Proyecto Fin de Carrera
Ingeniería Industrial

Sistema para monitorización del ritmo cardíaco mediante Smartphone

Autor: Andrés Manuel Cotorruelo Jiménez

Tutor: Daniel Rodríguez Ramírez

Dep. Sistemas y automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Carrera
Ingeniería Industrial

Sistema para monitorización del ritmo cardíaco mediante Smartphone

Autor:

Andrés Manuel Cotorruelo Jiménez

Tutor:

Daniel Rodríguez Ramírez

Profesor titular

Dep. de Teoría de la Señal y Comunicaciones

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Carrera: Sistema para monitorización del ritmo cardíaco mediante Smartphone

Autor: Andrés Manuel Cotorruelo Jiménez

Tutor: Daniel Rodríguez Ramírez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A Eloísa Sánchez Herrera, que probó la falsedad del dicho "madre no hay más que una"

Agradecimientos

Aprovecho estas líneas para agradecerles a aquellas personas cuyo incesante apoyo ha hecho de guía en estos años de carrera. Me gustaría hacer especial mención a mi abuela, que, aunque la dedicatoria del proyecto es en su nombre, jamás habrá un número de secciones de dedicatorias ni agradecimientos que le hagan justicia a la labor que ha desarrollado conmigo. A mi familia, que ante viento y marea me han apoyado. A mis amigos, junto a los cuales he disfrutado y sin duda disfrutaré en el futuro. A las personas especiales de mi vida, aquellas sin las cuales, no puedo relatar mi historia. No me gustaría concluir esta sección sin hacer especial mención a mi tutor, sin la ayuda del cual no podría haber concluido este proyecto, y que ha sobrepasado con creces las labores de tutor, cayendo más bien en la categoría de amigo. A todos y cada uno de vosotros: muchas gracias.

Resumen

Existen en el mundo una gran cantidad de personas que, desgraciadamente, sufren enfermedades cardiovasculares y se beneficiarían de una monitorización continua de sus constantes cardíacas. Este proyecto pretende presentar una solución preliminar haciendo uso de los actualmente generalmente extendidos *Smartphone* y de placas basadas en microcontroladores para tal fin. También abre una línea de posibles ampliaciones, todas ellas enfocadas al bienestar cardíaco del usuario.

.

Abstract

In the world, there are a vast amount of people who, unfortunately, suffer from cardiovascular diseases and could benefit from a continuous monitorization of their cardiac constants. This project intends to offer a solution using the currently globally extended *smartphones* and microcontroller-based boards for such purpose. It also opens up a range of possible extensions, all of them focused on the user's well-being.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Este proyecto surge de una necesidad real: por desgracia, existen una multitud de personas con enfermedades cardiacas graves. Su condición les hace candidatos de ser los receptores de un implante conocido como ICD, que, como se explicará más adelante, aplicará descargas eléctricas directamente al corazón. Estas descargas eléctricas, en ocasiones pueden dejar al paciente inconsciente. El objetivo del proyecto es desarrollar un dispositivo capaz de detectar anomalías en el ritmo cardíaco y avisar a las autoridades sanitarias. Esto es posible gracias a la popularidad de los *smartphones*, que aúnan en un solo dispositivo funcionalidades necesarias para el desarrollo de este proyecto, a saber: red telefónica, conexión GPS y comunicación por Bluetooth. Estas tres características se aprovecharán para proveer al usuario de un método automático de aviso en caso de alerta.

1.1 ICDs

Un desfibrilador cardioversor implantable (ICD) o un desfibrilador cardioversor implantable automático (AICD) es un dispositivo que se implanta en el interior del cuerpo, capaz de realizar cardioversión, desfibrilación y (en las versiones más modernas) marcar el paso del corazón. En consecuencia, el dispositivo es capaz de corregir la mayoría de arritmias peligrosas para la vida. El ICD es el tratamiento de primera línea y la terapia preventiva para pacientes con riesgo de sufrir muerte súbita por fibrilación ventricular y taquicardia ventricular. Los dispositivos modernos pueden ser programados para detectar ritmos cardíacos anormales y dar terapia a través de un marcado de pasos antitaquicárdico programable además de descargas de alta y baja energía.

Figura 1: ICD [1]

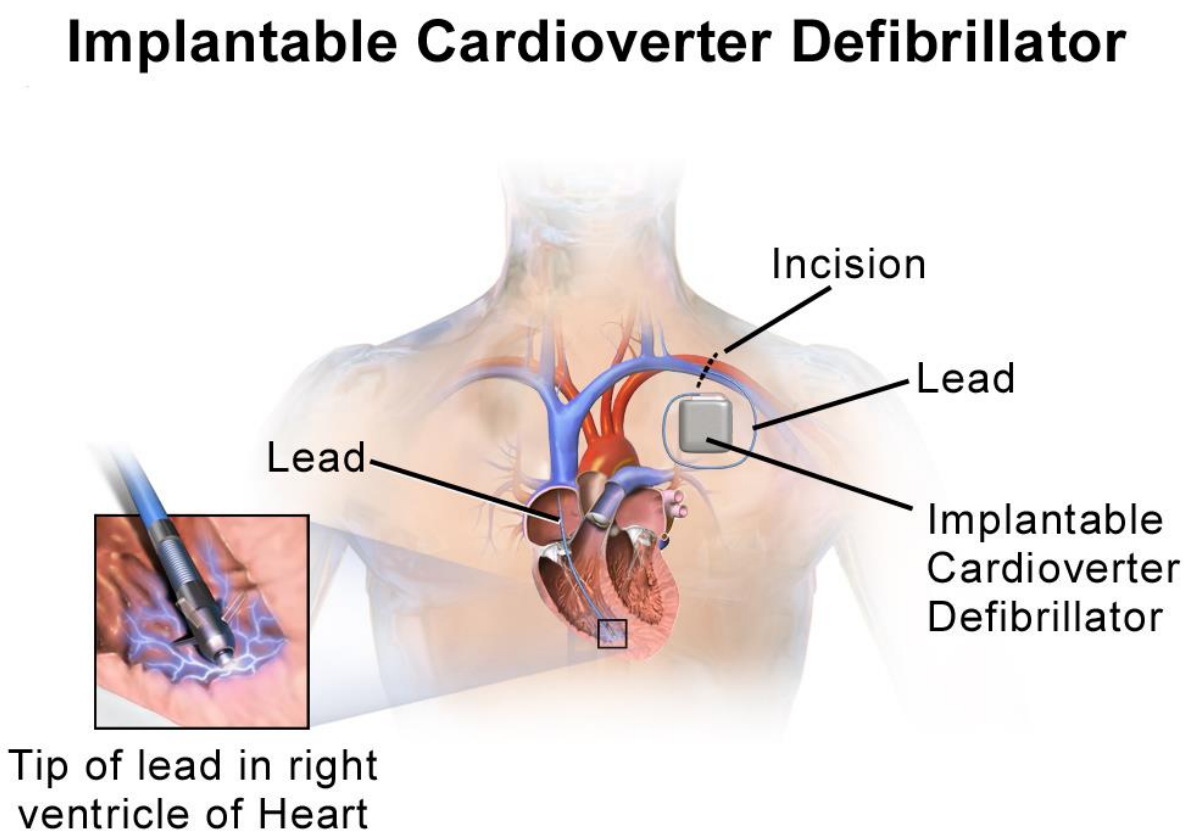


Figura 2: Colocación de un ICD [2]

Las baterías actuales duran entre 6 y 10 años. Con avances en la tecnología (baterías con más capacidad o potencialmente, baterías recargables) puede ser posible incrementar la duración hasta bien pasados los 10 años. El cable que conecta el dispositivo al corazón tiene una longevidad media mucho mayor, pero puede incurrir en varios tipos de fallos, específicamente fallos en el aislamiento o ruptura del conductor, requiriendo así un reemplazo de dicho cable.

El proceso de implantación de un sistema ICD es similar a la implantación de un marcapasos. Los ICDs están compuestos de un generador ICD y de cables. El primer componente, o generador, contiene un chip con memoria RAM, software programable, un condensador y una batería; la cual está normalmente implantada bajo la piel de la parte izquierda del pecho. La segunda parte del sistema es un cable o cables que, de forma similar a los marcapasos, se conectan al generador y se pasan a través de una vena a las cámaras derechas del corazón. El cable, generalmente, se aloja en el ápice o septum del ventrículo derecho. Igual que los marcapasos, los ICDs pueden tener un solo cable en el corazón (en el ventrículo derecho, ICD de una sola cámara), dos cables (en la aurícula derecha y ventrículo derecho, ICD de cámara doble) o tres cables (ICD biventricular, uno en la aurícula derecha, otra en el ventrículo derecho y otra en la pared exterior del ventrículo izquierdo).

La diferencia entre marcapasos e ICDs es que los marcapasos también están disponibles como unidades temporales y, generalmente, están diseñados para corregir ritmos cardiacos bajos (bradicardia) mientras que los ICDs son a menudo salvaguardas permanentes contra arritmias repentinas peligrosas para la vida.

El avance más reciente en este campo es el ICD subcutáneo (S-ICD).

La implantación de un ICD está pensada para prevenir la muerte cardiaca repentina y está indicada bajo varias condiciones. Dos categorías amplias pero distintas son la prevención primaria y secundaria. La prevención primaria se atribuye a pacientes que no han sufrido un episodio peligroso de arritmia. La prevención secundaria es la que presenta más beneficios, ya que se atribuye a los supervivientes de infarto derivado de fibrilación ventricular o taquicardia ventricular prolongada hemodinámicamente inestable, después de que las causas reversibles hayan sido excluidas. Similarmente, los ICDs se usan en prevención primaria para prevenir la muerte cardiaca en pacientes que tienen riesgo de taquicardia ventricular prolongada o fibrilación ventricular. Estos son la gran mayoría de casos en los que se implanta un ICD. Existe una multitud de directrices para el uso de ICDs en prevención primaria. Tanto el Colegio Americano de Cardiología (ACC), la Asociación Americana del Corazón (AHA) y la Sociedad Europea de Cardiología (ESC) proporcionan actualizaciones de estas directrices. Algunas de las directrices de clase I son las siguientes:

- Pacientes con LVEF (Fracción de eyección del ventrículo izquierdo) ≤ 35% debido a un infarto de miocardio previo, pasados al menos 40 días desde dicho infarto y están en la clase funcional NYHA II o III.
- Pacientes con disfunción en el ventrículo izquierdo debido a un infarto de miocardio, pasados al menos 40 días de dicho infarto, tienen un LVEF ≤ 30%, y están en la clase funcional NYHA I.
- Pacientes con cardiomiopatía no isquémica, con un LVEF ≤ 35% y que están en la clase funcional NYHA II o III
- Pacientes con taquicardia ventricular no prolongada debido a un infarto de miocardio previo, con LVEF ≤ 40%, con fibrilación ventricular inducible o taquicardia ventricular prolongada en un estudio electrofisiológico.
- Pacientes con enfermedades estructurales del corazón y taquicardias ventriculares prolongadas espontáneas, hemodinámicamente estables o inestables.
- Pacientes con un síncope de origen indeterminado, con taquicardia ventricular o fibrilación ventricular clínicamente relevante, así como hemodinámicamente significante inducidas en un estudio electrofisiológico. [3]

1.2 Electrocardiografía

La electrocardiografía (ECG o EKG) es la medida de la actividad cardíaca en un período de tiempo mediante unos electrodos dispuestos en puntos concretos del cuerpo. Este proceso no invasivo recoge los impulsos eléctricos generados por el corazón en su polarización y despolarización y los representa en una gráfica. A dicha representación se le llama electrocardiograma (también ECG o EKG). Tradicionalmente, esta representación se llevaba a cabo mediante un rollo de papel continuo y unas agujas que iban escribiendo sobre él, aunque actualmente los datos se representan de forma digital. El resultado de dicha representación es una onda bastante característica, en la que se pueden distinguir distintas secciones:

- Onda P: generada durante la despolarización auricular.
- Segmento PR: no se produce contracción en el corazón, los impulsos eléctricos viajan desde el fascículo atrioventricular hacia los ventrículos.
- Complejo QRS: producido por la rápida despolarización de los ventrículos.
- Segmento ST: sin actividad eléctrica.
- Onda T: indica la repolarización de los ventrículos.

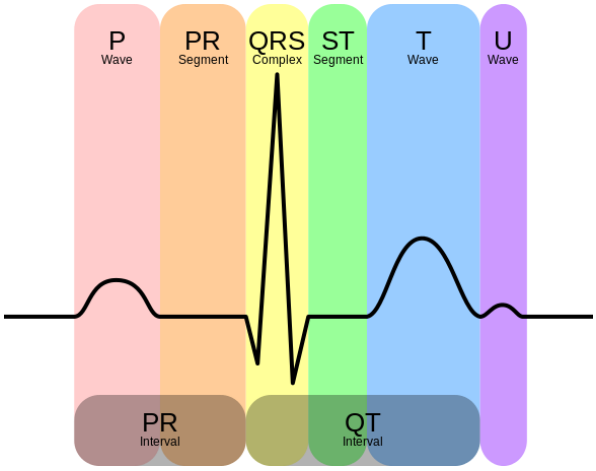


Figura 3: ECG [4]

Las primeras instancias históricas de la electrocardiografía se tienen cuando Alexander Muirhead colocó cables en las muñecas de un paciente con fiebre para detectar su pulso mientras estudiaba para su doctorado en ciencias en 1872 en el hospital de St. Bartholomew, en la ciudad de Londres, Reino Unido. Esta actividad fue tomada y visualizada directamente por el fisiólogo británico John Burdon usando un electrómetro de Lippmann: un dispositivo usado para detector pequeños pulsos de corriente eléctrica, inventada por Gabriel Lippmann en el 1873. Consiste en un tubo lleno hasta la mitad de mercurio y una pequeña cantidad de ácido sulfúrico diluido, con un extremo delgado y otro grueso, con cables conectados en estos dos extremos. Cuando una corriente circula a través del tubo, esta cambia la tensión superficial del mercurio, haciéndole ascender una corta distancia a través del tubo capilar.

La primera persona que tuvo un enfoque desde un punto de vista eléctrico del corazón fue Augustus Walter, que en el momento trabajaba en el hospital St. Mary en Paddington, Londres, Reino Unido. Su electrocardiógrafo consistía en un electrómetro de Lippmann conectado a un proyector. Se colocaba el electrómetro de Lippmann entre el proyector y una placa fotográfica, y ésta última a un tren de juguete, permitiendo el registro del latido del corazón en tiempo real. Aún en 1911, Walter no le vio demasiada aplicación clínica a su invento. El progreso más grande en este campo se produjo en 1901 cuando Willem Einthoven usó el galvanómetro de cuerdas para registrar la actividad eléctrica del corazón: un aparato que consistía en unos filamentos de cuarzo bañados en plata de varios metros de longitud, por los cuales transcurre una corriente eléctrica. Sobre estos filamentos actuaban unos potentes electroimanes, situados a ambos lados de los filamentos, lo que causaba que estos se movieran por acción del campo electromagnético. El movimiento de estas se ampliaba y se proyectaba a través de una ranura sobre una placa fotográfica en movimiento. En vez de usar los electrodos autoadhesivos que se utilizan hoy en día, Einthoven sumergía dos extremidades en cubos de solución salina. La nomenclatura que se usa hoy en día para las distintas secciones del ECG se le debe a éste inventor, que recibiría el premio Nobel en medicina en 1924. [5]

1.3 Arduino

Arduino es una familia de placas de desarrollo, que integran un microcontrolador Atmel AVR. Estas placas cuentan con pines de entrada y salida, uno o más puertos serie, dependiendo del modelo; y una interfaz USB. La placa viene con un entorno de desarrollo integrado (IDE, por sus siglas en inglés) gracias al cual se permite al usuario programar la placa. El IDE de Arduino está basado en el IDE del lenguaje de programación Processing, a su vez basado en Java.

Arduino es una plataforma de desarrollo de hardware libre, lo que quiere decir que sus especificaciones y diagramas son de acceso público, ya sea de forma gratuita o bajo pago. Las placas Arduino (o las compatibles con éste) pueden usar los llamados shields, unas placas que se conectan a los pines de la placa Arduino para expandir las utilidades de la misma. Existen shields para gran variedad de aplicaciones: control de motores, interfaz SD, NFC o conectividad WiFi, entre otras.

Figura 4: Logotipo de Arduino

Arduino comenzó en 2005 como un proyecto para los alumnos del Interaction Design Institute Ivrea (IDII). En aquel entonces, los alumnos hacían uso de unos microcontroladores BASIC Stamp, que costaban unos 100 dólares cada uno. Massimo Banzi, que se convertiría en uno de los fundadores de Arduino, en aquel momento profesor en el IDII, pensaba que el precio era demasiado elevado para los alumnos y los microcontroladores BASIC Stamp carecían de potencia de cálculo para ciertas aplicaciones. Además, necesitaba una plataforma que pudiera ejecutarse en ordenadores Macintosh, comunes entre los desarrolladores del IDII. Banzi tenía un amigo en el MIT que había desarrollado el lenguaje de programación Processing, que inspiró a Banzi a desarrollar un software similar para la que sería su plataforma. El primer prototipo vio la luz en 2005, aunque aún no se llamaba Arduino. Ese año, Banzi escogería dicho nombre para la plataforma. Banzi quería una placa que costara menos de 30 dólares, y, siguiendo esa filosofía, utilizaron componentes fáciles de encontrar en caso de que alguien quisiera montar la placa por su cuenta. Rápidamente y sin ayuda de anuncios o marketing, Arduino cobró fama gracias a internet. Debido a su facilidad de uso y precio reducido, a lo largo de los años siguientes fue creciendo hasta convertirse en una de las plataformas líderes de desarrollo de hardware libre que es hoy. [6]

1.4 Android

Android es un sistema operativo móvil de código libre desarrollado por Google basado en el kernel de Linux. Está enfocado a dispositivos con pantalla táctil, tanto teléfonos móviles, como tabletas electrónicas y recientemente, dispositivos de muñeca (Android wear). En 2013, el 60% de los sistemas (ordenadores personales, tabletas, teléfonos móviles y sistemas híbridos) usaban Android como sistema operativo. Dentro de éstos, la distribución de las distintas versiones del sistema en uso es la que se muestra debajo. Como se puede comprobar, la versión predominante es la Marshmallow (6.0.x, API 23). Este es un dato a tener en cuenta, ya que ciertos elementos pueden no estar disponibles para una versión determinada y resultaría en incompatibilidades.

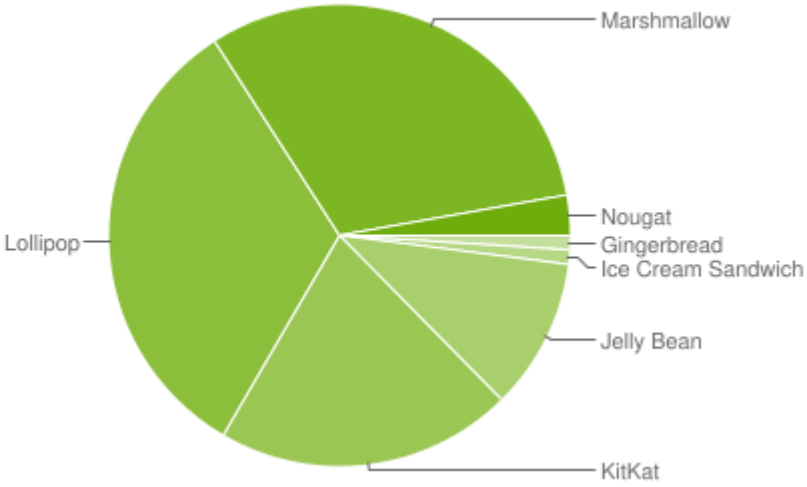


Figura 5: distribución de las versiones de Android con fecha de 6 de marzo de 2017 [7]

En el nivel más bajo del sistema operativo está el kernel de Linux. En informática, un kernel es un programa que gestiona las peticiones de entrada y salida de datos y las traduce en instrucciones de procesamiento de datos para la CPU. Sobre dicho kernel se encuentran las librerías y APIs escritas en C. Sobre éstas, el framework de aplicación en el que se encuentran las librerías compatibles con Java basadas en Apache-Harmony. Sobre el framework, en el nivel más alto, se encuentran las aplicaciones del teléfono.

Android comenzó en 2003, cuando sus fundadores Andy Rubin, Rich Miner, Nick Sears y Chris White decidieron desarrollar un sistema operativo avanzado para cámaras digitales. Al darse cuenta de que el mercado para tales dispositivos podría no ser lo suficientemente extenso, cambiaron el rumbo hacia el desarrollo de un sistema operativo para rivalizar con Symbian y Windows Mobile. Google adquirió Android en 2005 y Rubin, Miner y White se quedaron en la compañía. El primer dispositivo con Android, el teléfono móvil HTC Dream, fue lanzado al mercado el 22 de octubre de 2008. En 2010 Google lanza la familia de dispositivos Nexus, una línea de *smartphones* y tabletas electrónicas Android sin capas de software adicionales al propio Android. Desde 2008, Android ha lanzado un gran número de actualizaciones que han incrementado las funcionalidades del sistema operativo. El nombre de cada versión es el de un postre o dulce, y va en orden alfabético: Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow y Nougat. Esta última versión vio la luz en agosto de 2016.

