UNIVERSIDAD DE LOS ANDES

PROYECTO DE GRADO

Puesta en marcha y calibración de un calorímetro 2277 de ThermoMetric

---

Autor:
Juan BARBOSA

Director:
Edgar Francisco VARGAS, Dr.Sc.

Proyecto de Grado para optar por el título de Químico

Termodinámica de Soluciones
Departamento de Química

2 de diciembre de 2018

Índice general

Índice de figuras

Índice de tablas

Control de la temperatura

El sistema cuenta con dos calentadores de precisión al interior del baño termostatado de 25 litros. Estos calentadores junto con el baño externo determinan la temperatura de trabajo del equipo. El baño externo se usa con el objetivo de mantener siempre activos los calentadores internos en algún nivel de potencia intermedio, pues el baño externo, al tener una temperatura inferior agregar una carga permanente a los calentadores internos [1]. Para monitorear la temperatura del baño el calorímetro cuenta con dos termistores, el primero para trabajar a temperaturas inferiores a 50 °C y el segundo para temperaturas superiores a este valor. La señal generada por uno de estos termistores se compara con un valor de resistencia definido por el usuario. La resistencia y por ende la temperatura del equipo se fijan usando cuatro resistencias de década, esto es que cada resistencia es una potencia de 10 menor que la anterior, las cuales se encuentran en la parte inferior del calorímetro, en la Figura 1.1 corresponden A, B, C, y D, de esta manera se pueden realizar experimentos a temperaturas en el rango de 5 a 80 °C. El diagrama del sistema de control térmico del calorímetro se muestra en el Apéndice A como Figura A.1.

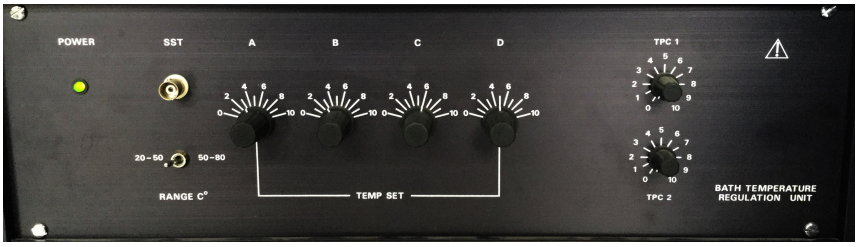


FIGURA 1.1: Resistencias de década que controlan la temperatura.

Para determinar la equivalencia de resistencia y temperatura el calorímetro cuenta con una tabla que relaciona los valores de resistencia y temperatura de baño externo que dan lugar a una temperatura constante en el baño interno. Dicha tabla se muestra en el **??** como Figura A.2, sin embargo, el uso de esa tabla está limitado a temperaturas

ambiente superiores a 20 °C por lo cual, en el caso particular de Bogotá rara vez se cumplen. Lo anterior es relevante dado que la resistencia eléctrica depende de la temperatura, en el caso de materiales metálicos, la resistencia aumenta con la temperatura, y para semiconductores se tiene el efecto contrario [2].

Una vez realizadas las conexiones eléctricas del calorímetro para su funcionamiento a 110 VAC, y antes de realizar una calibración química fue necesario estabilizar el equipo a una temperatura de 25 °C, puesto que el objetivo de la calibración química es contrastar los datos obtenidos con el calorímetro con los reportados en la literatura, los cuales por convención se reportan a estas condiciones. Para alcanzar esta temperatura se probó con las condiciones reportadas en la Figura A.2, en donde los valores de cada resistencia de década se muestra a continuación.

TABLA 1.1: Valores de las resistencias de década, para una temperatura de 25 °C, según calibración original del calorímetro.

Baño interno (°C)	A (10^4 Ω)	B (10^3 Ω)	C (10^2 Ω)	D (10^1 Ω)	**Baño externo (°C)**
25.0	3	2	4	9	22.0

La temperatura del equipo fue monitoreada por cuatro días, con al menos un registro diario, los resultados se muestran en la Tabla 1.2. En ella se puede observar que cuando inicialmente se creía que la temperatura del baño estaba cerca de estabilizarse a 25 °C, en realidad estaba en aumento de la temperatura, y cuando se creía que finalmente el calorímetro se había estabilizado en 26,86 °C el sistema se encontraba oscilando como lo muestra la última temperatura registrada.

