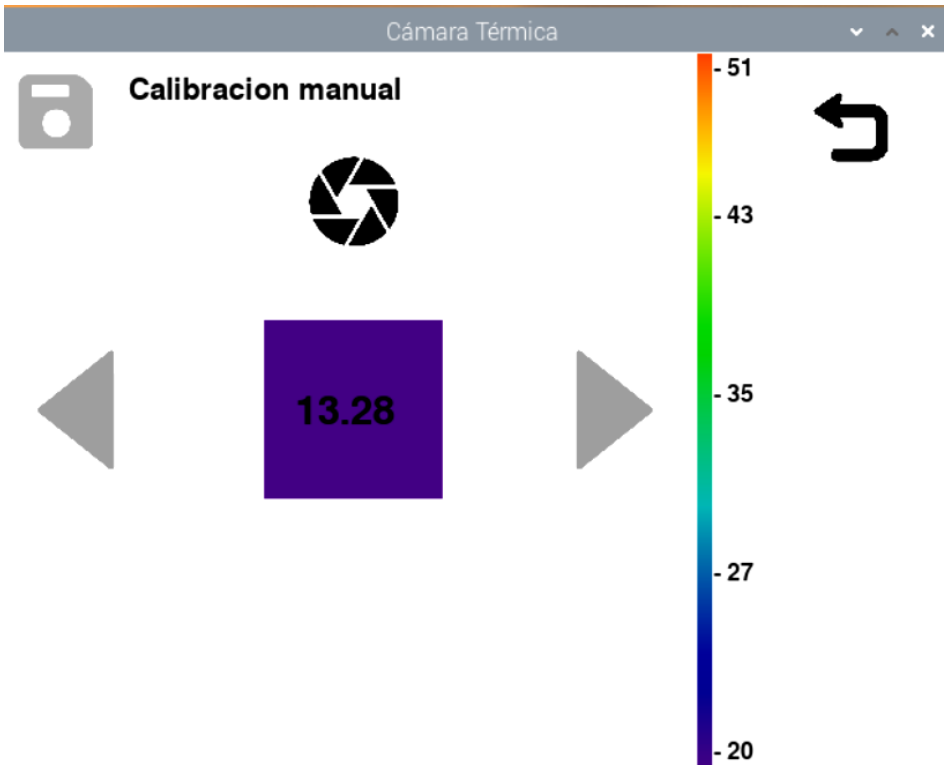
Annex 4a: main screen interface



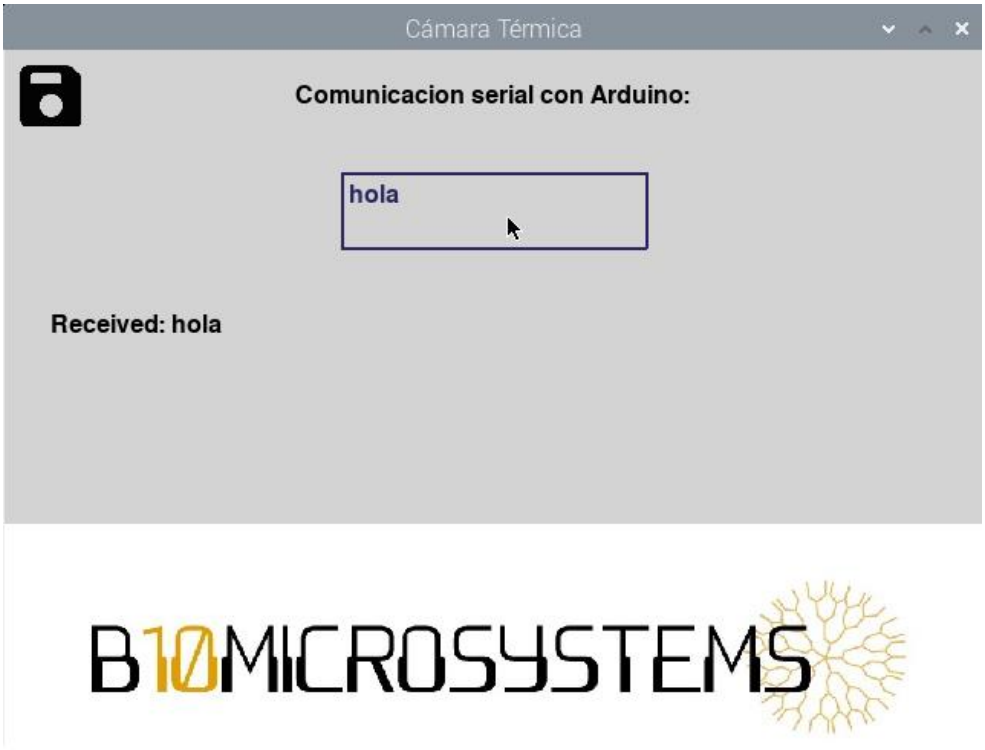
Annex 4b: Settings screen interface



Annex 4c: Temperature limit interface



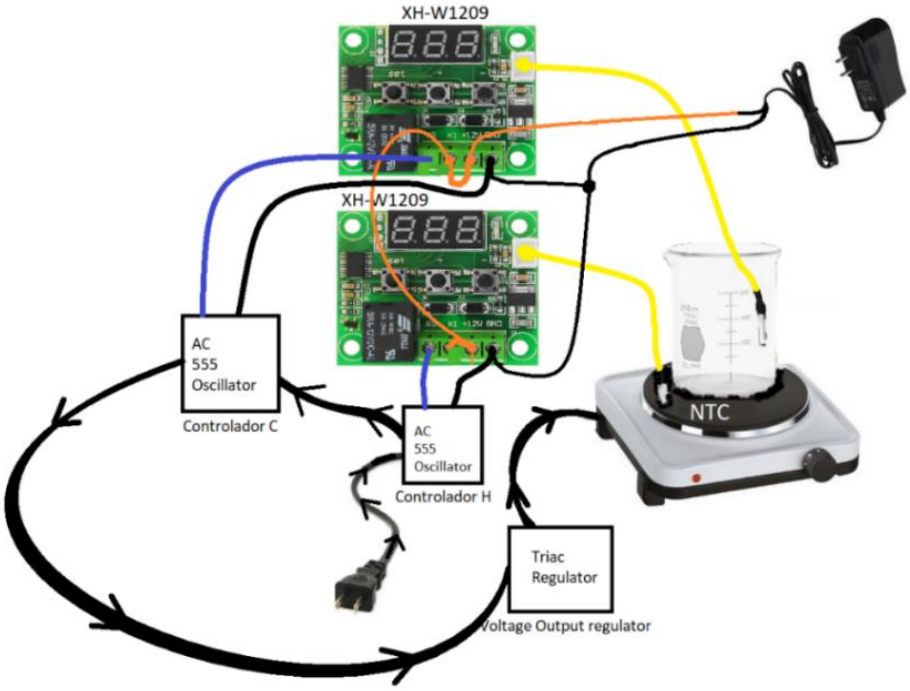
Annex 4d: Manual calibration interface



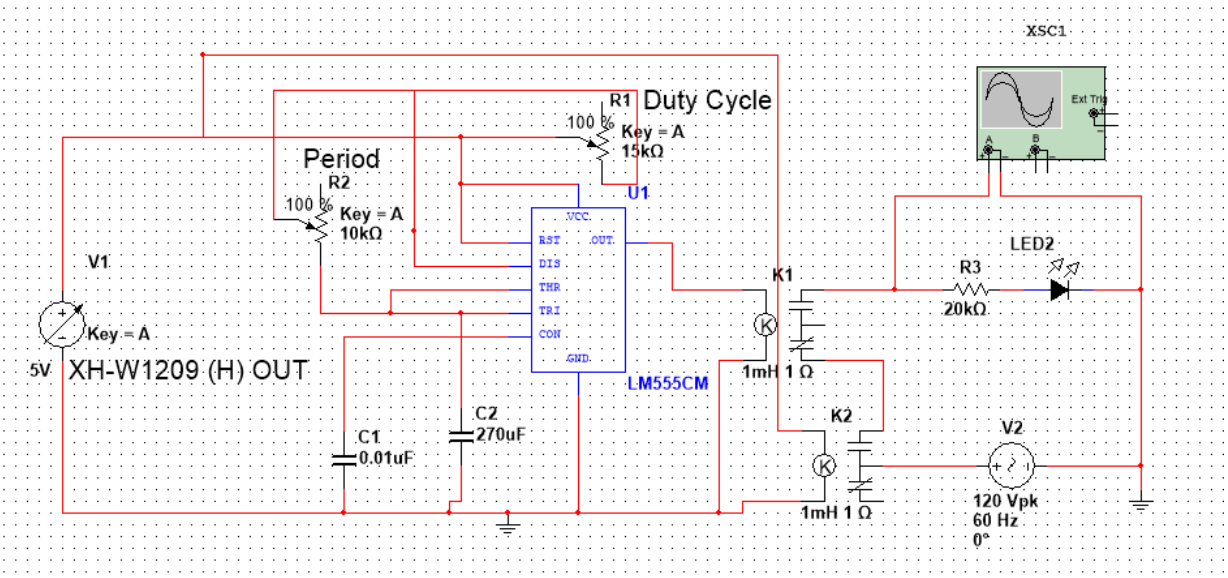
Annex 4e: Serial communication interface

```
Conectado a 192.168.0.18

Valid data types: "TXT, PNG, CSV"
.TXT is default for remote access
Video data types: "MP4, MKV"
.MKV is default for remote access

MAP (color map 640x480),
FULLSCREEN (full screen 640x480),
I_VIDEO (start video recording),
D_VIDEO (stop video recording),
V_EXPORT (export last video recorded)

TEMP (change to interpolation)
VALUES (change to sensor values),
I_DATOS (start data collection),
D_DATOS (stop data collection),
D_EXPORT (export last data collected)

MINTEMP## (change the lower limit),
MAXTEMP## (change the upper limit),
SERIALxxx (send xxx through Arduino serial)

ROOT for folder change
EXPORT change export file
C_EXIT (exit client, server running)
S_EXIT (exit client, server shutdown)
HELP to show again
```

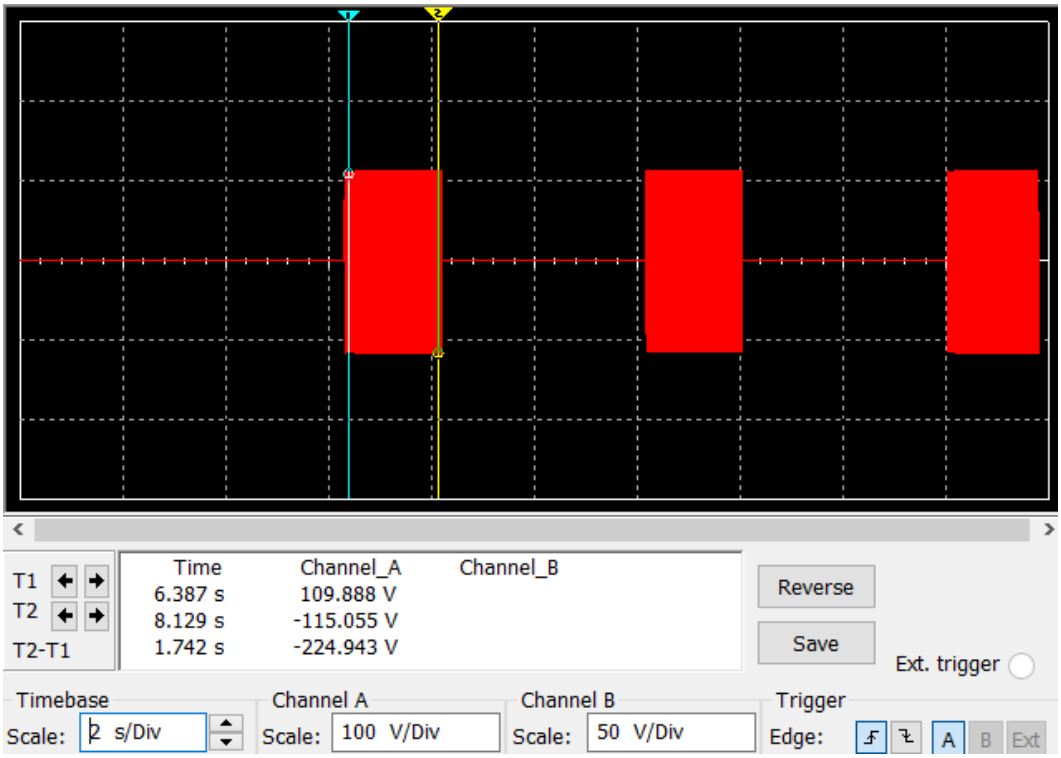
Annex 5: CSM client help menu



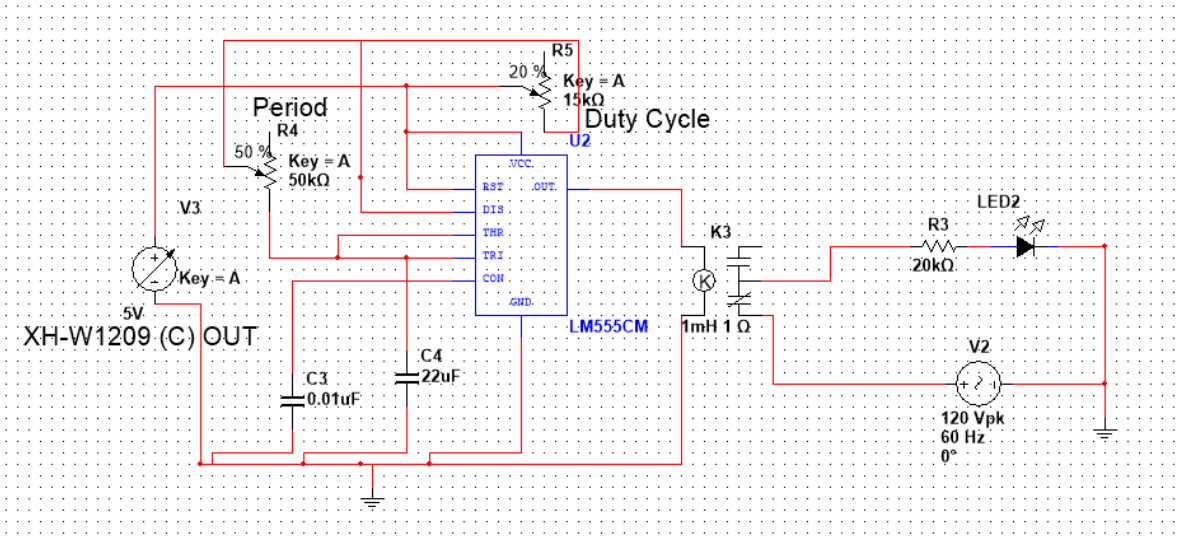
Annex 6: Heating plate diagram



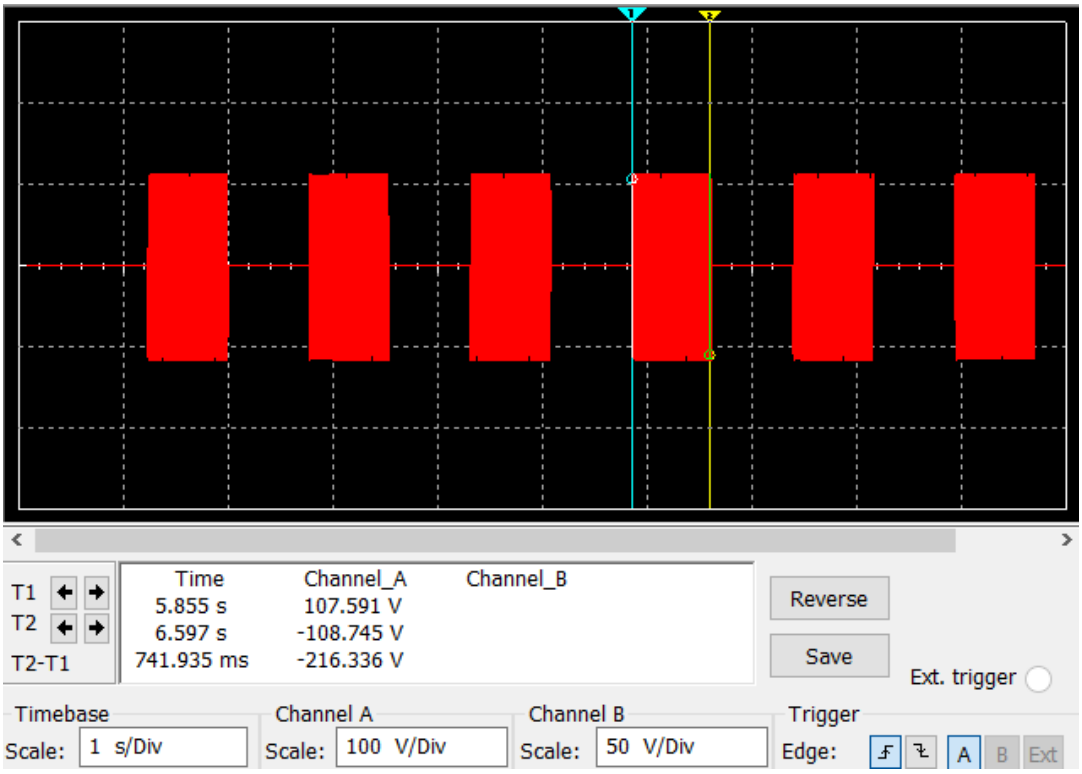
Annex 7: H-oscillator schematic



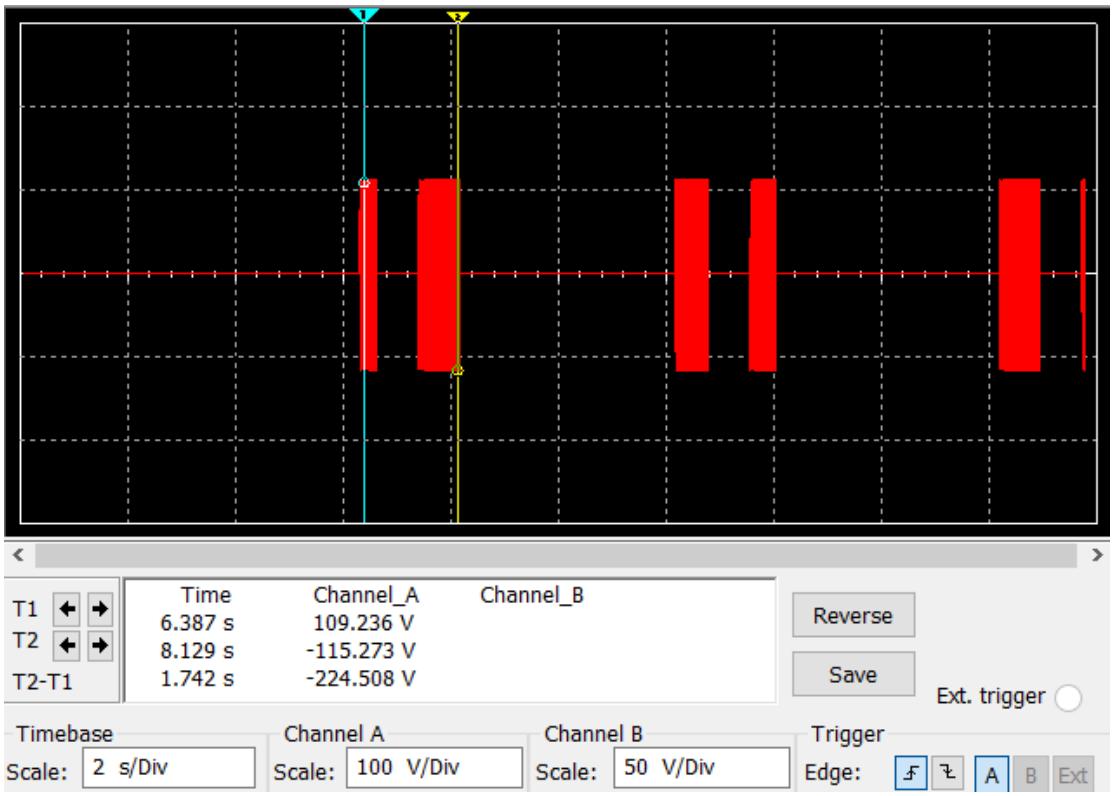
Annex 8: H-oscillator 75% duty cycle.



