

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

1. Control de la temperatura	1
Referencias	3
A. Anexos	5
1. Firmware microcontrolador	5
2. Software de Temperatura	8
2.1. Comunicaciones	8
2.2. Interfaz gráfica	12
3. Determinación de los valores de una calibración estática	19

Índice de figuras

1.1. Resistencias de década que controlan la temperatura.	2
A.1. Control térmico del calorímetro. Modificado de [1].	6
A.2. Equivalencia de resistencia y temperatura en una calibración de 1994 . .	21

Índice de tablas

Control de la temperatura

El sistema cuenta con dos calentadores de precisión al interior del baño termostata- do 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 ac- tivos los calentadores internos en algún nivel de potencia intermedio, pues el baño ex- terno, al tener una temperatura inferior agregar una carga permanente a los calen- tadores 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 termis- tores 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, es- to es que cada resistencia es una potencia de 10 menor que la anterior, las cuales se encuentran en la parte inferior del calorímetro ([Figura 1.1](#)), 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](#).

Para determinar la equivalencia de resistencia y temperatura el calorímetro cuenta con una tabla que relaciona los valores de resistencia y temperatura de baño ex- terno que dan lugar a una temperatura constante en el baño interno. Dicha tabla se muestra en el [Apéndice A](#) 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



FIGURA 1.1: Resistencias de década que controlan la temperatura.

cuales por convención se reportan a estas condiciones.

De todas las acciones que se hicieron a lo largo del proyecto, la estabilización de la temperatura es la que llevó más tiempo. En gran medida esto fue debido al tiempo de estabilización del equipo, el cual puede tomar cerca de 6 horas en completarse.

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.

Anexos

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;
```

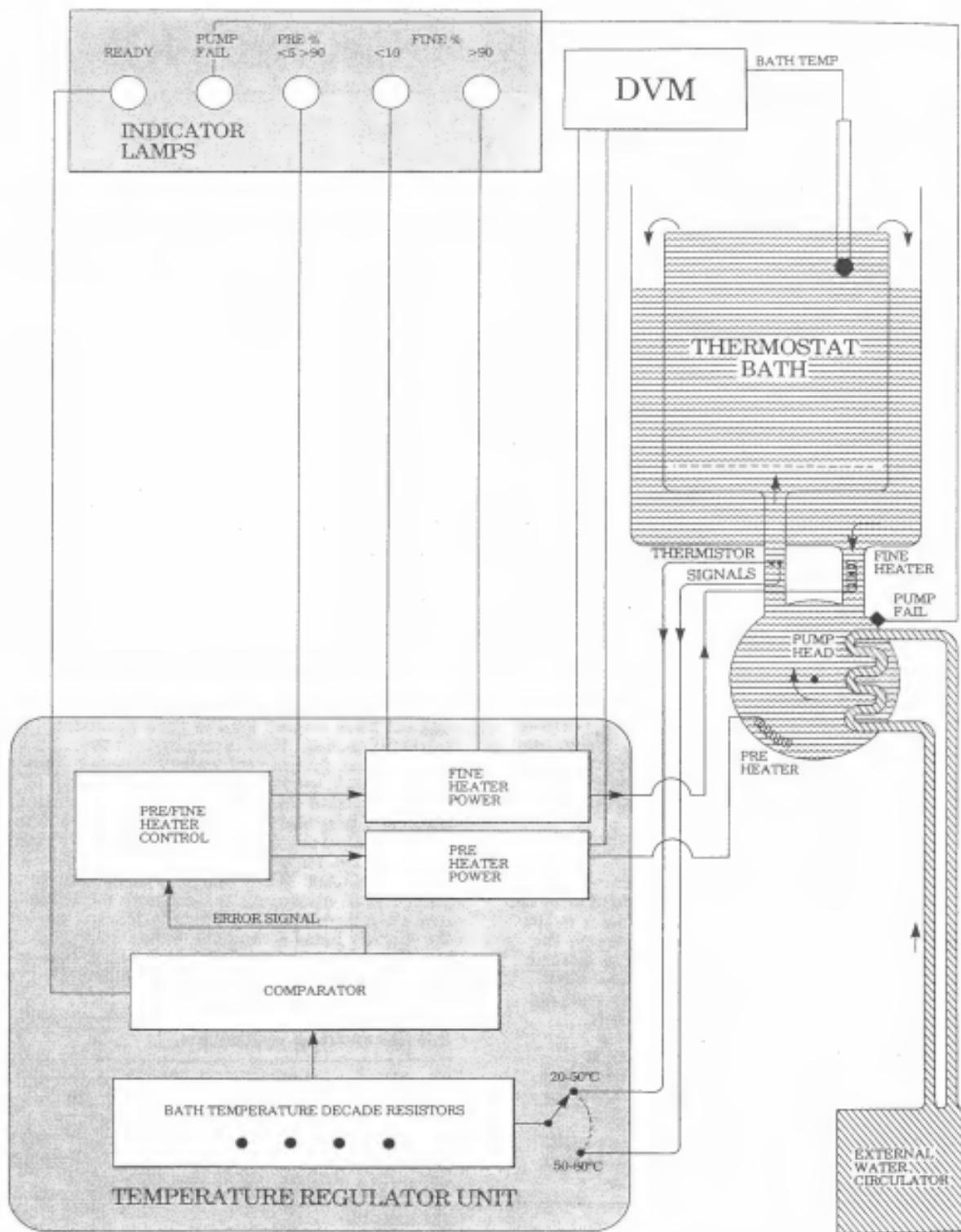


FIGURA A.1: Control térmico del calorímetro. Modificado de [1].

