Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER Nº 201807

Sistema basado en IoT de recogida y envío de datos ambientales, biométricos y mecánicos a bordo de un vehículo para SmartDriving

D. GARCÍA FERNÁNDEZ, Santiago
TUTOR: D. CORCOBA MAGAÑA, VÍCTOR
TUTOR EMPRESA: D. RIONDA RODRÍGUEZ, ABEL

FECHA: JULIO 2018

Resumen

Uno de los principales objetivos del Internet de las Cosas es el tener la capacidad de recoger una gran cantidad de datos con el fin de analizar e identificar lo que ocurre en cualquier tipo de situación o aplicación con el fin último de mejorar el desempeño de uno o varios elementos de un sistema en base a la corrección de errores o deficiencias.

Este trabajo se basa en la aplicación de este nuevo concepto de Internet de las Cosas al ámbito de la conducción, sin embargo el trabajo no se limita a estudiar únicamente el comportamiento del vehículo sino que se trata de estudiar también al conductor puesto que este es sin ninguna duda una parte esencial dentro de esta actividad. El uso de dispositivos captadores de variables biométricas del conductor será por tanto un punto diferencial y de interés en este trabajo. Se tratará por consiguiente de analizar la relación entre el binomio vehículo-conductor pero se añadirá también un tercer elemento que sería el ambiente el cual puede afectar al comportamiento de ambas partes de dicho binomio. El objetivo último por tanto sería el aumentar la eficiencia en la conducción y lo que es aún más importante, la seguridad al volante todo en ello en base a la generación de recomendaciones sonoras o visuales que llevarán a la implementación de un sistema de ayuda al Smart Driving a bordo.

Para la consecución de estos objetivos es necesario implementar un sistema que tenga la función principal de recoger datos de todas las diferentes fuentes dentro del vehículo. Este sistema será el objeto de estudio principal a lo largo de este Trabajo Fin de Master. Para el diseño de este sistema se estudiarán y justificarán convenientemente todas las decisiones tomadas que han llevado a un sistema principal basado en una SBC de tipo Raspberry Pi a la cual llegan datos a través de diferentes medios que van desde conexiones inalámbricas como Bluetooth o Wi-Fi hasta conexiones directas de cobre a los pines de entrada del dispositivo.

Con el objetivo de validar la implementación final del sistema se probará este en escenarios reales y se analizarán los datos recogidos para diferentes condiciones y en diferentes tipos de vehículos y trayectos. Finalmente se tratará de evaluar el prototipo final desarrollado con el objetivo de detectar posibles deficiencias o puntos de mejora.

Abstract

One of the main objectives of the Internet of Things is to have the ability to collect a large amount of data in order to analyze and identify what happens in any type of situation or application with the ultimate goal of improving the performance of one or several elements of a system based on the error correction.

This work is based on the application of this new concept of Internet of Things to the field of driving, however the work is not limited to studying only the behavior of the vehicle but it is also studying the driver since this is without no doubt an essential part of this activity. The use of biometric variable sensing devices together with the driver will surely be a differential point of interest in this work. Therefore, the relationship between the vehicle-driver binomial will be analyzed, but a third element that would be the environment which could affect the behavior of both parts of said binomial will be added. The ultimate goal therefore would be to increase driving efficiency and, what is even more important, safety at the wheel, all this based on the generation of sound or visual recommendations that will lead to the implementation of a Smart Driving aid system on board.

To achieve these objectives it is necessary to implement a system that has the main function of collecting data from all the different sources within the vehicle. This system will be the main study object throughout this Final Master's Project. For the design of this system, all the decisions taken that have led to a main system based on a Raspberry Pi type of SBC will be studied and justified. Data will arrive through different means, ranging from wireless connections such as Bluetooth or Wi-Fi up to direct copper connections to the input pins of the device.

In order to validate the final implementation of the system, it will be tested in real scenarios and the collected data will be analyzed for different conditions and in different types of vehicles and routes. Finally, we will try to evaluate the final prototype developed with the aim of detecting possible deficiencies or points of improvement.

Índice General

Índice de figuras

Fig. 6.7.- Resultados obtenidos para el tiempo de trayecto obtenido mediante el adaptador OBD-II conectado al puerto y el prototipo TS “breadboard”. Datos obtenidos durante distintos trayectos del viaje de ida y vuelta entre Asturias y Almería.

Índice de tablas

Lista de acrónimos y abreviaturas

TFM	Trabajo Fin de Máster
IoT	Internet of Things
IUTA	Instituto Universitario de Tecnología Industrial de Asturias
IdC	Internet de las Cosas
OBD	On-Board Diagnostics
CAN	Controller Area Network
VFC	Variabilidad de la Frecuencia Cardiaca
HRV	Heart Rate Variation
SNA	Sistema Nervioso Autónomo
SBC	Single Board Computer
GPU	Graphics Processing Unit
DHT	Digital Humidity and Temperature
LDR	Light Dependent Resistor
GPIO	General Purpose Input Output
EDA	ElectroDermal Activity
PPG	PhotoPlethysmoGraphy
GPS	Global Positioning System

1. Introducción

1.1.- CONTEXTO Y MOTIVACIÓN

Para poder comenzar a hablar del contexto y motivación de este proyecto es estrictamente necesario comenzar hablando del proyecto superior bajo el cual se desarrolla este Trabajo Fin de Máster (TFM).

El proyecto principal surge con el objetivo de buscar una reducción de la peligrosidad al volante o un aumento de la seguridad en carretera al mismo tiempo que se busca una conducción eficiente también en lo referente al ahorro de combustible.

Actualmente se asiste al crecimiento y a la entrada en casi todos los ámbitos del IoT (Internet Of Things) o Internet de las Cosas. Este crecimiento está ocasionando un interés de proporciones aún incalculables por los datos y por la monitorización de prácticamente todos los objetos y sistemas que antes eran sistemas aislados. Los vehículos son uno de esos sistemas que, por un lado facilitan esa monitorización y extracción de datos debido a la gran componente electrónica que han venido adquiriendo durante las últimas décadas y por otro lado presentan un interés enorme en lo que se refiere al control externo debido a que lo que está en juego en todo momento en que un vehículo está funcionando no es ni más ni menos que una o varias vidas humanas.

El proyecto surge en un momento en el que cada vez es más recurrente la temática de conducción autónoma o semi-autónoma. Este hecho no sería algo que fuera en contra de este proyecto sino que lo que aquí se pretende diseñar podría servir como complemento o como analizador de los nuevos modos de conducción y de la relación binomio vehículo-conductor que, pese a todo, aún parece estar bastante lejos de poder llegar a separarse.

Además resulta muy interesante pensar en este sistema como un complemento de ayuda al desarrollo de la conducción autónoma. El sistema podría desempeñar un papel realmente importante en el análisis de los datos recogidos para medir el confort de los pasajeros de un vehículo autónomo y el nivel de estrés que experimentan con el fin de reducir los problemas de aceptabilidad de estos vehículos.

Teniendo todo lo anterior en cuenta parece claro que la finalidad docente y la relación académica de este proyecto con los contenidos y aprendizajes conseguidos a los largo de los años de grado y master es muy alta. La temática de Internet de las Cosas ha sido uno de los temas más recurrentes de las asignaturas del master y esto tiene mucho sentido ya que este concepto engloba las tres ramas en las que se dividía el grado (electrónica, telemática y comunicaciones) y que de algún modo vuelven a unirse en este master.

Este trabajo fin de master se ha realizado en colaboración con la empresa ADN Mobile Solutions y ha recibido financiación en modalidad de beca del Instituto Universitario de Tecnología Industrial de Asturias (IUTA).

1.2.- ESTADO DEL ARTE

Existe un trabajo bastante extenso realizado por este mismo grupo de investigación en el campo del aprendizaje y la evaluación de la eficiencia en la conducción. Los trabajos previos proponen técnicas novedosas para evaluar la eficiencia y la seguridad al volante en base al comportamiento y las acciones del conductor en la carretera.

Existen varios estudios que han llegado a conclusiones de la mejora de la eficiencia y la seguridad al volante mediante dispositivos de ayuda a bordo. [1][2]

Existen trabajos similares realizados por grupos de investigación de otras universidades y países. En todos estos trabajos se trata y se trabaja principalmente con el análisis de la conducción en lo que se refiere al análisis de datos mecánicos del vehículo. Otros trabajos centran su investigación en la influencia de las condiciones ambientales tanto externas como internas de la cabina y tratan de analizar la influencia de los diferentes estados ambientales al rendimiento o a la eficiencia de la conducción. Incluso hay algunos estudios que tienen en cuenta ambos factores. [3]

Otro tipo de documentos y trabajos se centran en el estudio de las variables físicas del conductor de cada a monitorizar su estado durante la conducción con el objetivo principal de alertar de posibles problemas de salud pero únicamente tienen estos factores en cuenta. [4]

Uno de los puntos más novedosos de este proyecto tiene que ver con la integración de diferentes frentes de investigación en un mismo sistema o proyecto. Lo que se busca es la integración del análisis del entorno, las condiciones ambientales, los parámetros electromecánicos del vehículo y lo que es más importante y hace destacar a esta línea de investigación es el relacionar todo esto con el estado físico del conductor.

Y es que el estado físico del conductor es uno de los elementos que más afectan a la conducción y que por tanto más pueden influir en la mejora de la eficiencia y la seguridad.

Hoy en día existen cada vez más dispositivos wearables capaces de monitorizar cada vez más parámetros físicos humanos tal como está siendo analizado de manera muy activa en el campo de e-health.

Otro punto muy importante que no se tiene en cuenta en los trabajos anteriores relacionados es que la conducción es una actividad humana que se aprende y por lo tanto es una actividad cuyo desempeño en términos de eficiencia y de seguridad es

potencialmente mejorable. Es por ello que uno de los enfoques de este proyecto se basa en la idea de la motivación en la conducción con el objetivo de generar aprendizaje.

Por otro lado también hay varios trabajos que se centran en el estudio del entorno y de variables añadidas como las señales de tráfico o las condiciones externas al vehículo que no se han tenido en cuenta en este proyecto por el momento pero que podrían incluirse ya que serían un complemento muy valioso. [4]

2. Objetivos

2.1. OBJETIVOS DEL PROYECTO DE INVESTIGACIÓN COMPLETO

Como ya se ha comentado en apartados anteriores el principal objetivo de este proyecto es el de crear una solución para mejorar la seguridad en la conducción a la vez que se mejora la eficiencia en el consumo de combustible y el estado del conductor al volante.

Se pretende guiar al conductor a través de recomendaciones en tiempo real que le permitan guiar su comportamiento al volante hacia una forma de conducir eficaz y segura. Para llegar a generar estas recomendaciones es necesario establecer el contexto en el que se está produciendo la conducción en cada momento. Con este fin es necesario monitorizar múltiples variables tanto del entorno como del vehículo como de la propia persona que se agruparían en cuatro grandes grupos: estudio de la vía, del ambiente, del vehículo y sobre todo del estado del conductor.

Para conseguir todo esto el sistema que se pretende crear deberá tener una arquitectura a dos niveles tal como se representa en la figura 2.1, con un dispositivo instalado dentro del propio vehículo (sistema embarcado) y un elemento central desplegado en la nube.

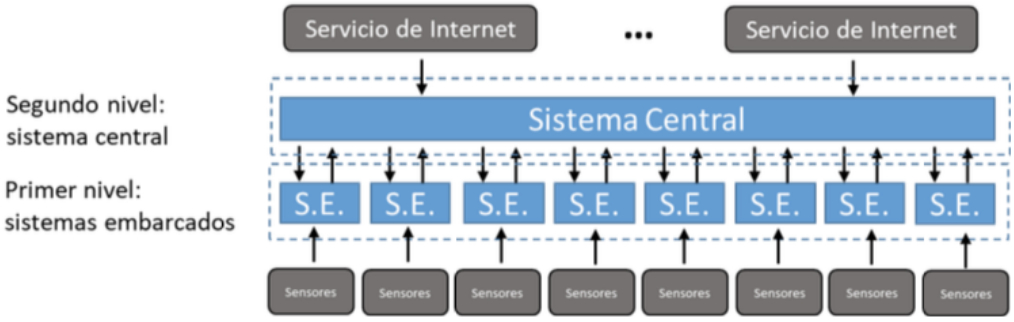


Fig. 2.1.- Esquema de la arquitectura en dos niveles en la que se basa el proyecto global

El dispositivo embarcado se encargará de la interacción con el binomio vehículo-conductor (lectura de sensores y generación de recomendaciones) utilizando para la toma de decisiones una base de conocimiento con situaciones y actuaciones. Esta base de conocimiento será actualizada de periódicamente desde el elemento central que llevará el peso computacional de las tareas analíticas. Dicho elemento tendrá una arquitectura Big-Data y en él se aplicarán técnicas de clustering, autoadaptación, aprendizaje y big data para determinar comportamientos individualizados para los conductores, o para grupos de características similares o generales cuando no sea posible determinar para un conductor individual. El elemento central asimismo se conectará con servicios de internet para complementar la información recogida de los vehículos y completarla con datos sobre tráfico y climatología entre otros.

2.1.1 Objetivos generales del proyecto

El objetivo general del proyecto será crear un sistema autoadaptativo que asista al conductor para lograr una conducción segura y eficiente. Para ello el sistema deberá reunir las siguientes características:

- Recibirá datos de múltiples fuentes y sensores (análisis del contexto). Entre ellas estarán medidas biométricas del conductor (frecuencia cardiaca, tensión arterial, etc.), del vehículo, situación de la cabina, situación del clima y la vía.

- El sistema detectará a través de la monitorización de los sonidos en la cabina patrones lingüísticos que confrontar con los datos biométricos y faciliten la determinación de estados de ánimo.
- Tendrá un funcionamiento inteligente y evolutivo (autoadaptativo). El sistema será capaz de adaptarse a la situación en la que se encuentre y, además, será capaz de aprender a lo largo del tiempo para mejorar sus actuaciones.
- Será mínimamente intrusivo. Utilizará un HUD como elemento de presentación visual, además de audio y vibraciones para interaccionar con el conductor (realidad aumentada). Para la recogida de la información biométrica utilizará una pulsera de actividad o dispositivo similar no intrusivo que no requiera ninguna actuación ni instalación por parte del conductor.

El proyecto se alinea con los retos de la sociedad de transporte inteligente, ecológico e integrado (transporte inteligente) y de energía segura, limpia y eficiente (Smart-Cities) definidos en el Horizonte 2020 de la UE (Incluidos también en la Estrategia Española de Ciencia y Tecnología y de Innovación). Adicionalmente, los temas TIC incluidos en el mismo se encuentran dentro de los recogidos en el H2020 en liderazgo industrial (computación avanzada, big data, Internet del futuro). En este sentido son numerosas las convocatorias en las que se solicitan los típicos presentes en este proyecto como el big-data, cloud computing, smart cyber-physical systems, smart system integration, etc.

Por otro lado, el proyecto reune dos condiciones muy demandadas en la investigación moderna: multidisciplinariedad y cooperación con la empresa.

Partiendo de la motivación y de los objetivos que se persiguen en el proyecto principal, pueden llegar a deducirse las necesidades que requeriría este tipo de aplicación. En este apartado se tratará de explicar sin entrar en demasiado detalle cada una de las partes que deberían componer el sistema final indicando que partes habían de ser desarrolladas y diseñadas en es Trabajo Fin de Máster así como la relación de estas partes con el resto del sistema.

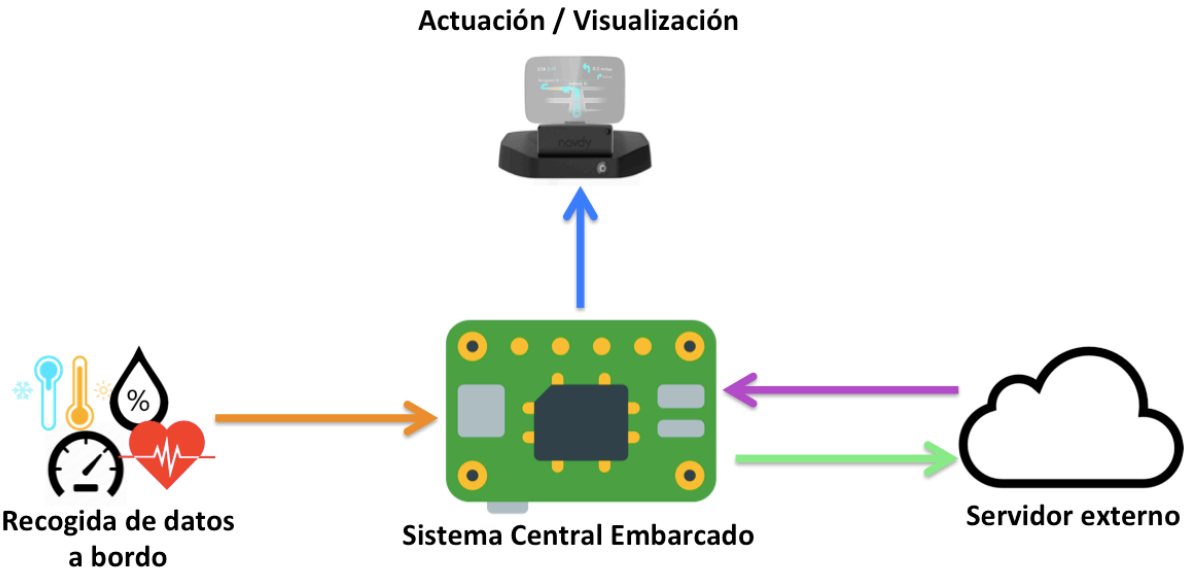


Fig. 2.2.- Esquema de las partes principales de las que estará compuesto el proyecto global [5-11]

En la figura 2.2 se puede ver un esquema resumen de las partes principales del sistema final .Las funciones principales de cada una de estas partes serían las siguientes:

- Sistema central embarcado

Esta parte se refiere al sistema que iría a bordo del vehículo y estaría directamente relacionado con las otras 3 partes del sistema tal como se puede ver en la 2.2.

De forma resumida las principales funciones que ha de desempeñar esta parte del sistema serían, por un lado la recolección en tiempo real de los datos procedentes de la parte de recogida de datos a bordo o lo que es lo mismo de todos los sensores embarcados o a los que es posible acceder desde el propio vehículo.

Por otro lado esta parte ha de establecer una comunicación permanente y bidireccional con el sistema central de inteligencia externo desplegado en la nube. Esto es necesario ya que por una parte el sistema embarcado tiene que enviar todos los datos recogidos a la nube y también ha de recibir las recomendaciones o las

indicaciones de cómo actuar frente a diferentes situaciones. Esta información es suministrada por el servidor externo después de un tratamiento adecuado de los datos y después de un proceso de aprendizaje continuo.

Además de las recomendaciones que vienen expresamente indicadas desde el exterior esta parte del sistema ha de ser capaz de generar sus propias recomendaciones o avisos en tiempo real para tratar de reaccionar a tiempo ante determinados eventos sin necesidad de tener que pasar por el servidor externo.

Por último el sistema embarcado deberá enviar sus órdenes al la parte de actuación y visualización de recomendaciones para que este muestre al conductor lo que convenga en cada momento.

- Recogida de datos a bordo

Esta parte del sistema tiene la función principal de recolectar múltiples tipos de datos mediante una serie de sensores a bordo del vehículo. Además deberá comunicar estos datos al sistema central embarcado para su posterior procesamiento y su utilidad en el funcionamiento del sistema.

En un principio los datos recogidos por este sistema serán de cuatro tipos: datos ambientales, datos de posicionamiento, datos mecánicos del vehículo y datos biométricos del conductor.

- Servidor externo

Esta parte del sistema será la encargada de asumir las tareas más complejas desde el punto de vista computacional. Entre estas tareas se encontrarían:

  - Recuperar información de servicios de internet como podrían ser datos climatológicos o de emergencias y situación de las vías.
  - Recepción de datos procedentes de los sistemas embarcados. Almacenamiento y procesamiento de los mismos.

- Análisis de los datos para determinar nuevas situaciones a tratar a través de la detección de situaciones peligrosas o ineficientes. Determinar la lista de actuaciones para cada una de las situaciones asignando diferentes niveles de urgencia.
- Actualizar base de conocimiento de los sistemas embarcados de forma periódica.

- Actuación/Visualización

Esta parte es la interfaz de comunicación con el conductor. Es uno de los elementos clave del sistema puesto que es algo bastante novedoso en proyectos de esta naturaleza.

En esta parte del sistema es en la que se trabajará en la motivación del conductor minimizando la intrusividad y buscando los mejores mensajes o formas de comunicación con el conductor.

Este sistema se debería caracterizar por algunos rasgos como serían por ejemplo el empleo de las filosofías de *coaching* y *mindfulness* cultivando el refuerzo positivo en lugar de ocuparse únicamente de advertencias y avisos.

Se deberá utilizar la interacción multimodal con el conductor empleando diferentes métodos de comunicación como HUDs, sistemas de sonido o sistemas de vibraciones.

2.2. OBJETIVOS DE ESTE TRABAJO FIN DE MÁSTER

Una vez se ha explicado de manera resumida en qué consistiría cada una de las partes que componen el proyecto completo es necesario especificar que puntos serán objeto de estudio y serán llevados a cabo con objeto de este Trabajo Fin de Máster.

Pues bien, este trabajo se centrará exclusivamente en el desarrollo e implementación de las partes del sistema que se encontrarían a bordo del sistema exceptuando la fase de actuación y visualización que corresponde más bien a una de las fases más avanzadas del proyecto y no se corresponde esa fase con el momento temporal en el que se desarrolla este trabajo.

En cuanto al objetivo de desarrollo de cada una de las partes se debería de desarrollar e implementar prácticamente al 100% la fase de recogida de datos a bordo y una gran parte del funcionamiento del sistema central embarcado, concretamente la parte que se relaciona con la fase de recogida de datos.

Un punto importante a tener en cuenta durante el desarrollo de este proyecto y dado que no es un proyecto aislado sino que está integrado en un proyecto de orden superior, es necesario tener en cuenta en todo momento la compatibilidad del trabajo realizado con el resto de trabajos futuros en el proyecto para permitir la integración final de todas las partes del sistema. Este punto será uno de los factores a tener en cuenta en la fase de diseño y toma de decisiones de las tecnologías y procesos a emplear.

Los objetivos principales de este Trabajo Fin de Master serán por tanto los siguientes:

- Creación de un sistema básico para la recogida de información a bordo
    - Diseño y construcción de equipo hardware
    - Diseño y construcción del software de adquisición y almacenamiento de datos
- Diseño e implementación de la red de interacción de los dispositivos embarcados
- Definición y diseño de relación entre el sistema central embarcado y el resto de partes del sistema.
- Asegurarse de que el sistema diseñado no inhabilite o dificulte la integración del mismo en un sistema de carácter superior habida cuenta de que no se está construyendo un sistema final aislado sino una parte de un sistema más complejo.

- Usabilidad del sistema muy sencilla, cómoda y al alcance de todo tipo de usuarios.

3. Metodología

3.1.. FUNDAMENTOS TEÓRICOS

En este apartado se tratará de explicar algunos de los conceptos más importantes en el desarrollo de este trabajo y cuyo conocimiento es necesario para comprender las necesidades y problemas a los que se pretende dar una respuesta por un lado y por otro lado para evaluar la solución que se propone.

3.1.1.- Internet de las cosas (IoT, Internet of Things)

El Internet de las Cosas (IdC), más comúnmente conocido por su término anglosajón *Internet of Things* (IoT), es un concepto que, si bien no es algo demasiado nuevo, sí que se encuentra en pleno auge actualmente y el potencial que se le atribuye es muy elevado.

Podría decirse que Internet de las Cosas es o será la próxima evolución de Internet. En su definición más simple IoT podría consistir en la interconexión de objetos cotidianos con Internet y con otros objetos y/o personas de su entorno. Toda la interacción que existía entre objetos del día a día con personas pasaría a estar gestionada por máquinas y no sólo eso si no que el nivel de interacción y de "observación" se multiplicaría.

La base de Internet de las Cosas solo tiene sentido si es posible establecer sistemas de recolección de datos o de medida de variables relacionadas con los objetos y a su vez es posible compartir estos datos con una red de nivel superior. La medida y recolección de datos pasa necesariamente por la sensorización, no sólo con el objetivo de medir variables físicas o mecánicas sino de todas los posibles parámetros de interés en relación al estado o al funcionamiento de una determinada "cosa" en todo momento. En cuanto a la interconexión entre objetos e Internet o entre varios objetos es necesario dotar a cada una de estas "cosas"

con conectividad, y más habitualmente será necesaria conectividad inalámbrica, bien para conectar las "cosas" con un nodo central con conexión a la red de nivel superior o bien para establecer esa conexión directamente con el objeto.

El gran auge actualmente de IoT viene precisamente de que la parte de sensorización y conectividad necesaria se ha facilitado mucho y también se ha abaratado y miniaturizado en los últimos años de manera que es mucho más operable ahora.

Todo esto nos lleva a la conclusión de que el número de dispositivos conectados crecerá exponencialmente y ya lo está haciendo, de hecho muchas compañías entre las que está Cisco IBSG prevén un total de 50 mil millones de dispositivos conectados en 2020, lo que sería casi triplicar en tres años los de 20 mil millones dispositivos conectados en 2017 según datos de *Strategy Analytics*.

Actualmente y en el futuro más cercano IoT está compuesto de una multitud de redes diferentes dispersas pero ese no es el objetivo real de IoT, el objetivo es que todas estas pequeñas redes se interconecten en la denominada "Red de Redes" consiguiendo asi una herramienta mucho más poderosa y que es lo que realmente da lugar a la evolución del Internet que conocíamos hasta ahora. Con IoT se consigue que Internet sea sensorial lo cual nos permite ser mucho más eficientes y proactivos.

La manera de evolucionar es y ha sido siempre compartiendo información. Y es que IoT es sinónimo de datos e información lo que lleva directamente a conocimiento y sabiduría. De este modo la importancia de IoT y su evolución va ligada también al desarrollo del *Big Data* que permita el procesamiento y el análisis conveniente sin el cual no tendría sentido la recolección de toda esa información.

En el momento en el que se combine la capacidad de percibir, recoger, transmitir distribuir y analizar datos masivamente con la manera en que las personas procesan esta información se producirá una evolución y un cambio en la manera de vivir de las personas y en la sociedad en general. [12-14]

3.1.2.- *On-Board Diagnostics* (OBD)

Desde aproximadamente los años 80 la electrónica interna de los vehículos no ha dejado de crecer y de ir convirtiéndose en parte cada vez más esencial del vehículo. La electrónica de una vehículo de hoy en día comprende elementos tan diversos y a su vez necesarios como los elevalunas eléctricos, el climatizador, el cierre centralizado, la electrónica de los asientos, la centralita de inyección, el cuadro de instrumentos, los sistemas multimedia o los botones y mandos del volante.

Antiguamente los interconexiones entre los sistemas electrónicos del vehículo se realizaban con cableados simples, sin embargo el aumento del número de dispositivos y componentes electrónicos a conectar hace inviable que todas las conexiones se realicen con cables entre cada uno de los sistemas. De esta idea surgió a principios de los 80 un protocolo de comunicaciones para la automoción centrado en un bus de comunicaciones denominado CAN (*Controller Area Network*) bus. Con este nuevo bus se pasó a tener un único bus (un solo "cable") que recorre todo el vehículo y los diferentes sistemas electrónicos se conectan a él para enviar y recibir mensajes. En realidad en la actualidad hay varios sub-buses para garantizar la rapidez y robustez en las comunicaciones.

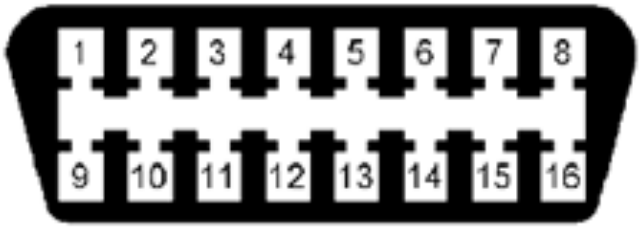
Uno de los usos de este protocolo es para los servicios de diagnosis o de diagnósticos y la recogida de datos del vehículo desde el exterior. De este uso surgen los conectores o puertos OBD (*On-Board Diagnostics*). Estos conectores son la interfaz entre el bus CAN y el exterior con el fin de que se puedan extraer diferentes datos de interés del funcionamiento del vehículo.

A finales de la década de los 80 surge OBD-I con el fin de estandarizar un conector que permitiera leer las cifras de emisiones de cada vehículo pero no todas las marcas lo incorporaron. Por esta razón se decidió mejorar la tecnología y hacerla obligatoria para todos los coches de nueva venta en Estados Unidos a partir de 1996, y ese es el origen de OBD-II.

El OBD-II se encarga de realizar la monitorización de los sistemas funcionales del vehículo y además es capaz de almacenar ciertos códigos de errores referentes a fallos comunes de los vehículos.

Por regulación el conector OBD-II se debe encontrar en el interior de la cabina, a 91 cm del conductor y ha de poder ser manipulado sin ninguna herramienta especial. Normalmente se encuentra situado debajo del volante o detrás del encendedor.

Normalmente los conectores cuentan con 16 pines como se muestra en la figura 3.1 aunque no es estrictamente necesario puesto que dentro de este protocolo existen 5 lenguajes diferentes y esto puede variar el número de conexiones necesarias.



PIN	DESCRIPTION	PIN	DESCRIPTION
1	Vendor Option	9	Vendor Option
2	J1850 Bus +	10	J1850 Bus –
3	Vendor Option	11	Vendor Option
4	Chassis Ground	12	Vendor Option
5	Signal Ground	13	Vendor Option
6	CAN (J-2234) High	14	CAN (J-2234) Low
7	ISO 9141-2 K-Line	15	ISO 9141-2 L-Line
8	Vendor Option	16	Battery Power

OBD-II Connector and Pinout

Fig. 3.1.- Esquemático de los pines que componen un conector OBD-II [15]

Actualmente existen lectores OBD-II universales que pueden conectarse al puerto del vehículo para retransmitir los datos que recogen del bus hacia otros dispositivos como pueden ser un ordenador o un teléfono móvil de forma que se puedan monitorizar personalmente los diferentes parámetros de nuestro vehículo. [16-17]

3.1.3.- Smart Driving (Conducción inteligente)

Con el concepto de "Smart Driving" o conducción inteligente se pretende hacer referencia en este trabajo y en la mayoría de ámbitos de trabajo actuales a la integración en un vehículo de diferentes soluciones principalmente tecnológicas que ayuden a que la conducción se convierta en una actividad más eficiente y sobre todo más segura.

En el concepto de Smart Driving existe por tanto una parte que hace referencia a la conducción eficiente o lo que es lo mismo, a la eficiencia energética y mecánica de la conducción y del uso de un vehículo así como de la eficiencia energética en términos medioambientales. En este ámbito se analizan principalmente los comportamientos y los hábitos de un conductor al volante y la relación de estos con el consumo del vehículo y con los puntos óptimos de trabajo de cada una de las partes que forman parte de la conducción y del funcionamiento del vehículo.

La importancia del análisis de la conducción se hace palpable cuando se observan los beneficios de aplicar la conducción eficiente entre los que estarían principalmente el ahorro económico en combustible y en mantenimiento del vehículo, la mejora en el confort en la conducción (los frenazos o acelerones desaparecerían en la conducción eficiente) o la reducción de la contaminación urbana.

Por otro lado el concepto de Smart Driving hace referencia a la parte encargada de hacer de la conducción conocida algo mucho más seguro y tratar de reducir el número de accidentes en carretera y la peligrosidad al volante, principalmente tratando de anticipar o prevenir las situaciones peligrosas al volante y tratando por tanto de reducir los errores

humanos que son uno de los factores más importante en la consecución de los accidentes hoy en día.

En esta segunda parte del concepto de Smart Driving se hace muy importante la monitorización del estado físico del conductor y de su relación con el ambiente y con el vehículo.

En los últimos años y con el auge del Internet de las Cosas el Smart Driving también se ha empezado a desarrollas de una manera mucho más seria y potente puesto que la facilidad de extraer cada vez una mayor cantidad de datos en tiempo real permite realizar unos análisis de comportamiento y de eficiencia mucho más precisos y mucho más adecuados y fiables.

Existen en la actualidad multitud de artículos e investigaciones realizadas desde diversas instituciones que indican y dan prueba de los beneficios de las ayudas al volante y las recomendaciones para el Smart Driving. [18-20]

3.1.4. Otros conceptos importantes

- **Bluetooth Low Energy:** Hasta la aparición de esta nuevo enfoque, Bluetooth era una tecnología que se basaba en el despliegue de redes inalámbricas de área personal (PAN) para establecer comunicaciones inalámbricas punto a punto. El principal problema es que en muchas ocasiones el uso de la energía no era ni mucho menos óptimo.

  Con el aumento exponencial del número de dispositivos conectados se plantea la optimización de la energía consumida por Bluetooth minimizando el radio de cobertura y la potencia de transmisión. Esta idea se comenzó a pensar en el 2001 se llamó *Wibree* que en el 2007 se integró dentro de las especificaciones de Bluetooth *Low Energy* y en 2010 dentro de la pila de protocolos de Bluetooth 4.0.

Las características de la comunicación Bluetooth pasaron a ser: una velocidad de transmisión de 1 Mbps, conexiones cifradas usando AES de 128 bits y uso de códigos de redundancia para reducir el número de errores.

Este nuevo enfoque de Bluetooth dio lugar a algo tan importante como el desarrollo y el abaratamiento de sensores con conectividad que son una parte esencial del Internet de las Cosas. [21]

- **SQLite:** SQLite es un sistema de gestión de bases de datos relacional . Es un sistema completo de bases de datos que soporta múltiples tablas, índices, triggers y vistas. La principal diferencia con otros sistemas de gestión de bases de datos como PostgreSQL o MySQL es que no necesita un proceso separado que funcione como servidor puesto que trabaja directamente sobre archivos que se encuentran en el propio disco duro.

  Las principales ventajas de SQLite tienen que ver con la velocidad o el rendimiento de las bases de datos y es que SQLite permite realizar operaciones de una forma mucho más eficiente y rápida.

- **Intervalo RR:** El intervalo o intervalos RR son las diferencias de tiempo que transcurren entre dos pulsaciones consecutivas, para ser concretos el tiempo entre una contracción ventricular y la siguiente. Los intervalos RR no ofrecen en bruto más información sobre el estado físico de una persona que la propia frecuencia cardiaca (FC).

  El VFC o Variabilidad de la Frecuencia Cardiaca (HRV en inglés) se basa en aplicar métodos matemáticos a los intervalos RR y con esto se consigue aportar una relación entre la variación del RR y el sistema nervioso autónomo (SNA). El SNA es la parte del sistema nervioso que se encarga de la regulación de los órganos en lo referente a su activación o inhibición. Estas regulaciones están directamente

relacionadas con las necesidades fisiológicas de modo que al hacer deporte por ejemplo se activarán aquellos órganos que protagonicen esta acción.

En ciencias del deporte se puede asociar la regulación del SNA con estados fisiológicos muy concretos, sin embargo durante la realización de actividad física por ejemplo existen órganos que no se ven afectados del mismo modo por la acción de SNA pudiendo generar resultados confusos. Por ejemplo existe una interrelación entre el sistema cardiovascular y el respiratorio de modo que cuando se realiza ejercicio de alta intensidad los pulmones presionan el corazón y alteran el ritmo de este con respecto al marcado por el SNA por lo que la medición de estos intervalos no se corresponde con lo que el SNA marca. Por esta razón, el estudio del VFC y de los intervalos RR es de gran importancia existiendo marcadores que pueden dar información sobre el estado de forma, los niveles de estrés o incluso la propensión o cercanía a sufrir un paro cardiaco. [22]

3.2 DISEÑO BÁSICO DEL SISTEMA GLOBAL

Partiendo de la motivación y de los objetivos que se persiguen en el proyecto, pueden llegar a deducirse las necesidades que requeriría este tipo de aplicación. En primer lugar tal y como se ha expuesto anteriormente el sistema que se pretende implementar se basa en algo similar a lo representado en la figura 3.2 donde se puede observar un sistema de procesamiento de datos central embarcado. A este sistema central llegan datos de dos tipos principalmente, por un lado llegan datos de distintos tipos recogidos a través de diferentes sistemas de recogida de datos embarcados en el vehículo y por otro lado llegan recomendaciones o instrucciones desde un servidor de procesamiento de datos en la nube o externo. Por otro lado se tiene el sistema de recogida de datos que tiene la función de recopilar datos de diferentes tipos dentro del vehículo y enviarlos en tiempo real al sistema central embarcado. La tercera parte del sistema estaría situada en el exterior del vehículo, será lo que denominaremos servidor de procesamiento de datos externo (servidor externo) que

tendrá la función principal de recibir datos del sistema central embarcado, de procesar estos datos en tiempo real o a lo largo del tiempo y generar y enviar diferentes tipos de recomendaciones de vuelta al sistema central embarcado. A partir de los elementos ya comentados se podrá llegar al último elemento presente en el esquema que sería la parte actuadora que se encargaría de mostrar las recomendaciones pertinentes al conductor mediante algún tipo de sistema audiovisual.

3.3. DESCRIPCIÓN DEL SISTEMA PROPUESTO

En este apartado de la memoria se procede a explicar detalladamente la solución propuesta como sistema embarcado que tiene por objeto satisfacer de manera satisfactoria los objetivos marcados en apartados anteriores.

La función básica del sistema a implementar a bordo del vehículo no es otra que la de recoger información. El sistema ha de ser capaz de recoger una gran cantidad de datos por minuto de muy diversa naturaleza. Como ya se ha comentado en apartados anteriores todos estos datos permitirán realizar un análisis de la situación y del contexto en el que se da la conducción en cada momento así como ir estableciendo relaciones entre unos datos y otros.

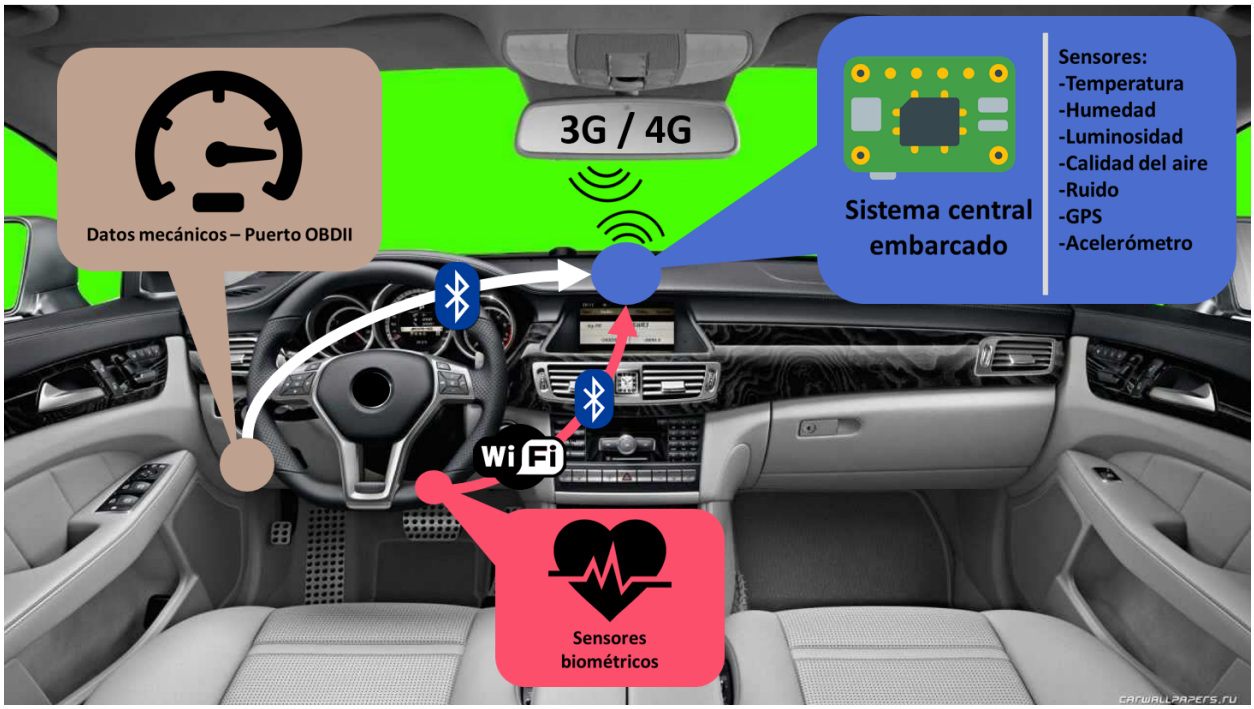


Fig. 3.2.- Esquema de la red que se creará dentro del vehículo y que gira en torno a sistema central embarcado. A este sistema llegan datos desde diferentes fuentes, tal como se muestra en la representación. Los datos del vehículo se obtienen en base a una comunicación Bluetooth mientras que los datos biométricos pueden llegar mediante enlaces Bluetooth o Wi-Fi en función del dispositivos que se emplee para su recolección. El resto de fuentes de datos están conectadas de manera directa por conexiones de cobre con la SBC principal. Esta SBC por último deberá tener la capacidad de comunicarse con un servidor externo mediante un enlace de comunicación móvil 3G o 4G.

Tal y como se puede intuir en el esquema de la figura 2.2 se ha decidido recoger datos de fundamentalmente los siguientes tipos:

- **Datos ambientales:** Con estos datos es posible establecer como afectan a la conducción las condiciones ambientales y también si estás afectan de igual manera a todos los conductores o si por el contrario la susceptibilidad a factores de este tipo varía mucho entre una persona y otra. A continuación se especifican los distintos tipos de datos ambientales que se propone recoger y el motivo de esta decisión:

- ***Temperatura:*** La temperatura puede afectar al conductor y tiene una gran relación demostrada con el sueño.

- ***Humedad:*** El nivel de humedad puede ayudar a la determinación del tiempo en el exterior y resulta interesante su estudio en combinación con la temperatura.

- ***Nivel de luminosidad***: El nivel de luminosidad, además de indicar si es de día o de noche ayuda a determinar pasos por túneles o zonas de mucha luz o reflejos en función de donde se sitúe el sensor. El nivel de luz en cabina es un factor muy importante para la visión del conductor.

- ***Nivel de sonido/ruido:*** Esta magnitud puede dar una idea del volumen de la música o de la radio y por tanto permitirá evaluar el efecto de este en la conducción. Además sería posible llegar a detectar patrones en la voz para extraer de este en la conducción. Además sería posible llegar a detectar patrones en la voz para extraer ciertas palabras y relacionarlas con determinadas situaciones. También sería posible llegar a determinar el número de personas a bordo del vehículo.

- ***Calidad del aire:*** Esta medida podría permitir en primer lugar analizar efectos en la salud de la calidad del aire más a largo plazo pero también podría ser útil para determinar diversas situaciones al volante como sería la de fumar. Puede resultar interesante como mecanismo de detección de calidad de aire mala en cabina y por consiguiente una mejora del confort pudiendo alertar al conductor de este hecho para que actúe en consecuencia.

- **Datos de posicionamiento GPS:** Será necesario recopilar datos de latitud, longitud y altitud GPS. Esta información será muy útil para relacionar los comportamientos del vehículo y del conductor con los diferentes tipos de vías y los emplazamientos de las mismas que pueden ser factores clave en situaciones de estrés tanto de conductor como del propio vehículo.

- **Datos de aceleración:** En este punto se podrían diferenciar de nuevo datos de dos naturalezas diferentes: los datos de aceleraciones correspondientes al vehículo y los correspondientes al conductor. A continuación se explica algo más en detalle en qué la razón de esta diferenciación:
    - Aceleración vehículo: Aquí se recogen datos en tiempo real de las aceleraciones en los 3 ejes que sufre el vehículo. Esta información combinada con la información mecánica resulta de vital interés ya que se pueden deducir situaciones como frenazos o acelerones bruscos y así poder ver como afectan a la generación o no de estrés u otros fenómenos que puedan afectar a la seguridad o a la eficiencia.

    - Aceleración del brazo del conductor: Esta medida puede ser útil para determinar la manera de conducir de una persona. Podría llegar a deducirse por ejemplo si la persona da virajes bruscos o incluso si conduce más o menos tiempo con la mano sobre la palanca de cambio.

    - Rotación de la muñeca del conductor: Este dato puede resultar útil combinado con el dato de aceleración del brazo. Ya que el origen de ambos datos debería ser el mismo dispositivo.

- **Datos biométricos:** Esta parte de los datos es una de las más importantes del sistema puesto que como ya se ha comentado es el punto más diferencial con otro proyectos y permite establecer la relación entre conductor, vehículo y ambiente. En el sistema que se propone se pretende trabajar fundamentalmente con datos de ritmo cardiaco e intervalo RR (tiempo entre latidos). Si fuera

posible se intentaría trabajar con algún dato extra como podría ser la conductividad y temperatura de la piel o el volumen de sangre, el problema de la obtención de estos datos es la dificultad existente en encontrar dispositivos fiables que permitan obtener estos datos de manera poco intrusiva.

- **Datos mecánicos:** Son otros de los datos más relevantes del proyecto puesto que aportan información tal como la velocidad, las revoluciones por minuto o la posición del acelerador. Esta información permite generar informes de eficiencia tanto de consumo como de emisiones y además permite evaluar el modo de conducir de una determinada persona en todo momento para luego relacionarlo con el resto de variables del sistema.

Con todos estos datos se estaría cubriendo una gran parte de los parámetros que potencialmente pueden afectar a la conducción y que permiten establecer las condiciones en las que se está realizando esta actividad.

Conocidos los datos que se pretenden recoger en el sistema propuesto es momento de explicar la manera de extraerlos a bordo.

Una de las decisiones más importantes en el proceso de diseño de este sistema es la elección de la SBC (*Single Board Computer*) a emplear como equipo procesador, recolector y almacenador de datos dentro del vehículo. Esta SBC será la parte principal y central de todos los dispositivos embarcados. Las principales funciones de la SBC serán las siguientes:

- Ser la parte encargada de la ejecución en tiempo real de todos los códigos de adquisición de datos que se creen con este fin. Deberá asegurarse además de su ejecución de manera autónoma en el arranque.
- La adquisición de datos ambientales, datos de GPS y datos de aceleración del vehículo será en base a una conexión directa cableada y/o soldada entre el

puerto GPIO de la SBC y la placa o tarjeta sobre la cual se organicen y conecten los diferentes sensores y dispositivos capaces de obtener los datos.

- Establecer un enlace Bluetooth con un adaptador universal conectado al puerto OBDII del vehículo con el fin de obtener los datos pertinentes del mismo.
- Crear un punto de acceso Wi-Fi accesible a otros dispositivos.
- Crear un servidor UDP en uno o varios puertos para recibir datos de los dispositivos encargados de recoger información biométrica del conductor.
- Poder acceder también a información biométrica de dispositivos *wearables* vía Bluetooth.
- Disponer de puertos USB de salida para futura conectividad con dispositivos de visualización y/o actuación.
- Posibilidad de acceso a conectividad 2G/3G/4G

De este modo y atendiendo a la forma de recepción de datos que se ha expresado en la definición de las funciones principales que tendrá la SBC se puede ahora explicar la manera de generar y transmitir cada uno de los tipos de datos. De nuevo se separará la explicación en función del tipo de datos:

- **Datos mecánicos del vehículo:** Para extraer estos datos se pretende emplear el puerto OBDII del vehículo del que ya se ha hablado anteriormente. Para extraer los datos y enviarlos a la SBC es necesario un adaptador universal que estaría conectado de manera permanente al puerto y establecería una comunicación Bluetooth con la SBC cuando esta así lo ordene.

- **Datos de aceleración del vehículo:** El modo de extraer estos datos es mediante un sensor acelerómetro de 3 ejes que se conecta a la SBC de manera similar a los sensores ambientales. La diferencia es que este dispositivo se comunica mediante una comunicación I2C en lugar de únicamente conectarse a un puerto de entrada/salida. Este componente se situará a ser posible en el mismo circuito electrónico que los sensores ambientales.

- **Datos de sensores ambientales:** Para extraer estos datos es necesario conectar estos sensores a los pines de entrada/salida del puerto GPIO de la propia SBC bien sea mediante cableado o mediante soldadura. Además en algunos casos será necesario añadir diferentes circuitos electrónicos intermedios. De estos puntos se hablará más en detalle en el capítulo de *Implementación.* Se tratará de agrupar todos los sensores de este tipo en un mismo circuito electrónico lo más compacto posible.

- **Datos de posicionamiento GPS:** La extracción de estos datos es similar a los dos tipos de datos previos ya que de nuevo este dispositivo se situará en el mismo circuito electrónico pero en este caso la comunicación con el mismo se realiza mediante el puerto serie de la SBC.

- **Datos biométricos:** Para la extracción de los datos biométricos del conductor será necesario el uso de dispositivos *wearables*. Se usarán fundamentalmente dos, uno de tipo Smart *Watch* y otro de tipo cinta torácica. El primero de ellos extraerá los datos en base al contacto físico del sensor con la muñeca del conductor y enviará estos datos a la SBC a través del punto de acceso Wi-Fi creado por esta al que habrá de conectarse el reloj. La comunicación se basará en un modelo cliente-servidor UDP. En el caso de la cinta torácica la comunicación entre el dispositivo *wearable* y la SBC se produce vía Bluetooth 4.0. Estos dos dispositivos serán los principales pero se hará alguna prueba más con otro tipo de *wearables* que se mostrarán más adelante y que necesitan emplear simultáneamente las comunicaciones vía Wi-Fi y vía Bluetooth.

4. Implementación

En este punto se tratará de explicar en detalle el proceso seguido a lo largo de todo el trabajo, una vez defina la idea del sistema a desarrollar, para crear un prototipo final que desempeñe las funciones que hagan cumplir los objetivos previstos para este trabajo.

