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

27 de noviembre de 2018

Índice general

Índice de figuras

Introducción

La termodinámica es el estudio de las transformaciones de energía, en la etapa más temprana de esta rama de las ciencias, se pensaba que el calor era una clase de fluido cuya cantidad neta en el universo permanecía siempre constante. El aumento de temperatura de un objeto era explicado a partir de la migración de calor de un objeto a otro [1, 2]. Usando esta teoría del calor como fluido, el ingeniero Sadi Carnot dio origen a la termodinámica con el análisis del problema, sobre cómo generar el mejor y más eficiente motor de vapor en 1824. Posteriormente, Julius von Mayer en 1842 descubrió una equivalencia entre el calor y el trabajo mecánico, trabajo también publicado por James Joule el siguiente año [2]. A raíz de esto hoy se considera el calor como una forma de transferencia de energía [2].

Los resultados de la termodinámica se encuentran implícitos en una serie de declaraciones que reciben el nombre de leyes de la termodinámica, las cuales en conjunto, nos permiten conocer la dirección natural en la que tienen lugar los cambios químicos y físicos en la materia [3]. Históricamente la termodinámica fue desarrollada antes que se tuviera un entendimiento sobre la estructura interna de la materia, además, sus leyes tampoco siguen un orden cronológico [1]. La primera ley surge del trabajo realizado por Mayer en 1842, una de las leyes más importantes de la ciencia en general: la conservación de la energía [1, 2]. La segunda se atribuye a Carnot y Clausius en 1824, y habla sobre la reversibilidad o direccionalidad de los procesos [1].

A pesar que el estudio de las transformaciones de energía parece un tema distante de la química, la termodinámica, estudiada a través de la fisicoquímica, ha demostrado ser de vital importancia tanto en la química como la biología [3]. No sólo permite entender la producción o consumo de energía en las reacciones químicas, además constituye una herramienta fundamental para responder preguntas que se encuentran en el corazón de la bioquímica, por ejemplo sobre cómo fluye la energía en una célula, y qué tan grande puede ser la agrupación de moléculas que forman estructuras complejas como

las células [3]. En otros casos, por ejemplo, existen propiedades de fácil determinación para un analista, como el color, masa y densidad. Sin embargo, hay propiedades que dependen de los enlaces, estructura molecular y naturaleza del material, entre estas se encuentran propiedades termodinámicas de interés químico: capacidad calorífica, entalpía, entropía, etc [4].

Dentro de la fisicoquímica, el estudio y medición de la transferencia de energía en forma de calor se denomina calorimetría, y constituye una de las áreas más viejas de esta rama de la química [5]. Se podría considerar que la historia de esta comienza en junio de 1783, con la presentación de *Memoria del calor* (Mémoire de la Chaleur) por Lavoisier y Laplace a la Academia Francesa [5]. La mayoría de los procesos físicos y químicos están acompañados de absorción o liberación de energía. Lo anterior hace de la calorimetría una técnica con un amplio rango de aplicaciones [6]. Entre estas aplicaciones se tienen titulaciones, flujos, reacciones y estudio de procesos de sorción [4].

El instrumento para realizar este tipo de mediciones se denomina calorímetro, y existen de diversos tipos. Pueden ser clasificados por el tipo de condiciones que imponen al sistema, por ejemplo, volumen constante, temperatura constante, calor constante, etc. Estos calorímetros reciben el nombre de: bombas calorimétricas, isotérmicos, y adiabáticos, correspondientemente [4, 6]. También pueden ser clasificados por el principio de funcionamiento, si compensan los flujos de energía o los acumulan [4]. Finalmente, dependiendo del rango de las potencias medidas, se tienen dos términos comúnmente usados: *microcalorimetría* para el caso de experimentos realizados en el rango de los microvatios [6, 7], mientras que para escalas de nanovatios son usados *nanocalorímetros* [7].

