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

30 de noviembre de 2018

Índice general

Índice de figuras

Índice de tablas

Calibración Eléctrica

Con el objetivo de asegurar que la información registrada por el calorímetro corresponde con un valor específico de potencia, es necesario realizar una calibración eléctrica, la cual es específica para cada canal de medición. Para ello, el equipo cuenta con una resistencia de precisión que envuelve el contenedor de la celda y permite simular, lo mejor posible, la energía liberada en forma de calor cuando una reacción química tiene lugar en la celda, la ubicación de esta resistencia se muestra en la Figura 1.1 [1].

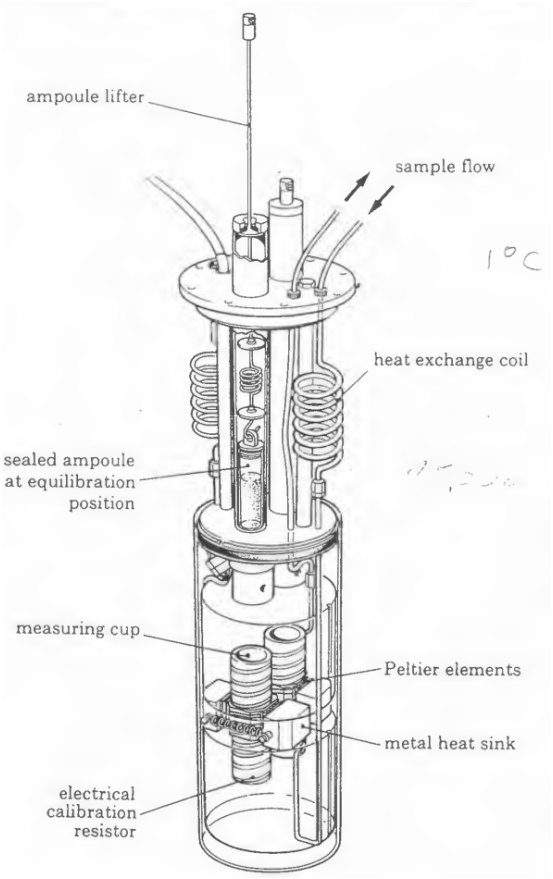


FIGURA 1.1: Interior del cilindro de medición, modificidado de [1].

Para calibrar el sistema se realizan lecturas consecutivas de la línea base, esto es la potencia que registra el calorímetro es ausencia de perturbación, y a partir de esto se ajusta el cero del canal, pues se busca que la línea base sea lo más cercana a 0 μW. Posteriormente, se aplica una potencia conocida sobre la resistencia y hacen lecturas de la potencia registrada por el calorímetro, a partir de esto se ajusta la ganancia, hasta que la potencia aplicada y la registrada tengan el valor más cercano posible. La potencia conocida debe concordar con el valor de amplificación del selector `RANGE` en la Figura 1.2.

FIGURA 1.2: Control del cero y la ganancia del calorímetro

Existen dos tipos de calibraciones, en la primera se modifican los valores de cero y ganancia en el equipo directamente, mientras que en la segunda los valores corresponden a ajustes en el software. Estos métodos de calibración reciben el nombre de estática y dinámica correspondientemente, y se debe asegurar que al momento de realizar una calibración dinámica, una calibración estática haya sido realizada previamente, pues los valores registrados por software son digitales (discretos), entonces la resolución de estos será determinada por la ganancia del circuito análogo, lo cual sólo es posible de controlar en el calorímetro directamente.

1. Estática

En la calibración estática, se debe manipular el equipo directamente. En primer lugar como fue mencionado anteriormente se ajusta el cero del canal en ausencia de perturbaciones. Para esto se manipula el potenciómetro de diez vueltas del lado izquierdo el cual está marcado como `ZERO` en la Figura 1.2, girando en sentido horario el valor

