Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

PROYECTO FIN DE CARRERA

Dispositivo sensorial biométrico autónomo con monitorización remota en tiempo real

Autor: Andrés Falo Morales
Directores: Luis Antonio Martín Nuez, Javier Solobera Abad

Ponente: José María López Pérez

Ingeniería Técnica Industrial
Especialidad Electrónica

Curso 2014/2015

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

PROPUESTA y ACEPTACIÓN DEL PROYECTO FIN DE CARRERA DE INGENIERÍA TÉCNICA

DATOS PERSONALES

APELLIDOS, Nombre: Falo Morales, Andrés

Nº DNI 72986708A Dirección Cuartal, 8

C.P. 50820 Localidad Zaragoza

Provincia Zaragoza Teléfono 619827582 NIA: 562280

Firma:

DATOS DEL PROYECTO FIN DE CARRERA

INGENIERIA TECNICA INDUSTRIAL, Especialidad Electrónica Industrial

TITULO Dispositivo sensorial biométrico autónomo con monitorización remota en tiempo real

DEPÓSITO EN: ZAGUAN (Obligatorio) ☒ y CD-ROM ☐ (si PFC es tipo B aplicación informática)

DIRECTOR José María López Pérez

VERIFICACIÓN EN SECRETARÍA

El alumno reúne los requisitos académicos **(1)** para la adjudicación de Proyecto Fin de Carrera

SELLO DEL CENTRO

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

EL FUNCIONARIO DE SECRETARIA

Fdo.:

SE ACEPTA LA PROPUESTA DEL PROYECTO **(2)**	SE ACEPTA EL DEPÓSITO DEL PROYECTO
En Zaragoza, a 21 de enero de 2.015	En Zaragoza, a 18 de febrero de 2.015
Fdo.: José María López	Fdo.: José María López
DIRECTOR DEL PFC	DIRECTOR DEL PFC

(1) Requisitos académicos: tener pendientes un máximo de 24 créditos o dos asignaturas para finalizar la titulación.

(2) Para que la propuesta sea aceptada por el Director, es imprescindible que este impreso esté sellado por la Secretaría de la EINA una vez comprobados los requisitos académicos.

Dispositivo sensorial biométrico autónomo con monitorización remota en tiempo real

RESUMEN

Este proyecto ha sido realizado en la empresa Libelium Comunicaciones Distribuidas S.L. en concreto en su sección Cooking Hacks en la que se realizan proyectos usando Software y Hardware libre.

Las plataformas utilizadas en este proyecto han sido Arduino, diseñada para utilizar la electrónica en proyectos multidisciplinares y distribuida por Libelium y la placa de adaptación de sensores biométricos e-Health creada por Cooking Hacks.

Con la realización de este proyecto no se pretende crear un nuevo producto sino desarrollar la posibilidades de integración de distintos módulos de Arduino con la placa e-Health y mostrar su capacidad a los usuarios.

La finalidad es crear un dispositivo que sea capaz de medir diferentes parámetros biométricos con varios sensores ya existentes para la placa e-Health durante el ejercicio físico o para usarlo con una persona accidentada en un lugar de difícil acceso y que a su vez mande la información de forma inalámbrica a un punto central, todo esto se tiene que llevar a cabo de forma autónoma por lo que se alimenta mediante baterías y que se puede alargar su autonomía mediante un panel solar. También incluye un pequeño display en el que el propio portador u otra persona puede ver la información.

Para ello se ha estudiado los diferentes módulos que compondrán el montaje final y se han adaptado las diferentes librerías para Arduino MEGA, posteriormente se ha creado una placa integradora para los distintos módulos y finalmente un código principal para cumplir todos los objetivos planteados.

Para concluir se han hecho pruebas para comprobar el correcto funcionamiento de todo el montaje y para testear la comunicación inalámbrica.

Tabla de contenidos

1 Introducción....8
1.1 Marco de trabajo: Libelium y sus productos....8
1.2 Tecnología Open Source....8
1.3 Motivación del PFC....10
1.4 Objetivos del PFC....10
2 Campo de aplicación....12
2.1 Posibles aplicaciones del sistema....12
2.2 Estado del arte....12
3 Especificaciones del sistema....16
3.1 Tecnologías de partida....16
3.1.1 Arduino....16
3.1.2 e-Health....18
3.2 Parámetros de interés para la monitorización....19
3.3 Otras especificaciones para el diseño final....20
4 Hardware....22
4.1 Selección del Hardware....22
4.2 Diagrama de bloques....22
4.2.1 Comunicación entre módulos....23
4.3 Arduino MEGA 2560....24
4.4 TFT LCD 2.2”....26
4.5 SX1272 LoRa + Multiprotocol....27
4.6 3G + GPS....28
4.7 Solar Charger Shield v2....29
4.7.1 Batería Li-Ion....30
4.7.2 Panel solar flexible....30
4.8 Sensores....31
4.9 e-Health Portable Board....34
4.9.1 Diagrama de circuito esquemático....36
4.9.2 Placa de circuito impreso....37

5 Software.....40
5.1 Introducción.....40
5.2 Software de base: Arduino IDE.....40
5.3 Lenguaje de programación: C/C++.....41
5.4 Programa principal.....42
5.5 Software TFT LCD.....44
5.6 Software asociado a los sensores.....45
5.7 Software creación de la trama para su envío.....47
5.8 Software LoRa.....47
5.9 Software 3G+GPS.....51
5.10 Otras funciones.....54
6 Diseño de la carcasa.....56
7 Pruebas.....58
7.1 Pruebas de consumo.....58
7.2 Pruebas cobertura.....59
7.3 Pruebas del sistema completo.....61
8 Conclusiones.....64
8.1 Conclusiones personales.....65
8.2 Agradecimientos.....65
9 Bibliografía y referencias.....66
Índice de Figuras.....68
Índice de Tablas.....70
Glosario.....72
Anexo A: Códigos de programación.....74
A.1. Código completo de ejemplo.....74
A.2. Funciones.....77
A.3. Modificaciones en las librerías.....86
Anexo B: Documentación online generada.....88

1 Introducción

1.1 Marco de trabajo: Libelium y sus productos

El presente Proyecto Fin de Carrera se ha realizado desde Julio de 2014 hasta Febrero de 2015 en la empresa Libelium Comunicaciones Distribuidas S.L. bajo la dirección de **D. Javier Solobera Abad**, encargado de producto de la empresa y Luis Antonio Martín Nuez encargado I+D de la sección Cooking Hacks, y en la tarea de Ponente se ha contado con la supervisión de **D. José María López Pérez**, Profesor del departamento de Ingeniería Electrónica y Comunicaciones de la Universidad de Zaragoza.

El proyecto se ha llevado a cabo íntegramente en las instalaciones de Libelium en el Centro Europeo de Empresas e Innovación de Aragón, donde se han facilitado todas las herramientas y materiales.

Libelium nace en 2006 como empresa spin-off de la Universidad de Zaragoza. Se dedica al diseño y fabricación de hardware para la implementación de redes sensoriales inalámbricas, redes malladas y protocolos de comunicación para todo tipo de redes inalámbricas distribuidas. A su vez dentro de la empresa se encuentra la sección Cooking Hacks dedicada a la venta de hardware para la creación de proyectos DIY (Do It Yourself) en la cual se enmarca este proyecto.

1.2 Tecnología Open Source

Como se ha dicho anteriormente el proyecto a sido realizado dentro de la sección Cooking Hacks, la división Open Hardware de Libelium, cuyo objetivo es extender la electrónica a todo el mundo haciéndola asequible, fácil de aprender, divertida y soportada por una gran Comunidad.

Fig. 1: Logo de Cooking Hacks

Por lo tanto una de las características más importantes a destacar de este proyecto es que en gran medida se ha usado tecnología Open Source tanto a nivel de software como hardware.

El término Open Source se acuñó específicamente para el desarrollo de software e implicaba que un programador diera a conocer el código para cierto programa, aplicación, etc. de manera que otros programadores pudieran utilizarlo e implementarlo en sus propios desarrollos. Lo más importante de este proceso sería, que al obtener el código otro programador este realizara mejoras al mismo y también las compartiera por la misma vía por la que había obtenido la información inicial, ayudando a la comunidad a generar mejores estrategias, productos, desarrollos, etc.

Fig. 2: Logo Open Software

Esto derivó también a lo que llamamos Open Hardware que son aquellos dispositivos electrónicos cuyas especificaciones y diagramas esquemáticos son de acceso público.

Fig. 3: Logo Open Hardware

En estos últimos años, la tecnología basada en hardware y software libre, como por ejemplo Arduino (sistema desarrollado basado en microcontroladores ATmega), ha iniciado un movimiento muy importante entre la sociedad electrónica, propulsado por las numerosas ventajas que este tipo de dispositivos facilitan. Ya sea en el ámbito personal como en el profesional, las personas comparten sus avances con el resto de la comunidad, lo que permite tener una referencia continua para la realización de distintos proyectos.

1.3 Motivación del PFC

El proyecto atiende a la motivación de Libelium de realizar un artículo para los desarrolladores demostrando la posibilidad de integración de distintas placas comercializadas por la empresa.

El área de aplicación principal es la medicina deportiva. Gracias a la conexión inalámbrica, las constantes vitales de los deportistas pueden ser monitorizadas en el terreno y ser enviadas a un punto receptor designado en tiempo real. De esta forma, entrenadores y médicos pueden evaluar de mejor manera las respuestas de los deportistas.

Dado su carácter experimental la plataforma ha sido especialmente pensada para su uso en deportes de montaña en donde el usuario puede portar el dispositivo en la mochila y mientras se realiza la actividad se pueden ir midiendo las diferentes constantes.

1.4 Objetivos del PFC

El objetivo del proyecto es integrar y adaptar el software de la placa **e-Health**, en un dispositivo tipo **Arduino** junto con más placas diseñadas para esta plataforma y construir un **dispositivo sensorial biométrico autónomo con monitorización remota en tiempo real**. Para lograrlo se plantearon los siguientes objetivos:

- En primer lugar, el estudio del ámbito del proyecto, las tecnologías Open Source y la monitorización médica.
- Selección de los sensores entre los disponibles para su aplicación.
- Estudio de las plataformas de comunicación a implementar para el envío de información, red 3G/GPS y la plataforma LoRa.
- Estudio de los elementos hardware restantes para su funcionalidad en el montaje final.
- El diseño de la placa de adaptación para todos los módulos usados.
- Finalmente, la realización del firmware o código de programación necesario para el control de la electrónica y la interpretación de los datos de acuerdo a las especificaciones.
- Pruebas para la comprobación del correcto funcionamiento del sistema.

2 Campo de aplicación

2.1 Posibles aplicaciones del sistema

La aplicación del sistema está orientada a la monitorización de las constantes biométricas en **medicina deportiva**, realizar el seguimiento de uno o varios deportistas a través de mediciones de sus parámetros biométricos de interés deportivo, tales como la temperatura corporal, pulso cardiaco, etc.

Además también está pensado como un elemento que se podría usar como un aparato médico en casos de accidentes en lugares remotos para monitorizar al paciente hasta su evacuación.

Finalmente cabe destacar que al ser un sistema prototipo y de carácter experimental se dejan abiertas sus posibles aplicaciones al usuario final ya que puede acceder a toda la información del hardware y software generado así como también servir de base para futuros proyecto más específicos en sus uso.

2.2 Estado del arte

Existen varios proyectos y productos comerciales con la finalidad de recoger datos biométricos y ser enviados de forma inalámbrica para posteriormente ser tratados. En concreto los parámetros biométricos más monitorizados son:

- Actividad eléctrica del corazón en forma de electrocardiograma (ECG).
- Pulsioximetría, saturación de oxigeno en sangre y pulsaciones por minuto.
- Presión arterial.
- Respiración y picos de flujo respiratorio.
- Temperatura corporal.
- Actividad muscular mediante sensores de electromiograma (EMG).
- Sudoración (GSR).
- Diferentes sensores como el giroscopio o el acelerómetro, relacionados con la posición del individuo, movimiento de las extremidades corporales, etc.
- Parámetros ambientales que pueden afectar en el estado del individuo, véase: Temperatura ambiental, altura, presión atmosférica, nivel de oxígeno ambiental...

Sistemas comerciales:

Entre los sistemas comerciales se encuentran los productos enfocados al deporte pero que generalmente se reducen a la medida de uno o dos datos biométricos como son la frecuencia cardiaca y la temperatura corporal. Muchos de ellos también miden parámetros ambientales y los más caros incluyen GPS para registrar la actividad pero no disponen de comunicación inalámbrica con otros dispositivos.

Hay muchas marcas comerciales, una de las mas grandes y conocidas es **POLAR**.

Fig. 4: Ejemplo producto POLAR

En el ámbito de la medicina existen aparatos profesionales capaces de medir la gran mayoría de parámetros biométricos indicados pero estos sistemas requieren su uso por parte de personal cualificado debido a su complejidad a parte su elevado coste solamente accesible para hospitales y equipos de investigación alejándolo de los usuarios independientes que quieran investigar de forma autónoma. Además estos aparatos carecen del componente de portabilidad debido a su tamaño y peso aparte de que normalmente requieren el ser conectados a la corriente eléctrica para su funcionamiento.

Un ejemplo de este tipo de sistemas son los comercializados por la compañía alemana **Medlab** que vende aparatos profesionales para la medicina.

Fig. 5: VITRO de Medlab

Proyectos y sistemas experimentales:

Entre los proyectos observados, uno muy interesante es el **codeBlue Project** de la Universidad de Harvard. En el cual han sido realizados tres módulos diferentes para ser utilizados con los motes TelosB y Mica2. El primer módulo incluye un sensor de pulsioximetría, el segundo integra un electrocardiograma de dos derivaciones y el tercer módulo introduce un acelerómetro, un giroscopio y electromiograma.

El proyecto **SUP** (Seguridad Urgencias Pirineos), realizado por el Instituto Tecnológico de Aragón en colaboración con médicos del SALUD. Trata de ser una UVI (Unidad de Vigilancia Intensiva) portátil que se coloca en el pecho del paciente. Ésta monitoriza las constantes vitales más importantes (frecuencia cardiaca, capacidad pulmonar, saturación de oxígeno, flujo sanguíneo y temperatura).

Con respecto al análisis de la sudoración de la piel, el proyecto **HandWave** refleja un diseño avanzado de este tipo de medición. Se trata de un dispositivo de tamaño reducido y fácilmente transportable, que es capaz de medir la conductividad de la piel y transmitir los datos inalámbricamente a través de una conexión Bluetooth. Está basado en un microcontrolador que regula la circuitería de amplificación del sistema y puede variar el factor de ganancia para conseguir mejor resolución.

En el proyecto **Vital Signs Monitoring System**, se estudia la monitorización de pacientes con el objetivo de conseguir una atención lo más temprana posible. Los parámetros que incluyen en su seguimiento son la temperatura corporal, el pulso cardíaco y la saturación de oxígeno en sangre. Estos datos son enviados a un centro de monitorización de forma inalámbrica.

3 Especificaciones del sistema

3.1 Tecnologías de partida

3.1.1 Arduino

Arduino es una plataforma de hardware abierto basada en la arquitectura de los micro-controladores AVR de la empresa Atmel. Se trata de una sencilla placa con entradas y salidas (E/S) analógicas y digitales que incluye un entorno de desarrollo para implementar el código en lenguaje C/C++.

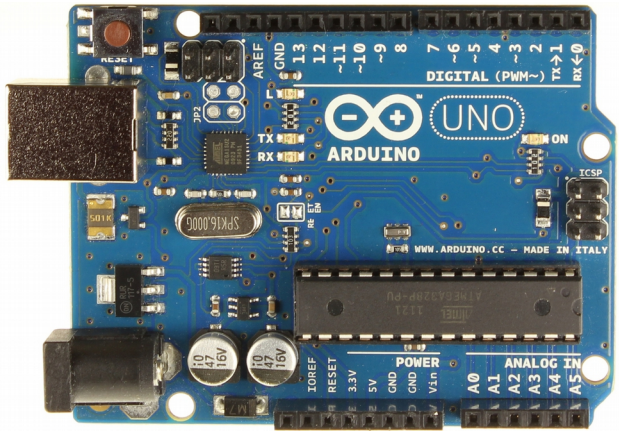

Fig. 6: Placa Arduino UNO

Nació como un proyecto para acercar la electrónica y la programación a la gente ya hacerlas más sencillas de aprender. Su propósito fue es de ser una placa de desarrollo de fácil programación a través de un lenguaje de muy alto nivel, en comparación con otras plataformas como PIC, que resultaba muy complicada para los iniciados y nuevos estudiantes. Pero Arduino ha transcendido más allá y hoy en día y no sólo se usa para prototipado, sino que incluso es una placa sobre la que desarrollan pequeños proyectos para empresas y entidades. Cabe destacar que hay muchos módulos pensados para pinchar directamente sobre Arduino, como módulos de comunicaciones (ZigBee, Bluetooth, 3G/GPRS), módulo para control de motores y robots, lectura de tarjetas RFID y muchos más, que han hecho que Arduino se haya ganado un hueco muy importante en el mundo de la electrónica.