TABLA 1.2: Registro de temperaturas del baño interno en el tiempo para la configuración recomendada por la calibración en la Figura A.2.

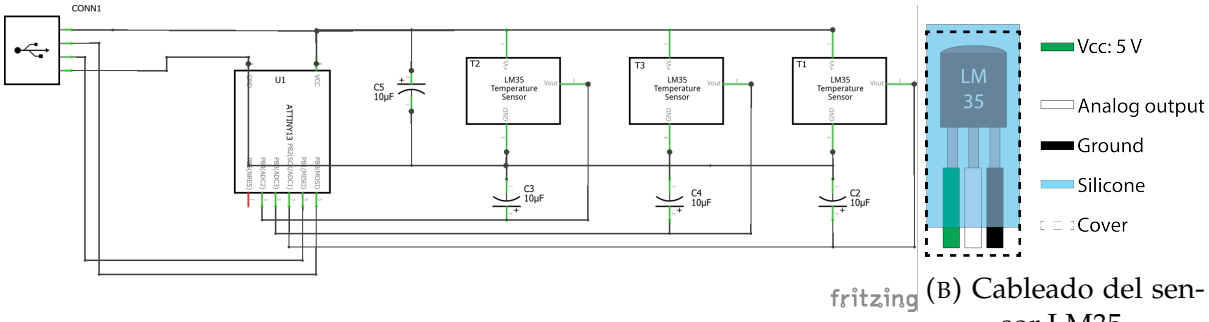
Fecha (DD/MM HH:MM)	**Temperatura (°C)**
07/09 11:13	25,14
07/09 11:50	25,19
10/09 09:43	26,64
11/09 09:22	26,86
11/09 10:00	26,86
12/09 08:30	26,86
12/09 10:30	26,86
12/09 15:48	26,64

Sin embargo, esta no fue la única configuración probada, pues también se trató de estabilizar el equipo usando: 2977 (A,B,C,D), que dio lugar a las siguientes temperaturas de baño interno 25,44 °C, y 28,38 °C luego de 5 días. De manera similar se probaron

3000, 3021, 3040, 3100, 3121, 3140, 3160 ninguna de las cuales presentó una temperatura estable. Debido a la dificultad de estabilizar el equipo a una temperatura determinada, se decidió construir un sistema de monitoreo automatizado para facilitar esta tarea, de esta manera se podría saber el histórico de una configuración, si el sistema está en calentamiento, estable, u oscilando.

1. Sistema de monitoreo

De todas las acciones que se hicieron a lo largo del proyecto, la estabilización de la temperatura es la que llevó más tiempo. Esto se debió a que existen dos variables que determinan una temperatura en el calorímetro, por un lado la configuración de las resistencias de década, y el baño externo, además se debe tener en cuenta el tiempo de estabilización del calorímetro, el cual puede tomar cerca de 6 horas en completarse. Para monitorear el estado de la temperatura del baño interno se construyó un circuito con tres sensores de temperatura, los sensores LM39 fueron elegidos debido a que presentan una respuesta lineal en voltaje, y tienen un rango de trabajo de 2 °C a 120 °C [3]. Cada uno de estos sensores fue conectado a tres cables como se muestra en la Figura 1.2b, y posteriormente fue cubierto con silicona y un cable termoencogible para aislar el sensor del agua.



(A) Conexión al microcontrolador.

(B) Cableado del sensor LM35.

FIGURA 1.2: Circuito del sistema de monitoreo de temperatura.

Los sensores a su vez fueron conectados a un microcontrolador ATtiny 13 [4].

Referencias

(1) Suurkuusk, J. *2277 Thermal Activity Monitor*; inf. téc.; Järfälla: Termometric AB.

(2) Simon, S. H., *The Oxford solid state basics*; OUP Oxford: 2013.