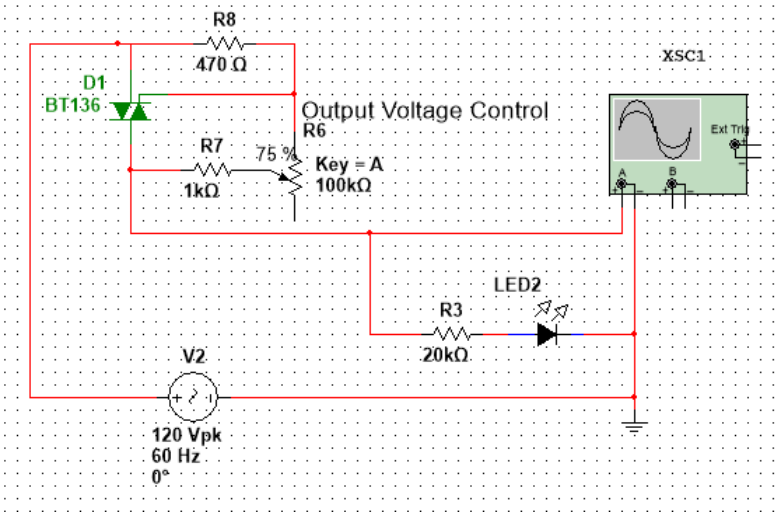
Annex 9: C-oscillator schematic



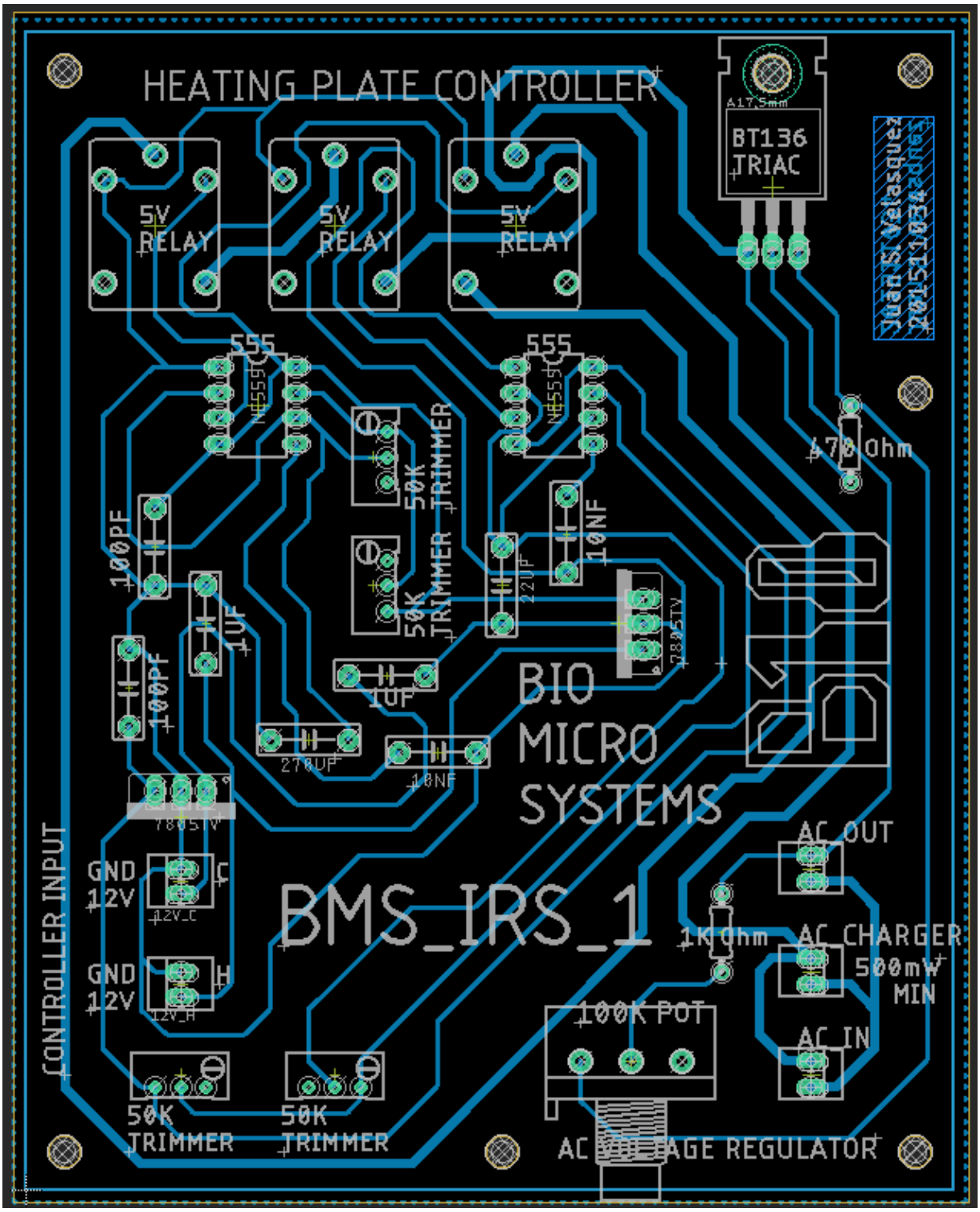
Annex 10: C-oscillator 54% Duty cycle



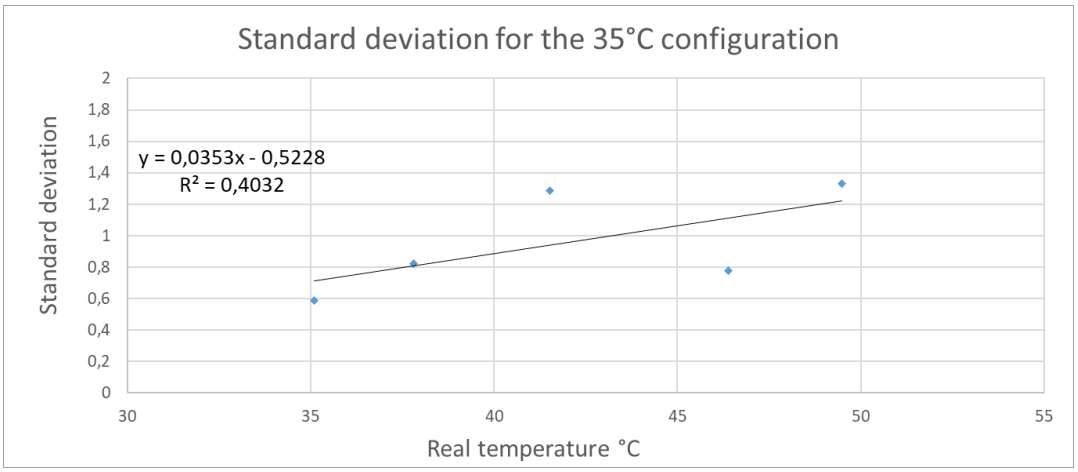
Annex 11: H and C oscillator output



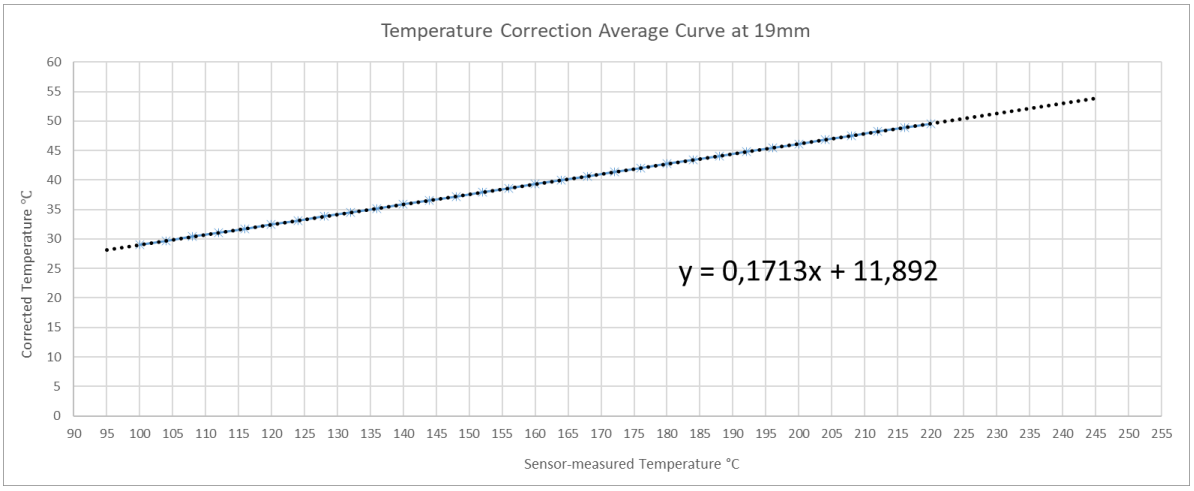
Annex 12: AC voltage regulator schematic



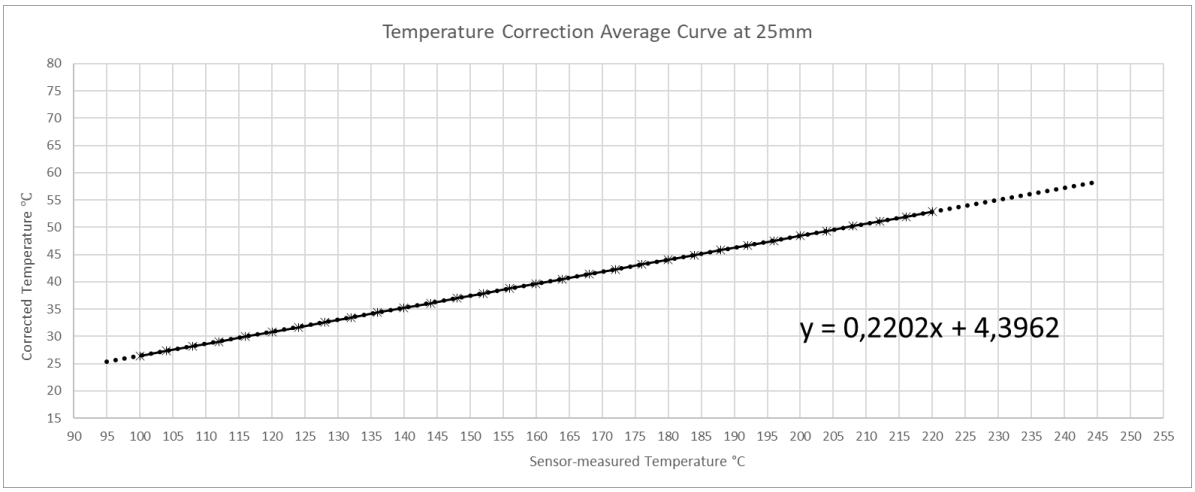
Annex 13: Heating plate PCB



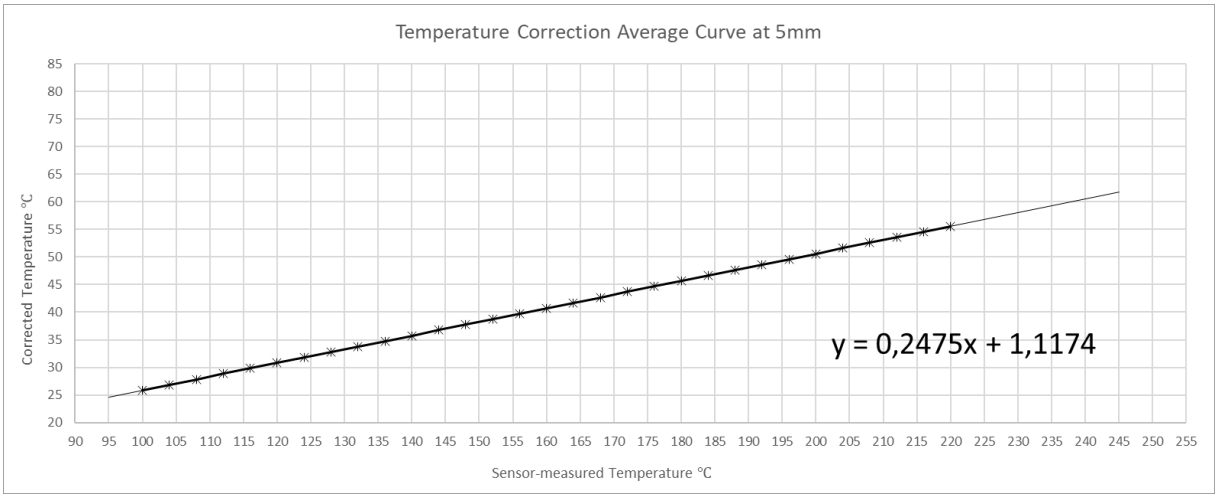
Annex 14: Heating plate standard deviation curve