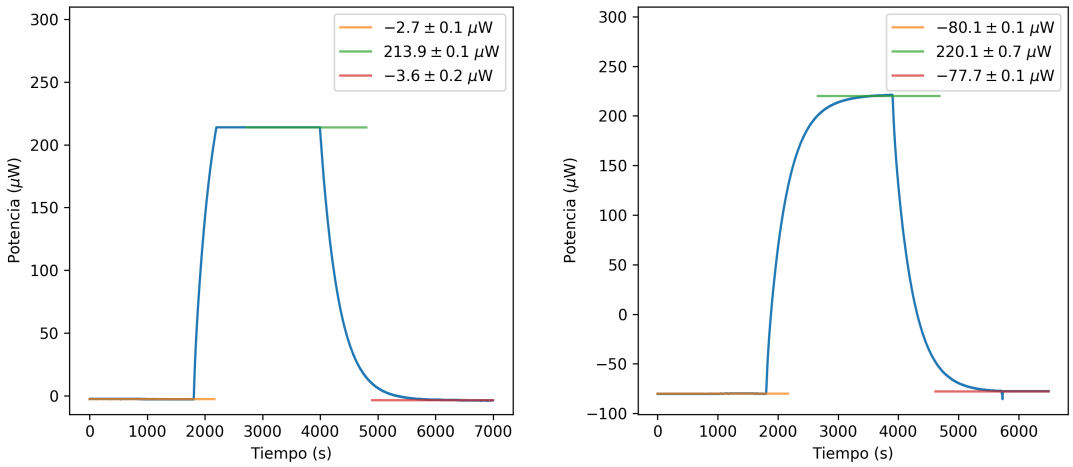
de potencia registrado será cada vez menor. Una vez el cero ha sido ajustado, se aplica sobre la resistencia una potencia conocida, por un tiempo determinado por el usuario, y se espera que la señal registrada por el calorímetro en su estado estacionario coincida con el valor aplicado de potencia. Para esto se debe esperar cerca de 20 minutos para que la resistencia alcance el estado estacionario y posteriormente se manipula el potenciómetro derecho (`FINE` en la Figura 1.2), donde giros en el sentido horario aumentan el valor de potencia registrado por el equipo, de esta forma se configura la ganancia del equipo.

Para activar la resistencia es posible hacerlo manualmente a través de los botones blancos que se observan en la Figura 1.2. En primer lugar se selecciona el canal sobre el cual se quiere realizar la calibración, para esto se oprime el botón correspondiente a cada canal (canal 1, botón izquierdo) del panel `SELECT`, esto ocasionará que la luz asociada a ese canal comience a oscilar. Posteriormente, si la luz continúa en este estado se debe volver a presionar el botón de `SELECT` asociado a este canal. El lado de la calibración A ó B se selecciona oprimiendo el botón `CAL A/B` donde la luz prendida indica A, y la potencia oprimiendo los botones de `RANGE` cuyo producto da el valor de potencia deseado en μW. Finalmente, para activar la calibración se oprimen de manera simultánea el botón de `SELECT` del canal y el botón `CAL A/B`. Una vez se haya calibrado el estado estacionario, la resistencia se apaga oprimiendo el botón del canal en `SELECT` y posteriormente, al mismo tiempo este botón y `CAL A/B`.

Otra manera de hacerlo es a través del software Digitam. En donde se realiza un método experimental que contenga la calibración eléctrica, este método ya se encuentra creado bajo el nombre `staticCalibration.tam`, el cual:

- Mide por 30 minutos la línea base.
- Aplica 300 μW sobre el lado B del canal 1 (referencia) por 35 minutos.
- Espera por 45 minutos a que la potencia se estabilice en la línea base.

Resultados de calibraciones estáticas se muestran en la Figura 1.3. En particular, para el caso de la Figura 1.3a se tiene que el cero está bien ajustado, sin embargo el valor de la amplitud no corresponde, pues el sensor se satura a un valor de 213,9 μW, sin una transición suave al estado estacionario. El efecto contrario se observa en la Figura 1.3b, pues la potencia aplicada corresponde con la leída, sin embargo, la línea base se encuentra a -80,1 μW.



(A) Potencia medida: 216,6 $\pm$ 0,2 μW. (B) Potencia medida: 300,2 $\pm$ 0,2 μW.

FIGURA 1.3: Ejemplos de calibraciones estáticas.

Para determinar las desviaciones a la línea base así como el valor del estado estacionario se implementó un algoritmo que se muestra en la Sección 3 y que realiza lo siguiente:

1. Toma el valor absoluto de la derivada de la potencia.
2. Realiza un filtro de medianas sobre los resultados anteriores, con un kernel de 101 puntos.
3. A los puntos con valores mayores o iguales a 0,5 se les asigna un valor de 1, y los restantes 0.
4. Se toma la derivada de los valores anteriores, de tal forma que aumentos corresponden con 1, y caídas con -1. Para el caso del estado estacionario y la última línea base, se toman el último 30 % de los datos, en donde ya se encuentran estables estos estados. Las fronteras de estos valores determinan las fronteras de las líneas base y el estado estacionario.
5. El promedio y desviación estándar se toma sobre cada uno de estos intervalos.

