

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

6 de diciembre de 2018



# Índice general

<b>1. Calibración Química</b>	<b>1</b>
1. Sistema de inyección . . . . .	1
1.1. Control por automatizado . . . . .	2
1.2. Calibración de la jeringa . . . . .	4
2. Calibración 1-propanol y agua . . . . .	5
2.1. Preparación de la solución 3 % . . . . .	6
2.2. Resultados . . . . .	7
3. Calibración con HCl y KHCO <sub>3</sub> . . . . .	7
3.1. Solución de HCl 0.25 mM . . . . .	8
3.2. Solución de KHCO <sub>3</sub> 0.17 mM . . . . .	9
3.3. Realización del experimento . . . . .	10
3.4. Resultados . . . . .	10
<b>Referencias</b>	<b>13</b>
<b>A. Imágenes de soporte</b>	<b>15</b>
<b>B. Códigos</b>	<b>17</b>
1. Firmware microcontrolador . . . . .	17
2. Determinación de intervalos estacionarios en la calibración del sistema de monitoreo de temperatura . . . . .	19
3. Software de Temperatura . . . . .	20
3.1. Comunicaciones . . . . .	20
3.2. Interfaz gráfica . . . . .	24
4. Disco 3D para sensor térmico en el calorímetro . . . . .	31
5. Determinación de los valores de una calibración estática . . . . .	32



# Índice de figuras

1.1. Sistema de inyección construido como alternativa al uso de las canulas. . . . .	2
1.2. Configuración de la jeringa en Digitam. . . . .	3
1.3. Panel del uso manual del controlador de la jeringa. . . . .	3
1.4. Panel de configuración de la jeringa en modo automático. . . . .	4
1.5. Curva de calibración de la jeringa usada. . . . .	5
1.6. Resultados obtenidos para la mezcla de agua con 1-propanol 2,96 %. . . . .	7
1.7. Determinación de la concentración de una solución de HCl usando valores de densidad reportados en la literatura [22]. . . . .	9
1.8. Las señales entre 0 y 100 minutos, corresponden con una calibración dinámica y estática. Posteriormente, las 2 inyecciones. . . . .	11
A.1. Control térmico del calorímetro. Modificado de [8]. . . . .	15
A.2. Equivalencia de resistencia y temperatura en una calibración de 1994. . . . .	16



# Índice de tablas

1.1. Masa dispensada por la jeringa usando diferentes longitudes y un $V_d = 100,000 \mu\text{L}$ . . . . .	5
1.2. Experimentos realizados con la solución 2,96 % de 1-propanol. . . . .	6
1.3. Densidades y masas medidas para preparar las soluciones con concentraciones 0,25 mM y 0,17 mM para el HCl y KHCO <sub>3</sub> correspondientemente. . . . .	10



# 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, esto es de vital importancia dado que las calibraciones eléctricas con frecuencia no transfieren a la celda el 100 % de la potencia aplicada, así mismo la distribución de calor puede ser considerablemente distinta a la de una reacción. Dos sistemas químicos fueron usados para la realización de la calibración química: la reacción del ácido clorhídrico con bicarbonato de potasio, y la disolución de 1-propanol en agua. Estas sistemas hacen uso de reactivos de fácil acceso, han sido estudiados previamente en los procesos de calibración de equipos calorimétricos, como el calorímetro de titulación NanoITC de *TA Instruments* con el que cuenta el grupo de **Termodinámica de Soluciones** [18-20]. En el caso de la neutralización del ácido clorhídrico es posible obtener parámetros termodinámicos como la entalpía de reacción, entropía y energía libre de Gibbs.

## 1. 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 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 1.1](#)).

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  $250 \mu\text{L}$  la cual cuenta con una canula soldada en la punta de esta. Al momento de realizar las calibraciones químicas, no se contaba con una jeringa de este tipo con la canula conectada, y a

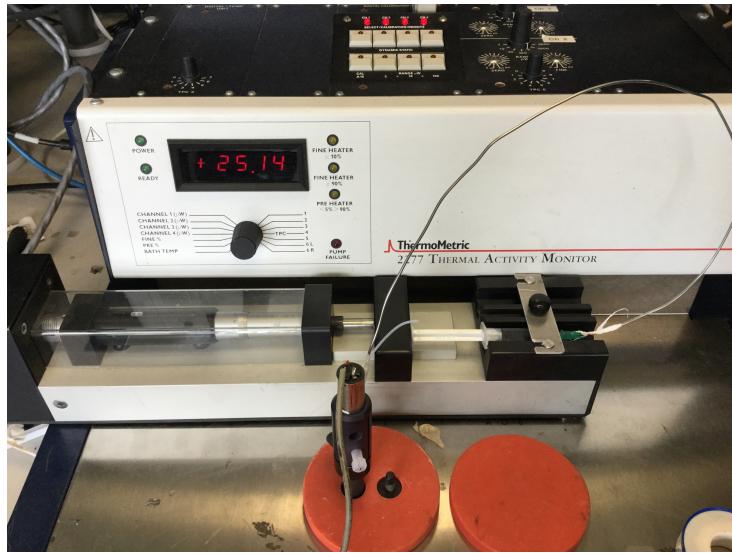


FIGURA 1.1: Sistema de inyección construido como alternativa al uso de las canulas.

pesar que varios intentos fueron realizados para soldar la punta con la canula de oro y acero inoxidable, no fue posible juntar las mismas, 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. Otro aspecto a considerar es que 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.

## 1.1. Control por automatizado

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 de inyección (Dispense volume).