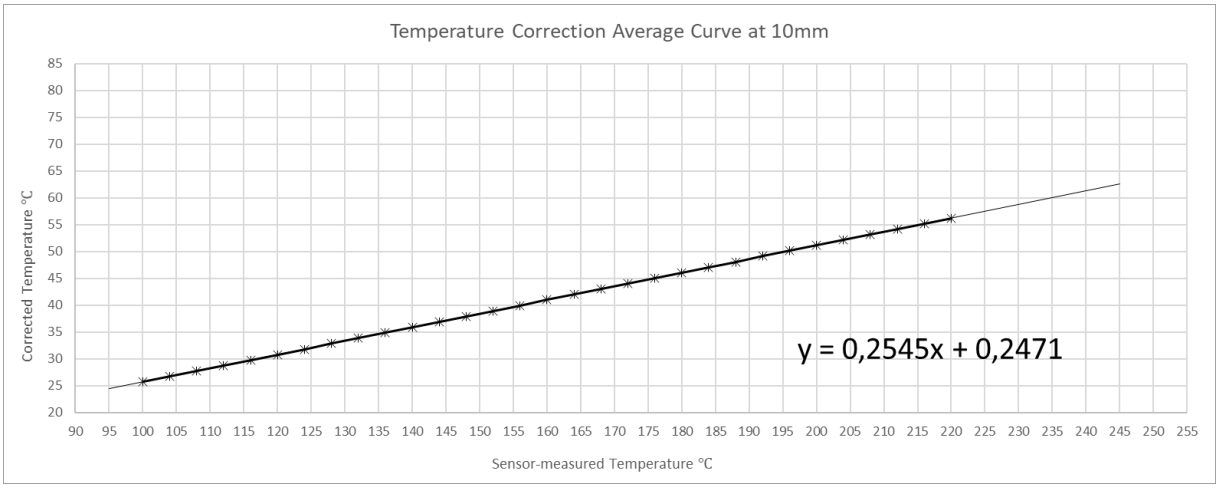
Annex 15: Thermal bath 19mm



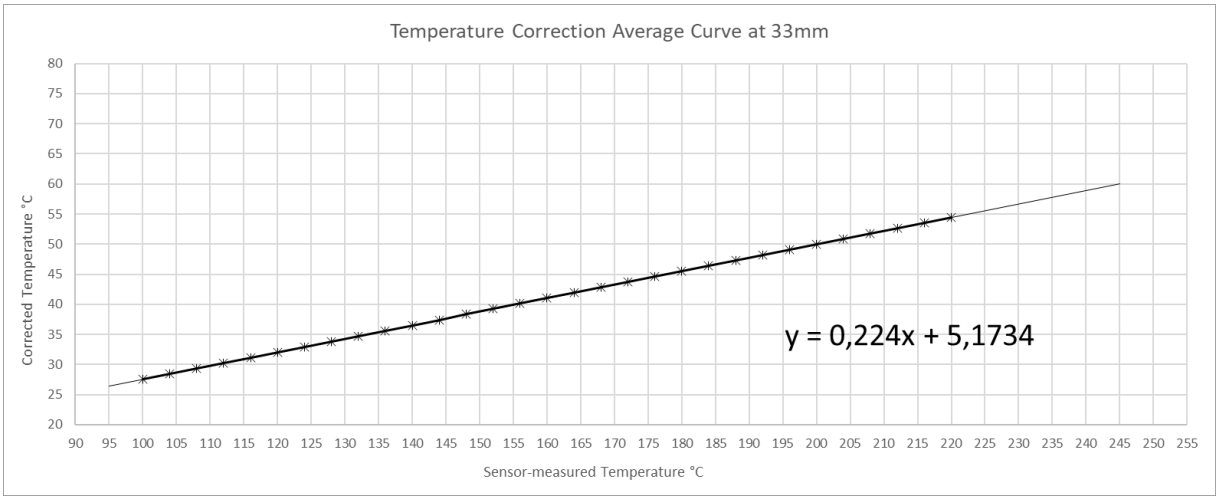
Annex 16: Thermal bath 25mm



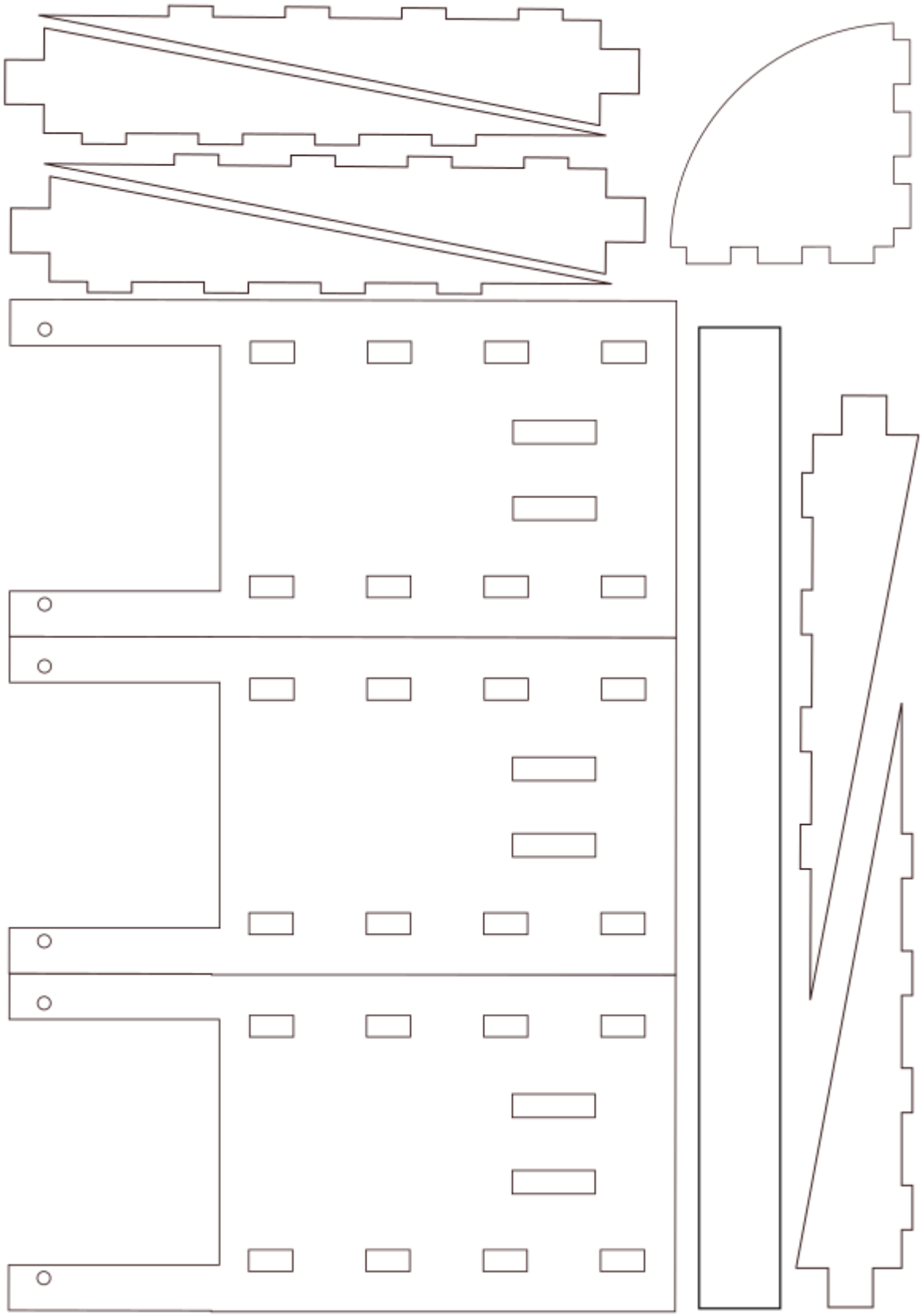
Annex 17: Heating plate 5mm



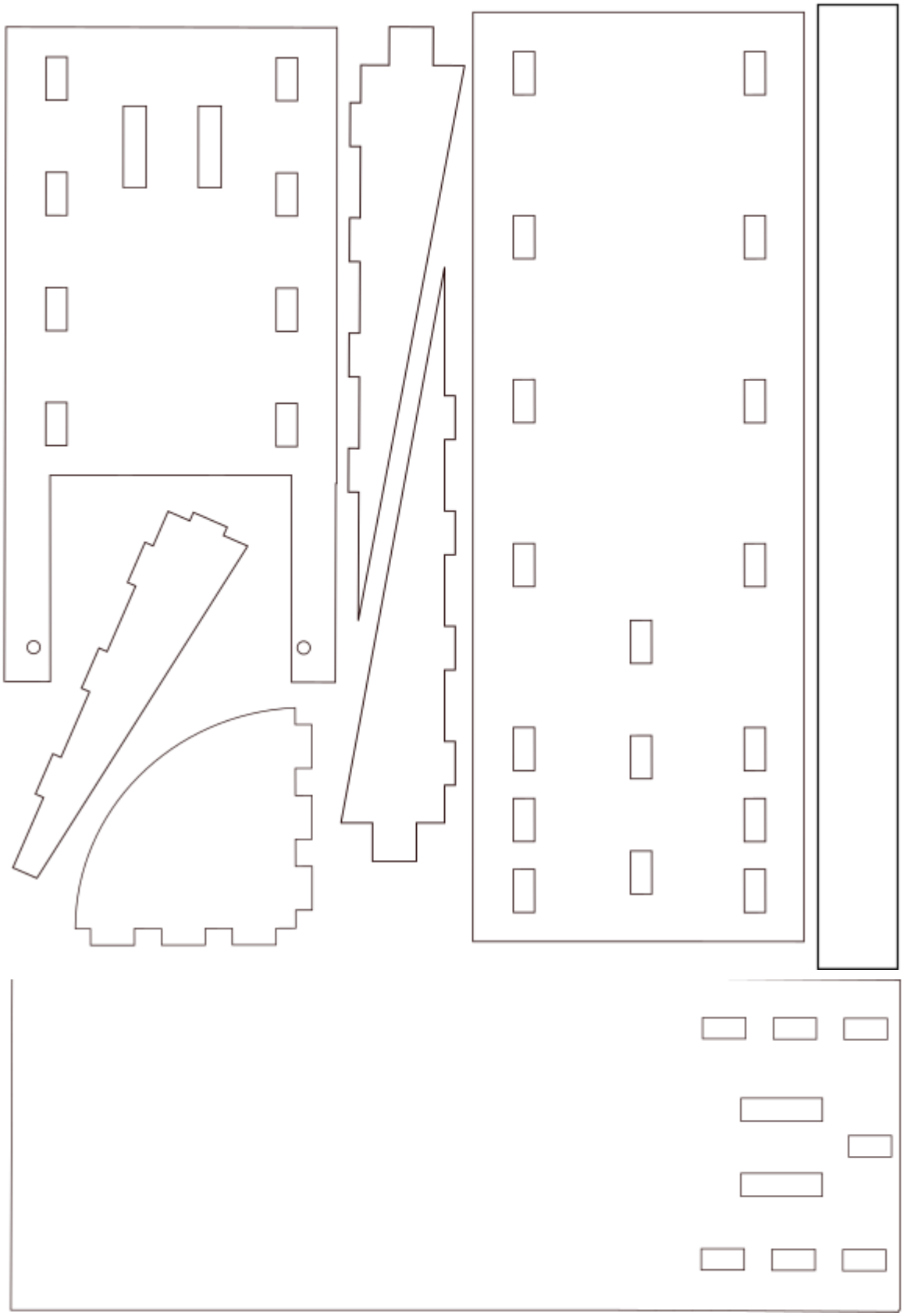
Annex 18: Heating plate 10mm



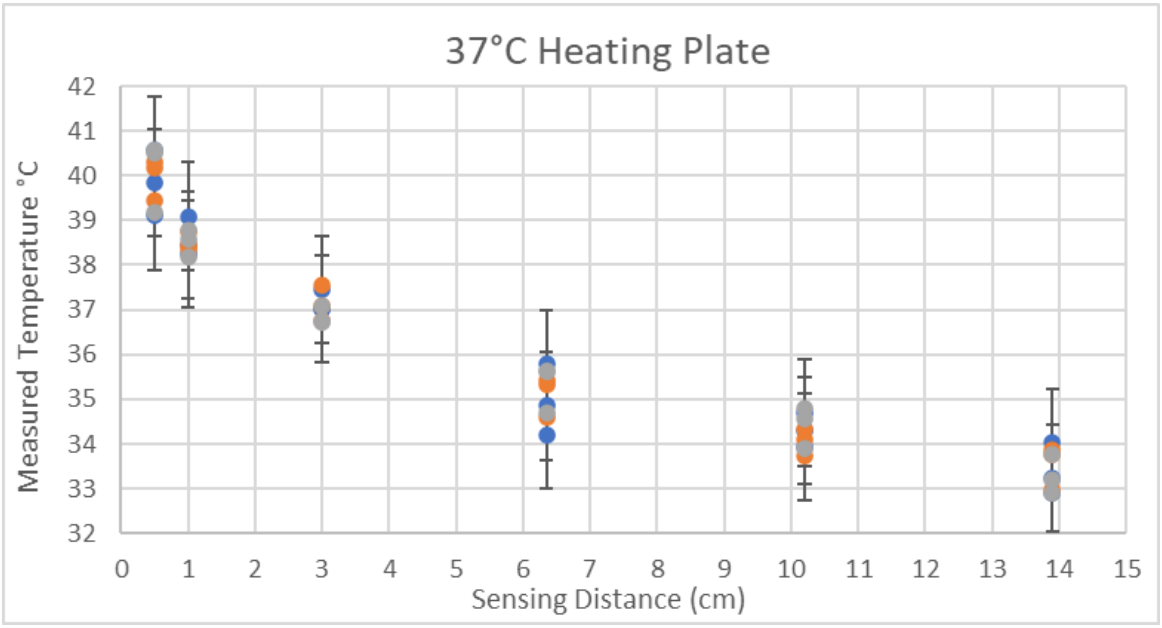
Annex 19: Heating plate 33mm



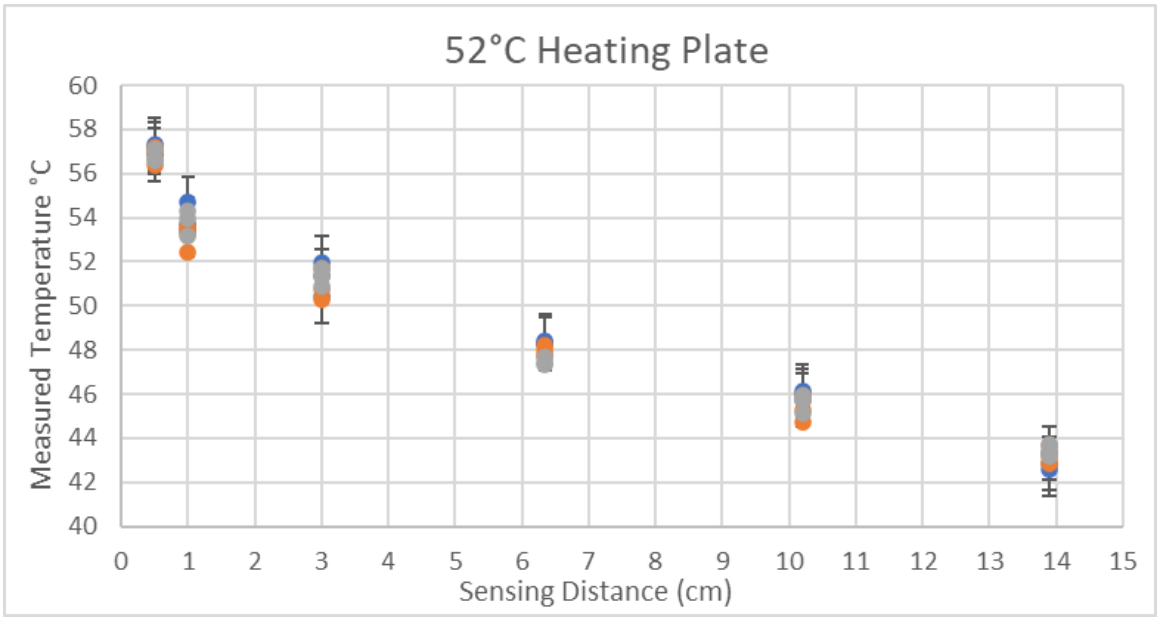
Annex 20: 5mm acrylic distance contraption design



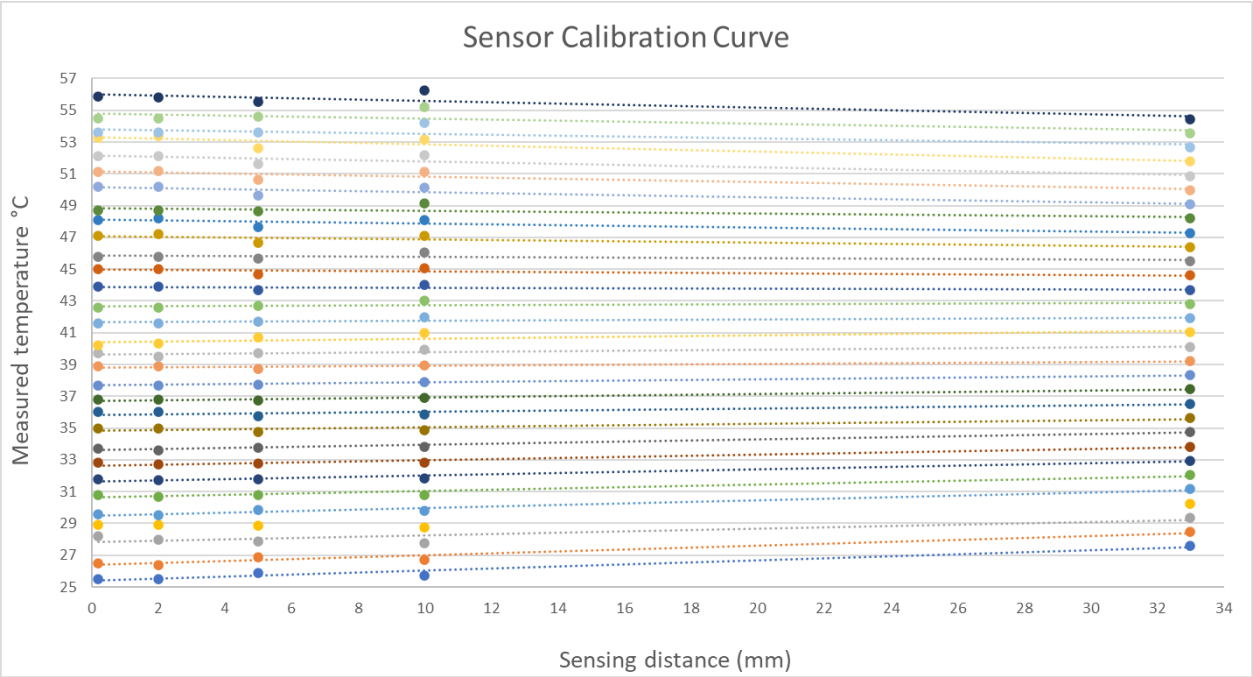
Annex 20: 5mm acrylic distance contraption design



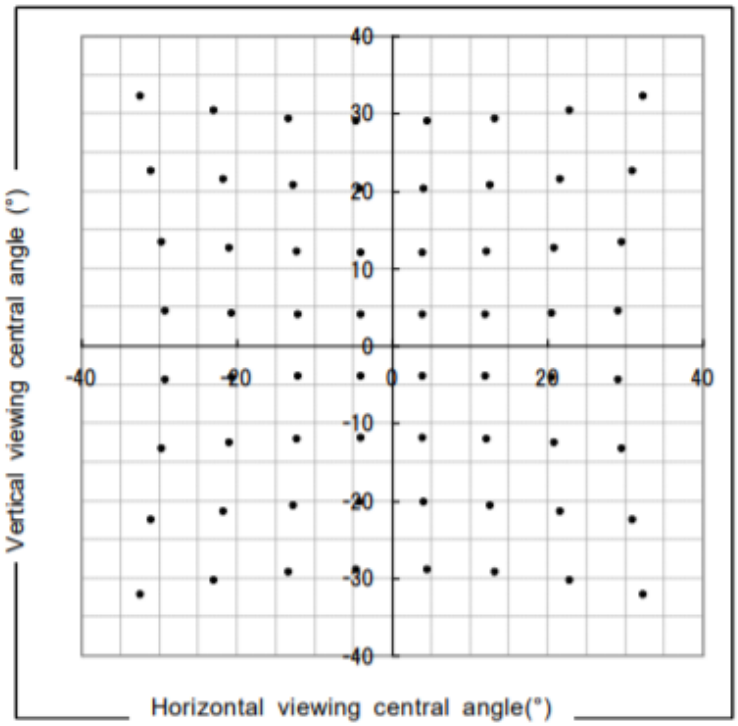
Annex 21: 37°C distance calibration curve