(3) Instruments, T. *LM35 datasheet, Aug* **1999**.

(4) Instruments, T. *ATtiny13 datasheet, Aug* **2010**.

Imágenes de soporte

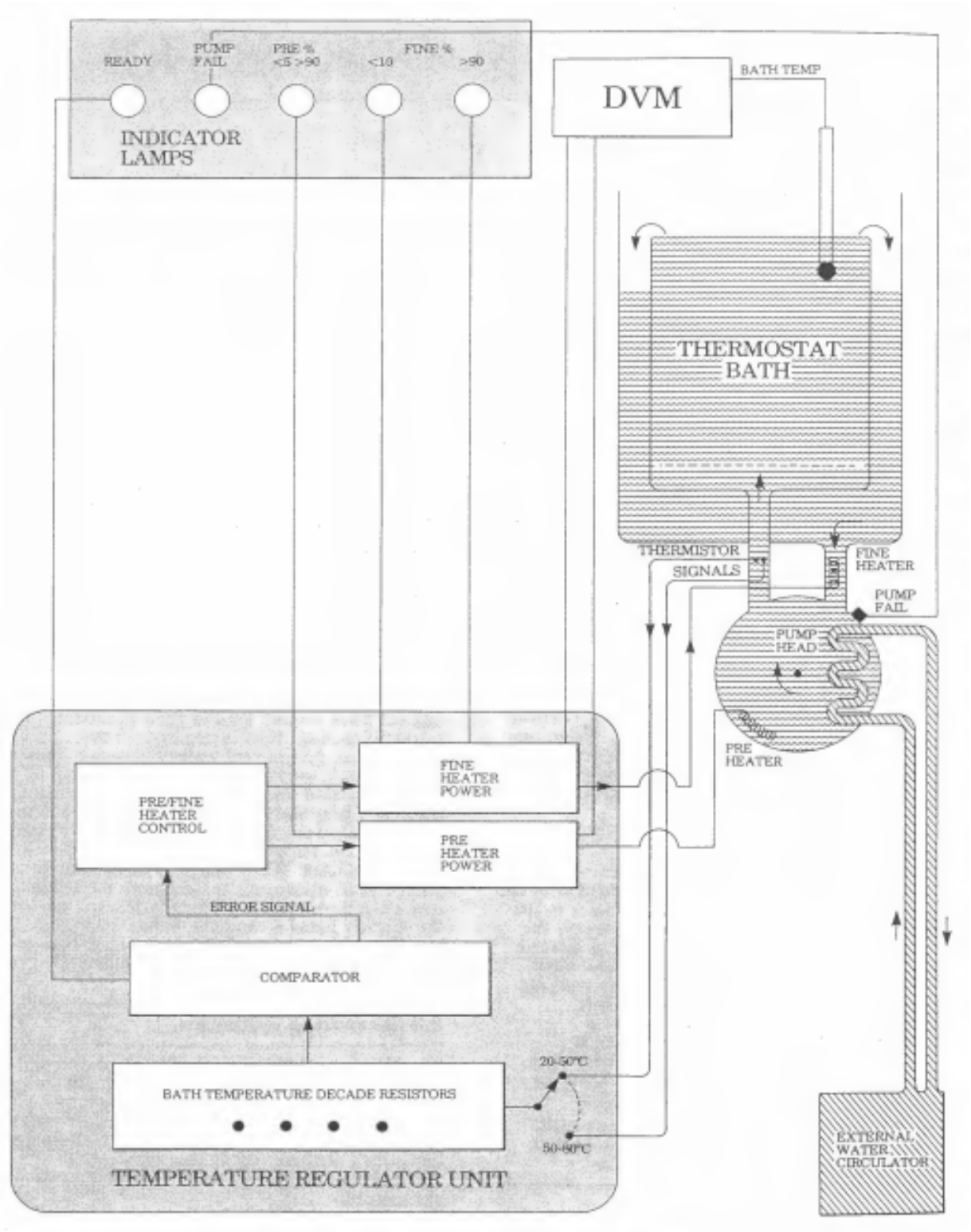


FIGURA A.1: Control térmico del calorímetro. Modificado de [1].