1.5 Programación orientada a objetos, Java y Android: aclaración previa

Para el completo entendimiento de gran parte de la programación utilizada en el proyecto se hace necesario se hace necesario aclarar algunos conceptos relativos a la programación orientada a objetos (OOP, por sus siglas en inglés) y el kit de desarrollo de software (SDK, también por sus siglas en inglés) de Android.

La OOP es un paradigma de programación que representa el concepto de objetos con campos de datos y procedimientos. La base de la OOP son las clases. Una clase es un conjunto de datos (similar a una estructura en C) que además de incluir variables incluye funciones, llamadas métodos. Declarar una clase para su posterior uso se llama instanciar o hacer instancia. El constructor es un método homónimo a la clase; al llamar al constructor se hace una nueva instancia de la clase. Los constructores se programan de manera que se les pasen como parámetros los valores iniciales del nuevo objeto. Con la OOP surge el concepto de herencia: una clase puede heredar de otra, es decir, implementar sus métodos y tener sus variables. Esto simplifica el código, pues, si se quisiera extender el comportamiento de una clase sólo habría que crear una clase nueva e indicar de qué clase hereda en vez de implementar todos los métodos y especificar todos los valores de las variables.

En Java existen unas clases llamadas *listeners* (oyentes, en español). Estas clases tienen unos métodos que se disparan cuando sucede un evento determinado. El SDK de Android los implementa y especializa, de manera que cada *widget* (botones, campos de texto…) tiene una amplia gama de listeners. Dependiendo del *widget* que sea, el listener puede escuchar cambios de estado, pulsaciones sobre él, la creación del mismo…, etc.

La forma de añadir un *listener* a un *widget* es la siguiente: cada *widget* tendrá un método por el cual se le puede asignar un *listener* para cada tipo de evento que acepte, por ejemplo: pulsaciones en los botones, cambios de estado en los botones redondos, cambios de texto en los cuadros de texto… etc. Este método tiene como argumento una clase que herede del *listener* del evento que se está intentando escuchar, es decir: si se quisiera añadir un listener a un botón de manera que realice una determinada acción cuando éste ha sido pulsado, se usaría el método setOnClickListener() y el tipo de argumento que este método aceptaría sería una clase que heredara de la subclase OnClickListener de la clase Button. La acción a realizar ante el evento gestionado por el listener que se está programando se especifica en el método de la subclase pasada por parámetro, normalmente homónima al evento. Este método puede o no llevar argumentos en función del evento para el que se esté implementando un listener. Siguiendo con el ejemplo anterior, dentro de la subclase Button.OnClickListener existe un método llamado OnClick() que hay que sobrescribir para añadirle la función deseada a dicho botón. No es necesario crear un archivo Java distinto para albergar la clase del listener, se puede declarar in situ.

1.6 Desarrollo de aplicaciones para dispositivos Android

Como se ha mencionado con anterioridad, los dispositivos Android están programados en Java, haciendo uso de los SDKs provistos por Google. Es por esto que se pueden desarrollar aplicaciones Android en cualquier entorno de desarrollo para Java siempre que se implemente dicho SDK. Hasta diciembre de 2014 la herramienta provista por Google para el desarrollo de aplicaciones Android era el *plugin* Eclipse ADT (Android Development Tools) para el entorno de desarrollo Eclipse. Este IDE se usa principalmente para desarrollar aplicaciones en Java, sin embargo, puede utilizarse para desarrollar aplicaciones en otros lenguajes de programación a través de plug-ins: Ada, ABAP, C, C++, COBOL, D, Fortran, Haskell, JavaScript, Julia, Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby (incluyendo el framework Ruby on Rails), Rust, Scala, Clojure, Groovy, Scheme, y Erlang. [8]

Figura 6: Logotipo de Eclipse

En mayo de 2103, Google anunció el lanzamiento de Android Studio, un entorno de desarrollo dedicado únicamente al desarrollo de aplicaciones para dispositivos Android en su conferencia Google I/O. En junio de 2014 la primera versión en beta vio la luz. La primera versión estable fue lanzada en diciembre de 2014. Este entorno de desarrollo contiene las mismas funcionalidades del *plugin* ADT (configurar proyectos Android en blanco, crear una interfaz gráfica para la aplicación, añadir paquetes basados en la API del framework de Android, depurar las aplicaciones y exportar archivos .apk firmados o sin firmar para distribuir las aplicaciones). [9]

Figura 7: Logotipo de Android Studio

1.7 Comunicación Bluetooth

Bluetooth es un estándar de comunicación sin cables, de corto alcance y entre dispositivos móviles o fijos. La comunicación Bluetooth tiene su origen en 1994, cuando la desarrolla la compañía de telecomunicaciones Ericsson como alternativa sin cables a los cables de datos RS-232. El nombre Bluetooth (en español: diente azul) es la forma inglesa del apodo de Harald Bjarkan, rey danés que unificó las tribus de Dinamarca e introdujo el cristianismo en la región. El logotipo proviene de la superposición de las runas de las iniciales de dicho rey.

Figura 8: Logotipo de Bluetooth

La comunicación Bluetooth opera en un rango de frecuencias entre 2402 y 2480 MHz. Esta comunicación hace uso de la técnica de modulación llamada "espectro ensanchado por salto de frecuencia" (del inglés *frequency hopping spread spectrum*, FHSS), inventada por la actriz americana Hedy Lamarr. El Bluetooth tiene una estructura maestro-esclavo, en la que un dispositivo actúa como servidor (maestro) y se conecta a hasta 7 dispositivos (esclavos), aunque no todos los dispositivos maestros son capaces de alcanzar este número de dispositivos esclavos. [10]

2 PROBLEMÁTICA Y OBJETIVOS DEL PROYECTO

2.1 Problemática

- Las descargas de los ICD pueden ser bastante altas, luego es peligroso para la electrónica en general
- Selección del mejor enfoque para la detección del evento

2.2 Objetivos

- Diseñar un filtro para la señal del electrocardiograma permita la obtención de la frecuencia cardiaca.
- Realizar la programación del sistema de lectura del electrocardiograma
    - Lectura del electrocardiograma
    - Tratado de la señal
    - Cálculo de la frecuencia cardiaca
    - Aviso en caso de que la frecuencia cardiaca salga de los límites estipulados
    - Conectividad con el Smartphone
- Idear el protocolo de comunicación entre ambos.
- Desarrollo de la aplicación en Android
    - Interfaz gráfica
    - Conectividad Bluetooth
    - Determinación de la posición
    - Posibilidad de configurar parámetros relativos a la aplicación
    - Aviso por SMS

3 ANÁLISIS DEL ESTADO DEL ARTE

Tradicionalmente el seguimiento de los pacientes y del estado de los ICDs se hace a través de una intervención quirúrgica. Estas intervenciones se hacen en intervalos de unos tres meses, y estaban inicialmente orientadas a la reforma de los condensadores del dispositivo. La característica de reforma automática de condensadores permitió a los doctores concentrarse en otros aspectos, entre los que se incluyen estado de la batería, impedancia de la descarga, historial de terapias, parámetros del marcado de paso para la bradicardia y electrocardiogramas almacenados en el dispositivo.

Un dispositivo candidato para la monitorización remota tiene la habilidad de transmitir un mensaje periódico y, en algunos casos, también mensajes activados por el paciente. La transmisión se puede realizar de varias formas, a través de la línea telefónica o mediante otras redes (por ejemplo: mediante satélite). En un modelo sacado al mercado hace años, los datos los recibe un dispositivo oyente que transmite un mensaje encriptado a un centro de servicio de monitorización doméstica. Allí, el mensaje se desencripta y enviado mediante fax al doctor encargado del paciente. Este mensaje contiene información básica sobre el diagnóstico. El primer sistema disponible para los ICDs es un ICD de una sola cámara Belos VR-T (Biotronik, Berlín, Alemania), en combinación con un dispositivo oyente RUC 1000-A (Biotronik, Berlín, Alemania). El dispositvo oyente es un teléfono GSM dedicado. [11]

4 JUSTIFICACIÓN DE LA SOLUCIÓN ADOPTADA

La primera cuestión a considerar es la detección del evento. Debido a que durante la totalidad de la duración del desarrollo del proyecto no se ha contado con un ICD, no se ha podido siquiera intentar el desarrollo de una comunicación entre éste y el dispositivo móvil. Por lo tanto, la única opción es desarrollar un sistema de detección propio. Ante este problema, se abren dos caminos posibles: por un lado, se podría detectar la descarga sobre el usuario y, por otro lado, se podría reaccionar ante situaciones peligrosas.

El problema con la primera metodología es que habría que detectar una descarga eléctrica potencialmente fuerte, por lo que habría no solo que aislar las partes más sensibles del dispositivo, sino que además habría que desarrollar un circuito que convirtiera una señal eléctrica de potencialmente decenas o centenas de voltios (no existe documentación fiable disponible al público al respecto) a una señal interpretable por dicho dispositivo.

Es por ello, que se decidió detectar comportamientos anómalos en el pulso del paciente. Si bien esto se puede conseguir con un pulsímetro de muñeca, se consideró que el muestreo del electrocardiograma sería más versátil, pues, de haberse continuado el desarrollo del proyecto, además de contabilizar el pulso, se podría analizar esta onda, pudiendo detectar comportamientos anómalos como las peligrosas fibrilaciones ventriculares.

Una vez hecho esto, se hace necesaria la elección de un sistema de medida de la señal cardiaca. El sistema que lea la señal, además, debe ser capaz de tratarla y de realizar una comunicación con el Smartphone. Para tal fin, se eligió una placa Arduino, por su versatilidad y facilidad de programación. Para la obtención del electrocardiograma, se compró un shield preparado para leer electrocardiogramas y electromiogramas. Para la comunicación entre la placa y el Smartphone, se eligió el Bluetooth, debido a que todos los Smartphone vienen equipados con esta característica.

Concretando lo decidido en los párrafos anteriores, la solución se compondrá de unos electrodos conectados al paciente y a un Arduino con un shield encargado de tratar la señal provinente de dichos electrodos. El Arduino se comunica por Bluetooth a un dispositivo Android. Este, podrá recibir una alerta en caso de que se produzca, consultar la frecuencia cardiaca actual del paciente, cambiar el límite superior e inferior de aviso y pedir los dos valores de los umbrales de aviso con los que está trabajando el Arduino.

4.1 Programas utilizados

A continuación, se expondrán los programas que se han utilizado para las diversas tareas que engloba el proyecto.

4.1.1 Arduino IDE

El lenguaje de programación usado para las placas Arduino es un conjunto de funciones de C/C++. Las placas Arduino pueden programarse a través del IDE proporcionado por los fabricantes, disponible de forma gratuita en su página web (www.arduino.org/download). Este IDE difiere de otros entornos de desarrollo disponibles para otros microcontroladores en que es extremadamente fácil comenzar a programar el microcontrolador, sin necesidad de conocimientos previos de electrónica. Como contraposición a este, cabe mencionar los microcontroladores de Texas Instruments, que programados mediante el IDE Code Composer Studio, requiere gestión de los registros del microcontrolador de forma explícita, mientras que con el Arduino IDE, todas estas gestiones se hacen de forma automática y sin interacción del usuario.

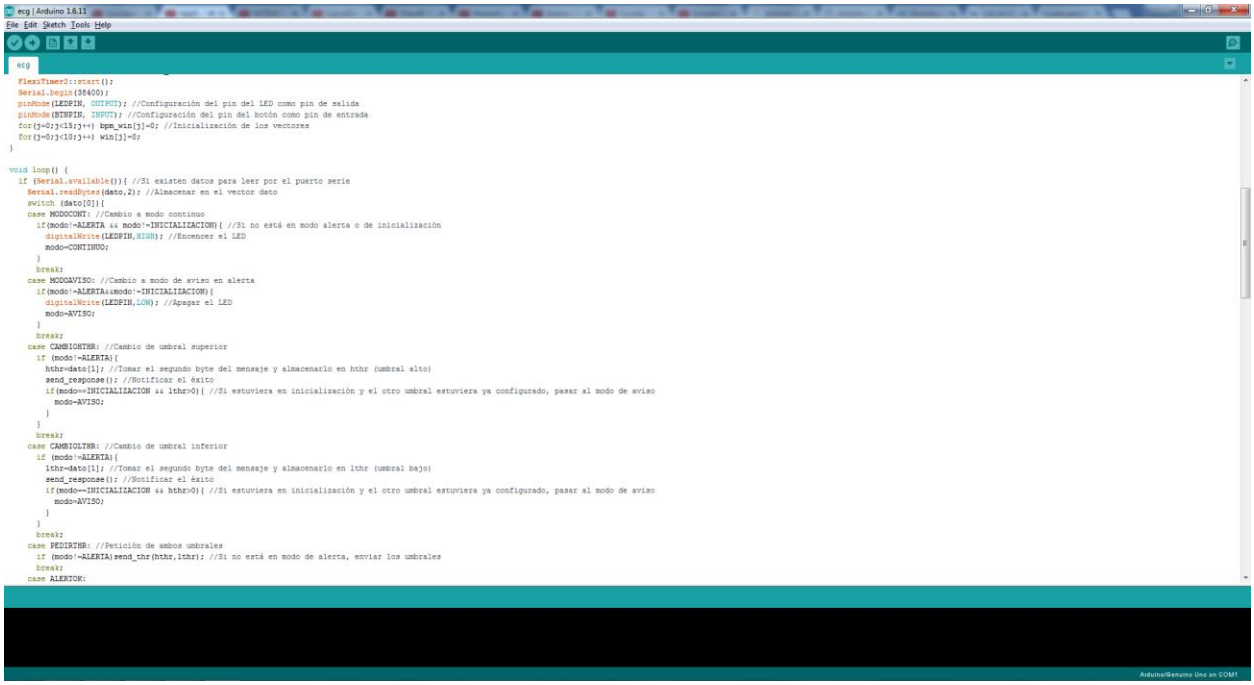


Figura 9: Apariencia visual de Arduino IDE

4.1.2 Android Studio

Este programa se ha expuesto en detalle en la sección anterior. El desarrollo del proyecto se comenzó en mayo de 2013, por lo que las primeras fases del desarrollo de la aplicación se realizaron en Eclipse con el *plugin* de ADT. Tras el lanzamiento de Android Studio, el desarrollo continuó en este entorno por ser más estable y estar soportado por Google.

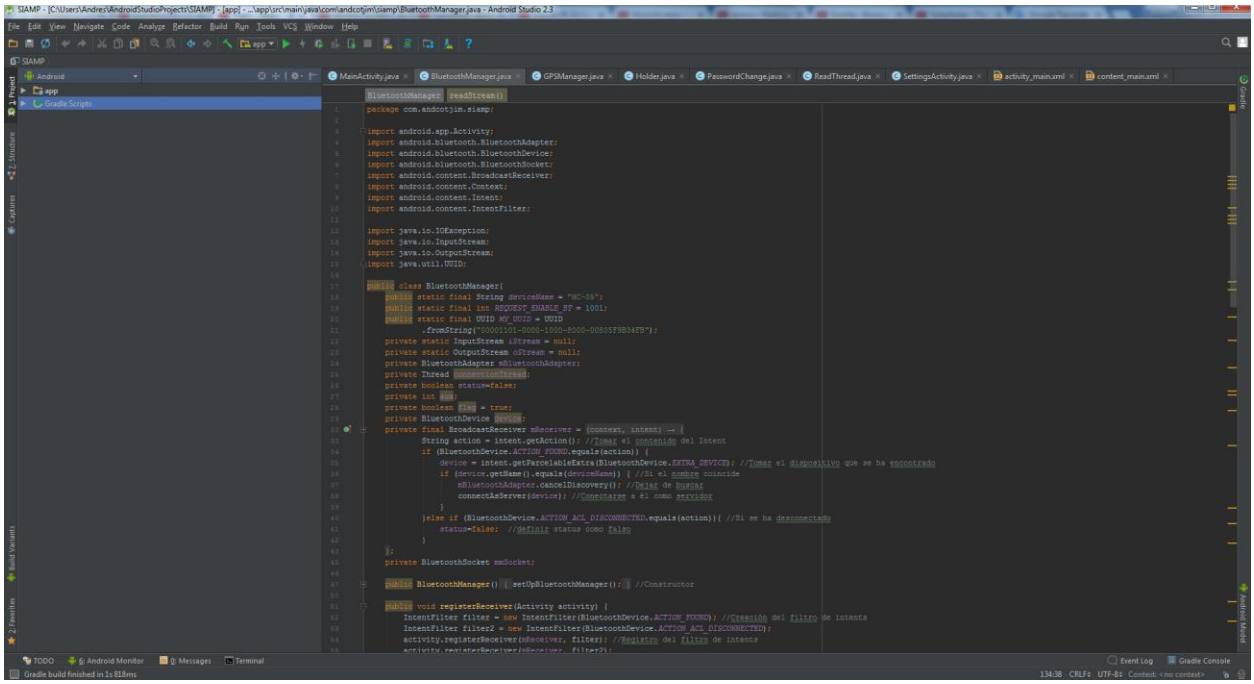


Figura 10: Apariencia visual de Android Studio

4.1.3 MATLAB

Tanto la toma de datos como el desarrollo del filtro se realizó usando MATLAB. Para la primera tarea se usó la utilidad de comunicación en serie, mientras que, para la segunda, se usó el *System Identification Toolbox*.

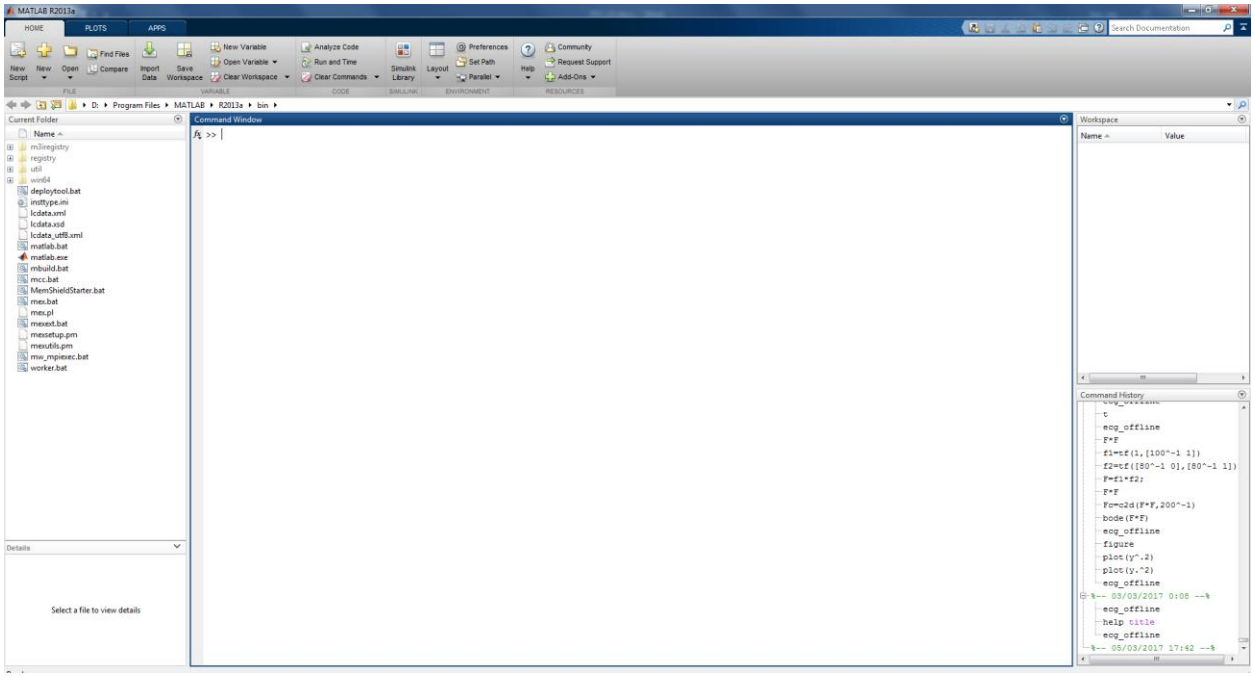


Figura 11: Apariencia visual de MATLAB

5 PLAN DE TRABAJO

A continuación, se muestra un diagrama con las distintas fases del proyecto, ordenadas de forma cronológica.

