UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Medidor inalámbrico de la frecuencia respiratoria para pacientes COVID-19

Autor:
D. David Fernández Farto

Tutor:
D. Jesús Manuel Hernández Mangas

Valladolid, 9 de Septiembre de 2021

TÍTULO:	**Medidor inalámbrico de la frecuencia respiratoria para pacientes COVID-19**
AUTOR:	**D. David Fernández Farto**
TUTOR:	**D. Jesús Manuel Hernández Mangas**
DEPARTAMENTO:	**Electricidad y Electrónica**

TRIBUNAL

PRESIDENTE:	**Dña. Lourdes Pelaz Montes**
VOCAL:	**D. Jesús Manuel Hernández Mangas**
SECRETARIO	**D. Iván Santos Tejido**
SUPLENTE	**D. Jesús Arias Álvarez**
SUPLENTE	**Dña. Ruth Pinacho**

FECHA:	
CALIFICACIÓN:	

RESUMEN

Recientes estudios han relacionado el empeoramiento de pacientes de COVID-19 con el aumento de su frecuencia respiratoria. Lamentablemente, no existen en el mercado sistemas de monitorización inalámbrica de dicha frecuencia respiratoria que sean de un coste reducido. El presente documento plantea una solución que resuelve este problema.

El resultado es un cinturón con un sensor que, abrochado en torno al vientre o pecho del paciente, es capaz de detectar cuándo se produce una inhalación o exhalación, de forma que se pueden cuantificar el número de respiraciones por minuto. Cada cierto período de tiempo, la información capturada por el sensor es enviada vía WiFi a un servidor, integrándose en una base de datos a partir de la cual se pueden generar, de manera sencilla, diferentes gráficas que faciliten el seguimiento de la evolución respiratoria del paciente. Además, el sensor cuenta con una pantalla, por lo que se puede ver en tiempo real el valor del número de respiraciones por minuto.

Palabras clave

Frecuencia respiratoria, ESP32, QMC5883L, MQTT, Arduino, MariaDB

ABSTRACT

Recent studies have shown that there is a relationship between the worsening of COVID-19 patients and the increase in their breathing rate. Unfortunately, there are no wireless monitoring devices in the market with a reduced price. This paper provides a solution to this problem.

The result is a belt with a sensor, which has to be fastened either around the abdomen or chest of the patient. Its function is to detect when an inhalation or exhalation takes place in order to quantify the number of breaths per minute. Every certain amount of time, the information is sent via WiFi to a server, where it is stored in a database. This database enables the generation of different graphs in order to follow the breathing evolution of the patient. In addition, the sensor includes a screen so that the number of breaths per minute can be seen in real time.

Keywords

Breathing rate, ESP32, QMC5883L, MQTT, Arduino, MariaDB

Agradecimientos

El Trabajo Fin de Grado ha sido el broche final a cuatro de años de carrera de los cuales, pese a los muchos momentos duros, puedo decir, sin lugar a duda, que forman parte de una de las mejores etapas de mi vida. A continuación, quería mostrar mis agradecimientos a todas aquellas personas sin las cuales no habría sido posible llegar hasta aquí.

En primer lugar, a mis padres, por la educación que me han brindado desde pequeño. Además, junto a mi hermano, han sido las personas con las que he podido poner a prueba el prototipo desarrollado. En segundo lugar, a Irene, por su apoyo diario e incondicional tanto en el plano académico como en el personal. Seguidamente, a mis compañeros y amigos de la carrera por haber hecho que esta etapa haya sido algo mucho más que estudiar. Tampoco quiero olvidarme de mi grupo de amigos de toda la vida, los cuales me han hecho disfrutar al máximo de mi tiempo libre.

Por último, mi más profundo agradecimiento para mi tutor Jesús. Gracias por haberme guiado con tanto interés y entusiasmo a lo largo de la realización del presente trabajo.

Índice de contenidos

o

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

La problemática surgida por la pandemia de coronavirus que ha azotado a nuestro país ha puesto de manifiesto la falta de recursos y de equipamiento en muchos hospitales. Concretamente, el área que nos atañe es el de la neumología, debido a que se ha descubierto la existencia de una estrecha relación entre el empeoramiento de pacientes afectados por COVID-19 y el aumento de su frecuencia respiratoria.

En la actualidad, la mayoría de hospitales disponen de tan solo dos formas diferentes de medir la respiración de los pacientes. La más simple consiste en contar el número de inspiraciones durante un minuto, bien sea utilizando un fonendoscopio o, directamente, colocando la mano en la zona tóraco-abdominal del paciente. Parece evidente que es un método bastante rudimentario, además de poco preciso, puesto que solo se tiene en cuenta una muestra de un minuto. La otra técnica, más precisa, es la que se utiliza para los pacientes que están ingresados y consiste en la monitorización del paciente por medio de la colocación de electrodos en su cuerpo. El problema se encuentra en que el precio de un equipo de estas características ronda los mil euros.

Por tanto, el diseño de un dispositivo de bajo coste para la monitorización inalámbrica de la frecuencia respiratoria supondría un gran avance en materia de equipamiento hospitalario. Además, la sucesiva información recogida por el sensor podría ser enviada a un servidor, de forma que los datos quedarían registrados digitalmente facilitando su tratamiento posterior para, por ejemplo, la observación de la evolución respiratoria del paciente o la inclusión de la información en la historia clínica del mismo.

1.2. Estado del arte

Términos como la salud digital, la eSalud o la telemedicina se utilizan hoy en día indistintamente para hablar de un mismo concepto. Sin embargo, la Organización Mundial de la Salud (OMS) considera diferentes significados para cada una de estas palabras. Por un lado, define eSalud como “el uso de las tecnologías de la información y las comunicaciones (TIC) como soporte en los campos de salud y los relacionados con la salud”, mientras que habla de la salud digital como un “concepto englobador que incluye nuevas disciplinas como la computación avanzada en *big data* o en genómica, y la inteligencia artificial” [1].

En cuanto a la telemedicina, la OMS reserva su utilización para el "uso de las TIC en la prestación a distancia de servicios asistenciales de salud" [1]. El origen de este término se remonta a la década de los cincuenta del siglo XX. Durante este período, numerosos investigadores de diferentes universidades del mundo comenzaron a interesarse en la aplicación de principios de ingeniería para la resolución de problemas médicos. Al apoyarse en el uso de las TIC, sus avances han sido paralelos al desarrollo tecnológico, experimentando un gran progreso en los últimos años [2].

La Agenda Digital para Europa 2020 tenía como acción clave la salud digital, bajo la premisa de que las TIC debían ayudar al sistema sanitario para paliar un problema que empezaba a ser serio y urgente, y que no es otro, que el de la saturación hospitalaria [3]. Justo en ese mismo año, la OMS declararía el inicio de la pandemia de COVID-19, la cual pondría aún más de manifiesto la mencionada necesidad de acelerar el proceso de digitalización del sistema sanitario.

En España, como en otros muchos países, la presencia del virus obligó al confinamiento domiciliario de la población durante un largo período de tiempo, por lo que se tuvo que recurrir, de forma inmediata, al uso de la telemedicina. Tras quedar patentes sus múltiples beneficios, se podría decir que esta forma de prestación de servicios médicos ha llegado para quedarse. Algunas de estas ventajas son:

- Se evitan desplazamientos innecesarios, descongestionando los centros sanitarios y proporcionando comodidad al paciente, puesto que no tiene que salir de casa.
- Permite a los habitantes de áreas despobladas o remotas contar con todo tipo de especialistas, garantizando así una equidad en el acceso a los servicios de salud que puede ayudar a frenar otro problema social como es el éxodo rural.
- Se optimizan los tiempos de respuesta, reduciendo los tiempos de espera y mejorando la asistencia proporcionada al paciente.

Todo esto propició que el Gobierno español decidiera crear una nueva secretaría, la Secretaría General de Salud Digital, Información e Innovación del Sistema Nacional de Salud, con el objetivo de abordar los proyectos de modernización, mejora y transformación del Sistema Nacional de Salud [4].

En línea con el apartado de motivación anterior, el desarrollo de sensores inalámbricos para uso médico resultaría un primer paso hacia un objetivo futuro de contar con "hospitales inteligentes", hospitales que, gracias al Internet de las Cosas (IoT), tengan interconectados todos sus dispositivos.

1.3. Objetivos y requisitos

El objetivo es diseñar y desarrollar un dispositivo de bajo coste que permita la monitorización en tiempo real de la frecuencia respiratoria de un ser humano. Además, se pretende que la información recogida sea enviada a un ordenador para su integración en una base de datos. Por tanto, se deben cumplir una serie de requisitos que pueden dividirse en cuatro materias: nodo principal, sensor, comunicación bidireccional y base de datos.

Para el nodo principal:

- Se quiere tener un dispositivo económico, por lo que el coste del nodo, así como del resto de los componentes, debe ser bajo.
- Se necesita conectividad, idealmente vía WiFi, para transmitir la información recogida.
- Debe contar con un conector de batería para poder dotar al dispositivo de autonomía.
- Interesa que pueda utilizarse un entorno de desarrollo integrado (IDE) de código abierto que cuente con un buen soporte a nivel de comunidad.
- Sería recomendable que dispusiera de una pantalla, de manera que se pudiera obtener algún tipo de realimentación de forma directa.

El sensor debe ser capaz de detectar cada una de las inspiraciones que se realicen. Para ello se plantean dos posibilidades, una mecánica, consistente en capturar el movimiento torácico, y otra acústica, a partir del ruido que se produce al respirar.

En cuanto a la comunicación bidireccional, además de que el sensor envíe mensajes al servidor, se quiere que la comunicación pueda realizarse de forma inversa, de tal manera que los parámetros de configuración del dispositivo puedan ser modificados en directo a tenor de las consideraciones del especialista que lo supervise. Por ejemplo, que se pueda variar el período de tiempo tras el cual se envían los datos al servidor.

Para el manejo de la base de datos, el lenguaje utilizado para desarrollar el *software* debe ser compatible con SQL, que es el lenguaje de dominio específico utilizado en programación para administrar, y recuperar información, de sistemas de gestión de bases de datos [5].

1.4. Proceso creativo y propuesta

El primer paso del proyecto era encontrar la forma de detectar cuándo se produce una inspiración, es decir, qué sensor utilizar y cómo utilizarlo. La opción acústica se descartó rápidamente, puesto que filtrar el sonido producido por la respiración frente a todo el ruido que lo rodea es prácticamente imposible. Por tanto, había que centrarse en cómo poder detectar mecánicamente cada elevación pectoral.

Tras un proceso de investigación se dio con la posibilidad de determinar los movimientos tóraco-abdominales por medio de la colocación de un sensor sobre una banda que vaya en torno al tórax o abdomen del paciente. Dicho sensor puede ser, entre otros, un acelerómetro o un sensor magnético. El único requisito es que sea capaz de correlacionar la expansión del volumen tóraco-abdominal con la fase respiratoria [6].

En primer lugar, se planteó el uso de un acelerómetro, con el objetivo de que, a partir de la aceleración provocada por el movimiento de la caja torácica, se obtuviera la velocidad y, a partir de esta, el tiempo. El problema está en que la aceleración provocada por dicho movimiento es ínfima frente a la aceleración fruto de la gravedad, por lo que se descartó el uso de un sensor de estas características.

A continuación, se pensó en utilizar un sensor magnético, que a la postre acabaría siendo el sensor escogido. Los sensores magnéticos funcionan a partir de la presencia de un campo magnético, por lo que en la banda se tiene que colocar un imán que lo genere. En función de la distancia en la que se encuentre el sensor respecto al imán, se podría conseguir establecer una relación directa entre la magnitud del campo magnético capturada por el sensor y el movimiento tóraco-abdominal. De esta idea nacería la propuesta final, representada en la Figura 1.1, consistente en colocar una cinta elástica entre el sensor y el imán, de forma que ambos elementos se acercasen y alejasen al ritmo de la respiración.

Figura 1.1: Representación esquemática del dispositivo.

1.5. Recursos utilizados

En cuanto al *hardware*, en un primer momento se pensó en utilizar un módulo ESP8266 como nodo principal, debido a dos motivos principales, su bajo coste, inferior a 10 euros, y su compatibilidad con Arduino, un IDE con numerosas bibliotecas *software*. El inconveniente que presentan estos módulos es que solo tienen un pin para el bus I2C, y los sensores magnéticos o magnetómetros necesitan dos. Los módulos ESP32, sucesores de los ESP8266, sí que satisfacen este requisito. Tanto ESP8266 como ESP32 son nombres de familias de chips SoC (*System on a chip)* desarrollados por Espressif Systems [7].

Para la elección del módulo ESP32 más conveniente, se hizo un primer cribado a partir de los modelos que disponían de pantalla. A continuación, se seleccionaron aquellos que contaban con un conector para la alimentación mediante batería LiPo (polímero de litio) y, finalmente, se eligió el dispositivo de menor coste. El modelo escogido fue el TTGO T-Display V1.1 de la marca LILYGO en su versión de 4MB de memoria flash.

Como sensor magnético se optó, en base a su precio, por el módulo GY-273, el cual cuenta con un chip con tecnología magnetorresistiva. La magnetorresistencia es una propiedad que tienen algunos conductores metálicos o semiconductores de variar su resistencia eléctrica cuando son sometidos a un campo magnético [8].

Por otro lado, respecto al protocolo de comunicación, se ha optado por MQTT, el protocolo por excelencia en el mundo del IoT, debido a su sencillez y ligereza. El tamaño de la cabecera de sus mensajes no supera los 5 bytes, por lo que la sobrecarga introducida es mínima, haciendo que sea manejable por cualquier dispositivo por muchas limitaciones de potencia, consumo o ancho de banda que tenga. MQTT se define como un protocolo de comunicación M2M (*machine to machine*) basado en la pila TCP/IP. Una máquina actúa como cliente mientras que otra actúa como servidor o bróker. El funcionamiento se basa en un servicio de mensajería *push* con patrón publicador/suscriptor, en donde el cliente puede publicar mensajes en el bróker o suscribirse a los mensajes publicados en él. Dentro del bróker los mensajes se organizan en *topics* [9]. Para ilustrar esto mediante un ejemplo, imaginemos una casa domótica en la que tenemos un frigorífico y una lavadora conectados a Internet que envían mensajes de su estado, vía MQTT, a un PC en

el que se ha instalado un bróker. Mediante un *smartphone*, el dueño del hogar quiere consultar el estado de sus dos electrodomésticos, de forma que va a haber tres clientes conectados al bróker, dos publicando y otro suscrito. El frigorífico publicará sus mensajes en un *topic* diferente al de la lavadora, de manera que el usuario desde su teléfono se suscribirá a uno u otro *topic* en función de lo que quiera consultar.

Aunque no se vaya a hacer uso de ello, cabe destacar que MQTT dispone de un mecanismo de calidad de servicio de 3 niveles, así como de distintas medidas de seguridad, por ejemplo, transporte SSL/TLS o autentificación de usuario [9].

1.6. Estructura del documento

A continuación, se describen las partes de las que consta el presente documento, el cual se encuentra dividido en tres capítulos, siendo este el primero de ellos. En el segundo capítulo se habla sobre los diferentes elementos que constituyen el *hardware* del dispositivo, mientras que en el Capítulo 3 se describe todo lo relativo al *software*, programas, IDEs y lenguajes utilizados. Por último, en el Apéndice A se recoge el código de los distintos ficheros mencionados a lo largo del tercer capítulo, mientras que en el Apéndice B se incluye una breve guía de instalación de los programas e IDEs utilizados.

Capítulo 2

Hardware

2.1. Visión general

Este capítulo se centra en el apartado *hardware*, especificaciones de los componentes utilizados y montaje. Para esto último, es importante tener en cuenta que el dispositivo está destinado a un entorno hospitalario, por lo que el diseño debe estar preparado para ser sometido a procesos de desinfección. En el apartado final del capítulo se incluye el presupuesto económico del dispositivo.

2.2. ESP32 TTGO T-Display V1.1

Como nodo principal se ha utilizado el módulo ESP32 TTGO T-Display V1.1 de la marca LILYGO, concretamente el modelo con un tamaño de memoria flash de 4MB.

El ESP32 es un SoC diseñado por la compañía china Espressif y fabricado por TSMC que integra en un único chip un procesador de doble núcleo de 32bits a 160MHz [8]. En cuanto al módulo, en la página web oficial de la empresa LILYGO [10] se pueden encontrar tanto las especificaciones como el *pinout* y las medidas del producto. A continuación, se adjuntan algunas de las características más relevantes acorde al uso que se le quiere dar:

- Módulo de conexión inalámbrica WiFi 802.11 b/g/n, lo cual permite la conexión a Internet.
- Pila TCP/IP instalada, por lo que se puede utilizar el protocolo MQTT de forma sencilla.
- Pantalla TFT LCD de 1.14 pulgadas.
- Dos botones y un botón de *reset*.
- Conector JST de dos pines, de forma que se puede conectar una batería que lo dote de autonomía.
- Puerto USB tipo C para la carga de la batería y la depuración del software.
- Dos pines GPIO para el bus I2C, el 21 y 22 mostrados en la Figura 2.1, lo cual permite conectar con un magnetómetro.
- Dimensiones: 51.52 mm x 25.04 mm x 8.54 mm.

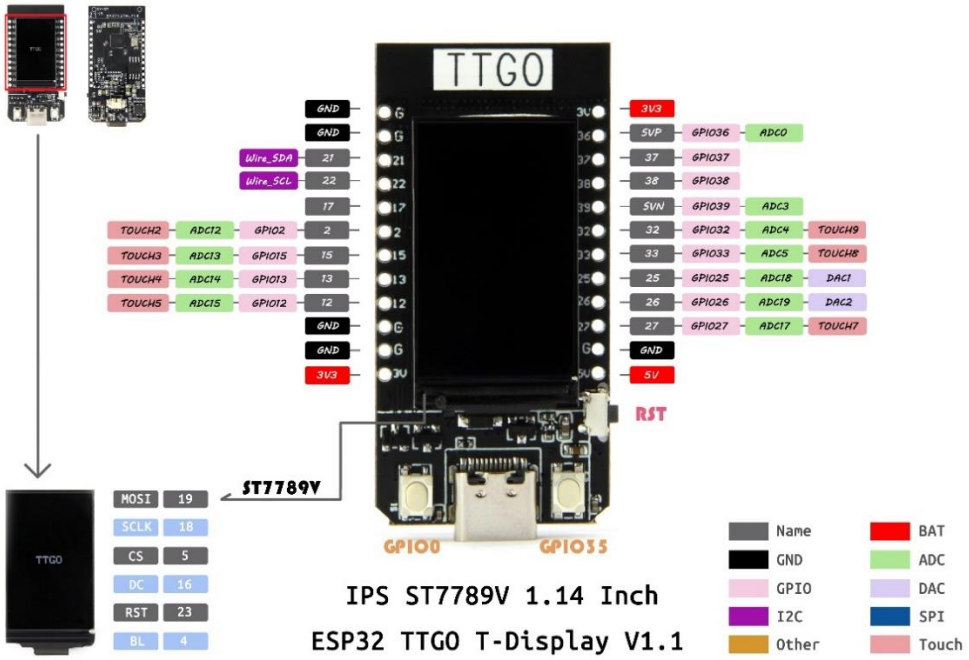

Figura 1.1: Pinout del ESP32 TTGO T-Display V1.1 [10].