Thermal Activity Monitor No. 404
Date: 05-17-1994

Sensor #: 411B **Lowtemp**

Calibration data

Resistans Ω	Temp °C	Error °C
40607.50	19.999	0.000
25895.50	30.235	-0.000
17268.70	39.999	0.000
11656.50	50.006	-0.000

Thermistor constants
R=A* EXP(B/T+C/T^2)

A=0.11332690E-01
B= 4957.60
C= -156384.66
T=temp. in Kelvin

Bath temp °C	Decade set Ω/10	Circ temp °C	Bath temp °C	Decade set Ω/10	Circ temp °C
10	6474	6.0	30	2616	27.0
11	6171	7.0	31	2507	28.0
12	5884	8.5	32	2403	29.0
13	5612	9.5	33	2304	30.0
14	5354	10.5	34	2209	31.0
15	5109	11.5	35	2119	32.0
16	4877	12.5	36	2033	33.0
17	4657	13.5	37	1951	34.5
18	4448	14.5	38	1873	35.5
19	4249	15.5	39	1798	36.5
20	4061	16.5	40	1727	37.5
21	3881	17.5	41	1659	38.5
22	3711	18.5	42	1594	39.5
23	3549	19.5	43	1531	40.5
24	3395	21.0	44	1472	41.5
25	3249	22.0	45	1415	42.5
26	3109	23.0	46	1361	43.5
27	2977	24.0	47	1309	44.5
28	2850	25.0	48	1259	45.5
29	2730	26.0	49	1211	46.5

Sensor #: 406A **Hightemp**

Calibration data

Resistans Ω	Temp °C	Error °C
61596.80	50.026	0.000
40850.50	60.007	-0.000
27224.10	70.413	0.000
19224.30	79.800	-0.000

Thermistor constants
R=A* EXP(B/T+C/T^2)

A=0.12585980E-01
B= 5542.77
C= -182502.18
T=temp. in Kelvin

Bath temp °C	Decade set Ω/10	Circ temp °C	Bath temp °C	Decade set Ω/10	Circ temp °C
50	6166	47.5	70	2765	68.0
51	5912	48.5	71	2663	69.0
52	5669	49.5	72	2564	70.0
53	5438	51.0	73	2469	71.5
54	5217	52.0	74	2379	72.5
55	5006	53.0	75	2292	73.5
56	4805	54.0	76	2209	74.5
57	4613	55.0	77	2129	75.5
58	4429	56.0	78	2052	76.5
59	4254	57.0	79	1979	77.5
60	4086	58.0	80	1909	78.5
61	3926	59.0	81	1841	79.5
62	3773	60.0	82	1776	80.5
63	3627	61.0	83	1714	81.5
64	3487	62.0	84	1654	82.5
65	3353	63.0	85	1597	83.5
66	3225	64.0	86	1541	84.5
67	3103	65.0	87	1488	85.5
68	2985	66.0	88	1437	86.5
69	2873	67.0	89	1388	87.5

FIGURA A.2: Equivalencia de resistencia y temperatura en una calibración de 1994.

