

**GRADO EN INGENIERÍA ELECTRÓNICA DE  
TELECOMUNICACIONES**



**VNIVERSITAT  
DE VALÈNCIA**

**TRABAJO DE FIN DE GRADO**

**DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE  
MONITORIZACIÓN APlicado A LA MEDIDA DE  
HUMEDAD**

**AUTOR:**

**ADRIÁN SÁNCHEZ PLA**

**TUTORIA:**

**DRA. INGA SILVIA CASANS BERGA**

**DRA. INGA ASUNCIÓN EDITH NAVARRO ANTÓN**

**OCTUBRE, 2019**



## AGRADECIMIENTOS

---

En primer lugar, me gustaría agradecer a mis tutoras de trabajo, Silvia Casans Berga y Edith Navarro Antón por haberme brindado la posibilidad de trabajar junto a ellas en el estudio de la medida de humedad.

También me gustaría agradecer a mi familia sus sabios consejos y su comprensión. En especial, a mi hermana Ruth por permitir que trabaje con su computadora. A mis tíos Agustina y M<sup>a</sup>Jesús por permitirme realizar el estudio del sótano de su casa.

Pero, sobre todo me gustaría agradecer a mi tío Juan Gabriel el haber despertado en mí el interés, que cuando era niño entendía por tecnología, pero que cuando crecí comprendí que era ingeniería.

Finalmente, nombrar a mis cuatro amigos Pepe, Jorge, Luis y Andrea. No solo nos aconsejamos entre nosotros en cuanto a asuntos académicos se refiere, sino que somos el apoyo o vía de escape que en muchas ocasiones necesitamos. Cito a mi amigo Luis: “Los amigos son la familia que uno elige”.

¡Muchas gracias a todos!

Adrián Sánchez Pla.



# ÍNDICE

---

<b>1. INTRODUCCIÓN.....</b>	<b>1</b>
1.1 RESUMEN .....	1
1.2 MOTIVACIÓN.....	1
1.3 OBJETIVOS .....	2
1.4 PLANIFICACIÓN .....	2
<b>2. SISTEMA DE MONITORIZACIÓN IMPLEMENTADO.....</b>	<b>4</b>
2.1 HARDWARE.....	4
2.1.1 Conversor resistencia – frecuencia .....	4
2.1.2 Sensores.....	8
• Acelerómetro.....	8
• Sensor de Temperatura y Humedad .....	10
2.1.3 Módulos inalámbricos .....	13
2.1.4 Módulo reloj.....	14
2.1.5 Módulo tarjeta SD .....	15
2.1.6 Arduino.....	16
• Arduino UNO.....	16
• Arduino NANO .....	18
<b>3. SOFTWARE.....</b>	<b>19</b>
3.1 ENTORNO DE PROGRAMACIÓN XCTU .....	19
3.2 ENTORNO DE PROGRAMACIÓN ARDUINO.....	20
3.3 PROGRAMACIÓN NODO DE MEDIDA.....	21
3.3.1 Declaración de variables.....	22
3.3.2 Configuración de valores por defecto – setup() .....	24
3.3.3 Bucle de programa – loop () .....	24
3.3.4 Subrutinas de interrupción .....	27
3.3.5 Funciones del programa adicionales .....	28
3.4 PROGRAMACIÓN NODO BASE .....	31
3.4.1 Librerías .....	33
3.4.2 Declaración de variables.....	35
3.4.3 Configuración de valores por defecto – setup() .....	36
3.4.4 Bucle de programa – loop() .....	36
<b>4. INTERFAZ DE USUARIO.....</b>	<b>40</b>
4.1 PANEL PRINCIPAL.....	40
4.2 SELECCIÓN DE PUERTO DE COMUNICACIONES .....	41
4.3 CALIBRACIÓN DE ACELERÓMETRO .....	42
4.4 CALIBRACIÓN CONVERTIDOR RESISTENCIA – FRECUENCIA .....	43
4.5 ALMACENAMIENTO DE COEFICIENTES.....	44
4.6 CAMBIO TCICLO .....	45
4.7 ADQUISICIÓN DE DATOS .....	45
4.7.1 Adquisición de datos en procesos de calibración.....	45
4.7.2 Adquisición de datos completa.....	46
<b>5. RESULTADOS EXPERIMENTALES .....</b>	<b>47</b>
5.1 VERIFICACIÓN DEL CIRCUITO .....	47
5.2 VERIFICACIÓN DEL SISTEMA DE MONITORIZACIÓN .....	48
5.2.1 Primera serie.....	51
5.2.2 Segunda serie .....	52

5.2.3 <i>Tercera serie</i> .....	53
5.2.4 <i>Cuarta serie</i> .....	53
<b>6. CONCLUSIONES Y PROPUESTAS DE FUTURO.....</b>	<b>55</b>
6.1 CONCLUSIONES .....	55
6.2 PROPUESTAS DE FUTURO.....	55
<b>7. REFERENCIAS BIBLIOGRÁFICAS.....</b>	<b>57</b>
<b>ANEXO 1: ESQUEMÁTICO NODO DE MEDIDA .....</b>	<b>59</b>
<b>ANEXO 2: ESQUEMÁTICO ARDUINO NANO .....</b>	<b>60</b>
<b>ANEXO 3: ESQUEMÁTICO ARDUINO UNO.....</b>	<b>60</b>
<b>ANEXO 1: LAYOUT NODO DE MEDIDA.....</b>	<b>62</b>
<b>ANEXO 5: CÓDIGO NODO BASE .....</b>	<b>62</b>
<b>ANEXO 6: CÓDIGO NODO DE MEDIDA.....</b>	<b>65</b>

# LISTA DE FIGURAS

---

FIGURA 1: DIAGRAMA DE BLOQUES DEL SISTEMA DE MONITORIZACIÓN IMPLEMENTADO .....	4
FIGURA 2: CIRCUITO CONVERSOR RESISTENCIA-FRECUENCIA .....	5
FIGURA 3: CRONOGRAMA DE CARGA Y DESCARGA DE CONDENSADOR C (ROJO). ONDA CUADRADA $V_{OA}$ (VERDE) .....	5
FIGURA 4: CRONOGRAMA DE SEÑALES $V_{OA}$ , $V_D$ Y $V_{OUT}$ .....	7
FIGURA 5: ACELERÓMETRO ADXL335 .....	8
FIGURA 6: POSICIONES ABSOLUTAS ACELERÓMETRO .....	9
FIGURA 7: SENSOR DE TEMPERATURA Y HUMEDAD RELATIVA SHT15.....	10
FIGURA 8: SECUENCIA DE INICIO DE COMUNICACIÓN CON SENSOR SHT15 .....	11
FIGURA 9: MÓDULO XBEE SERIES 2 .....	13
FIGURA 10: ADAPTADOR XBEE-USB .....	14
FIGURA 11: MÓDULO RELOJ (CHRONODOT V2.1) .....	14
FIGURA 12: MÓDULO DE ALMACENAMIENTO SD (WIRELESS SD SHIELD) .....	15
FIGURA 13: PLACA ARDUINO UNO.....	17
FIGURA 14:PLACA ARDUINO NANO .....	18
FIGURA 15: XCTU, VENTANA DE CONFIGURACIÓN.....	19
FIGURA 16: XCTU, VENTANA DE SIMULACIÓN.....	20
FIGURA 17: ENTORNO DE DESARROLLO ARDUINO.....	21
FIGURA 18: DIAGRAMA DE FUNCIONAMIENTO DEL NODO DE MEDIDA.....	22
FIGURA 19: DIAGRAMA DE FUNCIONAMIENTO DEL NODO BASE 1/3 .....	32
FIGURA 20:DIAGRAMA DE FUNCIONAMIENTO DEL NODO BASE 2/3.....	33
FIGURA 21: DIAGRAMA DE FUNCIONAMIENTO DEL NODO BASE 3/3 .....	34
FIGURA 22: INTERFAZ DE USUARIO ASOCIADA AL PROGRAMA PRINCIPAL.....	40
FIGURA 23:SELECTOR DESPLEGABLE DE PUERTO DE COMUNICACIONES .....	41
FIGURA 24: SELECCIÓN DE PUERTO DE COMUNICACIONES, BUCLE 1.....	41
FIGURA 25: SELECCIÓN DE PUERTO DE COMUNICACIONES, BUCLE 2 .....	42
FIGURA 26: PANEL FRONTAL SUBRUTINA CALIBRADO_ACELEROMETRO .....	42
FIGURA 27: PANEL FRONTAL DE LA SUBRUTINA CALIBRADO_RESISTENCIA.....	43
FIGURA 28:PANEL FRONTAL SUBRUTINA DE CÁLCULO DE RECTA DE AJUSTE .....	43
FIGURA 29: CODIFICACIÓN DE PARÁMETROS A ALMACENAR EN ARCHIVO DE TEXTO .....	44
FIGURA 30: PANEL PRINCIPAL SUBRUTINA CAMBIO_TCICLO .....	45
FIGURA 31: DISPOSICIÓN DE PUNTOS DE MEDIDA.....	47
FIGURA 32: DISPOSICIÓN DEL NODO DE MEDIDA .....	49
FIGURA 33: NODO DE MEDIDA .....	49
FIGURA 34: DISPOSICIÓN DEL NODO BASE .....	50

FIGURA 35: GRÁFICAS TEMPERATURA Y HUMEDAD RELATIVA, PRIMERA SERIE .....	51
FIGURA 36: GRÁFICA DE RESISTENCIA, PRIMERA SERIE.....	51
FIGURA 37: GRÁFICAS DE TEMPERATURA Y HUMEDAD RELATIVA, SEGUNDA SERIE .....	52
FIGURA 38: GRÁFICA DE RESISTENCIA, SEGUNDA SERIE .....	52
FIGURA 39: GRÁFICAS DE TEMPERATURA Y RESISTENCIA, TERCERA SERIE.....	53
FIGURA 40: GRÁFICA DE RESISTENCIA, TERCERA SERIE .....	53
FIGURA 41: GRÁFICA DE TEMPERATURA Y RESISTENCIA, CUARTA SERIE.....	54
FIGURA 42: GRÁFICA DE RESISTENCIA, CUARTA SERIE .....	54
FIGURA 43: ESQUEMÁTICO NODO DE MEDIDA .....	59
FIGURA 44: ESQUEMÁTICO PLACA ARDUINO NANO .....	60
FIGURA 45: ESQUEMÁTICO PLACA ARDUINO UNO .....	60
FIGURA 46:LAYOUT PLACA NODO DE MEDIDA.....	62

# LISTA DE TABLAS

---

TABLA 1: CARACTERÍSTICAS ELÉCTRICAS ADXL335 .....	9
TABLA 2: CARACTERÍSTICAS ELÉCTRICAS SENSOR SHT15 .....	10
TABLA 3: ECUACIÓN Y COEFICIENTES DE CONVERSIÓN A TEMPERATURA.....	12
TABLA 4: ECUACIÓN Y COEFICIENTES DE CONVERSIÓN A HUMEDAD RELATIVA. SIN COMPENSAR LOS EFFECTOS DE LA TEMPERATURA (IZQ.) Y COMPENSANDO LOS EFECTOS DE LA TEMPERATURA (DCHA.) .....	12
TABLA 5: ECUACIÓN Y COEFICIENTES PARA EL CÁLCULO DE LA TEMPERATURA DEL PUNTO DE CONDENSACIÓN (°C).....	12
TABLA 6: CARACTERÍSTICAS ELÉCTRICAS MÓDULO XBEE SERIES 2.....	13
TABLA 7: CARACTERÍSTICAS ELÉCTRICAS MÓDULO RELOJ.....	15
TABLA 8: RESULTADOS DEL ESTUDIO PRELIMINAR.....	48



# 1. INTRODUCCIÓN

---

## 1.1 RESUMEN

En el capítulo 1, se presentan los objetivos planteados para la realización del trabajo que se presenta. En éste se pretende diseñar e implementar un sistema de monitorización de medida de humedad basado en un convertidor resistencia-frecuencia, así como verificar su funcionamiento en un entorno real. Para poder llevarlo a cabo, se ha realizado una revisión de artículos científicos respecto a la posibilidad de obtener medidas de resistencia de un muro o pared.

En el capítulo 2, se describen los diferentes elementos hardware que conforman el sistema de monitorización y medida. El sistema diseñado se ha implementado en dos placas Arduino, y está compuesto por dos nodos y una interfaz ejecutable en computadora. El primer nodo es el nodo de medida, que se ha programado para adquirir medidas de tiempo, temperatura, humedad relativa y posición de un plano mediante un acelerómetro. El segundo es el nodo base, nodo que almacena los mensurandos recibidos del nodo de medida en una tarjeta SD. Y la interfaz se emplea para monitorizar y administrar las funcionalidades del sistema. La comunicación entre los elementos de la red se realiza inalámbricamente entre si.

Los capítulos 3 y 4, se dedican a la descripción software, distinguiendo entre herramientas software: XCTU, Arduino y LabVIEW.

En el capítulo 5, se muestran los resultados experimentales que verifican el funcionamiento del primer prototipo desarrollado. Estos resultados sugieren un rediseño del sistema para aislarlo de una componente de ruido electromagnético que introduce un deshumidificador presente en el entorno de verificación del sistema.

En el capítulo 6, se indican las conclusiones extraídas del trabajo realizado y trabajos futuros, ya que se trata de un primer prototipo al que se le pueden incorporar nuevas prestaciones y mejoras. Finalmente, se incluye un apartado de anexos en los que se presentan los esquemáticos de algunos componentes, así como los códigos de programa desarrollados.

## 1.2 MOTIVACIÓN

Esta idea dio comienzo en octubre de 2018 cuando visité a la profesora Silvia Casans para realizar mi trabajo de fin de grado acerca de un tema relacionado con electrónica e instrumentación.

Hablando de varias ideas para realizar el proyecto, y de una serie de problemas estructurales presentes en un inmueble debidos a problemas de humedades, surgió la que sería el eje de este proyecto. Tras consultar diferente bibliografía, [1-4], se comprobó que existía una relación clara y estudiada previamente en diversos trabajos entre resistencia y humedad relativa de una pared. Por ello, la idea que se planteó fue diseñar e implementar un sistema de medida de humedad, y verificar su funcionamiento en un entorno real.

## 1.3 OBJETIVOS

El objetivo de este proyecto es diseñar e implementar un sistema de monitorización aplicado a medidas de resistencia basado en un convertidor resistencia-frecuencia. Para cumplir con el objetivo, se proponen diferentes hitos a alcanzar:

- Medir temperatura
- Medir frecuencia mediante el empleo de una plataforma hardware libre como es Arduino.
- Implementar un circuito de acondicionamiento aplicado a la medida de resistencia.
- Controlar el instrumento mediante el uso de una interfaz desarrollada en LabVIEW. Así como controlar los módulos inalámbricos y las diferentes placas Arduino.
- Desarrollar una plataforma de monitorización y control.
- Desarrollar el software adecuado para la generación de informes.

La implementación del circuito y de los diferentes equipos hardware se describen en el Capítulo 2, en cambio, el desarrollo software se ubica en el Capítulo 3.

## 1.4 PLANIFICACIÓN

Para abordar el proyecto y lograr el objetivo, éste se dividió en seis fases de trabajo: estudio preliminar, familiarización con los dispositivos, desarrollo, pruebas y depuración, adquisición de datos y redacción.

El planteamiento inicial se vio alterado debido a un imprevisto académico, extendiendo la fase de desarrollo dos meses más de lo previsto, y por consiguiente la fecha de entrega de este proyecto.

La duración en meses de cada fase de trabajo y qué se realizó en ellas se expone a continuación. Decir que, por el tipo de trabajo desarrollado en cada una de ellas, hay fases que se ejecutan en paralelo, mientras que otras fases no lo permitieron.

1. Estudio preliminar: se realizaron diferentes series de medida empleando un multímetro de mano para verificar el orden de magnitud de resistencia que presentaba el objeto de estudio.

**DURACION:** octubre de 2018.

2. Familiarización con los dispositivos: se estudiaron y probaron los diferentes sensores que se podían emplear para cumplir con los objetivos. Así como con las diferentes plataformas de desarrollo, LabVIEW, Arduino y XCTU.

**DURACION:** octubre, noviembre y diciembre de 2018.

3. Desarrollo: en esta fase se implementó la componente hardware del proyecto, conformando los diferentes nodos. Y también se desarrolló el software necesario para que los nodos trabajen conjuntamente.

**DURACION:** enero, febrero, marzo, abril, mayo y junio de 2019.

4. Pruebas y depuración: durante esta fase se realizaron pruebas acerca de si se habían integrado correctamente la parte software y hardware.

**DURACION:** julio de 2019.

5. Adquisición de datos: esta fase consistió en establecer el sistema en el entorno seleccionado para realizar el estudio y que almacenara datos durante este tiempo.  
**DURACION**: julio, agosto y septiembre de 2019.
6. Redacción: en esta última fase se elaboraron una serie de borradores que tras su corrección y aprobación que convirtieron en este informe.  
**DURACION**: julio, agosto y septiembre de 2019.

## 2. SISTEMA DE MONITORIZACIÓN IMPLEMENTADO

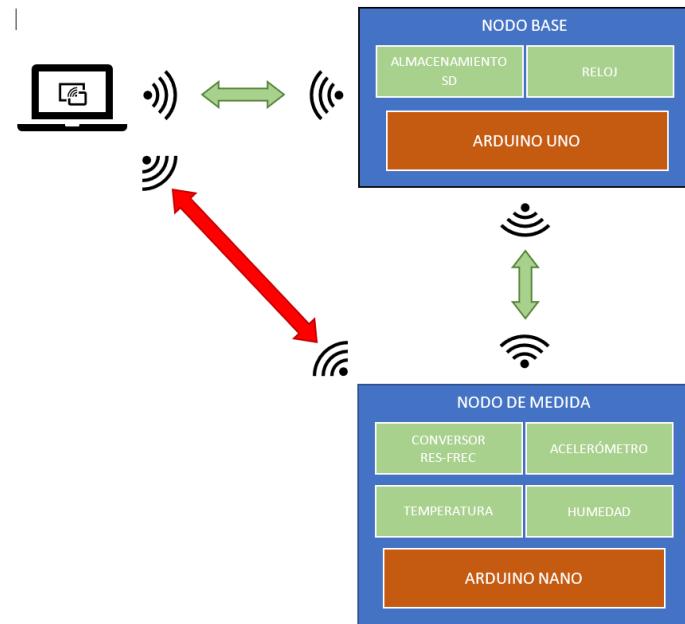


Figura 1: Diagrama de bloques del sistema de monitorización implementado

El esquema de la Figura 1, muestra cómo está planteado el sistema desarrollado. Como puede verse, dicho sistema está compuesto por una computadora, un nodo base y un nodo de medida. Los enlaces de comunicación que se pueden establecer son: entre computadora-nodo base-nodo de medida, representado por las flechas verdes, este es el enlace de funcionamiento habitual del sistema. O puede establecerse el enlace computadora-nodo de medida, representado en rojo, este se emplea para realizar diferentes funciones de monitorización.

También se muestran los diferentes módulos que incorpora cada nodo, así como el propio controlador que se emplea en cada uno. Siendo el controlador del nodo base la placa Arduino UNO, y para el nodo de medida la placa Arduino NANO.

### 2.1 HARDWARE

En esta sección se presenta cada uno de los módulos que componen la parte hardware del sistema implementado. Se ha hecho una distinción tipo de módulos, éstos son: conversor resistencia-frecuencia, sensores, módulos inalámbricos, módulo reloj, módulo tarjeta SD y Arduino.

#### 2.1.1 CONVERSOR RESISTENCIA – FRECUENCIA

El circuito empleado para actuar como conversor resistencia-frecuencia está basado en un oscilador de relajación, que se corresponde con la primera etapa del circuito mostrado en la Figura 2.

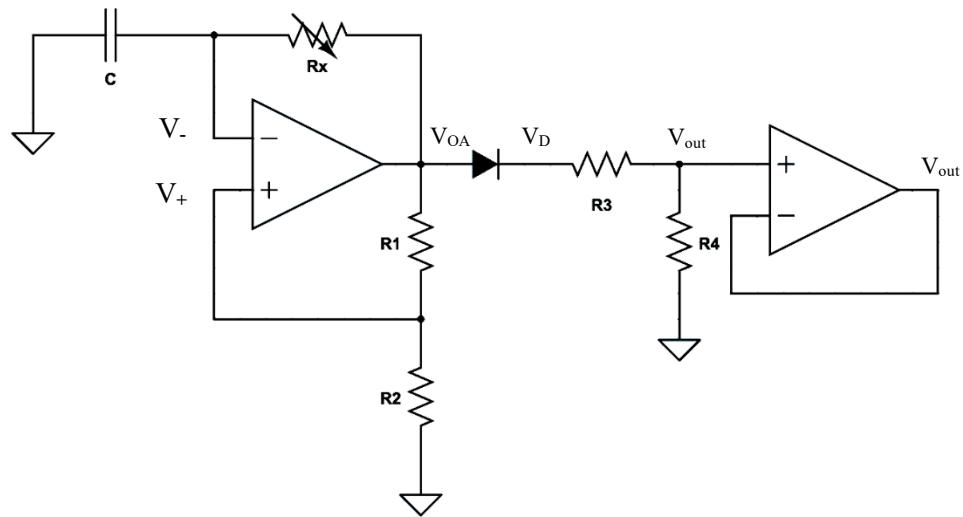


Figura 2: Circuito conversor resistencia-frecuencia

El circuito se ha diseñado para que la resistencia de carga del condensador  $C$  sea la incógnita,  $R_x$ . De modo que si se es capaz de medir el periodo de la señal se obtiene de forma indirecta el correspondiente valor de resistencia.

La Figura 3 muestra el cronograma de funcionamiento del oscilador de relajación asociado a la primera etapa de conversión.