Annex 22: 52°C distance calibration curve



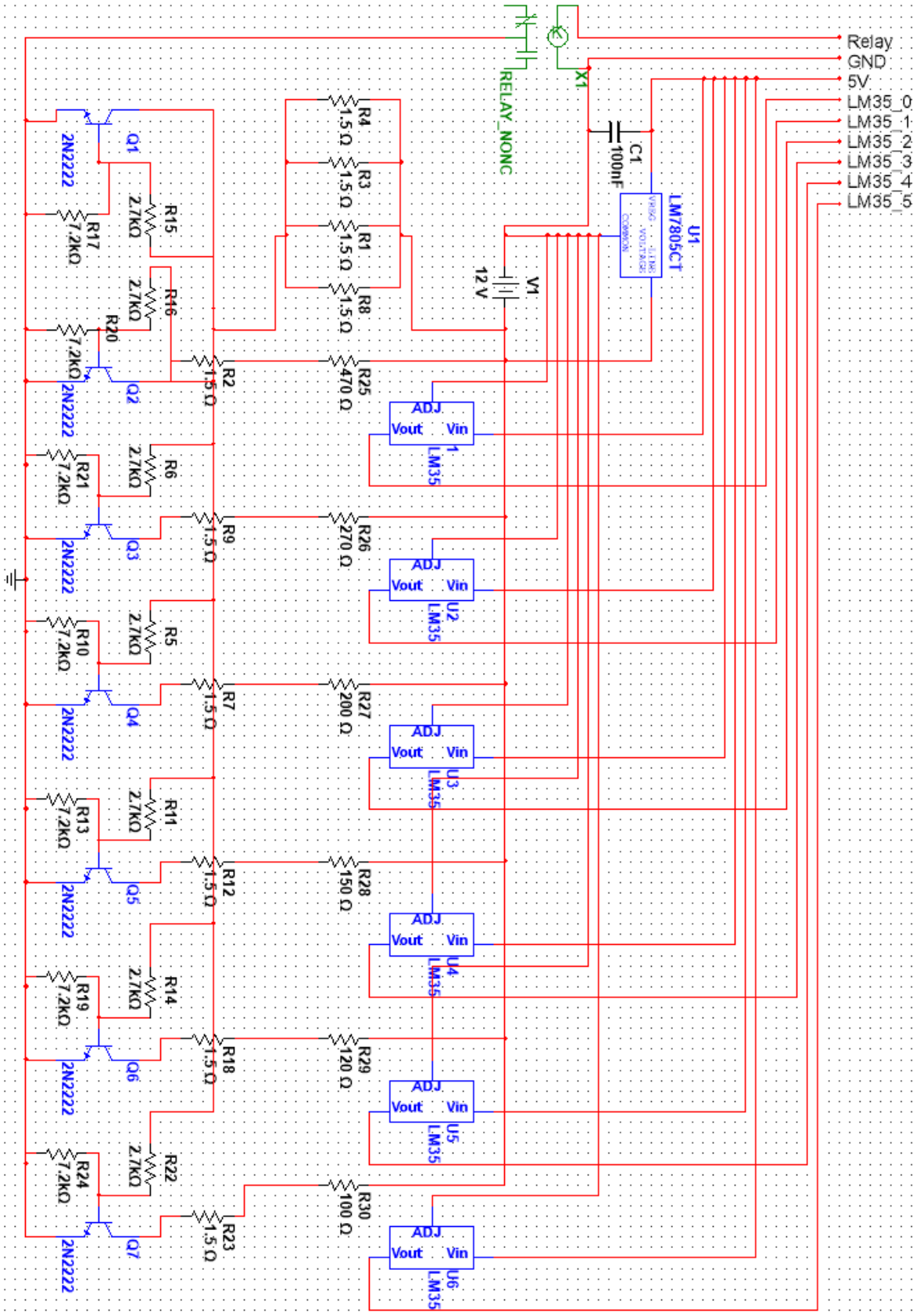
Annex 23: Prediction curves



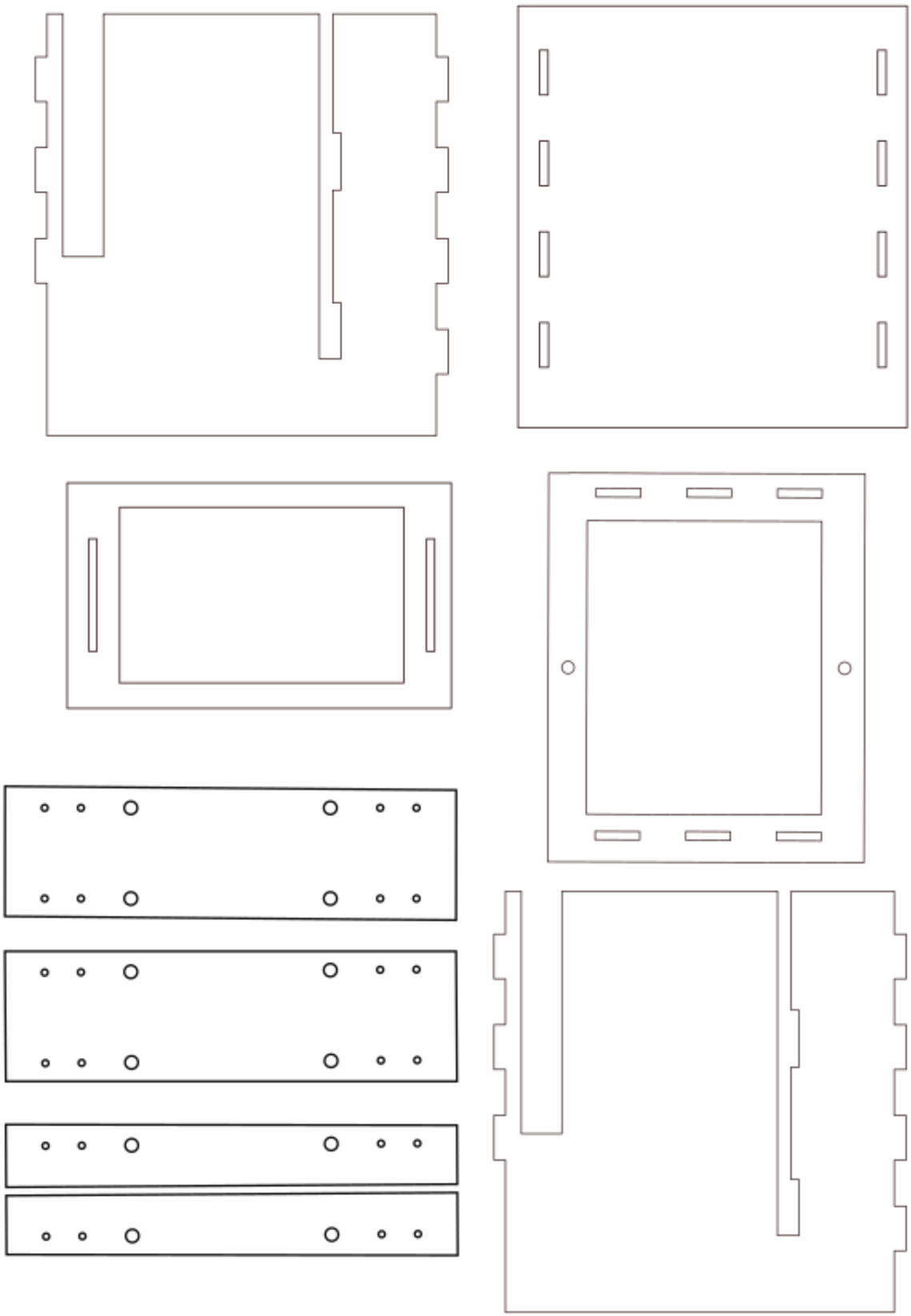
Annex 24: AMG8833 laser distribution

```
import math
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

d = float(raw_input('Distance to the surface (cm): '))
x_angle = [-32.5, -23, -13.5, -4.5, 4.5, 13.5, 23, 32.5, -31, -22, -13, -4, 4, 13, 22, 31, -30, -21, -12.5, -4, 4, 12.5,
21, 30, -29, -21, -12, -4, 4, 12, 21, 29, -29, -21, -12, -4, 4, 12, 21, 29, -30, -21, -12.5, -4, 4, 12.5, 21, 30, -31, -
22, -13, -4, 4, 13, 22, 31, -32.5, -23, -13.5, -4.5, 4.5, 13.5, 23, 32.5]

x_angle_radians = []
for x in x_angle:
x_angle_radians.append(x*math.pi/180)

y_angle = [32.5, 31, 29, 28.5, 28.5, 29, 31, 32.5, 22.5, 22, 21, 20.5, 20.5, 21, 22, 22.5, 13.5, 13, 12.5, 12, 12,
12.5, 13, 13.5, 4.5, 4, 4, 4, 4, 4, 4, 4.5, -4.5, -4, -4, -4, -4, -4, -4, -4.5, -13.5, -13, -12.5, -12, -12, -12.5, -13, -
13.5, -22.5, -22, -21, -20.5, -20.5, -21, -22, -22.5, -32.5, -31, -29, -28.5, -28.5, -29, -31, -32.5]

y_angle_radians = []
for y in y_angle:
y_angle_radians.append(y*math.pi/180)

x_distance = []
for x in x_angle_radians:
x_distance.append(d*math.tan(x))
y_distance = []
for y in y_angle_radians:
y_distance.append(d*math.tan(y))
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
#SensorLens
ax.scatter(0, 0, d, color='r', linewidth = 3, label = 'Sensor')

#SensorPoints
ax.scatter(x_distance, y_distance, 0, color = 'black', linewidth=1, label = 'Pixels')

#SensorLines
for i in range(0, len(x_distance)):
ax.plot([0,x_distance[i]], [0,y_distance[i]], [d, 0], color='r', linewidth=0.2)

ax.set_zbound(lower=0, upper=d)
ax.legend()
plt.title('Points seen by the sensor at\n' + str(d) + ' cm')
plt.xlabel('X distances (cm)')
plt.ylabel('Y distances (cm)')
plt.grid()
plt.show()
```