En particular el trabajo a realizar busca armar, poner en marcha y calibrar un microcalorímetro 2277 Thermal Activity Monitor. Esto implica un entendimiento del funcionamiento del calorímetro a nivel interno, desde las partes mecánicas, la etapa electrónica y los principios fisicoquímicos detrás del mismo. En este proyecto, por su naturaleza, se hace imperativo que una vez alcanzado el nivel de operación del calorímetro, se realice una calibración para constatar que el funcionamiento es correcto y concuerda con los resultados obtenidos por la literatura así como con un microcalorímetro moderno disponible en el grupo de investigación *Termodinámica de Soluciones*.

1. Objetivos

1.1. Objetivo general

Poner en funcionamiento el calorímetro 2277 Thermal Activity Monitor con el que se cuenta, en funcionamiento y adicionalmente calibrar el equipo para su uso en las investigaciones activas del grupo Termodinámica de Soluciones.

1.2. Objetivos específicos

- Ensamblar el equipo 2277 Thermal Activity Monitor.
- Realizar el cableado y conexiones electrónicas pertinentes al mismo.
- Calibración eléctrica, determinación de las señales de entrada y salida, flujo de las bombas hidráulicas y temperatura del baño.
- Calibración química, determinación de la entalpía molar, energía libre de Gibbs, entropía, y constante de equilibrio, del acomplejamiento del catión bario con éter 18-corona-6.

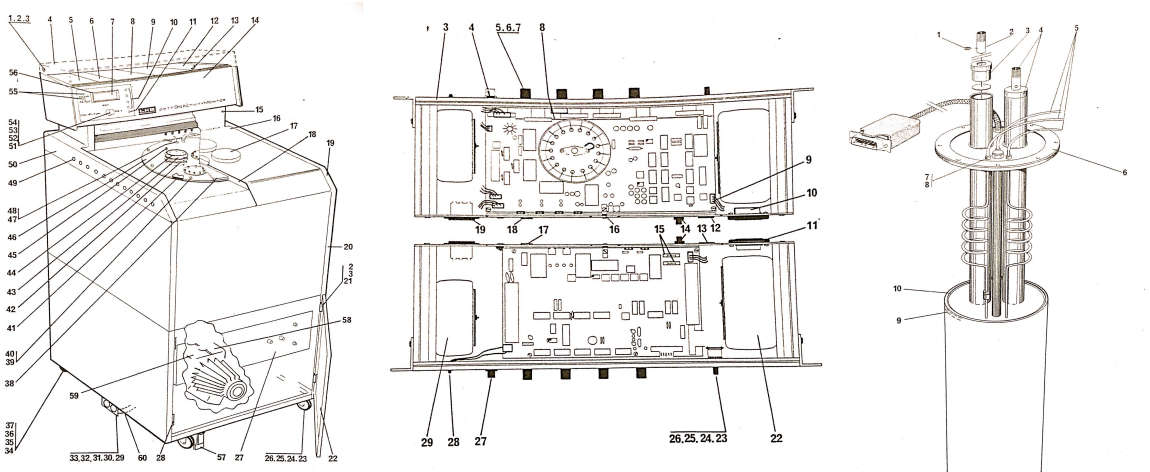
2. Justificación del proyecto

En general la calorimetría permite una gran variedad de análisis, muchos de ellos cuentan con aplicaciones industriales, comerciales, biológicas y químicas, permitiendo el entendimiento de las interacciones moleculares en soluciones [8]. Una ventaja de la calorimetría es que no es específica, ni invasiva además de no depender de las propiedades electroquímicas y ópticas de un sistema dado, siendo esto de vital importancia para las investigaciones de procesos biológicos, el estudio del crecimiento bacteriano y para la detección de compuestos biológicos [9]. Por otro lado la calorimetría también permite el estudio de la termodinámica en sistemas de absorción, bien sea por la universalidad de la absorción física en las superficies como catalizadores, o en procesos industriales como la separación de mezclas de gases [10]. Finalmente la calorimetría es el método clásico para la determinación de propiedades termodinámicas en las muestras, entre estas se encuentran la capacidad calorífica, la entalpía, entropía y energía libre de Gibbs, las cuales constituyen el punto de partida de gran cantidad de estudios teóricos, desarrollos y producción industrial de un compuesto químico [4, 11].