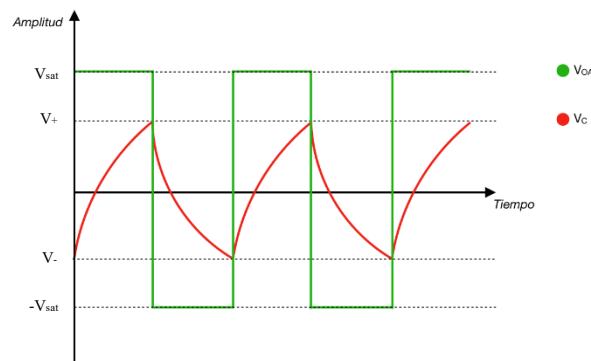


Figura 3: Cronograma de carga y descarga de condensador  $C$  (rojo). Onda cuadrada  $V_{OA}$  (verde)

El principio de funcionamiento de esta configuración es el siguiente:

- El condensador, se carga a través de la resistencia  $R_x$  a la tensión  $V_{sat}$ .
- En el momento que la tensión de las dos entradas del operacional se iguala,  $V_+ = V$ , se invierte la polaridad del circuito, cambiándose el estado de  $V_{OA}$ , que pasa de  $V_{sat}$  a  $-V_{sat}$ .

- En ese instante, el condensador comienza a descargarse hacia  $V_{sat}$ . Y de nuevo, al igualarse el valor de la entrada inversora del operacional con la referencia fijada en la entrada no inversora, se vuelve a invertir la polaridad y comienza a cargarse de nuevo el condensador. Este comportamiento queda reflejado en la Figura 3.

La velocidad con que se lleva a cabo esta carga y descarga, viene determinada por los valores de  $C$  y  $Rx$ . Que, a su vez, coincide con la frecuencia de la onda cuadrada generada,  $V_{OA}$ . Partiendo de la ecuación genérica de carga de un condensador, ecuación 1, y suponiendo que el condensador  $C$  se halla cargado en el instante inicial a  $V_-$ .

Conociéndose también, que, al invertirse la polaridad del circuito por alcanzar valores de pico,  $\pm V_+$ , el condensador comienza la descarga si se hallaba en proceso de carga, o la carga si se encontraba en proceso de descarga. Se ha denominado al periodo de carga, desde  $-V_+$  hasta  $+V_+$ , como tiempo  $t_I$ .

$$V_+ = V_{sat} + (-V_+ - V_{sat}) \cdot e^{-t_I/R_x \cdot C} \quad [1]$$

El valor,  $\pm V_+$ , se corresponde a la expresión analítica de un divisor de tensión en el nudo  $V_{OA}$ . Aplicando esa expresión en la Ecuación 1 y operando, se obtiene la ecuación 2. Expresión que relaciona el tiempo de carga en función de  $Rx$ ,  $C$  y un factor logarítmico.

$$t_I = R_x \cdot C \cdot \ln\left(\frac{2 \cdot R_2 + R_1}{R_1}\right) \quad [2]$$

Considerándose el tiempo de carga del condensador,  $t_I$ , igual al de descarga, puede reescribirse la ecuación 2 de la siguiente forma:

$$Rx = \frac{T_{V_{OA}}}{2 \cdot C \cdot \ln\left(\frac{2R_2 + R_1}{R_1}\right)} \quad [3]$$

La ecuación 3 permite obtener el valor de la resistencia,  $Rx$ , a partir de los valores de los componentes pasivos y del periodo de la señal  $V_{OA}$ .

Aclarar que por diseño se ha establecido que la tensión de alimentación de los amplificadores operacionales del circuito,  $V_{cc}$ , sea de  $\pm 15$  V. Del mismo modo los valores de  $R_1$  y  $R_2$ , que se han fijado a  $12 \text{ k}\Omega$  y  $33 \text{ k}\Omega$ , respectivamente. Y  $C$  de  $68 \text{ nF}$ .

En cuanto a la segunda etapa del circuito, decir que se ha empleado un diodo a modo de rectificador de media onda, dando lugar a una señal,  $V_D$ , cuyo estado alto se corresponde a la diferencia de tensión igual a la resta del valor de la señal  $V_{OA}$  y la tensión del diodo,  $V_d$ , cuando éste se encuentra en conducción. El estado bajo equivale a 0 V. Tras el diodo, se emplea un acondicionamiento que consiste en un divisor resistivo adecuando la amplitud de la señal  $V_D$ . Y finalmente, un amplificador operacional en configuración de seguidor de tensión para acoplar impedancias.

El divisor resistivo está formado por  $R_3$  y  $R_4$ , resistencias que se han diseñado para valores de  $33\text{ k}\Omega$  y  $12\text{ k}\Omega$ , respectivamente. Este divisor ofrece un factor 4:15. En el cronograma de la Figura 4 se muestra el resultado de esta segunda etapa frente a la primera.

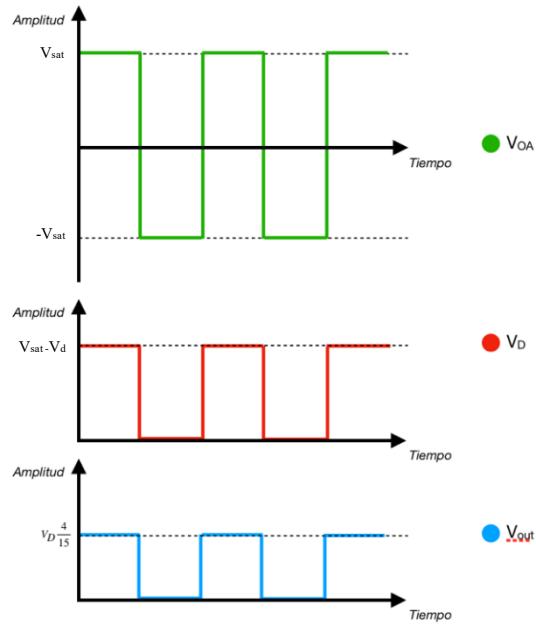


Figura 4: Cronograma de señales  $VOA$ ,  $VD$  y  $Vout$

## 2.1.2 SENSORES

Este subapartado está compuesto por los sensores empleados en la implementación del sistema de medida. Los sensores que se han empleado son acelerómetro y sensor de temperatura y humedad relativa.

- **Acelerómetro**

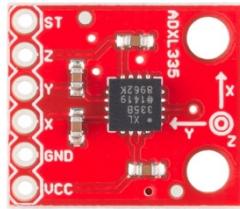


Figura 5: Acelerómetro ADXL335

La Figura 5 muestra el acelerómetro ADXL335, este sensor analógico es capaz de registrar variaciones de posición en cualquiera de sus tres ejes. En base a este comportamiento, presenta dos posibles aplicaciones:

- **Medida de inclinación**, mediante el registro del ángulo de cada plano que forman los tres ejes (XY, XZ e YZ). Esto permite registrar inclinación en paredes, techos, suelos, etc. Datos que, en viviendas o inmuebles antiguos, hay que considerar para la seguridad de aquellos que viven o visiten dicho espacio.
- **Medida de vibración**. Este atributo, podría aprovecharse como motivo de alarma en caso de registrar una fuerte variación repentina en cualquiera de sus ejes. El origen de esta vibración podría deberse a un derrumbe, hundimiento o seísmo. Fenómenos, que condicionan el bienestar de los presentes en un entorno semejante.

Las características eléctricas más relevantes de este sensor se muestran en la Tabla 1, valores extraídos de la hoja de características que proporciona el fabricante, [6].

Para utilizar el sensor e interpretar correctamente su respuesta, es necesario realizar su calibrado, ya que puede presentar ganancias diferentes en cada eje, así como, una tensión de offset distinta para cada uno de ellos.

CARÁCTERISTICAS ADXL335	
V <sub>cc</sub> (V)	1,8 a 3,6

Corriente de alimentación ( $\mu$ A)	350
Rango de operación (°C)	- 40 a 85
Rango de medida (g)	$\pm 3,6$
Sensibilidad (mV/g)	300
0 g X <sub>out</sub> , Y <sub>out</sub> , Z <sub>out</sub> (V)	1,5
0 g OFFSET vs. Temperatura (mg/°C)	$\pm 1$
Deriva térmica sensibilidad (%/°C)	$\pm 0,01$

Tabla 1: Características eléctricas ADXL335

El proceso de calibrado consiste en capturar las lecturas que ofrece el sensor en cada una de sus seis posiciones absolutas, Figura 6. Es decir, en las posiciones donde se aplica la fuerza de la gravedad sobre un único eje, siendo en los otros dos nula.

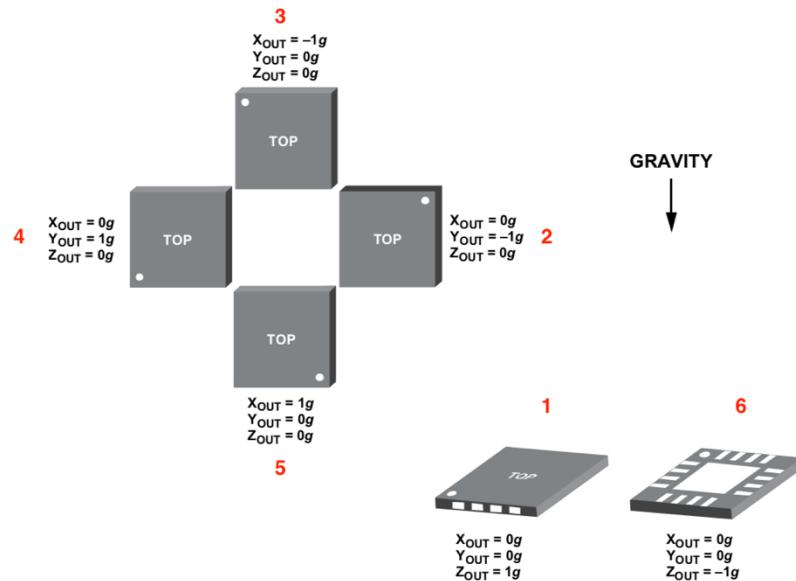


Figura 6: Posiciones absolutas acelerómetro

Una vez adquiridas estas medidas, para calcular la ganancia de cada eje se utiliza la ecuación 4:

$$G = \frac{1}{2} \cdot \frac{A_{+1g} - A_{-1g}}{1g} \quad [4]$$

Los términos  $A$  se corresponden con la lectura de un eje. El subíndice +1g, indica que esa lectura se corresponde a la de percepción de la gravedad como positiva. Y el subíndice que indica percepción de la gravedad como negativa, es -1g.

Empleando las mismas lecturas en la ecuación 5, se obtiene el valor de offset de ese eje.

$$Offset = \frac{A_{+1g} + A_{-1g}}{2} \quad [5]$$

Una vez realizada la calibración y calculados los coeficientes correspondientes a cada eje, para determinar el valor de aceleración de cada uno de ellos, ha de aplicarse la ecuación 6 con los coeficientes del eje que se esté midiendo:

$$A_{\text{compensada}} = \frac{A_{\text{medida}} - \text{Offset}}{G} \quad [6]$$

El valor de las lecturas por defecto se haya expresadas en radianes, para realizar la conversión a grados decimales se han de emplear las ecuaciones [7], [8] y [9].

$$\theta_x = \arctan \left( \frac{A_x}{\sqrt{A_y^2 + A_z^2}} \right) \quad [7]$$

$$\theta_y = \arctan \left( \frac{A_y}{\sqrt{A_x^2 + A_z^2}} \right) \quad [8]$$

$$\theta_z = \arctan \left( \frac{\sqrt{A_x^2 + A_y^2}}{A_z} \right) \quad [9]$$

- Sensor de Temperatura y Humedad



Figura 7: Sensor de temperatura y humedad relativa SHT15

La Figura 7 muestra el sensor digital SHT15, éste permite medir temperatura y humedad relativa del ambiente en el que se encuentre. Las características eléctricas más relevantes de este sensor se presentan en la Tabla 2. Estos valores típicos se han extraído de la hoja de datos proporcionada por el fabricante, [7].

CARACTERÍSTICAS SHT15	
Vcc (V)	2,4 a 5,5
Consumo ( $\mu$ W)	90
Rango de operación ( $^{\circ}$ C)	- 40 a 123,8
Temperatura	Humedad relativa
Resolución ( $^{\circ}$ C)	0,01
Precisión ( $^{\circ}$ C)	$\pm$ 0,3
Resolución (%)	0,05
Precisión (%)	$\pm$ 2

Tabla 2: Características eléctricas sensor SHT15

El protocolo que ha de emplearse para trabajar con este sensor es el *Inter-Integrated Circuit* (I<sup>2</sup>C), desarrollado en 1982 por Philips Semiconductors. Este tipo de comunicación con el sensor se realiza mediante los pines DATA y SCK, [15].

A modo de aclaración, decir que el protocolo de comunicación I<sup>2</sup>C se caracteriza por ser una comunicación maestro-esclavo. Siempre se inicializa la comunicación por parte del maestro, es entonces cuando el esclavo manifiesta algún comportamiento. Se trabaja en línea lógica positiva, es decir, el estado lógico alto se corresponde con un 1 en la línea de comunicación, y el estado bajo con un 0 en la misma línea.

Una vez hechas estas breves aclaraciones relacionadas con el protocolo de comunicación, se procede a presentar el cronograma que se ha de recrear para iniciar la comunicación con el sensor. Figura 8.

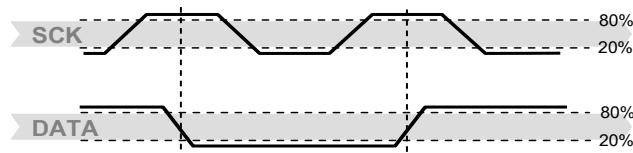


Figura 8: Secuencia de inicio de comunicación con sensor SHT15

Tras esta secuencia, sigue otra, denominada secuencia de instrucción, indicando el tipo de tarea que se solicita. Para ello, se envían 3 bits de dirección, que el fabricante advierte que solamente “000” está permitido, seguidos de 5 bits de comando. Las combinaciones de bits de comando útiles son:

- **00011** para medir temperatura.
- **00101** para medir humedad relativa.
- **00111** para leer el registro *Status*.
- **00110** para escribir le registro *Status*.
- **11110** para resetear el sistema, borrando el registro *Status* y volviendo a los valores por defecto de éste.

Introducida la secuencia de instrucción, en los casos que se solicita un mensurando, es necesario esperar un tiempo para proceder a recoger la medida. Este tiempo varía en función de los bits de resolución empleados, siendo 320 ms para el caso en que se emplean 14 bits para realizar la medida.

Una vez transcurrido ese tiempo, el mensurando está almacenado en la memoria interna del sensor y puede ser recogido. Ha de tenerse en cuenta que las medidas vienen dadas en 2 bytes, y los datos vienen codificados en prioridad de *Most Significant Bit* (MSB) primero. Finalmente, obtenida la lectura, si se requiere una nueva u otro tipo de instrucción, se ha de proceder nuevamente con la secuencia de inicio de comunicación, Figura 8. Y posteriormente, enviar la secuencia de instrucción deseada.

En cuanto al mensurando, en la hoja de datos del fabricante, [7] se facilitan las ecuaciones, y los coeficientes, Figura 9 y Figura 10, con los que traducir los valores leídos en términos de cuentas, a valores físicos como son la temperatura, en grados centígrados, o el porcentaje de humedad relativa del ambiente.

VDD	d <sub>1</sub> (°C)	d <sub>1</sub> (°F)	SO <sub>T</sub>	d <sub>2</sub> (°C)	d <sub>2</sub> (°F)
5V	-40.1	-40.2	14bit	0.01	0.018
4V	-39.8	-39.6	12bit	0.04	0.072
3.5V	-39.7	-39.5			
3V	-39.6	-39.3			
2.5V	-39.4	-38.9			

$$T = d_1 + d_2 \cdot SO_T$$

Tabla 3: Ecuación y coeficientes de conversión a temperatura

$$RH_{\text{linear}} = c_1 + c_2 \cdot SO_{RH} + c_3 \cdot SO_{RH}^2 \quad (\%RH)$$

$$RH_{\text{true}} = (T_{\circ C} - 25) \cdot (t_1 + t_2 \cdot SO_{RH}) + RH_{\text{linear}}$$

SO <sub>RH</sub>	c <sub>1</sub>	c <sub>2</sub>	c <sub>3</sub>
12 bit	-2.0468	0.0367	-1.5955E-6
8 bit	-2.0468	0.5872	-4.0845E-4

SO <sub>RH</sub>	t <sub>1</sub>	t <sub>2</sub>
12 bit	0.01	0.00008
8 bit	0.01	0.00128

Tabla 4: Ecuación y coeficientes de conversión a humedad relativa. Sin compensar los efectos de la temperatura (izq.) y compensando los efectos de la temperatura (dcha.)

En ambos casos, Tabla 3 y 4, las variables denominadas como SO<sub>T</sub> y SO<sub>RH</sub>, hacen referencia a las lecturas de temperatura y humedad relativa, respectivamente, expresadas en términos de cuentas.

Por último, para calcular a qué temperatura se produce la condensación del agua en estado gaseoso presente en el ambiente, ha de emplearse la expresión y coeficientes de la Tabla 5.

$$T_d(RH, T) = T_n \cdot \frac{\ln\left(\frac{RH}{100\%}\right) + \frac{m \cdot T}{T_n + T}}{m - \ln\left(\frac{RH}{100\%}\right) - \frac{m \cdot T}{T_n + T}}$$

Temperature Range	Tn (°C)	m
Above water, 0 – 50°C	243.12	17.62
Above ice, -40 – 0°C	272.62	22.46

Tabla 5: Ecuación y coeficientes para el cálculo de la temperatura del punto de condensación (°C)

### 1.1.3 MÓDULOS INALÁMBRICOS

Este dispositivo, Figura 9, comercializado por *Digi International*, ofrece la posibilidad de establecer una comunicación con otro dispositivo de la misma familia y a su vez establecer redes de comunicaciones, [10].

El protocolo de comunicación utilizado por estos dispositivos es *Zigbee*. Este protocolo, similar a *Bluetooth*, se caracteriza por consumos bajos y por su eficiente fabricación. Actúa en la banda de 2,4 GHz y alcanza una velocidad de transmisión máxima de 250 kbps.

El fabricante proporciona una gran variedad de datos técnicos, [9], acerca del funcionamiento, pero los más relevantes para el proyecto son los reflejados en la Tabla 4.



Figura 9: módulo Xbee series 2

PARÁMETROS ELÉCTRICOS MODULO XBEE S2				
Tensión de alimentación (V)	3 a 3,4			
Frecuencia (MHz)	2,5			
Rango de temperatura (°C)	- 40 a 80			
Corriente (mA)	Emitiendo	Recibiendo	IDLE	Modo bajo consumo
	295	45	15	3,5 $\mu$ A
Alcance en interior (m)	90			
Alcance en línea vista (m)	3200			
Velocidad de transmisión (kbps)	250			

Tabla 6: Características eléctricas módulo Xbee series 2

La configuración de los módulos puede llevarse a cabo también enlazando por puerto serie o modo Application Programming Interface (API), e introducir por la línea de comandos cada uno de los parámetros necesarios para configurar el dispositivo. Tanto se emplee el programa XCTU o la configuración en modo API, se necesita conectar la computadora con el dispositivo, para ello se emplean los adaptadores USB, Figura 10.



Figura 10: Adaptador Xbee-USB

Para configurar cada módulo existe una plataforma software llamada XCTU que permite escribir datos como el nombre, red a la que pertenece, máscara de red, gateway y muchos más indicadores que hacen posible la comunicación. Además, permite verificar que un dispositivo está emitiendo y/o recibiendo, ya que posee una ventana de simulación donde se puede escribir un mensaje, enviarlo y verificar que ha alcanzado el destino, siempre y cuando se hallen los dos dispositivos conectados a la misma computadora.

Para establecer esa comunicación entre módulos XBee, se precisa de mínimo dos dispositivos. Las redes compuestas por estos módulos requieren que al menos un integrante actúe como coordinador, siendo éste el que se encarga de enrutar los paquetes de datos. El otro integrante puede actuar como router, o como dispositivo final. Si actúa a modo de router, crea y mantiene la información de la red para determinar el camino más rápido hasta destino, en el caso de tener una red de mayor complejidad. Y si actúa como dispositivo final, pueden mandar información, pero únicamente a través de un router o de un coordinador.

#### 1.1.4 MÓDULO RELOJ

Para garantizar que, ante un posible apagón o desconexión de la alimentación del nodo base, se mantenga la fecha y hora real se precisa de un reloj. Razón por la que se hace uso de este módulo, el cual cuenta con un adaptador para una batería, logrando que el controlador interno no deje de contabilizar el tiempo pese a que la alimentación cese.



Figura 11: Módulo reloj (Chronodot V2.1)

La configuración de este subsistema se realiza, como puede presuponerse al ver la imagen de éste, Figura 11, mediante comunicación I<sup>2</sup>C, [8]. En cuanto a sus características eléctricas, además de las mostradas en la Tabla 5, decir que para que funcione con total normalidad, el fabricante recomienda que a modo de resistencias de *pull-up*, R<sub>1</sub> y R<sub>2</sub>, se empleen valores de 10 kΩ.

CARACTERÍSTICAS ELECTRICAS MODULO CHRONODOT V2.1	
Tensión de alimentación (V)	2,3 a 5,5
Rango de temperatura (°C)	- 40 a 85
Corriente (μA)	200
Precisión (ppm)	± 3,5

Tabla 7: Características eléctricas módulo reloj

### 1.1.5 MÓDULO TARJETA SD

El módulo SD (Wireless SD Shield), Figura 12, se monta superpuesto sobre la placa Arduino UNO, de manera que quedan disponibles todos los pines digitales, ya que cuenta con conexiones que se superponen con las de Arduino UNO, [11].

Las características que desencadenaron su uso en el proyecto son que además de contar con un puerto para conectar tarjetas microSD, cuenta con un zócalo preparado para un módulo de comunicación con encapsulado XBee. Además, cuenta con un interruptor que comuta la comunicación serie con la placa a través de un enlace directo por cable, posición USB. O a través del módulo de comunicación, posición MICRO, obsérvese en Figura 12.