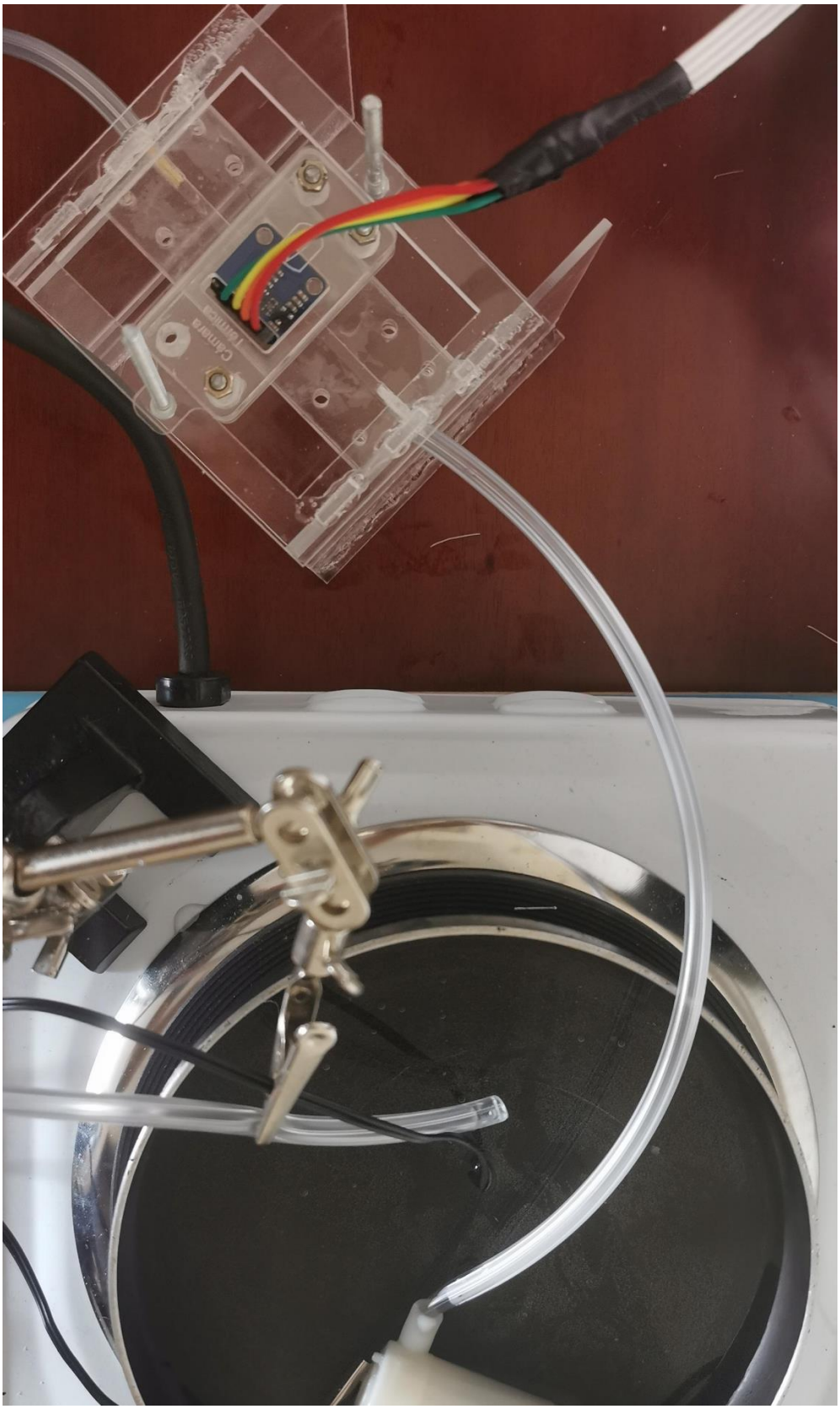
Annex 25: Thermal sensor geometry code



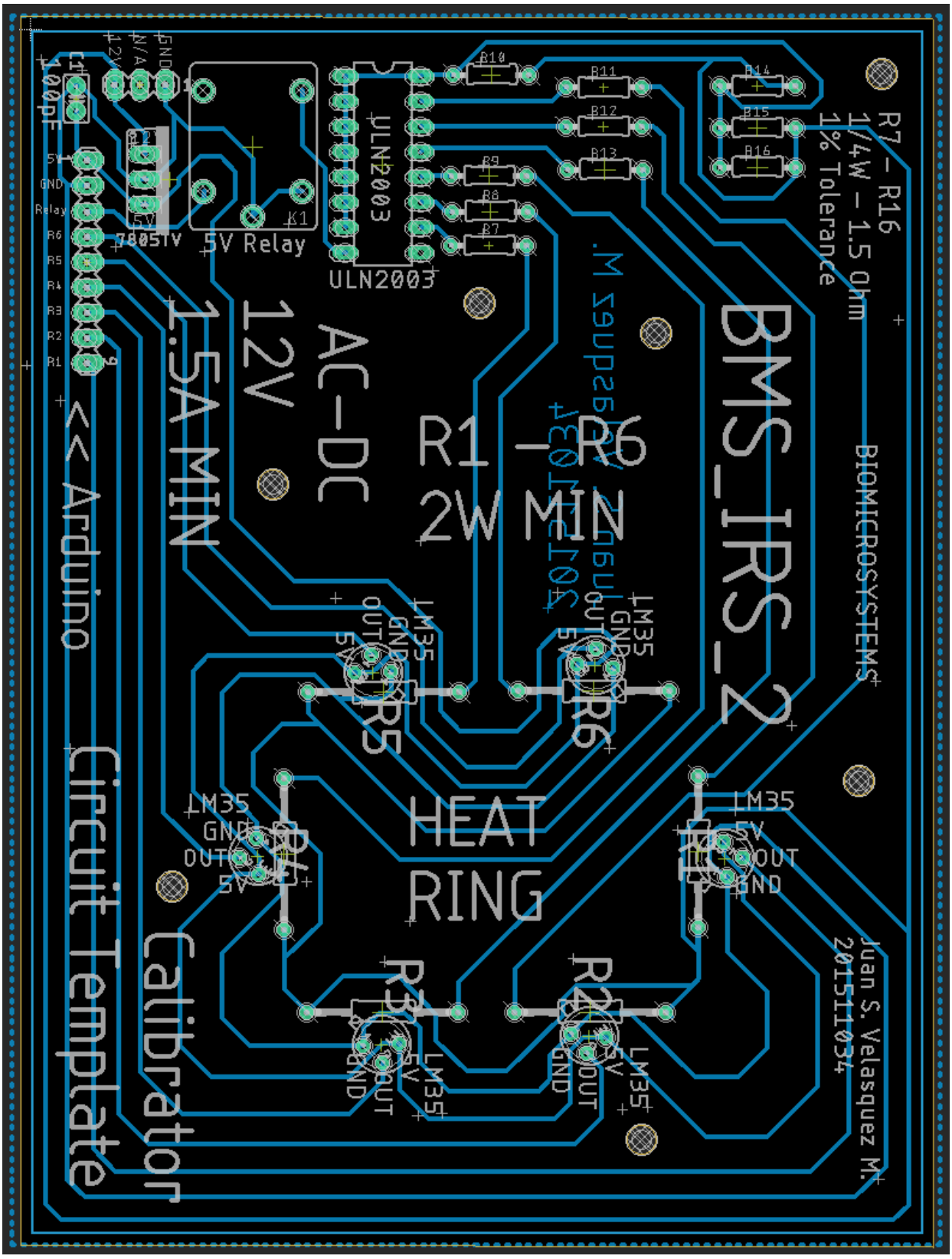
Annex 26: Current mirror schematic



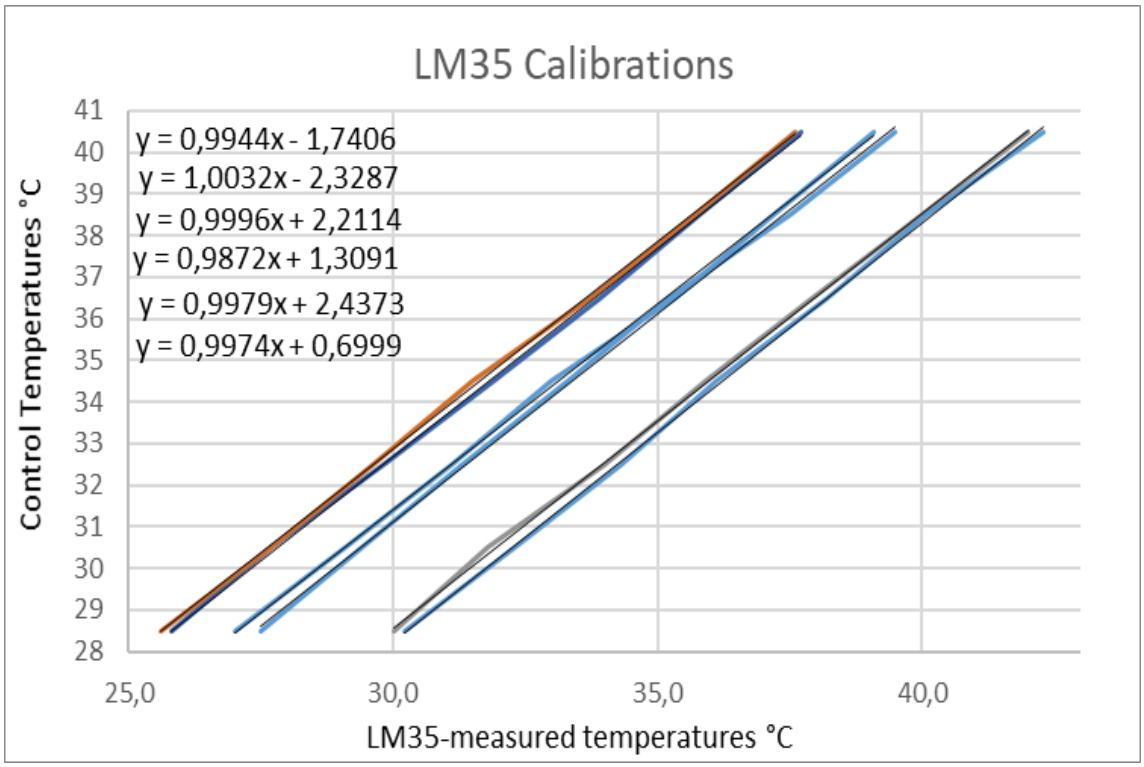
Annex 27: Initial casing



Annex 28: microfluidic system pump

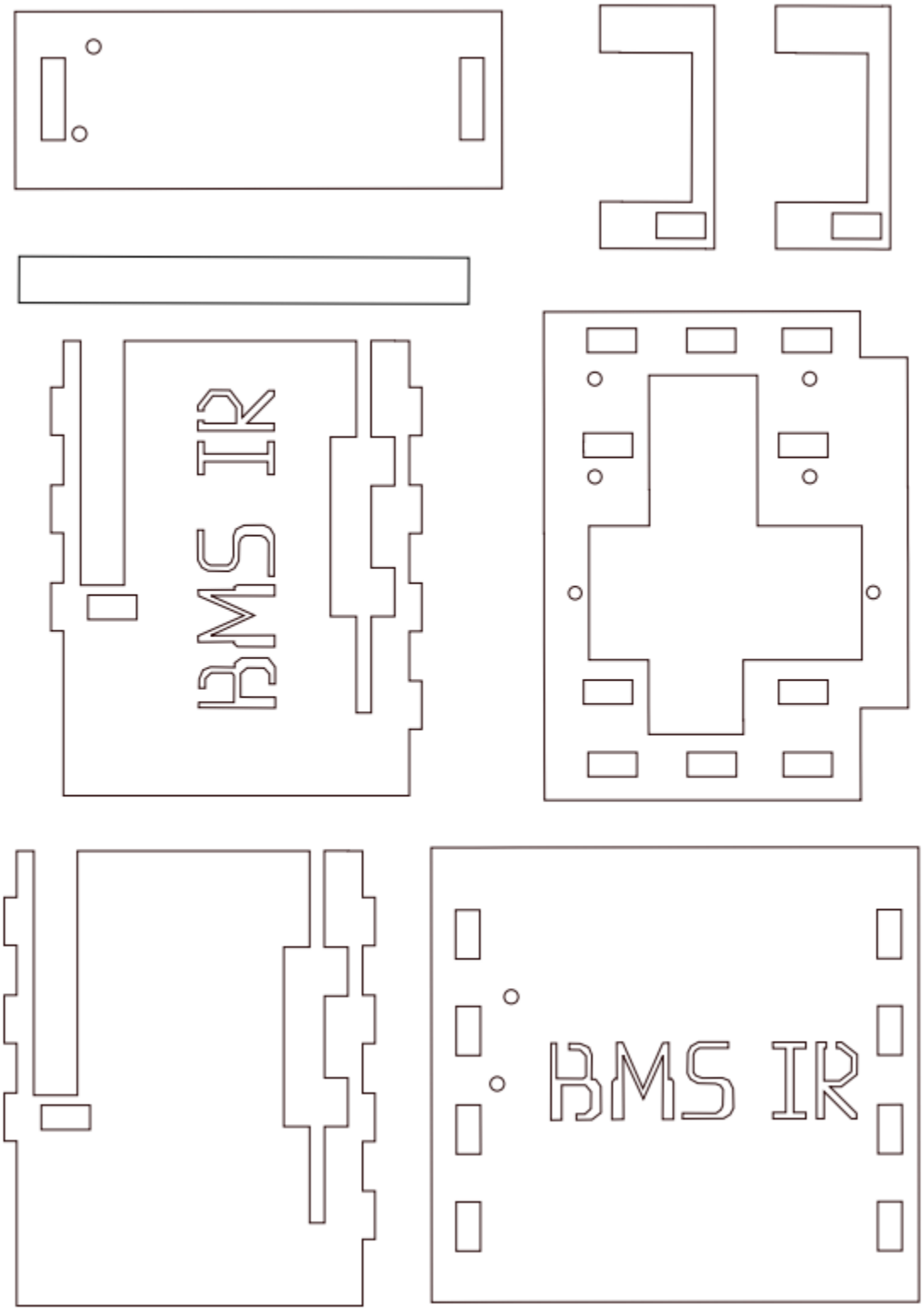


Annex 29: Calibrator circuit PCB

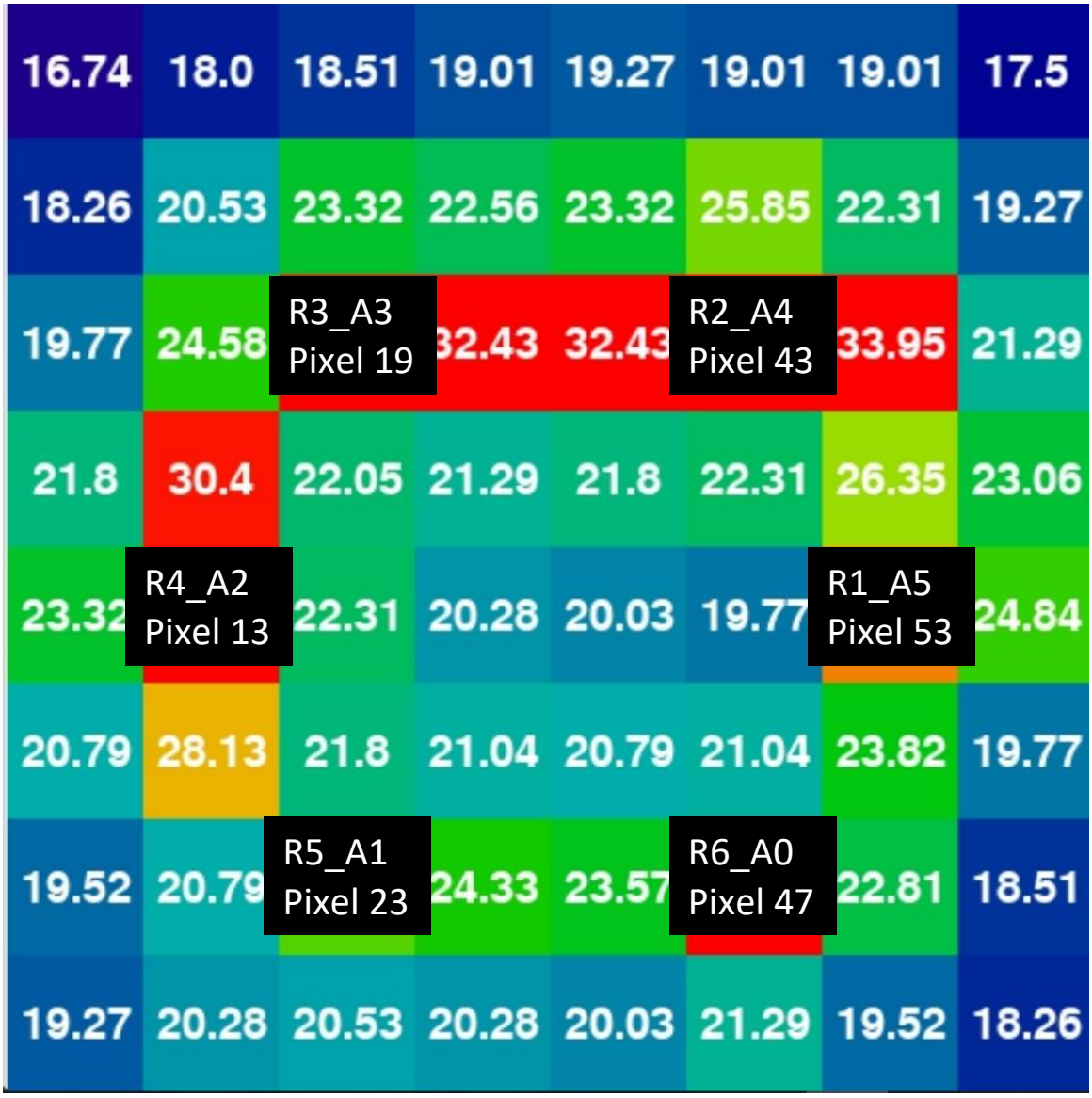


	LM35 °C	LM35 °C	LM35 °C	LM35 °C	LM35 °C	LM35 °C
Control temperature °C	S1	S2	S3	S4	S5	S6
28	30,0	30,2	25,8	27,0	25,6	27,5
30	31,8	32,3	27,7	29,1	27,7	29,4
32	34,0	34,3	29,8	31,1	29,6	31,3
34	35,9	36,1	31,9	33,0	31,5	33,3
36	37,9	38,2	33,9	35,3	33,7	35,2
38	40,0	40,1	35,8	37,2	35,7	37,5
40	42,0	42,3	37,7	39,1	37,6	39,5

Annex 30: LM35s' calibration



Annex 31: Final casing design



Annex 32: Calibrator circuit pixels

```
#include <EEPROM.h>

#define relay 8
#define lm35_0 A0
#define lm35_1 A1
#define lm35_2 A2
#define lm35_3 A3
#define lm35_4 A4
#define lm35_5 A5

String temp;
double temp0;

void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(relay, OUTPUT);
  digitalWrite(relay, LOW);
  pinMode(lm35_0, INPUT);
  pinMode(lm35_1, INPUT);
  pinMode(lm35_2, INPUT);
  pinMode(lm35_3, INPUT);
  pinMode(lm35_4, INPUT);
  pinMode(lm35_5, INPUT);
}

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

    if (temp.startsWith("clc")) {
      temp = "Received: " + temp + ", erasing EEPROM";
      Serial.println(temp);
      for (int i = 0 ; i < EEPROM.length() ; i++) {
        EEPROM.write(i, 0);
      }
    }

    else if (temp.startsWith("hlp")) {
      temp = "Received: " + temp + ", clc (clear EEPROM), get (get temperature data), cal
(calibration)";
      Serial.println(temp);
    }

    else if (temp.startsWith("get")) {
      temp = "Received: " + temp;
      for (int i = 0; i < 12; i++) {
        temp = temp + ", " + EEPROM.read(i);
```

```
  }
  Serial.println(temp);
 }

 else if (temp.startsWith("cal"))  {
  temp = "Received: " + temp + ", starting the calibration";
  Serial.println(temp);

  digitalWrite(LED_BUILTIN, HIGH);
  digitalWrite(relay, HIGH);
  for (int i = 0; i < 20; i++) {
   delay(1250);
   digitalWrite(LED_BUILTIN, LOW);
   delay(1250);
   digitalWrite(LED_BUILTIN, HIGH);
  }
  digitalWrite(LED_BUILTIN, LOW);
  digitalWrite(relay, LOW);
  temp0 = analogRead(lm35_0);
  temp0 = (5.0 * temp0 * 100.0) / 1023.0;
  temp0 = (0.9944 * temp0) - 1.7406;
  EEPROM.write(0, temp0);
  EEPROM.write(1, (temp0 - EEPROM.read(0)) * 100);
  temp0 = analogRead(lm35_1);
  temp0 = (5.0 * temp0 * 100.0) / 1023.0;
  temp0 = (1.0032 * temp0) - 2.3287;
  EEPROM.write(2, temp0);
  EEPROM.write(3, (temp0 - EEPROM.read(2)) * 100);
  temp0 = analogRead(lm35_2);
  temp0 = (5.0 * temp0 * 100.0) / 1023.0;
  temp0 = (0.9996 * temp0) + 2.2114;
  EEPROM.write(4, temp0);
  EEPROM.write(5, (temp0 - EEPROM.read(4)) * 100);
  temp0 = analogRead(lm35_3);
  temp0 = (5.0 * temp0 * 100.0) / 1023.0;
  temp0 = (0.9872 * temp0) + 1.3091;
  EEPROM.write(6, temp0);
  EEPROM.write(7, (temp0 - EEPROM.read(6)) * 100);
  temp0 = analogRead(lm35_4);
  temp0 = (5.0 * temp0 * 100.0) / 1023.0;
  temp0 = (0.9979 * temp0) + 2.4373;
  EEPROM.write(8, temp0);
  EEPROM.write(9, (temp0 - EEPROM.read(8)) * 100);
  temp0 = analogRead(lm35_5);
  temp0 = (5.0 * temp0 * 100.0) / 1023.0;
  temp0 = (0.9974 * temp0) + 0.6999;
  EEPROM.write(10, temp0);
  EEPROM.write(11, (temp0 - EEPROM.read(10)) * 100);
```

```
   }

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

Annex 33: Arduino code

```
# -*- coding: utf-8 -*-

import socket
import os
from requests import get
from time import sleep

#ip = get('https://api.ipify.org').text
#print 'My public IP address is:', ip

#h = socket.gethostname()
#ip=socket.gethostbyname(h)

s = socket.socket()
host = '192.168.0.18'
nombre = f = ''
port = 8081
videoFile = 'mkv'
dataFile = 'txt'
done = False
out = False
initialize = True
s.connect((host, port))
while True:
   try:
      print(s.recv(1024))
      if initialize:
          print('\n\rValid data types: "TXT, PNG, CSV"\n\r.TXT is default for remote
access\n\rVideo data types: "MP4, MKV"\n\r.MKV is default for remote access\n\n\rMAP
(color map 640x480),\nFULLSCREEN (full screen 640x480),\nI_VIDEO (start video
recording),\nD_VIDEO (stop video recording),\nV_EXPORT (export last video
recorded)\n\n\rTEMP (change to interpolation)\n\rVALUES (change to sensor values),
\n\rI_DATOS (start data collection),\nD_DATOS (stop data collection),\nD_EXPORT (export last
data collected)\n\n\rMINTEMP## (change the lower limit), \n\rMAXTEMP## (change the upper
limit),\n\rSERIALxxx (send xxx through Arduino serial)\n\n\rROOT for folder change\n\rEXPORT
```

```
change export file\n\rC_EXIT (exit client, server running)\n\rS_EXIT (exit client, server
shutdown)\n\rHELP to show again\n')
            s.send(str.encode('TXT'))
            print('Data extension changed to TXT as per default')
            initialize = False
            print(s.recv(1024))
        b = str.encode(raw_input('Waiting for Command: '))
        if b != 'HELP':
                s.send(b)
                if b == 'MP4':
                        videoFile = 'mp4'
                        print('Video extension changed to MP4')
                    elif b == 'MKV':
                        videoFile = 'mkv'
                        print('Video extension changed to MKV')
                    elif b == 'TXT':
                        dataFile = 'txt'
                        print('Data extension changed to TXT')
                    elif b == 'CSV':
                        dataFile = 'csv'
                        print('Data extension changed to CSV')
                    elif b == 'PNG':
                        dataFile = 'png'
                        print('Data extension changed to PNG')

                    elif b == 'V_EXPORT':
                        videoFile = str(s.recv(1024)).lower()
                        while(True):
                            try:
                                nombre = str(raw_input('Nombre del archivo:')).strip()
                                nombre = (nombre + '.' + videoFile).strip()
                                print(nombre)
                                f = open(nombre,'w')
                                break
                            except Exception as e:

                                print(e)
                                #print('Error parcing the file, try again')
                        l = s.recv(1024)
                        while(not out):
                            f.write(l)
                            if done:
                                break
                            l = s.recv(1024)
                            if(str(l).find('Done') > 0):
                                done = True
                        done = False
                        f.flush()
```

```
                f.close()
            elif b == 'D_EXPORT':
                dataFile = str(s.recv(1024)).lower()
                while(True):
                    try:
                        nombre = str(raw_input('Nombre del archivo:')).strip()
                        nombre = (nombre + '.' + dataFile).strip()
                        print(nombre)
                        f = open(nombre,'w')
                        break
                    except Exception as e:
                        print(e)
                l = s.recv(1024)
                while(not out):
                    f.write(l)
                    if done:
                        break
                    l = s.recv(1024)
                    if(str(l).find('Done') > 0):
                        done = True
                done = False
                f.flush()
                f.close()
            elif b == 'C_EXIT':
                print('Disconnected from ' + host)
                sleep(5)
                s.close()
                break
            elif b == 'S_EXIT':
                print('Server: '+ host + ' disconnected')
                sleep(5)
                s.close()
                break
    else:
        print('\n\rValid data types: "TXT, PNG, CSV"\n\r.TXT is default for remote
access\n\rVideo data types: "MP4, MKV"\n\r.MKV is default for remote access\n\n\rMAP
(color map 640x480),\nFULLSCREEN (full screen 640x480),\nI_VIDEO (start video
recording),\nD_VIDEO (stop video recording),\nV_EXPORT (export last video
recorded)\n\n\rTEMP (change to interpolation)\n\rVALUES (change to sensor values),
\n\rI_DATOS (start data collection),\nD_DATOS (stop data collection),\nD_EXPORT (export last
data collected)\n\n\rMINTEMP## (change the lower limit), \n\rMAXTEMP## (change the upper
limit),\n\rSERIALxxx (send xxx through Arduino serial)\n\n\rROOT for folder change\n\rEXPORT
```

