# Thermal Camera

## Introduction:

Thermal cameras are electronic devices which detect infrared wavelengths and allow to detect heat signatures [1]. One such use, is in infrared spectrophotometry, where information about exothermic or endothermic chemical reactions can prove important in determining the molecules present within compounds.

The implementation consists of a device that facilitates the observation of thermal reactions in microfluidic systems, using an AMG8833 thermal camera which uses the near-infrared spectrum (NIR: 780nm – 3000nm). The implementation uses a Raspberry PI to process the information from the thermal camera, and to display the information through a monitor connected through HDMI. The Raspberry PI also serves as a hub which not only runs the software, but serves the purpose of a server, which allows the user to export the data obtained by the thermal camera through the thermal imaging process. It also allows the user to interact with an Arduino microcontroller through a serial monitor interface, which serves the same purpose as the Arduino IDE monitor.

The calibration employed as well as the software implemented are also shown. The calibration is done through linear regression, using the temperature average of the 4 center pixels of the AMG8833 thermal camera. A calibrator circuit which is controlled by the Arduino microcontroller is proposed as to allow for temperature adjustments without having to resort to heating plates or thermal baths as a temperature reference. The calibrator circuit, uses current mirrors to heat resistors to different temperatures, creating a temperature gradient ring, which the Raspberry PI can interpret as temperature references for the linear regression curve.

## Materials:

1. 1x AMG8833 Thermal Camera
2. 1x 5V Relay
3. 1x ULN2003
4. 1x LM7805
5. 1x 100pF Capacitor
6. 10x 1.5 Ohm Resistor 1/4W
7. 1x 470 Ohm resistor 2W
8. 1x 270 Ohm resistor 2W
9. 1x 200 Ohm resistor 2W
10. 1x 150 Ohm resistor 2W
11. 1x 120 Ohm resistor 2W
12. 1x 100 Ohm resistor 2W
13. 6x LM35
14. 1x AC-DC 12V 1.5A Adapter

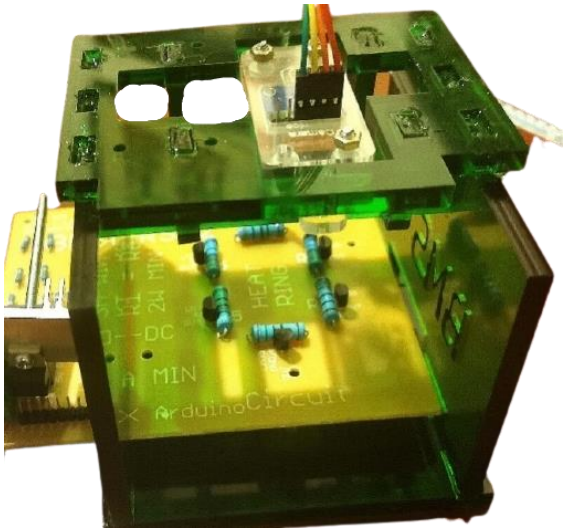


*Illustration 1: Thermal Camera Implementation*

## Component Explanation:

**AMG8833:**

The AMG833 is a thermal sensor which measures temperatures from 0°C to 80°C at up to 7 meters. The sensor which is developed by Panasonic features an 8x8 grid which allows to sense temperatures at about 10 fps. The sensor is able to communicate through I2C protocol and is compatible with both the Raspberry PI as well as an Arduino microcontroller. It has a sensing

uncertainty of 0.25°C, and is able to operate in the range of 3.3-5V. It should be noted that the sensor has a 15 second delay after start up to stabilize the output temperatures being measured by the thermistors [2]. It should also be noted that for the sensor to function properly, the sensor has to be calibration with respect to the distance from the sensed object.

**ULN2003:**

The ULN2003 is an integrated circuit which is composed by 7 identical and independent drivers, which can drive up to 500mA; each driver is composed by a Darlington Pair. The circuit has a maximum Vce of 50V, while to get the drivers into saturation it only requires between 1V and 1.3V [3]. For this implementation, the ULN2003 integrated circuit is operated as a current mirror. The implementation was opted using this circuit, due to the fact that for the current mirror to function properly, the transistors used need to have the same characteristics and the same gain (β) values. It should be noted that transistor arrays such as the CA3046 NPN array as well as the CA3081 in emitter configuration would work just as well.

The following illustration shows the Darlington pair of one of the seven drivers in the ULN2003 [4].

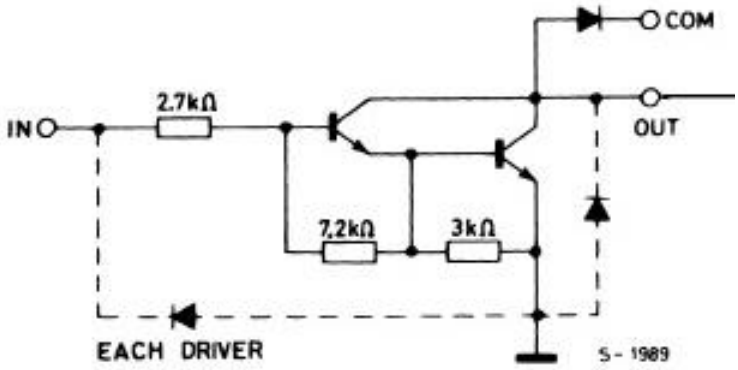



*Illustration 2: ULN2003 Darlington Pair*

**Arduino:**

An Arduino is a microcontroller board based on the ATmega328P. It has an unregulated power source in the VIN pin which supports 7-12V, and a regulated 5V power source [5]. For this implantation the Arduino is the device which controls the temperature gradient ring which serves as the AMG8833's calibration circuit. The Arduino microcontroller operates the relay which closes the current mirror circuit and allows for the gradient ring to heat up, as well as take the measurements from the LM35 temperature sensors in each of the ring's 2W resistors to correlate the data. The Arduino is this implementation communicates with the Raspberry PI through the serial's write() and read() functions.

**Raspberry PI:**

The Raspberry Pi is a low-cost, small-size computer which allows the user to run code [6]. For this implementation, the software as well as the serial interface with the Arduino microcontroller are implemented through python code. In this implementation, the job of the Raspberry PI is that of displaying the 8x8 temperature matrix obtained through the AMG8833, as well as the Arduino's IDE monitor. The Raspberry PI uses the Serial communication library to emulate the Arduino's monitor. The software also allows capture images and record video of the temperature matrix. The option to connect and control the software through a local network is also proposed.