VENTAJAS DE ARDUINO

El mundo de los micro-controladores y plataformas de desarrollo aunque es muy variado y tiene muchos fabricantes, es poco accesible a personas con pocos conocimientos de electrónica y programación. Arduino ha conseguido en apenas unos años, convertirse en la placa preferida para el desarrollo de prototipos y proyectos, tanto para aficionados como para profesionales del sector.

Las ventajas que presenta Arduino son las siguientes:

• Precio asequible: En comparación con otras placas, el precio es bastante bajo. Tenemos placas de Arduino por precios muy bajos. La nueva placa Arduino Leonardo, tiene un precio de unos 15€. La placa Arduino UNO, tiene un precio de unos 25€. Versiones más potentes de la placa como el Arduino Mega, tiene un precio de unos 55€.

• Multiplataforma: El IDE de Arduino funciona en los sistemas operativos más usados en el mercado, que son Windows, Linux y Macintosh, mientras que otros IDE's sólo están disponibles para Windows.

• Entorno de programación simple: a diferencia de sus competidores, el entorno de Arduino, está simplificado al máximo, y es muy intuitivo para aquel que lo utiliza por primera vez. Con pulsar un simple botón el código se carga en la placa y ya tenemos nuestro código funcionando dentro del micro-controlador.

• Funciones de alto nivel: crear un programa en Arduino es muy simple. Se basa en la sintaxis del lenguajes C, y dispone de un compilador para este lenguaje. Además, dispone de muchas funciones ya integradas en bibliotecas, así como muchos códigos de ejemplo para poder aprovechar código ya hecho por otros usuarios. No es necesario conocer los registros del micro ni programar en el complicado lenguaje ensamblador, aunque dispone de esta opción para usuarios más avanzados.

• Software de código abierto y fácilmente ampliable : El software libre, tiene una trayectoria mucho más antigua que el hardware libre, y ha demostrado ser una forma de trabajo y desarrollo que ha dado productos muy buenos, y gratuitos, con la colaboración de personas de todo el mundo que hacen su pequeña aportación o mejora. Arduino aprovecha esto y además hace que la integración de nuevas librerías sea tan sencillo como descargar un archivo y pegarlo en un directorio.

• Hardware ampliable y diseño abierto : Basado en los micro-controladores ATmega168, ATmega328, ATmega128 y ATmega2560 el diseño de la placa está bajo licencia Creative Commons, y esto le da al diseño la capacidad de mejorar con aportaciones y sugerencias constantes.

3.1.2 e-Health

La plataforma e-Health diseñada por Cooking Hacks de Libelium es una plataforma pensada para funcionar con Arduino y Raspberry Pi capaz de medir diferentes valores biométricos de las personas a través de los distintos sensores. Esta información puede ser usada para la monitorización en tiempo real del individuo o recoger la información para ser analizada en un posterior diagnóstico médico.

El módulo e-Health ha sido creado para ayudar a investigadores y desarrolladores a medir diferentes constantes biométricas con fines de experimentación, estudios o simple diversión. Cooking Hacks ofrece una alternativa abierta y barata comparado con el precio prohibitivo de las soluciones del mercado médico profesional. Sin embargo esta plataforma no tiene certificación médica por lo que no puede ser usada para monitorizar a pacientes críticos que necesiten disponer de datos muy precisos.

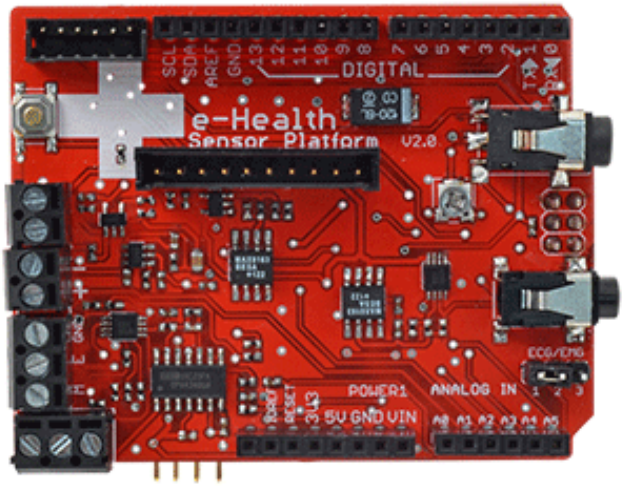

Fig. 7: Placa e-Health

e-Health dispone de la electrónica necesaria para adaptar las señales de los distintos sensores a las plataformas Arduino y Raspberry Pi.

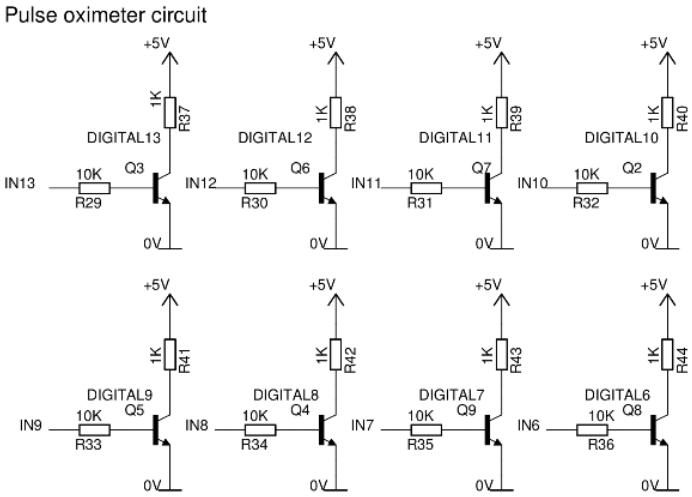

Fig. 8: Ejemplo electrónica de adaptación SPO2

Los sensores disponibles en la plataforma e-Health son los siguientes:

- Flujo de aire (frecuencia respiratoria)
- SPO2 (pulsioximetría)
- GSR (conductividad de la piel)
- Acelerómetro (posición del individuo)
- ECG (electrocardiograma)
- EMG (electromiografía)
- Tensionómetro
- Glucómetro

3.2 Parámetros de interés para la monitorización

Dado que el sistema está pensado para que pueda ser fácilmente transportable se escogieron los parámetros biométricos cuyos sensores se puedan colocar de forma sencilla por cualquier persona y además que pudiesen medir sus valores de forma simultánea sin realizar cambios en el montaje ni en el software.

Con estas premisas se estableció el objetivo de monitorizar, mediante software, con los sensores disponibles los siguientes parámetros biométricos:

1. Temperatura de la piel
2. Frecuencia respiratoria
3. Pulsioximetría
4. Conductividad de la piel
5. Posición del individuo

3.3 Otras especificaciones para el diseño final

Teniendo en cuenta las posibles aplicaciones del sistema se decidió que el diseño final debía cumplir las siguientes especificaciones.

- Portátil (uso de batería)
- Ligero
- Display de visualización
- Comunicación a un centro remoto por dos sistemas distintos inalámbricos
- Hardware sencillo de replicar por otros usuarios
- Funciones del software de alto nivel para poder ser usadas por otros usuarios de forma sencilla
- Bajo coste

4 Hardware

4.1 Selección del Hardware

Dado el carácter prototipo del sistema en el proceso de selección de hardware no se hizo un estudio de mercado para encontrar la mejor opción y más barata para cumplir con funcionalidades especificadas. A parte del hardware de inicio del que ya se ha hablado (Arduino y e-Health), el resto de módulos utilizados han sido productos de Libelium como las placas de comunicación 3G+GPS y LoRa o bien productos que llegaron de muestra de otras empresas como la pantalla TFT LCD o el Solar Charger Shield.

Así mismo el resto de accesorios como pueden ser las antenas o el panel solar son productos comercializados por la página de Cooking Hacks y que son de fácil acceso para el resto de usuarios.

4.2 Diagrama de bloques

Todo el hardware que compone el sistema conforma el **Dispositivo sensorial biométrico autónomo**, mediante el siguiente diagrama de bloques se puede ver de forma conceptual la comunicación de los distintos módulos respecto al microcontrolador del sistema.

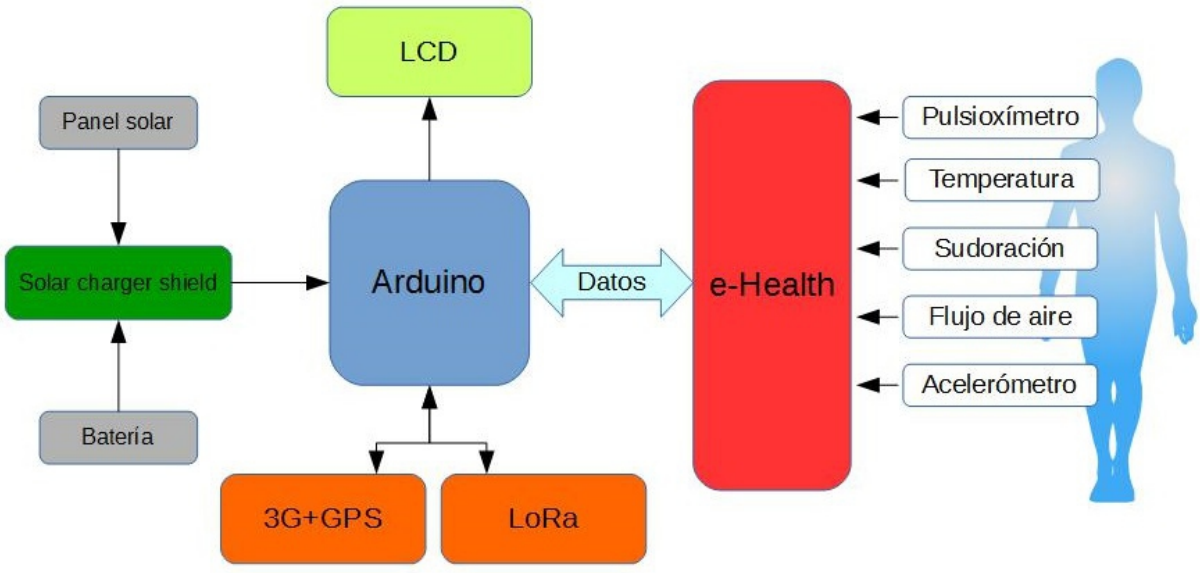

Fig. 9: Diagrama de bloques general

4.2.1 Comunicación entre módulos

Se han usado tres tipos de comunicación distintas entre Arduino y el resto de módulos para la transmisión de información en función de los distintos elementos seleccionados. A continuación se explicarán las características principales de cada uno.

SPI (Serial Peripheral Interface)

El **Bus SPI** es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus **SPI** es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj.

Incluye una línea de reloj dato entrante, dato saliente y un pin de chip select, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.

El **SPI** es un protocolo síncrono. La sincronización y la transmisión de datos se realiza por medio de 4 señales:

- **SCLK** *(Clock)*: Es el pulso que marca la sincronización. Con cada pulso de este reloj, se lee o se envía un bit.
- **MOSI** *(Master Output Slave Input)*: Salida de datos del Master y entrada de datos al Slave.
- **MISO** *(Master Input Slave Output)*: Salida de datos del Slave y entrada al Master.
- **SS/Select**: Para seleccionar un Slave, o para que el Master le diga al Slave que se active.

Tanto la pantalla TFT LCD como el módulo de comunicación LoRa se comunican con Arduino por **SPI**.

UART (Universal Asynchronous Receiver-Transmitter)

La **UART** controla los puertos y dispositivos serie. Se encuentra integrado en la placa base o en la tarjeta adaptadora del dispositivo. Las funciones principales de chip **UART** son de manejar las interrupciones de los dispositivos conectados al puerto serie y de convertir los datos en formato paralelo, transmitidos al bus de sistema, a datos en formato serie, para que puedan ser

transmitidos a través de los puertos y viceversa.

Son dos las lineas necesarias para la comunicación, Tx (transmisión) y Rx (recepción).

El módulo 3G+GPS es el único módulo que se comunica por **UART**.

I²C (Inter-Integrated Circuit)

I²C es un bus de comunicación en serie. La principal característica de **I²C** es que utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj. También es necesaria una tercera línea, pero esta sólo es la referencia (masa). Como suelen comunicarse circuitos en una misma placa que comparten una misma masa esta tercera línea no suele ser necesaria.

Las líneas se llaman:

- SDA: datos
- SCL: reloj
- GND: tierra

El único dispositivo que se comunica por **I²C** es el sensor de **posición** que va conectado a la placa de e-Health.

4.3 Arduino MEGA 2560

Inicialmente se comenzó usando la placa Arduino UNO como base para el montaje y también para la fase de pruebas de funcionamiento y aprendizaje del uso de los distintos módulos.

Durante esta fase de aprendizaje se detectaron las carencias del Arduino UNO debido entre otros a la falta de pines de entrada/salida necesarios así como la falta de los puertos de comunicación necesarios para la comunicación con todos los módulos. De este modo se decidió cambiar al Arduino MEGA.

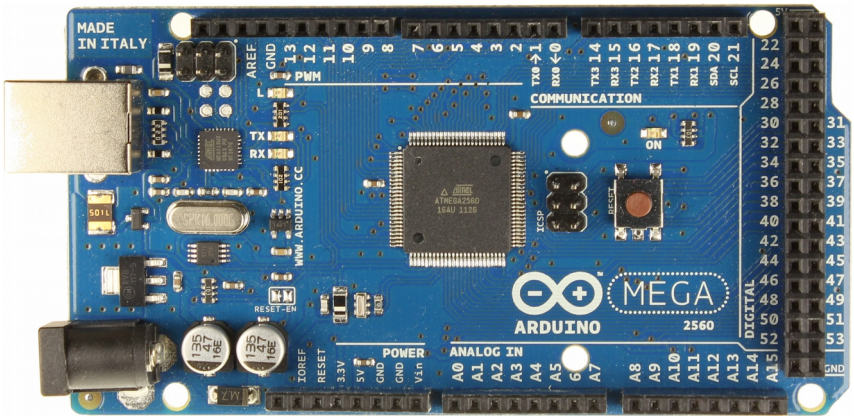

Fig. 10: Placa Arduino MEGA

El Arduino MEGA es la versión extendida del Arduino UNO y esta basado en el microcontrolador **ATMega2560** lo que ofrece más capacidad para realizar todas las funciones requeridas además de disponer de más memoria para flash para poder programar códigos más complejos.

Las características generales del Arduino MEGA 2560 son las siguientes:

Arduino MEGA 2560	
Micro-controlador	ATmega2560
Tensión de operación	5 V
Voltaje de entrada (recomendado)	7-12 V
Voltaje de entrada (límites)	6-20 V
Pines I/O digitales	54 (15 suministran salida PWM)
Pines analógicos	16
Memoria flash	256 KB de los cuales 8 KB usados por el bootloader
SRAM	8 KB
EEPROM	4 KB
Reloj	RTC (16 MHz)
Dimensiones	68.6 mm x 53.4 mm
Peso	25 g

Tabla 1: Características Arduino MEGA

4.4 TFT LCD 2.2”

Para la visualización de la información de los sensores por parte del usuario se ha optado por la pantalla TFT LCD a color de 2.2 pulgadas de diagonal y con una resolución de 240x320 pixels producida por **Elecfreaks**. Funciona mediante el controlador **ILI9341** y su comunicación con Arduino es a través de SPI. Además dispone de un lector de tarjetas SD para poder mostrar por pantalla imágenes.