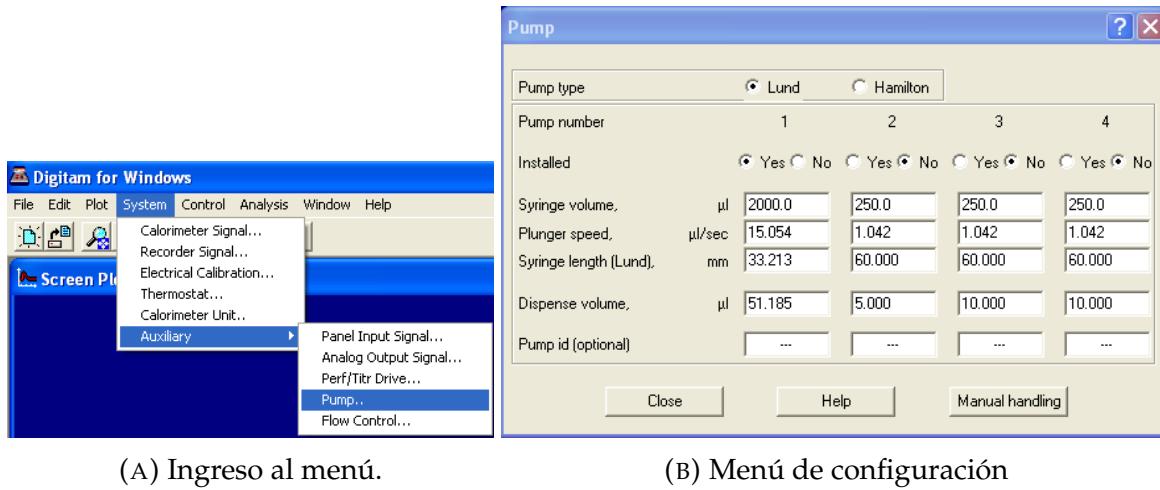


FIGURA 1.2: Configuración de la jeringa en Digitam.

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 la distancia entre el émbolo y el pistón. En este modo, también es posible aumentar esta distancia (Fast backward), avanzar un milímetro (One millimeter forward) y moverse la distancia requerida para que la jeringa dispense el volumen por inyección (Dispense). Por otro lado, 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.

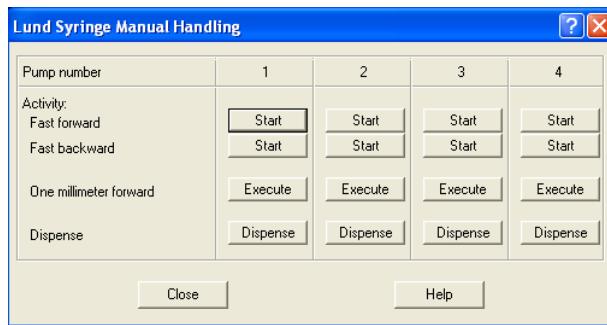


FIGURA 1.3: Panel del uso manual del controlador de la jeringa.

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 el Pump and flow control. Se debe tener en cuenta la sección

del experimento que se está configurando. En el caso de la [Figura 1.4](#), 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 1.2b](#), y 300 segundos de espera entre cada inyección.

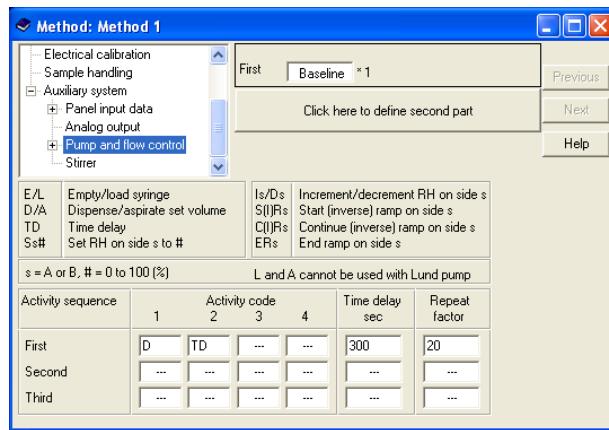


FIGURA 1.4: Panel de configuración de la jeringa en modo automático.

## 1.2. 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 1.1](#).

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 densímetro usado en la [Sección 3](#) fue configurado para realizar una lectura a esta misma temperatura, con lo cual se obtuvo un valor de densidad de 0.998679 g/cm<sup>3</sup>. Con esta información, es posible construir una curva que permite inferir la configuración a usar de la jeringa, la cual, junto con el valor escogido, se muestran en la [Figura 1.5](#). Para un volumen de dispensación de 100 µL, se obtuvo una longitud de 33,213 mm.

