UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

“ PLATAFORMA DE MONITORIZACIÓN eHealth A TRAVÉS DEL DISPOSITIVO PHOTON EN EL ÁMBITO IoT ”

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación

Mención: Sistemas Electrónicos

Autor: D. Iván J. Santana Sosa

Tutores: D. Félix B. Tobajas Guerrero

D. Valentín De Armas Sosa

Fecha: Junio de 2016

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

" PLATAFORMA DE MONITORIZACIÓN eHealth A TRAVÉS DEL DISPOSITIVO PHOTON EN EL ÁMBITO IoT "

HOJA DE FIRMAS

Tutor 1 | **Tutor 2**

Fdo. D. Félix B. Tobajas Guerrero | Fdo. D. Valentín De Armas Sosa

Alumno

Fdo. D. Iván J. Santana Sosa

Junio de 2016

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

“ PLATAFORMA DE MONITORIZACIÓN eHealth A TRAVÉS DEL DISPOSITIVO PHOTON EN EL ÁMBITO IoT ”

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo. : _____

Vocal **Secretario/a**

Fdo.: _____ Fdo.: _____

Junio de 2016

Agradecimientos

-

A mi abuelo y abuela por su amor incondicional y lecciones de vida, nunca os olvidaré.

A mis padres, por haberme proporcionado la mejor educación y apoyo para seguir adelante.

A mis tíos, que siempre me han mimado y ayudado.

A mis tutores, Félix y Valentín, por su gran esfuerzo, atención y conocimientos.

A mis amigos, con los que he compartido momentos memorables.

Índice de Contenidos

-

Índice de Figuras

-

Índice de Tablas

-

Listado de acrónimos

-

ADC	*Analog to Digital Conversion*
AO	*Amplificador Operacional*
API	*Application Programming Interface*
ARM	*Acorn RISC Machine*
ASCII	*American Standard Code For Information Interchange*
AV	*Auriculoventricular*
BPM	*Beats Per Minute*
CAN	*Controller Area Network*
CLI	*Command Line Interface*
COITT	*Colegio Oficial de Ingenieros Técnicos de Telecomunicación*
CPU	*Central Processing Unit*
DAC	*Digital to Analogue Converter*
DAC	*Digital-to-Analog Converter*
DPS	*Degrees Per Second*
ECG	*Electrocardiograma*
EEPROM	*Electrically Erasable Programmable Read Only Memory*
EITE	*Escuela de Ingeniería de Telecomunicación y Electrónica*
FPA	*Filtro Paso Alto*
FPB	*Filtro Paso Bajo*

GND	*Ground*
GPIO	*General Purpose Input-Output*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
HW	*Hardware*
HW/SW	*Hardware/Software*
I^2C	*Inter-Integrated Circuit*
IA	*Amplificador de Instrumentación*
IBI	*Inter Beat Interval*
IBM	*International Business Machines*
IBSG	*Internet Business Solutions Group*
IDE	*Integrated Development Environment*
IGIC	*Impuesto General Indirecto Canario*
IoT	*Internet of Things*
IP	*Internet Protocol*
ITU	*Unión Internacional de Telecomunicaciones*
KB	*Kilobyte*
LA	*Left Arm*
LCD	*Liquid Crystal Display*
LED	*Light-Emitting Diode*
LL	*Left Leg*
MAF	*Moving Average Filter*
MB	*Megabyte*
MIT	*Massachusetts Institute of Technology*
NPM	*Node Package Manager*
PCB	*Printed Circuit Board*
PFC	*Proyecto fin de carrera*
PnP	*Plug-and-play*
PPG	*Photoplethysmography*
PWM	*Pulse-Width Modulation*
RA	*Right Arm*
RAM	*Random Access Memory*

RGB	*Red, Green, Blue*
RL	*Right Leg*
RTOS	*Real-time Operating System*
SA	*Sinoauricular*
SCL	*Serial Clock line*
SDA	*Serial Data Line*
SMPS	*Switched-Mode Power Supply*
SMS	*Short Message Service*
SPI	*Serial Peripheral Interface*
SSE	*Server Sent Events*
TCP	*Transmission Control Protocol*
TFG	*Trabajo Fin de Grado*
TIC	*Tecnologías de la Información y las Comunicaciones*
UDP	*User Datagram Protocol*
ULPGC	*Universidad de Las Palmas de Gran Canaria*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*

Memoria

-

Capítulo 1. Introducción

-

El este capítulo se detallan los antecedentes del presente TFG (Trabajo Fin de Grado), además de describirse los objetivos que se pretenden satisfacer con su realización e introducirse una base de conceptos relacionados con el ámbito IoT. Para finalizar, se presenta un resumen de la estructura de la memoria de este TFG.

1.1 Antecedentes

La evolución de las TIC (Tecnologías de la Información y las Comunicaciones) a partir de mediados del siglo XX ha revolucionado significativamente la sociedad en la que vivimos hoy en día. Este hecho ha modificado el estilo de vida de mucha gente y en consecuencia, se ha generado una dependencia de la tecnología cada vez mayor, lo que ha propiciado la constante evolución de las TIC.

Esta revolución se ha visto favorecida por la aparición de la era digital en la década de 1970, produciéndose hasta la época actual una tendencia de reducir al máximo las dimensiones físicas de los dispositivos hardware, el consumo de potencia o la complejidad de los sistemas (tanto software como hardware), pero manteniendo al mismo tiempo, o incluso mejorando, sus funcionalidades y prestaciones.

Desde la década de 1960 Internet ha evolucionado drásticamente llegando a establecerse como un recurso accesible para muchas personas, y que además ofrece una amplia variedad de beneficios. Los usuarios tienen mayor acceso a todo tipo de información y utilizan esta herramienta como medio de comunicación virtual; por otra parte las empresas disponen de este nuevo mecanismo que es vital hoy en día para su progreso y mejora.

Cualquier tipo de objeto que nos rodea diariamente ya tiene cabida en Internet, y lo que antes eran personas conectadas a Internet, hoy en día se ha convertido en una complejo ecosistema donde multitud de dispositivos se interconectan entre sí para la transferencia de información útil.

Es en este punto donde surge el concepto de Internet de las Cosas (*Internet of Things*, IoT). Este término fue impulsado en 1999 desde el Instituto de Tecnología de Massachusetts (MIT) y se ha convertido en uno de los conceptos más populares y comentados en la industria tecnológica en los últimos años.

Según la Unión Internacional de Telecomunicaciones (ITU), IoT se define como "Una infraestructura global para la sociedad de la información, lo que permite servicios avanzados mediante interconexión de diferentes elementos (físicos y virtuales) en base a la información existente y a las Tecnologías de la Información y las Comunicaciones"[1].

La transcendencia de este ámbito tecnológico podría ser de suma importancia en los próximos años. Así, el Grupo de Soluciones Empresariales para Internet (IBSG) de Cisco calcula que en 2015 habían 25.000 millones de dispositivos conectados a Internet, mientras que en 2020 el numero será de 50.000 millones [2] , estando este incremento asociado fundamentalmente a la introducción de IoT.

En la Figura 1 se muestran los principales campos de aplicación de IoT en la actualidad [3].

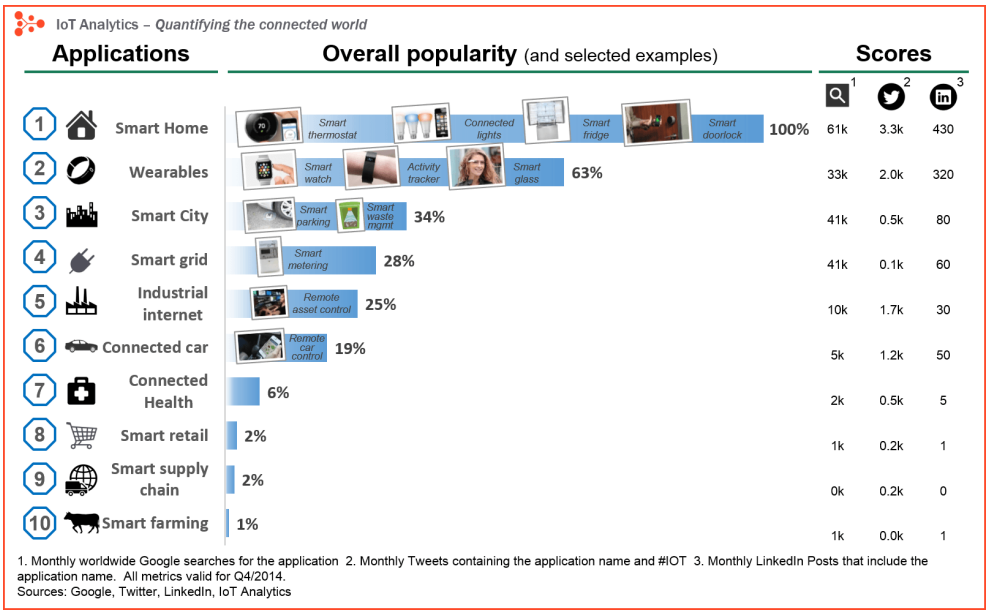


Figura 1. Principales campos de aplicación de IoT.

Así, se observa que en el punto 7 se encuentra el ámbito *Connected Health* y es hacia el que está orientado el presente TFG, recogiéndose unos niveles bajos de incidencia (6%) de donde resulta que actualmente puede estar desaprovechándose el enorme potencial que esta infraestructura ofrece en este campo de aplicación.

Desde el punto de vista de un usuario privado, la aplicación intensiva de IoT y de las TIC tiene el potencial de mejorar la autonomía de las personas, siendo visibles los efectos más evidentes en aspectos relacionados con el hogar y la salud personal. En este contexto, la domótica o la *eHealth* comprenden, respectivamente, dos de los principales ámbitos de aplicación de IoT [4]

Este tipo de tecnología, aplicado en el sector de la salud, sería de gran ayuda por ejemplo, para la monitorización de datos de las constantes vitales y parámetros médicos a través de Internet dirigido a la tercera edad, deportistas y en general, todo tipo de personas.

Puesto que se ofrece la posibilidad de usar esta tecnología en el hogar, los ciudadanos serían capaces de vivir de forma más independiente, aumentando el sentimiento de seguridad y de calidad de vida, además de ayudar a reducir el coste de equipos médicos y la necesidad de recursos de cuidadores para las personas mayores. Asimismo, los pacientes no tendrían que ir de manera reiterativa a la consulta, pudiendo

reducirse los tiempos de espera y lo que es aún más importante, los costes de personal y operaciones administrativas en Sanidad.

Por lo tanto, se requieren de nuevas tecnologías que ayuden en el análisis de los datos recogidos a través de los sensores y permitan, por ejemplo, predecir enfermedades de un paciente o avisar en caso de que se produzca alguna situación de emergencia. Además, otra ventaja significativa de IoT es que resulta factible el desarrollo de dispositivos de coste muy reducido capaces de garantizar estas facilidades.

Por otro lado, diversas empresas se dedican a la fabricación de sistemas empotrados para plataformas de desarrollo, microprocesadores o servicios en la nube en el ámbito de Internet de las Cosas, destacando Intel, Samsung, Google, IBM, ARM o Cisco, entre otras como la *start-up Particle* con su producto de bajo coste denominado "Photon", representado en la Figura 2 y que será el que se utilizará para el desarrollo del presente Trabajo Fin de Grado [5].

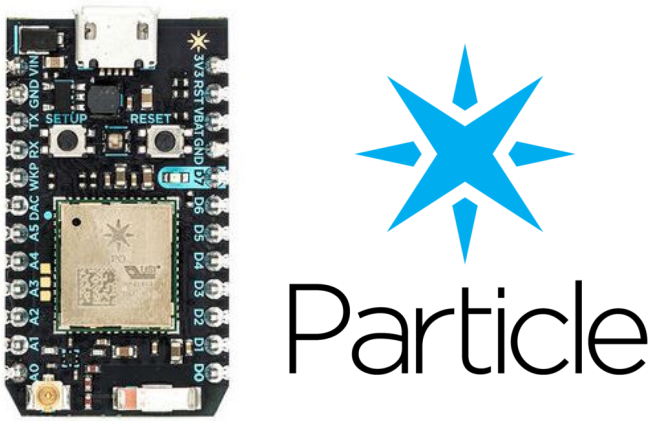


Figura 2. Plataforma de desarrollo hardware Photon.

1.2 Objetivos

El objetivo general de este TFG es el diseño e implementación de una aplicación de monitorización de parámetros de salud de bajo coste basado en el dispositivo Photon de la empresa *Particle*, permitiendo que el sistema/aplicación envíe datos biométricos, en este caso enfocados en la actividad eléctrica del corazón, recogidos a través de sensores, pudiendo estos ser recibidos en tiempo real a través de Internet con una latencia mínima.

El dispositivo Photon se basa en un kit de desarrollo *open source* para la creación de productos conectados a Internet a través de Wi-Fi, basado en la arquitectura *Wiced de Broadcom*, que combina un potente microcontrolador ARM Cortex M3 con un chip Broadcom Wi-Fi, alimentación a 3,3 VDC, así como 18 entradas / salidas de propósito general (GPIO) .

Como objetivos complementarios, se plantea la integración de un sensor con función de acelerómetro que detectará caídas de los usuarios que se comunicarán una vez se verifique por parte del propio paciente, o en caso de que no pueda, a través de un temporizador. Por consiguiente, en el desarrollo del presente TFG se plantean los siguientes objetivos específicos:

- Adquisición de los conocimientos acerca de Internet de las cosas (*Internet of Things,* IoT) y el funcionamiento básico de sus principales aplicaciones basadas en *eHealth* y su uso actual con fines dedicados en especial atención a personas de tercera edad o deportistas.
- Interpretar resultados electro cardiovasculares.
- Estudio de las características y funcionamiento del dispositivo Photon y sus componentes periféricos (sensores), así como la verificación del HW (hardware).
- Utilización de las herramientas software con las que se realizará el desarrollo del TFG.
- Desarrollo y diseño del *firmware* que permita enviar la señal ECG en tiempo real y el valor de los pulsos por minuto (*Beats per Minute,* BPM) a través de la red, además de detectar caídas.
- Integración y validación de la plataforma completa.

1.3 Peticionario

Actúa como peticionario del presente Trabajo Fin de Grado (TFG) la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE) de la Universidad de Las Palmas de Gran Canaria (ULPGC) como requisito indispensable para la obtención del título de

Grado en Ingeniería en Tecnologías de la Telecomunicación, tras haber superado con éxito las asignaturas especificadas en el Plan de Estudios.

1.4 Estructura del documento

La memoria está estructurada de la siguiente forma:

En el ***Capítulo 1*** se realiza una introducción que proporcionará al lector una visión básica acerca de los conceptos que se tratarán en el presente TFG. Además, se presentan también los objetivos a conseguir, un peticionario y la estructura que seguirá el documento.

El ***Capítulo 2*** explicará los principales conocimientos básicos biológicos y eléctricos del corazón, así como los principios de la electrocardiografía. Todo esto con el fin de poder interpretar y entender correctamente la señal ECG que proporcionará la plataforma, y adquirir conocimientos generales acerca de este órgano vital.

En el ***Capítulo 3*** se describe cómo funcionará la plataforma a nivel de hardware para posteriormente explicar detalladamente los distintos dispositivos que conformarán la plataforma final. Por otra parte se introduce el bus de comunicaciones I^2C presente en multitud de dispositivos actuales, y del cual se hará uso en este proyecto.

El ***Capítulo 4*** describe el proceso software llevado a cabo para la implementación de la aplicación *eHealth*. Se analizarán las herramientas que se han utilizado para desarrollar el *firmware* del presente TFG, así como una explicación detallada de las secciones más destacadas del código implementado. Una vez finalizado el capítulo se comprobará que se han completado los principales objetivos establecidos inicialmente en el presente TFG, que son la visualización del electrocardiograma en tiempo real y la detección de caídas.

El ***Capítulo 5*** comprende el funcionamiento y características de la plataforma final desarrollada. Primeramente se mostrará la interconexión final de la plataforma por medio de imágenes y tablas, para luego describir las modificaciones realizadas del *firmware* a lo largo del desarrollo del presente TFG. Finalmente se explicará el funcionamiento de la

plataforma de cara al usuario y los resultados obtenidos después de realizar diversas pruebas.

Para concluir, en el ***Capítulo 6*** se planteará una serie de conclusiones y líneas futuras en vista al trabajo realizado del presente TFG.

Capítulo 2. Conocimientos básicos cardiológicos

-

El propósito de este capítulo es el de presentar los conocimientos básicos sobre el corazón humano para posteriormente comprender el funcionamiento del prototipo desarrollado. Teniendo en cuenta que este tema ha sido ampliamente estudiado, la información contenida en las siguientes secciones se centrará en aquellos aspectos relevantes para el presente TFG.

2.1 Conocimientos biológicos sobre el corazón humano

Los tejidos del cuerpo humano requieren oxígeno y otros componentes esenciales para funcionar de manera correcta. El oxígeno se introduce en el organismo humano por medio de los pulmones y desde éstos se transporta hasta los receptores de oxígeno. De esta función de transporte se encargan los eritrocitos o los denominados glóbulos rojos.

La mayor parte de las personas conocen que el corazón es un órgano vital y que sin él es imposible vivir. Dicho órgano muscular tiene la función de bombear la sangre, y por consecuencia eritrocitos, a través de los vasos sanguíneos y capilares del cuerpo

humano, de modo que una vez lleguen los eritrocitos a todos las partes del cuerpo, se alimentará con oxígeno las células que conforman los tejidos, además de recoger el dióxido de carbono y las sustancias de desecho producidas por las células.

El corazón tiene un tamaño aproximadamente igual al de un puño cerrado, con un peso promedio de 250 gramos en el sexo femenino y 300 gramos en el masculino y cabe señalar que tiene menos del 0,5 % del peso corporal. Como datos interesantes, al final de la vida de una persona, este órgano puede llegar a latir (entendiéndose por dilatación y contracción) más de 3500 millones de veces y se debe agregar que diariamente, la media de latidos se encuentra entorno a las 100.000 veces, bombeando unos 7.570 litros de sangre.

A continuación se examinará la anatomía del corazón, que se puede observar en la Figura 3 [6]. Así, primeramente comentar que se ubica entre los pulmones y detrás del esternón. Es hueco en su interior y está dividido en cuatro cámaras o cavidades :

Dos superiores,

- Aurícula derecha (atrio derecho).
- Aurícula izquierda (atrio izquierdo).

Dos inferiores,

- Ventrículo derecho.
- Ventrículo izquierdo.

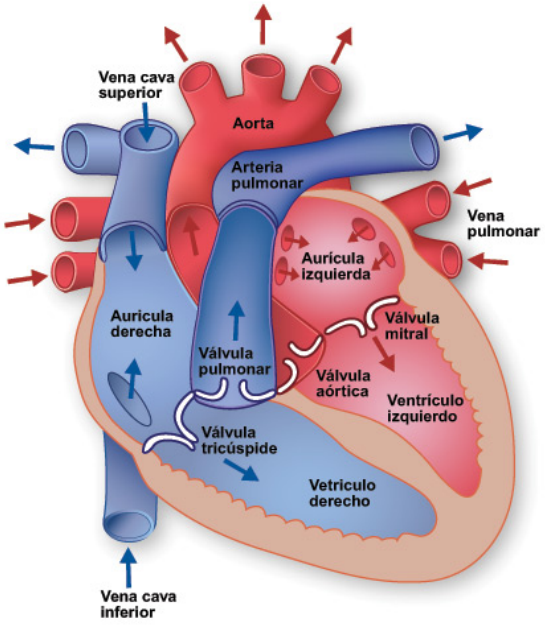


Figura 3. Anatomía del corazón

Una pared muscular, denominada "tabique", separa las aurículas izquierda y derecha y los ventrículos izquierdo y derecho. El ventrículo izquierdo es la cavidad más grande y fuerte del corazón. Las aurículas tienen la función de recepción de sangre del sistema sanguíneo y los ventrículos la función contraria (expulsión). Establecida la referencia del corazón, se procede a explicar su funcionamiento.

La mitad derecha recibe la sangre desoxigenada que ya ha pasado por los tejidos y capilares del cuerpo humano y la deriva hacia los pulmones (este circuito también es conocido como "circulación pulmonar"). La mitad izquierda hace el efecto contrario, recibe la sangre oxigenada que proviene de los pulmones y la bombea través de la arteria Aorta hacia los tejidos a del cuerpo humano por medio del sistema sanguíneo.

Cada vez que el corazón se contrae (o late), bombea sangre hacia afuera, las contracciones se denominan **sístoles** (significa contracciones en griego). El músculo cardíaco se contrae en dos etapas:

1. Se da cuando, tanto la aurícula derecha como la izquierda se contraen al mismo tiempo, bombeando o haciendo fluir la sangre hacia los ventrículos derecho e izquierdo.
2. Se da cada vez que se contraen simultáneamente los dos ventrículos, bombeando sangre hacia fuera del corazón.

Una vez finalizado el proceso sistólico (de contracción) el músculo cardíaco se relaja antes de la siguiente contracción o latido (sístole); esta relajación se denomina **diástole** (en griego significa dilatación) y permite que el corazón se llene de sangre. [7] Cuando se produce la relajación del órgano muscular, éste succiona la sangre de los conductos que la alimentan (las venas cava superior e inferior para la aurícula derecha y las venas pulmonares para la aurícula izquierda).

Dicho brevemente, el corazón es un órgano muscular autocontrolado, formado por dos bombas en paralelo que trabajan al mismo tiempo para propulsar la sangre hacia todos los órganos del cuerpo. Las aurículas son cámaras de recepción, que envían la sangre que reciben hacia los ventrículos, y estos funcionan como cámaras de expulsión.

2.2 Actividad eléctrica del corazón y fundamentos de la electrocardiografía

El electrocardiograma (ECG) es un registro muy útil de la función del corazón (actividad eléctrica) de modo que se registran los pulsos eléctricos que estimulan el corazón y producen su contracción.

Las células cardiacas en reposo se encuentran cargadas negativamente o polarizadas, pero la estimulación eléctrica, que es cuando se contrae el corazón, las despolariza y las carga positivamente. Por lo tanto, el corazón es recorrido por una onda progresiva de estimulación (despolarización) que produce contracción del miocardio (tejido muscular del corazón).

Las ondas de despolarización (células se vuelven positivas) y repolarización (células recuperan su carga negativa) se registran en el ECG y son fenómenos eléctricos. Cuando se produce la repolarización, el corazón no presenta ningún tipo de movimiento durante esta actividad.

Cuando esta actividad eléctrica recorre el corazón, se puede captar con electrodos externos (sobre la piel) y se puede registrar como ECG. El nodo sinoauricular (SA) se puede localizar visualmente en la Figura 4, y es donde se origina el impulso eléctrico que da origen a un latido cardiaco, que posteriormente se extiende como onda y estimula ambas aurículas.

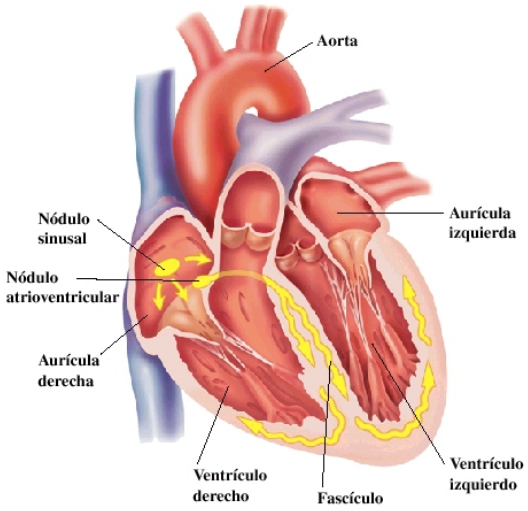


Figura 4. Sistema eléctrico cardíaco.

El impulso eléctrico generado por el nodo SA se registra como onda P en el ECG y por lo tanto representa la actividad eléctrica de la contracción (despolarización) de ambas aurículas. Después, este impulso llega al nodo auriculoventricular (AV), donde aparece una pequeña pausa, lo que permite que la sangre llegue a los ventrículos.

Después de esta pausa, el nodo AV es estimulado y se inicia un impulso eléctrico que se dirige hacia abajo por el haz de His y las ramas del mismo. En la Figura 5 se puede observar el recorrido del impulso eléctrico.

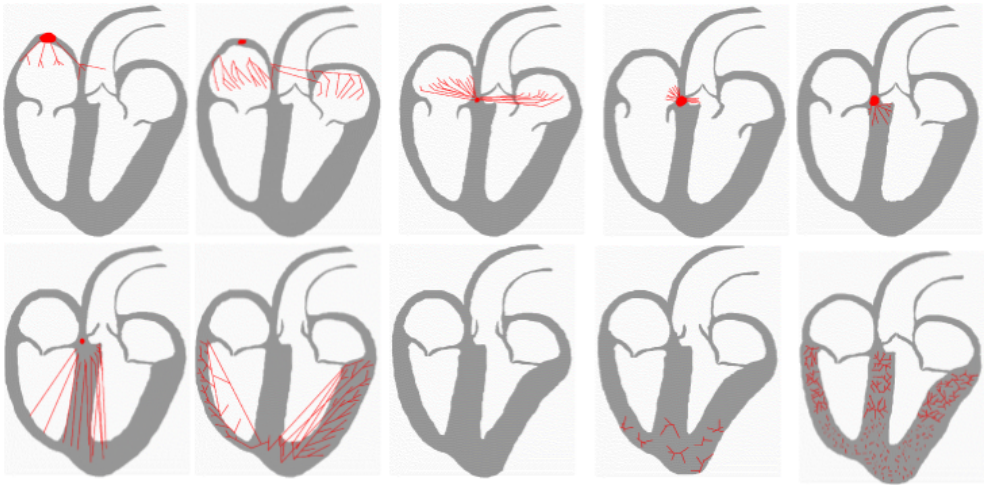


Figura 5. Camino del impulso eléctrico.

El complejo QRS se puede visualizar en la Figura 6, y representa el impulso eléctrico que se aleja del nodo AV, y por lo tanto se interpreta como la actividad eléctrica de la estimulación de los ventrículos (contracción). La onda Q es la primera deflexión hacia abajo del complejo QRS, seguida de la onda R hacia arriba. Cabe comentar que a menudo falta la onda Q. Luego, la onda R hacia arriba va seguida de una onda S hacia abajo.

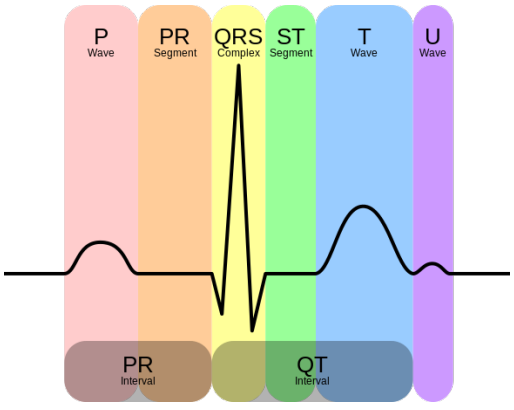


Figura 6. ECG completamente etiquetado.

Seguidamente, existe una pausa tras el complejo QRS y luego aparece la onda T, que representa la repolarización de los ventrículos para que se les pueda volver a estimular. Un ciclo cardiaco completo comprende la onda P, el complejo QRS y la onda T, repitiéndose este patrón continuamente. La altura o valor de la onda representa un voltaje y el eje horizontal representa el tiempo. Estos suelen representarse en un papel con una cuadricula milimétrica, en el que cada cuadrado de 1mm representa 40 ms (eje de las abscisas) y 1mV (eje de las ordenadas).

Como bien se comentaba anteriormente, para obtener el electrocardiograma se adhieren unos electrodos a la piel del sujeto, compuestos de almohadillas adhesivas que contienen un gel conductor y un borne, de forma que cada combinación de posiciones genera un punto de vista diferente, y cuantos más electrodos se hayan situado, mayor será en principio la información que se pueda tomar.

Hay cuatro electrodos principales, LA (*Left Arm*/Brazo Izquierdo), RA (*Right Arm*/ Brazo Derecho), LL (*Left Leg*/ Pierna Izquierda), RL (*Right Leg*/ Pierna derecha). Los electrocardiogramas más completos tienen estos cuatro electrodos, y otros seis denominados de V1 a V6, que se sitúan en el pecho. La colocación de estos electrodos se puede ver en el ejemplo de la Figura 7.

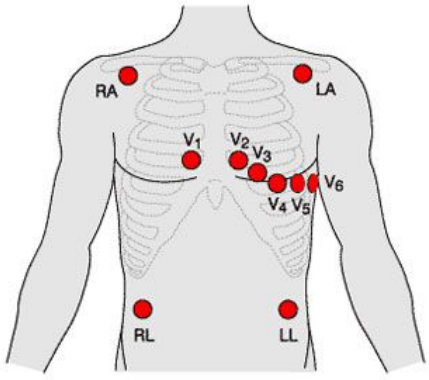


Figura 7. Posicionamiento de electrodos.

Cada punto de vista debido a la combinación de electrodos, se denomina *derivación*. La derivación más simple se halla realizando la resta de la lectura del potencial que se obtiene en el brazo derecho con la lectura del potencial en el brazo izquierdo, y se representa con el numero romano I (por tanto *derivación I*). Cada nueva derivación toma un registro desde un ángulo distinto, por lo que cada derivación es un aspecto diferente de la misma actividad cardiaca.

2.3 Frecuencia, ritmo cardíaco y eje eléctrico

Los tres puntos básicos en la descripción de un electrocardiograma son :

- La *frecuencia cardiaca*.
- El *ritmo cardiaco*.
- El *eje eléctrico*.