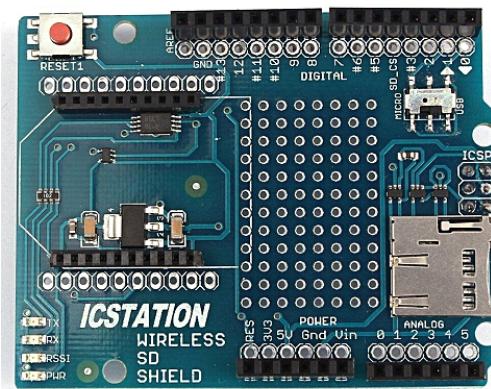


Figura 12: Módulo de almacenamiento SD (Wireless SD Shield)

Este protocolo de comunicación utilizado por la tarjeta SD, está formado por cuatro líneas de comunicación, la línea de reloj o SCLK, la línea CS, y las de datos *Master Output Slave*

*Input* (MOSI) y *Master Input Slave Output* (MISO), estas dos últimas son las que contienen los datos propios de un enlace de comunicaciones.

En la placa Arduino UNO, estas líneas se ubican en los pines digitales, concretamente, SCLK en el pin 13, MISO en el 12 y MOSI en el 11. Por lo que quedan reservados a este dispositivo sin poder hacer uso de ellos otro dispositivo.

La única particularidad que ha de tenerse en cuenta a la hora de emplear este módulo es que, por un lado, el formato de datos de la tarjeta de datos ha de ser FAT32 o ExFAT. Por otro, ha de averiguararse si el pin *Chip Select* (CS) del módulo, se corresponde con el del dispositivo de control sobre el que se monta. En este caso, el dispositivo de control es Arduino UNO, éste tiene este pin ubicado en el pin digital número 10, y el módulo SD lo tiene asignado en el el pin 4, por lo que se debe puentejar los pines digitales 4 y 10. La función de la línea CS del protocolo de comunicación *Serial Peripheral Interface* (SPI), [16], permite al dispositivo que cumple el rol de maestro, en este caso Arduino UNO, excitar al esclavo, módulo SD, para que este se active.

## 1.1.6 ARDUINO

Indicar que en este trabajo se ha desarrollado e implementado un primer prototipo donde el tiempo de desarrollo es limitado. Debido a ello se eligió como opción adecuada a la funcionalidad del proyecto los dispositivos de la familia Arduino. Ya que son versátiles y rápidamente configurables para desarrollar un cometido, permitiendo reducir el tiempo de desarrollo de un dispositivo de control.

Un control elaborado mediante microcontroladores, a priori, tiene un coste más reducido, pero el tiempo necesario para diseñar un sistema de control partiendo de cero es mayor. Y aunque muchas de las funcionalidades que ofrece Arduino se ven desaprovechadas o incluso obviadas, al encontrarse el proyecto en fase de desarrollo, prima que tengan un comportamiento correcto los diferentes subsistemas de sensado, comunicación y la interfaz gráfica para interconexión persona-equipo.

- Arduino UNO

La función de este dispositivo,[12], empleado como control en el nodo base, es actuar como coordinador entre una red de nodos de medida, a la que proporcione instrucciones, y la interfaz de usuario en una computadora. Además de administrar los módulos conectados a esta placa.



Figura 13: Placa Arduino UNO

Este dispositivo, Figura 13, está basado en el microprocesador ATmega328P,[5], y presenta una serie de características que hacen que se haya optado por él. Por un lado, en cuanto a parámetros electrónicos referentes a su funcionamiento, el fabricante recomienda que la tensión de alimentación del microcontrolador nunca supere los 5 V. Para alimentarlo, se plantean tres opciones, mediante la conexión directa al pin  $V_{in}$  con esta tensión. La segunda es empleando el cable de comunicación con una computadora, de USB mini B a USB tipo 2. Y la tercera opción es empleando un cable de alimentación de corriente continua de 12 V. Por otro lado, dispone de catorce pines configurables como salidas y entradas digitales, de las cuales seis, las marcadas con una virgulilla, “~”, junto al número de pin, permiten el uso de señales *Pulse Width Modulation* (PWM). Cuenta también con seis entradas analógicas. Otros pines de interés son:

- **AREF**, permite ajustar el rango de acción del *Analog-Digital Converter* (ADC) de 10 bits de las entradas analógicas, por defecto se haya en 5 V. Aplicando una tensión diferente y que no supere la asignada por defecto se ajustará el rango del convertidor. Esta referencia, también se puede ajustar por software a 1,1 V.
- **IOREF**, suministra la tensión de estado lógico alto al que trabajan las entradas y salidas digitales.
- **RX y TX**, son los pines necesarios para la comunicación serie.
- **Pines 2 y 3**, son pines digitales habilitados para las interrupciones externas. Este tipo de subrutinas, pueden configurarse para ser activadas en función de si se recibe en estos pines un estado alto, un estado bajo, un cambio de estado, un flanco de subida o un flanco de bajada.
- **Pines A4 y A5**, son las dos líneas de comunicación necesarias en el protocolo I<sup>2</sup>C. La línea de datos (SDA) en el A4 y la línea correspondiente al reloj (SCL) en A5.

Para programar el dispositivo, el fabricante suministra un entorno de programación gratuito llamado Arduino, el cual se describirá en el apartado 3.2, en el que probar los diferentes programas elaborados. Este software, suministra código ejemplo para comprender el funcionamiento del dispositivo. Así como la posibilidad de compilar el código en busca de errores antes de cargar el programa. También ofrece la posibilidad de interactuar con el programa abriendo una ventana de comunicación del puerto serie.

- **Arduino NANO**

La función de este dispositivo es actuar como dispositivo de control en el nodo de medida. Encargándose de atender las solicitudes del nodo base, y de coordinar los sensores que forman este nodo para realizar las medidas de los parámetros y enviarlas.

El dispositivo Arduino NANO, Figura 14, presenta prácticamente las mismas características que Arduino UNO. Pero, debido a su reducido tamaño, algunos aspectos se ven modificados, [13]. Por ejemplo, el número de pines digitales, siendo doce en lugar de catorce como en la placa Arduino UNO.

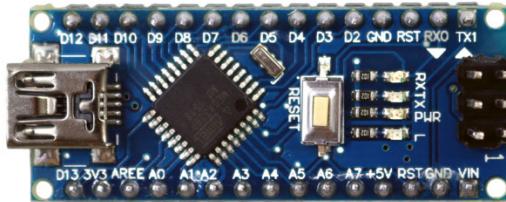


Figura 14:Placa Arduino NANO

La forma de programar el comportamiento de este dispositivo es mediante la plataforma que proporciona el fabricante. Para ello, se precisa es una computadora, un cable USB a mini-USB. Una vez programado, alimentándolo mediante los pines analógicos  $V_{in}$  y  $GND$ , ya está listo para funcionar.

# 3. SOFTWARE

Este capítulo está compuesto por cuatro apartados, el 3.1 está dedicado a presentar el programa empleado para configurar los módulos Xbee. El apartado 3.2 describe el entorno de desarrollo en el que se ha desarrollado el código para las placas Arduino. En los apartados 3.3 y 3.4 se describen los códigos de programación desarrollados diferenciando entre nodo de medida y el nodo base. Se describen: Librerías, funciones adicionales del programa, declaración de variables, configuración de valores por defecto y bucle del programa. En cada apartado se presenta el algoritmo de funcionamiento del nodo correspondiente y posteriormente se abordan los diferentes bloques con mayor detalle.

## 3.1 ENTORNO DE PROGRAMACIÓN XCTU

En este apartado se realiza una breve explicación acerca del programa XCTU empleado para configurar los dispositivos de comunicación inalámbrica Xbee.

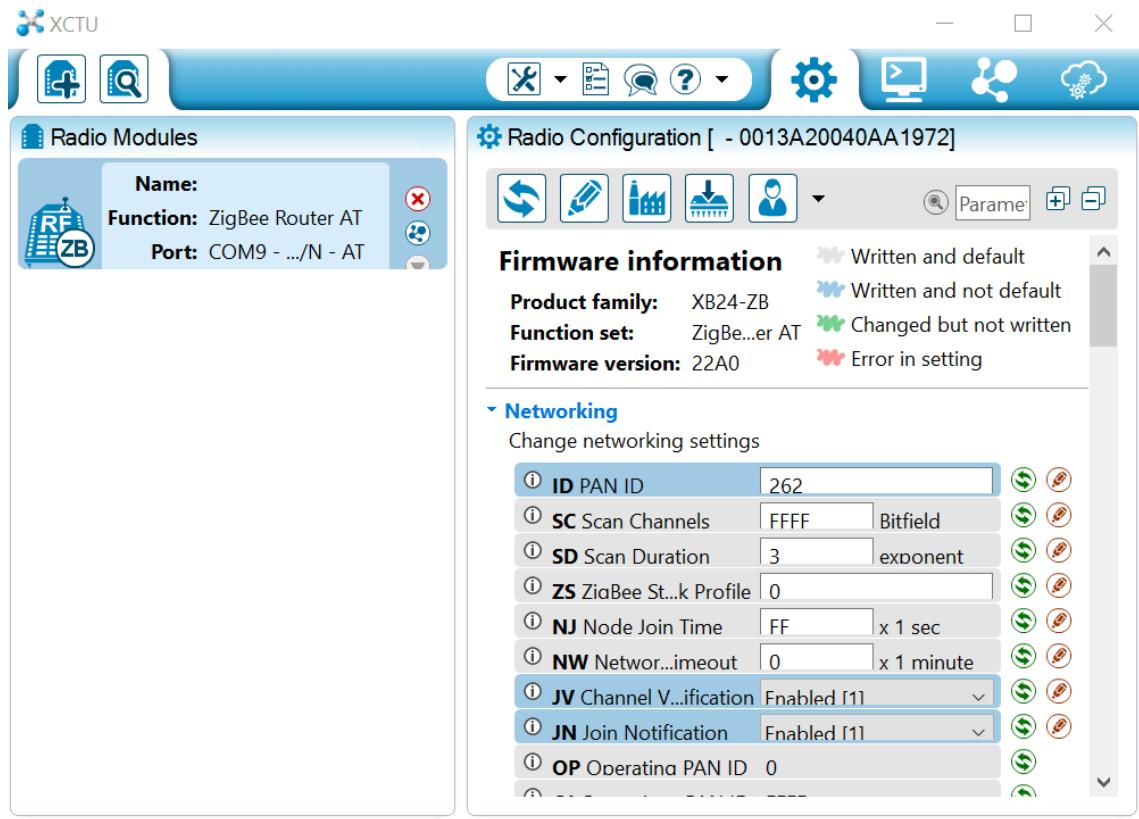


Figura 15: XCTU, ventana de configuración

Este programa permite configurar los dispositivos Xbee en función del rol que vayan a cumplir en la red de comunicación planteada. Estos roles son *End device*, *Router* y *Coordinator*. En cualquier caso, el menú de configuración “Radio Configuration” que se muestra en la Figura 15 es muy similar. En ese menú se puede configurar y modificar los parámetros de red. Por ejemplo, en el proyecto se ha establecido que la dirección de red *PAN ID* sea 262.

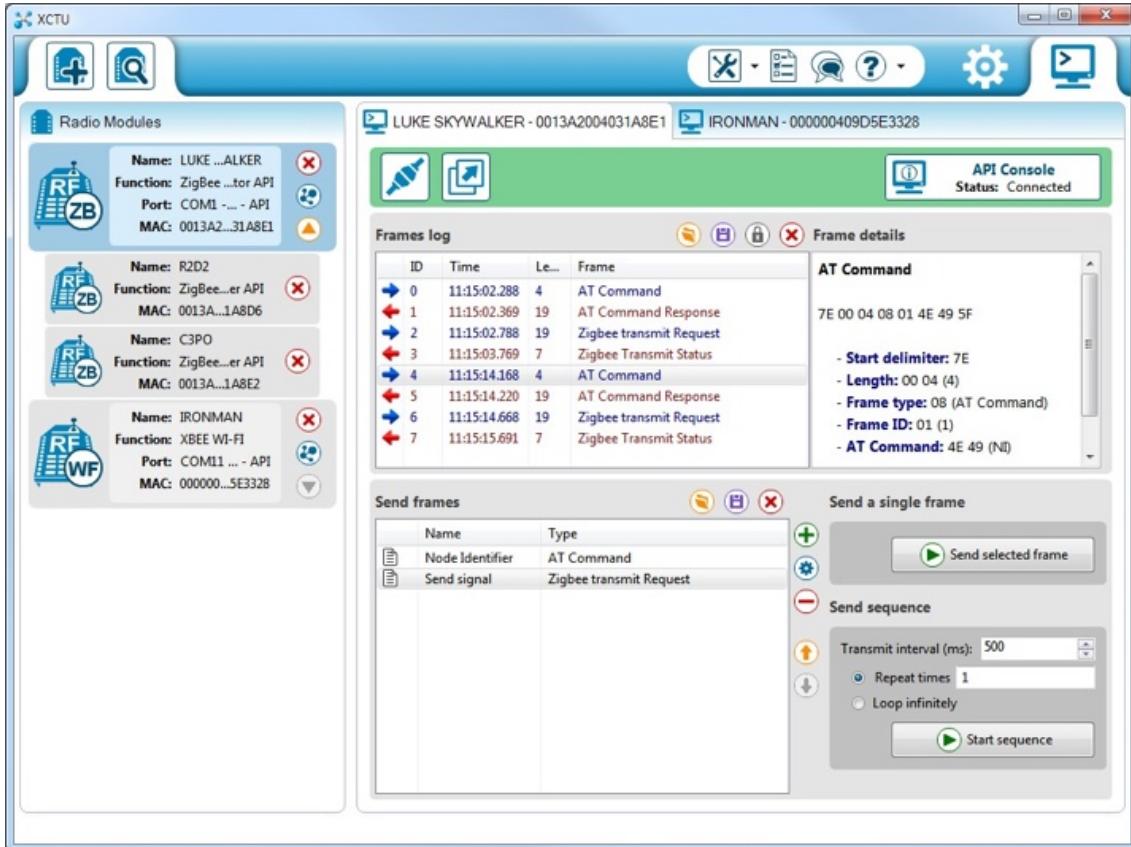


Figura 16: XCTU, ventana de simulación

También permite realizar una simulación para verificar que se han asignado los parámetros correctamente y que la comunicación se ha establecido, véase Figura 16.

### 3.2 ENTORNO DE PROGRAMACIÓN ARDUINO

En esta sección se presenta el programa Arduino empleado para implementar el código de funcionamiento del nodo de medida y nodo base.

Este entorno de desarrollo está basado en un compilador de lenguaje de programación C++,[14]. C cualquier programa desarrollado en esta plataforma ha de estructurarse en dos funciones principales, *setup()* y *loop()*, véase Figura 17. En la primera función, *setup()*, generalmente se localizan las líneas de código necesarias con los valores por defecto para poder ejecutar la función *loop()* sin ningún problema. La segunda función, *loop()*, está destinada a contener las líneas que determinan el funcionamiento del programa desarrollado.

El programa permite depurar el código escrito en función de la placa Arduino sobre la que se vaya a ejecutar, esta función se encuentra en el menú *Tools* y en el apartado de *board* ha de seleccionarse la empleada.

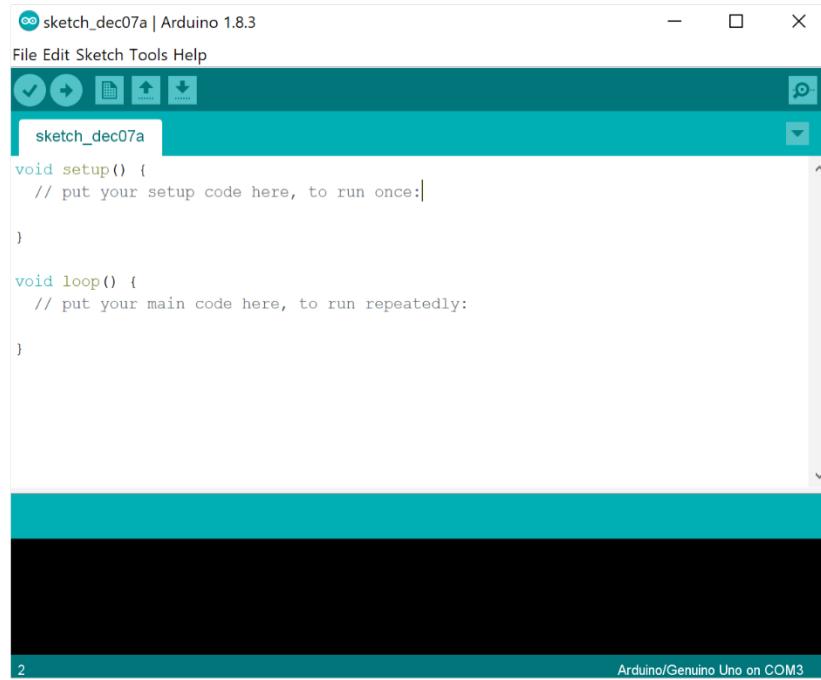


Figura 17: Entorno de desarrollo Arduino

Del mismo modo, puede seleccionarse el puerto por el que se comunican computadora y placa Arduino. Esta función se encuentra en el directorio *Tools/port*. También ofrece la posibilidad de interactuar con una placa Arduino que ejecute un programa, mediante una ventana de diálogo. Ésta emerge al clicar sobre el ícono azul que contiene una lupa, ubicado en la esquina superior derecha.

### 3.3 PROGRAMACIÓN NODO DE MEDIDA

La Figura 21 muestra el algoritmo de funcionamiento asociado al nodo medida. Seguidamente se describen los diferentes bloques que forman el programa desarrollado.

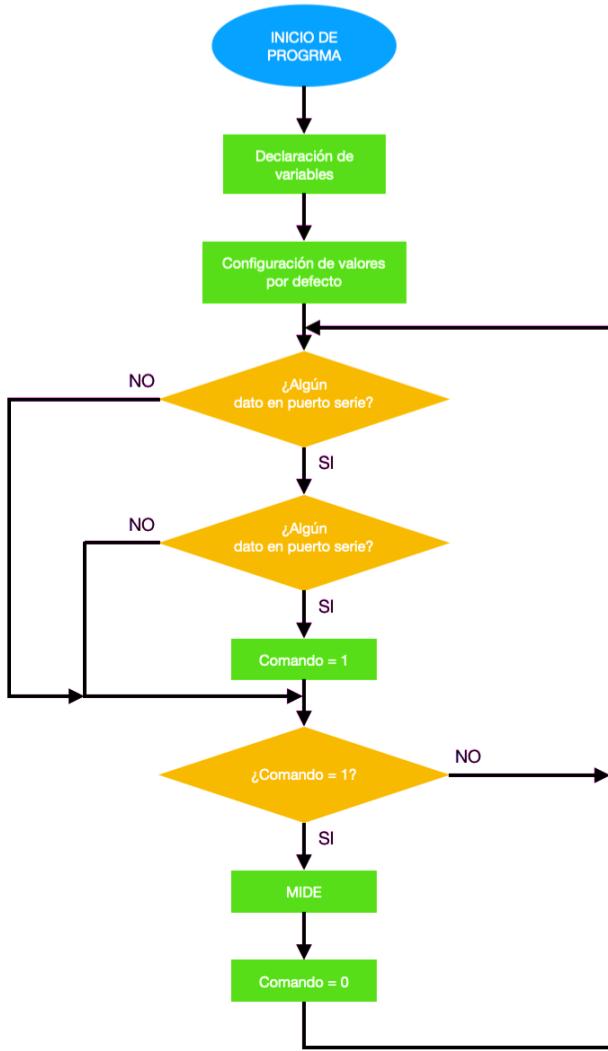


Figura 18: Diagrama de funcionamiento del nodo de medida

### 3.3.1 DECLARACIÓN DE VARIABLES

El código mostrado a continuación corresponde a las diferentes variables utilizadas en el nodo medida. Cabe distinguir entre variables de programa y variables asociadas a los diferentes sensores incluidos en el nodo medida (acelerómetro, sensor de humedad y temperatura y conversor resistencia-frecuencia).

- Variables de programa: En esta sección, por un lado, se ubica la principal variable, *comando*, este contenedor de tipo int, es el encargado de almacenar la instrucción recibida del nodo base, en caso de recibir un “1”, se procede a la medida, en cualquier otro caso, no se realiza ninguna tarea.

Por otro lado, se instancian otros dos tipos de variables que tienen como objeto medir la misma magnitud, tiempo. Estas variables son empleadas en las subrutinas de interrupción, y su función es almacenar una medida de tiempo, de ahí que se declaren como *unsigned*, ya

que no tiene sentido hablar de tiempo negativo. El motivo de declararlas como *volatile*, es debido a la frecuencia con la que sus valores van a ser actualizados.

```
int comando;      //X - NO HACE NADA
                  //1 - LECTURA DE MENSURANDOS
volatile unsigned long tn;
volatile unsigned long t0;
```

- Variables del acelerómetro: Se declaran tres constantes de tipo entero, *Xpin*, *Ypin* y *Zpin*, almacenan a qué pin analógico se conecta cada línea de mensurando correspondiente a un eje del acelerómetro. Se declaran de tipo constante, *const*, debido a, que los pines a los que se conectarán las líneas de lectura de los ejes no cambiarán.

Decir que la primera letra de cada variable hace referencia al eje que se pretende leer. Por ejemplo, puede entenderse que la variable *Xpin* procesa la información del eje X del acelerómetro a través del pin analógico 1, *A1*.

```
const int Xpin = A1;
const int Ypin = A2;
const int Zpin = A3;
```

- Variables del sensor de temperatura y humedad: Este grupo de variables contiene la información del sensor SHT15. Las constantes *ClkPin* y *DataPin*, se declaran de este tipo, porque al igual que en el caso del acelerómetro, se va a hacer uso exclusivo de estos pines para establecer las líneas de comunicación del protocolo I<sup>2</sup>C. Por lo tanto, el pin digital 4 se empleará para la señal de reloj, y el 5 para la señal de datos. Las dos constantes restantes albergan el comando a enviar a dicho sensor para que realice una medida u otra. La razón de declararlas de tipo constante es que el fabricante indica en la hoja de datos, [7], que los comandos para solicitar las medidas son siempre los mismos para cada magnitud. Siendo el valor decimal 3 para temperatura y 4 para humedad.