Códigos

1. Firmware microcontrolador

```
#include "uart.h"
#include <util/delay.h>
#include <avr/interrupt.h>

#define STOP 0x01
#define START 0x02
#define SET_CHANNEL_0 0x03
#define SET_CHANNEL_1 0x04
#define SET_CHANNEL_2 0x05
#define SET_CHANNEL_3 0x06
#define NAME 0x0f

#define N_MEAN 4096 // 4**6

void setupADC(void);
void setupUART(void);
void setChannel(uint8_t ch);
void sendADC(uint16_t value);

int main(void)
{
    setupADC();
    setupUART();

    uint8_t val;
    uint16_t i;
    uint32_t mean;
```

```
    sei();

    while(1)
    {
        val = uart_getc();// & (0x7f);

        if(val == START)
        {
            mean = 0;
            for(i = 0; i < N_MEAN; i++)
            {
                ADCSRA |= (1 << ADSC); // start conversion
                while(ADCSRA & (1 << ADSC));
                mean += ADC;
            }
            sendADC(mean / 64); // 2**10 * 4**6 / 2**16
        }
        else if(val == NAME)
        {
            uart_puts("Rutherford\n");
        }
        else if((val >= SET_CHANNEL_0) && (val <= SET_CHANNEL_3))
        {
            setChannel(val - SET_CHANNEL_0);
            uart_puts("C\n");
        }
        else
        {
            uart_putc(val);
        }
    }
    return 0;
}

void sendADC(uint16_t value)
{
    uint8_t high, low;
    high = value >> 8;
    low = value;

    uart_putc(high);
    uart_putc(low);
}
```

```
void setChannel(uint8_t ch)
{
    uint8_t mask = ~((1 << MUX1) | (1 << MUX0));
    ADMUX = (ADMUX & mask) | ch;
}

void setupADC(void)
{
    /*setup ADC*/
    ADMUX |= (1 << REFS0); // internal reference
    //~ ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // set
    division factor to 128
    ADCSRA |= (1 << ADPS2) | (1 << ADPS1); // set division factor to 64 //
    default
    //~ ADCSRA |= (1 << ADPS1) | (1 << ADPS0); // set division factor to 8
    ADCSRA |= (1 << ADEN);
}

void setupUART(void)
{
    PORTB &= ~(1 << UART_RX); // set low
    DDRB &= ~(1 << UART_RX); // input

    PORTB |= (1 << UART_TX); // set high
    DDRB |= (1 << UART_TX); // output
}
```

2. Software de Temperatura

2.1. Comunicaciones

```
from time import sleep
from serial import Serial
import serial.tools.list_ports as find_ports

GET_ADC = 0x02
SET_CHANNEL_0 = 0x03
SET_CHANNEL_1 = 0x04
SET_CHANNEL_2 = 0x05
SET_CHANNEL_3 = 0x06
NAME = 0x0f
```

```
SLOPES = [10.257, 10.3, 10.29]
INTERCEPTS = [-12.2, -10.2, -8.4]

BAUDRATE = 115200
TIMEOUT = 0.5

SAMPLING_TIME = 2
ADJUST_TIME = 0.4

SERIAL = None

V_REF = 1.1

LAST_CHANNEL = -1

class NoActiveSerialException(Exception):
    """
        There is no serial port active.
    """
    def __init__(self):
        super(Exception, self).__init__("There is no serial port active.")

class InvalidCalibrationPort(Exception):
    """
        Not a calibration port.
    """
    def __init__(self, port):
        super(Exception, self).__init__("%s is not a calibration port."%port
    )

class ADCException(Exception):
    """
        Critical error. It was not possible to retrieve a valid ADC value.
    """
    def __init__(self):
        super(Exception, self).__init__("Critical error. It was not possible
   to retrieve a valid ADC value.")

def testName(serial):
    serial.write([NAME])
    sleep(0.2)
    ans = serial.readline()
```

```
    try:
        ans = ans.decode()
    except:
        return False
    if "Rutherford" in ans:
        return True
    return False

def findDevices():
    ports = list(find_ports.comports())
    devs = []
    for port in ports:
        port = port.device
        try:
            ser = initPort(port)
            devs.append(port)
            ser.close()
        except Exception as e:
            print(e)
            # pass
    return devs

def initPort(port):
    ser = Serial(port = port, baudrate = BAUDRATE, timeout = TIMEOUT)
    if testName(ser):
        return ser
    else:
        ser.close()
        raise(InvalidCalibrationPort(port))

def setChannel(channel):
    global SERIAL, LAST_CHANNEL
    if (channel <= 3) and (channel >= 0):
        try:
            for i in range(10):
                SERIAL.write([SET_CHANNEL_0 + channel])
                ans = SERIAL.readline()
                try:
                    ans = ans.decode()
                    if "C" in ans:
                        break
                except: pass
                sleep(0.01)
```

```
        except AttributeError:
            raise(NoActiveSerialException())
    else:
        raise(Exception("%d is not a valid channel." %channel))

def getADC():
    for i in range(5):
        SERIAL.write([GET_ADC])
        try:
            high = SERIAL.read()[0]
            # if high <= 3:
            low = SERIAL.read()[0]
            value = (high << 8) | low
            return value
        except IndexError:
            pass
    raise(ADCException())

def getVoltage():
    global V_REF
    v = (V_REF * getADC()) / 0xFFFF
    return v

def getTemperatures():
    global SAMPLING_TIME, ADJUST_TIME
    t = [0]*3
    adjust = SAMPLING_TIME / 3 - ADJUST_TIME
    if adjust < 0: adjust = 0
    for i in range(3):
        setChannel(i + 1)
        val = (1000*getVoltage() - INTERCEPTS[i]) / SLOPES[i]
        t[i] = round(val, 2)
        sleep(adjust)
    return t

def setGlobalSerial(serial):
    global SERIAL
    SERIAL = serial

def isSerialNone():
    global SERIAL
    if SERIAL == None:
        return True
```

```
    return False

def close():
    global SERIAL
    try:
        SERIAL.close()
        SERIAL = None
    except: pass

if __name__ == '__main__':
    devs = findDevices()
    print(devs)
    if len(devs) == 1:
        SERIAL = initPort(devs[0])

        while True:
            try:
                text = []
                for i in range(4):
                    setChannel(i)
                    val = getADC()
                    text.append("C%d: %d"%(i, val))
                text = "\t".join(text)
                print(text)

                sleep(5e-2)
            except KeyboardInterrupt:
                break

        SERIAL.close()
```