La *frecuencia cardiaca* se define como el número de latidos que se producen por minuto (latidos por minuto) a nivel del corazón, y en pulsaciones por minuto a nivel de arterias periféricas. La medición principal se enfoca hacia la frecuencia ventricular analizada por el complejo QRS. Sin embargo, no es la única porción que se puede medir en relación a la frecuencia, ya que también se puede definir de esta manera la frecuencia auricular, que resulta muy útil en algunas arritmias.

Dicha frecuencia se mide en condiciones específicas (reposo, actividad, etc.) y se puede efectuar la medida en varios puntos del cuerpo, siendo lo más común en la muñeca, el cuello o tórax. Lo dicho anteriormente se resume en la siguiente expresión:

$$f_{cardiaca} = \frac{nº\ de\ latidos}{minuto}$$

La determinación del *ritmo cardiaco* es muy importante, ya que indica el origen anatómico eléctrico del latido cardiaco. Ocasionalmente se confunde el término de ritmo con el de frecuencia cardiaca, pero ya se revisó previamente que la frecuencia cardiaca determina la velocidad del latido, mientras que el ritmo determina su origen.

Un ritmo normal que se origine en el nodo sinoauricular (SA) se denomina *ritmo sinusal*. Los criterios para poder establecer que un ritmo es sinusal es la presencia de ondas P seguidas de todo complejo QRS en todos los latidos, y cada ciclo sucesivo dura el mismo tiempo, lo que da una frecuencia continua estable.

Una frecuencia mayor de 100 lat./min (con ritmo normal) se denomina *taquicardia sinusal* y aparece en personas normales con la actividad física, o bien

relacionado con enfermedades que requieran de un mayor consumo de oxigeno como infecciones, etc.

Una frecuencia inferior a 60 lat./min (con ritmo normal) se denomina *bradicardia sinusal* y es frecuente en deportistas y pacientes con tratamientos con fármacos que reduzcan la frecuencia cardiaca.

Las pulsaciones por minuto son diferentes para cada persona, y difieren dependiendo de la edad y del género. Existen diversos métodos para calcular dicha frecuencia, pudiéndose determinar el valor máximo con la siguiente expresión:

$$f_{max} = 208 - 0.7 * edad$$

En la Tabla 1 se muestran los valores normales de la frecuencia cardíaca en reposo y en ejercicio.

Tabla 1. Valores normales de la frecuencia cardíaca.

	Adulto Sedentario	**Adulto en forma**	**Deportista**
Reposo Pulsaciones por minuto	Entre 70 y 90	Entre 60 y 80	Entre 40 y 60
Ejercicio aeróbico Pulsaciones por minuto	Entre 110 y 130	Entre 120 y 140	Entre 140 y 160
Ejercicio intenso Pulsaciones por minuto	Entre 130 y 150	Entre 140 y 160	Entre 160 y 200

Fuente:http://www.frecuencia-cardiaca.com/frecuenciacardiacavalores.php

La determinación del *eje eléctrico* es una de las partes que se consideran más complejas a estudiar por presentar diversos métodos. El eje eléctrico indica la sumatoria de todos los vectores de despolarización, auricular en la onda P, ventricular en el complejo QRS, y la repolarización ventricular en la onda T.

En el caso de aplicaciones clínicas, se acostumbra a determinar el eje eléctrico ventricular a partir de la utilización del complejo QRS, siendo la determinación exacta del eje eléctrico meramente académica/teórica y no práctica, por lo que se prefieren métodos no tan precisos, pero más prácticos.

Capítulo 3. Análisis Hardware

-

En este capítulo se muestra una descripción detallada con objeto de comprender el funcionamiento y las principales características de los dispositivos HW que formarán parte de la plataforma desarrollada en el presente TFG.

3.1 Plataforma HW

La plataforma HW propuesta para satisfacer los objetivos establecidos en el presente TFG se compone de los siguientes dispositivos electrónicos, que se puede visualizar en la Figura 8 :

- **Sensor ECG que obtiene la señal de la actividad eléctrica del corazón.**
- **Acelerómetro / Sensor de movimiento que detectará una caída.**
- **Pulsómetro.**
- **Buzzer / Zumbador.**

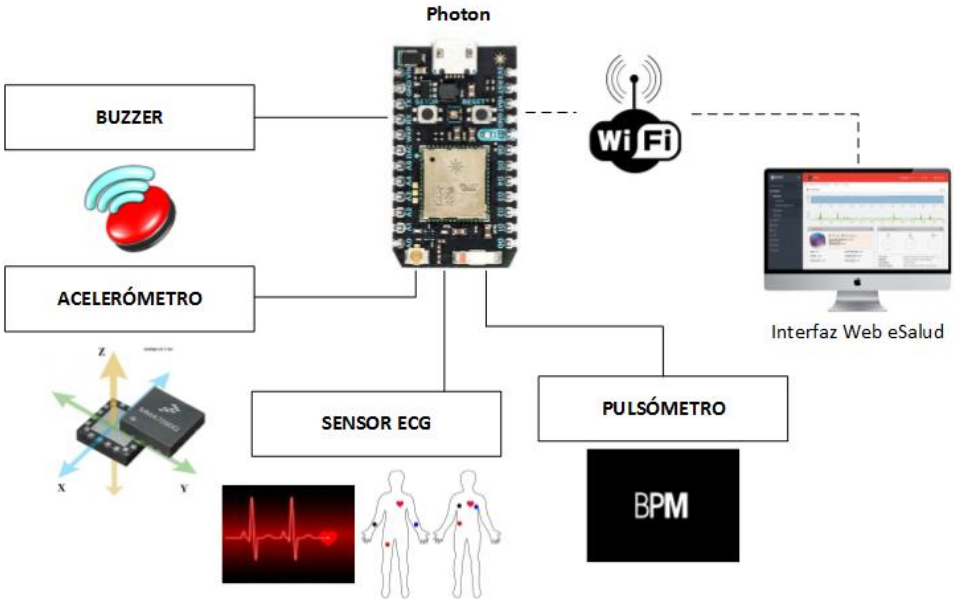


Figura 8. Diagrama general Hardware

Tal y como se comentó en el ***Capítulo 1***, la base para el desarrollo de la plataforma HW/SW en el campo de IoT, es el dispositivo Photon. Sin embargo, para llevar a cabo la realización de cada uno de los objetivos establecidos inicialmente, es necesario incorporar una serie de componentes adicionales a este dispositivo.

A continuación se describirá de manera genérica el funcionamiento de la plataforma hardware propuesta.

Para empezar, la señal ECG es obtenida a través de unos electrodos posicionados sobre el sujeto que captan la actividad eléctrica del corazón y a través de los cuales, mediante los bornes de éstos, se realiza un conexionado por cable hacia el sensor ECG.

Este sensor realiza el proceso de amplificación y filtrado de la señal ECG para poder ser recibida correctamente y enviada posteriormente al dispositivo Photon con el fin de ser procesada. Esta señal será enviada vía inalámbrica a través de la tecnología Wi-Fi desde el dispositivo Photon para poder ser visualizada en un navegador web. Adicionalmente, se detectará el complejo QRS con el objetivo de poder determinar el valor de la frecuencia cardíaca, valor que también será visualizado en el navegador.

El sensor ECG será capaz de detectar además si los electrodos están adheridos a la piel. En caso de detectarlo, y que no haya señal ECG durante un determinado periodo de tiempo, se indicaría un posible paro cardíaco del paciente, generando un evento que se mostraría en el navegador, enviando al mismo tiempo un SMS.

Por otro lado, el pulsómetro estará colocado principalmente en el dedo del paciente, a pesar de que haya más lugares donde ubicarlo, y detectará a través de de su sensor óptico las pulsaciones por minuto (BPM) y el intervalo entre latidos (IBI) . Este sensor se conectará al dispositivo Photon y estos valores serán enviados inalámbricamente para mostrarlos en el navegador.

Al mismo tiempo que se obtienen todos los datos anteriores, el acelerómetro estará continuamente conectado al dispositivo Photon a través de un bus I^2C, y en caso de que se produzca una caída del usuario, se detectará generando un sonido por parte del *buzzer* para avisarle de que se tiene que levantar, y en caso de que no pueda, se enviará un aviso mediante SMS al móvil del familiar correspondiente para que actúe con las medidas adecuadas. Además, este evento de caída será registrado en el navegador.

3.2 Componentes hardware

Una vez definido el diseño general utilizado como base para implementar la plataforma HW desarrollada en este PFC, se mostrará una descripción detallada de las especificaciones del dispositivo Photon y de los distintos dispositivos de extensión seleccionados para llevar a cabo el desarrollo de la aplicación de *eHealth*.

3.2.1 Dispositivo de desarrollo hardware Photon

El kit de desarrollo hardware de IoT Photon de la empresa *Particle* **[8]**, ofrece todo lo necesario para construir un producto conectado a Internet. Se ha escogido específicamente el dispositivo Photon para el desarrollo de este TFG dado su bajo coste, su programabilidad, su utilización de código abierto y por integrar un módulo WiFi habilitado para la creación de proyectos y prototipos conectados a Internet. Este dispositivo constituye la segunda generación de la plataforma dedicada a IoT de *Particle*,

la cual comenzó con el dispositivo *Spark Core* **[9]**. La principal diferencia entre ambos dispositivos se basa en que Photon utiliza un chip Wi-Fi de *Broadcom* en lugar de la empresa *Texas Instruments*, que es el utilizado en el caso de su antecesor. Además, el dispositivo Photon utiliza un procesador más rápido, con más memoria RAM y posee algunos pines con nuevas características. La Figura 9 muestra el dispositivo *Spark Core* a la izquierda y el Photon a la derecha.

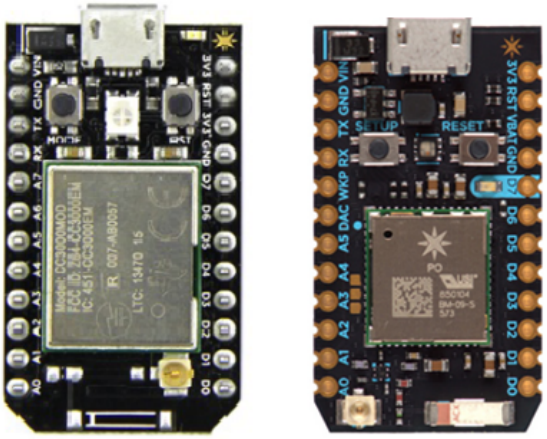


Figura 9. Comparativa Spark Core – Photon

En el caso del dispositivo Photon, *Particle* combina un potente microcontrolador ARM Cortex M3 con un chip Wi-Fi *Broadcom* en un pequeño módulo llamado PØ (P-cero). Además, se añaden un suministro de energía SMPS (*Switched-Mode Power Supply*) de 3.3VDC y componentes de radiofrecuencia y de interfaz de usuario al módulo PØ en una pequeña PCB (*Printed Circuit Board*) denominada Photon **[8]**.

Particle usa *Wiring*, el mismo *framework* que Arduino. Este *framework* es de código abierto y es usado para programar microcontroladores. Hay una gran comunidad donde los expertos, intermedios e iniciantes desarrolladores de todo el mundo comparten sus ideas, conocimiento y su experiencia.

Entre otras características, el dispositivo Photon dispone de 1MB de memoria Flash y 128KB de RAM, además de varios LED (*Light-Emitting Diode*) integrados, 18 entradas/salidas de propósito general, periféricos avanzados y un sistema operativo en tiempo real (FreeRTOS). Este dispositivo cuenta con una gran cantidad de interfaces analógicas, digitales y de comunicación, tal y como se muestra en la

Tabla 2 **[8]**.

Tabla 2. Periféricos del dispositivo Photon.

Tipo de Periférico	Cantidad	Entrada / Salida
Digital	18	Entrada / Salida
Analógico (ADC)	8	Entrada
Analógico (DAC)	2	Salida
SPI	2	Entrada / Salida
I2S	1	Entrada / Salida
I2C	1	Entrada / Salida
CAN	1	Entrada / Salida
USB	1	Entrada / Salida
PWM	9	Salida

La Figura 10 muestra un dispositivo Photon con sus principales componentes resaltados, mientras que en la Figura 11 se muestra su *pinout*, del que posteriormente se procederá a una descripción más detallada.

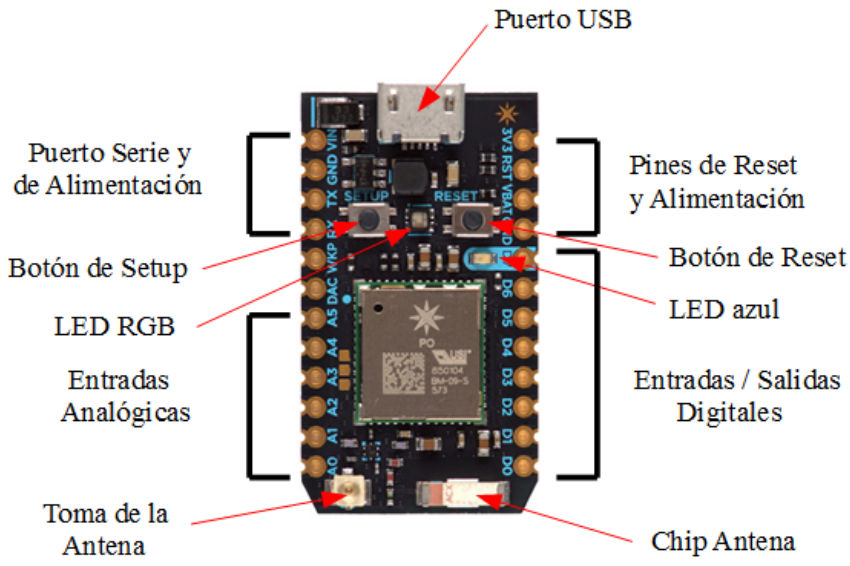


Figura 10. Componentes del dispositivo Photon.

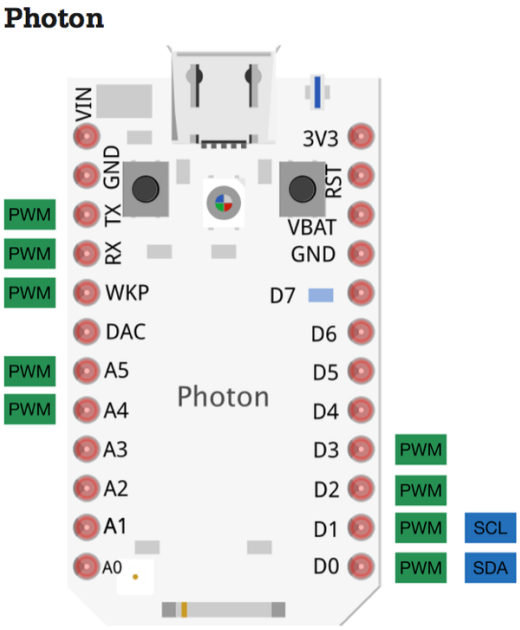


Figura 11. Photon pinout.

Los dos botones disponibles en la plataforma, *Setup* y *Reset*, permiten configurar nuevas credenciales Wi-Fi y reiniciar el dispositivo, y se pueden utilizar en conjunto para provocar el restablecimiento de los valores de fábrica por defecto. Entre ambos botones se encuentra un LED de tipo RGB (*Red, Green, Blue*) que proporciona información acerca del estado del dispositivo. Así, si se encuentra conectado a una red Wi-Fi estará parpadeando muy suavemente en color cian o, si se está cargando un programa en Flash, este LED parpadeará rápidamente en color magenta **[10]**.

En la parte superior se encuentra el puerto micro-USB. Aunque el objetivo principal de éste es proporcionar alimentación al dispositivo Photon, también se puede utilizar para la programación del dispositivo vía USB y para realizar comunicaciones USB con un ordenador. A la derecha del micro-USB están situados los pines de *reset* y alimentación (3V3, RST, VBAT y GND). El dispositivo Photon convierte la energía de entrada proporcionada a través de la alimentación del micro-USB o del pin VIN en un suministro de 3.3 Voltios, ya que toda la lógica del dispositivo funciona a esta tensión. El pin RST se puede utilizar, al igual que el botón *Reset*, para reiniciar el sistema.

El pin VBAT permite que una pequeña batería de reserva se conecte a la del dispositivo Photon para proporcionarle alimentación mientras éste se encuentra en modo *deep sleep*, conservando así el contenido de su memoria, de forma que cuando vuelva a operar en modo normal, pueda continuar en el estado en el que se encontraba previamente.

Los pines *D0* a *D7* son pines de propósito general que pueden actuar como entradas o salidas digitales. Además, los pines *D0* a *D3* también pueden actuar como salidas analógicas utilizando técnicas PWM (*Pulse-Width Modulation*). Tal y como se aprecia en la Figura 11, existe también un LED azul situado junto al pin *D7* que se encuentra conectado directamente a *D7*.

Este dispositivo posee un chip de antena que le permite operar correctamente cuando utiliza la tecnología Wi-Fi. Además, dispone también de una pequeña toma a través de la cual se puede conectar una antena externa. Con esto se conseguiría ampliar el rango de cobertura Wi-Fi, añadiendo una antena más sensible o direccional. Por defecto, el dispositivo Photon intentará elegir la mejor antena, pero también se puede controlar qué antena utilizar mediante *firmware*.

Los pines *A0* a *A5* constituyen entradas analógicas que trabajan con tensiones de entre 0 y 3.3V. Los pines analógicos también se pueden utilizar como entradas o salidas digitales, como los pines *D0* a *D7* y, al igual que los pines digitales, algunos pines analógicos (*A4*, *A5*) también se pueden utilizar como salidas analógicas PWM. A continuación se encuentra el pin del conversor analógico-digital (*Digital Analog Converter*, DAC), el cual se trata de un pin de salida analógica especial, capaz de proporcionar voltajes comprendidos entre 0 y 3.3V. A su lado está el pin WKP, el cual se utiliza para despertar al dispositivo Photon después de que se ha puesto en modo *deep sleep.*

Los pines TX y RX (Transmisión y Recepción, respectivamente) se utilizan para la comunicación serie. Por último, situados por encima de estos pines se encuentran un segundo pin GND y el pin VIN. Tal y como se comentó anteriormente, se puede alimentar al dispositivo Photon suministrando entre 3.6V y 5.5V al pin VIN, como una alternativa al uso del puerto USB.

3.2.2 Sensor ECG AD8232 “ Heart Monitor ”

Dado que una de las partes fundamentales es la obtención de la señal ECG, el dispositivo AD8232 facilitará dicha función. Este sensor se integra en una pequeña PCB distribuida por la empresa *SparkFun* con el propósito de medir la actividad eléctrica del corazón [11] . Las señales eléctricas del corazón son capturadas en la entrada con un conector *jack* de audio que difiere en 3 pads conectados a la piel. El circuito integrado se puede visualizar en la Figura 12 y está diseñado para ser conectado fácilmente con cualquier placa de desarrollo Arduino.

En la descripción del dispositivo se comenta que su intención no es la de diagnosticar o tratar alguna condición, por lo que los resultados obtenidos son básicamente para realizar pruebas sin ninguna intención de proporcionar un perfil de datos médicos a nivel profesional.

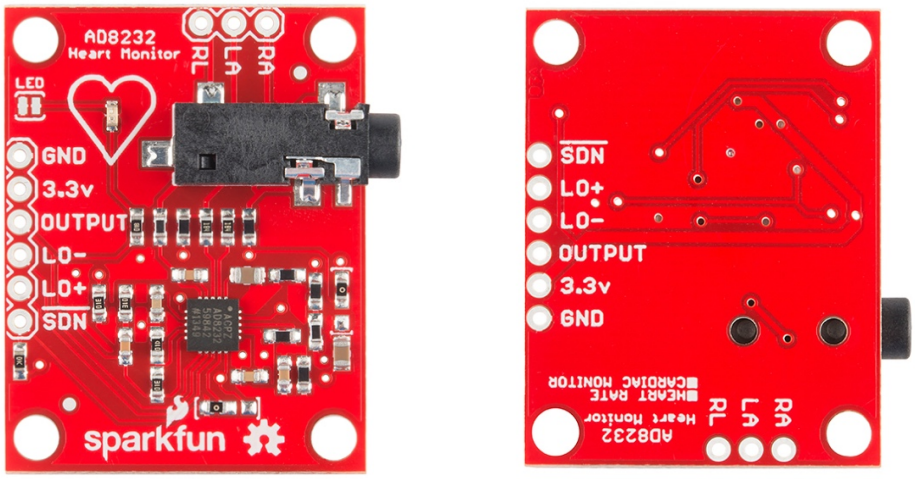


Figura 12. Placa AD8232.

El chip AD8232 fabricado por *Analog Devices* se ha diseñado para extraer, amplificar y filtrar electrocardiogramas sencillos en presencia de condiciones con ruido como las que se generan debido al movimiento. Este diseño permite que un sistema empotrado pueda adquirir la señal de salida analógica fácilmente. La fuente de alimentación del chip opera entre 2.0 y 3.5 V, y las dimensiones de la PCB son de 4mm x 4mm.

De manera más específica, éste implementa un filtro paso alto con 2 polos para eliminar los efectos del movimiento y el potencial de la semicelda [1] del electrodo. Este filtro está acoplado con la arquitectura del amplificador para que ambos puedan obtener una alta ganancia y un filtrado paso alto en una sola etapa. Además un Amplificador Operacional (AO) proporciona al chip la creación de un filtro paso bajo con 3 polos para filtrar ruido añadido. Más aún, se permite que el usuario pueda modificar estos filtros para adaptarlos a sus necesidades.

A continuación, se realizará una descripción general del diagrama de bloques del chip AD8232, proporcionado en el *datasheet* [11], mostrado en la Figura 13. Éste consiste en un amplificador de instrumentación (IA), un amplificador operacional (A1), otro amplificador (A2) y un *buffer* (A3). Además de todo esto, este chip incluye la detección del electrodo en la piel del usuario y un circuito de restauración automática que propicia la señal rápidamente una vez se vuelven a conectar los electrodos.

Así, el chip AD8232 tiene un amplificador de instrumentación que amplifica la señal ECG a la vez que elimina el potencial de la semicelda [1] del electrodo. Si se desea profundizar con más detalle en cada uno de los bloques, el *datasheet* ofrece un apartado explicativo para cada uno de ellos [11].

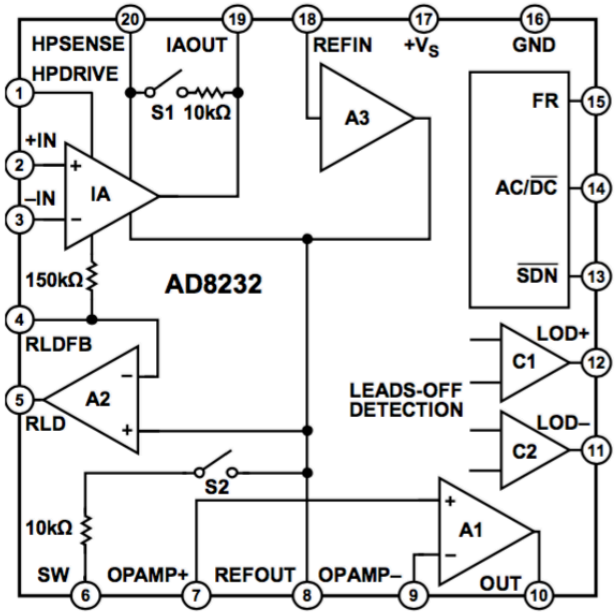


Figura 13. Diagrama de bloques funcional del chip AD8232.

1. Potencial de semicelda: El potencial de una celda es la diferencia entre los potenciales de dos semiceldas o de dos electrodos simples, uno relacionado con la semireacción del electrodo de la derecha (Eder) y el otro, con la semirreacción del electrodo de la izquierda (Eizq) .[11]

Llegados a este punto, a continuación se describirá cómo se colocarán los *pads* o electrodos. Entre más cerca estén posicionados del corazón, mejor será en principio la medida proporcionada y entre más lejos, más ruido de los músculos se podrá apreciar.

Los cables tienen distintos colores para ayudar a identificar un correcto posicionamiento. Así, en la Tabla 3 se indica el conexionado referenciado con los pines del sensor. Más aún, en la Figura 14 se explica la colocación óptima, que proporciona mejores resultados. Todas las pruebas realizadas a lo largo del presente TFG se han llevado a cabo siguiendo el punto 2 de la Figura 14.

Tabla 3. Identificacion del cableado.

Color	Ubicación
Negro	RA(Right Arm)
Azul	LA (Left Arm)
Red	RL (Right Leg)

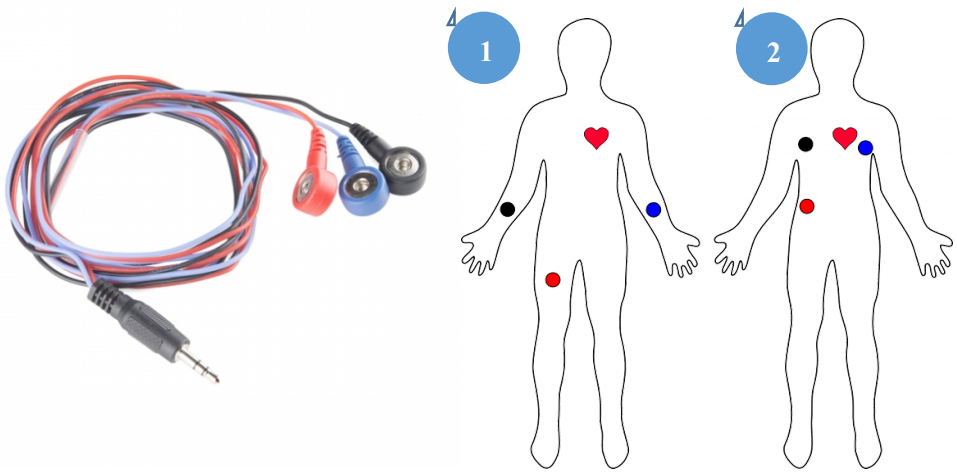


Figura 14. Colocación de los pads.

El dispositivo AD8232 "*Heart Rate Monitor*" de *SparkFun* tiene nueve conexiones / pines del circuito integrado. Estos son GND, VIN, OUTPUT,LO-,LO+. Su descripción se puede ver en la Tabla 4.

Tabla 4. Pinout AD8232.

PIN	Descripción
GND	Tierra
3.3v (VIN)	3.3v Fuente
OUTPUT	Output Signal
LO-	Leads-off Detect -
LO+	Leads-off Detect +
SDN	Shutdown

El pin VDD suministra la alimentación necesaria, que en este caso es de 3.3V para que el chip funcione correctamente. El pin de tierra (GND) se considera el punto cero de todas las tensiones eléctricas que puedan estar presentes en el dispositivo, y en principio debe conectarse a la Tierra física. Tanto el pin LO- como el LO+ son unos comparadores de salida. Cuando estos pines están en modo de detección, se ponen a nivel alto (*HIGH*) cuando el electrodo está desconectado, y a nivel bajo (*LOW*) cuando está conectado. El pin SDN no se usa, pero su función es la de entrar en un modo de bajo consumo. El pin OUTPUT es la salida del Amplificador Operacional (AO) y propicia la señal cardíaca completamente acondicionada.

3.2.3 Chip LSM9DS1

El circuito integrado LSM9DS1 "*HomeBreak*" representado en la Figura 15 es distribuido igualmente por *SparkFun*. Se compone de un acelerómetro de 3 ejes, un giroscopio de 3 ejes y un magnetómetro de 3 ejes. Este sensor es muy versátil y detecta la aceleración lineal, la velocidad angular y el campo magnético, respectivamente, con la intención de proveer un posicionamiento completo y la detección de movimiento.

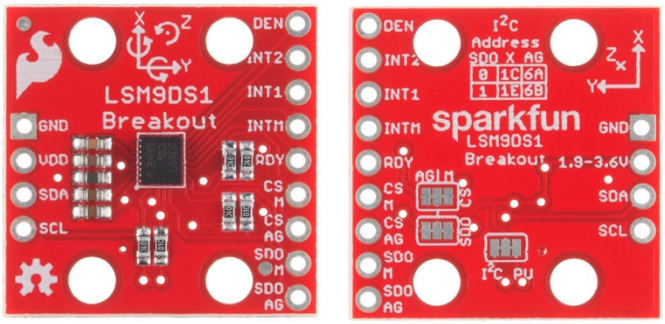


Figura 15. Breakout LSM9DS1 chip de SparkFun.

Así, al estar bien integrados y sincronizados, proveen 9 grados de libertad. Las siguientes escalas son programables y para la aceleración lineal se tienen unos valores de ±2g/±4g/±8/±16 g, para el campo magnético de ±4/±8/±12/±16 gauss, y finalmente una velocidad angular de ±245/±500/±2000 dps. [12]

El chip LSM9DS1 fabricado por *STMicroelectronics* soporta las interfaces I^2C bus serial con el modo estándar y rápido (100KHz y 400KHz), y por otro lado también incluye la interfaz SPI serial. Además tiene interrupciones programables, un modo de ahorro energético y un sensor de temperatura integrado.

En este TFG, únicamente se hará uso del acelerómetro, por lo que sólo se entrará en detalle en este ámbito. Los acelerómetros son dispositivos que miden la aceleración, con el fin de calcular cual es la tasa de cambio de la velocidad de un objeto.

Las unidades de medida son los metros por segundo (m/s^2) o la fuerza G (g). La gravedad de la Tierra es de 9.8 m/s^2, pero puede variar en función de la ubicación. Estos dispositivos suelen usarse para detectar vibraciones en los sistemas o aplicativos de orientación. Si una persona está sentada, el valor que se obtiene por norma general es de 1 g.

Los acelerómetros pueden medir la aceleración en uno, dos, o tres ejes. Los de 3 ejes son los más usuales debido a su bajo coste. Por norma general, estos dispositivos contienen placas capacitivas internas, algunas son fijas mientras que otras están unidas a muelles de tamaño diminuto que se mueven según la fuerza que actúa sobre el sensor. Si las placas se mueven entre ellas, la capacidad cambia, y debido a esta variación, puede calcularse la aceleración. En la Figura 16 y en la Figura 17 se puede ver el proceso comentado anteriormente.