El calorímetro 2277 Thermal Activity Monitor con el que cuenta el grupo de investigación se encuentra desarmado, a la espera de su ensamble y puesta en funcionamiento. Este instrumento permite monitorear una gran variedad de reacciones químicas y bioquímicas, lo anterior debido a su capacidad de cuantificar procesos exotérmicos y endotérmicos. Estas reacciones pueden ser estudiadas en el rango de 5 - 80 °C [12]. Este rango de temperaturas se debe al uso de un baño termostatado de 25 litros. Cuatro balones de medición independientes se encuentran sumergidos en este baño, permitiendo medidas con desviaciones de temperatura inferiores a $\pm 2 \times 10^{-4}$ °C, alcanzando de esta manera medidas en el rango de microvatios [12]. Es por esta razón que poner en marcha y calibrar el equipo con el que cuenta el grupo de investigación resulta una contribución importante para el mismo, así como para el Departamento de Química en general.

3. Metodología

Para el ensamble del calorímetro se cuenta con el catálogo de partes, y el manual de instrucciones. En el primero se detallan y enumeran las partes del equipo, en la Figura 1.1 se muestran ejemplos de los distintos sistemas con los que el calorímetro cuenta [12].



(A) Vista exterior del equipo con sus componentes.

(B) Parte del sistema eléctrico del calorímetro.

(C) Celda de medición.

FIGURA 1.1: Componentes exteriores del equipo y partes de los sistemas del calorímetro [12].

Usando esta información se llevará a cabo el ensamble del calorímetro así como las conexiones eléctricas. A nivel electrónico, la liberación o absorción de energía térmica se mide usando una celda Peltier, las cuales tienen una respuesta proporcional en voltaje, de esta forma se obtiene una señal eléctrica a partir de pequeñas variaciones de temperatura en la celda de reacción. La señal eléctrica es procesada por el sistema digital del instrumento [12]. De esta forma y siguiendo las instrucciones es posible realizar una calibración eléctrica del sistema.

La calibración química puede realizarse de distintas maneras. En principio cualquier reacción que pueda ser llevada a cabo en un calorímetro, con condiciones controladas y cuyas propiedades termodinámicas sean bien conocidas, puede ser usada como una reacción de calibración [6]. Sin embargo es recomendable que los reactivos involucrados en estas reacciones no requieran de algún tipo de purificación que pueda alterar el análisis [6].

Se propone realizar una calorimetría de titulación como método de calibración química, lo anterior es importante porque una calibración eléctrica puede dar lugar a distribuciones de temperatura distintas en la celda de reacción, o generar patrones de flujo de calor distintos a una reacción química, esto puede generar errores sistemáticos en las medidas realizadas [6]. La titulación es una técnica importante en la medida que permite la determinación simultánea de la entalpía molar y la constante de equilibrio de la reacción, por lo cual también se obtienen el cambio en la energía estándar de Gibbs y el

cambio en la entropía estándar [6].

Para esta calibración es necesario contar con soluciones acuosas de cloruro de bario ($BaCl_2$) y éter 18-corona-6 ($[C_2H_4O]_6$) [6, 13, 14]. La reacción de acomplejamiento es 1:1, de la siguiente forma:

$$Ba^{2+}(ac) + [C_2H_4O]_6(ac) \longrightarrow Ba^{2+} \cdot [C_2H_4O]_6(ac) \tag{1.1}$$

Dichas soluciones deben encontrarse en el rango de concentraciones de 1 mM a 10 mM para el éter y 10 mM hasta 100 mM para el cloruro de bario. Dentro de este rango, existe evidencia experimental que muestra que no hay una variación significativa de las cantidades termodinámicas medidas [6, 14]. Considerando el volumen de las celdas con las que cuenta el equipo, se propone usar 1.5 mL de la solución de éter. Las soluciones deben ser desgasificadas y termostatadas a 25 °C, con el objetivo de limitar el error experimental introducido por diferencias de temperatura causadas por burbujas [13-15].