2. Dinámica

En la calibración dinámica el sistema registra la línea base por 1 minuto, posteriormente aplica 40 % de la potencia seleccionada sobre la resistencia, dando lugar a una pendiente constante por tres minutos, posteriormente, incremente nuevamente la potencia, mientras registra las pendientes observadas. Finalmente, la potencia se lleva hasta el 95 % y se obtiene un corto estado estacionario, de esta forma el software ajusta los valores de ganancia y cero del canal [1].

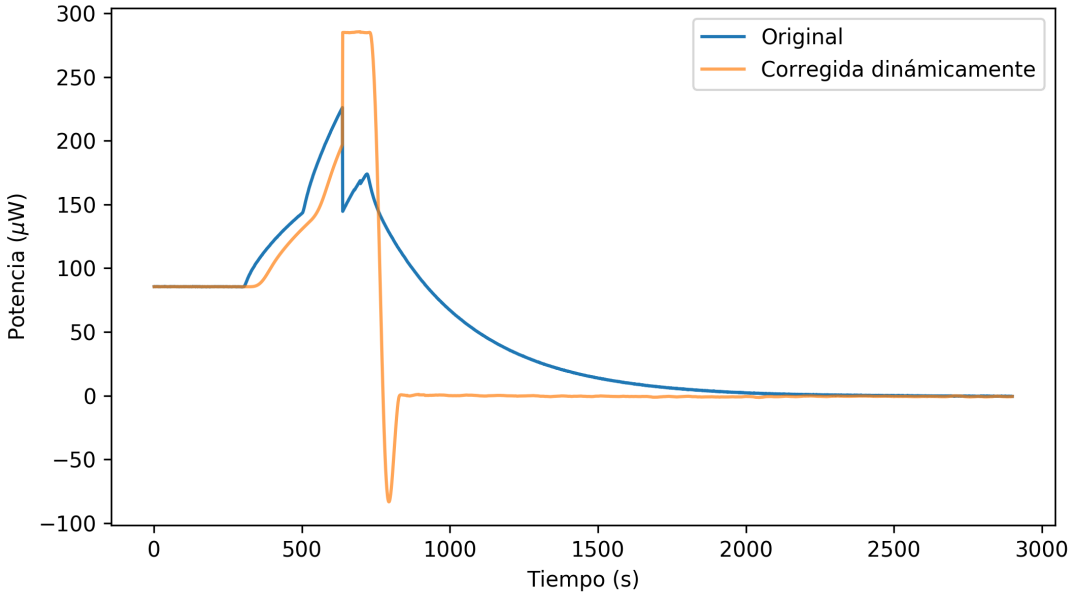


FIGURA 1.4: Ejemplo de una calibración dinámica.

En la Figura 1.4 se muestra un ejemplo de una calibración dinámica. En ella, la línea base se encuentra en 85,5 μW, luego de la calibración la línea base se estabiliza en -0,4 μW. La calibración dinámica permite caracterizar dos constantes que describen la inercia que presenta el calorímetro. Esto es que existe un tiempo de respuesta del equipo ante una perturbación. Luego de la calibración dinámica es posible tener una señal corregida dinámicamente, en donde se puede observar el estado estacionario al 95 % de la potencia aplicada, y los cambios de pendiente que fueron antes mencionados.

Finalmente, la calibración dinámica asegura la reproducibilidad de un experimento, como se muestra en la Figura 1.5, en donde luego de haber realizado una calibración estática, se hace una dinámica y posteriormente para confirmar sus resultados se ejecuta una nueva calibración estática, en donde no se modifican ninguno de los parámetros en

el calorímetro. En ella se observa como el inicio de las curvas de la calibración dinámica no coinciden, sin embargo, la parte final de estas se solapa, haciendo que las medidas sean reproducibles.