2.3. Sensor GY-273 QMC5883L

Como magnetómetro se ha utilizado el módulo GY-273 del fabricante ARCELI, el cual no proporciona ningún tipo de documentación del producto. A la vista de la Figura 2.2, se observa que tiene cinco pines y, pese a que aparece impreso una serigrafía con el nombre HMC5883L, cuenta en realidad con un chip QMC5883L. Tanto uno como otro son chips de superficie que incorporan un sensor de magnetorresistencia capaz de medir el valor del campo magnético en tres ejes. A continuación, se muestran algunas especificaciones extraídas del *datasheet* del QMC5883L [11]:

- Interfaz I2C con dirección 0x0D.
- Dispone de un modo de muestreo continuo en el cual el magnetómetro está constantemente realizando mediciones y actualizando los registros x, y, z de los ejes.
- Voltaje de alimentación entre 2.16 y 3.6 V.
- Rango de valores de los datos de los registros de salida entre -32768 y 32767.
- Dimensiones: 18.5 mm x 13.55 mm x 1 mm.

2.4. Batería

En el apartado de las baterías existe un compromiso entre la capacidad, el tamaño y el precio. El objetivo es que el dispositivo sea de bajo coste y poco invasivo para el paciente, por lo que hay que decantarse por una batería de reducido tamaño, a pesar de que esto limite la autonomía. Concretamente, se ha optado por una batería LiPo recargable de 1100 mAh del fabricante MakerFocus, cuyas dimensiones, mostradas en la Figura 2.3, son muy similares a las del TTGO.

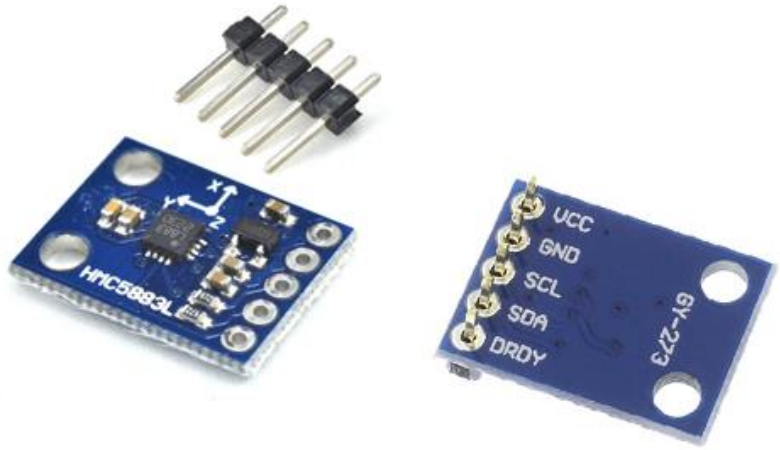

Figura 2.2: Módulo GY-273 ARCELI [12].

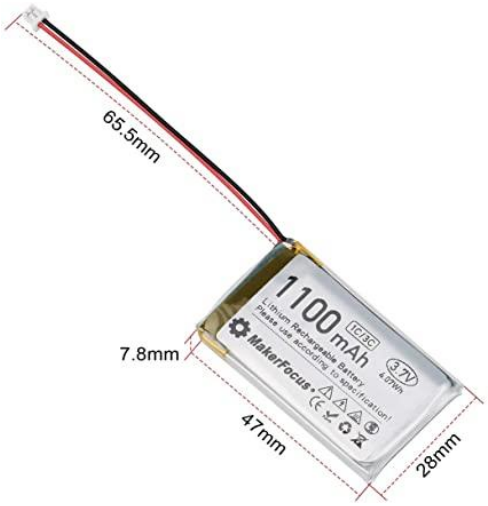

Figura 2.3: Batería 1100 mAh MakerFocus[13].

2.5. Imán

El imán escogido es un imán de neodimio de forma circular, cuyo radio es inferior a los 10 mm, como puede apreciarse en la Figura 2.4.

Figura 2.4: Tamaño del imán [14].

2.6. Montaje

En total, en este capítulo se ha hablado de cuatro componentes distintos, de los cuales los tres primeros forman un bloque, puesto que la batería y el magnetómetro van conectados al nodo principal. Viendo los nombres de los pines de las Figuras 2.1 y 2.2, se puede intuir el conexionado a realizar entre el GY-273 y el TTGO, el cual se muestra en la Figura 2.5. Este esquema ha sido generado mediante el programa Proteus [15]. En la Figura 2.6 se observa el resultado obtenido tras soldar con estaño los cuatro cables presentados en la Figura 2.5.

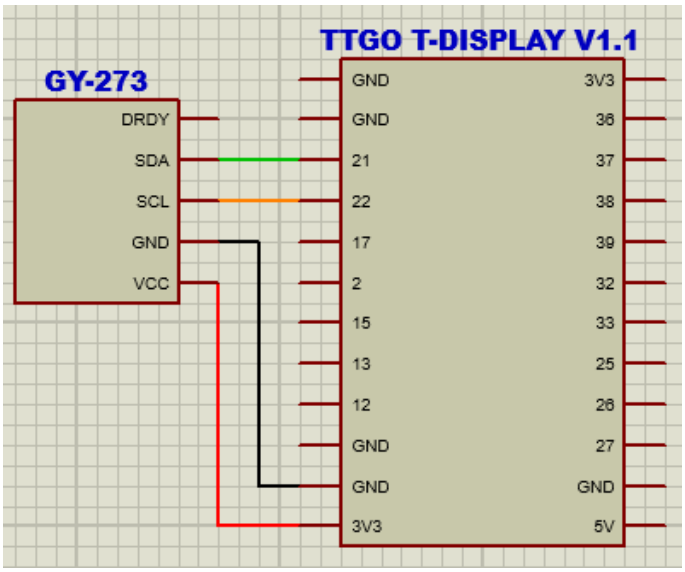

Figura 2.5: Esquema de conexión del nodo principal y el sensor.

Haciendo uso de una herramienta *online* de diseño 3D llamada Tinkercad [16] se han creado dos piezas, una que albergue el nodo principal, sensor y batería, y otra que contenga al imán. Los diseños resultantes se muestran en la Figura 2.7. En ambos casos se tiene una plancha con ranuras a los lados. En la pieza destinada al imán se ha colocado un pequeño cilindro hueco en la parte central de la plancha, el cual queda sellado al colocar la tapa, impidiendo que el imán se salga. En la otra pieza se han levantado cuatro paredes sobre la plancha, de forma que se tiene una caja cuya base sobresale por los extremos. En una de las paredes se ha hecho una salida para el puerto USB del nodo principal, mientras que en la tapa se ha hecho una ventana para que la pantalla quede visible. La hendidura rectangular de la plancha se corresponde con el tamaño de la batería. De esta forma, se reduce un par de milímetros la altura de la caja, puesto que la batería se coloca en la parte inferior de la caja, debajo del nodo, como se ve en la Figura 2.8. Ambas piezas han sido impresas con plástico PLA rojo mediante una impresora Anycubic i3 Mega [17].

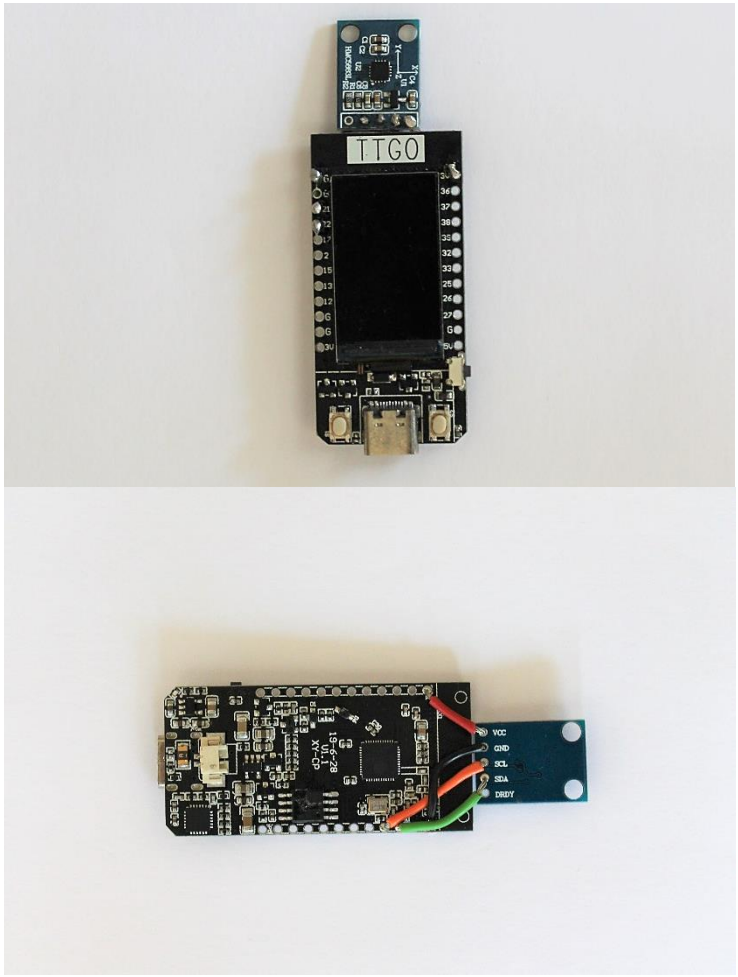

Figura 2.6: Conexión del nodo principal y el sensor.

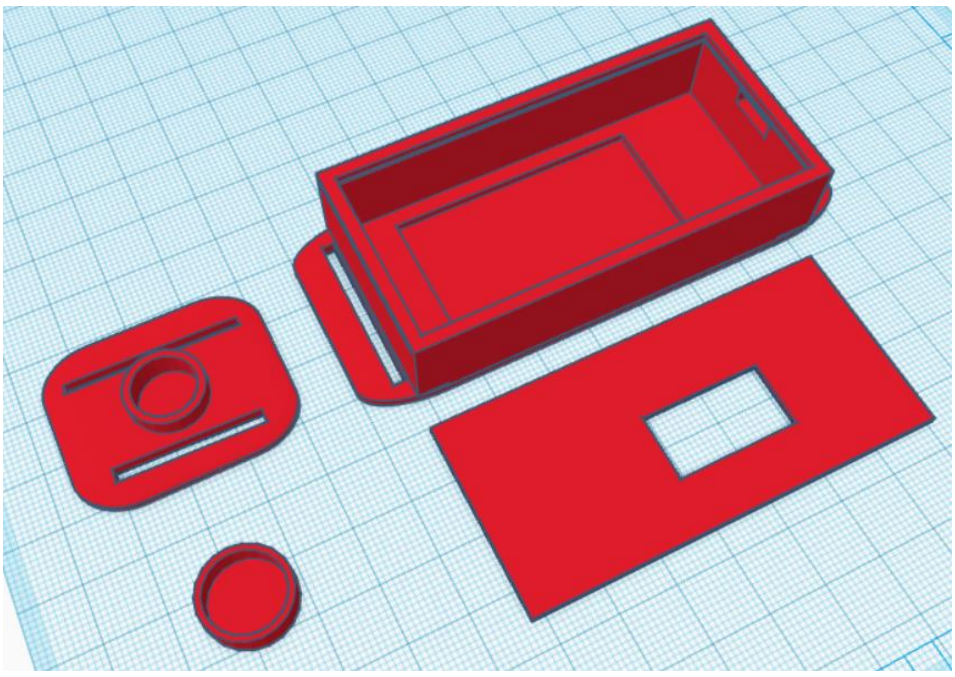

Figura 2.7: Diseños 3D realizados mediante Tinkercad.

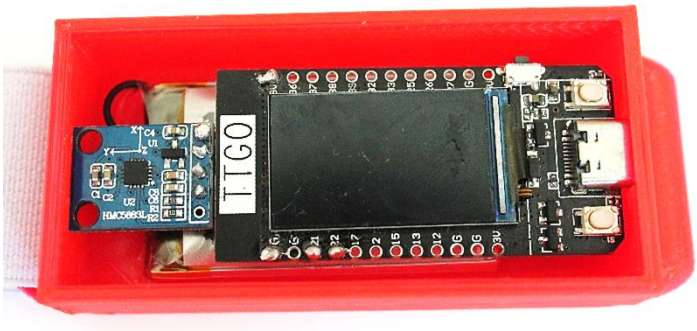

Figura 2.8: Disposición del nodo principal, sensor y batería dentro de la pieza.

Tomando como referencia el esquema de la Figura 1.1, en el prototipo se pueden distinguir cuatro partes. Por un lado, las dos piezas diseñadas, y por otro, las dos cintas. La representada en color negro es una cinta rígida que hace las veces de correa del cinturón, mientras que la pintada en azul es la cinta elástica que permite el acercamiento y alejamiento de las piezas al ritmo de la respiración. Para el cierre y ajuste del cinturón se han cosido varios trozos de velcro en la correa, de tal forma que esta cinta puede ser separada del resto del dispositivo para su higienización. La Figura 2.9 muestra el dispositivo, al completo, abrochado en torno al pecho de un varón. El extremo con menor proporción de velcro es el que se emplea para que la correa quede unida a una de las piezas diseñadas, Figura 2.10, mientras que el otro extremo es el que se utiliza para ajustar el cinturón en función de la anchura del paciente, Figura 2.11. El resto del dispositivo, Figura 2.12, va a estar siempre unido, puesto que la cinta elástica queda cosida uniendo a las dos piezas diseñadas. Al contar estas con tapa, los componentes albergados pueden ser extraídos en caso de que el proceso de desinfección llevado a cabo pudiese dañarlos.

Figura 2.9: Dispositivo en funcionamiento.

Figura 2.10: Extremo de fijación de la correa.

Figura 2.11: Extremo de ajuste de la correa.

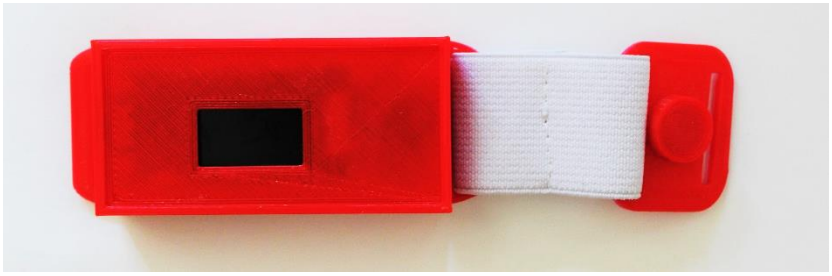

Figura 2.12: Las dos piezas diseñadas quedan permanentemente unidas por la cinta elástica.

2.7. Presupuesto económico

En primer lugar, la Tabla 2.1 recoge el coste material de cada uno de los componentes adquiridos para el montaje del presente dispositivo. Datos reales a día 3 de mayo de 2021. Cabe mencionar que no se han considerado ni los costes indirectos derivados de la necesidad de tener una impresora 3D, un soldador o una máquina de coser, ni costes ínfimos como los cuatro trocitos de cable y el estaño necesarios para la conexión del nodo principal al sensor.

La Tabla 2.2 muestra el importe del dispositivo desarrollado, lo cual incluye el coste anterior de los materiales y el coste del diseño, es decir, las horas de ingeniería empleadas. Patentado el prototipo, el precio que adquiriría la producción de un dispositivo englobaría solo el coste de los materiales y el costo de un operario con conocimientos de costura y soldadura que se encargarse del montaje. Esto se muestra en la Tabla 2.3.

CONCEPTO	PROVEEDOR	REFERENCIA	€ / CANTIDAD
TTGO T-Display V1.1	Banggood	1522925	10.97 / 1 unidad
GY-273	Amazon	B07DN1862K	6.99 / 1 unidad
Batería 1100 mAh	Amazon	B0867KDMY7	5.49 / 1 unidad
Imán	AliExpress	1005001536647667	0.16 / 1 unidad
Cinta rígida	Mercería local		1 / 1 metro
Cinta elástica	Mercería local		0.1 / 0.1 metros
Velcro	Mercería local		1 / 0.5 metros
PLA	LEON3D		0.4 / 18 gramos
		TOTAL	26.11 €

Tabla 2.1: Presupuesto de los materiales.

CONCEPTO	€ / H	HORAS	TOTAL €
Materiales			26.11
Ingeniero	20	300	6000
Operario	11	0.25	2.75
		TOTAL	6028.86 €

Tabla 2.2: Presupuesto de fabricación del producto final.

CONCEPTO	€ / H	HORAS	TOTAL €
Materiales			26.11
Operario	11	0.25	2.75
		TOTAL	28.86 €

Tabla 2.3: Presupuesto de fabricación de un dispositivo.

Finalmente, se plantean los costes de una producción en cadena si el precio de venta del dispositivo fuera de 35 €. Bajo una hipotética situación en la cual el Ministerio de Salud decide adquirir 10 dispositivos para cada uno de los 467 hospitales públicos que hay en nuestro país [18], el beneficio obtenido sería el mostrado en la Tabla 2.4.

CONCEPTO	€ / UNIDAD	UNIDADES	TOTAL €
Coste de materiales	26.11	4670	121933.7
Coste de diseño	6000		6000
Coste de operario	2.75	4670	12842.5
Costes indirectos	300		300
Precio dispositivo	35	4670	163450
		BENEFICIO	22373.8 €

Tabla 2.4: Producción en cadena.

Los 300 euros de costes indirectos son una aproximación del gasto que supondría la adquisición de una impresora 3D, un soldador, una máquina de coser, cables y estaño. El beneficio total es el resultado de restar al total del precio dispositivo, 163450 €, los demás conceptos. Evidentemente, si todos los materiales hubieran sido adquiridos en distribuidores asiáticos y se hubiera considerado que a mayor número de componentes adquiridos menor precio, el beneficio sería mayor.

Capítulo 3

Software

3.1. Visión general

En este último capítulo del documento se van a tratar todos los aspectos relativos al *software*. En el Apéndice A se incluye el código de los tres ficheros desarrollados, mientras que en el Apéndice B se adjunta una guía para la descarga e instalación de los diferentes programas que se mencionarán a lo largo del presente capítulo.

En primer lugar, se va a exponer de forma general cómo es el funcionamiento del sistema. Al ser el prototipo un dispositivo inalámbrico, se quiere prolongar al máximo su autonomía, por lo que el *software* desarrollado para el nodo principal va a seguir una dinámica repetitiva consistente en trabajar y dormirse. Esta idea es la que se representa en la Figura 3.1, donde "t_envio" simboliza el período de tiempo que tiene que transcurrir antes de realizar un envío al bróker. En el caso de que el tiempo entre envíos fuera de 60 segundos, la autonomía del dispositivo sería de algo más de 3 días, como se puede ver en la Figura 3.2. La fase de establecimiento de conexiones y envío supone siempre unos 5 segundos, por lo que el resto de los 60 segundos serían de fase de trabajo-sueño.