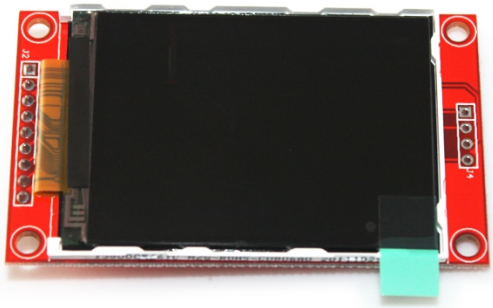

Fig. 11: Display TFT LCD de Elecfreaks

Los niveles de tensión de funcionamiento de la pantalla son a 3.3 voltios y los niveles de las entradas y salidas de Arduino MEGA son a 5 voltios por lo que ha sido necesaria la adaptación de los voltajes con el integrado **CD4050BC**. Este circuito integrado esta formado por 6 buffers de amplificación para poder hacer la transformación en la impedancia de la señal.

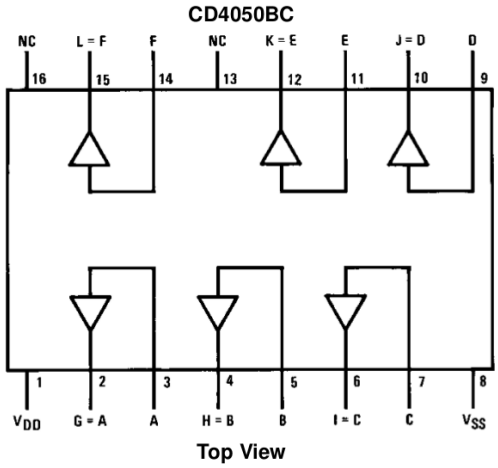

Fig. 12: Circuito integrado CD4050BC

4.5 SX1272 LoRa + Multiprotocol

Como módulo principal de comunicación se ha utilizado el nuevo **SX1272 LoRa** de Libelium. Dado su gran rango de alcance será ideal para poder enviar los datos recogidos por los sensores a un punto central que recogerá la información.

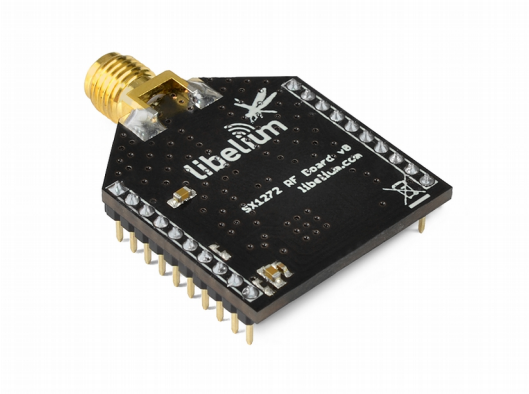

Fig. 13: Módulo SX1272

LoRa es una nueva técnica de modulación de espectro ensanchado que permite transmitir información con una baja tasa de datos pero a una gran distancia. El módulo trabaja en las bandas **ISM** 868 y 900 MHz, al ser frecuencias mucho menores que la popular banda de 2.4 GHz se consiguen mayor penetración en materiales (muros, árboles) y se reducen las posibles interferencias durante la transmisión.

LoRa	
Modulo	SX1272
Banda de frecuencia dual	863-870 MHz (Europa)
	902-928 MHz (USA)
Potencia de transmisión	14 dBm
Sensibilidad	-134 dBm
Canales	8 (868 MHz)
	13 (915 MHz)
Rango	LOS = 21 km
	NLOS = +2 km

Tabla 2: Características LoRa

Para comunicar el módulo con Arduino se utiliza la placa **Multiprotocol** también de Libelium. Esta placa permite conectar distintos módulos de comunicación inalámbrica mediante los dos socket que dispone, uno por el bus SPI y el otro por UART.

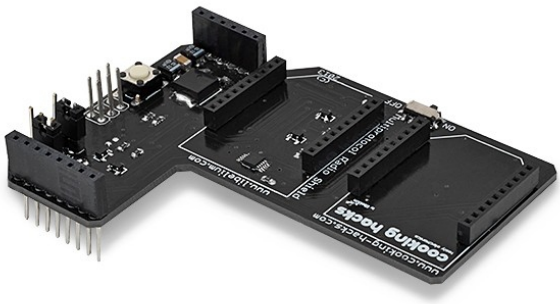

Fig. 14: Placa Multiprotocol

4.6 3G + GPS

El módulo utilizado como segunda fuente de comunicación inalámbrica en caso de fallar la principal es la placa **3G + GPS** de Libelium.

Su conectividad **3G/GPRS** permite al usuario poder realizar distintas tareas como cargar o descargar contenido de un servidor web con la navegación HTTP o también trabajar con el protocolo FTP para subir archivos a un servidor.

El módulo usado por esta placa es el **SIM5218** de SIMCom que además también dispone de **GPS** lo que nos servirá para poner realizar la geolocalización del individuo cuando requiramos.

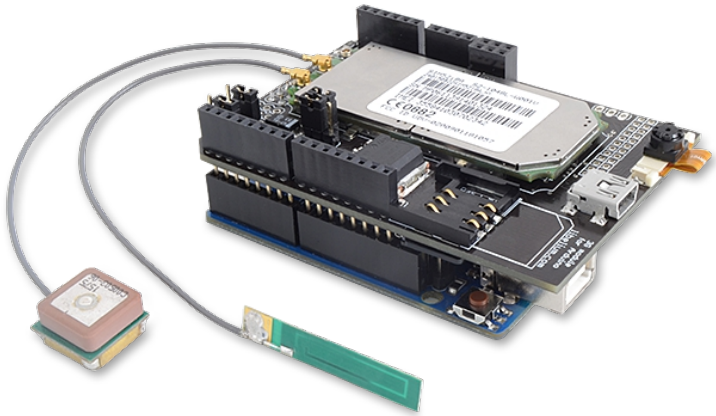

Fig. 15: Módulo 3G+GPS

4.7 Solar Charger Shield v2

Para asegurar la autonomía del sistema se ha utilizado el módulo **Solar Charger Shield v2** de la empresa Seeed Studio. Este módulo pensado para la plataforma Arduino permite conectar una batería recargable, que además mediante otro conector se puede ir recargando paulatinamente con un panel solar, lo que nos permite crear sistemas completamente autónomos.

Fig. 16: Solar Charger Shield v2

El módulo acepta baterías de entre 2.7 y 4.7 V elevando la tensión hasta los 5 Voltios con los que trabaja Arduino, además es capaz de suministrar una corriente máxima de hasta 700 mA. También dispone de protección a cortocirtuitos y un conector USB que permite cargar la batería.

4.7.1 Batería Li-Ion

La batería seleccionada para alimentar el sistema es una batería de iones de litio también comercializada por **Seed Studio** y que está recomendada para su uso con el Solar Chaguer Shield v2. Su poco peso y reducido tamaño la hacen perfecta para el sistema, ademas está compuesta por dos celdas de 2200 mAh lo que será una capacidad suficiente para alimentar el montaje durante horas.

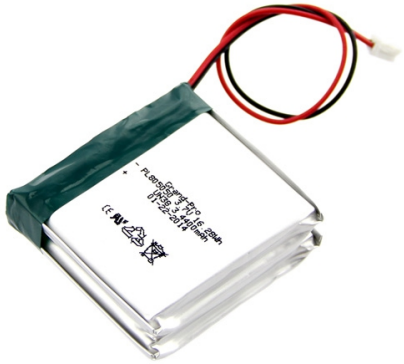

Fig. 17: Batería 4400 mAh

Batería 4400 mAh	
Tensión de carga	4.2 V
Voltaje nominal	3.7 V
Capacidad nominal	4400 mAh
Máxima corriente de carga	2.2 A
Máxima corriente de descarga	4.4 A
Peso de una celda	83 g
Dimensiones	53 mm x 51 mm x 17 mm

Tabla 3: Características batería 4400 mAh

4.7.2 Panel solar flexible

Dado que el sistema esta pensado para ser usado en el exterior se ha optado por el panel solar flexible disponible en la web de Cooking Hacks. Este panel será ideal para el montaje al ser muy ligero y resistente y nos permitirá alargar la autonomía del sistema.

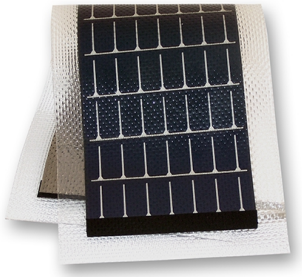

Fig. 18: Panel solar flexible

Panel solar flexible	
Dimensiones	180 mm x 280 mm
Voltaje típico	5.5 V
Corriente típica	100 mAh
Máximo voltaje en carga	6.4 V

Tabla 4: Características panel solar flexible

4.8 Sensores

A continuación se explicará brevemente el funcionamiento de los 5 sensores seleccionados, de entre los disponibles en la plataforma e-Health, que se van a utilizar.

El sensor de **flujo de aire** es básicamente dos termopares en serie para detectar los cambios de temperatura en las exhalaciones permite conocer la frecuencia respiratoria del portador.

Fig. 19: Sensor de flujo de aire

La conductividad de la piel, también conocida como **GSR**, es un método para medir la conductividad eléctrica de la piel en función de su humedad por lo que nos permite saber los niveles de sudoración que sufre el individuo. El sensor es esencialmente un ohmetro que mide la resistencia entre dos puntos. A mayor humedad en la piel la resistencia eléctrica baja y una piel seca supone mayor resistencia.

Fig. 20: Sensor de sudoración de la piel

El sensor de posición se basa en un **acelerómetro** y nos permite saber la posición en la que se encuentra el portador de entre 5 diferentes, de pie, tumbado hacia arriba, tumbado hacia abajo, tumbado de lateral derecho y tumbado de lateral izquierdo.

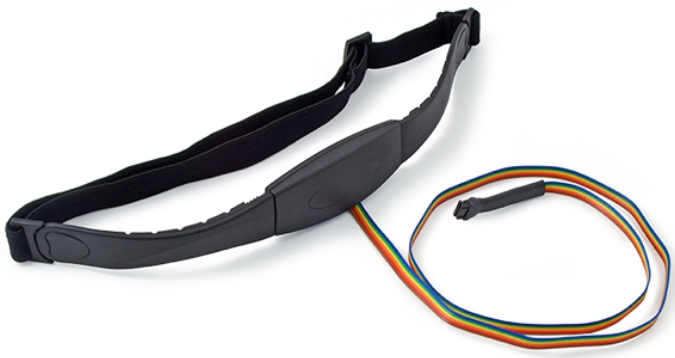

Fig. 21: Sensor de posición

El **pulsioxímetro** permite saber el porcentaje de saturación de oxígeno en la sangre y las pulsaciones, parámetros muy importantes en la medicina deportiva. La lectura de estos valores se realiza mediante técnicas fotoeléctricas.

Fig. 22: Sensor pulsioximetría

Este sensor se trata de un termistor tipo NTC que mide la **temperatura corporal** que es un parámetro básico en la medicina. La razón es que multitud de enfermedades vienen acompañadas por cambios de temperatura lo que nos puede ayudar a prevenir problemas antes de tiempo.

Fig. 23: Sensor de temperatura

4.9 e-Health Portable Board

Para aglutinar todo los elementos que compondrán el montaje final se ha creado una placa donde conectarán los distintos módulos y en la cual también se podrán extraer con facilidad si alguno de ellos falla. Para decidir las lineas necesarias a conectar entre los módulos y Arduino se realizaron múltiples pruebas cruzadas entre ellos con códigos sencillos de ejemplo a la vez que sirvió para su estudio y comprobar su correcto funcionamiento.

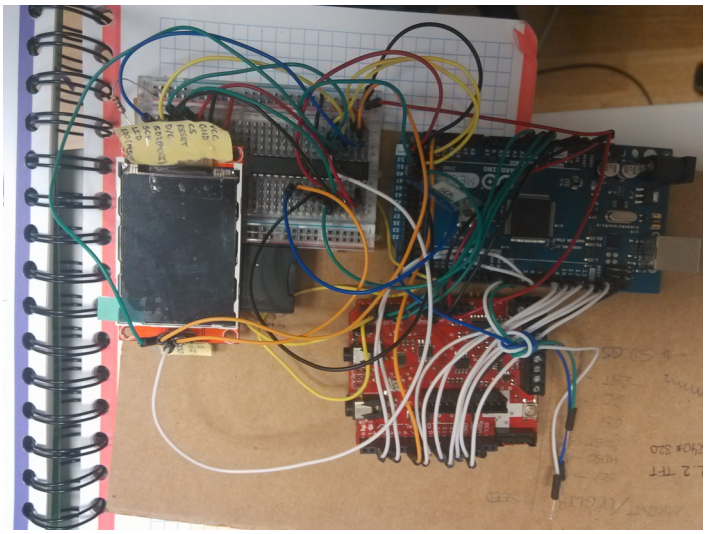

Fig. 24: Ejemplo pruebas previas

En la siguiente tabla se hace un resumen de las conexiones realizadas entre los módulos y Arduino.

Conexiones entre módulos				
Módulo	Pines módulo	Pines Arduino	Alimentación	Comentarios
e-Health	Airflow → A1 GSR → A2 Body temp. → A3 SPO2 → D6-D13 Acelerómetro → A4, A5, D3	Airflow → A1 GSR → A2 Body temp. → A3 SPO2 → D7-D13, D15 Acelerómetro → A4, A5, D3	5V, 3V3, GND	Acelerómetro a las lineas I²C de Arduino
Multiprotocol	MOSI, MISO, SCK, RX, TX, T1, T2, T3	D50-D52, RX1, TX1, D36, D38, D40	5V, GND	El conexionado de la UART puede permitir incluir otro módulo de comunicación

Módulo	Pines módulo	Pines Arduino	Alimentación	Comentarios
3G + GPS	RX, TX, D2	RX0, TX0, D22	5V, GND	
TFT LCD	CS_TFT, RESET, DC, MOSI, SCK, LED, MISO, CS_SD	D47-D52	5V, GND	Lineas SPI a través del integrado CD4050, pin LED a 3V3
Solar Charger Shield	5V, GND	5V, GND	Batería	

Tabla 5: Resumen conexiones

La placa se ha diseñado con el software **EAGLE 6.1.0.** y una vez finalizada se encargo su fabricación a la empresa profesional **Micron-20.**

Los componentes básicos para la placa han sido las tiras de pines THD, tanto hembra como macho, con un paso de 0.1 pulgadas (2.54 mm) que es el estándar en todos los módulos para Arduino. Esto nos permitirá conectar cada módulo a la placa y que queden fijados de forma segura pero con la posibilidad de extraerlos cuando queramos.

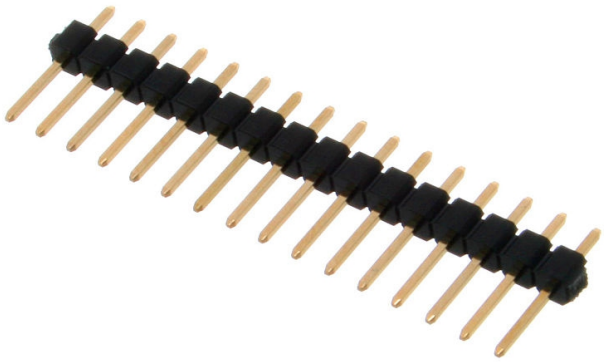

Fig. 25: Tira de pines macho

Además también será necesario el integrado CD4050BC para poder controlar el diplay y una resistencia de 10 Ohmios que regula el brillo de la pantalla. Todos los elementos han sido comprados al distribuidor internacional de componentes electrónicos **Farnell**.