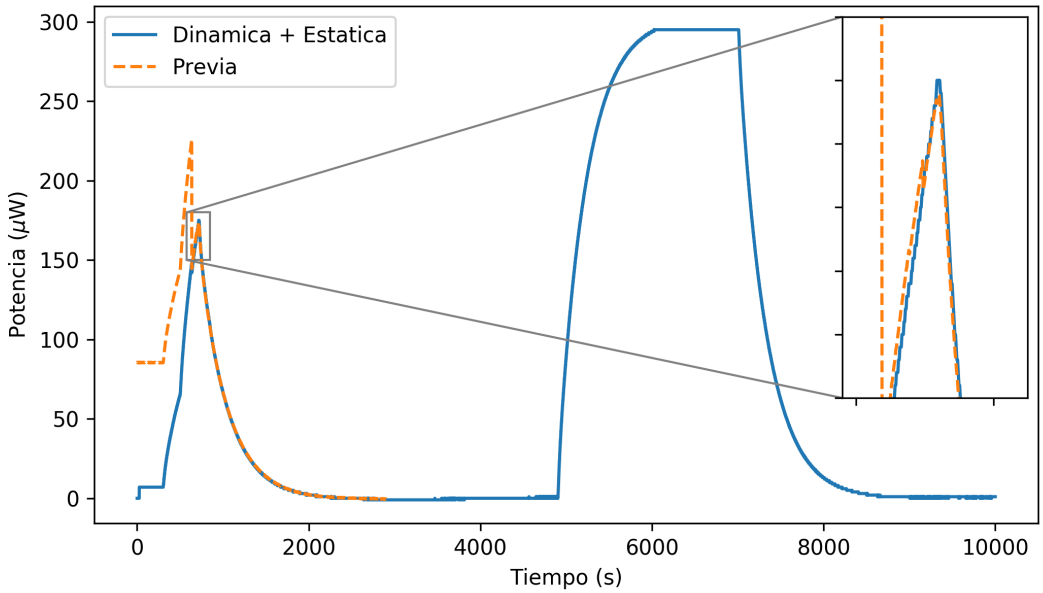


FIGURA 1.5: Calibraciones dinámica y estática que muestran el correcto funcionamiento del calorímetro.

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 Ohaus Analytical Plus (AP250D) y un densímetro Anton Paar DSA5000M. La balanza presenta incertidumbres de 1×10^{-5} y 1×10^{-4} g dependiendo del rango usado, para el primer caso la masa medida debe ser inferior a 80 g y en el segundo no puede superar los 250 g, siendo esta la capacidad máxima de la balanza. En el caso del densímetro, son necesarios volúmenes cercanos a 2 mL de una muestra líquida, con esto el equipo determina la densidad de la sustancia con incertidumbres de 1×10^{-6} g/cm^{-3} para una temperatura determinada por el usuario. 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 pipet4u Performance 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 [2]. 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.

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

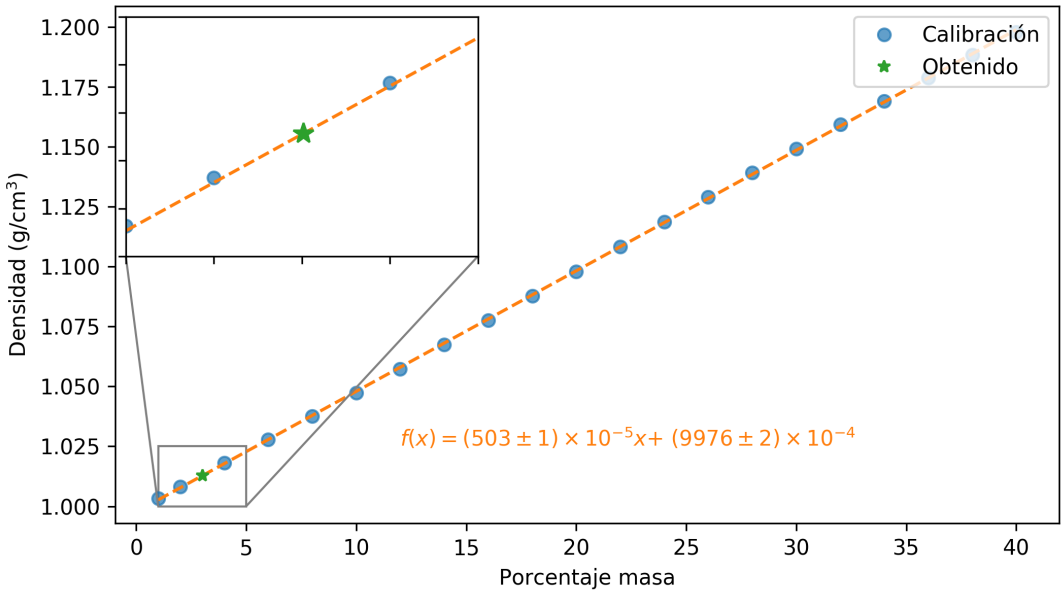


FIGURA 2.1: Determinación de la concentración de una solución de HCl usando valores de densidad reportados en la literatura [2].