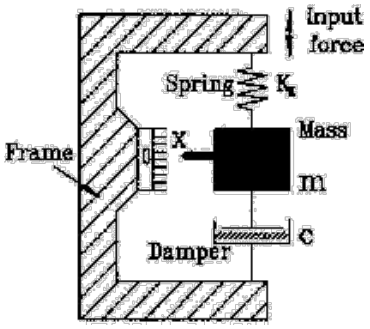


Figura 16. Funcionamiento de un acelerómetro básico.

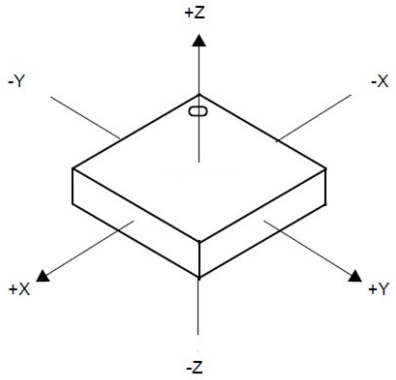


Figura 17. 3 Ejes en un modelo de acelerómetro.

Como bien se comentó con antelación, se soporta tanto SPI como I^2C, siendo SPI la interfaz más sencilla a la hora de la integración del sensor, pero requiere más conexionado, mientras que en I^2C sólo se requieren dos conexiones. En este TFG se hizo uso del bus de comunicación I^2C, que se explicará con mayor detalle en el apartado 3.3. Cabe comentar que el dispositivo LSM9DS1 es un esclavo I^2C y también se emplea para escribir los datos a los registros, cuyo contenido también puede ser leído. En este TFG no se ha hecho uso de los registros de 8 a 16 bits incluidos en el dispositivo.

El chip trabaja con un voltaje de alimentación entre 2.4 y 3.6 V. Éste tiene 13 pines, encontrándose los requisitos mínimos de conexionado en la parte izquierda del circuito integrado, que son el de alimentación y los pines de I^2C. En la Tabla 5 se describen cada uno de ellos.

Tabla 5. Pinout LSM9DS1

Pin Label	Pin Function
GND	Tierra (0V)
VDD	3.3 V Fuente
SDA	SPI: MOSI I^2C: Serial Data
SCL	Serial Clock

El pin VDD suministra la alimentación necesaria, que en este caso es de 3.3V para que el chip funcione correctamente. El pin de tierra (GND) se considera el punto cero de todas las tensiones eléctricas que puedan estar presentes en el. El pin SDA proporciona la funcionalidad de envío y recepción de datos, mientras que el el pin SCL se generan los pulsos de reloj que sincronizan el sistema.

3.2.4 Pulsómetro “ Pulse Sensor ”

El pulsómetro denominado “*Pulse Sensor*” [14] es un diseño *Plug&Play* (PnP) realizado por Joel Murphy & Yury Gitman, que puede ser conectado a un microprocesador sin tener que configurarlo previamente mediante *jumpers* o software proporcionado por el fabricante, ni proporcionar parámetros a sus controladores.

Este dispositivo puede ser posicionado en el dedo o en la oreja, siendo el logo del corazón blanco donde se realiza el contacto con la piel.

Figura 18. Vista frontal y trasera del Pulse Sensor

Se puede apreciar un agujero donde hay un led verde conectado, y un cuadrado justo debajo, que se corresponde con el sensor de luz, siendo el mismo que se usa en los móviles y aparatos que se suelen utilizar para ajustar el brillo según la luz ambiente. Su finalidad es que el LED brille en el dedo u oreja, y el sensor recoja la intensidad de la luz.

El esquemático de la parte trasera del sensor se muestra en la Figura 19. Se implementa un filtro paso bajo y un Amplificador Operacional (AO) para aumentar la amplitud de la onda de pulso y normalizar la señal en torno a un punto de referencia para el desarrollo de monitores ópticos de frecuencia cardíaca.

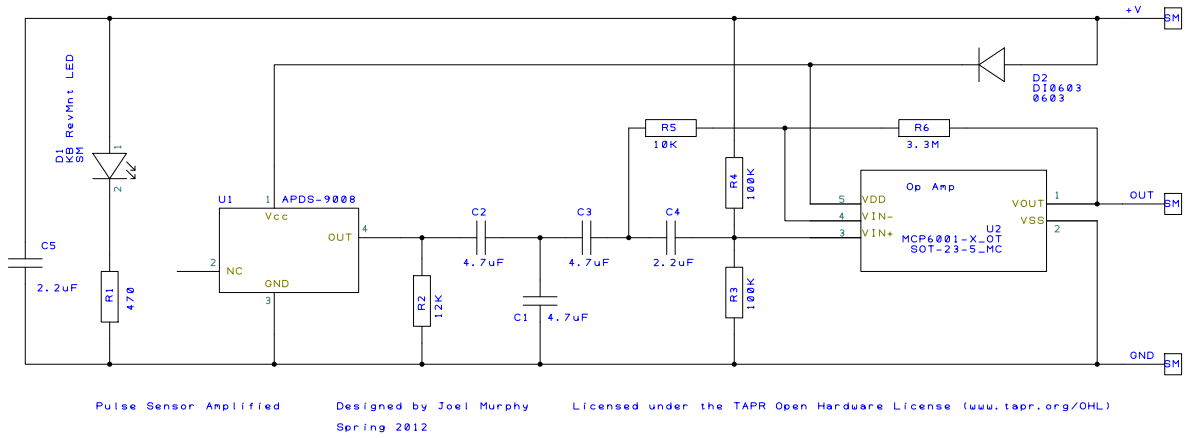


Figura 19. Esquemático del Pulse Sensor.

El conexionado o *pinout* es muy básico, y contiene los elementos descritos en la Tabla 6 :

Tabla 6. Pinout Pulse Sensor

PIN	Descripción
GND	Tierra
VIN	3v a 5V
SIGNAL	Datos, color púrpura

El pin VDD suministra la alimentación necesaria, que en este caso es de 3.3V para que el chip funcione correctamente. El pin de tierra (GND) se considera el punto cero de todas las tensiones eléctricas que puedan estar presentes en el dispositivo. El pin de SIGNAL proporciona datos analógicos de salida con valores comprendidos entre 0 a 4095 (12 bits).

3.2.5 Buzzer/Zumbador

El zumbador "*Grove*" de la Figura 20 es un altavoz piezoeléctrico fabricado por *Seedstudio* [15] basado en un cristal piezoeléctrico que se deforma cuando se aplica una tensión, y que por lo tanto se usa para la reproducción de sonido. Estos altavoces son sencillos, baratos y con un nivel de intensidad alta con muy poca potencia eléctrica.

Su respuesta en frecuencia es óptima en altas frecuencias y no suelen trabajar en el rango de las bajas frecuencias. Así, este elemento puede ser conectado a salidas digitales y emitir un tono cuando la salida está a nivel *HIGH*. Alternativamente, puede conectarse a una salida PWM (*Pulse-Width-Modulation*) para generar varios tonos.

Figura 20. Módulo Buzzer de Seedstudio

Este componente opera entre 4.0 y 8.0 Voltios, y genera un sonido mayor o igual a 85 decibelios. Tiene el pin de GND y VIN para su alimentación y el pin SIG para la conexión con un puerto digital del sistema empotrado, y generar sonido cuando pase a nivel alto.

3.3 Introducción al Bus de comunicaciones I^2C

3.3.1 Definición del Bus I^2C

I^2C es un bus de comunicaciones bidireccional serie que interconecta circuitos integrados. El objetivo de este bus es conectar microcontroladores con dispositivos periféricos de baja velocidad como memorias, convertidores, expansores I/O, drivers LCD, sensores, etc. con un diseño poco complejo y de bajo coste. Es muy popular y usado ampliamente en muchas aplicaciones.

Además, este bus proporciona una comunicación insensible a las variaciones de velocidad de los sistemas anexos, de forma que los dispositivos de baja velocidad no reducen el rendimiento o la tasa binaria ejercida por dispositivos de altas velocidades, pues cada sistema trabaja a su velocidad de procesado sin influir en los demás dispositivos.

Las características más destacadas del bus I^2C son:

- Bus de comunicación síncrono y comunicación controlada por señal de reloj.
- Uso de dos hilos, la de datos *Serial Data Line* (SDA) y la de reloj *Serial Clock Line* (SCL).
- Cada dispositivo tiene una dirección única.
- Distancia normalmente entre 2 y 3 metros.
- Velocidad de transmisión,
    - **Standard** : hasta 100 Kbits/s,
    - **Fast** : hasta 400 Kbits/s,
    - **High-speed**: 3,4Mbits/s y 5Mbits/s.
- Cada dispositivo conectado al bus obtiene un código de dirección seleccionable mediante el software establecido, existiendo permanentemente una relación *Master/ Slave* entre la CPU y los dispositivos conectados.
- El bus permite la conexión de varios *Masters*, ya que incluye un detector de colisiones.
- El protocolo de transferencia de datos y direcciones posibilita diseñar sistemas completamente definidos por software (*Maestro – Esclavo*).
- Los datos y direcciones se transmiten mediante palabras de 8 bits.

3.3.2 Señales del bus I^2C

Este bus se basa en tres señales, como se puede ver en la Figura 21 [16]. En este ejemplo se observa la comunicación entre un *Maestro* (RasPi) y varios esclavos anexos a las líneas SDA y SCL del bus I^2C.

- SDA (*System Data*) por la cual viajan los datos entre los dispositivos.
- SCL (*System Clock*) transitan los pulsos de reloj que sincronizan el sistema.
- GND (Masa) Interconectada entre todos los dispositivos conectados al bus.

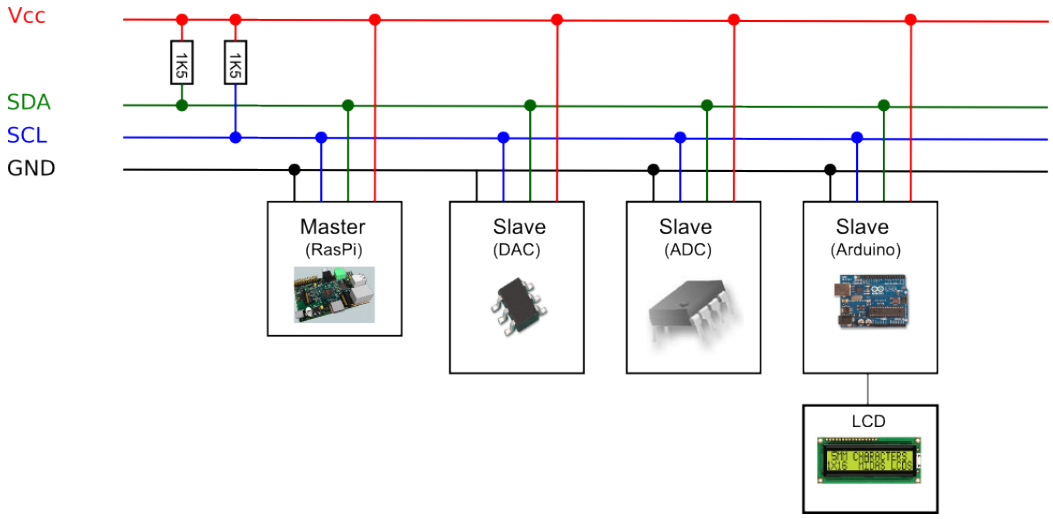


Figura 21. Ejemplo de interconexión I²C.

Las señales típicas de alimentación son +5 V o +3.3 V, pero también soportan otros sistemas con otros voltajes. En el caso en el que el voltaje entre dispositivos sea demasiado grande (a partir de unos 2.5 V por ejemplo), I²C ofrece un adaptador de niveles que normaliza las tensiones y además permite desactivar dispositivos indeseados del sistema.

También se establece una resistencia *pull-up* por cada línea. Estas resistencias pueden tener valores entre 1KΩ y 10 KΩ, dependiendo de la velocidad de comunicación. En el ejemplo de la Figura 21 se ha establecido un valor de 1,5 KΩ.

3.3.3 Protocolo de Comunicaciones del bus I²C

Como bien se sabe, para establecer la comunicación entre dispositivos conectados al bus I²C se debe seguir un protocolo, y en lo que se refiere a la transmisión de datos, los bits como bien se ha comentado, irán por la señal SDA. Por cada bit de información se requiere un pulso de SCL, y los datos solo pueden cambiar cuando SCL está a nivel bajo.

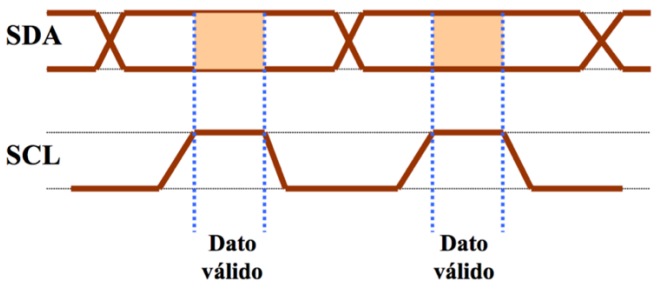


Figura 22. Protocolo establecido para la transmisión de datos

La transmisión es iniciada por el dispositivo *Maestro* (START), a través del flanco de bajada en la señal SDA con SCL a nivel alto, lo que se puede ver mejor en la Figura 23. Además, cuando el bus está inactivo se muestra a nivel alto, tanto la señal SDA, como SCL.

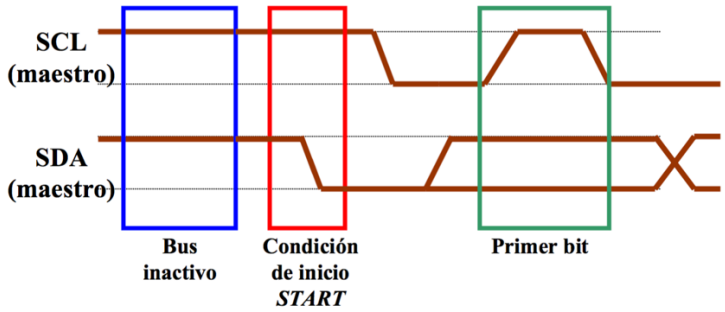


Figura 23. Inicio de la transmisión

Para la finalización de la transmisión (STOP) Figura 24, el dispositivo *Maestro* libera el bus (inactivo), siendo el proceso el inverso del establecido en el inicio de la transmisión con el flanco de SDA, que en este caso sería de subida.

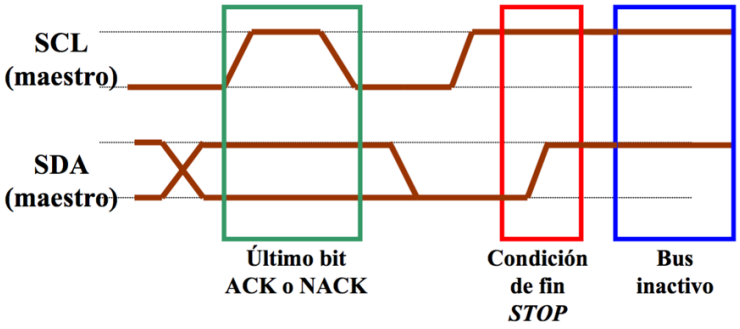


Figura 24. Finalización de la transmisión

Una trama I^2C está determinada por los campos mostrados a continuación, que pueden ser visualizada en la Figura 25 [17]:

1. **Bit de start**: Este es un bit especial, ya que la línea SDA no puede cambiar a menos que SCL este a nivel bajo. Este bit rompe dicha norma y provoca un cambio de nivel alto a nivel bajo cuando SCL está a nivel alto.
2. **Address**: El primer byte enviado empieza con 7 bits de dirección, el cual indica a quien se envía o se solicita el dato.

3. **R/W (Read/Write)**: El siguiente bit indica si se va a realizar una operación de lectura o de escritura.
4. **ACK**: Este bit está presente al final de cada byte que se envía y permite asegurarse de que el byte ha llegado a su destino. De este modo, el que envía deja el bit a nivel alto y si alguien ha recibido el mensaje, fuerza ese bit a nivel bajo. De esta manera confirma que le ha llegado el byte y la transmisión puede continuar.
5. **1º Byte de datos**: Este es el primer byte de datos propiamente dicho, ya que lo anterior no se puede elegir y viene impuesto por el protocolo. Aquí se puede poner el dato que se desee. En caso de comunicación con sensores remotos un uso habitual es poner el número de registro al que se desea escribir o leer. Después del byte de datos, se espera otro ACK del receptor.
6. Se repite el paso 5 tantas veces como sea necesario.
7. **Bit de Stop**. En este caso ocurre lo contrario al bit de *Start*, se pasa de nivel bajo a nivel alto cuando la línea SCL se encuentra a nivel alto. Esto finaliza la transmisión y deja el bus libre para que otro puede empezar a transmitir.

Figura 25. Trama I^2C

En la Figura 26 se mostrará el flujo que ha de seguir cuando el *Maestro* envía datos a sus dispositivos *Esclavos*, y viceversa.

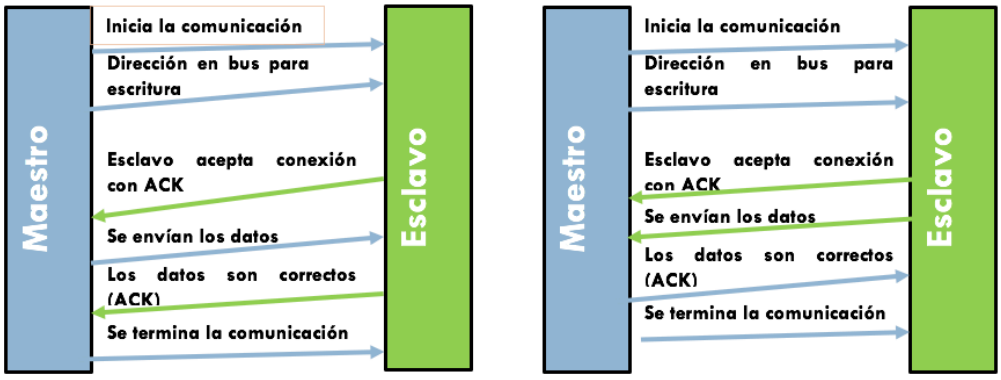


Figura 26. Flujo entre Maestro-Esclavo

3.3.4 Configuraciones posibles

El modo estándar (*Standard-Mode*) del bus I^2C, que es el que se utilizará en el acelerómetro para realizar el presente TFG, usa una tasa de datos de 100 Kbit/s y 7 bit para el direccionamiento. También se ha establecido el modo rápido (*Fast-Mode*) que soporta velocidades de 400 kbit/s.

Además de éste, se ha incrementado notablemente la tasa de datos a 3,4 Mbit/s con el modo de alta velocidad (*High-Speed-Mode*). En este modo, se hace uso de las señales SCLH y SDAH para establecer una distinción, siendo otras diferentes de las establecidas en el modo estándar.

Otra estructura es el *Ultra Fast Mode* de hasta 5 Mhz. Este modo, a diferencia de los anteriores, es unidireccional. En este modo los dispositivos *Esclavos* no transmiten ACKs y solo se permite la presencia de un dispositivo *Maestro*.

Adicionalmente, es posible la configuración con el direccionamiento de 10 bits introduciendo la posibilidad de comunicarse con 1024 dispositivos, que ya es un paso importante, y la escalabilidad que se podría implementar. La mayoría de los controladores I^2C soportan todas las velocidades y direccionamientos comentados anteriormente.

Además de lo comentado, en la configuración de ***Maestro* único** solo se encuentra un *Maestro* con capacidad multi-esclavo. Es la implementación más simple del estándar I^2C y prescinde de la necesidad de un arbitraje de dispositivos *Maestro*. En *Maestro* único, si no existe un dispositivo esclavo que necesite expandir la señal de reloj a una velocidad más baja, el dispositivo maestro puede escribir comandos sin la necesidad de leer el estado de la señal SCL. Esta técnica no es recomendable, pero funciona en muchos casos, como por ejemplo la escritura en EEPROM o dispositivos *Esclavo* que se implementan como máquinas de estado hardware.

La configuración ***Multimaestro*** aparece en entornos en los que varios dispositivos *Maestro* pretenden controlar el bus I^2C. Es importante que en un entorno *multimaestro*, todos los dispositivos *Maestro* soporten esta configuración, ya que si se emplea un

Maestro que únicamente implementa *Maestro* único, éste interrumpirá las comunicaciones de los dispositivos *Maestro multimaestros* en la mayoría de los casos.

Capítulo 4. Desarrollo del Firmware

-

En este capítulo se sentarán las bases del método de trabajo utilizado para desarrollar la parte correspondiente al *firmware* en el presente TFG, definiéndose el funcionamiento del sistema *eHealth* realizado con respecto a los principales objetivos establecidos inicialmente, como son la visualización del electrocardiograma en tiempo real y la detección de caídas.

Para el desarrollo del *firmware* es importante conocer cuáles son aquellas herramientas que posee el dispositivo Photon para llevar a cabo su programación, así como el lenguaje en el que se basa.

4. 1 Herramientas y lenguaje de programación

La programación del dispositivo Photon se basa en *Wiring* [18] al igual que Arduino. *Wiring* se define como un *framework* de código libre para microcontroladores, compuesto por un lenguaje de programación. Este *framework* fue desarrollado por Hernando Barragán en 2003 y la idea es la de poder escribir software capaz de controlar multitud de dispositivos compuestos por microcontroladores de manera sencilla. En la página web oficial se comenta que se propicia lo que se denomina como "*sketching with hardware*", una manera interactiva de que las personas observen por sí mismas cómo van surtiendo los efectos generados por el código.

-

En definitiva, este *framework* ha sido creado por diseñadores y artistas para incentivar a una comunidad con distintos niveles de programación en todo el mundo, con el fin de compartir ideas, conocimiento y experiencias.

El lenguaje de programación *Wiring* está basado en C/C++. Por otra parte, es importante tener en cuenta que la estructura que siguen todos los programas desarrollados para el dispositivo Photon (formato .ino) consta de dos funciones básicas: `setup()` y `loop()`. La función `setup()` es utilizada normalmente para inicializar pines y objetos, mientras que es dentro de la función `loop()` donde se desarrolla el código que define el funcionamiento de la aplicación.

Por lo tanto, la empresa *Particle* ha puesto mucho énfasis en realizar una programación lo más sencilla posible, cuenta con un entorno de desarrollo de fácil uso, así como un lenguaje de programación basado en *Wiring* y una amplia colección de librerías creadas, tanto por ellos, como por la contribución de terceros, que hacen posible sus objetivos.

Como el Photon es un dispositivo que principalmente hace uso de Internet, resulta ideal programarlo a través de este medio, por lo que la mayor parte del tiempo se escribirá el código en el navegador web para posteriormente enviárselo a través de la red, por lo que el dispositivo está en un continuo proceso de comprobación acerca de nuevas actualizaciones del *firmware*. Esto permite actualizar el software del Photon remotamente desde cualquier parte del mundo.

El dispositivo Photon viene pre-programado con un gestor de arranque (o *bootloader*) y una aplicación de usuario denominada "*Tinker*"[19]. Se trata de una librería de *firmware* capaz de ejecutar funciones básicas de GPIO a través de una API (*Application Programming Interface*) y funciona con una aplicación móvil para iOS y Android denominada de la misma manera, "Tinker". Permite conmutar fácilmente pines digitales y realizar lecturas analógicas y digitales sin necesidad de escribir ninguna línea de código.

Por lo tanto, se puede utilizar desde la web a través del IDE (*Integrated Development Environment*) "*Particle Build*", cuyo aspecto se muestra en la Figura 27 para desarrollar código, compilarlo y cargarlo en la memoria *Flash* del dispositivo de forma inalámbrica.

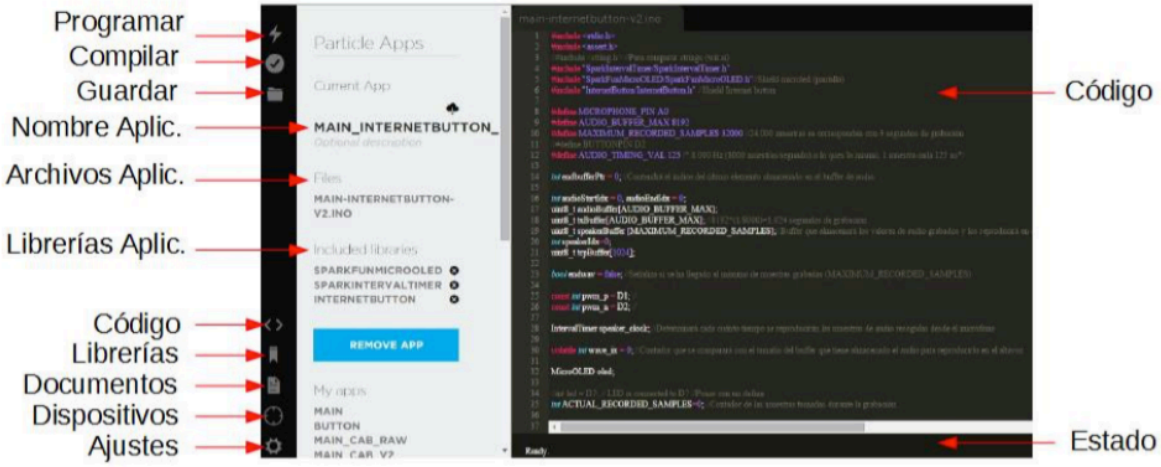


Figura 27. IDE Particle Build

También es posible la programación *offline* a través de una versión de escritorio de código abierto denominada "*Particle Dev*", basada en el proyecto *Atom* de *Github*. Esta versión es un editor de texto moderno, y un "*hackable* IDE", lo que permite personalizar el entorno a través de distintos paquetes descargables desde un buscador dentro del programa con el fin de facilitar las tareas de programación, entre otras muchas funciones. Este entorno está disponible tanto para Mac, como para Windows y Linux. [20]

Esta versión adaptada por *Particle* de *Atom* funciona casi de una manera muy similar al Web IDE, si bien, esta herramienta no es completamente *offline*, debido a que usa el servicio web para compilar la aplicación y realizar la programación del dispositivo Photon, aunque en un futuro será completamente independiente de una conexión a Internet.

Cabe destacar la importancia de *Particle Cloud,* representado en la Figura 28, el cual permite conectar todos los productos de IoT a Internet y ofrece una pasarela/puerta de enlace segura para interactuar con los dispositivos de IoT de la empresa *Particle*, a través de APIs modernas y servicios web entorno al producto. La nube también incluye actualizaciones del *firmware* de manera inalámbrica, lo cual representa un mecanismo seguro para actualizar el *firmware* que se ejecuta en el dispositivo IoT, con el fin de

mejorar la funcionalidad, corregir errores o aportar nuevos protocolos, entre otras funciones. [21]

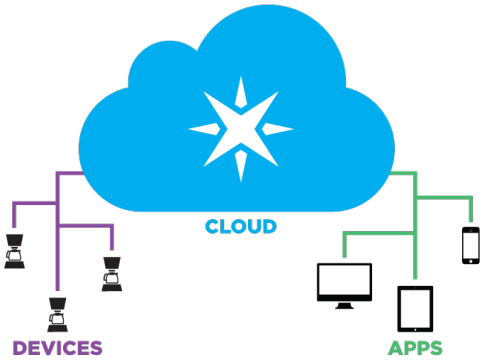


Figura 28. Diagrama de Particle Cloud.

En el caso concreto del desarrollo del *firmware* del presente TFG, se ha usado el método *Particle* Dev, cuya interfaz de usuario se muestra en la Figura 29.

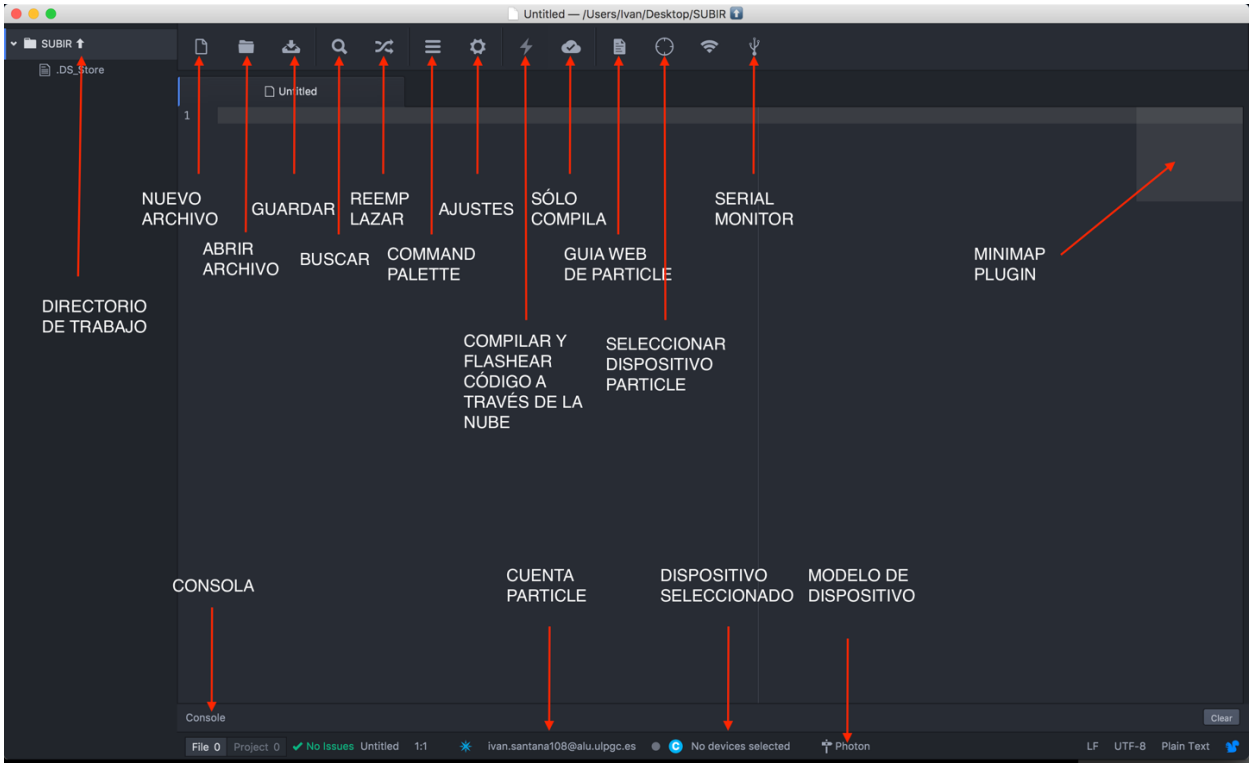


Figura 29. IDE Particle Dev

Este entorno ofrece opciones más avanzadas que *Particle Build*, indicándose a continuación una descripción de las opciones disponibles más destacadas.