De cara al usuario, cuando se inicializa el dispositivo por primera vez, el nodo principal se pone en modo de acceso AP (*Access Point*), es decir, se convierte en un punto de acceso, generando una red WiFi a la cual se puede uno conectar desde un smartphone. En este caso, como se observa en la Figura 3.3, se le ha dado el nombre de "RESP01" al SSID.

Si se ha introducido de forma correcta la contraseña, llegará una notificación de inicio de sesión como la mostrada en la Figura 3.4. Tras pulsarla, se iniciará un portal cautivo y aparecerá una ventana, como la de la Figura 3.5, en la que se indican los pasos a seguir, cuyo resultado se muestra en las Figuras 3.6 y 3.7. Una vez completados los parámetros de configuración que aparecen en la última Figura mencionada, simplemente se ha de pulsar en el botón "Guardar" y, a continuación, en el "Desconectar" que se observaba en la Figura 3.6. Realizado esto, los valores introducidos quedan almacenados en el sistema de ficheros del módulo ESP32 y se produce un reinicio. Al iniciarse, se realiza una comprobación de la existencia de datos guardados, lo cual en esta ocasión es cierto, por lo que se obvian los pasos mencionados de conversión en un punto de acceso y lanzamiento del portal cautivo. En esta situación el módulo se mantiene en un modo de acceso denominado STATION, en el cual busca una red WiFi a la que conectarse. Establecida la conexión con la red que se indicó en la Figura 3.7, se inicia el bucle mostrado en la Figura 3.1

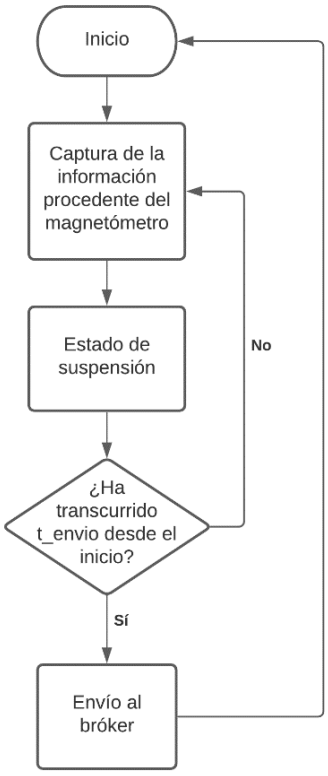

Figura 3.1: Diagrama de flujo del funcionamiento general del *software* implementado en el nodo principal.

- t_envio: 60 s
- Fase de sueño-trabajo: 11.2 mA
- Fase de establecimiento de conexiones y envío: 45.21 mA
- Capacidad de la batería: 1100 mAh

$$\text{Consumo medio durante t\_envio} = \frac{55}{60} \times 11.2 + \frac{5}{60} \times 45.21 = 14.033 \text{ mA}$$

$$\text{Duración del dispositivo} = \frac{1100}{14.03} = 78.386 \text{ horas} = 3.266 \text{ días}$$

Figura 3.2: Consumo eléctrico.

Figura 3.3: Punto de acceso.

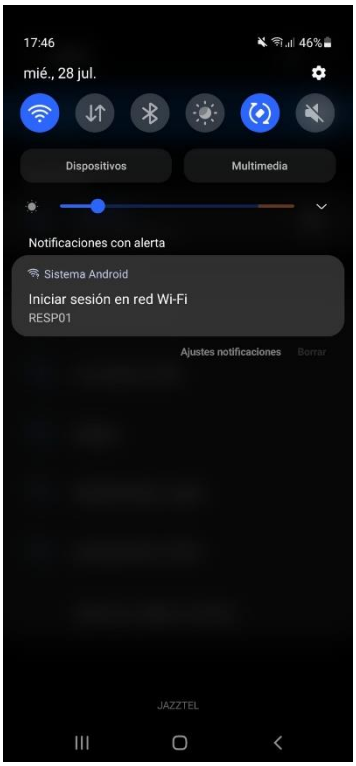

Figura 3.4: Notificación de inicio del portal cautivo.

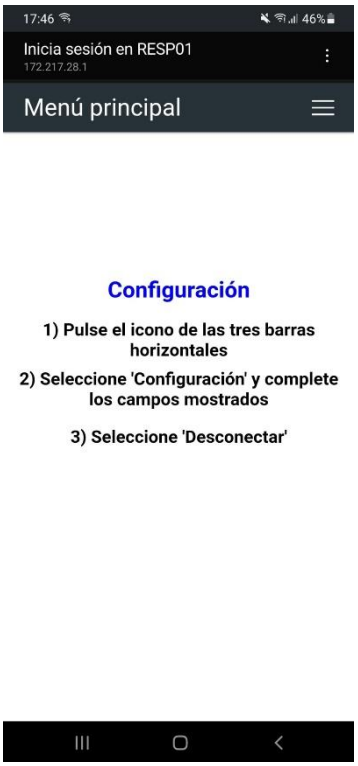

Figura 3.5: Portal cautivo.

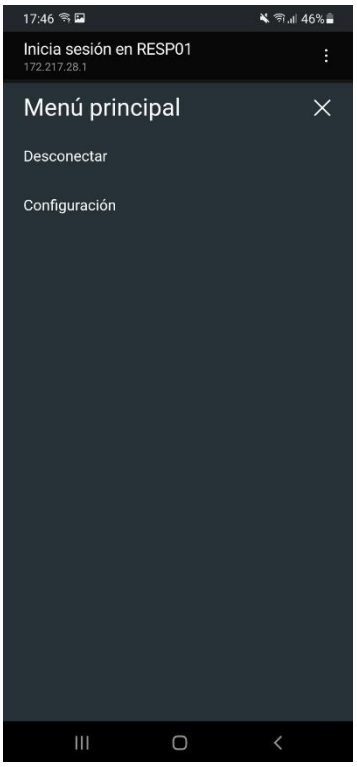

Figura 3.6: Menú desplegable del portal cautivo.

Figura 3.7: Campos de configuración en el portal cautivo.

Antes de concluir el apartado, se especifica de forma concisa el significado de cada uno de los campos de la Figura 3.7:

- SSID: red WiFi con la que se quiere establecer una conexión, es decir, la red a la que se encuentra conectado el ordenador donde se tiene instalado el bróker. Este dato es utilizado por la función "conectarWifi" del fichero "softwareTTGO" del Apéndice A.
- Contraseña: contraseña del SSID. Este dato es utilizado también por la función "conectarWifi".
- Servidor: dirección IP del servidor al que se quiere enviar la información, es decir, la dirección del ordenador donde se tiene instalado el bróker. Este dato es utilizado por la función "conectarMqtt" del fichero "softwareTTGO" del Apéndice A.
- Dispositivo: nombre del dispositivo. En el ejemplo, RESP01. Este dato es utilizado para diferenciar en la base de datos qué dispositivo realizó la medición.
- NHC: número de historia clínica del paciente. Conjunto de números que le identifican dentro del hospital. Este dato es utilizado para diferenciar en la base de datos a quién corresponde cada medición.
- Tpo. envío: período de tiempo entre cada envío al bróker. Este dato se almacena en la variable "t_envio" mencionada en la Figura 3.1.

3.2. Mosquitto

El bróker que se ha utilizado es Mosquitto, un servicio de código abierto desarrollado por la fundación Eclipse que implementa el protocolo MQTT. En la página oficial se puede encontrar toda la documentación necesaria para su uso [19].

El nodo principal va a enviar la información almacenada a un ordenador en el que está instalado Mosquitto y, a su vez, el bróker va a reenviar los datos a una base de datos. Por tanto, habrá un cliente publicador, el nodo principal, y un cliente suscriptor, el intermediario con la base de datos.

Por otro lado, se había planteado el requisito de comunicación bidireccional, de forma que se pudiesen reconfigurar parámetros sin necesidad de tener que formatear el nodo principal para que vuelva a aparecer el portal cautivo. En principio, los cuatro primeros campos de la Figura 3.7 se van a mantener constantes, por lo que interesa crear una aplicación que permita reconfigurar el NHC y el tiempo entre envíos. Para conseguirlo, va a haber un tercer cliente publicando en el bróker mensajes de reconfiguración. Esto hace que el módulo ESP32 tenga que estar suscrito al bróker para poder recibirlos, por lo que el nodo principal es un cliente publicador y suscriptor.

En la Figura 3.8 se muestra un esquema que aclara el panorama MQTT. En la situación actual se tiene un nodo principal, al cual se le ha dado el nombre de RESP01, que publica en un *topic* del mismo nombre, RESP01, al cual está suscrito el cliente de la base de datos. Si en vez de un nodo, se tuvieran 100, el cliente de la base de datos se suscribiría a 100 *topics*. Por su parte, el cliente utilizado para la reconfiguración publica mensajes en un *topic* único para cada módulo ESP32, en este caso, RESP01config, al cual está suscrito el módulo con nombre RESP01. De esta forma, si hubiera 100 módulos, el cliente de

reconfiguración tendría 100 *topics* diferentes en los que publicar, pero cada nodo principal estaría suscrito a un único *topic* de estos. Por tanto, y para concluir, siempre va a haber N+2 clientes, donde N será el número de módulos ESP32 que haya conectados al bróker, y 2 se corresponde al cliente de la base de datos y al de la reconfiguración.

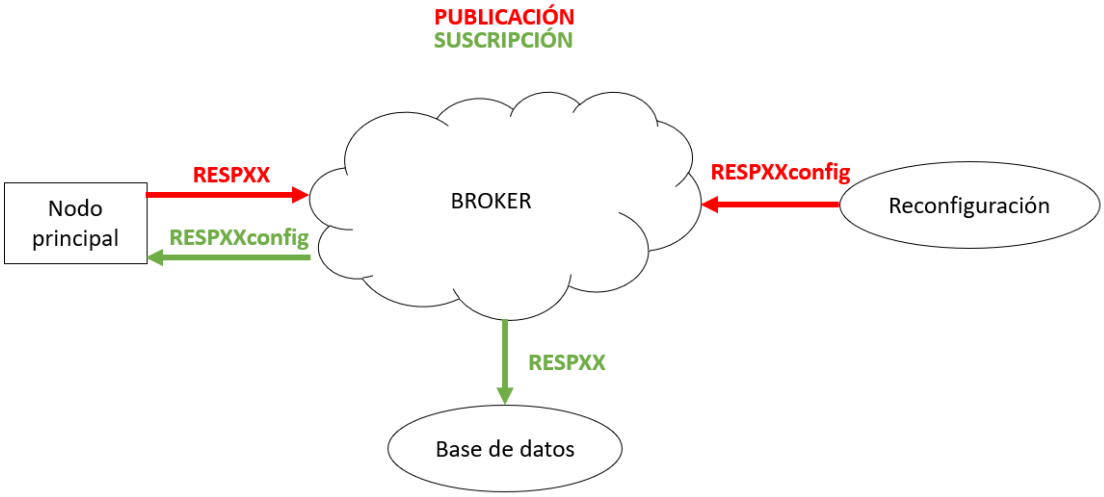

Figura 3.8: Esquema MQTT del sistema.

3.3. Arduino

Se ha utilizado el IDE de Arduino para desarrollar y grabar el código implementado en el módulo ESP32, el fichero "softwareTTGO" del Apéndice A. En la referencia de la página oficial de Arduino se exponen las diferentes funciones, tipos de variables, y estructuras disponibles, aunque es posible utilizar cualquiera de los comandos estándar de C++, puesto que el lenguaje de Arduino está basado en él [20]. La popularidad de Arduino se debe en gran parte a la multitud de librerías de las que dispone, así como al gran soporte ofrecido por la comunidad, pudiéndose encontrarse numerosos de foros de ayuda. Concretamente, para el programa desarrollado, se ha hecho uso de hasta 9 librerías diferentes:

- WiFi: librería que permite establecer conexiones WiFi. Además, da la posibilidad de configurar el modo de acceso, en función de si se quiere tener un punto de acceso o una estación, como bien se comentó anteriormente [21].
- AutoConnect: proporciona una interfaz web para la configuración del punto de acceso, es decir, el portal cautivo [22]. Implícitamente hace uso de las librerías ArduinoJson y PageBuilder. La primera facilita el manejo de ficheros Json (*JavaScript Object Notations*), ficheros de texto plano para el almacenamiento de datos estructurados [23]. La segunda permite generar de forma sencilla páginas HTML y enviárselas al cliente [24].
- PubSubClient [25]: librería que da soporte a la comunicación MQTT para poder enviar y recibir mensajes.
- TFT_eSPI: posibilita la utilización de la pantalla TFT [26].
- 18650CL: librería que proporciona funciones relacionadas con la obtención de estadísticas de la batería [27].

- SPIFFS: permite el manejo del SPIFFS (*SPI Flash File System*), que es el sistema de ficheros diseñado para funcionar en memorias flash conectadas por SPI [28].
- FS: librería que contiene objetos y funciones interesantes para la gestión de ficheros en SPIFFS [29].
- Wire: posibilita la comunicación con otros dispositivos mediante el bus I2C [30].
- Ticker: contiene una función que permite llamar a otras funciones de forma repetida cada cierto intervalo de tiempo [31].

En Arduino, los *scripts*, denominados *sketches*, cuentan siempre con dos funciones principales llamadas "setup" y "loop". Al ejecutar un *sketch*, se realiza en primer lugar lo indicado en "setup" y, seguidamente, se pasa a "loop". Esta función actúa como si de un bucle "while(1)" se tratara, repitiéndose de forma cíclica el código que alberga. A continuación, se van a enumerar las diferentes acciones que se producen dentro de estas funciones en el fichero "softwareTTGO". En el Apéndice A se puede encontrar el código de forma íntegra.

En "setup":

1. Se inicializa el Monito Serie de Arduino, el cual permite ver por pantalla, en un ordenador, la realimentación de la ejecución.
2. Se establece una frecuencia de CPU de 80MHz, que es la frecuencia mínima a la cual el TTGO es capaz de establecer conexiones WiFi.
3. Se inicializan los botones, así como las interrupciones asociadas a ellos. Si se presiona el botón izquierdo, se produce un formateo del sistema de ficheros y un reinicio del módulo. Si se pulsa el botón derecho, se enciende la pantalla, de forma que se van mostrando los sucesivos valores capturados de la frecuencia respiratoria.
4. Se inicializa el sistema de ficheros.
5. Se inicializa el portal cautivo y se cargan los valores almacenados para los parámetros de configuración.
6. Se establece la configuración del portal cautivo, su título, la opción del menú que se quiere mostrar, el nombre y contraseña del SSID generado como punto de acceso, y las URIs.
7. Si es la primera vez que se ejecuta el código, se inicializa el módulo, se muestra en la pantalla el nombre del SSID y se lanza el portal cautivo. Una vez se cierre este, se apaga la pantalla.
8. Se establece la conexión WiFi. Si transcurrido un minuto no se hubiera conseguido, se encendería la pantalla mostrando el siguiente mensaje de error: "Error WiFi".
9. Se establecen los detalles del servidor MQTT, nombre y puerto.
10. Se habilita la función "callback" para poder recibir los mensajes de reconfiguración.
11. Se establece la comunicación MQTT y se realiza la suscripción al *topic* RESPXXconfig correspondiente, como se expuso en la Figura 3.7. Si transcurrido un minuto no se hubiera conseguido establecer, se encendería la pantalla mostrando el siguiente mensaje de error: "Error MQTT".

12. Se configura la hora en el huso de España accediendo a un servidor NTP.
13. Se muestra en la pantalla un mensaje indicando que las conexiones se han establecido de forma satisfactoria: “Conectado!”.
14. Se finalizan las conexiones MQTT y WiFi.

Dentro de la función “loop”:

1. Se llama a la función “dormir”. Dentro de esta función se establece el modo *light sleep*, un modo de suspensión para reducir el consumo de la batería en el cual los periféricos digitales, parte de la RAM y las CPUs entran en estado de *clock-gating* [32]. El dispositivo permanece en este estado durante 90 milisegundos.
2. Transcurridos 100 milisegundos desde el inicio de la función “loop”, se accede a la función “getData”, que es donde se realiza la captura y filtrado de los datos obtenidos por el magnetómetro, para obtener la frecuencia respiratoria cuantificada en respiraciones por minuto.
3. Si ha transcurrido el tiempo suficiente como para hacer un envío, variable “t_envio” comentada previamente, se procede a:
   3.1. Activar “ticker” de “getData”.
   3.2. Establecer de nuevo las conexiones WiFi y MQTT.
   3.3. Obtener la hora.
   3.4. Recibir los mensajes del *topic* suscrito.
   3.5. Publicar un mensaje en el bróker. Este contiene el nombre del dispositivo, el NHC del paciente, la media de las respiraciones por minuto desde que se realizó el último envío, y la hora a la que se tomó la primera medición de las que componen la mencionada media.
   3.6. Finalizar las conexiones WiFi y MQTT.
   3.7. Desactivar “ticker” de “getData”.

   El “ticker” se utiliza para que se continúe accediendo a “getData” cada 100ms, puesto que la realización del proceso descrito en este tercer paso conlleva unos 5 segundos.

La Figura 3.9 ilustra en un diagrama de flujo todos estos pasos. Además del proceso secuencial explicado, existen dos interrupciones asociadas a los botones del TTGO. Si se pulsa el botón derecho, la pantalla del nodo principal se enciende, mostrando en tiempo real las respiraciones por minuto que se van obteniendo. Si se pulsa de nuevo el botón derecho, la pantalla se apaga. En el caso del botón izquierdo, al presionarlo, se produce un borrado de las credenciales guardadas y un reinicio del TTGO, por lo que al iniciarse de nuevo se lanza el portal cautivo. Estas dos interrupciones se reflejan en la Figura 3.10.