En cuanto a las variables *val* y *lectura*, se utilizan para realizar operaciones con las lecturas del sensor. Sin embargo, la variable *ack* está relacionada con el correcto funcionamiento del sensor, tomando valor “1” cuando no se ha medido correctamente. Y tomando valor “0” cuando la medida ha sido satisfactoria.

```
const int ClkPin = 4;
const int DataPin = 5;
const int ComandoTemp = 3;
const int ComandoHum = 4;
int val, ack, lectura;
```

- Variables del conversor resistencia-frecuencia: Finalmente, de nuevo se instancian variables de tipo *volatile*, y su función es almacenar la medida de tiempos. *TH*, para la medida de tiempo en estado alto de la señal cuadrada cuyo periodo sirve para calcular la resistencia. *TL*, para contabilizar el tiempo que permanece la misma señal en estado bajo.

```
volatile unsigned int TH;
volatile unsigned int TL;
```

### 3.3.2 CONFIGURACIÓN DE VALORES POR DEFECTO – SETUP()

Esta primera función tiene como propósito contener la información de los parámetros por defecto. De modo que, si se resetea el dispositivo, tenga predefinidos los valores básicos para el correcto funcionamiento del programa.

```
void setup()
{
Serial.begin(9600);
analogReference(DEFAULT);
pinMode(Xpin,INPUT);
pinMode(Ypin,INPUT);
pinMode(Zpin,INPUT);
comando = 0;
}
```

En primer lugar, se inicializa el puerto serie y lo habilita para la comunicación, concretamente a una velocidad de 9600 baudios. Esta velocidad es variable, pudiéndose trabajar a velocidades mayores.

En la segunda línea, se hace la función predefinida que asigna el rango de trabajo de los ADC de los puertos analógicos, en este caso se determina que sea el valor por defecto, para ello, se le pasa como parámetro la palabra reservada *DEFAULT*, por lo que el rango de trabajo será de 0 V a 5 V. Si en lugar de *DEFAULT*, se escribe *INTERNAL*, se ajusta la tensión limitante a 1,1V. Y en el caso de que se desee fijar manualmente la tensión de referencia, la cual se aplicaría sobre el pin *AREF*, habrá de escribirse como referencia la palabra reservada *EXTERNAL*.

Las siguientes tres instrucciones, sirven para determinar el modo de funcionamiento de los pines analógicos por los que se recibirá las tensiones correspondientes al ángulo de cada eje del acelerómetro. Como puede verse, además de indicarle qué pin, empleando la variable correspondiente en cada caso, se les asigna la funcionalidad de que sean entradas, mediante la palabra reservada *INPUT*. Si se necesitara emplear un pin analógico como salida habría de escribirse en este lugar la palabra *OUTPUT*. Y en el caso de necesitar una entrada con resistencia de polarización, existe la posibilidad de realizarse sin la necesidad de emplear resistencias de pull-up físicas. Para ello basta con escribir *INPUT\_PULLUP*.

Finalmente, se encuentra la línea que asigna el valor por defecto a la variable de instrucciones, *comando*.

### 3.3.3 BUCLE DE PROGRAMA – LOOP()

La función principal del programa desarrollado designada como *loop()*, en ella se plasma el comportamiento tal y como se requiere para el correcto funcionamiento del nodo de medida de acuerdo con los criterios de diseño.

Estos criterios establecen que, cada vez que el nodo base lo solicite, éste ha de realizar una medida de todos los mensurandos y enviarlos vía puerto serie. El formato establecido para la cadena de datos enviada se corresponde con un primer indicador de qué se está midiendo (X, Y, Z, Temp, Hum, TH, TL), seguido del carácter “;”, de esta forma, al exportarlo a

formato `.csv`, se emplea como separador de columna. Tras esta cabecera, se envía la lectura correspondiente al mensurando seguido de nuevo del carácter “;”, esto se lleva a cabo con el propósito de encadenar medidas del mismo modo.

Por ejemplo, la cadena de medidas correspondiente al acelerómetro será:

X;LecturaEjeX;YLecturaEjeY;Z;LecturaEjeZ;

Tras esta información, se envía la correspondiente a las lecturas de temperatura y humedad relativa. Siendo los identificadores *Temp*, para la medida de temperatura, y *Hum*, para la humedad relativa del ambiente. Por tanto, la cadena tendrá la forma:

Temp;LecturaTemperatura;Hum;LecturaHumedad;

Finalmente, cierran la línea de datos las medidas temporales de la señal cuyo periodo permite calcular el valor de resistencia del material, para ello, se emplea a modo de distintivo *TH* para el tiempo en estado alto, y *TL* para el tiempo en estado bajo. Esta última fracción de la línea de información presenta la estructura:

TH;LecturaEstadoAlto;TL;LecturaEstadoBajo

Para realizar la medida del período se utiliza una interrupción que requiere el uso de tres parámetros. Para que su ejecución sea correcta, en primer lugar, se indicará el número de interrupción que se está configurando, 0 para la interrupción asociada al pin digital número 2, y 1, como es el caso, para la asociada al pin digital 3. En segundo lugar, se debe indicar el nombre asociado a la interrupción, en el caso del código desarrollado, *flanco\_subida*. Para configurar cualquiera de los modos posibles debe indicarse como tercer parámetro en la función *attachInterrupt()* una de las siguientes palabras reservadas: *HIGH*, *LOW*, *RISING* o *FALLING*.

El código que se muestra a continuación corresponde al bucle de programación:

```
void loop() { //INICIO DE PROGRAMA
attachInterrupt(1, flanco_subida, RISING);
if(Serial.available()>0)
{
if((Serial.peek() == 49))
{
comando = Serial.read();
}
}
```

En primer lugar, se encuentra la instancia de la subrutina de interrupción 1, correspondiente a un pin digital 3. Ésta se configura para activarse en el momento que se detecte un flanco de subida en la señal de ese pin digital. Ya se mencionaron anteriormente los diferentes eventos que activan las subrutinas de interrupción, estado lógico alto, estado bajo, flanco de subida y flanco de bajada.

Posteriormente, se encuentra la primera estructura condicional, donde se evalúa, mediante la función *Serial.available()*, si en el puerto serie hay información para ser procesada. Esta

función, toma el valor de “-1” cuando el buffer se encuentra vacío. En cambio, cuando hay bytes para leer devuelve como dato el número total de ellos. En el condicionante anidado, se comprueba si el byte de lectura del puerto serie se corresponde con el valor “49” de la tabla ASCII, el cual equivale al valor decimal “1” y se corresponde con la instrucción de medida, de ser así, se le atribuye este valor a la variable comando.

La diferencia entre la función *Serial.peek()* y *Serial.read()* es que la primera consulta el valor del primer byte mientras la segunda lo lee, y por lo tanto lo elimina del buffer.

El siguiente código corresponde a la obtención de la información asociada al acelerómetro:

```
if(comando == 49) //TECLA – 1
{
//LECTURA DE MENSURANDOS //ACELERÓMETRO
Serial.print("X:"); Serial.print(analogRead(Xpin)); Serial.print(";");
delay(2);
Serial.print("Y:"); Serial.print(analogRead(Ypin)); Serial.print(";");
delay(2);
Serial.print("Z:"); Serial.print(analogRead(Zpin)); Serial.print(";");
delay(2);
```

En este fragmento se evalúa si la variable comando es 1. Además, se encuentra la primera fracción de la cadena de medida total, en concreto la correspondiente a las medidas del acelerómetro. Éstas se llevan a cabo leyendo la cantidad de cuentas correspondiente a cada eje, para ello se emplea la función nativa del entorno de programación *analogRead()*, esta precisa como parámetro el valor del pin analógico que se va a leer.

Es recomendable que entre lecturas de pines analógicos haya un pequeño fragmento de tiempo de espera para evitar errores de conversión por parte del ADC, con unos pocos milisegundos es suficiente para que éste se recupere de una medida anterior. Es por ello que se han incluido en el código los retardos de 2 ms.

El siguiente fragmento de código, representa cómo se lleva a cabo la adquisición de los mensurandos correspondientes a la temperatura y humedad relativa al ambiente. Además de su envío por puerto serie.

```
//TEMPERATURA
sendCommandSHT(ComandoTemp, DataPin, ClkPin);
waitForResultSHT(DataPin);
val = getData16SHT(DataPin, ClkPin);
skipCrcSHT(DataPin, ClkPin);
Serial.print("Temp,");Serial.print(val);Serial.print(";");
val=0;
//HUMEDAD
sendCommandSHT(ComandoHum, DataPin, ClkPin);
waitForResultSHT(DataPin);
val = getData16SHT(DataPin, ClkPin);
skipCrcSHT(DataPin, ClkPin);
Serial.print("Hum,");Serial.print(val);Serial.print(";");
val=0;
```

Las nuevas líneas que llaman la atención son las que contienen las funciones *sendCommandSHT()*, *waitForResutlSHT()*, *getData16SHT()* y *skipCrcSHT()*. Estas funciones, contienen el código necesario para iniciar la comunicación con el sensor, solicitar una medida, esperar hasta que ésta se realice, y una vez recibida, expresarla en escala decimal, separando la parte entera de la parte decimal. Posteriormente, se analizarán en profundidad en el apartado de funciones de programa adicionales.

Finalmente, en el último fragmento del programa se envía la última cadena de medidas. El código es el siguiente:

```
//LECTURA DE RESISTENCIA
Serial.print("TH;"); Serial.print(TH); Serial.print(";");
Serial.print("TL;"); Serial.print(TL); Serial.println("");
}
comando = 0;
}//FIN DE PROGRAMA
```

Se corresponden con las relacionadas con el periodo de la señal proveniente de medir la resistencia del material aplicando corriente alterna. Como puede verse, una vez termina el ciclo de medida se vuelve a reasignar el valor “0” a la variable *comando*, de esta forma se garantiza que hasta que no se vuelva a recibir la orden no se realizan más medidas, por lo tanto, el sistema permanece en reposo a la espera de una nueva instrucción.

### 3.3.4 SUBRUTINAS DE INTERRUPCIÓN

En este subapartado, se explica la configuración de las interrupciones para lograr una medida de tiempo en estado alto y tiempo en estado bajo. Antes de dar comienzo a analizar el código, realizar una breve explicación acerca de la única función nativa del entorno empleada, *micros()*. Esta función no requiere que se le pasen parámetros para realizar su cometido, una vez invocada, devuelve el número de microsegundos que han transcurrido desde que el programa ha comenzado a funcionar.

```
/*INTERRUPCIONES*/
void flanco_subida () {
t0 = micros();
attachInterrupt(1, flanco_bajada, FALLING);
}

void flanco_bajada (){
tn = micros();
TH = tn - t0;
t0 = tn;
```

```

attachInterrupt(1, flanco_subida2, RISING);
}

void flanco_subida2 () {
tn = micros();
TL = tn - t0;
attachInterrupt(1, flanco_bajada, FALLING);
}

```

A simple vista, llama la atención que la subrutina declarada en *Setup()*, *flanco\_subida()*, contenga en su interior una nueva declaración de subrutina. Esto es debido a que la metodología seguida para realizar la medida es una actualización del evento que activa la subrutina de interrupción 1.

Puede verse que la variable *t0* registra el tiempo de referencia la primera vez que se detecta un flanco de subida. Posteriormente, se reconfigura la interrupción 1 de forma que, se la instancia bajo el nombre de *flanco\_bajada()*, y seleccionando la excitación en flanco de bajada. Una vez se detecta el flanco de bajada y se activa la nueva subrutina de interrupción 1, *flanco\_bajada()*, se vuelve a registrar el tiempo actual en la variable *tn*. De forma que ya se puede realizar el cálculo del tiempo en estado alto de la señal, para ello se calcula la diferencia entre *tn* y *t0*, diferencia que se almacena en la variable *TH*. Ahora como el objetivo es determinar el tiempo de estado bajo de la misma señal, se toma el tiempo *tn* como referencia para futuros cálculos, por ello que la variable *t0* toma este valor. Con objeto de lograr la medida de ese periodo, se vuelve a instanciar la subrutina de interrupción 1 bajo el nombre, en este caso, de *flanco\_subida2()*. El evento que condiciona su ejecución es la detección de un flanco de subida.

Del mismo modo que en la subrutina *flanco\_bajada()*, en *flanco\_subida2()* se vuelve a solicitar el tiempo en primera instancia de modo que permita calcular la diferencia entre *tn* y *t0*, diferencia que se corresponde con el tiempo que ha transcurrido la señal en estado bajo, y que se almacena en la variable *TL*. Una vez calculada y asignada la medida, se vuelve a instanciar la subrutina de interrupción, volviendo al punto de origen, *flanco\_bajada()*.

Resumiendo, el método empleado para determinar el periodo de la señal es cíclico. Es decir, la subrutina de interrupción 1 se reajusta cada vez que se produce un evento, reconfigurándose para buscar otro tipo de evento.

Para garantizar el continúo funcionamiento, se instancia la subrutina de interrupción 1 en el estadio inicial en cada ciclo del programa principal, es decir, se instancia *flanco\_subida()*. De esta forma, se asegura que en caso de error no se quede el sistema bloqueado en una fase, se vuelve a iniciar el ciclo en esta función.

### 3.3.5 FUNCIONES DEL PROGRAMA ADICIONALES

En este subapartado se va a proceder a analizar una por una las funciones adicionales empleadas en el programa principal asociadas al sensor de temperatura y humedad SHT15. Se distinguen un total de cinco funciones adicionales, éstas son: *shiftIn(int dataPin, int clockPin, int numBits)*, *sendCommandSHT(int command, int dataPin, int clockPin)*,

`waitForResultSHT(int dataPin), getData16SHT(int dataPin, int clockPin) y skipCrcSHT(int dataPin, int clockPin).`

- Función `shiftIn(int dataPin, int clockPin, int numBits)`: Tiene como propósito realizar la conversión de escala binaria a escala decimal de la medida proporcionada por el sensor SHT15. Para ello, además del pin de datos y pin de reloj, precisa del número de bits de extensión que contiene la palabra. Este último dato es la condición de salida del bucle que ejecuta la conversión. Para llevarla a cabo, se procede de la siguiente manera: cada bit de datos se envía durante el estado alto de la señal de reloj, por lo tanto, el método de lectura consiste en enviar al sensor un estado alto de reloj, leer el bit y cambiarlo a base decimal, y finalmente, escribir estado bajo en la señal de reloj para proceder con el siguiente. Una vez completado el ciclo de lectura, se devuelve como resultado de la función el valor de esa lectura.

```
int shiftIn(int dataPin, int clockPin, int numBits)
{
    int ret = 0; for (int i = 0; i < numBits; ++i)
    {
        digitalWrite(clockPin, HIGH);
        ret = ret * 2 + digitalRead(dataPin);
        digitalWrite(clockPin, LOW);
    }
    return (ret);}
```

- Función `sendCommandSHT(int command, int dataPin, int clockPin)`: Tiene como objetivo establecer comunicación con el sensor SHT15 y solicitar medir la temperatura o la humedad. Es por ello que además de los pines de comunicación, se le proporciona el comando de medida, *command*, que puede tomar el valor 3 o 4, solicitando que se mida la temperatura o la humedad respectivamente.

Las primeras líneas de código, concretamente hasta la línea 7, inclusive, están dedicadas a recrear la secuencia de comunicación inicial con el sensor reflejada en la Figura 8. Para ello, se configuran inicialmente las líneas de comunicación como salidas digitales. Una vez enviada la secuencia, se realiza un cambio de base de la variable *command* mediante la función *shiftOut()*, cambiando de escala decimal a escala binaria y transmisiéndolo por el canal de datos. Además, se establece que el bit más significativo, *Most Significative Bit* (MSB), ocupa el primer lugar a la hora de realizar la conversión.

Posteriormente, en la línea 9, se cambia el modo de comportamiento de la línea de datos, pasando de ser una salida a ser una entrada. Entrada que en las siguientes líneas se lee para verificar que la cadena de instrucciones se ha recibido correctamente en el sensor. De ser así, mantiene en estado bajo la línea cuando la señal de reloj se haya en estado alto, y en estado alto cuando la señal de reloj está en estado bajo. En caso de no haber recibido correctamente la cadena de instrucciones se envía por puerto serie el carácter "\$", como identificador de que ha habido un problema en la comunicación con el sensor.

```
void sendCommandSHT(int command, int dataPin, int clockPin)
{
    int ack; pinMode(dataPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    digitalWrite(dataPin, HIGH);
```

```

digitalWrite(clockPin, HIGH); digitalWrite(dataPin, LOW);
digitalWrite(clockPin, LOW); digitalWrite(clockPin, HIGH);
digitalWrite(dataPin, HIGH); digitalWrite(clockPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, command);
digitalWrite(clockPin, HIGH);
pinMode(dataPin, INPUT);
ack = digitalRead(dataPin);
if (ack != LOW)Serial.println("$");
digitalWrite(clockPin, LOW); ack = digitalRead(dataPin);
if (ack != HIGH)
Serial.println("$");
}

```

- Función `waitForResultSHT(int dataPin)`: Esta tercera función adicional se emplea para esperar a que el sensor, tras recibir la instrucción para que mida, realice una espera hasta que la medida esté lista para ser recepcionada. Para llevar a cabo este propósito de forma eficiente precisa el parámetro de la línea de datos, ya que, una vez la medida esté preparada para ser enviada, ésta toma el valor lógico de estado bajo. El estado de la línea se comprueba cada 70 ms. Y de nuevo, si ha habido un infortunio en la medida, la misma línea tomará el estado lógico alto. De darse el caso, se volvería a enviar por puerto serie el carácter “\$”.

```

void waitForResultSHT(int dataPin)
{
int ack; pinMode(dataPin, INPUT);
for (int i = 0; i < 100; ++i)
{
delay(70); ack = digitalRead(dataPin); if (ack == LOW)
break;
}
if (ack == HIGH) Serial.println("$");
}

```

- Función `getData16SHT(int dataPin, int clockPin)`: Es la encargada de leer la medida realizada por el sensor y precisa dos parámetros para realizar la comunicación con éste. Éstos son el pin de la línea de datos y el pin asignado a la señal de reloj. Una vez realizada la lectura, devuelve el valor real de la lectura en escala decimal.

La medida se recoge en dos bytes, los cuales son procesados uno a uno, de ahí que pueda observarse en el código el desplazamiento del primer byte a la posición más significativa, línea 3. Posteriormente, tras indicar que ya se ha procesado el primer byte, se solicita al sensor el segundo byte, líneas de la 4 a la 6. Una vez disponible el segundo byte de datos se lee y se incorpora en la parte menos significativa de la variable `val`, variable donde se halla el otro byte de información.

```

int getData16SHT(int dataPin, int clockPin)
{
int val; pinMode(dataPin, INPUT); pinMode(clockPin, OUTPUT);
val = shiftIn(dataPin, clockPin, 8);
val *= 256; pinMode(dataPin, OUTPUT);
digitalWrite(dataPin, HIGH); digitalWrite(dataPin, LOW);
}

```

```

digitalWrite(clockPin, HIGH); digitalWrite(clockPin, LOW);
pinMode(dataPin, INPUT);
val |= shiftIn(dataPin, clockPin, 8);
return val;
}

```

- Función `skipCrcSHT(int dataPin, int clockPin)`: Esta función tiene como propósito indicarle al sensor que ha de obviar el checksum *Cyclic Redundancy Check* (CRC). Este checksum de un byte de extensión tiene la información para garantizar que cualquier dato erróneo que sea detectado y se elimine. En este caso no se ha considerado dicho byte de información.

```

void skipCrcSHT(int dataPin, int clockPin)
{
pinMode(dataPin, OUTPUT); pinMode(clockPin, OUTPUT);
digitalWrite(dataPin, HIGH);
digitalWrite(clockPin, HIGH);
digitalWrite(clockPin, LOW);
}

```

### 3.4 PROGRAMACIÓN NODO BASE

Para comprender el comportamiento del sistema de control del nodo base, además del código de programa, se aportan tres imágenes del diagrama de funcionamiento del propio programa, Figura 22, Figura 23 y Figura 24. Recordar, que este nodo es el encargado de comunicarse tanto con el nodo de medida como con la computadora. Enviará la orden de adquirir mensurandos al nodo medida, y por parte de la computadora, podrá recibir la instrucción para cambiar el tiempo de ciclo, configurado por defecto a 15 minutos.