6 OBTENCIÓN DE LA SEÑAL Y DESARROLLO DEL FILTRO

El primer paso es obtener una muestra de la señal a tratar, para desarrollar un sistema de filtrado sobre ella, a fin de identificar los pulsos cardíacos. En esta sección se desarrolla el proceso seguido para llegar a tal fin.

6.1 Selección de la frecuencia de muestreo

Como bien es sabido, la correcta elección de la frecuencia de muestreo es esencial para el tratamiento de una señal. El seleccionar una tasa de muestreo menor que la adecuada puede llevar a la pérdida de información, mientras que mediante una tasa de muestreo excesivamente elevada se podría recaer en el tratamiento de demasiados datos, pudiéndose obtener la misma información con una frecuencia de muestreo algo menor. La onda del ECG que se ha decidido tomar para el cálculo de la frecuencia cardiaca es la onda R: el pico más alto del complejo QRS, de unos 90 Hz de frecuencia [11]. El teorema de Shannon-Nyquist dice que, para evitar la pérdida de información, hay que muestrear, al menos, al doble de la frecuencia de la onda que se muestrea. Esto impone un límite inferior de 180 muestras por segundo, o un tiempo de muestreo de 0,00556 segundos. En aras de obtener números lo más redondos posibles para simplificar el cálculo de los intervalos de tiempo, se fijó la frecuencia de muestreo en 200 Hz, es decir, un tiempo de muestreo de 5 milésimas de segundo.

6.2 Obtención de una muestra de la señal

Para el diseño del filtro, es necesaria una muestra de la señal a filtrar. A tal fin, se programó un script en MATLAB y Arduino. El microcontrolador se encargará de leer la señal analógica proveniente de los electrodos y enviarla por comunicación serie a través del puerto USB cada 5 milésimas de segundo, la frecuencia de muestro. El script de MATLAB, por su parte, lee del puerto serie durante un tiempo, lo almacena en una variable, y posteriormente guarda el archivo para posterior tratado.

Se tomaron datos de tres personas distintas, para comprobar que el filtro funciona de igual manera en los tres casos. Como puede apreciarse en la imagen de debajo, aunque las formas del electrocardiograma son diferentes, la forma del complejo QRS es similar en todas, así como su magnitud, llegando en todas ellas a un valor superior a 1000.

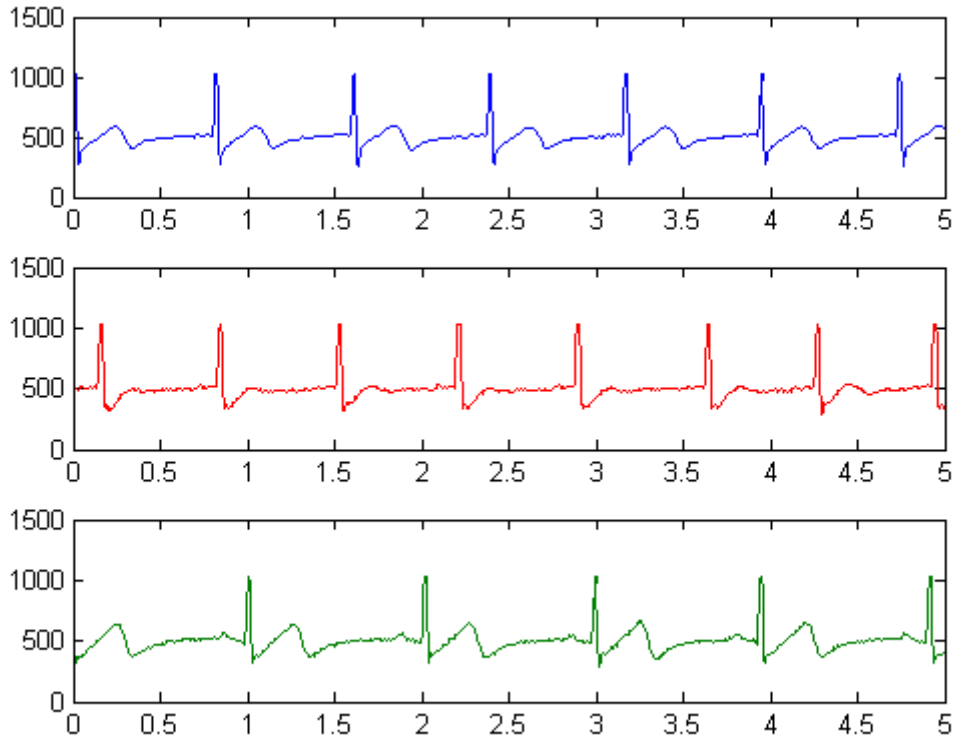


Figura 12: Muestra de ECGs de 3 personas distintas

6.3 Colocación de los electrodos

La situación de los electrodos es de vital importancia para la obtención de un buen electrocardiograma. Como se detallará a continuación, los electrodos utilizados fueron diseñado para ser colocados en las extremidades: los electrodos izquierdo y derecho en las muñecas correspondientes, y la derivación en el tobillo derecho. Sin embargo, tras consultar con personal sanitario y realizar pruebas con distintas disposiciones de los electrodos, el mejor resultado se obtuvo al colocar los electrodos izquierdo y derecho bajo el pectoral mayor izquierdo y derecho, respectivamente, hacia la mitad de la clavícula; y con la derivación colocada en la espalda. De esta forma se consigue eliminar las señales eléctricas de los músculos cercanos a los electrodos que pueden deformar el electrocardiograma. Además, al estar los electrodos más cerca del corazón, el electrocardiograma resulta más amplio y es más sencillo de tratar, pues aumenta la relación señal-ruido.

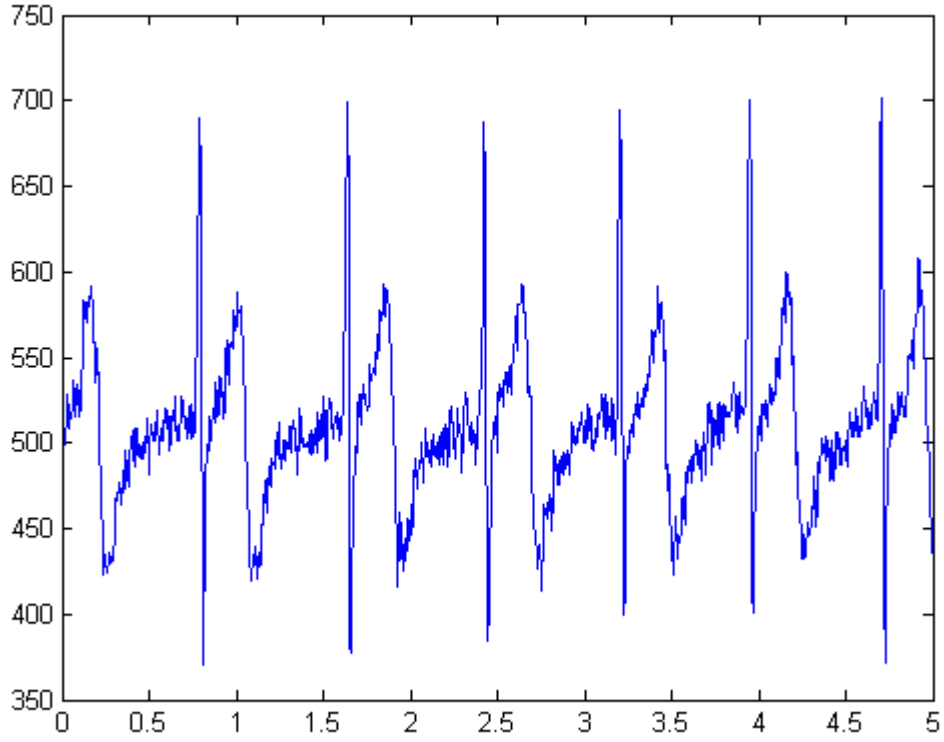


Figura 13: Electrodos en las extremidades, músculos en reposo

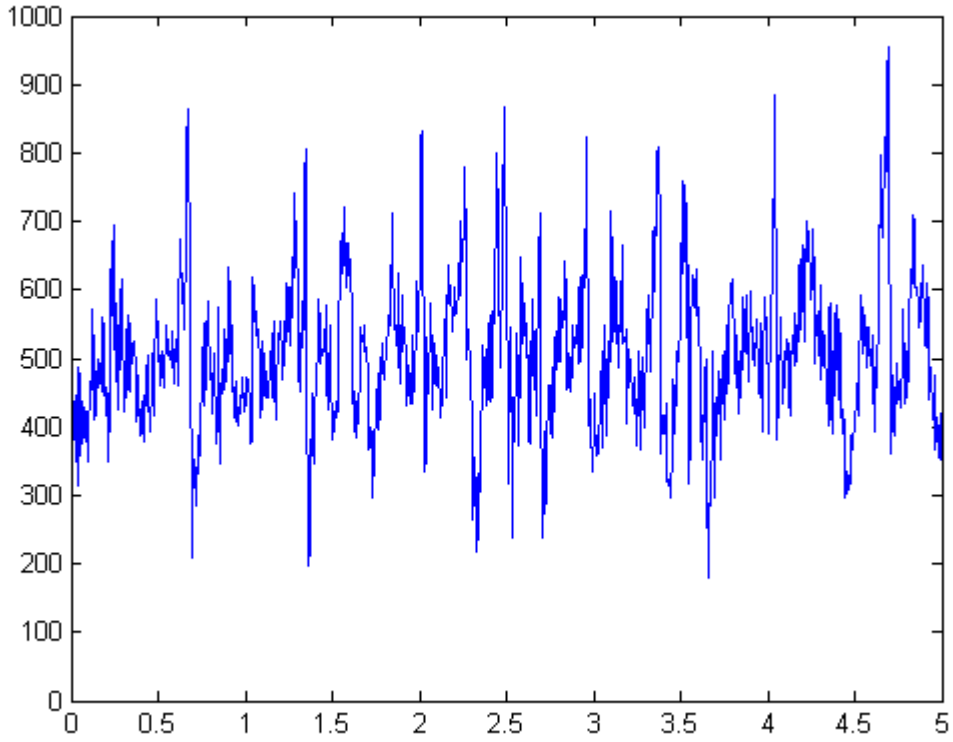


Figura 14: Electrodos en las extremidades, músculos en movimiento

Arriba se muestra una comparación de la señal obtenida con los electrodos colocados en las extremidades. En la primera imagen, durante el registro de la actividad cardiaca se realizaron movimientos de los músculos cercanos a los puntos de colocación de los electrodos (flexor del carpo izquierdo y derecho para los electrodos izquierdo y derecho, gemelos para la derivación) y en la segunda, con los músculos en reposo. Como se puede apreciar, aparece una gran cantidad de ruido proveniente de las señales eléctricas de dichos músculos.

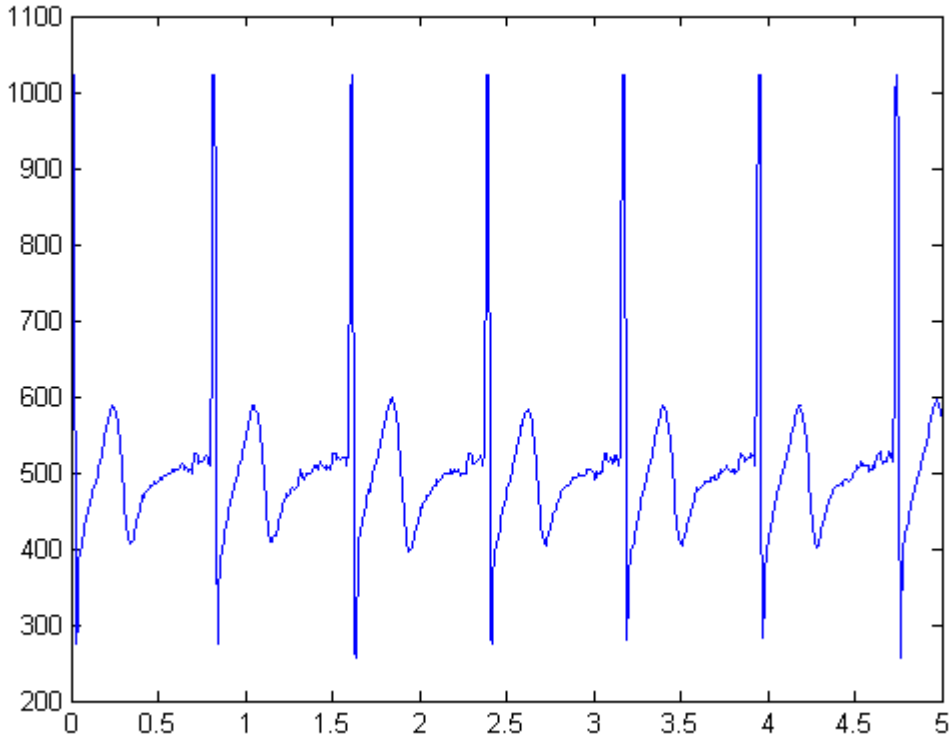


Figura 15: Electrodos en el pecho, músculos en reposo

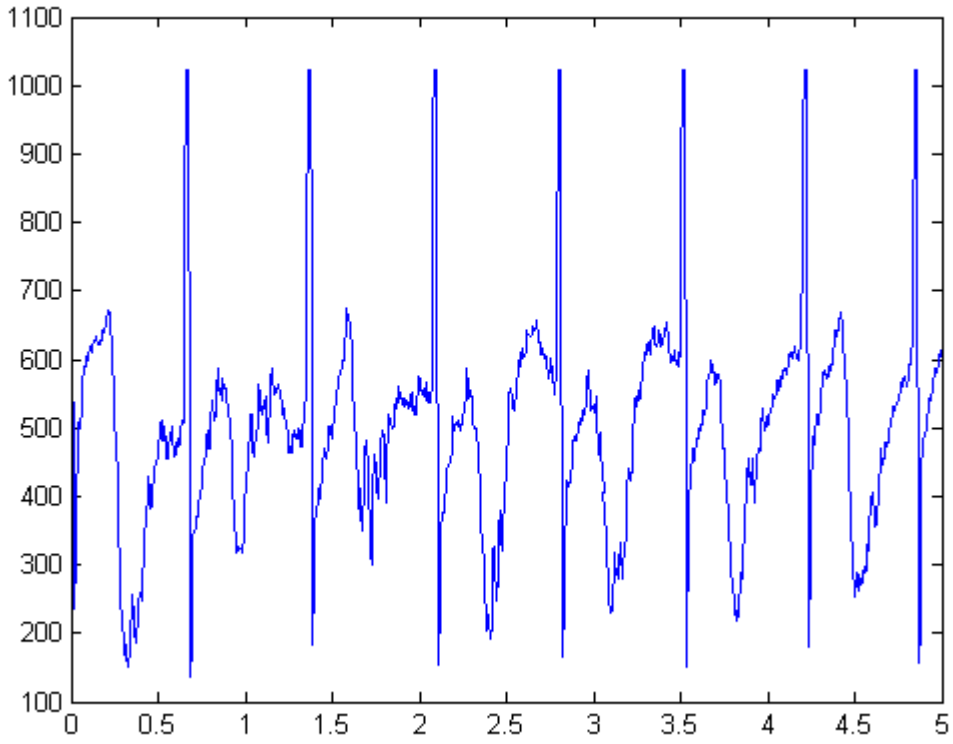


Figura 16: Electrodos en el pecho, músculos en movimiento

En este caso, la señal mostrada es la obtenida en el mismo caso anterior, pero con los electrodos colocados en el pecho. De la misma forma que en la anterior, la gráfica superior muestra la señal obtenida con un movimiento continuo de los músculos más cercanos a los electrodos (serrato, dorsal ancho), mientras que durante el tiempo de captación de la señal de la imagen inferior se han mantenido dichos músculos en reposo. La señal también aparece deformada por las señales eléctricas transmitidas por los músculos, pero el complejo QRS queda prácticamente inalterado, por lo que a la hora de identificar el pulso se buscará filtrar el resto del ECG y aislar dicho complejo. Como ventaja añadida, esta disposición de los electrodos es más compacta y cómoda para el individuo que la lleva.

6.4 Diseño del filtro

Como se argumentó anteriormente, para la identificación de la frecuencia cardiaca se buscará aislar el complejo QRS, más concretamente, la onda R. Ya que dicha onda tiene una frecuencia instantánea de unos 90 Hz [11], el diseño del filtro irá encaminado a filtrar las frecuencias que caigan fuera del entorno de los 90 Hz. Por lo tanto, la primera etapa del filtrado es filtrar las frecuencias que caigan fuera del entorno de los 90 Hz. Esto se puede conseguir con un filtro paso banda, compuesto por un filtro paso bajo y un filtro paso bajo en serie, con frecuencias de corte de 80 Hz y 100 Hz respectivamente.

La fórmula de la función de transferencia de un filtro paso bajo en el dominio de la frecuencia se puede expresar como sigue:

$$G_{paso\ bajo}(s) = \frac{1}{\tau_B s + 1}$$

Donde τ_B representa la inversa de la frecuencia de corte. Del mismo modo, la fórmula de un filtro paso alto en el dominio de la frecuencia se expresa de la siguiente forma:

$$G_{paso\ alto}(s) = \frac{\tau_A s}{\tau_A s + 1}$$

Donde τ_A vuelve a representar la frecuencia de corte del filtro.

Para colocar dos sistemas continuos en serie, no hay más que multiplicar sus funciones de transferencia, por lo que se obtiene:

$$G_{paso\ banda}(s) = G_{paso\ bajo}(s) \cdot G_{paso\ alto}(s) = \frac{\tau_A s}{\tau_A \tau_B s^2 + (\tau_A + \tau_B)s + 1}$$

Y particularizando para los valores $\tau_A = 80^{-1} = 0{,}0125$ y $\tau_B = 100^{-1} = 0{,}01$

$$= \frac{0{,}0125s}{1{,}25 \cdot 10^{-4}s^2 + 0{,}0225s + 1}$$

Gracias a este filtrado se atenúa el efecto del ruido de red (50 Hz), se elimina la componente continua (0 Hz) y se eliminan posibles ruidos de más alta frecuencia (>100 Hz). A continuación, se muestra el diagrama de Bode del filtro paso alto y bajos continuos, y su combinación en un filtro paso banda.