Para obtener la concentración 0,84 $\pm$ 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:

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

Las densidades de las soluciones inicial y final se muestran en la Tabla 2.1.

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 0,10131 ± 0,00001 M, donde la incertidumbre se calcula usando:

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

Posteriormente se tomó una alícuota de 0,1709 g, la cual fue diluída en 99,5657 g de agua. Usando la Ecuación 2.4, se obtiene una concentración de 0,17265 mM. El resumen de las cantidades usadas para la dilución de las soluciones de ácido y bicarbonato, así como las densidades obtenidas a 20 °C se muestran en la Tabla 2.1.

2. Sistema de inyección

El sistema de inyección consiste en un motor de pasos acoplado a un tornillo de precisión, el cual controla el desplazamiento del émbolo de la jeringa de inyección. El

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

	$[\mathbf{M}]_\mathbf{i}$ (M)	$\mathbf{m_s}$ (g)	$\mathbf{m}_{H2O}$ (g)	ρ_s (g/cm^3)	ρ_f (g/cm^3)	$[\mathbf{M}]_\mathbf{f}$ (mM)
HCl	$0{,}84 \pm 0{,}04$	0,0297	99,5553	1,012832	0,998205	0,25
$KHCO_3$	$0{,}10131 \pm 0{,}00001$	0,1709	99,5657	1,003662	0,998215	0,17265

fluido saliente de la jeringa se desvía usando una manguera de cromatografía líquida de acero inoxidable, el cual se conecta en el otro extremo a un canal que lleva el fluido hasta la celda de medición.

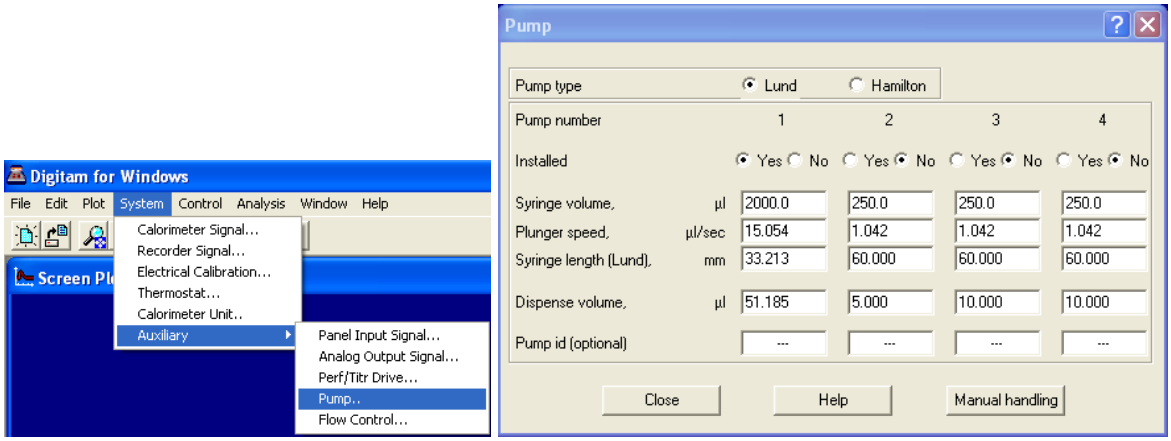
FIGURA 2.2: Sistema de inyección construido como alternativa al uso de las canulas.

Tanto la aguja de la jeringa como la manguera de cromatografía, corresponden con una alternativa para ingresar el fluido hasta la celda que difiere del mecanismo original en los volúmenes usados y el tipo de jeringas. Originalmente, para introducir una sustancia en la celda se hace uso de una jeringa de vidrio Hammilton de 250 μL la cual cuenta con una canula soldada en la punta de esta. Al momento de realizar el experimento, no se contaba con una jeringa de este tipo con la canula conectada, y a pesar que varios intentos fueron realizados para soldar la punta con la canula de oro y acero inoxidable no fue posible juntar las partes, en parte dado que el diámetro de la canula es considerablemente pequeño, siendo difícil de ver a simple vista. Lo anterior tiene consideraciones especiales, pues los diámetros de la jeringa y la manguera no son compatibles, por lo cual se hace necesario usar cinta de teflón para evitar al máximo fugas

en el sistema. Además el volumen interno de la manguera cromatográfica es mucho mayor al de la canula, por lo cual se debe tener especial cuidado por los volúmenes introducidos, pues no se debe exceder la capacidad de 4 mL de la celda.