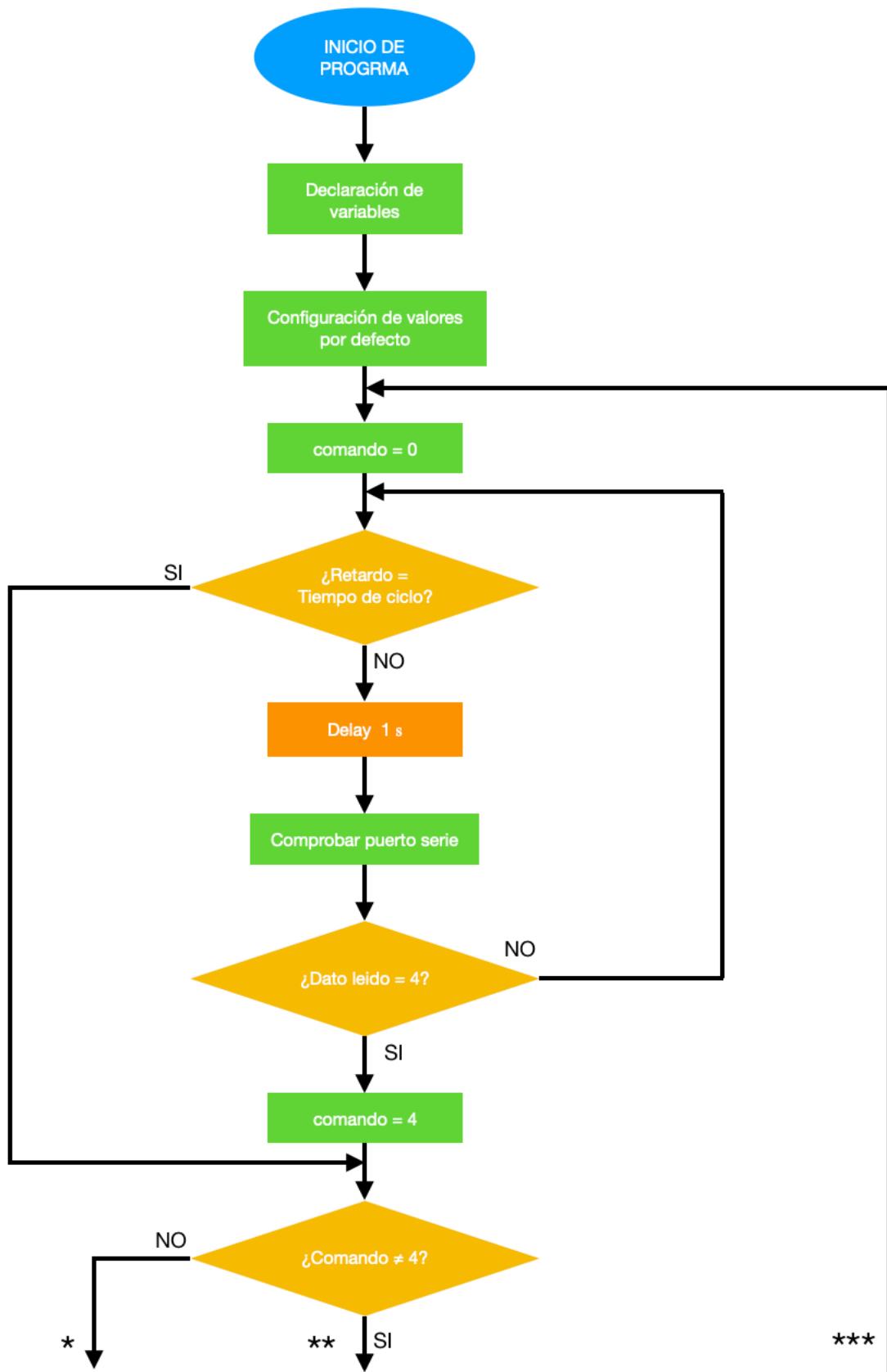


Figura 19: Diagrama de funcionamiento del nodo base 1/3

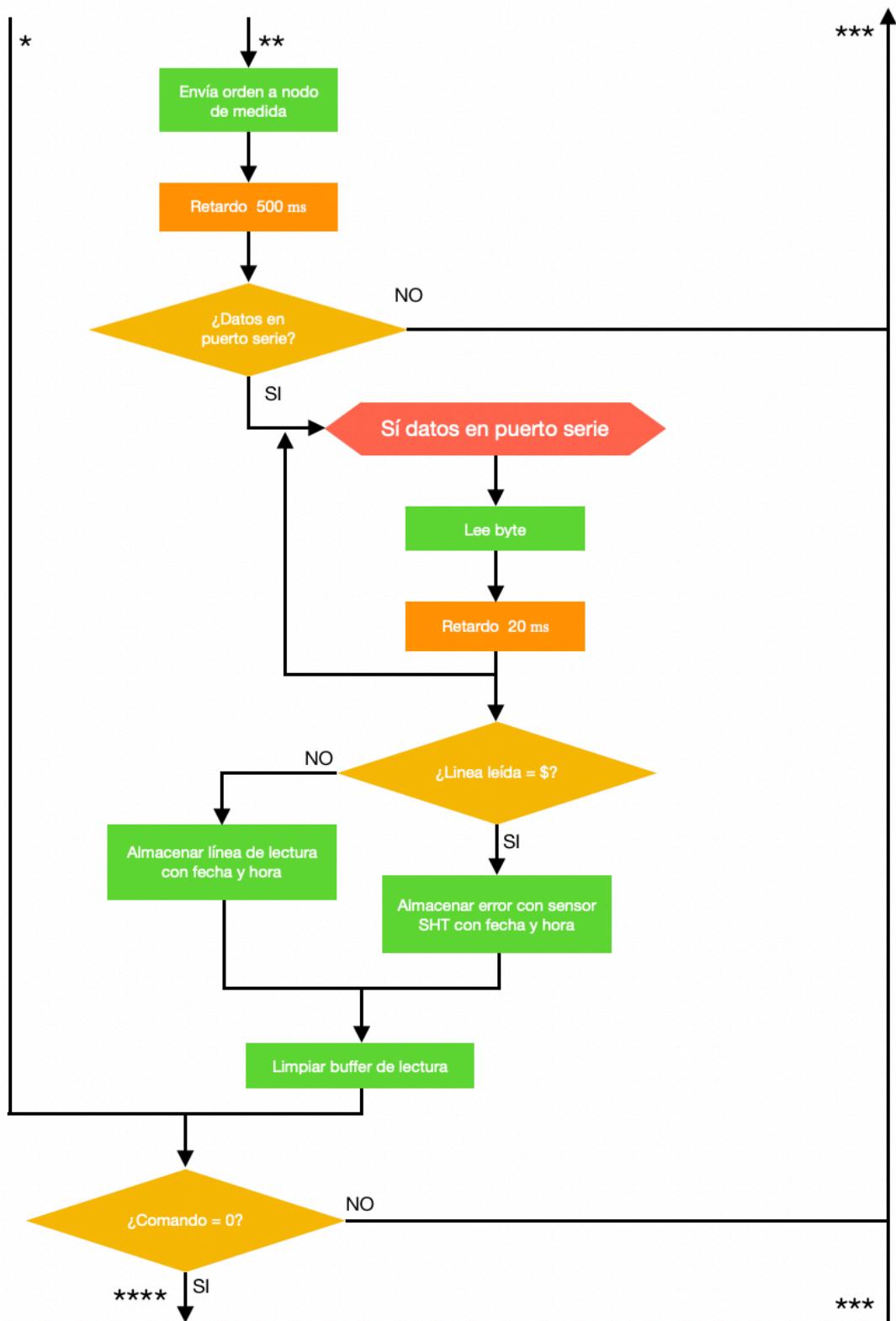


Figura 20: Diagrama de funcionamiento del nodo base 2/3

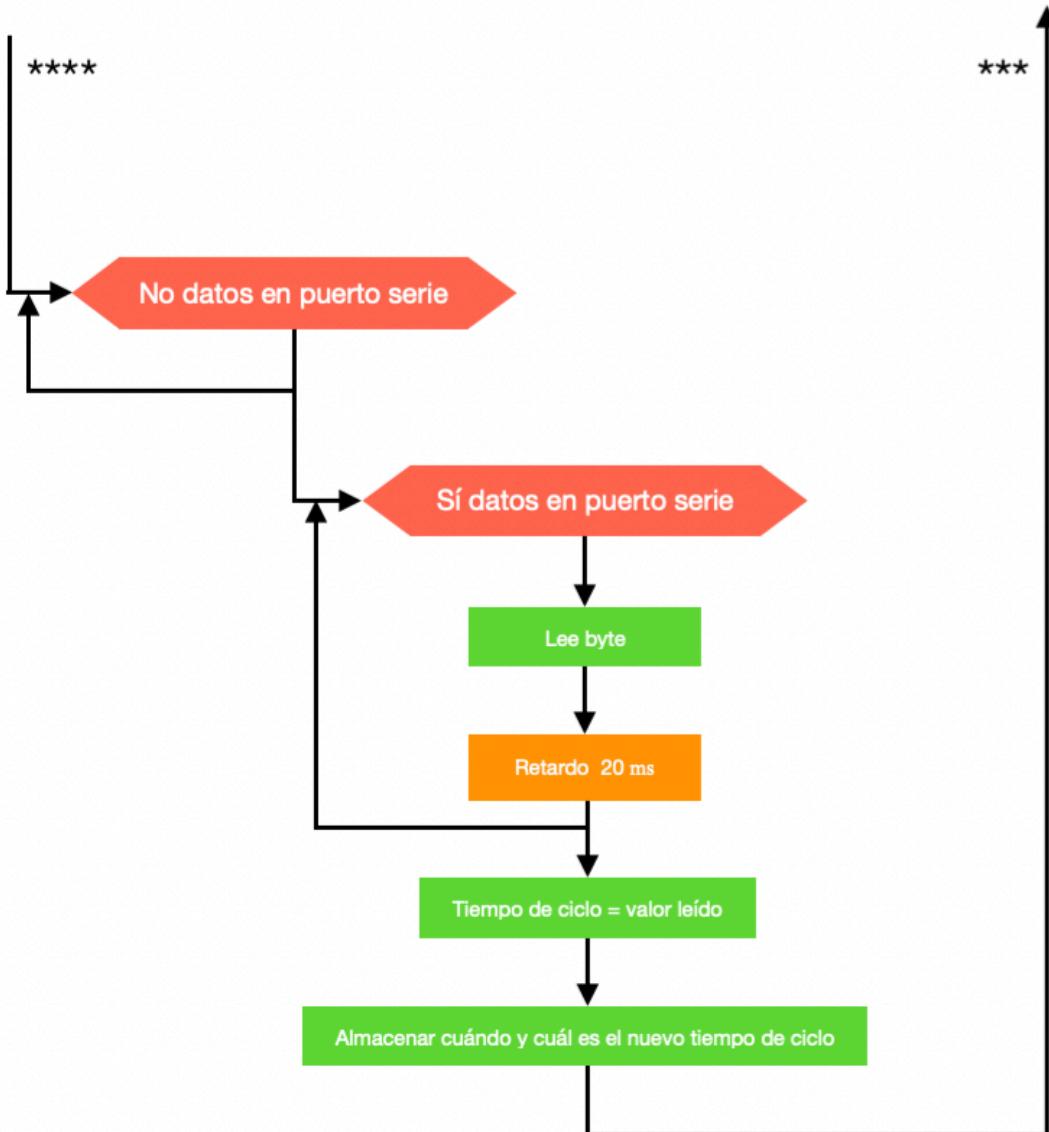


Figura 21: Diagrama de funcionamiento del nodo base 3/3

### 3.4.1 LIBRERÍAS

A continuación, se muestran las librerías de programa empleadas para que Arduino UNO trabaje conjuntamente con el módulo reloj y el módulo de almacenamiento SD, *RTCLib.h*, *RTC\_DS3221.h* y *SD.h*. Y también, las empleadas para establecer comunicación mediante el protocolo SPI e I<sup>2</sup>C, *SPI.h* y *Wire.h*. Todas ellas, contienen funciones que agilizan la configuración de los módulos, así como la fácil obtención de los parámetros de interés, como pueden ser en el caso del reloj la fecha y hora. O, en el caso del módulo SD, la posibilidad de leer y escribir ficheros de texto.

- Librería <RTCLib.h> : Esta librería, contiene las funciones generales empleadas para trabajar con el módulo reloj. Permitiendo obtener y ajustar la fecha y hora.

- Librería <RTC\_DS3231.h> : Esta librería, también relacionada con el módulo reloj, se emplea ya que contiene definiciones de objetos y funciones que permiten, a las contenidas en la librería anterior, ejecutarse.
- Librería <SD.h> : Esta librería contiene los objetos y funciones que permiten almacenar en el módulo SD información.
- Librería <SPI.h> : Esta librería se incluye en el programa principal para poder llevar a cabo comunicaciones que se rigen por el protocolo SPI. Las funciones que contiene esta librería se emplean para comunicarse con el módulo SD.
- Librería <Wire.h> : Esta librería, contiene las funciones necesarias para establecer una comunicación basada en protocolo I<sup>2</sup>C. Comunicación que se establece para interactuar con el módulo reloj.

### 3.4.2 DECLARACIÓN DE VARIABLES

Al igual que en el nodo de medida, se desglosan las variables globales en tres grupos, éstos se diferencian en función del cometido o la relación que tengan entre si las variables. Así pues, se distinguen variables de programa, variables RTC y variables SD. Éstas se describen a continuación:

- Variables de programa: Repercuyen generalmente durante el funcionamiento del programa, pero no están relacionadas con ningún módulo adicional a la propia placa Arduino UNO, éstas son *comando* y *Tciclo*. La primera de ellas, *comando*, toma valores numéricos, de ahí que se instancie de tipo *int*. La función de esta variable es, dependiendo de su valor, 0 para ejecutar la secuencia que solicita los mensurandos al nodo de medida cada periodo de tiempo determinado, o, 4 para ejecutar la secuencia relacionada con la actualización de ese periodo de tiempo que por defecto es 15 minutos.

La segunda variable, *Tciclo*, es el contenedor que alberga el intervalo de minutos que hay entre cada solicitud de mensurandos. Por ello y por qué alberga un número que hace referencia a tiempo, se declara de tipo *unsigned int*.

```
//VARIABLES DE PROGRAMA
int comando;
unsigned int Tciclo;
```

- Variables RTC : En este segundo grupo de variables se encuentra la relacionada con el módulo de reloj, de ahí el empleo de las siglas *Real Time Clock* (RTC). Ésta variable, *rtc*, se instancia como objeto de tipo *RTC\_DS3231* debido a que es este el microcontrolador que se sirve para controlar el tiempo.

```
//VARIABLES RTC
RTC_DS3231 rtc;
```

- Variables SD: La última clase de variables son las relacionadas con el módulo SD, en este caso, la conforma únicamente un objeto del tipo *File*, llamado *archivo*. Este objeto permitirá realizar escritura de datos en una memoria microSD empleando el módulo Wireless SD shield.

```
//VARIABLES
```

SD File archivo;

### 3.4.3 CONFIGURACIÓN DE VALORES POR DEFECTO – SETUP()

Del mismo modo que con el dispositivo Arduino NANO, en esta función se realizan los ajustes y asignación de valores por defecto, para que, en caso de cese de actividad del equipo, por ejemplo, por desconexión de la alimentación, al recuperar la actividad tenga los parámetros necesarios para llevar a cabo un arranque de acuerdo al funcionamiento diseñado.

```
void setup() {
Serial.begin(9600);
Wire.begin();
rtc.begin();
if (!rtc.isrunning())
{
  rtc.adjust(DateTime(__DATE__, __TIME__));
}
//CONFIGURACION SD
if (!SD.begin(10))
{
  Serial.println("Error en inicio de modulo SD ");
  while(!SD.begin(10)){}
}
comando = 0; cambio = 0; Tciclo = 15; //DEFAULT 15, expresado en minutos
}
```

En esta función, se inicializa la comunicación del puerto serie, también la propia con el reloj, *Wire.begin()*. Además de la puesta en marcha del mismo, realizando una actualización de fecha y hora como puede verse en el interior de la estructura condicional.

A continuación de esta secuencia, se encuentra la inicialización de la comunicación con el módulo de almacenamiento SD. Para ello ha de consultarse en la hoja de datos de la placa Arduino cuál es el pin digital con la función CS. En este caso, según la hoja de datos de la placa Arduino UNO, [12], esta función se localiza en el pin digital 10. Una vez verificado que el módulo se ha iniciado correctamente, se procede a asignar los valores por defecto de las variables de programa, así como se establece el tiempo de ciclo por defecto a 15 minutos.

### 3.4.4 BUCLE DE PROGRAMA – LOOP()

Como se muestra en el diagrama anterior, Figura 22, Figura 23 y Figura 24, la secuencia que ha de desarrollar el programa es:

- Si no se recibe instrucción por parte de la computadora, solicitar las medidas de los sensores del nodo de medida, una vez recibidas se almacenan en el archivo *DATA19.csv*, indicando la fecha y la hora de esa medida.
- Si se recibe el carácter “4” por parte de la computadora, se ejecuta la secuencia de actualización de cambio del tiempo de ciclo de medida. Una vez recibido el nuevo intervalo, se reporta en el archivo de datos indicando cuándo y a qué valor se ha actualizado el valor.

El código de la función *loop()* descrito en este apartado se ha desglosado en cinco partes, las cuales se desarrollan a continuación.

```
void loop() {
comando = 0; int cambio = 0; int i; int j; char inputByte; String buff;
DateTime actual = rtc.now();
archivo = SD.open("DATA19.csv", FILE_WRITE); //ARCHIVO MEMORIA
```

En este primer fragmento de código, se encuentran las instancias de las variables locales, así como la actualización de valor de las variables globales.

Además, se localiza la línea de código que realiza la apertura del archivo de datos donde se almacenan las lecturas recibidas del nodo de medida. Para llevar a cabo la apertura del archivo se precisa la función *open()*, incluida en la librería *SD.h*. En la referencia al lenguaje de esta librería, [14], se indica como ha de emplearse y las diferentes modalidades que ofrece para la apertura de archivos.

```
*****MODO AUTONOMO*****
//BUCLE QUE SOLICITA LA MEDIDA CADA Tciclo
for (i = 0; (i < Tciclo) && (cambio == 0); i++)
{ for (j = 0; (j < 60) && (cambio == 0); j++)
{ comando = Serial.read();
if(comando == 52) cambio = 1;
else delay(1000);
}
cambio = 0;
```

Este segundo fragmento de programa contiene las líneas de código que plasman cómo se realiza la espera de tiempo para solicitar una nueva línea de medidas. Mas puede observarse que, en caso de que se reciba la instrucción de cambio de tiempo de ciclo, *comando* igual a “52”, este valor se corresponde con el carácter decimal “4” en la tabla ASCII. En ese caso, se actualiza el valor de la variable *cambio* a “1”, y se cesa la actividad en los bucles que generan la espera de tiempo correspondiente al tiempo de ciclo. Además, se vuelve a re establecer el valor de *cambio* a “0”, así en la iteración siguiente se inicia la espera de nuevo.

```
//CODIGO RECEPCION DE INFORMACION Y ALMACENAMIENTO EN SD
if(comando != 52)
{
Serial.print("1"); //ORDEN DE LECTURA DIRIGIDA AL NODO DE MEDIDA
delay(500); //TIEMPO DE ESPERA HASTA CALCULO DE LECTURAS
if ((Serial.available()>0) && (Serial.peek()!=36))
```

```

{
while(Serial.available()>0) { //LECTURA DE LINEAS DE MENSURANDOS
buff+=(char)Serial.read();
delay(20);
}
archivo.print(actual.day()); archivo.print("/"); archivo.print(actual.month()); archivo.print("/");
archivo.print(actual.year()); archivo.print(" - "); archivo.print(actual.hour());
archivo.print(":"); archivo.print(actual.minute()); archivo.print(":"); archivo.print(buff);
}

```

En el tercer fragmento, se muestra el código que se ejecuta cuando *comando* toma un valor distinto de “52”. Este caso se corresponde con el envío de la orden de medir al nodo de medida. Para ello se envía el carácter “1”, se realiza una pequeña espera para que el nodo de medida prepare y envíe todas las medidas.

Tras la espera, se comprueba que el buffer del puerto serie contiene bytes de información, y que el primero de ellos no se corresponde con el valor “36”, índice representativo del símbolo del dólar en la tabla ASCII. De ser así, significaría que ha habido un problema de lectura del sensor SHT15. En cambio, si el primer byte no se corresponde dicho número, se almacena la línea de mensurandos indicando fecha y hora de medida.

```

else{
if(Serial.read() == 36) { //ERROR EN SENSOR HUMEDAD/TEMPERATURA
archivo.print(actual.day()); archivo.print("/"); archivo.print(actual.month()); archivo.print("/");
archivo.print(actual.year()); archivo.print(" - "); archivo.print(actual.hour()); archivo.print(":");
archivo.print(actual.minute());
archivo.print(";Lectura erronea del de temperatura y humedad");
while(Serial.available()>0){ Serial.read(); }
} } buff = "";
}

```

El cuarto fragmento contiene el código a ejecutar en el caso de que el sensor SHT15 no haya podido realizar la medida, caso en el que se recibe el carácter “\$”. El procedimiento que se lleva a cabo consiste en reportar el fallo en el archivo de datos, indicando fecha y hora, y además, se limpia el buffer ya que pueden haberse recibido medidas incongruentes.

```

//MODO CAMBIO DE TIEMPO DE CICLO - 4
if (comando == 52)
{
while(Serial.available()<=0){}
while(Serial.available()>0)
{
buff += (char) Serial.read(); delay(20);
Tciclo = buff.toInt(); buff = "";
archivo.print(actual.day()); archivo.print("/"); archivo.print(actual.month()); archivo.print("/");
archivo.print(actual.year()); archivo.print(" - ");
archivo.print(actual.hour()); archivo.print(":"); archivo.print(actual.minute());
archivo.print(";Tiempo de ciclo actualizado a "); archivo.print(Tciclo);
archivo.print(" minutos\n");
}
}

```

}//FIN DE PROGRAMA

En este quinto y último fragmento de código correspondiente al programa principal, se muestra el código a ejecutar en el caso de que se reciba la orden de cambio de tiempo de ciclo. Inicialmente, se produce una espera hasta recibir el nuevo tiempo, una vez recibido, se asigna el nuevo valor a la variable *Tciclo*. Y se genera un reporte en el archivo de datos indicando el nuevo tiempo de ciclo, así como la fecha y hora de la modificación del mismo.

## 4. INTERFAZ DE USUARIO

En este capítulo se describirán las partes más relevantes del código desarrollado en la plataforma de desarrollo software LabVIEW. LabVIEW es un software de ingeniería de sistemas que requiere pruebas, medidas y control con acceso rápido a hardware e información de datos, [17].

Se ha desarrollado una aplicación ejecutable llamada *TFGApp.exe*, y se ha implementado en la plataforma digital LabVIEW, concretamente en la versión del año 2018. Todos los archivos del proyecto relacionados con el código de esta interfaz se aportan en el CD adjunto, así como la aplicación ejecutable *TFGApp.exe*.

Se distinguirán varios apartados considerando las diferentes opciones de adquisición, control y visualización. En el apartado *panel principal*, se describirán las opciones de visualización que se plantean, así como la forma de interacción con el programa desarrollado. El apartado dedicado a describir las opciones de adquisición es: *Adquisición de datos*. Los apartados relacionados con las opciones de control son: *Selección de puerto de comunicaciones*, *calibración de acelerómetro*, *calibración convertidor resistencia-frecuencia*, *almacenamiento de coeficientes* y *cambio Tciclo*.