Dado que durante el proceso se han ido implementando diferentes versiones con diferente nivel de evolución de una misma parte, la explicación no se limitará a la implementación del prototipo final sino que se hará hincapié en el proceso de mejora continua del sistema y en como se han ido encontrando y corrigiendo errores casi en cada una de las etapas y partes de la implementación del sistema.

A causa de la naturaleza de las diferentes partes del sistema ha sido posible desarrollarlas de una manera relativamente independiente y por tanto será de esa misma forma de la que se explique su implementación. Se diferenciará principalmente en tres partes: parte de tarjeta de sensores de cabina, parte de recogida de datos biométricos y parte de recogida de datos del vehículo.

En este apartado no se entrará demasiado en detalle en la parte de funcionamiento o software del sistema que se tratará más en detalle en el capítulo de “Funcionamiento del sistema”.

Antes de entrar en la implementación de cada parte resulta justificar la selección de un elemento común a todas las partes y elemento central del sistema embarcado como es la SBC. En el siguiente apartado además se explicarán las características principales del modelo seleccionado así como sus fortalezas y debilidades frente a las funciones que debería desempeñar en el sistema.

4.1 SELECCIÓN DE SBC DEL SISTEMA

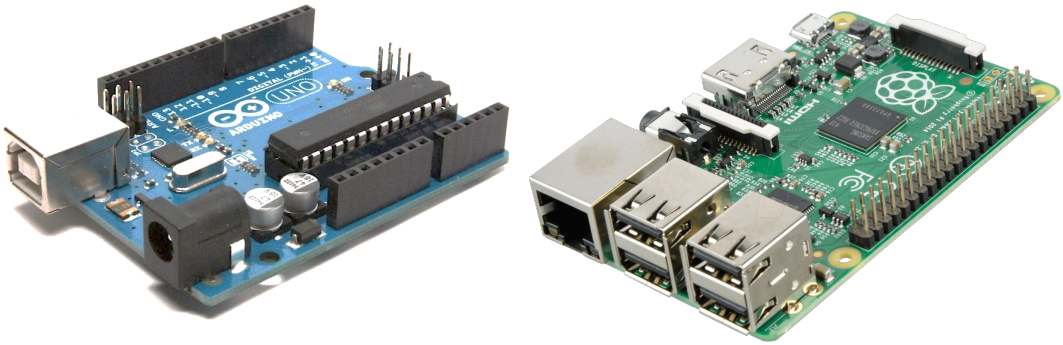
Para seleccionar o elegir una SBC entre las múltiples opciones disponibles en el mercado es necesario tener en cuenta varios factores. En primer lugar la SBC tendrá que desempeñar unas determinadas funciones, las cuales se han especificado en el capítulo 3. Además la SBC o el sistema computador que se seleccione habrá de cumplir unos determinados requisitos en lo referente a sus dimensiones y su peso teniendo en cuenta el sistema dentro del cual va a desempeñar su labor.

Como resumen, las funciones principales a desempeñar por esta SBC y algunas de sus características o requisitos principales serían: Almacenamiento y procesamiento de datos en tiempo real, conectividad Wi-Fi, conectividad Bluetooth, puertos USB disponibles, múltiples pines de entrada/salida para conectividad de sensores analógicos.

En un primer momento las dos opciones principales que se plantearon fueron dos de las más comúnmente utilizadas para el registro de datos con sensores que serían las placas hardware tipo Arduino y Raspberry Pi en sus distintas versiones o modelos. En las imágenes de la figura 4.1 a) y b) pueden verse ejemplos de estas placas.

En una comparativa inicial se pueden apreciar puntos a favor y en contra de la elección de cada una de las familias en detrimento de la otra. Suponiendo que comparamos los modelos más avanzados de cada una de las marcas como sería el Arduino Uno y la Raspberry Pi 3 se puede apreciar inicialmente y tras haber investigado en el uso de cada una de ellas en diversos proyectos basados en recogida de datos que las placas Arduino aportan una mayor facilidad en la adquisición de datos de sensores simples (que son algunos de los que se pretenden emplear) dado que incorporan un conversor A/D interno y pines de entrada analógicos. La Raspberry Pi 3 en este punto complica algo más el proceso

ya que únicamente tiene pines de entrada/salida digitales. Sin embargo las ventajas del Arduino no van mucho más lejos.



a) b)

Fig. 4.1.- Imagen de dos de los principales tipos de SBC comparados y estudiados como posibles alternativas para ser el elemento fundamental del sistema central embarcado. Se representan tanto una SB de tipo Arduino (a) [23] como una SBC de tipo Raspberry Pi (b).

La conectividad es uno de los puntos en los que la placa "básica" de Arduino pierde frente a la Raspberry Pi que ya incorpora conectividad Ethernet, Wi-Fi y Bluetooth. El Arduino Uno puede conseguir estas funcionalidades añadiendo *Shields* que aumentarían el tamaño y complicarían el sistema. Lo mismo ocurriría con las necesidades de puertos USB.

La gran diferencia en realidad entre ambas placas radica en que la Raspberry Pi 3 es un computador en el que es posible instalar diferentes sistemas operativos mientras que el Arduino Uno simplemente no lo es. Esto hace que, entre otras cosas, la Raspberry Pi 3 tenga capacidad multitarea y pueda ejecutar varios procesos o funciones de manera simultanea mientras que el Arduino sólo puede ejecutar un código al mismo tiempo lo cual ralentiza mucho el procesamiento y la ejecución de varias actividades, cosa que será esencial para que el sistema procese y almacene datos en tiempo real procedentes de múltiples fuentes.

De este modo la opción de emplear Arduino o al menos la opción de que sea el único sistema de procesamiento del sistema queda totalmente descartada.

Continuando por la línea de las SBC tipo Raspberry Pi 3 existen en el mercado otros muchos modelos de diferentes fabricantes. Después de haber examinado muchas de ellas se llegó a la conclusión de que la única que en principio podía competir con la Raspberry Pi 3 era la Orange Pi Prime, de la familia de Orange Pi la cual se puede ver en la figura 4.2.

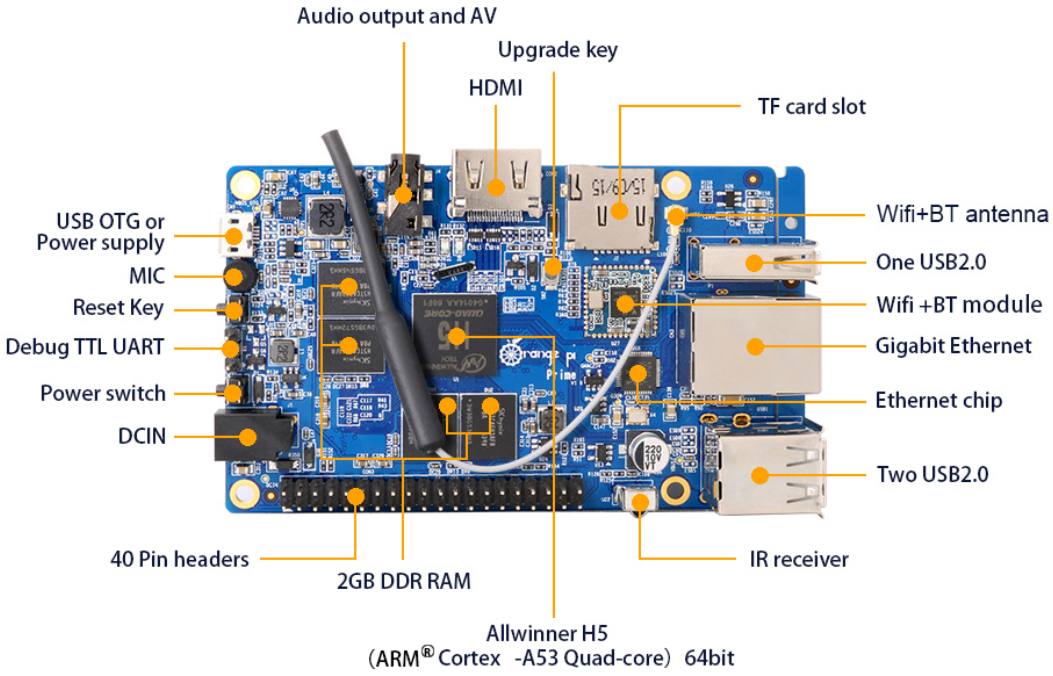


Fig. 4.2.- Representación de la parte superior de una SBC de tipo Orange Pi , concretamente una Orange Pi Primer la cual se ha estudiado como principal alternativa a la Raspberry Pi por su buena compatibilidad con sistemas operativos Android. . [24]

Observando simplemente la figura 4.2 es posible determinar que en lo referente al hardware la Orange Pi Prime parece estar mucho más cerca de la Raspberry Pi 3 de lo que estaba el Arduino Uno. Si se compara parte a parte el hardware de ambas SBC se encuentra que por ejemplo el procesador de ambas tarjetas es similar pero la GPU es

bastante superior en la Orange Pi que en la Raspberry Pi, contando además con una memoria RAM superior, 2GB frente a 1GB de la Raspberry Pi 3.

En lo referente a la conectividad ambas ofrecen las mismas opciones de conectividad inalámbrica, si bien la Raspberry Pi 3 ofrece unas antenas de Bluetooth y Wi-Fi incorporadas mucho más estéticas y de menores dimensiones ya que vienen directamente incorporadas en la PCB. En cuanto a la conexión Ethernet la Orange Pi es de nuevo superior con su tarjeta de red Gigabit Ethernet frente a la Fast Ethernet de la Raspberry Pi 3.

Si se examinan los puerto USB se ve que ambas tienen USB 2.0 pero la Raspberry Pi 3 tiene 4 frente a los 3 de la Orange Pi Prime.

En líneas generales la Orange Pi Prime parece superior, al menos en la parte hardware, a la Raspberry Pi 3 sin embargo las cualidades de la Raspberry Pi 3 no son malas ni mucho menos para el proyecto que se pretende desarrollar.

Si se habla ahora de la parte software de ambas SBC, la Raspberry Pi dispone de uno de los mejores sistemas operativos para mini-ordenadores que existe actualmente como es Raspbian. Raspberry Pi permite además instalar otros sistemas operativos de manera oficial sin embargo ni Ubuntu ni Android se encuentran entre ellos (a excepción del sistema operativo Android Things que sí que tiene soporte oficial de Google pero que dispone aún de funcionalidades extremadamente limitadas). Orange Pi sin embargo permite instalar en sus tarjetas estos dos sistemas operativos además de todas los demás instalables en Raspberry Pi 3 así como los basados en Debian. Orange Pi es conocida por ser una de las SBC que mejor funcionan con Android. Sabiendo todo esto la gran diferencia en el software de ambas SBC radica en el soporte. Raspberry Pi tiene un soporte prácticamente imposible de igualar y un gran apoyo por parte de la comunidad y de los desarrolladores. Además es una de las SBC más utilizadas desde hace unos años con

mucha diferencia en proyectos de este estilo. La Orange Pi sin embargo aunque en teoría es más compatible con algún sistema operativo extra no dispone de soporte en ninguno de ellos.

En principio el sistema a implementar, al menos su parte principal no va a necesitar un sistema operativo Android ni Ubuntu. Además la parte de recolección de datos con sensores es a priori bastante más sencillo y existen muchas más librerías disponibles para python y concretamente para ejecutar en Raspberry Pi con Raspbian.

Todo ello hace que la SBC seleccionada sea la Raspberry Pi 3. De todos modos la Orange Pi es un modelo válido a simple vista y no se descarta su utilización si en un futuro hiciese falta un sistema operativo Android dentro del sistema para interconectar algún dispositivo que sólo funcione en Android. Pero de esto se hablará en el capítulo de conclusiones y líneas futuras.

A continuación se describen las principales características de la SBC seleccionada que sería concretamente la Raspberry Pi 3 modelo B, la cual se muestra en la figura 4.3.

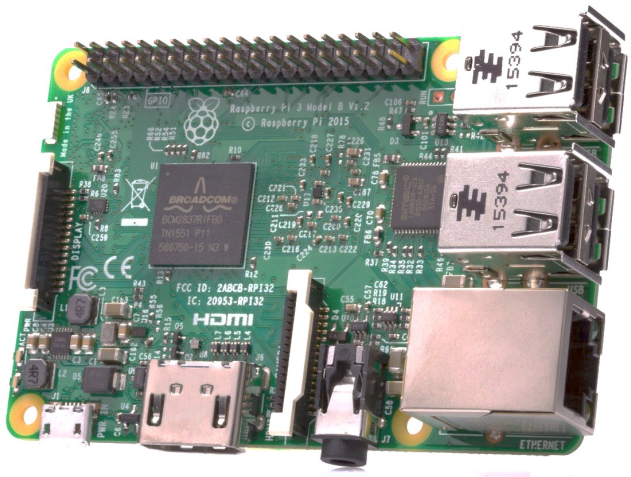


Fig. 4.3.- Raspberry Pi 3 model B como la que se ha empleado a lo largo de las diferentes fases de desarrollo de este trabajo así como en el prototipo final del sistema. [25]

Características Raspberry Pi 3 model B [25]:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN y Bluetooth Low Energy (BLE)
- 40-pin GPIO extendido
- 4 puertos USB 2.0
- Puerto HDMI
- Puerto para cámara Raspberry Pi
- Puerto display DSI para conectar Raspberry Pi 3 con display táctil
- Ranura Micro SD para almacenar datos y sistema operativo
- Puerto Micro USB de alimentación conmutable hasta 2.5 A

4.2 IMPLEMENTACIÓN DE TARJETA DE SENSORES (TS) DE CABINA

En esta parte del sistema se podrían diferenciar 2 prototipos. Cada uno de estos 2 prototipos podrían relacionarse con las fases inicial, intermedia y final del desarrollo de esta parte del sistema y también del sistema en sí. Además la diferencia fundamental en entre el segundo prototipo y el primero radica en el uso de *breadboard* o pcb para el conexionado de los componentes electrónicos. Denominaremos por tanto a los prototipos con los nombres "Prototipo TS *breadboard*" y "Prototipo TS PCB".

Esta parte es la que incorpora más componentes electrónicos y más componentes en general y es por tanto la parte que más evoluciona con el paso del tiempo del proyecto.

Será además una de las que a priori vaya a limitar el tamaño del sistema final y por tanto su evolución tiene gran importancia.

4.2.1 Selección de componentes

En este apartado se tratarán de explicar y razonar las decisiones tomadas en la selección de todos los dispositivos y componentes que formarán parte de los dos prototipos. Las pocas diferencias existentes entre los componentes de cada uno de los prototipos también quedarán convenientemente explicadas en este apartado. Además se tratarán y definirán las principales características de cada uno de estos componentes.

4.2.1.1 Sensores ambientales

Tal como se ha especificado en capítulos anteriores es necesario adquirir sensores capaces de medir las siguientes variables ambientales:

- Temperatura
- Humedad
- Nivel de luminosidad
- Nivel de calidad del aire
- Nivel de sonido/ruido

La finalidad del proyecto no es tener datos extremadamente precisos del ambiente sino que se pretende tener datos en tiempo real para saber el estado ambiental aproximado pudiendo cometerse errores de carácter leve. La tasa de recogida de datos de estos sensores tampoco será por tanto demasiado elevada.

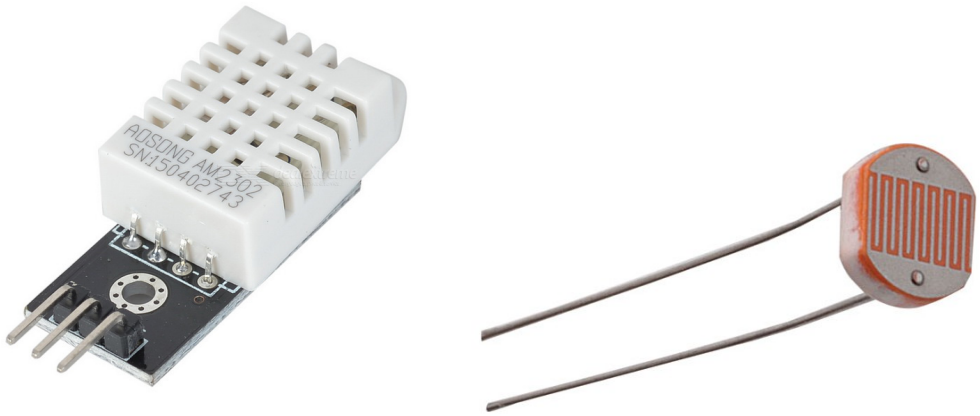
Otro punto a tener en cuenta es que se está creando un primer prototipo y no es necesario por el momento buscar la optimización máxima de variables como el peso o las dimensiones sin haber testeado primero el funcionamiento del sistema completo.

Por todo ello los sensores escogidos serán dentro de estas premisas sensores de uso común en el ámbito de la recogida de datos para proyectos de internet de las cosas y serán por tanto sensores seleccionados primando su precio siempre que se asegure un funcionamiento correcto dentro de los márgenes previstos.

A continuación se especifican los sensores escogidos junto a sus principales especificaciones.

- **Sensor de temperatura y humedad (figura 4.4 (a))**
  - Modelo elegido: DHT-22
  - Rango de humedad: 0 – 100 (%)
  - Rango de temperatura: -40 – 80 (°C)
  - Precisión temperatura: 0.5 °C
  - Precisión humedad: 2%
  - Tasa de muestreo: 0.5 Hz
  - Tensión de alimentación: 3 – 6 (V)
  - Corriente máxima: 2.5 mA
  - Dimensiones: 27 x 59 x 13.5 (mm)

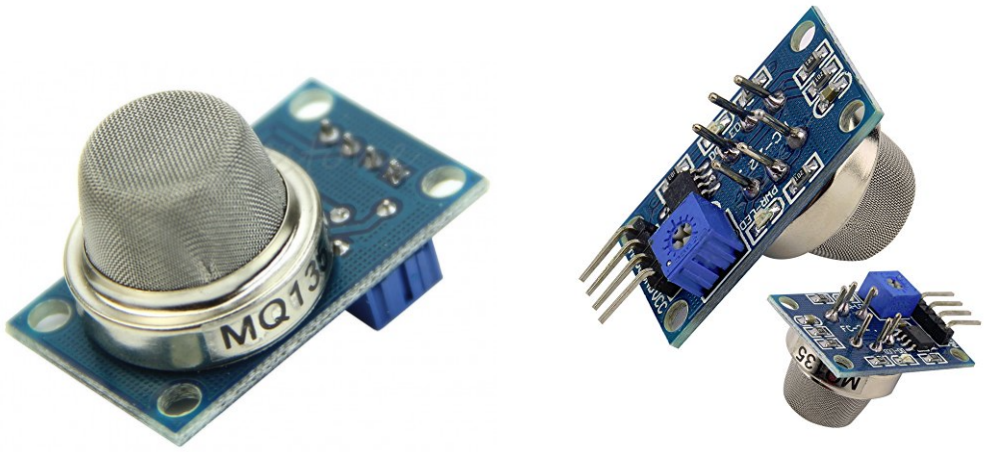
- **Sensor de nivel de luminosidad (figura 4.4 (b))**
  - Fotoresistencia
  - Rango de temperaturas: -30 – 60 (°C)
  - Dimensiones: 5.3 x 4.6 x 3 (mm)



a) b)

Fig. 4.4.- a) Sensor DHT empleado para la adquisición de la temperatura y la humedad del aire [26] y b) LDR empleado para la obtención del nivel de luminosidad. Ambos sensores se han empleado desde el principio y forman parte del prototipo final del sistema. [27]

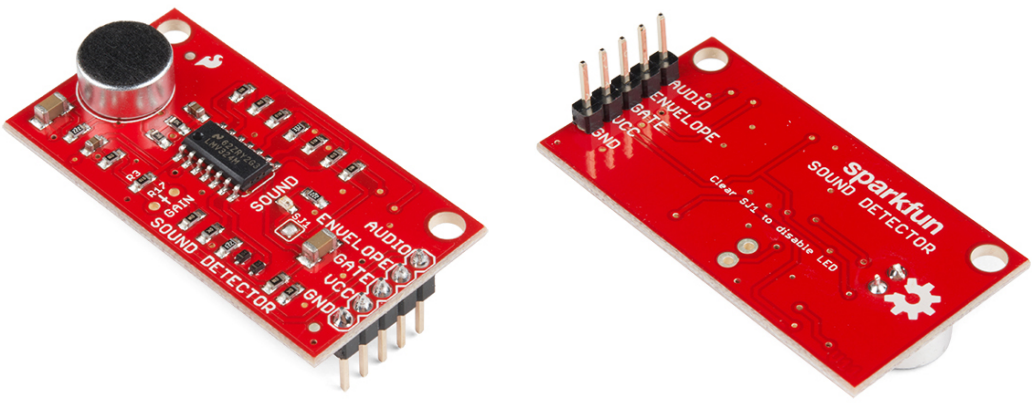
- **Sensor de calidad del aire (figura 4.5)**
  - Modelo MQ-135
  - Gases detectados:
    - Benceno
    - Etanol
    - Humo
    - Propano
    - Monóxido de carbono
    - Gas LPG
    - Metano
    - Amoniaco
    - Óxidos de nitrógeno
  - Tensión de alimentación: 5V
  - Rango de detección: 10-1000 (ppm)
  - Dimensiones: 32 x 22 x 24 (mm)



a) b)

Fig. 4.5.- Parte frontal (a) y trasera (b) del sensor MQ-135 empleado para la obtención de la calidad del aire. Es el módulo que se ha empleado durante todo el desarrollo del prototipo y también en el prototipo final. [28-29]

- **Sensor de nivel de sonido/ruido (figura 4.6) [30]**
  - Modelo: Sparkfun SEN-14262
  - Tensión de alimentación: 3.5 – 5.5 (V)
  - 3 salidas
    - Envolvente: Da una idea de la amplitud del sonido.
    - Audio: Directamente desde el micrófono.
    - Gate: Pulso positivo cuando el sonido sobrepasa un cierto nivel.
  - Sensibilidad/ganancia variables
  - Dimensiones: 25 x 45 x 17 (mm)



a) b)

Fig. 4.6.- Parte frontal (a) y trasera (b) del sensor Sparkfun SEN-14262 empleado para la obtención de la información de nivel de sonido o ruido en el ambiente. Este sensor se ha empleado a partir de la segunda mitad del desarrollo del sistema para mejorar los resultados que se obtenían con sensores de peor calidad. [30]

4.2.1.2 Acelerómetro

El acelerómetro buscado ha de permitir una tasa de lectura sustancialmente superior a al requisito impuesto para los sensores ambientales puesto que las variaciones en la aceleración se producen de una manera mucho más rápida y es información de alto interés el detectar esos cambios.

Se ha decidido emplear un acelerómetro con comunicación I2C. Este bus, a parte de la facilidad de lectura, permite dejar espacio libre en otros pines para sensores que no puedan funcionar con este bus como son los sensores ambientales escogidos.

A continuación se muestran las principales especificaciones del sensor acelerómetro escogido.

- **Acelerómetro (figura 4.7)**
  - Modelo: MMA7455
  - Número de ejes: 3
  - Dimensiones: 25 x 13 x 10 (mm)
  - Soporta 5V/3.3V de entrada
  - Salida digital I2C/SPI
  - Bajo consumo de corriente

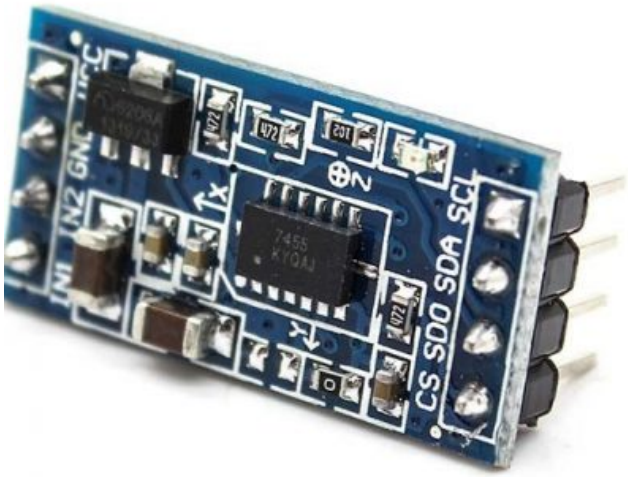


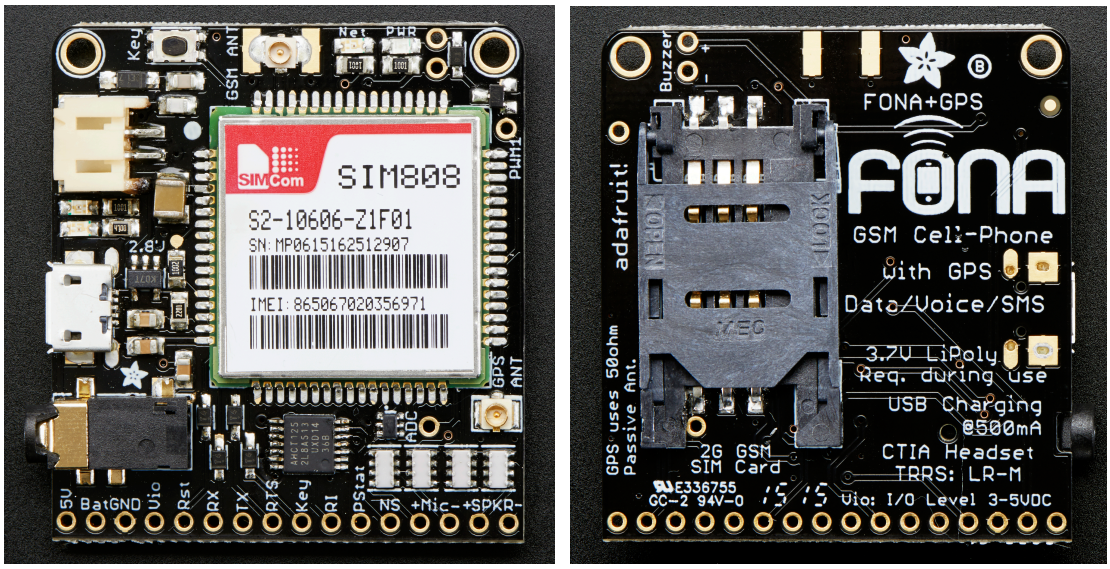
Fig. 4.7.- Módulo acelerómetro MMA7455 con capacidad de medida en los 3 ejes. Es el módulo que se ha empleado durante todo el desarrollo del prototipo y también en el prototipo final. [31]

4.2.1.3 Posicionamiento GPS y conexión GSM

En este punto se busca tener un dispositivo fiable capaz de determinar la posición GPS en 3 ejes, con una fiabilidad elevada y con un error no demasiado elevado. Todas estas condiciones habrán de darse además en un dispositivo que no ocupe un tamaño demasiado elevado que haga aumentar mucho el tamaño final del circuito electrónico.

Por otro lado sería conveniente que la antena de GPS no fuera fija sino que se pudiera sustituir por cualquier otra con diferentes longitudes de cable o de calidades diferentes para que se pudieran corregir posibles errores de nivel de señal dado que inicialmente, se pretende trabajar con una antena básica que no eleve demasiado el precio ya que los módulos en sí son una de las partes más costosa económicamente en el mundo de los sensores más típicos.

Si se tiene en cuenta nuevamente que este sistema tiene el objetivo de formar parte de un sistema mayor, resulta necesario el que muchos de sus aspectos estén abiertos o sean flexibles para interactuar correctamente en el futuro con el resto de partes del sistema final y de la aplicación concreta. Por ello, dentro de la existencia de multitud de módulos GPS se ha buscado un tipo de módulos que aportan una funcionalidad extra que no se empleará dentro de este trabajo pero que sí llegará a poder ser de mucha utilidad en el futuro. Esta funcionalidad es la conectividad por red móvil. Esta funcionalidad permitirá al sistema disponer de conexión en todo momento si así se requiriese y podría ser útil tanto para las conexiones con la nube o servidores externos como para la comprobación o descarga de determinados tipos de datos en tiempo real.



a) b)

Fig. 4.8.- Parte frontal (a) y trasera (b) del módulo GPS/GSM Adafruit FONA 808 que aporta los datos de posicionamiento GPS/GLONASS y permite además establecer una comunicación 2G. Esta segunda funcionalidad aún no se tiene en cuenta en este trabajo pero puede resultar útil en posteriores fases de desarrollo del sistema global. [32]

Ya que el volumen de información con el que se va a trabajar a priori no va a ser elevado sino que la comunicación en principio debería estar reducida a peticiones de datos individuales o de archivos de texto se decidió seleccionar el módulo con conectividad

móvil de segunda generación, GSM. El módulo seleccionado es de una empresa muy fiable y con mucho soporte detrás como es Adafruit Industries y el modelo seleccionado y sus características principales se especifican a continuación.

- Módulo GPS y GSM [32]
  - Modelo: Adafruit FONA 808, celular + GPS breakout
  - Combinación de módulos GSM/GPRS y GPS
  - Tipo de interfaz: UART/USB
  - Alimentación: 2.8 – 5 (V)
  - Dimensiones: 44 x 43 x 8 (mm)
  - Peso: 12.3 (g)
  - Quad-band 850/900/1800/1900 MHz. Conecta con cualquier red GSM global con la incorporación de una SIM 2G.
  - Envía y recibe datos GPRS (TCP/IP, http, etc)
  - Especificaciones GPS:
    - 66 canales de adquisición
    - 22 canales de seguimiento
    - GPS L1 código C/A
    - Chip GPS MT3336 con sensibilidad de -165 dBm
    - Sensibilidad en arranque lento de -147 dBm
    - Tiempo típico de arranque 30s
    - Precisión aproximada de 2.5 m (depende de antena)

Para el uso y montaje del módulo que se explicará en apartados posteriores son necesarios una serie de elementos electrónicos añadidos sin los cuales el sistema no sería capaz de funcionar como se le requiere. A continuación se detallan los elementos necesario para el funcionamiento del módulo:

- Batería de tipo Lipoly o LiIon de 500 mAh o superior
- Para obtener datos de GPS es necesario:
  - Antena GPS con conector final de tipo uFL. Se ha seleccionado inicialmente una antena pasiva como la que se puede ver en la figura 4.9. La cual presenta las siguientes características:
    - Dimensiones 9 x 9 x 6.5 (mm)
    - Ganancia: -2 dBi
    - Peso: 2.4 (g)

- Para el correcto funcionamiento de la conexión GPRS es necesario:
  - Antena GSM con conector final tipo uFL
  - Tarjeta SIM 2G

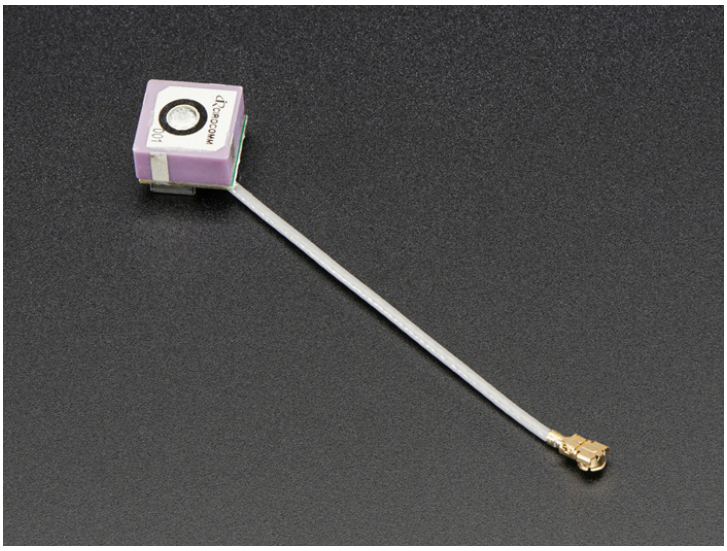


Fig. 4.9.- Antena GPS pasiva empleada en todas las fases de este proyecto. El conector de esta antena es el que necesita para conectarse al módulo FONA 808 y es de tipo uFl. [33]

4.2.1.4 Otros componentes electrónicos

Hasta este punto se han especificado todos los componentes principales para la implementación de la tarjeta de sensores. A parte de estos es necesario el uso de otro tipo de componentes electrónicos que permitan integrar y dar funcionalidad a esta placa. A continuación se resume el resto de elementos empleados para la implementación de esta parte del sistema.

- ***Breadboard*:** Es el elemento que servirá para interconectar todos los componentes electrónicos de manera ordenada en el "prototipo TS *breadboard*". Por las dimensiones finales de este prototipo debido a su alto número de componentes y a las dimensiones de los mismos será necesario emplear un total de 2 *breadboards* de 630 + 2x100 como las que se ven en la figura 4.10. [34]

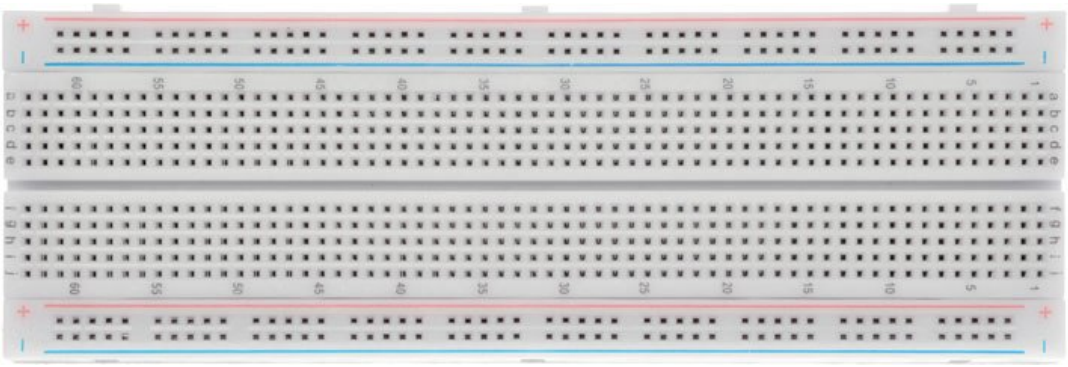


Fig. 4.10.- Ejemplo de *breadboard* similar a la que se ha empleado para las fases iniciales de desarrollo del sistema embarcado. Es un elemento fundamental para el conexionado de los diferentes componentes electrónicos de una manera relativamente segura y ordenada cuando no se dispone de una placa de circuito impreso. [34]

- **Resistencias:** Para el correcto funcionamiento de alguno de los sensores o de los leds del sistema será necesario el uso de algunas resistencias cuyo encapsulado se modificará entre el "prototipo TS *breadboard*" y el "prototipo TS PCB" tal como se verá más adelante.

- **Conversor A/D:** El conversor A/D será un elemento fundamental en el sistema para posibilitar la lectura de los sensores debido a que la Raspberry Pi no dispone de entradas analógicas con las que medir directamente la salida de un sensor y es necesario una conversión analógico-digital previa. El conversor A/D seleccionado por su uso común y su buen resultado además de su bajo coste es el modelo MCP3008 el cual se puede ver en la figura 4.11.

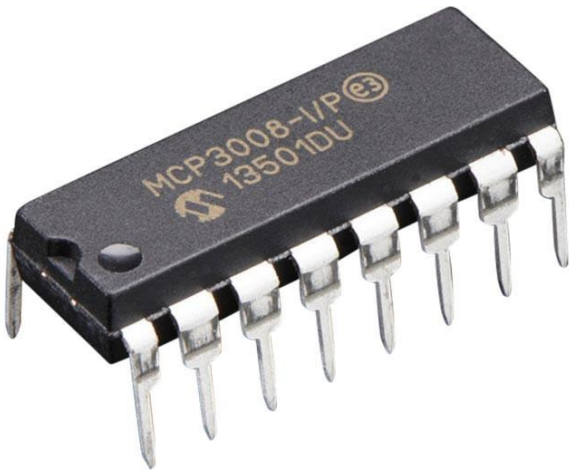
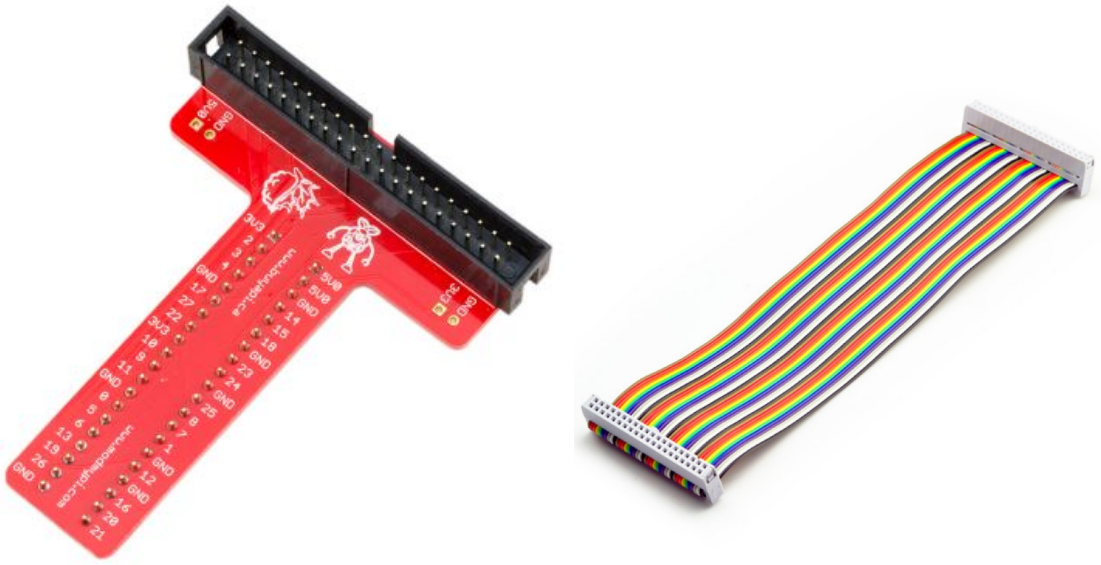


Fig. 4.11.- Conversor A/D MCP 3008 de 8 puertos empleado en los dos prototipos principales diseñados a lo largo del trabajo. Es esencial dado que la Raspberry Pi no dispone de entradas analógicas. [35]

- **Breakout board:** Este elemento (figura 4.12 (a)) permite trasladar una imagen del GPIO de la Raspberry Pi 3 a la *breadboard* con el fin de facilitar el conexionado. Una vez colocada la conexión entre esta y el GPIO de la SBC se realiza mediante una cinta de cables de 40 conexiones como la que se ve en la figura 4.12 (b).



a) b)

Fig. 4.12.- (a) *Breakout board* como el empleado en este proyecto que permite replicar sobre una *breadboard* el GPIO de la Raspberry Pi 3 para permitir conexionados más limpios. (b) Cinta de cables de 40 conexiones similar a la empleada en el proyecto para conectar la *Breakout board* (a) con el GPIO de la Raspberry Pi 3 y así conseguir un conexionado mucho más ordenado y limpio a la vez que seguro en lo referente a evitar desconexiones de cables debido a movimientos del sistema. [36]

- **Cables de conexiones:** En el "prototipo TS *breadboard*" es necesario el uso de cables de diferentes longitudes como los que se ven en la figura 4.12 (b) para realizar todo el conexionado entre los distintos elementos que no van conectados internamente por las conexiones de la *breadboard*..

- **LEDs:** Será necesario también el uso de varios LEDs principalmente para verificación y depuración ajena a la aplicación final pero también como prevención o anticipación ante posibles usos futuros de los mismos. Del mismo modo que ocurría con las resistencias y con los conexionados los LEDs serán de diferente tipo en los dos prototipos implementados.

- **Convertidor de nivel lógico:** Este componente electrónico permite la conversión de un valor de tensión entre dos referencias de voltaje dadas, una superior a la otra. Este elemento será necesario ya para adaptar la tensión de funcionamiento de alguno de los sensores que difiere de la tensión de funcionamiento del conversor A/D. De nuevo el componente en sí encargado de esta función será diferente en el "prototipo TS *breadboard*" y en el "prototipo TS PCB". En la figura 4.13 se puede ver el componente empleado en el "prototipo TS *breadboard*".

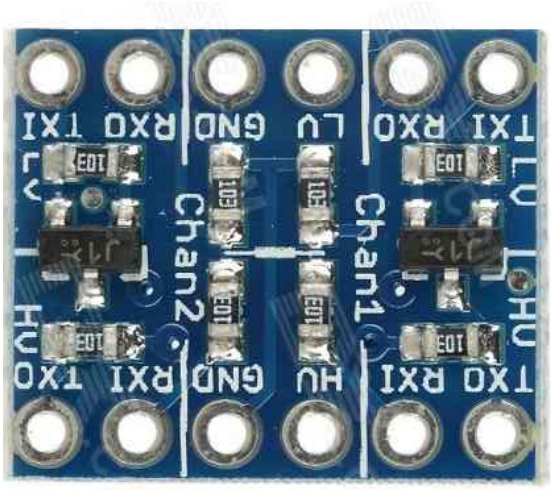


Fig. 4.13.- Parte frontal del convertidor de nivel lógico empleado en el primer prototipo (TS *breadboard*) para realizar los cambios de tensión convenientes para adaptar la tensión de funcionamiento de los sensores a la tensión de referencia del conversor A/D que viene dada por la tensión de funcionamiento de 3.3 V de las entradas digitales de la Raspberry Pi 3. [37]

- **Adaptador toma de corriente vehículo:** Adaptador necesario para pasar de la toma del mechero del vehículo a una toma USB con una corriente necesaria para que el sistema pueda alimentarse totalmente a través de esta toma sin que hay deficiencias en ninguno de los dispositivos conectados. Se ha elegido un adaptador como el que se ve en la figura 4.14 de la marca Aukey que suministra en cada una de sus puertos USB una corriente de 2.4 A lo cual es sobradamente suficiente para el número de dispositivos conectados y también será suficiente si se llega a conectar algún periférico extra como pudiese ser una pantalla.

Fig. 4.14.- Adaptador de toma de corriente del vehículo (mechero) a doble USB con una corriente de 2.4 A que es suficiente para alimentar a la Raspberry Pi y a los diferentes dispositivos que depende de ella.

- **Tarjeta micro SD:** Es una parte muy importante del sistema puesto que es la parte que hace de disco duro. En esta tarjeta es donde se instala toda la parte de software incluido el sistema operativo. Se ha seleccionado una memoria de 32 GB que se ha considerado suficiente para almacenar el sistema operativo que ocupa unos 4 GB y también para dejar espacio disponible de sobra para almacenar grandes cantidades de datos con el objetivo de realizar pruebas donde los datos queden almacenados en local y luego se extraigan al finalizar el trayecto.

Fig. 4.15.- Tarjeta de memoria micro SD con una capacidad de 32 GB. En esta tarjeta irá instalado el sistema operativo Raspbian con el que funcionará la SBC y también hará de disco de almacenamiento de datos. La capacidad de 32 GB es la suficiente para poder realizar almacenamiento de datos de varias semanas sin ningún tipo de problema.

4.2.2 Implementación prototipo TS breadboard

Una vez explicados todos los componentes involucrados en este prototipo y teniendo en cuenta el conexionado electrónico de los mismos, el cual se puede deducir del esquemático presente en el anexo A, no faltaría mucho más que realizar el conexionado físico de todos los componentes sobre las *breadboards* reales.

Este prototipo no se fabricó directamente sino que es fruto de un proceso de evolución necesario al contar con muchos sensores diferentes, con modos de funcionamiento diferentes lo cual requiere la realización de muchas pruebas individuales hasta conseguir el funcionamiento de cada sensor. Después se tuvieron que ir integrando todos los sensores juntos y se hubo que ir asegurando la no influencia o interferencia entre sensores funcionando al mismo tiempo.

Por todo ello resulta interesante presentar de forma muy resumida este proceso de evolución del prototipo sobre la *breadboard*.

En la figura 4.16 se ve una de las etapas más iniciales del prototipo donde se comenzó por la conexión de los principales sensores ambientales de ruido, calidad del aire, luz y temperatura y humedad. Tal como se aprecia en la figura 4.16 en un primer momento se emplearon condensadores para realizar la conversión analógico digital de la que ya hemos hablado pero se vio que esto era bastante poco operativo y muy lento para emplear en un prototipo definitivo.

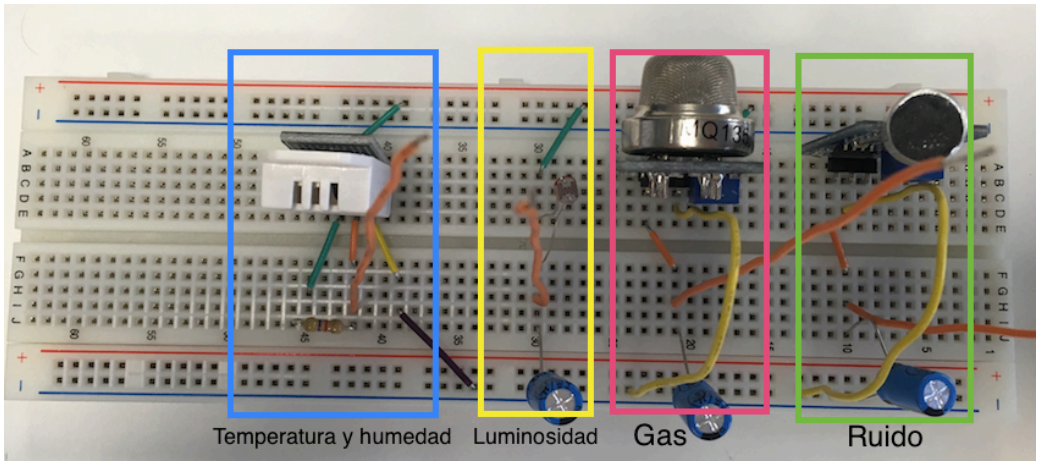
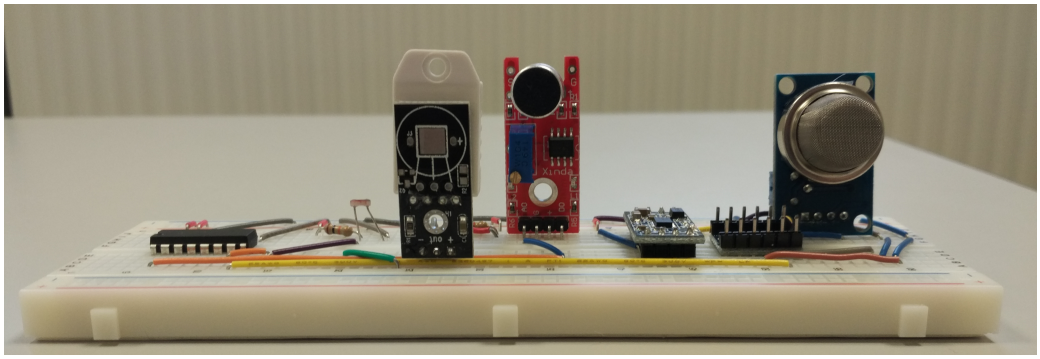


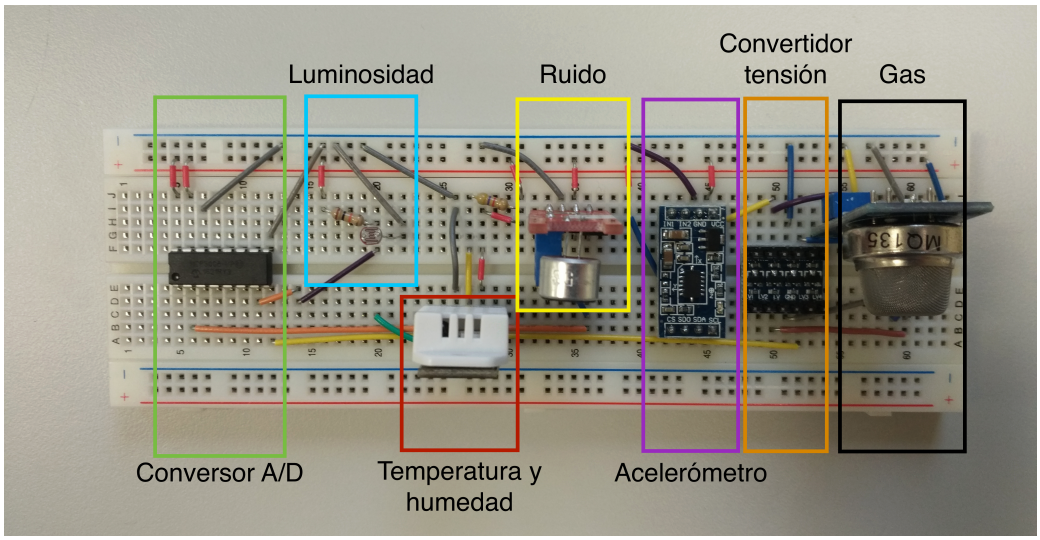
Fig. 4.16.- Prototipo TS *breadboard* en la fase más inicial del desarrollo donde se puede ver que únicamente se cuenta con cuatro sensores ambientales y se emplean condensadores para la conversión analógico-digital.

Se pasa entonces a la siguiente fase en el proceso de evolución la cual se presenta en las figuras 4.17 (a) y (b). En ellas se ve que se ha añadido el conversor A/D, el convertidor de tensión y el acelerómetro. Además se puede apreciar como el sensor de ruido también se ha sustituido debido a que el mostrado anteriormente era únicamente para realizar pruebas ya que era un sensor con salida digital. Otra característica de la evolución que se aprecia claramente es que la disposición de las conexiones se ha realizado de un modo mucho más ordenado y seguro.

Sin embargo a estas alturas de la evolución aún restaba por añadir la parte del módulo GPS y sobre todo la *breakout board* que permitirá la conexión de las *breadboards* con el GPIO de la Raspberry Pi de manera segura para realizar pruebas de campo ya que hasta este punto el conexionado se realizaba con cables cortos simples tal como se aprecia en la figura 4.18, lo cual era suficiente para realizar pruebas pero inviable para un escenario real.