TABLA 1.1: Masa dispensada por la jeringa usando diferentes longitudes y un  $V_d = 100,000 \mu\text{L}$

Longitud (mm)	Masa 1 (g)	Masa 2 (g)	Masa 3 (g)	Promedio (g)	Desviación (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

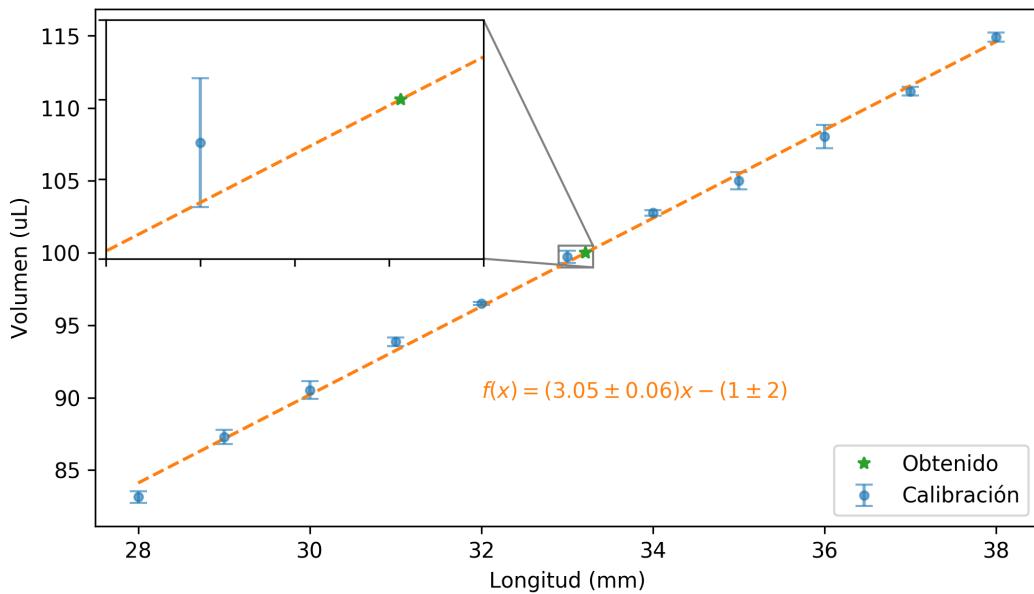


FIGURA 1.5: Curva de calibración de la jeringa usada.

## 2. Calibración 1-propanol y agua

La disolución de 1-propanol en agua constituye otro método reportado por varios autores para realizar una calibración química [18-20, 25]. Un gráfico de energía transferida en forma de calor en función del número de inyecciones o el volumen total inyectado, debe seguir una tendencia lineal [18-20].

## 2.1. Preparación de la solución 3 %

Para conocer la masa de agua que se debe adicionar para obtener una solución con concentración final  $P$  en porcentaje masa, a partir de una masa  $m'_{\text{total}}$  de una solución con concentración  $P'$  conocida, se usa la siguiente ecuación:

$$P = \frac{m_{\text{propanol}}}{m_{\text{total}}} = \frac{P'm'_{\text{total}}}{m_{\text{H}_2\text{O}} + m'_{\text{total}}} \longrightarrow m_{\text{H}_2\text{O}} = \left( \frac{P'}{P} - 1 \right) m'_{\text{total}} \quad (1.1)$$

Para un gramo de 1-propanol (99,8 % LiChrosolv para cromatografía líquida), fueron medidos 33,6457 g de agua tipo 1 previamente desgasificada de manera análoga a las soluciones de HCl y KHCO<sub>3</sub>. Sobre el agua se adicionaron 1.0290 g de 1-propanol, para obtener una solución final de 1-propanol de 2,96 %, cuya densidad a 20 °C es 0.993487 g/cm<sup>3</sup>.

Con esta solución tres experimentos fueron realizados, las cantidades usadas en cada uno de ellos se muestra en la [Tabla 1.2](#).

TABLA 1.2: Experimentos realizados con la solución 2,96 % de 1-propanol.

Experimento	Masa H <sub>2</sub> O (g)	# inyecciones
1	2.41273	30
2	2.49389	1
3	2.56668	1

El método experimental procedió de la misma forma que en la [Subsección 3.3](#), sin embargo como se muestra en la [??](#), luego de la primera inyección el sistema se saturó imposibilitando, nuevamente la cuantificación de la potencia generada. Una vez se determinó que el calorímetro había detectado, pero no cuantificado la disolución, se procedió preparar una solución 2,96 % de 1-propanol. Para esto se disolvieron 1,0290 g de 1-propanol en 33.6457 g de agua, y su densidad fue de un valor de 0.993487 g/cm<sup>3</sup>. Con el objetivo de limitar el efecto del sistema de inyección sobre las potencias registradas, para este experimento se realizaron 30 inyecciones con 1 minuto de espera entre cada una de ellas, además de ignorar la calibración estática, pues había sido realizada previamente. Los resultados se muestra en la [Figura 1.6](#), para realizar la integral se toman en cuenta el estado del calorímetro antes y después de las inyecciones, a partir de estas se obtiene la linea base durante las inyecciones (línea naranja). Posteriormente, se realiza la resta de la señal de potencia con la linea base para obtener que esta se encuentre justo sobre el cero (línea verde), y finalmente se integran los datos numéricamente usando la regla del trapecio [26] implementado en la librería numpy de Python [27].