4.9.1 Diagrama de circuito esquemático

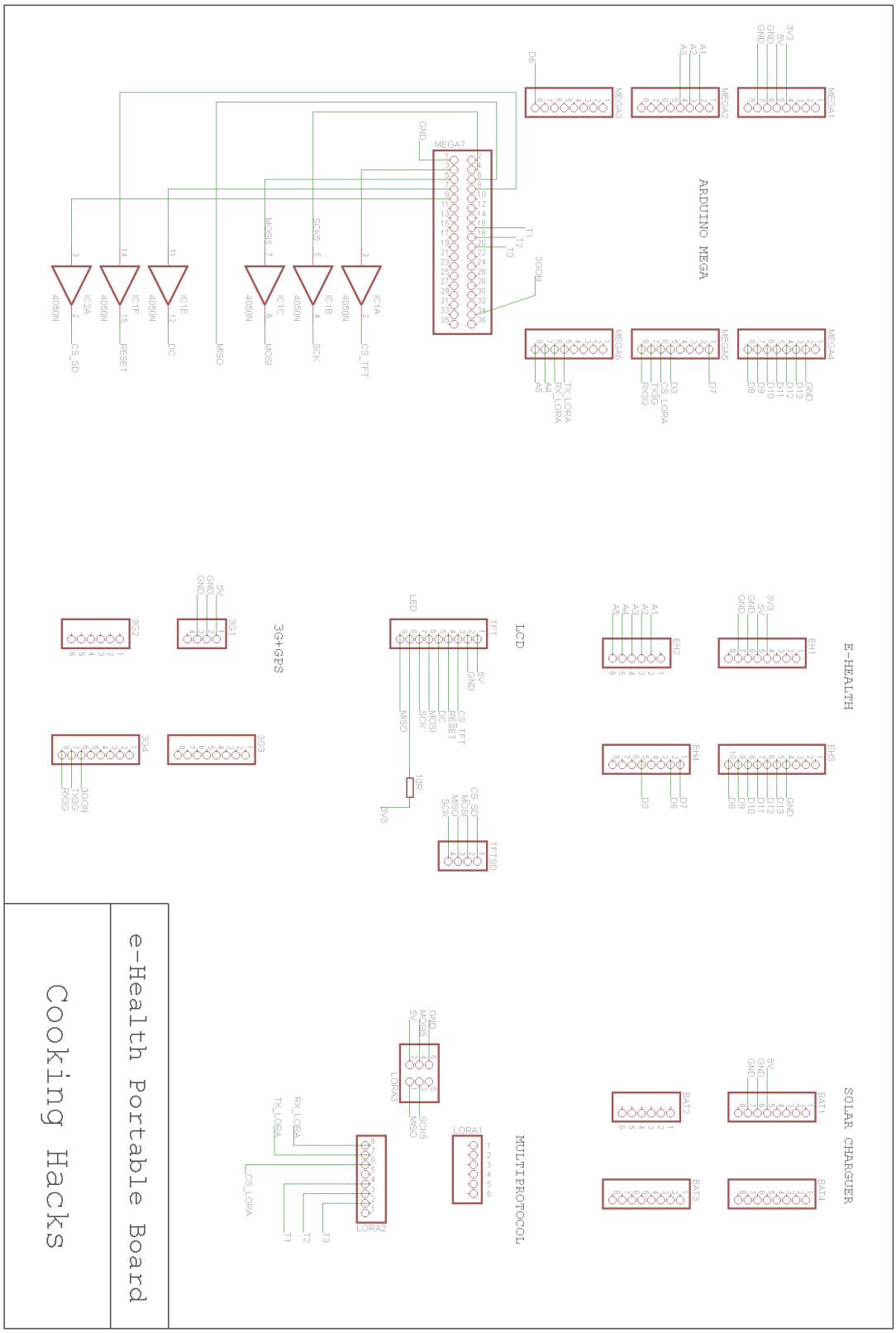

Fig. 26: Circuito esquemático

4.9.2 Placa de circuito impreso

A la hora de realizar la disposición espacial de los componentes hay que tener en cuenta los conectores de cada placa a los que tendremos que tener acceso una vez esté en el encapsulado final y además intentar que el tamaño de la placa sea el más reducido posible.

Después de algunas pruebas se eligió la siguiente disposición de los módulos:

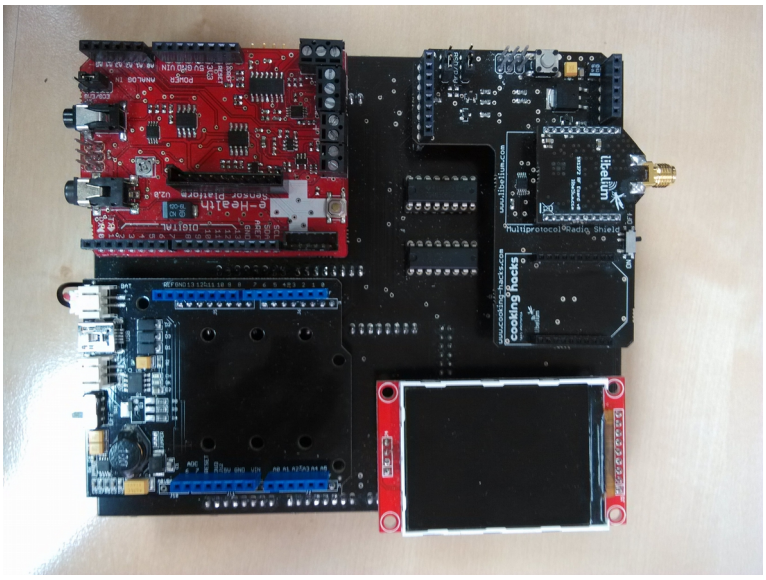

Fig. 27: Módulos cara TOP

Fig. 28: Módulos cara BOTTOM

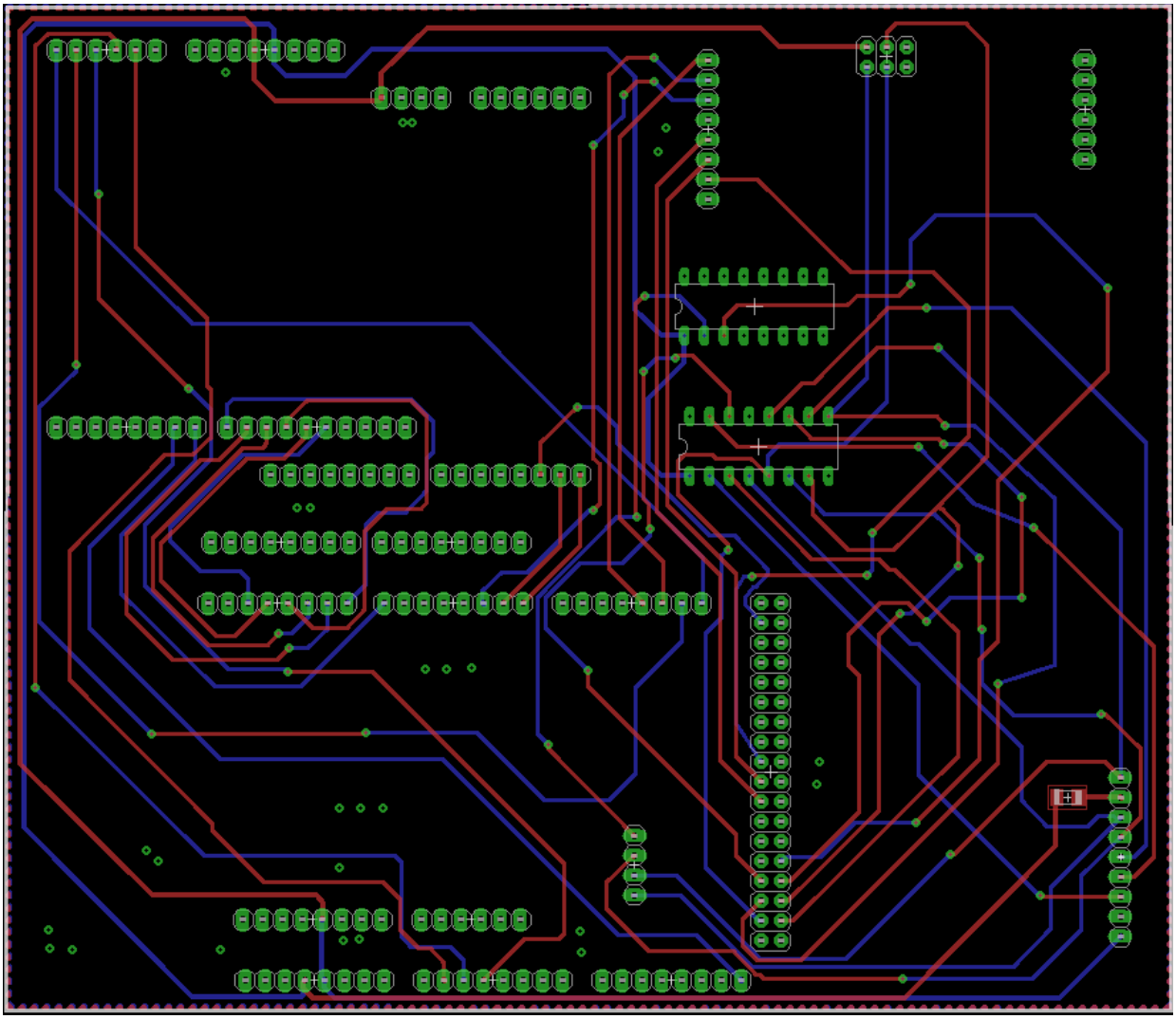

Fig. 29: Ruteado PCB

5 Software

5.1 Introducción

En este aparatado se explica el funcionamiento de las distintas partes que componen el código principal mediante diagramas de flujo o tablas explicando las funciones utilizadas más importantes. También se cuenta las distintas librerías usadas en el código así como las funciones de alto nivel creadas para facilitar la compresión del código y el uso a posibles usuarios. Igualmente se habla de las pequeñas modificaciones que han sido necesarias realizar en alguna de ellas para el correcto funcionamiento de todos los módulos.

Aunque se hayan utilizado librerías ya existentes ha sido necesario mucha programación en el código para dotarlo de inteligencia y que sea robusto.

5.2 Software de base: Arduino IDE

Para realizar cualquier tipo de programación con la tecnología de Arduino, es necesario instalar el IDE (Integrated Development Environment). Se puede encontrar en la página web www.arduino.cc y está disponible para descargar gratuitamente por cualquier usuario.

Es importante usar la última versión que se encuentra en la web. El entorno de programación incluye todas las librerías de API necesarias para compilar los programas. Una vez tenemos un programa cargado en el microcontrolador, el funcionamiento de Arduino se basa en el código cargado. La estructura de códigos se divide en dos partes fundamentales: setup y loop. Ambas partes del código tienen un comportamiento secuencial, ejecutándose las instrucciones en el orden establecido. El setup es la primera parte del código que se ejecuta, haciéndolo solo una vez al iniciar el código. En esta parte es recomendable incluir la inicialización de los módulos, alimentación, entre otros, que se vayan a utilizar. El loop es un bucle que se ejecuta continuamente, formando un bucle infinito.

En el entorno existen una serie de botones que sirven para un manejo rápido del código:

Verificar: Chequea el código en busca de errores.

Cargar: Compila el código y lo vuelca en la placa E/S de Arduino.

Nuevo: Crea un nuevo sketch.

Abrir: Presenta un menú de todos los programas sketch de su "sketchbook" (librería de sketch). Un clic sobre uno de ellos lo abrirá en la ventana actual.

Guardar: Salva el programa sketch.

Monitor Serial: Inicia la monitorización serie.

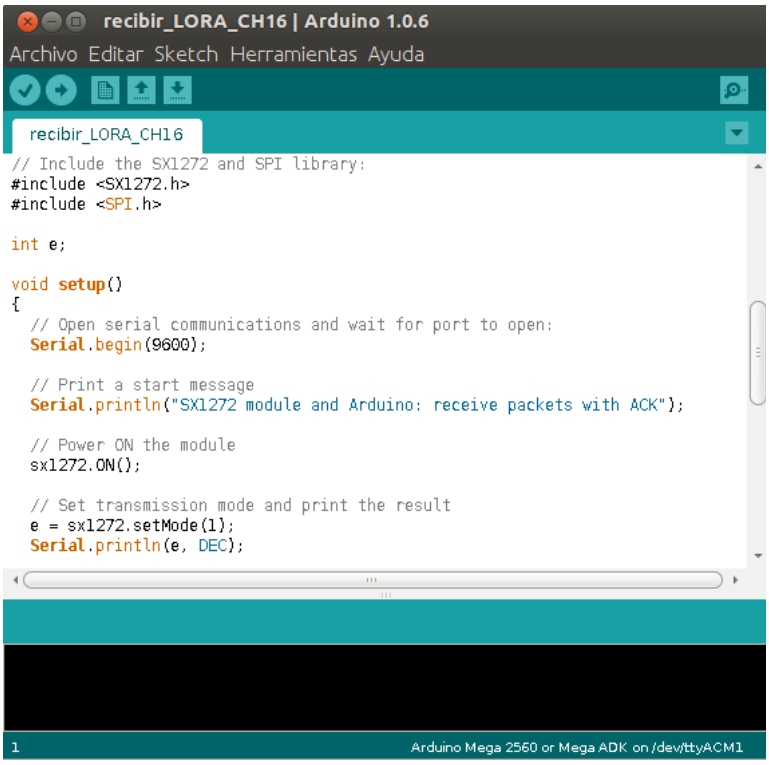

Fig. 30: IDE de programación de Arduino

5.3 Lenguaje de programación: C/C++

C es un lenguaje de programación orientado a la implementación de Sistemas Operativos. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones. Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

C++ es un lenguaje de programación diseñado con la intención de extender el exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos.

Las propiedades de este tipo de lenguajes son:

- Un núcleo del lenguaje simple, con funcionalidades añadidas importantes, como funciones matemáticas y de manejo de archivos, proporcionadas por bibliotecas.
- Es un lenguaje muy flexible que permite programar con múltiples estilos.
- Un sistema de tipos que impide operaciones sin sentido.
- Usa un lenguaje de pre-procesado.
- Acceso a memoria de bajo nivel mediante el uso de punteros.
- Interrupciones al procesador con uniones.
- Un conjunto reducido de palabras clave.
- Tipos de datos agregados (struct) que permiten que datos relacionados se combinen y se manipulen como un todo.

5.4 Programa principal

Para mostrar las capacidades del sistema se ha generado un código de ejemplo usando las distintas posibilidades que nos ofrecen los módulos integrados.

Al crear funciones de alto nivel se deja a decisión del usuario el poder modificar el código en función de la aplicación que vaya ha hacer del sistema.

El siguiente diagrama de flujo muestra el funcionamiento básico del programa completo de ejemplo:

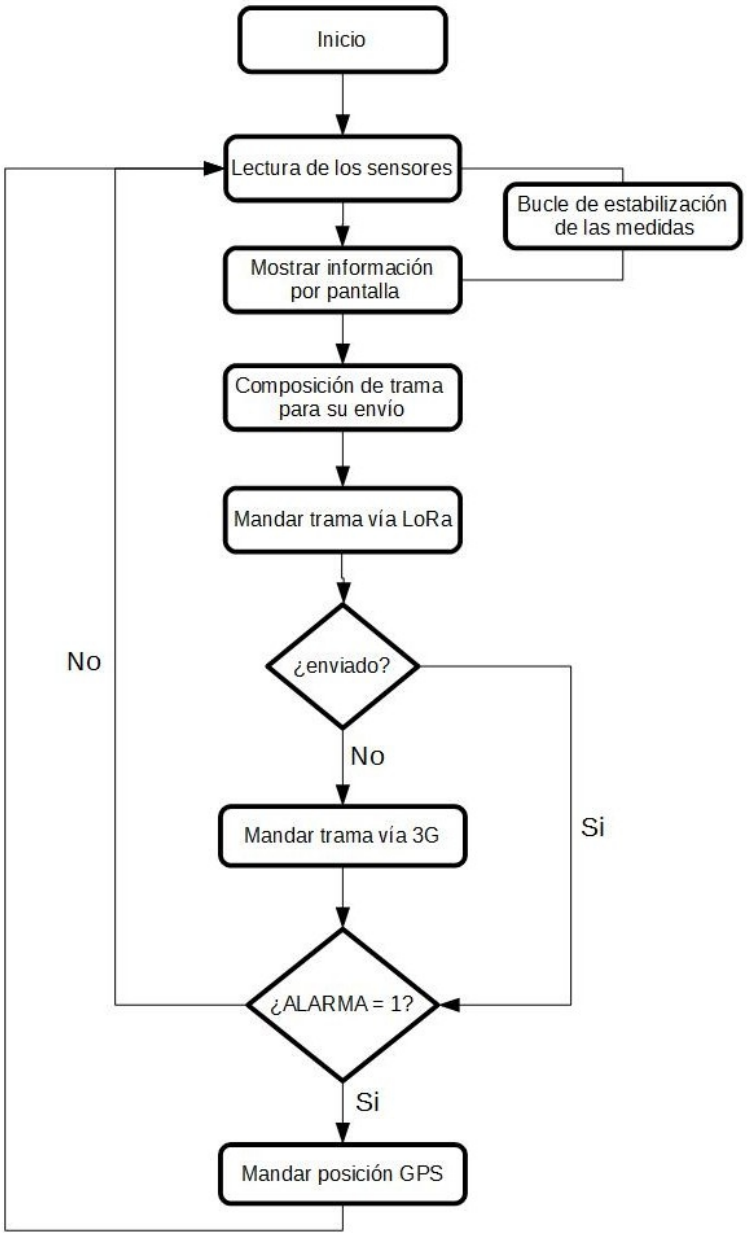

Fig. 31: Diagrama de flujo código de ejemplo

5.5 Software TFT LCD

El control del display de visualización se realiza utilizando las librerías **Adafruit_GFX** y **Adafruit_ILI9341** a parte de la librería de Arduino **SPI**.