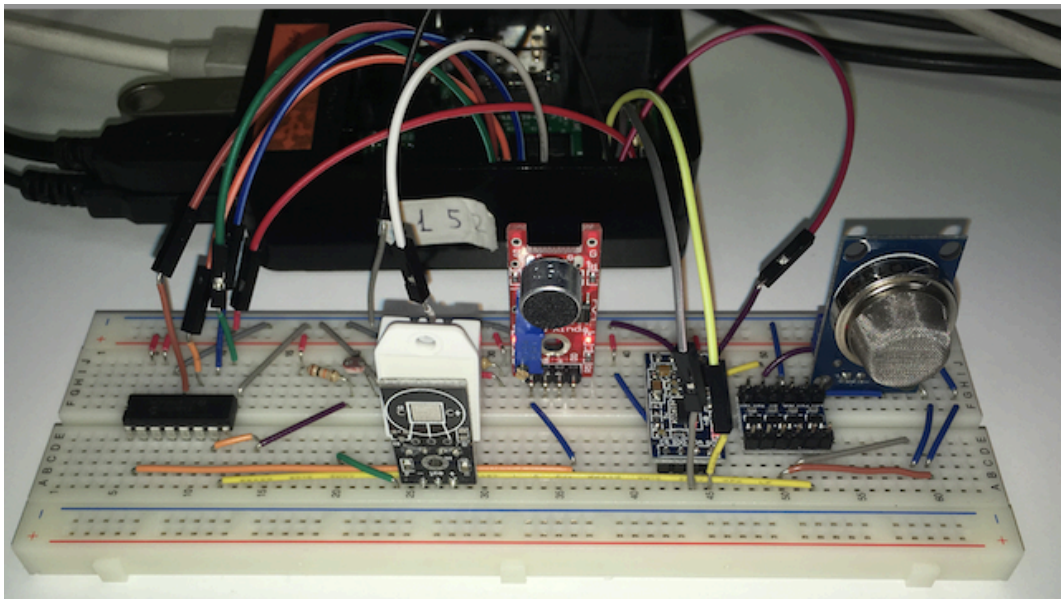


a)

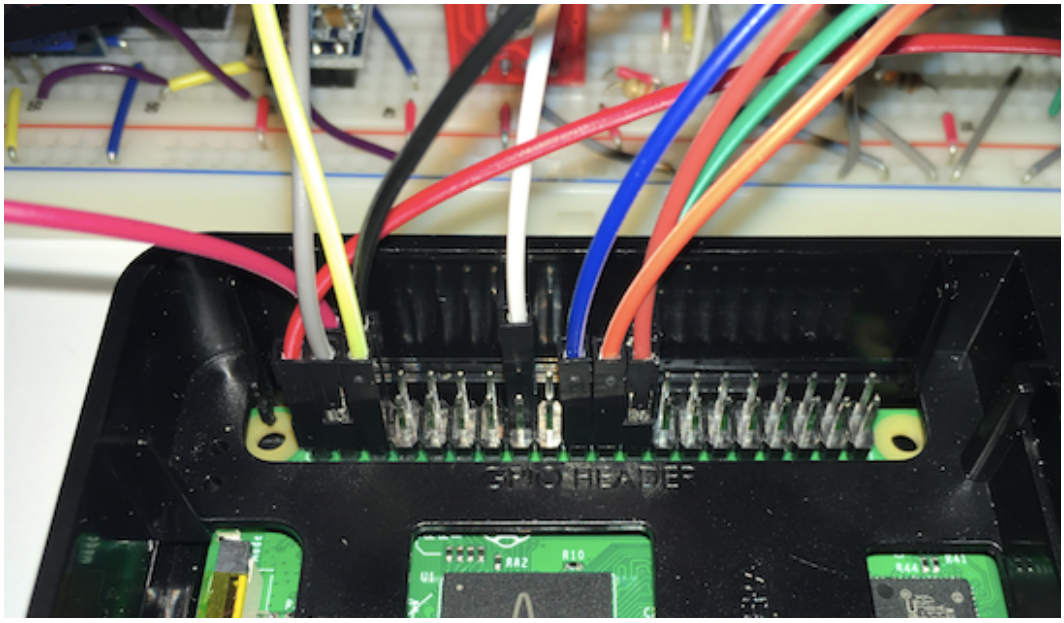


b)

Fig. 4.17.- Prototipo TS *breadboard* en una fase de desarrollo más avanzada que la figura 4.16 donde ya se incluyen todos los sensores ambientales así como el acelerómetro. Además se han sustituido los condensador por un conversor A/D real, concretamente un MCP3008 como el que se representa en la figura 4.11. Se observa el prototipo tanto de perfil (a) como desde arriba e indicando las diferentes partes que lo componen (b).



a)



b)

Fig. 4.18.- Conexionado de la *breadboard* de sensores representada en la figura 4.17 con la Raspberry Pi 3 mediante *jumpers* y cables cortos individuales (a). Puede verse también en b) el detalle del conexionado de estos *jumpers* en el GPIO de la Raspberry Pi 3.

El prototipo definitivo sobre la *breadboard* se puede ver en las figura 4.19 y 4.20. La evolución de este prototipo se hace notable en puntos como son la inclusión del módulo

GPS/GSM con su antena y su batería incorporadas, la inclusión de un nuevo sensor de ruido de mayor calidad que permite obtener la amplitud del sonido, la inclusión de la *breakout board* para el GPIO de la Raspberry Pi 3 y por último la inclusión de una serie de LEDs con funcionalidades principales en el entorno de desarrollo y pruebas pero disponibles para cualquier otra funcionalidad futura.

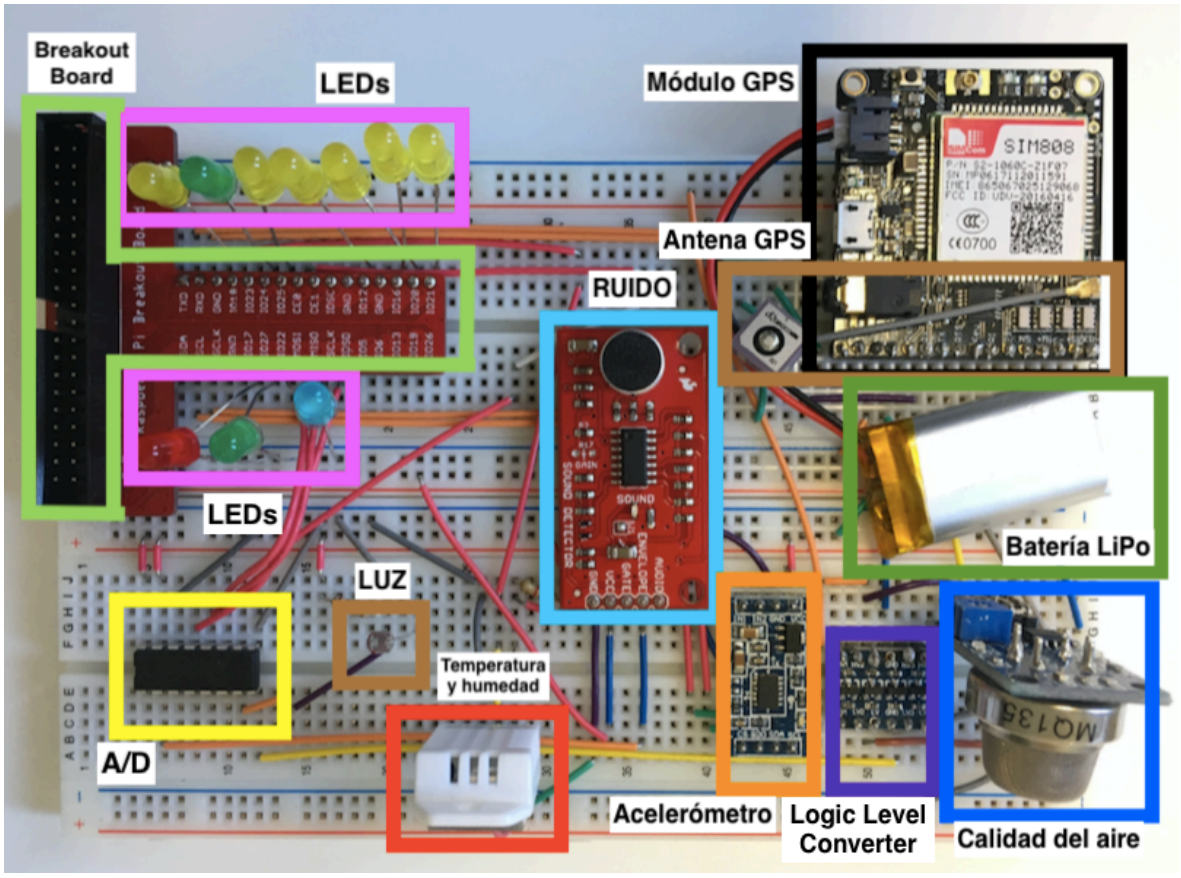


Fig. 4.19.- Prototipo TS *breadboard* final visto desde arriba. En la imagen se describen todos los componentes principales que componen este sistema. Únicamente faltaría la cinta de conexiones que conectan directamente la *breakout board* con el GPIO de la Raspberry Pi 3.

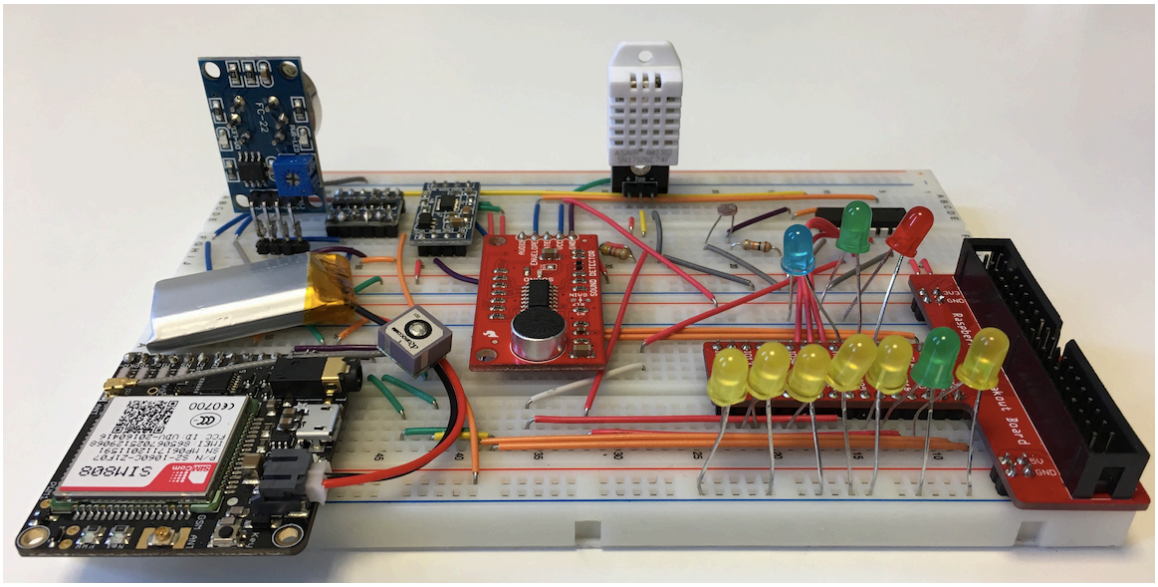


Fig. 4.20.- Prototipo TS *breadboard* final visto desde un lateral. En la imagen se han incluido ya todos los componentes y únicamente faltaría la cinta de conexiones que conectan directamente la *breakout board* con el GPIO de la Raspberry Pi 3.

En la figura 4.21 se puede ver como sería el prototipo final en *breadboard* colocado dentro de una caja de cartón normal y conectado convenientemente a la Raspberry Pi. Esta disposición permitiría realizar pruebas de campo y de hecho con este prototipo es con el que se han recogida la mayor parte de los datos reales a día de hoy.

Una vez definido el prototipo se ve que este presenta varias deficiencias. La primera de ellas es el tamaño. Las dimensiones de prototipo son de 19 x 19 x 9 (cm) por tanto no es nada cómodo de utilizar. Sin embargo no son las dimensiones la principal preocupación de usar este sistema, el principal problema viene de las conexiones de los componentes. El prototipo está ideado para ir en algún lugar dentro de la cabina de un vehículo o lo que es lo mismo, el sistema tendrá que soportar vibraciones continuas e incluso bruscas aceleraciones o frenazos. Tal como está el prototipo hasta este punto se ofrece muy poca seguridad y fiabilidad en las medidas ya que en cualquier momento cualquiera de los componentes electrónicos podrían desconectarse o cortocircuitarse y podrían dar al traste con una jornada de pruebas de varias horas.

Fig. 4.21.- Prototipo TS *breadboard* final ya incluyendo el conexionado con la Raspberry Pi 3. En la imagen se incluye ya también un encapsulado provisional de cartón que es el que se ha empleado en las pruebas iniciales para evitar movimientos de la *breadboard* y proteger algo al sistema.

Otro punto en contra de este sistema es la capacidad que ofrece a los implementadores de replicación. El proyecto en el que se enmarca este sistema buscará recoger la mayor cantidad de datos posibles y realizar la mayor cantidad de pruebas de campo en el menor tiempo. Es por ello que probablemente se quiera replicar este sistema para disponer de varios prototipos iguales con los que poder realizar pruebas simultáneamente.

Por todas las razones expuestas se decidió implementar un segundo prototipo que corrigiera alguna de las deficiencias explicadas. La implementación de este segundo prototipo se explica en el siguiente apartado de la memoria.

4.2.3 Implementación prototipo TS PCB

Tal como se acaba de explicar en el punto anterior, el prototipo basado en *breadboards* no era lo suficientemente adecuado para el desarrollo de pruebas futuras. Por esta razón se decidió tratar de trasladar la parte más incómoda y problemática del prototipo a algo como es una placa de circuito impreso o PCB (Printed Circuit Board) que es algo mucho más seguro y manejable además de ofrecer una reducción significativa de variables tan importantes como el tamaño y el peso.

En este proceso es en el que prioritariamente se trabajó y colaboró proactivamente con la empresa ADN Mobile Solutions.

El programa empleado para el diseño de la PCB ha sido KiCad. KiCad es un entorno de software empleado para el diseño de circuitos electrónicos impresos entre otros. Es un software muy intuitivo y además muy flexible y adaptable disponiendo de un gran número de componentes en sus librerías y pudiendo crear componentes nuevos en función de las necesidades del diseño y del circuito en cuestión.

Los pasos seguidos para el diseño y fabricación de la PCB con KiCad son muy similares a los que previsiblemente se seguirían en otros programa de diseño de este estilo:

1. Diseño de circuito esquemático
2. Verificación de reglas eléctricas en esquemático
3. Asociación de componentes a huellas

4. Generar listado de redes
5. Generar BOM (Bill Of Materials)
6. Diseño de placa de circuito impreso
7. Verificar reglas de diseño
8. Generación de Gerbers
9. Verificación de errores
10. Envío de Gerbers para fabricación

Una vez recibida la PCB o PCBs diseñada se han de seguir los siguientes pasos para la finalización del prototipo:

1. Soldadura de conectores y demás componentes en PCB
2. Verificación de correcto funcionamiento de la placa
3. Diseño y fabricación de encapsulado

Se tratará ahora por tanto de explicar los puntos más importantes de este proceso de fabricación del segundo prototipo

4.2.3.1. Diseño e implementación de PCB

1. Diseño de circuito esquemático

La primer parte del sistema consistió en trasladar el esquema del circuito que se tenía montado ya en el prototipo primero al programa y editor de esquemáticos de KiCad. En la figura 4.22 se aprecia el esquemático final diseñado que es el mismo que se puede examinar de manera más detallada en el anexo A.

A la hora de trasladar el circuito del prototipo en *breadboard* al programa se trató de optimizar el diseño teniendo en cuenta que en la PCB final se podrían sustituir algunos de los componentes por sus equivalentes miniaturizados con encapsulado SMD que podrían

aportar un ahorro significativo de espacio. Esto se verá mejor cuando se asignen los componentes concretos a cada uno de los elementos o huellas del esquemático.

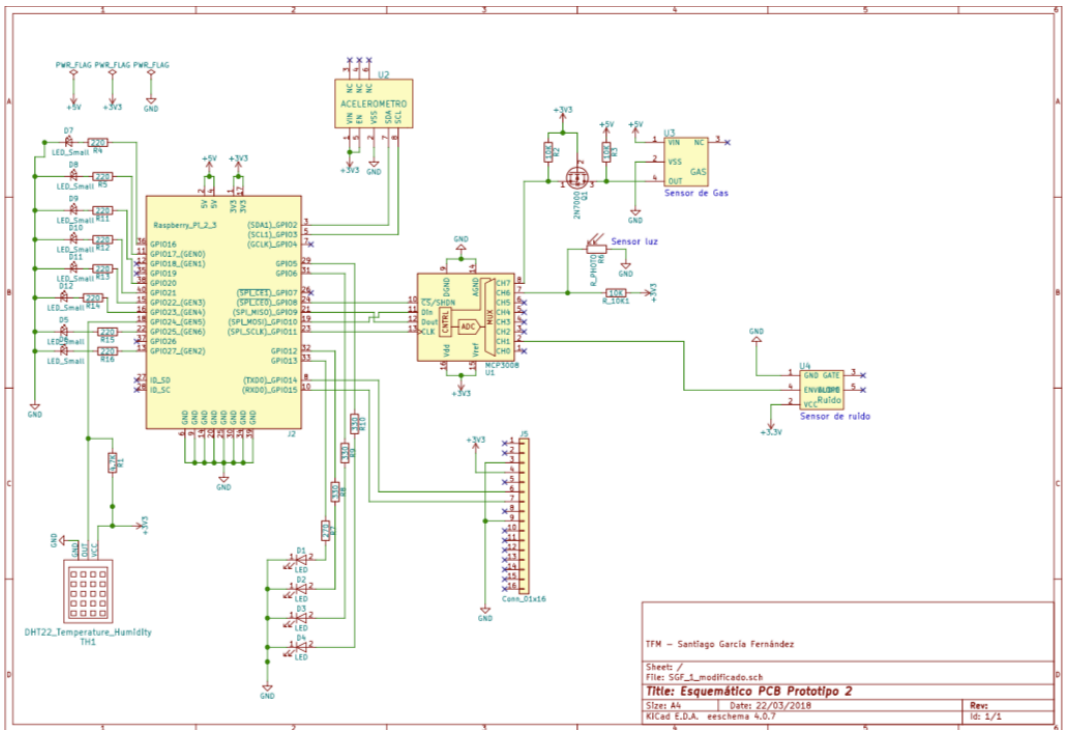


Fig. 4.22.- Esquemático realizado con herramienta software KiCad para el diseño de la placa de circuito impreso del prototipo TS PCB. Mismo esquemático que el representado en detalle en el anexo A.

Como reseña sobre el manejo de la herramienta cabe desatacar que la mayoría de componentes empleados para diseñar el esquemático ya estaban disponibles en las librerías propias del programa. Sin embargo algunos componentes menos comunes como los sensores tuvieron que ser importados pero el proceso de importación resultó bastante intuitivo.

2. Verificación de reglas eléctricas en esquemático

Este paso resulta esencial para comprobar si existe algún error en el diseño del esquemático y permite ir corrigiéndolos todos antes de pasar a siguientes pasos.

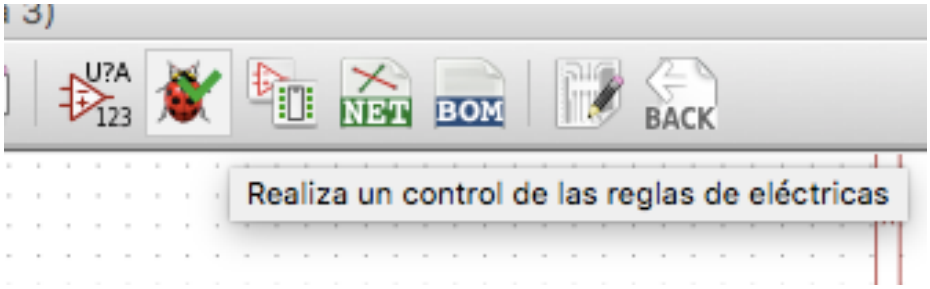


Fig. 4.23.- Botones de KiCad para ir realizando las diferentes fases del diseño. Resaltado el botón de control y verificación de reglas eléctricas en el esquemático.

En la figura 4.23 se representa el icono que hace esta función en el software KiCad.

3. Asociación de componentes a huellas

Este paso resulta muy importante para el paso posterior del esquemático a circuito impreso. En este punto indicamos que huella o que forma tendrá dentro del circuito impreso cada uno de los componentes que forman el esquemático.

La asignación de huellas componentes para este circuito sería la que se especifica en la figura 4.24. Casi la totalidad de las huellas se encontraban disponibles en el programa tras haber incorporado las correspondientes librerías extra para los sensores. Casi la totalidad porque del acelerómetro no fue posible encontrar una librería específica en KiCad para el modelo que se emplea en este proyecto. Por ello ha sido necesario explorar otra de las utilidades de este programa como es el editor y creador de huellas. Se ha tenido por tanto que tomar las medidas correspondientes sobre el componente real disponible en el laboratorio y trasladarlas al programa para diseñar la huella que permita que una vez fabricada la PCB el componente encaje perfectamente en el hueco reservado para él. En la figura 4.25 se puede ver la huella diseñada.

1	D1	LED	LEDs:LED_D5.0mm
2	D2	LED	LEDs:LED_D5.0mm
3	D3	LED	LEDs:LED_D5.0mm
4	D4	LED	LEDs:LED_D5.0mm
5	D5	LED_Small	LEDs:LED_0805
6	D6	LED_Small	LEDs:LED_0805
7	D7	LED_Small	LEDs:LED_0805
8	D8	LED_Small	LEDs:LED_0805
9	D9	LED_Small	LEDs:LED_0805
10	D10	LED_Small	LEDs:LED_0805
11	D11	LED_Small	LEDs:LED_0805
12	D12	LED_Small	LEDs:LED_0805
13	J2	Raspberry_Pi_2_3	Pin_Headers:Pin_Header_Straight_2x20_Pitch2.54mm
14	J5	Conn_01x16	Pin_Headers:Pin_Header_Straight_1x16_Pitch2.54mm
15	Q1	2N7000	TO_SOT_Packages_THT:TO-92_Molded_Narrow
16	R1	4.7K	Resistors_SMD:R_0805
17	R2	10K	Resistors_SMD:R_0805
18	R3	10K	Resistors_SMD:R_0805
19	R4	220	Resistors_SMD:R_0805
20	R5	220	Resistors_SMD:R_0805
21	R6	R_PHOTO	Pin_Headers:Pin_Header_Straight_1x02_Pitch2.54mm
22	R7	270	Resistors_SMD:R_0805
23	R8	330	Resistors_SMD:R_0805
24	R9	330	Resistors_SMD:R_0805
25	R10	330	Resistors_SMD:R_0805
26	R11	220	Resistors_SMD:R_0805
27	R12	220	Resistors_SMD:R_0805
28	R13	220	Resistors_SMD:R_0805
29	R14	220	Resistors_SMD:R_0805
30	R15	220	Resistors_SMD:R_0805
31	R16	220	Resistors_SMD:R_0805
32	R_10K1	10K	Resistors_SMD:R_0805
33	TH1	DHT22_Temperature_Humidity	Pin_Headers:Pin_Header_Straight_1x03_Pitch2.54mm
34	U1	MCP3008	Housings_DIP:DIP-16_W7.62mm
35	U2	ACELEROMETRO	ACEL_foot:ACELEROMETRO
36	U3	GAS	Pin_Headers:Pin_Header_Straight_1x04_Pitch2.54mm
37	U4	Ruido	Pin_Headers:Pin_Header_Straight_1x05_Pitch2.54mm

Fig. 4.24.- Ficha resumen de la asignación de huellas a los diferentes componentes que forman el esquemático de la figura 4.22. Alguna de las huellas han surgido de la importación de librerías propias de determinadas familias de componentes y alguna otra como la del acelerómetro ha tenido que ser diseñada y creada manualmente con el editor de huellas.

4. Generar listado de redes

Este paso se hace automáticamente a través del programa donde simplemente se selecciona generar el listado de redes para el programa que se vaya a emplear para la colocación de las diferentes huellas en la PCB que en este caso será el “Pcbnew” que es la propia herramienta de KiCad para el trabajo con placas de circuito impreso.

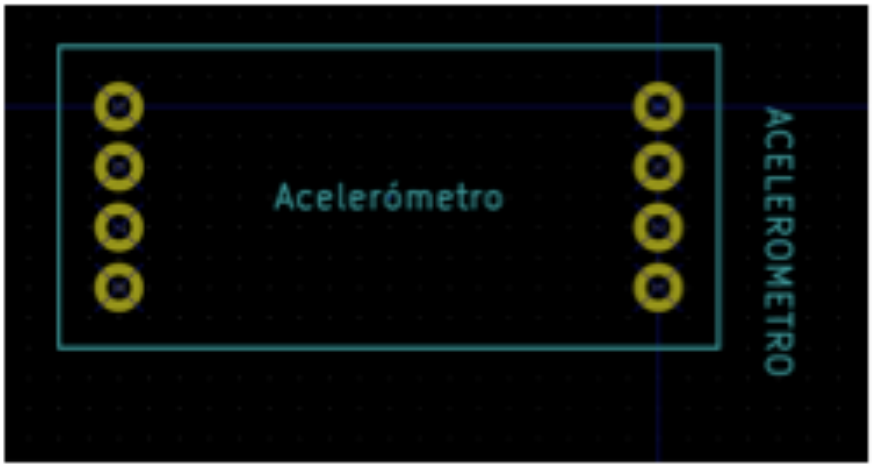


Fig. 4.25.- Huella del acelerómetro diseñada manualmente en KiCad gracias a la herramienta de edición y creación de huellas. La creación de esta huella ha sido necesaria al no encontrar una librería que contara con la huella de este modelo de acelerómetro.

5. Generar BOM (Bill Of Materials)

El siguiente paso en el proceso, no cronológicamente de manera obligada, es el de la redacción de la lista de materiales o componentes necesarios para montar la placa real una vez esta esté fabricada. Se creó por tanto un BOM donde vienen especificados estos materiales sabiendo que la mayoría de ellos pueden ser sustituidos por otros equivalentes de otro fabricante o ligeramente diferentes siempre y cuando cumplan la misma función.

6. Diseño de placa de circuito impreso

En este punto se trabajó con la herramienta Pcbnew de KiCad. El primer paso para el diseño de la PCB es importar el listado de redes que se había generado en el paso cuarto de este proceso.

Una vez importado el listado se deben de colocar los componentes sobre la superficie que se defina para la totalidad de la PCB de la manera más conveniente teniendo en cuenta

la optimización del espacio, la aplicación posterior de la placa o las dimensiones de los componentes que irán insertados sobre la misma entre otros aspectos.

Cuando ya se tienen los componentes con una primera posición definida en el espacio de la placa es hora de comenzar con el conexionado de los nodos con la ayuda de las conexiones que el programa indica gracias al listado de redes que se ha importado. En este punto por tanto es necesario definir los anchos de cada tipo de pista. En este caso tendremos cuatro tipos de pistas a saber pistas de masa, pistas de alimentación a 3.3 V, pistas de alimentación a 5 V y pistas de conexión normales. Los anchos de línea seleccionados nos han sido demasiado exigentes puesto que no se precisa en una miniaturización tan elevada al tener muchos componentes y dispositivos que irán insertados y que por ellos mismos ocuparán un gran espacio. Se han diferenciado 2 anchos de pista diferentes, un ancho de 0.889 mm (35 mils) para los dos tipos de pistas de alimentación y un ancho de pista de 0.305 mm (12 mils) para el resto.

Completada la interconexión de todos los nodos resulta útil definir un plano de masa en el espacio de PCB vacío lo que además reducirá sustancialmente el número de líneas de conexiones de masa y resultará más eficaz a la par que estético.

Por último se crear los orificios para la posible tornillería en las esquenas y se coloca toda la serigrafía que se pretenda mostrar en la placa real, tanto la serigrafía propia de los componentes indicando a cual corresponde cada huella como la serigrafía informativa del trabajo para el que se ha realizado la PCB o las entidades colaboradoras en el proyecto.

7. Verificar reglas de diseño

Este paso no es necesariamente el último sino que se puede ir aplicando el comprobador de errores a medida que se van realizando modificaciones en el diseño de la PCB. Lo que si es necesario es que el diseño que sea el definitivo cumpla las reglas de diseño y paso el chequeo.

Una vez llegados a este punto se tenía ya el diseño de la PCB final que sería el que se representa en la figura 4.26.

8. Generación de Gerbers

Con el diseño de la PCB finalizado se han de generar los archivos Gerber que son los archivos que el fabricante necesita para poder fabricar la PCB que se ha diseñado.

9. Verificación de errores [38]

Un paso muy aconsejable cuando se crea una PCB como esta es el chequeo de posibles errores más allá de las reglas de diseño que puedan dar lugar al fabricante a complicaciones o a fallos a la hora de fabricar la placa real.

Muchos fabricantes incorporan en su página web una herramienta que permite subir los Gerbers y esta examina todos los datos con el fin de buscar este tipo de errores de los que se está hablando. En este caso se ha decidido emplear la herramienta online de un fabricante que no es el que se va a contratar para la fabricación de la PCB pero que ofrece este servicio de forma gratuita en su sitio web y es una herramienta muy útil. Se está hablando del fabricante belga Eurocircuits.

En la web de Eurocircuits existe un apartado a la hora de hacer un pedido que se llama "Analyse your data". Una vez se realiza el registrado de manera gratuita y se accede a este apartado basta con poner un nombre al archivo y cargar en la página un archivo comprimido con los Gerbers de la PCB diseñada.

Una vez cargados se presenta la una página donde se indica tal como se aprecia en la figura 4.27 que los archivos están siendo analizados.

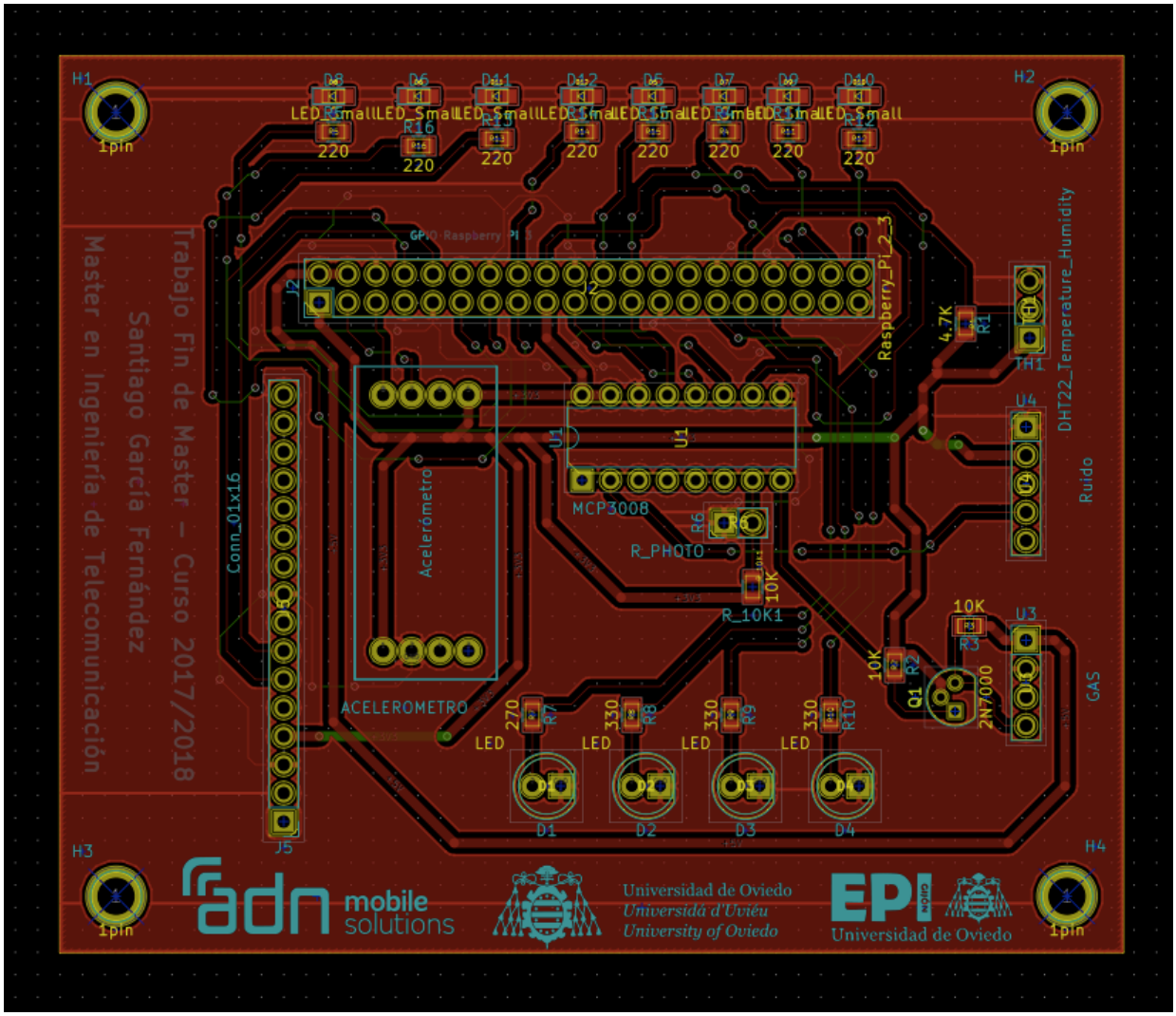


Fig. 4.26.- Diseño final de la PCB realizado con KiCad, el cual se usará en el prototipo TS PCB. La colocación de las distintas huellas sobre la placa se ha realizado pensando en la colocación y en las dimensiones de los diferentes sensores y sobre todo en la funcionalidad que desempeña cada uno de ellos y por tanto en las necesidades de orientación que tienen.

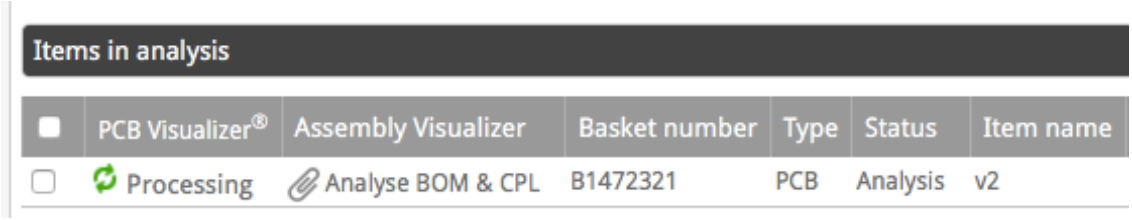


Fig. 4.27.- Apartado dentro de la web de Eurocircuits donde se indica que se están analizando los archivos de PCB que se han cargado en la herramienta.

Este proceso generalmente dura pocos minutos o incluso menos. Cuando los archivos hayan sido analizados se mostrará tal como aparece en la figura 4.28.

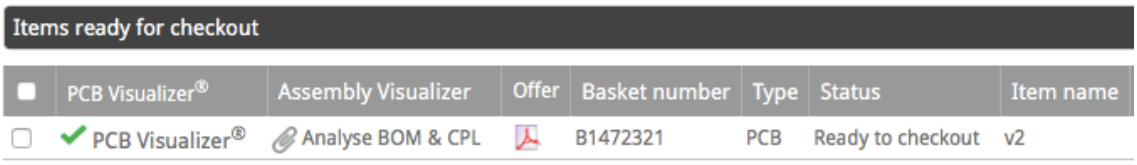


Fig. 4.28.- Apartado dentro de la web de Eurocircuits donde se indica que se han analizado correctamente los archivos de PCB y ya es posible acceder al análisis.

Si se entra en la herramienta "PCB Visualizer" es posible acceder a multitud de opciones de inspección de la PCB que se ha diseñado. Una de las principales opciones disponibles es una denominada PCB *Checker* donde se informa entre otras cosas del número de errores o advertencias (*warnings*) que presenta la placa que se ha cargado en la aplicación. Como se ha remarcado la herramienta es muy útil porque además en caso de haber errores te indica exactamente el punto en el que se produce el error y cuál es ese error para que sea posible volver al KiCad y corregirlo de una manera rápida y simple.

Tal como se ve en la figura 4.29 el PCB *Checker* deberá mostrar 0 errores para asegurarnos de que el fabricante no tendrá problemas a la hora de crear la placa que se le encargue.

Por comentar alguna funcionalidad más de la herramienta es posible ver por ejemplo la clase de la PCB (que en este caso sería una 6c tal como se ve en la figura 4.30) o distintas vistas de las partes que componen la PCB como las que se presentan en la figura 4.31 e 4.32.

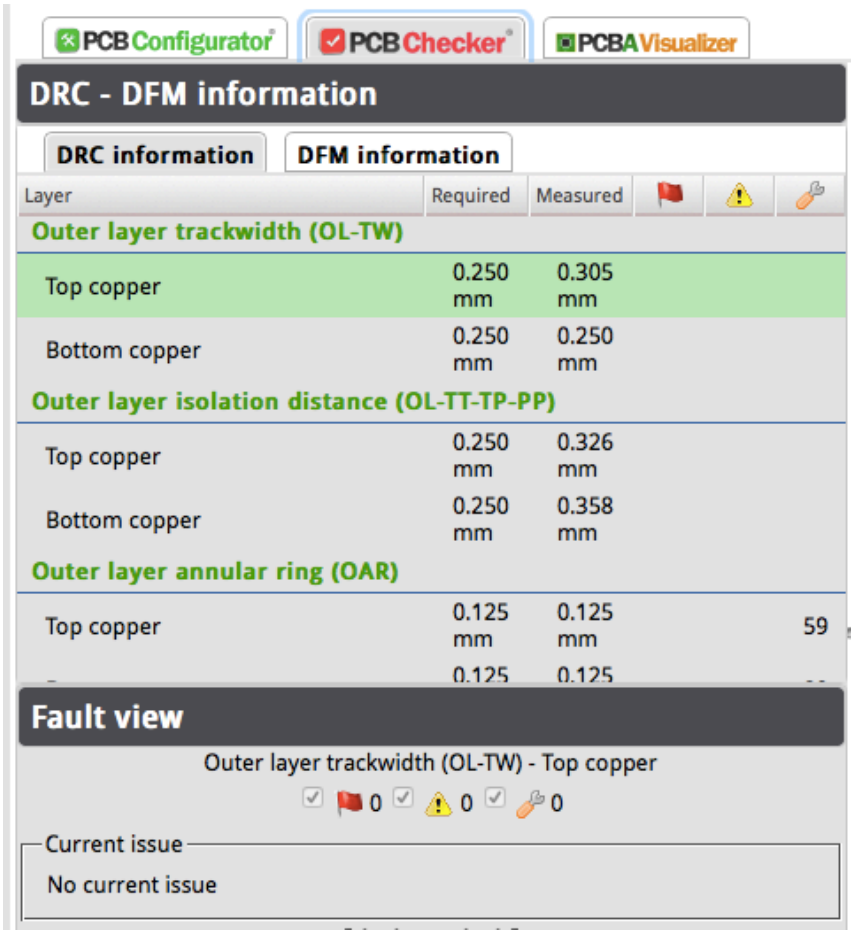


Fig. 4.29.- Resumen de errores y advertencias en herramienta PCB Checker de Eurocircuits.

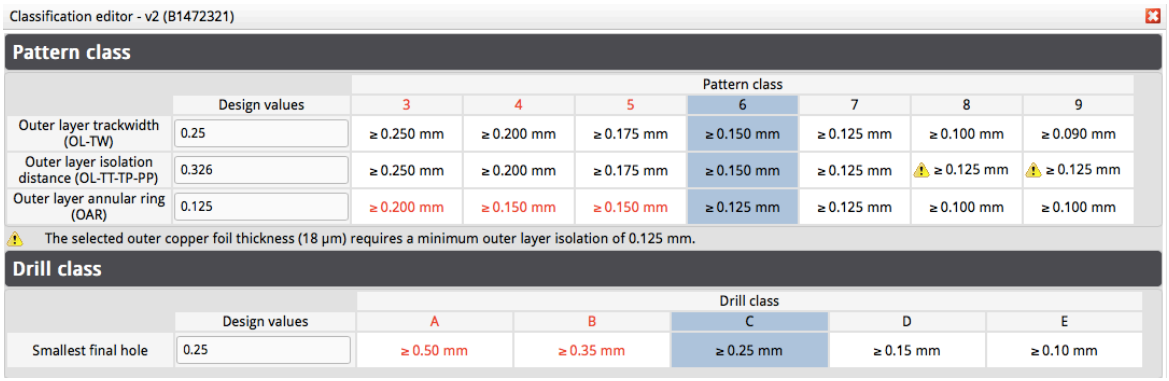


Fig. 4.30.- Resumen de categoría de la PCB diseñada suministrado por la herramienta de análisis de Eurocircuits.

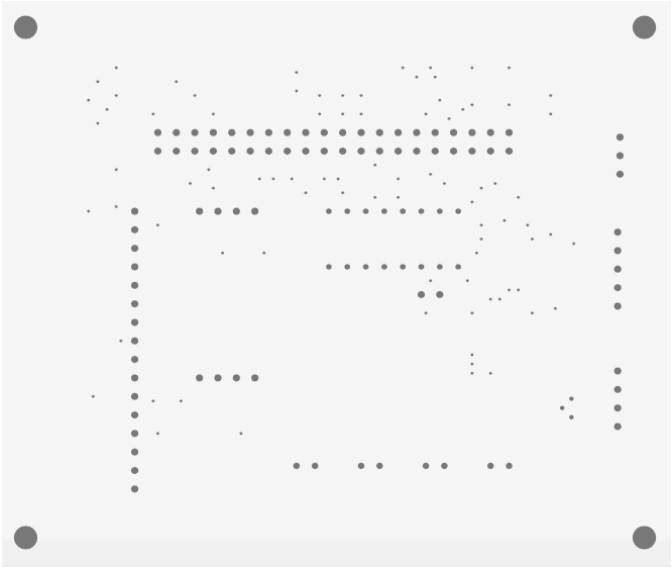
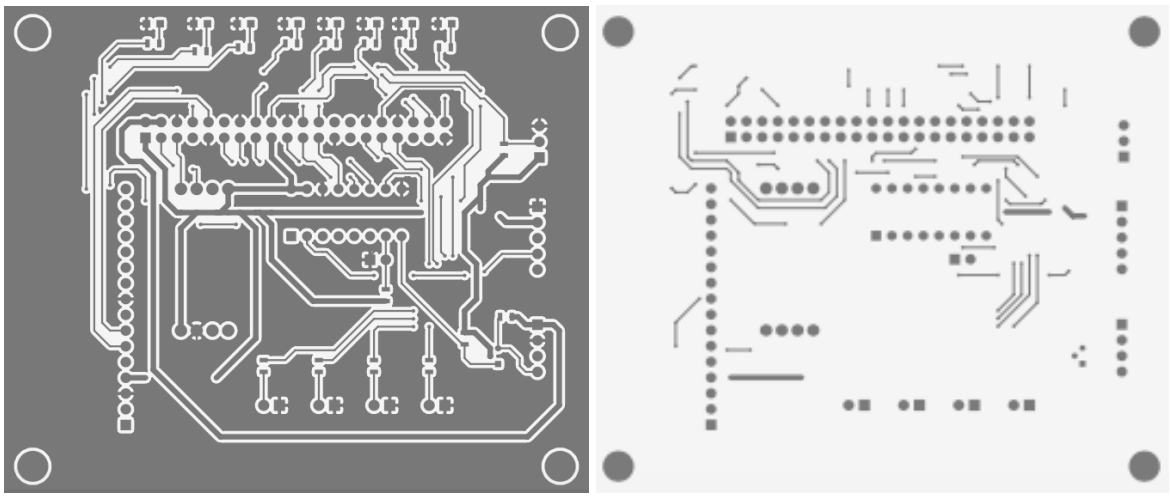


Fig. 4.31.- Imagen suministrada por la herramienta de análisis de Eurocircuits que indica todos los agujeros que serán necesarios para la fabricación de la PCB situándolos en sus posiciones correspondientes y con los tamaños que tendrán.



a) b)

Fig. 4.32.- Imágenes suministradas por la herramienta de análisis de Eurocircuits que muestran las pistas de cobre que tendrán la PCB tras su fabricación así como sus anchos tanto en la capa *top* (a) como en la capa *bottom* (b).

10. Envío de Gerbers para fabricación

Con todas las comprobaciones pertinentes realizas ya se ha asegurado la correcta fabricación de la PCB con lo que sólo faltaría enviar los Gerbers al fabricante que se desee y esperar a la llegada de la placa real.

Las PCBs ya fabricadas tienen el aspecto que se muestra en las figuras 4.33 (capa top) y 4.34 (capa *bottom*).

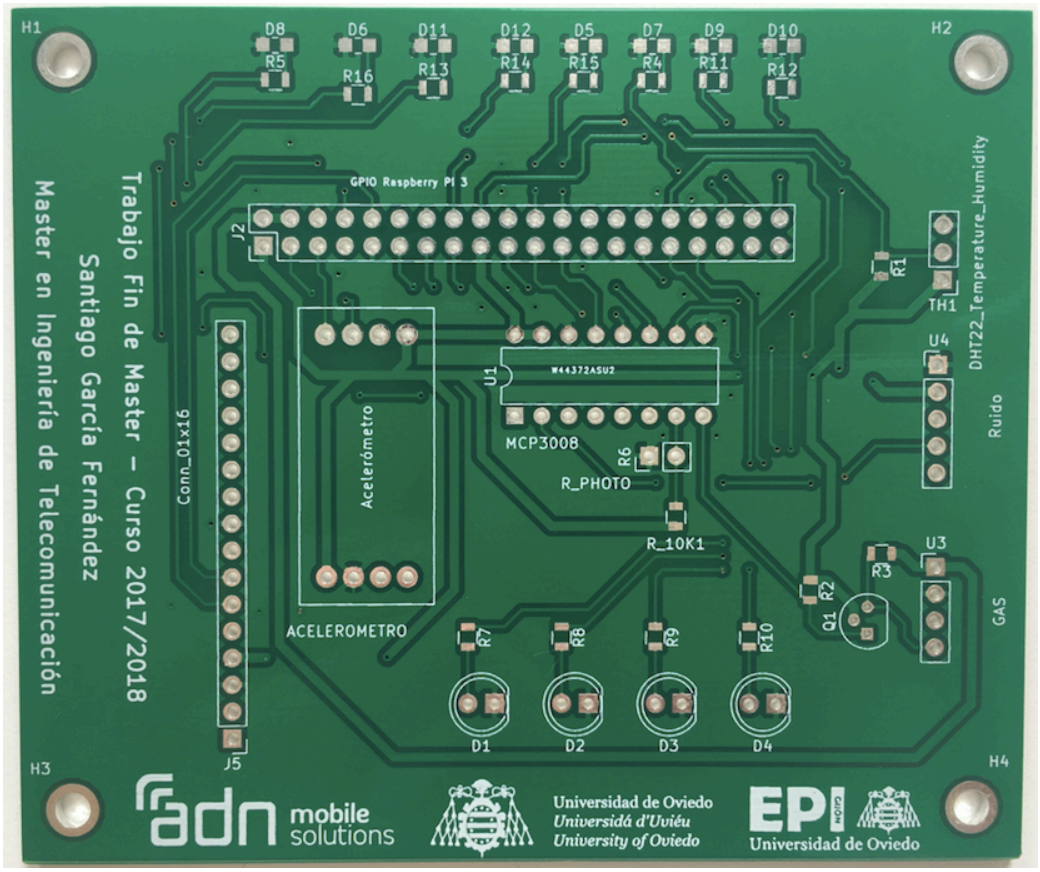


Fig. 4.33.- Imagen de capa *top* de la PCB real una vez fabricada.

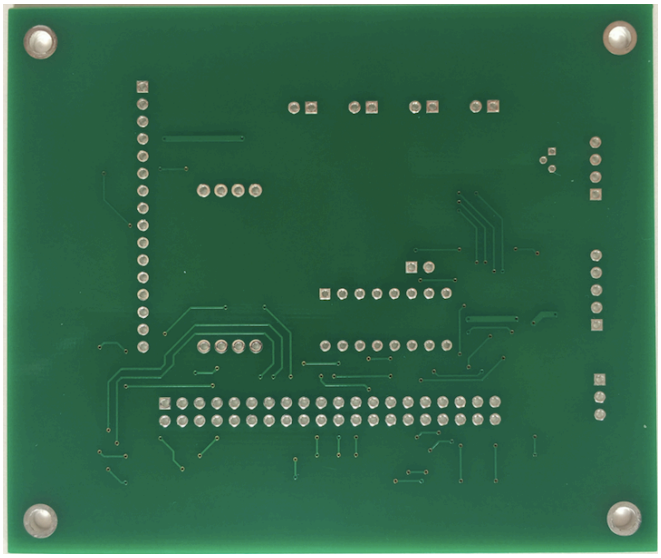


Fig. 4.34.- Imagen de capa *bottom* de la PCB real una vez fabricada.

4.2.3.2. Montaje del prototipo

Sobre esta PCB han de seguirse los pasos anteriormente especificados hasta llegar a completar el prototipo definitivo.

1. Soldadura de conectores y demás componentes en PCB

Sobre la placa se soldaron los distintos conectores y componentes presentes en el BOM u otros equivalentes. Para esta fase se contó de nuevo con la ayuda de personal de la empresa ADN con el fin de que el acabado de la soldadura fuese mejor al disponer de material más especializado.

En esta primera placa se soldaron conectores para cada uno de los componentes, incluidos los sensores, en una segunda placa quizás se podrían llegar a soldar los sensores directamente sobre la placa para ahorrar el espacio de los conectores pero es esta primera el objetivo era sobre todo probar el funcionamiento de cada componente sobre la PCB y comprobar que no había habido ningún error en la fabricación o en el diseño de la misma.