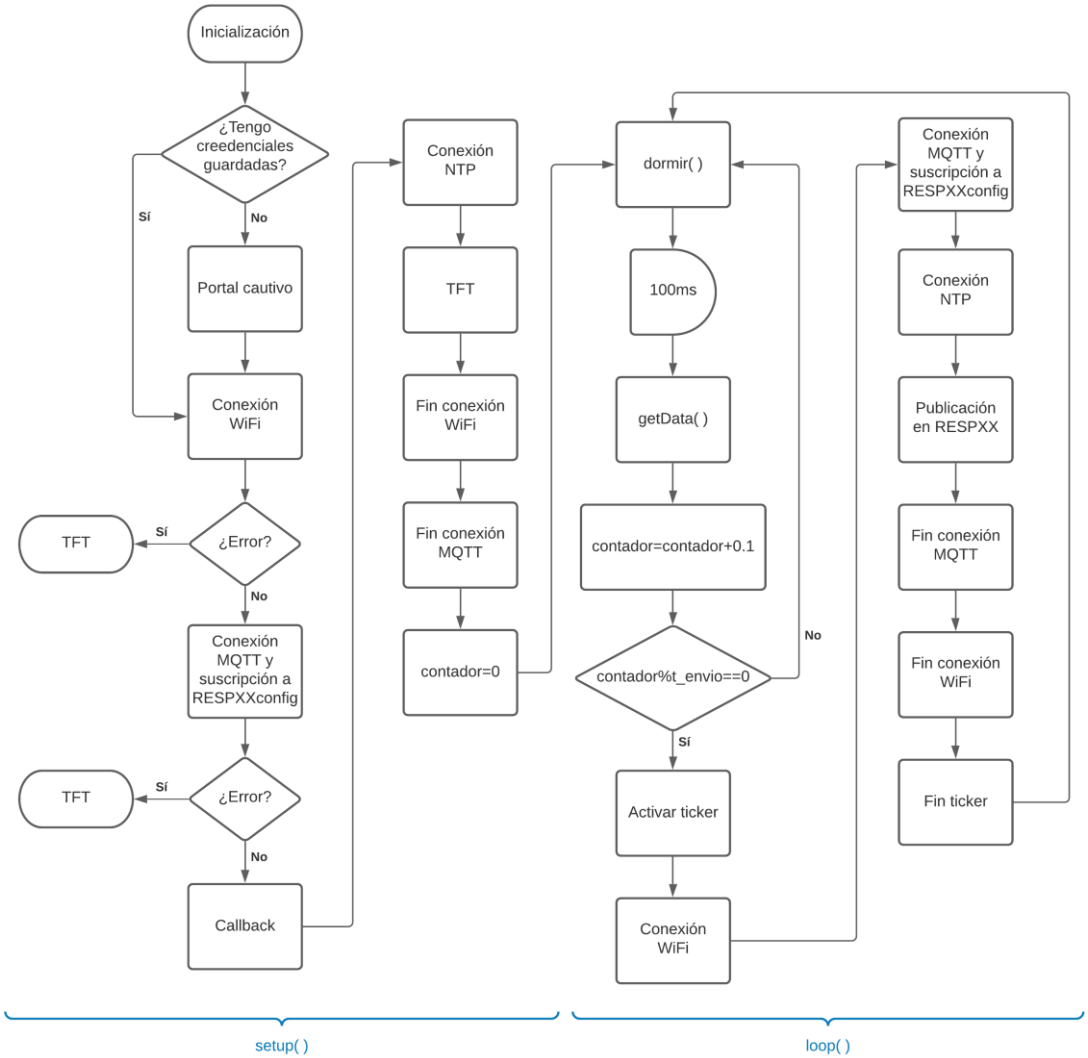

Figura 3.9: Diagrama de flujo del fichero “softwareTTGO”.

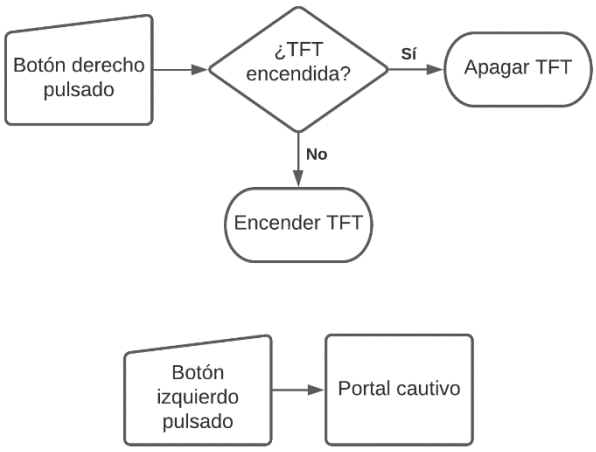

Figura 3.10: Diagrama de flujo de las interrupciones del fichero “softwareTTGO”.

3.3.1. Función "getData"

La clave del funcionamiento del dispositivo se encuentra en el filtrado digital que se realiza a partir de la señal eléctrica generada por el magnetómetro. La función donde se produce esto es "getData". La Figura 3.11 muestra el circuito de filtrado que implementa dicha función, donde se tienen tres filtros de tipo IIR, filtros de respuesta infinita al impulso. Las etiquetas en rojo se corresponden con los nombres que tienen las variables en el código del fichero "softwareTTGO".

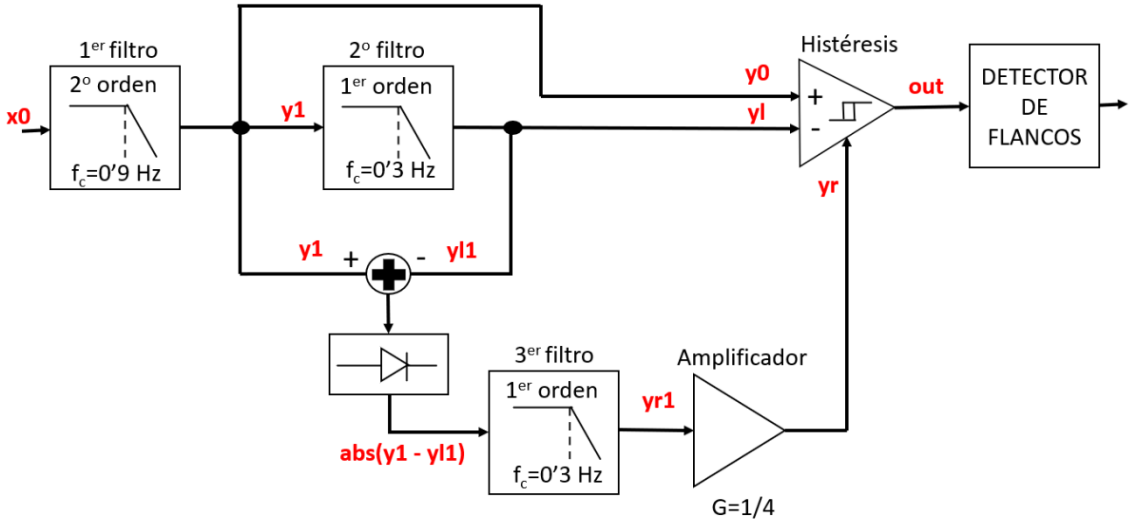

Figura 3.11: Esquema del filtrado digital implementado en la función "getData".

La señal etiquetada como x0 es igual al módulo de un vector cuyas coordenadas son los valores del campo magnético para los tres ejes obtenidos por el magnetómetro. Esta señal se hace pasar por un filtro paso bajo de segundo orden con frecuencia de corte de 0.9 Hz. A mayor orden, mayor rechazo de las frecuencias superiores a la de corte, pero también mayor complejidad, por lo que supone un aumento de la carga computacional. La respiración humana nunca va a ser mayor o igual a una respiración por segundo, por lo que interesa atenuar las frecuencias superiores a 1 Hz, de ahí el valor de 0.9 Hz para la frecuencia de corte. En la Figura 3.12 se puede ver la función de transferencia asociada a este filtro de Butterworth, así como la señal de salida resultante. Los tres filtros son de este tipo, puesto que son los que tienen menor rizado en la banda de paso.

A la salida del segundo filtro se va a tener una señal atenuada a partir de 0.3 Hz. Al restar esta señal a la salida del primer filtro, se obtiene el mismo resultado que si se hubiera aplicado un filtro paso banda con una banda de paso entre 0.3 Hz y 0.9 Hz. A continuación, esta señal es sometida a un detector de envolvente, representado en la Figura 3.11 por el bloque del diodo. Seguidamente, la señal resultante es filtrada de nuevo con un filtro igual al segundo, cuyas ecuaciones se muestran en la Figura 3.13, y amplificada por un factor de ¼.

Finalmente, se tiene un comparador con histéresis no inversor, cuya salida se ha etiquetado como out. Si vale 0 y la tensión de la señal y0 es superior a la suma de las tensiones de las otras dos señales, yl e yr, el valor de out cambiará a 1. De la misma forma, si la salida es 1 y la tensión de la señal y0 es inferior a la diferencia entre yl e yr, el valor de out pasaría a ser 0.

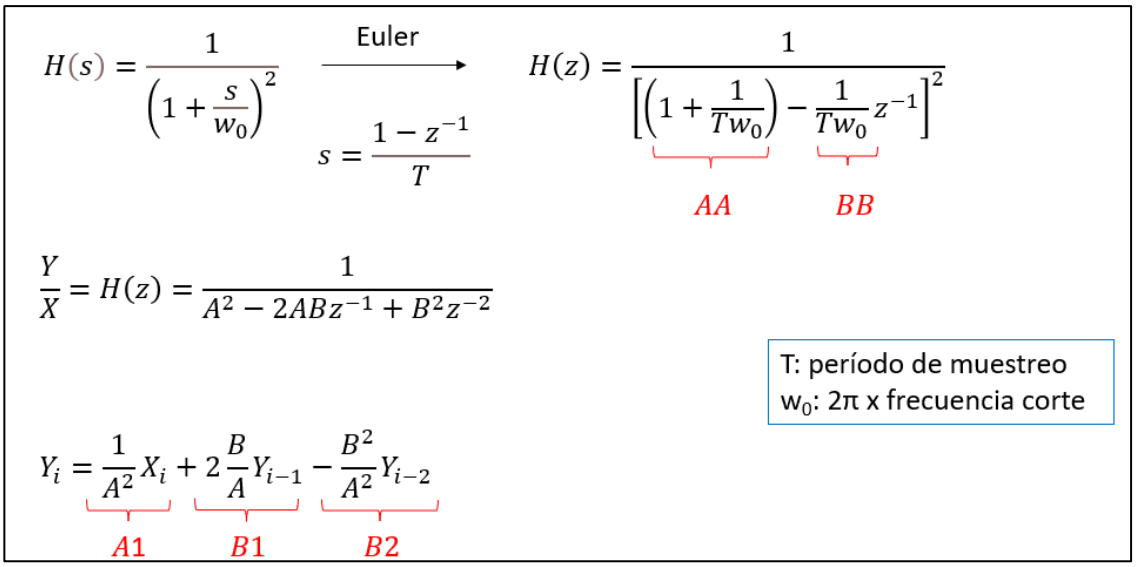

Figura 3.12: Ecuaciones matemáticas asociadas al primer filtro

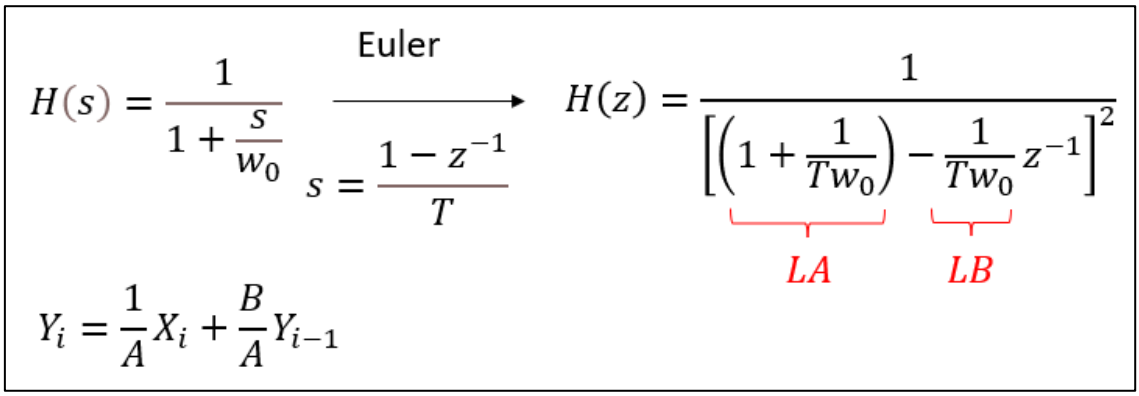

Figura 3.13: Ecuaciones matemáticas asociadas al segundo y tercer filtro.

La Figura 3.14 muestra un ejemplo de la evolución de las mencionadas señales. En la parte central de la imagen se ven cuatro líneas prácticamente superpuestas, las cuales se corresponden con las señales x0, color azul, y0, en rojo, yl+yr, en verde, e yl-yr, en naranja. En la parte inferior, en color rosa, aparecen los flancos de subida y bajada asociados a la señal de salida out. Se puede apreciar como el flanco de bajada se produce cuando el valor de la señal roja, y0, pasa a ser inferior al valor de la señal naranja, yl-yr, tal y como se había comentado previamente. De igual manera, en el momento en que el valor de y0 supera al de yl+yr, es cuando se produce el flanco de subida. Por su parte, se puede observar que la forma de la señal roja, x0, es similar a la de la señal azul, y0, pero menos abrupta, como consecuencia del paso por el primer filtro.

Por otra parte, la señal en color gris se corresponde con los valores finales de la frecuencia respiratoria. A partir de un detector de flancos, bloque final de la Figura 3.11, se capturan las transiciones producidas durante los ciclos de histéresis para, mediante una media móvil, cuantificar el número de respiraciones por minuto. En la Figura 3.14 se puede comprobar como la frecuencia respiratoria aumenta en concordancia con la disminución de la anchura de los pulsos en rosa. De forma inversa, a menor número de transiciones en el ciclo de histéresis, menor es el número de respiraciones por minuto.

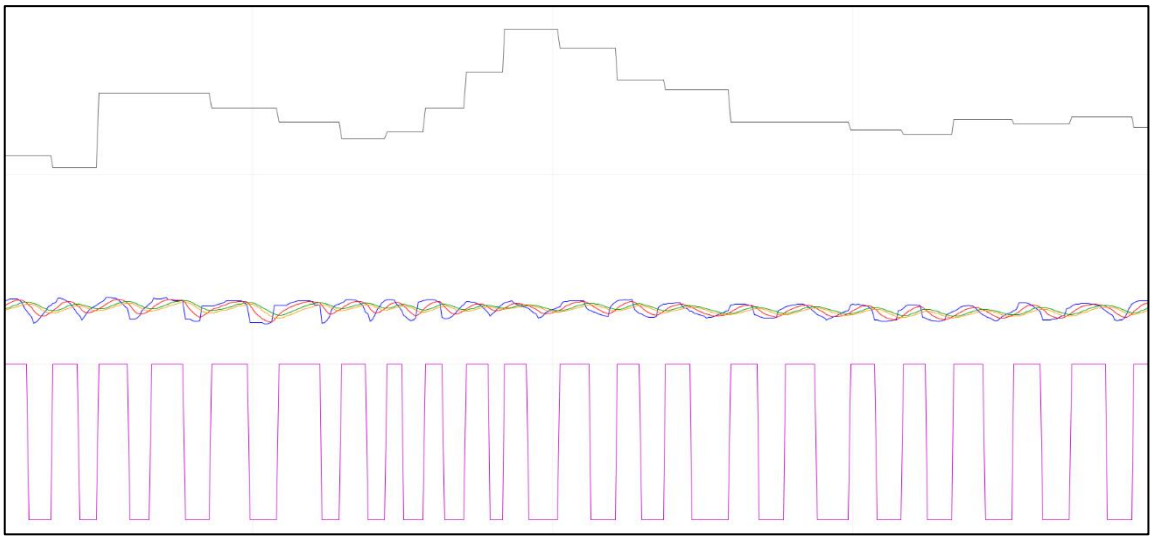

Figura 3.14: Formas de onda.

3.4. MariaDB

Una base de datos es una colección de tablas, las cuales muestran datos en formato estructurado de filas y columnas. Para el manejo de bases de datos, se utilizan DBMS (Database Management System). Un DBMS es una colección de *software* orientado al manejo de bases de datos, cuya función es servir de interfaz entre la base de datos, el usuario y las distintas aplicaciones utilizadas. Para interactuar con una base de datos se utilizan consultas SQL o *queries*.

En este caso, se ha utilizado el DBMS MariaDB [33]. La instalación del servidor MariaDB trae consigo un programa llamado HeidiSQL [34], el cual permite visualizar de forma gráfica el contenido de las bases de datos. Para la creación de las tablas que componen las bases, se ha utilizado el programa SQLWorkbench [35]. En la Figura 3.15 se muestra el código, en lenguaje SQL, utilizado para generar una tabla de nombre "respirometro". Cada línea define el nombre y tipo de una variable, la cual dará el nombre a una columna de la tabla. Por tanto, la ejecución de este código da lugar a una tabla de 6 columnas. Las cinco primeras coinciden con los datos que contiene el mensaje publicado vía MQTT por el nodo principal, nombre del dispositivo, hora a la que se inició la medición, NHC, media de la respiración por minuto y porcentaje de batería restante. El sexto parámetro, "hora_medida", es añadido por el cliente de la base de datos e indica el instante en el que se integró el mensaje en la base de datos. Por último, en una tabla siempre debe existir una clave primaria, que es un valor único que identifica inequívocamente a cada fila de una tabla. En este caso, los parámetros "dispositivo" y "hora_medida" forman dicha clave, puesto que nunca va a haber dos mediciones realizas en el mismo instante por parte de un mismo dispositivo.

En la Figura 3.16 se muestra cómo se ve en HeidiSQL la tabla resultante de ejecutar el código de la Figura 3.15. En este ejemplo se puede observar la información enviada por un dispositivo con nombre "RESP01" sobre un paciente con número de historia clínica 3245.

```
CREATE TABLE respirometro(
    dispositivo VARCHAR(10),
    NHC INT,
    hora_medida DATETIME,
    rpm FLOAT,
    bateria VARCHAR(4),
    hora_volcado DATETIME,
    PRIMARY KEY (dispositivo, hora_medida)
);
```

Figura 3.15: Código desarrollado en SQLWorkbench.

dispositivo	NHC	hora_medida	rpm	bateria	hora_volcado
RESP01	3.245	2021-08-31 20:04:06	14,27	56	2021-08-31 20:05:06
RESP01	3.245	2021-08-31 20:03:06	15,34	56	2021-08-31 20:04:06
RESP01	3.245	2021-08-31 20:02:05	14,79	56	2021-08-31 20:03:06
RESP01	3.245	2021-08-31 20:01:06	10,8	56	2021-08-31 20:02:06
RESP01	3.245	2021-08-31 20:00:05	14,28	56	2021-08-31 20:01:07
RESP01	3.245	2021-08-31 19:59:05	11,65	56	2021-08-31 20:00:05
RESP01	3.245	2021-08-31 19:58:05	11,05	56	2021-08-31 19:59:05
RESP01	3.245	2021-08-31 19:57:06	13,01	56	2021-08-31 19:58:05
RESP01	3.245	2021-08-31 19:56:05	12,77	56	2021-08-31 19:57:06
RESP01	3.245	2021-08-31 19:55:05	12,94	56	2021-08-31 19:56:05

Figura 3.16: Datos almacenados en la base de datos vistos en HeidiSQL.

3.5. Python

Tanto el fichero que genera al cliente del bróker que realiza la inserción de mensajes en la base de datos, como el que origina al cliente encargado de la reconfiguración, han sido desarrollados en lenguaje Python [36]. El motivo de no haber utilizado también Arduino se encuentra en que en Python existe una librería, llamada MySQLdb [37], que permite establecer de forma sencilla una conexión con la base de datos. Para la ejecución de estos dos ficheros, basta con escribir en línea de comandos la instrucción *python nombre_fichero.py*. En el caso del primer cliente mencionado, el nombre del fichero que lo genera es “database”, mientras que para el segundo es “reconfig”. En el Apéndice A se puede encontrar el código de ambos ficheros.