Las funciones de las librerías más importantes que se han usado durante todo el software para el manejo del display la representación de la información son las siguientes:

Función	Descripción
`tft.begin()`	Inicializa el display
`tft.fillScreen(uint16_t color)`	Llena la pantalla de un color
`tft.setRotation(uint8_t x)`	Establece la orientación
`tft.setCursor(int16_t x, int16_t y)`	Establece las coordenadas donde se empezará a dibujar/escribir
`tft.setTextColor(uint16_t c)`	Selecciona el color del texto a escribir
`tft.setTextSize(uint8_t s)`	Selecciona el tamaño del texto a escribir
`tft.println()`	Escribe por pantalla con los parámetros definidos con anterioridad

Tabla 6: Funciones control TFT LCD

Además se han creado dos funciones de alto nivel para facilitar al usuario ciertas tareas.

- **`printLogo()`** → dibuja el logo inicial al encender el sistema
- **`setScreen()`** → prepara la pantalla para representar los datos de los sensores

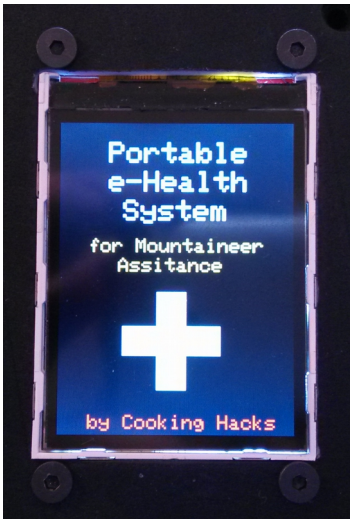

Fig. 32: pintLogo

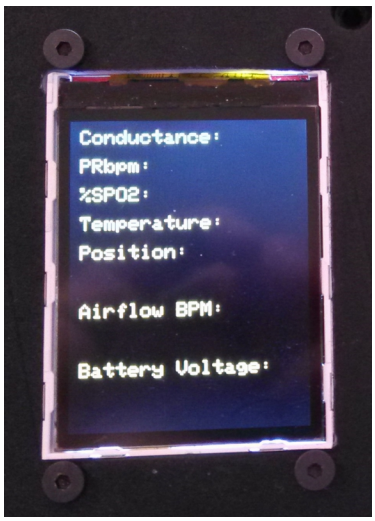

Fig. 33: setScreen

5.6 Software asociado a los sensores

Para programar el software relacionado con los sensores se han utilizado las librerías existentes del módulo e-Health. Estas librerías han sido diseñadas específicamente para el Arduino UNO por lo que ha sido necesario realizar pequeñas modificaciones tanto en el código como las librerías para que todos los sensores seleccionados funcionasen correctamente en el Arduino MEGA.

Para cada sensor se ha creado una función en las que se combinan funciones de las librerías de e-Health y de la pantalla TFT LCD.

Función	Descripción
`getShowSPO2()`	Lee y muestra por pantalla los valores del sensor de pulsioximetría
`getShowConductance()`	Lee y muestra por pantalla los valores del sensor de sudoración de la piel
`getShowTemp()`	Lee y muestra por pantalla los valores del sensor de temperatura
`getShowPosition()`	Lee y muestra por pantalla los valores del sensor de posición
`getShowAirflow()`	Lee y muestra por pantalla los valores del sensor de flujo de aire

Tabla 7: Funciones lectura y muestra sensores

Puesto que el refresco de la pantalla se observo que era muy lento, a la hora de actualizar los datos de los sensores por pantalla para optimizar el tiempo de espera se decidió usar la técnica de dibujar encima el mismo dato recogido en la lectura anterior.

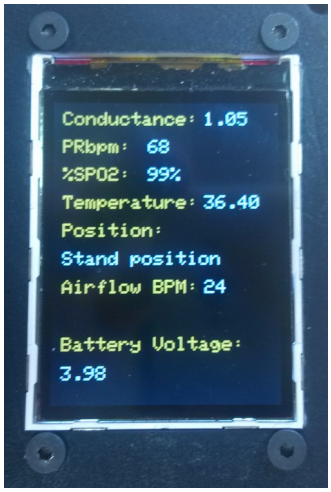

Fig. 34: Mostrando datos sensores

El funcionamiento genérico de estas funciones se explica con el siguiente diagrama de flujo:

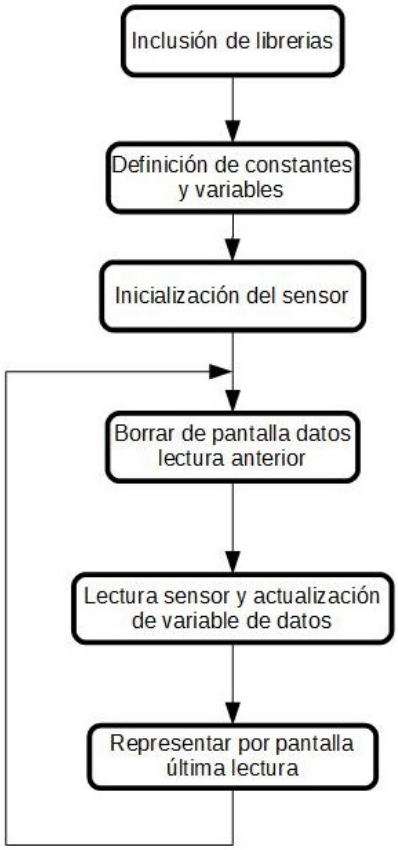

Fig. 35: Diagrama flujo sensores

En el caso del sensor de flujo de aire ha sido necesario crear una nueva función en las librerías de e-Health para que nos diese un valor numérico de las respiraciones y poder ser representado por pantalla. Esta función llamada `airflowBPM()` cuenta el número de respiraciones en 10 segundos para poder calcular las respiraciones totales en un minuto.

También ha sido necesario modificar en las librerías la función que recoge los datos del pulsioxímetro. Estas modificaciones se pueden consultar en los anexos.

5.7 Software creación de la trama para su envío

Dado que la información se va a enviar de forma inalámbrica se ha creado un tipo de trama en una cadena de caracteres ya que es compatible con los dos sistemas de envío, para ello se ha creado una función de alto nivel llamada **`compoundFrame()`** en la cual se cogen todas las variables donde se han guardado los valores de los sensores y se transforman si es necesario al tipo cadena de caracteres. Además se van numerando las tramas desde que se enciende el sistema para saber si alguna se ha perdido.

A la hora de decidir la forma de la trama se ha buscado el equilibrio entre legibilidad y tamaño ya que en las comunicaciones inalámbricas interesa que el peso de los paquetes a enviar sea reducido para conseguir que el tiempo que tarda en enviar sea el menor posible.

El formato de la trama creada es el siguiente:

#	N:	int	#	COND:	float	#	BPM:	int	#	SPO2:	int	#	TEMP:	float	#	AirBPM:	int	#	POS:	char	#	ALARM:	int

Tabla 8: Formato trama de envío

5.8 Software LoRa

El código del software que envía la trama a través del módulo **SX1272 LoRa** usando su librería especifica para Arduino, se divide en dos partes, configuración del módulo y envío de la trama. Para ello se han creado dos funciones de alto nivel, **`startLora()`** y **`sendLora()`**.

La primera de ellas configura todos los parámetros necesarios para el correcto envío de la trama, las funciones más importantes a la hora de configurar y controlar LoRa son las siguientes:

Función	Descripción
`sx1272.ON()`	Enciende el módulo
`sx1272.setMode(uint8_t mode)`	Establece modo de envío
`sx1272.setChannel(uint32_t ch)`	Selecciona canal
`sx1272.setPower(char p)`	Selecciona potencia de envío
`sx1272.setNodeAddress(uint8_t addr)`	Asigna una dirección en la red al módulo
`sx1272.setRetries(uint8_t ret)`	Establece número máximo de reintentos
`sx1272.OFF()`	Apaga el módulo

Tabla 9: Funciones configuración LoRa

Una vez encendido el módulo hay que establecer el modo de envío, en el cual se configuran 3 parámetros distintos, ancho de banda (BW), *coding rate* (CR) y *spreading factor* (SF).

- Ancho de banda → Establece el ancho de la señal de transmisión.
- Coding rate → Ratio de uso de la técnica de codificación de información a nivel de byte para hacerla mas resistente al ruido e interferencias.
- Spreading factor → Grado de uso de la técnica de ensanchamiento espectral para mayor resistencia a interferencias.

Estos parámetros también pueden ser configurados de forma unitaria.

Modo	BW	CR	SF	Sensibilidad (dB)	Tiempo de transmisión (ms) para un paquete de 100 bytes	Comentarios
1	125	4/5	12	-134	4245	Máximo rango, baja tasa de datos
2	250	4/5	12	-131	2193	-
3	125	4/5	10	-129	1208	-
4	500	4/5	12	-128	1167	-
5	250	4/5	10	-126	674	-
6	500	4/5	11	-125.5	715	-
7	250	4/5	9	-123	428	-
8	500	4/5	9	-120	284	-
9	500	4/5	8	-117	220	-
10	500	4/5	7	-114	186	Mínimo rango, alta tasa de datos, menor consumo

Tabla 10: Modos de envío LoRa

A continuación se configura el canal por el cual se enviarán los datos. Hay que destacar que el módulo LoRa puede trabajar tanto en la banda de frecuencia 868 MHz como en la banda 900 MHz por lo que se podrá usar el sistema en todos los países según su legislación.

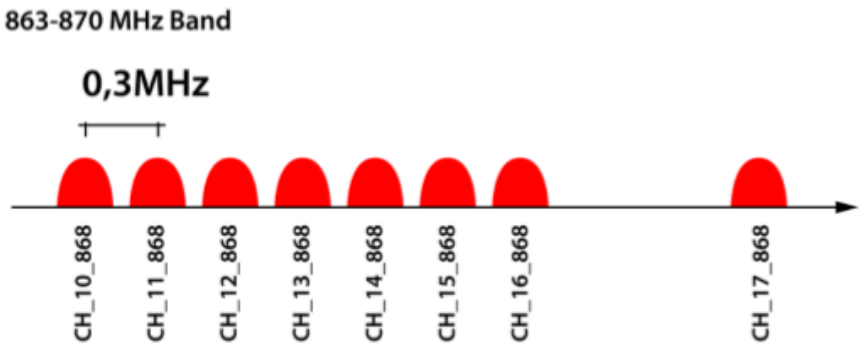

Fig. 36: Canales LoRa banda 868

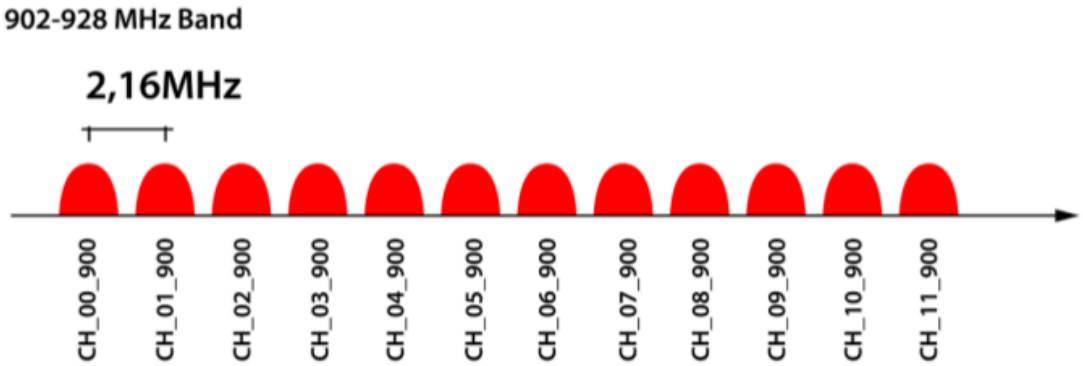

Fig. 37: Canales LoRa banda 900

Y finalmente habrá que seleccionar el nivel de potencia de entre los 3 valores disponibles.

Parámetro	SX1272 nivel de potencia
'L'	0 dBm
'H'	7 dBm
'M'	14 dBm

Tabla 11: Configuración potencia de envío

Una vez está el módulo completamente configurado pasaremos a la segunda parte donde simplemente habrá que seleccionar el modo de envío de entre los disponibles:

Función	Descripción
`sx1272.sendPacketTimeout(uint8_t dest, char *payload)`	Envía datos antes de que el tiempo de espera finalice
`sx1272.sendPacketMAXTimeout(uint8_t dest, char *payload)`	Envía datos antes de que el tiempo de espera máximo finalice
`sx1272.sendPacketTimeoutACK(uint8_t dest, char *payload)`	Envía datos antes de que el tiempo de espera finalice con ACK
`sx1272.sendPacketMAXTimeoutACK(uint8_t dest, char *payload)`	Envía datos antes de que el tiempo de espera máximo finalice con ACK
`sx1272.sendPacketTimeoutACKRetries(uint8_t dest, char *payload)`	Envía datos antes de que el tiempo de espera finalice con ACK y reintentos
`sx1272.sendPacketMAXTimeoutACKRetries(uint8_t dest, char *payload)`	Envía datos antes de que el tiempo de espera máximo finalice con ACK y reintentos

Tabla 12: Funciones envío LoRa

De la misma manera habrá que configurar y establecer el modo de recibir los datos del módulo que se encargará de recoger la información que estemos enviando.

Función	Descripción
`sx1272.receivePacketTimeout(uint8_t dest, char *payload)`	Recibe datos antes de que el tiempo de espera finalice
`sx1272.receivePacketMAXTimeout(uint8_t dest, char *payload)`	Recibe datos antes de que el tiempo de espera máximo finalice
`sx1272.receivePacketTimeoutACK(uint8_t dest, char *payload)`	Recibe datos antes de que el tiempo de espera finalice con ACK
`sx1272.receivePacketMAXTimeoutACK(uint8_t dest, char *payload)`	Recibe datos antes de que el tiempo de espera máximo finalice con ACK
`sx1272.receivePacketTimeoutACKRetries(uint8_t dest, char *payload)`	Recibe datos antes de que el tiempo de espera finalice con ACK y reintentos
`sx1272.receivePacketMAXTimeoutACKRetries(uint8_t dest, char *payload)`	Recibe datos antes de que el tiempo de espera máximo finalice con ACK y reintentos

Tabla 13: Funciones recepción LoRa

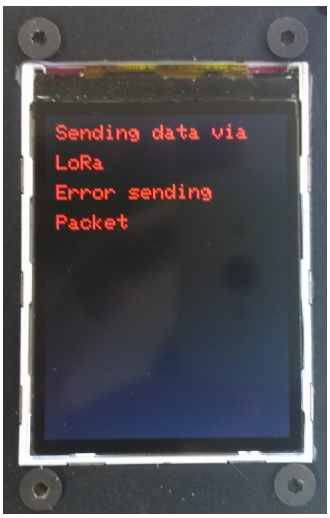

Fig. 38: Envío fallido LoRa

Fig. 39: Envío satisfactorio LoRa

5.9 Software 3G+GPS

El módulo 3G+GPS se utilizará en el programa para subir la información es caso de fallo de envío del modulo de comunicación principal. Se aprovechara su conectividad 3G para subir los datos de los sensores a un servidor con el protocolo **FTP** en forma de archivo de texto.

También se usará el módulo para establecer la posición donde se encuentra el portador del sistema mediante el **GPS**.

La comunicación con el **SIM5218** se realiza mediante el conjunto de **comandos Hayes.** Este es un lenguaje desarrollado por la compañía Hayes Communications que prácticamente se convirtió en estándar abierto de comandos para configurar y parametrizar modems. Los caracteres «AT», que preceden a todos los comandos, significan «Atención», e hicieron que se conociera también a este conjunto de comandos como **comandos AT**.

Todas las instrucciones del envío de la información se reúnen en la función `send3G()`, a continuación se explican los comandos AT utilizados para subir la información al servidor FTP.