En las figura 4.35 y 4.36 se pueden ver los resultados de la soldadura de componentes tanto en la capa *top* (figura 4.35) como en la capa *bottom* (4.36)

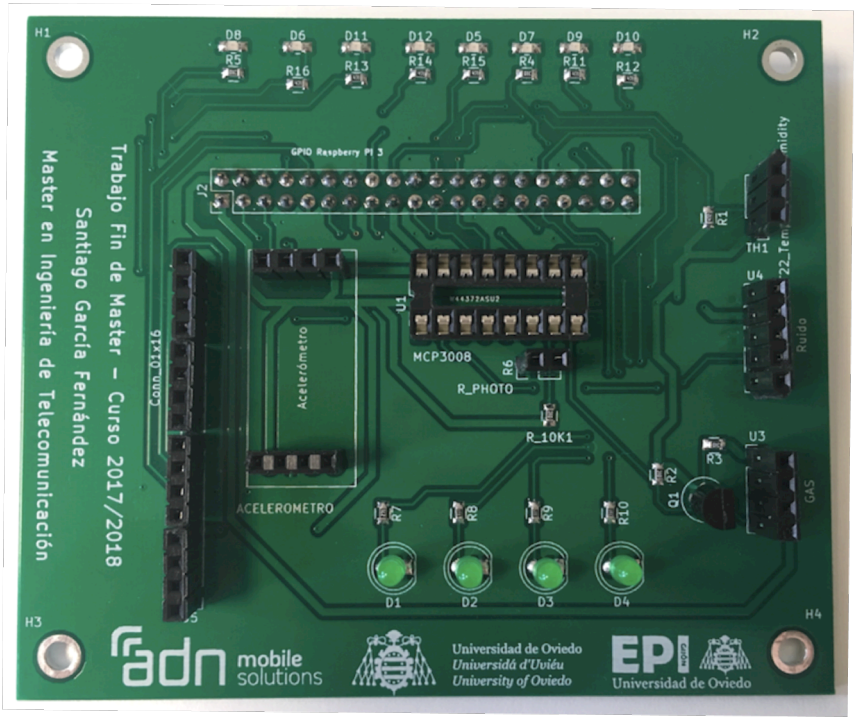


Fig. 4.35.- Imagen de capa top de la PCB real una vez se han soldado los diferentes componentes sobre la placa.

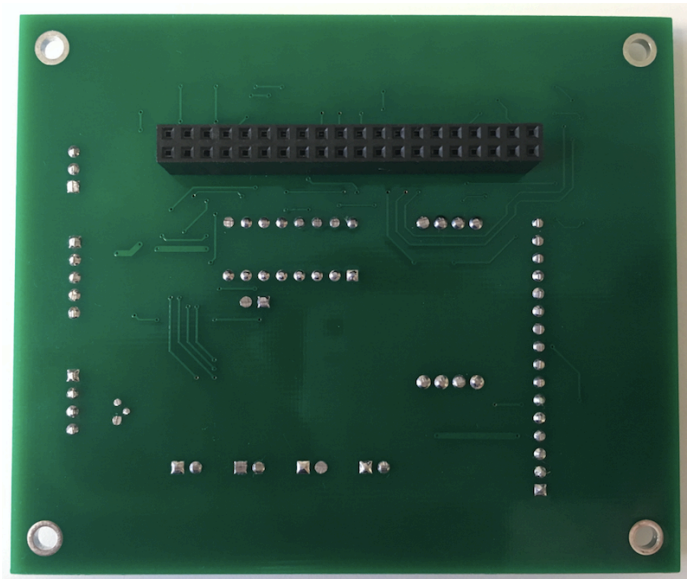


Fig. 4.36.- Imagen de capa *bottom* de la PCB real una vez se han soldado los diferentes componentes sobre la placa.

2. Verificación de correcto funcionamiento de la placa

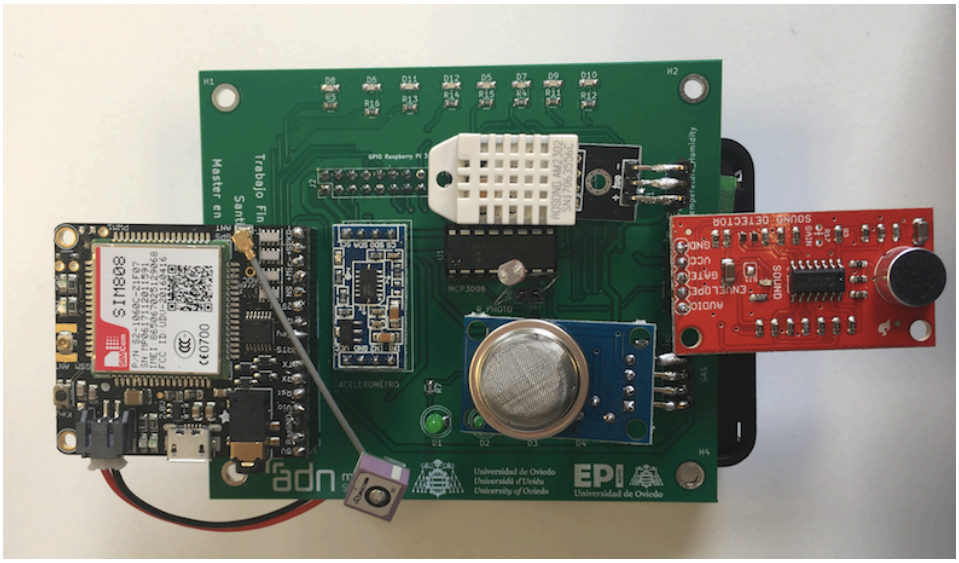
Una vez soldados sobre la PCB todos los conectores necesarios así como los componentes de montaje superficial de menor tamaño como resistencias, transistores y LEDs, es hora de colocar cada uno de los componentes restantes e ir probando el funcionamiento de la placa poco a poco. Los componentes que restan por colocar en la PCB serían:

- Conversor A/D
- Acelerómetro
- Sensor de nivel de luminosidad
- Sensor de temperatura y humedad
- Sensor de calidad del aire
- Sensor de nivel de sonido
- Módulo GPS/GSM

Tal como se estaba diciendo es conveniente ir colocando los componentes, sobre todo los sensores uno a uno e ir verificando su correcto funcionamiento. Las razones tienen que ver por un lado con la seguridad de los componentes que pudieran verse dañados por algún error en el diseño de la PCB de modo que cuantos menos daños se causen mejor. La otra razón es que si existe algún error en el diseño será mucho más fácil de detectar siguiendo este procedimiento.

La verdad es que esta fase del proceso fue una de las más cortas con diferencia ya que no se detectó ningún tipo de problema. De este modo, una vez conectados todos los componentes y sensores sobre la PCB y ya con la Raspberry Pi acoplada en la parte inferior el resultado final sería algo como lo que se puede apreciar en la figura 4.37. El encapsulado de la Raspberry Pi se ha dejado por comodidad y estabilidad a la hora de

agregar un segundo encapsulado al sistema, pero podría sustituirse o eliminarse en cualquier momento.



a)

b)

Fig. 4.37.- Imagen de la PCB con todos los módulo montados sobre ella y ya colocada sobre la Raspberry Pi 3. Este sería el prototipo TS PCB a falta del encapsulado. Imagen del prototipo desde arriba (a) y desde un lateral (b).

4.2.3.3. Diseño y fabricación del encapsulado del prototipo

Por un lado se está trabajando simplemente con un prototipo que es más que susceptible de ser modificado con el fin de corregir o mejorar algunos defectos, pero por otro lado se pretende que el prototipo pueda probarse en entornos reales y por diferentes usuarios. Estos dos hechos llevan a pensar en la realización de un encapsulado que haga sencillo, cómodo de usar y relativamente atractivo al prototipo final pero si que ello conlleve un gasto de tiempo y dinero excesivo en el diseño ya que puede que haya que modificarlo.

Se pensó entonces en la realización de un primer encapsulado mediante el uso de la tecnología de impresión en 3D.

Los pasos seguidos para la fabricación con impresión 3D del encapsulado serían los siguientes:

- Toma de decisiones de posición interna del cableado necesario
- Toma de decisiones referentes a la fijación del prototipo dentro del encapsulado
- Toma de decisión de los orificios necesarios para conexión del prototipo con el exterior
- Toma de medidas de las dimensiones necesarias para proteger y asegurar el prototipo
- Diseño del encapsulado teniendo en cuenta las reglas de diseño para impresión 3D
- Generación de archivos de impresión
- Verificación de diseño correcto mediante programa de impresión
- Decisión de nivel de relleno necesario del prototipo
- Impresión de la pieza o piezas del encapsulado
- Unión de las piezas si hiciese falta

- Análisis del resultado

En cuanto a las diferentes tomas de decisiones iniciales, las más importantes han sido la decisión de que los cables de alimentación internos de los diferentes módulo del prototipo se quedaran dentro del encapsulado para así hacer un prototipo más manejable y evitar posibles desconexiones en los traslados o con el uso normal del sistema. Este punto es importante ya que generará un aumento considerable de las dimensiones del encapsulado y en consecuencia del prototipo final dado que los conectores USB de los que se dispone ocupan un gran espacio al ir conectados en un lateral de la Raspberry Pi tal como se puede apreciar en la figura 4.37. Este tema podría solucionarse seguramente con algún tipo de adaptador o conector USB especial que permita más libertad de movimiento pero dado que este encapsula es factible que no sea el definitivo no se ha hecho demasiado hincapié en solucionar este problema.

Dado que se pretende aumentar la facilidad de uso del prototipo se ha decidido también dotar al encapsulado de un único orificio para conexión con el exterior que es el que está destinado a la alimentación a través del cable micro-USB-USB que iría directamente conectado al adaptador USB del mechero del vehículo y que sería la única fuente de alimentación del prototipo.

Teniendo estos dos premisas en cuenta se procedió a la toma de medidas y se comenzó a crear un primer diseño.

La herramienta empleada para el diseño ha sido “TinkerCad”. Esta herramienta permite acceso web libre y gratuito para realizar diseños de una manera bastante intuitiva y con una gran variedad de opciones básicas de diseño que sorprende para ser una herramienta gratuita y para principiantes. El programa permite trabajar con formas básicas rellenas y huecas y desplazarlas por un espacio de trabajo en 3D que está perfectamente acotado en todo momento y en el cual puedes ajustar la rejilla de trabajo a las dimensiones de interés. Todas estas características hacen que el diseño de piezas no demasiado

complejas sea muy sencillo para personas sin conocimientos avanzados de CAD. Además este método de diseño es mucho más sencillo que el que ofrecen otras herramientas gratuitas en las cuáles las formas se generan mediante un programa que es necesario escribir.

El primer diseño realizado se puede ver en la figura 4.38. Como se ve el diseño es de lo que sería la base del encapsulado y faltaría la parte superior cuyo diseño sería una segunda pieza que iría atornillada a la base gracias a los 4 orificios realizados.

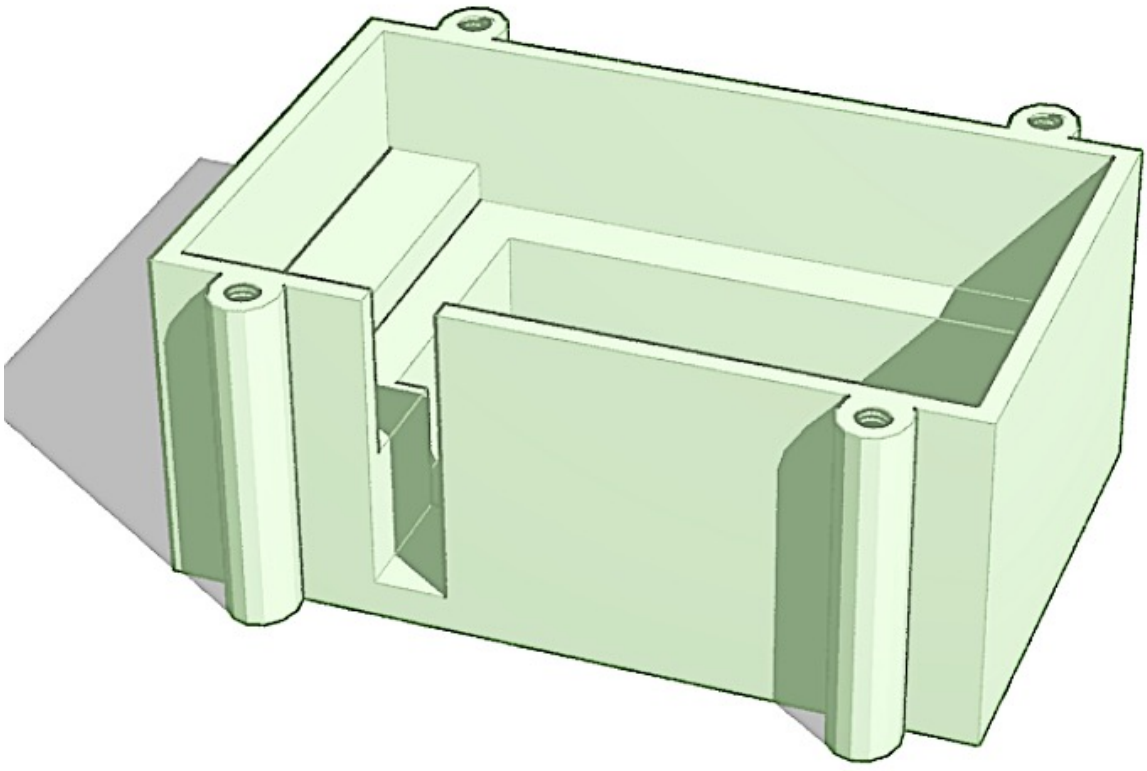


Fig. 4.38.- Diseño de encapsulado sin tapa para impresión en 3D. Base diseñada en una pieza que nunca llegó a ser impresa.

Para el diseño de los orificios para los tornillos se ha empleado una plantilla ya prediseñada de tornillos de diferentes tamaños de los cuales se han empleado los más

pequeños ya que en sí las medidas disponibles en esta plantilla son bastante grandes. Se han empleando por tanto huecos para tornillos M10. La plantilla de los diversos tornillos se puede ver sobre el espacio de trabajo de "TinkerCad" en la figura 4.39.

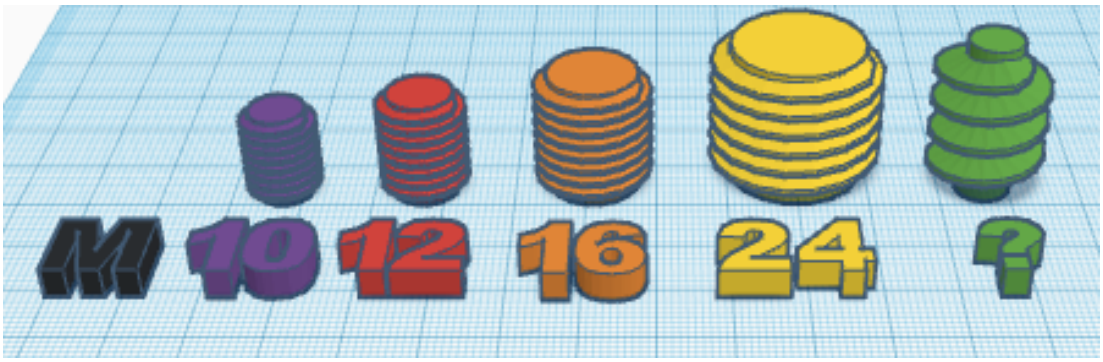


Fig. 4.39.- Plantilla de tornillos importada a TinkerCad para su uso en el diseño de las piezas.

Una vez realizado el diseño se generaron los archivos de impresión y se pasaron al programa de impresión de la impresora empleada que en este caso es la Ultimaker 2+ la cual se puede ver en al figura 4.40. El software de impresión es el "Ultimaker Cura" para cuyo uso, tanto del software como de la impresora se ha necesitado y se ha contado con la colaboración del Área de Teoría de la Señal y Comunicaciones (TSC) de la Universidad de Oviedo.

Fig. 4.40.- Impresora 3D Ultimaker 2+ empleada para la impresión de la base del encapsulado del prototipo final.

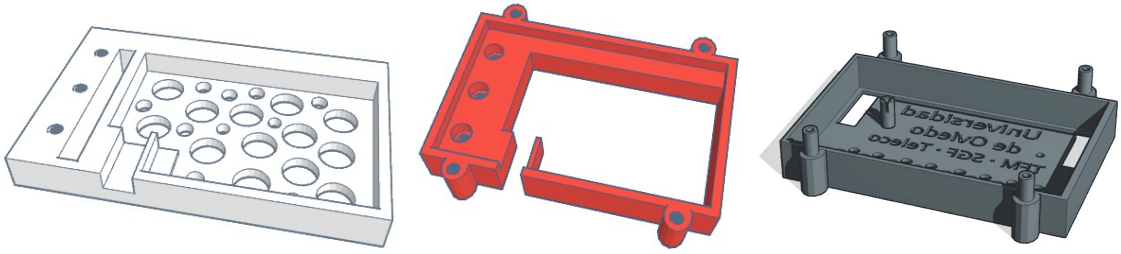
Antes de comenzar con la impresión ya se encontró el primer problema y es que la pieza diseñada tenía demasiado relleno y la pieza impresa, según el cálculo del software y aún reduciendo el relleno que hace la impresora a un 40% tendría un peso cercano a un kilo, que es mucho para un encapsulado. De todas formas se decidió seguir adelante e imprimir la pieza. Sin embargo, por alguna deficiencia de la impresora no se pudo llegar a imprimir por completo la misma puesto que la impresora se atascaba y dejaba de hacer contacto con la pieza después de algunas horas de trabajo.

La solución a este problema se pensó que podría pasar por dividir la pieza en partes de modo que el tiempo de impresión de cada una se redujese y así poder llegar a tener un encapsulado aunque fuese muy provisional.

Se decidió por tanto seguir por ese camino y dividir la pieza en partes. Concretamente se dividió la pieza correspondiente a la base en tres partes. Además se trató de reducir el

relleno de las piezas siempre que se pudiese para reducir el peso y además mejorar la ventilación del interior del prototipo.

En las figuras 4.41 a), b) y c) se muestran los diseños finales de las tres partes que componen el encapsulado final del prototipo.



a) **b)** **c)**

Fig. 4.41.- Piezas del encapsulado del prototipo final diseñadas e impresas

Una vez finalizados los nuevos diseños se procedió a su impresión en la impresora Ultimaker 2+, la cual se expone en la figura 4.40.

El proceso de ensamblado ha sido diferente entre las partes 1 y 2 (figuras 4.41 a) y b)) y entre las partes 2 y 3 (figuras 4.41 b) y c)) puerto que las partes 1 y 2 forman la base su unión ha de ser permanente, por esta razón la unión se ha realizado mediante tirafondos en los 3 orificios que se ven en las imágenes, siendo estos tirafondos de nylon para reducir el peso final. Además se ha añadido pegamento especial para plásticos con el objetivo de asegurar la unión. Con la unión de estas dos partes quedaría formada la base del encapsulado.

La unión entre la tapa y la base debe ser una unión temporal de modo que la tapa pueda quitarse y ponerse sin demasiados problemas. De este modo la tapa simplemente encaja en los 4 orificios realizados en la base.

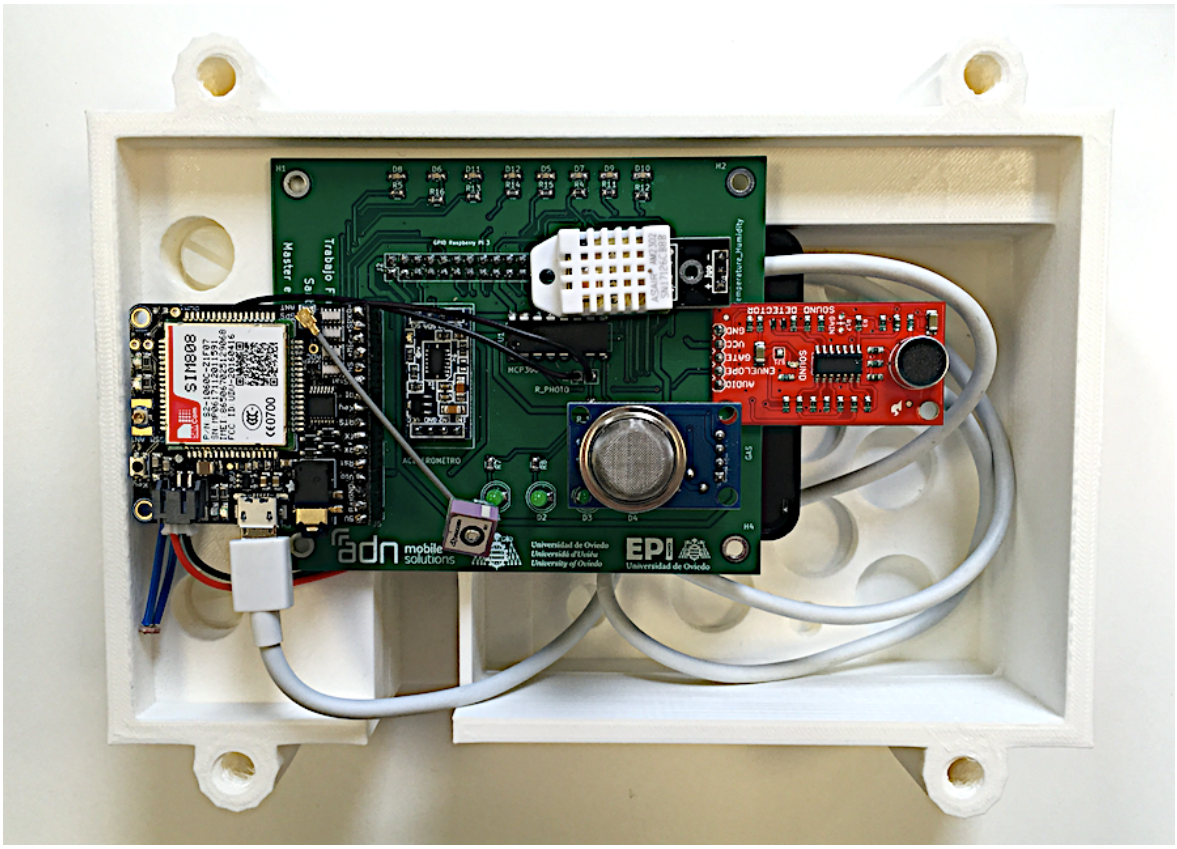


Fig. 4.42.- Prototipo final basado en TS PCB colocado sobre la base del encapsulado con todas las conexiones internas realizadas.

Finalmente en las figuras 4.42 y 4.43 se muestra el resultado final del prototipo TS PCB ya dentro del encapsulado completo. Entre otras cosas puede verse que se ha dejado un orificio en la tapa para dejar salir al sensor de nivel de luminosidad pero el resto de sensores están dentro del encapsulado. Parece obvio que el sensor de luz tenga que estar fuera pero podría pensarse que otros sensores ambientales deberían estar mucho más expuestos también. En este punto se vuelve de nuevo a tratar de explicar que la precisión no es el fin último de este proyecto si no que basta con obtener resultados razonables y poder observar tendencias en los datos. Por ello no es crítico que el resto de sensores se guarden dentro del encapsulado, además la cantidad de orificio y ranuras realizadas sobre la tapa es suficiente como para que el sonido o el aire del ambiente inunden el interior de encapsulado.

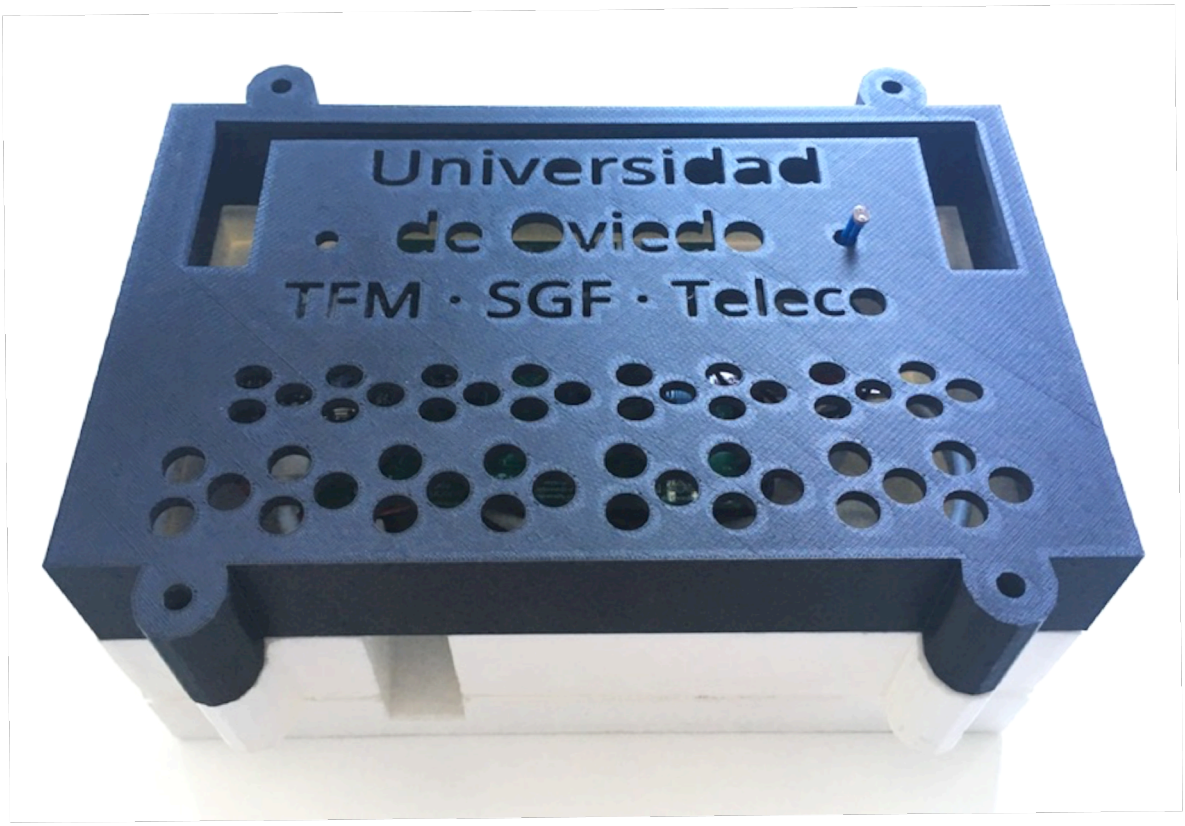


Fig. 4.43.- Prototipo final basado en TS PCB colocado dentro del encapsulado impreso en impresión 3D. Todas las conexiones internas han sido realizadas y el prototipo sólo dispone de un interfaz con el exterior a través del hueco rectangular visible en la base (parte blanca) del encapsulado a través del cual se conectaría la alimentación mediante un cable micro-USB.

4.3 RECOGIDA DE DATOS BIOMÉTRICOS

Para al recogida de datos biométricos se han empleado dispositivos *wearables*. Por ello en este apartado se procede a la descripción de los diferentes dispositivos empleados así como de la manera en la que cada uno de ellos se comunicará y enviará los datos al sistema de almacenamiento de los mismos.

4.3.1 Selección de componentes

Se emplearán un total de tres dispositivos *wearables* con distintas características y que ofrecen la posibilidad de registrar diferentes tipos de datos biométricos. El objetivo del uso de hasta tres dispositivos diferentes no es el de emplear los tres de manera simultánea en un escenario de pruebas real sino que se pretende evaluar cada uno de ellos de manera individual con el objetivo de definir cual de los tres tiene mejores condiciones para funcionar dentro del sistema que se está proponiendo y por qué. Los tres dispositivos empleados, los cuales se explicarán de manera independiente son: Huawei Watch, Polar H10 y *Empatica* E4.

- **Huawei Watch 2**

  En este primer caso el dispositivo que se presenta es un SmartWatch común y por lo tanto es un dispositivo que cualquier persona podría llevar hoy en día. El modelo Huawei Watch 2 es el que se presenta en la figura 4.44. La decisión de emplear este tipo de dispositivo se tomó en primer lugar porque eran uno de los dispositivos de los que ya se disponía en el laboratorio a la hora de comenzar este trabajo y en segundo lugar por la comodidad en su utilización. [39]

  A continuación se exponen algunas de las principales características de este modelo:

  - Dimensiones: 49 x 45 x 12.6 (mm)
  - Pantalla: 1.2 “
  - Memoria: 4GB (no ampliable)
  - RAM: 768 MB
  - SO: Android Wear
  - Resistencia al agua IP68
  - Conectividad Wi-Fi y Bluetooth 4.0

- Batería: 410 mAh
- Precio: 400 -700 €

Fig. 4.44.- Smartwatch Huawei Watch 2 empleado para la adquisición de datos de ritmo cardiaco, aceleración y rotación del brazo y posicionamiento GPS.

A parte de estas características, de las cuales se podrán obtener puntos a favor y en contra cuando se compare este dispositivo con alguno de los otros, existen otra característica o capacidad muy importante de este dispositivo teniendo en cuenta su objetivo. La parte importante es la referente a los sensores de los que dispone el reloj.

Sensores disponibles de interés en Huawei Watch 2:

- Sensor de ritmo cardiaco
- Acelerómetro
- Giroscopio
- GPS + Glonass

Posiblemente el sensor de mayor interés sea el sensor de ritmo cardiaco dado que el uso de este dispositivo será fundamentalmente para monitorizar variables biométricas del conductor. Sin embargo los otros tres sensores podrían resultar también bastante útiles para

analizar comportamientos al volante o modos de conducción y sobre todo el sensor de GPS podría servir para comparar o contrastar datos del módulo GPS fijo.

- **Polar H10**

Con este segundo dispositivo se pasa a hablar de algo muy diferente a un SmartWatch. En este caso el dispositivo consiste en una cinta o elástico pectoral con un sensor acoplado de modo que ya se pasa a tratar un dispositivo que no es algo tan común como un reloj. [40] El dispositivo puede verse en la figura 4.45 y sus principales especificaciones y características de interés serían las siguientes:

- Dimensiones conector: 34 x 65 x 10 (mm)
- Peso total: 60 g
- Batería: CR 2025, duración de 400 horas transmitiendo
- Temperatura de funcionamiento: -10 – 50 (°C)
- Precio: 90 €

Fig. 4.45.- Cinta pectoral Polar H10 empleada para la obtención de datos de ritmo cardiaco e intervalos RR.

A continuación se exponen las principales funciones de este dispositivo:

- Sensor de frecuencia cardiaca con precisión alta

- Conectividad Bluetooth 4.0
- Dispositivo actualizable
- Relativamente cómodo de llevar

De estas funciones en principio podría destacar que la medida del ritmo cardiaco debería ser bastante más precisa en este dispositivo que en el reloj Huawei. Otro punto muy a favor de este dispositivo sería el precio que es entre 6 y 8 veces menor que el Smart Watch. Y a priori no tiene muchos más puntos a favor aunque, como se verá en puntos posteriores sí que presenta un funcionamiento mucho más robusto que otros dispositivos, al menos integrado dentro de este sistema.

- **Empatica E4**

El último dispositivo probado en el transcurso de este trabajo ha sido un brazalete de la compañía *Empatica*, concretamente el modelo E4. [41] Este dispositivo puede verse en la figura 4.46 y sus especificaciones principales se detallan a continuación:

- Dimensiones: 44 x 40 x 16 (mm)
- Peso: 25 g
- Batería: + 24 horas en modo *streaming*
- Conectividad Bluetooth Low Energy Smart
- Memoria: Hasta 60h de datos
- Certificados:
  - CE Cert. No. 1876/MDD (93/ 42/EEC Directive, Medical Device class 2a)
  - FCC CFR 47 Part 15b
  - IC (Industry Canada)
  - RoHS
  - MIC Japan: BLE112 has type approval certification ID R 209-J00046
- Precio: 1400 – 1500 €

a) b)

Fig. 4.46.- Pulsera de actividad *Empatica* E4 empleada para adquisición de datos biométricos muy fiables, precisos y con certificados de calidad. Vista delantera (a) y trasera (b).

Funciones principales del dispositivo:

- Sensor PPG: Mide el pulso de volumen de sangre (BVP) del cual se puede extraer la variabilidad del ritmo cardiaco [58]
- Acelerómetro en 3 ejes
- Sensor EDA (sensor GSR): Mide las fluctuaciones constantes en ciertas propiedades eléctricas de la piel
- Lectura de la temperatura de la piel
- Reloj interno de tiempo real con resolución de 0.2 segundos

Atendiendo a las funcionalidades ofrecidas por este dispositivo parece realmente superior a los otros dos, y en realidad lo es. Si se habla de recogida de datos biométricos la comparativa entre este dispositivo y los otros dos resulta prácticamente ridícula puesto que este está a un nivel muy superior. Esta diferencia radica no sólo en la capacidad de recoger más tipos de datos biométricos sino en las certificaciones médicas de las que dispone y las

cuales validarían directamente todos los datos que se obtengan y permitirá hacer estudio mucho más profesionales y correctos.

Por comentar alguna de los puntos en contra que se pueden ver a priori sería el tema del precio que parece demasiado elevado para un sistema como este el cual debería ser en principio algo mucho más accesible.

Este ha sido un primer análisis de cada uno de los dispositivos pero será en el apartado de funcionamiento donde se pueda relacionar cada uno de ellos con el sistema y se verán algunas ventajas o inconvenientes que presenta cada uno de ellos que en este punto aún no es posible vislumbrar.

Algo que sí se puede determinar ya es la manera de conectarse de cada uno de los dispositivos con el sistema de almacenamiento de datos. En el esquema de la figura 4.47 se representa eso y se aprecia que tanto la pulsera *Empatica* como la cinta pectoral Polar H10 han de enviar sus datos vía Bluetooth mientras que el *Smartwatch* permite el envío de esos datos tanto vía Wi-Fi como vía Bluetooth.

Fig. 4.47.- Esquema de la interconexión entre los distintos dispositivos wearables y la SBC del sistema central embarcado.

4.4. RECOGIDA DE DATOS DEL VEHÍCULO

En este apartado tal como se ha hecho en apartados anteriores del capítulo de implementación, se hablará únicamente de la arquitectura y de los componentes hardware que la forman sin entrar a explicar a fondo el funcionamiento software de sistema de recogida de datos. Esta parte se explicará posteriormente en el apartado de funcionamiento del sistema.

La arquitectura de esta parte del sistema es muy simple y se puede apreciar esquematizada en la figura 4.48. Donde el acceso directo a los datos del vehículo se realiza a través de un adaptador conectado al puerto OBDII del vehículo. Este adaptador será el interfaz entre el sistema central embarcado y el bus CAN y tendrá por tanto las funciones de por un lado realizar las peticiones correspondientes cuando así se le solicite desde la SBC y también enviar de vuelta los resultados de estas peticiones obtenidos del bus CAN.

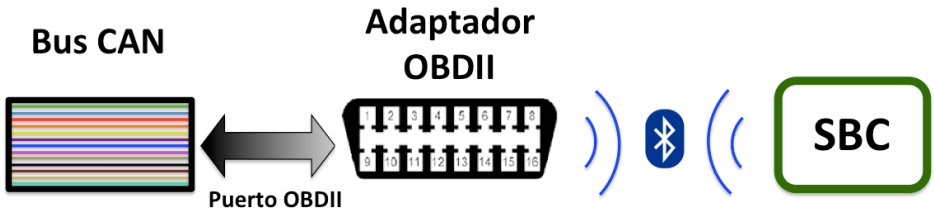


Fig. 4.48.- Esquema de la interconexión entre los distintos elementos que intervienen en la adquisición de datos del mecánicos del vehículo.

4.4.1 Selección de componentes

En esta parte del sistema la selección de componentes se reduce a la elección del adaptador OBDII más adecuado.

La primera decisión a tomar tiene que ver con la manera en la que el adaptador se comunicará con terceros dispositivos. Las dos opciones principales en la inmensa mayoría de adaptadores OBDII universales y comerciales son la conectividad vía Wi-Fi o vía Bluetooth. En este caso cualquiera de las dos sería válida dado que la Raspberry Pi 3 incorpora ambos módulos, tanto Wi-Fi como Bluetooth. Finalmente se decidió seleccionar el tipo de adaptador que emplea la conectividad Bluetooth dado que previsiblemente el Wi-Fi de la SBC va a tener un número de usos mayor aunque esto no debería de ser un inconveniente tampoco.

Se han adquirido un total de dos adaptadores equivalente que deberían de ser internamente iguales o muy parecidos y qué únicamente se diferencia en su apariencia externa. Ambos son adaptadores universales ELM-327 con soporte para todos los protocolos OBDII. Ambos modelos se pueden ver en la figura 4.49.

a) b)

Fig. 4.49.- Adaptadores OBD-II Bluetooth empleados y probados en este proyecto, uno de menores dimensiones (a) que el otro (b).

El adaptador de menor tamaño (color azul) tiene unas dimensiones de 48 x 35 x 25 (mm) y un peso de 45 g mientras que el mayor (color negro) tiene unas dimensiones de 90

x 50 x 30 (mm) y un peso de 75 g. Siendo más práctico el primero de ellos una vez se ha comprobado que el funcionamiento de ambos es correcto. [42-43]

5. Funcionamiento del sistema

En este apartado se tratará de explicar el funcionamiento interno del sistema y se relacionará todo este funcionamiento interno con la parte de hardware ya explicada en el capítulo anterior. Puesto que una parte importante de este trabajo se ha basado en desarrollo software este capítulo resulta esencial para entender el funcionamiento del sistema final sobre todo en la parte referente a la recogida, tratamiento inicial y almacenamiento de datos.

De un modo similar al seguido durante el desarrollo de todo este trabajo se realizará una explicación de cada parte de manera individual puesto que es la mejor manera de entender posteriormente el propósito global.

Cabe destacar o diferenciar dos tipos de programas en función de si estos se emplean para recibir y almacenar datos dentro de la propia SBC o si son programas que se ejecutan en dispositivos externos a la SBC para realizar la adquisición de datos y enviar posteriormente estos hacia la SBC para que un segundo programa se encargue de la recepción y almacenamiento. Será por tanto en esos dos grandes grupos donde se realice la primera bifurcación en la explicación.

5.1. Software de almacenamiento, adquisición y/o recepción de datos integrado en la SBC

En esta parte se encuentra el grueso de los programas generados a lo largo del trabajo a excepción de los que se han creado para ejecutarse en alguno de los dispositivos

wearables con el fin de conseguir una adquisición de datos y una comunicación con el sistema central embarcado satisfactoria.

Previamente a la explicación de cada uno de los programas de extracción y almacenamiento de cada tipo de dato es conveniente explicar el funcionamiento general del sistema así como lo puntos comunes compartidos por la totalidad o al menos por la gran mayoría de los scripts generados a lo largo de este trabajo.

5.1.1. Funcionamiento general y puntos en común de los programas

El funcionamiento del sistema en lo que a la parte software se refiere está basado en la ejecución de manera simultánea de una serie de programas que tienen el principal objetivo de realizar una adquisición o recepción de datos de uno o varios dispositivos y almacenar esos datos en una base de datos SQLite. El motivo de la elección de las bases de datos SQLite ha sido principalmente por su sencillez y por la rapidez que aportan en el almacenamiento de datos lo cual puede resultar vital en una aplicación que pretende funcionar total o parcialmente en tiempo real como es el caso de esta.

Resulta importante destacar que todos los programas generados para ejecutarse dentro en la SBC del sistema han sido programados en lenguaje Python.

Además de la ejecución de manera simultánea de todos los programas es necesario exigir que estos se ejecuten de manera automática en el arranque de la Raspberry Pi. Para lograr satisfacer estos dos requerimientos se ha hecho uso del archivo de configuración ***crontab.*** Lo que se está haciendo en realidad es emplear el programador de tareas de *Unix* que se denomina *Cron*. Lo que hace este servicio es lo que hace cualquier otro programador de tareas en Windows o en cualquier otro sistema operativo, es decir, establece un instante temporal absoluto o relativo en el que ejecutar un determinado comando o un determinado programa. En el archivo de configuración crontab es necesario

por tanto escribir una línea correspondiente a la ejecución de cada uno de los scripts a ejecutar. Con esto se consigue que la Raspberry Pi ejecute todos los archivos como tareas independientes. Faltaría indicar en qué momento ha de ejecutar esas instrucciones. Para ello el servicio cron dispone de la opción de indicar una fecha y hora o concreta pero también existen diferentes comando predefinidos que hacen referencia a diferentes situaciones. La que interesa en este caso sería la opción de @reboot que indica al programa que ha de ejecutar los comandos que le siguen una única vez y justo en el arranque del sistema operativo. Esta explicación queda mucho más clara si se observa el archivo de configuración crontab ya finalizado, el cual se muestra en la figura 5.1. En apartados posteriores se volverá a hacer referencia a esta captura para aclarar algunos temas referentes a algún otro detalle observable sobre este archivo.

```
###### Sensores ambientales

@reboot sudo python Sensores.py

#####  Huawei Watch 2

@reboot sudo python serverudp_huawei.py #Heart Rate
@reboot sudo python serverudp_huawei_2.py  # Acelerometro y Giroscopio

#####  Modulo GPS

@reboot sudo python serial_gps.py

#####  OBDII

@reboot sudo python obd_mejorado.py #Rapido

#####  Sensor ruido

@reboot sudo python ruido.py


#####  EMPATICA E4

#@reboot sudo python serverudp_empatica.py

#####  POLAR H10

@reboot sudo python hdiez1.py
@reboot sudo python hdiez2.py

#####  Acelerometro MMA7455

@reboot sudo python accel.py &
```

Fig. 5.1.- Captura del archivo de configuración "Crontab" donde se pueden ver los diferentes scripts que se han creado para recoger datos de data dispositivo y los cuales se ejecutan en el arranque del sistema operativo de la Raspberry Pi 3.

Tal y como se puede apreciar en la figura 5.1 se han creado una serie de programas los cuales están clasificados principalmente en los siguientes bloques en función del dispositivo del que pretenden recoger datos:

1. Sensores ambientales
2. Sensor de ruido
3. Acelerómetro fijo
4. Módulo GPS/FONA
5. OBDII
6. Cinta Polar H10
7. Smartwatch Huawei
8. Pulsera Empatica E4

En todos estos programas existe una serie de elementos comunes. El primero de ellos sería un bucle principal que sigue una estructura similar en todos los programas y donde se incluye además un mecanismo de prevención ante errores normales en la adquisición de datos de manera que el programa no deje de recoger datos de manera inesperada. El esquema de este mecanismo preventivo puede verse en el esquema de la figura 5.2.

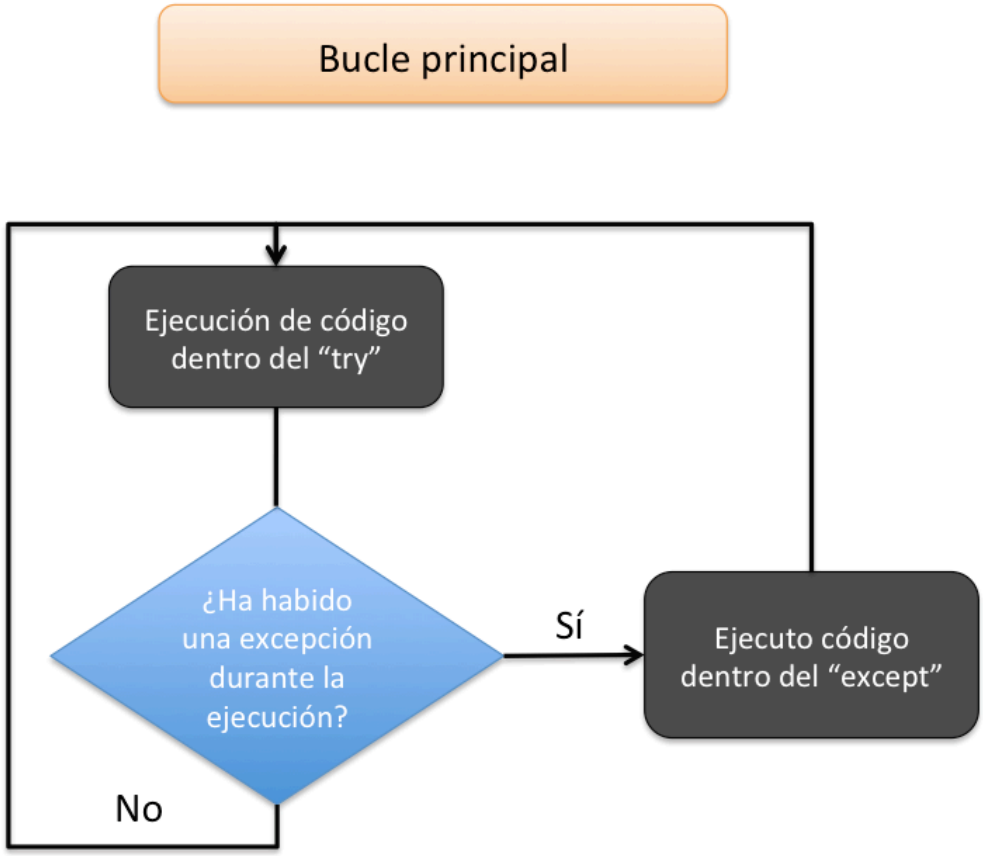


Fig. 5.2.- Diagrama de flujo del sistema de protección frente a errores normales "Try-Except" que se ha implementado dentro del bucle principal de todos los programas desarrollados.

Además, antes de entrar en el bucle principal, cada uno de los programas principales de cada bloque realiza las siguientes tareas:

- Importación de librerías necesaria
- Tiempo de espera de duraciones diferentes para asegurar correcta inicialización de todos los dispositivos implicados
- Conexión con base de datos SQLite donde se almacenarán los datos (misma BD para todos los datos)
- Definición de función para crear tantas tablas como tipos de datos diferentes que se vayan a recoger y almacenar en cada programa

- Definición de función para agregar un dato nuevo a una tabla
- Creación de las tablas pertinentes llamando a la función de crear tablas previamente definida
- Establecer valores iniciales de variables y constantes necesarias para el resto de la ejecución

5.1.2. Funcionamiento específico de cada uno de los programas

A continuación se tratará de explicar brevemente cada uno de los programas con el fin de entender como funciona cada uno de los dispositivos del sistema y de alguna manera justificar alguna de los hechos expuestos como la separación de algunos componentes pertenecientes al mismo circuito físico en programas de adquisición diferentes.

1. Sensores ambientales

En este bloque, tal como se ve en el archivo de configuración representado en la figura X, únicamente se tiene un script que es el denominado “Sensores.py”. En este programa se recogerán datos de los siguientes tres sensores:

- Sensor de temperatura y humedad (DHT-22)
- Sensor de nivel de luminosidad (LDR) [50]
- Sensor de calidad del aire (MQ-135) [46][49]

La primera tarea relevante de este programa, la cual tiene lugar justo antes de entrar en el bucle principal, es la etapa de calibración del sensor de calidad del aire MQ-135. Esta calibración se realiza con una función ya definida en la librería de los sensores de gases de la familia MQ que se ha empleado en este trabajo. Esta calibración debería de mejorar los resultados obtenidos con este sensor pero no es tan importante en este caso ya que no se pretende obtener la concentración concreta de un determinado gas en el aire sino que únicamente se medirá en tanto por ciento cómo de bueno es el aire en cada momento con el fin de mostrar tendencias o variaciones bruscas más que un valor preciso.

Ya dentro del bucle principal se registra la marca de tiempo y se va accediendo a los datos de cada uno de los tres sensores. La manera de acceder al dato de los sensores de calidad del aire y de nivel de luminosidad es directamente a través del valor de tensión devuelto por el conversor A/D en las patillas correspondientes a cada sensor. En el caso de los datos de temperatura y humedad su extracción se realiza mediante una librería propia de los sensores DHT-11 y DHT-22 que permite deducir el valor de ambos datos examinando directamente la patilla de entrada a la Raspberry Pi que está directamente conectada al sensor.

Antes de pasar al almacenamiento de datos, tanto el valor de tensión del sensor de nivel de luminosidad como el del sensor de calidad del aire han de ser convertidos a las unidades de interés en este caso a lux y a tanto por ciento respectivamente.

Para obtener el valor del nivel de luminosidad en lux es preciso tener en cuenta el conexionado eléctrico el cual se ha esquematizado en la figura 5.3. A partir de esta condición se puede obtener el valor convertido aplicando la expresión 5.1. [52]

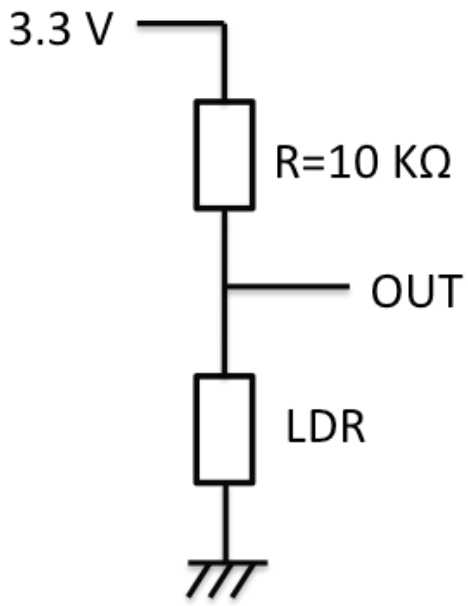


Fig. 5.3.- Esquemático del circuito electrónico en el que está involucrado el sensor LDR de nivel de luminosidad a partir del cuál es posible deducir la expresión (5.1) .

$$V_{OUT} = V_{IN} \cdot \frac{\frac{500}{Data_{lux}}}{\frac{500}{Data_{lux}} + R(K\Omega)} \rightarrow Data_{lux} = \frac{165}{V_{OUT}} - 50 \qquad (5.1)$$

Para la conversión del valor del dato de calidad del aire se ha realizado una calibración en función de los resultados obtenidos en las pruebas de modo que el valor normal en interiores esté en un valor comprendido entre el 50% y el 70% aproximadamente. Esta calibración no es una calibración que pretenda ser correcta en absoluto sino que se busca una calibración muy aproximada que permita ver variaciones en los datos obtenidos que puedan ser interpretadas de manera sencilla. Para realizar una calibración correcta habría que disponer de equipos de medida ya calibrados y/o de condiciones ambientales conocidas que son en ambos casos elementos que no se podrían reproducir fácilmente y que no son de demasiado interés para este trabajo.