En el calorímetro el baño debe encontrarse a 25 °C y deben usarse dos de las cuatro celdas disponibles. La primera se considera como referencia y debe contener al disolvente únicamente, en este caso 1.5 mL de agua [12]. En la segunda se adiciona el mismo volumen de la solución de éter, una vez la línea base del calorímetro se encuentra estable, se adicionan una a una 30 alícuotas de la solución de cloruro de bario, con tiempos entre inyección mayores a $\tau = 6$ minutos, tanto a la celda de referencia como a la de reacción [14, 15]. Cada inyección modifica la temperatura de la celda respecto a la referencia, razón por la cual las termopilas Peltier, deberán compensarla, la energía requerida por las celdas se cuantifica en función del tiempo, esto da lugar a una gráfica de potencia en función del tiempo [15]. Para cada inyección, es posible calcular el cambio en la energía integrando la potencia usada por el sistema. La gráfica de los cambios energéticos en función de la fracción molar ($[Ba^{2+}]/[\text{éter}]$) permite obtener las propiedades termodinámicas [14, 15]. La diferencia entre el valor mínimo de la isotérma y la línea base de esta corresponde con la entalpía molar estándar ΔH_m. La derivada de la isotérma en el punto equimolar, da lugar a la constante de afinidad K_a. Finalmente con esta es posible determinar los cambios en la energía libre de Gibbs y la entropía.

$$\Delta G_m = -RT \ln K_a = \Delta H_m - T\Delta S_m \tag{1.2}$$

Para determinar la capacidad calorífica es necesario realizar el mismo experimento a distintas temperaturas, dado que:

$$C_{p,m} = \left(\frac{\partial H_m}{\partial T}\right)_p \tag{1.3}$$

En el caso de esta titulación los valores reportados por IUPAC corresponden con [6]:

- $\Delta H_m = -(31{,}42 \pm 0{,}20)$ kJ mol^{-1}
- $K_a = (5{,}90 \pm 0{,}20) \times 10^3$ mM
- $C_{p,m} = 126$ J K^{-1} mol^{-1}

Calibración Química

La calibración química consiste en contrastar las propiedades termodinámicas de un sistema químico, obtenidas usando el calorímetro con aquellas reportadas en la literatura. Para esto se estudia la reacción del ácido clorhidrico con bicarbonato de potasio. Esta reacción presenta varias ventajas: por un lado hace uso de reactivos de fácil acceso, es una reacción con esqueometría 1:1, ha sido estudiada previamente, y además, es usada en calibraciones de equipos calorimétricos., como el calorímetro de titulación NanoITC de *TA instruments*.

1. Preparación de las soluciones de HCl

Para la preparación de las soluciones, se hace uso de forma sistemática de una balanza XXXX y un densímetro YYYY. Además, en el proceso de preparación se hace necesaria la toma de dos alícuotas de 30 μL y 170 μL, para el HCl y $KHCO_3$ correspondientemente, por lo cual se usa una micropipeta ZZZZ con rango de 20 a 200 μL.