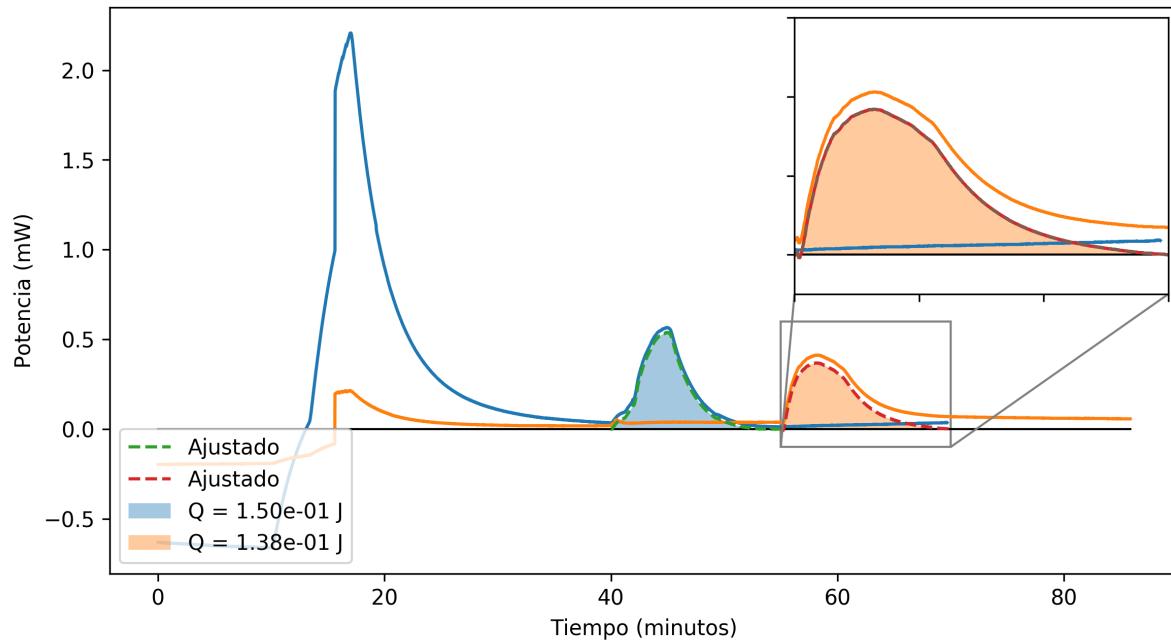


FIGURA 1.6: Resultados obtenidos para la mezcla de agua con 1-propanol 2,96 %.

## 2.2. Resultados

La masa de 1-propanol contenida en cada volumen de inyección  $V_{\text{iny}}$  está determinada por:

$$m_p = \% \rho V \quad (1.2)$$

La energía transferida en forma de calor corresponde con: 149.2 mJ. Ahora, considerando la densidad de la solución y el volumen de inyección se tiene que por cada una se introducen:

## 3. Calibración con HCl y KHCO<sub>3</sub>

Para la preparación de las soluciones de HCl y KHCO<sub>3</sub> y 1-propanol, se hace uso de forma sistemática de una balanza Ohaus Analytical Plus (AP250D) y un densímetro Anton Paar DSA5000M. La balanza presenta incertidumbres de  $1 \times 10^{-5}$  y  $1 \times 10^{-4}$  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 solución, con esto el equipo determina la densidad de la sustancia con incertidumbres

de  $1 \times 10^{-6}$  g/cm<sup>-3</sup> para una temperatura determinada por el usuario. Además, en el proceso de preparación se hace necesaria la toma de dos alícuotas de 30  $\mu\text{L}$  y 170  $\mu\text{L}$ , para el HCl y KHCO<sub>3</sub> correspondientemente, por lo cual se usa una micropipeta pipet4u Performance con rango de 20 a 200  $\mu\text{L}$ , que en el rango trabajado tienen precisiones superiores al 99.4 % [21].

### 3.1. 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  $\rho$  [22]. 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 1.7.

$$\delta[w_t] = \sqrt{\left(\frac{\rho - b}{m^2} \delta m\right)^2 + \left(\frac{\delta b}{m}\right)^2 + \left(\frac{\delta \rho}{m}\right)^2} \quad (1.3)$$

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]:

$$[M] = 10 \frac{[w_t]\rho}{m_m} \quad m_m \text{ la masa molecular del HCl} \quad (1.4)$$

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

$$\delta[M] = [M] \sqrt{\delta[w_t]^2 + \delta\rho^2} \quad (1.5)$$

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. En la cual, la concentración final se calcula a partir de las densidades de las soluciones inicial ( $\rho_s$ ) y final ( $\rho_f$ ), las masas de agua ( $m_{\text{H}_2\text{O}}$ ) y de solución inicial ( $m_s$ ) usando la Ecuación 1.6:

$$[M]_f = [M]_i \frac{V_s}{V_f} = [M]_i \frac{m_s / \rho_s}{(m_s + m_{\text{H}_2\text{O}}) / \rho_f} = [M]_i \left( \frac{m_s}{m_s + m_{\text{H}_2\text{O}}} \right) \left( \frac{\rho_s}{\rho_f} \right) \quad (1.6)$$