Por último, una vez recogidos en las correspondientes variables los datos de interés, se llama a la función encargada de almacenar esta información en la base de datos con su correspondiente marca de tiempo.

Además tanto en este programa como en los del resto de bloques existe una parte de código al final que se encarga de hacer parpadear un LED. El objetivo de esto es el de que el usuario y sobre todo los desarrolladores puedan ver si los programas están recogiendo datos en realidad o si ha habido un error y es necesario abortar la sesión de pruebas y corregirlo.

2. Sensor de ruido

La manera de extraer los datos de interés a partir de este sensor es muy similar al proceso que se seguía con los anteriormente mencionados sensores de luminosidad y calidad del aire ya que se puede obtener la tensión de la patilla del conversor A/D correspondiente al sensor de ruido en este caso. Sin embargo se ha decido aislar a este sensor y generar un programa a parte para él, el denominado "ruido.py" en el archivo de configuración del cron. La razón de esta separación se debe a que el muestre que se necesita con el sensor de ruido es bastante mayor que el que se puede obtener con la serie de sensores de calidad del aire, luminosidad y temperatura y humedad.

El bucle principal consiste de nuevo en recoger la marca de tiempo en primer lugar, posteriormente tomar 20 muestras del valor de tensión de interés a lo largo de 0.2 segundos y extraer el máximo de esas muestras. Esto se ha realizado por la inestabilidad que se obtenía al seleccionar simplemente una muestra de cada vez ya que la señal que viene del sensor no se corresponde durante todo el tiempo con el valor de la amplitud del sonido sino que es un poco más compleja. Una vez seleccionado ese máximo ha de convertirse ese valor de tensión a decibelios con respecto a un valor determinado de referencia. Se ha obtenido ese valor de referencia como el valor de tensión que obtiene el sensor de media en un laboratorio del departamento en situación de relativo silencio. En concreto el paso a decibelios se ha realizado mediante la expresión 5.2.

$$Data_{dB} = 20 \log \frac{V_{Max}}{0.024} \tag{5.2}$$

El valor de ruido en decibelios junto a la marca de tiempo se agregan a la tabla correspondiente en la base de datos mediante la función creada.

Por último y antes de salir del bucle principal se pasa a la parte de control del parpadeo del led correspondiente a este sensor.

3. Acelerómetro fijo

De nuevo se tiene un caso de un sensor presente en la placa que se separa y se le crea un programa individual, en este caso el denominado "accel.py" en el archivo de configuración del cron. De nuevo la razón de este aislamiento es la necesidad de un muestro mucho más alto dado que el sensor y el bus comunicación lo permiten. La aceleraciones y frenazos bruscos ocurren de una manera muy rápida y por tanto está velocidad de muestreo elevada permitirá detectar ese tipo de situaciones.

En cuanto al programa en sí del acelerómetro, en primer lugar y antes del bucle principal se ha definido una función basada en una librería que se ha importado previamente para el control del bus serie I2C. Esta función se encarga de inicializar el acelerómetro y calibrarlo así como de devolver los valores de aceleración en los 3 ejes.

Dentro del bucle principal tras haber obtenido la marca de tiempo se procede a la llamada de esta función y esta ya devolvería el valor de las tres aceleraciones X, Y y Z.

Por último se añaden los datos con sus correspondientes marcas de tiempo a las tablas correspondientes de la base de datos de trabajo.

Cabe destacar que el muestro de este valor se ha fijado a poco más de una décima de segundo. [51][53]

4. Módulo GPS/FONA

En este bloque se dispone al igual que en los anteriores de un único programa independiente, en este caso sería el denominado "serial_gps.py". Este programa es ligeramente más complejo que los anteriores debido a que no existen (o al menos no se han

encontrado) librerías que faciliten el trabajo y es necesario desarrollar la comunicación serie e interpretar las respuesta desde cero en el script.

La primera tarea dentro del bucle principal consiste en establecer la conexión con el dispositivo a través de puerto serie "serial0" de la Raspberry Pi, concretamente a una velocidad de 19200 baudios, la cual resultó ser la que mejores resultados conseguía.

Una vez se establece la conexión se procede a enviar órdenes. La primera de ellas sería un "AT+CGPSPWR=1\r" que es para encender el módulo GPS por si no lo estuviera. En la segunda orden ya se procede a la solicitud de la información con la orden "AT+CGNSINF\r". La respuesta a esta orden sigue una estructura como la que se muestra a continuación: [45]

<GNSS run status>,<Fix status>,<UTC date & Time>,<Latitude>,<Longitude>, <MSL Altitude>,<Speed Over Ground>,<Course Over Ground>,<Fix Mode>, <Reserved1>,<HDOP>,<PDOP>,<VDOP>,<Reserved2>,<GNSS Satellites in View>,<GNSS Satellites Used>,<GLONASS Satellites Used>,<Reserved3>,<C/N0 max>,<HPA>,<VPA>

El paso siguiente por tanto será el de analizar esta respuesta y extraer los datos de interés que sería principalmente la latitud (<Latitude>), la longitud (<Longitude>), la fecha y hora (<UTC date & Time>), la altitud (<MSL Altitude>), el número de satélites con visión directa (<GNSS Satellites in View>) y el número de satélites usados actualmente (<GNSS Satellites Used>).

Por último se añaden los datos a las tablas correspondientes de la base de datos SQLite y se pasa por la etapa de parpadeo de los LEDs correspondientes al módulo GPS.

- **Modificación hora SBC**

Uno de los puntos importantes para el sistema es la marca de tiempo que se registra y asigna a todos los datos cuando se almacenan en la base de datos. Esta marca de tiempo ha de provenir de una única fuente y a ser posible ha de ser una marca de tiempo real.

Teniendo todo lo anterior en cuenta se ha decido emplear la hora obtenida mediante las señales de los satélites GPS para establecer la hora de la SBC o Raspberry Pi 3.

Una vez recogidos los datos se plantea una etapa de modificación de la hora extraída para adecuarla al horario en el estemos (verano o invierno). Es obvio que esta parte está programa para el año actual y habrá de modificarse para los días exactos de cambio de hora en años posteriores.

Una vez definida la hora se asigna dicha hora a la de la Raspberry Pi con el comando "date" y a partir de ahí todos los programas podrán estar referenciados a la hora del GPS.

5. OBDII [47]

Este programa (OBD_mejorado.py) es el que permite la comunicación Bluetooth entre el adaptador OBDII, conectado directamente al vehículo, y la Raspberry Pi 3.

Para establecer la conexión Bluetooth desde la Raspberry Pi se empleará el puerto "rfcomm0". Por lo tanto el primer paso de este código será el poner a dicho puerto a la espera de una conexión por parte del adaptador OBDII del cual se ha de indicar su dirección MAC.

El siguiente paso es el de pasar por una etapa de establecimiento de la conexión con el adaptador dentro de un bucle tal como se aprecia en el esquema de la figura 5.4.

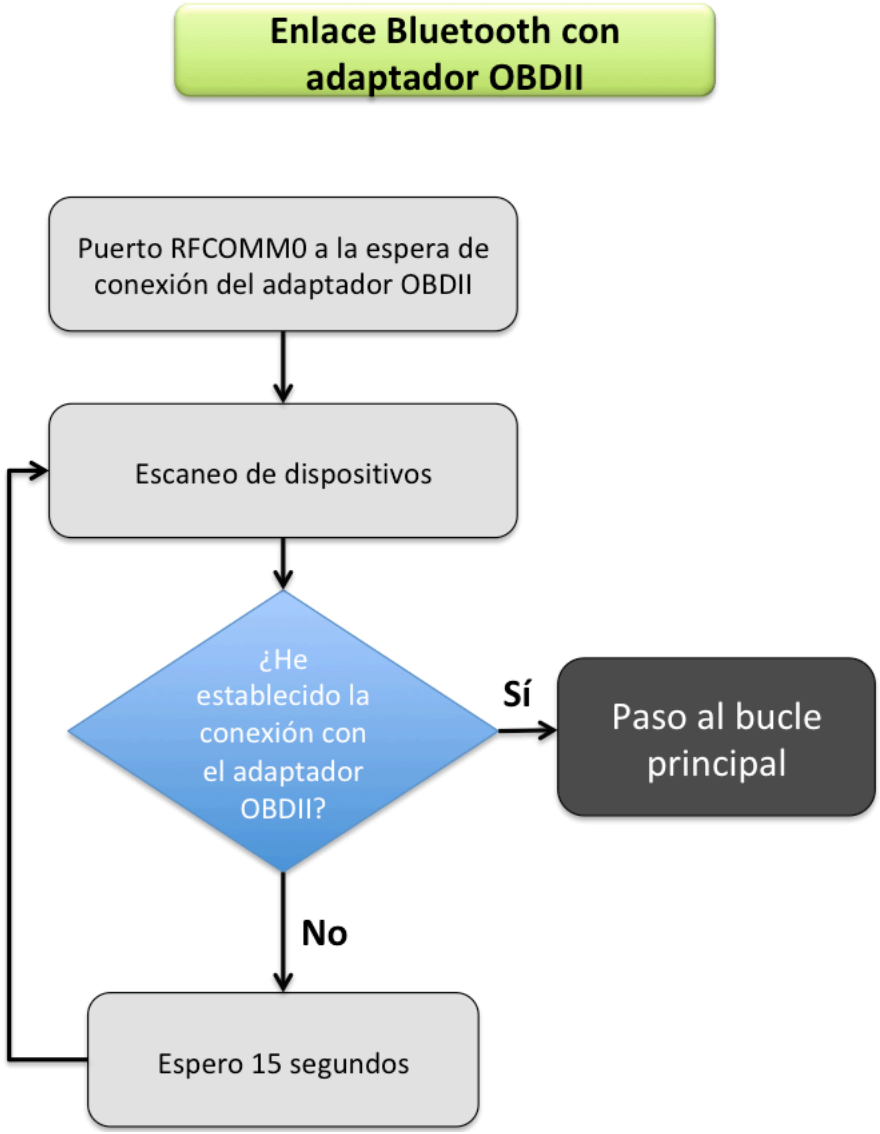


Fig. 5.4.- Diagrama de flujo de proceso de establecimiento de conexión con adaptador OBD-II Bluetooth.

Una vez dentro del bucle principal el proceso es idealmente simple. Este consiste en, registrar la marca de tiempo, realizar una petición Bluetooth al adaptador, gracias a una librería para trabajar con este tipo de adaptadores, y esperar por la respuesta para luego almacenarla en la base de datos.

El proceso que se acaba de explicar no tiene en sí complejidad sin embargo este código es uno de los que más problemas ha dado y también el que ha conllevado un mayor tiempo de desarrollo y una mayor evolución hasta tener el programa actual. El gran problema viene de la gran cantidad de datos disponibles sobre los cuales se pueden llegar a hacer consultas a través de OBDII. Esto nos lleva a que, si tenemos en cuenta que el tiempo aproximado del conjunto petición respuesta es del orden de 0.15 segundos y existen más de 60 tipos de peticiones diferentes el hacerlas todas llevaría directamente a un tiempo de refresco de datos tan importantes como la velocidad o las revoluciones del vehículo que superaría los 6 o 7 segundos. Este tiempo de refresco es totalmente inadmisible para este tipo de datos. Un punto a favor en este sentido es que una mayoría de estos datos no están disponibles en muchos vehículos de modo que al hacer la petición de estos parámetros no disponibles la respuesta siempre es algo inválido. De este modo sería posible no realizar la petición de estos datos, sin embargo surge otro problema. El problema sería que no todos los vehículos disponen de los mismos parámetros accesibles sino que esto depende mucho del fabricante y de su política de cesión de estos datos. De este modo no se puede crear un código tal como se estaba pensando que sea válido para cualquier vehículo. [44]

La solución propuesta es la siguiente:

- Se crea una lista donde se recogen todos los parámetros posibles
- Se hace una segunda lista paralela llena de unos que indican si hay que solicitar ese parámetro (“1”) o si no hay que solicitarlo (“0”)
- Se sigue el esquema de la figura 5.5

Con este esquema de funcionamiento se ha conseguido reducir el retardo de 7 u 8 segundos hasta quedarse en torno a los 2 segundos, ya muy cerca de los 1.5 segundos que suele ser el tiempo de actualización de la mayoría de datos en los vehículos que se han probado.

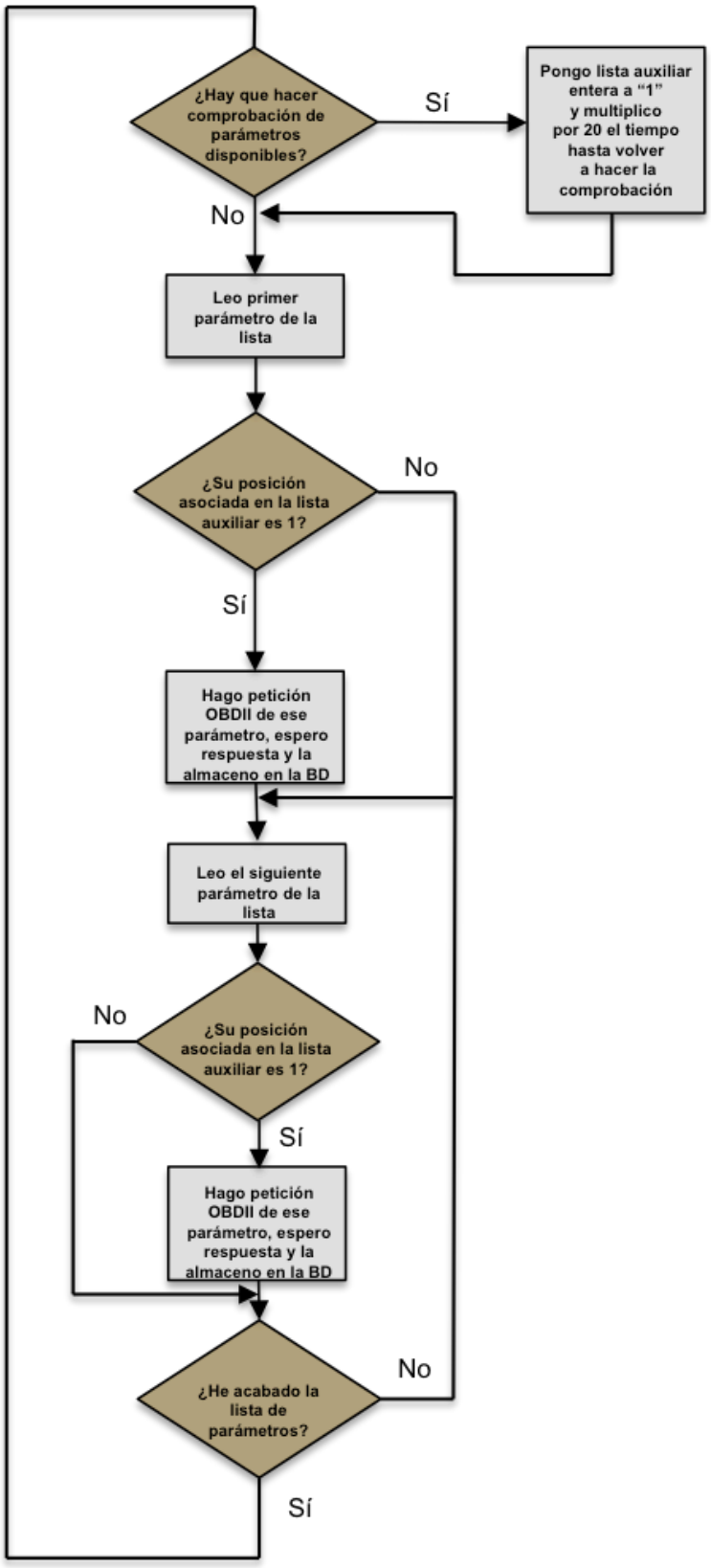


Fig. 5.5.- Diagrama de flujo de funcionamiento de la adquisición de datos OBD-II para optimizar el tiempo

6. Cinta Polar H10

En este caso, tal como se ve en el archivo de configuración "*crontab*" para la adquisición de datos de la banda pectoral Polar H10 ha sido necesario crear dos scripts diferentes. El primero de ellos, denominado "hdiez1.py" se basa únicamente en la ejecución de un mini programa en *perl* a través de la línea de comandos. Este mini programa emplea la herramienta "*Gatttool*" con la que es posible acceder a los servicios que corren dentro de los dispositivos Bluetooth. En el programa en *perl* se especifica a la herramienta que escriba un determinado valor en un determinado campo dentro del dispositivo Bluetooth que en este caso sería la banda Polar H10. El objetivo de esta escritura no será otra que decirle al dispositivo que se quiere recibir las notificaciones periódicas del servicio de medida de ritmo cardiaco. Además de escuchar y recibir estas notificaciones, el programa en *perl* indica que cada vez que se reciben esos datos desde el dispositivo estos sean almacenados en un archivo de texto, sobrescribiendo el contenido de este. [57]

Una vez se ha explicado el funcionamiento del primer programa de este bloque queda claro que si todo va bien se tendrá un archivo de texto que se va actualizando con una determinada información sobre el ritmo cardiaco a medida que el dispositivo envía notificaciones y la SBC las recibe. Esto lleva directamente al segundo de los códigos creados para este bloque. Este segundo código denominado (hdiez2.py) tiene la principal función de realizar lecturas del archivo de texto mencionado y analizar lo que lee con el fin de extraer los datos de ritmo cardiaco. Una vez identificado el dato ha de almacenarse en la tabla correspondiente de la base de datos *SQLite* como en el resto de bloques del sistema.

El periodo de emisión de la cinta Polar H10 es de aproximadamente un segundo con lo que el código de lectura de del fichero de texto sólo necesita leer cada un segundo ahorrando así carga al sistema.

Por último cabe destacar que, como ya se ha comentado no se registra únicamente el ritmo cardiaco sino también el intervalo entre pulsos (RR).

5.2. Software de adquisición y transmisión de datos desde un dispositivo externo a la SBC

En esta parte entrarían los dos bloques restantes que se corresponden con el *Smartwatch Huawei Watch* 2 y la pulsera *Empatica* E4. La razón de la apertura de un apartado a parte o diferente para estos dos dispositivos radica en que para la extracción de sus datos es necesario la ejecución de programas en dispositivos externos a la propia SBC.

7. Smartwatch Huawei

Tal como se ha explicado en apartados anteriores, el objetivo del uso de este dispositivo *wearable* es el de extraer datos de ritmo cardiaco principalmente pero además datos también de aceleración, de rotación de la muñeca del conductor y de posicionamiento GPS. Para ello es necesario el uso de los sensores incorporados en el dispositivo.

El sistema operativo de este dispositivo es una versión de Android, concretamente la denominada Android *Wear*. Surge por tanto la idea de que para tener acceso y control de estos datos es necesario emplear algún tipo de aplicación Android y, dado que se trata de un proyecto en desarrollo con necesidades particulares, es necesario contar con una aplicación que cumpla con los objetivos concretos requeridos. Estos objetivos serán principalmente dos:

- Adquisición de datos en tiempo real de los 4 sensores de interés (medidor de pulso, acelerómetro y giroscopio y GPS).
- Envío de estos datos al sistema central embarcado de manera inalámbrica para su posterior almacenamiento y tratamiento.

Algunos objetivos secundarios de la aplicación serían:

- Bajo consumo de batería
- Facilidad en su lanzamiento

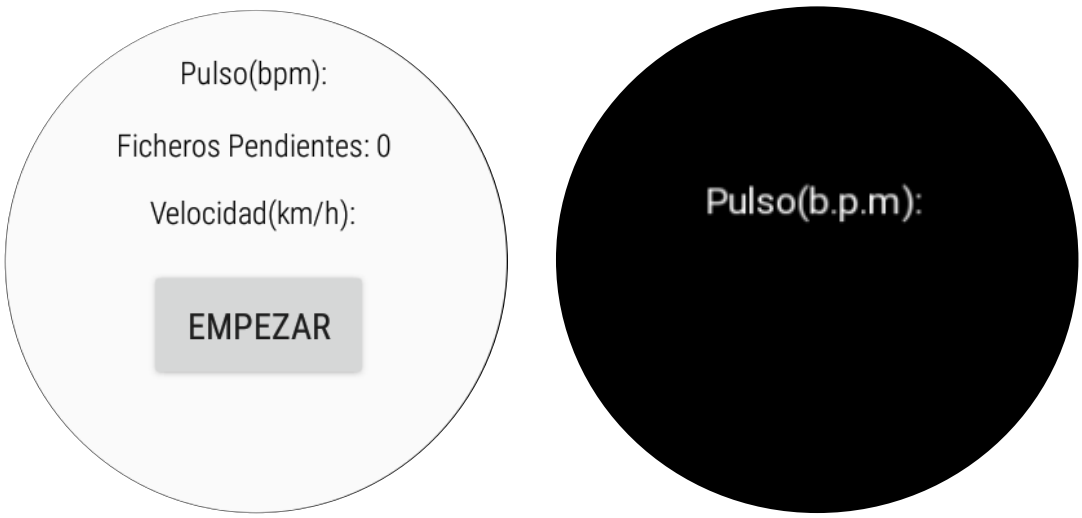
En el anexo F se exponen las partes de la aplicación que se han programado así como las más importantes de esta aplicación, de todas formas a continuación se tratará de explicar en que ha consistido el trabajo de programación en esta parte.

Para el desarrollo de esta *app* no se partió desde cero sino que se trató de modificar una aplicación creada para este dispositivo pero para un proyecto diferente en la cual se realizaba toda la parte de adquisición de datos.

Una vez se dispuso del proyecto de generación de esta aplicación hubo que añadir principalmente dos tareas y anular algunas de las tareas que se realizaban en la aplicación antigua como la posibilidad de almacenamiento de datos que a priori no será necesaria en este caso. La primera se basaba en el desarrollo de un cliente UDP encargado de enviar los datos recogidos de los sensores al servidor que estará implementado y esperando mensajes en el sistema central embarcado (Raspberry Pi), parte de la que se hablará un poco más adelante. La segunda tarea tiene que ver con el inicio cómodo y sencillo de la aplicación para el conductor.

Para el desarrollo de la primera tarea ha sido necesario crear una nueva clase denominada “EnviarMensajem.java” la cual se puede ver completa en el anexo F.3 y cuyas funciones básicas son las de cualquier cliente UDP normal, es decir, establecer el puerto y la dirección IP destino del mensaje, formar el datagrama a enviar con el mensaje que se le indique y finalmente enviar dicho datagrama al puerto y dirección definidos previamente. Esta clase será empleada cada vez que se realice la adquisición de los datos para enviárselos al servidor. [54-55]

Para el desarrollo de la segunda tarea se pensó que la manera más simple y cómoda para el usuario de lanzar esta aplicación sería directamente no tener que lanzarla. Esto podría realizarse de algún modo al arrancar el vehículo cuando el dispositivo *wearable* identificara la presencia de la Raspberry Pi principal, sin embargo esto haría que el usuario tuviera que ir con el dispositivo encendido permanentemente lo cual no sería lo ideal si no fuese porque los relojes que se están empleando en este proyecto disponen de una autonomía relativamente baja y no conviene malgastarla fuera del vehículo. En consecuencia se ha decidido implementar o más bien incluir una clase que permita a la aplicación arrancar de manera autónoma cuando el reloj se enciende. [56] De este modo se consigue ahorrar batería sin que el lanzamiento de la *app* sea demasiado complicado para el usuario ya que únicamente tendrá que encender su dispositivo *wearable* cuando se introduce en su vehículo.



a) b)

Fig. 5.6.- Representación de la modificación de la interfaz de usuario de la aplicación Android para la recogida del ritmo cardiaco con el *smartwatch* Huawei. De la interfaz antigua (a) sólo se conserva la información de pulso que es el dato que se recoge fundamentalmente y el fondo de la pantalla pasa de blanco a negro para conseguir un aumento de la autonomía de la batería con la aplicación corriendo. Se consigue finalmente una interfaz como la que se muestra en b).

La última modificación realizada sobre la aplicación de partida puede entenderse de manera visual gracias a la figura 5.6. En la aplicación de partida se tenía una interfaz como la que se muestra en la figura 5.6 (a). Sin embargo muchos de los campos no son necesarios en la *app* modificada, tanto es así que el único que puede resultar útil es la muestra del pulso en tiempo real con el único fin de comprobar que la aplicación está recogiendo datos correctamente. Por otro lado se aprecia que se ha cambiado el color del fondo de blanco a negro, la razón de este cambio tiene que ver con el ahorro de batería que como se comentará más adelante es uno de los puntos del uso de este dispositivo que más problemas genera. Con este simple cambio es posible reducir el consumo de batería.

En el lado de sistema central serán necesarios también un par de programas en python tal como se aprecia en el archivo de configuración *crontab*, denominados "serverudp_huawei.py" y "serverudp_huawei_2.py". Ambos son sencillos servidores UDP que esperan por tramas enviadas desde el reloj Huawei. La diferencia entre ambos programas es el puerto en el que esperan. La razón de esta diferenciación radica en que los datos de aceleración y rotación no se generan en intervalos fijos de tiempo mientras que los datos de ritmo cardiaco sí que deben hacerlo. Si se ejecuta todo en el mismo código y se reciben todos los datos por el mismo puerto es posible que en ocasiones se generen colas indeseadas que generen retardos en la recepción que interfieran en los tiempos de almacenamiento de los datos de ritmo cardiaco. De todas formas se ha puesto un limitador para los sensores de aceleración y rotación de modo que sólo envíen mensajes cuando se produzcan movimientos bruscos ya que el muestreo de estos sensores es muy rápido y podría generar esas colas en recepción que no son de interés ya que tal como se ha diseñado el código la marca de tiempo se pone en el momento en el que se recibe el dato en el servidor para que no existan diferentes relojes que suministren la marca de tiempo en los diferentes bloques.

Por otro lado, los datos de posicionamiento GPS se recogen también dentro del programa de recogida del ritmo cardiaco.

8. Pulsera *Empatica* E4

El primer punto abordar en este apartado es el porqué de incluir este dispositivo en la parte de dispositivos externos ya que, aparentemente podría comportarse de manera muy similar a cómo lo hace la cinta pectoral Polar H10.

La principal razón para incluir este dispositivo en este apartado tiene que ver con la hermeticidad del dispositivo que el fabricante a diseñado. En un primer momento se trató de extraer los datos y establecer una conexión con el dispositivo vía Bluetooth desde la Raspberry Pi principal sin embargo el proceso de acceso a los datos y de establecimiento de conexión en esta pulsera de actividad no tienen nada que ver con lo que se daba en la cinta Polar. En este caso el sistema es mucho más cerrado y ha de emplearse necesariamente una api suministrada por el propio fabricante.

Las apis proporcionadas por el fabricante están en Android y eso genera un gran inconveniente al tener todo el resto del sistema funcionando sobre un sistema operativo Raspbian. Existía la opción de tratar de replicar el funcionamiento de esa api sobre el sistema operativo de la Raspberry Pi principal pero esta opción por el momento se ha descartado principalmente por falta de tiempo para dedicar a esta tarea.

La opción que parecía más factible al menos para comenzar a probar las ventajas de la integración de este dispositivo en el sistema era la de desarrollar una arquitectura como la que se esquematiza en la figura 5.7. En este esquema puede verse como ha sido necesario incluir una segunda SBC intermedia entre el dispositivo y la SBC principal. El hardware de esta segunda SBC será el mismo que el de la principal, es decir será otra Raspberry Pi 3 y la diferencia estará basada en el sistema operativo que corre sobre ella que se sabe que tendrá que ser Android pero en la versión concreta se centrará un subapartado posterior.

Fig. 5.7.- Esquema básico de la transmisión de datos y las interconexiones entre los diferentes dispositivos para la adquisición y almacenamiento de datos biométricos con pulsera *Empatica* E4.

La arquitectura y el funcionamiento en sí del sistema consistiría en los siguientes pasos:

1. Establecimiento de conexión Bluetooth entre pulsera *Empatica* y SBC secundaria. Entrada de pulsera *Empatica* en modo *Streaming*
2. Adquisición de los datos transmitidos por la pulsera por parte de la SBC secundaria
3. Reenvío de los datos recogidos por la SBC secundaria hasta la SBC principal en tiempo real y a través de un proceso cliente-servidor UDP similar al que se daba en la conexión con el *Smartwatch* Huawei
4. Almacenamiento en tiempo real de los datos recibidos en por el servidor UDP en la base de datos *SQLite* de la Raspberry Pi principal.

Este es el funcionamiento del sistema básico pero es necesario hablar de tres problemas extra que presenta el empleo de este dispositivo *wearable* además de tener que emplear esta nueva arquitectura. Estos tres problemas están relacionados directamente en primer lugar con la fase de establecimiento de conexión o emparejamiento entre la pulsera Empatica E4 y la SBC secundaria, en segundo lugar con la compatibilidad y fiabilidad de los sistemas operativos Android sobre Raspberry Pi y en tercer lugar con la necesidad de verificar la clave del producto para poder darle uso. A continuación tratarán de explicarse cada uno de esos problemas por separado.

- **Problema 1: Fase de establecimiento de conexión y puesta en modo *Streaming* de la pulsera *Empatica* E4**

Este problema está relacionado directamente con el nivel de interacción necesaria entre usuario y dispositivo *wearable* para que este pueda funcionar correctamente en el sistema. Este nivel si se habla de la pulsera de actividad *Empatica* E4 es muy elevado principalmente por el tiempo que necesita para llevarse a cabo la conexión de este dispositivo.

El dispositivo tiene por así decirlo 3 fases en su proceso de iniciación del modo *streaming* en el cual la pulsera emite de manera periódica señales bluetooth con la información que recoge de todos sus sensores.

- **Fase "desconectado":** En esta fase la api no ha verificado todavía la clave del producto y no es posible establecer la conexión.
- **Fase de escaneo:** En esta fase la api ya ha verificado la clave del producto y permanece escaneando por intentos de emparejamiento bluetooth. En este momento es necesario pulsar el botón de encendido de la pulsera. Una vez pulsado el botón la api detecta el intento de conexión por parte de este dispositivo y verifica si es el dispositivo asignado a la clave. De ser así se establece la conexión de manera satisfactoria y se pasa a la fase de conexión.
- **Fase de "conexión":** En esta fase se ha producido ya el emparejamiento entre dispositivo y api ejecutada sobre el receptor bluetooth. El dispositivo se encuentra enviando datos periódicamente en modo *streaming*.

Para indicar al usuario en qué fase se está en cada momento se han empleado una serie de LEDs. El usuario ha de esperar a que se enciendan los LEDs que indican que se ha entrado en la fase de escaneo para encender su pulsera y esperar a que la conexión se realice de manera correcta, lo cual se verifica si se encienden los LEDs correspondientes.

Para el encendido de cada LED se ha programado en el cliente UDP de la Raspberry Pi secundaria el envío de unos mensajes de aviso diferentes para cada fase de manera que cuando el servidor de la SBC principal los reciba pueda actuar en consecuencia y encender los LEDs apropiados.

Este proceso lleva un tiempo que sin producirse errores puede llegar a los 2.5 minutos y que de producirse algún error puede alargarse varios minutos más ya que es necesario producir un reseteo del sistema. Estos posibles errores vienen derivados de los otros dos problemas que a continuación se explicarán.

- **Problema 2: Compatibilidad y fiabilidad de sistemas operativos Android sobre Raspberry Pi**

Tal como se explicó en los capítulos de selección de los componentes, uno de los principales inconvenientes de las SBC de tipo Raspberry Pi frente a otras contrincantes como la Orange Pi es su compatibilidad limitada con sistemas operativos Android ya que no existen versiones con todas las funcionalidades de Android disponibles para instalar en Raspberry Pi con un soporte detrás, la única que actualmente tiene soporte (de Google en este caso) es Android *Things* pero aún está en pleno desarrollo y en el momento de ejecución de este trabajo las funcionalidades con las que contaba eran extremadamente limitadas.

Sí que existen sin embargo diferentes imágenes de sistemas operativos Android para instalar en Raspberry Pi sin embargo o bien no tienen soporte o bien tienen el soporte de Android pero sus funcionalidades son pocas. En este proyecto se han llegado a probar varios de estos sistemas operativos con el fin de ver si alguno de ellos era válido totalmente para el fin de este proyecto o en su defecto cuál de ellos era el que presentaba menor número de deficiencias.

A continuación se resumen las versiones de Android con las que se ha probado analizando sus ventajas e inconvenientes en relación a la finalidad que se busca en este proyecto.

- **Android *Things*:** Es el sistema operativo Android más prometedor para aplicaciones de Internet de las Cosas y fundamentalmente para su uso en Raspberry Pi 3. Su gran potencial tiene que ver con que tiene detrás el soporte de la comunidad de Android *Developers* y es un sistema operativo lanzado por Google. Desafortunadamente, pese a que el potencial del sistema es grande, durante el desarrollo de esta proyecto también este sistema operativo se encontraba en fase de desarrollo y las versiones que fueron sacando no tenían las funcionalidades requeridas por esta aplicación. De hecho hasta la última versión este sistema operativo no aportaba la funcionalidad de Bluetooth lo cual hace inviable su uso para esta aplicación.

  De nuevo es conveniente remarcar que, el hecho de no haberla podido emplear hasta el momento no es una razón para dejar de estar pendiente de la evolución de este SO para su posible futuro uso. [48]

- **Android Emteria:** Se ha probado son diferentes versiones de este sistema operativo. Este sistema presenta aparentemente las funcionalidades de cualquier sistema Android incluyendo una interfaz gráfica que facilita mucho las pruebas y la instalación de aplicaciones (Android *Things* no dispone de tal interfaz por le momento). El problema de este sistema tenía que ver aparentemente con la funcionalidad Bluetooth ya que no era posible el paso de la fase 2 a la fase 3 en la conexión de la pulsera Empatica E4. En la parte de escaneo el la api instalada en este sistema operativo era incapaz de encontrar al dispositivo de manera que nunca se llegaba a establecer una conexión y por tanto no se pasaba a la fase 3 de *streaming* de datos.

- **Android Lineage:** Este es el tercer sistema operativo que se ha probado. La apariencia en cuanto a lo que interfaz se refiere es muy similar a los sistemas operativos Android comúnmente utilizados y también al Android Emteria. En este caso la funcionalidad de Bluetooth está disponible y operativa totalmente después de las pruebas realizadas. La otra parte importante del sistema que sería el Wi-Fi es la que ha generado algún tipo de problema.

Es necesario que el sistema se conecte automáticamente en el arranque al punto de acceso Wi-Fi creado en la Raspberry Pi principal y sobre todo que se mantenga conectado de manera permanente, sin desconexiones momentáneas. Tras la realización de varias pruebas se puede concluir que el Wi-Fi es capaz de conectarse al punto de acceso en el arranque pero no el 100% de las veces. Tampoco se mantiene la conexión estable durante el 100% de las veces que se establece la conexión. Este hecho ha dado lugar a que se puedan realizar pruebas y recogidas de datos de pruebas que se expondrán más adelante pero no permite su uso en un prototipo final por la poca seguridad y robustez que ofrece.

De todos modos, es el único sistema operativo con el cual se ha llegado a establecer la conexión entre todos los dispositivos de la manera que se preveía.

En cuanto a las posibles soluciones, puesto que el problema está en la conexión Wi-Fi, necesaria para el envío de datos entre las dos *SBCs* pero no para la adquisición de los mismos en la SBC secundaria, se ha tratado de buscar la forma de conectar ambas SBC vía Ethernet pero no se ha conseguido debido a las dificultades encontradas para asignar una dirección IP concreta al puerto Ethernet desde el sistema operativo Android. Sí que se ha conseguido sin embargo la conexión a través de un router con lo cual la inclusión de un router de dos puertos de pequeñas dimensiones podría ser una opción factible para solucionar el problema de interconexión de *SBCs*.

- **Problema 3: Verificación de clave del producto en fase de conexión**

El último problema es probablemente también el que presenta una solución más sencilla. El problema tiene que ver con que en el proceso de conexión, la api de *Empatica* tiene que verificar la existencia de la clave del producto y para ello tiene que realizar una petición web por lo que es necesario que el sistema disponga de conexión a internet.

La conexión a internet del sistema es un apartado que será necesario cuando el proyecto entre en fases posteriores pero que aún no se ha implementado.

Por otro lado se ha comprobado que esta petición sólo es realizada por la api la primera vez que se registra una determinada clave. A partir de ahí la confirmación de esa clave queda almacenada en la memoria de la aplicación y ya no es necesaria la conexión a internet. De este modo para las primera pruebas realizadas se ha registrado la clave en el laboratorio con conexión a internet y a partir de ahí ya se ha podido trabajar sin problemas en modo offline.

6. Validación experimental

Antes de exponer algunos de los principales resultados obtenidos durante el transcurso de las diferentes pruebas y validaciones experimentales realizadas principalmente con los dos prototipos finales del sistema cabe hacer algunas consideraciones.

En primer lugar es necesario destacar que no se han realizado una pruebas concretas al finalizar el prototipo con el fin de realizar una validación del mismo al completo sino que este proceso de validación de cada una de las partes del prototipo ha ido prácticamente a la par que el desarrollo de los propios prototipos. Y es que ambos ámbitos están directamente relacionados. De este modo el desarrollo de los prototipos dentro del cual una aprte fundamental es la corrección de errores ha estado basada casi en su totalidad en las pequeñas pruebas experimentales que se han ido realizando sobre el sistema casi desde el primer día. Así, casi todo cambio software o hardware en el sistema se ha hecho al detectar en alguna de las pruebas alguna deficiencia o algún punto de mejora. Del mismo modo la manera de recoger los datos y las pruebas experimentales se han ido adecuando a las necesidades del prototipo de forma que se pudieran detectar fallos o prever diferentes situaciones que se puedan llegar a dar en escenarios reales.

Dado que el objetivo final de este trabajo es la recogida en sí de datos, siendo esta correcta, no se ha llegado a hacer un análisis ni un tratamiento exhaustivo de los datos de las pruebas en su conjunto Sin embargo si que se ha ido comprobando durante las diferentes fases del desarrollo y por supuesto sobre los prototipos finales, que estos datos recogidos son fiables en lo referente a que concuerdan con las condiciones o con el entorno en el que se han realizado las medidas de los mismos.

Tal como se ha visto en los capítulos 4 y 5, se ha estudiado el uso de diferentes dispositivos *wearables* para la adquisición de datos biométricos. En este punto se descartarán algunos de ellos y se justificará esa decisión.

Por todo lo que se acaba de exponer a continuación se procede al análisis de la calidad de los datos obtenidos con los distintos prototipos implementados (principalmente con el prototipo basado en TS *breadboard* (figura 4.21) y el basado en TS PCB (figura 4.43)) en lo que se refiere a la evaluación de parámetros tales como el tiempo de muestreo de los datos, sus retardos o su precisión. Esto no impide que se vayan a analizar algunos de los datos obtenidos con el fin de adelantar algunas conclusiones al estudio posterior.

6.1. CONSIDERACIONES GENERALES

6.1.1 Situación en cabina del prototipo

Uno de los puntos a tener en cuenta a la hora de realizar las pruebas es la colocación del equipo en el interior del vehículo. Parece claro que algunos datos pueden verse afectados por la orientación del equipo, la estabilidad y en definitiva por la posición exacta del mismo. Así no se obtienen los mismos resultados de nivel de luminosidad si se coloca el equipo sobre el salpicadero que si se coloca en el suelo del asiento trasero.

Para determinar la posición óptima para las pruebas se han tenido en cuenta muchos factores que indicaban que el salpicadero era el lugar más adecuado, sin embargo la estabilidad del sistema frente a aceleraciones o frenazos bruscos no es ni mucho menos buena y existe la posibilidad de desplazamientos o caídas del sistema durante el trayecto lo que daría al traste con las pruebas. De este modo se decidió colocar el prototipo en todas

las pruebas en el sobre el asiento trasero derecho pudiendo de este modo tener una mayor seguridad de la integridad del equipo durante las pruebas.

Lo que parece claro es que una vez elegido el lugar de colocación del sistema este ha de permanecer fijo para todas las pruebas que se realicen en el futuro al menos en el mismo vehículo para poder realizar comparativas adecuadas.

6.1.2 Base de datos de almacenamiento de resultados

Si se habla del almacenamiento de los datos localmente en la propia SBC principal del sistema embarcado el proceso y el formato de almacenamiento es el mismo para todas las pruebas realizadas a lo largo del trabajo.

Como se ha comentado en los primeros capítulos de este documento, se ha decidido emplear bases de datos *SQLite* principalmente por su velocidad al ser esta necesaria para trabajar en tiempo real de manera simple.

Se ha decidido emplear una tabla para cada dato y una fila para cada nueva entrada. En cada fila por lo general se almacenará la marca de tiempo, un nombre para el dato y el propio valor de dicho dato.

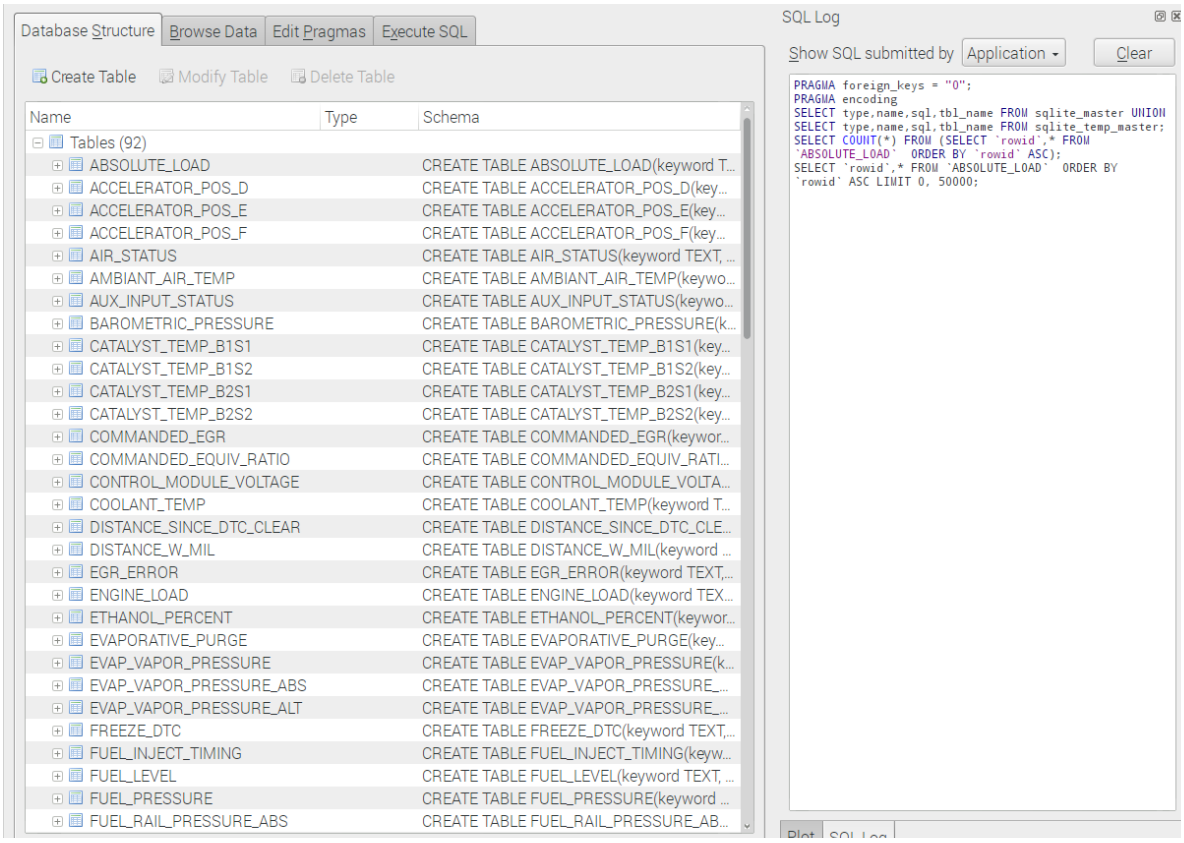


Fig. 6.1.- Ejemplo de tablas que forman una de las bases de datos obtenida durante las pruebas de alguno de los prototipos creados durante este trabajo. Imagen obtenida con *SQLite Database Browser*.

De este modo todos los resultados obtenidos tendrán una estructura o una apariencia muy similar a la que se muestra en las figuras 6.1. y 6.2 donde respectivamente se pueden ver el conjunto de tablas que forman una de las bases de datos de pruebas y los datos que componen unas de esas tablas, concretamente la tabla de revoluciones por minuto del vehículo que es un tipo de dato mecánico obtenido mediante OBD-II.

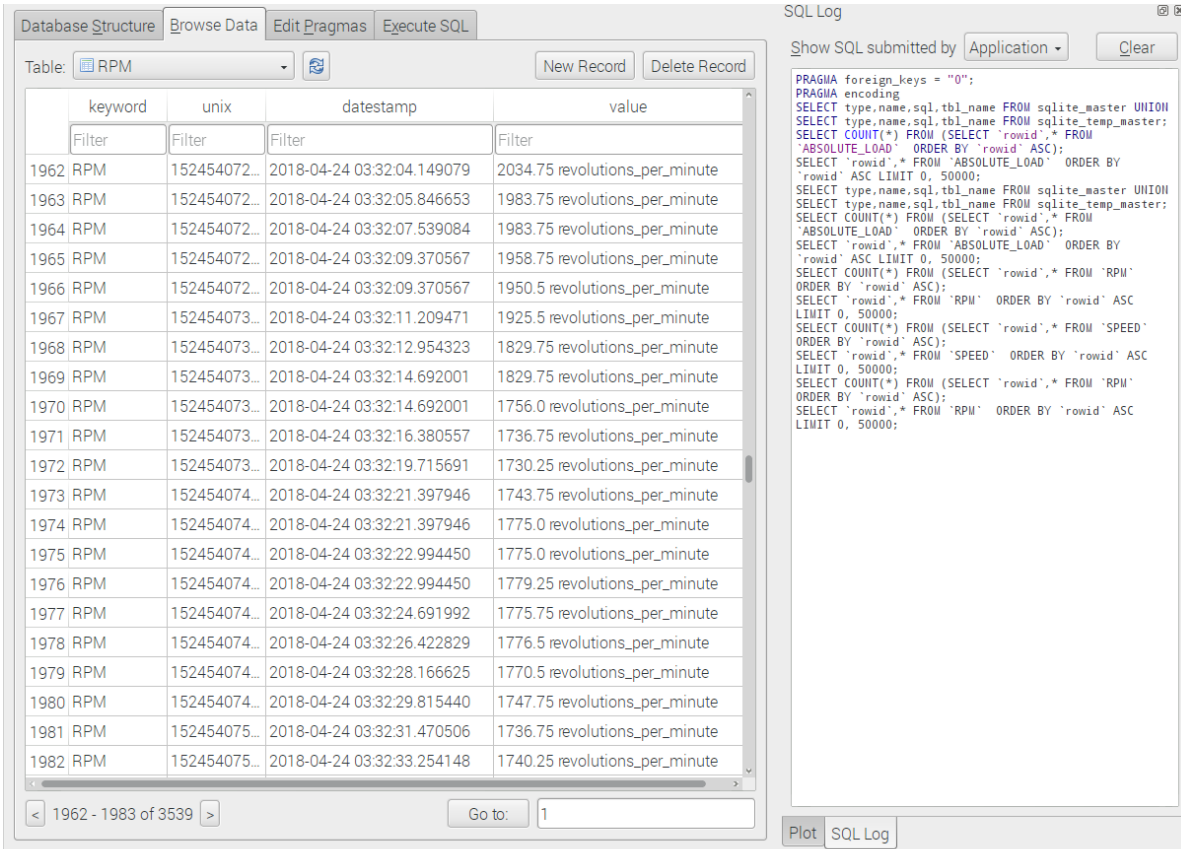


Fig. 6.2.- Ejemplo de tabla de datos correspondiente a las revoluciones por minuto (RPM) del vehículo. Datos obtenidos durante las pruebas de alguno de los prototipos creados durante este trabajo. Imagen obtenida con *SQLite Database Browser*.

6.2. EVALUACIÓN DEL PROTOTIPO BASADO EN TS "BREADBOARD"

Sobre este prototipo ha sido sobre el cual se ha realizado el grueso de las pruebas. El objetivo principal de este hecho no era otro que el de tratar de detectar y subsanar la mayor parte de problemas, o al menos los más evidentes, antes de implementar el prototipo basado en PCB sobre el que las modificaciones son mucho más costosas en tiempo y en esfuerzo.

En la tabla 6.1 se muestra la relación de datos que se recogen con este prototipo así como sus tiempos de muestreo experimentales. De manera clara se aprecia que algunos de los datos son obtenidos con un periodo bajo y además fijo como podría ser el caso del ritmo cardiaco procedente de la cinta Polar H10 pero también vemos otros muchos datos cuyos periodos de adquisición son muy elevados o son muy variables.