Comando AT	Descripción
`AT+CPIN`	Introducir el código pin de la tarjeta SIM
`AT+CREG`	Establece conexión con el módulo para conectarse a la red
`AT+CGSOCKCONT`	Establece el protocolo y el nombre del punto de acceso

AT+CFTPSERV	Establece el nombre del dominio del servidor FTP
AT+CFTPPORT	Establece el puerto del servidor FTP
AT+CFTPMODE	Configura el modo FTP
AT+CFTPUN	Establece el nombre para el acceso al servidor
AT+CFTPPW	Establece la contraseña para acceder al servidor
AT+CFTPPUT	Sube un archivo al servidor

Tabla 14: Comandos AT para envío por 3G

En el siguiente diagrama de flujo se puede ver la secuencia de funcionamiento del envío de los datos al servidor con el uso de los comandos AT:

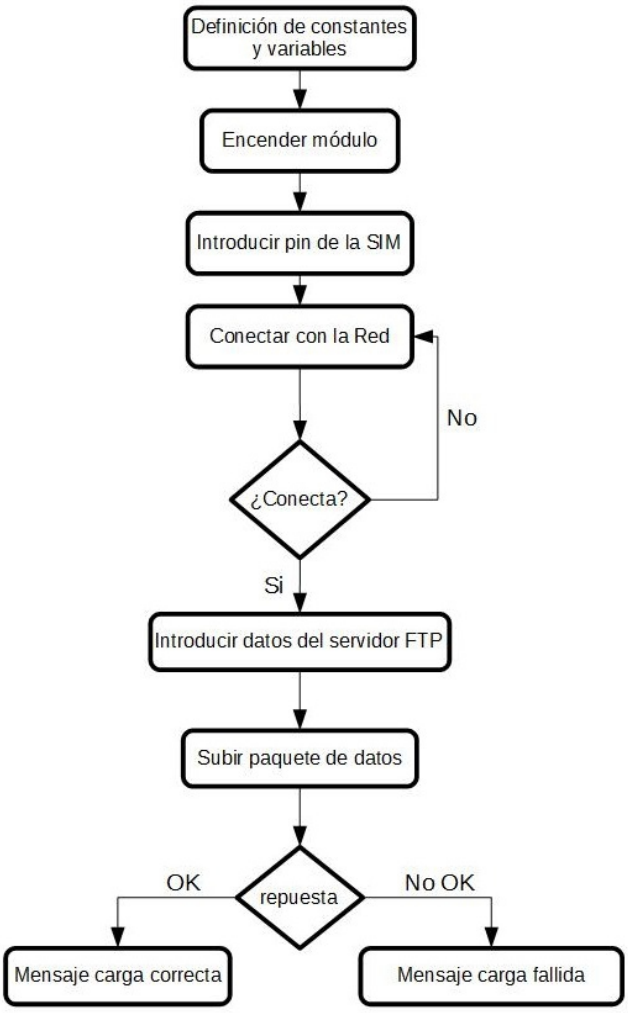

Fig. 40: Diagrama flujo subir datos por 3G

Cabe destacar que en el código se han establecido mecanismos para que en caso de que tarde más de un tiempo determinado en conectar con la red por falta de cobertura o en caso de fallo al encender el módulo aborte y continúe con el programa principal.

De la misma manera se ha creado la función **`getGPSposition()`**, con la que otra vez con los comandos AT se enviarán instrucciones al módulo para establecer conexión y conseguir las coordenadas de nuestra posición. Los comandos usados esta vez son los siguientes:

Comando AT	Descripción
`AT+CGPS`	Activa el modo GPS del módulo
`AT+CGPSINFO`	Recoge las coordenadas de la posición actual

Tabla 15: Comandos AT para GPS

Igualmente que en el envío de la trama con los datos de los sensores se usarán las funciones creadas para el envío de las coordenadas GPS tanto vía LoRa como vía 3G.

Fig. 41: Obteniendo datos de posición

5.10 Otras funciones

Para completar el código se han creado otras funciones que servirán al usuario:

- `setAlarm()` → Esta función es la que se encarga de establecer en que momento se activa la alarma para mandar la señal de alerta. En el caso de que alguno de los parámetros biométricos a superado el nivel inferior o superior establecido por el usuario la variable **alarm** sera **'1'** y se podrá observar en la trama enviada. Además que la variable alarma esté activada indicará que se deben enviar las coordenadas GPS para una posible evacuación.
- `showBatLevel()` → Esta función calcula el nivel de tensión de la batería, lo que nos ayudará a saber aproximadamente cuanto tiempo nos queda antes de que deje de funcionar el sistema.

6 Diseño de la carcasa

Para la realización del encapsulado prototipo del sistema completo se ha optado por la tecnología de impresión 3D ya que en Libelium disponemos de impresoras 3D con las que hemos podido realizar pruebas de cara al diseño del encapsulado final.

Para realizar el modelo 3D se ha utilizado el software online **Tinkercad** que permite de forma fácil e intuitiva el diseño de modelos en 3D y posteriormente la exportación de los archivos en diferentes formatos estándar para su impresión 3D.

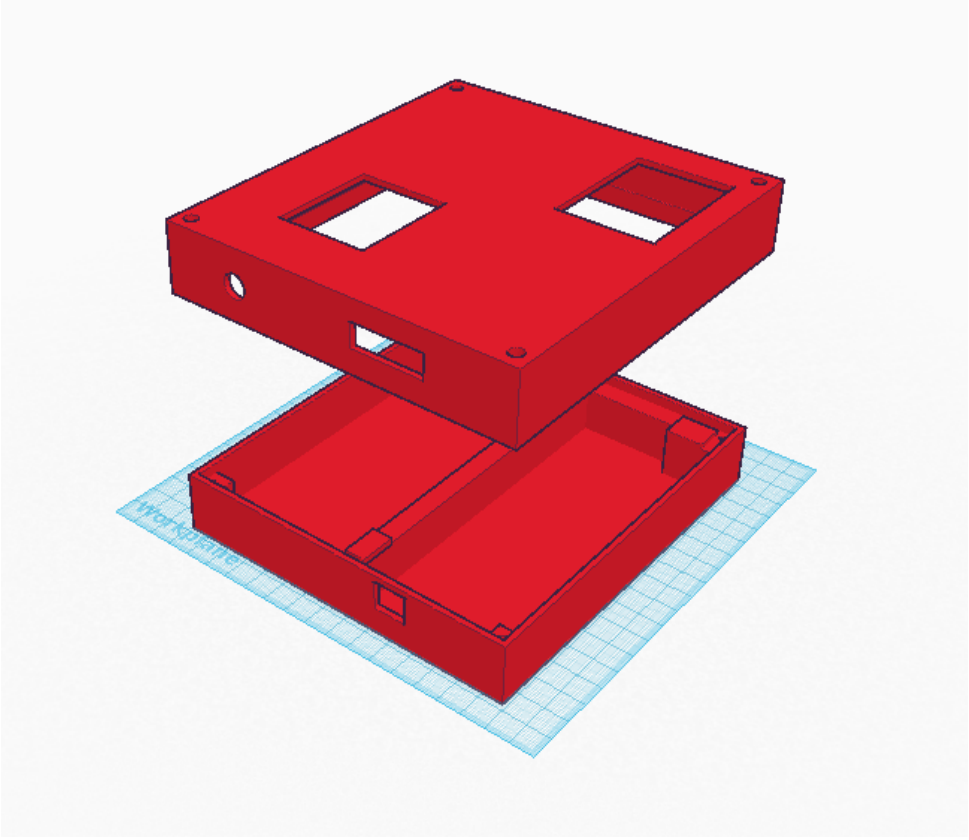

Fig. 42: Diseño de caja para el sistema

Una vez que estuvo claro el diseño final se encargo su impresión definitiva a la empresa **Shapeways** dedicada profesionalmente a la impresión 3D ofreciendo un gran acabado final y permitiendo elegir entre una gran cantidad de materiales y colores.

Fig. 43: Montaje dentro de la carcasa con sensores

Para poder transportar el sistema prototipo se compró una mochila a la que se le hicieron modificaciones para poder sacar los cables y la antena además de una ventana por la que se puede visualizar el display.

Fig. 44: Montaje completo

7 Pruebas

Con todo el sistema ya terminado se han realizado pruebas tanto de consumo y también de cobertura de los módulos de comunicación para establecer su alcance. Asimismo se realizaron pequeñas pruebas en exterior con los códigos de ejemplo generados para comprobar el funcionamiento del conjunto completo.

Como punto de control para recibir las tramas durante las pruebas se ha utilizado un Arduino con un módulo LoRa para recibir a través de este sistema y también el servidor de pruebas de Libelium para recibir las tramás subidas al mismo por el protocolo FTP.

7.1 Pruebas de consumo

El consumo del sistema variará considerablemente según el software cargado en Arduino y la forma de envío por lo que se han hecho pruebas con dos códigos distintos para ver cuanta autonomía tiene el sistema en sus límites, un código es el que tiene menor consumo y el otro el que más consume.

- Código toma de medidas y envío por LoRa → duración de la batería **17.5 horas** → consumo medio aproximado de **250 mAh**.
- Código toma de medidas y envío por 3G, toma de posición GPS y envío de posición por 3G → duración de la batería **6.5 horas** → consumo medio aproximado de **677 mAh**.

El panel solar nos permitirá aumentar la autonomía con una corriente de carga típica de **100 mAh** y de hasta 150 mAh en caso de máxima incidencia solar. En cualquier caso se podrá aumentar la autonomía con una batería extra conectándola al puerto USB del módulo Solar Charguer Shield.

Con estas pruebas se demostró que el sistema puede servir para monitorizar a una persona realizando una actividad física durante varias horas y poder ir estudiando sus constantes desde un punto de control y poder detectar si hay algún parámetro fuera de los rangos normales.

7.2 Pruebas cobertura

Dado que el módulo **LoRa** es un nuevo producto de Libelium se colaboró en las pruebas para establecer el alcance que puede llegar a tener nuestro sistema hasta el punto donde se recogerán los datos.

Las primeras pruebas fueron con linea a vista (LOS) dónde se enviaron paquetes de 90 bytes entre dos puntos a 21 km de distancia con más de 100 intentos por prueba.

Fig. 45: Puntos de las pruebas LOS

LoRa Mode	Rango	Potencia	Canal	Éxito (%)	RSSI paquete (dBm)	Sensibilidad (dB)	Margen (dB)
Modo 1	21.6 km	High	CH_12_868	100	-126.79	-134	7.21
		Max		100	-121.76	-134	12.24
		High	CH_16_868	100	-127.06	-134	6.94
		Max		100	-120.21	-134	13.79
Modo 3	21.6 km	High	CH_12_868	95	-127.29	-129	1.71
		Max		95	-120.73	-129	8.27
Modo 6	21.6 km	High	CH_12_868	99	-125.77	-125.5	-0.27
		Max		100	-119.43	-125.5	6.07
Modo 9	21.6 km	High	CH_12_868	0	-	-117	-
		Max		49	-120.95	-117	-3.95

Tabla 16: Pruebas LoRa LOS

También se realizaron pruebas de cobertura en entorno urbano (NLOS) para establecer el alcance con obstáculos donde se puede ver que el alcance se reduce considerablemente.

Estas pruebas se realizaron con las siguientes características:

- Modo de envío 1
- Máxima potencia de envío: 14 dBm
- Canal 12 de la banda de frecuencias 868 mHz
- Paquetes de 80 bytes
- 50 intentos para cada punto

Punto de envío	Rango (m)	Número de edificios que atraviesa la señal	Éxito (%)	RSSI paquete (dBm)	Margen (dB)
Punto 1	830	4	96	-124.89	9.11
Punto 2	960	14	92	-131.26	2.74
Punto 3	1070	6	98	-120.24	13.76
Punto 4	1530	14	98	-130.16	3.84
Punto 5	863	6	100	-120.42	13.58

Tabla 17: Pruebas LoRa NLOS

En el caso de la subida de datos al servidor por la red **3G** siempre ha sido exitoso dado lo extendido que esta esta red y gracias también a la conectividad **GPRS** que dispone el módulo todavía más extendida que la anterior aunque con menor tasa de transferencia de datos.

En cuanto al posicionamiento con el **GPS** todas las pruebas también han sido satisfactorias dando coordenadas con muy buena precisión aunque con el inconveniente de un alto tiempo de obtención de la posición de entre 15 y 20 segundos.

7.3 Pruebas del sistema completo

Para comprobar el correcto funcionamiento del sistema se han hecho pruebas en el exterior durante varias horas para testearlo. En este caso se ha usado un código de ejemplo en el que se recogen las medidas biométricas y se envían por LoRa o 3G.

En las siguientes fotos se puede observar de como queda el sistema puesto sobre la persona a monitorizar.

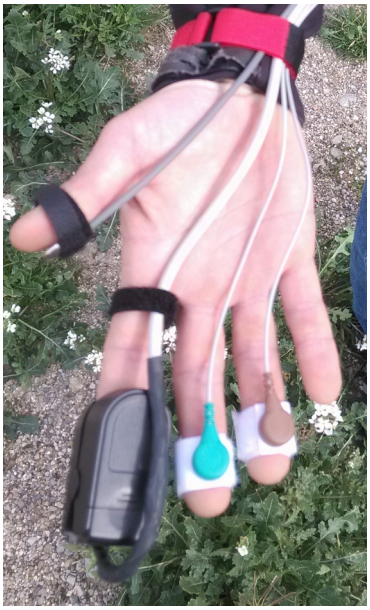

Fig. 46: Colocación de los sensores en la mano

Fig. 47: Portando el sistema con los sensores

Fig. 48: Portando el sistema con los sensores

Con los datos recogidos por los sensores se han realizado gráficas para corroborar su correcto funcionamiento, en este caso se puede observar la evolución de las pulsaciones y de las respiraciones por minuto en el mismo periodo de tiempo en el cual se aumento la actividad física. Estas tramas corresponden a aproximadamente una hora de actividad.