- Directorio de Trabajo: Apartado del programa donde se ubican los archivos con los que se va a trabajar. Ofrece una interacción con la interfaz para la transferencia de textos, datos, archivos u objetos de una ubicación a otra.

- Command Palette: Este apartado dentro de la interfaz ofrece la opción de buscar y ejecutar los comandos disponibles para interactuar de manera avanzada con el programa.

- Ajustes: En este apartado se establece la configuración del programa de manera que se ajuste a las necesidades del usuario, pudiéndose personalizar los temas, instalar paquetes, etc.

- Compilar y programar: Se utiliza para compilar el código desarrollado a través del servicio en la nube de *Particle*, en busca de posibles errores y para generar el archivo *firmware* que se enviará al dispositivo Photon.

- Compilar: Apartado que usa el servicio en la nube de *Particle* para compilar el código y generar el archivo *firmware,* sin enviarlo al dispositivo.

- Guía Web de Particle: Se obtiene el conjunto de libreras y documentación suministrada por *Particle*, proporcionando toda la información necesaria para el uso de sus productos, tanto a nivel de hardware como *firmware*.

- Seleccionar Dispositivo: En esta pestaña se detecta y selecciona el dispositivo, lo cual proporciona también el identificador del dispositivo que se esté utilizando.

- Serial monitor: Es una aplicación terminal de puerto serie para visualizar el intercambio de datos con el hardware conectado a través del puerto serie seleccionado.

- Parte central: Se encuentra localizado el panel en el que se escribirá todo el código necesario para la aplicación que se desee implementar.

Además de estas funcionalidades, la pestaña *Particle* del programa ofrece otras opciones. En la Figura 30 se puede apreciar cómo es posible visualizar las variables y funciones de la nube de *Particle* lo que constituye una herramienta muy útil a la hora de realizar pruebas. Adicionalmente, y de manera personalizable, incluye una opción para renombrar el dispositivo de *Particle*.

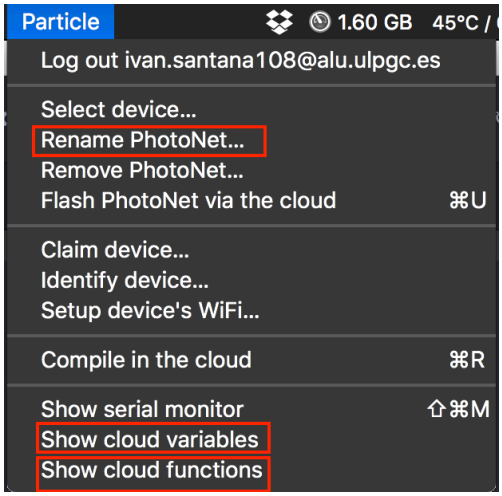


Figura 30. Pestaña Particle

A continuación, los siguientes subapartados explicarán en detalle el *firmware* correspondiente a las distintas funcionalidades de la plataforma *eHealth* desarrollada en el presente TFG. Además, estos están ordenados de manera cronológica, del mismo modo en el que se fueron desarrollando hasta su versión final.

4. 2 Obtención datos ECG y representación visual vía puerto serie

Como se indicó en el **Capítulo 3,** para la obtención de la señal ECG se hace uso del sensor AD8232 "*Heart Rate Monitor*". Este subapartado comprende la obtención de dicha señal por medio del *firmware* ejecutado en el dispositivo Photon en tiempo real a través del puerto serie.

Así, en el código correspondiente al *firmware* relativo a este sensor se tendrán que definir los pines de entrada utilizados para su interconexión con el fin de establecer una

comunicación analógica de la señal de salida del sensor AD8232. De implementar esta funcionalidad se encargan las sentencias descritas a continuación :

- `pinMode(pin,mode)`: Configura el pin especificado para establecerlo como entrada (*Input* con o sin *pull-up* o resistencia *pull-down*) o como salida (*Output*). Tiene dos argumentos y no retorna nada :
    - `pin`: El pin que se quiere establecer como entrada/salida.
    - `mode`: INPUT, INPUT_PULLUP,INPUT_PULLDOWN o OUTPUT.

Para esta primera prueba, los datos son enviados vía puerto serie y es por ello que hay que insertar la función en el código que permita establecer la comunicación entre el dispositivo Photon y el ordenador, en este caso. La siguientes funciones permiten inicializar la funcionalidad de comunicación serie y escribir datos :

- `Serial.begin(speed)`**:** Establece la tasa de datos en bits por segundos (baudios) para la transmisión de datos vía serie. Tiene un argumento y no devuelve nada.
    - `Speed`**:** Parámetro que especifica la tasa en baudios. Se aceptan los siguientes valores, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, o 115200. En este caso se usará siempre el valor de 9600 baudios.

- `Serial.println(val)`**:** Imprime los datos al puerto serie a modo texto con el estándar ASCII, además añade un retorno de carro (\r) para realizar un salto de línea.
    - `val`: El valor a imprimir de cualquier tipo de dato.

Puesto que se tienen que leer, tanto los valores que proporciona la salida del circuito integrado AD8232, como los pines LO+ y LO- para detectar el conexionado de los pads, se hará uso de los siguientes métodos :

- `digitalRead(pin)`**:** Lee el valor de un pin específico. Tiene un argumento y devuelve tanto valores lógicos altos (*HIGH*) como bajos (*LOW*).

- `pin`: El pin del que se desea leer el valor digital.

- `AnalogRead(pin)`: Lee el valor analógico de un pin específico con 12 bits de resolución, lo que significa que los valores de voltaje comprendidos entre 0V a 3.3V serán mapeados en valores enteros de 0 a 4096, obteniéndose la siguiente expresión : $Resolucion = \frac{3.3}{4095} = 0.0008\ voltios/valor$
  - `pin`: El pin del que se desea leer el valor analógico.

La Figura 31 muestra el código que establece todo lo comentado anteriormente. La función `setup()` inicializa todos los componentes y en la función `loop()` se comprobará si el usuario tiene conectados los *pads*. En caso de estarlo, se transmiten los valores muestreados vía puerto serie, y en caso contrario se imprime un valor constante cero.

```
int pinL0_plus = D2;
int pinL0_minus = D3;
int analogPin = A3;
String ECG;

void setup() {
      // inicializa la comunicación serial y los pines:
      Serial.begin(9600);
      pinMode(analogPin, INPUT);
      pinMode(pinL0_plus, INPUT);
      pinMode(pinL0_minus, INPUT);

}

void loop() {
// Comprueba que los pads están conectados
if((digitalRead(pinL0_plus) == 1)&&(digitalRead(pinL0_minus) == 1)) {
            Serial.println('0');
}
      else {
            Serial.println(analogRead(analogPin));

      }
      //Espera 1ms para no saturar los datos
      delay(1);
}
```

Figura 31. Código Arduino para transmitir datos AD8232 vía puerto Serie.

A la hora de visualizar los datos vía puerto serie, o lo que es la señal ECG, la empresa *SparkFun* ha publicado un repositorio en *Github* con un archivo en lenguaje *Processing*.

A modo introductorio, *Processing* [22] es un un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital.

El archivo `Heart_Rate_Display.pde` se puede descargar en el enlace [23] y trabaja con librerías/API asociadas a la comunicación a través de puerto serie. Este programa, una vez reciba un valor 0, que es cuando no están conectados los pads a la piel del usuario, generará un valor constante de color azul, en caso contrario generará la señal ECG registrada en color rojo, como puede visualizarse en la Figura 32.

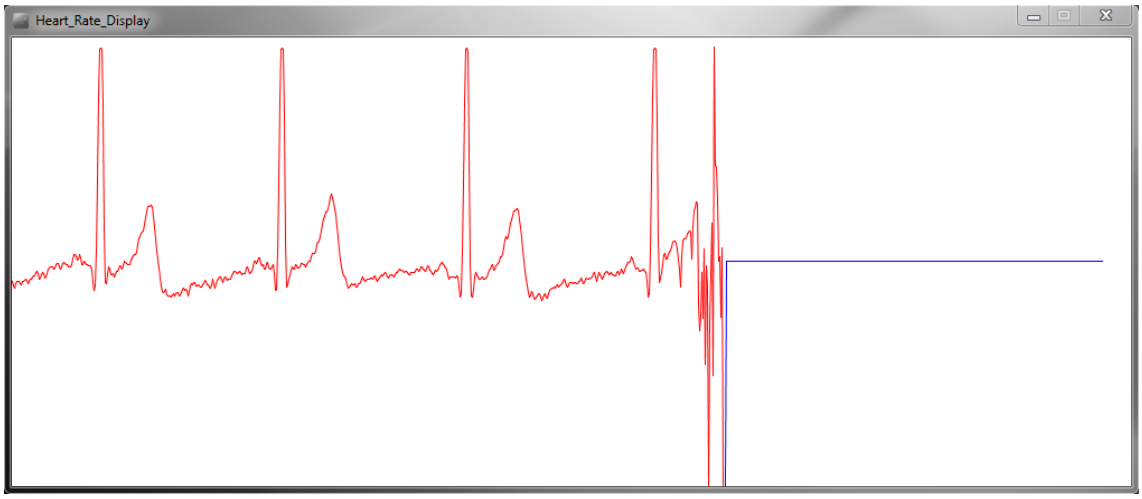


Figura 32.Heart Rate Display a través de Processing.

Fuente: https://learn.sparkfun.com/tutorials/ad8232-heart-rate-monitor-hookup-guide?_ga=1.179030311.2025795697.1452716986#

La intención final es la de representar esta misma señal a través de Internet, en un navegador web, por lo que no se modificará ni hará ningún cambio al archivo *Processing* desarrollado por *Sparkfun*. En definitiva, este apartado inicial del proyecto ha resultado útil para verificar que el sensor AD8232 funciona correctamente.

En la Figura 33 se puede observar una prueba realizada con la misma filosofía a partir del código desarrollado inicialmente, de donde resulta que el sistema desempeña su función adecuadamente.

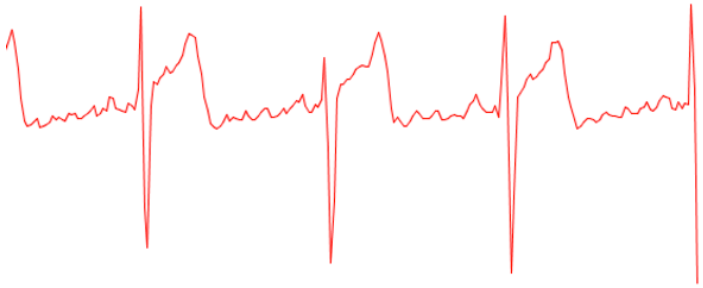


Figura 33. Prueba realizada a través de Processing.

4. 3 Obtención de los datos BPM e IBI a través del dispositivo "Pulse Sensor"

Se ha establecido que el pulsómetro "*Pulse Sensor*" facilite de manera precisa, tanto los valores de la frecuencia cardiaca o pulsaciones por minuto (BPM), como el intervalo entre latidos (IBI) representado en milisegundos. El pulsómetro es básicamente una fotopletismografía (PPG del inglés *photoplethysmogram*), obtenida de manera óptica a través de una medición de la absorción de luz de la piel y generando una gráfica de valores analógicos variante con el tiempo. En este TFG no se representará dicha gráfica, pero se abre la posibilidad de implementarse en trabajos futuros.

Cuando el corazón late, como se vió en el **Capítulo 2,** propulsa sangre por todo el cuerpo, viajando por las arterias, vasos sanguíneos y capilares y consecuentemente generando una onda de pulso. El dispositivo "*Pulse Sensor*" captará esta onda una vez esté posicionado en el cuerpo (por lo general en el dedo).

El *firmware* asociado a este dispositivo se ha desarrollado para medir el IBI cuando la señal de la onda de pulso supera el 50 % de la amplitud total de la onda de pulso, y las pulsaciones por minuto se determinan con una media de los 10 valores IBI previos al pulso.

Primeramente es necesario tener una frecuencia de muestreo constante para obtener una medida correcta del tiempo entre cada pulso. Esta tarea se realiza a través de un *Timer* de interrupción hardware con una frecuencia de muestreo de 500 HZ y por lo tanto, 2 milisegundos de resolución entre pulsos, aunque en principio esta tasa puede establecerse como se desee. Esta interrupción usa la API de Paul Kourany denominada *SparkIntervalTimer,* a la que se puede acceder desde el repositorio de *GitHub* [24].

SparkIntervalTimer usa interrupciones para realizar llamadas a funciones con un intervalo temporal fijo y de manera muy precisa. Pueden ejecutarse hasta 5 *Timers* de forma simultánea en el Photon (*TMR3, TMR4, TMR5, TMR6 y TMR7*). Dichos *Timers* se mapean con los pines establecidos en la Tabla 7.

Tabla 7. Asignación de pines con Timers en Photon.

PIN	TMR3	TMR4	TMR5	TMR6	TMR7
D0		■		NO TIENEN ASIGNACIÓN	
D1		■			
D2	■				
D3	■				
A4	■				
A5	■				
WKP			■		

Para usar la librería *SparkIntervalTimer*, se debe crear inicialmente una variable global. A continuación, se debe inicializar con la siguiente función:

- `begin(function, time, timebase)`: Inicializa el objeto *Interval Timer*. Tiene 3 parámetros y devuelve verdadero en caso de que la inicialización tenga éxito, en caso contrario devuelve falso (por ejemplo, cuando otros objetos de *IntervalTimer* usan el mismo recurso).
    - `function`: Función a la que se hace la llamada.
    - `time` : valor del tiempo en microsegundos.
    - `timebase`: El intervalo está indicado en microsegundos o la mitad de milisegundos. Los valores posibles son *uSec* y *hmSec*, respectivamente.

Por lo tanto, se leen cada 2ms los valores obtenidos por el dispositivo "*Pulse Sensor*" con la finalidad de determinar las pulsaciones por minuto. Lo primero que se tiene que hacer es recoger la lectura analógica, implementándose posteriormente la variable `SampleCounter` con la finalidad de recoger el valor del tiempo y la variable `N` sirve para ayudar a eliminar el ruido en pasos posteriores. En la Figura 34 puede verse el código asociado a este proceso del archivo `PulseSensor_Spark.cpp` .

```
void interruptSetup(void){
  // Timer que genera una interrupción cada 2ms
  pulseTimer.begin(pulseISR, 2000, uSec);  // (2000 * 1us period)

}

// TIMER3 INTERRUPT SERVICE ROUTINE.
 void pulseISR(void) {
  noInterrupts();
  Signal = analogRead(pulsePin); // lee el pin del pulsómetro
  sampleCounter += 2;  // recoge el valor del tiempo en ms.
  int N = sampleCounter - lastBeatTime; // monitoriza el tiempo
desde el último latido para evitar ruido
…
```

Figura 34. Fichero PulseSensor_Spark.cpp. Lectura datos del sensor.

Seguidamente, en el código de la Figura 35, se muestra el valor más alto y mínimo de la onda PPG, con la finalidad de obtener una medida precisa de la amplitud. Las variables P y T establecen el valor de pico que se inicializa a 2048 (mitad de la resolución de 12 bits del dispositivo Photon), aunque éste irá cambiando a medida que se ejecuta el código del fichero `PulseSensor_Spark.cpp`.

```
// encuentra el pico de la onda de pulso
if(Signal < thresh && N > (IBI/5)*3){ // evitar ruido esperando 3/5
partes del ultimo IBI

if (Signal < T){ // T es el pico
      T = Signal; //Recoge el valor minimo de la onda de pulso

}
  }

  if(Signal > thresh && Signal > P){ // thresh condición que ayuda
a eliminar ruido
    P = Signal; // P es el pico máximo de la onda de pulso.
  }
…
```

Figura 35. Fichero PulseSensor_Spark.cpp. Valor máximo y mínimo onda PPG.

A continuación se comprobará si se ha generado un pulso, para ello, tendrá que haber transcurrido un tiempo mínimo. En la Figura 36 del fichero `PulseSensor_Spark.cpp`, la condición establecida con la variable `N` establece el límite de 240 bpm (250ms), una vez se cumpla la condición y las 3/5 partes del IBI, se obtendrá un pulso. Posteriormente se calculará el tiempo transcurrido desde el ultimo pulso para obtener el IBI.

```
if (N > 250){ // establece limite de pulsaciones
    if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3)
){
      Pulse = true; // se establece un flag que indica el pulso
      digitalWrite(blinkPin,HIGH); // enciende el led D7 del Photon
      IBI = sampleCounter - lastBeatTime; // mide el intervalo
entre pulsos (ms)
      lastBeatTime = sampleCounter; // recoge el valor del tiempo
para el próximo pulso

…
```

Figura 36. Fichero PulseSensor_Spark.cpp. Comprobación de un pulso.

En la siguiente porción de código, mostrado en la Figura 37 del fichero `PulseSensor_Spark.cpp`, se calculan las pulsaciones por minuto (BPM). Primeramente, se inicializa una variable, denominada `runningTotal`, para recopilar los IBIs. El valor IBI más lejano, (11 pulsos anteriores) se posiciona en 0 en el contenido de `rate[]`, y el valor más cercano se establece en la posición 9. A continuación, se calcula una media del *array,* teniendo en cuenta los 10 valores de IBI, y se procede a calcular el valor BPM. Seguidamente se establece un *flag* `QS` para indicar que se ha encontrado un pulso.

```
uint16_t runningTotal = 0; // inicializa la variable

      for(int i=0; i<=8; i++){ // desplaza los datos del array
        rate[i] = rate[i+1]; // y coloca el valor IBI más lejano
        runningTotal += rate[i]; // añade los 9 valores IBI más
lejanos
      }

rate[9] = IBI; // añade el valor IBI mas cercano al array
      runningTotal += rate[9]; // y lo asigna a runningTotal
      runningTotal /= 10; // media de los 10 valores IBI
      BPM = 60000/runningTotal; // pulsos por minuto, de ahí el
valor
      QS = true; // se establece flag
    }

…
```

Figura 37. Fichero PulseSensor_Spark.cpp. Calculo de bpm.

La siguiente porción de código, mostrada en la Figura 38 sirve para identificar cuándo no hay un pulso. El *flag* `Pulse` se declaró verdadero una vez se supere el *threshold* superior, y en este caso, cuando sobrepase el inferior, se declarará falso. Esta referencia indica si se ha generado un pulso. Posteriormente se realiza una medida de la onda de pulso y se actualiza el valor *threshold,* modificando las variables P y T.

Para finalizar, en el código de la Figura 39 se muestra que si en 2.5 segundos no se detecta ningún pulso, se inicializan todos los valores y el valor BPM e IBI se asigna a 0. Así se concluye la parte correspondiente a la obtención de los valores BPM e IBI en el *firmware* desarrollado en el archivo `PulseSensor_Spark.cpp`.

```
if (Signal < thresh && Pulse == true){   // cuando los valores
superan el threshold inferior

digitalWrite(blinkPin,LOW); // es apaga el blink D7 del Photon
    Pulse = false; // se restablece el valor del Pulso

  amp = P - T; // se obtiene la amplitud de la onda de pulso
    thresh = amp/2 + T; // se establece el threshold a la mitad de
la amplitud
    P = thresh; // Se restablece los valores P y T para la próxima
medida
    T = thresh;
  }

…
```

Figura 38. Fichero PulseSensor_Spark.cpp. Identificar cuando no hay pulso.

```
if (N > 2500) { // si no surge un pulso en 2.5 segundos
             thresh = 2048; // thresh default
             P = 2048; // P default
             T = 2048; // T default
             lastBeatTime = sampleCounter; //lastBeatTime up to
date
             firstBeat = true; // evitar ruido
             secondBeat = false; // cuando obtenemos el pulso
             BPM=0; // se inicializa los valores BPM e IBI
             IBI=0;
      }

…
```

Figura 39. Fichero PulseSensor_Spark.cpp. Tiempo para detección de pulso.

A modo conclusión, se muestra una tabla resumen con los valores útiles generados en el fichero `PulseSensor_Spark.cpp` .

Tabla 8. Resumen Variables fichero PulseSensor_Spark.cpp

Variable	**Descripción**
Signal (*int*)	Valor analógico del dispositivo *Pulse Sensor*
IBI (*int*)	Intervalo entre pulsos (ms)
BPM (*int*)	Pulsos por minuto
QS (*boolean*)	Detección de un pulso

A continuación se describirá el código en lenguaje *Wiring* que se ejecutará en el dispositivo Photon. Así, la función `loop()` mostrada en la Figura 40 se ejecuta cada 20 ms y contiene el programa principal, siendo su función básica la de encender un LED una vez se detecte un pulso. La función `setup()` se encargará de inicializar los pines y la comunicación serie.

```
void setup(){
pinMode(blinkPin,OUTPUT); // pin que se encedera con el pulso!
Serial.begin(9600); // Comunicación puerto serie
interruptSetup// llamada a la función que lee la señal del
pulsómetro cada 2ms
}

void loop(){
if (QS == true){     //  Se detecta el pulso
  // BPM and IBI son variables externas
digitalWrite(blinkPin,HIGH);     // enciende el LED.
QS = false;                      // inicializa el flag
}

else {

  digitalWrite(blinkPin,LOW); // no se detecta el pulso, se apaga
el led
}

delay(20);

}

…
```

Figura 40. Fichero PulseSensorAmped_Arduino_1dot4.ino. Código principal en la función loop().

4. 4 Algoritmo para la detección del complejo QRS

En este apartado se muestra el *firmware* correspondiente al algoritmo que detecta en tiempo real el complejo QRS. Este algoritmo se basa según lo descrito en el artículo redactado por HC Chen y SW Chen [25]. El siguiente diagrama de bloques muestra cómo funciona dicho algoritmo.

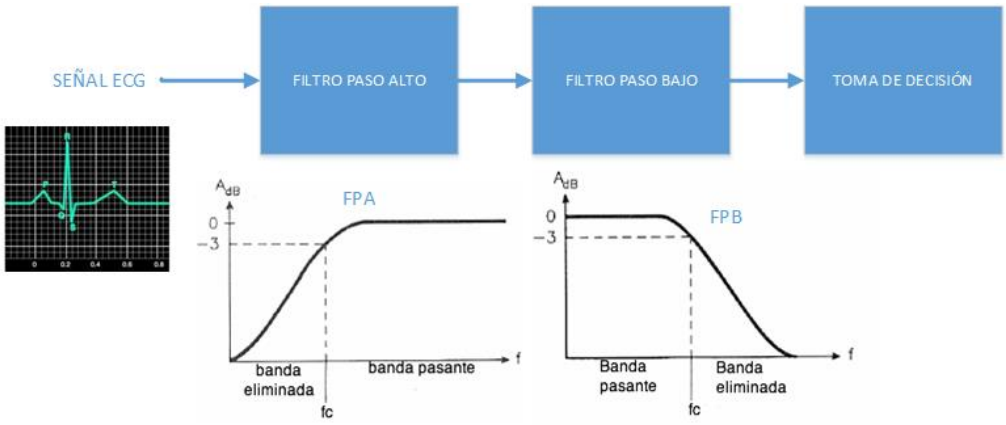


Figura 41. Diagrama de bloques del algoritmo de detección QRS.

Como se puede apreciar, se requieren filtros digitales para el preprocesamiento de la señal ECG, lo que conlleva una reducción del coste computacional. En concreto, se compone de un Filtro Paso Alto (FPA), un Filtro Paso Bajo (FPB) y una etapa de toma de decisión. El FPA se encarga de acentuar el complejo QRS, además de eliminar las ondas P o T debido al interés principal en el complejo QRS. Por una parte, el FPB se encarga de obtener una envolvente de la onda QRS. Finalmente, un *threshold* adaptativo se aplica a esta envolvente para decidir si se ha detectado el complejo QRS.

El desarrollo del *firmware* se ha realizado tomando como referencia el código diseñado por Blake Milner para una plataforma Arduino [26]. A continuación, se procederá a explicar las partes del código más destacadas del *firmware* desarrollado.

En primer lugar, la función `detect()` contendrá todo el código necesario asociado a la funcionalidad del algoritmo. Tiene un parámetro de entrada que serán los datos obtenidos del sensor AD8232, y devuelve verdadero o falso en caso de detectar, o no, el complejo QRS, respectivamente. A continuación, la función `loop()` hallará el

valor de las pulsaciones por minuto (BPM) que viene siendo la principal funcionalidad de este código.

El primer elemento será el Filtro Paso Alto con fase lineal, que se incluye en la Figura 42 del fichero `QRS.ino`. Éste se compone de un punto `M` (MAF, *Moving Average Filter*) y un retardo de grupo de `(M+1)/2` muestras. Es importante elegir correctamente el valor `M`, ya que determina la respuesta en frecuencia del filtro. En relación al espectro de potencia de la señal ECG, se obtiene que la mayor parte de la energía del complejo QRS se concentra entre 5-15Hz, mientras que las onda P o T suelen ser inferiores a 5Hz. Es por ello que se elige un valor 5 para `M`, debido a tener la mejor respuesta en frecuencia para la señal de estudio, con la finalidad de atenuar las ondas P y T y amplificar el complejo QRS.

```
/* Filtro Paso Alto */

if(number_iter < M) {
      // se rellena el buffer con suficientes puntos para el filtro
paso alto.
      hp_sum += ecg_buff[ecg_buff_RD_idx];
      hp_buff[hp_buff_WR_idx] = 0;
}
else{
      hp_sum += ecg_buff[ecg_buff_RD_idx];

      tmp = ecg_buff_RD_idx - M;
      if(tmp < 0) tmp += M + 1;

      hp_sum -= ecg_buff[tmp];

      float y1 = 0;
      float y2 = 0;

      tmp = (ecg_buff_RD_idx - ((M+1)/2));
      if(tmp < 0) tmp += M + 1;

      y2 = ecg_buff[tmp];
      y1 = HP_CONSTANT * hp_sum;
      hp_buff[hp_buff_WR_idx] = y2 - y1;
}

// lectura del buffer ECG, se incrementa la posicion
ecg_buff_RD_idx++;
ecg_buff_RD_idx %= (M+1);

// Escritura del buffer ECG, se incrementa la posicion
hp_buff_WR_idx++;
hp_buff_WR_idx %= (N+1);
…
```

Figura 42. Fichero QRS.ino. Implementación filtro paso alto lineal.

El Filtro Paso Bajo no lineal puede identificarse como un detector de envolvente, y hay que tener en cuenta que el tamaño de la ventana es dinámico. En este caso, el tamaño se ha determinado obteniendo una frecuencia de muestreo (`winSize`) de 250 Hz y un intervalo de la suma de la ventana (`N`) de 30, por lo que se corresponde con 120 ms en tiempo real. En la Figura 43 puede verse la implementación software del filtro del fichero `QRS.ino`.

```
/* Filtro Paso Bajo */
// desplaza en una nueva lectura del filtro paso alto
lp_sum += hp_buff[hp_buff_RD_idx] * hp_buff[hp_buff_RD_idx];

if(number_iter < N) {
      // se rellena el buffer con suficientes puntos para el filtro
paso bajo.
      next_eval_pt = 0;
}
else{
      // desplaza/elimina el punto mas antiguo
      tmp = hp_buff_RD_idx - N;
      if(tmp < 0) tmp += (N+1);
      lp_sum -= hp_buff[tmp] * hp_buff[tmp];
      next_eval_pt = lp_sum;
}

// lectura del buffer HP, se incrementa la posición
hp_buff_RD_idx++;
hp_buff_RD_idx %= (N+1);

…
```

Figura 43. Fichero QRS.ino. Implementación filtro paso bajo.