Dato	Tiempo de muestreo	Tipo
Posición D acelerador	2-4	OBD-II
Posición E acelerador	2-4	OBD-II
Acelerómetro (X, Y, Z)	0.5-10	Sensores ambientales
Calidad del aire (%)	0.5-10	Sensores ambientales
Heart rate Huawei *Watch*	5-8	*Smartwatch*
Humedad (%)	0.5-10	Sensores ambientales
Latitud módulo FONA 808	2-4	Módulo GPS/GSM
Longitud módulo FONA 808	2-4	Módulo GPS/GSM
Latitud Huawei *Watch*	5-8	*Smartwatch*
Longitud Huawei *Watch*	5-8	*Smartwatch*
Nivel de luminosidad (lux)	0.5-10	Sensores ambientales
MAF (Sensor de flujo de aire)	2-4	OBD-II
Carga motor	2-4	OBD-II
Presión admisión aire	2-4	OBD-II
Heart rate Polar H10	1	Cinta pectoral Polar
RPM	2-4	OBD-II
Nivel de sonido/ruido	0.5-2	Ruido
Tiempo de trayecto (*Run time*)	2-4	OBD-II
Temperatura	0.5-10	Sensores ambientales
Temperatura del aire de admisión	2-4	OBD-II
Temperatura refrigerante	2-4	OBD-II
Velocidad	2-4	OBD-II

Tabla. 6.1.- Datos recogidos con prototipo basado en TS "breadboard" con tiempos de muestreo asociados a cada una de ellos.

Con este prototipo se realizó una prueba a gran escala consistente en una serie de trayectos por la geografía española durante un viaje entre Asturias y Almería de ida y

vuelta. En las figuras 6.3 y 6.4 se pueden ver los datos de posicionamiento GPS recogidos por el módulo GPS FONA 808.

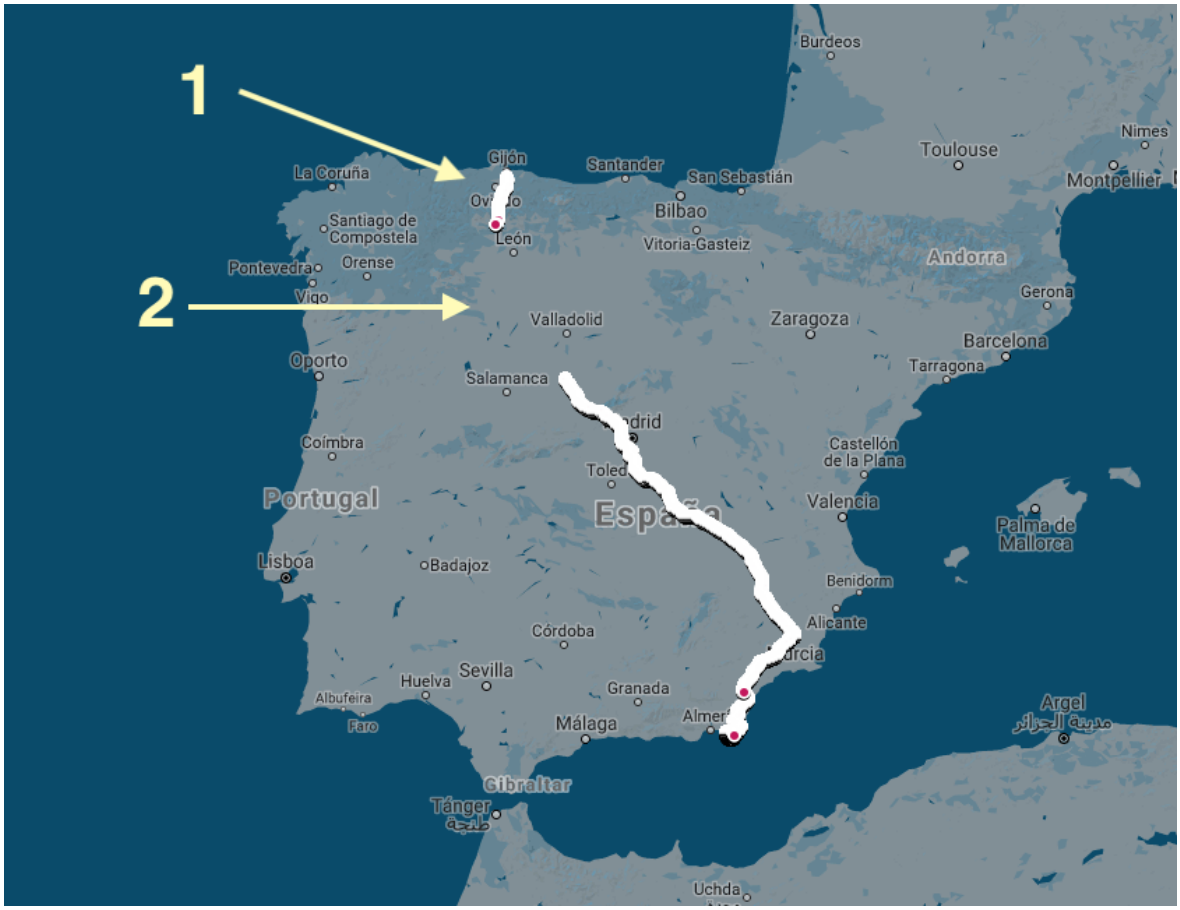


Fig. 6.3.- Datos de posicionamiento GPS recogidos por el módulo GPS FONA 808 incluido en el prototipo TS "*breadboard*". Datos recogidos durante los diferentes trayectos del viaje de ida entre Asturias y Almería.

Si se hace un pequeño análisis de los datos recogidos se pueden hacer varias consideraciones que tienen relación con los zonas que se han numerado en los mapas de las figuras 6.3 y 6.4. En la zona 1 de ambos mapas simplemente se indica que en la zona de Asturias se vuelven a tener muestras de posicionamiento. En la zona 2 de ambos mapas se ve un vacío lo cual parece indicar que el módulo GPS no ha sido capaz de obtener información al atravesar la zona de Castilla y León. Sin embargo si se comprueban otros tipos de datos para estas mismas zonas e instantes franjas temporales se observa que no existen información de ninguno de ellos lo cual indica y así se ha verificado con el usuario

que realizó las pruebas que el sistema no había estado conectado durante estos tramos. En la zona 3 también se ven una serie de puntos correspondientes a medidas que tienen errores en la obtención de la longitud, pero como se observa son errores aislados que pueden deberse a diferentes razones.

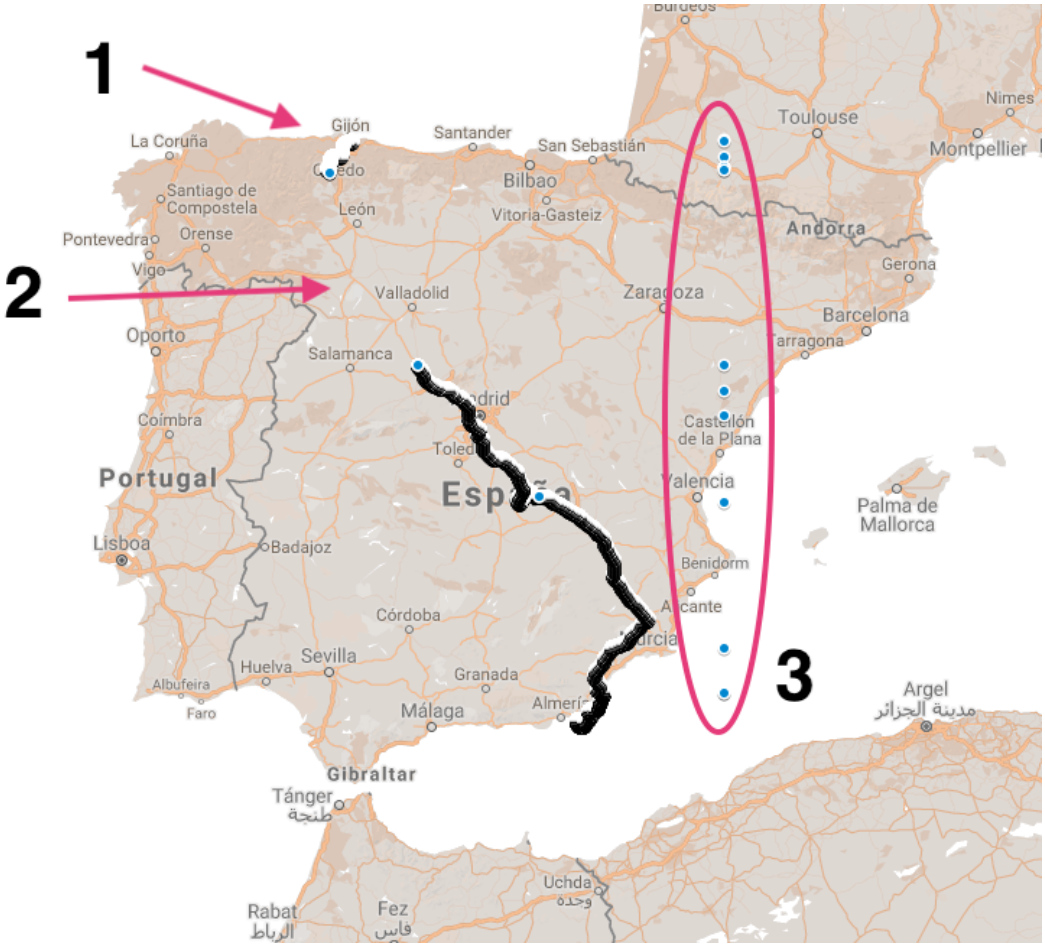


Fig. 6.4.- Datos de posicionamiento GPS recogidos por el módulo GPS FONA 808 incluido en el prototipo TS "*breadboard*". Datos recogidos durante los diferentes trayectos del viaje de vuelta entre Almería y Asturias.

Este ejemplo da una idea de la capacidad de análisis que aporta el tener una gran cantidad de datos y de muy diversos orígenes.

Dado que el objetivo del trabajo no es el análisis propiamente dicho de los datos para obtener tendencias, se mostrarán algunos de los datos obtenidos durante este viaje de pruebas para observar el desempeño del prototipo.

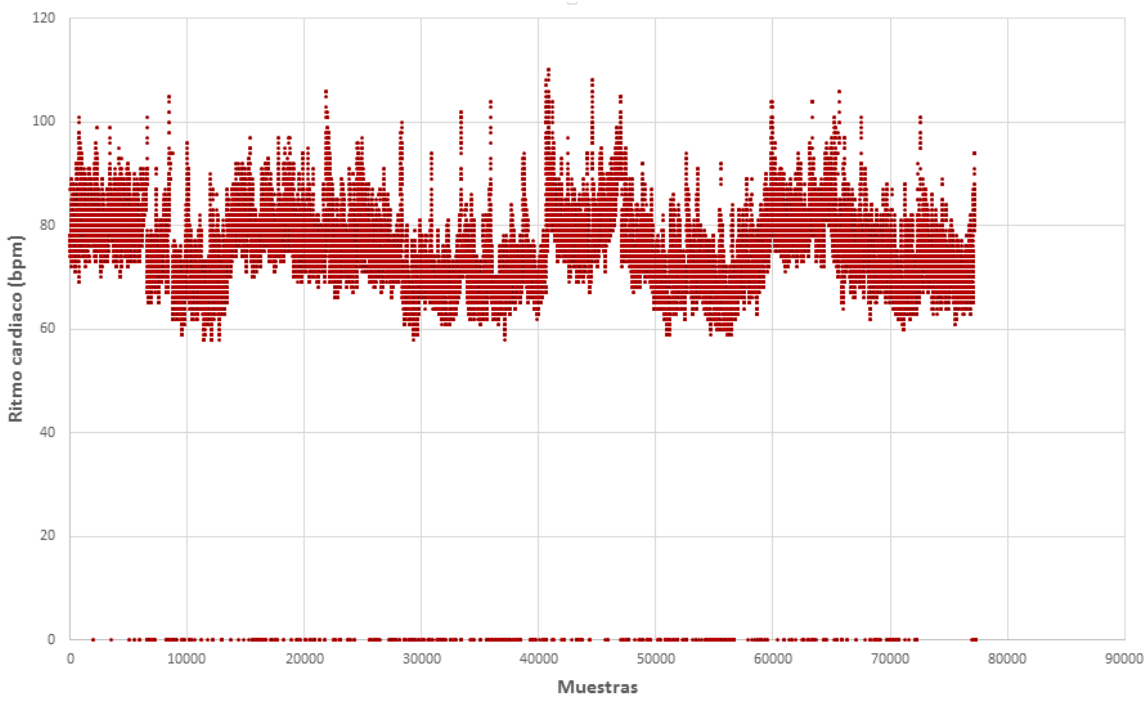


Fig. 6.5.- Resultados obtenidos para el ritmo cardiaco con el dispositivo Polar H10 y el prototipo TS "*breadboard*". Datos obtenidos durante distintos trayectos del viaje de ida y vuelta entre Asturias y Almería.

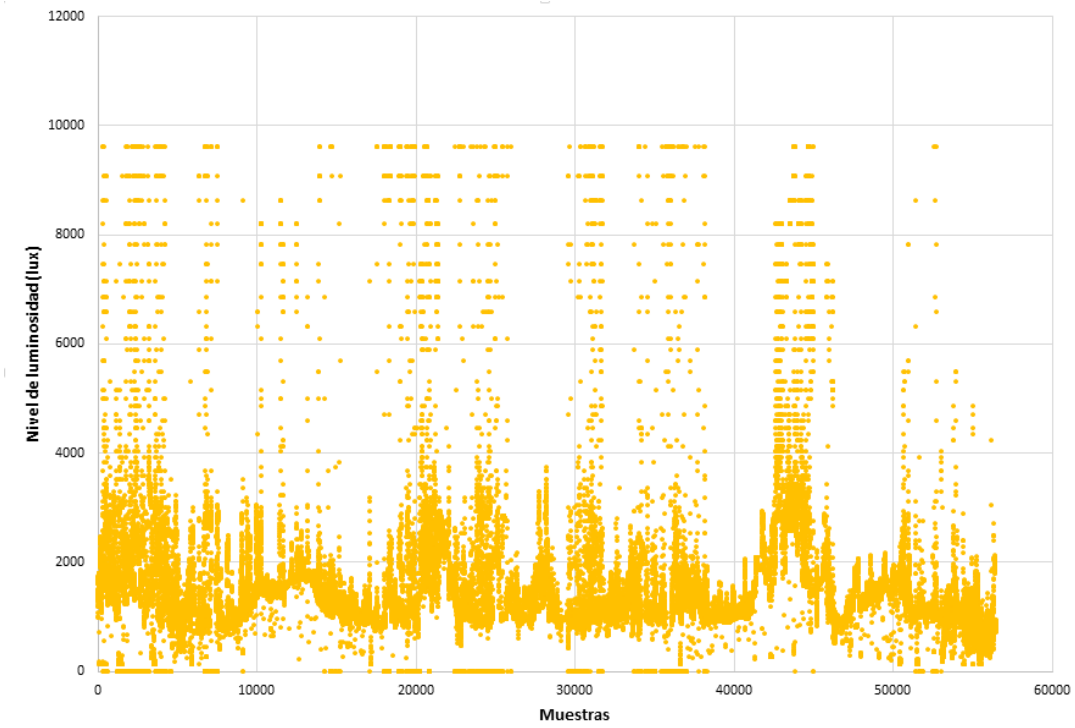


Fig. 6.6.- Resultados obtenidos para el nivel de luminosidad y el prototipo TS "*breadboard*". Datos obtenidos durante distintos trayectos del viaje de ida y vuelta entre Asturias y Almería.

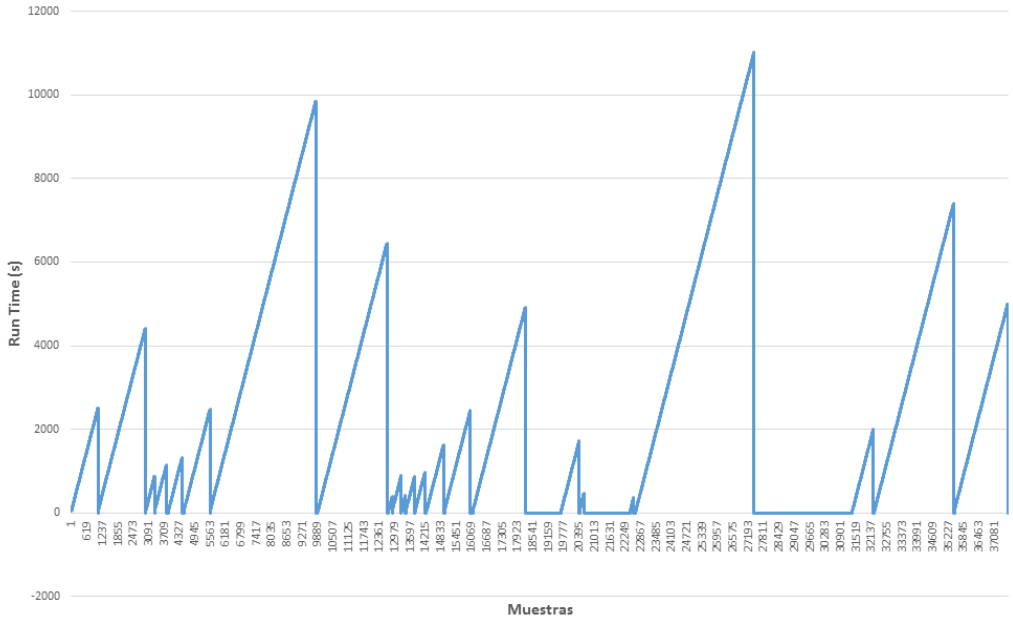


Fig. 6.7.- Resultados obtenidos para el tiempo de trayecto obtenido mediante el adaptador OBD-II conectado al puerto y el prototipo TS "*breadboard*". Datos obtenidos durante distintos trayectos del viaje de ida y vuelta entre Asturias y Almería.

En las figuras 6.5, 6.6 y 6.7 se ha representado el eje x como muestras para facilitar la representación pero para cada una de las muestras como es obvio se tiene registrada en la base de datos la marca de tiempo correspondiente para poder relacionar los datos provenientes de distintas fuentes y hacer en el futuro estudio compuestos.

6.1.2. Evaluación experimental de uso de pulsera *Empatica* E4

Uno de los dispositivos disponibles de mayor interés y cuyas pruebas no se pudieron realizar en el viaje de pruebas entre Almería y Asturias es la pulsera *Empatica* E4.

Dado el interés existente en la integración de este dispositivo en el sistema por la calidad de los datos que aporta ha sido necesario probar la pulsera para verificar que se pueden extraer los datos de ellas con mayores o menores dificultades.

Finalmente se pudieron obtener todos los datos que proporciona la pulsera *Empatica* E4 mediante los programas especificados para dicho fin en el anexo G y mediante la arquitectura hardware descrita en el capítulo 5.

Es necesario indicar que en las pruebas quedaron de nuevo patentes algunos de los problemas expuestos en el capítulo 5, sobre todo los problemas 1 y 2 puesto que el 3 presenta una solución ya indicada temporal que hace que pueda obviarse. Estos problemas hacen en la práctica que el usuario tenga que pasarse varios minutos tratando de conectar la pulsera con el sistema antes de iniciar la marcha, pudiendo estos 5 minutos extenderse a más del doble si ha habido algún fallo inesperado debido a problemas como la no conexión de la SBC secundaria al punto de acceso Wi-Fi de la principal. Además el sistema debería estar pensado para que pudiese ser empleado e instalado de manera sencilla por usuarios sin conocimientos de esta temática e incluso por empresas u otras instituciones que dispongan de flotas de vehículos en los cuales no es permisible una espera de 5 minutos cada vez que se quiere iniciar la marcha.

Por todo ello se ha decidido aparcar el eso de este dispositivo que, pese a ser el que aporta mejores datos con mucha diferencia es también el que dificulta en mayor medida el uso del sistema además de tener un coste mucho mayor que el de otros dispositivos *wearables* probados y que tienen mejor rendimiento.

En principio y con el software y hardware de *Empatica* que suministra a día de hoy la compañía parece muy lejano el que esta pulsera pueda llegar a emplearse en este sistema de la manera que se pretende y no se ha conseguido idear una solución buena a los problemas puesto que la parte de conexión automática entre el dispositivo y el sistema no es posible.

6.3. EVALUACIÓN DE PROTOTIPO BASADO EN TS “PCB”

En este apartado no se volverán a mostrar datos aislados obtenidos mediante este prototipo ya que no es el objetivo. En lugar de eso lo que se hará es mostrar las principales ventajas o soluciones que implementa este prototipo con respecto al anterior en lo referente a los resultados que a partir de él es posible obtener.

Dato	Tiempo de muestreo	Tipo	Tiempo muestreo antiguo
Posición D acelerador	1.5-4	OBD-II	2-4
Posición E acelerador	1.5-4	OBD-II	2-4
Acelerómetro (X, Y, Z)	1	Sensores ambientales	0.5-10
Calidad del aire (%)	0.5-10	Sensores ambientales	0.5-10
Heart *rate* Huawei Watch	-	*Smartwatch*	5-8
Humedad (%)	0.5-10	Sensores ambientales	0.5-10
Latitud módulo FONA 808	2-4	Módulo GPS/GSM	2-4
Longitud módulo FONA 808	2-4	Módulo GPS/GSM	2-4
Latitud Huawei Watch	-	*Smartwatch*	5-8
Longitud Huawei Watch	-	*Smartwatch*	5-8
Nivel de luminosidad (lux)	0.5-10	Sensores ambientales	0.5-10
MAF (Sensor de flujo de aire)	1.5-4	OBD-II	2-4
Carga motor	1.5-4	OBD-II	2-4
Presión admisión aire	1.5-4	OBD-II	2-4
Heart rate Polar H10	1	Cinta pectoral Polar	1
RPM	1.5-4	OBD-II	2-4
Nivel de sonido/ruido	0.5-2	Ruido	0.5-2
Tiempo de trayecto (Run time)	1.5-4	OBD-II	2-4
Temperatura	0.5-10	Sensores ambientales	0.5-10
Temperatura del aire de admisión	1.5-4	OBD-II	2-4
Temperatura refrigerante	1.5-4	OBD-II	2-4
Velocidad	1.5-4	OBD-II	2-4
Altitud módulo FONA 808	2-4	Módulo GPS/GSM	2-4
Intervalo RR Polar H10	1	Cinta pectoral Polar	1
Resto de parámetros OBD disponibles en el vehículo en cada momento	1.5-4	OBD-II	-

Tabla. 6.2.- Datos recogidos con prototipo basado en TS “PCB” con tiempos de muestreo asociados a cada una de ellos. Se muestran también los tiempos de muestreo obtenidos con el prototipo anterior, el TS “bredaboard”.

En la tabla 6.2. se pueden observar varias cosas. En primer lugar hay algunos datos que se recogían en el prototipo antiguo y que no se recogen en este nuevo. Estos datos son los extraídos del smartwatch Huawei. La principal razón de descartar estos datos en el nuevo prototipo no es otra que la de facilitar el uso del sistema y por tanto la recogida de datos para el usuario. Con el reloj Huawei era posible obtener el ritmo cardiaco y otros datos como la aceleración y la rotación de la muñeca, sin embargo la batería del dispositivo hacía que el periodo de muestreo de estos datos tuviese que ser alto para que la batería del mismo pudiese soportar trayectos de varias horas sin agotarse. Este alto periodo de muestreo hacía que los datos no fuesen tan valiosos, principalmente los datos de ritmo cardiaco que serían mucho menos útiles que los obtenidos mediante la cinta Polar H10. Sin embargo el uso de un reloj de este estilo con una autonomía de la batería superior que permitiese una tasa de frecuencia de muestreo mayor sería algo a tener en cuenta para incorporar en el sistema final ya que al haberse incluido el inicio automático de la aplicación del reloj, el usuario sólo tiene que encender el reloj al entrar al vehículo y es un proceso que podría llegar a ser asumible, no como en el caso de la pulsera Empatica.

Volviendo a la tabla 6.2 puede verse como se ha reducido hasta 10 veces el periodo de muestreo del acelerómetro lo cual era un gran problema en los resultados del prototipo antiguo puesto que, tener datos de aceleración del vehículo cada 10 segundos es prácticamente como no disponer de datos puesto que estos carecen de validez real. El tomar datos cada segundo se ha conseguido realizando la adquisición de datos del acelerómetro en paralelo y no en serie con los otros sensores ambientales. El periodo de muestreo de 1 segundo podría incluso ser demasiado pero podría llegar a rebajarse más si se quisiera.

El punto de mayor avance en este prototipo en lo referente a la recogida de datos es sin duda la recogida de datos mecánicos del vehículo mediante el puerto OBD-II. Con el nuevo programa de adquisición y almacenamiento de datos del vehículo, el cual se expone en el anexo E.5, se ha pasado a tener la capacidad de recoger todos los parámetros OBD-II disponibles en un vehículo y a la vez mantener o reducir el periodo de muestreo.

Tras varias semanas de pruebas en trayectos cortos con este prototipo se ha verificado su correcto funcionamiento así como un aumento muy sustancioso de la estabilidad física del sistema a la vez que se aumenta la protección de la electrónica gracias al encapsulado diseñado.

Será este prototipo en principio con el que se siga adelante en los siguientes pasos del proyecto en el que se enmarca este TFM.

7. Conclusiones y líneas futuras

La primera conclusión que se puede extraer después de haber estado trabajando durante el último año en este proyecto es sin duda la importancia que ya está empezando a tener y que tendrá Internet de las Cosas en el futuro y en el avance tecnológico de la sociedad en general. Se ha comprobado como esta tecnología abre una abanico enorme de posibles aplicaciones que sin lugar a duda generarán una mejora en la vida de la gente y en la forma en la que esta gente se relaciona con otras personas y con su entorno. Este proyecto aún no llega a influir en el comportamiento de las personas ya que sólo es una parte de un proyecto más amplio pero si que parece claro que la idea y los objetivos que se están comenzando a cumplir indican que las ayudas a la conducción inteligentes cómo las que se proponen con este sistema son una opción muy importante para aumentar la seguridad a bordo de un vehículo así como mejorar el desempeño de las personas al volante.

En cuanto a las conclusiones más concretas de este Trabajo Fin de Máster se podría empezar por volver a recordar los objetivos que se habían planteado en el subapartado segundo del capítulo 2.

En primer lugar se planteaba la necesidad de crear un sistema para la recogida de información a bordo, tanto la parte software como la parte hardware. Se podría decir que el grado de consecución de este objetivo es del 100%. Se ha diseñado e implementado un equipo completo y totalmente funcional capaz de recoger y almacenar en tiempo real multitud de datos de fuentes muy diferentes. Además se han implementado satisfactoriamente las comunicaciones entre los diferentes elementos y dispositivos que forman el sistema.

En segundo lugar se trataba de buscar una compatibilidad futura del sistema con el resto de componentes a incluir y de ideas a desarrollar con el avance del proyecto global. Mucho de este trabajo se realizó ya al seleccionar el hardware que formaría parte de este sistema, principalmente el uso de la Raspberry Pi 3 como SBC facilita mucho el tener todo tipo de opciones posibles para llevar a cabo sin tener problemas de compatibilidades futuras ni nada por el estilo. Además el sistema creado cuenta con detalles que dejan lugar para ideas o desarrollos futuros como por ejemplo la inclusión de varios LEDs adicionales que podrían emplearse para depuración o para otros muchos fines o el uso de un módulo GPS que es además GSM y que puede ser útil en los siguientes pasos del proyecto.

Como conclusiones que se pueden obtener del desempeño del prototipo final creado, podría decirse que el sistema es bastante sencillo y cómodo de usar por un usuario que era en sí otro de los objetivos que se marcaban al inicio. El uso de la cinta pectoral Polar H10 es la clave en la consecución de este objetivo.

El prototipo final diseñado presenta sin embargo algunas deficiencias que llevan directamente a comenzar a hablar de las posibles mejoras futuras de este sistema. Uno de los puntos menos atractivos del prototipo diseñado es su peso y tamaño, debido en gran parte a su encapsulado y a las conexiones internas necesarias para que funcionen algunos módulos como es el caso del módulo GPS/GSM. Si en algún momento se llegará a diseñar una nueva versión de este prototipo sería conveniente sustituir casi todos los sensores por sus versiones miniaturizadas dado que el espacio que ocupan es una de los factores que hacen aumentar el tamaño del sistema. Por otro lado, sería conveniente que el módulo GPS fuese también de menor tamaño o que el cable USB que necesita para la alimentación pudiese ocupar un menor tamaño o incluso buscar algún módulo que pudiera ser alimentado de una forma más eficiente en tamaño.

Si se continúa hablando del encapsulado, se ha comprobado que la fabricación de este se ha alargado varios meses al no disponer de una impresora 3D propia y además el resultado ha sido una estructura bastante pesada y grande. Si se pudiese imprimir el diseño

en 3D de una manera más eficiente y obtener un resultado menos pesado sería lo ideal. Sin embargo aunque se consiguiera ese encapsulado impreso en 3D más ligero es posible que el precio no disminuyese o incluso se incrementara. Parece un coste muy elevado para la función que realiza esta parte del sistema. De este modo lo mejor sería buscar otra opción para el encapsulado ya que si se pretenden crear varios prototipos el coste y el tiempo de fabricación de un encapsulado como el que tiene el prototipo final creado podría generar muchos problemas.

Con el final de este trabajo se ha dejado implementado un sistema capaz de recoger datos de multitud de fuentes de manera autónoma y por tanto el sistema está listo para que se comiencen a realizar recogidas de datos masivas en local con el objetivo de comenzar a analizar los datos e ir analizando patrones y generando respuestas a esos patrones.

Los siguientes pasos del proyecto que tienen relacionan con el sistema pasan sin duda por dotar al sistema embarcado de conectividad 2G/3G/4G de modo que los datos puedan cargarse en tiempo real en el servidor o nube de inteligencia externo y poder establecer un canal de comunicación que envíe información desde la cabina y reciba órdenes o instrucciones para el conductor o para el propio vehículo.

Otro de los siguientes pasos inmediatos es el de dotar al sistema de una pantalla o de algún sistema de visualización para permitir interacciones visuales con el conductor. En esta pantalla se mostraría las órdenes que llegasen del servidor pero también otros datos que se pudiesen ir analizando en tiempo real desde el propio sistema embarcado que sería otro de los siguientes pasos a realizar.

Del mismo modo que se quiere dotar al sistema de equipos de visualización, es necesario implementar algún sistema de audio e incluso de micrófonos para establecer un canal de audio con el usuario. Este equipo podría ser algún tipo de altavoz o directamente la radio del vehículo si se consiguiese implementar un transmisor FM en la SBC.

Surge también la idea de implementar algún tipo de aviso o de interacción con el conductor diferente como podrían ser vibraciones en el volante o asientos o cosas similares.

Sería por último muy interesante tratar de incorporar al sistema dispositivos como la *Empatica* E4 que aportaran datos biométricos más valiosos que los que se tienen ahora mismo con la Polar H10 pero sin rebajar el nivel de comodidad y facilidad de uso del sistema.

8. Referencias

[1] Stewart A. Birrell and Mark S. Young, "The impact of Smart driving aids on driving performance", "Transportation Research Part F: Traffic Psychology and Behaviour", Volume 14, Issue 6. November 2011. Consultado en agosto de 2017.

[2] Stewart A. Birrell, Mark Fowkes and Paul A. Jennings, "Effect of Using an In-Vehicle Smart Driving Aid on Real-World", IEEE Transactions on Intelligent Transportation Systems, Volume 15, Issue 4. 20 June 2014. Consultado en agosto de 2017.

[3] Shahina Begum, "Intelligent driver monitoring systems base don physiological sensor signals: A review". Publicado en "16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)", 30 January, 2014, The Hague, Netherlands. Consultado en septiembre de 2017.

[4] Mario Muñoz-Organero y Víctor Corcoba Magaña, "Valdating the Impact on Reducing Fuel Consumption by Using an EcoDriving Assistant Based on Traffic Sign Detection and Optimal Deceleration Patterns". IEEE Transactions on Intelligent Transportation Systems, Volume 14, Issue 2. Publicado el 12 de Marzo de 2013. Consultado en septiembre de 2017.

[5] "What Is A Normal Healthy Heart Rate", disponible en http://www.hearthealthonline.org/what-is-a-normal-healthy-heart-rate . Imagen asociada a ritmo cardiaco. Consultado en septiembre de 2017.

[6] Imagen asociada a temperatura disponible en: http://www.transparentpng.com/details/climate-control-home-temperature_2707.html

[7] Imagen asociada a humedad, disponible en: https://www.onlinewebfonts.com/icon/116176

[8] "This tiny projector puts Smartphone apps on your car's windshield", 5 August 2014, disponible en : https://www.theverge.com/2014/8/5/5970705/this-tiny-projector-puts-smartphone-apps-on-your-cars-windshield . Imagen asociada a HUD.

[9] Imagen asociada a velocímetro disponible en : https://www.iconfinder.com/icons/306029/dashboard_gauge_guage_odometer_speed_speedometer_widget_icon#size=256

[10] Imagen asociada a SBC disponible en: https://iconos8.es/icon/54088/frambuesa-pi-zero

[11] Imagen asociada a nube o servidor externo, disponible en: https://icon-icons.com/es/icono/compartida-nube/4316

[12] "Internet de las cosas", artículo de Wikipedia en español. Disponible en https://es.wikipedia.org/wiki/Internet_de_las_cosas . Consultado en septiembre de 2017 y mayo de 2018.

[13] "¿Qué es y cómo funciona el Internet de las Cosas?". Hipertextual, autor: "jjtorres". 20 Octubre 2014. Artículo disponible en:

https://hipertextual.com/archivo/2014/10/internet-cosas/ . Consultado en septiembre de 2017 y mayo de 2018.

[14] Dave Evans, "Internet de las cosas. Cómo la próxima evolución de Internet lo cambia todo". Cisco Internet Business Solutions Group (IBSG). Abril 2011. https://www.cisco.com/c/dam/global/es_mx/solutions/executive/assets/pdf/internet-of-things-iot-ibsg.pdf . Consultado en septiembre de 2017 y mayo de 2018.

[15] Imagen asociada a pines de un puerto OBD-II. Disponible en : https://mechanics.stackexchange.com/questions/23047/obd-ii-power-when-key-not-in-ignition .

[16] Héctor Mañón, "Todo lo que debes saber sobre el puerto OBD-II", 8 Junio 2016. Artículo disponible en: https://noticias.autocosmos.com.mx/2016/06/08/todo-lo-que-debes-saber-sobre-el-puerto-obd-ii . Consultado en septiembre de 2017 y mayo de 2018.

[17] Gonzalo Lara, "CAN bus: la forma de transmitir información en el automóvil", 24 Enero 2013. Artículo disponible en: https://www.motorpasion.com/coches-hibridos-alternativos/can-bus-como-gestionar-toda-la-electronica-del-automovil . Consultado en septiembre de 2017 y mayo de 2018.

[18] Víctor Corcoba Magaña y M. Muñoz-Organero, "Artemisa: An eco-driving assistant for Android Os", publicado en "2011 IEEE International Conference on Consumer Electronics –Berlin (ICCE-Berlin), 6-8 September 2011. Consultado en agosto septiembre de 2017

[19] Unidad de intervención educativa de la Dirección General de Tráfico, "Conducción eficiente". Disponible en: http://www.dgt.es/PEVI/documentos/catalogo_recursos/didacticos/did_adultas/

Conduccion_eficiente.pdf . Consultado en septiembre de 2017 y mayo de 2018.

[20] "Smart Driving Solutions". Artículo del sitio web de GNSD Automotive, disponible en: http://www.gnsd.co.kr/home/html/sub01/sub01_0102.html . Consultado en septiembre de 2017 y mayo de 2018.

[21] JJ. Velasco, "En qué consiste Bluetooth LE?", 10 Diciembre 2013. Artículo disponible en: https://hipertextual.com/2013/12/que-es-bluetooth-le . Consultado en mayo de 2018.

[22] "Variabilidad de la frecuencia cardíaca e intervalos RR", Apta Vital Sport, 2 Mayo 2012. Artículo disponible en: https://aptavs.com/articulos/variabilidad-de-la-frecuencia-cardiaca-e-intervalos-rr junio de 2018.

[23] Imagen asociada a Arduino. Disponible en: https://www.theengineeringprojects.com/product/arduino-uno-r3-pakistan

[24] Imagen asociada a Orange Pi Prime. Disponible en: https://www.redeszone.net/2017/04/12/raspberry-pi-3-vs-orange-pi-prime/

[25] Características de Raspberry Pi 3 modelo B. Imagen asociada a Raspberry Pi 3. Disponible en: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[26] Imagen asociada a sensor de temperatura y humedad DHT-22. Disponible en: https://www.inventelectronics.com/wp-content/uploads/2015/01/dht-22.jpg

[27] Imagen asociada a LDR. Disponible en: https://nomada-e.com/store/consumibles/156-fotoresistencia.html

[28] Imagen asociada a sensor de calidad del aire MQ-135, incluye además de características. Disponible en: http://www.geekbotelectronics.com/producto/mq-135-modulo-sensor-de-calidad-del-aire/

[29] Imagen asociada a sensor de calidad del aire MQ-135. Disponible en: https://glowroad.com/rees52-mq135-mq-135-air-quality-sensor-hazardous-gas-detection-module-for-arduino/az/B018GUYRKO#&gid=1&pid=1

[30] Sparkfun, "Sound Detector Hookup Guide". Disponible en: https://learn.sparkfun.com/tutorials/sound-detector-hookup-guide?_ga=2.195587374.346622441.1530704870-1774217988.1507549321 . Consultado el . Consultado en marzo y abril de 2018.

[31] "Acelerómetro Digital MMA7455, AV Electronics. Artículo disponible en: http://snskart.in/mma7455. Imagen asociada a acelerómetro MMA755.

[32] "Lady Ada", "Adafruit FONA 808 Cellular + GPS Breakout". Disponible en : https://learn.adafruit.com/adafruit-fona-808-cellular-plus-gps-breakout/overview . Consultado en febrero y marzo de 2018.

[33] Características e imagen asociada a antena GPS pasiva con conector uFl. Disponible en: https://www.electronicaembajadores.com/es/Productos/Detalle/AT1A010/antenas/antenas-de-gps/antena-gps-pasiva-ufl-2-dbi . Consultado el

[34] Imagen asociada a breadboard. Disponible en : https://www.wiltronics.com.au/product/10335/830-hole-breadboard-630-2x100/

[35] Imagen asociada a conversor A/D MCP 3008. Imagen disponible en: https://thepihut.com/products/adafruit-mcp3008-8-channel-10-bit-adc-with-spi-interface

[36] Imagen asociada a breakout board. Disponible en: https://www.modmypi.com/raspberry-pi/prototyping-and-breakout-boards/gpio-breakout-boards-1026/rpi-b-plus-gpio-t-cobbler-breakout-board-kit-assembled

[37] Conversor de nivel lógico. Imagen y características disponibles en: https://articulo.mercadolibre.com.ar/MLA-624082347-logic-level-converter-5v-33v-arduino-pic-raspberry-pdiy--_JM

[38] Sitio web oficial de Eurocircuits. Acceso disponible en; https://www.eurocircuits.com

[39] Imagen asociada a smartwatch Huawei Watch 2 Sport. Imagen disponible en : https://www.amazon.com/Huawei-Watch-Sport-Smartwatch-Warranty/dp/B06XDMCH6Z

[40] Imagen asociada a banda pectoral Polar H10. Disponible en: https://support.polar.com/e_manuals/H10_HR_sensor/Polar_H10_user_manual_Espanol/Content/Introduction.htm

[41] Sitio web oficial de Empatica. Empatica E4. Acceso en: https://www.empatica.com/research/e4/

[42] Imagen asociada a adaptador OBD-II Bluetooth ELM 327. Disponible en: https://articulo.mercadolibre.com.ar/MLA-617526548-scanner-bluetooth-obd2-escaner-obdii-elm327-v21-torque-_JM

[43] Imagen asociada a adaptador OBD-II Bluetooth ELM 327 grande. Disponible en: https://www.dpciwholesale.com/obdii-bluetooth-car-diagnostic-code-scanner-2094.html

[44] Lista de parámetros OBD-II. HEM Data Corporation. Lista disponible en: http://hemdata.com/support/list-of-obd-ii-parameters . Consultado en noviembre de 2017 y junio de 2018.

[45] Artículo sobre el uso de módulos FONA y lista de parámetros que devuelve ante solicitudes GPS. Disponible en: https://github.com/initialstate/fona-pi-zero/wiki/Part-4.-GPS-Usage . Consultado en febrero y marzo de 2018.

[46] "Tutorial sensores de gas MQ2, MQ3, MQ7 y MQ135", Naylan Mechatronics. Artículo disponible en: https://naylampmechatronics.com/blog/42_Tutorial-sensores-de-gas-MQ2-MQ3-MQ7-y-MQ13.html . Consultado en octubre y noviembre de 2017.

[47] Documentación para manejo de OBD-II con python. Documentación disponible en: http://python-obd.readthedocs.io/en/latest/ Consultado en noviembre de 2017.

[48] Android Developers, primeros pasos con Android Things y Raspberry Pi. Documentación disponible en: https://developer.android.com/things/hardware/raspberrypi . Consultado en noviembre de 2017 y mayo de 2018.

[49] Tutorial de instalación y explicación de funcionamiento de sensor de calidad del aire MQ-135. Información disponible en: https://tutorials-raspberrypi.com/configure-and-read-out-the-raspberry-pi-gas-sensor-mq-x/ . Consultado en noviembre de 2017 y mayo de 2018.

[50] Tutorial de instalación y uso de sensor de nivel de luminosidad LDR con conversor A/D MCP3008 complementario. Información en: http://www.pibits.net/code/read-ldr-raspberry-pi-using-mcp3008.php . Consultado en septiembre y octubre de 2017.

[51] Video-tutorial de uso e instalación de acelerómetro MMA7455. Vídeo disponible en: https://www.youtube.com/watch?v=OYXXvaypUKQ . Consultado en octubre de 2017.

[52] "Measure Light Intensity using Light Dependent Resistor (LDR)". Artículo disponible en: http://emant.com/316002.page . Consultado en septiembre y octubre de 2017.

[53] "Playing with a MMA7455 Accelerometer", artículo sobre usos de un módulo acelerómetro MMA7455, May 2015. Artículo disponible en: https://linux.activityworkshop.net/raspberry_pi/accelerometer.html . Consultado en octubre de 2017.

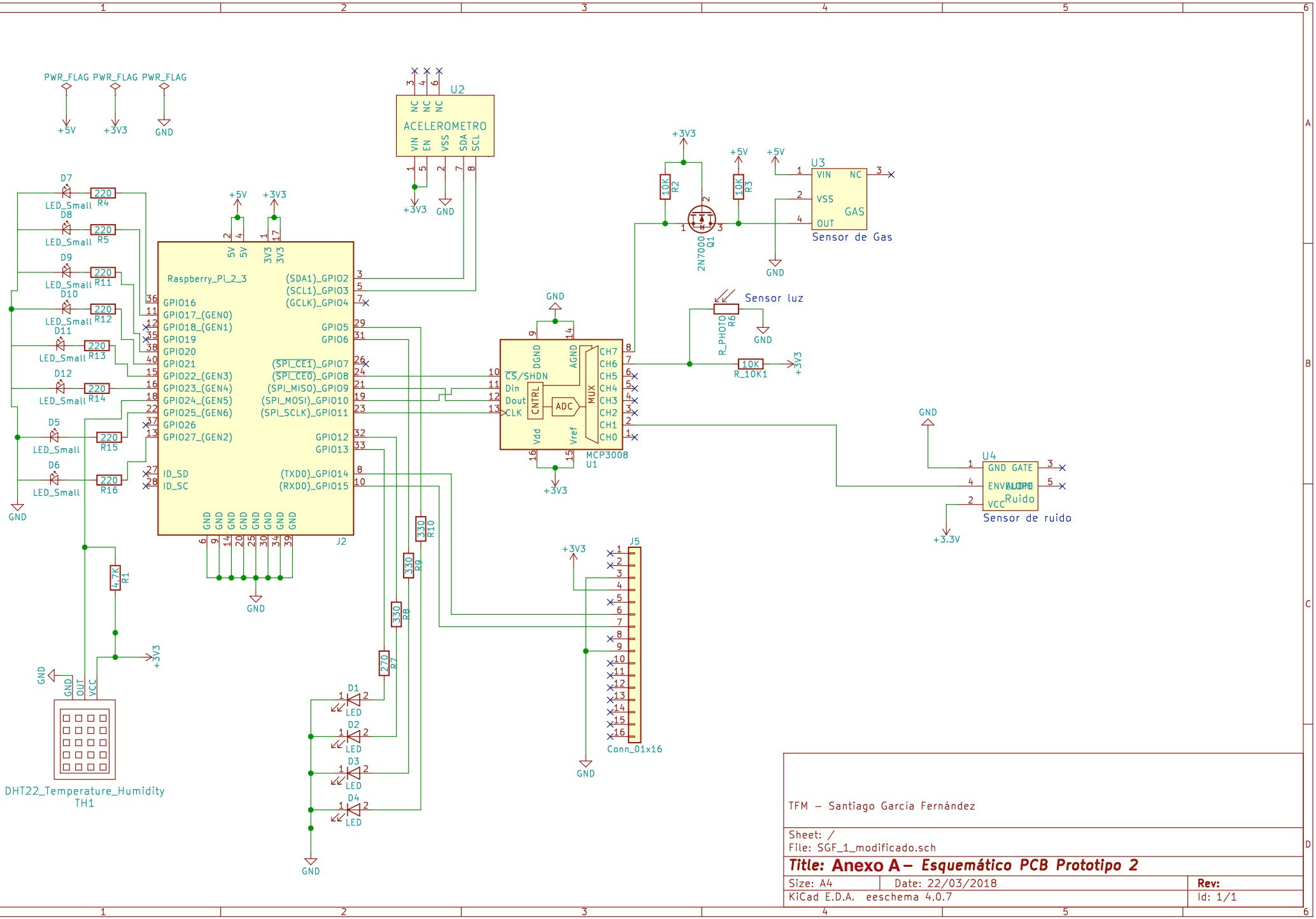
[54] Mustafa Alparslan, "Android Udp Client Example", 9 March 2014. Disponible en: http://msdalp.github.io/2014/03/09/Udp-on-Android/ . Consultado en noviembre de 2017.

[55] Android Developers: Datagram Sockets. Disponible en: https://developer.android.com/reference/java/net/DatagramSocket . Consultado en noviembre de 2017.

[56] Eduardo Ismael García Pérez, "Auto-Ejecución De Una Aplicación Al Inicio De Un Dispositivo Con Android", Códigofacilito. Artículo disponible en: https://codigofacilito.com/articulos/auto-ejecucion-de-una-aplicacion-al-inicio-de-un-dispositivo-con-android . Consultado en noviembre y diciembre de 2017.

[57] "Connect Polar H10 Bluetooth LE heart rate sensor to Ubuntu", Nob, 5 August 2017. Artículo disponible en: https://nob.ro/post/polar_h10_ubuntu/ . Consultado en febrero y marzo de 2018.

[58] "Utilizing the PPG/BVP signal", Empatica Support, 31 March 2016. Artículo disponible en https://support.empatica.com/hc/en-us/articles/204954639-Utilizing-the-PPG-BVP-signal . Consultado en noviembre de 2017 y febrero de 2018.