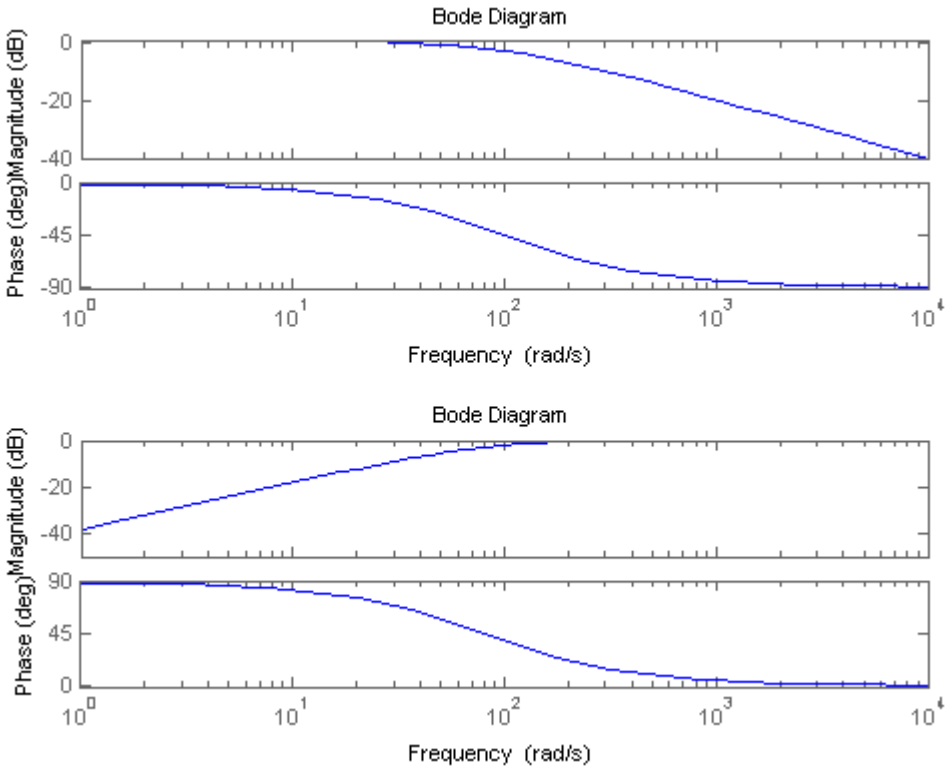


Figura 17: Diagramas de bode del filtro paso bajo (arriba) y del filtro paso alto (abajo)

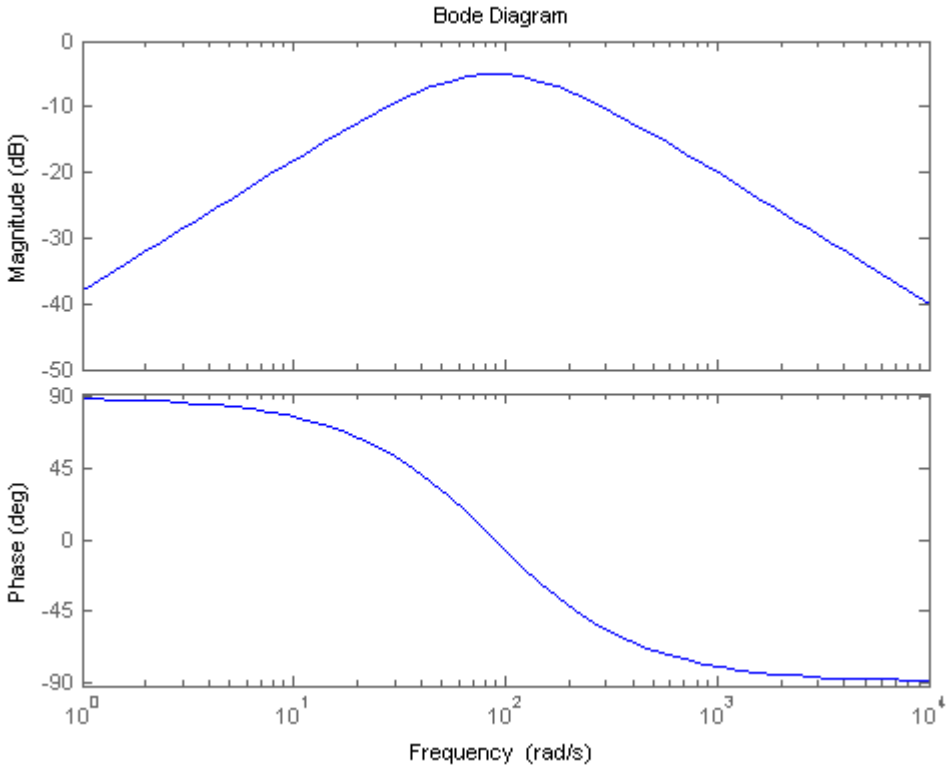


Figura 18: Diagrama de Bode del filtro paso banda

El filtro no se puede aplicar directamente a la señal; esto es debido a que en la realidad la toma de datos no es un proceso continuo sino discreto, y por lo tanto no se puede aplicar un filtro de carácter continuo. Para

realizar la conversión, es necesario realizar una aproximación de la señal partiendo de un número finito de valores de ella. Esta aproximación se puede hacer de tres formas: mediante la aproximación de Euler hacia delante, la de Euler hacia detrás y la de Tustin o bilineal. A continuación, se presenta un esquema de las tres:

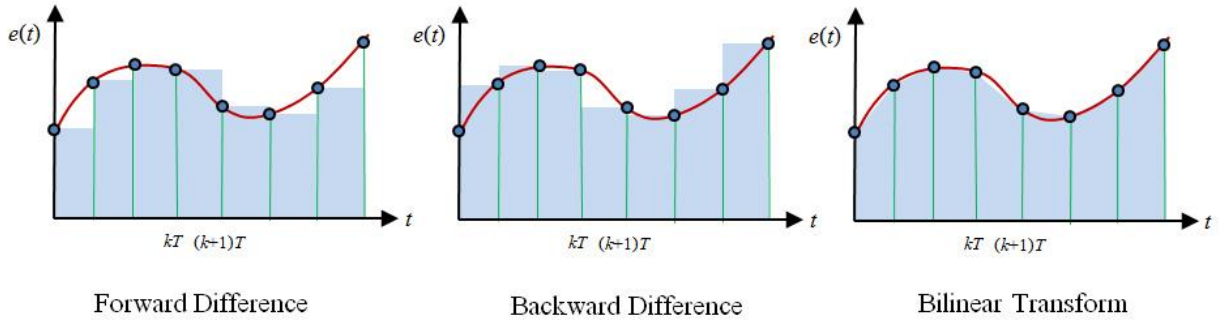


Figura 19: Métodos de aproximación [12]

Cada una de ellas dará lugar a una aproximación discreta distinta del filtro. Puede demostrarse que para realizar la aproximación de un sistema lineal continuo de forma discreta solo hay que sustituir la variable s por la expresión correspondiente:

Para la aproximación de Euler hacia delante:

$$s \sim \frac{z-1}{T_S}$$

Para la aproximación de Euler hacia detrás:

$$s \sim \frac{z-1}{z\,T_s}$$

Y finalmente, para la aproximación bilineal:

$$s \sim \frac{2(z-1)}{T_s(z+1)}$$

Debido a que la aproximación bilineal es la que mejor se ajusta a la señal original, es la que producirá una mejor aproximación en tiempo discreto. Sin embargo, en el desarrollo del filtro se ha usado la aproximación de Euler hacia delante por simplicidad, ya que el microcontrolador del Arduino no permitía demasiada complejidad en el programa si había de hacerse 200 veces por segundo. Sustituyendo la expresión de Euler hacia atrás en la ecuación del filtro paso banda:

$$G_{paso\ banda}(z) = \frac{0{,}3189 - 0{,}3189\,z^{-1}}{1 - 1{,}277\,z^{-1} + 0{,}466\,z^{-2}}$$

Que expresada en forma de ecuación queda:

$$y_k = 1{,}277\,y_{k-1} - 0{,}466\,y_{k-2} + 0{,}3189\,(u_k - u_{k-1})$$

A continuación, se representa la señal inicial y la señal filtrada usando la ecuación anterior.

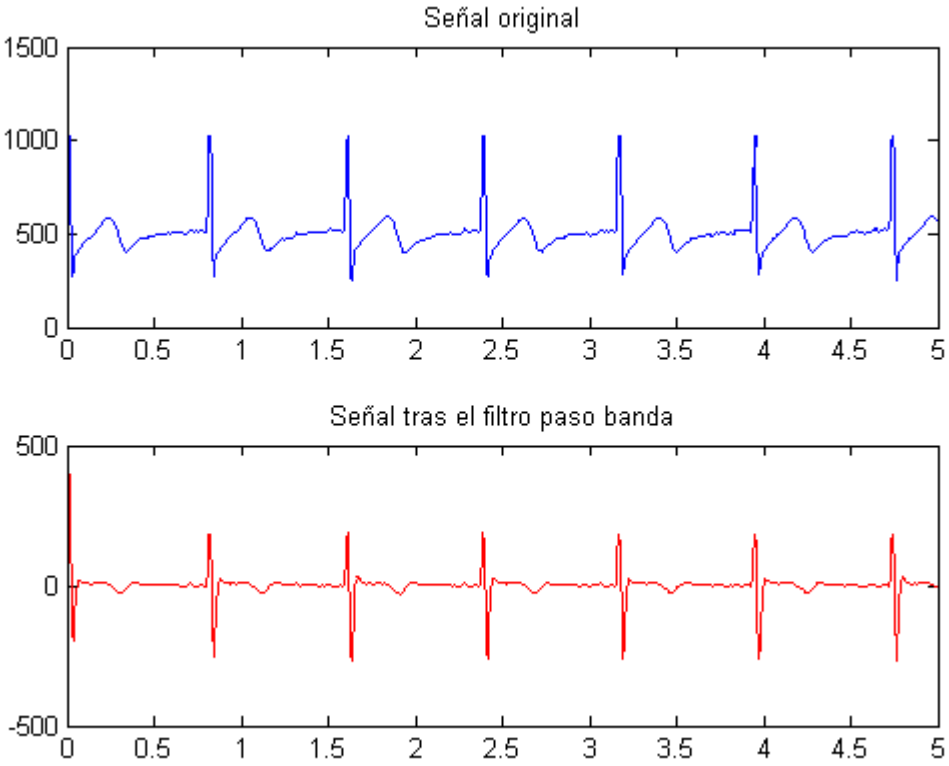


Figura 20: Señal antes y después del filtro paso banda

Como puede apreciarse, mediante este filtro se aísla el complejo QRS, dejando la señal prácticamente a 0 en la mayoría del resto del ECG. También se aprecia al comienzo un transitorio, llegándose a obtener un valor de hasta 400 en la señal filtrada. Este transitorio puede interferir en la medida del pulso, pero no afectará al valor final por varias razones. La primera de ellas es la corta duración del transitorio: la duración de éste es inferior a 0,1 segundos, por lo que afectará como máximo a una medida del pulso. A esta razón se le añade el hecho de que el transitorio solo tiene lugar al comienzo del filtrado. Finalmente, la frecuencia cardiaca a evaluar se calculará como la media en 15 pulsos en lugar de calcular la frecuencia cardiaca instantánea, filtrando aún más el error en el que se pueda incurrir.

El paso siguiente en el filtrado es calcular el valor absoluto de la señal, de esta manera, los picos obtenidos serán siempre mayores que cero.

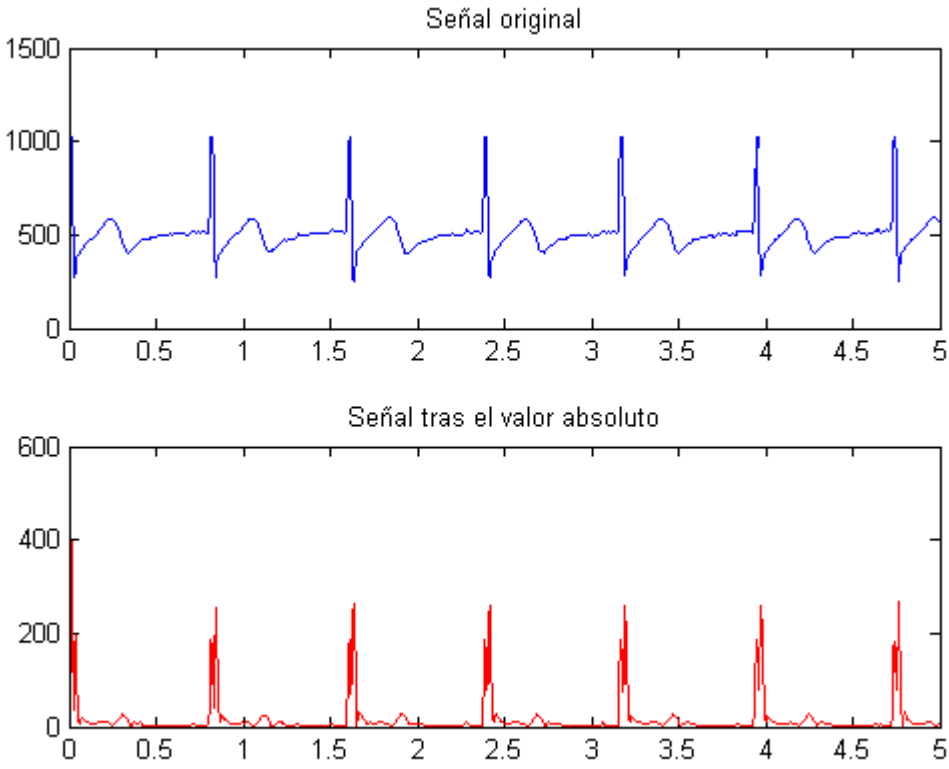


Figura 21: Señal antes y después de realizar el filtro paso banda y el cuadrado de la señal

Tras esta operación, los picos destacan y quedan completamente definidos, pero demasiado ruidosos. Para paliar este efecto, se le aplica la última fase del filtrado: un filtro de la media móvil, con una ventana de 10 tiempos de muestreo:

$$y_k = \frac{1}{10}\sum_{i=0}^{9} u_{k-i}$$

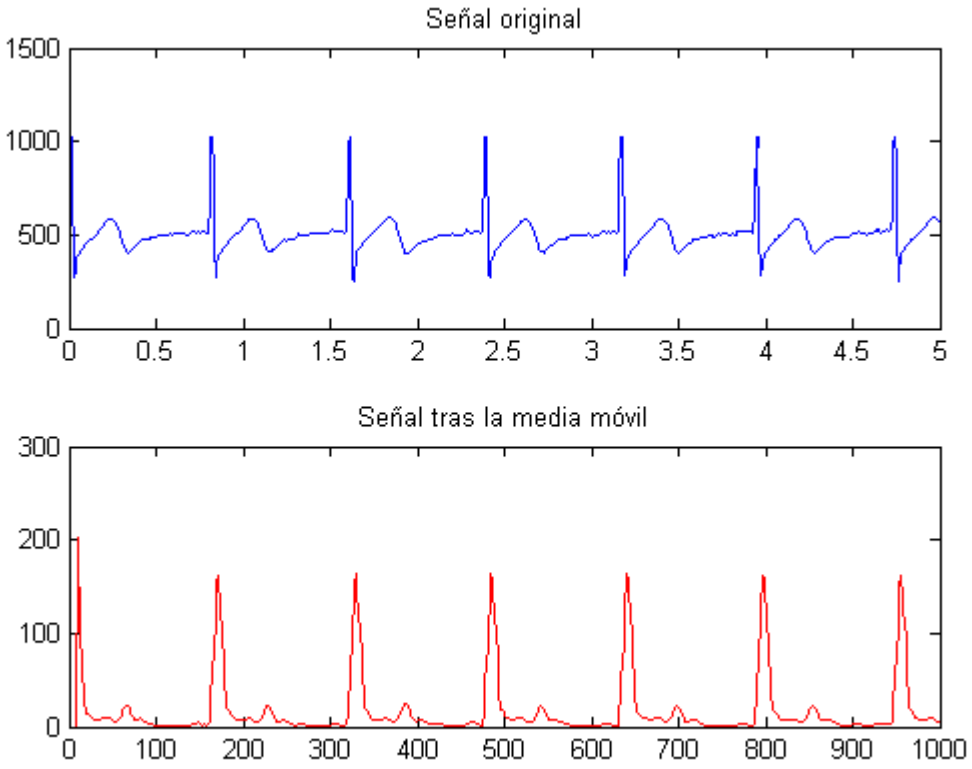


Figura 22: señal antes y después del filtrado completo

Tras el proceso completo de filtrado, se obtiene una señal con picos bien definidos y sin ruido donde se aprecia claramente el momento en el que ocurre la onda R. A continuación, se muestran las tres ondas que se obtuvieron en el apartado anterior. Puede apreciarse, que el comportamiento es como el esperado: picos bien definidos de un valor constante en el tiempo y consistentes entre sí. Por lo tanto, se tomará como medida de pulso cuando la señal tras el filtrado completo sobrepase un cierto umbral, que se estableció como 100 en la versión final del código.

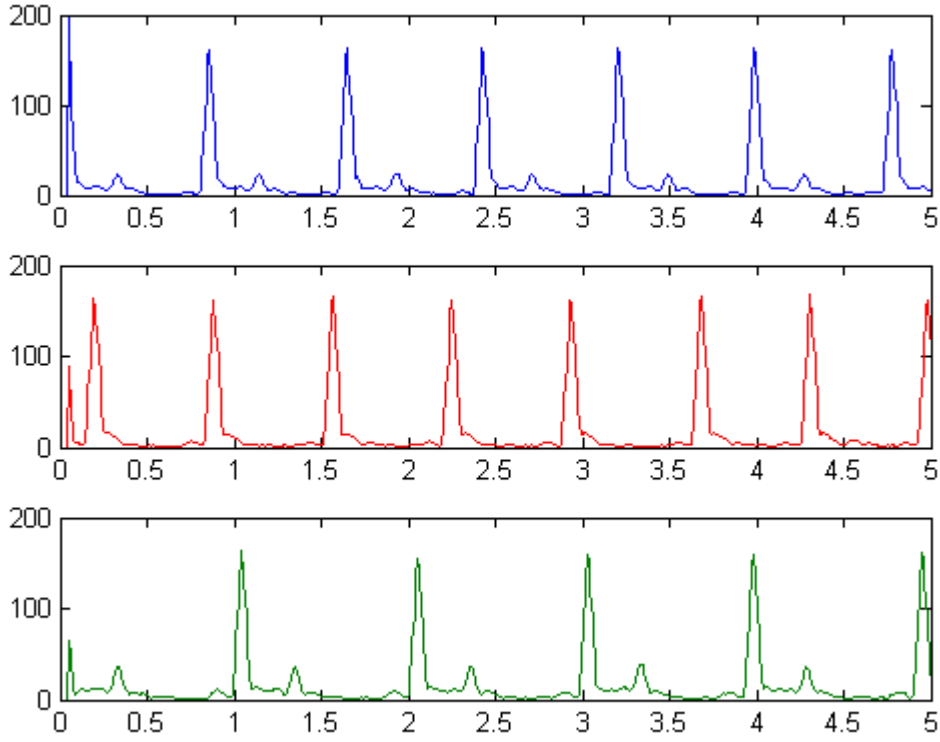


Figura 23: Filtrado de los 3 ECGs tomados en el apartado anterior

7 SISTEMA ARDUINO

7.1 Programación de un Arduino

Los microcontroladores de la familia Arduino se programan a través del IDE proporcionado por la propia empresa, que se puede encontrar de forma gratuita en su web http://www.arduino.org. La conexión entre el terminal desde el cual se va a programar el microcontrolador y el propio Arduino se realiza mediante un cable USB. Antes de poder programar la placa, hay que especificar en el IDE el puerto COM al cual está conectado la placa y qué tipo de placa es. El lenguaje de programación de Arduino, como se mencionó previamente, es una variante de C++. Esta variante incluye dos funciones auxiliares que simplifican la programación de la placa para los usuarios menos familiarizados con la programación de microcontroladores. Estas funciones son **setup()** y **loop()**.

- **void** setup (**void**): esta función se llama una única vez al arrancarse el microcontrolador y no se vuelve a llamar durante todo el ciclo de ejecución del programa. Debido a que es la primera función que se ejecuta, resulta útil tanto para inicializar variables, como para abrir puertos serie o configurar pines.
- **void** loop (**void**): en esta función se suele colocar la carga de trabajo de los microcontroladores Arduino o similares a ellos. Esta función se llama ad infinitum, por ello, se programa el comportamiento del microcontrolador en esta función.

Al terminar de programar el código, el entorno verifica el programa y busca en él fallos de sintaxis. En caso de no haberlos, se compila y se envía a la placa a través del puerto serie. La placa empleada para la realización del proyecto, un Arduino Uno, solo dispone de un puerto serie. Este puerto serie se usa tanto para programar la placa, como para comunicar la placa a través de UART. Por ello, es de vital importancia desconectar cualquier sistema de los pines que toman control del puerto serie (D0 y D1) en el momento que se quiera reprogramar la placa.

7.2 Componentes

El subsistema Arduino está compuesto por los siguientes componentes:

7.2.1 Arduino Uno

La placa Arduino Uno es una placa basada en el ATmega328. Tiene 14 pines de entrada/salida digitales (de los cuales, 6 pueden ser usadas como salidas PWM), 6 entradas analógicas, un resonador cerámico de 16MHz, un conector USB, un *jack* de alimentación, una entrada ICSP (programación en serie en circuito, por sus siglas en inglés) que permite la programación del microcontrolador implementado en un circuito sin necesidad de extraerlo; y un botón de reset.

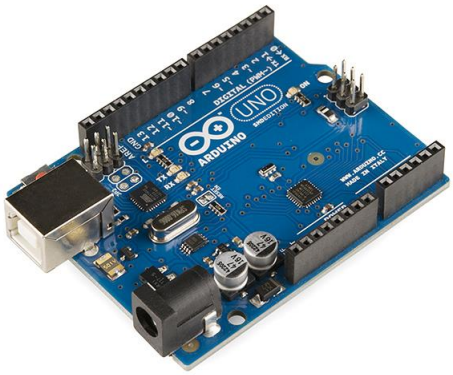


Figura 24: Placa Arduino Uno [13]

7.2.1.1 Alimentación

La placa Arduino Uno puede ser alimentada a través de la conexión USB o mediante una fuente de alimentación externa. La fuente de alimentación se elige automáticamente. La alimentación externa (no por USB) puede provenir de un convertidor de alterna a continua o de una batería. El adaptador puede ser conectado enchufando un conector de 2,1 mm de centro positivo en el *jack* de alimentación de la placa. Para alimentarse mediante batería, las bornas de ésta han de conectarse en los pines GND y Vin del conector POWER.

La placa puede operar mediante una fuente externa de 6 a 20 V. Sin embargo, si se le provee con menos de 7 V, el pin de 5V puede proporcionar menos de 5V y la placa puede ser inestable. Si se usan más de 12 V, el regulador de voltaje puede sobrecalentarse y dañar la placa. El rango recomendado de voltaje es de 7 a 12V.

7.2.1.2 Memoria

El ATmega328 tiene 32 KB (0,5 KB se usan para el *bootloader*). También tiene 2KB de SRAM y 1 KB de EEPROM (de donde se puede leer y escribir con la librería EEPROM)

7.2.1.3 Entradas y salidas

Cada uno de los 14 pines digitales en la placa Uno puede ser usado como entrada o como salida, usando las funciones **pinMode()**, **digitalWrite()** y **digitalRead()**. Trabajan a 5V. Cada pin puede dar o recibir un máximo de 40 mA y tiene una resistencia de *pull-up* interna (desconectada por defecto) de 20-60 kΩ.