La Figura 3.17 refleja el funcionamiento del algoritmo implementado en “database”. En primer lugar, gracias a la mencionada librería MySQLdb, se establece la comunicación con la base de datos. A continuación, se hace lo propio con el bróker, haciendo esta vez uso de una librería llamada paho.mqtt.client [38]. Ambas conexiones permanecen abiertas hasta que el usuario decide finalizarlas por medio del comando *ctrl+c*. Cada mensaje publicado por el módulo ESP32 es separado en segmentos, uno por cada columna de las mostradas en la Figura 3.16. Además, como se indicó en el apartado anterior, se añade un segmento más con la “hora_volcado”. Mediante una *query*, se hace que todos estos segmentos queden integrados en la columna de la tabla correspondiente.

Por su parte, la Figura 3.18 muestra el diagrama de flujo del fichero “reconfig”. En este caso, al ejecutar el comando *python reconfig.py*, el usuario se va a encontrar con una aplicación interactiva. En ella, se solicita, en primer lugar, el *topic* en el que se va a publicar (RESPXXconfig) y, a continuación, se presentan los dos parámetros disponibles para modificar, NHC y “t_envio”. En el ejemplo de la Figura 3.19 se ha publicado en el *topic* RESP01config un nuevo número de historia clínica con valor 687.

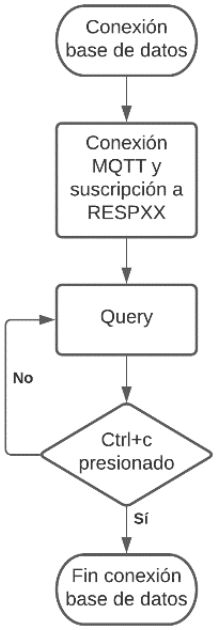

Figura 3.17: Diagrama de flujo del fichero “database”.

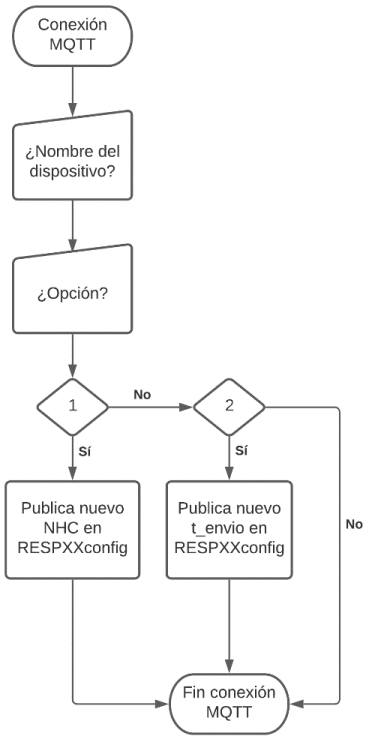

Figura 3.18: Diagrama de flujo del fichero “reconfig”.

```
C:\Program Files\mosquitto>python reconfig.py
Nombre del dispositivo [RESPXXconfig]:RESP01config
1:NHC
2:Tiempo entre envios
Introduzca el numero del parametro a modificar:1
Introduzca el nuevo NHC:687
```

Figura 3.19: Resultado de la ejecución del fichero “reconfig”.

3.6. CSS

A grandes rasgos, CSS (*Cascading Style Sheets*) es el lenguaje utilizado para manejar el diseño y presentación de páginas web, es decir, cómo lucen frente al usuario. Funciona junto al lenguaje HTML, que es el que se encarga del contenido de las páginas [39].

En este lenguaje están desarrollados los ficheros fuente de la librería AutoConnect. Por defecto, al iniciarse el portal cautivo comentado previamente, la pantalla principal debería presentar el aspecto mostrado en la Figura 3.20. Para logar el resultado observado en la Figura 3.5, se ha tenido que modificar el código del fichero “AutoConnectPage.cpp”. En el Apéndice A se muestran las líneas retocadas.

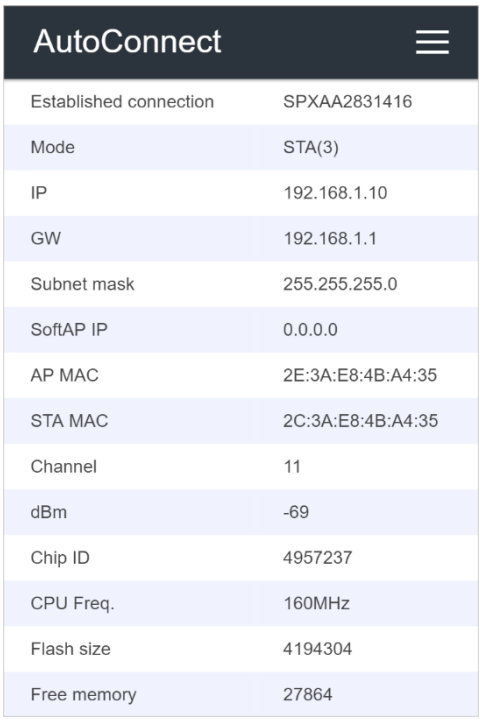

Figura 3.20: Pantalla de inicio del portal cautivo de la librería AutoConnect.

Capítulo 4

Casos prácticos

4.1. Visión general

En este apartado se muestran algunas gráficas obtenidas a partir de los datos almacenados en la base de datos. A pesar de que existen programas más sofisticados y elegantes para la elaboración de gráficas, se ha optado por usar una herramienta sencilla y conocida como es Excel, puesto que desde HeidiSQL se pueden exportar los datos de las tablas en formato Excel CSV.

En la Figura 4.1 se pueden ver los rangos de frecuencia respiratoria típica en estado de reposo de un ser humano en función de la edad. En base a esto, una persona adulta no debería superar las 20 respiraciones por minuto.

años_vida	meses_vida	dias_vida	limite_inf	límite_sup
0 - 0,5	0 - 6	0 - 183	35	60
0,5 - 1	6 - 12	183 - 365	25	40
1 - 3	12 - 36	365 - 1096	20	30
3 - 6	36 - 72	1096 - 2192	20	25
6 - 12	72 - 144	2192 - 4383	14	22
>12 años	> 144 meses	> 4383 días	12	20

Figura 4.1: Respiraciones por minuto en función de la edad [40].

En primer lugar, se muestra una gráfica, Figura 4.2, en la que se ha monitorizado a un varón de 22 años durante un período de una hora en estado de reposo. La frecuencia obtenida es bastante regular, aunque algo elevada, oscilando principalmente entre 18 y 22 respiraciones por minuto. En la Figura 4.3, se puede observar la gráfica resultante para la misma situación de reposo en el caso de un varón de 60 años. Se observa claramente que la frecuencia es inferior a la anterior, adecuándose más al rango esperado.

En la Figura 4.4 se tienen de nuevo los datos del varón de 22 años. En esta ocasión, transcurrido un tiempo de reposo, se comienza a hacer ejercicio, lo cual se refleja drásticamente en la gráfica, tardando un poco en recuperar la monotonía inicial.

Por último, se muestran dos gráficas que capturan la evolución respiratoria durante el período de sueño de un varón de 22 años, Figura 4.5, y de una mujer de 57 años, Figura 4.6. La media de los datos del primer caso resulta ser de 19.20 respiraciones por minuto,

mientras que para el segundo caso es de 16.8. De nuevo, se ve como el primer paciente tiene una frecuencia respiratoria más elevada de lo normal. En ambos casos, existen grandes diferencias entre los valores máximos y mínimos, lo cual está asociado a fases de sueño profundo y fases de movimiento del paciente en la cama.

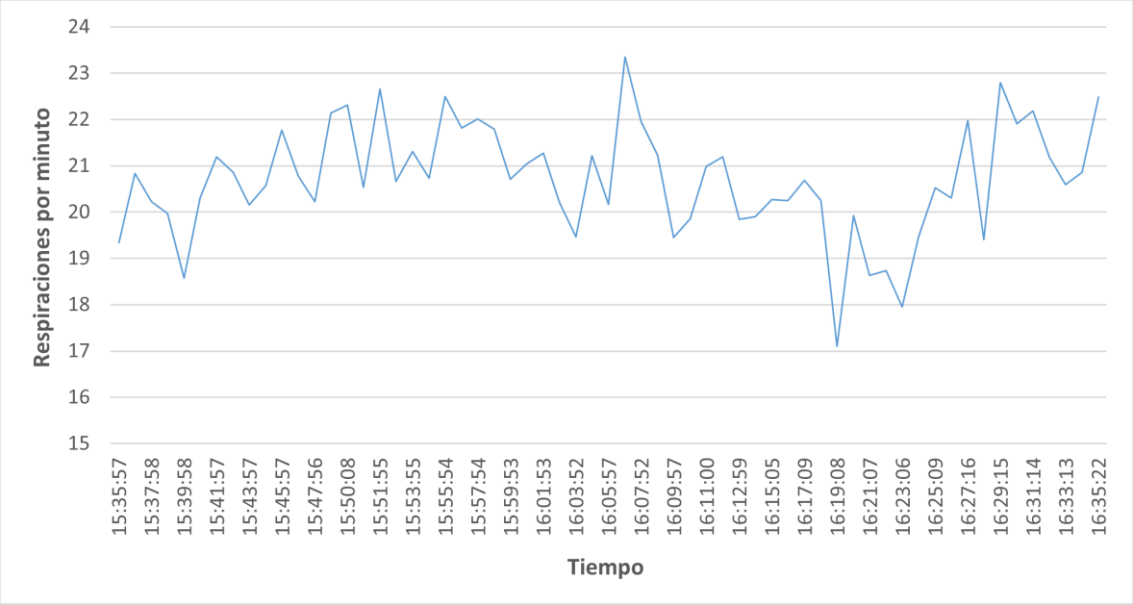

Figura 4.2: Evolución respiratoria de un varón de 22 años en estado de reposo.

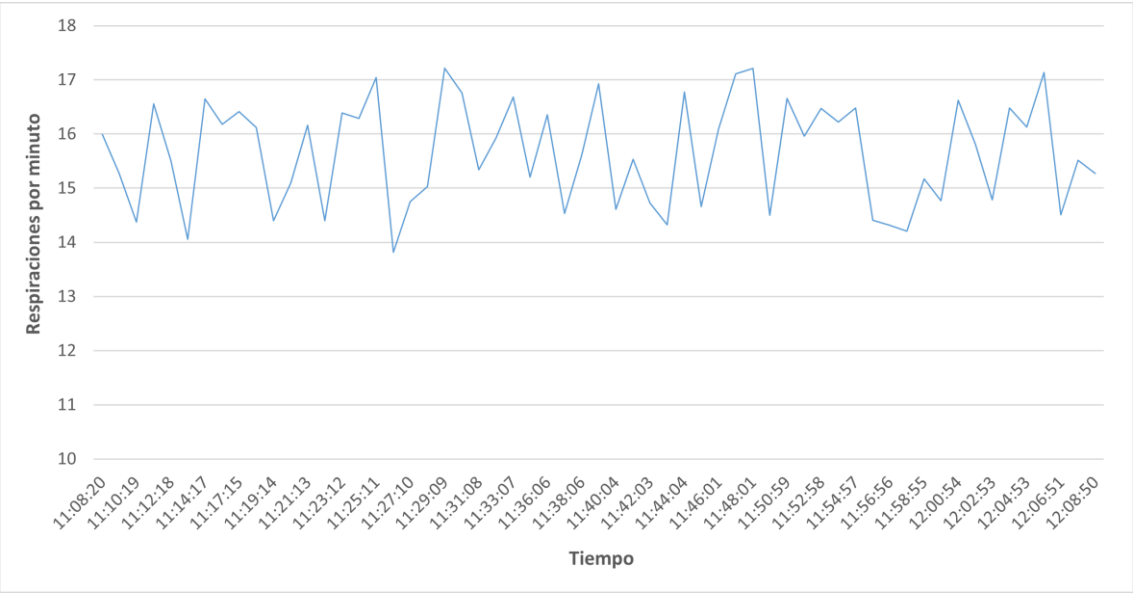

Figura 4.3: Evolución respiratoria de un varón de 60 años en estado de reposo.

.

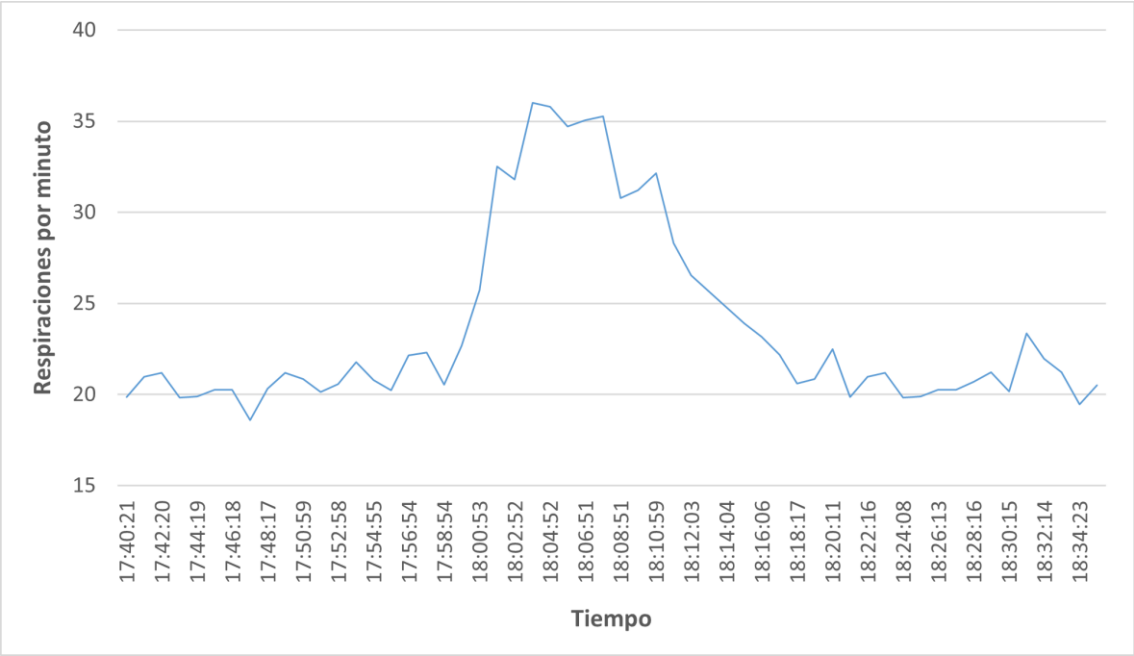

Figura 4.4: Evolución respiratoria de un varón de 22 años haciendo ejercicio.

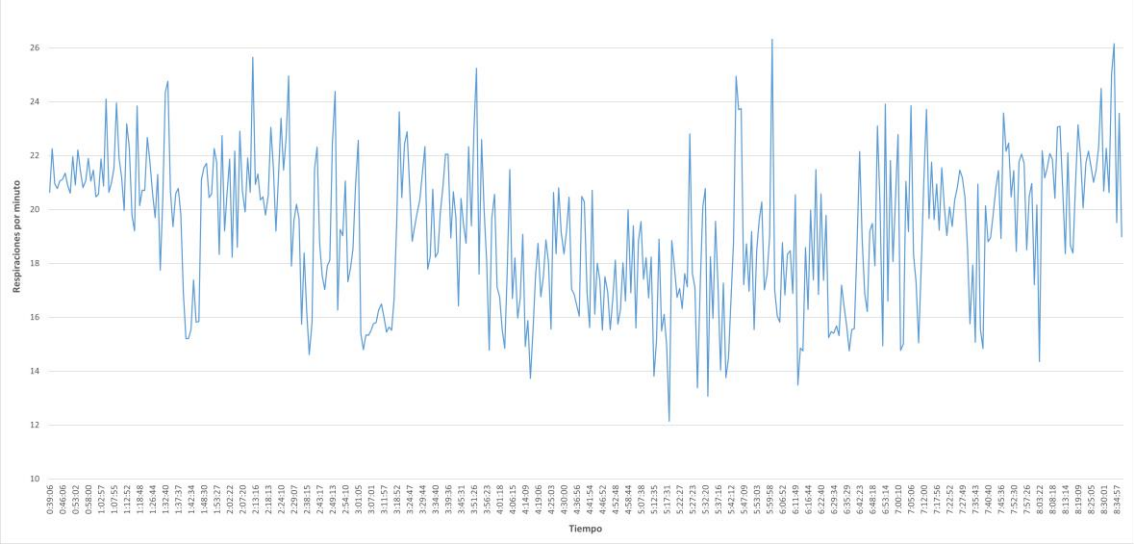

Figura 4.5: Evolución respiratoria de un varón de 22 años durante su período de sueño.

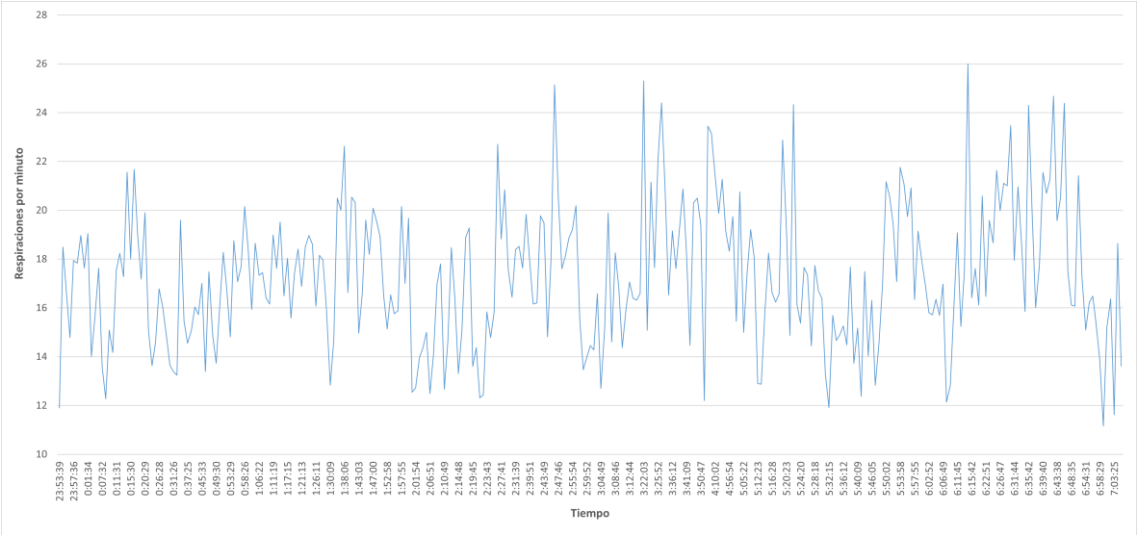

Figura 4.6: Evolución respiratoria de una mujer de 57 años durante su período de sueño.

Capítulo 5

Conclusiones

5.1. Obstáculos encontrados

Los problemas y desafíos encontrados a lo largo del presente trabajo han sido numerosos y variados.