**Calibration Circuit:**
The calibration circuit implemented consists of a series of current mirrors which are connected to a series of resistors, which heat up as the current passes through them. The circuit has pins to interact with the Arduino microcontroller as well as a terminal block to connect the 12V 1.5 AC-DC adapter, which provides the current for the temperature gradient ring.

The Arduino has a pin connected to the circuit's relay which grounds the current mirror and allows for the current to flow through the 2W resistors. It should be noted that the relay grounds all of the drivers in the ULN2003 at the same time, since the drivers are independent. The time the current flows through the resistors can be modified in the Arduino code to alter the temperature reached by the gradient ring. It should also be noted that the ULN2003 may get hot, as each of the drivers will experience approximately 250mA for the duration that the relay remains active. As such, a heat sink may need to me employed.

The temperature gradient ring has LM35 sensors in close proximity (1mm) to each of the 2W resistors, to get the resistor temperature and allow for a correlation between the LM35 values and the values measured by the AMG8833 thermal camera. It should be noted that the temperature gradient ring shape can be modified, as long as all of the resistors are visible in the 8x8 matrix and are not too close together as to create temperature overlapping.
The circuit diagram can be observed in illustration 4. The left section which incorporates the transistors, is an emulation of the ULN2003 as such, the ULN2003 would replace that section of the circuit.

## Overall Diagram:

The overall circuit diagram with both the Arduino microcontroller as well as the Raspberry PI and the calibration circuit are shown below. It should be noted that for the proper operation of the device, a mouse, a keyboard as well as a screen monitor should be connected to the Raspberry PI to allow the user to interact with the thermal camera software. The overall implementation is to use two AC-DC adapters, since one powers the Raspberry PI and the second one is to power the calibration circuit, which is shown as a 12V power source in illustration 4.