```
change export file\n\rC_EXIT (exit client, server running)\n\rS_EXIT (exit client, server
shutdown)\n\rHELP to show again\n')
   except Exception as e:
          print(e)
          s.close()
       break
s.close()
```

Annex 34: Client code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

##############################################################################Librerias

import serial
import pygame
import select
import socket
import re
import os
import math
import time
import threading
import numpy as np
import glob
import csv
import subprocess
import signal
import spidev
import RPi.GPIO as GPIO
import Tkinter, tkFileDialog
from scipy.interpolate import griddata
from scipy.stats import linregress
from pygame.locals import *
from time import time
from colour import Color
from os import listdir
from botonesPrincipales import *
from Adafruit_AMG88xx import Adafruit_AMG88xx
from time import sleep
from requests import get

##############################################################################CLASE
PARA INPUTBOX
global sensibility
sensibility = 0
```

```
#initialize the sensor
global sensor
sensor = Adafruit_AMG88xx()

class InputBox:
    global sensor, sensibility
    sensor = Adafruit_AMG88xx()
    def __init__(self, x, y, w, h, text=''):
        self.rect = pygame.Rect(x, y, w, h)
        self.color = YELLOW
        self.text = text
        self.txt_surface = textFont_temp.render(text, True, self.color)
        self.active = False
        self.ser = serial.Serial('/dev/ttyACM0', 115200, timeout=1)
        self.ser.flush()
        self.line = ''

    def handle_event(self, event):
        global sensor
        if event.type == pygame.MOUSEBUTTONDOWN:
            if self.rect.collidepoint(event.pos):
                self.active = not self.active
            else:
                self.active = False
            self.color = AZUL if self.active else YELLOW
        if event.type == pygame.KEYDOWN:
            if self.active:
                if event.key == pygame.K_RETURN:
                    self.text = bytes(self.text+'\n')
                    self.ser.write(self.text)
                    self.line = self.ser.readline().decode('utf-8').rstrip()
                    if self.text == "get\n":
                        pixelsMap = sensor.readPixels()
                        x = []
                        x.append(pixelsMap[11])
                        x.append(pixelsMap[26])
                        x.append(pixelsMap[43])
                        x.append(pixelsMap[45])
                        x.append(pixelsMap[31])
                        x.append(pixelsMap[13])
                        a = self.line
                        correction = [0,0,0,0,0,0]
                        y = []
                        for i in range(0, 6):
                            b = a.split(",", 2)[1].strip()
                            a = a.split(",", 1)[1]
                            c = a.split(",", 2)[1].strip()
```

```
                    a = a.split(",", 1)[1]
                    y.append(float(b + "." + c))
                    x[i] = (0.251*4*x[i]) + 2.4622
                    correction[i] = (x[i] - y[i])*(1+sensibility)
                    x[i] = (x[i] + 5.311)/ (0.2531*4)
                    x[i] = (x[i] - correction[i])*4
                xParameter, bParameter = regLineal(x, y)
                guardarVariables(xParameter, bParameter, fileNameCalibration)

            sleep(1)
            self.text = ''
        elif event.key == pygame.K_BACKSPACE:
            self.text = self.text[:-1]
        else:
            self.text += event.unicode
        # Re-render the text.
        self.txt_surface = textFont_temp.render(self.text, True, self.color)

    def update(self):
        width = max(200, self.txt_surface.get_width()+10)
        self.rect.w = width

    def draw(self, lcd):
        lcd.blit(self.txt_surface, (self.rect.x+5, self.rect.y+5))
        pygame.draw.rect(lcd, self.color, self.rect, 2)

################################################################################Atributos

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

calib_scale240 = float(240)/3788   # Likely about 285
calib_scale320 = float(320)/3777   # Likely about 384
calib_offset240 = calib_scale240 * 180   # Likely about 28
calib_offset320 = calib_scale320 * 318   # Likely about 25

################################################################################Variables
del sistema

salirMainScrn = False
corriendoMainScrn = False
salir = False
imprimirTem = True
grabandoCSV = False
salirGrabacionCSV = True
grabandoVideo = False
grabarVideo = False
configurar = False
```

```
configurando = False
confiMax = False
confiMin = False
confiCal = False
configurandoCal = False
tomaHabilitada = True
archivoVideoNuevo = ''
archivoDatosNuevo = ''

calX = []
calY = []

promedioTem = 0
promedioDatos = 0

box_active = False
input_box = pygame.Rect(79, 185, 82, 31)

color_inactive = (0,0,0)
color_active = pygame.Color('dodgerblue2')

csv_pixels = []
tiempo_inicio_csv = time()

################################################################################Variables
de configuración

MINTEMP = 20
MAXTEMP = 30

servidor = False
servidorThread = False
extActual = 'CSV'
extActual2 = 'MKV'
videoCon = 'MAP'
ip = 'N/A'
newCommand = 'N/A'

fileNameLimits = 'source/colorLimits.txt'
fileNameCalibration = 'source/calibration.txt'

try:
   colorFile = open(fileNameLimits,'r')
   colorStr = colorFile.read().split(",")
   colorInt = [int(x) for x in colorStr]
   MINTEMP = colorInt[0]
   MAXTEMP = colorInt[1]
except:
```

```
   print("Color por defecto")

xParameter = 0.2959
bParameter = -2.4836
try:
   calibrationFile = open(fileNameCalibration,'r')
   calibrationStr = calibrationFile.read().split(",")
   calibrationInt = [float(x) for x in calibrationStr]
   xParameter = calibrationInt[0]
   bParameter = calibrationInt[1]
except:
   print("Calibración por defecto")


##########################################################################Configura
ción de colores
#how many color values we can have
COLORDEPTH = 1024
nPixels = 64

os.putenv('SDL_FBDEV', '/dev/fb1')
pygame.init()
global root

#the list of colors we can choose from
blue = Color("indigo")
colorsList = list(blue.range_to(Color("red"), COLORDEPTH))
FONDO = (32, 30, 32)
BLANCO = (255, 255, 255)
COLOR_TEXTO = (50, 60, 80)
GRIS =(211,211,211)
AZUL = (42, 39, 96)
NEGRO = (0,0,0)
YELLOW = (255, 255, 0)

#create the array of colors
colors = [(int(c.red * 255), int(c.green * 255), int(c.blue * 255)) for c in colorsList]
# colorsTrans = [(int(c.red * 255), int(c.green * 255), int(c.blue * 255), 50) for c in colorsList]

points = [(math.floor(ix / 8), (ix % 8)) for ix in range(0, 64)]

grid_x, grid_y = np.mgrid[0:7:32j*float(scalable-0.031), 0:7:32j*float(scalable-0.031)]

##########################################################################Configura
ción de la Pantalla
#Screen is 240x320

scalable = 2
```

```
scrn_height = 240*scalable
scrn_width = 320*scalable

#sensor is an 8x8 grid so lets do a square
map_height = min(scrn_height,scrn_width)
map_width = min(scrn_height,scrn_width)

displayPixelWidthCubic = float(map_width) / 30
displayPixelHeightCubic = float(map_height) / 30

displayPixelWidth = map_width / 8
displayPixelHeight = map_height / 8

clock = pygame.time.Clock()
click = False
################################################################################Botones
principales
def posBotonRec(pos, dim):
   rec = (pos[0]-dim[0]/2,pos[1]-dim[1]/2 , dim[0], dim[1])
   return rec

def posBotonEsquina(pos, dim):
   rec = [pos[0]-dim[0]/2, pos[1]-dim[1]/2]
   return rec

tam_boton = (scrn_width - map_width)/2
botones=[]
botonesMainScreen(pygame, botones)

botonesConfig = []
botonesSettingScreen(pygame, botonesConfig)

botonesMaximo = []
botonesLimScreen(pygame, botonesMaximo)

botonesCalibracion = []
botonesCalScreen(pygame, botonesCalibracion)

################################################################################Interfaz

#Se crea la pantalla que contendra la interfaz
flags = pygame.DOUBLEBUF | pygame.SRCALPHA
lcd = pygame.display.set_mode((scrn_width, scrn_height), flags)
pygame.display.set_caption('Cámara Térmica')
textFont = pygame.font.SysFont("comicsansms", int(displayPixelWidth/2))
textFont_botones = pygame.font.SysFont("comicsansms", int(tam_boton/3))
textFont_temp = pygame.font.SysFont("comicsansms", int(displayPixelWidth/2.5))
```

```
textFont_rango_confi = pygame.font.SysFont("comicsansms", 60)
textFont_rango_cal = pygame.font.SysFont("comicsansms",40 )
textFont_titulo = pygame.font.SysFont("comicsansms", 30)
lcd.fill(BLANCO)
pygame.display.update()

directorios = listdir("/media/pi")
if len(directorios)>0:
   raiz = "/media/pi/" + directorios[0] +"/"
else:
   raiz = "/home/pi/Desktop/Datos_CT/"

raiz = "/home/pi/Desktop/Datos_CT/"

input_box1 = InputBox(220, 80, 400, 50)

#Funciones
#Guardar variables tlimites
def guardarVariables(var1, var2, archivo):
   file = open(archivo,"w")
   text = str(var1) + "," + str(var2)
   file.write(text)
   file.close()

def regLineal(datX, datY):
   regresion = linregress(datX, datY)
   pendiente = regresion.slope
   corte = regresion.intercept
   return pendiente,corte

def actualizarTemCal(var):
   global text_recV
   var = round(var,2)
   posMenos = botonesCalibracion[2]['pos']
   posMas = botonesCalibracion[3]['pos']
   colorText = color_active if box_active else color_inactive
   textLabelVar = textFont_rango_cal.render(str(var), True, colorText)
   text_recV = textLabelVar.get_rect()
   text_recV.center = ((posMenos[0] + posMas[0]) / 2, (posMenos[1] + posMas[1]) / 2)
   pygame.draw.rect(lcd, BLANCO, input_box)
   lcd.blit(textLabelVar, text_recV)

def dibujar_botones_calibracion():
   lcd.fill(BLANCO)
   for boton in botonesCalibracion:
      if boton['habilitado']:
         lcd.blit(boton['imagen'], boton['rect'])
      else:
```

```
            lcd.blit(boton['imagen_dos'], boton['rect'])
    texto = "Calibracion manual"
    textFont_titulo = pygame.font.SysFont("comicsansms", 30)
    textLabel = textFont_titulo.render(texto, True, NEGRO)
    text_rec =  textLabel.get_rect()
    text_rec.center = (180, 25)
    lcd.blit(textLabel, text_rec)

def dibujarBarraColor(posHorizontal):
    widthBar = 10
    heightBar = scrn_height
    for i in range(heightBar):
        pygame.draw.rect(lcd, colors[int(i*(4/scalable))],(posHorizontal+1, heightBar-i, widthBar,
2))
    rango = MAXTEMP - MINTEMP
    nTxtBar = 5
    tam_marcas = rango/float(nTxtBar-1)
    tam_marcas_pixel = map_height/(nTxtBar-1)
    for i in range(nTxtBar):
        txtBar = "- "+str(int(MINTEMP+int(tam_marcas*i)))
        textLabel = textFont_temp.render(txtBar, True, NEGRO)
        lcd.blit(textLabel, (posHorizontal + 12, map_height - tam_marcas_pixel*i -
int(displayPixelWidth/3)))
    txt = "- "+str(MAXTEMP)
    textLabel = textFont_temp.render(txt, True, NEGRO)
    lcd.blit(textLabel, (posHorizontal + 12, 1))

def dibujar_botones_configuracion():
    global ip, newCommand, lcd, input_box1, configurar
    for boton in range(0, 13):
        rec = posBotonRec(botonesConfig[boton]['pos'], (85,85))
        pygame.draw.rect(lcd, BLANCO, rec)
        lcd.blit(botonesConfig[boton]['imagen'], botonesConfig[boton]['rect'])
        textLabel = textFont_temp.render(botonesConfig[boton]['texto'][0], True, NEGRO)
        lcd.blit(textLabel, (rec[0]-20, rec[1]+rec[3] - 105))
    if servidor:
        rec = posBotonRec(botonesConfig[13]['pos'], (85,85))
        pygame.draw.rect(lcd, BLANCO, rec)
        lcd.blit(botonesConfig[13]['imagen'], botonesConfig[13]['rect'])
        textLabel = textFont_temp.render(botonesConfig[13]['texto'][0], True, NEGRO)
        lcd.blit(textLabel, (rec[0]-20, rec[1]+rec[3] - 105))
    else:
        rec = posBotonRec(botonesConfig[13]['pos'], (85,85))
        pygame.draw.rect(lcd, BLANCO, rec)
        lcd.blit(botonesConfig[13]['imagen_dos'], botonesConfig[13]['rect'])
        textLabel = textFont_temp.render(botonesConfig[13]['texto'][0], True, NEGRO)
        lcd.blit(textLabel, (rec[0]-20, rec[1]+rec[3] - 105))
    textLabel = textFont_temp.render("Carpeta actual: "+ raiz, True, NEGRO)
```

```
    lcd.blit(textLabel, (10, 450))
    textLabel = textFont_temp.render("Extension de archivo de datos:", True, NEGRO)
    lcd.blit(textLabel, (10, 20))
    textLabel = textFont_temp.render(extActual, True, NEGRO)
    lcd.blit(textLabel, (10, 38))
    textLabel = textFont_temp.render("Extension de archivo de video:", True, NEGRO)
    lcd.blit(textLabel, (10, 150))
    textLabel = textFont_temp.render(extActual2 + '/' + videoCon, True, NEGRO)
    lcd.blit(textLabel, (10, 168))
    textLabel = textFont_temp.render('Direccion IP local:', True, NEGRO)
    lcd.blit(textLabel, (470, 400))
    textLabel = textFont_temp.render(ip, True, NEGRO)
    lcd.blit(textLabel, (470, 418))
    textLabel = textFont_temp.render('Configuracion manual:', True, NEGRO)
    lcd.blit(textLabel, (10, 300))
    textLabel = textFont_temp.render('Limites      Regresion', True, NEGRO)
    lcd.blit(textLabel, (10, 320))
    #textLabel = textFont_temp.render(newCommand, True, NEGRO)
    #lcd.blit(textLabel, (470, 436))
    pygame.display.update()

def dibujar_botones_rango(texto, var):
    global botonesMaximo, MAXTEMP, MINTEMP
    lcd.fill(GRIS)
    if texto == 'save':
        lcd.blit(botonesMaximo[2]['imagen'], botonesMaximo[2]['rect'])
        lcd.blit(botonesMaximo[5]['imagen'], botonesMaximo[5]['rect'])
    else:
        for boton in range(0, 5):
            lcd.blit(botonesMaximo[boton]['imagen'], botonesMaximo[boton]['rect'])
        posMenos = botonesMaximo[0]['pos']
        posMas = botonesMaximo[1]['pos']
        textLabelVar = textFont_rango_confi.render(str(MAXTEMP), True, NEGRO)
        text_recV = textLabelVar.get_rect()
        text_recV.center = ((posMenos[0] + posMas[0]) / 2, (posMenos[1] + posMas[1]) / 2)
        lcd.blit(textLabelVar, text_recV)
        textLabel = textFont_titulo.render('Tope temperatura maxima', True, NEGRO)
        text_rec =  textLabel.get_rect()
        text_rec.center = (210, 85)
        lcd.blit(textLabel, text_rec)
        posMenos = botonesMaximo[3]['pos']
        posMas = botonesMaximo[4]['pos']
        textLabelVar = textFont_rango_confi.render(str(MINTEMP), True, NEGRO)
        text_recV = textLabelVar.get_rect()
        text_recV.center = ((posMenos[0] + posMas[0]) / 2, (posMenos[1] + posMas[1]) / 2)
        lcd.blit(textLabelVar, text_recV)
        textLabel = textFont_titulo.render('Tope temperatura minima', True, NEGRO)
        text_rec =  textLabel.get_rect()
```

```
        text_rec.center = (210, 285)
        lcd.blit(textLabel, text_rec)
        dibujarBarraColor(500)

def constrain(val, min_val, max_val):
    return min(max_val, max(min_val, val))

def map(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def dibujar_botones_iniciales(lista_botones):
    lcd.fill(BLANCO)
    for boton in lista_botones:
        if boton['selected']:
            lcd.blit(boton['imagen_dos'], boton['rect'])
            textBoton = boton['texto2']
        else:
            lcd.blit(boton['imagen'], boton['rect'])
            textBoton = boton['texto1']
        for i in range(len(textBoton)):
            texto=textBoton[i]
            textLabel = textFont_botones.render(texto, True, NEGRO)
            lcd.blit(textLabel, (scrn_width - tam_boton - 30,boton['rect'][1]-18 +
(i*displayPixelWidth/3))  )
    dibujarBarraColor(map_width)

def cerrarTodo():
    global confiCal, configurar, grabarVideo, salirGrabacionCSV, salir
    confiCal = configurar = grabarVideo = False
    salirGrabacionCSV = salir = True

def connection():
    global MINTEMP, MAXTEMP, fileNameLimits, raiz, configurar, salirMainScrn, botones,
botonesConfig, salirGrabacionCSV, grabandoCSV, ip, servidorThread, newCommand, s, c,
```

```
servidor, extActual, extActual2, videoCon, grabarVideo, grabandoVideo, archivoDatosNuevo,
archivoVideoNuevo
    #host = socket.gethostname()
    #ip = socket.gethostbyname(host)
    clientDisconnect = False
    if servidor:
        #ip = get('https://api.ipify.org').text
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(('8.8.8.8', 80))
        ip = s.getsockname()[0]
        s = socket.socket()
        try:
            port = 8081
            s.bind(('', port))
            s.listen(1)
            c, address = s.accept()
            c.send(str.encode('Conectado a ' + ip))
            while not salir:
                servidorThread = True
                nuevaRaiz = True
                nuevoArchivo = True
                understood = False
                global newCommand
                #ready = select.select([s], [], [], 1)
                #if ready[0]:
                newCommand = (c.recv(1024))
                if newCommand == 'PNG':
                    extActual = 'PNG'
                    understood = True
                elif newCommand == 'CSV':
                    extActual = 'CSV'
                    understood = True
                elif newCommand == 'TXT':
                    extActual = 'TXT'
                    understood = True
                elif newCommand == 'MP4':
                    extActual2 = 'MP4'
                    understood = True
                elif newCommand == 'MKV':
                    extActual2 = 'MKV'
                    understood = True
                elif newCommand == 'MAP':
                    videoCon = 'MAP'
                    understood = True
                elif newCommand == 'FULLSCREEN':
                    videoCon = 'FULLSCREEN'
                    understood = True
                elif newCommand == 'I_VIDEO':
```

```
            configurar = salirMainScrn = botones[2]['selected'] = botonesConfig[2]['selected'] =
False
            sleep(1)
            grabarVideo = True
            understood = True
          elif newCommand == 'D_VIDEO':
            grabarVideo = False
            grabandoVideo = True
            understood = True
          elif newCommand == 'I_DATOS':
            configurar = salirMainScrn = botones[2]['selected'] = botonesConfig[2]['selected'] =
False
            sleep(1)
            salirGrabacionCSV = grabandoCSV = False
            understood = True
          elif newCommand == 'D_DATOS':
            salirGrabacionCSV = grabandoCSV = True
            understood = True
          elif newCommand == 'TEMP':
            botones[3]['selected'] = False
            understood = True
          elif newCommand == 'VALUES':
            botones[3]['selected'] = True
            understood = True
          elif newCommand == 'ROOT':
            while nuevaRaiz:
              c.send(str.encode('\n\rRaiz es: ' + raiz +'\n\rDesea cambiarla? (SI/NO)'))
              newCommand = (c.recv(1024))
              if newCommand == 'SI':
                raizInfo = ('Ingrese nueva raiz: \n\r\n\rCarpetas: ')
                for name in os.listdir(raiz):
                  if os.path.isdir(os.path.join(raiz, name)):
                    raizInfo = raizInfo + ('\n\r'+ name)
                c.send(str.encode(raizInfo + '\n\r".." para regresar\n\r'))
                newCommand = c.recv(1024)
                if newCommand == '..':
                  raizInfo = raiz.split('/', -2)
                  raizInfo2 = ''
                  for b in range(0, len(raizInfo)-2):
                    raizInfo2 = raizInfo2 + raizInfo[b] + '/'
                  raiz = raizInfo2
                else:
                  raiz = (raiz + newCommand + '/')
              elif newCommand == 'NO':
                nuevaRaiz = False
            understood = True
          elif newCommand == 'EXPORT':
            while nuevoArchivo:
```

```
                archivoInfo = '\n\rRaiz es: ' + raiz +'\n\rArchivos exportables en carpeta:'