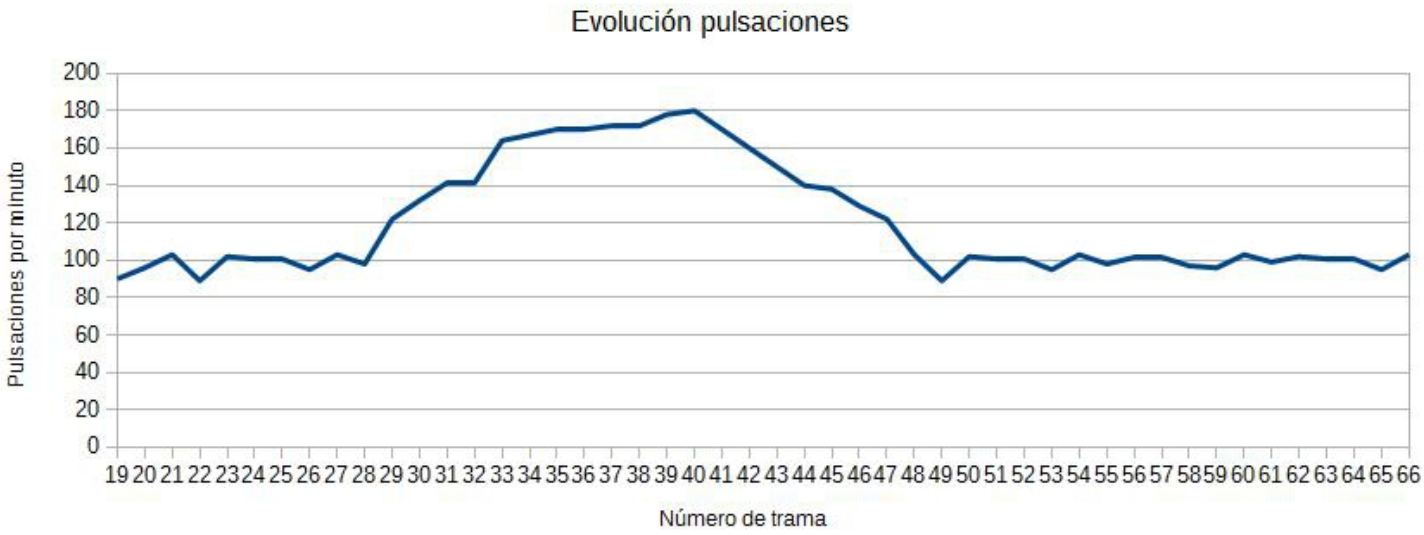

Fig. 49: Gráfica pulsaciones

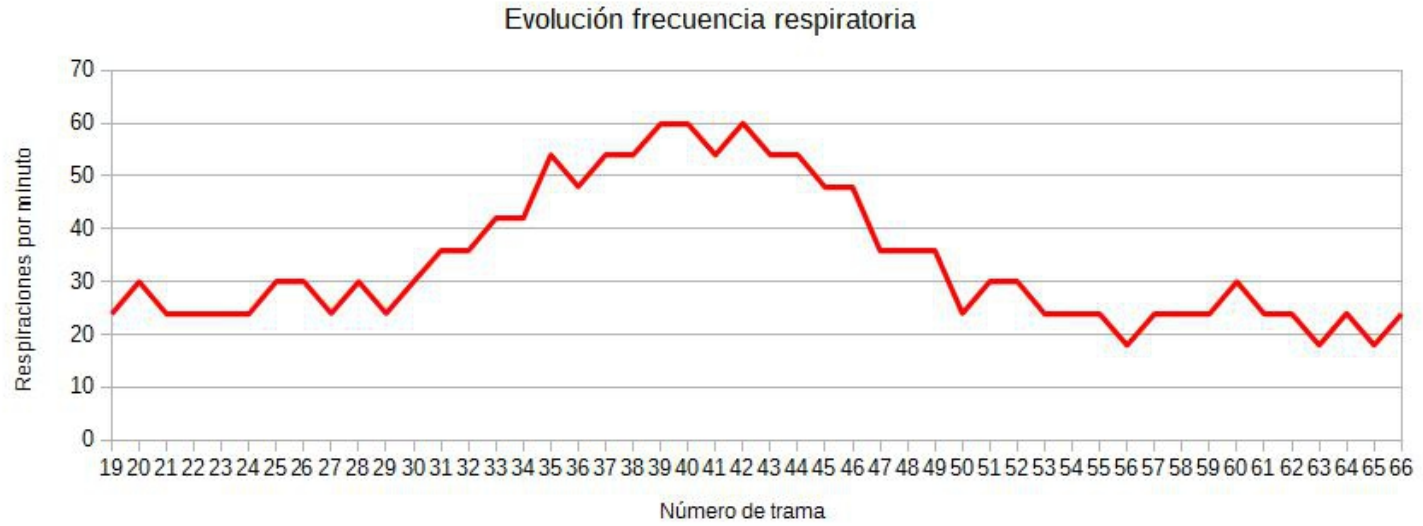

Fig. 50: Gráfica respiraciones

8 Conclusiones

Se ha conseguido alcanzar los objetivos que se habían planteado, consiguiendo crear un sistema prototipo de un producto, gracias a la modularidad de Arduino, que es capaz de monitorizar en tiempo real de forma autónoma a un individuo con el fin de establecer unos limites de seguridad entorno a los parámetros biométricos de su cuerpo pudiendo alertar de valores extremos por encima de los valores establecidos .

Este conjunto final completo, al que hemos denominado Portable e-Health System trabajando con Arduino supone un buen punto de partida para crear futuros productos y para que gente interesada en el campo de la medicina pueda experimentar con un bajo presupuesto.

Como el acceso al firmware final es sencillo y está disponible, cualquier persona con algún conocimiento de programación puede usar, personalizar o mejorar su sistema más allá del original.

Los objetivos marcados y cumplidos son:

- Estudio básico de la tecnología que se iba a trabajar: Arduino y el mundo Open Source.
- Comprender en profundidad el hardware utilizado (datasheet, documentación técnica…).
- Diseño de la placa de adaptación para todos los módulos utilizados.
- Conocer el lenguaje de programación C/C++ siendo este la base de todos los códigos realizados dentro del entorno de desarrollo de Arduino.
- Desarrollo del código firmware necesario para controlar los módulos, con la consiguiente comprensión de todas las librerías utilizadas.
- Pruebas del sistema.

En adición a estos objetivos, se han conseguido adquirir y mejorar habilidades gracias a las tareas desarrolladas durante el proyecto, como:

- Estudio de los diferentes protocolos de comunicación.
- Aprendizaje y manejo del entorno EAGLE.

- Búsqueda de proveedores para ciertos componentes.
- Soldadura de componentes.

8.1 Conclusiones personales

Ha sido una experiencia muy gratificante tener la oportunidad de trabajar como ingeniero de investigación y desarrollo de Libelium, y más concretamente en el departamento Cooking Hacks, donde he podido aprender y donde se me ha provisto de todo el hardware y el conocimiento que he necesitado. La aplicación real de los conocimientos adquiridos durante la carrera y la adquisición de otros muchos durante estos meses de trabajo, ha hecho que tantos años de estudio hayan merecido la pena.

He podido ver de primera mano las dificultades por las que pasa un proyecto real en una empresa y con las que el ingeniero debe lidiar para sacarlo adelante como pueden ser la búsqueda de material entre los distintos distribuidores, tiempos de entrega, tiempos estimados para realizar las tareas.

8.2 Agradecimientos

En primer lugar quiero dar las gracias a todos los compañeros de Libelium, y en especial al departamento de Cooking Hacks, Luis Martín y Jorge Casanova y además a Javier Solobera encargado de producto de la empresa por aceptarme para realizar este proyecto con ellos y haberme guiado en el proceso de diseño de un producto real. También quiero dar las gracias a toda la gente del departamento de investigación y desarrollo, como Marcos Yarza, Ahmad Saad, Alejandro Gallego, Javier Siscart o Yuri Carmona entre otros, por haberme ayudado con todas mis dudas.

También quiero dar las gracias a José María López Pérez, tanto por haber sido el tutor de mi proyecto, como profesor de la carrera, y al resto de profesores de la Escuela de Ingeniería que me han dado clases en algún momento.

Por último quiero dar las gracias a mi familia por todo su apoyo durante todo este tiempo, a mis amigos y compañeros de trabajo, como Marcos Martínez, Víctor Boria, Luis Miguel Martí o Víctor Lapuente. Todos han aportado algo necesario en algún momento en particular del proyecto.

A todos muchísimas gracias.

9 Bibliografía y referencias

http://www.libelium.com/

http://www.cooking-hacks.com/

http://es.wikipedia.org/wiki/H%C3%A1galo_usted_mismo

http://opensource.org/

http://arduino.cc/

http://wm.sim.com/

http://www.seeedstudio.com/

http://www.elecfreaks.com/

http://micron20.com/

http://es.farnell.com/

https://github.com/adafruit/Adafruit_ILI9341

https://github.com/adafruit/Adafruit-GFX-Library

https://www.tinkercad.com/

http://www.shapeways.com/

Índice de Figuras

Fig. 1: Logo de Cooking Hacks....8
Fig. 2: Logo Open Software....9
Fig. 3: Logo Open Hardware....9
Fig. 4: Ejemplo producto POLAR....13
Fig. 5: VITRO de Medlab....13
Fig. 6: Placa Arduino UNO....16
Fig. 7: Placa e-Health....18
Fig. 8: Ejemplo electrónica de adaptación SPO2....19
Fig. 9: Diagrama de bloques general....22
Fig. 10: Placa Arduino MEGA....25
Fig. 11: Display TFT LCD de Elecfreaks....26
Fig. 12: Circuito integrado CD4050BC....26
Fig. 13: Módulo SX1272....27
Fig. 14: Placa Multiprotocol....28
Fig. 15: Módulo 3G+GPS....29
Fig. 16: Solar Charger Shield v2....29
Fig. 17: Batería 4400 mAh....30
Fig. 18: Panel solar flexible....31
Fig. 19: Sensor de flujo de aire....31
Fig. 20: Sensor de sudoración de la piel....32
Fig. 21: Sensor de posición....32
Fig. 22: Sensor pulsioximetría....33
Fig. 23: Sensor de temperatura....33
Fig. 24: Ejemplo pruebas previas....34
Fig. 25: Tira de pines macho....35
Fig. 26: Circuito esquemático....36
Fig. 27: Módulos cara TOP....37
Fig. 28: Módulos cara BOTTOM....37
Fig. 29: Ruteado PCB....38
Fig. 30: IDE de programación de Arduino....41
Fig. 31: Diagrama de flujo código de ejemplo....43
Fig. 32: pintLogo....44

Fig. 33: setScreen....44
Fig. 34: Mostrando datos sensores....45
Fig. 35: Diagrama flujo sensores....46
Fig. 36: Canales LoRa banda 868....49
Fig. 37: Canales LoRa banda 900....49
Fig. 38: Envío fallido LoRa....51
Fig. 39: Envío satisfactorio LoRa....51
Fig. 40: Diagrama flujo subir datos por 3G....52
Fig. 41: Obteniendo datos de posición....53
Fig. 42: Diseño de caja para el sistema....56
Fig. 43: Montaje dentro de la carcasa con sensores....57
Fig. 44: Montaje completo....57
Fig. 45: Puntos de las pruebas LOS....59
Fig. 46: Colocación de los sensores en la mano....61
Fig. 47: Portando el sistema con los sensores....61
Fig. 48: Portando el sistema con los sensores....61
Fig. 49: Gráfica pulsaciones....62
Fig. 50: Gráfica respiraciones....62
Fig. 51: Ejemplo diagrama explicativo para la web....88

Índice de Tablas

Tabla 1: Características Arduino MEGA....25
Tabla 2: Características LoRa....27
Tabla 3: Características batería 4400 mAh....30
Tabla 4: Características panel solar flexible....31
Tabla 5: Resumen conexiones....35
Tabla 6: Funciones control TFT LCD....44
Tabla 7: Funciones lectura y muestra sensores....45
Tabla 8: Formato trama de envío....47
Tabla 9: Funciones configuración LoRa....47
Tabla 10: Modos de envío LoRa....48
Tabla 11: Configuración potencia de envío....49
Tabla 12: Funciones envío LoRa....50
Tabla 13: Funciones recepción LoRa....50
Tabla 14: Comandos AT para envío por 3G....52
Tabla 15: Comandos AT para GPS....53
Tabla 16: Pruebas LoRa LOS....59
Tabla 17: Pruebas LoRa NLOS....60

Glosario

PCB Printed Circuit Board (Placa de circuito impreso)

DIY Do It Yourself (Hágalo usted mismo)

IDE Integrated Development Environment (Entorno de desarrollo integrado)

SPI Serial Peripheral Interface (Interfaz de comunicación serie)

UART Universal Asynchronous Receiver-Transmitter

I^2C Inter-Integrated Circuit (Circuitos Inter-Integrados)

TFT LCD Thin Film Transistor-Liquid Crystal Display (Pantalla de cristal liquido de transistores de película fina)

ISM Industrial, Scientific and Medical

SDA Serial Data Line (Línea de Datos Serie)

SCL Serial Clock Line (Línea de Reloj Serie)

ACK Acknowledgement (Asentimiento)

EAGLE Easily Applicable Graphical Layout Editor

Tx Transmission (Transmisión)

Rx Receiver (Recepción)

LOS Line Of Sight

NLOS Non Line Of Sight

Anexo A: Códigos de programación

A.1. Código completo de ejemplo

```
/*
 *  PORTABLE eHealth SYSTEM complete example.
 *
 *  Description: Example with all modules working
 *
 *  Copyright (C) 2015 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/> .
 *
 *  Version 1.0
 *  Author: ANDRES FALO
 */

//Libraries eHealth
#include <eHealth.h>
#include <PinChangeInt.h>

//Libraries TFT
#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"

//Libraria LoRa
#include "SX1272.h"

//Definition of used pins for TFT
#define TFT_DC 49
#define TFT_CS 53
#define TFT_MOSI 51
#define TFT_CLK 52
#define TFT_RST 48
#define TFT_MISO 50

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST, TFT_MISO);

//eHealth
int BPM;
int SPO2;
float conductance;
//float resistance;
//float conductanceVol;
float temperature;
uint8_t position;
char* position2;
int countsPerMinute;

char POS1[]="Supine position";
```

```
char POS2[]="Left position";
char POS3[]="Rigth position";
char POS4[]="Prone position";
char POS5[]="Stand position";
char POS6[]="non-defined";

//LoRa
char conductanceSEND[10];
char temperatureSEND[10];

int e;

//3G + GPS
int8_t answer;
int onModulePin = 22, aux;
int data_size = 0;
int end_file = 0;

char gps_data[150];
int counter;

char aux_str[150];

char response[150];
int x = 0;
long previous;

int cont3G;
unsigned long start;

int errorGPS;
int datagpsOK;

int y = 1;

//BATTERY
const int analogInPin = A0;  // Analog input pin that the VBAT pin is attached to
int BatteryValue = 0;        // value read from the VBAT pin
float outputValue = 0;

//other variables
char dataFrame[150];

int stabilize;//how many times you read de sensors before reading

int cont = 0;

int error;

int alarm;

int numFrame = 0;

// the setup routine runs once when you press reset:
void setup() {

  tft.begin();

  printLogo();

  eHealth.initPulsioximeter();

  eHealth.initPositionSensor();

  //Attach the inttruptions for using the pulsioximeter.
  pinMode(A15, INPUT);
  digitalWrite(A15, HIGH);
  PCintPort::attachInterrupt(A15, &readPulsioximeter, RISING);
```

```
  pinMode(onModulePin, OUTPUT);

  Serial.begin(115200);

}

// the loop routine runs over and over again forever:
void loop() {

  setScreen();

  //loop to stabilize measures
  for (stabilize=3; stabilize>1; stabilize--)
  {

    showBatLevel();

    getShowSPO2();

    getShowConductance();

    getShowTemp();

    getShowPosition();

    getShowAirflow();

  }

  tft.fillScreen(ILI9341_BLACK);

  setAlarm();

  compoundFrame();

  startLora();

  sendLora();

  if (error == 1)
  {
    send3G();
  }

  if ((alarm == 1) || (numFrame == 10*y))
  {
    errorGPS=0;

    getGPSposition();

    if (errorGPS==0 && datagpsOK==1)
    {
      strncpy(dataFrame,gps_data,150);

      y++;

      startLora();

      sendLora();

      if (error == 1)
      {
        send3G();
      }

    }
    else
    {
      delay(1000);
```

```
    }
  }
}
```