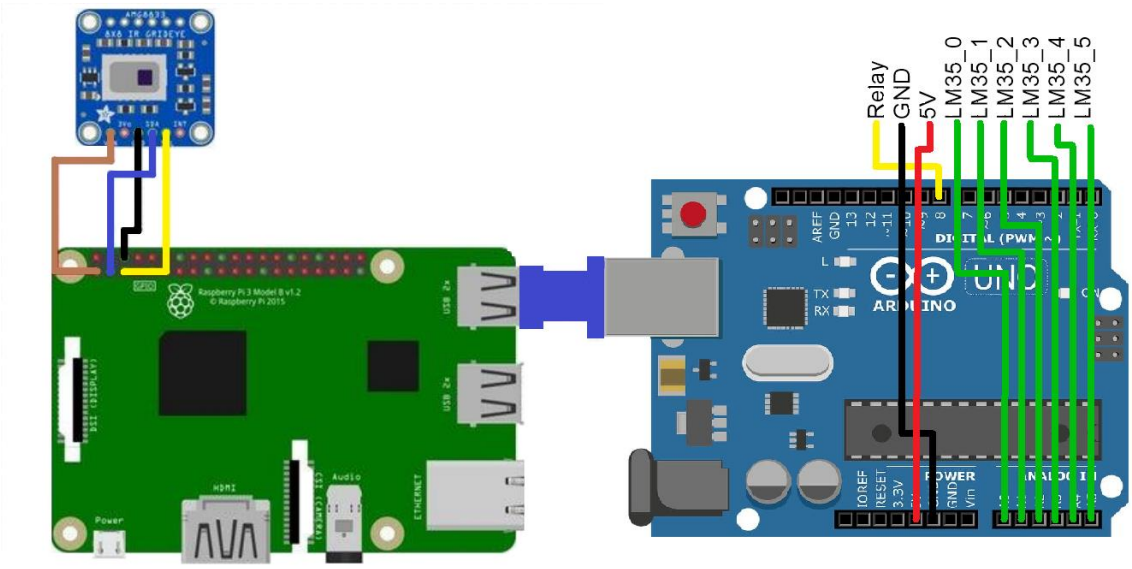



*Illustration 3: Processor Diagram*

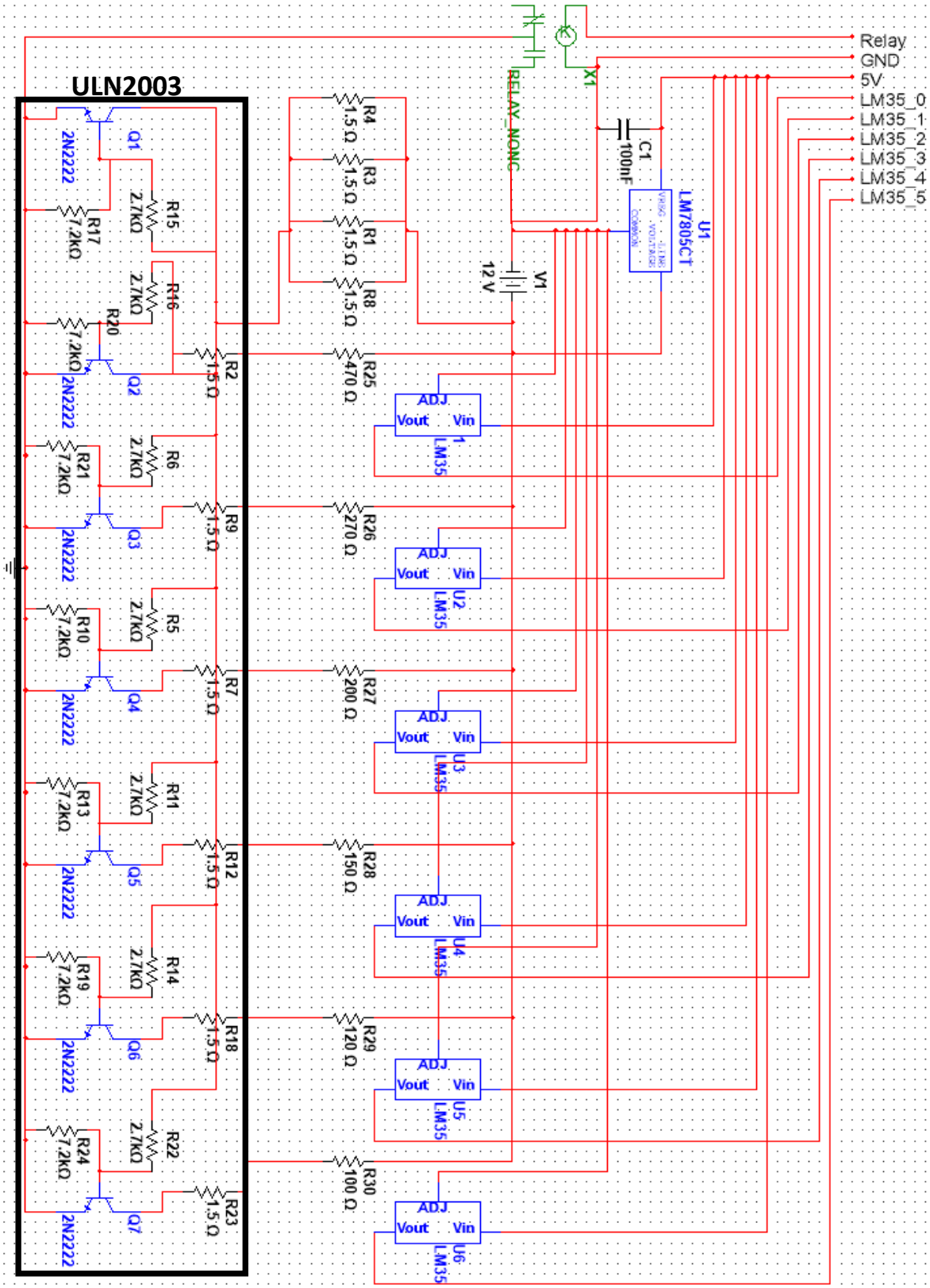



*Illustration 4: Calibration Circuit Diagram*

## Implementation:

**Casing:**

The casing for the thermal camera is done using acrylic. The devices casing is implemented using 5mm acrylic, while the AMG8833 has its own separate casing as to protect the thermal camera using 2mm acrylic. The devices casing holds the AMG8833 casing as well as the calibration circuit. The 5mm acrylic casing can also hold a microfluidic system at a specified distance from the thermal camera's lens.

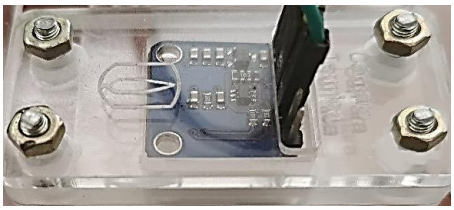


*Illustration 5: AMG8833 Casing*

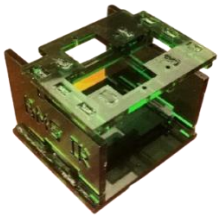


*Illustration 6: Device Casing*

Illustration 5 shows the AMG8833's 2mm casing; the illustration 6 shows the devices 5mm acrylic casing. The 2mm and 5mm casing designs can be observed in Annex 1 and 3, respectively. The microfluidic system design can also be observed in Annex 2. It should be noted that acrylic is transparent for a portion of the wavelength in the NIR spectrum, as such, using the device in an environment with thermal noise, could create uncertainties in the measuring process.

**PCB:**

The calibration circuit's PCB is shown below. The PCB has anchor points which allow the circuit to be secured to the 5mm acrylic casing. It should be noted that the heat ring's position is absolute in the Raspberry PI's code, thus, if the circuit's 2W resistor positions are modified, the code should also be modified to account for the changes. A higher resolution diagram can be observed in Annex 4.

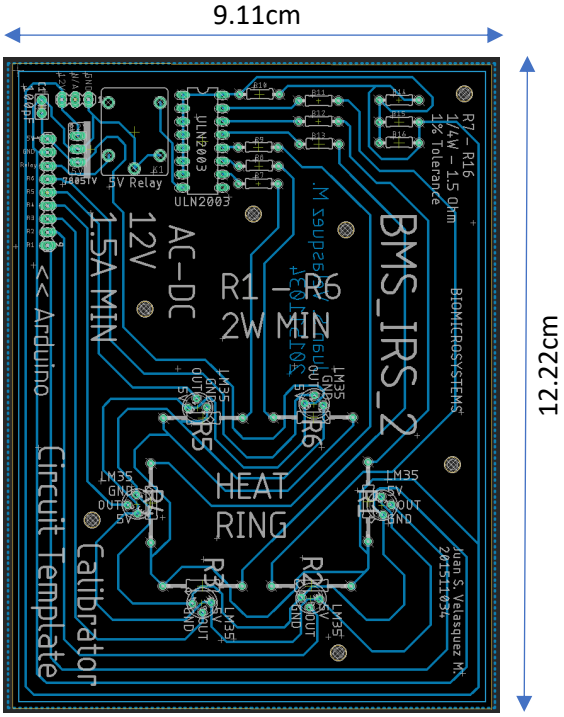



*Illustration 7: Calibration Circuit PCB Diagram*

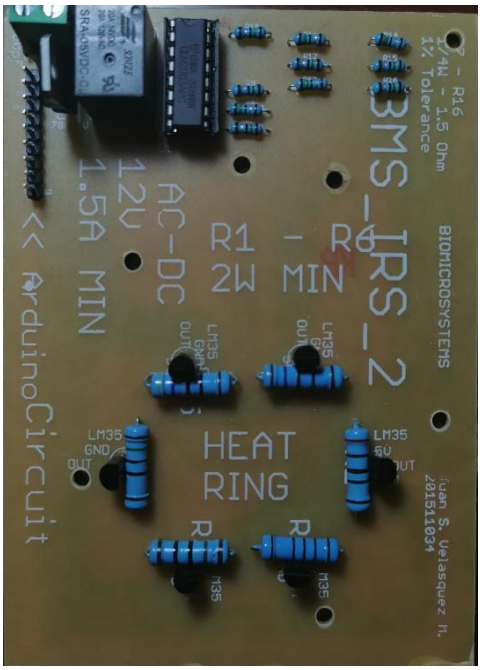



*Illustration 8: Calibration Circuit PCB Implementation*

## Calibration:

**AMG8833 Geometry:**

To properly calibrate the thermal camera, it is necessary to observe the laser distribution of the AMG8833 sensor. [2] shows the laser distribution as a function of their angle. A python code is implemented to transform the laser distribution as a function of distance. The python code can be observed in Annex 5. This tool allows to observe what the AMG8833 sees at a certain distance; this information allows to know which pixel readings accurately represent the thermal images obtained from measuring the microfluidic system. This implementation is needed as the area covered by the thermal camera is not a square, but a shape with elongated corners.

The python code also allows to observe which pixels of the AMG8833 read the 2W resistors in the calibration circuit. This allows the device to correlate the information obtained form the LM35 sensors in the calibration circuit, with the temperature measured by the thermal camera. The following illustrations show the laser distribution of the AMG8833 sensor at a distance of 0.5cm using the python code implemented.