7.2.1.4 Comunicación

El Arduino Uno tiene una serie de herramientas para la comunicación con un ordenador, otro Arduino u otros microcontroladores. El ATmega328 provee de comunicación por puerto serie UART TTL a 5v, disponible a través de los pines digitales 0 (RX) y 1 (TX). Este microcontrolador también soporta comunicación I2C y SPI.

7.2.2 SHIELD-EKG-EMG

El shield EKG-EMG es un shield de hardware libre, compatible con las placas arduino y similares vendido por la página de Olimex (http://www.olimex.com). Provee a la placa en cuestión la funcionalidad de tomar señales electrocardiográficas o electromiográficas, posibilitando, según la web del productor, la captura del ritmo cardiaco o el reconocimiento de gestos mediante la monitorización y el análisis de la actividad muscular.

Mediante los *jumpers* AIN_SEL del borde superior (derecho en la imagen) es posible configurar el canal por el que el shield enviará la señal leída al Arduino. Al haber 6 distintos canales por los que enviar la señal, se pueden apilar hasta 6 SHIELD-EKG-EMG en caso de necesitarse más electrodos. El shield también incluye un potenciómetro, usado para la calibración de la señal obtenida. Según el *datasheet* del fabricante, todos los shields vienen pre calibrados de fábrica, por lo que no se ha hecho necesario reajustarlo.

Figura 25: Shield EKG-EMG [14]

7.2.3 Módulo Bluetooth HC-05

El modulo Bluetooth HC-05 es un SPP (protocolo de puerto serie, por sus siglas en inglés) Bluetooth, diseñado para crear conexiones sin cables. El HC-05 tiene una modulación de 3Mbps Bluetooth V2.0+EDR (Tasa de datos mejorada, por sus siglas en inglés). Usa el sistema Bluetooth CSR Bluecore de un solo chip con tecnología CMOS y con AFH (característica de salto de frecuencia, por sus siglas en inglés). Tiene un tamaño de 12,7x27mm.

7.2.3.1 Pines

- STATE: en este pin se copia la tensión existente en el pin positivo del LED de estado del módulo Bluetooth.
- RXD: pin de recepción, uno de los dos canales necesarios para la comunicación por UART.
- TXD: pin de transmisión, el segundo de los canales necesarios para la comunicación por UART.
- GND: puesta a tierra.
- VCC: tensión de alimentación (3,3V)
- KEY: pin de configuración. Si se mantiene a un nivel lógico alto es posible configurar algunos parámetros del módulo Bluetooth (nombre del dispositivo, tasa de baudios a la cual transmite los datos… etc.).

7.2.3.2 Características de hardware

- Sensibilidad de -80dBm
- Hasta +4dBm de potencia de transmisión RF
- Operación de baja potencia a 1,8V
- Control PIO (entrada/salida programada, por sus siglas en inglés)
- Interfaz UART con tasa de baudios programable
- Antena integrada

7.2.3.3 Características de software

- Tasa de baudios por defecto de 38400
- Tasas de baudios soportadas: 9600, 19200, 38400, 57600, 115200, 230400, 460800.
- Por defecto realiza una auto conexión al último dispositivo al conectarse
- Reconexión automática tras el transcurso de 30 minutos si se desconecta como resultado de que se haya salido del rango

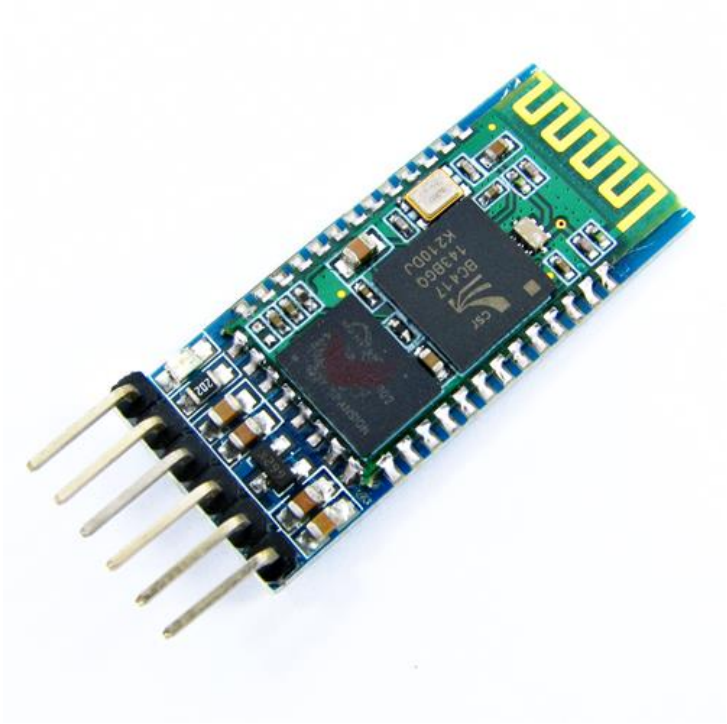


Figura 26: Módulo Bluetooth HC-05 [15]

7.2.4 Electrodos pasivos

Los electrodos que se han utilizado para el proyecto también han sido provistos por Olimex. Son tres electrodos pasivos diseñados para el EKG-EMG-SHIELD. Los tres electrodos van designados cada uno por una letra: L el electrodo izquierdo, R el electrodo derecho y D la derivación.

Figura 27: Electrodos pasivos [16]

7.2.5 Otros componentes electrónicos

Además de todo lo mencionado anteriormente, se hace uso de botones, diodos LED y resistencias.

7.2.6 Conexión

Al Arduino se conecta el SHIELD-EKG-EMG y a éste, una placa blanca de prototipado en la que está montado el siguiente circuito. En la imagen siguiente no se muestra el SHIELD-EKG-EMG. Éste, sin embargo, montado sobre los pines del Arduino, y se le conectan los electrodos por el conector *jack*.

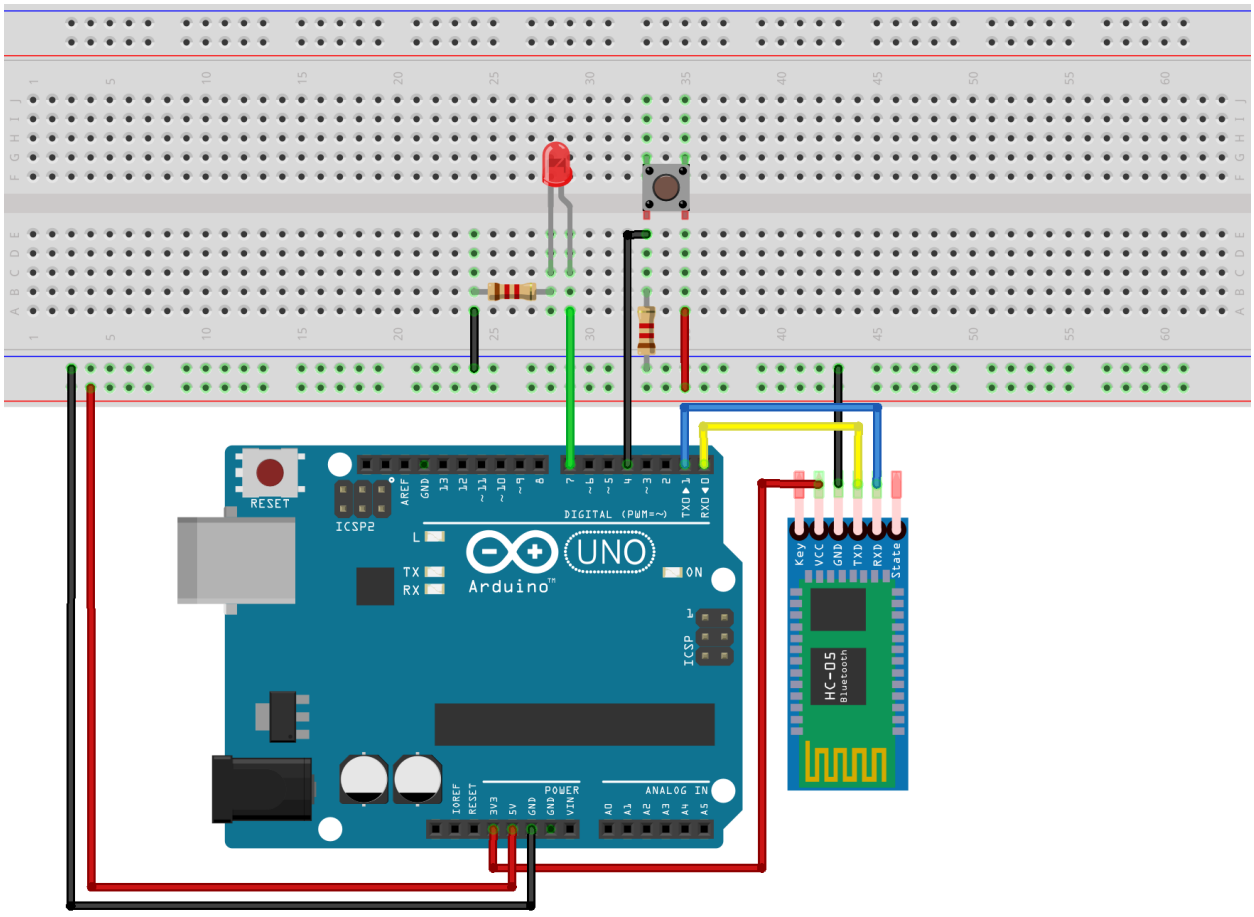


Figura 28: Circuito del arduino

La placa, a través del shield, alimenta a 5V el botón y el LED; y a 3,3V el módulo de comunicación Bluetooth. En serie con el LED existe una resistencia de limitación de corriente de 1,2 KΩ para evitar que se queme el diodo. En un terminal del botón se tienen 5V, y en el otro una resistencia de pull-down de 1,2 KΩ. Entre las bornas de la resistencia y el botón se conecta el pin 4 de la placa, configurado como pin de entrada para detectar la pulsación del botón. En el ánodo (polo positivo del LED) se conecta el pin 7 de la placa, configurado como salida. De esta manera podemos controlar el estado del LED para así mostrar gráficamente el estado en el que se encuentra el sistema. El cátodo del led se conecta con la arriba mencionada resistencia de limitación y ésta con tierra. La tierra también proviene de la placa.

7.3 Funcionamiento

Mientras el microcontrolador está encendido, éste está esperando leer del dispositivo Bluetooth, conectado al puerto serie, a través del cual se le mandarán distintos comandos, normalmente cambios de umbrales de aviso o de estados del sistema. El comportamiento del Arduino se ha desarrollado como una máquina de estados, en la que las transiciones están determinadas por el usuario o por la aparición de una anomalía en el ritmo cardiaco. Los siguientes son los estados en los que puede encontrarse el microcontrolador:

- **Modo de inicialización:** el Arduino al encenderse comienza en este estado. La única funcionalidad que tiene el microcontrolador en este estado es, aparte de la lectura del puerto serie, el parpadeo del LED de estado con una frecuencia de medio segundo, señalizando que está listo para que se le envíe la configuración inicial. Una vez que se ha configurado, éste pasa al estado de aviso en caso de alerta.
- **Modo de aviso en caso de alerta:** en este modo de funcionamiento se lee la señal proporcionada por el EMG-EKG-SHIELD y se trata con el filtro diseñado. A partir de la señal filtrada, se calcula el ritmo cardiaco y en caso de sobrepasar algún límite, ya sea por exceso o por defecto, se inicia la rutina de alerta. Mientas el microcontrolador se mantiene en este estado el LED de estado se encuentra apagado.
- **Modo continuo:** el funcionamiento de este modo es análogo al funcionamiento del modo de aviso en alerta, excepto que cada vez que se calcula la frecuencia cardiaca se envía por puerto serie. Durante el tiempo en el que el microcontrolador se encuentra en este estado, el LED de estado está encendido.
- **Modo de alerta:** tras detectar una anomalía en la frecuencia cardiaca, el microcontrolador entra en este estado. Éste estado es un estado de bloqueo: el microcontrolador no responderá a ninguna de las órdenes pasadas por puerto serie hasta que no reciba la orden de que la alerta se atendió, o que ha sido falsa. En caso de recibir esta señal, el microcontrolador volvería a pasar al modo por defecto, al de aviso en caso de alerta. En este modo el LED de estado parpadea con una frecuencia de una décima de segundo.

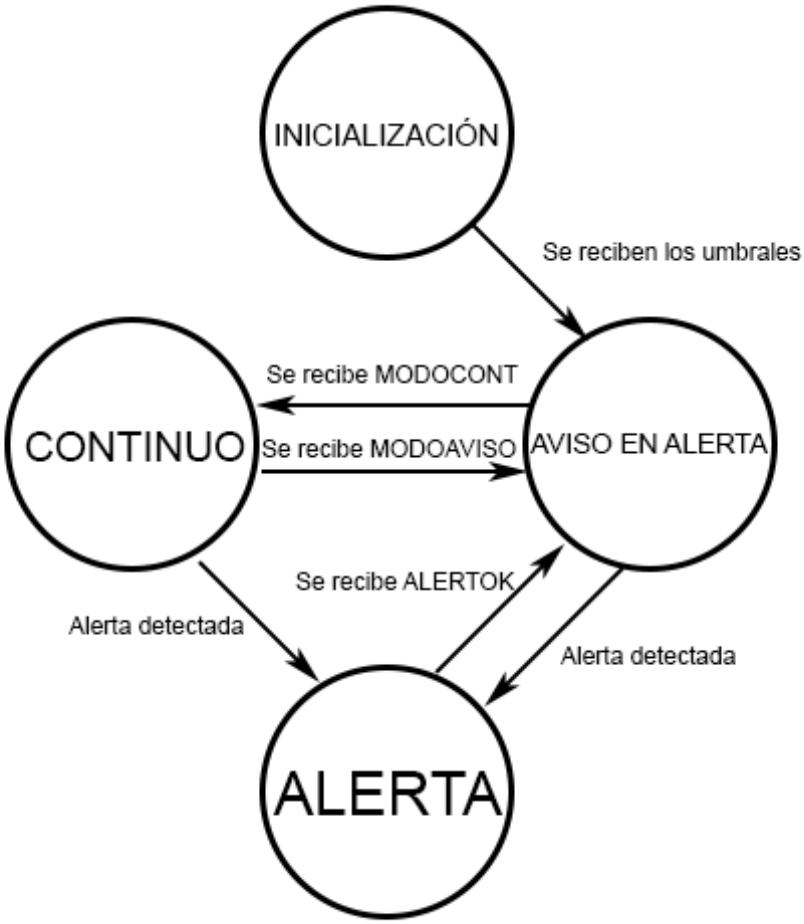


Figura 29: Máquina de estados

El cálculo de la frecuencia cardíaca es la tarea base del sistema, y lo realiza el microcontrolador con ayuda de un temporizador. El uso de un temporizador es para regularizar la frecuencia con la que se toman los datos, ya que si se programara el comportamiento sobre la función **loop**() la medida se tomaría en el momento que fuera posible. Esto es, la frecuencia de muestreo variará en función de la carga de trabajo del microcontrolador, resultando en fluctuaciones de la misma. Esto imposibilitaría el correcto funcionamiento de un filtro sencillo.

La frecuencia del temporizador es la frecuencia de muestreo a la que se ha diseñado el filtro, 5 ms. En caso de que el sistema se encuentre en cualquiera de los dos modos de funcionamiento normal (aviso en caso de alerta o continuo) en cada llamada a la función del timer se realiza el ciclo de aplicación del filtro. Se lee la señal analógica proporcionada por el SHIELD-EKG-EMG, una vez tomada esta medida, se le aplica el filtro explicado previamente. Un pulso es detectado cuando la señal al final del proceso de filtrado supera cierto umbral. Al detectar un pulso cardiaco se calcula la frecuencia cardíaca entre los dos últimos pulsos multiplicando la frecuencia de muestreo por sesenta y dividiendo todo ello entre un contador. Este contador se incrementa cada vez que se llama a la función del timer y se pone a cero cada vez que se detecta un pulso. Una vez calculada la frecuencia cardiaca instantánea se almacena en un vector circular de 15 posiciones. Siempre

que se detecta un pulso, y siempre que el vector esté lleno, se realiza la media aritmética de las 15 últimas frecuencias cardíacas almacenadas en dicho vector. Esta práctica le da robustez a la medida de la frecuencia cardíaca y evita que, en caso de venir dos pulsos seguidos el algoritmo dé una frecuencia falsa. Además, como se comentó anteriormente, se consigue mitigar el efecto del transitorio. Sin embargo, se ha de esperar como máximo 15 pulsos hasta que se detecta el sobrepaso de algún umbral. Esta rutina también se encarga del parpadeo del LED en caso de estar en estado de inicialización o de alarma.

7.4 Explicación del código

Con el objetivo de hacer el código lo más legible posible, se han definido unos macros para los valores que puede tomar la variable **modo**, la que refleja el estado del subsistema. Éstos son: **AVISO**, **CONTINUO**, **ALERTA** e **INICIALIZACION**, tomando valores de **0** a **3**, respectivamente. Igualmente, se encuentran definidos otros macros orientados a proveer facilidad a la hora de cambiar el circuito: **LEDPIN** y **BTNPIN**. Estos dos macros contienen el pin donde se halla conectado el ánodo del LED y el botón. Si se cambiara la distribución de los pines del circuito, sólo habría que cambiar el valor de estos macros en lugar de tener que cambiar todas las llamadas que se le puedan haber hecho a lo largo del código. Para la facilitación de la programación del microcontrolador, se han declarado una serie de funciones orientadas al envío de datos al subsistema Android. Todas ellas siguen un comportamiento similar: declaran una variable local de tipo **String** y la escriben por UART mediante la función **Serial.print**(). Esta función envía por el puerto serie el argumento que se le pase a la función convirtiéndolo (si no lo es en el momento en el que se le pasa) en una variable de tipo cadena. Todas las funciones siguientes funciones siguen el protocolo que se expondrá en el próximo apartado:

- **void** send_bpm (**int** _bpm): esta función está diseñada para enviar una medida de la frecuencia cardíaca. Toma como parámetro la frecuencia cardiaca que se pretende enviar. Se llama cada vez que se calcula la frecuencia cardíaca en modo continuo.

- **void** send_alert (**void**): esta función está pensada para el envío de una alerta. Se llama cada vez que la medida de la frecuencia cardíaca sobrepasa por exceso o por defecto los umbrales especificados.

- **void** send_thr (**int** _hthr, **int** _lthr): está función se ocupa del envío de los umbrales del Arduino al dispositivo Android en caso de que éste último lo solicitase. Toma como parámetro los dos umbrales que se desean enviar. Se llama cada vez que se recibe la orden por parte del dispositivo Android cuando el microcontrolador se encuentra en estado de monitorización continua o de aviso en caso de alerta.

- **void** send_response (**void**): esta función está ideada para notificar al dispositivo Android que se ha cambiado con éxito el umbral que se ha pedido cambiar. Se llama cada vez que se cambia exitosamente un umbral de aviso, ya sea superior o inferior.

Aparte de las funciones arriba mencionadas, también se requiere la función que realizará el temporizador. El temporizador se ha utilizado mediante la librería **FlexiTimer2.h**, no incluida en las librerías por defecto de Arduino. Este temporizador se declara en la función **setup**() mediante el método **FlexiTimer2::set**(**long**, **void (*f)()**). Los parámetros que toma esta función son, en primer lugar, el tiempo que transcurre entre llamada y llamada a la función de timer (*overflow*), y en segundo lugar la propia función de timer. Esta función no puede tener ningún argumento. El temporizador se inicia con el método **FlexiTimer2::start()**.

La función más compleja es la función **timer_fcn**, que como su propio nombre indica, es la que realiza el timer en cada *overflow*. Como se comentó anteriormente, tras comprobar que el sistema se encuentra en un estado de funcionamiento normal, se lee la señal y se aplica el filtro. La señal se guarda en la variable entera **xk**, mientras que la señal filtrada se almacena en la variable de doble precisión **yk**. Mediante la función **abs** se calcula el valor absoluto de **yk** mediante la llamada abs(yk). Seguidamente, se le asigna a la variable **sum** el valor 0, ya que se usará para sumar los 10 últimos valores de la señal para calcular la media móvil. Esto se hace mediante un bucle que, además de sumar los valores del valor absoluto de la señal almacenados en **win**, va actualizando este vector eliminando la última componente, corriendo el resto a la izquierda y añadiendo al final la señal leída. Tras esto, se divide la variable que almacena la suma del vector completo entre la longitud de dicho vector, resultando así el valor definitivo de la señal filtrada.