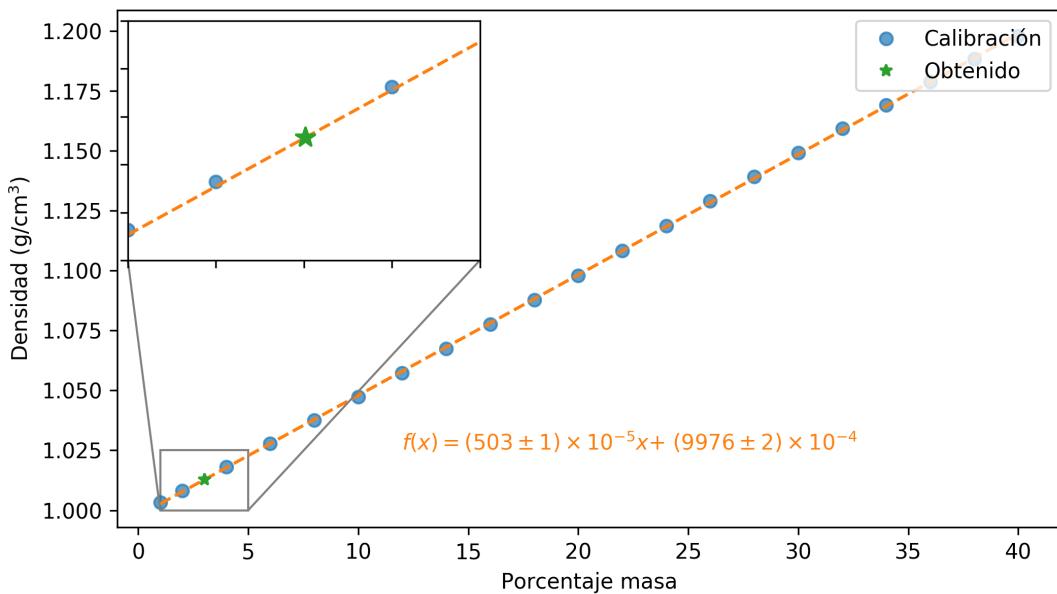


FIGURA 1.7: Determinación de la concentración de una solución de HCl usando valores de densidad reportados en la literatura [22].

Las densidades de las soluciones inicial y final se muestran en la [Tabla 1.3](#).

### 3.2. Solución de KHCO<sub>3</sub> 0.17 mM

En un balón aforado de 10 mL fueron adicionados 0,10143 g de KHCO<sub>3</sub>, junto con 9,93551 g de H<sub>2</sub>O. La densidad fue medida a 25 °C y su valor fue 1,003662 g/cm<sup>3</sup>. La concentración de esta solución se calculó usando la siguiente ecuación:

$$[M] = \frac{n}{V} = \frac{m\rho}{m_m(m + m_{H_2O})} \quad (1.7)$$

Obteniendo un valor de  $0,10131 \pm 0,00001$  M, para la cual la incertidumbre se calcula usando:

$$\delta[M] = \sqrt{\frac{m^2}{m_m^2(m + m_{H_2O})^2}\delta\rho^2 + \frac{\rho^2 m_{H_2O}^2}{m_m^2(m + m_{H_2O})^4}\delta m^2 + \frac{\rho^2 m^2}{m_m^2(m + m_{H_2O})^4}\delta m_{H_2O}^2} \quad (1.8)$$

Posteriormente se tomó una alícuota de 0,1709 g, que fue diluida en 99,5657 g de agua. Usando la [Ecuación 1.6](#), 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 1.3](#).

TABLA 1.3: Densidades y masas medidas para preparar las soluciones con concentraciones 0,25 mM y 0,17 mM para el HCl y KHCO<sub>3</sub> correspondientemente.

	[M] <sub>i</sub> (M)	<b>m<sub>s</sub></b> (g)	<b>m<sub>H2O</sub></b> (g)	$\rho_s$ (g/cm <sup>3</sup> )	$\rho_f$ (g/cm <sup>3</sup> )	[M] <sub>f</sub> (mM)
<b>HCl</b>	0,84 ± 0,04	0,0297	99,5553	1,012832	0,998205	0,25
<b>KHCO<sub>3</sub></b>	0,10131 ± 0,00001	0,1709	99,5657	1,003662	0,998215	0,17265

### 3.3. 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 120 minutos consecutivos.
2. **Baseline:** Luego de tener datos sobre el estado sin perturbar del sistema, se realiza una calibración dinámica para obtener un ajuste fino de los parámetros de ganancia y nivel del cero de la señal.
3. **Pause:** En este punto se realiza una calibración estática para confirmar que la calibración dinámica fue correcta. Para esto, se aplican 300  $\mu$ W 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  $\mu$ /s y el volumen de inyección es de 51,185  $\mu$ L.

En la celda de medición fueron adicionados 1,6029 g de la solución de KHCO<sub>3</sub> 0,17265 mM, y la jeringa fue cargada con 2,0 mL de la solución de HCl 0,2469 mM, ambas soluciones fueron desgasificadas por 15 minutos a 44 °C en un sonicador, además, la temperatura del baño térmico se mantuvo en 25,05 ± 0,06 °C a lo largo del experimento.

### 3.4. Resultados

Considerando un proceso a presión constante, se tiene una expresión que relaciona la potencia con la entalpía por inyección, donde el signo negativo en la [Ecuación 1.9](#)

viene de la polarización del equipo, lecturas de potencia positivas corresponden con reacciones exotérmicas:

$$\int_t^{t+\Delta t_{\text{iny}}} P dt = Q_{\text{iny}} = -\Delta H_{\text{iny}} \quad (1.9)$$