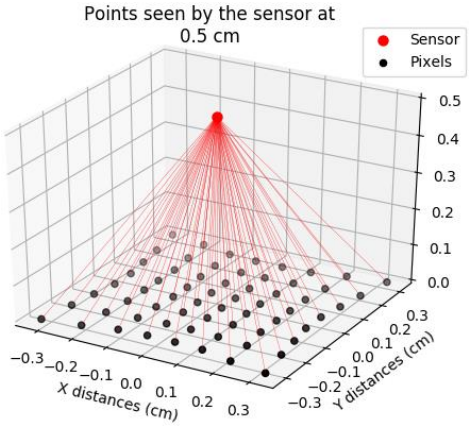



*Illustration 9: Laser Distribution Isometric View*

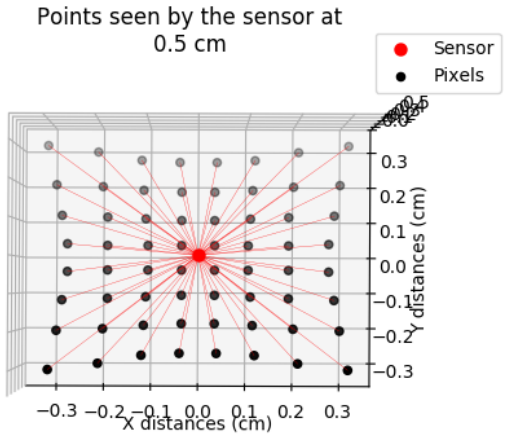



*Illustration 10: Laser Distribution Top View*

It should be noted that the distance between the thermal camera and the microfluidic system is optimized to allow the lasers to contact the microfluidic system's canal.

**AMG8833 Distance vs. Temperature:**

Since the thermal camera uses laser refraction to measure the temperature of the desired surface, the distance from the lens to the surface creates uncertainties in the measured value. Thus, it is necessary to create a calibration curve to account for the laser refraction variations due to distance.

This process can be recreated by placing the camera at a known distance from a heated surface. The calibration to account for the refraction variations was done at both 0.2cm as well as at 4.75cm. The former is the distance between the thermal camera and the microfluidic system and the latter is the distance between the thermal camera and the calibration circuit's gradient ring. The calibration process was conducted using the same temperatures at both distances to create a calibration curve which accounts for the refraction variations.

This calibration curve would also allow to correlate the data obtained by measuring the temperature gradient ring at 4.75cm and infer how the sensor should compensate to properly measure the

temperatures at 0.2cm and corroborate the calibration process. This calibration was conducted at both 37°C and 52°C as follows.

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

*Table 1: Distance Correction*

These correction values allow for an estimate at any other temperature at a distance of 0.2cm. However, it should be noted that more points would provide for a more accurate prediction, using linear regression. Using this information, the correction for a temperature of 35°C at 0.2cm would approximately 5.1°C.

**Calibration Circuit LM35 Sensors:**

For the device to correctly correlate the data form the calibration circuit, the temperature sensors in the circuit need to be calibrated. The calibration is conducted using a heated surface as well as a commercially calibrated thermal camera or thermometer. Each of the LM35 sensors are placed in contact with the heated surface and a linear regression is conducted in the range of 20°C to 60°C. The temperature range can be defined by allowing current to flow through the heat ring for a specific amount of time and measuring the heat produced by each for the resistors using the commercial thermal camera or high-resolution thermometer.

The linear regression equations obtained for each of the temperature sensors, can then be implemented into the Arduino microcontroller to compensate for measurement uncertainties in the heat ring. It should be noted that the higher the resolution of the device used for this calibration, the better the correlation between the calibration circuit and the thermal camera.

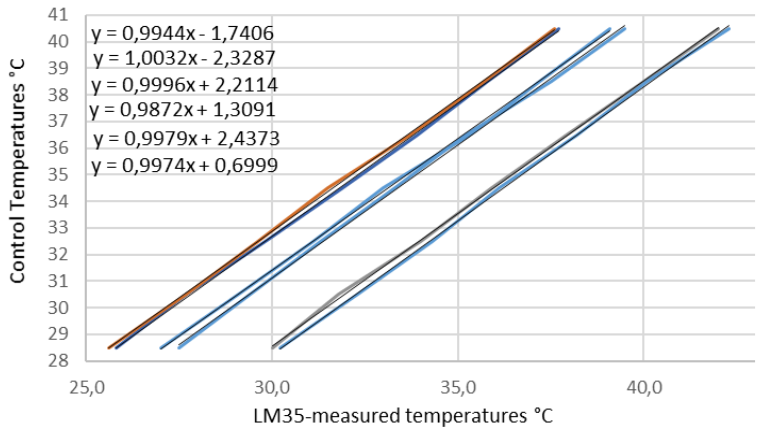



*Illustration 11: LM35 calibration Curves*

Although the LM35 sensors show a linear response, some have an offset. The data obtained show a maximum offset response of approximately 2.4°C. To further improve the measurements of the gradient heat ring by the LM35 sensors, thermal paste can be applied to increase the thermal conductivity to the temperature sensor. The thermal paste decreases the variations of the measurements as can be observed as follows. An illustration of the HY510 thermal paste used is also shown.


*Illustration 12: Thermal Paste*

| LM35 measurements 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 |

The gradient heat ring's resistor temperatures after 50 seconds of operation are observed above, using the HY510 thermal paste. Four measurements were taken to make sure that the resistors reach the same temperature after a certain operation time. The calibration circuit had a maximum variation of 9.2%, however, it should be noted that the variation is also seen by the AMG8833 thermal camera, thus, making the correlation possible.

## Linear Regression:

**Manual Regression:**

The linear regression process implemented in the software for the AMG8833 thermal camera consists of a .txt file which holds the constant of the Y = mX + b, where x is the average temperature of the four center pixels of the AMG8833 thermal camera. The calibration using this method takes a snapshot of the temperature array measured by the sensor and using the linregress function from the SciPy python library.

As such, the linear regression equation if no calibration is done to the device, is Y = 0.25X. Using the manual regression to calibrate the device, the equation obtained is Y = 0.2531X – 4.7765, which is then applied to a surface at a distance of 0.2cm. Using these calibration constants, the device is then used to measure the temperature of a microfluidic system at 35°C to observe is the calibration process is reliable. It should be noted that the 'b' constant already has the distance vs. temperature calibration correction of 5.1°C for 0.2cm at 35°C, resulting in a value of -4.7765.