2.2. Interfaz gráfica

```
import sys
import time

import __images__
from com import initPort, findDevices, setChannel, getTemperatures,
    setGlobalSerial, close, isSerialNone

import pyqtgraph as pg
try:
    from PyQt5 import QtCore, QtGui
```

```
    from PyQt5.QtWidgets import QLabel, QWidget, QMainWindow, QHBoxLayout,\
            QGroupBox, QFormLayout, QSystemTrayIcon, QApplication, QMenu, \
            QStyleFactory, QMessageBox
except ImportError:
    from PyQt4 import QtCore, QtGui
    from PyQt4.QtGui import QLabel, QWidget, QMainWindow, QHBoxLayout, \
            QGroupBox, QFormLayout, QSystemTrayIcon, QApplication, QMenu,\
            QStyleFactory, QMessageBox

import datetime
import numpy as np

START_TIME = 0
KERNEL_FILTER = 3

MAX_HOLDER = 45e3

# https://gist.github.com/iverasp/9349dffa42aeffb32e48a0868edfa32d

class TimeAxisItem(pg.AxisItem):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.setLabel(text='Time', units=None)
        self.enableAutoSIPrefix(False)

    def tickStrings(self, values, scale, spacing):
        return [datetime.datetime.fromtimestamp(value).strftime("%d/%H:%M")
    for value in values]

class RingBuffer(object):
    def __init__(self, size_max):
        self.max = int(size_max)
        self.data = []

    class __Full:
        """ class that implements a full buffer """
        def append(self, x):
            """ Append an element overwriting the oldest one. """
            self.data[self.cur] = x
            self.cur = (self.cur+1) % self.max

        def get(self):
            """ return list of elements in correct order """
```

```
            return self.data[self.cur:]+self.data[:self.cur]

    def append(self, x):
        """append an element at the end of the buffer"""
        self.data.append(x)
        if len(self.data) == self.max:
            self.cur = 0
            self.__class__ = self.__Full # Permanently change self's class
    from non-full to full

    def get(self):
        """ Return a list of elements from the oldest to the newest. """
        return self.data

class DataHolder(object):
    def __init__(self):
        self.x = RingBuffer(MAX_HOLDER)
        self.y = [RingBuffer(MAX_HOLDER), RingBuffer(MAX_HOLDER), RingBuffer
    (MAX_HOLDER)]

    def timestamp(self):
        return time.mktime(datetime.datetime.now().timetuple())

    def addValues(self, temperatures):
        now = time.localtime()
        self.x.append(self.timestamp())
        for i in range(3):
            y = self.y[i]
            y.append(temperatures[i])
            if (len(y.get()) > KERNEL_FILTER) and (KERNEL_FILTER > 0):
                temp = sorted(y.get()[-KERNEL_FILTER:])[(KERNEL_FILTER - 1)
    // 2]
                y.get()[-(KERNEL_FILTER - 1)] = temp

        self.save(now)

    def save(self, now):
        with open("TemperatureData.txt", "a") as file:
            now = time.strftime("%Y-%m-%d %H:%M:%S", now)
            data = ["%.2f"%self.getY(i)[-1] for i in range(3)]
            line = [now] + data
            file.write("\t".join(line) + "\r\n")
```

```
    def getX(self):
        return self.x.get()

    def getY(self, i):
        return self.y[i].get()

    def clear(self):
        self.x = RingBuffer(MAX_HOLDER)
        self.y = [RingBuffer(MAX_HOLDER), RingBuffer(MAX_HOLDER), RingBuffer
    (MAX_HOLDER)]

    def __len__(self):
        return len(self.x.get())

class FindDevicesThread(QtCore.QThread):