El último bloque que compone el desarrollo del algoritmo, es el que toma la decisión en cuanto a establecer si se ha detectado o no el complejo QRS, para ello se aplica un *threshold* adaptativo y se formula según la siguiente expresión :

$$Threshold = \alpha * \gamma * Pico + (1 - \alpha) * Threshold,$$

dónde *Pico* es el valor máximo actualizado según la forma de onda detectada, y α es un factor positivo que varía entre $0 \leq \alpha \leq 1$. El valor γ es un factor de peso que puede ser tanto 0.15 como 0.2. Se detectará el complejo QRS cuando la señal ECG supere el valor de *Threshold,* y este valor de umbral irá variando constantemente al detectar un nuevo complejo QRS. En la Figura 44 se muestra la implementación de este último bloque en el fichero `QRS.ino`.

```
/* Detección dinámica del threshold del pulso  */
// establece el threshold inicial
if(number_iter < winSize) {
	if(next_eval_pt > treshold) {
		treshold = next_eval_pt;
	}
// solo incrementa el valor number_iter si es menor que winSize, en
caso de ser mayor, el contador no sirve de ningún propósito
	number_iter++;
}
if(triggered == true) {
	trig_time++;

	if(trig_time >= 100) {
		triggered = false;
		trig_time = 0;
	}
}

// Buscar si se tiene un nuevo valor máximo
if(next_eval_pt > win_max) win_max = next_eval_pt;

// Buscar si estamos por encima del threshold adaptativo
if(next_eval_pt > treshold && !triggered) {
	triggered = true;

	return true;
}
//en caso contrario se finalizará la función antes de devolver
faslo para cambiar potencialmente el threshold

// Se ajusta el threshold adaptativo usando el máximo de la señal
encontrada en la ventana previa
if(win_idx++ >= winSize) {
	// Factor de peso para determinar la contribución del valor
del pico actual con la adaptación del threshold.
	float gamma = 0.175;

	// Se evalúa las observaciones antiguas y elegimos un valor
aleatorio comprendido entre 0.01 y 0.1
	float alpha = 0.01 + ( ((float) random(0, RAND_RES) / (float)
(RAND_RES)) * ((0.1 - 0.01)));

	// Se establece el nuevo threhold
	treshold = alpha * gamma * win_max + (1 - alpha) * treshold;

	// Se reinicia el index de la venta
	win_idx = 0;
	win_max = -10000000;
	}

// Devuelve falso si no se detecta un nuevo QRS
return false;

…
```

Figura 44. Fichero QRS.ino. Implementación del bloque de la toma de decisión .

Para finalizar, el algoritmo de detección QRS puede procesar, tanto la señal analógica generada por el sensor AD8232, como un *array* de datos de tamaño *x*. Es por ello que para comprobar que el algoritmo funciona según lo previsto, se ha realizado una prueba en la que se registran los datos durante un periodo de tiempo en un archivo. Posteriormente, estos datos han sido insertados en el *array* para su posterior procesamiento con la función `detect()`, implementándose en el código del dispositivo Photon una funcionalidad *Serial* para transmitir un aviso por puerto serie una vez el algoritmo detecta el complejo QRS, además del instante de muestreo en la que lo detecta.

Estos datos se han incluido en una hoja Excel y se han representado gráficamente todas las muestras tomadas durante 1 minuto. Los resultados se muestran en la Figura 45, en la que los puntos rojos indican la muestra en la que se detecta el complejo QRS, pudiéndose comprobar que el algoritmo funciona adecuadamente, realizándose un registro de 0 a 20.000 muestras, si bien en la figura sólo se representan las muestras de 0 a 4.000.

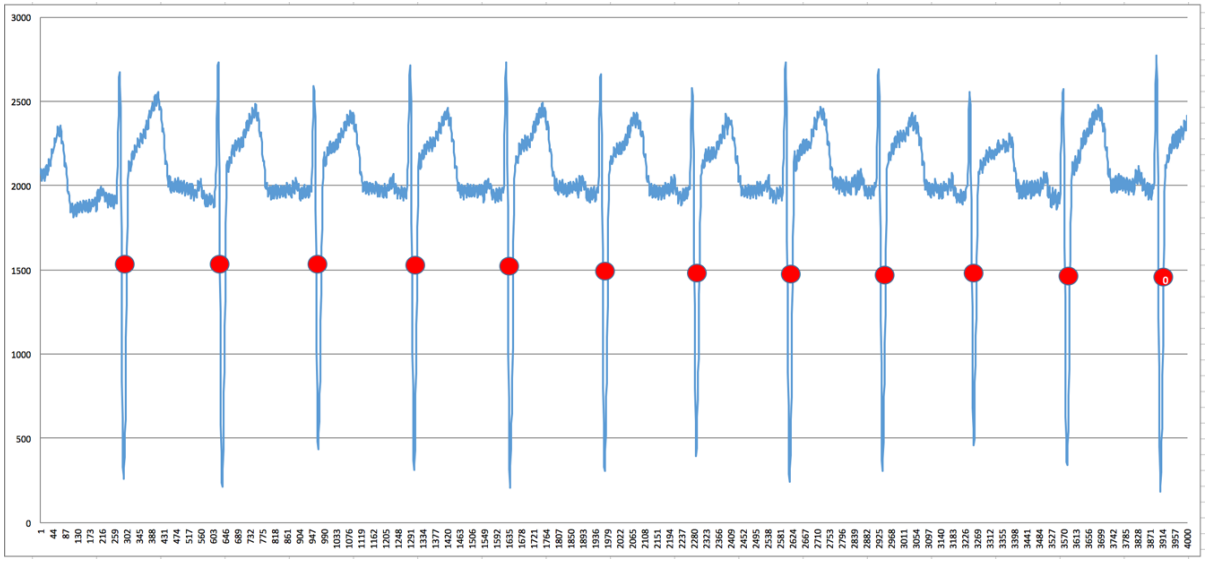


Figura 45. Prueba realizada Algoritmo detección complejo QRS.

4. 5 Detección de caídas

4.5.1 Introducción

Las caídas son la principal causa de daño en las personas ancianas (roturas de cadera, lesiones mortales, etc.) y a partir de los 75 años, es la causa del 70 % de las muertes accidentales [27]. Los más afectados por las caídas son los ancianos que viven solos y no se pueden levantar por ellos mismos. Por otra parte, el hecho de no detectar una caída en tiempo y forma, suele acarrear problemas adicionales. Es por ello que puede resultar conveniente disponer de un sistema automático que detecte una caída y envíe una petición de ayuda en caso de que el paciente no pueda levantarse.

Durante los últimos años se han desarrollado diferentes aproximaciones con respecto al área de detección de caídas, que se categorizan en los siguientes tipos [27] :

- Dispositivos *wearables*.
- Sensores de entorno.
- Sensores ópticos.

La categoría en la que tiene cabida el desarrollo del algoritmo de detección de caídas desarrollado en el presente TFG es la de un dispositivo *wearable*. Este tipo de dispositivos hacen uso de sensores integrados tales como acelerómetros, magnetómetros o giroscopios y su coste suele ser relativamente bajo. Además, tienen la ventaja de instalarse fácilmente, un aspecto importante a tener en cuenta para las personas de avanzada edad.

El objetivo de este apartado es el de presentar el desarrollo de un algoritmo software para la detección de caídas por medio del uso del acelerómetro integrado en el chip LSM9DS1. El ámbito de trabajo de este sistema estará principalmente enfocado en entornos de hogares, zonas de trabajo, etc. El requisito fundamental es la necesidad de la tecnología Wi-Fi.

El algoritmo se encargará de encontrar un modelo particular de caída, tal y como se muestra en la Figura 46. Como bien se introdujo en el **Capítulo 3**, el valor de la gravedad sometido en la superficie terrestre es de 1g = 9,8 m/s^2. Si surge una caída libre, se someterá a una acción desaceleradora de la gravedad y en caso de un impacto, ésta se intensificará. El intervalo temporal de la caída está normalizado a 1 segundo y los valores de caída libre suelen ser inferiores a 0,4 g, mientras que en el impacto son mayores a 2 g.

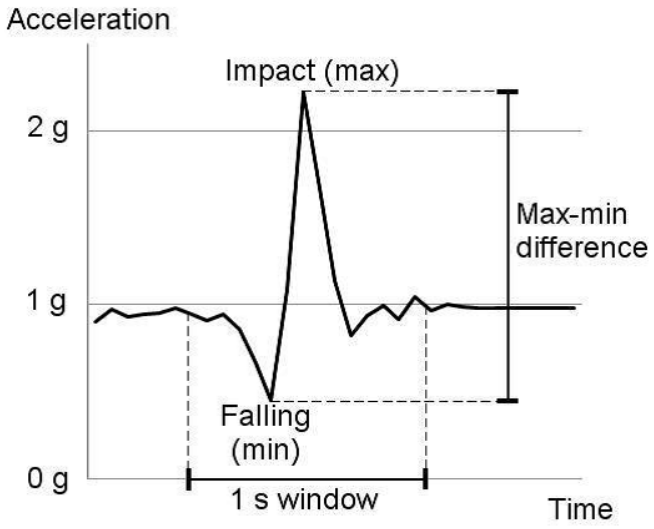


Figura 46. Modelo común en una caída.

Una vez se detecte un modelo de caída por parte del sistema, el siguiente proceso será el de determinar o reconocer la orientación del afectado (horizontal/vertical). En caso de que siga en el suelo en posición tumbada, se generará un aviso de emergencia. No obstante, si está de pie, el sistema detectará un falso positivo. Por consiguiente, hay que prestar especial atención a la hora de posicionar el sensor en el cuerpo, existiendo un acuerdo general por parte de los investigadores en que la mejor ubicación es la cintura [28][29].

El diagrama de flujo mostrado en la Figura 47 resume la funcionalidad del algoritmo de detección de caídas desarrollado.

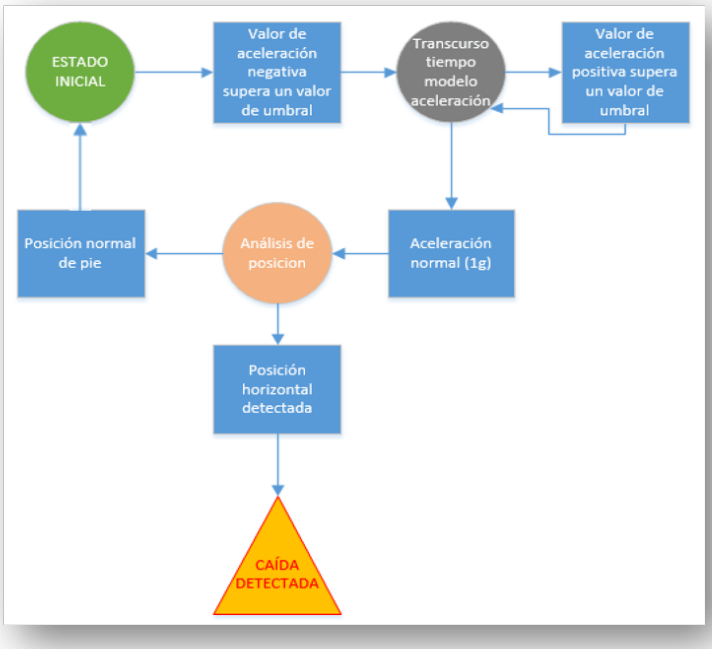


Figura 47. Diagrama de flujo algoritmo detección de caídas.

4.5.2 Algoritmo detección de caída de personas

El algoritmo desarrollado se basa en el concepto de que durante una caída, una persona experimenta una caída libre o disminución de la aceleración, seguido de un gran pico de aceleración, y finalmente un cambio en la orientación. El algoritmo comprueba que la magnitud de la aceleración sobrepasa el umbral inferior. En caso de superarlo, el algoritmo procede a detectar en otro periodo de 0.5 segundos si hay un pico de aceleración que excede el umbral superior.

Seguidamente, el algoritmo comprueba si en otro periodo temporal de 0.5 segundos la persona cambia de orientación. En caso de que cambie la orientación, se comprueba la orientación en un periodo temporal de 10 segundos, lo cual indicará si la persona está inmovilizada y no puede ponerse en pie por sí misma, resultando en una detección positiva de caída.

Cualquier fallo en cada uno de los pasos anteriores hará reinicializar los *triggers* (para conocer los estados del algoritmo) y volver al estado inicial. El diagrama de flujo del algoritmo se muestra a continuación, en la Figura 48.

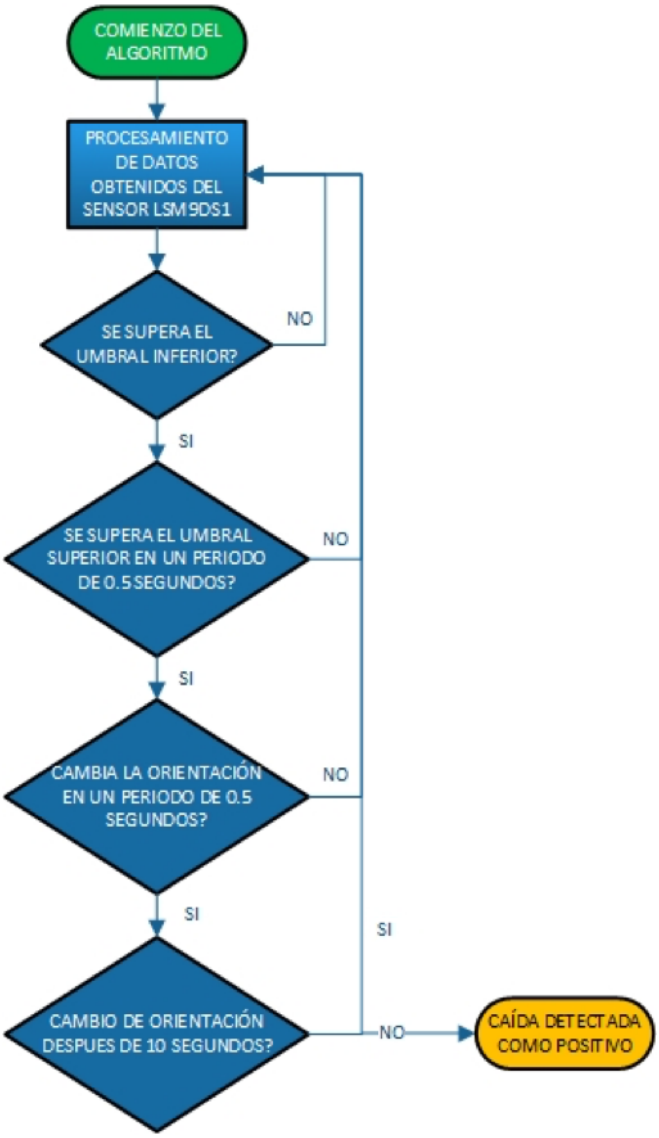


Figura 48. Diagrama de flujo del algoritmo de detección de caída.

En la Figura 49 se presenta la implementación del código relativo al algoritmo de detección de caídas, el cual se ejecutará cada 100ms y está contenido en el fichero `LSM9DS1_Basic_I2C.ino`. A la hora de realizar pruebas, se ha generado fichero de registro a través de la comunicación vía puerto serie, en los que se indicará si los *triggers* están activados, desactivados, y si se ha detectado una caída o no. Finalmente, un LED rojo se encenderá cuando se haya detectado una caída y un *buzzer* sonará para avisar al usuario de que tiene que levantarse para desactivar el aviso de emergencia.

```
//Referencia de aceleración superior
const int bx=0; const int by=0; const int bz=256;
//Vector de referencia
int vector_ref=pow(pow(bx,2)+pow(by,2)+pow(bz,2),0.5);

//Función que contiene el algoritmo desarrollado
void detect_Fall() {

double accel_mag; // Magnitud de la aceleración
double cambio_orientacion=0;

// Para leer datos del acelerómetro se tiene que hacer la llamada a la
función readAccel() que actualizará los valores ax, ay y az de
aceleración gravitatoria.
imu.readAccel();

//magnitud normalizada de las 3 componentes de la aceleracion
accel_mag=pow(pow(imu.calcAccel(imu.ax),2)+pow(imu.calcAccel(imu.ay),2)+p
ow(imu.calcAccel(imu.az),2),0.5);

//a partir de las componentes de la aceleracion se puede determinar la
orientación del dispositivo con la función arcoseno
cambio_orientacion=acos((imu.calcAccel(imu.ax)*(double)bx+imu.calcAccel(i
mu.ay)*(double)by+imu.calcAccel(imu.az)*(double)bz)/(double)accel_mag/(do
uble)vector_ref);

double valor =
((imu.calcAccel(imu.ax)*(double)bx+imu.calcAccel(imu.ay)*(double)by+imu.c
alcAccel(imu.az)*(double)bz)/(double)accel_mag/(double)vector_ref);

//si se supera el threshold inferior (0.4g)
if (accel_mag <=0.4) {
      trigger1=true;
      Serial.println("TRIGGER 1 ACTIVADO");
}

if (trigger1==true) {
      trigger1count++;

//si se supera el threshold superior (3g)
      if (accel_mag >=3) {
            trigger2=true;
            Serial.println("TRIGGER 2 ACTIVADO");
            trigger1=false;
            trigger1count=0;
            }
      }
//espera 0.5s para que la magnitud de la aceleración supere el threshold
superior.
      if (trigger1count>=6) {
      trigger1=false;
      trigger1count=0;
      Serial.println("TRIGGER 1 DESACTIVADO");
      }

      if (trigger2==true) {
      trigger2count++;
```

```
      cambio_orientacion=acos((imu.calcAccel(imu.ax)*(double)bx+imu.calc
Accel(imu.ay)*(double)by+imu.calcAccel(imu.az)*(double)bz)/(double)accel_
mag/(double)vector_ref);
//si la orientación está en posición horizontal (tumbado)
      if (cambio_orientacion >=0.000 && cambio_orientacion <=0.435) {
            trigger3=true;
            trigger2=false;
            trigger2count=0;
            Serial.println(cambio_orientacion);
            Serial.println("TRIGGER 3 ACTIVADO");
            //se activa el buzzer produciendo un sonido de alarma
            digitalWrite(pin_Buzzer, HIGH);
      }
}
//verifica si en 0.5s se produce un cambio en la orientación
if (trigger2count>=6) {
      trigger2=false;
      trigger2count=0;
      Serial.println("TRIGGER 2 DESACTIVADO");
}

if (trigger3==true) {
      trigger3count++;
//*****100******//Permite 10 s al usuario para retomar la posición normal
      if (trigger3count>=100) {
      cambio_orientacion=acos((imu.calcAccel(imu.ax)*(double)bx+imu.calc
Accel(imu.ay)*(double)by+imu.calcAccel(imu.az)*(double)bz)/(double)accel_
mag/(double)vector_ref);
            Serial.print(cambio_orientacion);
      if (cambio_orientacion >=0.000 && cambio_orientacion <=0.435) {
                  fall=true;
                  trigger3=false;
                  trigger3count=0;
                  Serial.println(cambio_orientacion);
      }else if(cambio_orientacion >=2.9 && cambio_orientacion <=3.2) {
                  fall=true;
                  trigger3=false;
                  trigger3count=0;
                  Serial.println(cambio_orientacion);
      }
//El usuario retomó la orientación normal (80-100 grados)
      else if (cambio_orientacion >=1.396 && cambio_orientacion <=1.745)
{
                  trigger3=false;
                  trigger3count=0;
                  Serial.println("TRIGGER 3 DESACTIVADO. FALSE FALL");
                  //se desactiva el buzzer
                  digitalWrite(pin_Buzzer, LOW);
            }
      }
}

//evento de la detección de caída
      if (fall==true) {
            Serial.println("FALL DETECTED");
            //se enciende un LED rojo.
            digitalWrite(digital_blink_RED, HIGH);
            //se desactiva el buzzer
            digitalWrite(pin_Buzzer, LOW);
```

```
                //delay para apagar el LED rojo
                delay(10000);
                digitalWrite(digital_blink_RED, LOW);
                //se reinicia la variable para ir al estado inicial
                fall=false;

        }
}
```

Figura 49. Fichero LSM9DS1_Basic_I2C.ino. Implementación del bloque de la toma de decisión .

4. 6 Transmisión de la señal ECG a través de Internet y representación gráfica

En este subapartado se explicará el desarrollo del *firmware* desarrollado para transmitir los datos obtenidos del sensor AD8232 y enviarlos a través de Internet, concretamente vía TCP, a un servidor *Node.js*, con la finalidad de que posteriormente sea visualizado en una página web del navegador. Este proceso puede observarse en la Figura 50.

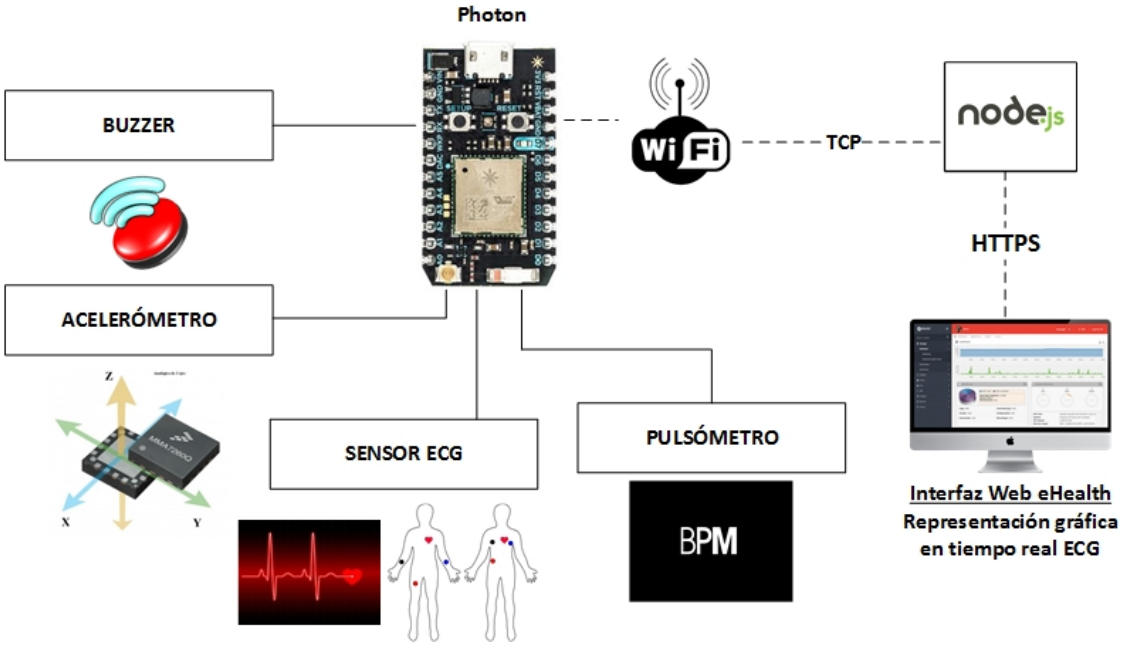


Figura 50. Diagrama de la plataforma con la integración del servidor Node.js

4.6.1 Introducción al servidor Node.js

El servidor *Node.js* es una plataforma construida sobre el motor *JavaScript V8* de *Chrome*, la cual está diseñada para elaborar de forma sencilla aplicaciones de red rápidas y escalables. Utiliza un modelo de E/S no bloqueante orientado a eventos, que lo hace ligero y eficiente. Esto resulta ideal para aplicaciones en tiempo real en las que uno o varios dispositivos realizan transferencias de datos de forma intensiva. De esta forma, *Node.js* se define como una solución para el desarrollo de aplicaciones en *JavaScript* y, en particular, para recibir y responder a peticiones HTTP (*Hypertext Transfer Protocol*).[30]

Esta plataforma se utiliza como entorno de desarrollo para aplicaciones de red en el lado del servidor. Además, *Node.js* utiliza código abierto y proporciona una amplia librería de módulos *JavaScript*, lo que simplifica en gran medida el desarrollo de aplicaciones web. Por otra parte, es importante señalar que, pese a que los módulos pueden instalarse como archivos simples, normalmente se instalan utilizando el gestor de paquetes de *Node.js* (*Node Package Manager*, NPM). Se trata de una herramienta CLI (*Command Line Interface*) que facilita la compilación, instalación y actualización de módulos, así como la gestión de las dependencias.

Otra de las características más significativas del servidor *Node.js* es que trabaja de forma asíncrona. Todas las APIs de la librería *Node.js* son asíncronas, es decir, no bloqueantes. En esencia, esto significa que un servidor basado en *Node.js* nunca espera por los datos de respuesta de una API. En su lugar, el servidor pasa a la siguiente API después de invocarla, utilizando un mecanismo de notificación de eventos para ayudar al servidor a obtener la respuesta correspondiente a la llamada de la API anterior, como se representa en la Figura 51.

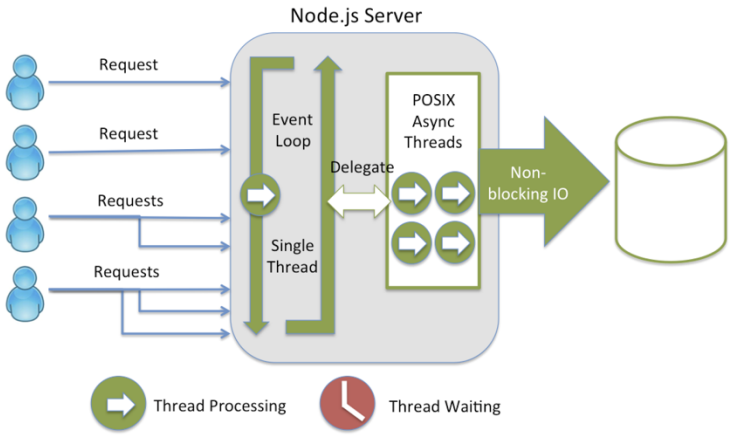


Figura 51. Node.js funcionamiento.

Fuente: http://programaenlinea.net/wp-content/uploads/2016/05/threading_node.png

4.6.2 Representación gráfica en tiempo real a través de TCP

TCP/IP es la base de Internet, y se compone de un conjunto de protocolos denominados *Transmission Control Protocol* (TCP) e *Internet Protocol* (IP). En el desarrollo del presente TFG se usará el protocolo TCP. En resumidas cuentas, la tarea del protocolo TCP es la de controlar, ordenar y manejar los datos con la finalidad de que lleguen a su destino sin que se pierdan paquetes, éste puede actuar como emisor o receptor de paquetes.

El emisor (cliente TCP) divide los datos a transmitir en paquetes y los envía por separado, siendo un punto fuerte de este protocolo su fiabilidad, principal motivo por el que se usa en este aplicativo *eHealth* en lugar del protocolo UDP (*User Data Protocol*), puesto que se asegura que todos los paquetes lleguen al destino, retransmitiéndolos si alguno no se entrega correctamente.

El receptor (servidor TCP) tendrá que ordenar los paquetes secuencialmente para poder obtener la información transmitida. Otros campos importantes a tener en cuenta cuando se trabaja con TCP son el identificador del puerto fuente y puerto de destino, compuesto de 16 bits (de 0 a 65535 valores) y utilizado para identificar, tanto la aplicación emisora como la receptora, respectivamente.

La dirección IP es la identificación específica que el protocolo IP establece en un dispositivo y de la red a la que está conectada. Se compone de un identificador de 32 bits (4 bloques de 8 bits separados por puntos), en el desarrollo de este TFG se establecerá la dirección *localhost* para acceder al servicio web proporcionado por el servidor *Node.js*.

A continuación se describirá el código en lenguaje *Javascript* que hará uso el servidor *Node.js* para la aplicación requerida, donde se recibirán los datos del sensor AD8232 enviados por TCP desde el dispositivo Photon. Primeramente se desarrolla la implementación de un servidor HTTP. El puerto donde actúan las peticiones web es el 8080, mientras que el puerto 8081 es en el que se escucha, de manera que accediendo al enlace http://localhost:8080/ se podrá acceder al archivo de datos correspondiente en lenguaje HTML (en este caso es `index.html`).

Además se hace uso del envío de los datos a través de SSE (*Server Sent Events*), la idea de este concepto, que la aplicación web se subscriba a un *stream* de datos generado por el servidor, y cuando ocurre un nuevo evento, se envía una notificación al cliente. [31]. En la Figura 52 se muestra la implementación del servidor HTTP en el archivo `nodeserver.js`, siendo necesario para su ejecución hacer uso del comando CLI (*Command Line Interface*) siguiendo los pasos mostrados a continuación:

1. Establecerse en el directorio donde está el archivo del servidor por medio del comando : cd
2. Ejecutar el siguiente comando : node <archivo.js>

```
//el puerto del navegador web al que está escuchando. Este se abre
en el navegador a través de http://localhost:8080/
var httpPort = 8080;

// es el puerto TCP que escucha al Photon. Este valor está
codificado en el firmware del Photon.
var dataPort = 8081;

// Este array contiene los clientes (objetos de respuesta http
server) para enviar los datos a través de SSE (Server Sent Events)
var clients = [];

//  For the static files we server out of the
var contentTypeByExtension = {
            '.css':  'text/css',
            '.gif':  'image/gif',
            '.html': 'text/html',
            '.jpg':  'image/jpeg',
```

```
            '.js':   'text/javascript',
            '.png':  'image/png',
};

// Creación del servidor HTTP
http.createServer(function (request, response) {
      try {
            var requestUrl = url.parse(request.url);
            var pathname = path.normalize(requestUrl.pathname);

            if (pathname == '/data' || pathname == '\\data') {
                  // Devuelve los datos SSE
                  var headers = {
                        'Content-Type': 'text/event-stream',
                              'Cache-Control': 'no-cache',
                              'Connection': 'keep-alive'
                  };
                  response.writeHead(200, headers);

                  console.log("starting sse");
                  clients.push(response);
            }
            else {
// Permite http://localhost:8080/ que se use como la URL que se
asociará con el archivo index.html

                  if (pathname == '/' || pathname == '\\') {
                        pathname = 'index.html';
                  }
// Maneja los archivos estáticos como index.html y main.js además
incluye el contenido apropiado para los archivos .html, .js,.css,
etc.
                  var headers = {};
                var contentType =
contentTypeByExtension[path.extname(pathname)];
                if (contentType) {
                  headers['Content-Type'] = contentType;
                }

// Para entregar los archivos en el directorio publico
                  var fsPath = path.join(baseDirectory, pathname);

                  var fileStream = fs.createReadStream(fsPath);
                  response.writeHead(200, headers);
                  fileStream.pipe(response);
                  fileStream.on('error',function(e) {
                        response.writeHead(404);
                        response.end();
                  });
            }
      } catch(e) {
            response.writeHead(500);
            response.end();
            console.log(e.stack);
      }
}).listen(httpPort)
…
```

Figura 52. Fichero nodeserver.js. Implementación del servidor http en Javascript.