                for name in os.listdir(raiz):
                  if name.endswith('.txt') or name.endswith('.csv') or name.endswith('.png') or
name.endswith('.mp4') or name.endswith('.mkv'):
                      archivoInfo = archivoInfo + ('\n\r'+ name)
                archivoInfo = archivoInfo + '\n\n\rPara modificar carpeta usar ROOT\n\rDesea
exportar algun archivo? (SI/NO)'
                c.send(str.encode(archivoInfo))
                newCommand = (c.recv(1024))
                if newCommand == 'SI':
                  archivoInfo = ('\n\rPara archivos TXT, CSV, PNG use D_EXPORT\n\rPara archivos
MP4, MKV use V_EXPORT\n\n\rIngrese archivo a exportar:')
                  c.send(str.encode(archivoInfo))
                  newCommand = c.recv(1024)
                  archivoActual = (raiz + newCommand)
                  if archivoActual.endswith('.txt') or archivoActual.endswith('.csv') or
archivoActual.endswith('.png'):
                    extActual = newCommand[-3:].upper()
                    archivoDatosNuevo = archivoActual
                  elif archivoActual.endswith('.mp4') or archivoActual.endswith('.mkv'):
                    extActual2 = newCommand[-3:].upper()
                    archivoVideoNuevo = archivoActual
                  nuevoArchivo = False
                elif newCommand == 'NO':
                  nuevoArchivo = False
              understood = True
            elif newCommand == 'V_EXPORT':
              archivo = open(archivoVideoNuevo, 'rb')
              sleep(10)
              c.send(str.encode(extActual2))
              sleep(10)
              buff = archivo.read(1024)
              while(buff):
                c.send(buff)
                buff = archivo.read(1024)
              c.send(str.encode('Done'))
              sleep(10)
              understood = True
            elif newCommand == 'D_EXPORT':
              archivo = open(archivoDatosNuevo, 'rb')
              sleep(10)
              c.send(str.encode(extActual))
              sleep(10)
              buff = archivo.read(1024)
              while(buff):
                c.send(buff)
                buff = archivo.read(1024)
              c.send(str.encode('Done'))
```

```
            sleep(10)
            understood = True
        elif newCommand.startswith('MINTEMP'):
            MINTEMP = int(newCommand[7:])
            guardarVariables(MINTEMP, MAXTEMP, fileNameLimits)
            understood = True
        elif newCommand.startswith('MAXTEMP'):
            MAXTEMP = int(newCommand[7:])
            guardarVariables(MINTEMP, MAXTEMP, fileNameLimits)
            understood = True
        elif newCommand.startswith('SERIAL'):
            serialTemp = str(newCommand[6:])
            serialTemp = bytes(serialTemp+'\n')
            input_box1.ser.write(serialTemp)
            print(input_box1.ser.readline().decode('utf-8').rstrip())
            understood = True
        elif newCommand == 'C_EXIT':
            sleep(5)
            clientDisconnect = True
            understood = True
        elif newCommand == 'S_EXIT':
            servidor = False
            ip = 'N/A'
            understood = True
        if not (newCommand == 'D_EXPORT') or not (newCommand == 'V_EXPORT'):
            if understood:
                c.send(str.encode('Recibido'))
            else:
                c.send(str.encode('Revisar comando'))
        if clientDisconnect:
            c.close()
            clientDisconnect = False
            c, address = s.accept()
            c.send(str.encode('Conectado a ' + ip))
        if not servidor:
            s.close()
            break
    c.close()
    s.close()
  except Exception as e:
    print(e)
    s.close()

def interfaz():
  global newCommand, ip, salir, map_height, map_width, displayPixelWidth,
displayPixelHeight, lcd, botones, salirGrabacionCSV, grabarVideo, configurar, confiCal,
box_active, promedioTem, confiMax
  text = "0"
```

```
    clock = pygame.time.Clock()
    while not salir:
        if confiMax:
            for event in pygame.event.get():
                input_box1.handle_event(event)
            input_box1.update()
            input_box1.draw(lcd)
            pygame.display.update()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                cerrarTodo()
                quit()
            if event.type == MOUSEBUTTONUP:
                mouse = event.pos
                if input_box.collidepoint(event.pos) and not tomaHabilitada:
                    # Toggle the active variable.
                    box_active = True
                else:
                    box_active = False
                # Change the current color of the input box.
                presionarPantalla(mouse)
            text = "0" if tomaHabilitada else text
            if event.type == pygame.KEYDOWN:
                # Permite terminar el programa
                if event.key == pygame.K_q:
                    cerrarTodo()
                # Alterna entre 'pantalla completa' y 'ventana'.
                elif event.key == pygame.K_f:
                    pygame.display.toggle_fullscreen()
                elif not tomaHabilitada and box_active:
                    if event.key == pygame.K_BACKSPACE:
                        text = text[:-1]
                    else:
                        if not re.match(r'^-?\d+(?:\.\d+)?$', event.unicode) is None or event.unicode ==
".":
                            text += event.unicode
                    try:
                        text = "0" if text == "" else text
                        print(text)
                        promedioTem = float(text)
                    except Exception as e:
                        print(e)
        pygame.time.wait(5)

def botonesPantallaConfiguracion():
    global terminar, s, ip, servidorThread, servidor, videoCon, extActual, extActual2, root, raiz,
salirGrabacionCSV, grabarVideo, botones, configurar, salirMainScrn, confiMax, MAXTEMP,
MINTEMP, confiMin, salir, confiCal, c
```

```
if botonesConfig[0]['selected']:
    confiMax = True
elif botonesConfig[1]['selected']:
    confiCal = True
elif botonesConfig[2]['selected']:
    configurar = salirMainScrn = botones[2]['selected'] = botonesConfig[2]['selected'] = False
elif botonesConfig[3]['selected']:
    confiMin = True
elif botonesConfig[4]['selected']:
    grabarVideo = False
    salirGrabacionCSV = True
    pygame.time.wait(20)
    try:
        root = Tkinter.Tk()
        root.title('Si cierran esto, el programa se muere LOLZ')
        root.geometry('500x1')
        root.wm_state('iconic')
        root.protocol('WM_DELETE_WINDOW', preventClosing)
        root.directory = tkFileDialog.askdirectory()
        raiz = root.directory + "/"
        root.destroy()
    except Exception as e:
        print(e)
        root.destroy()
elif botonesConfig[5]['selected']:
    extActual = 'TXT'
elif botonesConfig[6]['selected']:
    extActual = 'CSV'
elif botonesConfig[7]['selected']:
    extActual2 = 'MKV'
elif botonesConfig[8]['selected']:
    extActual2 = 'MP4'
elif botonesConfig[9]['selected']:
    videoCon = 'MAP'
elif botonesConfig[10]['selected']:
    videoCon = 'FULLSCREEN'
elif botonesConfig[11]['selected']:
    extActual = 'PNG'
elif botonesConfig[13]['selected']:
    if not servidor:
        servidor = True
        print('servidor inicializado')
        threading.Thread(target=connection).start()
    else:
        if servidorThread:
            servidor = False
            print('servidor apagado')
            ip = 'N/A'
```

```
            try:
                c.send(str.encode('Servidor desconectado'))
                c.close()
                s.close()
            except Exception as e:
                print(e)
                c.close()
                s.close()
    for boton in range(0,12):
        botonesConfig[boton]['selected'] = False
    botonesConfig[12]['selected'] = servidor

def preventClosing():
    pass

def botonesConfiMax():
    global salirGrabacionCSV, grabarVideo, botones, configurar, salirMainScrn, confiMax,
MAXTEMP, MINTEMP, confiMin, salir, confiCal
    if botonesMaximo[0]['selected']:
        MAXTEMP = constrain(MAXTEMP + 1, MINTEMP + 1, 100)
    elif botonesMaximo[1]['selected']:
        MAXTEMP = constrain(MAXTEMP - 1, MINTEMP + 1, 100)
    elif botonesMaximo[2]['selected']:
        guardarVariables(MINTEMP, MAXTEMP, fileNameLimits)
        confiMax = False
    for boton in botonesMaximo:
        boton['selected'] = False

def botonesConfiMin():
    global salirGrabacionCSV, grabarVideo, botones, configurar, salirMainScrn, confiMax,
MAXTEMP, MINTEMP, confiMin, salir, confiCal
    if botonesMaximo[0]['selected']:
        MAXTEMP = constrain(MAXTEMP + 1, MINTEMP + 1, 100)
    elif botonesMaximo[1]['selected']:
        MAXTEMP = constrain(MAXTEMP - 1, MINTEMP + 1, 100)
    elif botonesMaximo[2]['selected']:
        guardarVariables(MINTEMP, MAXTEMP, fileNameLimits)
        confiMin = False
    elif botonesMaximo[3]['selected']:
        MINTEMP = constrain(MINTEMP + 1, 0, MAXTEMP - 1)
    elif botonesMaximo[4]['selected']:
        MINTEMP = constrain(MINTEMP - 1, 0, MAXTEMP - 1)
    # dibujar_botones_rango("Tope temperatura maxima", MAXTEMP)
    for boton in botonesMaximo:
        boton['selected'] = False

def botonesConfiCal():
```

```
  global salirGrabacionCSV, grabarVideo, botones, configurar, salirMainScrn, confiMax,
MAXTEMP, MINTEMP, confiMin, salir, confiCal, promedioTem, tomaHabilitada, xParameter,
bParameter, calY, calX
  if botonesCalibracion[0]['selected']:
    xParameter, bParameter = regLineal(calX, calY)
    guardarVariables(xParameter, bParameter, fileNameCalibration)
  elif botonesCalibracion[1]['selected']:
    calX = []
    calY = []
    confiCal = False
  elif botonesCalibracion[2]['selected']:
    promedioTem -= 0.1
  elif botonesCalibracion[3]['selected']:
    promedioTem += 0.1
  elif botonesCalibracion[4]['selected']:
    if not tomaHabilitada:
      calX.append(int(promedioDatos))
      calY.append(promedioTem)
    tomaHabilitada ^= 1
    for bot in range(2,6):
      botonesCalibracion[bot]['habilitado'] ^= 1
    if len(calX) >= 2:
      botonesCalibracion[0]['habilitado'] = True
  elif botonesCalibracion[5]['selected']:
    tomaHabilitada ^= 1
    for bot in range(2,6):
      botonesCalibracion[bot]['habilitado'] ^= 1
  for boton in botonesCalibracion:
    boton['selected'] = False

def presionarPantalla(mouse):
  global salirGrabacionCSV, grabarVideo, botones, configurar, salirMainScrn, confiMax,
MAXTEMP, MINTEMP, confiMin, salir, confiCal, click
  scrnConfi = confiMax or confiMin or confiCal
  if corriendoMainScrn:
    for boton in botones:
      boton['selected'] = boton['selected'] ^ boton['rect'].colliderect([mouse[0]-1, mouse[1], 1,
1])
    salirGrabacionCSV = not botones[1]['selected']
    grabarVideo = botones[0]['selected']
    configurar = botones[2]['selected']
    if not configurar:
      dibujar_botones_iniciales(botones)
  elif configurando and not scrnConfi:
    for boton in botonesConfig:
      boton['selected'] = boton['rect'].colliderect([mouse[0] - 1, mouse[1], 1, 1])
    botonesPantallaConfiguracion()
  elif confiMax:
```

```
      for boton in botonesMaximo:
        boton['selected'] = boton['rect'].colliderect([mouse[0] - 1, mouse[1], 1, 1])
      botonesConfiMax()
    elif confiMin:
      for boton in botonesMaximo:
        boton['selected'] = boton['rect'].colliderect([mouse[0] - 1, mouse[1], 1, 1])
      botonesConfiMin()
    elif confiCal:
      for boton in botonesCalibracion:
        boton['selected'] = boton['rect'].colliderect([mouse[0] - 1, mouse[1], 1, 1]) and
boton['habilitado']
      botonesCalibracion[4]['selected'] = botonesCalibracion[4]['rect'].colliderect([mouse[0] - 1,
mouse[1], 1, 1])
      botonesConfiCal()
    click = True

def mainSrcn():
    global corriendoMainScrn, csv_pixels
    corriendoMainScrn = True
    dibujar_botones_iniciales(botones)
    while not salirMainScrn:
      #read the pixels
      try:
        pixelsMap = sensor.readPixels()
        # pixelsMap = np.random.rand(64,1)*100
        pixels = [map(p, MINTEMP, MAXTEMP, 0, COLORDEPTH - 1) for p in pixelsMap]
        if botones[3]['selected']:
          for ix in range(8):
            for jx in range(8):
              pygame.draw.rect(lcd, colors[constrain(int(pixels[ix * 8 + jx]), 0, COLORDEPTH - 1)],
(displayPixelHeight * ix, displayPixelWidth * jx, displayPixelHeight, displayPixelWidth))
              textTemp = textFont.render(str(round((pixelsMap[ix * 8 + jx] / 0.25) * xParameter
+bParameter, 2)), True, (255, 255, 255))
              textTempRec = textTemp.get_rect()
              textTempRec.center = ( int( displayPixelHeight * (ix+0.5) ) , int( displayPixelWidth *
(jx + 0.5) ))
              lcd.blit(textTemp, textTempRec)
        else:
          # perdorm interpolation
          bicubic = griddata(points, pixels, (grid_x, grid_y), method='cubic')

          # draw everything
          for ix, row in enumerate(bicubic):
            for jx, pixel in enumerate(row):
              pygame.draw.rect(lcd, colors[constrain(int(pixel), 0, COLORDEPTH - 1)], (7.5 * ix,
7.5 * jx, displayPixelHeightCubic, displayPixelWidthCubic))
        if grabandoCSV:
          tiempo_actual_csv = time() -tiempo_inicio_csv
```

```
            pixelsMap.insert(0,tiempo_actual_csv)
            csv_pixels.append(pixelsMap)

          if not salirMainScrn:
            pygame.display.update()
        except Exception as e:
          print(e)
    corriendoMainScrn = False

x_file_csv = 0
def grabar_csv():
    global salirGrabacionCSV, botones, extActual, grabandoCSV, x_file_csv,tiempo_inicio_csv,
csv_pixels, archivoDatosNuevo
    grabandoCSV = True
    dir = listdir(raiz)
    if not "dataFiles" in dir:
      try:
        os.system("mkdir " +raiz+ "dataFiles")
      except Exception as e:
        print(e)
    filename = raiz +"dataFiles/camara_termica." + extActual.lower()
    if not raiz + "dataFiles/camara_termica" + str(x_file_csv) + "." + extActual.lower() in
glob.glob(raiz +"dataFiles/*." + extActual.lower()):
      filename =raiz + "dataFiles/camara_termica" + str(x_file_csv) + "." + extActual.lower()
    else:
      x_file_csv += 1
      grabar_csv()
    tiempo_inicio_csv = time()
    archivoDatosNuevo = filename
    if extActual == 'PNG':
      rect = pygame.Rect(0, 0, 480, 480)
      sub = lcd.subsurface(rect)
      pygame.image.save(sub, filename)
      salirGrabacionCSV = True
    else:
      #salirGrabacionCSV = False
      archivo = open(filename, "w")
      csv_escritor = csv.writer(archivo)
        #Encabezado del archivo
      text_header = ["Tiempo"]
      for i in range(nPixels):
        text_header.append("Pixel "+ str(i))
      csv_escritor.writerow(text_header)
      archivo.flush()
      archivo.close()
      while not salirGrabacionCSV:
        archivo = open(filename, "a")
        csv_escritor = csv.writer(archivo)
```

```
            for i in range(len(csv_pixels)):
                csv_escritor.writerow(csv_pixels[i])
            csv_pixels = []
            archivo.flush()
            archivo.close()
            pygame.time.wait(1000)
        csv_pixels = []
        grabandoCSV = False

x_file_video = 0
def iniciarGrabarVideo():
    print('video Iniciado')
    global videoCon, lcd, p, grabandoVideo, x_file_video, archivoVideoNuevo
    grabandoVideo = True
    filename = raiz + "videoFiles/camara_termica" + str(x_file_video) + "." + extActual2.lower()
    dir = listdir(raiz)
    if not "videoFiles" in dir:
        try:
            os.system("mkdir " +raiz+ "videoFiles")
        except Exception as e:
            print(e)
    if not raiz +"videoFiles/camara_termica" + str(x_file_video) + "." + str(extActual2.lower()) in
glob.glob(raiz +"videoFiles/*." + extActual2.lower()):
        filename = raiz +"videoFiles/camara_termica" + str(x_file_video) + "." + extActual2.lower()
    else:
        x_file_video += 1
        iniciarGrabarVideo()
    archivoVideoNuevo = filename
    if videoCon == 'MAP':
        comando = 'ffmpeg -video_size 480x480 -framerate 30 -f x11grab -i :0.0+0,30 -crf 0 -preset:v
ultrafast -af aresample=async=1:first_pts=0 ' + filename
    elif videoCon == 'FULLSCREEN':
        comando = "ffmpeg -framerate 30 -f alsa -r 10 -f x11grab -s $(xdpyinfo | grep dimensions |
awk '{print $2;}') -i ${DISPLAY} -c:v libx264rgb -crf 0 -preset:v ultrafast -af
aresample=async=1:first_pts=0 "+ filename
    p = subprocess.Popen(comando, shell=True, stdin=None, stderr=subprocess.STDOUT,
stdout=subprocess.PIPE, close_fds=True)

def detenerVideo():
    print('video detenido')
    global p, grabandoVideo
    os.killpg(os.getpgid(p.pid), signal.SIGINT)
    grabandoVideo = False

def configuracionPantalla():
    global salirMainScrn, configurando, click, lcd, input_box1, clock
    configurando = True
    salirMainScrn = True
```

```
  while configurar:
    #if click:
      lcd.fill(BLANCO)
      if confiMax:
        dibujar_botones_rango('save', MAXTEMP)
        textLabel = textFont_temp.render("Comunicacion serial con Arduino:", True, NEGRO)
        lcd.blit(textLabel, (190, 20))
        textLabel = textFont_temp.render(input_box1.line, True, NEGRO)
        lcd.blit(textLabel, (30, 170))
        input_box1.update()
        input_box1.draw(lcd)
        pygame.display.update()
      elif confiMin:
        dibujar_botones_rango('todos', MINTEMP)
      elif confiCal:
        dibujar_botones_calibracion()
        dibujarBarraColor(map_width - 10)
        pygame.time.wait(20)
      else:
        dibujar_botones_configuracion()
      if not confiCal:
        pygame.display.update()
      pygame.time.wait(5)
      click = False
  configurando = False

#############################################################################MODIFICA
R calibracion
#############################################################################AUTOMA
TICA

def confiCalibracion():
  global configurandoCal, configurar, promedioTem, promedioDatos
  configurandoCal = True
  pygame.display.update()
  pixelsMap = []
  while confiCal:
    try:
      if tomaHabilitada:
        pixelsMap = sensor.readPixels()
        # pixelsMap = np.random.rand(64, 1) * 100
      pixels = [map(p, MINTEMP, MAXTEMP, 0, COLORDEPTH - 1) for p in pixelsMap]
      promTem = 0
      promDatos = 0

      for ix in range(3,5):
        for jx in range(3,5):
```

```
                pygame.draw.rect(lcd, colors[constrain(int(pixels[ix * 8 + jx]), 0, COLORDEPTH - 1)],
(displayPixelHeight * ix, displayPixelWidth * jx, displayPixelHeight, displayPixelWidth))
                promTem +=round((pixelsMap[ix * 8 + jx] / 0.25) * xParameter +bParameter, 2)
                promDatos += pixelsMap[ix * 8 + jx] / 0.25
          if tomaHabilitada:
            promedioTem = promTem / 4
            promedioDatos = promDatos / 4
          actualizarTemCal(promedioTem)
          if confiCal:
            pygame.display.update()
        except Exception as e:
          print(e)
    configurandoCal = False

def principal():
    global a, salirMainScrn, salirGrabacionCSV, s
    while not salir:
        if not salirMainScrn and not corriendoMainScrn:
            threading.Thread(target=mainSrcn).start()
        if not salirGrabacionCSV and not grabandoCSV:
            a = threading.Thread(target=grabar_csv).start()
        if grabarVideo and not grabandoVideo:
            iniciarGrabarVideo()
        elif not grabarVideo and grabandoVideo:
            detenerVideo()
        if configurar and not configurando:
            threading.Thread(target=configuracionPantalla).start()
        if confiCal and not configurandoCal:
            threading.Thread(target=confiCalibracion).start()
        pygame.time.wait(50)
    salirMainScrn = True
    salirGrabacionCSV = True

threading.Thread(target=interfaz).start()
threading.Thread(target=principal).start()
```