2.1. Control por software

Para acceder al control de la jeringa, en el menú superior: `System >Auxiliary >Pump`. En el momento se cuenta con un único agitador para la celda, por lo cual sólo se encuentra instalado uno de los controladores de jeringa tipo `Lund`, por esta razón el sistema debe detectar automáticamente únicamente el primer controlador (`Installed = yes`). La configuración de una jeringa consiste en escribir el volumen de esta (`Syringe volume`), la velocidad con la que se quiere mover el émbolo (`Plunger speed`), su longitud (`Syringe length`) y el volumen por inyección (`Dispense volume`).



(A) Ingreso al menú. (B) Menú de configuración

FIGURA 2.3: Configuración de la jeringa en Digitam.

Una vez se encuentra configurada la jeringa, es posible usarla de dos maneras distintas. Por un lado se tiene el modo manual, donde el usuario tiene la posibilidad de avanzar rápidamente (`Fast forward`), por ejemplo para disminuir al distancia entre el émbolo y el pistón. También es posible aumentar esta distancia (`Fast backward`), avanzar un milímetro (`One milimeter forward`) y moverse la distancia requerida para que la jeringa dispense el volumen por inyección (`Dispense`). El control manual resulta útil antes de iniciar un experimento, pues con frecuencia será necesario ajustar la distancia entre el pistón y el émbolo para que haya contacto. Además en el caso de ser requerida una calibración de la jeringa es posible usar el botón de dispensar para determinar si el volumen dispensado corresponde con el volumen configurado.

FIGURA 2.4: Panel del uso manual del controlador de la jeringa.

El modo automático se configura al momento de definir el método del experimento. Para esto es necesario expandir la opción de `Auxiliary system` en el panel izquierdo del menú y seleccionar ítem `Pump and flow control`. Se debe tener en cuenta la sección del experimento que se está configurando. En el caso de la Figura 2.5, sólo existe la sección de `Baseline`, en esta sección se busca realizar 20 inyecciones cada una con el volumen de dispensación y velocidad configurados en la Figura 2.3b y 300 segundos de espera entre cada inyección.

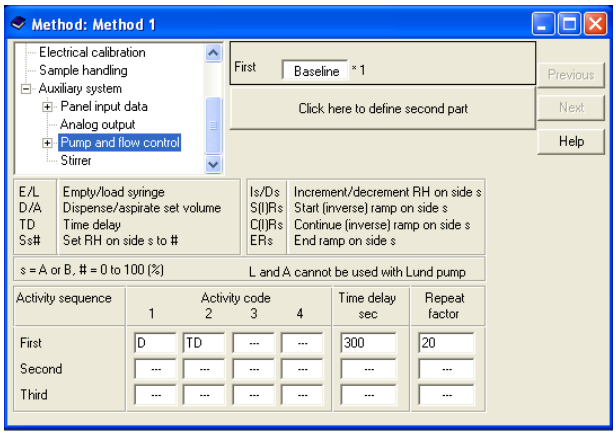


FIGURA 2.5: Panel de configuración de la jeringa en modo automático.

2.2. Calibración de la jeringa

Si bien el volumen de una jeringa es bien conocido, la longitud del émbolo es un parámetro que con frecuencia no se encuentra fácilmente. El controlador de la jeringa depende de este valor para relacionar la longitud que debe expandir el pistón para dispensar el volumen V_d. Por esta razón, para determinar la longitud correcta de la jeringa que debe ser configurada, se usaron distintos valores de esta y se midió la masa

de agua tipo 1 desplazada, para cada valor de longitud se tomaron 3 medidas, los cuales se muestran en la Tabla 2.2.