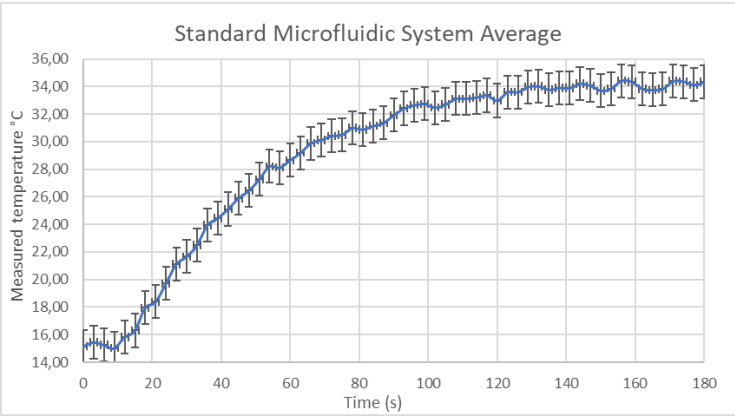



*Illustration 13: Device Calibration Results for 35°C*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 |

*Illustration 14: Device's Thermal Map for 35°C*

Although there are some variations, the maximum relative error present in the temperature array is approximately 5.7%. Thus, the calibration process is proven reliable. This calibration process is then scaled for the temperature gradient ring.

**Temperature Gradient Ring:**
This process is then automated using the calibration circuit to create the linear regression using the linregress function of the SciPy library. The calibration circuit and the AMG8833 thermal camera are placed in the casing and the pixels which correspond to the resistors are identified. Annex 6 shows the pixel number as well as the pin number which corresponds to the Arduino microcontroller. The pixel number is important, as the python code which measures the temperature array needs to correlate the resistor's pixel with the temperature measured to properly create a calibration curve using the data obtained by the calibration circuit.

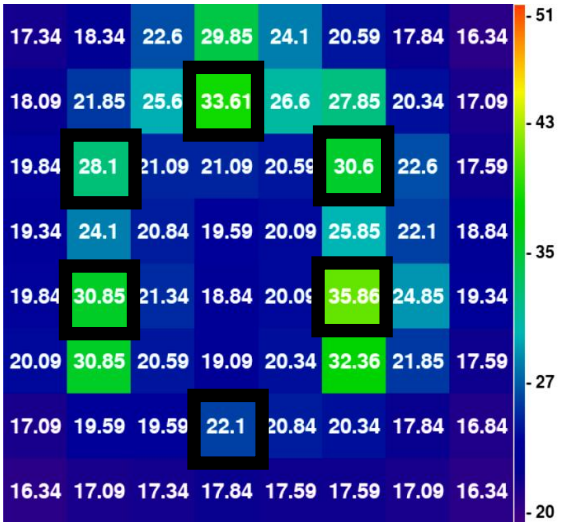


*Illustration 15: Pixelated Thermal Map*

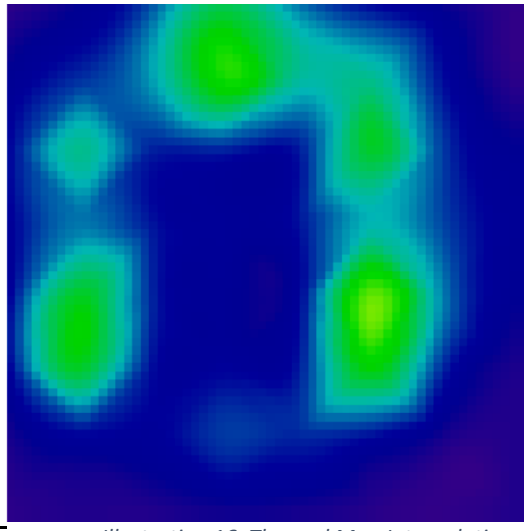


*Illustration 16: Thermal Map Interpolation*

It should be noted, that the illustration shown does not represent the temperature of the gradient heat ring after the 50 seconds of operation, but rather it shows the gradient heat ring as it heats up. Both illustrations are obtained using the software's .png capture feature.

Using this implementation, the calibration process is done four times to corroborate if it is reliable. The calibration equation obtained is Y = 0.274X – 4.955, which shows that there is about an 8% variation when compared with the manual linear regression calibration process performed.

## Code:

**Arduino:**
The Arduino code makes use of Serial communication to interpret the commands which the user sends through the Raspberry PI to control the calibration circuit. It records the data obtained by the calibration circuit and stores it using the EEPROM library. The device allows for various commands such as clc (clear), hlp (help), get and cal (calibrate).

It should be noted that if the command sent indicates for the Arduino to start the calibration process using the calibration circuit, the Arduino will be unresponsive for the duration of the calibration. Meaning, while the resistor's heat up, the Arduino can receive commands, but will not process them

until the relay which controls the current flow to the calibration circuit's resistors is turned off. After which, the user can ask the Arduino for the data using the get command.

It should be noted that all of the LM35 calibration equations are implemented at the moment the Arduino measures the resistor's temperature. The Code can be observed in Annex 7.

**Raspberry PI:**
The Raspberry PI's code is shown in Annex 8.

**Interface:**
The device's interface consists of a window created using PyGame which shows the heat map seen by the thermal camera. It also shows, a list menu which allows to capture both images and video, a slider which allows to modify the pixelated heat map into a more detailed image by using interpolation, as well as a settings menu, which allows to modify the parameters of the device.

The interpolation is implemented using the SciPy interpolation library which allows to modify the pixelated heat which shows an 8x8 matrix to a 64x64 image; as seen in illustration 16. It should also be noted that the illustrations 15 and 16 are not the same image, as the device and interface work in real time. Thus, it is only possible to observe either the pixelated heat map or the interpolated image. The photo and video capture are implemented using a subprocess, thus the device takes snapshots of the interface to create the different files. It should be noted that if an implementation is done using a different library which allows to capture both photo and video directly using the information form the AMG8833 thermal camera, it'll be possible to capture both types of images. The device allows to choose between different extension types when capturing video or saving data, whether it be in a .png format or in text such as .txt and .csv. The video capture allows for both .mp4 as well as .mkv and either full screen or just the heat map. The device also features a file explorer using a tKinter file directory, which allows to select the folder which will store the files created by the device.

The interface settings screen allows to modify the temperature limits, which determine the color range observed by the user. Using the same file which holds the constants for the Y = mX +b calibration equation, the temperature limits are saved for when the device is power off. The settings screen also allows to access the manual linear regression calibration, which creates a linear regression using the linregress function of the SciPy library.