Respecto al apartado *hardware*, dos problemas destacaron sobre el resto. El primero de ellos referido a cómo proteger la electrónica. En un principio se intentó hacer una caja con ranuras en las paredes laterales para hacer pasar las cintas, y con una base curva para que no hubiera vértices que se pudieran clavar en el cuerpo del paciente. Tras imprimir varios prototipos diferentes, no se consiguió ninguno con un buen acabado y, además, debido a la necesidad de hacer unas paredes robustas para que no se rompieran al tensar las cintas, la caja resultaba bastante voluminosa. Esto propicio que se planteara otra alternativa, consistente en imprimir unas simples planchas con ranuras en los extremos para las cintas. La electrónica iría colocada sobre dichas planchas, y todo ello iría envuelto bajo un plástico termorretractil transparente. El nuevo problema surgió al intentar calentar dicho plástico, puesto que su punto de fusión era mucho más alto que el del PLA utilizado para la impresión de las planchas. Además, aplicar calor teniendo una batería de por medio no es muy seguro, pues podría explotar. Por tanto, esta idea también fue descartada. Finalmente, se juntaron ambas ideas para dar lugar al diseño final, es decir, añadir unas paredes y tapa a la plancha.

Por otro lado, se tuvo una pequeña confusión con el magnetómetro. Como se explicó en el apartado 3 del Capítulo 2, existen dos chips con la misma funcionalidad, el QMC5883L y el HMC5883L. En la Figura 2.2 se ve que aparece impreso el nombre HMC5883L, sin embargo, en Amazon el producto se comercializa bajo el nombre de QMC5883L. Erróneamente, se pensó que lo cierto sería lo indicado en la serigrafía, por lo que en un primer intento se realizó la configuración *software* intentando acceder a la dirección I2C indicada para el HMC5883L.

En cuanto al *software*, el problema principal ha sido cómo plantear el algoritmo del módulo ESP32 para intentar prolongar al máximo la duración de la batería. Además, se ha visto que, bien Arduino, o bien el propio módulo, presenta problemas de estabilidad cuando están activas diferentes interrupciones al mismo tiempo. Por ello, en la función "dormir" se activa y desactiva la interrupción en cada llamada. Asimismo, la librería Ticker tampoco ha resultado funcionar como debiera, puesto que deja de activarse cuando el módulo está dormido. Por este motivo solo se hace uso de la librería Ticker en el

fragmento de la función “loop” en el que se trata de publicar el mensaje en el bróker, evitando así el conflicto con la interrupción de la función “dormir”.

5.2. Posibles mejoras y objetivos futuros

A corto plazo, la principal mejora a lograr pasaría por aumentar la autonomía del dispositivo. Para ello, habría que buscar cuál es el módulo ESP32 del mercado que menos energía consume en estado *light sleep*. Igualmente, resultaría oportuno utilizar el mismo lenguaje para los tres ficheros que se han desarrollado, de manera que se redujesen el número de programas a manejar e instalar.

Con vistas a futuro, sería interesante la creación de una aplicación Android en donde se pudieran ver en tiempo real, y desde un teléfono, las gráficas de la evolución respiratoria. Con ello, podría darse al dispositivo un enfoque destinado al mercado de *wearables* deportivos. Aumentando más las expectativas, se podría plantear también la posibilidad de medir más parámetros corporales, como por ejemplo la presión sanguínea.

Apéndice A

Listado de códigos

A.1. Fichero softwareTTGO.ino

```
//Librerías utilizadas:
#include <WiFi.h>
#include <AutoConnect.h>
#include <PubSubClient.h>
#include <TFT_eSPI.h>
#include <Pangodream_18650_CL.h>
#include "time.h"
#include <FS.h>
#include <SPIFFS.h>
fs::SPIFFSFS& FlashFS = SPIFFS;
#include <Wire.h>
#include <Ticker.h>
Ticker ticker;

#define PARAM_FILE      "/config.json" //Fichero donde se guardan
todas las creendeciales
#define AUX_SETTING_URI "/mqtt_setting"
#define AUX_SAVE_URI    "/mqtt_save"

#define BOTON_DCHA 35 //Botón derecho
#define BOTON_IZQ 0 //Botón izquierdo

#define ADC_PIN 34
#define CONV_FACTOR 1.8
#define READS 20

#define ADDR 0x0D //Dirección I2C para el magnetómetro
#define UMBRAL 32767 //Umbral de saturación de los registros de datos
#define RANGO_UMBRAL 65536 //Rango del umbral
#define PI 3.141592
#define DUMMY 0xBEBECAFE
#define QMC5883_Write(a,d) Wire.beginTransmission(ADDR);
Wire.write(a); Wire.write(d); Wire.endTransmission();
/* Inicializa la transmisión al dispositivo esclavo. En la dirección
a, escribo un byte d.
  Transmito y libero pines.*/

AutoConnect portal;
AutoConnectConfig config;
WiFiClient   wifiClient;
PubSubClient client(wifiClient);
```

```
TFT_eSPI tft = TFT_eSPI();
Pangodream_18650_CL BL(ADC_PIN, CONV_FACTOR, READS);

//Variables que el usuario introduce:
String ssid_;
String contrasena_;
String servidor_;
String dispositivo_;
String NHC_;
String tiempo_;

String payload; //Variable para el payload del mensaje mqtt a enviar
char msg[100]; //En el envío de mensajes mqtt el payload tiene que ser
de tipo char
bool conf = false; //Auxiliar para comprobar si se han introducido los
parámetros
bool auxMostrar = false; //Auxiliar para que se muestren las rpm en la
pantalla
int errorMqtt = 0; //Contador del número de intentos de conexión MQTT
int errorWifi = 0; //Contador del número de intentos de conexión WiFi
int t0 = 0, t1 = 0; //Auxiliares para llevar la cuenta del tiempo en
la función loop
int t_envio = 0; //Cada cuanto tiempo se envía al bróker
float medicion[1000]; //Matriz con los valores de las mediciones
String horas[1000]; //Matriz con las horas a las que se realizaron las
mediciones
int n_mensaje = 0; //Tamaño de las dos matrices anteriores

//Para la hora en España:
const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 3600;
const int   daylightOffset_sec = 3600;

//Variables para la función getData:
#define M_SIZE 512
int16_t M[M_SIZE]; //Matriz de almacenamiento de módulos
int n_getData = 0; //Contador del número de iteraciones de getData
int magne = 0; //Contador para reiniciar el magnetómetro en previsión
de un posible bloqueo

//************************************
//        ASPECTO DE LA PÁGINA
//************************************
ACStyle(estilo,
"label+input,label+select{position:sticky;left:300px;width:210px!impor
tant;box-sizing:border-box}");
ACInput(ssid, "", "SSID", "{1,}", "Red WiFi");
ACInput(contrasena, "", "Contraseña", "{1,}", "Contraseña del SSID");
ACInput(servidor, "", "Servidor", "{1,}", "Dirección IP");
ACInput(dispositivo, "", "Dispositivo", "{1,}", "Número de serie
RESPXX");
ACInput(NHC, "", "NHC", "^[0-9]{1,10}", "Número de historia clínica");
ACInput(tiempo, "", "Tpo. envío", "^[0-9]{1,10}", "Tiempo en
segundos");

ACSubmit(guardar, "Guardar", "/mqtt_save");
AutoConnectAux aux1("/mqtt_setting", "Configuración", true, {estilo,
ssid, contrasena, servidor, dispositivo, NHC, tiempo, guardar});

ACText(texto, "Parámetros guardados:", "text-
align:center;color:#2f4f4f;padding:10px;");
```

```
ACText(parameters);
ACText(texto2, "Pulse hacia atrás si quiere modificarlos", "text-
align:center;font-family:serif;color:#4682b4;padding:10px");
AutoConnectAux aux2("/mqtt_save", "Configuración", false, {texto,
parameters, texto2});

//*************************************
//                  GET
//*************************************
void getParams(AutoConnectAux& aux)
{
  ssid_ = aux["ssid"].value;
  ssid_.trim();
  contrasena_ = aux["contrasena"].value;
  contrasena_.trim();
  servidor_ = aux["servidor"].value;
  servidor_.trim();
  dispositivo_ = aux["dispositivo"].value;
  dispositivo_.trim();
  NHC_ = aux["NHC"].value;
  NHC_.trim();
  tiempo_ = aux["tiempo"].value;
  tiempo_.trim();
  t_envio = tiempo_.toInt();
}

//*************************************
//                  SAVE
//*************************************
String saveParams(AutoConnectAux& aux, PageArgument& args)
{
  AutoConnectAux&   mqtt_setting = *portal.aux(portal.where());
  getParams(mqtt_setting);
  AutoConnectInput& mqttserver =
mqtt_setting["mqttserver"].as<AutoConnectInput>();

  File param = FlashFS.open(PARAM_FILE, "w");
  mqtt_setting.saveElement(param, { "ssid", "contrasena", "servidor",
"dispositivo", "NHC", "tiempo"});
  param.close();

  //Parámetros que se mostrarán tras pulsar el botón de guardar:
  AutoConnectText&  echo = aux["parameters"].as<AutoConnectText>();
  echo.value = "SSID: " + ssid_ + + "<br>";
  echo.value += "Contraseña: " + contrasena_ + "<br>";
  echo.value += "Servidor: " + servidor_ + "<br>";
  echo.value += "Dispositivo: " + dispositivo_ + "<br>";
  echo.value += "NHC: " + NHC_ + "<br>";
  echo.value += "Tpo. envío: " + tiempo_ + "<br>";

  Serial.println("Fichero de configuración actualizado");
  conf = true;
  return String("");
}

//*************************************
//                  LOAD
//*************************************
String loadParams(AutoConnectAux& aux, PageArgument& args)
{
  (void)(args);
```

```
  File param = FlashFS.open(PARAM_FILE, "r");
  if (param)
  {
    if (aux.loadElement(param))
    {
      getParams(aux);
      Serial.println("Fichero de configuración cargado");
      conf = true;
    }
    else
    {
      Serial.println("El fichero de configuración no existe");
      param.close();
    }
  }
  else
  {
    Serial.println("Error en el fichero de configuración");
    param.close();
  }
  return String("");
}

//*****************************
//          BATERIA
//*****************************
int bateria()
{
  float bat = (BL.pinRead() - 1615) / 11.7; //La función pinRead
devuelve valores en un rango entre 2788 y 1615
  if (bat <= 0) bat = 1;
  if (bat > 100) bat = 100;
  return bat;
}

//*****************************
//          HORA
//*****************************
String hora()
{
  struct tm timeinfo;
  char timeStringBuff[50];
  if (!getLocalTime(&timeinfo)) return "Error_hora";
  strftime(timeStringBuff, sizeof(timeStringBuff), "%Y-%m-%d
%H:%M:%S", &timeinfo);
  return timeStringBuff;
}

//*****************************
//        PANTALLA
//*****************************
void encenderPantalla(int opcion)
{
  String str = "";
  tft.begin();
  tft.setRotation(1); //Para que se considere la pantalla horizontal
  tft.fillScreen(TFT_BLACK);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
  if (opcion == 1) str = config.apid;
  if (opcion == 2) str = "Error MQTT";
  if (opcion == 3) str = "Error WiFi";
```

```
  if (opcion == 4) str = "Conectado!";
  if (opcion == 5) str = String(medicion[n_mensaje], 2);
  tft.drawCentreString(str, 130, 40, 4); //frase, x, y, fuente
}

void apagarPantalla()
{
  digitalWrite(4, LOW);
}

//*****************************
//           MQTT
//*****************************
void conectarMqtt()
{
  while (!client.connected())
  {
    Serial.println("Intentando conexión MQTT...");
    if (client.connect(dispositivo_.c_str()))
    {
      Serial.println("Conexión MQTT establecida!");
      errorMqtt = 0;
      client.subscribe("RESP01config"); //Este topic tiene que
coincidir con el de publish del fichero resp01.py
    }
    else
    {
      Serial.print("Error -> ");
      Serial.println(client.state());
      delay(500);
      errorMqtt++;
      if (errorMqtt == 120) encenderPantalla(2); //Si llevo 1 minuto
intentado conectarme:
    }
  }
}

void publicarMqtt()
{
  int cont = 0; //Contador del número de mediciones que forman parte
de la media
  float acumulado = 0;
  float media = 0;
  for (int j = 0; j < n_mensaje; j++)
  {
    if (medicion[j] < 40 && medicion[j] > 10) //Se descartan las
mediciones fuera del rango de 10-40 rpm
    {
      acumulado += medicion[j];
      cont++;
    }
  }
  if (cont > 0)
  {
    media = acumulado / cont;
    payload = NHC_ + " " + horas[0] + " " + media + " " + bateria();
    payload.toCharArray(msg, sizeof(msg));
    client.publish(dispositivo_.c_str(), msg);
    Serial.println("[" + dispositivo_ + "]: " + payload );
  }
  n_mensaje = 0;
```

```
}

//*****************************
//            WIFI
//*****************************
void conectarWifi()
{
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid_.c_str(), contrasena_.c_str());
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.println("Conectándose a " + ssid_);
    delay(500);
    if (errorWifi == 120) //Si llevo 1 minuto intentando conectarme
    {
      encenderPantalla(3);
      delay(2000);
      ESP.restart();
    }
    errorWifi++;
  }
  Serial.println("Conectado a " + ssid_);
  Serial.println("Mi dirección IP es " + WiFi.localIP().toString());
  Serial.println("Mi dirección MAC es " + WiFi.macAddress());
  errorWifi = 0;
}

void desconectarWifi()
{
  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);
  Serial.println("WiFi desconectado");
}

//*****************************
//         DORMIR
//*****************************
void dormir(int ms)
{
  esp_sleep_enable_timer_wakeup(ms * 1000);
  Serial.flush();
  esp_light_sleep_start();
  esp_sleep_disable_wakeup_source(ESP_SLEEP_WAKEUP_TIMER);
}

//*****************************
//         REINICIAR
//*****************************
void IRAM_ATTR reiniciar()
{
  Serial.println("Boton izquierdo de reset pulsado");
  FlashFS.format();
  ESP.restart();
}

//*****************************
//         MOSTRAR
//*****************************
void IRAM_ATTR mostrar()
{
  if (auxMostrar == false)
```

```
  {
    auxMostrar = true;
    Serial.println("Encender pantalla");
  }
  else
  {
    auxMostrar = false;
    Serial.println("Apagar pantalla");
  }

}

//*************************************
//                CALLBACK
//*************************************
void callback(char* topic, byte* payload, unsigned int length)
{
  String mensaje = "";
  String opcion, valor;
  String mensajeOld = "";
  Serial.print("Mensaje recibido [");
  Serial.print(topic);
  Serial.print("]: ");
  for (int i = 0; i < length; i++)
  {
    mensaje += (char)payload[i];
  }
  mensaje.trim();
  Serial.println(mensaje);
  for (int i = 0; i < mensaje.length(); i++)
  {
    if (mensaje.substring(i, i + 1) == " ")
    {
      opcion = mensaje.substring(0, i);
      valor = mensaje.substring(i + 1);
      break;
    }
  }
  if (mensaje == mensajeOld) return;
  else
  {
    if (opcion.toInt() == 1) NHC_ = valor.toInt();
    if (opcion.toInt() == 2) t_envio = valor.toInt();
  }
}

//*************************************
//            MAGNETOMETRO
//*************************************
void reset_QMC5883()
{
  Wire.begin(21, 22, 4000000); //21 pin SDA,22 pin SCL,400kHz
velocidad de reloj máxima soportada
  //Configuración para el modo continuo:
  QMC5883_Write(0x0B, 0x01); //Set/Reset period
  QMC5883_Write(0x09, 0x11); // Mode:continuos, ODR=10Hz, Full
scale=2G, OSR=512
}

void getData() //Duración aproximada de la función: 1ms
{
```

```
  /*** CAPTURA ***/
  int16_t x, y, z, state; //Eje x, eje y, eje z, estado
  int m; //Módulo
  static int i = 0; //Cada incremento de esta variable supone 100
milisegundos

  Wire.beginTransmission(ADDR);
  Wire.write(0x00); //0x00: X LSB Register
  Wire.endTransmission();

  Wire.requestFrom(ADDR, 7); //Solicito 7 bytes al magnetómetro
  if (7  <= Wire.available()) {
    x = Wire.read() | (Wire.read() << 8); //Primer registro lsb,
segundo msb
    z = Wire.read() | (Wire.read() << 8);
    y = Wire.read() | (Wire.read() << 8);
    state = Wire.read(); //Posibles estados: ready, overflow, data
skip
  }

  //El magnetometro toma valores dentro del rango (-32678,326767).
  //x, y, z son variables enteras de 16 bits (2^16=65536), para darle
signo negativo:
  if (x > UMBRAL) x -= RANGO_UMBRAL;
  if (y > UMBRAL) y -= RANGO_UMBRAL;
  if (z > UMBRAL) z -= RANGO_UMBRAL;

  m = sqrt(x * x + y * y + z * z);

  //Si hubiera alguna irregularidad, omito el valor del módulo actual
y tomo el anterior:
  if (state != 0 && magne > 0) m = M[magne - 1]; //state=0 -> ready
  if (m > UMBRAL && magne > 0) m = M[magne - 1];
  M[magne] = m;
  magne++;
  if (magne == M_SIZE) //Cada 512 iteraciones reseteo el magnetómetro
para evitar un posible bloqueo
  {
    magne = 0;
    reset_QMC5883();
  }

  // FILTRADO
  #define TS (0.1) //Período de muestreo en segundos

  #define W0 (2*PI*0.9) //Frecuencia de corte del primer filtro en
rad/s
  #define AA (1.0+1.0/TS/W0)
  #define BB (1.0/TS/W0)
  #define A1 ((int)((1.0/AA/AA)*65536))
  #define B1 ((int)((2.0*BB/AA)*65536))
  #define B2 ((int)((-BB*BB/AA/AA)*65536))

  #define W0L (2.0*PI*0.3) //Frecuencia de corte del segundo y tercer
filtro en rad/s
  #define LA (1.0+1.0/TS/W0L)
  #define LB (1.0/TS/W0L)
  #define LA1 ((int)((1.0/LA)*65536))
  #define LB1 ((int)((LB/LA)*65536))

  #define NFLANCOS (4) //Tamaño del buffer de flancos
```

```
  static int x0, y0, y1 = 0, y2 = 0; //Parámetros de la ecuación en
diferencias del primer filtro
  static int yl, yl1 = 0; //Salida y salida anterior del segundo
filtro
  static int yr, yr1 = 0; //Salida y salida anterior del tercer filtro
  static unsigned int out; //Booleana para el estado de la histéresis
  static float frec = 0; //Frecuencia de respiración instantánea
  static unsigned int tfl = 0, flix = 0; //Duración entre flancos e
índice
  //Buffer circular para los tiempos de los flancos inicializado con
valores dummy:
  static unsigned int flt[NFLANCOS] = {DUMMY, DUMMY, DUMMY, DUMMY};

  x0 = m; //Tomo el dato

  //Salida del primer filtro:
  y0 = (A1 * x0 + B1 * y1 + B2 * y2) >> 16; //2^16=65536, anulo el
factor 65536 los defines

  //Salida del segundo filtro;
  yl = (LA1 * y1 + LB1 * yl1) >> 16;

  //Salida del tercer filtro:
  yr = (LA1 * abs(y1 - yl1) + LB1 * yr1) >> 16; //El diodo provoca el
valor absoluto

  y2 = y1; y1 = y0; yl1 = yl; yr1 = yr; //Las salidas anteriores se
actualizan a las actuales

  if ((i - tfl) > 200) frec = 0; //Si no hay cambios durante 20
segundos, frecuencia=0

  //Histéresis
  yr >>= 2; //Divido entre 2^2 porque la ganancia del amplificador es
1/4
  //Si el resultado del comparador da y0 como grande y el flanco está
bajo:
  if ((y0 > (yl + yr)) && (out == 0))
  {
    out = 1;
    if (flt[flix] != DUMMY) frec = (NFLANCOS * 6000 + (i - flt[flix])
/ 2) / (i - flt[flix]); //Hago una media móvil
    flt[flix] = tfl = i; //Guardo el tiempo que duró el flanco
    if (++flix >= NFLANCOS) flix = 0; //Si ya he guardado 4, reinicio
el indice
  }
  //Si el resultado del comparador da y0 como pequeño y el flanco está
alto:
  if ((y0 < (yl - yr) && (out == 1))) out = 0;

  if (i % 10 == 0) //i=10 implica 1 segundo
  {
    printf("%4d %4d %4d %4d %4d %4d\n", x, y, z, m, state, i);
    medicion[n_mensaje] = frec / 10;

    if (auxMostrar == true) encenderPantalla(5);
    else apagarPantalla();

    horas[n_mensaje] = hora();
    Serial.println(horas[n_mensaje] + " " + medicion[n_mensaje]);
```

```
    n_mensaje++;
  }
  i++;
  n_getData++;
}


//*************************************
//                SETUP
//*************************************
void setup()
{
  Serial.begin(115200);
  setCpuFrequencyMhz(80);

  pinMode(BOTON_DCHA, INPUT);
  pinMode(BOTON_IZQ, INPUT);
  attachInterrupt(digitalPinToInterrupt(BOTON_DCHA), mostrar,
FALLING);
  attachInterrupt(digitalPinToInterrupt(BOTON_IZQ), reiniciar,
FALLING);

  FlashFS.begin(true);

  portal.join({aux1, aux2});
  AutoConnectAux& mqtt_setting = *portal.aux(AUX_SETTING_URI);
  PageArgument  args;
  loadParams(mqtt_setting, args);

  config.apid = "RESP01"; //Nombre del SSID
  config.psk = "contraseña"; //Contraseña del SSID
  config.menuItems = AC_MENUITEM_DISCONNECT;
  config.title = "Menú principal";
  config.immediateStart = true;
  portal.config(config);

  portal.on(AUX_SETTING_URI, loadParams);
  portal.on(AUX_SAVE_URI, saveParams);

  if (conf == false)
  {
    Serial.println("Conéctese a la red WiFi " + config.apid);
    encenderPantalla(1);
    portal.begin();
  }
  apagarPantalla();

  conectarWifi();
  client.setServer(servidor_.c_str(), 1883); //Puerto 1883 por defecto
  client.setCallback(callback); //Para poder recibir mensajes
  conectarMqtt();

  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

  encenderPantalla(4);
  delay(2000);
  apagarPantalla();

  client.disconnect();
  delay(1000);
  desconectarWifi();
```

```
}

//*************************************
//                  LOOP
//*************************************
void loop()
{
  t0 = micros();
  dormir(90); //Duración aproximada de la función: 93 ms
  while (micros() - t0 < 100000 ); // Espera a que transcurran 100 ms
  getData(); //Duración aproximada de la función: 2 ms
  if (n_getData > 0 && n_getData % (t_envio * 10) == 0) // Se
multiplica por 10 porque se toman 10 muestras por segundo (getData
cada 100 ms)
  {
    ticker.attach(0.1, getData); //Se activa el ticker con llamada
cada 0.1 segundos
    conectarWifi();
    t1 = millis();
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
    while (millis() - t1 < 1000)
    {
      if (!client.connected()) conectarMqtt();
      else client.loop(); //Muy irregular, a veces entra al callback a
los 100 ms y otras a los 3 seg
    }
    publicarMqtt();
    client.disconnect();
    t1 = micros();
    while (micros() - t1 < 200000); //Retardo imprescindible para que
DISCONNECT llegue al broker
    desconectarWifi();
    ticker.detach(); //Se desactiva el ticker
  }
}
```