1.1. Solución de HCl 0.25 mM

Para determinar la concentración de una solución de 1.0 mL de HCl concentrado en 25.0 mL de agua tipo 1, se midió la densidad de la solución, posteriormente usando como referencia los datos reportados en la literatura fue realizada una regresión lineal que permitió relacionar la concentración con la densidad de la solución ρ [16]. De esta manera se estableció el valor de la concentración en: 3,02 $\pm$ 0,05 % (fracción de masa $[w_t]$), donde la incertidumbre se obtiene de la pendiente (m) e intercepto (b) de la regresión lineal que se muestra en la Figura 2.1.

$$\delta[w_t] = \sqrt{\left(\frac{\rho - b}{m^2}\delta m\right)^2 + \left(\frac{\delta b}{m}\right)^2} \tag{2.1}$$

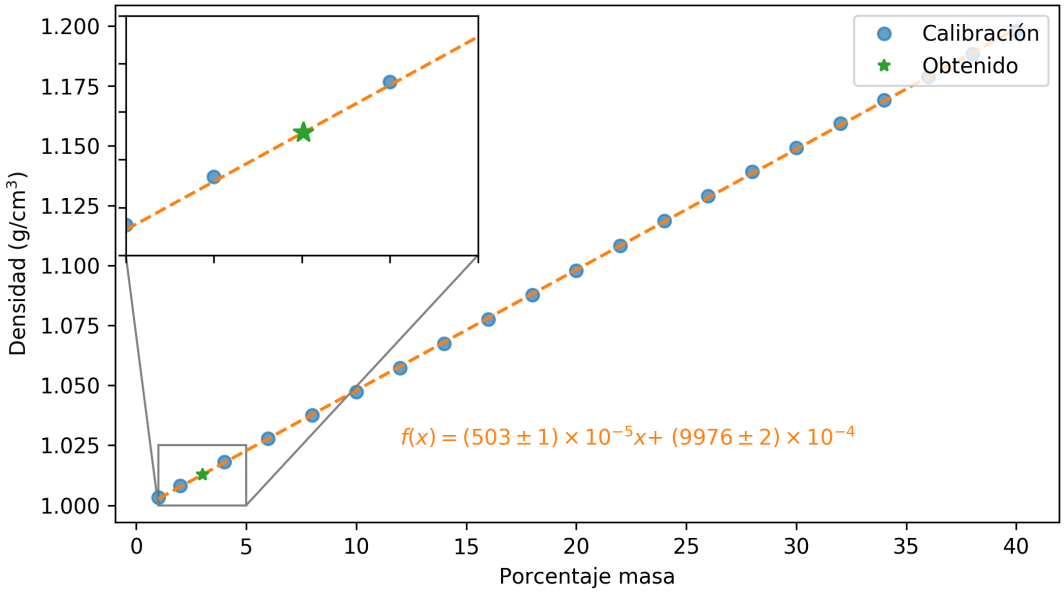


FIGURA 2.1: Determinación de la concentración de una solución de HCl usando valores de densidad reportados en la literatura [16].

Para obtener la concentración 0,84 ± 0,04 M, se usa la siguiente ecuación, la cual relaciona la concentración en fracción de masa con la molaridad $[M]$:

$$[M] = 10\frac{[w_t]\rho}{m_m} \qquad m_m \text{ la masa molecular del HCl} \tag{2.2}$$

Se tiene entonces que la incertidumbre en la concentración estará dada por:

$$\delta[M] = [M]\sqrt{\delta[w_t]^2 + \delta\rho^2} \tag{2.3}$$

Una alícuota de 0.0297 g de esta solución fue disuelta en 99.5553 g de agua, dando lugar a una solución 0.2469 mM. Donde la concentración final se calcula a partir de las densidades de las soluciones inicial (ρ_s) y final (ρ_f), las masas de agua (m_{H_2O}) y de solución inicial (m_s) usando la siguiente ecuación:

$$[M]_f = [M]_i\frac{V_s}{V_f} = [M_i]\frac{m_s/\rho_s}{(m_s + m_{H_2O})/\rho_f} = [M]_i\left(\frac{m_s}{m_s + m_{H_2O}}\right)\left(\frac{\rho_s}{\rho_f}\right) \tag{2.4}$$

	$[\mathbf{M}]_\mathbf{i}$ (M)	$\mathbf{m_s}$ (g)	$\mathbf{m}_{H2O}$ (g)	$\mathbf{æ_s}$ (g/cm^3)	$\mathbf{æ_f}$ (g/cm^3)	$[\mathbf{M}]_\mathbf{f}$ (mM)
HCl	$0{,}84 \pm 0{,}04$	0,0297	99,5553	1,012832	0,998205	0,24690
$KHCO_3$	0,1022577	0,1709	99,5657	1,003662	0,998215	0,1742689

TABLA 2.1: Densidades y masas medidas para alcanzar las soluciones con concentraciones 0,25 mM y 0,17 mM para el HCl y $KHCO_3$.