Sin embargo

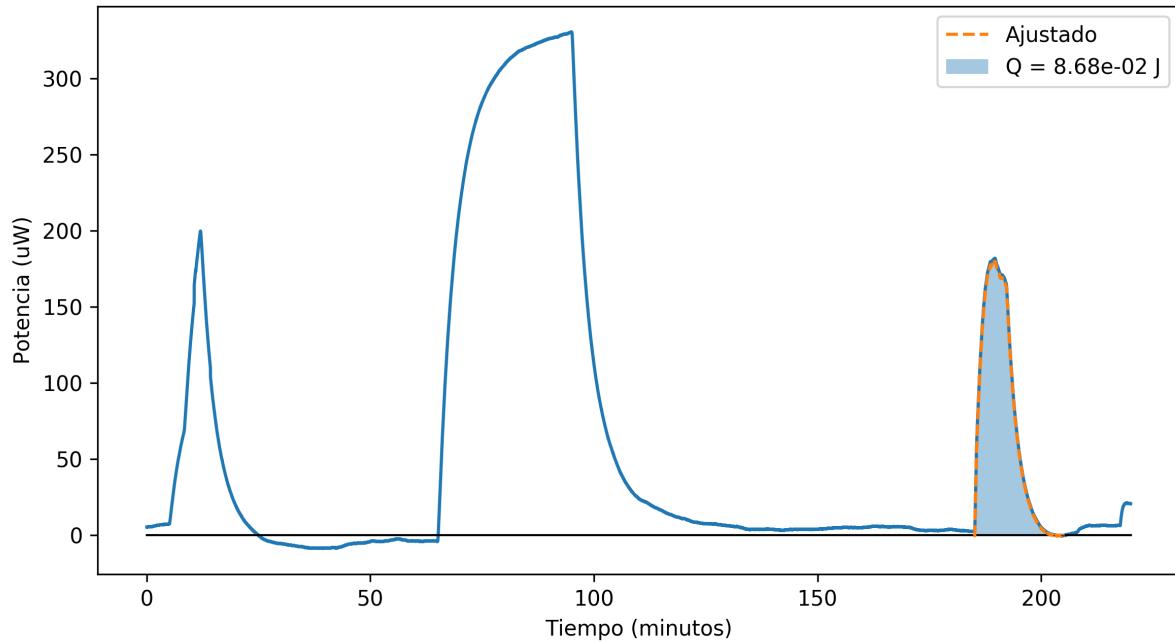


FIGURA 1.8: Las señales entre 0 y 100 minutos, corresponden con una calibración dinámica y estática. Posteriormente, las 2 inyecciones.

Dado que el calorímetro registra valores de potencia y esta corresponde con energía por unidad de tiempo, es necesario integrar en el tiempo la señal obtenida para cada inyección, de esta manera se determina el calor generado por cada una de estas. Además, de la primera ley de la termodinámica se tiene que  $U = Q - \int p dV$  y  $H \equiv U + pV$ , la entalpía está dada por:

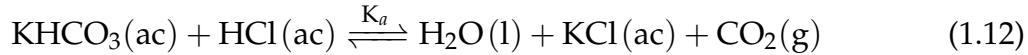
$$\Delta H = Q + V\Delta p \quad (1.10)$$

Para obtener la entalpía de la reacción se grafican las entalpías de inyección en función de la razón molar, y se toma la diferencia entre la asíntota inicial y final, cuyo valor corresponde con la entalpía total de la reacción [20]. Posteriormente, para determinar la entalpía molar se divide el resultado anterior por el número de moles de titulante usado. Por otro lado se tiene que en el punto de equivalencia la pendiente corresponde

con la constante de afinidad, para la cual se cumple la siguiente relación [20, 23, 24]:

$$\Delta G = -RT \ln K_a \quad (1.11)$$

Donde  $K_a$  se denomina la constante de afinidad y se obtiene del equilibrio de la reacción:



Usando la [Ecuación 1.11](#) se obtiene el cambio en energía libre de Gibbs, lo cual a su vez permite determinar la entropía, pues de la definición de esta junto con la primera ley de la termodinámica se obtiene:

$$G \equiv H - TS = (U + pV) - TS \quad (1.13)$$

$$\begin{aligned} dG &= dU + pdV + VdP - SdT - TdS \\ &= dQ - pdV + pdV + VdP - SdT - TdS \\ &= (dQ + VdP) - SdT - TdS \\ &= dH - SdT - TdS \end{aligned} \quad (1.14)$$

Lo cual para condiciones isotérmicas y cambios grandes ( $d \rightarrow \Delta$ ) se reduce a:  $\Delta G = \Delta H - T\Delta S$ , de donde se obtiene:

$$\Delta S = \frac{\Delta H - \Delta G}{T} \quad (1.15)$$

Para poder calcular las cantidades mencionadas anteriormente, es necesario obtener la potencia de cada inyección, sin embargo, como se observa en la [Figura 1.8](#), existe una segunda inyección (10 minutos después que la otra) que registra una gran potencia, posterior a esta se desequilibra el calorímetro.

La energía medida en las inyecciones se calcula con la [Ecuación 1.9](#). Entre  $t = 127$  min, hasta  $t + \Delta T = 141$  min, como se muestra en color naranja en la [Figura 1.8](#), dando lugar a un valor de 45,88 mJ. Considerando ahora que se agregaron 102,37  $\mu\text{L}$  (2 inyecciones) de una solución 0,2469 mM de HCl, fueron adicionados 25,28 nmoles de HCl, por lo cual se tiene que  $Q = \Delta H = -1815 \text{ kJ/mol}$ , valor que es muy superior al esperado de  $-9,0 \pm 0,9 \text{ kJ/mol}$  [20]. Además, dicho resultado no explica la desestabilización posterior del calorímetro, así como la no observación de más inyecciones. Por lo cual se procedió a probar el sistema la disolución de propanol en agua.

# 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) Suurkuusk, J. 2277 *Thermal Activity Monitor*; inf. téc.; Järfälla: Termometric AB.
- (9) Simon, S. H., *The Oxford solid state basics*; OUP Oxford: 2013.
- (10) 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.
- (11) Winkelmann, M; Hüttl, R y Wolf, G *Thermochimica acta* **2004**, 415, 75-82.
- (12) Morrison, J. *Pure and applied chemistry* **1987**, 59, 7-14.
- (13) 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.
- (14) Instruments, T. *LM35 Precision Centigrade Temperature Sensors*; inf. téc.; 1999.
- (15) AVR, M. *8-bit AVR microcontroller with 1K Bytes In-System Programmable Flash*; inf. téc.; 2010.