A.2. Fichero database.py

```
# -*- coding: utf-8 -*- #Permite usar tildes

import paho.mqtt.client as mqtt
import sys
import MySQLdb

# Abrir conexión con bases de datos
try:
    db = MySQLdb.connect("127.0.0.1","root","root","basedatos")
    # host, usuario, contraseña, base de datos
except:
    print("No se pudo conectar con la base de datos")
    print("Cerrando...")
    sys.exit()

# Preparando cursor
cursor = db.cursor()

# Callback para cuando se recibe un CONNACK del servidor:
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Conectado al broker")
        client.subscribe("RESP01")
    else:
        print(u"Conexión fallida - Error: " + str(rc))
        sys.exit()

# Callback para cuando el broker recibe un mensaje en el tópico
suscritoFin:
def on_message(client, userdata, msg):
    print(msg.topic + "-> " + msg.payload)
    segmentoPay = msg.payload.split(" ")

    sql = """INSERT INTO `basedatos`.`respirometro`
(`dispositivo`,`NHC`, `hora_medida`, `rpm`, `bateria`, `hora_volcado`)
VALUES ('""" +msg.topic + """', '""" + segmentoPay[0] + """', '""" +
segmentoPay[1] + " " + segmentoPay[2] + """', '""" + segmentoPay[3] +
"""', '""" + segmentoPay[4] + """', CURRENT_TIMESTAMP);"""
    try:
        # Ejecutar el comando SQL anterior
        cursor.execute(sql)
        # Commit para que los valores queden en la base de datos
        db.commit()
        print("Guardando en base de datos...OK")
    except:
        db.rollback() #Devuelve la base de datos a un estado previo
(descarta los cambios de la última sentencia)
        print("Guardando en base de datos...Error")

client = mqtt.Client("Base de datos")       # Creo un nuevo cliente
client.on_connect = on_connect              # Llamo a on_connect
client.on_message = on_message              # Llamo a on_message

try:
    client.connect("127.0.0.1", port=1883, keepalive=60)
except:
    print("No se pudo conectar con el MQTT Broker...")
    print("Cerrando...")
    db.close() #Importante cerrar la base de datos
```

```
    sys.exit()

# Bucle infinito hasta que se preciosa Crtl + C para salir
try:
    client.loop_forever()
except KeyboardInterrupt:
    print("Cerrando...")
    db.close()
    sys.exit()
```

A.3. Fichero softwareTTGO.ino

```
# -*- coding: utf-8 -*-

import paho.mqtt.client as mqtt
import sys

# Callback para cuando se recibe un CONNACK del servidor:
def on_connect(client, userdata, flags, rc):
    if rc != 0:
        print(u"Conexión fallida - Error: " + str(rc))
        sys.exit()

client = mqtt.Client("Reconfig")           # Creo un nuevo cliente
client.on_connect= on_connect              # Llamo a on_connect

try:
    client.connect("127.0.0.1", port=1883, keepalive=60)
except:
    print("No se pudo conectar con el MQTT Broker...")
    print("Cerrando...")
    sys.exit()

client.loop_start()

dispositivo=raw_input('Nombre del dispositivo [RESPXXconfig]: ')
parametro=int(raw_input('1:NHC\n2:Tiempo entre envios\nIntroduzca el
numero asociado a la opcion del parametro a modificar: '))
if parametro==1: valor=raw_input('Introduzca el nuevo NHC: ')
if parametro==2: valor=raw_input('Introduzca cada cuantos segundos se
quiere realizar el volcado: ')
mensaje=str(parametro) + " " + valor
publicacion=client.publish(dispositivo,mensaje,qos=1,retain=True) #Si
nadie esta suscrito a mi publicacion y no retengo el mensaje, se
pierde
publicacion.wait_for_publish() #No me desconecto hasta que se haya
publicado el mensaje

client.disconnect()
client.loop_stop()
```

Apéndice B

Guía de instalación

Para el correcto funcionamiento del proyecto, hacen falta una serie de programas. Los archivos necesarios para la instalación se adjuntarán en una carpeta llamada "Instalación". Dentro de ella se encontrarán cinco carpetas: "instMosquitto", "python", "baseDatos", "arduino" y "scripts". A continuación, se explicará cómo instalar cada uno de los programas necesario.

B.1 Instalación de Mosquitto

Los tres archivos necesarios están en la carpeta "instMosquitto". También pueden obtenerse, al menos para un sistema operativo Windows 10 de 64 bits, a través de los siguientes enlaces:

- Win64OpenSSL-1_1_1k.exe: http://slproweb.com/download/Win32OpenSSL-1_1_1k.exe [41].
- mosquitto-1.6.12a-install-windows-x64.exe: https://mosquitto.org/files/binary/win64/mosquitto-1.6.12a-install-windows-x64.exe [41].
- mosquittorar: https://mega.nz/file/bZdB2KTR#KUTrA_80hsA0o5GNOrt0kXe8CpOeVGEhgPqApbGz9Vg [41].

Pasos a seguir:

1. En "instMosquitto", doble click a "Win64OpenSSL-1_1_1k.exe". Aparecerán diferentes pasos para la instalación, hay que darle sucesivamente a "Ejecutar", "I accept the agreement", "The OpenSSL binaries (bins) directory", "Install", deseleccionar la opción de donar, y "Finish". Si no está instalado "Microsoft Visual C++ 2017 Redistributable (x64)", se pedirá su instalación. Basta con darle a "Sí", comenzará la descarga automáticamente, doble clic, "Instalar", "Cerrar".
2. En "instMosquitto", doble clic a "mosquitto-1.6.12a-install-windows-x64.exe" y, sucesivamente, clic en "Next", "Next", "Install", y "Finish". Si no está instalado "Microsoft Visual C++ 2015-2019 Redistributable (x64), se pedirá su instalación. Basta con darle a "Sí", comenzará la descarga automáticamente, doble clic, "Instalar", "Cerrar".
3. En "C:\Program Files\OpenSSL-Win64", copiar la carpeta "bin" y pegarla en "C:\Program Files\mosquitto".

4. Abrir “mosquittorar” y copiar los tres ficheros con extensión dll “libeay32.dll”, “pthreadVC2.dll” y “ssleay32.dll, para pegarlos en la dirección “C:\Program Files\mosquitto”.
5. Dentro de “instMosquitto”, en “mosquittorar”, doble clic en “mosquitto-1.4.9-install-win32.exe”, seguidamente “Next”, “Next”, “Next”. Antes de darle a “Install”, hay que asegurarse de que la dirección sea “C:\Program Files\mosquitto” y no “C:\Program Files(x86)\mosquitto”. Finalmente, “Finish”.

B.2 Instalación de Python

Los archivos necesarios están en la carpeta “python”. También pueden obtenerse, al menos para un sistema operativo Windows 10 de 64 bits, a través de los siguientes enlaces:

- python-2.7.13.msi: https://www.python.org/ftp/python/2.7.13/python-2.7.13.msi [42].
- paho-mqtt: https://pypi.org/project/paho-mqtt/1.3.0/#files [43].
- MySQL-python-1.2.5.win32-py2.7.exe: https://pypi.org/project/MySQL-python/ [44].

Pasos a seguir:

1. En “python”, doble clic en “python-2.7.13.msi”, “Install for all users”, “Next”. En la ventana “Customize Python”, seleccionar en “Add Python.exe to Path” la opción “Will be installed on local hard drive” para así poder ejecutar Python en cualquier directorio. Después “Next” y, finalmente, “Finish”.
2. En el buscador de la barra de Windows, escribir “Símbolo del sistema” para abrir la consola de comandos y, a continuación, ejecutar el comando “pip install --user paho-mqtt”.
3. En “python”, doble clic en “MySQL-python-1.2.5.win32-py2.7.exe”, “Siguiente”, “Siguiente”, “Siguiente”.

B.3 Instalaciones para el manejo de bases de datos

Los archivos necesarios están en la carpeta “baseDatos”. También pueden obtenerse, al menos para un sistema operativo Windows 10 de 64 bits, a través de los siguientes enlaces:

- mariadb-10.5.8-winx64.ms: https://downloads.mariadb.org/mariadb/10.5.8/ [45].
- Workbench-Build115: https://www.sql-workbench.eu/archive/Workbench-Build115.zip [46].
- jre-8u291-windows-x64.exe: https://www.oracle.com/es/java/technologies/javase-jre8-downloads.html [47].

Pasos a seguir:

1. En “baseDatos”, abrir “mariadb-10.5.8-winx64.msi” e ir dándole a “I accept the terms in the License Agreement”, “Next”, introducir la contraseña que se desee, “Next”, “Next”, “Install”, “Finish”. En el escritorio aparecerá un icono del acceso directo HeidiSQL. Clicando en él y dándole a “Nueva”, aparecerá una pantalla con diferentes opciones que hay que dejar tal como se muestra en la Figura B.1, escribiendo en el apartado de contraseña la establecida en el paso anterior. A continuación, clicando en “Abrir”, aparecerá lo mostrado en la Figura B.2. “Test” es el nombre por defecto de la base de datos generada. Haciendo clic derecho sobre ella, aparecerá la opción a “Editar” para cambiar el nombre.

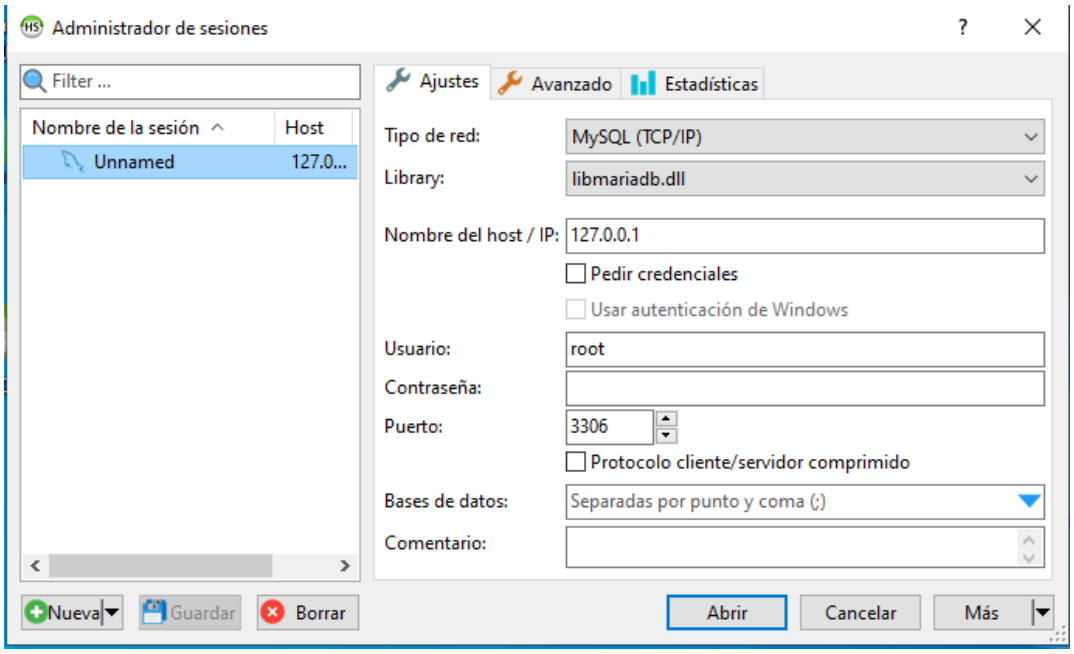

Figura B.1: Administrador de sesiones en HeidiSQL

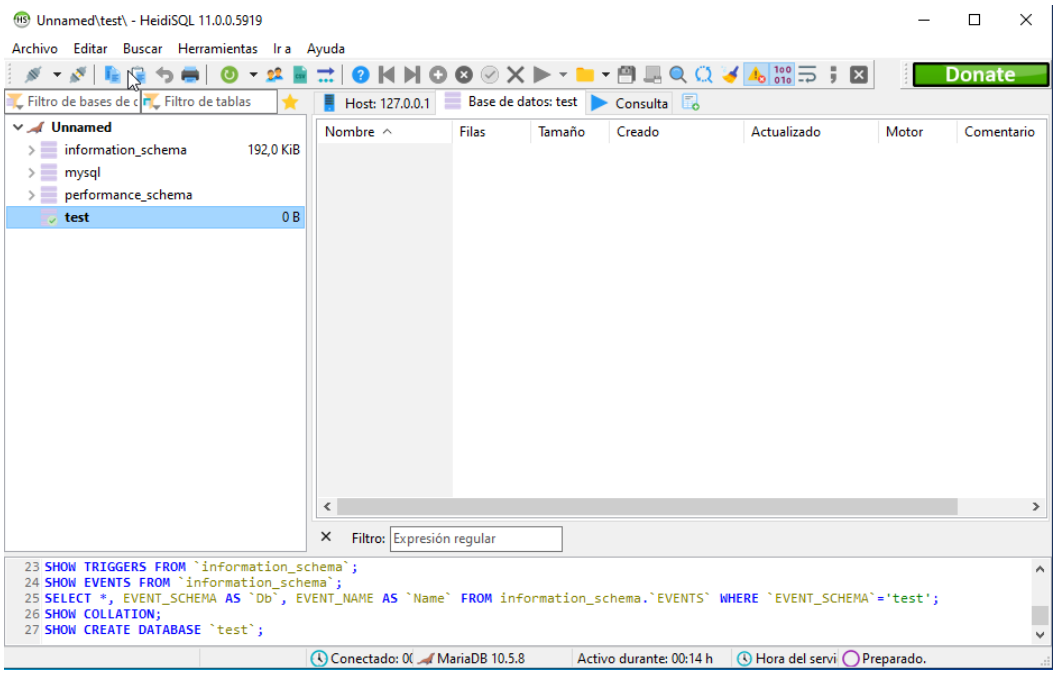

Figura B.2: Administrador de bases de datos en HeidiSQL.

2. En “baseDatos”, abrir la carpeta “Workbench-Build115” y hacer doble clic en “SQLWorkbench.exe”. Si al abrir apareciera un error de Java, habría que volver a “baseDatos” y hacer doble clic en “jre-8u291-windows-x64.exe”. En este caso, aparecerán una serie de campos a completar con el contenido mostrado en la Figura B.3, escribiendo de nuevo en el apartado de contraseña la establecida anteriormente. Al seleccionar la opción de “Driver” saldrá una ventana emergente, en la cual hay que clicar en “Yes” y luego “OK”. Se generará automáticamente una “URL”, en donde hay que sustituir la palabra *hostname* por la IP del ordenador en donde se está trabajando, y *port* por “3306”. El nombre de la base de datos dependerá de lo que se haya hecho en el paso anterior, “test” si no se ha se modificado. De nuevo, el usuario sería “root” y la contraseña la que se hubiése elegido antes. Para finalizar, clic en “OK”.