1.2. Solución de $KHCO_3$ 0.17 mM

En un balón aforado de 10 mL fueron adicionados 0.10143 g de $KHCO_3$, junto con 9.93551 g de H_2O. La densidad fue medida a 25 °C y su valor fue 1.003662 g/cm^3. La concentración de esta solución se calculó usando la siguiente ecuación:

$$[M] = \frac{n}{V} = \frac{m\rho}{m_m(m + m_{H_2O})} \tag{2.5}$$

Obteniendo un valor de $0{,}10100 \pm 0{,}00001$ M, donde la incertidumbre se calcula usando:

$$\delta[M] = \sqrt{\frac{\delta\rho^2 m^2}{m_m^2\left(m + m_{H_2O}\right)^2} + \frac{\delta m^2 \rho^2 m_{H_2O}^2}{m_m^2\left(m + m_{H_2O}\right)^4} + \frac{\delta {m_{H_2O}}^2 \rho^2 m^2}{m_m^2\left(m + m_{H_2O}\right)^4}} \tag{2.6}$$

2. Sistema de inyección

2.1. Calibración de la jeringa

2.2. Control por software

3. Realización del experimento

4. Resultados

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;
```

```
    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()
```

Referencias

(1) Feynman, R. P.; Leighton, R. B. y Sands, M., *The Feynman lectures on physics, Vol. I: The new millennium edition: mainly mechanics, radiation, and heat*; Basic books: 2011; vol. 1.

(2) Fermi, E., *Notes on thermodynamics and statistics*; University of Chicago Press: 1986.

(3) Atkins, P. y De Paula, J., *Physical chemistry for the life sciences*; Oxford University Press, USA: 2011.

(4) Gaisford, S.; Kett, V. y Haines, P., *Principles of thermal analysis and calorimetry*; Royal society of chemistry: 2016.

(5) Zielenkiewicz, W. y Margas, E., *Theory of calorimetry*; Springer Science & Business Media: 2006; vol. 2.

(6) Wadsö, I. y Goldberg, R. N. *Pure and Applied Chemistry* **2001**, *73*, 1625-1639.

(7) Wadsö, I. y Wadsö, L. *Thermochimica acta* **2003**, *405*, 15-20.

(8) Blandamer, M. J.; Briggs, B.; Cullis, P. M.; Irlam, K. D.; Engberts, J. B. y Kevelam, J. *Journal of the Chemical Society, Faraday Transactions* **1998**, *94*, 259-266.

(9) Winkelmann, M; Hüttl, R y Wolf, G *Thermochimica acta* **2004**, *415*, 75-82.

(10) Morrison, J. *Pure and applied chemistry* **1987**, *59*, 7-14.

(11) Wang, M.-H.; Tan, Z.-C.; Sun, X.-H.; Zhang, H.-T.; Liu, B.-P.; Sun, L.-X. y Zhang, T. *Journal of Chemical and Engineering Data* **2005**, *50*, 270-273.

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

(13) Tellinghuisen, J. *The Journal of Physical Chemistry B* **2007**, *111*, 11531-11537.

(14) Mizoue, L. S. y Tellinghuisen, J. *Biophysical chemistry* **2004**, *110*, 15-24.

(15) Duff, M. R. y col. *Journal of visualized experiments: JoVE* **2011**.

(16) Perry, R. H.; Green, D. W.; Maloney, J. O. y col. Perry's chemical engineers' handbook., 200722.