- (16) Grewal, H. *Oversampling the ADC12 for higher resolution*; inf. téc.; 2006.
- (17) Alexander, C. K.; Sadiku, M. N. y Sadiku, M., *Fundamentals of electric circuits*; McGraw-Hill New York: 2009; vol. 3.
- (18) Demarse, N. A.; Quinn, C. F.; Eggett, D. L.; Russell, D. J. y Hansen, L. D. *Analytical biochemistry* **2011**, 417, 247-255.
- (19) Adão, R.; Bai, G.; Loh, W. y Bastos, M. *The Journal of Chemical Thermodynamics* **2012**, 52, 57-63.
- (20) Instruments, T. *Nano Isothermal Titration Calorimeter (ITC), test procedures*; inf. téc.; 2013.
- (21) Gmb, A. B. *Pipetas manuales alta precisión, pipet4u 20 - 200 uL*; inf. téc.; 2012.
- (22) Perry, R. H.; Green, D. W.; Maloney, J. O. y col. *Perry's chemical engineers' handbook.*, 200722.
- (23) Matsuyama, B. Y.; Krasteva, P. V. y Navarro, M. V. en *c-di-GMP Signaling*; Springer: 2017, págs. 403-416.
- (24) Velazquez-Campoy, A. y Freire, E. *Nature protocols* **2006**, 1, 186.
- (25) Briggner, L.-E. y Wadsö, I. *Journal of biochemical and biophysical methods* **1991**, 22, 101-118.
- (26) Landau, R. H.; Mejía, M. J. P.; Páez, J. y Bordeianu, C. C., *A survey of computational physics: introductory computational science*; Princeton University Press: 2008; vol. 1.
- (27) Walt, S. v. d.; Colbert, S. C. y Varoquaux, G. *Computing in Science & Engineering* **2011**, 13, 22-30.

# Imágenes de soporte

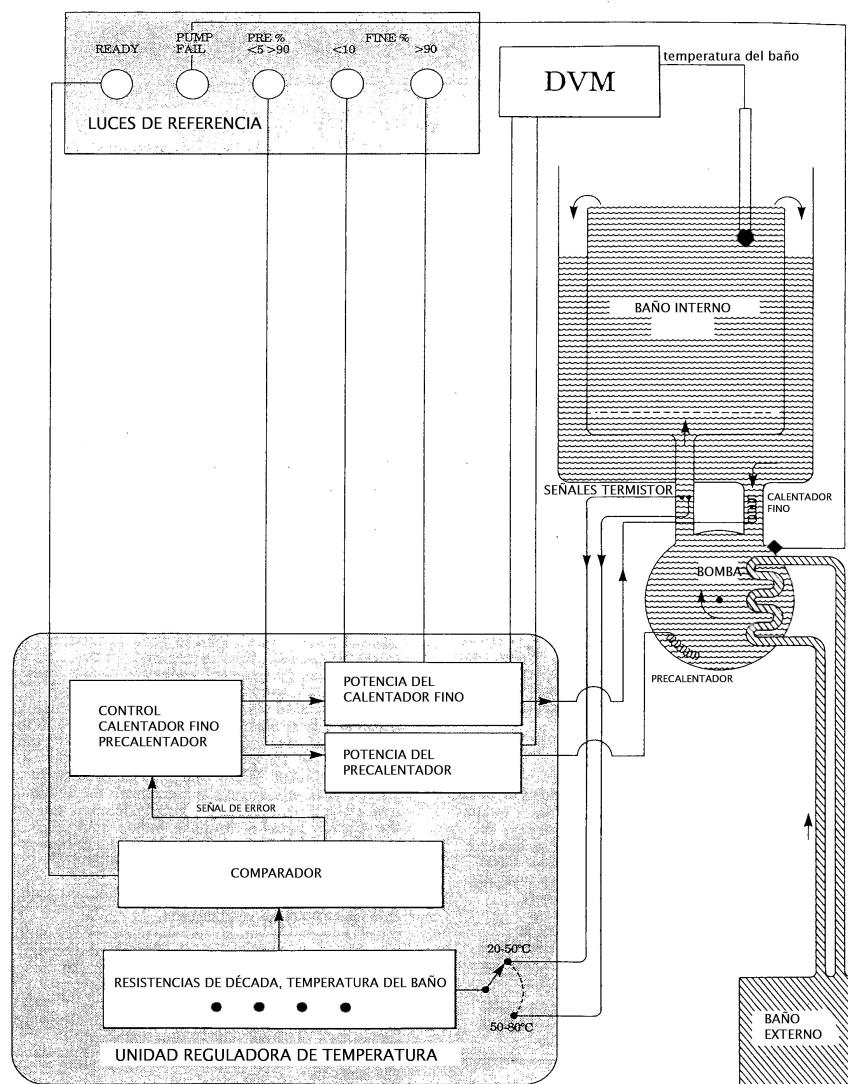


FIGURA A.1: Control térmico del calorímetro. Modificado de [8].