Annex 35a: Raspberry Pi code (server)

```
#Screen is 240x320
scalable = 1.95

scrn_height = 240*scalable
scrn_width = 320*scalable

#sensor is an 8x8 grid so lets do a square
map_height = min(scrn_height,scrn_width)
map_width = min(scrn_height,scrn_width)
```

```
def posBotonEsquina(pos, dim):
        rec = [pos[0]-dim[0]/2, pos[1]-dim[1]/2]
        return rec

def botonesMainScreen(pygame, botones):
  #Cargamos las imagenes que serviran como botones
  play_boton_imagen = pygame.image.load("imagenes/play.png")
  stop_boton_imagen = pygame.image.load("imagenes/stop.png")
  config_boton_imagen = pygame.image.load("imagenes/config.png")
  csv_start_imagen = pygame.image.load("imagenes/record.png")
  csv_stop_imagen = pygame.image.load("imagenes/StopRecord.png")
  swicth_on_imagen = pygame.image.load("imagenes/swicthOn.png")
  swicth_off_imagen = pygame.image.load("imagenes/swicthOff.png")
  log_boton_imagen = pygame.image.load("imagenes/LogoBiomicrosystemsCorto.png")

  #Se escalan las imagenes cargadas
  tam_boton = int(scrn_width - map_width)/2
  play_boton_imagen = pygame.transform.scale(play_boton_imagen, [int(tam_boton*0.8),
int(tam_boton*0.8)])
  stop_boton_imagen = pygame.transform.scale(stop_boton_imagen, [int(tam_boton*0.8),
int(tam_boton*0.8)])
  config_boton_imagen = pygame.transform.scale(config_boton_imagen, [int(tam_boton*0.8),
int(tam_boton*0.8)])
  csv_start_imagen = pygame.transform.scale(csv_start_imagen, [int(tam_boton*0.8),
int(tam_boton*0.8)])
  csv_stop_imagen = pygame.transform.scale(csv_stop_imagen, [int(tam_boton*0.8),
int(tam_boton*0.8)])
  swicth_off_imagen = pygame.transform.scale(swicth_off_imagen, [int(tam_boton*0.8),
int(tam_boton*0.8)])
  swicth_on_imagen = pygame.transform.scale(swicth_on_imagen, [int(tam_boton*0.8),
int(tam_boton*0.8)])
  log_boton_imagen = pygame.transform.scale(log_boton_imagen, [int(tam_boton*1),
int(tam_boton*1)])
  csv_start_imagen.set_alpha(20)

  #Se crea el rectangulo de cada boton
  r_boton_log = log_boton_imagen.get_rect()
  r_boton_confg = config_boton_imagen.get_rect()
  r_boton_csv = csv_start_imagen.get_rect()
  r_boton_on = swicth_on_imagen.get_rect()
  r_boton_play = play_boton_imagen.get_rect()

  #Asignacion de posiciones
  #top_left = map_width + int(((scrn_width - map_width) - tam_boton)/2)
  top_left = scrn_width - tam_boton
  top_up = scrn_height/10
  boton_separacion = scrn_height/4
```

```
    r_boton_play.topleft = [top_left, top_up ]
    r_boton_csv.topleft = [top_left, top_up + boton_separacion*0.9]
    r_boton_confg.topleft = [top_left, top_up + boton_separacion*1.8]
    r_boton_on.topleft = [top_left, top_up + boton_separacion*2.7]
    r_boton_log.topleft = [545, 410]

    botones.append({'imagen': play_boton_imagen, 'imagen_dos': stop_boton_imagen, 'rect':
r_boton_play, 'selected': False, 'texto1': ["Video"], 'texto2': ["Video"]})
    botones.append({'imagen': csv_start_imagen, 'imagen_dos': csv_stop_imagen, 'rect':
r_boton_csv, 'selected': False, 'texto1': ["Datos"], 'texto2': ["Datos"]})
    botones.append({'imagen': config_boton_imagen, 'imagen_dos': config_boton_imagen, 'rect':
r_boton_confg, 'selected': False,'texto1': ["Ajustes"], 'texto2': ["Ajustes"]})
    botones.append({'imagen': swicth_off_imagen, 'imagen_dos': swicth_on_imagen, 'rect':
r_boton_on, 'selected': True,'texto1': ["Temp."], 'texto2': ["Valores"]})
    botones.append({'imagen': log_boton_imagen, 'imagen_dos': log_boton_imagen, 'rect':
r_boton_log, 'selected': True,'texto1': [""], 'texto2': [""]})

################################################################################
Botones configuracion   ###########################################################

def botonesSettingScreen(pygame, botonesConfig):
    # Variables interfaz
    widthBoton = 55
    heightBoton = 55

    posMin = (int(50), 375)
    posCal = (int(150), 365)
    posMax = (int(400), 365)
    posSalir = (567, 75)
    posCar = (567, 220)
    posTxt = (67, 90)
    posCsv = (137, 90)
    posDoc = (267, 85)
    posMkv = (67, 235)
    posMp4 = (137, 235)
    posMap = (210, 205)
    posFulls = (210, 270)
    posPng = (207, 90)
    posSoc = (567, 355)
    posSer = (450, 320)
    posLog = (380, 60)

    # Cargamos las imagenes que serviran como botones
    minTem_boton_imagen = pygame.image.load("imagenes/limit.png")
    maxTem_boton_imagen = pygame.image.load("imagenes/serial.png")
    calibra_boton_imagen = pygame.image.load("imagenes/calibrate.png")
    salir_boton_imagen = pygame.image.load("imagenes/return.png")
    apagar_boton_imagen = pygame.image.load("imagenes/Folder.png")
```

```
   txt_boton_imagen = pygame.image.load("imagenes/txt.png")
   csv_boton_imagen = pygame.image.load("imagenes/csv.png")
   #doc_boton_imagen = pygame.image.load("imagenes/doc.png")
   mkv_boton_imagen = pygame.image.load("imagenes/mkv.png")
   mp4_boton_imagen = pygame.image.load("imagenes/mp4.png")
   map_boton_imagen = pygame.image.load('imagenes/map.png')
   fulls_boton_imagen = pygame.image.load('imagenes/fulls.png')
   png_boton_imagen = pygame.image.load('imagenes/png.png')
   soc_boton_imagen = pygame.image.load('imagenes/cloud2.png')
   soc_boton_imagen2 = pygame.image.load('imagenes/cloud2.png')
   log_boton_imagen = pygame.image.load('imagenes/LogoBiomicrosystemsCorto.png')

   # Se escalan las imagenes cargadas
   minTem_boton_imagen = pygame.transform.scale(minTem_boton_imagen, [int(widthBoton),
int(heightBoton)])
   maxTem_boton_imagen = pygame.transform.scale(maxTem_boton_imagen,
[int(widthBoton), int(heightBoton)])
   calibra_boton_imagen = pygame.transform.scale(calibra_boton_imagen,
[int(widthBoton*1.3), int(heightBoton*1.3)])
   salir_boton_imagen = pygame.transform.scale(salir_boton_imagen, [widthBoton,
heightBoton])
   apagar_boton_imagen = pygame.transform.scale(apagar_boton_imagen, [widthBoton,
heightBoton])
   txt_boton_imagen = pygame.transform.scale(txt_boton_imagen, [widthBoton, heightBoton])
   csv_boton_imagen = pygame.transform.scale(csv_boton_imagen, [widthBoton, heightBoton])
   #doc_boton_imagen = pygame.transform.scale(doc_boton_imagen, [widthBoton,
heightBoton])
   mkv_boton_imagen = pygame.transform.scale(mkv_boton_imagen, [widthBoton,
heightBoton])
   mp4_boton_imagen = pygame.transform.scale(mp4_boton_imagen, [widthBoton,
heightBoton])
   map_boton_imagen = pygame.transform.scale(map_boton_imagen, [int(widthBoton/1.3),
int(heightBoton/1.3)])
   fulls_boton_imagen = pygame.transform.scale(fulls_boton_imagen, [int(widthBoton/1.5),
int(heightBoton/1.5)])
   png_boton_imagen = pygame.transform.scale(png_boton_imagen, [int(widthBoton),
int(heightBoton)])
   soc_boton_imagen = pygame.transform.scale(soc_boton_imagen, [int(widthBoton),
int(heightBoton)])
   soc_boton_imagen2 = pygame.transform.scale(soc_boton_imagen, [int(widthBoton),
int(heightBoton)])
   log_boton_imagen = pygame.transform.scale(log_boton_imagen, [int(widthBoton*2),
int(heightBoton*2)])

   # Se crea el rectangulo de cada boton
   r_boton_minTem = minTem_boton_imagen.get_rect()
   r_boton_maxTem = maxTem_boton_imagen.get_rect()
   r_boton_calibra = calibra_boton_imagen.get_rect()
```

```
    r_boton_salir = salir_boton_imagen.get_rect()
    r_boton_apagar = apagar_boton_imagen.get_rect()
    r_boton_txt = txt_boton_imagen.get_rect()
    r_boton_csv = csv_boton_imagen.get_rect()
    #r_boton_doc = doc_boton_imagen.get_rect()
    r_boton_mkv = mkv_boton_imagen.get_rect()
    r_boton_mp4 = mp4_boton_imagen.get_rect()
    r_boton_map = map_boton_imagen.get_rect()
    r_boton_fulls = fulls_boton_imagen.get_rect()
    r_boton_png = png_boton_imagen.get_rect()
    r_boton_soc = soc_boton_imagen.get_rect()
    r_boton_log = log_boton_imagen.get_rect()

    # Asignacion de posiciones
    r_boton_maxTem.topleft = posBotonEsquina(posMax, (int(widthBoton/2),
int(heightBoton/2)))
    r_boton_minTem.topleft = posBotonEsquina(posMin, (int(widthBoton/2),
int(heightBoton/2)))
    r_boton_calibra.topleft = posBotonEsquina(posCal, (int(widthBoton/2), int(heightBoton/2)))
    r_boton_salir.topleft = posBotonEsquina(posSalir, (widthBoton, heightBoton))
    r_boton_apagar.topleft = posBotonEsquina(posCar, (widthBoton, heightBoton))
    r_boton_txt.topleft = posBotonEsquina(posTxt, (widthBoton, heightBoton))
    r_boton_csv.topleft = posBotonEsquina(posCsv, (widthBoton, heightBoton))
    #r_boton_doc.topleft = posBotonEsquina(posDoc, (widthBoton, heightBoton))
    r_boton_mkv.topleft = posBotonEsquina(posMkv, (widthBoton, heightBoton))
    r_boton_mp4.topleft = posBotonEsquina(posMp4, (widthBoton, heightBoton))
    r_boton_map.topleft = posBotonEsquina(posMap, (int(widthBoton/1.3),
int(heightBoton/1.3)))
    r_boton_fulls.topleft = posBotonEsquina(posFulls, (int(widthBoton/1.5),
int(heightBoton/1.5)))
    r_boton_png.topleft = posBotonEsquina(posPng, (widthBoton, heightBoton))
    r_boton_soc.topleft = posBotonEsquina(posSoc, (widthBoton, heightBoton))
    r_boton_log.topleft = posBotonEsquina(posLog, (widthBoton*2, heightBoton*2))

    # Se agregan los botones con sus respectivas propiedades
    botonesConfig.append(
        {'imagen': maxTem_boton_imagen, 'rect': r_boton_maxTem, 'selected': False, 'texto':
["Serial"],
         'pos': posMax})
    botonesConfig.append(
        {'imagen': calibra_boton_imagen, 'rect': r_boton_calibra, 'selected': False, 'texto': [""],
         'pos': posCal})
    botonesConfig.append(
        {'imagen': salir_boton_imagen, 'rect': r_boton_salir, 'selected': False, 'texto': ["Volver"],
'pos': posSalir})
    botonesConfig.append(
        {'imagen': minTem_boton_imagen, 'rect': r_boton_minTem, 'selected': False, 'texto': [""],
         'pos': posMin})
```

```
    botonesConfig.append(
        {'imagen': apagar_boton_imagen, 'rect': r_boton_apagar, 'selected': False, 'texto':
["Carpeta"], 'pos': posCar})
    botonesConfig.append(
        {'imagen': txt_boton_imagen, 'rect': r_boton_txt, 'selected': False, 'texto': [""], 'pos':
posTxt})
    botonesConfig.append(
        {'imagen': csv_boton_imagen, 'rect': r_boton_csv, 'selected': False, 'texto': [""], 'pos':
posCsv})
    #botonesConfig.append(
        #{'imagen': doc_boton_imagen, 'rect': r_boton_doc, 'selected': False, 'texto': [""], 'pos':
posDoc})
    botonesConfig.append(
        {'imagen': mkv_boton_imagen, 'rect': r_boton_mkv, 'selected': False, 'texto': [""], 'pos':
posMkv})
    botonesConfig.append(
        {'imagen': mp4_boton_imagen, 'rect': r_boton_mp4, 'selected': False, 'texto': [""], 'pos':
posMp4})
    botonesConfig.append(
        {'imagen': map_boton_imagen, 'rect': r_boton_map, 'selected': False, 'texto': [""], 'pos':
posMap})
    botonesConfig.append(
        {'imagen': fulls_boton_imagen, 'rect': r_boton_fulls, 'selected': False, 'texto': [""], 'pos':
posFulls})
    botonesConfig.append(
        {'imagen': png_boton_imagen, 'rect': r_boton_png, 'selected': False, 'texto': [""], 'pos':
posPng})
    botonesConfig.append(
        {'imagen': log_boton_imagen, 'rect': r_boton_log, 'selected': False, 'texto': [""], 'pos':
posLog})
    botonesConfig.append(
        {'imagen': soc_boton_imagen, 'imagen_dos': soc_boton_imagen2,'rect': r_boton_soc,
'selected': False, 'texto': ["Servidor"], 'pos': posSoc})

################################################################## Botones ajuste
limites ###########################################################################

def botonesLimScreen(pygame, botonesMaximo):
    widthBoton = 80
    heightBoton = 80

    posMas = (367, 160)
    posMenos = (153, 160)
    posSave = (30, 30)
    posMas2 = (367, 360)
    posMenos2 = (153, 360)
    posLog = (30, 350)
```

```
    # Cargamos las imagenes que serviran como botones
    menos_boton_imagen = pygame.image.load("imagenes/back.png")
    mas_boton_imagen = pygame.image.load("imagenes/fordward.png")
    save_boton_imagen = pygame.image.load("imagenes/save.png")
    log_boton_imagen = pygame.image.load("imagenes/BioMicroSystemsLinea.jpeg")

    # Se escalan las imagenes cargadas
    menos_boton_imagen = pygame.transform.scale(menos_boton_imagen, [widthBoton,
heightBoton])
    mas_boton_imagen = pygame.transform.scale(mas_boton_imagen, [widthBoton,
heightBoton])
    save_boton_imagen = pygame.transform.scale(save_boton_imagen, [40, 40])
    log_boton_imagen = pygame.transform.scale(log_boton_imagen, [670, 174])

    # Se crea el rectangulo de cada boton
    r_boton_mas = mas_boton_imagen.get_rect()
    r_boton_menos = menos_boton_imagen.get_rect()
    r_boton_save = save_boton_imagen.get_rect()
    r_boton_mas2 = mas_boton_imagen.get_rect()
    r_boton_menos2 = menos_boton_imagen.get_rect()
    r_boton_log = log_boton_imagen.get_rect()

    # Asignacion de posiciones
    r_boton_mas.topleft = posBotonEsquina(posMas, (widthBoton, heightBoton))
    r_boton_menos.topleft = posBotonEsquina(posMenos, (widthBoton, heightBoton))
    r_boton_mas2.topleft = posBotonEsquina(posMas2, (widthBoton, heightBoton))
    r_boton_menos2.topleft = posBotonEsquina(posMenos2, (widthBoton, heightBoton))
    r_boton_save.topleft = [10, 10]
    r_boton_log.topleft = posBotonEsquina(posLog, (widthBoton, heightBoton))

    botonesMaximo.append(
        {'imagen': mas_boton_imagen, 'rect': r_boton_mas, 'selected': False, 'texto': [""], 'pos':
posMas})
    botonesMaximo.append(
        {'imagen': menos_boton_imagen, 'rect': r_boton_menos, 'selected': False, 'texto': [""], 'pos':
posMenos})
    botonesMaximo.append(
        {'imagen': save_boton_imagen, 'rect': r_boton_save, 'selected': False, 'texto': [""], 'pos':
posSave})
    botonesMaximo.append(
        {'imagen': mas_boton_imagen, 'rect': r_boton_mas2, 'selected': False, 'texto': [""], 'pos':
posMas2})
    botonesMaximo.append(
        {'imagen': menos_boton_imagen, 'rect': r_boton_menos2, 'selected': False, 'texto': [""],
'pos': posMenos2})
    botonesMaximo.append(
        {'imagen': log_boton_imagen, 'rect': r_boton_log, 'selected': False, 'texto': [""], 'pos':
posLog})
```

```
############################################################################ Botones Calibracion ######################################################
def botonesCalScreen(pygame, botonesCalibracion):
    widthBoton = 80
    heightBoton = 80

    margen = 0

    posMas = [map_width - margen - (widthBoton / 2)-13, 240]
    posMenos = [margen + (widthBoton / 2)+13, 240]
    posSave = [40, 40]
    posReturn = [scrn_width - 50, 50]
    posObturador = [240, 100]
    posReTry = [350,100]
    posVarCalibracion = [scrn_height / 2, 240]

    # Cargamos las imagenes que serviran como botones
    menos_boton_imagen = pygame.image.load("imagenes/back.png")
    menos_boton_imagen_dos = pygame.image.load("imagenes/backGray.png")
    mas_boton_imagen = pygame.image.load("imagenes/fordward.png")
    mas_boton_imagen_dos = pygame.image.load("imagenes/fordwardGray.png")
    save_boton_imagen = pygame.image.load("imagenes/save.png")
    save_boton_imagen_dos = pygame.image.load("imagenes/saveUnavailable.png")
    return_boton_imagen = pygame.image.load("imagenes/return.png")
    shutter_boton_imagen = pygame.image.load("imagenes/shutter.png")
    shutter_boton_imagen_dos = pygame.image.load("imagenes/checkedShutter.png")
    reTry_boton_imagen = pygame.image.load("imagenes/reTry.png")
    reTry_boton_imagen_dos = pygame.image.load("imagenes/empty.png")

    # Se escalan las imagenes cargadas
    menos_boton_imagen = pygame.transform.scale(menos_boton_imagen, [widthBoton,
heightBoton])
    menos_boton_imagen_dos = pygame.transform.scale(menos_boton_imagen_dos,
[widthBoton, heightBoton])

    mas_boton_imagen = pygame.transform.scale(mas_boton_imagen, [widthBoton,
heightBoton])
    mas_boton_imagen_dos = pygame.transform.scale(mas_boton_imagen_dos, [widthBoton,
heightBoton])

    save_boton_imagen = pygame.transform.scale(save_boton_imagen, [50, 50])
    save_boton_imagen_dos = pygame.transform.scale(save_boton_imagen_dos, [50, 50])

    return_boton_imagen = pygame.transform.scale(return_boton_imagen, [50, 50])
    shutter_boton_imagen = pygame.transform.scale(shutter_boton_imagen, [60, 60])
    shutter_boton_imagen_dos = pygame.transform.scale(shutter_boton_imagen_dos, [60, 60])
```

```
    reTry_boton_imagen = pygame.transform.scale(reTry_boton_imagen, [40,40])
    reTry_boton_imagen_dos = pygame.transform.scale(reTry_boton_imagen_dos, [40,40])

    # Se crea el rectangulo de cada boton
    r_boton_mas = mas_boton_imagen.get_rect()
    r_boton_menos = menos_boton_imagen.get_rect()
    r_boton_save = save_boton_imagen.get_rect()
    r_boton_shutter = shutter_boton_imagen.get_rect()
    r_boton_return = return_boton_imagen.get_rect()
    r_boton_reTry = reTry_boton_imagen.get_rect()


    # Asignacion de posiciones
    r_boton_mas.center = posMas
    r_boton_menos.center = posMenos
    r_boton_save.center = posSave
    r_boton_shutter.center = posObturador
    r_boton_return.center = posReturn
    r_boton_reTry.center = posReTry

    botonesCalibracion.append(
        {'imagen': save_boton_imagen, 'imagen_dos': save_boton_imagen_dos, 'rect':
r_boton_save, 'selected': False, 'pos': posSave, 'habilitado': False})
    botonesCalibracion.append(
        {'imagen': return_boton_imagen, 'imagen_dos': return_boton_imagen, 'rect':
r_boton_return, 'selected': False, 'pos': posReturn, 'habilitado': True})
    botonesCalibracion.append(
        {'imagen': menos_boton_imagen, 'imagen_dos': menos_boton_imagen_dos, 'rect':
r_boton_menos, 'selected': False, 'pos': posMenos, 'habilitado': False})
    botonesCalibracion.append(
        {'imagen': mas_boton_imagen, 'imagen_dos': mas_boton_imagen_dos, 'rect': r_boton_mas,
'selected': False, 'pos': posMas, 'habilitado': False})
    botonesCalibracion.append(
        {'imagen': shutter_boton_imagen, 'imagen_dos': shutter_boton_imagen_dos, 'rect':
r_boton_shutter, 'selected': False, 'pos': posObturador, 'habilitado': True})
    botonesCalibracion.append(
        {'imagen': reTry_boton_imagen, 'imagen_dos': reTry_boton_imagen_dos, 'rect':
r_boton_reTry, 'selected': False, 'pos': posReTry, 'habilitado': False})
```

Annex 35b: Raspberry Pi code (button interface)