TFM – Santiago García Fernández

Sheet: /
File: SGF_1_modificado.sch

Title: Anexo A – *Esquemático PCB Prototipo 2*

Size: A4	Date: 22/03/2018	Rev:
KiCad E.D.A. eeschema 4.0.7		Id: 1/1

B. Lista de parámetros OBD-II sobre los que se pueden hacer peticiones mediante los adaptadores universales

Activo Sí/No (VW Beetle 2012)	PID	Name	Description	Response Value	Comentarios
Si	0	PIDS_A	*Supported PIDs [01-20]*	*bitarray*	*PID. Performance Information Data soportados. http://www.totalcardiagnostics.com/support/Knowledgebase/Article/View/43/6/reading-performance-information-data-pid*
Si	1	STATUS	*Status since DTCs cleared*	*special*	*Indicador de mal funcionamiento (luz roja del coche). Debería estar siempre. También debería dar información sobre los sistemas disponibles en cuanto a información.*
No	2	FREEZE_DTC	*DTC that triggered the freeze*	*special*	*DTC. Diagnostic Trouble Codes. Resultado del diagnóstico de los sistemas y explicación de cada código.*
No	3	FUEL_STATUS	*Fuel System Status*	*(string, string)*	*Dos strings (generalmente uno de ellos vacíos) indicando el estado de los dos sistemas de combustibles del vehículo (la mayoría solo tienen uno).*
Si	4	ENGINE_LOAD	*Calculated Engine Load*	*Unit.percent*	*Carga motor sobre el total de su capacidad. Nos indica lo forzado que trabaja el motor*
Si	5	COOLANT_TEMP	*Engine Coolant Temperature*	*Unit.celsius*	*Temperatura del refrigerante? Entiendo que sea del aceite. En principio NO SERÍA NECESARIO*
No	6	SHORT_FUEL_TRIM_1	*Short Term Fuel Trim - Bank*	*Unit.percent*	*Short Term Fuel Trim (STFT): factor de corrección (%) que aplica la PCM en ciclo cerrado para mantener una mezcla de fuel equilibrada. Si el sistema de combustible fuerse de ciclo abierto daría 0%. El valor debería ser ±5. Si fuese mayor de 10, estaríamos ante un problema de la mezcla.*
No	7	LONG_FUEL_TRIM_1	*Long Term Fuel Trim - Bank 1*	*Unit.percent*	
No	8	SHORT_FUEL_TRIM_2	*Short Term Fuel Trim - Bank*	*Unit.percent*	
No	9	LONG_FUEL_TRIM_2	*Long Term Fuel Trim - Bank 2*	*Unit.percent*	*Long Term Fuel Trim (LTFT): factor de corrección (%) que aplica la PCM tanto en ciclo abierto como en ciclo cerrado. El*
No	0A	FUEL_PRESSURE	*Fuel Pressure*	*Unit.kilopascal*	*Mismo caso que el anterior*
Si	0B	INTAKE_PRESSURE	*Intake Manifold Pressure*	*Unit.kilopascal*	*Presión en la admisión de aire*
Si	0C	RPM	*Engine RPM*	*Unit.rpm*	*Régimen motor*
Si	0D	SPEED	*Vehicle Speed*	*Unit.kph*	*Velocidad km/h*
No	0E	TIMING_ADVANCE	*Timing Advance*	*Unit.degree*	*Indica el punto en que se produce la chispa en el cilindro, en grados de giro del cigüeñal, con respecto al punto muerto superior (posición de máxima compresión). Le afectan muchos factores http://e-auto.com.mx/manual_detalle.php?manual_id=247.*
Si	0F	INTAKE_TEMP	*Intake Air Temp*	*Unit.celsius*	*Temperatura del aire de admisión. Interesante para control de consumo. Verificar para futuro)*
Si	10	MAF	*Air Flow Rate (MAF)*	*Unit.grams_per_s*	*Flujo de aire (g/s). Verificar para consumo.*
Si	11	THROTTLE_POS	*Throttle Position*	*Unit.percent*	*Posición del "empuje". Grado de apertura de la válvula de combustible.*
No	12	AIR_STATUS	*Secondary Air Status*	*string*	*Punto de admisión de aire secundario (recirculación de gases)*
Si	13	O2_SENSORS	*O2 Sensors Present*	*special*	*Indica la presencia de los sensores de O2 (2 "bancos" de 4 sensores)*
No	14	O2_B1S1	*O2: Bank 1 - Sensor 1 Voltag*	*Unit.volt*	*Respuesta (en voltios) de los sensores de O2*
No	15	O2_B1S2	*O2: Bank 1 - Sensor 2 Voltag*	*Unit.volt*	
No	16	O2_B1S3	*O2: Bank 1 - Sensor 3 Voltag*	*Unit.volt*	
No	17	O2_B1S4	*O2: Bank 1 - Sensor 4 Voltag*	*Unit.volt*	
No	18	O2_B2S1	*O2: Bank 2 - Sensor 1 Voltag*	*Unit.volt*	
No	19	O2_B2S2	*O2: Bank 2 - Sensor 2 Voltag*	*Unit.volt*	
No	1A	O2_B2S3	*O2: Bank 2 - Sensor 3 Voltag*	*Unit.volt*	
No	1B	O2_B2S4	*O2: Bank 2 - Sensor 4 Voltag*	*Unit.volt*	
Si	1C	OBD_COMPLIANCE	*OBD Standards Compliance*	*string*	
No	1D	O2_SENSORS_ALT	*O2 Sensors Present (alternat*	*special*	*Indica la presencia de los sensores de O2 (4 " bancos" de 2 sensores)*
No	1E	AUX_INPUT_STATUS	*Auxiliary input status (power*	*boolean*	*Debe ser 1 para el correcto funcionamiento de los sistemas.*

Activo Sí/No (VW Beetle 2012)	PID	Name	Description	Response Value	Comentarios
Si	1F	RUN_TIME	*Engine Run Time*	*Unit.second*	*Tiempo desde que se arranca el motor (s)*
Si	20	PIDS_B	*Supported PIDs [21-40]*	*bitarray*	*PID. Performance Information Data soportados.*
Si	21	DISTANCE_W_MIL	*Distance Traveled with MIL or*	*Unit.kilometer*	*Distancia recorrida con la MIL (Malfunction Indicator Lamp) encendida.*
No	22	FUEL_RAIL_PRESSURE_V	*Fuel Rail Pressure (relative to*	*Unit.kilopascal*	*Presión absoluta en el circuito de combustible.*
Si	23	FUEL_RAIL_PRESSURE_D	*Fuel Rail Pressure (direct injec*	*Unit.kilopascal*	*Presión relativa (a la atmósfera) en el circuito de combustible (manométrica)*
Si	24	O2_S1_WR_VOLTAGE	*02 Sensor 1 WR Lambda Volt*	*Unit.volt*	*Respuesta (en voltios) de los diferentes sensores de O2 en la sonda lambda (WR - Wide Ratio).*
No	25	O2_S2_WR_VOLTAGE	*02 Sensor 2 WR Lambda Volt*	*Unit.volt*	
No	26	O2_S3_WR_VOLTAGE	*02 Sensor 3 WR Lambda Volt*	*Unit.volt*	
No	27	O2_S4_WR_VOLTAGE	*02 Sensor 4 WR Lambda Volt*	*Unit.volt*	
No	28	O2_S5_WR_VOLTAGE	*02 Sensor 5 WR Lambda Volt*	*Unit.volt*	
No	29	O2_S6_WR_VOLTAGE	*02 Sensor 6 WR Lambda Volt*	*Unit.volt*	
No	2A	O2_S7_WR_VOLTAGE	*02 Sensor 7 WR Lambda Volt*	*Unit.volt*	
No	2B	O2_S8_WR_VOLTAGE	*02 Sensor 8 WR Lambda Volt*	*Unit.volt*	
Si	2C	COMMANDED_EGR	*Commanded EGR*	*Unit.percent*	*Dice lo que el PCM (Powertrain Control Module) le manda hacer a la EGR (Exhaust Gas Recirculation).* *Debería indicar 0% cuando no está comandada (OFF, al ralentí), 100% cuando lo está (ON, a baja carga, por ejemplo en inercia), y entre 0 y 100% cuando la solenoide del EGR entra en un ciclo de ON y OFF en el PCM dependiendo de la velocidad, carga motor y temperatura).*
Si	2D	EGR_ERROR	*EGR Error*	*Unit.percent*	*Error calculado en % respecto al actual estado del EGR. Si % < 0, estaría dando menos de lo demandado y si es % > 0 más. Cuanto mayor el valor, más probable que la válvula se haya quedado pegada.*
No	2E	EVAPORATIVE_PURGE	*Commanded Evaporative Purg*	*Unit.percent*	*Está normalizado para cada tipo de sistema de control de evaporaciones. 0% indica OFF y 100% ON.Valor importante cuando LTFTy STFT dan valores muy bajos (mezcla demasiado rica). La válvula de la purga podría estar soltando vapor (combustible, al fin y al cabo) en la admisión.*
No	2F	FUEL_LEVEL	*Fuel Level Input*	*Unit.percent*	*Porcentaje de llenado del tamque de combustible.*
Si	30	WARMUPS_SINCE_DTC_	*Number of warm-ups since co*	*Unit.count*	*Número de veces que se calentó el motor desde la última eliminación de DTCs. En cada ciclo la temperatura del refrigerante debe subir al menos 22ºC, y la del motor superar los 70ºC (60ºC en motores diésel).*
Si	31	DISTANCE_SINCE_DTC_	*Distance traveled since codes*	*Unit.kilometer*	*Distancia desde que los DTCs fueron eliminados por última vez.*
No	32	EVAP_VAPOR_PRESSURE	*Evaporative system vapor pre*	*Unit.pascal*	*Presión en el sistema de control de emisiones de vapor (emisiones producidas por la evaporación del combustible en depósito y conductos*
Si	33	BAROMETRIC_PRESSURE	*Barometric Pressure*	*Unit.kilopascal*	*Presión barométrica (atmosférica). Podría no coincidir con la de algunos servicios metereológicos, que la ajustan al nivel del mar (aunque no lo estén)*

Activo Sí/No (VW Beetle 2012)	PID	Name	Description	Response Value	Comentarios
No	34	O2_S1_WR_CURRENT	*02 Sensor 1 WR Lambda Curr*	*Unit.milliampere*	*Respuesta (en mA) de los diferentes sensores de O2 en la sonda Lambda (WR - Wide Ratio).*
No	35	O2_S2_WR_CURRENT	*02 Sensor 2 WR Lambda Curr*	*Unit.milliampere*	
No	36	O2_S3_WR_CURRENT	*02 Sensor 3 WR Lambda Curr*	*Unit.milliampere*	
No	37	O2_S4_WR_CURRENT	*02 Sensor 4 WR Lambda Curr*	*Unit.milliampere*	
No	38	O2_S5_WR_CURRENT	*02 Sensor 5 WR Lambda Curr*	*Unit.milliampere*	
No	39	O2_S6_WR_CURRENT	*02 Sensor 6 WR Lambda Curr*	*Unit.milliampere*	
No	3A	O2_S7_WR_CURRENT	*02 Sensor 7 WR Lambda Curr*	*Unit.milliampere*	
No	3B	O2_S8_WR_CURRENT	*02 Sensor 8 WR Lambda Curr*	*Unit.milliampere*	
No	3C	CATALYST_TEMP_B1S1	*Catalyst Temperature: Bank 1*	*Unit.celsius*	*Temperatura en el catalizador.*
No	3D	CATALYST_TEMP_B2S1	*Catalyst Temperature: Bank 2*	*Unit.celsius*	
No	3E	CATALYST_TEMP_B1S2	*Catalyst Temperature: Bank 1*	*Unit.celsius*	
No	3F	CATALYST_TEMP_B2S2	*Catalyst Temperature: Bank 2*	*Unit.celsius*	
Si	40	PIDS_C	*Supported PIDs [41-60]*	*bitarray*	*PID. Performance Information Data soportados.*
Si	41	STATUS_DRIVE_CYCLE	*Monitor status this drive cycle*	*special*	*Debe estar en 1 para que funcione la monitorización del ciclo de conducción.*
Si	42	CONTROL_MODULE_VOL	*Control module voltage*	*Unit.volt*	*Tensión (en V) en el módulo de control*
No	43	ABSOLUTE_LOAD	*Absolute load value*	*Unit.percent*	*Valor normalizado de masa de aire (en %) en cada ciclo de admisión.*
No	44	COMMANDED_EQUIV_RA	*Commanded equivalence ratio*	*Unit.ratio*	*Los sistemas que usan sensores convencionales de oxigen muestran la relación de equivalencia comandada por la PCM mientras que están ciclo abierto. Los sistemas en ciclo cerrado deberían mostrar el 100%.*
Si	45	RELATIVE_THROTTLE_P	*Relative throttle position*	*Unit.percent*	*Posición relativa (o aprendida) de la admisión.*
Si	46	AMBIANT_AIR_TEMP	*Ambient air temperature*	*Unit.celsius*	*Temperatura ambiente.*
No	47	THROTTLE_POS_B	*Absolute throttle position B*	*Unit.percent*	
No	48	THROTTLE_POS_C	*Absolute throttle position C*	*Unit.percent*	
Si	49	ACCELERATOR_POS_D	*Accelerator pedal position D*	*Unit.percent*	*Posición absoluta del acelerador.*
Si	4A	ACCELERATOR_POS_E	*Accelerator pedal position E*	*Unit.percent*	
No	4B	ACCELERATOR_POS_F	*Accelerator pedal position F*	*Unit.percent*	
Si	4C	THROTTLE_ACTUATOR	*Commanded throttle actuator*	*Unit.percent*	*This shohuld be 0% when commanded CLOSED (inercia) and 100% when commanded OPEN (full throttle).*
No	4D	RUN_TIME_MIL	*Time run with MIL on*	*Unit.minute*	*Minutos acumulados (totales, desde que existe el coche) de funcionamiento del motor con el MIL encendido.*
No	4E	TIME_SINCE_DTC_CLEA	*Time since trouble codes clea*	*Unit.minute*	*Tiempo acumulado desde la última limpieza de DTCs*

Activo Sí/No (VW Beetle 2012)	PID	Name	Description	Response Value	Comentarios
No	4F	*unsupported*	*unsupported*		
No	50	MAX_MAF	*Maximum value for mass air fl*	*Unit.grams_per_s*	*Valor máximo del sensor de flujo de aire (g/s)*
No	51	FUEL_TYPE	*Fuel Type*	*string*	*Tipo de combustible*
No	52	ETHANOL_PERCENT	*Ethanol Fuel Percent*	*Unit.percent*	*Porcentaje de etanol (para vehículos de combustible etanol)*
No	53	EVAP_VAPOR_PRESSURE	*Absolute Evap system Vapor*	*Unit.kilopascal*	*Presión de vapor absoluta en el sistema de evaporación (normalmente en el depósito)(kPa).*
No	54	EVAP_VAPOR_PRESSURE	*Evap system vapor pressure*	*Unit.pascal*	*Medida alternativa de la presión de vapor en el sistema de evaporación (Pa).*
No	55	SHORT_O2_TRIM_B1	*Short term secondary O2 trim*	*Unit.percent*	
No	56	LONG_O2_TRIM_B1	*Long term secondary O2 trim*	*Unit.percent*	
No	57	SHORT_O2_TRIM_B2	*Short term secondary O2 trim*	*Unit.percent*	
No	58	LONG_O2_TRIM_B2	*Long term secondary O2 trim*	*Unit.percent*	
No	59	FUEL_RAIL_PRESSURE_A	*Fuel rail pressure (absolute)*	*Unit.kilopascal*	*Presión absoluta en el rail de inyección de combustible (kPa).*
No	5A	RELATIVE_ACCEL_POS	*Relative accelerator pedal positi*	*Unit.percent*	*Posición relativa del pedal de aceleración.*
No	5B	HYBRID_BATTERY_REMA	*Hybrid battery pack remaining*	*Unit.percent*	
No	5C	OIL_TEMP	*Engine oil temperature*	*Unit.celsius*	*Temperatura del aceite del motor.*
No	5D	FUEL_INJECT_TIMING	*Fuel injection timing*	*Unit.degree*	*Momento de la inyección de combustible (en grados de giro del cigüeñal).*
No	5E	FUEL_RATE	*Engine fuel rate*	*Unit.liters_per_hou*	*Consumo instantáneo (l/h).*
No	5F	*unsupported*	*unsupported*		

C. Presupuesto y diagrama de Gantt.

La elaboración del presupuesto que a continuación se detalla consta de elementos de los cuales se tiene su coste exacto como son los equipos y el material adquiridos específicamente para el desarrollo de este trabajo y también de elementos materiales o servicios de los cuales se ha intentado dar un coste aproximado. Se ha considerado para los materiales principales empleados un periodo de amortización de 10 años y se ha establecido el coste en base a la fracción de ese tiempo de amortización en el cual su uso ha sido destinado a la realización de este Trabajo Fin de Master. Además se ha tratado de estimar el precio de la mano de obra que correspondería a las distintas fases en las que se ha dividido el trabajo teniendo en cuenta el diagrama de Gantt que se detalla en este mismo anexo. Los cálculos de este presupuesto se refieren únicamente al trabajo realizado en torno al sistema principal del trabajo y no a las versiones más iniciales del mismo.

	Partida 1.- Materiales necesarios		
Ud.	*Concepto*	*Precio unitario (€)*	*Subtotal (€)*
1	Raspberry Pi 3 model B	29,47	29,47
1	Sensor Temperatura y Humedad DHT-22	3,89	3,89
1	LDR	0,40	0,40
1	Sensor calidad del aire MQ-135	4,72	4,72
1	Sensor de nivel de sonido Sparkfun SEN-14262	9,80	9,80
1	Acelerómetro MMA7455	9,60	9,60

1	Conversor A/D MCP3008	2,16	2,16
1	Módulo GPS/GSM Adafruit FONA 808	40,96	40,96
1	Antena GPS pasiva uFl	4,06	4,06
1	Batería LiPoly 3.7 V / 500 mAh	6,90	6,90
1	Antena GSM	2,42	2,42
1	Cable USB – micro USB - Corto	3,60	3,60
1	Cable USB – micro USB (alimentación) - Largo	4,60	4,60
1	Fabricación PCB	1,30	1,30
1	Componentes electrónicos varios soldados sobre la PCB	16,20	16,20
1	Adaptador OBDII Bluetooth	7,20	7,20
1	Adaptador alimentación coche a USB – 2.4 A	6,40	6,40
1	Cinta pectoral Polar H10	65,00	65,00
1	Tall header GPIO Raspberry Pi 3	1,60	1,60
1	Encapsulado Raspberry Pi 3	6,30	6,30
1	Encapsulado prototipo en impresión 3D	82,00	82,00
1	Tarjeta SD 32 GB – Clase 10 A1	11,50	11,50
0,10	Equipos informáticos	3 400,00	340,00
		Total 1:	**660,08**

Tabla C.1. Partida 1 del presupuesto del sistema principal del TFM.

	Partida 2.- Horas de trabajo		
Ud.	*Concepto*	*Precio unitario (€)*	*Subtotal (€)*
450	Implementación del prototipo	50,00	22.500
200	Pruebas experimentales	40,00	8.000
200	Documentación del proyecto	30,00	6.000
40	Dirección de proyecto	75,00	3.000
		Total 2:	39.500
	TOTAL EJECUCIÓN MATERIAL		**40.160,08**
	IVA (21%)		8.433,62
	BENEFICIO INDUSTRIAL (6%)		2.409,60
	TOTAL EJECUCIÓN POR CONTRATA		**51.003,30**

Tabla C.2. Partida 2 del presupuesto del sistema principal del TFM.

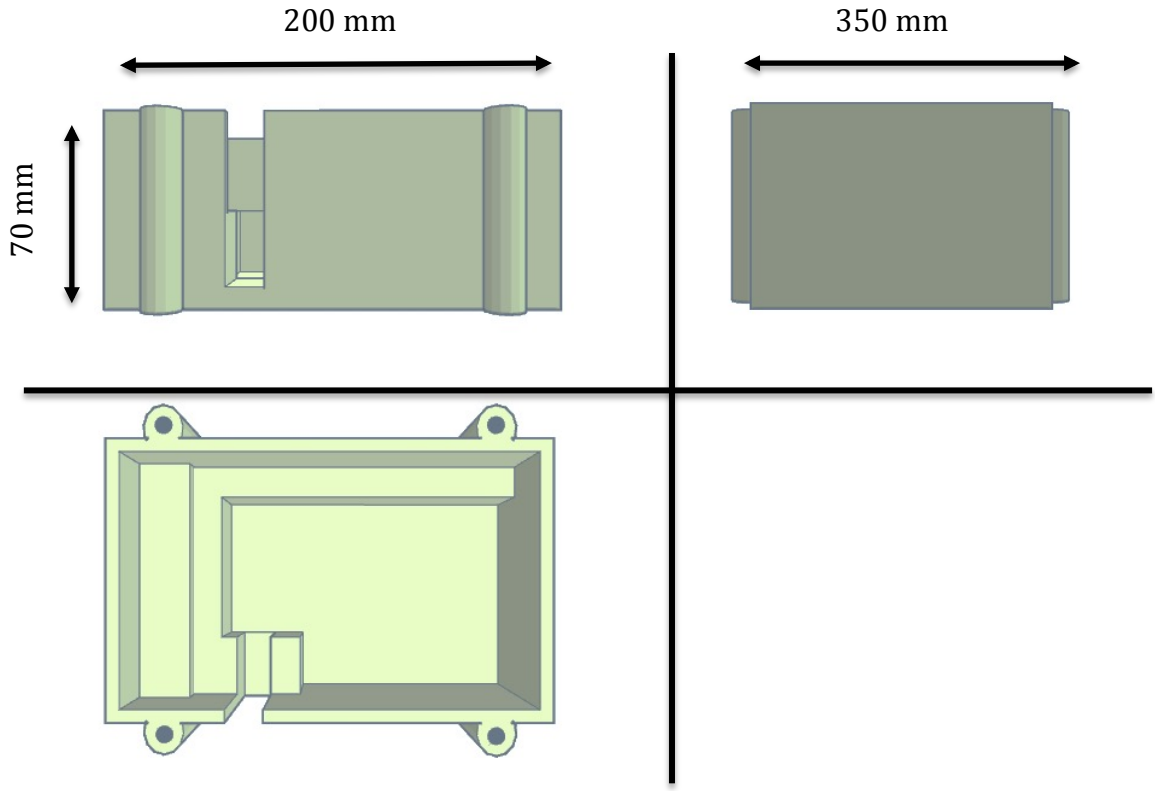
	2017					2018						
	Ago	Sep	Oct	Nov	Dic	Ene	Feb	Mar	Abr	May	Jun	Jul
1. Análisis del problema												
1.1. Estudio del problema, búsqueda y selección de posibles soluciones.	30 h											
1.2. Fundamentos teóricos	20 h											
2. Diseño												
2.1. Diseño inicial del sistema propuesto		30 h										
3. Implementación hardware												
3.1. Selección de SBC del sistema		12 h										
3.2. Implementación tarjeta de sensores de cabina												
3.2.1. Implementación prototipo TS "breadboard" + software		82 h					30 h					
3.2.2. Implementación prototipo TS "PCB"												
3.2.2.1. Diseño e implementación de PCB							70 h					
3.2.2.2. Montaje prototipo + software										20 h		
3.2.2.3. Diseño y fabricación encapsulado										70 h		
3.3. Implementación de recogida de datos biométricos +software				175 h								10 h
3.4. Implementación de recogida de datos del vehículo + software			80 h						50 h			
4. Validación experimental												
4.1. Pruebas en laboratorio dentro del desarrollo		56 h					125 h					
4.2. Pruebas en entornos reales con prototipo TS "breadboard"							30 h					
4.3. Pruebas en entornos reales con prototipo TS "PCB"										55 h		
5. Análisis de resultados												
5.1. Extracción de conclusiones y análisis de validez del método.											20 h	
6. Otros												
6.1. Redacción y preparación TFG										200 h		

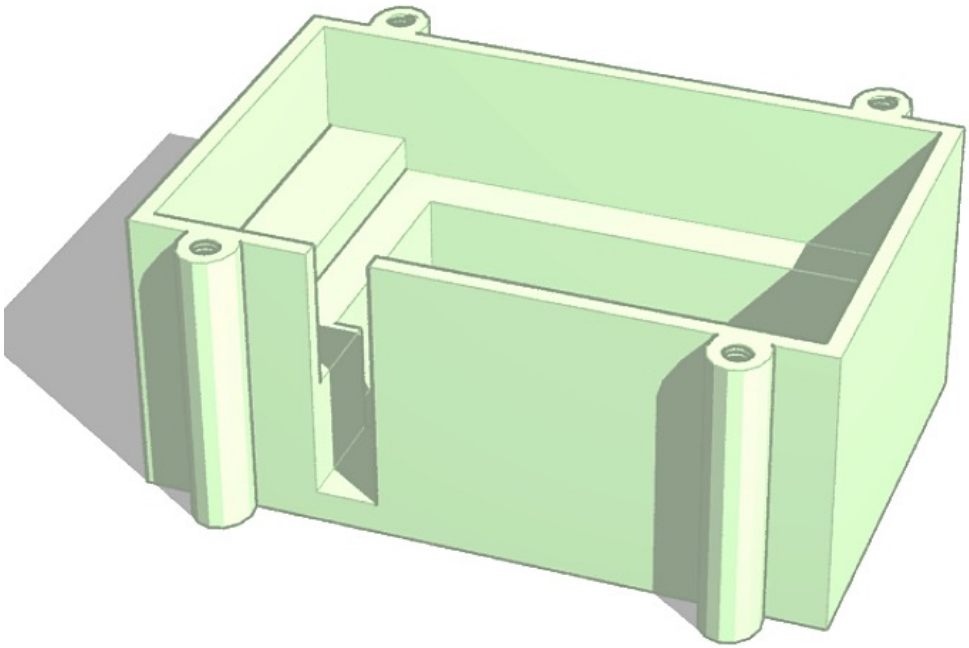
Tabla C.3. Diagrama de Gantt.

D.- DIMENSIONES BÁSICAS DE PIEZAS DISEÑADAS PARA IMPRESIÓN 3D

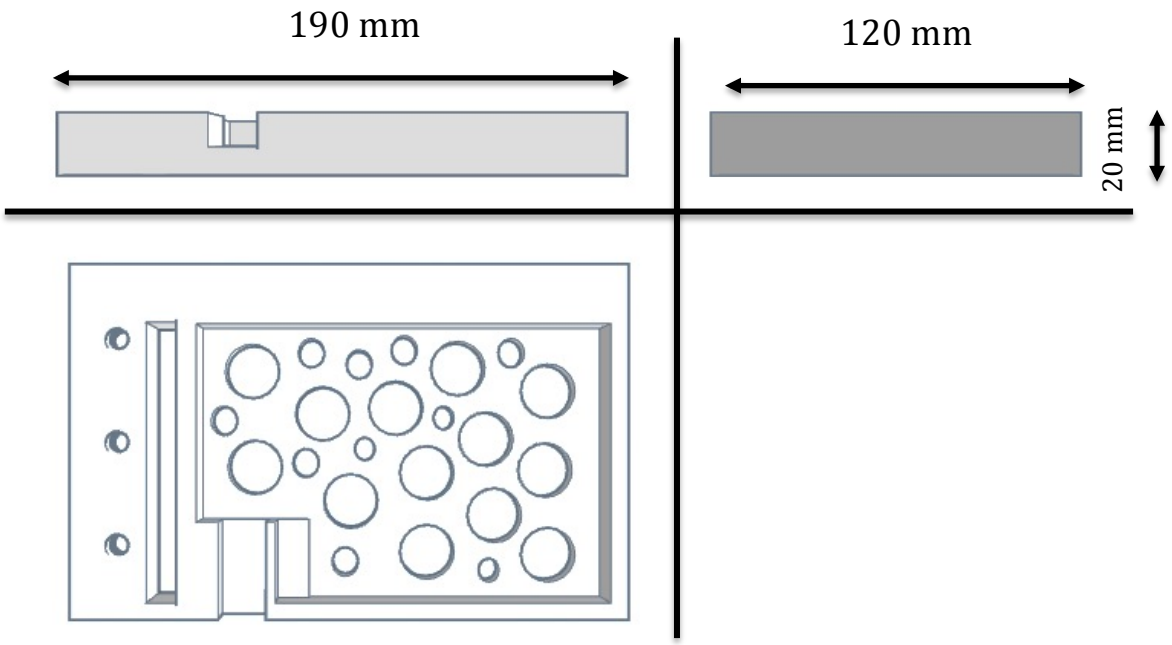
Las vistas aquí mostradas no se corresponden con vistas de alzado, perfil y planta sino que tienen perspectiva. La exposición de estas imágenes es informativa y no pretende en ningún caso dar medidas detalladas de la totalidad de las dimensiones de las piezas diseñadas.

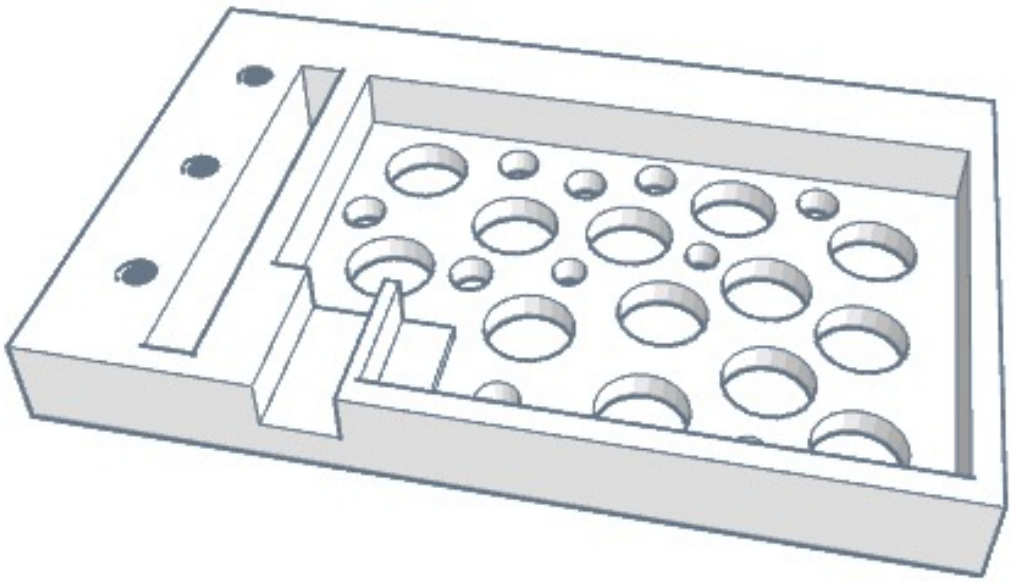
PIEZA Nº 1: Encapsulado sin tapa diseñado en una sola pieza (no impreso).



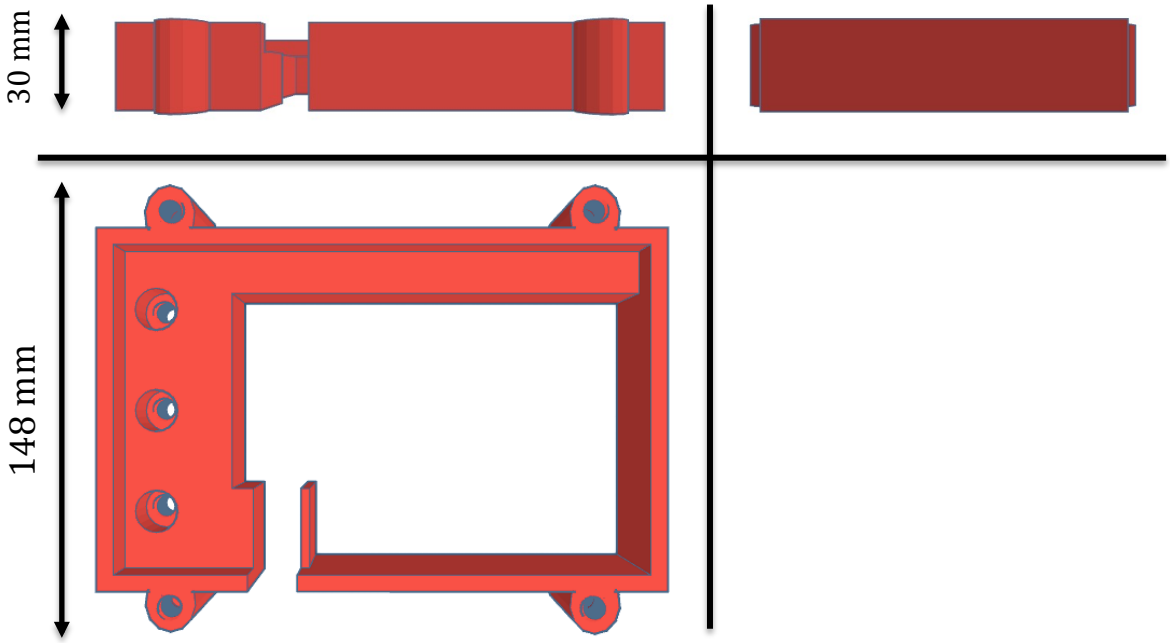


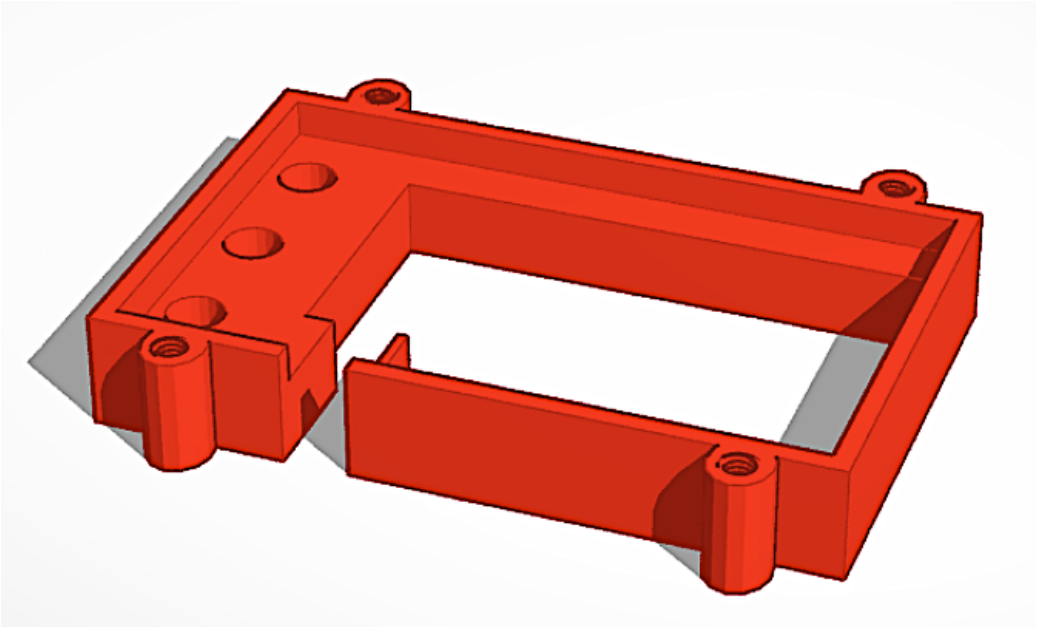
PIEZA Nº 2: Encapsulado con tapa diseñado en 3 partes (impreso), parte inferior.



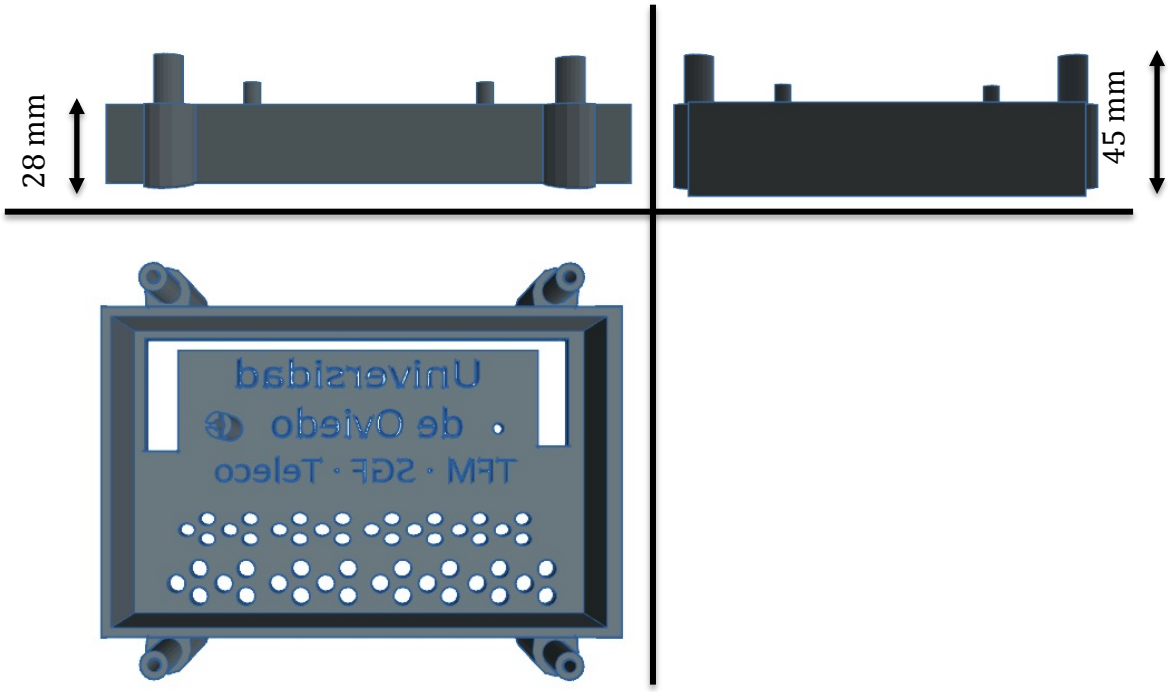


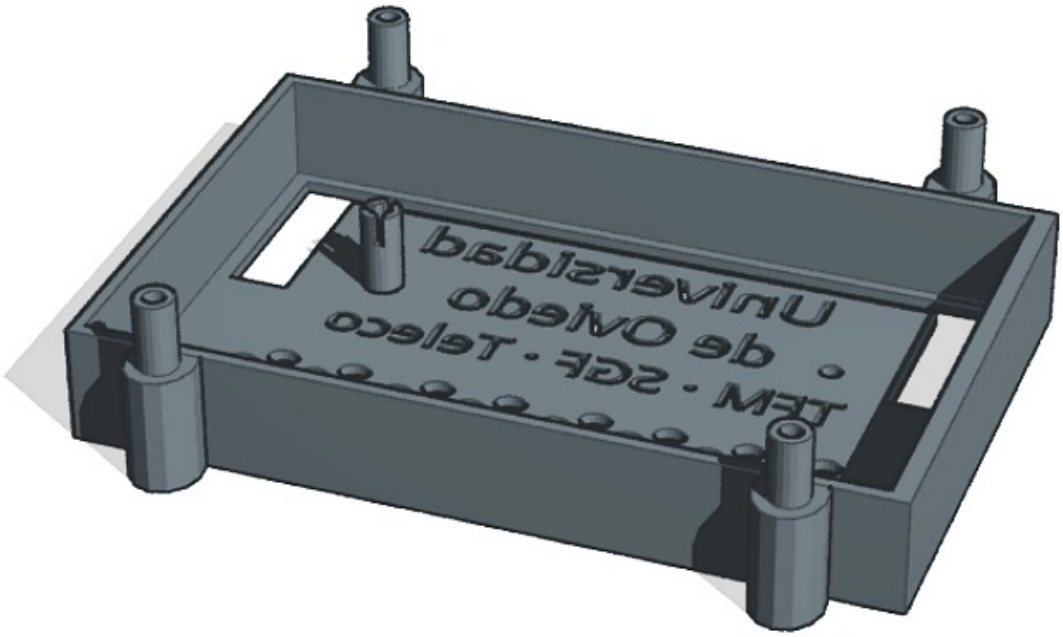
PIEZA Nº 3: Encapsulado con tapa diseñado en 3 partes (impreso), parte intermedia.





PIEZA Nº 4: Encapsulado sin tapa diseñado en 3 partes (impreso), parte superior (tapa).





E.- SCRIPTS PYTHON EMPLEADOS EN EL PROTOTIPO FINAL

E.1.- SCRIPT Nº 1- *"accel.py"* : Programa para la adquisición y almacenamiento de datos con el módulo acelerómetro.

```
# -*- coding:utf-8 -*-
#!/usr/bin/python

#Importación de librerías necesarias
import smbus
import time
import os
import RPi.GPIO as GPIO
import sys
import sqlite3
import datetime


#Establecimiento de conexión con base de datos SQLite
conn=sqlite3.connect('DATOS_P1.db')
curs=conn.cursor()


#Definición de funciones para crear tabla y para añadir dato a tabla en SQLite
def create_table():
	curs.execute('CREATE TABLE IF NOT EXISTS Acelerometro(keyword TEXT, unix REAL,
datestamp TEXT, value REAL)')


def dynamic_data_entry(sensor,valor,tabla,unix,date):
	keyword=sensor
	value=valor
	curs.execute("INSERT INTO "+ tabla +" (keyword, unix, datestamp, value) VALUES(?, ?, ?,
?)",(keyword, unix, date, value))
	conn.commit()


#Defincón de pines del GPIO a utilizar
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(16, GPIO.OUT)
pinparp=16
```

```
#Contador para parpadeo de LED en funcionamiento
contador=0

#Creación de tabla/s
try:
        create_table()
except:

        create_table()


# Define a class called Accel
class Accel():
   myBus=""
   if GPIO.RPI_INFO['P1_REVISION'] == 1:
      myBus=0
   else:
      myBus=1
   b = smbus.SMBus(myBus)
   def setUp(self):
      self.b.write_byte_data(0x1D,0x16,0x55) # Setup the Mode
      self.b.write_byte_data(0x1D,0x10,0) # Calibrate
      self.b.write_byte_data(0x1D,0x11,0) # Calibrate
      self.b.write_byte_data(0x1D,0x12,0) # Calibrate
      self.b.write_byte_data(0x1D,0x13,0) # Calibrate
      self.b.write_byte_data(0x1D,0x14,0) # Calibrate
      self.b.write_byte_data(0x1D,0x15,0) # Calibrate
   def getValueX(self):
      return self.b.read_byte_data(0x1D,0x06)
   def getValueY(self):
      return self.b.read_byte_data(0x1D,0x07)
   def getValueZ(self):
      return self.b.read_byte_data(0x1D,0x08)


#Bucle principal
while True:
   try:
                #Registro de marca de tiempo
         unix=time.time()
         date=str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d %H:%M:%S'))

                #Llamo a funciones de iniciación del módulo acelerómetro
      MMA7455 = Accel()
      MMA7455.setUp()

      #Adquisición de datos
      x = MMA7455.getValueX()
      y = MMA7455.getValueY()
      z = MMA7455.getValueZ()
                #Tiempo de espera porque no necesitamos el rendimiento al máximo
      time.sleep(0.1)
      os.system("clear")
      #Almacenamiento de datos en base de datos SQLite
                dynamic_data_entry('X', x,"Acelerometro",unix,date)
                dynamic_data_entry('Y', y,"Acelerometro",unix,date)
                dynamic_data_entry('Z', z,"Acelerometro",unix,date)
```

```
    #PARPADEO de LED para avisar de funcionamiento del script
    if (contador==20):
            GPIO.output(pinparp, GPIO.HIGH)
            contador=0
    elif (contador==10):
            GPIO.output(pinparp, GPIO.LOW)
            contador=contador+1
    else:
            contador=contador+1


except:
      #Ante un error volvemos al bucle principal
    pass
```

E.2.- SCRIPT Nº 2- *"COPY_P1.py"* : Programa para la actualización de las bases de datos SQLite en el arranque para evitar posibles errores en el anterior cierre de la base de datos.

```
# -*- coding:utf-8 -*-
#!/usr/bin/python

#Importación de librerías necesarias
import time
import sys
import subprocess
import commands


try:
                #Creo una copia de la base de datos "DATOS_P1.db"
    try:
        subprocess.call(["sudo cp /home/pi/DATOS_P1.db /home/pi/DATOS_P33.db"],shell=True)
    except:
        pass

    #Borro base de datos "DATOS_P1.db"
    try:
        subprocess.call(["sudo rm /home/pi/DATOS_P1.db"],shell=True)
    except:
        pass

    #Borro el archivo temporal que indica que la base de datos está abierta, si es que lo hubiera
    try:
        subprocess.call(["sudo rm /home/pi/DATOS_P1.db-journal"],shell=True)
    except:
         pass

    #Vuelvo a crear el archivo con el nombre original a partir de la copia que se había creado
    try:
        subprocess.call(["sudo cp /home/pi/DATOS_P33.db /home/pi/DATOS_P1.db"],shell=True)
    except:
        pass

except:
    time.sleep(1)
```

El código para la copia de la base de datos "DATOS_P2" es idéntico a este, sustituyendo "DATOS_P1" por "DATOS_P2" y "DATOS_P33" por "DATOS_P44".

E.3.- SCRIPT Nº 3- *"hdiez1.py"* : Programa para la adquisición de datos de ritmo cardiaco e intervalo RR de la cinta pectoral Polar H10

```
# -*- coding:utf-8 -*-
#!/usr/bin/python

#Importación de librerías necesarias
import time
import sqlite3
import sys
import datetime
import subprocess
import commands

#Lista de las MACs correspondientes a los diferentes dispositivos Polar H10 disponibles
lista=["C9:9F:07:2B:37:2C","F1:76:A8:90:68:E6"]

#Bucle principal
while True:
        #Miro qué dispositivo está empleándose
        for mac in lista:
            try:
                #Conexión con el dispositivo y petición para escuchar por las
                #notificaciones del servicio de ritmo cardiaco
                texto="sudo gatttool -t random -b "+mac+" --char-write-req --handle=0x0011 --
value=0100 --listen | perl -ne 'if(/.*value: (\w+) (\w+) (\w+) (\w+)/){ ($x,$y,$z,$a) = ($1,$2,$3,
$4);$rr=hex("+""+"$a$z"+""+"); open (FILE, "+""+">hdiez33.txt"+""+") or die "+""+"Cannot open
file"+""+"; printf FILE ("+""+"%d, %d\n"+""+", hex($y), $rr); close (FILE);}'"
                subprocess.call([texto],shell=True)
            except:
                time.sleep(0.01)
```

E.4.- SCRIPT Nº 4- *"hdiez2.py"* : Programa para el almacenamiento de datos de ritmo cardiaco e intervalo RR de la cinta pectoral Polar H10

```
# -*- coding:utf-8 -*-
#!/usr/bin/python

#Importación de librerías necesarias
import time
from gpiozero import MCP3008
import RPi.GPIO as GPIO
import sys
import sqlite3
import datetime
import os
import subprocess
import commands

time.sleep(2)

#Establecimiento de conexión con base de datos SQLite
conn=sqlite3.connect('DATOS_P1.db')
curs=conn.cursor()

#Definición de funciones para crear tabla y para añadir dato a tabla en SQLite
def create_table():
	curs.execute('CREATE TABLE IF NOT EXISTS Pulso_H10(keyword TEXT, unix REAL,
datestamp TEXT, value REAL)')
	curs.execute('CREATE TABLE IF NOT EXISTS RR_H10(keyword TEXT, unix REAL, datestamp
TEXT, value REAL)')

def dynamic_data_entry(sensor,valor,tabla,unix,date):
	keyword=sensor
	value=valor
	curs.execute("INSERT INTO "+ tabla +" (keyword, unix, datestamp, value) VALUES(?, ?, ?,
?)",(keyword, unix, date, value))
	conn.commit()

#Defincón de pines del GPIO a utilizar
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(23, GPIO.OUT) #
pinparp=23

#Contador para parpadeo de LED en funcionamiento
contador=0

#Creación de tabla/s
create_table()


#Definición de variables
cont=0
pulso2="NO"
#Bucle principal
while True:
```

```
    try:
      #Registro de marca de tiempo
                    unix=time.time()
                    date=str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d
%H:%M:%S'))
                    #Abrimos archivo de texto y leemos su contenido
                    f=open("hdiez33.txt","r")
                    linea=f.readline()

                    time.sleep(0.8)
                    #Analizamos contenido del texto y vamos extrayendo los datos de interés
                    tope=100
                    cont3=0
                    rr=" "
                    pulso=" "
                    for u in linea:

                                  if u==",":
                                                tope=cont3
                                                cont3=cont3+1
                                  elif cont3>tope:
                                                rr=rr+u
                                                cont3=cont3+1
                                  else:
                                          pulso=pulso+u
                                          cont3=cont3+1

                    #Compruebo si el pulso lleva rato sin modificarse lo que podría indicar desconexión
                    if pulso==pulso2:
                            cont=cont+1
                    else:
                            cont=0

                    #Almaceno datos en base de datos
                    if cont>10:
                                  dynamic_data_entry('Ritmo cardiaco
(bpm)',"Null","Pulso_H10",unix,date)
                                  print("Pulso: Null")
                    else:
                                  dynamic_data_entry('Ritmo cardiaco
(bpm)',pulso,"Pulso_H10",unix,date)
                                  dynamic_data_entry('Intervalo RR',rr,"RR_H10",unix,date)
                            print("Pulso: "+pulso)
                            print("RR: "+rr)
                    pulso2=pulso


                    #PARPADEO de LED para avisar de funcionamiento de parte sensores
                    if (contador==8):
                                  GPIO.output(pinparp, GPIO.HIGH)
                                  contador=0
                    elif (contador==4):
                                  GPIO.output(pinparp, GPIO.LOW)
                                  contador=contador+1
                    else:
                                  contador=contador+1

      except:
            #Almaceno "Null" en caso de fallo
        dynamic_data_entry('Ritmo cardiaco (bpm)',"Null","Pulso_H10",unix,date)
```