```
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, QBoxLayout,\n    QGroupBox, QFormLayout, QSystemTrayIcon, QApplication, QMenu,\n    QStyleFactory, QMessageBox\nexcept ImportError:\n    from PyQt4 import QtCore, QtGui\n    from PyQt4.QtGui import QLabel, QWidget, QMainWindow, QBoxLayout,\n        QGroupBox, QFormLayout, QSystemTrayIcon, QApplication, QMenu,\n        QStyleFactory, QMessageBox\n\nimport datetime\nimport numpy as np\n\nSTART_TIME = 0\nKERNEL_FILTER = 3\n\nMAX HOLDER = 45e3\n\n# https://gist.github.com/iverasp/9349dff42aeffb32e48a0868edfa32d\n\nclass TimeAxisItem(pg.AxisItem):\n    def __init__(self, *args, **kwargs):\n        super().__init__(*args, **kwargs)\n        self.setLabel(text='Time', units=None)\n        self.enableAutoSIPrefix(False)\n\n    def tickStrings(self, values, scale, spacing):\n        return [datetime.datetime.fromtimestamp(value).strftime("%d/%H:%M")\nfor value in values]\n\n\nclass RingBuffer(object):\n    def __init__(self, size_max):\n        self.max = int(size_max)\n        self.data = []\n\n    class __Full:\n        """ class that implements a full buffer """\n        def append(self, x):\n            """ Append an element overwriting the oldest one. """\n            self.data[self.cur] = x\n            self.cur = (self.cur+1) % self.max\n\n        def get(self):\n            """ 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 = QBoxLayout(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_occurred.\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]
```

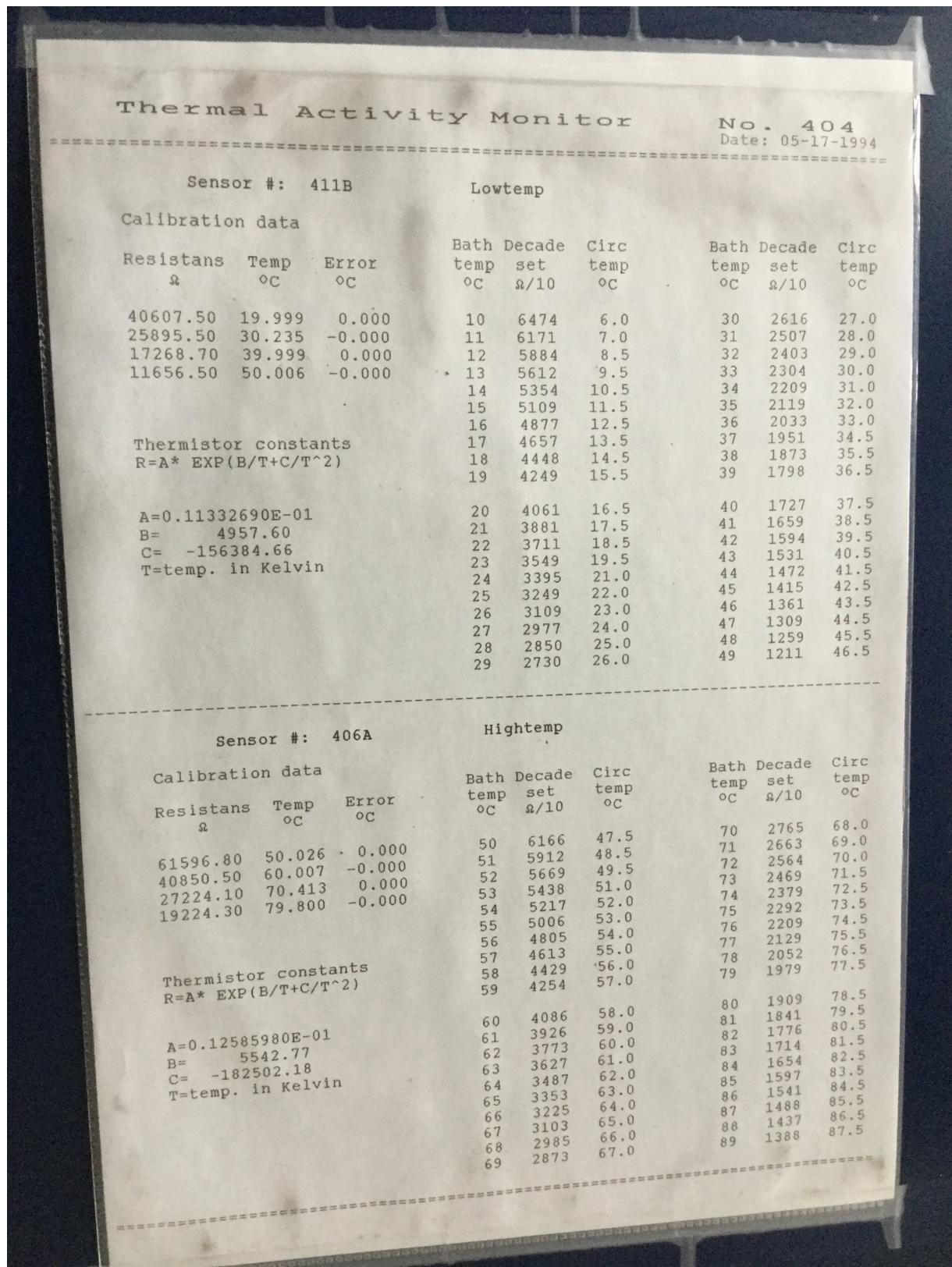


FIGURA A.2: Equivalencia de resistencia y temperatura en una calibración de 1994.