### 4.1 PANEL PRINCIPAL

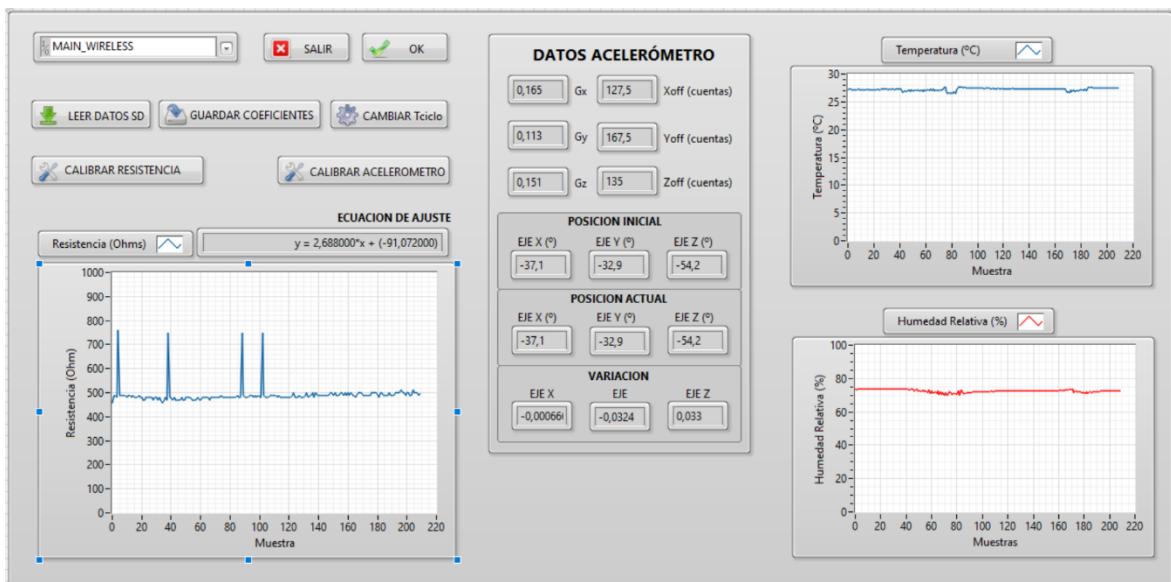


Figura 22: Interfaz de usuario asociada al programa principal

El programa principal se ubica en el archivo Principal.vi, la apariencia de éste se muestra en la Figura 22, a simple vista pueden distinguirse cuatro zonas de interacción, la primera y más relevante la correspondiente al conjunto de elementos, en la esquina superior izquierda. Estos elementos están relacionados con las opciones de control, el selector desplegable se relaciona con la opción de *Selección de puerto de comunicaciones*. Los botones CALIBRADO ACELERÓMETOR y CALIBRADO RESISTENCIA al presionarse ejecutan

la opción de control *Calibración de acelerómetro* o *Calibración convertidor resistencia-frecuencia*.

Del mismo modo sucede para los botones GUARDAR COEFICIENTES y CAMBIA TCiclo, activan respectivamente las opciones asociadas a ellos, *Almacenamiento de coeficientes* y *Cambio Tciclo*.

La excepción de controles presentes en ese cuadro es el botón LEER DATOS SD, que se relaciona con la opción *Adquisición de datos*.

En cuanto a las opciones de visualización, en la esquina inferior izquierda, se encuentra la zona dedicada a la representación de las medidas de resistencia, además puede observarse un cuadro de texto dedicado a contener la ecuación de conversión de tiempo a resistencia. La zona central, está dedicada a albergar los datos correspondientes al acelerómetro, en ella, en primer lugar, se hayan los contenedores que mostrarán los valores de ganancia y offset de cada eje, parámetros determinados en el proceso de calibrado de éste. Además, registra la posición inicial de referencia en la que se coloca el sensor, y también la actual, la cual se corresponde con la última posición registrada leída del fichero que proporciona los datos, llamado *DATA19.csv*. En la parte inferior se muestra cómo ha variado, en referencia a la posición inicial, el ángulo de cada eje.

## 4.2 SELECCIÓN DE PUERTO DE COMUNICACIONES

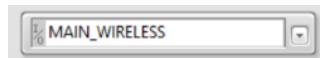


Figura 23: Selector desplegable de puerto de comunicaciones

Este elemento, está dedicado a la selección del puerto serie por el que se va a establecer la comunicación, en este caso *MAIN\_WIRELESS* es el escogido. Para proceder a ejecutar el programa ha de seleccionarse un puerto para la comunicación, de no ser así no se ejecuta ninguna instrucción. Para cumplir con este cometido, se han empleado dos estructuras en bucle que se comentarán a continuación.

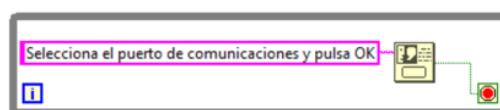


Figura 24: Selección de puerto de comunicaciones, bucle 1

En la Figura 24 se muestra el primer bucle, denominado bucle 1, que tiene la función de una vez iniciado el programa indicarle al usuario qué ha de hacer.

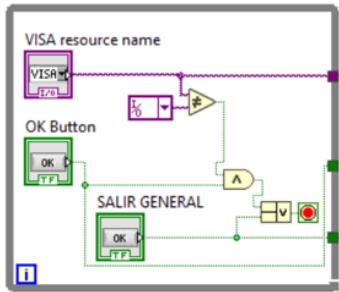


Figura 25: Selección de puerto de comunicaciones, bucle 2

El bucle 2, Figura 25, se emplea para verificar que se ha seleccionado un canal de comunicación, y que éste es válido.

### 4.3 CALIBRACIÓN DE ACELERÓMETRO



Figura 26: Panel frontal subrutina CALIBRADO\_ACELEROMETRO

En la Figura 26 se muestra el panel frontal de la subrutina de calibrado del acelerómetro, *CALIBRADO\_ACELEROMETRO.vi*. En él puede observarse un selector de puerto de comunicaciones, un cuadro de texto, botones de interacción y los valores de los parámetros de calibración.

La finalidad de esta subrutina es llevar a cabo el cálculo de las ganancias y valores de offset de cada eje, y la determinación de la posición de referencia del acelerómetro. Para ello, como puede verse en el cuadro de texto presente en el panel frontal, se coloca el sensor en las posiciones absolutas correspondientes a las de la Figura 6. Una vez se ha adquirido el valor de cada posición absoluta, se calculan los parámetros, y se solicita colocar el sensor en la posición de referencia.

## 4.4 CALIBRACIÓN CONVERTIDOR RESISTENCIA – FRECUENCIA

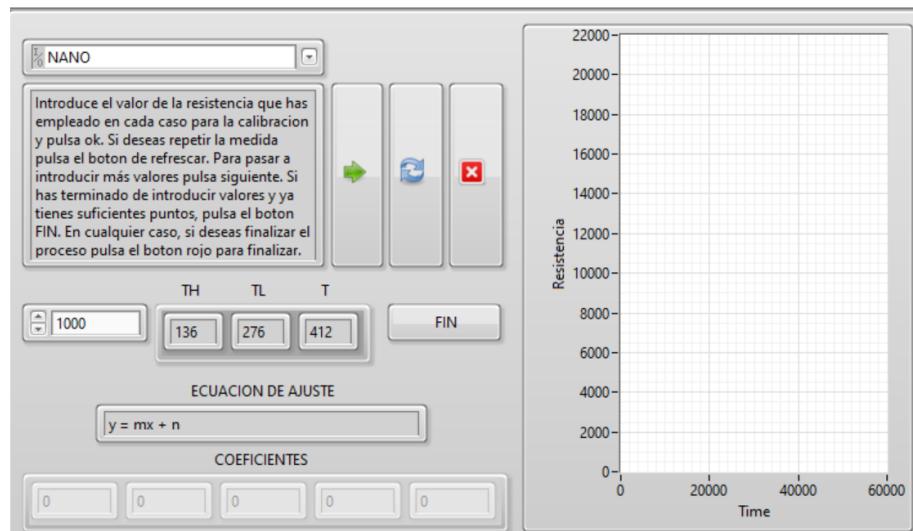


Figura 27: Panel frontal de la subrutina CALIBRADO\_RESISTENCIA

En la Figura 27 se muestra el panel principal de la subrutina de calibrado del convertidor resistencia-frecuencia *CALIBRADO\_RESISTENCIA.vi*. El funcionamiento de esta subrutina consiste en generar un vector que contenga el valor de resistencia medido y la lectura de tiempo que se toma de esa resistencia.

Para introducir el valor de resistencia que se está midiendo se emplea el dial numérico de la izquierda. Una vez asignado el valor, se pulsa el botón de siguiente, y se adquiere la medida de tiempos que se muestra en los tres contenedores contiguos al dial, estos contenedores representan el tiempo en estado alto, *TH*; el tiempo de estado bajo, *TL*; y la suma de ambos, *T*. (tiempos expresados en microsegundos).

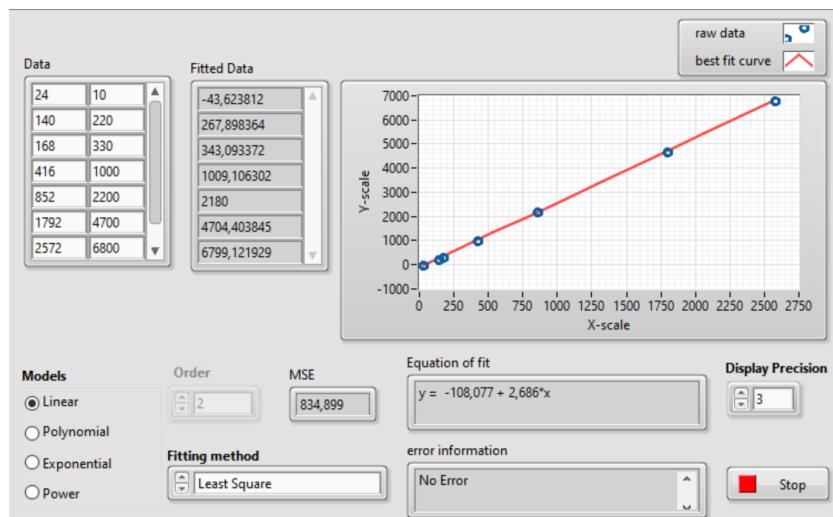


Figura 28:Panel frontal subrutina de cálculo de recta de ajuste

Si se desea repetir la medida ha de pulsarse el botón de refrescar. Una vez se haya concluido la toma de muestras, se pulsa el botón FIN, y se ejecuta una subrutina preconstruida en el entorno de desarrollo que permite seleccionar la recta que más se ajuste a la distribución de puntos suministrada. En la Figura 28 puede verse el panel principal de esta subrutina.

## 4.5 ALMACENAMIENTO DE COEFICIENTES

El propósito de esta opción es almacenar en un fichero de texto los valores de calibración con los que se esté trabajando en un momento dado. De esta forma, se pueden rescatar los parámetros posteriormente para procesar los datos en el mismo escenario. El codificado de esta información es el mismo que se presenta en el apartado Adquisición de datos. En la Figura 29 se muestra cómo se lleva a cabo esta codificación.

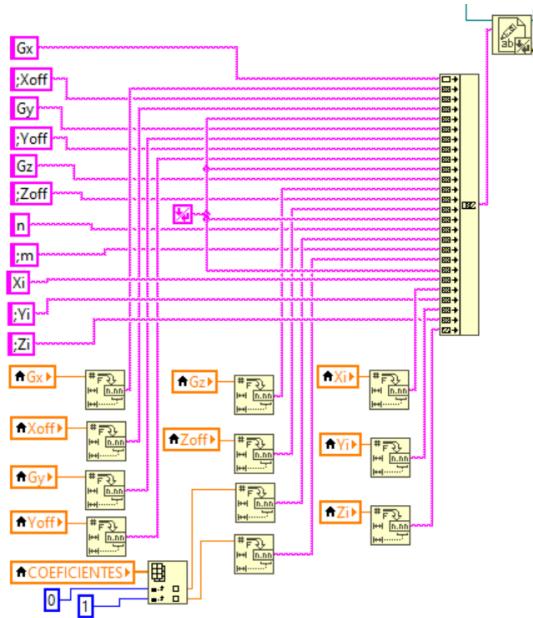


Figura 29: Codificación de parámetros a almacenar en archivo de texto

A continuación, se encuentra un ejemplo de esta codificación, mostrando los datos de un archivo de texto generado tras ejecutar esta opción es la siguiente:

```

Gx0,165000;Xoff127,500000
Gy0,113000;Yoff167,500000
Gz0,151000;Zoff135,000000
n-91,072000;m2,688000
Xi34,100000;Yi55,500000;Zi78,200000
  
```

En la codificación, los identificadores correspondientes al sensor acelerómetro son *Gx*, *Xoff*, *Gy*, *Yoff*, *Gz*, *Zoff*, *Xi*, *Yi*, *Zi*. Correspondiéndose los valores de las ganancias de cada eje a los números contiguos a los identificadores *Gx*, *Gy* y *Gz*. En cuanto a los valores de offset de los ejes se determinan observando los índices *Xoff*, *Yoff* y *Zoff*. También contiene este archivo de texto la información relativa a la posición de referencia del acelerómetro, se identifica la posición de cada eje mediante los índices *Xi*, *Yi* y *Zi*.

Los coeficientes de la recta de calibración del convertidor resistencia-frecuencia se identifican, por un lado, mediante el carácter *n* el factor linear de la recta, y por otro lado, mediante el carácter *m*, el valor de la pendiente de la recta de calibración.

Aclarar que para distinguir entre valores se emplea el carácter “;”.

## 4.6 CAMBIO TCICLO

Esta función es la que se emplea para comunicarse con el nodo base y asignar un nuevo tiempo de ciclo, *Tciclo*. En el panel frontal correspondiente a esta función ha de introducirse el nuevo periodo de muestreo, expresado en minutos. Una vez se ha indicado este tiempo, se pulsa el botón OK y se envía el nuevo valor al nodo, véase Figura 30.

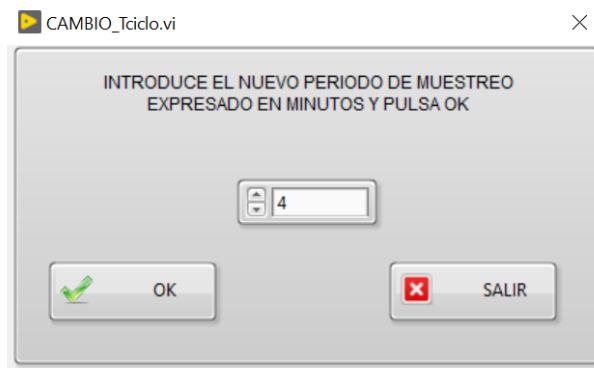


Figura 30: Panel principal subrutina CAMBIO\_Tciclo

## 4.7 ADQUISICIÓN DE DATOS

Para llevar a cabo el correcto funcionamiento de algunas funciones anteriores, ha de presentarse la subrutina empleada para realizar la adquisición de datos, *SUBVI\_AdquisicionDatos.vi*. Esta subrutina, se encarga de procesar una cadena de texto que contiene los mensurandos, los extrae y los proporciona en un formato numérico. En función del tipo de adquisición de datos que realiza esta subrutina se han distinguido dos modalidades, la adquisición de datos en procesos de calibración, y la adquisición de datos completa.

### 4.7.1 ADQUISICIÓN DE DATOS EN PROCESOS DE CALIBRACIÓN

Esta modalidad realiza un procesado de mensurandos selectivo. Si se invoca la subrutina de adquisición de datos desde la subrutina de calibrado del acelerómetro, se procesan únicamente los mensurandos relativos al sensor acelerómetro. Y en caso de ser invocada por la subrutina de calibrado del convertidor, se procesan los datos correspondientes al convertidor. Para lograr distinguir qué datos han de procesarse, la subrutina de adquisición de datos posee un índice numérico de entrada para realizar la distinción. Si se pretende procesar datos referentes al acelerómetro, habrá que asignar a este índice el valor numérico 1. En caso de requerir los datos relevantes al convertidor, el índice deberá ser 2.

Realizada cualquiera de los dos tipos calibraciones, o ambas, existe la posibilidad de generar un archivo de texto que contiene los coeficientes de calibración. Para realizar esta acción el programa se sirve del código presentado en el apartado 4.5.

## **4.7.2 ADQUISICIÓN DE DATOS COMPLETA**

La segunda modalidad de adquisición de datos es la adquisición de todos los elementos de la linea de datos. Del mismo modo que se realizaba en los casos anteriores, para distinguir este modo de adquisición de datos de los dos anteriores, ha de invocarse la función indicando que el índice numérico 3. De este modo, no se realiza ninguna discriminación de datos, sino que se procesa la totalidad de la cadena de datos.

## 5. RESULTADOS EXPERIMENTALES

---

En este capítulo se van a exponer los resultados de las diferentes pruebas realizadas en el entorno de estudio, éste se ubica en la localidad de Chiva, Valencia, concretamente en un sótano localizado en la vivienda familiar del autor. En dicho espacio subterráneo se cuenta con un equipo de des humidificación, electrodoméstico que se pone en funcionamiento diario en la franja de 23:00 a 07:00.

En el primer apartado, dedicado a la verificación del circuito, se muestran los datos recopilados que verifican el circuito empleado para obtener medidas de resistencia. Y, en el segundo apartado, dedicado a la verificación del sistema de monitorización, se plasman los resultados recopilados en cuatro experiencias.

### 5.1 VERIFICACIÓN DEL CIRCUITO

Este primer estudio tuvo como objeto verificar si el circuito conversor resistencia-frecuencia se ajustaba a las condiciones resistivas del muro. El circuito, estaba inicialmente configurado para medir altas resistencias [10 M, 100 G]  $\Omega$ .

Para realizar la comprobación del orden de magnitud de la resistencia del muro se empleó un multímetro de mano. Se tomaron medidas en dos puntos alineados respecto del plano vertical, pero a diferente altura, véase Figura 31.

El procedimiento consistió en realizar medidas con la sonda en ambas polaridades, es decir,



Figura 31: Disposición de puntos de medida

positivo-negativo (PN) y negativo-positivo (NP). Y debido a las oscilaciones que presentaban los mensurandos desde que se conectaba la sonda, se decidió adquirir la medida transcurrido un periodo de 1 minuto tras conectar la sonda.

ESTUDIO PRELIMINAR RESISTENCIA DEL MURO PUNTO INFERIOR									
Resistencia <sub>PN</sub> ( $\Omega$ )	17220	17080	17040	17020	16560	16660	16540	17760	17020
Resistencia <sub>NP</sub> ( $\Omega$ )	17380	16680	17020	16720	16720	16920	16880	16660	16380
ESTUDIO PRELIMINAR RESISTENCIA DEL MURO PUNTO SUPERIOR									
Resistencia <sub>PN</sub> ( $\Omega$ )	3180	3160	3080	2980	3060	2760	2780	3240	3180
Resistencia <sub>NP</sub> ( $\Omega$ )	8120	8260	8160	8260	8040	8460	8560	8160	8540

Tabla 8: Resultados del estudio preliminar

Los resultados de este estudio preliminar se presentan en la Tabla 6, puede verse que no se trata de resistencias de gran magnitud, por lo tanto, hubo de modificarse el circuito conversor para adaptarlo a este orden de magnitud.

Para adaptarlo, se tuvieron en cuenta los valores máximos y mínimos recogidos, valores destacados en la tabla. Así como la resolución temporal de la placa Arduino NANO, que es de  $1 \mu\text{s}$ .

Teniendo en cuenta estos parámetros, y aplicando la ecuación 3, se estimó el orden del condensador en nF y se escogió el valor estándar de 68 nF.

Empleando este condensador y la resolución temporal del Arduino NANO, puede calcularse mediante la Ecuación 2 que el mínimo valor de resistencia mesurable es  $3,93 \Omega$ .

## 5.2 VERIFICACIÓN DEL SISTEMA DE MONITORIZACIÓN

En este segundo apartado, se encuentran cuatro series de medidas con condiciones de entorno diferentes para cada caso. Aclarar que se han llevado a cabo sobre los mismos puntos de medida que el estudio preliminar. Se muestra la disposición del nodo de medida, así como los diferentes componentes de éste en la Figura 32.

En la imagen se destacan los elementos del nodo. En amarillo, el sensor acelerómetro ADXL335. En el recuadro verde se destaca el contenedor del nodo de medida, puede verse en el frente de este contenedor el sensor de temperatura y humedad relativa SHT15. En rojo se destacan los bornes sobre los que se realiza la medida. En el recuadro azul se encuentra la fuente de alimentación continua que proporciona una tensión de  $\pm 15 \text{ V}$ , así como una referencia a masa. Además, en la imagen puede verse una placa de prototipada empleada para extender la alimentación al nodo de medida.