A continuación se muestra el desarrollo software del servidor TCP que recibe los datos generados por el dispositivo Photon. Cabe comentar que los datos enviados por TCP desde el dispositivo Photon estarán compuestos por 2 dígitos hexadecimales, para facilitar el proceso con *Javascript*, puesto que con números binarios conllevaría una dificultad adicional. Posteriormente se indicará la implementación del código correspondiente a la transmisión de datos a través de TCP por parte del dispositivo Photon. En la Figura 53, fichero se muestra el código correspondiente al servidor TCP del fichero `nodeserver.js`.

```
// Ejecucion del servidor TCP. Se reciben los datos del Photon.
net.createServer(function (socket) {
	console.log('data connection started from ' +
socket.remoteAddress);

// El servidor envía un valor de 8bit (byte) para cada muestra. Se
usa setEncoding para leer cada byte de datos compuesto por 2 digitos
hexadecimales.

	socket.setEncoding('hex');

	socket.on('data', function (data) {
// Se reciben los datos en esta conexión. Y luego se envían a los
clientes SSE.
		sendDataToClients(data);
	});
	socket.on('end', function () {
		console.log('data connection ended');
	});
}).listen(dataPort);

// Funcion que se hace uso en el servidor TCP y envía los datos a
todos los clientes SSE web browser, los datos tienen que ser de tipo
String.
function sendDataToClients(data) {
	var failures = [];

	clients.forEach(function (client) {
		// console.log("sending data");
		if (!client.write('data: ' + data + '\n\n')) {
			failures.push(client);
		}
	});

	failures.forEach(function (client) {
		console.log("ending sse");
		removeClient(client);
		client.end();
	});
}

…
```

Figura 53. Fichero nodeserver.js. Implementación del servidor TCP.

A continuación se describirá el cliente SSE en lenguaje *Javascript* necesario para recibir el flujo del *stream* de datos y la representación gráfica de dichos datos. Se añade la función `AddEventListener()`,donde se obtienen los datos, y la función `handleData(data)`, en la que se representan gráficamente los datos del ECG. Para la representación gráfica se hizo uso de *HTML5 Canvas*, un elemento definido en el código HTML que usa los parámetros "*width*" (ancho) y "*height*"(alto) y la API *HTML5 Canvas* desarrollada en *Javascript*, con el fin de acceder a un conjunto de funcionalidades de representación gráfica, o incluso generar gráficos dinámicos. La gráfica resultante de la representación del ECG a través de la funcionalidad *canvas* se mostrará en el **Capítulo 5** y presentándose en la Figura 54 el código correspondiente a su implementación, incluido en el fichero `main.js`.

```
// Los datos se almacenan en este array
var dataSet = [];

document.addEventListener('DOMContentLoaded', function() {

  if (!!window.EventSource) {

    var source = new EventSource('data');
    source.addEventListener('message', function(e) {
// e.data es el dato SSE, conformado por dos caracteres
hexadecimales tipo String representando un valor analógico del
sensor AD8232
      for (var ii = 0; ii < e.data.length; ii += 2) {
        handleData(parseInt('0x' + e.data.substr(ii, 2)));
      }
    }, false);
  } else {
    console.log('sse not supported');
  }
}, false);

// Representación gráfica de los datos obtenidos en 2d.
function handleData(data) {
  // Los datos es un valor numérico comprendido entre 0 y 255.

  var canvas = document.getElementById("canvas");
  var ctx = canvas.getContext("2d");

// Añadir al conjunto de datos, eliminar de la izquierda si se pone
más ancho establecido de canvas
  dataSet.push(data);
  if (dataSet.length > (canvas.width - 1)) {
    dataSet.shift();
  }

  // Borrado o despejar la ventana canvas
  ctx.fillStyle = "#ffffff";
  ctx.fillRect(0, 0, canvas.width, canvas.height);
```

```
  // Función de dibujado
  ctx.fillStyle = "#e71d1d";

  for (var ii = 0; ii < dataSet.length; ii++) {
       var yy = 255 - dataSet[ii];
       var yy2 = 255- dataSet[ii+1];
       var ii2= ii+1;

    ctx.lineWidth = 1.5;
    ctx.beginPath();
    ctx.moveTo(ii, yy);
    ctx.lineTo(ii2, yy2);
    ctx.strokeStyle="#e92424";
    ctx.stroke();
  }

}

…
```

Figura 54. Fichero main.js. Recepción de datos SSE y dibujado gráfico.

Finalmente, la transmisión de datos TCP realizada desde el dispositivo Photon usa la función de *Particle* disponible. Concretamente se crea un objeto *TCPClient* y luego se usa la función `connect(IP,port)` de forma que el dispositivo se conecta a la IP y al puerto especificado. Además, devuelve un valor indicando si se ha realizado con éxito o no la conexión. También se añade la funcionalidad para saber si el cliente está conectado a través de `connected()` y así descartar cualquier dato entrante. El envío de los datos al servidor al que se está conectado, se realiza por medio de la función `write(val)`, siendo `val` de tipo *char* o *byte*.

La tasa de envío de datos al servidor TCP es de 5ms, siendo la óptima, ya que disminuyéndola implicaría una saturación y una incorrecta representación de los datos. Esta frecuencia de envío de datos demuestra que el aplicativo puede trabajar en tiempo real. Además de lo dicho anteriormente, se realiza una conversión de valores analógicos de 0 - 4095 (12 bits) a 0-255 (8bits) para trabajar con el método establecido en el servidor TCP. El código desarrollado para esta función, disponible en el fichero `TCP_Photon.ino`, se muestra en la Figura 55.

```
// Frecuencia de envío de muestras
const unsigned long SEND_PERIOD_MS = 5;
// Direccion IP y puerto del servidor. El servidor usa dos puertos,
uno para el web browser y otro para los datos analógicos de recepción
```

```
TCP. Este es el del puerto de datos y no el del web server.
IPAddress serverAddress(192,168,1,8);
int serverPort = 8081;

// Máquina de estado finita
enum {CONNECT_STATE, SEND_DATA_STATE};

// Creacion objeto TCPClient
TCPClient client;
unsigned long lastSend = 0;
int state = CONNECT_STATE;


void setup() {
	Serial.begin(9600);
	pinMode(INPUT_PIN, INPUT); //PIN datos analógicos del AD8232
}

void loop() {
	switch(state) {
	case CONNECT_STATE:
		Serial.println("connecting...");
		if (client.connect(serverAddress, serverPort)) {
			state = SEND_DATA_STATE;
		}
		else {
			Serial.println("connection failed");
			delay(15000);
		}
		break;

	case SEND_DATA_STATE:
		if (client.connected()) {
		// Descartar cualquier dato de entrada, no debería haber
ninguno…
			while(client.available()) {
				client.read();
			}
			// Envío de datos al servidor cada 5ms
			if (millis() - lastSend >= SEND_PERIOD_MS) {
				lastSend = millis();

// analogRead devuelve valores comprendidos entre 0 - 4095; se
eliminan los bits mas bajos y por lo tanto obtenemos de  0 – 255.
Con el fin de enviar un caracter de 8 bits.
				int val = analogRead(INPUT_PIN) >> 4;

				client.write((unsigned char)val);

// En caso de que los electrodos estén desactivados se envía un valor
constante 0
			if((digitalRead(LEADS_OFF_PLUS_PIN) ==
1)&&(digitalRead(LEADS_OFF_MINUS_PIN) == 1)) {
	unsigned char intermediate = 0;
	client.write(intermediate);
}
			else {
				client.write((unsigned char)val_ECG);
			}
```

```
                }
            }
            else {
                // Disconnected
                Serial.println("disconnected...");
                client.stop();
                state = CONNECT_STATE;
                delay(5000);
            }
            break;
    }}
```

Figura 55. Fichero TCP_Photon.ino. Implementación envío datos TCP.

4.6.3 Desarrollo de la interfaz web basada en el lenguaje HTML

En este subapartado se mostrarán las capturas de la página web finalizada y las porciones de código más importantes de su desarrollo. Cabe destacar que se trata de una versión que abre las puertas a mayores funcionalidades como la interacción con bases de datos (*MySQL*), entre otras. Se compone de 4 archivos fundamentales, `login.html` que simula un *login* por parte del usuario/a, `index.html` donde se muestra la ventana principal, `panels.html` donde se visualizarán las alertas generadas por una caída, y `charts.html` donde se expone la gráfica en tiempo real correspondiente al ECG del paciente, así como las variables de pulsaciones por minuto o IBI (intervalo entre pulsos).

El diseño de la página web se ha desarrollado basándose en la estructura *dashboard*, que es una interfaz gráfica donde se albergan *widgets* o miniaplicaciones como pueden ser calendarios, lista de tareas, mensajes, entre otras. En este diseño se ha planteado el uso de un calendario para que el usuario pueda gestionar las citas con los pacientes, un panel de mensajería con los pacientes para interactuar con ellos cuando sea necesario, unos datos estadísticos que muestran diversos parámetros, tales como los pacientes que están *online* transmitiendo datos por medio del dispositivo Photon, el índice de respuesta a los mensajes recibidos por los pacientes, un parámetro informativo de la cantidad de pacientes que se le han asignado al usuario recientemente, y para concluir, el índice de alerta de caídas en los pacientes.

Ahora bien, la utilización de los *widgets* o la fase de *login*, la configuración de la cuenta, el campo de búsqueda y el estado de conectividad de los pacientes, no funcionan como realmente lo deberían hacer, simplemente se trata de una plantilla que muestra todas estas características, dejando para futuros trabajos la implementación de su correcto funcionamiento. El ambiente está recreado alrededor de las posibles herramientas que un usuario necesitaría.

Comenzando por `login.html`, se visualiza una ventana básica en el que se introducirá el email, DNI, u otros campos de identificación del usuario, como se muestra en la Figura 56. A continuación se proporciona una contraseña y se muestra un botón para entrar. En este caso no se ha establecido una condición para comprobar los datos, y el botón *login* derivará directamente a la siguiente página web `index.html`.

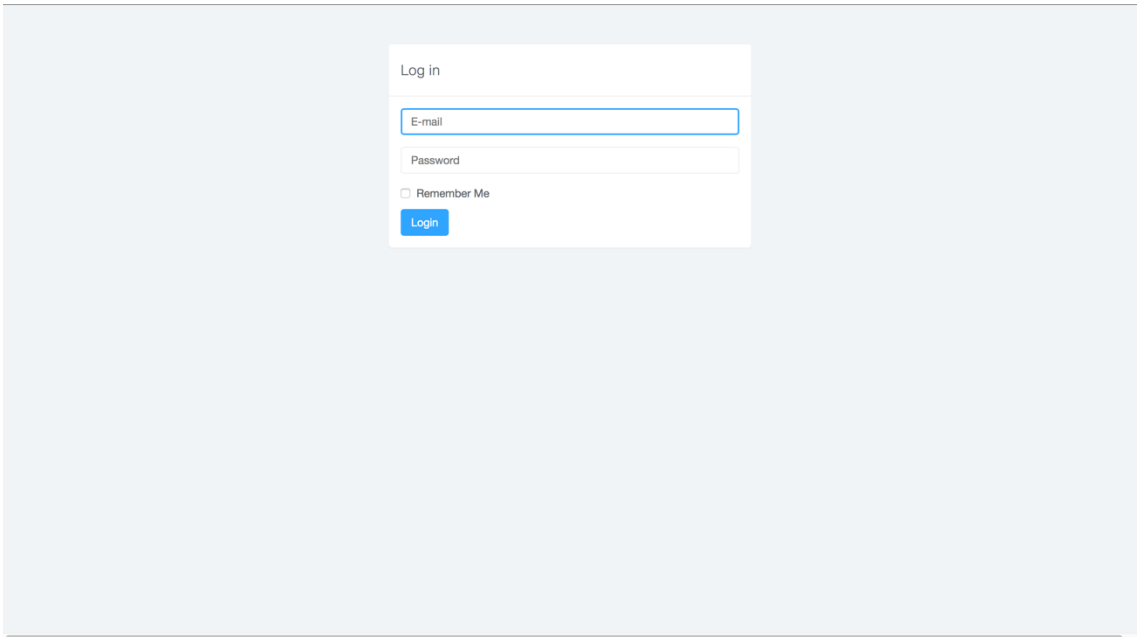


Figura 56. Fichero login.html. Visualización en el navegador.

A continuación, en la página `index.html` visualizada en la Figura 57, se muestran los widgets comentados anteriormente. Se ha establecido un diminutivo a la aplicación web como *Heartify*, y se ha añadido un panel de configuración para cerrar la sesión, configurar el perfil o para la configuración de la página web. A la izquierda se ofrece un campo de búsqueda, y justo debajo, si se selecciona la opción *Alertas,* derivará al archivo `panels.html`.

Figura 57. Fichero index.html. Dashboard principal.

`Panels.html` muestra una ventana donde se indicarán los eventos generados al detectarse una caída. Existe también otra pestaña, denominada *Pacientes,* que desplegará todos los pacientes asociados al usuario y sólo para los que aparezcan en modo *online* se podrá acceder a los datos en tiempo real. Si se selecciona un paciente *online*, derivará al fichero `charts.html`, el cual se muestra en la Figura 58, y comprende un recuadro con la visualización de la señal ECG.

Puesto que en el siguiente capítulo se describirá cómo obtener los datos proporcionados de la nube de *Particle*, esta sección se enfocará a la representación gráfica de la señal ECG. Como bien se comentó, se hará uso de *canvas* y de su implementación en HTML, mostrada en la Figura 59. Es importante destacar que hay que establecer un *id* en *canvas*, que se ha denominado *canvas,* y que enlazará con el archivo `main.js` para la representación gráfica desarrollada.

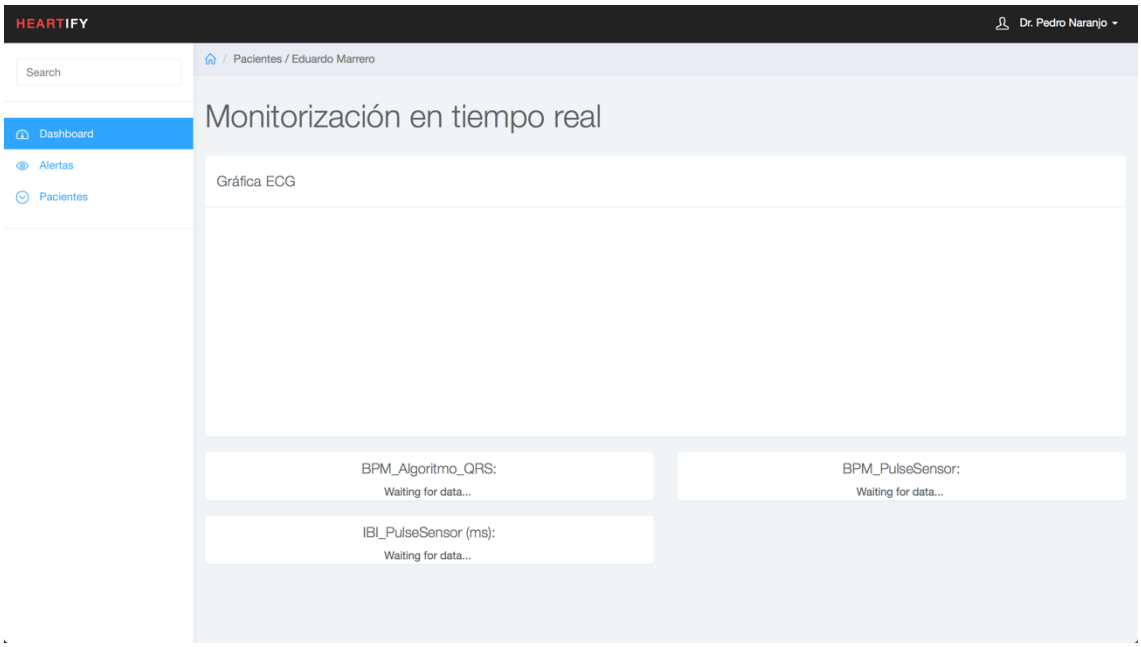


Figura 58. Fichero charts.html. Representación gráfica de ECG en tiempo real y variables BPM,IBI.

```
<div class="row">
                    <div class="col-lg-12">
                          <div class="panel panel-default">
                    <div class="panel-heading">Gráfica ECG</div>
                                <div class="panel-body">
                                      <div id="graphDiv">
        <canvas id="canvas" width="500" height="256"></canvas>
                                      </div>
                                      <!-- graphDiv -->
                                </div>
                          </div>
                    </div>
              </div>
...
```

Figura 59. Fichero charts.html. Código para la implementación de canvas.

Para más detalles del código HTML generado, se puede consultar el contenido del CD-ROM adjunto a este documento.

Capítulo 5. Diseño de la plataforma final

-

5. 1 Interconexión de la plataforma final

Una vez definida la distribución de los distintos elementos en la plataforma final, el siguiente paso consistió en determinar el modo en el que se conectarían entre sí. Para ello, en este apartado se muestran, una a una, las conexiones establecidas entre cada uno de los sensores o componentes que forman parte de la plataforma, y el dispositivo Photon. A continuación, en la Tabla 9, 10, 11 y 12 se muestra dicha información.

- Sensor AD8232 "*Heart Monitor*" :

Tabla 9. Conexionado AD8232 final con el dispositivo Photon.

PIN	Descripción	Photon Conexión
GND	Tierra	GND
3.3v (VIN)	3.3v Fuente	3.3
OUTPUT	Output Signal	A3
LO-	Leads-off Detect -	D3
LO+	Leads-off Detect +	D2
SDN	Shutdown	No se usa

- Sensor LSM9DS1 "*Breakout*":

Tabla 10. Conexionado Acelerómetro final con el dispositivo Photon.

Pin Label	Pin Function	Photon Conexión
GND	Tierra (0V)	GND
VDD	3.3 V Fuente	3.3
SDA	SPI: MOSI I^2C: Serial Data	D0
SCL	Serial Clock	D1

- "*Pulse Sensor*"

Tabla 11. Conexionado "Pulse Sensor" final con el dispositivo Photon.

PIN	Descripción	Photon Conexión
GND	Tierra	GND
VIN	3V a 5V	3.3
SIGNAL	Datos, color púrpura	A2

- "*LEDs*"

Tabla 12. Conexionado de LEDs final con el dispositivo Photon.

	PIN	Descripción	Photon Conexión
LED Rojo	**Cátodo**	Tierra	GND
	Ánodo	1.8V a 3.6V	D6
LED Verde	**Cátodo**	Tierra	GND
	Ánodo	1.8V a 3.6V	D4

Fritzing es un programa de automatización de diseño electrónico libre que busca ayudar a diseñadores y artistas para que puedan pasar de prototipos (por ejemplo, placas de pruebas) a productos finales. Permite a los usuarios documentar sus prototipos basados en Arduino y crear esquemas de circuitos impresos para su posterior fabricación [32]. En la Figura 60 se muestra el logotipo del programa.

Figura 60. Logotipo del programa Fritzing.

A continuación se mostrarán los esquemáticos del diseño final (tanto gráfico, Figura 61, como eléctrico Figura 62) de la plataforma de monitorización *eHealth* a través de dicha aplicación. El color negro del cableado que se muestra en la Figura 61 establece la referencia a tierra (GND) y el cable rojo, la alimentación de 3.3V para suministrar a todos los componentes.

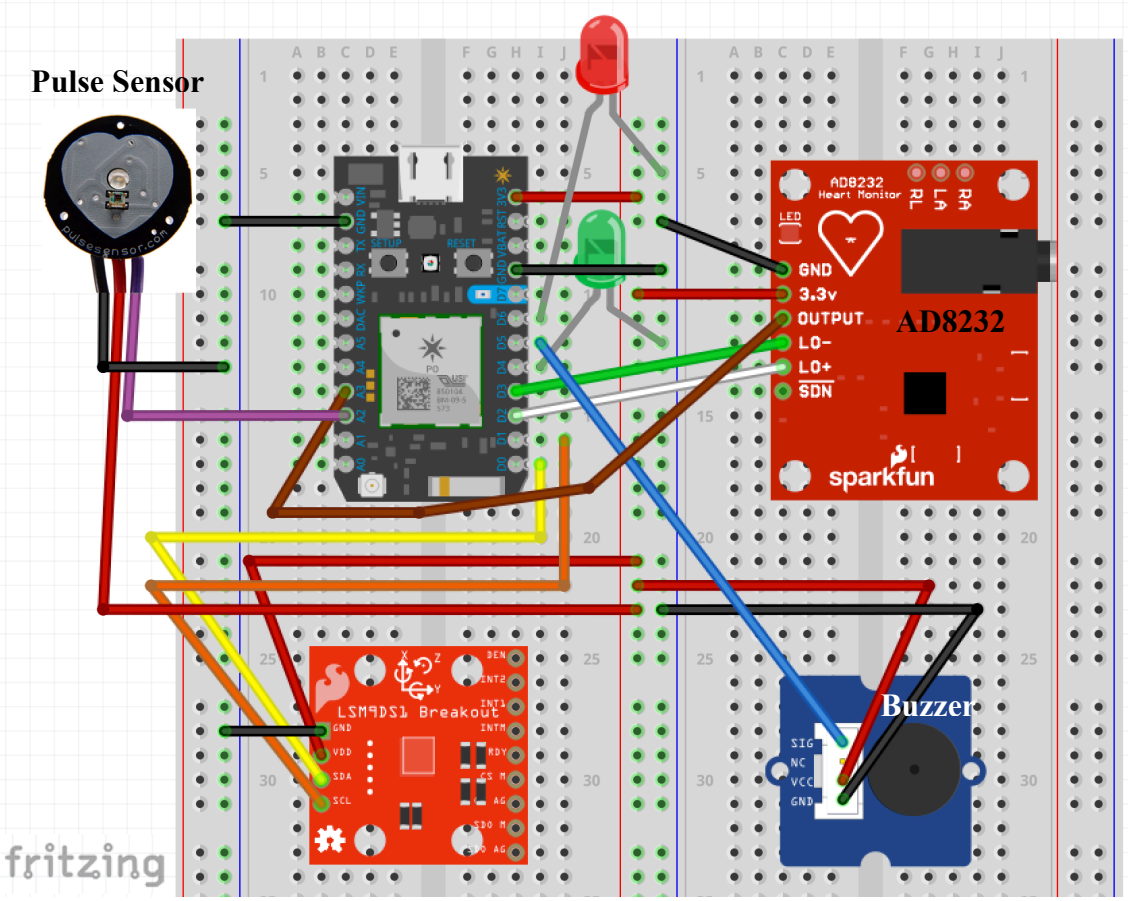


Figura 61. Esquemático gráfico final de la plataforma eHeatlh.

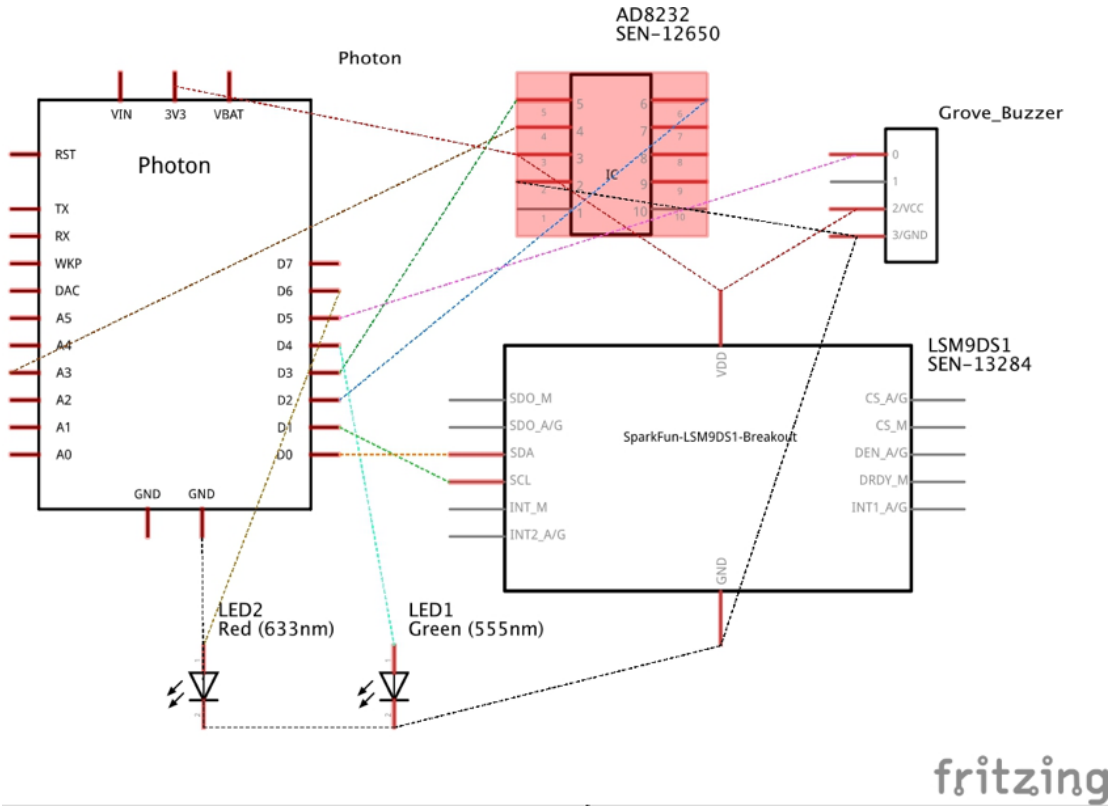


Figura 62. Esquema eléctrico final de la plataforma eHealth.

5. 2 Modificaciones en el firmware e implementación final

En el **Capítulo 4** se describieron de manera independiente cada uno de los elementos software que componen el proyecto (Obtención datos ECG, Obtención datos BPM e IBI, Algoritmo de detección complejo QRS, Algoritmo detección de caídas y la transmisión de la señal ECG a través de Internet). Ahora bien, lo que se pretende es conformar todos estos elementos independientes en un único código que se ejecutará en el dispositivo Photon, haciendo uso de librerías/APIs externas.

El desarrollo del *firmware* ha pasado por diversas versiones, concretamente por cinco. Cada versión funcionó correctamente con los elementos incorporados, y añadiendo en cada caso nuevas funcionalidades. Se explicará una por una hasta llegar a la conformación del código final, así como los problemas encontrados y cómo se llegaron a solventar. Los ficheros de cada versión se denominan `eHealth_Platform_vX.ino` dónde `X` corresponde al número de versión.

- La primera versión comprende la unificación del código correspondiente a la obtención de datos ECG y la representación visual vía puerto serie junto con la obtención de los datos BPM e IBI a través del dispositivo "*Pulse Sensor*". Además se realizó una prueba de envío de datos ECG a través de UDP, puesto que la idea inicial era transmitir los datos mediante este protocolo. Los datos BPM e IBI eran mostrados vía puerto serie.

- La segunda versión integra todos los componentes de la primera versión, pero se le añade el algoritmo de detección QRS y el encendido del LED *D7* del dispositivo Photon una vez lo detecte. En este apartado surgió un problema inesperado al llamar la función `analogRead()` dentro del `loop()` en el código del dispositivo Photon. Para solventarlo se incorporó el algoritmo de detección QRS en el archivo `SparkIntervalTimer.cpp`.

- En la tercera versión se modificó la representación de los datos ECG, se analizaron diferentes métodos para representarlos remotamente y poder visualizarlos en tiempo real. El abanico de opciones que se testeó para realizar esta tarea se muestra a continuación y finalmente se optó por usar el servidor *Node.js + canvas* por su representación adecuada en tiempo real con una tasa de envío de 5ms y fiabilidad de los datos mediante el uso del protocolo TCP.

    - THINGSPEAK: Es una plataforma y API para IoT que permite recolectar, almacenar, analizar y visualizar datos, en dispositivos tales como Arduino, Raspberry Pi, y otros dispositivos de IoT. Se hace uso de canales que reciben los datos, los procesan y los visualizan en código Matlab. Además se pueden establecer alarmas [33]. Se realizó una prueba y se comprobó que por medio de la API de *ThingsPeak* ejecutándose el código en el dispositivo Photon, la visualización no se realizaba en tiempo real por mucho que se redujera la tasa de envío, por lo tanto se descartó esta opción y se buscó otra más eficiente.

- BLYNK: Se trata de un *dashboard* de aplicación en iOS y Android en el que se pueden crear interfaces graficas a través de *widgets* para controlar Arduino, Raspberry Pi y otros dispositivos de IoT a través de Internet. [34] No se ha optado por esta opción por la simple razón de que no puede implementarse en un navegador web.

- UBIDOTS: Es un servicio en la nube que permite almacenar e interpretar información de sensores en tiempo real, haciendo posible la creación de aplicaciones para IoT de una manera fácil [35]. Las ventajas de *Ubidots* son que se puede implementarse con pocas líneas de código en cada dispositivo, mientras que la principal desventaja es que no trabaja 100 % a tiempo real, puesto que la tasa máxima de envío del dispositivo Photon sin que se distorsionaran o perdieran los paquetes UDP de la señal ECG, era de 50ms. En la Figura 63 se pueden apreciar ciertos paquetes perdidos o corrupción de datos, descartándose por este motivo esta opción.

Figura 63. Ubidots con una tasa de recepción UDP de 50 ms.

- Además, en la tercera versión, se añadió un LED de color verde para que destelle cuando el dispositivo "*Pulse Sensor*" detecta un pulso, así como el envío de las variables BPM e IBI por medio de la funcionalidad de *Particle Cloud* que se describirá en el apartado 5.1.1

- La cuarta versión contiene la funcionalidad desarrollada en la versión anterior, pero añadiéndole el algoritmo de detección de caídas por medio del software *Timer* que se explicará en el apartado 5.1.1, debido a que debe operar a una frecuencia de muestreo establecida en 100ms. También se añade un LED rojo que se encenderá cuando detecte una caída, un *buzzer* para avisar al usuario que se tiene que levantar, y la funcionalidad *IFTT* que se detallará en el apartado 5.1.2 para enviar un SMS de emergencia. Además se incorporó en el servidor *Node.js* la generación de un archivo en el que se almacenan todos los datos transmitidos por el dispositivo Photon, con el fin de analizar los datos y poder representarlos posteriormente por medio de programas tales como Excel.