A.2. Funciones

```
//************************************************************

void printLogo(){

  tft.fillScreen(ILI9341_BLACK);
  tft.setRotation(2);
  tft.setCursor(50, 20);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(3);
  tft.println("Portable");
  tft.setCursor(50, 50);
  tft.println("e-Health");
  tft.setCursor(65, 80);
  tft.println("System");
  tft.setCursor(30, 120);
  tft.setTextColor(ILI9341_YELLOW);
  tft.setTextSize(2);
  tft.println("for Mountaineer");
  tft.setCursor(60, 140);
  tft.println("Assitance");

  //(x,y,width,length,colour)
  tft.fillRect(100, 175, 30, 100, ILI9341_WHITE);
  tft.setRotation(1);
  tft.fillRect(80, 65, 30, 100, ILI9341_WHITE);

  tft.setRotation(2);
  tft.setCursor(30, 300);
  tft.setTextColor(ILI9341_RED);
  tft.setTextSize(2);
  tft.println("by Cooking Hacks");

  delay(2000);

  tft.setRotation(2);
  tft.setCursor(50, 20);
  tft.setTextColor(ILI9341_BLACK);
  tft.setTextSize(3);
  tft.println("Portable");
  tft.setCursor(50, 50);
  tft.println("e-Health");
  tft.setCursor(65, 80);
  tft.println("System");
  tft.setCursor(30, 120);
  tft.setTextColor(ILI9341_BLACK);
  tft.setTextSize(2);
  tft.println("for Mountaineer");
  tft.setCursor(60, 140);
  tft.println("Assitance");

  //(x,y,width,length,colour)
  tft.fillRect(100, 175, 30, 100, ILI9341_BLACK);
  tft.setRotation(1);
  tft.fillRect(80, 65, 30, 100, ILI9341_BLACK);

  tft.setRotation(2);
  tft.setCursor(30, 300);
```

```
  tft.setTextColor(ILI9341_BLACK);
  tft.setTextSize(2);
  tft.println("by Cooking Hacks");
}

//************************************************************

void setScreen(){

  tft.setCursor(10, 10);
  tft.setTextColor(ILI9341_YELLOW);
  tft.setTextSize(2);
  tft.println("Conductance:");
  tft.setCursor(10, 40);
  tft.setTextColor(ILI9341_YELLOW);
  tft.setTextSize(2);
  tft.println("PRbpm:");
  tft.setCursor(10, 70);
  tft.setTextColor(ILI9341_YELLOW);
  tft.setTextSize(2);
  tft.println("%SPO2:");
  tft.setCursor(10, 100);
  tft.setTextColor(ILI9341_YELLOW);
  tft.setTextSize(2);
  tft.println("Temperature:");
  tft.setCursor(10, 130);
  tft.setTextColor(ILI9341_YELLOW);
  tft.setTextSize(2);
  tft.println("Position:");
  tft.setCursor(10, 190);
  tft.setTextColor(ILI9341_YELLOW);
  tft.setTextSize(2);
  tft.println("Airflow BPM:");
  tft.setCursor(10, 250);
  tft.setTextColor(ILI9341_YELLOW);
  tft.setTextSize(2);
  tft.println("Battery Voltage:");
}

//************************************************************

void readPulsioximeter(){

  cont ++;

  if (cont == 50) { //Get only of one 50 measures to reduce the latency
    eHealth.readPulsioximeter();
    cont = 0;
  }
}

//************************************************************

void showBatLevel(){

  tft.setCursor(10, 280);
  tft.setTextColor(ILI9341_BLACK);
  tft.setTextSize(2);
  tft.println(outputValue);

  BatteryValue = analogRead(analogInPin);
  outputValue = ((float(BatteryValue)*5)/1023*2)-0.15;


  tft.setCursor(10, 280);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println(outputValue);
}
```

```
//************************************************************

void getShowSPO2(){

  tft.setTextColor(ILI9341_BLACK);
  tft.setCursor(100,40);
  tft.println(BPM);

  tft.setTextColor(ILI9341_BLACK);
  tft.setCursor(100,70);
  tft.println(SPO2);

  BPM=eHealth.getBPM();

  SPO2=eHealth.getOxygenSaturation();

  tft.setTextColor(ILI9341_WHITE);
  tft.setCursor(100,40);
  tft.println(BPM);

  tft.setTextColor(ILI9341_WHITE);
  tft.setCursor(100,70);
  tft.println(SPO2);
}

//************************************************************

void getShowConductance(){

  tft.setTextColor(ILI9341_BLACK);
  tft.setCursor(160, 10);
  tft.println(conductance);

  conductance = eHealth.getSkinConductance();

  tft.setTextColor(ILI9341_WHITE);
  tft.setCursor(160, 10);
  tft.println(conductance);
}

//************************************************************

void getShowTemp(){

  tft.setTextColor(ILI9341_BLACK);
  tft.setCursor(160,100);
  tft.println(temperature);

  temperature = eHealth.getTemperature();

  tft.setTextColor(ILI9341_WHITE);
  tft.setCursor(160,100);
  tft.println(temperature);
}

//************************************************************

void getShowPosition(){

  tft.setTextColor(ILI9341_BLACK);
  tft.setCursor(10,160);
  tft.println(position2);

  position = eHealth.getBodyPosition();

  if (position == 1) {
    position2=POS1;
  }
```

```
  else if (position == 2) {
    position2=POS2;
  }
  else if (position == 3) {
    position2=POS3;
  }
  else if (position == 4) {
    position2=POS4;
  }
  else if (position == 5) {
    position2=POS5;
  }
  else  {
    position2=POS6;
  }

  tft.setTextColor(ILI9341_WHITE);
  tft.setCursor(10,160);
  tft.println(position2);
}

//************************************************************

void getShowAirflow(){

  tft.setCursor(160, 190);
  tft.setTextColor(ILI9341_BLACK);
  tft.println(countsPerMinute);

  countsPerMinute=eHealth.airFlowBPM();

  tft.setCursor(160, 190);
  tft.setTextColor(ILI9341_WHITE);
  tft.println(countsPerMinute);

}

//************************************************************
void setAlarm(){

  if ( SPO2<90 || BPM<30){

    alarm=1;
  }
  else{
    alarm=0;
  }

}

//************************************************************

void compoundFrame(){

  dtostrf(conductance,2,2,conductanceSEND);
  dtostrf(temperature,2,2,temperatureSEND);

  snprintf(dataFrame,150,
  "#N:%d#COND:%s#BPM:%d#SPO2:%d#TEMP:%s#AirBPM:%d#POS:%s#ALARM:
%d",numFrame,conductanceSEND,BPM,SPO2,temperatureSEND,countsPerMinute,position2,alarm);

  numFrame++;
  datagpsOK=0;

}

//************************************************************

void startLora(){
```

```
  tft.setTextColor(ILI9341_RED);
  tft.setCursor(10,10);
  tft.println("Sending data via");
  tft.setCursor(10,40);
  tft.println("LoRa");

  sx1272.ON();
  e = sx1272.setMode(1);

  // Select frequency channel
  e = sx1272.setChannel(CH_16_868);

  // Select output power (Max, High or Low)
  e = sx1272.setPower('H');

  // Set the node address and print the result
  e = sx1272.setNodeAddress(4);

}

//************************************************************

void sendLora(){

  e = sx1272.sendPacketTimeoutACK(3, dataFrame);

  sx1272.OFF();

  error=0;

  if (e == 0)
  {
    tft.setCursor(10,70);
    tft.println("Packet sent");
  }
  else
  {
    tft.setCursor(10,70);
    tft.println("Error sending");
    tft.setCursor(10,100);
    tft.println("Packet");
    error=1;
  }

  delay(2000);
  tft.fillScreen(ILI9341_BLACK);

}

//************************************************************

void send3G(){

  tft.setTextColor(ILI9341_RED);
  tft.setCursor(10,10);
  tft.println("Sending data via 3G");

  power_on();

  if (cont3G == 0)

  {
    tft.setCursor(10,40);
    tft.println("Problem turning on the module");
    delay(2000);
    tft.fillScreen(ILI9341_BLACK);
  }
```

```
  else{

    delay(3000);

    start = millis();

    while( ((sendATcommand("AT+CREG?", "+CREG: 0,1", 500) || sendATcommand("AT+CREG?", "+CREG: 0,5",
500)) == 0) && (millis() - start < 20000) );

    tft.setCursor(10,40);
    tft.println("Waiting to connect to the network...");

    // sets APN, user name and password
    sendATcommand("AT+CGSOCKCONT=1,\"IP\",\"******\"", "OK", 2000);
    sendATcommand("AT+CSOCKAUTH=1,1,\"*****\",\"*****\"", "OK", 2000);

    sendATcommand("AT+CFTPSERV=\"*****\"", "OK", 2000);
    sendATcommand("AT+CFTPPORT=**", "OK", 2000);
    sendATcommand("AT+CFTPMODE=**", "OK", 2000);
    sendATcommand("AT+CFTPUN=\"*****\"", "OK", 2000);
    sendATcommand("AT+CFTPPW=\"*****\"", "OK", 2000);


    // uploads data to the FTP server
    if (datagpsOK==0)
    {
      sprintf(aux_str, "AT+CFTPPUT=\"/*****/sensor%03u.txt\"", numFrame);
    }
    else
    {
      sprintf(aux_str, "AT+CFTPPUT=\"/*****/position%03u.txt\"", numFrame);
    }

    answer = sendATcommand(aux_str, "+CFTPPUT: BEGIN", 20000);

    if (answer == 1)
    {
      Serial.print(dataFrame);

      // Sends <Ctrl+Z>
      aux_str[0] = 0x1A;
      aux_str[1] = 0x00;
      answer = sendATcommand(aux_str, "OK", 20000);

      if (answer == 1)
      {
        tft.setCursor(10,100);
        tft.println("Upload done");
      }
      else
      {
        tft.setCursor(10,100);
        tft.println("Upload fail");
      }

    }
    else
    {
      tft.setCursor(10,100);
      tft.println("Error with server");
    }

  }

  delay(2000);
  tft.fillScreen(ILI9341_BLACK);

}
```

```
//************************************************************

void power_on(){

  uint8_t answer=0;

  // checks if the module is started
  answer = sendATcommand("AT", "OK", 2000);
  if (answer == 0)
  {
    // power on pulse
    digitalWrite(onModulePin,HIGH);
    delay(3000);
    digitalWrite(onModulePin,LOW);

    cont3G=15;

    // waits for an answer from the module
    while(answer == 0 && cont3G>0){

      cont3G--;

      // Send AT every two seconds and wait for the answer
      answer = sendATcommand("AT", "OK", 2000);
    }
  }

}

//************************************************************

int8_t sendATcommand(char* ATcommand, char* expected_answer1,
unsigned int timeout)
{

  uint8_t x=0,  answer=0;
  char response[100];
  unsigned long previous;

  memset(response, '\0', 100);    // Initialize the string

  delay(100);

  while( Serial.available() > 0) Serial.read();    // Clean the input buffer

  Serial.println(ATcommand);    // Send the AT command

    x = 0;
  previous = millis();

  // this loop waits for the answer
  do{

    if(Serial.available() != 0){
      response[x] = Serial.read();
      x++;
      // check if the desired answer is in the response of the module
      if (strstr(response, expected_answer1) != NULL)
      {
        answer = 1;
      }
    }
    // Waits for the asnwer with time out
  }
  while((answer == 0) && ((millis() - previous) < timeout));

  return answer;
}
```

```
//************************************************************

void getGPSposition(){

  tft.setTextColor(ILI9341_RED);
  tft.setCursor(10,10);
  tft.println("Alarm detected!");

  tft.setCursor(10,40);
  tft.println("Geting position");

  power_on();

  if (cont3G == 0)

  {
    tft.setCursor(10,70);
    tft.println("Problem turning on the module");
    delay(2000);
    tft.fillScreen(ILI9341_BLACK);
    errorGPS=1;
  }

  else

  {

    tft.setCursor(10,70);
    tft.println("Turn on the board");

    delay(2000);

    // starts GPS session in stand alone mode
    answer = sendATcommand("AT+CGPS=1,1","OK",1000);
    if (answer == 0)
    {
      //Serial.println("Error starting the GPS");
      //Serial.println("The code stucks here!!");
      while(1);
    }

    start = millis();

    datagpsOK=0;

    while ((millis() - start < 60000) && (datagpsOK==0))

    {

      answer = sendATcommand("AT+CGPSINFO","+CGPSINFO:",1000);    // request info from GPS

      if (answer == 1)
      {

        counter = 0;
        do{
          while(Serial.available() == 0);
          gps_data[counter] = Serial.read();
          counter++;
        }
        while(gps_data[counter - 1] != '\r');
        gps_data[counter] = '\0';

        if(gps_data[0] == ',')
        {

          tft.setCursor(10,130);
          tft.println("GPS data:");
          tft.setCursor(10,160);
```

```
          tft.println("NO DATA");
          tft.setTextColor(ILI9341_BLACK);
          delay(200);
          tft.setCursor(10,160);
          tft.println("NO DATA");
          tft.setTextColor(ILI9341_RED);

        }
        else
        {
          datagpsOK=1;
          tft.setCursor(10,130);
          tft.println("GPS data:");
          tft.setCursor(10,160);
          tft.println(gps_data);
          delay(2000);
        }

      }
      else
      {
        tft.setCursor(10,160);
        tft.println("ERROR");
      }
    }

    delay(3000);
    tft.fillScreen(ILI9341_BLACK);
  }
}
```

A.3. Modificaciones en las librerías

```
//!*****************************************************************************
//!             Name: airflowBPM()                                               *
//!             Description: It prints air flow BPM in the serial monitor        *
//!             Param : void                                                     *
//!             Returns: void                                                    *
//!             Example: eHealth.airflowBPM();                                   *
//!*****************************************************************************

int eHealthClass::airFlowBPM(void)
{
  count = 0;
  timePreviousMeassure = millis();

  while (millis()-timePreviousMeassure < 10000) {

    valRead = analogRead(1);

    valRead   = map(valRead, 0, 1023, 5, 45);
    delay(100);

    if(valRead> 7)      waveState=1;

    if(waveState==1)
    {
      if(valRead<6)
      {
        count=count+1;
        waveState=0;
        delay(100);
      }
    }
  }

  if(count ==0 ){
    Serial.println("Apnea");
    countsPerMinute = 0;
  }

  else {
    countsPerMinute = 6*count;
    Serial.println(countsPerMinute);
    delay(20);
  }

  return countsPerMinute;

}
```

```
//!*****************************************************************************
//!             Name:   readPulsioximeter()                                    *
//!             Description: It reads a value from pulsioximeter sensor.       *
//!             Param : void                                                   *
//!             Returns: void                                                  *
//!             Example: readPulsioximeter();                                  *
//!*****************************************************************************

void eHealthClass::readPulsioximeter(void)
{
  uint8_t digito[41];

  uint8_t A = 0;
  uint8_t B = 0;
  uint8_t C = 0;
  uint8_t D = 0;
  uint8_t E = 0;
  uint8_t F = 0;
  uint8_t G = 0;

  for (int i = 0; i<41 ; i++) { // read all the led's of the module
    A = !digitalRead(13);
    B = !digitalRead(12);
    C = !digitalRead(11);
    D = !digitalRead(10);
    E = !digitalRead(9);
    F = !digitalRead(8);
    G = !digitalRead(7);

    digito[i] = segToNumber(A, B, C ,D ,E, F, G);
    delayMicroseconds(300); //300 microseconds

  }

  SPO2 = 10 * digito[24] + digito[20];
  BPM  = 100 * digito[18] + 10 * digito[2] + digito[0];

}
```

Anexo B: Documentación online generada

Se ha realizado una documentación online del proyecto que se colgará en la web de Cooking Hack, como un método para compartir los avances realizados con todos los usuarios del mundo, fomentando el hardware y software libre, y a su vez obtener la ayuda de las personas interesadas por la información, intercambiando emails y compartir experiencias, obteniendo conocimientos de manera recíproca.

Dicha documentación servirá en el futuro como tutorial de uso para los futuros usuarios que quieran replicar el sistema por ellos mismos añadiéndoles mejoras o los cambios necesarios para adecuarlo al uso de sus necesidades.

Dado que la documentación irá colgada en internet, ésta se ha generado con el lenguaje estándar para la elaboración de páginas web **HTML**. Este artículo se ha hecho en conjunto con el departamento de diseño de la empresa.

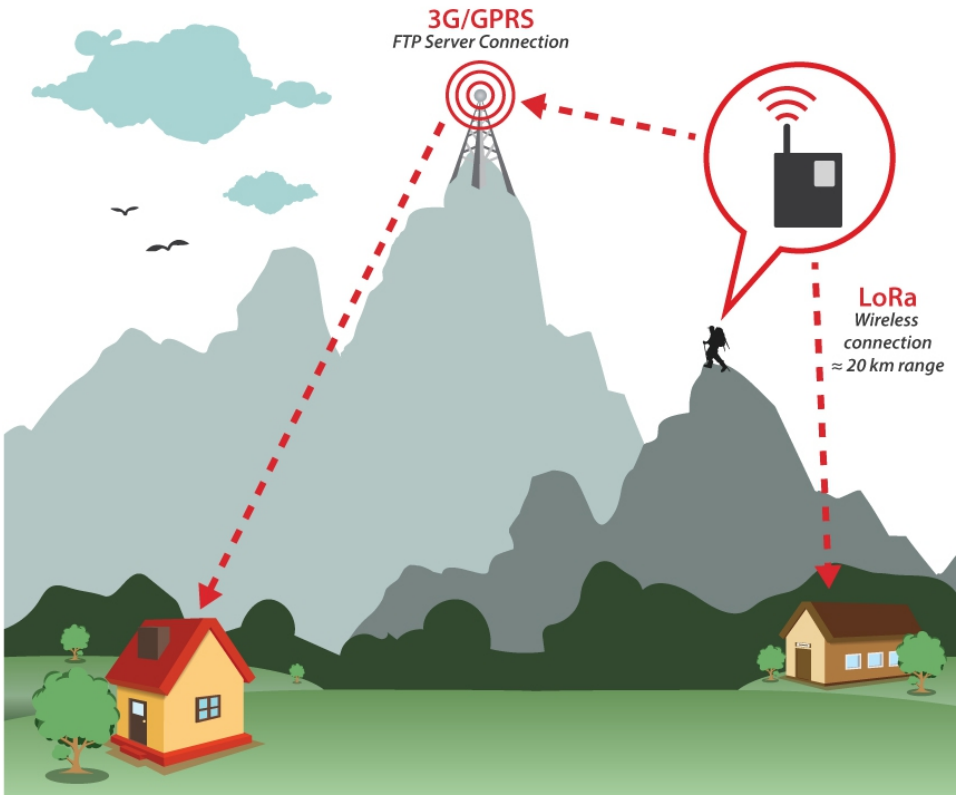

Fig. 51: Ejemplo diagrama explicativo para la web