    def __init__(self):
        super(QtCore.QThread, self).__init__()

    def run(self):
        while True:
            if isSerialNone():
                devs = findDevices()
                if len(devs) == 1:
                    try:
                        setGlobalSerial(initPort(devs[0]))
                    except:
                        setGlobalSerial(None)
                else:
                    setGlobalSerial(None)
            else:
                time.sleep(10)

class RequestDataThread(QtCore.QThread):
    def __init__(self, holder):
        super(QtCore.QThread, self).__init__()
        self.holder = holder
        self.exception = None

    def run(self):
        global START_TIME
        while True:
            if not isSerialNone():
                try:
```

```
                    self.holder.addValues(getTemperatures())
                except Exception as e:
                    self.stop()
                    self.exception = e
            else:
                time.sleep(5)

    def stop(self):
        close()

class MainWindow(QMainWindow):
    SAMPLING_DEFAULT = 1 # seconds
    MINIMUM_PLOT_UPDATE = 2000

    def __init__(self):
        super(QMainWindow, self).__init__()
        self.setWindowTitle("Lector de temperatura")
        widget = QWidget()
        self.setCentralWidget(widget)

        self.main_layout = QHBoxLayout(widget)
        self.main_layout.setContentsMargins(11, 11, 11, 11)
        self.main_layout.setSpacing(6)

        self.settings_frame = QGroupBox()
        self.settings_layout = QFormLayout(self.settings_frame)

        self.label_0 = QLabel("00.00")
        self.label_1 = QLabel("00.00")
        self.label_2 = QLabel("00.00")
        self.settings_layout.addRow(self.label_0, QLabel("\tInterno (C)"))
        self.settings_layout.addRow(self.label_1, QLabel("\tExterno (C)"))
        self.settings_layout.addRow(self.label_2, QLabel("\tAmbiente (C)"))

        self.main_layout.addWidget(self.settings_frame)
        ### pyqtgraph
        pg.setConfigOptions(leftButtonPan = False, foreground = 'k',
    background = None)
        # pg.setConfigOptions(, antialias = True)
        self.temperature_plot = pg.PlotWidget(
            labels={'left': 'Temperatura (C)', 'bottom': 'Hora'},
            axisItems={'bottom': TimeAxisItem(orientation='bottom')}
            )
```

```
        self.temperature_plot.addLegend()
        self.main_layout.addWidget(self.temperature_plot)

        symbol = None #
        symbolSize = 3
        self.data0_line = self.temperature_plot.plot(pen = "b", symbol =
symbol, symbolPen = "b", symbolBrush="b", symbolSize=symbolSize, name="
Interno")
        self.data1_line = self.temperature_plot.plot(pen = "m", symbol =
symbol, symbolPen = "m", symbolBrush="m", symbolSize=symbolSize, name="
Externo")
        self.data2_line = self.temperature_plot.plot(pen = "g", symbol =
symbol, symbolPen = "g", symbolBrush="g", symbolSize=symbolSize, name="
Ambiente")

        #### signals
        self.update_plots_timer = QtCore.QTimer()
        self.update_plots_timer.setInterval(self.MINIMUM_PLOT_UPDATE)
        self.update_plots_timer.timeout.connect(self.updatePlots)

        self.find_thread = FindDevicesThread()
        self.find_thread.start()

        self.data = DataHolder()
        self.data_thread = RequestDataThread(self.data)
        self.data_thread.start()
        self.update_plots_timer.start()