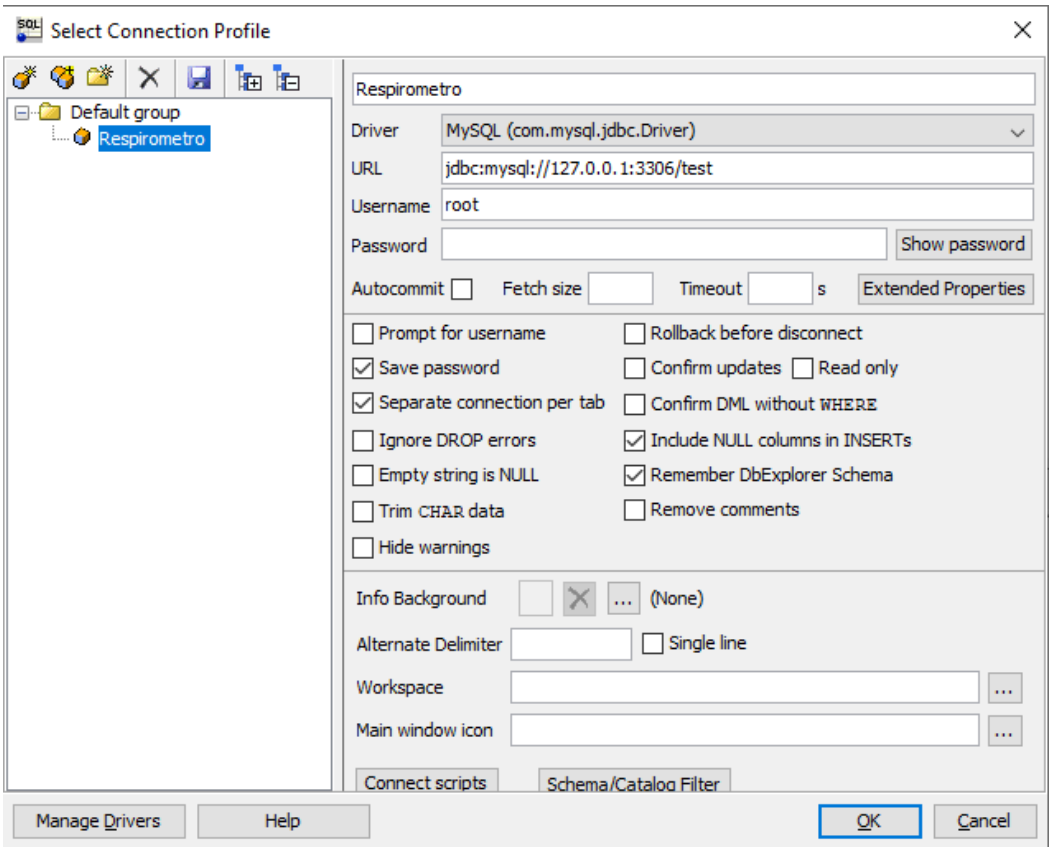

Figura B.3: Administrador de sesiones en SQLWorkbench.

B.4 Instalación de Arduino

Los archivos necesarios están en la carpeta “arduino”. También pueden obtenerse a través de los siguientes enlaces

- Arduino-1.8.15-windows.exe: https://www.arduino.cc/en/software [48].
- AutoConnect: https://github.com/Hieromon/AutoConnect.git [49].
- ArduinoJson: https://github.com/bblanchon/ArduinoJson.git [50].
- PageBuilder: https://github.com/Hieromon/PageBuilder.git [51].

- PubSubClient: https://github.com/knolleary/pubsubclient.git [52].
- TFT_eSPI: https://github.com/Bodmer/TFT_eSPI.git [53].
- 18650CL: https://github.com/pangodream/18650CL [17].

Pasos a seguir:

1. En la carpeta "arduino", doble clic en "Arduino-1.8.15-windows.exe". A continuación, se va seleccionando "I agree", "Next" e "Install". Cuando termine de instalarse, saltará una ventana preguntando si se quiere instalar un *software* de Adafruit Industries. Seleccionar "No instalar". A continuación, saltará una nueva ventana, preguntando por un *software* de Arduino srl. Esta vez elegimos "Instalar". Finalmente, "Close".
2. En el escritorio se habrá creado un acceso directo a Arduino. Para llegar a la ventana de la Figura B.4 hay que acceder al menú "Archivo>Preferencias" dentro de Arduino y pegar la URL https://dl.espressif.com/dl/package_esp32_index.json dentro del campo "Gestor de URLs adicionales de Tarjetas". Clic en "Ok". A continuación, se tiene que ir al menú "Herramientas>Placa>Gestor de tarjetas", buscar la palabra "esp32" y darle a "Instalar", Figura B.5. Una vez instalado, "Cerrar". Ahora, al ir de nuevo a "Herramientas>Placa", aparecerá una nueva categoría, "ESP32 Arduino", en la que hay que seleccionar "TTGO T1", puesto que es el modelo de ESP32 utilizado.
3. En la carpeta "librerias" de la carpeta "arduino" se encuentran una serie de librerías de las que requiere nuestro código fuente. Para añadir estas librerías, hay que irr al menú "Programa>Incluir Librería>Añadir biblioteca.ZIP…", acceder a la dirección donde se encuentre la carpeta "librerias" y seleccionar la librería correspondiente.

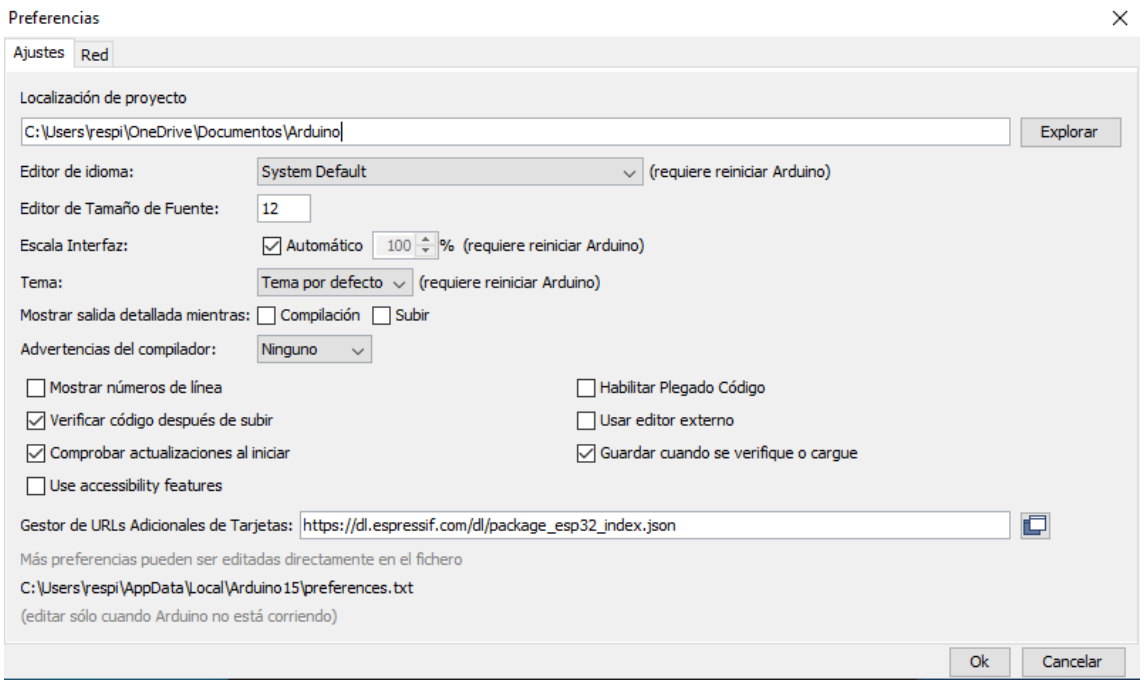

Figura B.4: Arduino Archivo>Preferencias

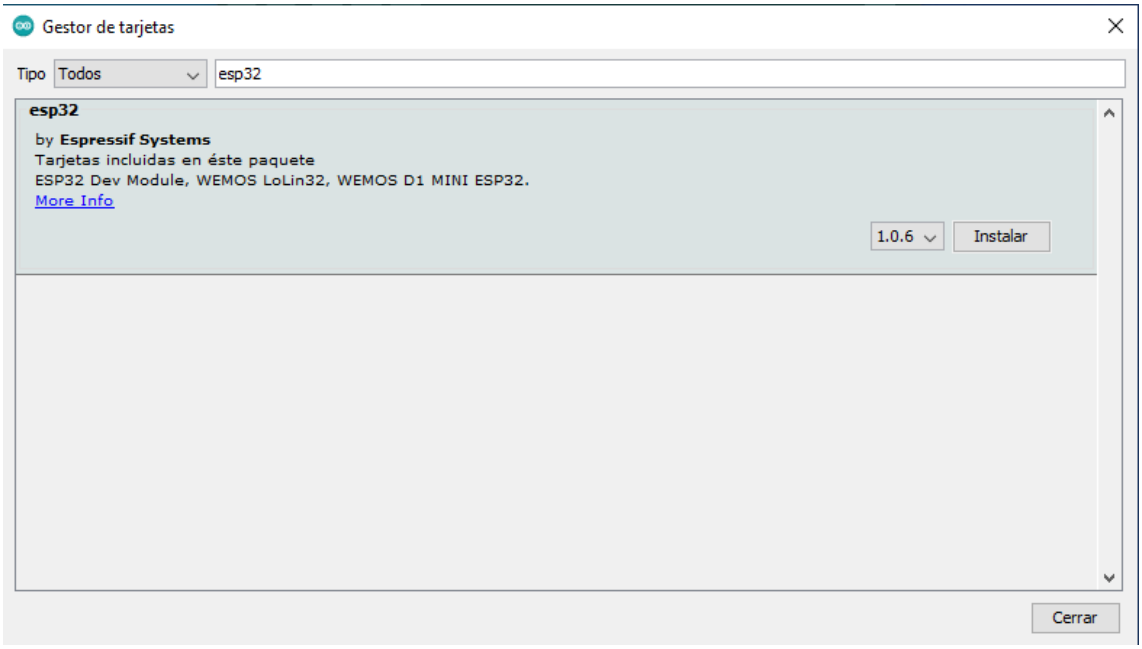

Figura B.5: Arduino Herramientas>Placa>Gestor de tarjetas

B.5 Manejo de los ficheros de código desarrollados

En la carpeta “scripts” se encuentran los ficheros de código que se han desarrollado: “AutoconnectPage.cpp”, “database.py”, “reconfig.py” y “softwareTTGO”.

Pasos a seguir:

1. El fichero “AutoconnectPage.cpp” se tiene que pegar en la dirección “C:\Users\nombre_usuario\OneDrive\Documentos\Arduino\libraries\/Autoconnect-master\src”. Saltará una ventana emergente diciendo que ya existe un archivo con ese nombre, por lo que hay que seleccionar la opción de “Reemplazar el archivo en el destino”.
2. Los ficheros “database.py” y “resp01.py” van en la dirección “C:\Program Files\mosquitto”.

Referencias

[1] World Health Organization (2019). *WHO guideline recommendations on digital interventions for health system strengthening*. (2019). https://apps.who.int/iris/bitstream/handle/10665/311941/9789241550505-eng.pdf?ua=1.

[2] Marqués, F. L. (2016, 1 septiembre). *Historia de la Telemedicina desde sus inicios hasta hoy*. Clinic Cloud. https://clinic-cloud.com/blog/historia-de-la-telemedicina/#historia-de-la-telemedicina

[3] Comisión Europea. (2010, agosto). *Una Agenda Digital para Europa*. https://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:52010DC0245R(01)&from=el

[4] Gabinete de Prensa del Ministerio de Sanidad del Gobierno de España (2020, 4 agosto). *El Gobierno refuerza la estructura del Ministerio de Sanidad con la Secretaría de Estado de Sanidad y la Secretaría General de Salud Digital, Información e Innovación del SNS.* Notas de Prensa. https://www.mscbs.gob.es/gabinete/notasPrensa.do?id=5022

[5] Morteo, F., & Bocalandro, N. (2004). *Un enfoque práctico de SQL* (2.ª ed.). Ediciones Cooperativas.

[6] Nieto, N., & Vega, M. L. (2017, marzo). *Diseño de un prototipo de medición de señales fisiológicas utilizadas en Biofeedback*. Universidad Nacional de Córdoba – Facultad de Ciencias Exactas Físicas y Naturales. https://rdu.unc.edu.ar/bitstream/handle/11086/4808/Informe%20PI%20-%20Nieto-Vega.pdf?sequence=1

[7] *Chipsets*. Espressif Systems. https://www.espressif.com/en/products/socs

[8] Magnetorresistencia. En *Diccionario de la lengua española*. Real Academia Española. https://dle.rae.es/magnetorresistencia

[9] Llamas, L. (2019, 17 abril). *¿Qué es MQTT? Su importancia como protocolo IoT*. Luis Llamas. https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/

[10] *LILYGO® TTGO T-Display ESP32 WiFi and Bluetooth Module Development Board For Arduino 1.14 Inch LCD(1)(1)(1)*. Lilygo. http://www.lilygo.cn/claprod_view.aspx?TypeId=62&Id=1274

[11] QST. (2016). *3-Axis Magnetic Sensor QMC5883L*. QST corporation. https://img.filipeflop.com/files/download/Datasheet-QMC5883L-1.0%20.pdf

[12] Luis Llamas. [Brújula magnética GY-273]. https://www.luisllamas.es/brujula-magnetica-con-arduino-compass-digital-hmc5883/

[13] Amazon. [Dimensiones de la batería 1100 mAh MakerFocus]. https://www.amazon.com/-/es/bater%C3%ADa-recargable-protecci%C3%B3n-aislada-desarrollo/dp/B0867KDMY7

[14] AliExpress. [Dimensiones del imán de neodimio]. https://es.aliexpress.com/item/1005001536647667.html

[15] *Proteus*. Proteus Labcenter. https://www.labcenter.com/

[16] *Tinkercad*. Tinkercad. https://www.tinkercad.com/

[17] *Anycubic i3 Mega*. Anycubic. https://www.anycubic.com/products/anycubic-i3-mega

[18] Ministerio de Sanidad, Consumo y Bienestar Social (2019). *Hospitales, Camas en funcionamiento y Puestos de Hospital de Día (PHD) del Sistema Nacional de Salud (SNS), número y tasa por 1.000 habitantes y número de Centros, Servicios y Unidades de Referencia (CSUR) según comunidad autónoma*. Sanidad en datos. https://www.mscbs.gob.es/estadEstudios/sanidadDatos/tablas/tabla22.htm

[19] *Mosquitto*. Eclipse Mosquitto. https://mosquitto.org/

[20] *Language Reference*. Arduino. https://www.arduino.cc/reference/en/

[21] *WiFi library*. Arduino. https://www.arduino.cc/en/Reference/WiFi

[22] *Autoconnect*. Arduino. https://www.arduino.cc/reference/en/libraries/autoconnect/

[23] *ArduinoJson*. Arduino. https://www.arduino.cc/reference/en/libraries/arduinojson/

[24] *PageBuilder*. Arduino. https://www.arduino.cc/reference/en/libraries/pagebuilder/

[25] *PubSubClient*. Arduino. https://www.arduino.cc/reference/en/libraries/pubsubclient/

[26] *TFT_eSPI*. Arduino. https://www.arduino.cc/reference/en/libraries/tft_espi/

[27] Pangodream, *18650CL*. GitHub. https://github.com/pangodream/18650CL

[28] Espressif Systems. *espressif/arduino-esp32/libraries/SPIFFS/*. GitHub. https://github.com/espressif/arduino-esp32/tree/master/libraries/SPIFFS

[29] Espressif Systems. *espressif/arduino-esp32/libraries/FS/*. GitHub. https://github.com/espressif/arduino-esp32/tree/master/libraries/FS

[30] Espressif Systems. *espressif/arduino-esp32/libraries/Wire/*. GitHub. https://github.com/espressif/arduino-esp32/tree/master/libraries/Wire

[31] *Ticker*. Arduino. https://www.arduino.cc/reference/en/libraries/ticker/

[32] *Sleep Modes*. Espressif Systems API. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html

[33] *MariaDB Foundation*. MariaDB. https://mariadb.org/

[34] *HeidiSQL Home*. HeidiSQL. https://www.heidisql.com/

[35] Kellerer, T.. *SQL Workbench - Home*. SQL Workbench. https://www.sql-workbench.eu/

[36] *Python*. Python Software Foundation. https://www.python.org/

[37] Dustman, A.. *MySQLdb's documentation*. MySQLdb. https://mysqlclient.readthedocs.io/

[38] Eclipse Foundation. *Eclipse Paho*. Paho MQTT Python Client. https://www.eclipse.org/paho/index.php?page=documentation.php

[39] Bos, B.. *Cascading Style Sheets home page*. CSS. https://www.w3.org/Style/CSS/#specs

[40] Hospital 12 de Octubre. [Rango de frecuencias respiratorias en función de la edad].

[41] Rodrigo Cortes. (2020, 5 noviembre). *¿Cómo instalar mosquitto en windows 10?* [Vídeo]. YouTube. https://www.youtube.com/watch?v=b655n9qZTug

[42] *Python* (2.7.13). (2016). Python [Software de ordenador]. https://www.python.org/downloads/release/python-2713/

[43] *Paho mqtt* (1.3.0). (2017). [Librería de Python]. PyPi. https://pypi.org/project/paho-mqtt/1.3.0/#files

[44] *MySQL* (1.2.5). (2014). [Librería de Python]. PyPi. https://pypi.org/project/MySQL-python/

[45] *MariaDB* (10.5.8). (2020). [Sistema de gestión de bases de datos]. MariaDB. https://downloads.mariadb.org/mariadb/10.5.8/

[46] *SQL Workbench* (115). [Sistema de gestión de bases de datos]. SQLWorkbench https://www.sql-workbench.eu/download-archive.html

[47] *Java SE Runtime Environment* (8). (2019). Oracle. [Utilidades para la ejecución de programas Java]. https://www.oracle.com/es/java/technologies/javase-jre8-downloads.html

[48] *Arduino IDE* (1.8.16). (2020). Arduino. [Entorno de desarrollo]. https://www.arduino.cc/en/software

[49] *AutoConnect* (1.2.2). (2020). Hieromon. [Librería de Arduino]. https://github.com/Hieromon/AutoConnect

[50] *ArduinoJson* (6.18.1). (2020). Bblanchon. [Librería de Arduino]. https://github.com/bblanchon/ArduinoJson

[51] *PageBuilder* (1.4.3). (2021). Hieromon. [Librería de Arduino]. https://github.com/Hieromon/PageBuilder.git

[52] *PubSubClient* (2.8). (2020). Knolleary [Librería de Arduino]. https://github.com/knolleary/pubsubclient

[53] *TFT_eSPI* (2.3.6). (2021). Bodmer [Librería de Arduino] https://github.com/Bodmer/TFT_eSPI