Como se vio en secciones anteriores, se considera que existe un pulso cuando la señal filtrada supera un cierto umbral umbral, aunque no ésta no es condición única, pues hay que considerar el caso en el que un pulso contenga varios valores mayores al umbral. Es por esto que también ha de cumplirse que la variable entera **flg** esté a 1. Esta variable solo se encuentra a 1 cuando se permite detectar un pulso, en caso contrario está a 0. Al detectar un pulso se le da el valor 0 y tras 20 llamadas a la función del temporizador, se vuelve a permitir la detección del pulso dejando **flg** a 1 de nuevo. La variable que lleva esta cuenta es la variable entera **cont** que se resetea cada vez que llega a 20 y se actualiza en cada vez que se llama a la función del temporizador con **flg** a 0. En el caso de detectarse un pulso, se calcula la frecuencia cardiaca y se almacena en otro vector circular, este de tipo carácter, **bpm_win**. También en este momento se pone a 0 la variable **t**, variable que se va incrementando en cada llamada a la función del timer en la que no se ha detectado pulso. Sirve para llevar una cuenta del tiempo entre pulsos y así poder calcular el tiempo entre ellos. El índice **i**, usado para recorrer el vector **bpm_win**, comienza con valor 0, cada vez que se almacena un valor de frecuencia cardiaca se actualiza y cuando alcanza 15 (longitud total de **bpm_win**) se reinicia a 0. La primera vez que llega a su valor máximo le asigna 0 a la variable entera **primero**, inicializada a 1. Esta variable sirve como variable bandera para comprobar si el vector de frecuencias cardíacas está lleno o no. Si esto es cierto, es decir, primero es igual a 0, se calcula el valor de la media del contenido del vector **bpm_win** y se almacena en la variable tipo carácter **bpm_avg**. En caso de encontrarse el sistema en modo continuo, se enviaría este valor por el puerto serie. Tras esto, se actualizan los valores de las variables empleadas en el cálculo del filtro y finalmente, en caso de que **bpm_avg** excediera los umbrales estipulados, se realizaría una llamada a la función **send_alert**() previamente descrita.

8 PROTOCOLOS

8.1 Protocolo de comunicación Arduino-Android

El microcontrolador se comunica con el dispositivo móvil mediante mensajes codificados en ASCII. El dispostivo Android es capaz de distinguir entre los tipos de mensaje (medida del pulso, alerta, umbrales almacenados o respuesta) gracias al carácter de cabecera, que define el tipo de mensaje que le sigue. Todos los mensajes terminan con el carácter de nueva línea 0x10.

Este protocolo de comunicación, a pesar de no ser el más óptimo en términos de bytes usados, es el más cómodo a la hora de depurar y, de los que se probaron, el único que conseguía una comunicación sin errores entre la placa Arduino y el dispositivo Android. Se intentó implementar un protocolo sin byte de fin de mensaje, pero Android tenía problemas leyendo los mensajes. A continuación, se muestra el protocolo completo.

Tabla 1: Protocolo Arduino-Android

Nombre del macro en el código	Tipo de mensaje	Carácter de cabecera	Formato del mensaje completo
BPM	Medida del pulso	#	#xxx
ALERT	Alerta	!	!
HTHR/LTHR	Envío de umbrales	^	^xxxvxxx
RESPONSE	Respuesta	~	~

- **BPM:** esta cabecera indica que el número siguiente corresponde a una medida de frecuencia cardiaca.
- **ALERT:** esta cabecera indica que se ha excedido algún umbral, ya sea el superior o el inferior. Ya que lo que es necesario transmitir es el hecho de que se ha producido una alerta, este mensaje no lleva más datos.
- **HTHR/LTHR:** esta cabecera señaliza al dispositivo Android que los datos que siguen son los umbrales de aviso que están siendo utilizados. El número entre la cabecera y el carácter "v" corresponde al umbral de aviso superior, y el número entre el carácter "v" y el carácter de nueva línea es el umbral de aviso inferior.

- **RESPONSE:** de nuevo, lo único relevante en este mensaje es la cabecera, por lo que no contiene más datos. Este paquete se usa para señalizar que se han cambiado con éxito los umbrales en el microcontrolador.

8.2 Comunicación Android-Arduino

A diferencia de la comunicación en el sentido opuesto, el microcontrolador no necesita de bytes de terminación de mensaje, por lo que capta correctamente el orden de los bytes. Debido a que los datos que se pasen nunca van a ser de más de 255, es posible utilizar un solo byte como mensaje, en lugar de codificar cada cifra del mensaje como un byte separado, como es el caso de la comunicación en sentido contrario. Por lo tanto, siempre interpretará como cabecera el primer byte que llega y como dato el segundo. Las cabeceras de este protocolo son todos caracteres ASCII de letras mayúsculas para facilitar la depuración mediante consola en la fase de desarrollo. Los nombres son los macros que se le han dado en el código:

Tabla 2: Protocolo Android-Arduino

Nombre del macro en el código	Carácter	Hexadecimal
MODOCONT	A	0x41
MODOAVISO	B	0x42
CAMBIOHTHR	C	0x43
CAMBIOLTHR	D	0x44
PEDIRTHR	E	0x45
ALERTOK	F	0x46

- **MODOCONT:** cambia el estado del Arduino al modo de envío continuo de frecuencia cardiaca, el byte de datos se ignora. Esta orden solo tiene efecto cuando el estado del Arduino es de aviso en caso de alerta.

- **MODOAVISO:** cambia el estado del Arduino a modo de aviso en alerta, el byte de datos se ignora. Análogamente, esta orden solo tiene efecto cuando el microcontrolador se encuentra en modo continuo.

- **CAMBIOHTHR:** señaliza el cambio del umbral de aviso superior, el segundo byte del paquete es el umbral deseado. Esta orden puede realizarse en cualquiera de los dos modos de funcionamientos normales (continuo, aviso en caso de alerta).

- **CAMBIOLTHR:** señaliza el cambio del umbral de aviso inferior, el comportamiento es análogo a **CAMBIOHTHR**.

- **PEDIRTHR:** señaliza la petición de los umbrales almacenados en el microcontrolador, el segundo byte es ignorado, ya que lo relevante del paquete es la cabecera.

- **ALERTOK:** indica que se salga del modo de alerta, bien porque se ha atendido o porque el usuario la ha calificado como falso positivo. El contenido del segundo byte es ignorado.

9 APLICACIÓN ANDROID

9.1 Funcionamiento y composición general de una aplicación Android

Todas las aplicaciones Android están compuestas principalmente por dos tipos de archivos: XML y Java. Mientras que los archivos XML proveen a la aplicación del apartado gráfico, el espacio de nombres, las cadenas de texto u otras propiedades, los archivos Java dotan a la aplicación de funcionalidad: es en ellos en los que se programa el comportamiento de la aplicación y es donde reside la mayor parte de la carga de programación.

XML (eXtensible, Markup Language, lenguaje de marcas extensible) es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C). El objetivo de este lenguaje es proveer de un sistema de etiquetas estructurado de propósito múltiple. Por estructurado se entiende que la información se divide en partes, que a su vez se dividen en otras partes. Entre los usos más importantes del lenguaje XML destacan los feed RSS o los archivos de configuración de las aplicaciones del framework .NET de Microsoft. Apple, a su vez, tiene una implementación de un registro basado en XML. En XML hay tres tipos de etiqueta: de comienzo <parte>, de final </parte> y etiquetas de elementos vacíos <elemento />.

Java es un lenguaje de programación orientado a objetos y basado en clases. Se basa en el principio WORA (write once, run anywhere), es decir, que una vez escrito en una plataforma, no ha de ser compilado de nuevo para correr en otra plataforma. Este lenguaje fue creado por James Gosling de Sun Microsystems (adquirida por Oracle). La sintaxis de este lenguaje de programación proviene en gran parte de C y C++, aunque implementa menos utilidades de bajo nivel, como, por ejemplo, los punteros. Las aplicaciones Java pueden ser ejecutadas en cualquier máquina virtual java (JVM, por sus siglas en inglés) sin importar la arquitectura del procesador. En java se pueden definir variables de las clases como estáticas. Esto significa que sólo se creará una instancia de la variable, sea cual sea el número de instancias que se declaren de la clase que lo contiene. También se pueden definir como finales, lo que significa que sólo se puede dar valor una vez, aunque no tiene por qué ser inicializada al declararse. También se pueden definir las variables y métodos como privados, protegidos o públicos. Si una variable o método se define como privado, su acceso queda restringido a la clase que lo contiene; en lugar de esto, si se define como protegido/a, el acceso a éste método se extiende a las subclases de la clase que lo contiene. Finalmente, si se declara como público/a, el acceso al método o variable está permitido a todo el mundo.

En Android Studio, el IDE que se ha utilizado para desarrollar la aplicación, al crear un nuevo proyecto de aplicación, el entorno genera automáticamente una carpeta en el disco duro de la máquina que está ejecutando el IDE. Todos los dados a ser modificados o creados se encontrarán en el directorio nombre_del_proyecto/app/src/main.

En la carpeta raíz se encuentra **AndroidManifest.xml**, el archivo XML que define gran cantidad de parámetros de la aplicación, entre ellos se incluyen, por ejemplo, la ubicación de la instalación de la aplicación en el dispositivo móvil, los permisos que se le conceden a la aplicación (a los componentes del móvil que la aplicación tiene derecho a acceder), la versión de la aplicación, el nombre… También en este fichero se encuentran recogidas todas las actividades. En Android, una actividad es lo que comúnmente se definiría como "pantalla". Cualquier error en la sintaxis de este archivo y la aplicación no se ejecutará. La aplicación fallará al arrancar también si dentro del código se hace uso de algún elemento que requiera algún permiso especial y no se haya especificado en este archivo.

En este directorio raíz se halla un directorio llamado **res** (de *resources*, recursos en inglés). En el directorio **res** se encuentran las imágenes y los ficheros XML que van a determinar el carácter visual de la aplicación. Dentro de **res**, se pueden hallar cuatro carpetas que comienzan por **drawable**. Estas contienen tanto las imágenes que puedan ser mostradas por la aplicación, como el icono de la misma. El porqué de la existencia de cuatro carpetas es la flexibilidad del sistema Android. Ya que Android está instalando en gran variedad de dispositivos con grandes cantidades de resoluciones de pantalla es impráctico usar las mismas imágenes en unas pantallas y en otras: una misma imagen se verá más grande en una pantalla con menos densidad de píxeles que en una con una densidad mayor. Por tanto, pantallas con mayor densidad de píxeles necesitan imágenes mayores. Los sufijos que siguen a los nombres de estas carpetas son **mdpi**, **hdpi**, **xhdpi**, **xxhdpi** y **xxxhdpi** (medium, high, extra high, extra extra high, extra extra extra high dots per inch; puntos por pulgada medios, altos, extra altos, extra extra altos y extra extra extra altos). En función de la densidad de puntos de la pantalla de cada dispositivo usará las imágenes de una u otra carpeta. Dentro de **res** también existe una carpeta llamada **layout**. En esta carpeta se almacenan los XML que confieren de carácter visual a las actividades: contienen los widgets (botones, campos de texto…) y la ordenación que éstos deben tener. También existe una carpeta llamada **menu**, en la que se almacenan los diversos XML que confieren a las actividades con un menú. Estos XML solo definen un orden de elementos y, en todo caso, un submenú. La siguiente subcarpeta en el directorio es **values**, en esta se encuentran tres ficheros XML: **dimens**, **strings** y **styles**. El primero sirve para definir las dimensiones que se pueden usar en el apartado visual de una aplicación, como márgenes y espaciados. El segundo contiene todas las cadenas de texto de la aplicación, cada una con una etiqueta. Esta práctica permite la fácil traducción de las aplicaciones, pues solo hay que añadir un nuevo fichero XML con los mismos nombres en las etiquetas y otro contenido. Finalmente, en el archivo **styles** es posible definir diversos temas gráficos para la aplicación. La última de las subcarpetas de **res** es **xml**. En esta carpeta entran todos los xml adicionales que puedan necesitarse.

En el directorio principal se encuentra una carpeta llamada **java**. Dentro de ella se encuentran todos los archivos java de los que hace uso la aplicación, generalmente, uno por clase.

Por último, cabe mencionar el archivo **R**, este es el que contiene el espacio de nombres de la aplicación. **R** contiene las diversas ids de los elementos gráficos (botones, menús… etc.) así como las direcciones de memoria de estos. Este archivo lo genera y actualiza automáticamente el entorno, por lo que no es ni necesario ni recomendable modificar el archivo, ya que la más mínima discrepancia puede hacer que la aplicación no se arranque.

9.2 Ciclo de vida de una actividad en Android

En Android, generalmente se identifica actividad con una pantalla, ya sea a pantalla completa, u ocupando una porción rectangular de la pantalla del dispositivo. El estado de ejecución de una actividad está regido por su ciclo de vida. Se denomina ciclo de vida de una actividad a la serie de estados de ejecución por la que dicha actividad puede pasar. La transición entre estados está controlada en parte por el usuario y en última instancia, por el sistema. Una actividad en ejecución puede encontrarse en tres estados: en ejecución en primer plano, en ejecución en segundo plano y en segundo plano sin proceso. Las transiciones entre estados provocan eventos (identificados con métodos de la clase **Activity**) que el programador puede utilizar para controlar ciertas acciones ligadas al ciclo de vida de la actividad, como, por ejemplo, hacer una copia de seguridad.

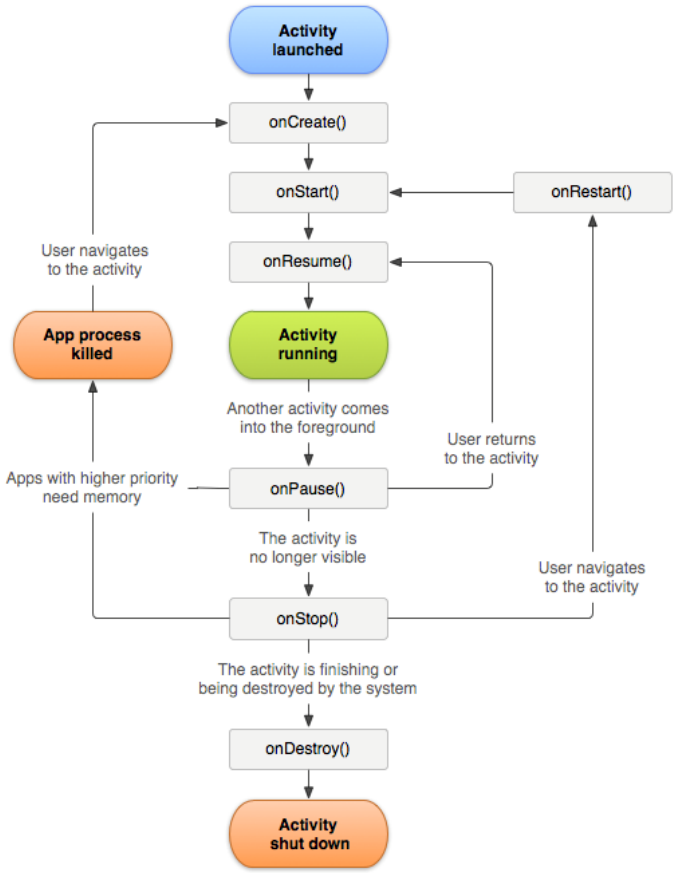


Figura 30: Ciclo de vida de una actividad en Android [17]

Cuando la actividad comienza, antes de que la actividad se muestre, ocurren tres eventos: **onCreate**, **onStart** y **onResume.** El primero de los eventos, **onCreate**, se dispara cuando la actividad se crea. En éste método deben incluirse las inicializaciones requeridas para la aplicación y la construcción de la interfaz gráfica de usuario. El segundo, **onStart**, se llama después de **onCreate** o después de **onRestart**. La diferencia entre éste y **onCreate** es que **onCreate** sólo se llama una vez, mientras que **onStart** se llama siempre que la aplicación vuelve a ser visible para el usuario. El último de los tres eventos que se disparan al comenzar, **onResume**, se llama cuando la actividad comienza a interactuar con el usuario. En éste método se deberían programar los accesos a los dispositivos de acceso exclusivo, como la cámara, o comenzar a dibujar las animaciones que la actividad pudiera incluir. Después de ejecutarse estos tres métodos, la actividad entra en modo de ejecución. Si desde este estado se inicia una nueva actividad, se llama al método **onPause**. Este método es antagónico a **onResume**, se llama cuando la actividad está a punto de ir a segundo plano. Si una actividad A llama a una actividad B, la actividad B no se iniciará hasta que se haya completado la llamada de **onPause** en la actividad A. Normalmente, la llamada a **onPause** va seguida de una llamada a **onStop**, aunque, como se explicará más adelante, en algunos casos pudiera suceder que la actividad volviera a primer plano antes de la llamada a **onStop**. Esta vuelta a primer plano la realizaría con otra llamada a **onResume**. El método **onSto** se llama cuando la actividad deja de mostrarse al usuario. En situaciones en las que no hay demasiada memoria disponible podría ocurrir que el sistema destruya directamente el proceso y no se llame a este método después del **onPause.** Después del método **onStop**, si la actividad vuelve al primer plano, se llama al método **onRestart** antes de que esto ocurra, y tras la llamada a **onRestart**, como se mencionó previamente, se vuelve a llamar a **onStart**. También podría suceder que el sistema requiera los recursos invertidos en la actividad, por lo que se detendría el proceso de la actividad. Si el usuario reabriera la actividad, se volvería a crear desde cero, esto es, llamando a **onCreate**. Desde **onStop** también se puede cerrar definitivamente la aplicación. Si esto sucede, se llama al método **onDestroy**, que es la última llamada antes de que se destruya por completo la actividad. Este método se debería utilizar como medio para realizar las últimas limpiezas en la actividad, no como último lugar para salvar datos. Se suele implementar este método para liberar recursos como *threads* de ejecución, de manera que, si la aplicación se destruye, no se queda el hilo en segundo plano consumiendo recursos.

9.3 Intents

En Android, la comunicación entre clases se realizan mediante Intents. Un Intent puede entenderse como una retransmisión a nivel global. Un Intent se diferencia de otro por su acción: la acción de un intent es una variable tipo cadena signada a éste, que hace las veces de "nombre" del Intent. Además de acciones, los Intent pueden llevar extras: datos que puede llevar el Intent y ser extraídos por cualquier clase que lo escuche. Estos extras pueden ser de casi todos los tipos de variables. Una clase puede escuchar un Intent determinado si implementa un filtro para la acción del Intent en cuestión.

El sistema libera un Intent cuando ocurre algún evento en el dispositivo en base al cual haya que actuar. Para el desarrollo de esta aplicación, se escuchan algunos de los Intents correspondientes a la conectividad Bluetooth: los que indican cambio en el estado del adaptador Bluetooth, así como los que indican la conexión y desconexión de un dispositivo Bluetooth.

A su vez, el desarrollador puede no solo escuchar Intents sino enviarlos. El envío de un Intent puede hacerse para tres fines distintos. El primero: para interactuar con el usuario a través del sistema. Existen funcionalidades en Android sobre las cuales no se puede tomar el control directamente, es necesario pedir permiso al usuario. Para que el usuario pueda conceder el permiso, se notifica al sistema mediante un Intent el tipo de petición que se quiere realizar. El sistema hace de intermediario entre la aplicación y el usuario para así evitar posibles comportamientos maliciosos. El segundo fin es establecer un sistema de notificaciones entre clases, con posible intercambio de datos. Por ejemplo, si se crea una clase cuyo propósito es realizar una tarea en concreto, esta clase puede notificar a otra que esté esperando el resultado de la tarea que ha terminado de realizarla mediante un Intent, y puede incluir el resultado de dicha tarea en un extra. Finalmente, el tercer propósito de los Intents es para iniciar actividades nuevas: se configura la acción como la clase de la actividad que se quiere inciar y se retransmite el Intent.

9.4 La aplicación

9.4.1 Descripción general

Se ha bautizado tanto a la aplicación como al sistema compuesto por el microcontrolador, el shield y el circuito SiAMP (Sistema de Aviso y Monitorización del Pulso). La aplicación se encargará de pasarle al microcontrolador los umbrales para el pulso, pasados los cuales, el microcontrolador asumirá que se ha provocado una alerta; de monitorizar esta medida del pulso, y de varias tareas más que se explicarán en detalle más adelante.

Figura 31: Icono de la aplicación SiAMP

9.4.2 Apariencia visual

La aplicación SiAMP tiene como funcionalidad proveer al usuario con una interfaz gráfica con la que poder comunicarse con el microcontrolador. Para aprovechar todas las funcionalidades instaladas el dispositivo Android debe tener conexión Bluetooth, GPS y acceso a la red telefónica para enviar SMS. La aplicación se compone de tres actividades.