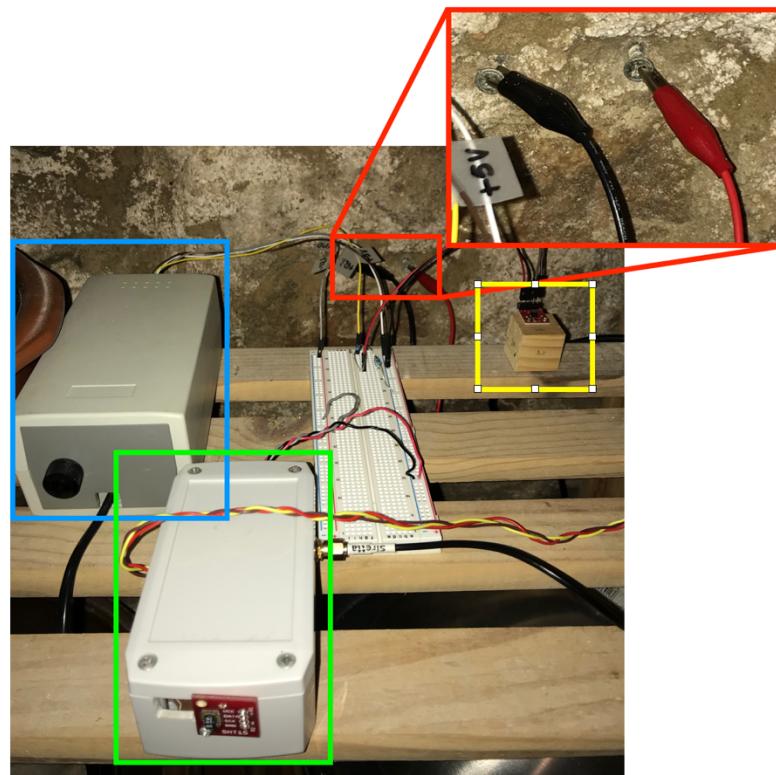


Figura 32: Disposición del nodo de medida

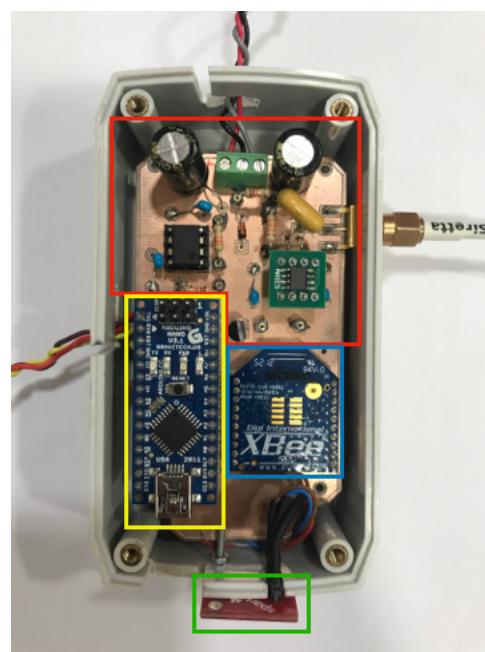


Figura 33: Nodo de medida

En la Figura 33 se muestra el interior del contenedor del nodo de medida. En la imagen puede observarse, destacado en un recuadro rojo el circuito correspondiente al convertidor resistencia-frecuencia. Circunscrito por un recuadro de color amarillo se destaca el módulo Arduino NANO. Contenido en un recuadro azul, se muestra el módulo de comunicación Xbee. Y, en verde se destaca la posición del sensor de humedad relativa y temperatura SHT15.

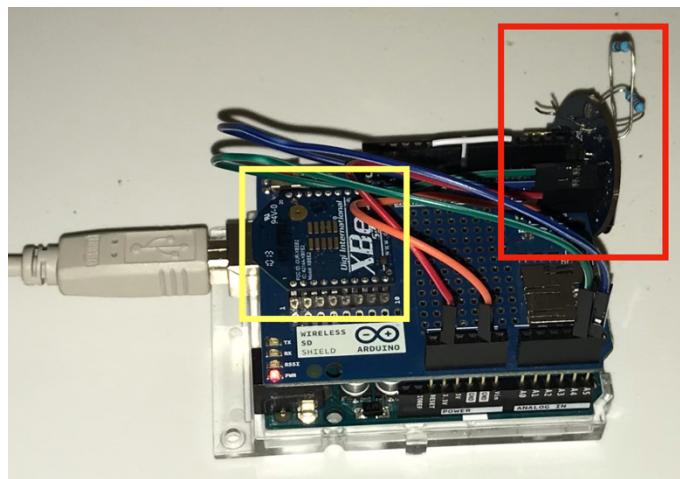


Figura 34: Disposición del nodo base

En cuanto al nodo base, se ubicó en la cocina de la vivienda, Figura 34. Destacado en amarillo puede verse el módulo de comunicación inalámbrico, y en color rojo el módulo reloj. También puede apreciarse la superposición del módulo SD sobre la placa Arduino UNO.

## 5.2.1 PRIMERA SERIE

En la primera serie de medidas se monitorizó el punto de medida inferior en condiciones de no funcionamiento del deshumidificador. Se llevó a cabo desde el 16 de julio de 2019 a las 12:32 horas hasta el 18 de julio de 2019 a las 16:53. Las medidas se tomaron a razón de quince minutos, generando 210 muestras.

Puede encontrarse el fichero con los resultados experimentales en el CD que se adjunta bajo el nombre de DATA19\_S1.csv.

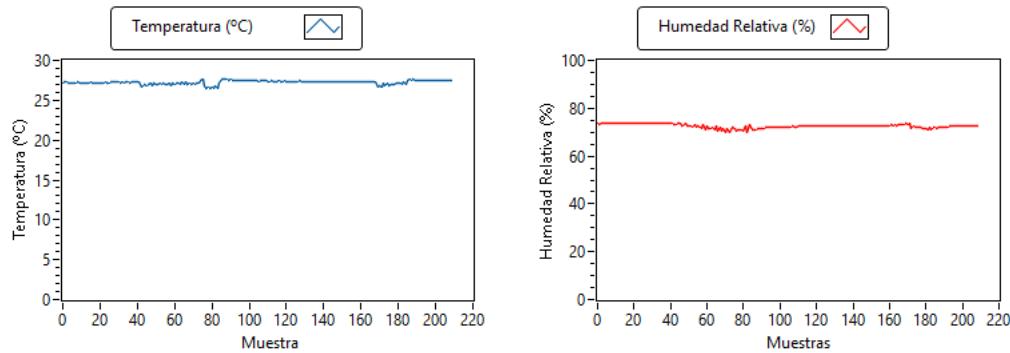


Figura 35: Gráficas temperatura y humedad relativa, primera serie

En las gráficas que se muestran en la Figura 35, puede observarse que tanto la temperatura, izquierda, como la humedad, derecha, permanecen constantes independientemente del horario. Manteniéndose la temperatura en torno a 27 °C, y la humedad aproximadamente en 75 %.

El comportamiento que desarrolla la resistencia del muro es similar, mantiene un valor constante de aproximadamente 500 Ω. Aunque en esta gráfica, en la Figura 36, puede observarse la aparición de cuatro picos con valores más altos y atípicos. Estos picos pueden ser debidos a un fallo en la sincronización de los nodos, produciéndose un error en el valor asignado.

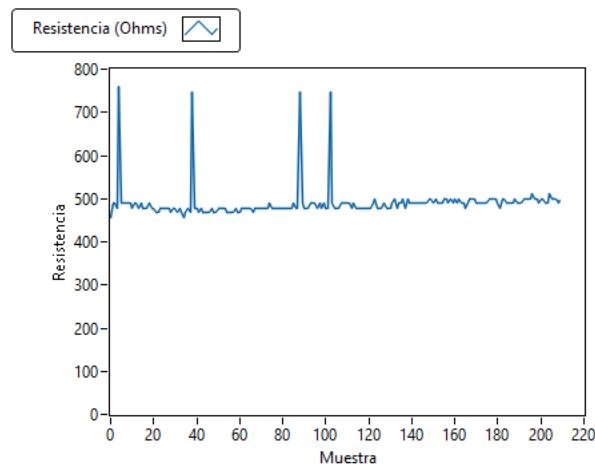


Figura 36: Gráfica de resistencia, primera serie

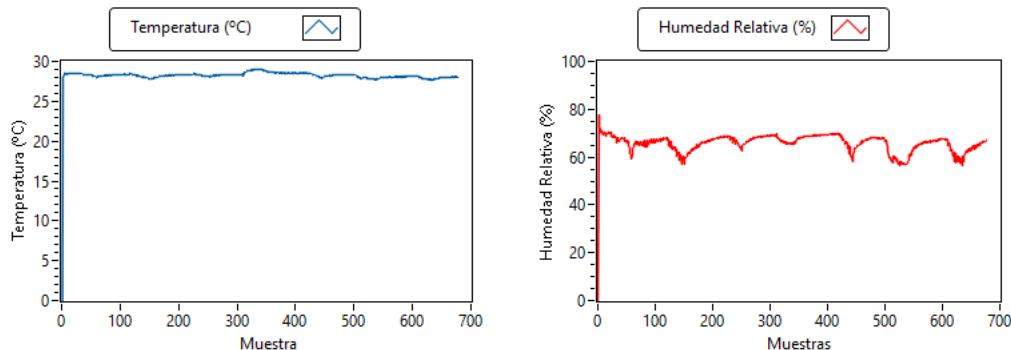
## 5.2.2 SEGUNDA SERIE

La fecha de realización de estas medidas fue de las 21:06 del día 30 de julio de 2019, hasta el día 6 de agosto de 2019 a las 22:08.

Realizándose un total de 678 medidas sobre el punto superior, con un intervalo de quince minutos entre sí.

En este caso se pretende observar la repercusión del deshumidificador empleado en la estancia, así como conocer el valor de resistencia de este punto. El fichero de resultados correspondiente a esta experiencia se encuentra en el CD adjunto bajo el nombre de DATA19\_S2.csv.

En estas gráficas, Figura 37, puede observarse que el equipo de deshumidificación cumple con su cometido de forma satisfactoria durante su periodo de funcionamiento, reduciendo el porcentaje de humedad en un 15 % casi un 20 % en algunos casos. En cuanto a cómo afecta este equipo a la temperatura del espacio, decir que reduce aproximadamente 1 °C la temperatura, puede observarse en la gráfica un leve rizado entorno al valor de 28 °C.



La gráfica de la Figura 38, muestra las medidas de resistencia adquiridas durante el periodo

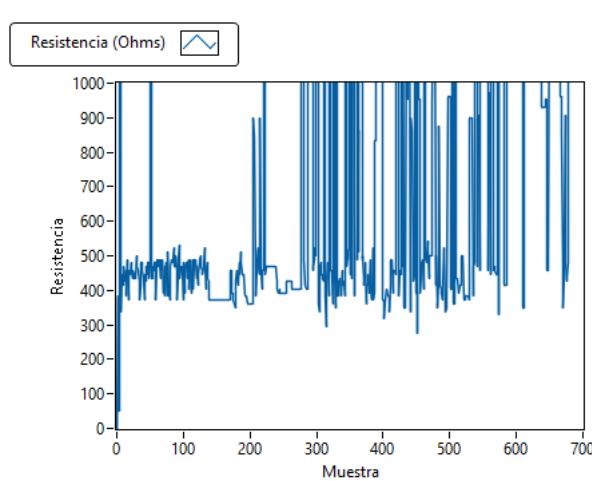


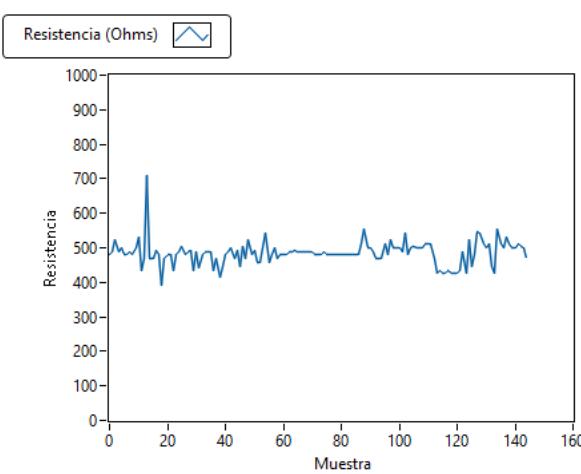
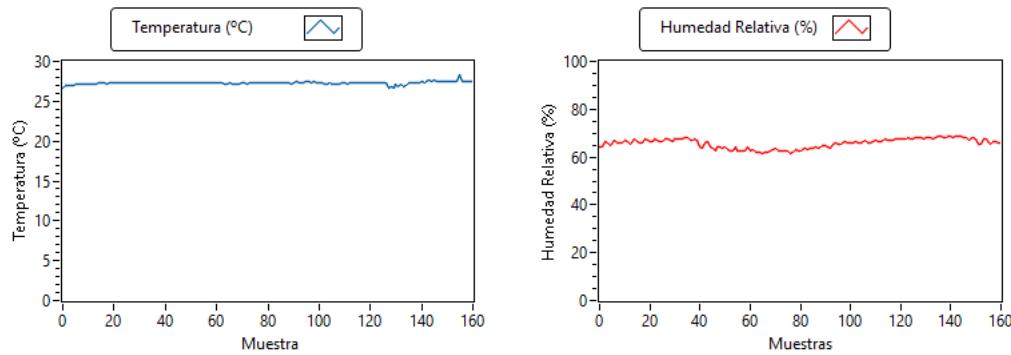
Figura 37: Gráfica de resistencia, segunda serie

de tiempo de esta serie. Puede observarse como estas medidas han sido contaminadas, interpretándose erróneamente, se supone que es debido a ruido electromagnético introducido por el equipo de deshumidificación. Para verificarlo es necesaria la realización de más pruebas experimentales.

### 5.2.3 TERCERA SERIE

La adquisición de medidas en esta tercera fase tuvo lugar en el periodo comprendido entre las 17:12 del día 26 de septiembre de 2019, y las 09:15 del 28 de septiembre de 2019. Del mismo modo que en las dos series anteriores, se tomaron las medidas cada quince minutos.

En este caso, se pretendía comprobar si el origen de la contaminación las medidas de la serie anterior era el deshumidificador. Para ello se tomaron muestras en el punto superior con el equipo sin ejercer acción alguna. El fichero de resultados correspondiente a esta experiencia se encuentra en el CD adjunto bajo el nombre de DATA19\_S3.csv.



Puede observarse en las gráficas de la Figura 39, que la temperatura y humedad relativa en este caso permanecen bastante centradas sobre un valor.

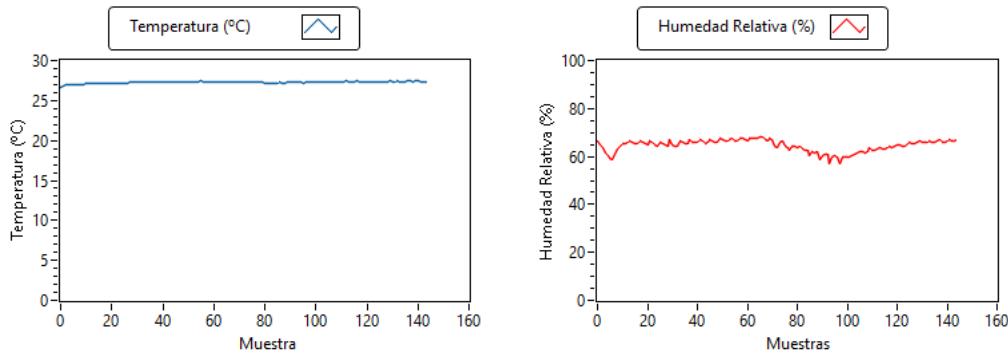
También puede observarse en la Figura 40 que, al detener el funcionamiento del deshumidificador, las muestras de resistencia tomadas no parecen estar contaminadas por ningún ruido ajeno al circuito. Aunque es cierto que de nuevo aparece un pico discordante entorno a la muestra diez, que se estima que es debido a problemas de sincronización.

### 5.2.4 CUARTA SERIE

La cuarta serie de medidas se realizó del día 28 de septiembre de 2019 a las 9:23 horas hasta las 21:23 del día 29 de septiembre de 2019. En este caso, las condiciones serían deshumidificador en funcionamiento, y punto de medida inferior. Obteniendo mensurandos cada quince minutos.

El objeto de esta cuarta serie es, asegurar que la contaminación de las muestras correspondientes a la resistencia es debida al ruido electromagnético generado por el deshumidificador en funcionamiento.

El fichero de resultados correspondiente a esta experiencia se encuentra en el CD adjunto bajo el nombre de DATA19\_S4.csv.



La gráfica correspondiente a humedad relativa de la Figura 41 pone de manifiesto la acción del deshumidificador durante este tiempo de adquisición. A la vista del resultado de medidas de resistencia que se muestran en la gráfica de la Figura 42, puede asegurarse que las medidas de resistencia, empleando el convertidor resistencia-frecuencia, se ven perturbadas por ruido de componente electromagnética introducido por la acción del equipo de deshumidificación.

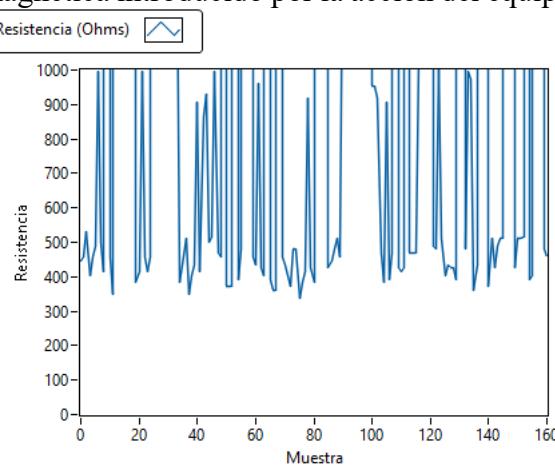


Figura 42: Gráfica de resistencia, cuarta serie

## 6. CONCLUSIONES Y PROPUESTAS DE FUTURO

---

### 6.1 CONCLUSIONES

Como se ha descrito en el documento presentado, se ha logrado desarrollar e implementar un sistema de monitorización de medidas de resistencia basado en un conversor resistencia-frecuencia. Además, se han alcanzado los hitos planteados inicialmente:

- Medir temperatura del ambiente.
- Medir humedad relativa al ambiente.
- Medir frecuencia mediante el empleo de una plataforma hardware libre como es Arduino.
- Implementar un circuito de acondicionamiento aplicado a la medida de resistencia en el rango de  $k\Omega$ .
- Desarrollar una interfaz de monitorización y control empleando LabVIEW.
- Controlar el instrumento mediante el uso de una interfaz de usuario, así como los módulos inalámbricos y las diferentes placas Arduino.
- Desarrollar el software adecuado para la generación de informes.
- Se ha verificado experimentalmente el funcionamiento del sistema de monitorización y control de medidas de resistencia desarrollado realizando medidas sobre un muro.

La realización de cada apartado me ha permitido ampliar los conocimientos adquiridos en asignaturas cursadas de perfil electrónico y de instrumentación como son: *Dispositivos Electrónicos y Fotónicos*, *Electrónica Analógica I y II*, *Sistemas y Equipos de Medida*, *Instrumentación Electrónica*, *Sensores e Instrumentación Digital* o *Diseño de Productos Electrónicos*. El implementar un sistema real y verificar su funcionamiento ha hecho posible adquirir nuevos conocimientos de programación descubriendo otros entornos de software, como el XCTU y el compilador de Arduino. Además, se han ampliado los conocimientos adquiridos del software LabVIEW dotándome de una mayor capacidad de trabajo.

### 6.2 PROPUESTAS DE FUTURO

Una vez verificado el correcto funcionamiento de los diferentes sistemas y subsistemas que conforman el sistema de monitorización de medida de resistencia, se plantean a continuación una serie de ideas a desarrollar para mejorar el funcionamiento del sistema actual. Estas propuestas son:

1. Sustituir los módulos Arduino por los microcontroladores en los que se basan, prescindiendo de las extensiones que éstos poseen. Esta modificación se plantea con el propósito de abaratar los costes por unidad, ya que el precio de los módulos

Arduino empleados es de 20 €, mientras que un microcontrolador ATmega328/P tiene un coste de entorno a 2 €, según proveedor.

2. Acondicionar el sistema del nodo de medida para poder aislarlo de las interferencias introducidas por el deshumidificador.
3. Optimizar los códigos desarrollados para reducir tiempos de ejecución, y solucionar los problemas de sincronización entre nodo base y nodo de medida que daban lugar a pérdidas de información en algunas tramas.
4. Dotar de capacidad al nodo base para compartir los mensurandos por internet, de forma que se pueda acceder a esos datos telemáticamente.
5. Desarrollar la alarma de detección de seísmos.
6. Dotar al sistema con la capacidad de programar el funcionamiento del deshumidificador en función de los niveles de humedad deseados.