5.1.1 Envío de variables, generación de eventos a través de Particle Cloud y software Timers

Particle ofrece una API con funciones en la nube muy útiles. En primer lugar, para usar esta API se deberá asociar el dispositivo Photon con el que se trabajará, a la cuenta de *Particle*. Cada dispositivo tiene una URL que puede ser usada para obtener variables o exponer funciones, entre otras funciones, de modo que trabajan como recursos en el dispositivo. Después se generará un "*Device ID*" y "*Access Token*" que podrán obtenerse a través del IDE *Particle Dev* y que serán necesarios para poder usar estas funciones. Las funciones que se usarán en el presente TFG son las que se muestran a continuación :

- `Particle.variable("nombre", variable)`: Expone una variable a través del *cloud* de *Particle* de manera que puede ser llamada fácilmente con el método HTTP denominado "GET" en cualquier página web. Se pueden usar hasta 20 variables.
  - `nombre`: está limitado a 12 caracteres.
  - `variable`: se soportan los tipos *Int*, *Double* y *String*.

- `Particle.publish("name",data)`: Publica un evento a través de *Particle Cloud* al cual se subscribirán por medio de *callbacks* o SSE. Esta característica permite al dispositivo generar un evento basado en una condición. Se establece por defecto un tiempo de vida (*TTL*) del evento de 60 segundos.
  - `name`: Caracteres ASCII.
  - `data`: Hasta 255 bytes

Ahora bien, en el aplicativo desarrollado se desean exponer las variables de las pulsaciones por minuto e IBI generadas por el dispositivo "*Pulse Sensor*" y los pulsos por minuto del algoritmo de detección QRS. Hay que mencionar, además, que se generará un evento cuando se detecte una caída. Estas variables y eventos se deberán visualizar en el aplicativo web. En la Figura 64 se muestran las porciones del código que se ejecutan en el dispositivo Photon por medio del fichero `eHealth_Platform_v4.ino` y que implementa estas funcionalidades de la nube de *Particle*.

```
//Variables externas al fichero.ino donde se generan los valores de
las variables

extern int bpm_value2;
extern int BPM;
extern int IBI;

//Función setup() en la que se integra la función
Particle.variable()
void setup(){
…
Particle.variable("bpmQrs",BPM_QRS);
Particle.variable("pulseBPM",BPM_Pulse);
Particle.variable("pulseIBI",IBI_Pulse);
…
}
//Función detect_Fall() en la que se integra la función
Particle.publish() para generar un evento cuando se detecta una
caída.

void detect_Fall() {
…

if (fall==true) {
//Se genera un evento con el nombre simulado de un paciente y la
hora en la que se cayó por medio de la función Time de Particle.
      Serial.println("FALL DETECTED");
      Time.zone(1);
      String caida = "Caída Paciente Javier Ascanio Detectada a las
"+Time.timeStr();
      Particle.publish("TrueFall",caida);
      digitalWrite(digital_blink_RED, HIGH);
```

```
        digitalWrite(pin_Buzzer, LOW);
        delay(10000);
        digitalWrite(digital_blink_RED, LOW);

        fall=false;

}}
…
```

Figura 64. Fichero eHealth_Platform_v4.ino. Implementación de las funciones de Particle Cloud.

Finalmente, en lo que respecta al algoritmo de detección de caídas, se ejecutará a una frecuencia de muestreo de 100ms. Si bien las interrupciones en el fichero `PulseSensor_Spark.cpp` se ejecutan cada 2ms y la función *loop()* del código se ejecuta a otra frecuencia predeterminada sin ningún *delay*, hay que hacer uso de los *Software Timers* proporcionados en la API de *Particle* para generar de manera independiente la función `detect_fall()` y que se ejecute como se ha establecido, con una frecuencia de 100ms.

Los *Software Timers* establecen un medio para ejecutar acciones temporales en el programa (*callbacks functions*). *FreeRTOS*, que es el sistema operativo que se ejecuta en el dispositivo Photon, permite soportar hasta 10 *Software Timers* simultáneamente con una resolución mínima de 1ms [36]. A continuación se muestra la sintaxis del uso de los *Software Timers* y el código implementado en la Figura 65 del archivo `eHealth_Platform_v5.ino`.

- `Timer timer(period, callback)`: Generación del objeto *timer*.
    - `period` : Es el periodo del *timer* en milisegundos (*unsigned int*)
    - `callback` Es la función *callback* que se llamará cuando el *timer* expire.

- `start()`: Da comienzo al *Timer* que está detenido, y si un *Timer* se está ejecutando, se vuelve a reiniciar.

```
#define PRINT_SPEED 100
Timer timer(PRINT_SPEED, detect_Fall);

void setup(){
…
timer.start();
…
}
```

```
void detect_Fall() {
…
…
}
```

Figura 65. Fichero eHealth_Platform_v4.ino. Implementación del Software Timer.

5.1.2 Servicio IFTT y logs con node.js

IFTTT o "*If This Then That*" es un servicio web que integra otras aplicaciones web en un solo lugar y con las que se pueden realizar acciones según un conjunto de criterios. De hecho, su uso es muy sencillo e intuitivo y ofrece multitud de características que pueden usarse diariamente.

Después de registrarse en *IFTTT* gratuitamente, se podrán crear "Recetas". Estas recetas incluyen dos aplicaciones web, una aplicación que establece una condición, y la otra aplicación para realizar una acción. El funcionamiento sigue el siguiente razonamiento, " Si algo ocurre, luego desempeña una acción" como se muestra en la Figura 66.

Figura 66. Funcionamiento de IFTT.

Se incluyen aplicaciones como *Dropbox*, *Feedly*, *Facebook*, *Twitter*, *Craigslist*, *ESPN*, *Foursquare*, *Google Calendar*, *Instagram*, *Google Drive*, *Gmail*, *Skydrive*, *Tumblr*, *Bitly* y *Particle***,** entre muchos otras. La funcionalidad que se desea obtener con esta herramienta es la de generar un SMS una vez se detecte una caída por medio del algoritmo desarrollado, y registrar este evento en un archivo en la plataforma *Dropbox*.

Para ello se monitorizará si la función `Particle.publish ("trueFall",data)` publica un evento asociado a la detección de una caída y en su caso, actuará generando un SMS a un número de teléfono específico. En la Figura 67 se muestra cómo se ha implementado el envío de SMS y en la Figura 68, el registro de los datos del evento por medio de un archivo "*.txt*" en *Dropbox*.

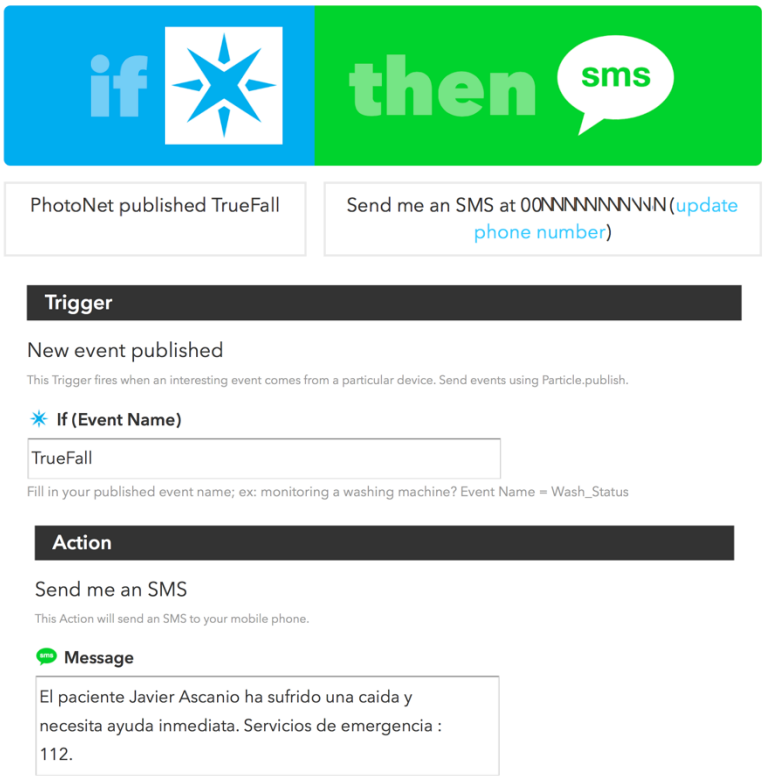
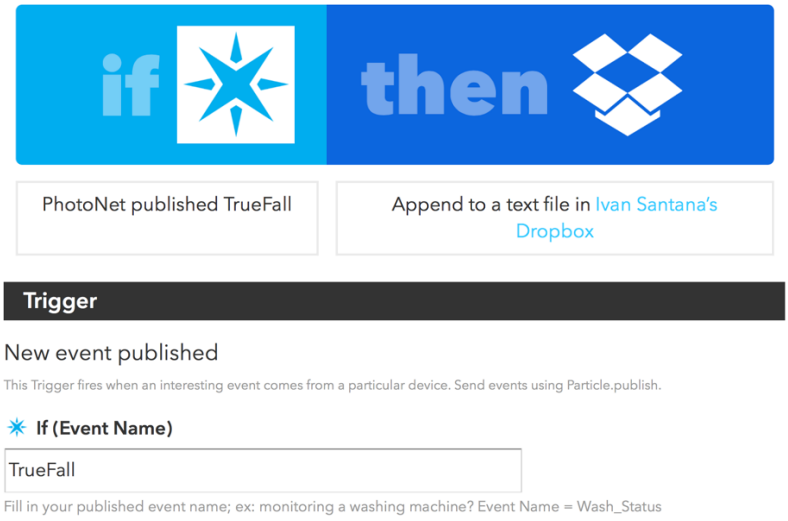


Figura 67. Implementación del servicio de mensajería SMS por medio de IFTT



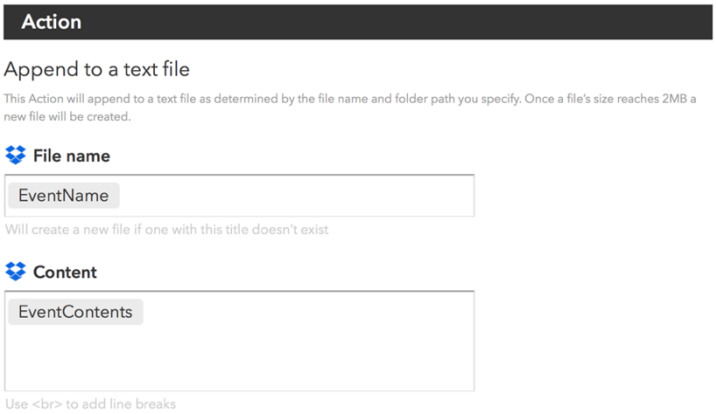


Figura 68. Implementación del registro de eventos en Dropbox por medio de IFTT

A continuación, se examinará brevemente cómo registrar todos los datos TCP recibidos en un archivo mediante un servidor *Node.js*. La implementación se ubica en la función del servidor TCP (archivo `livegraph.js`). A continuación, en la Figura 69 se puede ver que con una línea de código puede efectuarse dicha tarea, concretamente se usa la función `outStream.write()`.

```
// Se genera un archivo .txt en el mismo directorio donde se ubica
livegraph.js
var outStream = fs.createWriteStream("LOG_ECG.txt");

net.createServer(function (socket) {
      …
      socket.on('data', function (data) {
// Escribir en el archivo txt y que separa los valores por comas
            outStream.write(data.toString()+",");
            sendDataToClients(data);
      });
      socket.on('end', function () {
            console.log('data connection ended');
      });
}).listen(dataPort);

…
```

Figura 69. Fichero livegraph.js. Implementación del registro de datos en JavaScript.

5.1.3 Visualización de variables y eventos en HTML

Para mostrar las variables enviadas por `Particle.variable()` en el navegador web se necesitan los ingredientes AJAX y jQuery. AJAX es un modo de usar

Javascript para realizar funcionalidades de forma asíncrona en una página web. Hay muchas formas de usarlo, pero se usará la forma más sencilla, a través de una solicitud HTTP GET como se mostró el apartado 5.1.1.

Una forma segura de transmitir datos por Internet es en formato JSON. Por suerte, el equipo de *Particle* entiende esto y todas las variables de *Particle* devuelven sus datos en formato JSON. La nube de *Particle* también se ha configurado de la misma manera a través del uso de "*Access tokens*" para identificar a los usuarios.

Por lo tanto, se usa la API AJAX *jQuery* y luego la función *getJSON* para realizar la tarea de obtención de las variables. Debido a que los datos están en el formato JSON, en el *callback* de la función se puede usar su argumento como una estructura para acceder a distintos campos donde se publican los datos.

Para recargar las variables automáticamente en la página web se hace uso de AJAX, realizando un *polling* de manera eficiente, siendo unos valores adecuados de actualización de variables son cada 1 o 2 segundos. Se ha añadido *getJSON* con una función `window.setInterval()` que llama al conjunto de la función *JavaScript* cada segundo (1000ms). Se debe proporcionar el *device ID*, *Access Token* y nombre de la variable para poder hacer uso de esta funcionalidad. A continuación se muestra el código del fichero `charts.html` para poder mostrar las variables de *Particle* en la web.

```
<span id="temp">Waiting for data...</span>
<br>
<script type="text/javascript">

      window.setInterval(function() {
      var deviceID = "XXXXXXXXXXXXXXXXXXXXXXXX";
      var accessToken = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
      var varName = "nombre_evento";

      requestURL = "https://api.spark.io/v1/devices/" + deviceID +
"/" + varName + "/?access_token=" + accessToken;

      $.getJSON(requestURL, function(json) {
      document.getElementById("temp").innerHTML = json.result;
      document.getElementById("temp").style.fontSize = "28px";
                                    });
                                          }, 1000);
                                    </script>
```

Figura 70. Fichero "charts.html". Visualización de variables de la nube de Particle en HTML.

Ahora se explicará la visualización de eventos obtenidos de la nube de *Particle* en el navegador, por medio del lenguaje HTML. Un evento de flujo de datos SSE se abre y el navegador web comienza a escuchar eventos. Se debe proporcionar el *device ID*, *Access Token* y nombre del evento para poder hacer uso de esta funcionalidad. A continuación en la Figura 71 se muestra el código del archivo `panels.html`.

```
<span id="uptime"></span>
<br>
<span id="tstamp"></span>

<script type="text/javascript">

      var deviceID = " XXXXXXXXXXXXXXXXXXXXXXX";
      var accessToken = " XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
var eventSource = new EventSource("https://api.spark.io/v1/devices/"
+ deviceID + "/events/?access_token=" + accessToken);

eventSource.addEventListener('TrueFall', function(e) {
            var parsedData = JSON.parse(e.data);
            var tempSpan = document.getElementById("uptime");
            var tsSpan = document.getElementById("tstamp");

<!– Para posicionar el contenido del evento en la página web -->
            tempSpan.innerHTML += "<div style='position: relative;
padding-bottom: 85px; border-style:solid; border-width: medium;
border-color: #000000 ; background-color: #ff0000; color:#ffffff;
left: 270px; top: 15px; height: 50px; width:
500px;'><b>"+parsedData.data+"</b></div>";

            tempSpan.style.fontSize = "25px";

            }, false);
</script>
```

Figura 71. Fichero "panels.html". Visualización de eventos de la nube de Particle en HTML.

5. 3 Funcionamiento de la plataforma final y resultados obtenidos

El funcionamiento de la plataforma final *eHealth,* desarrollada en el presente TFG se basa en la ejecución de las siguientes funciones :

- **Obtención de datos ECG y representación gráfica en el navegador web en tiempo real a través de la red.**

- **Envío de variables referentes a las pulsaciones por minuto (BPM) e intervalo entre pulsos (IBI) y representación en el navegador web a través de la red.**

- **Detección de caídas y en caso de detectarse, se muestra un aviso en el navegador y en el móvil a través de un SMS.**

- **Registro de datos para un análisis posterior.**

Para llevar a cabo cada una de las tareas anteriores, fue necesario definir la distribución de los componentes que conformarán la plataforma HW/SW final. En la Figura 72 se muestra el diagrama de conexión final de la plataforma y en la Figura 73 una imagen de la *protoboard* de la plataforma final.

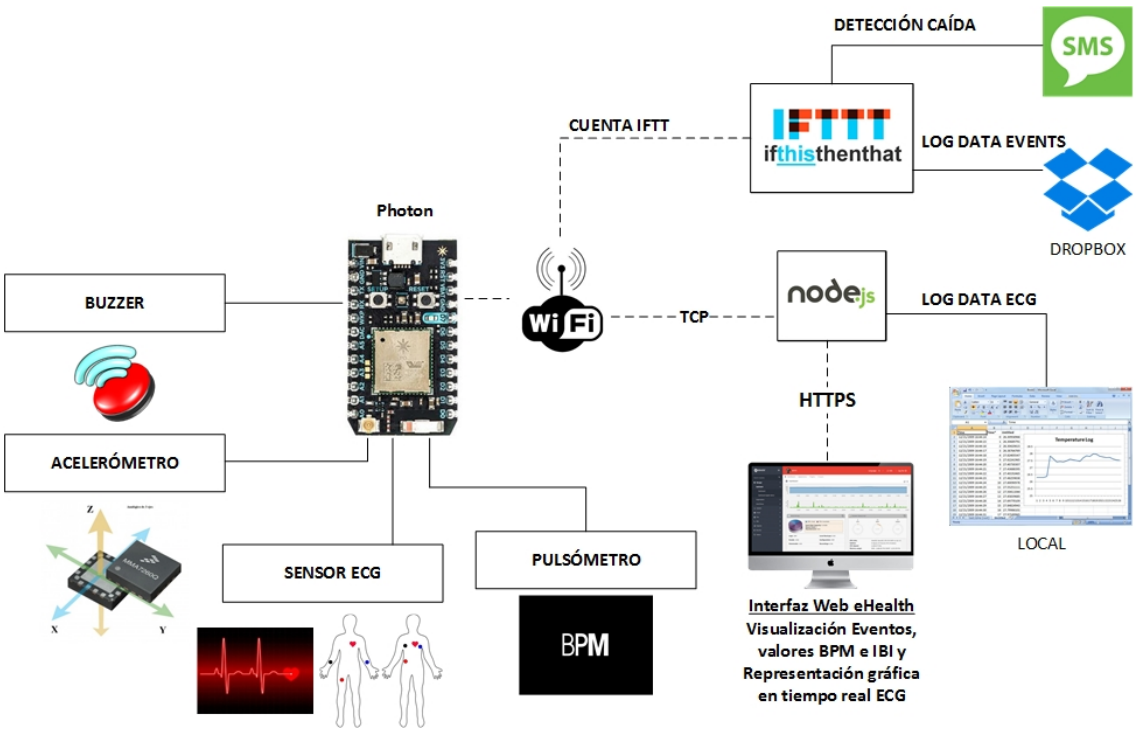


Figura 72. Diagrama de conexión final de la plataforma eHealth.

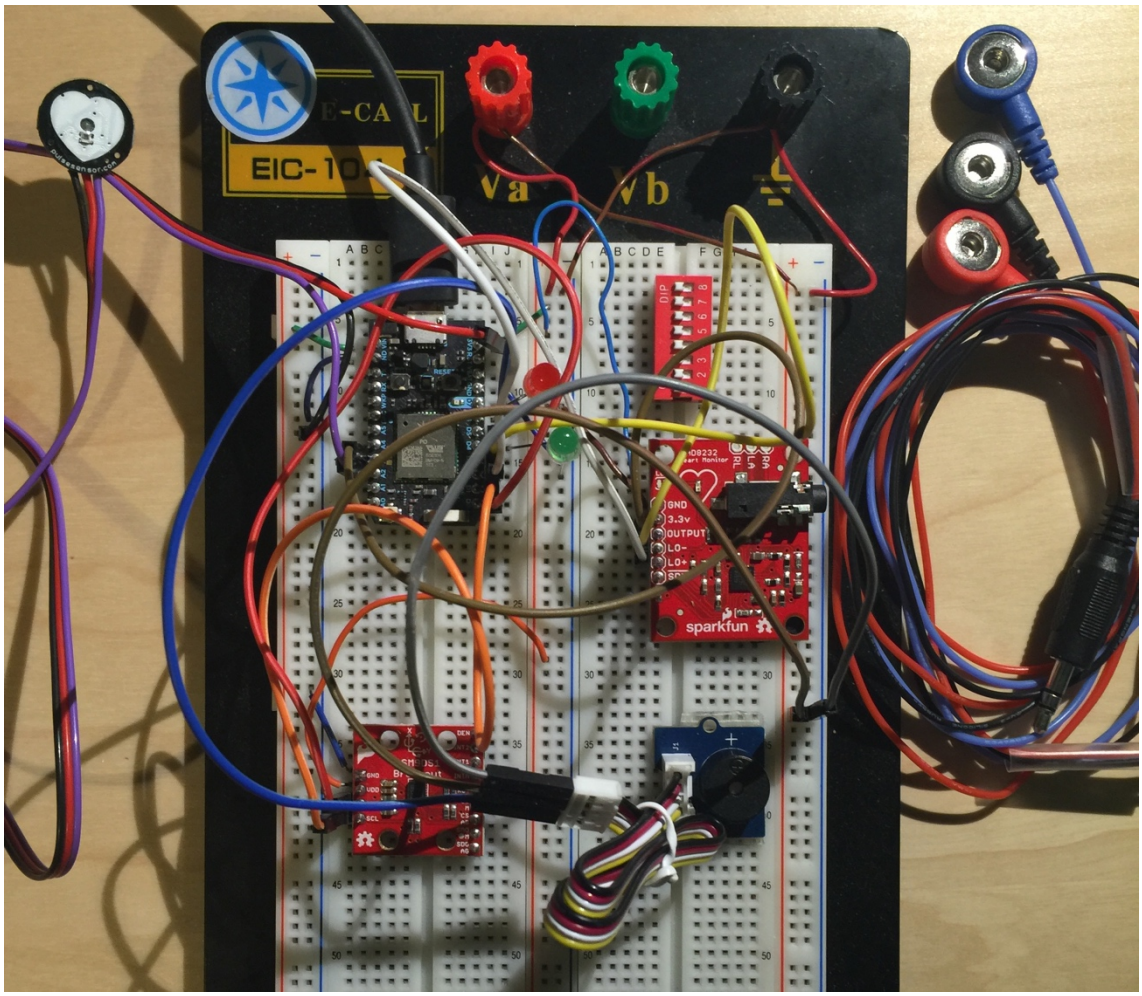


Figura 73. Foto de la protoboard con la interconexión de la plataforma final.

En primer lugar, para comenzar a usar la aplicación, el paciente deberá conectarse los *pads* o electrodos en las posiciones indicadas del cuerpo, siendo importante no moverse demasiado para obtener una señal correcta de ECG. El LED *D7* incorporado en el dispositivo Photon se encenderá cada vez que el algoritmo detecte un complejo QRS.

A continuación, esta señal se envía por TCP al servidor *Node.js* y se representa gráficamente en el navegador, almacenándose además todos estos datos localmente en un fichero para su posterior análisis. Por otra parte, el paciente deberá colocarse el pulsómetro en el dedo índice. Éste tiene un velcro con el que se puede fijar cómodamente. Así, en caso de detectar un pulso, se encenderá el LED verde que incorpora, y así se entrará en una dinámica hasta que lo retire del dedo, dando como resultado un pulso 0.

Para concluir, el último elemento de la plataforma es la detección de caídas. El paciente tendrá el dispositivo posicionado en su cintura y en caso de producirse una caída, el elemento *buzzer* actuará, generando un tono durante 10 segundos, con el fin de indicar al paciente que tiene que levantarse. En caso de que no pueda efectuar dicha acción, se encenderá el LED rojo durante 10 segundos, mostrándose visualmente en la plataforma de monitorización del navegador un aviso, además de enviarse un SMS a un número establecido y almacenarse esta incidencia en un archivo de *Dropbox*. En contraste con lo anterior, en caso de que se detecte una caída y el usuario se levante en el periodo indicado de 10 segundos, se generará una falsa alarma y se desactivarán, todos los elementos que indican una verdadera caída.

A continuación, se presentarán los resultados obtenidos de la plataforma *eHealth*, para comprobar su funcionalidad y correcto funcionamiento. Con este fin, se realizaron una serie de experimentos sobre la plataforma, que consistieron, principalmente, en efectuar los pasos asociados a su uso. Los resultados que se muestran a continuación se componen de capturas de pantallas, por lo que, si se desea verificar el funcionamiento del encendido de LEDs o el sonido emitido por el *buzzer*, se presenta un vídeo demostrativo contenido en el CD-ROM del presente TFG.

El primer paso es el de programar el dispositivo Photon, y por lo tanto enviar inalámbricamente el *firmware* final. Luego se ejecutará el servidor *Node.js* y se abrirá el navegador web en la dirección http://localhost:8080. En la Figura 74 se muestra el flujo de eventos generados a través de terminal ejecutando los comandos CLI.

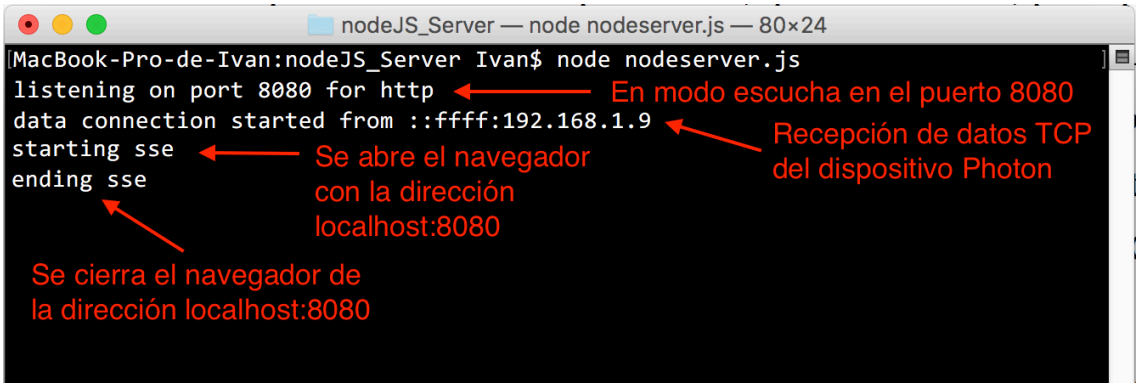


Figura 74. CLI Node.js Server

Posteriormente se accederá a la interfaz web diseñada, mostrada en la Figura 75. Para ver la representación gráfica del ECG se presiona la pestaña de pacientes *online*. Los resultados se muestran a continuación, una vez el paciente tenga conectados los electrodos y el pulsómetro.

Figura 75. Resultado 1 de la plataforma de monitorización eHealth.

Como se puede comprobar, los resultados obtenidos son adecuados y satisfactorios. Si el paciente se quita el pulsómetro y se deja conectado los electrodos, los valores del dispositivo "*Pulse Sensor*" se reinicializan a un valor 0, como se puede ver en la Figura 76.

Figura 76. Resultados 2 de la plataforma de monitorización eHealth.

Llegados a este punto, si el paciente se quita los electrodos, la interfaz gráfica generará un valor 0, al igual que en el resto de valores. Este punto puede verificarse en la Figura 77.

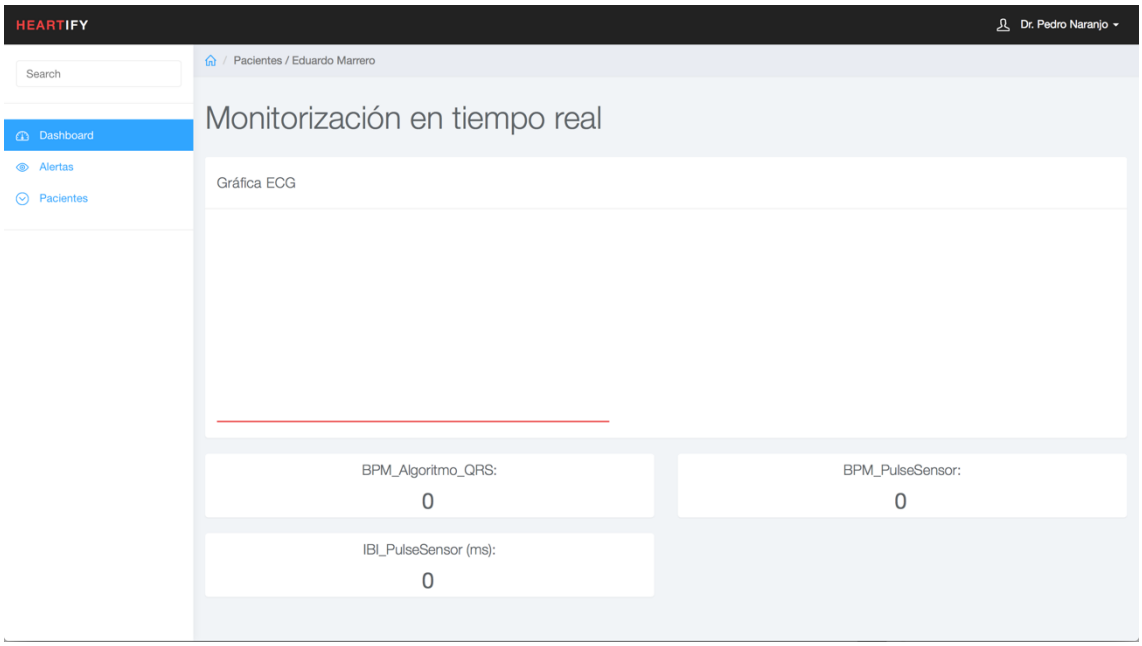


Figura 77. Resultados 3 de la plataforma de monitorización eHealth.