9.4.2.1 Actividad principal

La actividad principal se compone de tres elementos gráficos básicos: un botón tipo *switch* y dos campos de texto, uno de ellos oculto, situado a la derecha del *switch* y otro situado debajo de él. El primer campo de texto informa de la frecuencia cardíaca si el sistema se encuentra en modo continuo. El segundo, del estado de la conexión con el microcontrolador o de los detalles de la alerta en caso de darse. El botón tipo *switch* permite cambiar de modo continuo a modo de aviso en alerta y viceversa. Este botón se encuentra deshabilitado hasta el momento en el que la conexión Bluetooth con el microcontrolador se establece y el microcontrolador se inicializa.

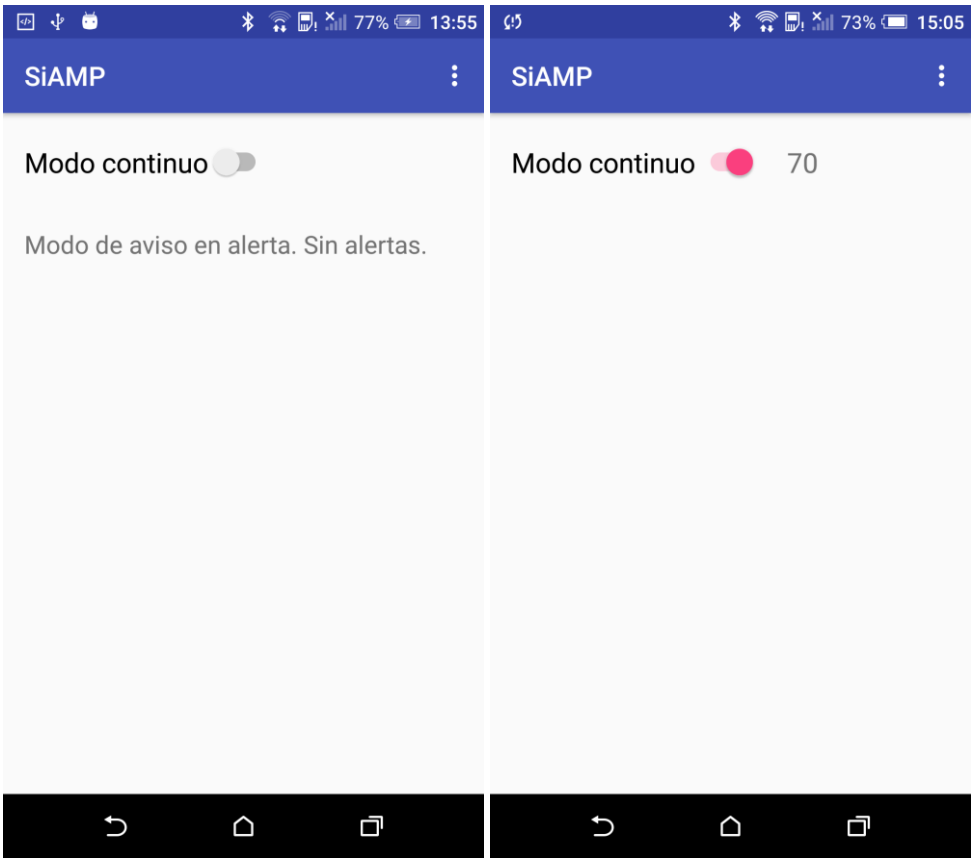


Figura 32: Actividad principal: modo aviso en alerta y continuo

Esta actividad también posee un menú, desde el que se puede conectar al dispositivo, pedir los umbrales de aviso almacenados actualmente en el microcontrolador, acceder a la actividad de configuración y reiniciar el estado del microcontrolador. Las opciones del menú se habilitan o deshabilitan de forma dinámica dependiendo del estado de la conexión con el microcontrolador: la opción de "Conectar SiAMP" solo está habilitada cuando no hay conexión con SiAMP, mientras que "Pedir umbrales de aviso" y "Configurar SiAMP" están habilitadas únicamente cuando hay conexión y el sistema está inicializado. Finalmente, la opción de reiniciar sólo se activa cuando el sistema haya informado de una alerta. Ésta última opción no reinicia realmente el microcontrolador, sino que el teléfono envía al microcontrolador el carácter "~" y el sistema vuelve al modo de aviso en caso de alerta.

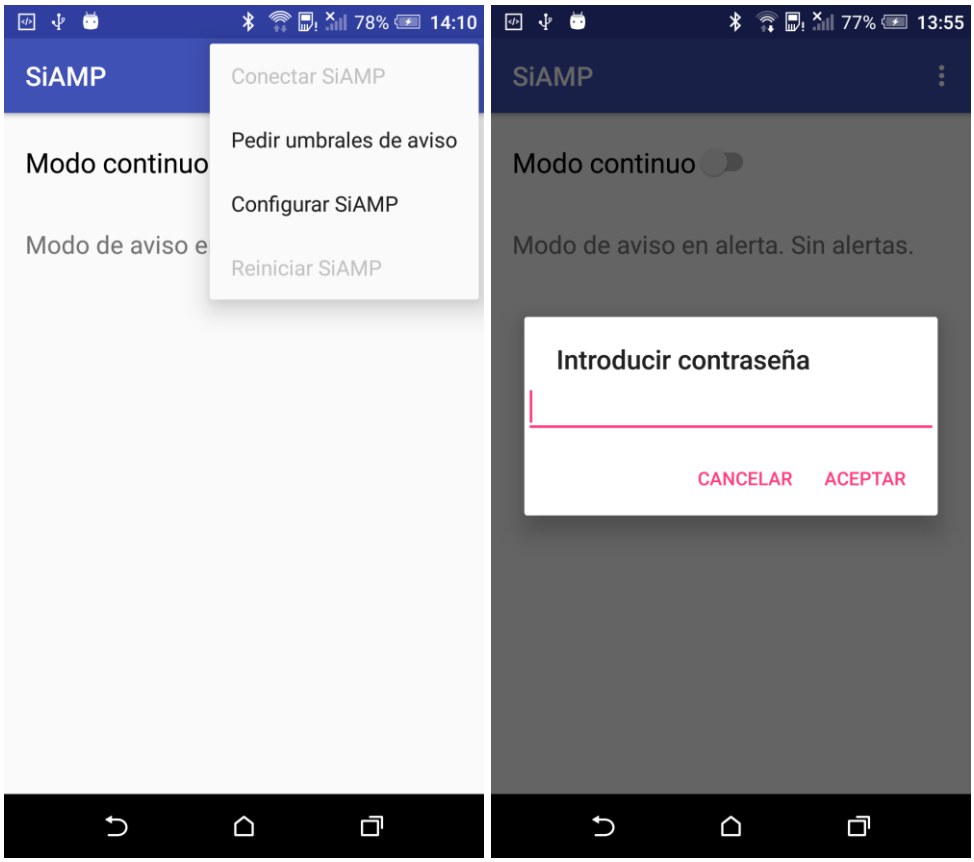


Figura 33: Actividad principal: menú y *pop-up* de contraseña

9.4.2.2 Actividad de configuración

Desde esta actividad es posible configurar diversos parámetros del sistema. Se accede a ella desde el ítem de menú de la actividad principal "Configurar SiAMP". Por motivos de seguridad, para acceder a esta actividad se requiere una contraseña. La contraseña por defecto es **aaaa**. Esta contraseña es almacenada en la configuración de la propia aplicación. Una vez introducida correctamente la contraseña se accede a la actividad de configuración, se tiene acceso a cuatro opciones:

- **Nuevo umbral de aviso superior:** al seleccionar esta opción la aplicación muestra un cuadro de diálogo con un cuadro de texto numérico en el que introducir el nuevo umbral de aviso. Al tocar en aceptar la aplicación comprueba que este número no excede 255, que es el máximo valor que se puede transmitir en un byte, en caso contrario, configura el valor del umbral como 255. La orden de cambiar el umbral no se realiza inmediatamente, sino cuando se vuelve a la actividad principal. Cuando esto sucede, se envía un paquete **CAMBIOHTHR** con el nuevo umbral como dato. Si todo ha ido correctamente, el microcontrolador responderá con un mensaje de respuesta (~). Cuando la aplicación reciba este mensaje, avisará por un mensaje tipo *toast* (mensaje desplegado desde el borde inferior de la pantalla) indicando que los umbrales se han cambiado con éxito.

- **Nuevo umbral de aviso inferior:** el funcionamiento es análogo a la opción de menú superior.
- **Nombre del paciente:** al seleccionar esta opción de menú la aplicación vuelve a desplegar un cuadro de diálogo con un cuadro de texto, esta vez alfanumérico, en el que se pide que se introduzca el nombre del paciente. Esta información se guarda en la aplicación y sirve para, en caso de alerta, avisar del nombre del paciente a través de un SMS.
- **Cambiar la contraseña:** para mayor seguridad, se permite la opción de cambiar la contraseña. Al seleccionar esta opción se abre la tercera y última actividad de la aplicación: la actividad de cambio de contraseña.

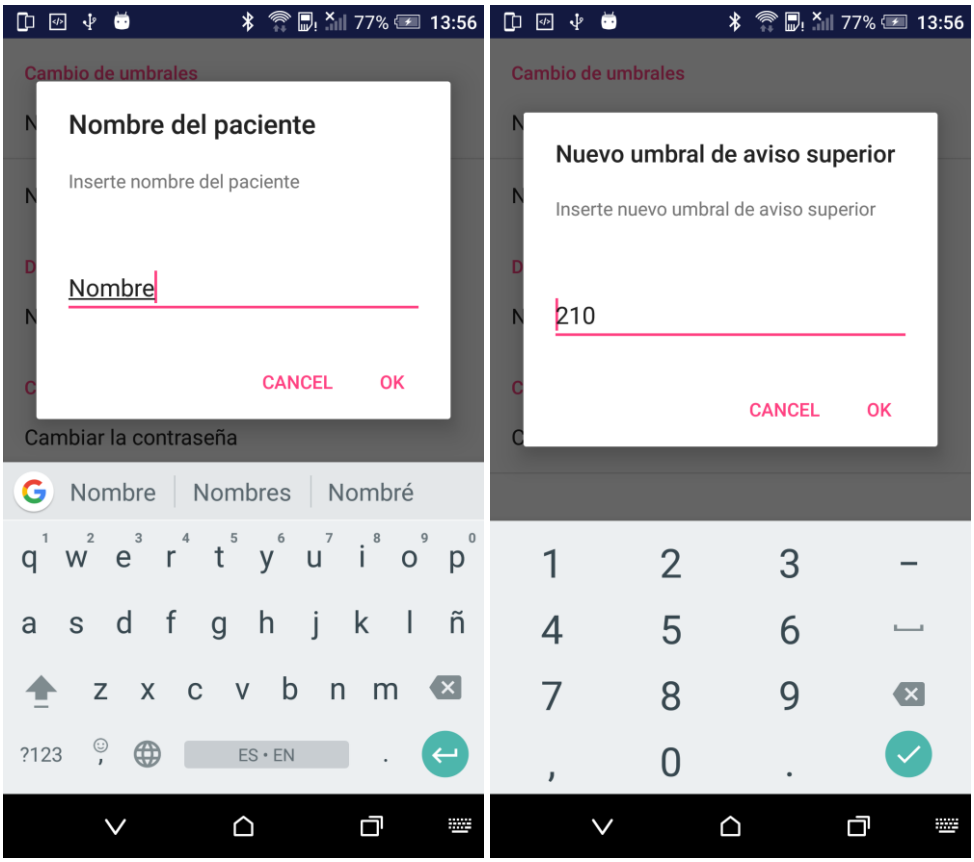


Figura 34: Actividad de configuración: nombre del paciente y umbral

9.4.2.3 Actividad de cambio de contraseña

Esta actividad dispone de tres campos de textos alfanuméricos: contraseña actual, nueva contraseña y confirmar contraseña, todos ellos configurados como contraseña para evitar que se pueda ver el contenido. También contiene dos botones: confirmar cambios y atrás. Al introducir todos los datos pedidos y pulsar sobre confirmar cambios la aplicación comprueba que la contraseña original es correcta, que las dos nuevas contraseñas coinciden y que tienen más de cuatro caracteres. Si todo se cumple, la contraseña se actualiza, se notifica al usuario de que la operación de cambio de contraseña se ha cambiado con éxito mediante un toast y la aplicación vuelve a mostrar la actividad de configuración.

Si se alguna de las condiciones no se cumpliera, se avisa al usuario por medio de un *pop-up*. En caso de pulsarse el botón "Atrás", el sistema ignora los campos de texto, no guarda ningún cambio y vuelve a la actividad de configuración.

Figura 35: Actividad de cambio de contraseña

9.4.3 Ciclo de funcionamiento

La primera vez que se inicia la aplicación, el sistema pide al usuario la concesión del permiso de acceso a la localización. Esto se hace así desde el nivel 23 de la API, cuando se definen algunos permisos como "peligrosos". Los permisos peligrosos son aquellos que pueden poner la privacidad del usuario en riesgo, y por ello, el usuario tiene que dar su consentimiento explícito para que la aplicación acceda a la información en cuestión.

Al iniciarse la aplicación y pulsarse sobre el ítem de menú "Conectar SiAMP" el sistema comprueba que el dispositivo cuenta con un adaptador Bluetooth y que éste está encendido. En caso de no estarlo, la aplicación pide al usuario que encienda la conectividad Bluetooth mediante un cuadro de diálogo del sistema. Esto es así ya que en Android no se puede forzar la habilitación de la conectividad Bluetooth: el usuario ha de dar la autorización. Tanto si se ha encendido el Bluetooth mediante el cuadro de diálogo del sistema o se encontraba

encendido antes de pulsar “Conectar SiAMP”, la aplicación comienza a buscar dispositivos Bluetooth cercanos con el nombre esperado. En caso de encontrar el SiAMP (y de que éste esté emparejado con el dispositivo Android) la aplicación configura la comunicación entre ambos. Cuando se termina de conectar, la aplicación comienza el proceso de inicialización.

La inicialización consiste en el envío de los umbrales de aviso superior e inferior. Una vez que la conexión se ha terminado de configurar, se inicia un temporizador de 3 segundos. Cuando el temporizador llega a su fin, se envía el primer umbral al microcontrolador. Se realiza esta espera porque, en ocasiones, el microcontrolador no recibía correctamente la primera transmisión por realizarse antes de que estuviera completada la conexión Bluetooth del lado del microcontrolador. Una vez enviado el primer umbral, se espera a la recepción del mensaje de respuesta para enviar el segundo umbral. Cuando se recibe el segundo mensaje de respuesta, el proceso de inicialización se da por terminado.

En todo momento de la conexión se informa al usuario del estado de esta a través de mensajes tipo Toast: la conexión del dispositivo, el comienzo del proceso de inicialización y el final de éste. Cuando la conexión termina de inicializarse, se habilita el botón tipo *switch* que permite el cambio de modo de funcionamiento del microcontrolador. Si por cualquier motivo la conexión Bluetooth se rompiera, la aplicación avisaría al usuario por medio de una notificación de tipo *ongoing* (no se puede descartar), indicando que el dispositivo se halla desconectado. Esta notificación también aparecerá si se cierra la aplicación. Al tocar sobre dicha notificación se abre la actividad principal, independientemente de desde qué pantalla se haya tocado la notificación. Al volver a conectarse el dispositivo, la notificación desaparece.

Si el dispositivo recibe una alerta por medio de un mensaje de alerta, la aplicación mostrará por pantalla un pop-up, notificando que se ha recibido una alerta. Este *pop-up* incluye un temporizador de cuenta atrás de 15 segundos. En caso de que el temporizador llegara a cero se iniciaría el proceso de envío de la alerta. El usuario puede elegir cancelar la alerta o enviarla mediante los botones en la parte inferior del *pop-up* que se muestra en pantalla. Si dicha alerta ocurre fuera de la aplicación, el usuario recibe una notificación, que al pulsar sobre ella le manda a la aplicación donde se le muestra el pop-up. En caso de elegir no enviarse la alerta, la aplicación envía un paquete tipo ALERTOK para devolver al microcontrolador al estado de aviso en caso de alerta, el *pop-up* desaparece y el temporizador se detiene de modo que no envíe la alerta. Si la alerta se envía, ya sea por inacción o por pulsar el botón de envío; se inicia la secuencia de envío de alerta. Ésta consiste en recibir del GPS del dispositivo la última posición conocida y leer la hora del reloj del sistema. Estos datos se muestran en el campo de texto estático de la actividad principal. Además de esto, la aplicación, a través del servicio de SMS del teléfono móvil, envía los datos de la localización, la hora del suceso y el nombre del paciente a un número de teléfono (si llegara a ponerse en uso, los servicios de atención sanitaria). Después de esto, se habilita en el menú de la actividad principal la opción de reiniciar el sistema para volver al uso normal. El éxito o fracaso en la operación de envío de SMS se notifica al usuario mediante notificaciones y mensajes de tipo Toast.

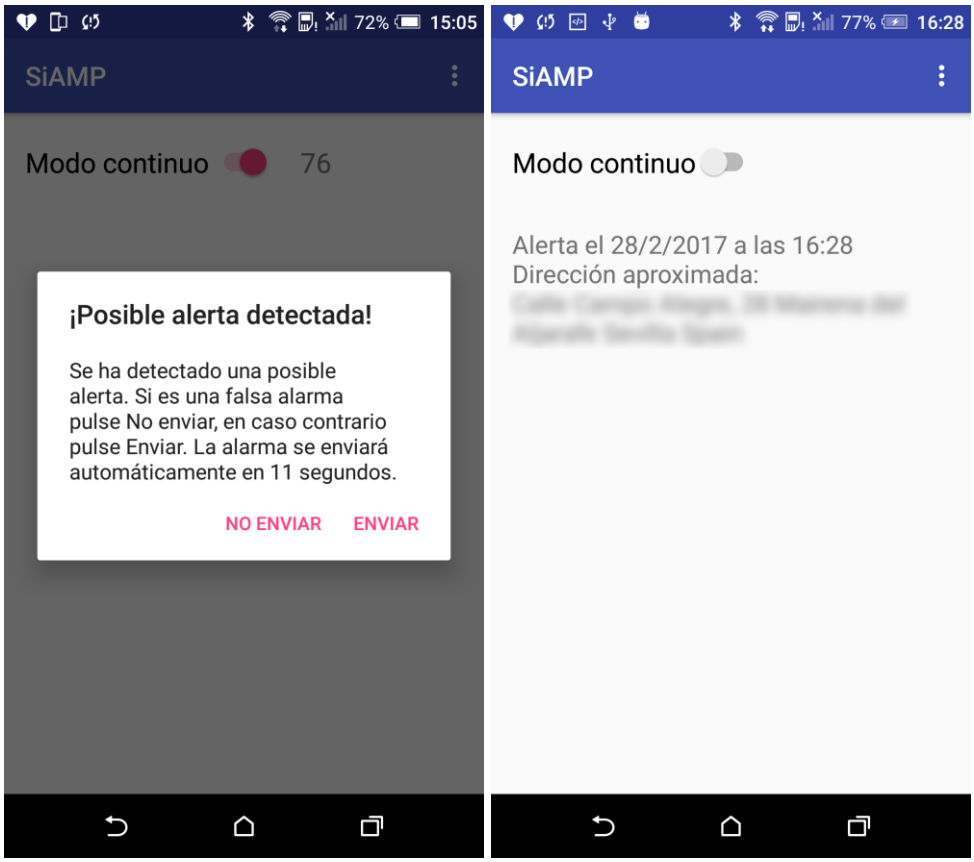


Figura 36: *Pop-up* de alerta y actividad principal al recibirse una alerta

9.5 Funcionamiento de la conectividad Bluetooth

Como se ha explicado antes, al abrir la aplicación y pulsar sobre el ítem de menú "Conectar SiAMP" se comprueba que la conectividad Bluetooth esté encendida. En esta sección se entrará en más detalle acerca del funcionamiento del Bluetooth. Para la comunicación Bluetooth, se tienen las funciones auxiliares **FindSiAMP**, **SetupBTConnection**, **StartListening** y **SendSiAMP**.

La primera en ejecutarse es **FindSiAMP**. Esta función no tiene parámetros y no devuelve nada. El papel de esta función es instanciar el adaptador Bluetooth del dispositivo Android y comprobar que éste está habilitado. En caso de no estarlo, como se expuso anteriormente, pide su habilitación a través de un cuadro de diálogo del sistema mediante un Intent de acción **BluetoothAdapter.ACTION_REQUEST_ENABLE**. Cuando el Bluetooth se enciende o se apaga, el sistema retransmite un Intent de acción **BluetoothAdapter.ACTION_STATE_CHANGED**, que la aplicación esperará escuchar para continuar con la conexión. Cuando ha ocurrido esto (o bien la conexión Bluetooth estaba habilitada antes de intentar realizar la conexión con SiAMP), la función busca entre los dispositivos emparejados con el sistema el que se corresponda con el nombre especificado. Cuando lo encuentra, guarda este dispositivo en una variable miembro de la clase de la actividad principal para que se prosiga la conexión más adelante.