TABLA 2.2: Masa dispensada por la jeringa usando diferentes longitudes y un $V_d = 100{,}000\ \mu$L

Longitud (mm)	**Masa 1 (g)**	**Masa 2** (g)	**Masa 3 (g)**	**Promedio (g)**	**Desviacion (g)**
28,000	0,08194	0,08196	0,08262	0,0822	0,0004
29,000	0,08573	0,08673	0,08648	0,0863	0,0005
30,000	0,09003	0,08884	0,08970	0,0895	0,0006
31,000	0,09270	0,09322	0,09263	0,0928	0,0003
32,000	0,09551	0,09536	0,09538	0,0954	0,0001
33,000	0,09895	0,09825	0,09870	0,0986	0,0004
34,000	0,10172	0,10176	0,10137	0,1016	0,0002
35,000	0,10409	0,10318	0,10422	0,1038	0,0006
36,000	0,10679	0,10593	0,10757	0,1068	0,0008
37,000	0,10958	0,11015	0,10993	0,1099	0,0003
38,000	0,11384	0,11327	0,11360	0,1136	0,0003

Con el termómetro de mercurio usado en la calibración térmica de los sensores de temperatura se midió la temperatura del agua usada en la calibración de la jeringa, dando un valor de 17.7 °C. El densimetro usado en la Sección 1 fue configurado para realizar una lectura a esta misma temperatura, con lo cual se obtuvo un valor de densidad de 0.998679 g/cm^3. Con esta información es posible construir una curva que permite inferir la configuración a usar de la jeringa, esta curva junto con el valor escogido se muestran en la Figura 2.6. Para un volumen de dispensación de 100 μL, se obtuvo una longitud de 33,213 mm.

3. Realización del experimento

El método experimental está dividido en cuatro partes:

1. **Pause**: En la etapa inicial del experimento se busca medir el estado de la línea base por 5 minutos consecutivos.

2. **Baseline**: Luego de tener datos sobre el estado sin perturbar del sistema, se realiza una calibración dinámica para realizar un ajuste fino de los parámetros de ganancia y nivel del cero de la señal. Para esto se aplica primero

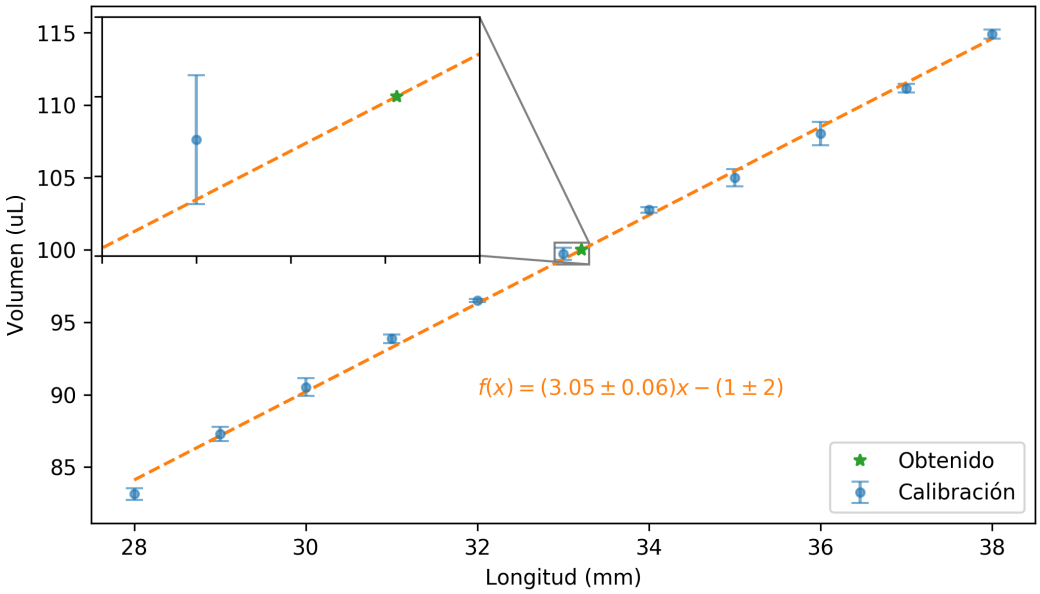


FIGURA 2.6: Curva de calibración de la jeringa usada.

3. **Pause:** En este punto se realiza una calibración estática para confirmar que la calibración dinámica fue correcta. Para esto, se aplican 300 μW sobre la celda por 30 minutos, posteriormente se retira la potencia y se esperan 50 minutos para la estabilización de la linea base.

4. **Main:** Una vez estabilizada la linea base, se realizan 30 inyecciones sucesivas con 10 minutos de espera entre cada inyección, la velocidad de inyección es de 50,018 μ/s y el volumen de inyección es de 51,185 μL.

4. Resultados

Referencias

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

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

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()
```

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]
```