E.5.- SCRIPT Nº 5- *"obd_mejorado.py"* : Programa para la adquisición y almacenamiento de datos OBD-II

```
# -*- coding:utf-8 -*-
#!/usr/bin/python

#Importación de librerías necesarias
import obd
import time
import sqlite3
import sys
import datetime
import subprocess
import RPi.GPIO as GPIO
import os

#Establecimiento el puerto de conexión y la mac del dispositivo a buscar
host="00:1D:A5:68:98:8D"
port="1"
subprocess.call(["sudo rfcomm bind rfcomm0 00:1D:A5:68:98:8D"],shell=True)

#Establecimiento de conexión con base de datos SQLite
conn=sqlite3.connect('DATOS_P2.db')
curs=conn.cursor()

#Definición de pines del GPIO a utilizar
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
pinparpa=17
GPIO.setup(pinparpa, GPIO.OUT) #OBD

#Contador para parpadeo de LED en funcionamiento
parpa=0

#Definición de funciones para crear tabla y para añadir dato a tabla en SQLite
def create_table():
    curs.execute('CREATE TABLE IF NOT EXISTS STATUS(keyword TEXT, unix REAL, datestamp TEXT,
value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS FREEZE_DTC(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS FUEL_STATUS(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS ENGINE_LOAD(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS COOLANT_TEMP(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS SHORT_FUEL_TRIM_1(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS LONG_FUEL_TRIM_1(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS SHORT_FUEL_TRIM_2(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS LONG_FUEL_TRIM_2(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS FUEL_PRESSURE(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS INTAKE_PRESSURE(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS RPM(keyword TEXT, unix REAL, datestamp TEXT, value
TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS SPEED(keyword TEXT, unix REAL, datestamp TEXT, value
TEXT)')
```

```
curs.execute('CREATE TABLE IF NOT EXISTS TIMING_ADVANCE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS INTAKE_TEMP(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS MAF(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS THROTTLE_POS(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS AIR_STATUS(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_SENSORS(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_B1S1(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_B1S2(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_B1S3(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_B1S4(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_B2S1(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_B2S2(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_B2S3(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_B2S4(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_SENSORS_ALT(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS AUX_INPUT_STATUS(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS RUN_TIME(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS DISTANCE_W_MIL(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS FUEL_RAIL_PRESSURE_VAC(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS FUEL_RAIL_PRESSURE_DIRECT(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_S1_WR_VOLTAGE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_S2_WR_VOLTAGE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_S3_WR_VOLTAGE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_S4_WR_VOLTAGE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_S5_WR_VOLTAGE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_S6_WR_VOLTAGE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_S7_WR_VOLTAGE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS O2_S8_WR_VOLTAGE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS COMMANDED_EGR(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS EGR_ERROR(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS EVAPORATIVE_PURGE(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS FUEL_LEVEL(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS WARMUPS_SINCE_DTC_CLEAR(keyword TEXT, unix REAL, datestamp TEXT, value TEXT)')
curs.execute('CREATE TABLE IF NOT EXISTS DISTANCE_SINCE_DTC_CLEAR(keyword TEXT, unix
```

```
REAL, datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS EVAP_VAPOR_PRESSURE(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS BAROMETRIC_PRESSURE(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS O2_S1_WR_CURRENT(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS O2_S2_WR_CURRENT(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS O2_S3_WR_CURRENT(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS O2_S4_WR_CURRENT(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS O2_S5_WR_CURRENT(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS O2_S6_WR_CURRENT(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS O2_S7_WR_CURRENT(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS O2_S8_WR_CURRENT(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS CATALYST_TEMP_B1S1(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS CATALYST_TEMP_B2S1(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS CATALYST_TEMP_B1S2(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS CATALYST_TEMP_B2S2(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS STATUS_DRIVE_CYCLE(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS CONTROL_MODULE_VOLTAGE(keyword TEXT, unix
REAL, datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS ABSOLUTE_LOAD(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS COMMANDED_EQUIV_RATIO(keyword TEXT, unix
REAL, datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS RELATIVE_THROTTLE_POS(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS AMBIANT_AIR_TEMP(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS THROTTLE_POS_B(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS THROTTLE_POS_C(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS ACCELERATOR_POS_D(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
                curs.execute('CREATE TABLE IF NOT EXISTS ACCELERATOR_POS_E(keyword TEXT, unix
REAL, datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS ACCELERATOR_POS_F(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS THROTTLE_ACTUATOR(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS RUN_TIME_MIL(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS TIME_SINCE_DTC_CLEARED(keyword TEXT, unix
REAL, datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS MAX_MAF(keyword TEXT, unix REAL, datestamp TEXT,
value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS FUEL_TYPE(keyword TEXT, unix REAL, datestamp TEXT,
value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS ETHANOL_PERCENT(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS EVAP_VAPOR_PRESSURE_ABS(keyword TEXT, unix
REAL, datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS EVAP_VAPOR_PRESSURE_ALT(keyword TEXT, unix
REAL, datestamp TEXT, value TEXT)')
```

```
        curs.execute('CREATE TABLE IF NOT EXISTS SHORT_O2_TRIM_B1(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS LONG_O2_TRIM_B1(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS SHORT_O2_TRIM_B2(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS LONG_O2_TRIM_B2(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS FUEL_RAIL_PRESSURE_ABS(keyword TEXT, unix
REAL, datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS RELATIVE_ACCEL_POS(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS HYBRID_BATTERY_REMAINING(keyword TEXT, unix
REAL, datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS OIL_TEMP(keyword TEXT, unix REAL, datestamp TEXT,
value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS FUEL_INJECT_TIMING(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS FUEL_RATE(keyword TEXT, unix REAL, datestamp TEXT,
value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS GET_DTC(keyword TEXT, unix REAL, datestamp TEXT,
value TEXT)')
        curs.execute('CREATE TABLE IF NOT EXISTS GET_CURRENT_DTC(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')

def dynamic_data_entry(sensor,valor,tabla,unix,date):
        keyword=sensor
        value=valor
        curs.execute("INSERT INTO "+ tabla +"  (keyword, unix, datestamp, value) VALUES(?, ?, ?, ?)",(keyword,
unix, date, value))

        conn.commit()

#Creación de tabla/s
create_table()

#Establecemos modo de funcionamiento del OBD
obd.logger.setLevel(obd.logging.DEBUG)

#Bucle para esperar a la conexión satisfactoria con el dispositivo
bal=0

while (bal==0):
        try:

            print("bucle")

                ports=obd.scan_serial()

                con=obd.OBD(ports[0])

                commands=con.supported_commands

                bal=1

        except:
                bal=0

                print "Reintentando conexion  en 5 segundos..."

                time.sleep(5)

#Definimos la lista de posibles parámetros e inicializamos la variable de comprobación de
#los mismos como todo "1" inicialmente
lista_buena=['STATUS','FREEZE_DTC','FUEL_STATUS','ENGINE_LOAD','COOLANT_TEMP','SHORT_FUEL_
TRIM_1','LONG_FUEL_TRIM_1','SHORT_FUEL_TRIM_2','LONG_FUEL_TRIM_2','FUEL_PRESSURE','INTA
KE_PRESSURE','RPM','SPEED','TIMING_ADVANCE','INTAKE_TEMP','MAF','THROTTLE_POS','AIR_STATU
```

```
S','O2_SENSORS','O2_B1S1','O2_B1S2','O2_B1S3','O2_B1S4','O2_B2S1','O2_B2S2','O2_B2S3','O2_B2S4','O2_SE
NSORS_ALT','AUX_INPUT_STATUS','RUN_TIME','DISTANCE_W_MIL','FUEL_RAIL_PRESSURE_VAC','FU
EL_RAIL_PRESSURE_DIRECT','O2_S1_WR_VOLTAGE','O2_S2_WR_VOLTAGE','O2_S3_WR_VOLTAGE','O
2_S4_WR_VOLTAGE','O2_S5_WR_VOLTAGE','O2_S6_WR_VOLTAGE','O2_S7_WR_VOLTAGE','O2_S8_WR
_VOLTAGE','COMMANDED_EGR','EGR_ERROR','EVAPORATIVE_PURGE','FUEL_LEVEL','WARMUPS_SI
NCE_DTC_CLEAR','DISTANCE_SINCE_DTC_CLEAR','EVAP_VAPOR_PRESSURE','BAROMETRIC_PRESS
URE','O2_S1_WR_CURRENT','O2_S2_WR_CURRENT','O2_S3_WR_CURRENT','O2_S4_WR_CURRENT','O2_
S5_WR_CURRENT','O2_S6_WR_CURRENT','O2_S7_WR_CURRENT','O2_S8_WR_CURRENT','CATALYST_
TEMP_B1S1','CATALYST_TEMP_B2S1','CATALYST_TEMP_B1S2','CATALYST_TEMP_B2S2','CONTROL_
MODULE_VOLTAGE','ABSOLUTE_LOAD','COMMANDED_EQUIV_RATIO','RELATIVE_THROTTLE_POS','
AMBIANT_AIR_TEMP','THROTTLE_POS_B','THROTTLE_POS_C','ACCELERATOR_POS_D','ACCELERAT
OR_POS_E','ACCELERATOR_POS_F','THROTTLE_ACTUATOR','RUN_TIME_MIL','TIME_SINCE_DTC_CLE
ARED','MAX_MAF','FUEL_TYPE','ETHANOL_PERCENT','EVAP_VAPOR_PRESSURE_ABS','EVAP_VAPOR
_PRESSURE_ALT','SHORT_O2_TRIM_B1','LONG_O2_TRIM_B1','SHORT_O2_TRIM_B2','LONG_O2_TRIM_
B2','FUEL_RAIL_PRESSURE_ABS','RELATIVE_ACCEL_POS','HYBRID_BATTERY_REMAINING','OIL_TE
MP','FUEL_INJECT_TIMING','FUEL_RATE']
res_ful=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]

#Definimos otras variables y constantes
comprobacion=0
limitexp=5

#Bucle principal
while True:
    try:
            #Registro de marca de timepo
        unix=time.time()
        date=str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d %H:%M:%S'))

                        #inicialización variables
        contador=0
        resval=" "
        comprobacion=comprobacion+1

                        #Comprobación de parámetros a solicitar al puerto OBD-II
        if (comprobacion>limitexp):

res_ful=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
            comprobacion=0
            limitexp=limitexp*20

        for par in lista_buena:
            contador=contador+1
            if res_ful[contador]==1:
                res=con.query(obd.commands[par])
                dynamic_data_entry(par,str(res.value),par,unix,date)
                resval=str(res.value)

                if (resval=="None"):
                    res_ful[contador]=0

        #PARPADEO de LED para avisar de funcionamiento de parte sensores
        if (parpa==2):
            GPIO.output(pinparpa, GPIO.HIGH)
            parpa=0
        elif (parpa==1):
            GPIO.output(pinparpa, GPIO.LOW)
            parpa=parpa+1
        else:
            parpa=parpa+1

    except:
        pass

con.close()
```

E.6.- SCRIPT Nº 6- *"ruido.py"* : Programa para el almacenamiento y adquisición de datos del sensor de nivel de sonido/ruido

```
# -*- coding:utf-8 -*-
#!/usr/bin/python

#Importación de librerías necesarias
import time
from gpiozero import MCP3008
import math
import RPi.GPIO as GPIO
import time
import sys
import sqlite3
import datetime
import os

#Establecimiento de conexión con base de datos SQLite
conn=sqlite3.connect('DATOS_P1.db')
curs=conn.cursor()

#Definición de pines del GPIO a utilizar
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(21, GPIO.OUT) #RUIDO LED


#Contador para parpadeo de LED en funcionamiento
contador=0

#Definición de funciones para crear tabla y para añadir dato a tabla en SQLite
def create_table():
	curs.execute('CREATE TABLE IF NOT EXISTS Ruido(keyword TEXT, unix REAL, datestamp
TEXT, value REAL)')

def dynamic_data_entry(sensor,valor,tabla,unix,date):
	keyword=sensor
	value=valor
	curs.execute("INSERT INTO "+ tabla +" (keyword, unix, datestamp, value) VALUES(?, ?, ?,
?)",(keyword, unix, date, value))
	conn.commit()



#Creación de tabla/s
create_table()

#Definición de variables
max=0

#Bucle principal
while True:
	try:
		#Registro de marca de tiempo
		unix=time.time()
		date=str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d %H:%M:%S'))
```

```
                    #Asignación de variable a pin correspondiente del conversor A/D al
                    #que está conectado el sensor de ruido
        divider = MCP3008(1)


                    #Bucle para obtener el máximo de los 20 últimos valores
        max=0
        for i in range(20):
            if (divider.value>max):
                max=divider.value
            time.sleep(0.05)
        #Paso de valor obtenido en voltios a decibelios con respecto a un valor
        #de referencia definido en el laboratorio
        max=20*(math.log10(max/0.024))
        #Almacenamiento de dato en base de datos
        dynamic_data_entry('Nivel de ruido (dB)',max,"Ruido",unix,date)



        #PARPADEO de LED para avisar de funcionamiento de parte sensores
        if (contador==4):
            GPIO.output(21, GPIO.HIGH)
            contador=0
        elif (contador==2):
            GPIO.output(21, GPIO.LOW)
            contador=contador+1
        else:
            contador=contador+1
    except:
        pass
```

E.7.- SCRIPT Nº 7- *"sensores.py"* : Programa para el almacenamiento y adquisición de datos de sensores de luz, calidad del aire, temperatura y humedad

```
# -*- coding:utf-8 -*-
#!/usr/bin/python
#Importación de librerías necesarias
import RPi.GPIO as GPIO
import time
import sys
import Adafruit_DHT
import sqlite3
import datetime
from gpiozero  import MCP3008
from mq import *
import smbus
import os

#Definición de pines del GPIO de la Raspberry Pi a utilizar
pin_ruido=3
pin_gas=11
pin_luz=6
pin_tyh=24 #Este es nomenclatura diferente al resto

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(25, GPIO.OUT) #Tabla sensores

#Establecimiento de conexión con base de datos SQLite
conn=sqlite3.connect('DATOS_P1.db')
curs=conn.cursor()

#Contador para parpadeo de LED en funcionamiento
contador=0

#Definición de funciones para crear tabla y para añadir dato a tabla en SQLite
def create_table():
        curs.execute('CREATE TABLE IF NOT EXISTS Temperatura(keyword TEXT, unix REAL,
datestamp TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Humedad(keyword TEXT, unix REAL, datestamp
TEXT, value REAL)')
    curs.execute('CREATE TABLE IF NOT EXISTS Luz(keyword TEXT, unix REAL, datestamp
TEXT, value REAL)')
        curs.execute('CREATE TABLE IF NOT EXISTS Calidad_Aire(keyword TEXT, unix REAL,
datestamp TEXT, value REAL)')

def dynamic_data_entry(sensor,valor,tabla,unix,date):
    keyword=sensor
    value=valor
    curs.execute("INSERT INTO "+ tabla +" (keyword, unix, datestamp, value) VALUES(?, ?, ?,
?)",(keyword, unix, date, value))
    conn.commit()

#Creación de tabla/s
```

```
try:
        create_table()
except:

        create_table()

#Bucle para esperar a la calibración del sensor de calidad del aire MQ-135
salir=0

while salir==0:
        try:
                mq = MQ()
                salir=1

        except:
                salir=0

#Bucle principal
try:
        while True:
                try:

                        #Registro de marca de tiempo
                        unix=time.time()
        date=str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d %H:%M:%S'))



                        #Registro valor del sensor de luz y convierto a lux
                        luz_divider=MCP3008(pin_luz)
                        luz_vol=luz_divider.value
                        luz=str((165/(luz_vol))-50)


                        #Registro valor de temperatura y humedad mediante librería propia del sensor
                humidity, temperature = Adafruit_DHT.read_retry(22, pin_tyh)

                        #Registro valor de calidad del aire y lo paso a porcentaje en base a
                        #una referencia definida previamente
                        aire_divider=MCP3008(7)
                        aire_val=aire_divider.value
                        aire=15/(aire_val)

                        #Almaceno datos de calidad del aire y nivel de luminosidad en base de datos
                        dynamic_data_entry('Calidad aire (%)',aire,"Calidad_aire",unix,date)
                        dynamic_data_entry('Nivel de luz (lux)', luz,"Luz",unix,date)


        #PARPADEO de LED para avisar de funcionamiento de parte sensores
                        if (contador==1):
                        GPIO.output(25, GPIO.HIGH)
                        contador=0
                elif (contador==0):
                        GPIO.output(25, GPIO.LOW)
                        contador=1

                        #Almaceno datos de temperatura y humedad en base de datos
                        if humidity is not None and temperature is not None:
                                dynamic_data_entry('Temperatura (Grados C)',
temperature,"Temperatura",unix,date)
```

```
                    dynamic_data_entry('Humedad (%)', humidity,"Humedad",unix,date)
                else:
                    print('Failed to get reading. Try again!')

            except:
                pass


except KeyboardInterrupt:
    print ("ERROR")
    pass

finally:
    GPIO.cleanup()
```

E.8.- SCRIPT Nº 8- *"serial_GPS.py"* : Programa para el almacenamiento y adquisición de datos de GPS del módulo FONA 808

```
# -*- coding:utf-8 -*-
#!/usr/bin/python

#Importación de librerías necesarias
import time
import sqlite3
import sys
import datetime
import serial
import time
import subprocess
import RPi.GPIO as GPIO
import os

#Establecimiento de conexión con base de datos SQLite

conn=sqlite3.connect('DATOS_P1.db')
curs=conn.cursor()

#Definición de pines del GPIO de la Raspberry Pi a utilizar
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(20, GPIO.OUT) #Tabla sensores

#Contador para parpadeo de LED en funcionamiento
contador=0

#Definición de funciones para crear tabla y para añadir dato a tabla en SQLite
def create_table():
    curs.execute('CREATE TABLE IF NOT EXISTS Time_GPS(keyword TEXT, unix REAL,
datestamp TEXT, value REAL, date TEXT, vis TEXT, usados TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS LAT_GPS(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS LONG_GPS(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS ALTITUD_GPS(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')

def dynamic_data_entry(sensor,valor,tabla,unix,date):
    keyword=sensor
    value=valor
    curs.execute("INSERT INTO "+ tabla +"  (keyword, unix, datestamp, value) VALUES(?, ?, ?,
?)",(keyword, unix, date, value))

    conn.commit()

#Creación de tabla/s

create_table()

#Definición de variables
A=1

#Bucle principal
```

```
while True:
    try:
        #Registro de marca de tiempo
        unix=time.time()
date=str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d %H:%M:%S'))

        #Conexión serie con módulo FONA
        ser = serial.Serial('/dev/serial0', 19200, timeout=0.5)

        #Encendemos módulo y pedimos información GPS
ser.write("AT+CGPSPWR=1\r")
        time.sleep(1)
ser.write("AT+CGNSINF\r")
        time.sleep(1)
response =  ser.readlines()

        #Analizamos respuesta y extraemos datos de interés
        for i in response:
            if "+CGNSINF:" in i:
                comas=0
                tiempo=" "
                latitud=" "
                longitud=" "
                altitud=" "
                sat_vis=" "
                sat_usa=" "
            print i
                for ch in i:
                    if ch==",":
                        comas=comas+1

                    if comas==2:
                        tiempo=tiempo+ch
                    if comas==3:
                        latitud=latitud+ch
            if comas==4:
                longitud=longitud+ch
              if comas==5:
                altitud=altitud+ch

            if comas==14:
                sat_vis=sat_vis+ch
            if comas==15:
                sat_usa=sat_usa+ch

                tiempo_2="Timestamp: " + tiempo[2:]


            lat2="Latitud : " + latitud[2:]
            print lat2

            lon2="Longitud: " + longitud[2:]
            print lon2

            alt2="Altitud: " + altitud[2:]
            print alt2

            vis="Numero de satelites visibles : " + sat_vis[2:]
            print vis
```

```
                usa="Numero de satelites usados: " + sat_usa[2:]
                print usa

        try:
                date_gps=str(datetime.datetime.fromtimestamp(float(tiempo[2:])).strftime('%Y-%m-
%d %H:%M:%S'))
        except:

                #Código para tener en cuenta los cambios de hora de España en 2018. En otro año
hay que cambiar el código y seleccionar los días de octubre y marzo que toquen ese año
                if (int(tiempo[6:8])>2) and (int(tiempo[6:8])<11):
                        if (int(tiempo[6:8])==3) and (int(tiempo[8:10])<25):
                                hora_gps=str(int(tiempo[10:12])+1)
                        elif (int(tiempo[6:8])==10) and (int(tiempo[8:10])>27):
                                hora_gps=str(int(tiempo[10:12])+1)
                        else:
                                hora_gps=str(int(tiempo[10:12])+2)
                else:
                        hora_gps=str(int(tiempo[10:12])+1)


                date_gps=str(tiempo[2:6]+"-"+tiempo[6:8]+"-"+tiempo[8:10]+"
"+hora_gps+":"+tiempo[12:14]+":"+tiempo[14:16])
                print(date_gps)
        curs.execute("INSERT INTO "+ "Time_GPS" +"  (keyword, unix, datestamp, value, date, vis,
usados) VALUES(?, ?, ?, ?, ?, ?, ?)",('Timestamp modulo GPS', unix, date, tiempo[2:], date_gps,
sat_vis[2:], sat_usa[2:]))

        conn.commit()

                #Almacenamiento de datos en base de datos

        dynamic_data_entry('Latitud modulo GPS',latitud[2:],"LAT_GPS",unix,date)
        dynamic_data_entry('Longitud modulo GPS',longitud[2:],"LONG_GPS",unix,date)
        dynamic_data_entry('Altitud modulo GPS',altitud[2:],"ALTITUD_GPS",unix,date)


        #PARPADEO de LED para avisar de funcionamiento de parte sensores
                if (contador==1):
            GPIO.output(20, GPIO.HIGH)
            contador=0
         elif (contador==0):
            GPIO.output(20, GPIO.LOW)
            contador=1

              if (int(hora_gps)<10):
                        hora_gps="0"+hora_gps
              elif (int(hora_gps)==24):
                        hora_gps="00"

              #Cambio de hora en la Raspberry Pi según hora de GPS
              subprocess.call(["sudo date
"+tiempo[6:8]+tiempo[8:10]+hora_gps+tiempo[12:14]+tiempo[2:6]+"."+tiempo[14:16]],shell=True)

        except:
                A=1
                time.sleep(1)


ser.close()
```

E.9.- SCRIPT Nº 9- *"serverudp_huawei_2.py"* : Programa para el almacenamiento de datos del acelerómetro y giroscopio del smartwatch Huawei Watch 2

```
# -*- coding:utf-8 -*-
#!/usr/bin/python

#Importación de librerías necesarias
import socket
import time
import sqlite3
import sys
import datetime
import select

#Establecimiento de conexión con base de datos SQLite
conn=sqlite3.connect('DATOS_P1.db')
curs=conn.cursor()

#Definición de funciones para crear tabla y para añadir dato a tabla en SQLite
def create_table():
    curs.execute('CREATE TABLE IF NOT EXISTS ACC_X_Reloj(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS ACC_Y_Reloj(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS ACC_Z_Reloj(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS GYRO_X_Reloj(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS GYRO_Y_Reloj(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')
    curs.execute('CREATE TABLE IF NOT EXISTS GYRO_Z_Reloj(keyword TEXT, unix REAL,
datestamp TEXT, value TEXT)')

def dynamic_data_entry(sensor,valor,tabla,unix,date):
    keyword=sensor
    value=valor
    curs.execute("INSERT INTO "+ tabla +"  (keyword, unix, datestamp, value) VALUES(?, ?, ?,
?)",(keyword, unix, date, value))

    conn.commit()

#Creación de tabla/s
create_table()

#Definición de dirección y puerto del servidor UDP
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
host="192.168.42.1" #RASPI IP
port1=2223
s.bind((host,port1))

#Inicialización de variables
rate=" "
latwt=" "
lonwt=" "
```

```
#Bucle principal
while True:
        try:
                rate=" "
                latwt=" "
                lonwt=" "

                cont=0

                #Esperamos a recibir datos del cliente
                 data, addr=s.recvfrom(1024)

                #Cuando recibo datos registro marca de tiempo
                 unix=time.time()
                 date=str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d %H:%M:%S'))

                #Analizo los datos recibidos para ver que tipo son
                #luego trato la información y la almaceno en la base de datos
                if (data[0]=='X'):
                                datasp=data.split(",")
                                ratesp=datasp[0].split(":")
                                latesp=datasp[1].split(":")
                                lonesp=datasp[2].split(":")

                                dynamic_data_entry('ACC_X',ratesp[1],"ACC_X_Reloj",unix,date)
                                dynamic_data_entry('ACC_Y',latesp[1],"ACC_Y_Reloj",unix,date)
                                dynamic_data_entry('ACC_Z',lonesp[1],"ACC_Z_Reloj",unix,date)

                elif(data[0]=='G'):
                                datasp2=data.split(",")
                                ratesp2=datasp2[0].split(":")
                                latesp2=datasp2[1].split(":")
                                lonesp2=datasp2[2].split(":")

                                dynamic_data_entry('GYRO_X',ratesp2[1],"GYRO_X_Reloj",unix,date)
                                dynamic_data_entry('GYRO_Y',latesp2[1],"GYRO_Y_Reloj",unix,date)
                                dynamic_data_entry('GYRO_Z',lonesp2[1],"GYRO_Z_Reloj",unix,date)

                #Si hay algún error almaceno "Null"
        except:

                unix=time.time()
                date=str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d %H:%M:%S'))

                print("error")
                dynamic_data_entry('ACC_X',"Null","ACC_X_Reloj",unix,date)
                dynamic_data_entry('ACC_Y',"Null","ACC_Y_Reloj",unix,date)
                dynamic_data_entry('ACC_Z',"Null","ACC_Z_Reloj",unix,date)
                dynamic_data_entry('GYRO_X',"Null","GYRO_X_Reloj",unix,date)
                dynamic_data_entry('GYRO_Y',"Null","GYRO_Y_Reloj",unix,date)
                dynamic_data_entry('GYRO_Z',"Null","GYRO_Z_Reloj",unix,date)
```

El script "serverudp_huawei.py" es equivalente a este, simplemente cambia el puerto del servidor y el tratamiento de datos que se adapta a la recepción de la información de frecuencia cardiaca y GPS.

E.10.- SCRIPT Nº 10- *"serverudp_empatica.py"* : Programa para la recepción y almacenamiento de datos procedentes de la pulsera Empatica E4. Gestiona también el proceso de conexión con el dispositivo

```
# -*- coding:utf-8 -*-
#!/usr/bin/python

#Importación de librerías necesarias
import socket
import time
import sqlite3
import sys
import datetime
import RPi.GPIO as GPIO


#Establecimiento de conexión con base de datos SQLite
conn=sqlite3.connect('DATOS_P1.db')
curs=conn.cursor()

#Definición de pines del GPIO de la Raspberry Pi a utilizar
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(22, GPIO.OUT)
GPIO.setup(27, GPIO.OUT)
GPIO.setup(17, GPIO.OUT)

#Definición de funciones para crear tabla y para añadir dato a tabla en SQLite
def create_table():
	curs.execute('CREATE TABLE IF NOT EXISTS BVP(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
	curs.execute('CREATE TABLE IF NOT EXISTS GSR(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
	curs.execute('CREATE TABLE IF NOT EXISTS IBI(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')
	curs.execute('CREATE TABLE IF NOT EXISTS TEMP(keyword TEXT, unix REAL, datestamp
TEXT, value TEXT)')

def dynamic_data_entry(sensor,valor,tabla,unix,date):
	keyword=sensor
	value=valor
	curs.execute("INSERT INTO "+ tabla +"  (keyword, unix, datestamp, value) VALUES(?, ?, ?,
?)",(keyword, unix, date, value))

	conn.commit()


#Creación de tabla/s
create_table()


#Definición de dirección y puerto del servidor UDP
```

```
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
host="192.168.42.1" #RASPI IP
port=12345
s.bind((host,port))

#Inicialización de variables
rate=" "
latwt=" "
lonwt=" "


#Bucle principal
while True:
        try:
                #Esperamos a recibir datos del cliente
                cont=0
           data, addr=s.recvfrom(1024)

                #Cuando recibo datos registro marca de tiempo
                unix=time.time()
           date=str(datetime.datetime.fromtimestamp(unix).strftime('%Y-%m-%d %H:%M:%S'))

                #Analizo los datos recibidos y compruebo si se envían notificaciones de estado
                #de conexión o se están transmitiendo datos

                #ESi se recibe una notificación se procede a encender un determinado led para
                #informar al usuario de cómo debe proceder en el proceso de conexión.

                if ("rojo" in data):
                        GPIO.output(17, GPIO.HIGH)
                        GPIO.output(27, GPIO.LOW)
                        GPIO.output(22, GPIO.LOW)

      elif ("azul" in data):
         GPIO.output(17, GPIO.LOW)
          GPIO.output(27, GPIO.LOW)
          GPIO.output(22, GPIO.HIGH)

      elif ("verde" in data):
          GPIO.output(17, GPIO.LOW)
          GPIO.output(27, GPIO.HIGH)
          GPIO.output(22, GPIO.LOW)

      #Si recibimos datos se procede a tratar la información y almacenarla en la BD
                else
                        datasp=data.split(",")
                        bvpesp=datasp[0].split(":")
                        gsresp=datasp[1].split(":")
                        ibiesp=datasp[2].split(":")
                        tempesp=datasp[3].split(":")

                        #Almaceno los datos de interés en la base de datos
                        dynamic_data_entry('BVP',bvpesp[1],"BVP",unix,date)
                        dynamic_data_entry('GSR',gsresp[1],"GSR",unix,date)
                        dynamic_data_entry('IBI',ibiesp[1],"IBI",unix,date)
                        dynamic_data_entry('Temperatura',tempesp[1],"TEMP",unix,date)

        except:
                pass
```

F.- APP ANDROID EMPLEADA PARA ADQUISICIÓN Y ENVÍO DE DATOS MEDIANTE EL SMARTWATCH HUAWEI WATCH 2

En este documento se exponen los archivos y programas de la aplicación Android que se corresponde con una modificación de una aplicación empleada en proyectos anteriores del grupo de investigación. Esta modificación permite adaptar la aplicación para los nuevos objetivos de este trabajo fin de master, que se basan en recoger datos de ritmo cardiaco, aceleración, rotación y posicionamiento GPS mediante los sensores que proporciona el reloj Huawei Watch 2. Los archivos que aquí se muestran tiene como objetivo entender las funcionalidades que se incluyen en la aplicación y ver principalmente las tareas o modificaciones que se han llevado a cabo en este TFM.

F.1.- SCRIPT Nº 1- *"MyService.java"*

Este servicio tiene la función principal de recoger los datos de diferentes tipos de sensores del smartwatch. Estos sensores serían el sensor de posicionamiento GPS, el giroscopio, el acelerómetro y sobre todo el sensor de ritmo cardiaco. Tras registrar el dato la aplicación deberá enviar mediante datagramas UDP estos datos al servidor (que será la SBC principal del sistema central emabrcado). Se pueden ver algunos detalles en este servicio como que el envío de mensajes de aceleración y rotación se ha limitado para sólo enviar valores altos y no saturar el servidor ya que la capacidad de muestreo de este sensor es muy elevada.

```
package es.uniovi.victorcorcobamagana.heartratetracker;

import android.Manifest;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.location.Location;
import android.net.wifi.WifiManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.PowerManager;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.app.ActivityCompat;
import android.util.Log;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.common.api.ResultCallback;
import com.google.android.gms.common.api.Status;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.wearable.Wearable;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.util.Timer;
import java.util.TimerTask;

public class MyService extends Service implements LocationListener, SensorEventListener,
GoogleApiClient.ConnectionCallbacks,
      GoogleApiClient.OnConnectionFailedListener {

   private File sdcard;
```

```
    private File dir;
    private File file;
    private Boolean first = true;
    public static float velocidad, latitud, longitud, precision, latitudAnterior, longitudAnterior, distancia=0;
    public static String latstr, lonstr;
    private FileOutputStream fOut;
    private PrintStream ps;
    Runnable runnable;
    public static String hearrate ="0.0";
    SensorManager mSensorManager;
    Sensor mHeartRateSensor;
    SensorEventListener sensorEventListener;
    private GoogleApiClient mGoogleApiClient;

// ACELEROMETRO
    SensorManager mSensorManager2;
    Sensor mAccelSensor;
    public static String accx ="0.0";
    public static String accy ="0.0";
    public static String accz ="0.0";
    public static Integer contador2 =0; //Contador para que envíe el dato del acelerometro cada más tiempo, para
ahorrar batería en trayectos largos

// GIROSCOPIO
    SensorManager mSensorManager3;
    Sensor mGyroSensor;
    public static String gyrox ="0.0";
    public static String gyroy ="0.0";
    public static String gyroz ="0.0";
    public static Integer contador3 =0; //Contador para que envíe el dato del acelerometro cada más tiempo, para
ahorrar batería en trayectos largos


    private Boolean primeraVezLocation = true;
    private Location locationAnterior;
    Timer timer;
    Thread thread;
    public static Integer contador =0; //Contador para que envíe el pulso cada más tiempo, para ahorrar batería en
trayectos largos


    public MyService() {
    }

    private class PerformBackgroundTask extends AsyncTask {

        @Override
        protected Object doInBackground(Object[] objects) {
            new Thread() {
                public void run() {

                }
            }.start();

            return null;
        }
    }
    public void callAsynchronousTask() {
        final Handler handler = new Handler();
        timer = new Timer();
        TimerTask doAsynchronousTask = new TimerTask() {
            @Override
            public void run() {
                handler.post(new Runnable() {
                    public void run() {
```

```
                try {
                    MyService.PerformBackgroundTask performBackgroundTask = new
MyService.PerformBackgroundTask();
                    // PerformBackgroundTask this class is the class that extends AsynchTask
                    performBackgroundTask.execute();
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                }
            }
        });
    }
};

timer.scheduleAtFixedRate(doAsynchronousTask, 0, 1000); //execute in every 50000 ms
}

private static WifiManager.WifiLock wifiLock;

public static void keepWiFiOn(Context context, boolean on) {
    if (wifiLock == null) {
        WifiManager wm = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
        if (wm != null) {
            wifiLock = wm.createWifiLock(WifiManager.WIFI_MODE_FULL, "dsfd");
            wifiLock.setReferenceCounted(true);
        }
    }
    if (wifiLock != null) { // May be null if wm is null
        if (on) {
            wifiLock.acquire();
            Log.d("", "Acquired WiFi lock");
        } else if (wifiLock.isHeld()) {
            wifiLock.release();
            Log.d("", "Released WiFi lock");
        }
    }
}

PowerManager.WakeLock mWakeLock;
@Override
public void onCreate() {
    super.onCreate();

    // HEARTRATE
    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mHeartRateSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
    first=true;
    mSensorManager = ((SensorManager) getSystemService(SENSOR_SERVICE));
    mSensorManager.registerListener(this, mHeartRateSensor, SensorManager.SENSOR_DELAY_FASTEST);

    //ACELEROMETRO
    mSensorManager2 = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mAccelSensor = mSensorManager2.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    first=true;
    mSensorManager2 = ((SensorManager) getSystemService(SENSOR_SERVICE));
    mSensorManager2.registerListener(this, mAccelSensor, SensorManager.SENSOR_DELAY_FASTEST);

    //GIROSCOPIO
    mSensorManager3 = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mGyroSensor = mSensorManager3.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
    first=true;
    mSensorManager3 = ((SensorManager) getSystemService(SENSOR_SERVICE));
    mSensorManager3.registerListener(this, mGyroSensor, SensorManager.SENSOR_DELAY_FASTEST);

    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addApi(LocationServices.API)
```

```
            .addApi(Wearable.API)  // used for data layer API
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .build();
    mGoogleApiClient.connect();
    callAsynchronousTask();


    final PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
    mWakeLock = pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "My Tag");
    mWakeLock.acquire();



}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    super.onStartCommand(intent, flags, startId);
    return START_STICKY;
}

@Override
public void onDestroy() {
    super.onDestroy();
    mGoogleApiClient.disconnect();
    mSensorManager.unregisterListener(this);

    mSensorManager2.unregisterListener(this);

    mSensorManager3.unregisterListener(this);

    try {
        fOut.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    ps.close();
    timer.cancel();


}
@Override
public IBinder onBind(Intent intent) {
    // TODO: Return the communication channel to the service.
    throw new UnsupportedOperationException("Not yet implemented");
}


@Override
public void onLocationChanged(Location location) {
    velocidad=location.getSpeed();
    latitud= (float)location.getLatitude();
    longitud= (float)location.getLongitude();
    latstr=Float.toString(latitud);
    lonstr=Float.toString(longitud);
    precision= location.getAccuracy();

    if(primeraVezLocation){
        locationAnterior=location;
        primeraVezLocation=false;
    }
    else{
        distancia += distancia+location.distanceTo(locationAnterior);
```

```
            locationAnterior = location;
        }

    }


    @Override
    public void onConnected(@Nullable Bundle bundle) {
        Log.d ("Depuración","Conectado");
        LocationRequest locationRequest = LocationRequest.create()
                .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
                .setInterval(300)
                .setFastestInterval(300);

        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
return;
        }
        LocationServices.FusedLocationApi
                .requestLocationUpdates(mGoogleApiClient, locationRequest, this)
                .setResultCallback(new ResultCallback<Status>() {

                    @Override
                    public void onResult(Status status) {
                        if (status.getStatus().isSuccess()) {
                            if (Log.isLoggable("Depuración", Log.DEBUG)) {
                                Log.d("Depuración", "Successfully requested location updates");
                            }
                        } else {
                            Log.e("Depuración",
                                    "Failed in requesting location updates, "
                                            + "status code: "
                                            + status.getStatusCode()
                                            + ", message: "
                                            + status.getStatusMessage());
                        }
                    }
                });

    }

    @Override
    public void onConnectionSuspended(int i) {

    }

    @Override
    public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {

    }

    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_HEART_RATE) {
            if (contador == 5) {
                hearrate = "" + (int) event.values[0];
                thread = new Thread(new EnviarMensagem("Ritmo: " + hearrate + ", Latitud: " + latstr + ", Longitud: " +
lonstr + ","));

                thread.start();

                contador = 0;
            } else {
                contador = contador + 1;
            }
```

```
    }
        else if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            if (contador2==1) {
                accx = "" + (int) event.values[0];
                accy = "" + (int) event.values[1];
                accz = "" + (int) event.values[2];

                if (Integer.parseInt(accx)>11 || Integer.parseInt(accx)<-11 || Integer.parseInt(accy)>11 ||
Integer.parseInt(accy)<-11 || Integer.parseInt(accz)>11 || Integer.parseInt(accz)<-11) {


                    thread = new Thread(new EnviarMensagem2("X: " + accx + ", Y: " + accy + ", Z: " + accz + ","));

                    thread.start();

                    contador2 = 0;
                }
            }
        else {
            contador2 = contador2 + 1;
        }
    }
        else if (event.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
            if (contador3==1) {
                gyrox = "" + (int) event.values[0];
                gyroy = "" + (int) event.values[1];
                gyroz = "" + (int) event.values[2];

                if (Integer.parseInt(gyrox)>7 || Integer.parseInt(gyrox)<-7 || Integer.parseInt(gyroy)>7 ||
Integer.parseInt(gyroy)<-7 || Integer.parseInt(gyroz)>7 || Integer.parseInt(gyroz)<-7) {
                    Log.d("", "GYRO  XXXXXX: " + gyrox);
                    Log.d("", "GYRO  YYYYYY: " + gyroy);
                    Log.d("", "GYRO  ZZZZZZ: " + gyroz);

                    thread = new Thread(new EnviarMensagem2("GYROX: " + gyrox + ", GYROY: " + gyroy + ",
GYROZ: " + gyroz + ","));

                    thread.start();

                    contador3 = 0;
                }
            }
            else {
                contador3 = contador3 + 1;
            }
        }
    else
        Log.d("", "Unknown sensor type");
  }

  @Override
  public void onAccuracyChanged(Sensor sensor, int accuracy) {

  }

}
```

F.2.- SCRIPT Nº 2- *"Monitor.java"*

Clase necesaria para el arranque automático de la aplicación con un arranque del dispositivo wearable.

```
package es.uniovi.victorcorcobamagana.heartratetracker;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;


public class Monitor extends BroadcastReceiver {

   public void onReceive(Context context,Intent intent) {
      Intent i = new Intent(context, MainActivity.class);
      i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
      context.startActivity(i);
   }
}
```

F.3.- SCRIPT Nº 3- *"EnviarMensagem.java"*

Clase necesaria para el envío de datagramas UDP al servidor. Se han creado un total de 2 clases variando el puerto de recepción del servidor para realizar balanceo de carga y no saturar un puerto del servidor con todos los datos, de este modo GPS y ritmo cardiaco se envían a un puerto y los datos del acelerómetro y el giroscopio a otro.

```
package es.uniovi.victorcorcobamagana.heartratetracker;

import android.util.Log;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

//Created by Lucas Altmann on 30/05/2017.

public class EnviarMensagem implements Runnable {

   private String mensagem;
   public EnviarMensagem(String mensagem) {
      this.mensagem = mensagem;
   }

   @Override
   public void run() {
      try {
         int PORTA_SERVIDOR = 2222;
         DatagramSocket s = new DatagramSocket(2007);
         InetAddress ADDRESS = InetAddress.getByName("192.168.42.1");
         byte[] message = this.mensagem.getBytes();
         DatagramPacket p = new DatagramPacket(message, this.mensagem.length(), ADDRESS,
PORTA_SERVIDOR);
         s.send(p);
         s.close();
         //Log.d("", mensagem);
      } catch (Exception e) {
         Log.d("EXCEPTION", e.getMessage());

      }
   }
}
```

G.- APP ANDROID EMPLEADA PARA ADQUISICIÓN Y ENVÍO DE DATOS MEDIANTE LA PULSERA DE ACTIVIDAD EMPATICA E4

En este documento se exponen los archivos y programas de la aplicación Android que se corresponde con una modificación de la aplicación de ejemplo suministrada por los propietarios de Empatica. Sobre los archivos se pueden ver las principales modificaciones que tienen que ver con el envío de mensajes a un servidor externo implementando para ello un cliente UDP. Su puede observar también el envío de los diferentes datos así como el envío de notificaciones que se identifican con las palabras "verde", "azul" y "rojo" que pretenden informar al servidor del estado en el que se encuentra la conexión entre la pulsera Empatica E4 y el receptor Bluetooth con el objetivo de realizar los pasos necesarios para lograr un emparejamiento satisfactorio de ambos dispositivos y una entrada en el modo de funcionamiento de streaming de la pulsera.

G.1.- SCRIPT Nº 1- *"MainActivity.java"* :

```
package com.empatica.sample;

import android.Manifest;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.provider.Settings;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.text.TextUtils;
import android.view.View;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;
import com.empatica.empalink.ConnectionNotAllowedException;
import com.empatica.empalink.EmpaDeviceManager;
import com.empatica.empalink.config.EmpaSensorStatus;
import com.empatica.empalink.config.EmpaSensorType;
import com.empatica.empalink.config.EmpaStatus;
import com.empatica.empalink.delegate.EmpaDataDelegate;
import com.empatica.empalink.delegate.EmpaStatusDelegate;


public class MainActivity extends AppCompatActivity implements EmpaDataDelegate,
EmpaStatusDelegate {

    private static final int REQUEST_ENABLE_BT = 1;
    private static final int REQUEST_PERMISSION_ACCESS_COARSE_LOCATION = 1;

    private static final long STREAMING_TIME = 10000; // Stops streaming 10 seconds after connection

    private static final String EMPATICA_API_KEY = "40b6adf5edd94541a299b896b2da9869"; //
TODO insert your API Key here

    private EmpaDeviceManager deviceManager = null;

    private TextView accel_xLabel;
    private TextView accel_yLabel;
    private TextView accel_zLabel;
    private TextView bvpLabel;
    private TextView edaLabel;
    private TextView ibiLabel;
    private TextView temperatureLabel;
    private TextView batteryLabel;
    private TextView statusLabel;
```

```
    private TextView deviceNameLabel;
    private RelativeLayout dataCnt;
    Thread thread;
    Thread thread2;
    Thread thread3;
    Thread thread4;
    public Float GSR_v;
    public Float TEMP_v;
    public Float IBI_v;
    public Integer red_v=0;
    public Integer blue_v=0;
    public Integer green_v=0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Initialize vars that reference UI components
        thread = new Thread(new EnviarMensagem("Hola! Estoy enviando..."));
        thread.start();

        statusLabel = (TextView) findViewById(R.id.status);
        dataCnt = (RelativeLayout) findViewById(R.id.dataArea);
        accel_xLabel = (TextView) findViewById(R.id.accel_x);
        accel_yLabel = (TextView) findViewById(R.id.accel_y);
        accel_zLabel = (TextView) findViewById(R.id.accel_z);
        bvpLabel = (TextView) findViewById(R.id.bvp);
        edaLabel = (TextView) findViewById(R.id.eda);
        ibiLabel = (TextView) findViewById(R.id.ibi);
        temperatureLabel = (TextView) findViewById(R.id.temperature);
        batteryLabel = (TextView) findViewById(R.id.battery);
        deviceNameLabel = (TextView) findViewById(R.id.deviceName);

        initEmpaticaDeviceManager();
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull
int[] grantResults) {
        switch (requestCode) {
            case REQUEST_PERMISSION_ACCESS_COARSE_LOCATION:
                // If request is cancelled, the result arrays are empty.
                if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                    // Permission was granted, yay!
                    initEmpaticaDeviceManager();
                } else {
                    // Permission denied, boo!
                    final boolean needRationale = ActivityCompat.shouldShowRequestPermissionRationale(this,
Manifest.permission.ACCESS_COARSE_LOCATION);
                    new AlertDialog.Builder(this)
                            .setTitle("Permission required")
                            .setMessage("Without this permission bluetooth low energy devices cannot be found,
allow it in order to connect to the device.")
                            .setPositiveButton("Retry", new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int which) {
                                    // try again
                                    if (needRationale) {
```

```
                        // the "never ask again" flash is not set, try again with permission request
                        initEmpaticaDeviceManager();
                    } else {
                        // the "never ask again" flag is set so the permission requests is disabled, try open
app settings to enable the permission
                        Intent intent = new
Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS);
                        Uri uri = Uri.fromParts("package", getPackageName(), null);
                        intent.setData(uri);
                        startActivity(intent);
                    }
                }
            })
            .setNegativeButton("Exit application", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    // without permission exit is the only way
                    finish();
                }
            })
            .show();
        }
        break;
    }
  }

  private void initEmpaticaDeviceManager() {
    // Android 6 (API level 23) now require ACCESS_COARSE_LOCATION permission to use BLE
    if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this, new String[] {
Manifest.permission.ACCESS_COARSE_LOCATION },
REQUEST_PERMISSION_ACCESS_COARSE_LOCATION);

    } else {
      // Create a new EmpaDeviceManager. MainActivity is both its data and status delegate.
      deviceManager = new EmpaDeviceManager(getApplicationContext(), this, this);


      if (TextUtils.isEmpty(EMPATICA_API_KEY)) {
        new AlertDialog.Builder(this)
            .setTitle("Warning")
            .setMessage("Please insert your API KEY")
            .setNegativeButton("Close", new DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int which) {
                // without permission exit is the only way
                finish();
              }
            })
            .show();
        return;
      }
      // Initialize the Device Manager using your API key. You need to have Internet access at this
point.
      deviceManager.authenticateWithAPIKey(EMPATICA_API_KEY);
    }
  }

  @Override
```

```
    protected void onPause() {
        super.onPause();
        if (deviceManager != null) {
            deviceManager.stopScanning();
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (deviceManager != null) {
            deviceManager.cleanUp();
        }
    }

    @Override
    public void didDiscoverDevice(BluetoothDevice bluetoothDevice, String deviceName, int rssi, boolean
allowed) {
        // Check if the discovered device can be used with your API key. If allowed is always false,
        // the device is not linked with your API key. Please check your developer area at
        // https://www.empatica.com/connect/developer.php
        if (allowed) {
            // Stop scanning. The first allowed device will do.
            deviceManager.stopScanning();


            try {
                // Connect to the device
                deviceManager.connectDevice(bluetoothDevice);
                updateLabel(deviceNameLabel, "To: " + deviceName);
            } catch (ConnectionNotAllowedException e) {
                // This should happen only if you try to connect when allowed == false.
                Toast.makeText(MainActivity.this, "Sorry, you can't connect to this device",
Toast.LENGTH_SHORT).show();
            }
        }
    }

    @Override
    public void didRequestEnableBluetooth() {
        // Request the user to enable Bluetooth
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);

    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        // The user chose not to enable Bluetooth
        if (requestCode == REQUEST_ENABLE_BT && resultCode == Activity.RESULT_CANCELED)
{
            // You should deal with this
            return;
        }
        super.onActivityResult(requestCode, resultCode, data);
    }

    @Override
    public void didUpdateSensorStatus(EmpaSensorStatus status, EmpaSensorType type) {
```

```
    // No need to implement this right now
}

@Override
public void didUpdateStatus(EmpaStatus status) {
    // Update the UI
    updateLabel(statusLabel, status.name());

    // The device manager is ready for use
    if (status == EmpaStatus.READY) {

        while (blue_v<3){
            thread3 = new Thread(new EnviarMensagem("azul"));
            thread3.start();
            blue_v=blue_v+1;
        }
        updateLabel(statusLabel, status.name() + " - Turn on your device");
        blue_v=0;

        // Start scanning
        deviceManager.startScanning();
        thread4 = new Thread(new EnviarMensagem("azul"));
        thread4.start();

        // The device manager has established a connection
    } else if (status == EmpaStatus.CONNECTED) {
        // Stop streaming after STREAMING_TIME
        while (green_v<3){
            thread3 = new Thread(new EnviarMensagem("verde"));
            thread3.start();
            green_v=green_v+1;
        }
        green_v=0;
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                dataCnt.setVisibility(View.VISIBLE);

            }
        });
        // The device manager disconnected from a device
    } else if (status == EmpaStatus.DISCONNECTED) {
        while (red_v<3){
            thread3 = new Thread(new EnviarMensagem("rojo"));
            thread3.start();
            red_v=red_v+1;

        }
        red_v=0;
        updateLabel(deviceNameLabel, "");


    }
}

@Override
public void didReceiveAcceleration(int x, int y, int z, double timestamp) {
    updateLabel(accel_xLabel, "" + x);
    updateLabel(accel_yLabel, "" + y);
```

```
        updateLabel(accel_zLabel, "" + z);
        thread3 = new Thread(new EnviarMensagem("verde"));
        thread3.start();
    }

    @Override
    public void didReceiveBVP(float bvp, double timestamp) {
        updateLabel(bvpLabel, "" + bvp);
        thread2 = new Thread(new EnviarMensagem("BVP: "+bvp+ ", GSR: "+GSR_v+", IBI: "+IBI_v+",
Temperatura: "+TEMP_v+ ", Timestamp: "+ timestamp +","));
        thread2.start();

    }

    @Override
    public void didReceiveBatteryLevel(float battery, double timestamp) {
        updateLabel(batteryLabel, String.format("%.0f%%", battery * 100));
    }

    @Override
    public void didReceiveGSR(float gsr, double timestamp) {
        updateLabel(edaLabel, "" + gsr);
        GSR_v=gsr;
    }

    @Override
    public void didReceiveIBI(float ibi, double timestamp) {
        updateLabel(ibiLabel, "" + ibi);
        IBI_v=ibi;
    }

    @Override
    public void didReceiveTemperature(float temp, double timestamp) {
        updateLabel(temperatureLabel, "" + temp);
        TEMP_v=temp;
    }

    // Update a label with some text, making sure this is run in the UI thread
    private void updateLabel(final TextView label, final String text) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                label.setText(text);
            }
        });
    }
}
```

G.2.- SCRIPT Nº 2- *"EnviarMensagem.java"*

Clase empleada para hacer de cliente UDP y enviar los datagramas pertinentes con los datos recogidos a el servidor UDP que espera en la SBC del sistema central embarcado.

```
package com.empatica.sample;

import android.util.Log;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

//Created by Lucas Altmann on 30/05/2017.

public class EnviarMensagem implements Runnable {

   private String mensagem;

   public EnviarMensagem(String mensagem) {
      this.mensagem = mensagem;
   }

   @Override
   public void run() {
      try {
         int PORTA_SERVIDOR = 12345;
         DatagramSocket s = new DatagramSocket(2007);
         InetAddress ADDRESS = InetAddress.getByName("192.168.42.1");
         byte[] message = this.mensagem.getBytes();
         DatagramPacket p = new DatagramPacket(message, this.mensagem.length(), ADDRESS,
PORTA_SERVIDOR);
         s.send(p);
         s.close();
      } catch (Exception e) {
         Log.d("EXCEPTION", e.getMessage());
      }
   }
}
```