**Server:**
The python code allows to start a socket connection which allows a user in a local network to control the device's interface and collect data using the AMG8833 thermal camera. The server also allows to export files to the client using a file explorer implemented through the client's command line. To show the user running the server, the interface shows if a client has connected; it shows both the status of the client as well as the local IP which connects to it. The server, however, only allows for a client to connect, once the server function is called by the server interface.

The server implementation recognizes commands which allow to modify the temperature limits as well as the data and video extension. When the user calls for data collection or video capture through the client, the server automatically sends the file created through the TCP connection. However, the server allows to export files of the extension supported by the device by using the file explorer through the command line.

For the client to successfully disconnect from the server, a command is also sent, which either disconnects the current user from the current socket connection, or shuts off the server. At which point if another client is to connect, the server needs to be restarted from the server interface.

**Serial Communication:**
The serial communication with the Arduino microcontroller is implemented using the Serial library. A textbox is implemented using PyGame which sends the communication to the microcontroller, which then answers with the command sent and operates the calibration circuit. It should be noted that for the serial communication to work, the microcontroller's address is needed for the Raspberry PI to know to which port the information is headed. For this implementation, the Arduino's port address is directly linked to the text box. This would allow to communicate with multiple microcontroller's by implementing more instances of the text box class.

When the command to get the calibration, information is sent to the Arduino, the Raspberry PI overwrites the calibration file, once the data is received. This calibration is done using the SciPy library.

**Calibration:**
The manual calibration process implemented in the interface, allows to take multiple readings (at least two) of heated surfaces to correct or adjust the measured temperature to create a linear regression equation. The calibration screen features a save button to create or modify the file with the calibration constants as well as arrows to adjust the measured temperature.

In the case of the calibration circuit data, once the Arduino sends the data, the software loads the data in vectors to create the calibration constants using the linregress function.

**Client Code:**
The client implementation to control the device is done through the command line. The code allows to control the main implementation through a local network as well as modify the temperature constants for the heat map.

It features a help command which shows the user all of the commands supported by the code. The code can be observed in Annex 9.

## Recommendations:

Regarding the calibration process, the procedure should be performed din a place with little to no thermal noise apart from the heated surface. Since the AMG8833 behaves like an infrared thermometer, any thermal fluctuations in the air can throw off the reading and create variations which in turn create uncertainties in the collection of data with the device.

It should also be noted that the warm air which is a consequence of the heated surface can also create thermal fluctuations which create uncertainty in the data collected. As the temperature of the heated surface gets higher, it is more likely that the calibration process would have errors. Thus, it is recommended that for temperatures above 50°C, the calibration process should be performed three times to make sure that the calibration constants obtained are reliable and accurately represent the data. This also has to be considered when using the calibration circuit, which depending on the time of operation can reach temperatures of up to 80°C.

## Observations:

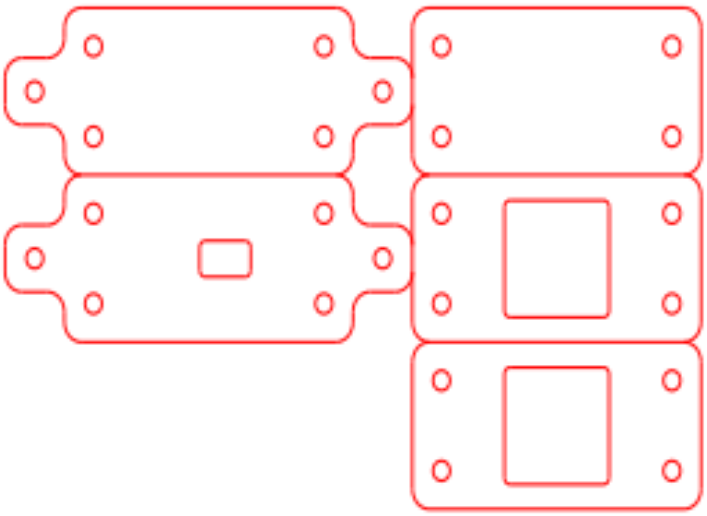
Regarding the implementation's code, it should be noted that the use of threads allows for the parallel operation the AMG8833 as well as the server which allows the interface to be controlled through a local network. Although this implementation allows for a very flexible data collection system, if the processor running the code has to run other processes, the program may become unresponsive. This software was used and tested as the only program running in the Raspberry PI at the time, thus allowing for very smooth data collection.

It should also be noted that the casing implemented allows for an Adafruit AS7262 6-channel visible light sensor next to the AMG8833 thermal camera. Using this implementation, would allow to make a simple spectrophotometer.

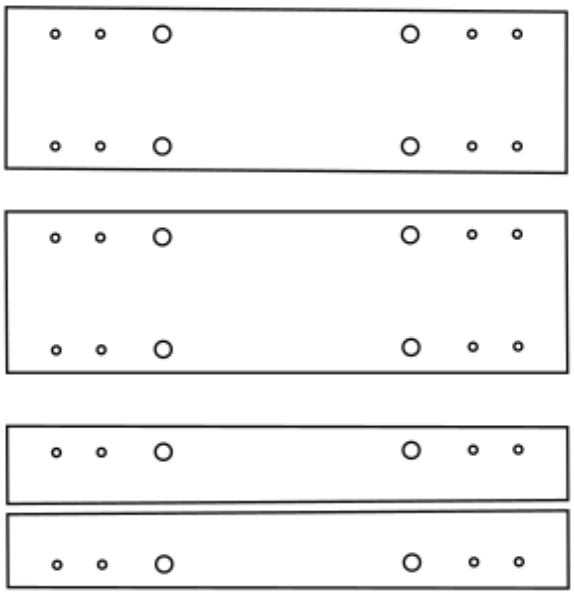
## References:


1. *Everything You Need To Know About Thermal Imaging Cameras*. Everything You Need To Know About Thermal Imaging Cameras | RS Components. (n.d.). Retrieved September 9, 2021, from https://uk.rs-online.com/web/generalDisplay.html?id=ideas-and-advice%2Fthermal-imaging-cameras-guide.
2. Panasonic Corporation. (2011, August 30). *SPECIFICATIONS FOR Infrared Array Sensor AMG8833*. Google Drive. Retrieved September 9, 2021, from https://drive.google.com/file/d/1wcXSsOnkCn4Oi7sN5cG3tklMu7beDR9c/view.
3. Gabriel. (2018, February 23). *EL ULN2003: DRIVER DE SALIDA PARA MICROCONTROLADORES*. Inventable. Retrieved September 9, 2021, from https://www.inventable.eu/2018/02/09/uln2003-driver-salida-microcontroladores.
4. *ULN2003A Darlington Array 7 NPN 500mA*. Hobby Electronics. (n.d.). Retrieved September 9, 2021, from https://www.hobbytronics.co.uk/uln2003a-darlington-array.
5. *¿Qué es Arduino?* Arduino. (2017, July 12). Retrieved September 9, 2021, from https://arduino.cl/que-es-arduino/.
6. Rodríguez de Luis, E. (2018, September 18). *De cero a maker: todo lo necesario para empezar con Raspberry Pi*. Raspberry Pi: todo lo necesario para iniciarse como maker desde cero. Retrieved September 9, 2021, from https://www.xataka.com/makers/cero-maker-todo-necesario-para-empezar-raspberry-pi.