    def updatePlots(self):
        if self.data_thread.exception != None:
            self.errorWindow(self.data_thread.exception)
        else:
            try:
                x = np.array(self.data.getX())
                for i in range(3):
                    label = getattr(self, "label_%d" %i)
                    plot = getattr(self, "data%d_line" %i)
                    data = np.array(self.data.getY(i))
                    plot.setData(x, data)
                    label.setText("%.2f"%data[-1])
            except Exception as e:
                print(e)
```

```
    def deviceConnection(self):
        if self.find_thread.success:
            self.deviceExists()
        else:
            self.noDevice()

    def noDevice(self):
        self.data_thread.stop()
        self.update_plots_timer.stop()

    def warning(self, text):
        msg = QMessageBox()
        msg.setIcon(QMessageBox.Warning)
        msg.setText('Warning.\n%s' % text)
        msg.setWindowTitle("Warning")
        msg.setStandardButtons(QMessageBox.Ok)
        msg.exec_()

    def errorWindow(self, exception):
        self.noDevice()
        text = str(exception)
        msg = QMessageBox()
        msg.setIcon(QMessageBox.Warning)
        msg.setText('An error ocurred.\n%s' % text)
        msg.setWindowTitle("Error")
        msg.setStandardButtons(QMessageBox.Ok)
        msg.exec_()

    def closeEvent(self, event):
        event.ignore()
        self.hide()

class SystemTrayIcon(QSystemTrayIcon):
    def __init__(self, icon, parent = None):
        QSystemTrayIcon.__init__(self, icon, parent)
        menu = QMenu(parent)
        openAction = menu.addAction("Open")
        exitAction = menu.addAction("Exit")
        self.setContextMenu(menu)

        openAction.triggered.connect(self.openMain)
        exitAction.triggered.connect(self.exit)
        self.activated.connect(self.systemIcon)
```

```
        self.main_window = MainWindow()
        self.openMain()

    def exit(self):
        QtCore.QCoreApplication.exit()

    def systemIcon(self, reason):
        if reason == QSystemTrayIcon.Trigger:
            self.openMain()

    def openMain(self):
        self.main_window.show()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    QApplication.setStyle(QStyleFactory.create('Fusion'))
    app.setWindowIcon(icon)
    app.processEvents()

    w = QWidget()
    trayIcon = SystemTrayIcon(icon, w)

    trayIcon.show()

    app.exec_()

    close()
```

3. Determinación de los valores de una calibración estática

```
import numpy as np
from scipy.signal import medfilt

def getStaticData(name, from_ = 0, to_ = -1):
    sCal0 = np.genfromtxt(name, skip_header = 2)
    sCal0 = sCal0[from_:to_, :2] - sCal0[0, 0]
    return sCal0.T

def getIntervals(p, threshold = 0.5):
```

```
    d = np.diff(p)
    ad = medfilt(abs(d), kernel_size = 101)

    filtered = ad.copy()
    up = ad >= threshold
    low = ad < threshold
    filtered[up] = 1
    filtered[low] = 0

    changes = np.diff(filtered)
    ones = np.where(changes == 1)[0]
    negative = np.where(changes == -1)[0]

    t = 0.7
    baseline = (0, ones[0])
    top = int(t * (ones[1] - negative[0])) + negative[0], ones[1]
    baseline2 = int(t * (len(p) - negative[1])) + negative[1], len(p)

    return baseline, top, baseline2

def getStatistics(p, intervals):
    data = [p[i[0] : i[1]] for i in intervals]
    return [(np.mean(d), np.std(d)) for d in data]
```