## Thermal Activity Monitor

No. 404

Date: 05-17-1994

Sensor #: 411B

Lowtemp

## Calibration data

Resistans	Temp	Error	Bath temp	Decade set	Circ temp	Bath temp	Decade set	Circ temp
Ω	°C	°C	°C	Ω/10	°C	°C	Ω/10	°C
40607.50	19.999	0.000	10	6474	6.0	30	2616	27.0
25895.50	30.235	-0.000	11	6171	7.0	31	2507	28.0
17268.70	39.999	0.000	12	5884	8.5	32	2403	29.0
11656.50	50.006	-0.000	13	5612	9.5	33	2304	30.0
			14	5354	10.5	34	2209	31.0
			15	5109	11.5	35	2119	32.0
			16	4877	12.5	36	2033	33.0
			17	4657	13.5	37	1951	34.5
			18	4448	14.5	38	1873	35.5
			19	4249	15.5	39	1798	36.5
<b>Thermistor constants</b>			20	4061	16.5	40	1727	37.5
$R = A * \exp(B/T + C/T^2)$			21	3881	17.5	41	1659	38.5
$A = 0.11332690E-01$			22	3711	18.5	42	1594	39.5
$B = 4957.60$			23	3549	19.5	43	1531	40.5
$C = -156384.66$			24	3395	21.0	44	1472	41.5
$T = \text{temp. in Kelvin}$			25	3249	22.0	45	1415	42.5
			26	3109	23.0	46	1361	43.5
			27	2977	24.0	47	1309	44.5
			28	2850	25.0	48	1259	45.5
			29	2730	26.0	49	1211	46.5

Sensor #: 406A

Hightemp

## Calibration data

Resistans	Temp	Error	Bath temp	Decade set	Circ temp	Bath temp	Decade set	Circ temp
Ω	°C	°C	°C	Ω/10	°C	°C	Ω/10	°C
61596.80	50.026	0.000	50	6166	47.5	70	2765	68.0
40850.50	60.007	-0.000	51	5912	48.5	71	2663	69.0
27224.10	70.413	0.000	52	5669	49.5	72	2564	70.0
19224.30	79.800	-0.000	53	5438	51.0	73	2469	71.5
			54	5217	52.0	74	2379	72.5
			55	5006	53.0	75	2292	73.5
			56	4805	54.0	76	2209	74.5
			57	4613	55.0	77	2129	75.5
			58	4429	56.0	78	2052	76.5
			59	4254	57.0	79	1979	77.5
<b>Thermistor constants</b>			60	4086	58.0	80	1909	78.5
$R = A * \exp(B/T + C/T^2)$			61	3926	59.0	81	1841	79.5
$A = 0.12585980E-01$			62	3773	60.0	82	1776	80.5
$B = 5542.77$			63	3627	61.0	83	1714	81.5
$C = -182502.18$			64	3487	62.0	84	1654	82.5
$T = \text{temp. in Kelvin}$			65	3353	63.0	85	1597	83.5
			66	3225	64.0	86	1541	84.5
			67	3103	65.0	87	1488	85.5
			68	2985	66.0	88	1437	86.5
			69	2873	67.0	89	1388	87.5

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

# Códigos

## 1. Firmware microcontrolador

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

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

#define N_MEAN 4096 // 4**6

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

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

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

```
sei();

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

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

return 0;
}

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

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

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

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

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

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

## 2. Determinación de intervalos estacionarios en la calibración del sistema de monitoreo de temperatura

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import medfilt

def findZeros(ch1, ch2, ch3):
    y = np.zeros((len(ch1), 3))
    y[:, 0] = ch1[:, 1]
    y[:, 1] = ch2[:, 1]
    y[:, 2] = ch3[:, 1]
```

```
y = np.mean(y, axis = 1)
dy = abs(np.diff(y))
abs_dy = np.zeros(len(dy) + 1)
abs_dy[0] = dy[1]
abs_dy[1:] = dy

dy = medfilt(abs_dy, kernel_size = 507)
pos = (dy > 5e-2)
dy[pos] = 0.5
dy[np.logical_not(pos)] = 0
return abs_dy, dy
```

### 3. Software de Temperatura

#### 3.1. 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__("{} is not a calibration port.".format(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:\u2022%d" %(i, val))
        text = "\t".join(text)
        print(text)

        sleep(5e-2)
    except KeyboardInterrupt:
        break

SERIAL.close()
```

### 3.2. 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/9349dff42aeffb32e48a0868edfa32d

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("%A %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_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()
```

## 4. Disco 3D para sensor térmico en el calorímetro

```
thick = 1.2;
d_screw = 4;
r_screw = (86.5 - d_screw) / 2;
$fn = 40;

difference()
{
    cylinder(d = 92, h = thick, center = true);

    translate([0, r_screw, 0])
    cylinder(d = d_screw, h = 2*thick, center = true);

    translate([r_screw, 0, 0])
    cylinder(d = d_screw, h = 2*thick, center = true);

    translate([(r_screw)/sqrt(2), (r_screw)/sqrt(2), 0])
    cylinder(d = d_screw, h = 2*thick, center = true);

    translate([0, -r_screw, 0])
    cylinder(d = d_screw, h = 2*thick, center = true);

    translate([-r_screw, 0, 0])
    cylinder(d = d_screw, h = 2*thick, center = true);

    translate([- (r_screw)/sqrt(2), -(r_screw)/sqrt(2), 0])
    cylinder(d = d_screw, h = 2*thick, center = true);
```

```
translate([- (r_screw) / sqrt(2), (r_screw) / sqrt(2), 0])
cylinder(d = d_screw, h = 2 * thick, center = true);

translate([(r_screw) / sqrt(2), -(r_screw) / sqrt(2), 0])
cylinder(d = d_screw, h = 2 * thick, center = true);

cylinder(d = 10, h = 2 * thick, center = true);
}
```

## 5. 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]
```