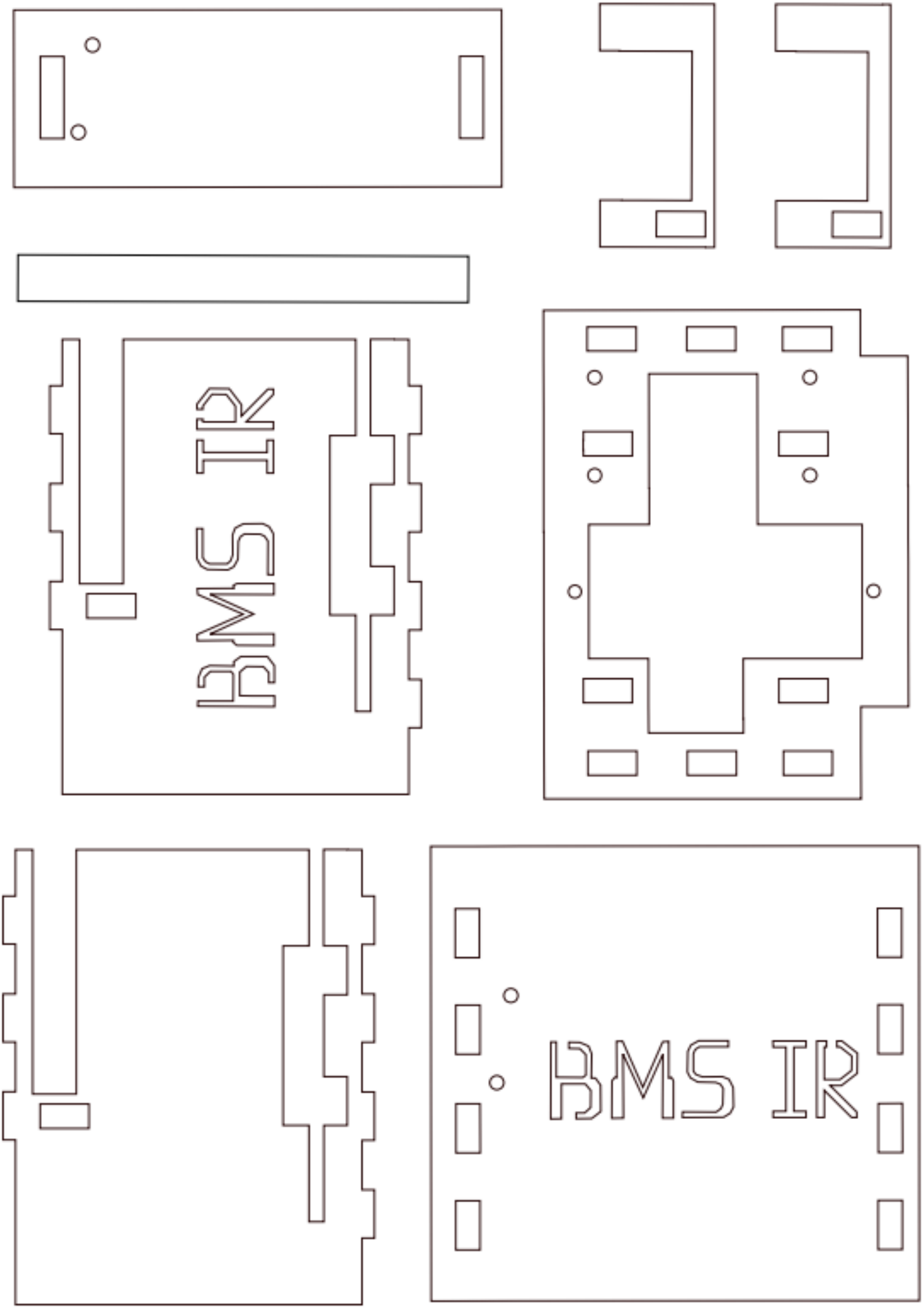
**Annexes:**



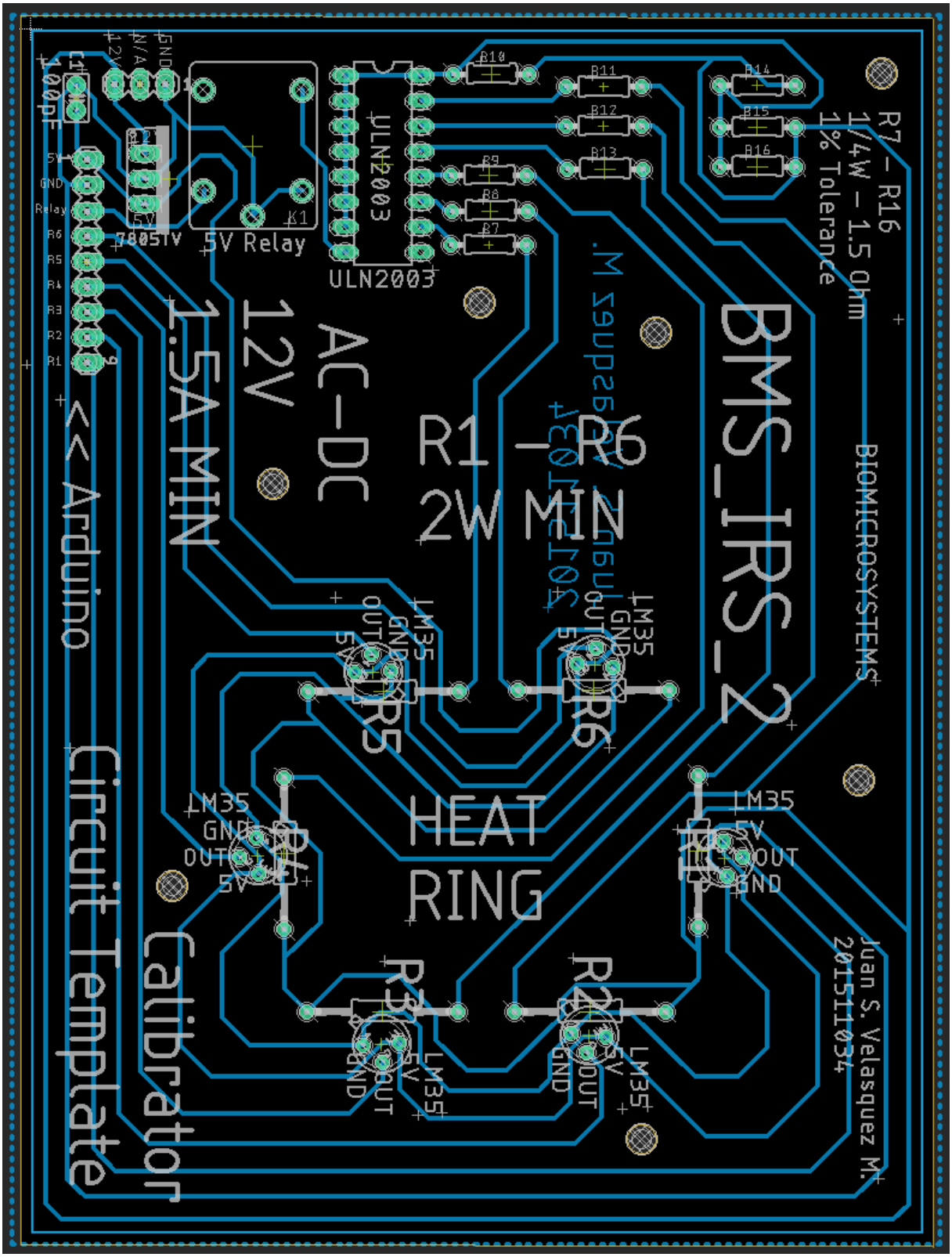
Thermal Camera Casing



Microfluidic System


BMS IR
BMS IR

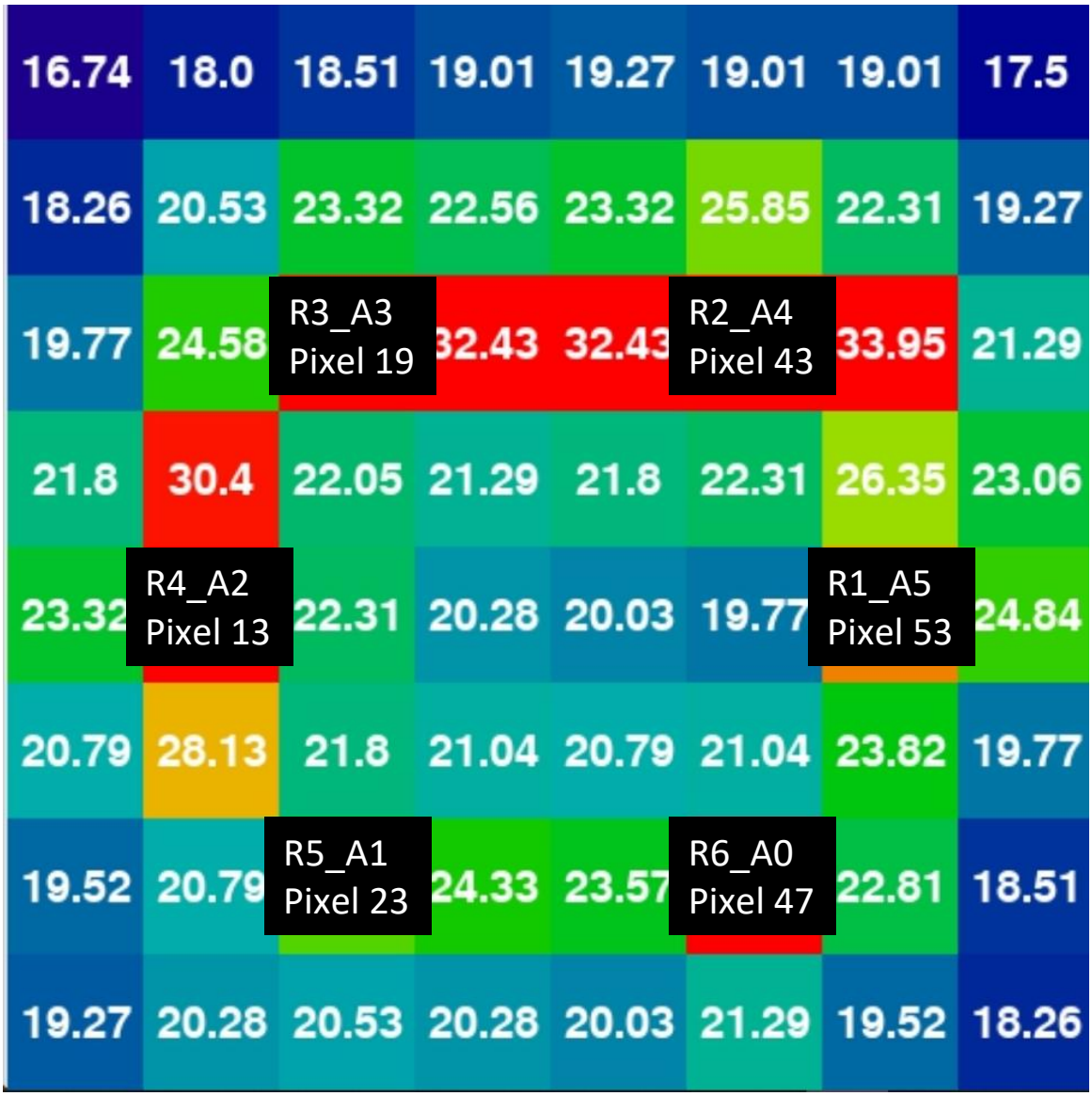
Device Casing




PCB Diagram

```
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()
```

Laser Distribution Code




Thermal Map Pixel Distribution

```
#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);
    }
  }
}
```

Arduino 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()
```

Raspberry PI Code (a)

```
#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})
```

Raspberry PI Code (b)

```
# -*- 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()
```

Client Code