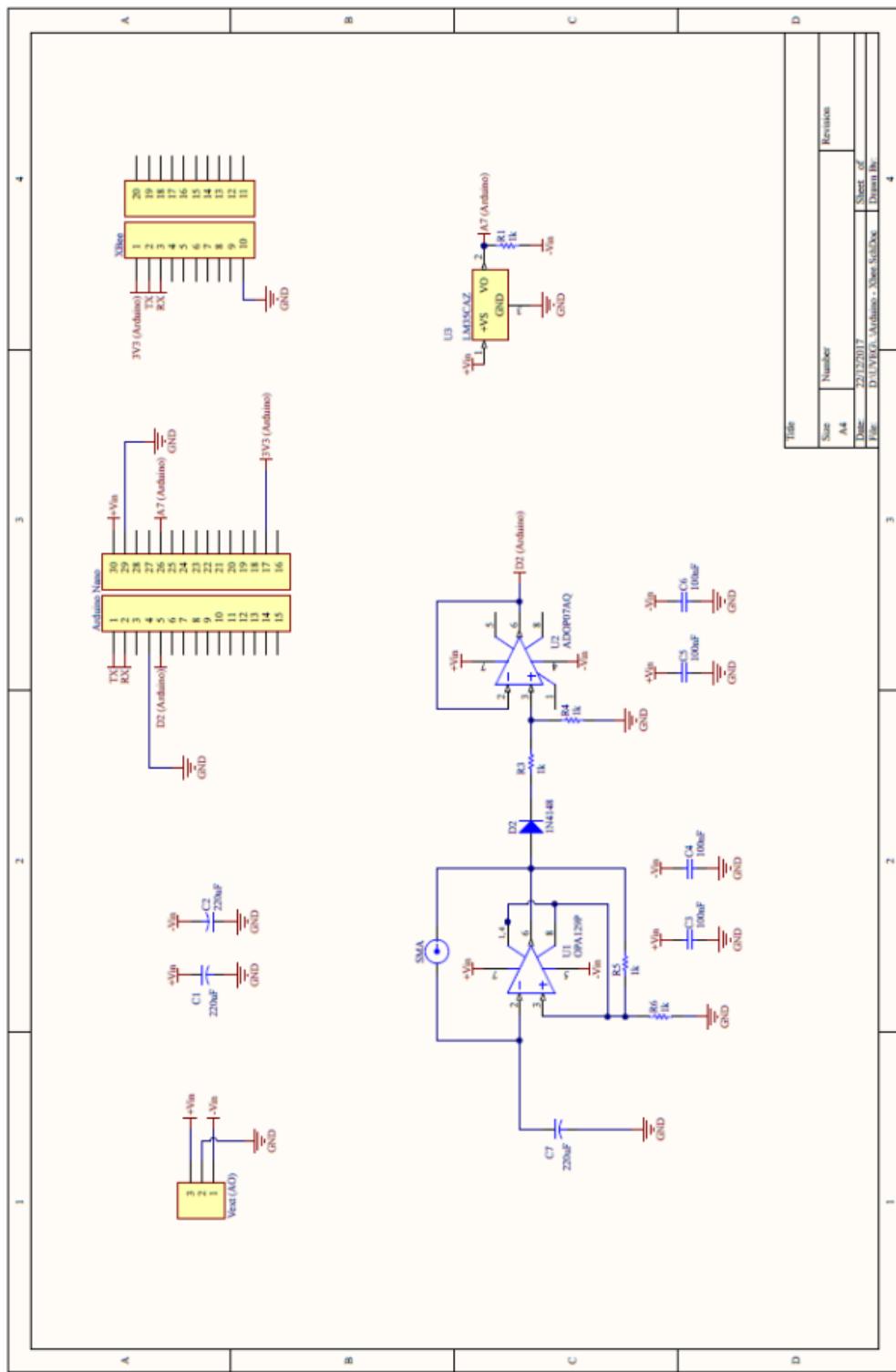
## 7. REFERENCIAS BIBLIOGRÁFICAS

---

- [1] (2016, Abril). Guía para el Control de la Humedad en el Diseño, Construcción y Mantenimiento de Edificaciones. U.S. Environmental Protection Agency. [Online]. Disponible en: [https://espanol.epa.gov/sites/production-es/files/2016-07/documents/moisture\\_control\\_guidance\\_spanish\\_april\\_2016\\_508\\_final.pdf](https://espanol.epa.gov/sites/production-es/files/2016-07/documents/moisture_control_guidance_spanish_april_2016_508_final.pdf)
- [2] C. Andrade. (2011, Mayo-Agosto). La resistividad eléctrica como parámetro de control del hormigón y de su durabilidad. ALCONPAT. Vol. 1 (2). 93-101. [Online]. Disponible en: <http://dx.doi.org/10.21041/ra.v1i2.8>
- [3] J.M.Cruz, J.Payá, L.F. Lalinde, I.C. Fita. (2011, Enero-Marzo). Evaluación de las propiedades eléctricas de morteros de cemento con puzolanas. Materiales de Construcción. [Online]. Vol. 61 (301). 7-26. Disponible en: <http://materconstrucc.revistas.csic.es/index.php/materconstrucc/article/view/216/262>
- [4] Chih-Yuan Chang. (2017, Mayo). Study on the Correlation between Humidity and Material Strains in Separable Micro Humidity Sensor Design. Sensors 2017. Vol. 17 (1066). Disponible: <https://www.mdpi.com/1424-8220/17/5/1066>
- [5] Atmel. (2016). ATmega328/P. Atmel. San Jose, CA, USA. [Online]. Disponible en: <https://datasheet.octopart.com/ATMEGA328P-MU-Microchip-datasheet-65729177.pdf>
- [6] Analog Devices. (2009). ADXL335. Analog Devices. Norwood, MA, USA. [Online]. Disponible en: <https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf>
- [7] Sensirion. (2008, Julio). Datasheet SHT1x. Sensirion. Laubisrütistrasse, Suiza. [Online]. Disponible en: [https://www.sparkfun.com/datasheets/Sensors/SHT1x\\_datasheet.pdf](https://www.sparkfun.com/datasheets/Sensors/SHT1x_datasheet.pdf)
- [8] Maxim Integrated. (2015, Marzo). Extremely Accurate I2C-Integrated RTC/TCXO/Crystal. Maxim Integrated. [Online]. Disponible en: <https://datasheets.maximintegrated.com/en/ds/DS3231.pdf>
- [9] Digi. (2018, Mayo). Zigbee RF Modules. Digi. [Online]. Disponible en: <https://www.digi.com/resources/documentation/digidocs/PDFs/90000976.pdf>
- [10] Faludi Robert, *Wireless Sensor Networks*, 1. Sebastopol: O'Reilly Media, Inc., 2010.
- [11] Arduino, «Arduino SD Wireless Shield» Arduino, [Online]. Disponible en: <https://store.arduino.cc/arduino-wirelss-sd-shield>. [Último acceso: Julio 2019].
- [12] Arduino, «ARDUINO UNO REV3» Arduino, [Online]. Disponible en: <https://store.arduino.cc/arduino-uno-rev3>. [Último acceso: Julio 2019].
- [13] Arduino, «ARDUINO NANO» Arduino, [Online]. Disponible en: <https://store.arduino.cc/arduino-nano>. [Último acceso: Julio 2019].

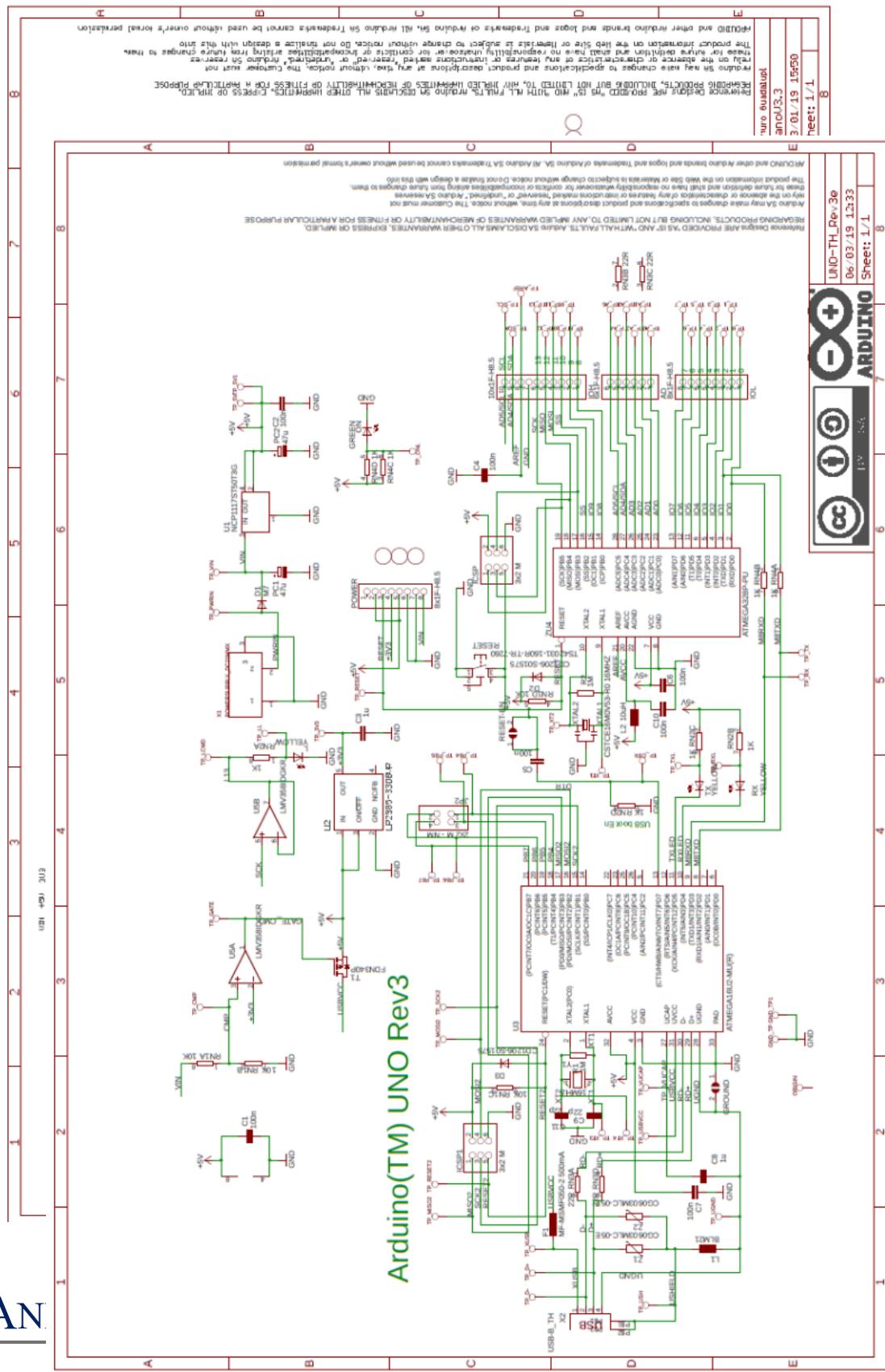
- [14] Arduino, «Language Reference» Arduino, [Online]. Disponible en: <https://www.arduino.cc/reference/en/>. [Último acceso: Julio 2019].
- [15] Wikipedia, «I<sup>2</sup>C» Wikipedia, [Online]. Disponible en: <https://es.wikipedia.org/wiki/I%2BC>. [Último acceso: Agosto 2019].
- [16] Wikipedia, «Serial Peripheral Interface» Wikipedia, [Online]. Disponible en: [https://es.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface](https://es.wikipedia.org/wiki/Serial_Peripheral_Interface). [Último acceso: Agosto 2019].
- [17] National Instruments, «¿Qué es LabVIEW?» National Instruments, [Online]. Disponible en: <https://www.ni.com/es-es/shop/labview.html> . [Último acceso: Septiembre 2019].

## ANEXO 1: ESQUEMÁTICO NODO DE MEDIDA



*Figura 43: Esquemático nodo de medida*

## ANEXO 2: ESQUEMÁTICO ARDUINO NANO



*Figura 45: Esquemático placa Arduino UNO*



## ANEXO 4: LAYOUT NODO DE MEDIDA

---

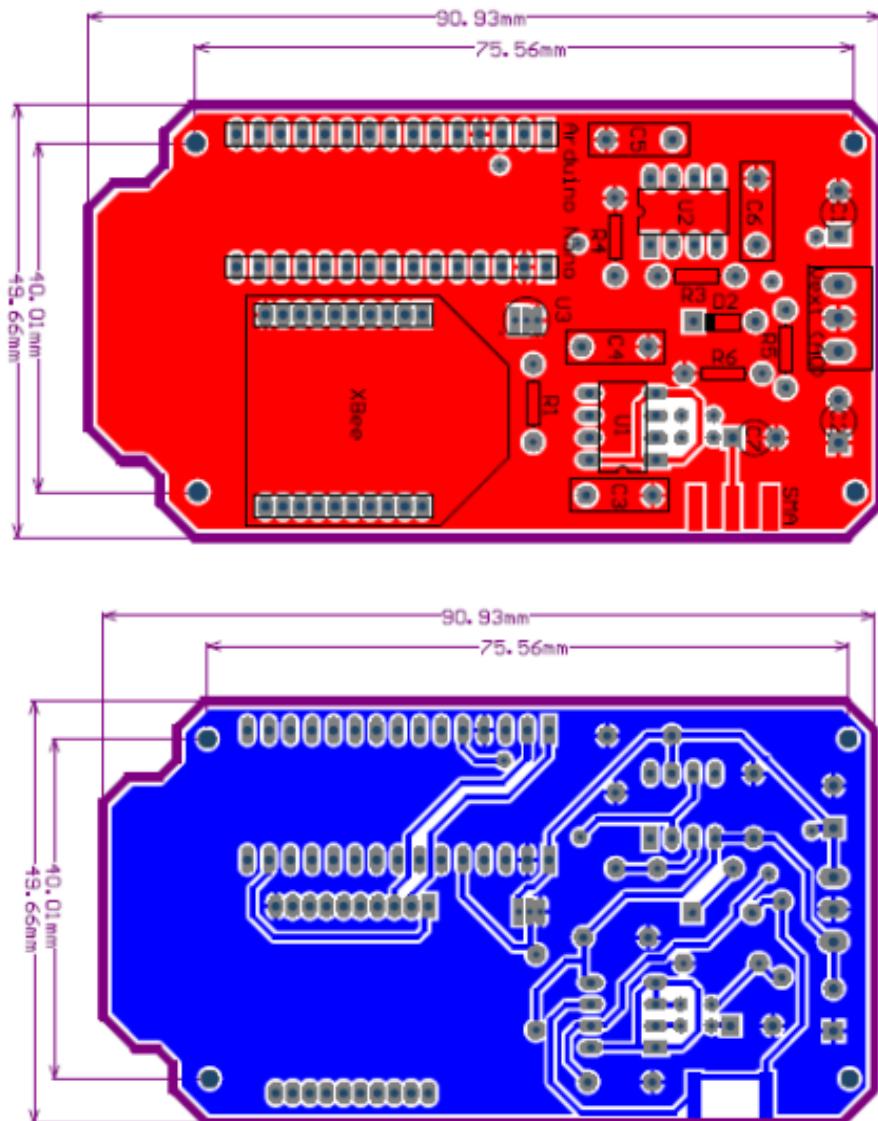


Figura 46: Layout placa nodo de medida

## ANEXO 5: CÓDIGO NODO BASE

---

```

#include <SPI.h>
#include <RTClib.h>
#include <RTC_DS3231.h>
#include <SD.h>
#include <Wire.h>

/*MODOS DE FUNCIONAMIENTO
X - MODO AUTONOMO
4 - MODO CONFIGURACIÓN TIEMPO DE CICLO
*/
//VARIABLES DE PROGRAMA
int comando;
int Tciclo;

//VARIABLES RTC
RTC_DS3231 rtc;
int dia;

//VARIABLES SD
File archivo;

void setup() {
Serial.begin(9600);
//CONFIGURACION RTC
Wire.begin();
rtc.begin();
if (!rtc.isrunning()) {
rtc.adjust(DateTime(__DATE__, __TIME__));
}
DateTime actual = rtc.now();
dia = actual.day();

//CONFIGURACION SD
if (!SD.begin(10)) {
Serial.println("Error en inicio de modulo SD ");
while(!SD.begin(10)){}
}
comando = 0;
Tciclo = 15; //DEFAULT 15, expresado en minutos
}

void loop() {
comando = 0;
int cambio = 0;
int i;
int j;
char inputByte;
String buff;
DateTime actual = rtc.now();

```

```

archivo = SD.open("DATA19.csv", FILE_WRITE); //ARCHIVO MEMORIA

/*********************MODO AUTONOMO*****************/
//BUCLE QUE SOLICITA LA MEDIDA CADA Tciclo
for (i = 0; (i < Tciclo) && (cambio == 0); i++) {
  for (j = 0; (j < 60) && (cambio == 0); j++) {
    comando = Serial.read();
    if(comando == 52){
      cambio = 1;
    }
    else {
      delay(1000);
    }
  }
  cambio = 0;
}
//CODIGO RECEPCION DE INFORMACION Y ALMACENAMIENTO EN SD
if(comando != 52){
  Serial.print("1"); //ORDEN DE LECTURA DIRIGIDA AL NODO DE MEDIDA
  delay(500); //TIEMPO DE ESPERA HASTA CALCULO DE LECTURAS
  if ((Serial.available()>0) && (Serial.peek()!=36)){
    while(Serial.available()>0){ //LECTURA DE LINEAS DE MENSURANDOS
      buff+=(char)Serial.read();
      delay(20);
    }
    archivo.print(actual.day());
    archivo.print("/");
    archivo.print(actual.month());
    archivo.print("/");
    archivo.print(actual.year());
    archivo.print(" - ");
    archivo.print(actual.hour());
    archivo.print(":");
    archivo.print(actual.minute());
    archivo.print(";");
    archivo.print(buff);

  } else{
    if(Serial.read() == 36){ // ERROR DE SENSOR HUMEDAD/TEMPERATURA
      archivo.print(actual.day());
      archivo.print("/");
      archivo.print(actual.month());
      archivo.print("/");
      archivo.print(actual.year());
      archivo.print(" - ");
      archivo.print(actual.hour());
      archivo.print(":");
      archivo.print(actual.minute());
      archivo.print(",Lectura erronea del de temperatura y humedad");
    }
  }
}

```

```

while(Serial.available()>0){
Serial.read();
    }})
buff = "";
}

//MODO CAMBIO DE TIEMPO DE CICLO - TECLA 4
if (comando == 52) {
while(Serial.available()<=0){}
while(Serial.available()>0){
buff += (char) Serial.read();
delay(20);
    }

Tciclo = buff.toInt();
buff = "";

archivo.print(actual.day());
archivo.print("/");
archivo.print(actual.month());
archivo.print("/");
archivo.print(actual.year());
archivo.print(" - ");
archivo.print(actual.hour());
archivo.print(":");
archivo.print(actual.minute());
archivo.print(";Tiempo de ciclo actualizado a ");
archivo.print(Tciclo);
archivo.print(" minutos\n");
}
}

```

## ANEXO 6: CÓDIGO NODO DE MEDIDA

---

/\*CÓDIGO ARDUINO NANO  
Adquirir las lecturas del acelerómetro en sus 3 ejes para determinar el angulo de inclinacion.  
Adquirir medidas de temperatura y humedad relativa.  
Adquirir medidas de resistencia electrica mediante tension alterna.

```

*/
//VARIABLES DE PROGRAMA
int comando; //0 - NO HACE NADA
//1 - LECTURA DE MENSURANDOS
volatile unsigned long tn;
volatile unsigned long t0;

```

```

//VARIABLES ACELERERÓMETRO EXPRESADAS EN CUENTAS
const int Xpin = A1;
const int Ypin = A2;
const int Zpin = A3;

//VARIABLES LECTURA TEMPERATURA, HUMEDAD Y RESISTENCIA
const int ClkPin = 4;
const int DataPin = 5;

const int ComandoTemp = 3;
const int ComandoHum = 4;
int val;
int ack;
int lectura;

volatile unsigned int TH;
volatile unsigned int TL;

void setup() {
Serial.begin(9600);
analogReference(DEFAULT);
pinMode(Xpin,INPUT);
pinMode(Ypin,INPUT);
pinMode(Zpin,INPUT);
comando = 0;
attachInterrupt(1, flanco_subida, RISING); //SE HABILITA LA SUBRUTINA PARA MEDIR
RESISTENCIA

}

void loop() {
if(Serial.available()>0){
if((Serial.peek() == 49)){
comando = Serial.read();

}

if(comando == 49){ //TECLA - 1
//LECTURA DE MENSURANDOS
//ACELERÓMETRO
Serial.print("X:");
Serial.print(analogRead(Xpin));
Serial.print(",");
delay(2);

Serial.print("Y:");
}
}
}

```

```

Serial.print(analogRead(Ypin));
Serial.print(";");
delay(2);

Serial.print("Z;");
Serial.print(analogRead(Zpin));
Serial.print(",");
delay(2);
//TEMPERATURA y HUMEDAD
//TEMPERATURA
sendCommandSHT(ComandoTemp, DataPin, ClkPin);
waitForResultSHT(DataPin);
val = getData16SHT(DataPin, ClkPin);
skipCrcSHT(DataPin, ClkPin);
Serial.print("Temp;");
Serial.print(val);
Serial.print(",");
val=0;

//HUMEDAD
sendCommandSHT(ComandoHum, DataPin, ClkPin);
waitForResultSHT(DataPin);
val = getData16SHT(DataPin, ClkPin);
skipCrcSHT(DataPin, ClkPin);
Serial.print("Hum;");
Serial.print(val);
Serial.print(",");
val=0;

//LECTURA DE RESISTENCIA
Serial.print("TH;");
Serial.print(TH);
Serial.print(",");
Serial.print("TL;");
Serial.print(TL);
Serial.println("");
}

comando = 0;
}//FIN DE PROGRAMA

```

```

/*INTERRUPCIONES*/
void flanco_subida () {
t0 = micros();
attachInterrupt(1, flanco_bajada, FALLING);
}

void flanco_bajada () {
tn = micros();

```

```

TH = tn - t0;
t0 = tn;
attachInterrupt(1, flanco_subida2, RISING);
}

void flanco_subida2 () {
tn = micros();
TL = tn - t0;
t0 = tn;
//attachInterrupt(1, flanco_bajada, FALLING);
}

/*FUNCIONES ADICIONALES*/
int shiftIn(int dataPin, int clockPin, int numBits)
{
int ret = 0;
for (int i = 0; i < numBits; ++i) {
digitalWrite(clockPin, HIGH);
ret = ret * 2 + digitalRead(dataPin);
digitalWrite(clockPin, LOW);
}
return (ret);
}

void sendCommandSHT(int command, int dataPin, int clockPin) {
int ack;
pinMode(dataPin, OUTPUT);
pinMode(clockPin, OUTPUT);
digitalWrite(dataPin, HIGH);
digitalWrite(clockPin, HIGH);
digitalWrite(dataPin, LOW);
digitalWrite(clockPin, LOW);
digitalWrite(clockPin, HIGH);
digitalWrite(dataPin, HIGH);
digitalWrite(clockPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, command);
digitalWrite(clockPin, HIGH);
pinMode(dataPin, INPUT);
ack = digitalRead(dataPin);
if (ack != LOW)
Serial.println("$");
digitalWrite(clockPin, LOW);
ack = digitalRead(dataPin);
if (ack != HIGH)
Serial.println("$");
}

void waitForResultSHT(int dataPin) {
int ack;
pinMode(dataPin, INPUT);

```

```

for (int i = 0; i < 100; ++i) {
delay(70);
ack = digitalRead(dataPin);
if (ack == LOW)
break;
}
if (ack == HIGH)
Serial.println("$ ");
}

int getData16SHT(int dataPin, int clockPin) {
int val;
pinMode(dataPin, INPUT);
pinMode(clockPin, OUTPUT);
val = shiftIn(dataPin, clockPin, 8);
val *= 256;
pinMode(dataPin, OUTPUT);
digitalWrite(dataPin, HIGH);
digitalWrite(dataPin, LOW);
digitalWrite(clockPin, HIGH);
digitalWrite(clockPin, LOW);
pinMode(dataPin, INPUT);
val |= shiftIn(dataPin, clockPin, 8);
return val;
}

void skipCrcSHT(int dataPin, int clockPin) {
pinMode(dataPin, OUTPUT);
pinMode(clockPin, OUTPUT);
digitalWrite(dataPin, HIGH);
digitalWrite(clockPin, HIGH);
digitalWrite(clockPin, LOW);
}

```