Finalmente se procederá a mostrar los resultados relativos a la detección de caídas. A modo de prueba, se dejará caer la *protoboard* con todos los componentes bien fijados sobre una superficie amortiguada como es, una cama. Los eventos generados vía puerto serie, representados en la Figura 78, ayudan a identificar el proceso del algoritmo, activándose en primer lugar el *trigger 3*, comenzando a sonar el *buzzer*, se dejó en posición horizontal para así generar el caso positivo de caída. Por otro lado, se generó un aviso en el navegador y en torno a unos 10 segundos después de la caída, un aviso por SMS. En la Figura 79 y en la Figura 80 se muestran los resultados obtenidos, a modo de simulación, con información ficticia.

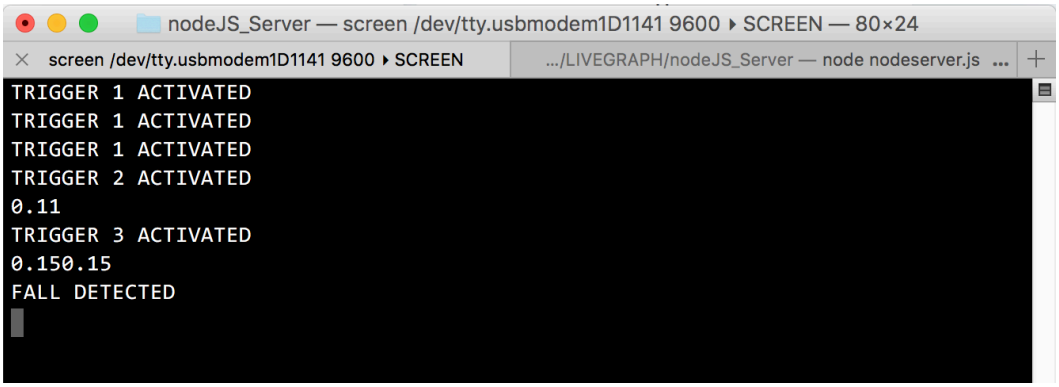


Figura 78. Estados del algoritmo de detección de caídas via puerto serie.

Figura 79. Visualización del evento de la detección de caída en el navegador web.

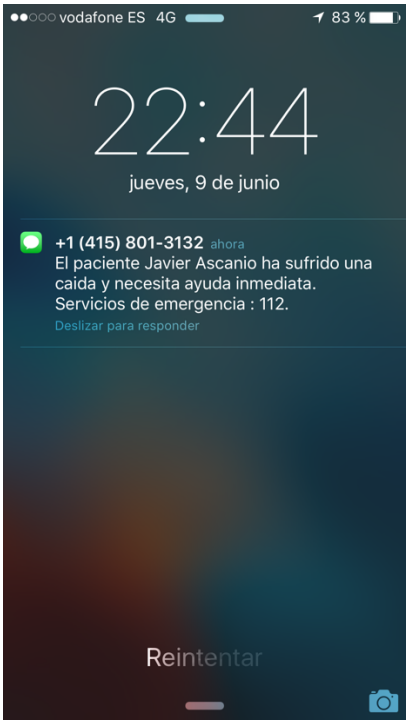


Figura 80. Recepción de SMS debido a una caída.

En caso de efectuarse una caída, y cambiar con posterioridad el paciente de orientación horizontal a vertical, se activará un falso positivo, que puede ser visualizado en la Figura 81.

Figura 81. Estados del algoritmo de detección de caídas vía puerto serie.

Llegados a este punto, se puede determinar que el funcionamiento de la plataforma HW/SW desarrollada es capaz de registrar y mostrar adecuadamente la señal ECG de un paciente de manera remota, además de detectar posibles caídas, y con ello satisfacer el objetivo fundamental del presente TFG.

Capítulo 6. Conclusiones y trabajos futuros

A lo largo de esta sección se presentan las conclusiones obtenidas tras el desarrollo del presente TFG. Además, también se incluye un apartado de líneas futuras para formular la implementación de nuevas características y mejoras en el prototipo de la plataforma actual.

6. 1 Conclusiones

Durante el presente Trabajo Fin de Grado, y en vista de los resultados favorables obtenidos, se ha conseguido el objetivo de crear una plataforma *eHealth* sencilla de coste reducido, con características desarrolladas en el ámbito de Internet de las Cosas (IoT). Este aplicativo es capaz de representar información biométrica en tiempo real y generar avisos en caso de detectar caídas de personas.

La intención del proyecto es que los profesionales puedan consultar los datos básicos de los pacientes, de manera que resulte cómodo, tanto por su parte como por la de los pacientes, ya que no tendrían que ir de manera reiterativa a la consulta, pudiendo reducirse los tiempos de espera, y lo que es aún más importante, los costes de personal y operaciones administrativas en Sanidad.

La aplicación desarrollada se ha implementado de manera que tenga la menor latencia en cuanto a la transmisión de datos por Internet, y en su conjunto es una plataforma ligera, con componentes de reducidas dimensiones, que hacen posible crear un producto final atractivo.

Gracias a la realización de este TFG se ha logrado profundizar un poco más en el ámbito de los sensores y sistemas empotrados. Como se ha podido comprobar en este trabajo, el dispositivo Photon ofrece muchas funcionalidades y su bajo coste es un factor que se valoró de manera significativa a la hora de elegir un dispositivo IoT.

A continuación, tras los resultados obtenidos al finalizar el desarrollo de la plataforma *eHealth*, se pueden establecer las siguientes conclusiones:

- El dispositivo Photon posee un amplio conjunto de características que lo hacen idóneo para este tipo de aplicaciones basadas en salud. No obstante, en la actualidad carece de capacidad para realizar conexiones seguras debido a la exposición de elementos identificativos por parte de *Particle Cloud* en una página web (*Access token*, *device ID*), a excepción del uso de *webhooks*, por lo que en general sería necesario disponer de un servidor para llevar a cabo este tipo de conexiones.

- A la hora de realizar las comunicaciones entre el dispositivo Photon y el servidor *Node.js* se decidió utilizar el protocolo TCP frente a UDP. Aunque UDP es más rápido, resulta menos fiable, ya que se podrían perder paquetes de datos biométricos durante su envío o éstos podrían llegar desordenados. Sin embargo, aunque en principio el hecho de que se perdieran algunos datos no haría que la aplicación dejara de funcionar, se decidió utilizar TCP por fiabilidad.

- El entorno de desarrollo IDE *Particle Dev* es una editor de texto avanzado con multitud de características, lo que facilita en gran medida las tareas de programación. Además, posee la capacidad de instalar paquetes que, por

ejemplo, realicen tareas de indentación, autocompletado, referencias en código, entre muchas más funcionalidades.

- La extensa cantidad de librerías desarrolladas, tanto por la empresa *Particle* como por terceros, resulta de mucha utilidad para generar, a partir de ellas, aplicaciones o proyectos más elaborados. El hecho de que el dispositivo Photon trabaje con *firmware* de código abierto como en el caso de Arduino, ha contribuido al rápido desarrollo de la programación de la aplicación propuesta.

- Los foros de *Particle* constituyen un gran apoyo, ya que se componen de una comunidad muy abundante y activa. Esto permitió resolver dudas específicas acerca de errores a lo largo del desarrollo de la presente aplicación por medio de publicación en los foros, o directamente por mensajes privados a usuarios, obteniendo respuestas muy rápidas y claras por parte del equipo de *Particle* (ingenieros, jefes del departamento de *firmware*, etc.) o de terceros.

Aunque el prototipo aquí presentado ofrece una funcionalidad acorde a las especificaciones inicialmente establecidas, no deja de ser un prototipo. Esto significa que queda recorrido de cara a presentar un producto final. Para facilitar dicho recorrido, se presenta a continuación un conjunto de mejoras y oportunidades de expansión del prototipo.

6. 2 Líneas futuras

Con este apartado se pretenden plantear nuevas vías que tengan como punto de partida el presente TFG. También se pretende indicar cuáles son los puntos a mejorar en el diseño con el fin de obtener unos resultados óptimos. Por lo tanto, se abren diversas áreas para el desarrollo futuro. A continuación se muestran las líneas futuras más destacadas que se plantean tras el desarrollo del presente TFG :

- Integración de electrodos inalámbricos que eliminen la conexión requerida por medio de un cable analógico tipo *jack*, facilitando la comodidad en los usuarios.
- Integración de todas las funcionalidades simuladas en la página web para que un usuario haga uso de ellas por medio de bases de datos remota, así como la detección automática de posibles anomalías y aspectos más específicos por parte del servidor, que den lugar a un aplicativo más profesional en el campo de la cardiología.

- Implementación de metodologías para establecer conexiones seguras con el dispositivo Photon y garantizar así la privacidad y seguridad de los datos biométricos de los usuarios.

- Inclusión de un micrófono de manera que el usuario pueda comunicarse con sus contactos en caso de emergencia, por medio, por ejemplo, de voz IP.

- Inclusión de batería externa para su portabilidad y la posibilidad de reducir el tamaño del conjunto por medio del desarrollo de un único circuito integrado o PCB.

- Ejecución de pruebas en condiciones reales para la detección de caídas y la integración de todas las funcionalidades que ofrece el dispositivo LSM9DS1, con el objetivo de mejorar el algoritmo desarrollado, con nuevas categorías físicas de condiciones cotidianas, entre otras.

- Integración con la programación de iOS/Android para permitir usar las funcionalidades de los *smartphones* por medio de aplicaciones.

Bibliografía

-

Relación de bibliografía consultada para la elaboración del presente Trabajo Fin de Grado. Incluye los documentos de las especificaciones de los componentes electrónicos utilizados, así como una relación de páginas web relacionadas con la materia.

[1] Internet of Things Global Standards Initiative.(n.d.).Available: http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx. Last Access: May, 2016.

[2] Evans, D. (2011). Internet de las cosas Internet de las cosas Cómo la próxima evolución de Internet lo cambia todo. *Cisco Internet Bussiness Solutions Group - IBSG*,41,1.Available:https://www.cisco.com/web/ES/assets/executives/pdf/Internet_of_Things_IoT_IBSG_0411FINAL.pdf . Last Access: May, 2016.

[3] Iot, O. (2015). The 10 most popular Internet of Things applications right now The Internet of Things applications ranking. From http://iot-analytics.com/10-internet-of-things-applications/. Last Access: May, 2016.

[4] Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. Computer, Networks, 54(15), 2787–2805. Last Access: May, 2016. http://doi.org/10.1016/j.comnet.2010.05.010

[5] Internet of Things: 10 Most Innovative Companies | WT VOX. (2015). Retrieved from https://wtvox.com/internet-of-things-iot/internet-of-things-10-innovative-companies/. Last Access: May, 2016.

[6] "Anatomia del Corazón" @ www.texasheart.org. (2015). Retrieved from http://www.texasheart.org/HIC/Anatomy_Esp/anato_sp.cfm . Last Access: May, 2016.

[7] "Que es el Corazón." @ www.youbioit.com (2015). Retrieved from http://www.youbioit.com/es/article/shared-information/24402/que-es-el-corazon. Last Access: May, 2016.

[8] "Photon Datasheet" @ docs.particle.io. (2016). Retrieved from https://docs.particle.io/datasheets/photon-datasheet/. Last Access: May, 2016.

[9] "Spark Core Datasheet" @ docs.particle.io. (n.d.). Retrieved from https://docs.particle.io/datasheets/core-datasheet/. Last Access: May, 2016

[10] Monk, Simon, (2015), Getting Started with the Photon: Making Things with the Affordable, Compact, Hackable WiFi Module. Maker Media Inc. ISBN 1457186993, 9781457186998.

-

[11] Sheet, D. (2013). Single-Lead , Heart Rate Monitor Front End. Last Access: 1 June, 2016
[12] Potencial_de_reducción @ es.wikipedia.org. (n.d.). Retrieved from https://es.wikipedia.org/wiki/Potencial_de_reducción#Potenciales_de_semicelda Last Access: 1 June, 2016.
[13] Fifo, E., & Lsm, T. (2013). LSM9DS0 Datasheet. iNEMO inertial module: (August), 1–74. Last Access: 2 June, 2016.
[14] "PulseSensor" @ pulsesensor.com. (2016). Retrieved from http://pulsesensor.com/ . Last Access: 3 June, 2016.
[15] Bazaar, B. (2015). Grove – Buzzer User Manual. Last Access: 2 June, 2016.
[16] "El bus I2C"@ www.prometec.net. (n.d.). Retrieved from http://www.prometec.net/bus-i2c/ . Last Access: 2 June, 2016.
[17] I2C @ www.electroensaimada.com. (n.d.). Retrieved from http://www.electroensaimada.com/i2c.html . Last Access: 2 June, 2016.
[18] "Wiring" @ wiring.org.co. (2016). Retrieved from http://wiring.org.co/ Last Access: 4 June, 2016.
[19] "Tinker & Mobil App" @ docs.particle.io. (n.d.). Retrieved from https://docs.particle.io/guide/getting-started/tinker/core/. Last Access: 4 June, 2016.
[20] "Particle dev" @ www.particle.io. (2016). Retrieved from https://www.particle.io/dev Last Access: 7 June, 2016.
[21] "Particle prototype" @ www.particle.io. (2016). Retrieved from https://www.particle.io/prototype Last Access: 7 June, 2016.
[22] Index @ Processing.Org. (n.d.). Retrieved from http://processing.org/ . Last Access: 4 June, 2016.
[23] Heart_Rate_Display @ github.com. (n.d.). Retrieved from https://github.com/sparkfun/AD8232_Heart_Rate_Monitor/tree/master/Software/Heart_Rate_Display_Processing/Heart_Rate_Display. Last Access: 4 June, 2016.
[24] SparkIntervalTimer @ github.com. (n.d.). Retrieved from https://github.com/pkourany/SparkIntervalTimer Last Access: 4 June, 2016.
[25] HC Chen, SW Chen, "A Moving Average based Filtering System with its Application to Real-time QRS Detection", Computers in Cardiology, 2003. Last Access: 6 June, 2016.
[26] real_time_QRS_detection @ github.com. (n.d.). Retrieved from https://github.com/blakeMilner/real_time_QRS_detection. Last Access: 6 June, 2016.
[27] Liu, J., & Lockhart, T. E. (2014). Development and evaluation of a prior-to-impact fall event detection algorithm. IEEE Transactions on Biomedical Engineering, 61(7), 2135–2140. http://doi.org/10.1109/TBME.2014.2315784 . Last Access: 6 June, 2016.
[28] Kangas, M., Konttila, A., Lindgren, P., Winblad, I., & Jämsä, T. (2016). Comparison of low-complexity fall detection algorithms for body attached accelerometers. Gait & Posture, 28(2), 285–291. http://doi.org/10.1016/j.gaitpost.2008.01.003. Last Access: 6 June, 2016.
[29] Bourke, A. K., & Lyons, G. M. (2008). A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. Medical Engineering and Physics, 30(1), 84–90. http://doi.org/10.1016/j.medengphy.2006.12.001 Last Access: 6 June, 2016.

[30] nodejs_introduction @ www.tutorialspoint.com. (n.d.). Retrieved from http://www.tutorialspoint.com/nodejs/nodejs_introduction.htm Last Access: 7 June, 2016.
[31] "Stream Updates with Server-Sent Events" @ www.html5rocks.com. (n.d.). Retrieved from http://www.html5rocks.com/en/tutorials/eventsource/basics/ Last Access: 8 June, 2016.
[32] Fritzing @ es.wikipedia.org. (n.d.). Retrieved from https://es.wikipedia.org/wiki/Fritzing Last Access: 9 June, 2016.
[33] getting-started-with-thingspeak @ es.mathworks.com. (n.d.). Retrieved from https://es.mathworks.com/help/thingspeak/getting-started-with-thingspeak.html Last Access: 8 June, 2016.
[34] "Blynk" @ www.blynk.cc. (n.d.). Retrieved from http://www.blynk.cc/. Last Access: 8 June, 2016.
[35] "introduccion a ubidots" @ ubidots.com. (n.d.). Retrieved from http://ubidots.com/docs/es/get_started/introduccion.html Last Access: 8 June, 2016.
[36] "Particle Device Firmware" @ docs.particle.io. (n.d.). Retrieved from https://docs.particle.io/reference/firmware/photon/ Last Access: 9 June, 2016.

Pliego de condiciones

-

El Pliego de Condiciones expone las condiciones bajo las que se ha desarrollado el presente Trabajo Fin de Grado. A continuación, se muestran el conjunto de herramientas *hardware, software* y *firmware* empleadas durante su realización.

PL. 1 Condiciones Hardware

En la Tabla PL.P- 1 se presentan los equipos *hardware* usados.

Tabla PL.P- 1. Relación de equipos hardware.

Equipo	Modelo	Fabricante
Ordenador Portátil	MacBook Pro (Retina, 15-inch, Early 2013)	Apple
Kit desarrollo Photon	WRL-13774	Particle
Sensor AD8232 "Heart Rate Monitor"	SEN-12650	SparkFun Electronics
Cable audio Jack conexión electrodos ECG	CAB-12970	SparkFun Electronics
Electrodos de ECG	SEN-12969	SparkFun Electronics
Pulsómetro	SEN-11574	SparkFun Electronics
Sensor LSM9DS1 "Breakout"	SEN-13284	SparkFun Electronics
Buzzer	SKU 107020000	Seedstudio
LED	COM-09590	SparkFun Electronics

-

Protoboard	104-1	E-Call Enterprise
Jumpers conexión Protoboard	Conexión machp-macho y hembra-macho	Lopacan

PL. 2 Condiciones Software

En la Tabla PL.P- 2 se exponen las herramientas *software* utilizadas, especificando su versión.

Tabla PL.P- 2. Relación de herramientas software.

Software	Versión	Fabricante
Sistema Operativo Mac	OS X El Capitan	Apple
Sistema Operativo Windows	Microsoft Windows 10	Microsoft
Microsoft Office para Mac	15	Microsoft
Atom Particle Dev	1.7.3	Atom
Microsoft Visio Professional 2013	13	Microsoft
Mendeley desktop version	1.16.1	Medeley Ltd.
Fritzing	0.9.3	Fritzing Org
Processing	3.0.2	Processing Foundation

PL. 3 Condiciones firmware

Para concluir, se muestra el *firmware* utilizado, y su versión, en la Tabla PL.P- 3.

Tabla PL.P- 3. Relación de firmware.

Firmware	Versión
Dispositivo Photon	v0.5.1
Node.js	v6.2.0
Particle CLI	v1.12.0
Canvas	v 4.8.5.1

Presupuesto

-

Este capítulo contiene el presupuesto que recoge los gastos generados durante el desarrollo del presente Trabajo Fin de Grado. Una vez analizados cada uno de los criterios establecidos, se aplicarán los impuestos vigentes y se procederá a la obtención del coste total del TFG.

P. 1 Trabajo tarifado por tiempo empleado

Este concepto contabiliza los gastos que corresponden a la mano de obra, según el salario correspondiente a la hora de trabajo de un Ingeniero Técnico de Telecomunicación. Se propone utilizar la siguiente fórmula:

$$H = C_t \times 74,88 \times H_n + C_t \times 96,72 \times H_e\,, \qquad \text{(P.1)}$$

donde:

- H representa los honorarios totales por el tiempo dedicado.
- H_n detalla las horas normales trabajadas dentro de la jornada laboral.
- H_e especifica las horas especiales trabajadas.
- C_t es un factor de corrección función del número de horas trabajadas.

Para la realización del presente TFG se han invertido un total de 300 horas. Todas ellas se han realizado dentro del horario normal, por lo que el número de horas especiales es cero. Además, de acuerdo a lo establecido por el Colegio Oficial de Ingenieros Técnicos de Telecomunicación (COITT), el factor de corrección C_t a aplicar para 300 horas trabajadas es de 0,60, tal y como se puede comprobar en la Tabla P-1.

Tabla P-1. Coeficientes reductores para trabajo tarifado (COITT)

Horas	Factor de corrección
Hasta 36	1,00
Exceso de 36 hasta 72	0,90
Exceso de 72 hasta 108	0,80
Exceso de 108 hasta 144	0,70
Exceso de 144 hasta 180	0,65
Exceso de 180 hasta 360	**0,60**
Exceso de 360 hasta 510	0,55
Exceso de 510 hasta 720	0,50
Exceso de 720 hasta 1080	0,45
Exceso de 1800	0,40

Teniendo en cuenta estos datos, el coste total de honorarios asciende a:

$$H = 0,6 \times 74,88 \times 300 + 0,6 \times 96,72 \times 0 = 13.478,40\,€ \qquad \text{(P.2)}$$

Por lo tanto, el trabajo tarifado por tiempo empleado asciende a la cantidad de *trece mil cuatrocientos setenta y ocho euros con cuarenta céntimos*.

P. 2 Amortización del inmovilizado material

En el inmovilizado material se consideran, tanto los recursos hardware como software empleados para la realización de este TFG.

Para estipular el coste de amortización en un periodo de 3 años se utiliza un sistema de amortización lineal, en el que se supone que el inmovilizado material se deprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula de la siguiente forma:

$$Cuota\ anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Número\ de\ años\ de\ vida\ útil} \quad \text{(P.3)}$$

Donde el valor residual se corresponde con el valor teórico que se supone que tendrá el elemento en cuestión después de su vida útil.

P.2.1 Amortización del material hardware

Debido a que la duración de este Trabajo Fin de Grado es de tan solo 4 meses, siendo este periodo muy inferior al de 3 años estipulado para el coste de amortización, los costes se calcularán en base a los derivados de los primeros 4 meses.

En la Tabla P-2 se especifica el *hardware* amortizable necesario para la realización del trabajo, indicando su valor de adquisición y su amortización, teniendo en cuenta un tiempo de uso de 4 meses.

Tabla P-2. Costes y amortización del hardware (I)

Elemento	Valor de adquisición	Amortización
Ordenador portátil MacBook Pro (Retina, 15-inch, Early 2013)	2000,00 €	222,22 €
Total	**2000,00 €**	**222,78 €**

Por otro lado, en la Tabla P-3 se muestra el resto de material *hardware* utilizado y para el que, debido a su bajo precio, su amortización coincide con su valor de adquisición.

Tabla P-3. Costes y amortización del hardware (II)

Elemento	Valor de adquisición	Amortización
Kit desarrollo Photon	26,77 €	26,77 €
Sensor AD8232 "Heart Rate Monitor"	17,62 €	17,62 €
Cable audio Jack conexión electrodos ECG	4,38 €	4,38 €
Electrodos de ECG	7,04 €	7,04 €
Pulsómetro	22,09 €	22,09 €
Sensor LSM9DS1 "Breakout"	22,09 €	22,09 €
Buzzer	1,68 €	1,68 €
LED	0,62 €	0,62 €
Protoboard	37,00 €	37,00 €
Total	**139,29 €**	**139,29 €**

Finalmente, tras realizar la suma de ambos se obtiene el coste total del material hardware, tal y como se muestra en la Tabla P-4.

Tabla P-4. Costes y amortizaciones totales del hardware

Concepto	Coste
Costes y amortización del *hardware* (I)	222,78 €
Costes y amortización del *hardware* (II)	139,29 €
Total	**362,07 €**

El coste total del material *hardware* asciende a *trescientos sesenta y dos euros con siete céntimos*.

P.2.2 Amortización del software

Para el cálculo de los costes de amortización del material *software* se considerarán, al igual que con el material hardware, los costes derivados de los primeros 4 meses.

La Tabla P-5 muestra los elementos *software* necesarios para la realización del trabajo, así como su valor de adquisición y su amortización.

Tabla P-5. Costes y amortización del software

Elemento	Valor de adquisición	Amortización
Sistema Operativo Mac	0,00 €	0,00 €
Sistema Operativo Windows	0,00 € (*)	0,00 € (*)
Microsoft Office para Mac	0,00 € (*)	0,00 € (*)
Atom Particle Dev	0,00 €	0,00 €
Microsoft Visio Professional 2013	0,00 € (*)	0,00 € (*)
Mendeley desktop version	0,00 €	0,00 €
Total	**0,00 €**	**0,00 €**

(*) *Licencia de uso proporcionada por la ULPGC.*

Por tanto, el coste total del material software asciende a la cantidad de *cero euros*.

P. 3 Redacción del trabajo

Se ha utilizado (P.4) para determinar el coste asociado a la redacción de la memoria del presente Trabajo Fin de Grado.

$$R = 0,\ 07 \times P \times C_n\ , \qquad \text{(P.4)}$$

donde:

- R son los honorarios por la redacción del trabajo.
- P es el presupuesto.
- C_n es el coeficiente de ponderación en función del presupuesto.

El valor del presupuesto *P* se calcula sumando los costes del trabajo tarifado por tiempo empleado y de la amortización del inmovilizado material, tanto *hardware* como *software*. El resultado de los costes se muestra en la Tabla P-6.

Tabla P-6. Presupuesto, incluyendo trabajo tarifado y amortización del inmovilizado material

Concepto	**Coste**
Trabajo tarifado por tiempo empleado	13.478,40 €
Amortización del material *hardware*	362,07 €
Amortización del *software*	0,00 €
Total	**13.840,47 €**

Como el coeficiente de ponderación C_n para presupuestos menores de 30.050,00€ viene definido por el COITT con un valor de 1.00, el coste derivado de la redacción del Trabajo Fin de Grado es de:

$$R = 0,07 \times 13.840,47 \times 1 = 968,83€ \quad \text{(P.5)}$$

Ascendiendo de esta forma el coste de la redacción del trabajo a *novecientos sesenta y ocho euros con ochenta y tres céntimos*.

P. 4 Derechos de visado del COITT

El COITT establece que para proyectos técnicos de carácter general, los derechos de visado para 2016 se calculan en base a (P.6).

$$V = 0,006 \times P_1 \times C_1 + 0,003 \times P_2 \times C_2\,, \quad \text{(P.6)}$$

donde:

- V es el coste de visado del trabajo.
- P_1 es el presupuesto del proyecto.
- C_1 es el coeficiente reductor en función del presupuesto.

- P_2 es el presupuesto de ejecución material correspondiente a la obra civil.
- C_2 es el coeficiente reductor en función a P_2.

El valor del presupuesto P_1 se halla sumando los costes de las secciones correspondientes al trabajo tarifado por tiempo empleado, a la amortización del inmovilizado material y a la redacción del documento. Esta suma se muestra en la Tabla P-7. Al igual que en el caso anterior, el coeficiente C_1 para proyectos de presupuesto inferior a 30.050,00€ es de 1,00, asimismo el valor de P_2 es de 0,00€, ya que no se realiza ninguna obra.

Tabla P-7. Presupuesto, incluyendo trabajo tarifado, amortización y redacción del trabajo

Concepto	**Coste**
Trabajo tarifado por tiempo empleado	13.478,40 €
Amortización del material hardware	362,07 €
Amortización del software	0,00 €
Redacción del trabajo	968,83€
Total	**14.809,30 €**

De esta forma, aplicando a (P.6) los datos de la tabla P-7 y el coeficiente especificado se obtiene:

$$V = 0{,}006 \times 14.809{,}30 \times 1 = 88{,}85\ € \qquad \text{(P.7)}$$

Los costes por derechos de visado del presupuesto ascienden a *ochenta y ocho euros con ochenta y cinco céntimos*.

P. 5 Gastos de tramitación y envío

Los gastos de tramitación y envió están estipulados en seis euros (6,00€) por cada documento visado de forma telemática.

P. 6 Material fungible

Además de los recursos *hardware* y *software*, en este trabajo se han empleado otros materiales, como los folios y el tóner de la impresora entre otros, que quedan englobados como material fungible.

En la Tabla P-8 se muestran los costes derivados de estos recursos.

Tabla P-8. Costes de material fungible

Concepto	Coste
Folios	10,00 €
Tóner de la impresora	30,00 €
Encuadernado	4,00 €
Tres CDs	6,00 €
Total	**50,00 €**

Los costes de material fungible ascienden a *cincuenta euros*.

P. 7 Aplicación de impuestos y coste total

La realización del presente TFG está gravada por el Impuesto General Indirecto Canario (IGIC) en un siete por ciento (7 %). En la Tabla P-9 se muestra el presupuesto final con los impuestos aplicados.

Tabla P-9. Presupuesto total del Trabajo Fin de Grado

Concepto	Coste
Trabajo tarifado por tiempo empleado	13.478,40 €
Amortización del material hardware	362,07 €
Amortización del software	0,00 €
Redacción del trabajo	968,83€

Derechos de visado del COITT	88,85 €
Gastos de tramitación y envío	6,00 €
Costes de material fungible	50,00 €
Total (Sin IGIC)	**14.954,15 €**
IGIC (7%)	1.046,79 €
Total	**16.000,94 €**

El presupuesto total del Trabajo Fin de Grado "*Plataforma de monitorización eHealth a través del dispositivo Photon en el ámbito IoT*" asciende a *dieciséis mil euros con noventa y cuatro céntimos*.

Fdo.: D. Iván J. Santana Sosa

En Las Palmas de Gran Canaria a 10 de Junio de 2016

Anexo

Contenido del CD

En este Anexo se presenta el contenido del CD adjunto a este documento, el cual está formado por las siguientes partes:

- En el directorio raíz se encuentra el fichero `Memoria_TFG_final.pdf` con la memoria del TFG "Plataforma de monitorización *eHealth* a través del dispositivo Photon en el ámbito *IoT*" en formato PDF.

- En el directorio `eHealth_Platform_v4` se encuentra el Código fuente desarrollado para la gestión de la aplicación *eHealth* en el dispositivo Photon utilizando lenguaje *Wiring*.

- En el directorio `nodeJS_Server` se encuentran los ficheros del servidor *node.JS* y del navegador web.

- En el directorio `Vídeo_Demostrativo` se incluyen dos vídeos que muestran el funcionamiento de la plataforma desarrollada.