A la ejecución de la primera la sigue **SetupBTConnection**, también sin parámetros ni valor devuelto. Esta función define una UUID para el dispositivo Android en la conexión Bluetooth y toma el *socket* de comunicación del dispositivo que guardó la función anterior y lo guarda en otra variable miembro de la clase. De este *socket* extrae dos *streams* de datos, uno de entrada y uno de salida y los vuelve a guardar en las variables miembros correspondientes. Tras hacer esto, esta función llama a **StartListening**, que se explicará en el párrafo siguiente. Después de esta llamada, cancela la notificación que informa al usuario de que no hay conexión con SiAMP, espera tres segundos y comienza el proceso de inicialización mandando el primer umbral.

La función **StartListening** es la función que se ocupa de la recepción de datos y de la notificación de la llegada de éstos mediante un Intent al hilo de ejecución principal. Una vez más, no recibe parámetros y no devuelve nada. Esta función declara una instancia de la clase Thread (hilo de ejecución). Este hilo de ejecución realizará la lectura de los datos del *stream* de entrada del *socket* del dispositivo Bluetooth. Se hace uso de un hilo de ejecución porque la llamada de lectura es bloqueante, y si se hiciera desde el hilo de ejecución principal se bloquearía la interfaz gráfica de usuario hasta recibir algún dato, momento en el que se volvería a bloquear. Este hilo de ejecución comprueba que haya bytes disponibles para ser leídos en el *stream* de datos de entrada. Cuando haya alguno, los lee uno a uno. Cuando el byte leído es el byte de terminación (0x10, especificado en la propia función **StartListening**) crea un vector de tipo byte, con tantas posiciones como bytes se hayan leído desde el último mensaje. Seguidamente, vuelca los bytes que han ido entrando por el *stream* al vector que se ha creado. Una vez hecho esto, mediante un Handler (herramienta que permite a los hilos el envío de mensajes) libera un intent con la acción **DATA_RECEIVED** y añadiéndole el vector de bytes leídos como extra. Esta acción la escuchará el hilo de ejecución principal, extraerá los datos y actuará en función del tipo de datos que sean.

La finción **SendSiAMP** se encarga de enviar los datos al SiAMP. Toma dos parámetros de tipo entero y no devuelve nada. Esta función realiza una tarea relativamente simple: crea un vector de dos posiciones de tipo byte, y asigna el primer parámetro al primer byte, y el segundo parámetro al segundo byte, haciendo en ambos casos una conversión de entero a byte. Una vez hecho esto, escribe en el *stream* de salida este vector.

Finalmente, en el método **onDestroy** de la actividad principal, se señaliza que se detenga el hilo de ejecución que lee del *stream* de datos de entrada y se cierran los *streams* de datos de entrada y salida, así como el *socket* de conexión.

9.6 Funcionamiento del GPS

De la funcionalidad GPS se encarga la clase auxiliar **GPSManager**. Cuando se crea una instancia de la clase **GPSManager** y se llama a la función miembro **getLocation**, el programa, a través del servicio de localización del dispositivo móvil, comprueba que está encendido el GPS y la red móvil. Si ninguna de las dos está operativa, la aplicación no puede proveer una localización y no devuelve nada. En caso de que estén habilitadas alguna de ellas (o las dos), toma la posición del proveedor de red. En caso de no poder obtenerse

por este método, toma la posición provista por el GPS. Esto se hace así siguiendo la recomendación de Google en su página para desarrolladores de Android, ya que la posición provista por el GPS puede no estar disponible en todo momento (ésta no funciona cerca de edificios altos y requiere conexión directa con el satélite), consume más batería y puede tardar más en obtenerse. Debajo se muestra una tabla con las características de los dos métodos de obtención de la posición [18].

Tabla 3: Características de los proveedores de localización

Tecnología	Precisión	Consumo	Características
GPS autónomo, provisto por el GPS	6 m	Alto	• Usa el chip GPS del dispositivo • Necesita estar en el campo visual de los satélites • Necesita conexión a varios satélites a la vez • Requiere tiempo obtener la posición
GPS asistido (AGPS), provisto por la red	60 m	Medio-Bajo	• Usa el chip GPS del dispositivo, así como información de la red móvil para dar una posición inicial rápidamente • Consumo de energía muy bajo • Muy preciso • Funciona sin tener el cielo en el campo visual • Depende de que el teléfono y la operadora soporten este método

9.7 Funcionamiento de los SMS

El envío de SMS se hace a través de la función **sendSMS**. Primero, se declaran dos instancias de PendingIntent. Un **PendingIntent** es una descripción de un Intent y de la acción que se desea realizar con él. Uno de los PendingIntent estará destinado a escuchar si el SMS se ha enviado con éxito, mientras que el otro lo estará a escuchar si ha sido recibido con éxito. También implementa un receptor para cada PendingIntent. En cada uno de estos receptores, se tiene en consideración la casuística completa del envío (envío correcto, error genérico, falta de cobertura, red deshabilitada, o protocolo inválido) y recepción del SMS (recibido, no recibido), para así proveer al usuario con la información más detallada posible. Implementados los receptores, se crea una instancia del gestor de SMS por defecto del sistema y se le pasa el mensaje a enviar, el número de teléfono del receptor y los dos PendingIntent mediante una función para que éste se ocupe del envío.

9.8 Archivos XML

Como se ha expuesto anteriormente, el apartado gráfico de una aplicación en Android viene definido por los archivos XML dentro de la carpeta **layout**. Una actividad puede componerse de uno o varios archivos de este tipo, dando así lugar a un diverso rango de opciones para el apartado gráfico de las aplicaciones. A continuación, se van a comentar los diversos archivos de distribución usados en la aplicación.

9.8.1 Actividad principal

9.8.1.1 /layout/activity_main.xml

Este archivo es uno de los dos que dotan a la actividad principal de interfaz gráfica. Contiene dos elementos: la barra de tareas, la cual está presente únicamente en esta actividad y que proporciona acceso al menú de configuración; y una referencia al segundo archivo que da forma a la actividad principal, **content_main.xml**, que se explicará en la sección siguiente

9.8.1.2 /layout/content_main.xml

Como su propio nombre indica, este archivo es en el que se encuentra la información sobre el contenido de la actividad principal. Define el comportamiento de los bordes de la pantalla, la orientación que debe tomar, así como las declaraciones y propiedades de los elementos de la actividad principal: el campo de texto principal, el campo de texto para la frecuencia cardíaca y el botón tipo *switch* que permite cambiar el modo de funcionamiento del microcontrolador.

9.8.1.3 /menu/menu_main.xml

En este archivo se definen los ítems del menú presentes en el menú desplegable de la actividad principal. A cada ítem le corresponde una entrada, con su cadena de texto a mostrar y un número que determinará el orden en el que aparecerán los ítems de menú.

9.8.2 Actividad de configuración (/xml/preferences.xml)

Android permite la creación de actividades de configuración (preferencias) a partir únicamente de un archivo xml. También posibilita agrupar las preferencias en categorías. A cada una de las preferencias se le puede asignar una entrada de **SharedPreferences**, que, como se verá más tarde, es la forma de guardar datos en la propia aplicación. De esta forma, se ha asignado una preferencia al nombre del paciente y al umbral de aviso superior e inferior. Al seleccionar las preferencias Android ofrece la posibilidad de abrir un cuadro de diálogo para editarlas, con campos de entrada personalizables en función del tipo de preferencia (numérica, alfanumérica... etc). Adicionalmente, permite asignar a una preferencia un Intent en lugar de una entrada de **SharedPreferences**, por lo que posibilita iniciar una actividad distinta desde la actividad de preferencias. Es así como se lanza la actividad de cambio de contraseña desde la pantalla de configuración.

9.8.3 Actividad de cambio de contraseña (/layout/activity_password_change.xml)

Este archivo contiene la información de los elementos presentes en la actividad de cambio de contraseña: los tres campos de texto estático, los tres campos de texto editables y los dos botones.

9.9 Clases

La aplicación hace uso de diversas clases: algunas vienen contenidas en el SDK de Android, otras han sido desarrolladas para definir objetos y otras definen el comportamiento de actividades. Las que definen objetos son clases auxiliares, existentes para simplificar el código de las clases que determinan el comportamiento de las actividades.

9.9.1 Clases incluidas en el SDK de Android

Las clases siguientes son comunes a todas las aplicaciones de Android. Ya que no han sido desarrolladas específicamente para este proyecto, se dispone a exponer su funcionalidad sin entrar en gran detalle sobre sus miembros y variables.

- **Activity:** la clase que rige las actividades. A pesar de que normalmente se presentan al usuario a través de una GUI (interfaz gráfica de usuario, por sus siglas en inglés) como ventanas en pantalla completa también se pueden encontrar actividades en forma de ventanas flotantes o incrustadas en otras actividades.

- **AlertDialog:** subclase de Dialog. Muestra un *pop-up* que puede presentar uno, dos o tres botones configurables.

- **Notification:** clase que representa como una notificación persistente se presentará al usuario.

- **NotificationManager:** clase gestora de notificaciones. Existe para notificar al usuario de las notificaciones existentes. La gestión de las notificaciones se realiza mediante la id de cada notificación. De esta clase no se hace instancia directamente, sino que se toma de un servicio de sistema.

- **PendingIntent:** descripción de un Intent y de una acción asociada a él. Al pasar un PendingIntent a una aplicación se le da acceso a la acción que con él va. Si la aplicación que creó el PendingIntent se cierra, este seguirá siendo utilizable por aquellos procesos a los que se le haya pasado dicho PendingIntent.

- **Service:** servicio, es un componente de la aplicación capaz de realizar tareas o dar funcionalidades a otras aplicaciones sin interactuar con el usuario. Estos servicios se ejecutan en el hilo de ejecución de la aplicación que lo creó, por lo que, si va a realizar tareas con gran coste computacional o con llamadas bloqueantes, los servicios habrán de ser declarados en un hilo de ejecución aparte.

- **BluetoothAdapter:** representa el adaptador de Bluetooth local. Permite realizar tareas fundamentales relativas al Bluetooth

- **BluetoothDevice:** representa un dispositivo Bluetooth remoto. Las instancias de clase no se hacen directamente, si no que se reciben a través del adaptador Bluetooth.

- **BluetoothSocket:** representa un socket de conexión Bluetooth conectado o en proceso de conexión. Su comportamiento es similar a los sockets de conexión TCP.

- **BroadcastReceiver:** clase que implementa un sistema para recibir Intents tanto de otras aplicaciones como internos a la propia aplicación.

- **Context:** clase que refleja el contexto del estado actual de la aplicación. Típicamente se llama al contexto para obtener información de otra parte de la aplicación.

- **DialogInterface:** clase que contiene los elementos necesarios para trabajar con la clase AlertDialog. Estos elementos son listeners e ids de los botones, y métodos para gestionar el diálogo de alerta.

- **Intent:** esta clase se explica en profundidad en el apartado 8.3

- **IntentFilter:** clase que tiene la información de los Intents que se escucharán en la aplicación. Pueden ser modificados dinámicamente mediante Java o quedar completamente determinados antes de la compilación mediante el archivo AndroidManifest.xml.

- **SharedPreferences:** clase que se ocupa del almacenamiento de preferencias a nivel de aplicación. Este almacenamiento se realiza a través de variables, de forma similar a como se almacenan en los Bundles, que se expondrán más adelante.

- **Address:** clase que representa una dirección física: un conjunto de cadenas de texto que describen una localización.

- **Geocoder:** clase que se ocupa del *geocoding* y el *geocoding* inverso. El *geocoding* es el proceso de convertir una dirección física en coordenadas. El *geocoding* inverso es la conversión de unas coordenadas a una dirección parcial.

- **RingtoneManager:** clase que da acceso a los tonos de llamada, tonos de notificación y otros sonidos.

- **Bundle:** clase que puede almacenar datos sobre el estado de ejecución de una aplicación en un momento dado. Los datos que se guardan en el Bundle no se seleccionan automáticamente: es el programador el que decide qué datos se guardan. Es en este método en el que hay que guardar el estado de la aplicación, identificando cada aspecto con una variable. La forma de guardar datos en un Bundle es mediante los métodos **void putInt**(**String** key, **int** value), **void putDouble**(**String** key, **double** value), **void putString**(**String** key, **String** value)…etc. El primer argumento, key, es el nombre que se le da al atributo que se guarda, para luego poder recuperarlo. La recuperación de los datos se hace mediante los métodos de la clase Bundle **int getInt**(**String** key), **double getDouble**(**String** key), **String getString**(**String** key)… etc. Aquí se pone de manifiesto el papel del argumento *key* del método *onRestoreInstanceState*: distinguir unos datos del Bundle de otros. En caso de llamar a cualquier método del tipo *get* y recibir un tipo de datos que no concuerde con el tipo de datos que se esperaba, la aplicación se detiene.

- **CountDownTimer:** clase que permite la implementación de un timer de cuenta atrás, produce eventos en cada *tick* y al finalizar. Tanto el tiempo total del timer como el intervalo entre *ticks* son completamente configurables.

- **PreferenceManager:** clase que permite instanciar la clase PreferenceManager de una actividad.

- **SmsManager:** clase que gestiona las operaciones con mensajes SMS.

- **InputType:** clase que contiene constantes que definen el contenido de los objetos editables (como campos de introducción de texto).

- **GregorianCalendar:** clase que proporciona utilidades para la obtención y el formateo como cadena de la hora y la fecha a partir del calendario gregoriano.

- **Menu:** interfaz para el manejo de los objetos de un menú.

- **MenuItem:** clase que hace referencia a un ítem concreto de un menú concreto.

- **View:** representa el bloque básico para la interfaz de usuario de una aplicación. Una View ocupa un espacio rectangular en la pantalla y se ocupa de dibujar en él y de manejar los eventos que ocurren en dicho espacio. Es la clase base de los *widgets*.

- **Button:** clase representa un *widget* de botón. Los botones pueden ser pulsados por el usuario para efectuar una acción.

- **CompoundButton:** botón con dos estados: apagado y encendido. Cuando el botón se pulsa el estado se cambia automáticamente. A pesar de que en la aplicación no se ha utilizado ningún botón de este tipo, el *listener* del *switch* se encuentra en esta clase.

- **EditText:** representa un campo de texto editable.
- **Switch:** al igual que el *CompoundButton*, el *Switch* tiene dos estados. La diferencia entre estos dos *widgets* es que, para cambiar el estado, el usuario puede o bien arrastrar el *switch* hacia alguno de los dos lados, o pulsar sobre él.
- **TextView:** clase que representa un campo de texto estático.
- **Toast:** clase mediante la cual se pueden crear y mostrar mensajes tipo Toast. Estos mensajes aparecen en la parte inferior de la pantalla y únicamente pueden mostrar texto. Su utilidad se limita a mostrar información y no son útiles si se requiere alguna acción por parte del usuario.
- **PreferenceFragment:** clase que permite mostrar un conjunto de objetos Preference como una lista. Estos objetos Preference tienen asignados una entrada en la clase SharedPreferences, de manera que se puede interactuar con las preferencias almacenadas directamente a través de un elemento Preference. La clase Preference no se ha utilizado en el código, ya que, en vez de haberse programado esta lista en java, se ha hecho en XML. Al hacerse así, todas las preferencias se guardan como cadena, aunque se haya introducido un valor numérico, por lo que antes de manejarlas directamente hay que hacer conversiones. En caso de no hacerse se provoca el cierre de la aplicación.
- **Location:** clase que contiene datos de una localización física. Además de las coordenadas también incluye datos como a la hora que se tomó, orientación, altitud y velocidad.
- **LocationListener:** listener que se ocupa de detectar cambios en la localización física.
- **LocationManager**: esta clase proporciona acceso a los sistemas de localización del sistema. Permite a las aplicaciones obtener datos periódicos sobre la posición del dispositivo.

9.9.2 Clases auxiliares

9.9.2.1 GPSManager

Esta clase gestiona todo el apartado de geolocalización y lo hace de una manera similar a como la clase BluetoothManager gestiona la conectividad Bluetooth. En este caso no es necesario incluirla en Holder, ya que GPSManager, además de a un servicio de sistema, no se conecta a un dispositivo físico, por lo que no importa que se instancie de nuevo cada vez que la aplicación vuelva a primer plano. Esta clase hereda de Service e implementa LocationListener, es decir, que adquiere las características de ésta última, una clase empleada para recibir notificaciones cuando la localización cambia.

- Variables:
    - **isGPSEnabled:** variable booleana privada que indica el estado de la localización GPS en el dispositivo Android.

- **isNetworkEnabled:** variable booleana privada que indica el estado de la localización por red en el dispositivo Android
- **mLocation:** instancia privada de la clase Location que contiene todos los datos sobre la localización del dispositivo Android.
- **latitude:** variable privada de doble precisión que contiene la latitud en la que se encuentra el dispositivo
- **longitude:** variable privada de doble precisión que contiene la longitud en la que se encuentra el dispositivo
- **mLocationManager:** instancia privada de la clase LocationManager. Esta clase se ocupa de gestionar el sistema de localización: conexión al servicio, obtención del estado de éste, toma de medidas de posición…
- **canGetLocation:** variable booleana pública que indica si es posible obtener la localización.
- **mContext:** instancia privada de la clase Context. Es necesario pasársela a la clase GPSManager para que ésta pueda conectarse con el servicio de localización del sistema.

- Métodos:
  - **public Location** getLocation(): primero comprueba que el GPS está activado, en caso afirmativo y a través del gestor de localización, se conecta con el GPS del dispositivo y pide la última localización almacenada, almacenando la latitud y la longitud en las variables correspondientes.
  - **public double** getLatitude(): devuelve el valor de latitud de la última posición almacenada en el dispositivo.
  - **public double** getLongitude: devuelve el valor de la longitud de la última posición almacenada en el dispositivo.

- Constructor
  - **public** GPSManager(**Context** context): en el constructor se asigna el contexto desde el que se le llama a mContext. Adicionalmente se llama al método getLocation().

9.9.2.2 Protocol

En esta clase está almacenado todo el protocolo ya explicado anteriormente de comunicación entre los dos dispositivos del sistema. Se decidió almacenar el protocolo en una clase aparte para, en caso de tener que modificar dicho protocolo, no tener que buscar todas las referencias al protocolo en el código y sólo cambiarlo

en la clase. Ésta clase se compone únicamente de variables públicas estáticas, para que se pueda acceder desde cualquier clase de la aplicación.

9.9.3 Clases de actividades

9.9.3.1 Introducción

Cuando se quiere que una clase rija el comportamiento de una actividad en Android, es necesario que esta herede de la clase Activity. Esta clase tiene una gran variedad de métodos que permiten controlar un amplio abanico de aspectos de la actividad a programar. Cada uno de estos métodos se llama automáticamente al ocurrir un suceso determinado. Debido a la vasta extensión de la lista de estos métodos y su gran diversidad de usos, sólo se introducirán los que se han utilizado.

9.9.3.1.1 Métodos heredados de la clase Activity ligados al ciclo de vida

Estos métodos están ligados al previamente explicado ciclo de vida de la aplicación. Debido a que cada uno se llama cuando ocurre el suceso del ciclo de vida homónimo, sólo se enumerarán y, se explicarán los argumentos en caso de haberlos.

- **protected void** onCreate(**Bundle** savedInstanceBundle): En caso de que el sistema mate el proceso de la aplicación podrían recuperarse los datos más relevantes por medio de esta instancia de la clase Bundle.
- **protected void** onResume()
- **protected void** onDestroy()

9.9.3.1.2 Otros métodos heredados de la clase Activity

- **public boolean** onCreateOptionsMenu(**Menu** menu): el sistema llama a este método la primera vez que se muestran los ítems del menú. El parámetro menu es el menú que se crea.
- **public boolean** onMenuItemSelected(**int** featureId, **MenuItem** item): el sistema llama a este método cada vez que se pulsa un ítem de menú. El parámetro *featureId* es el panel en el que se encuentra el menú, e *ítem*, el ítem de menú seleccionado.
- **public boolean** onPrepareOptionsMenu(**Menu** menu): este método es similar a onCreateOptionsMenu, solo que en lugar de llamarse en la creación del menú, se llama cada vez que se muestra por pantalla.
- **protected void** onSaveInstanceState(**Bundle** outState): el sistema llama a este método cada vez que se dispone a guardar el estado de la aplicación en el Bundle.

- **protected void** onRestoreInstanceState(**Bundle** savedInstanceState): el sistema llama a este método cada vez que se dispone a cargar el estado de la aplicación del Bundle correspondiente. En éste método se debe programar la recuperación de los datos y la reestructuración de la aplicación en consonancia con ellos.

- **public void** onRequestPermissionsResult(**int** requestCode,**String** permissions[], **int**[] grantResults): el sistema llama a este método cuando se ha obtenido la respuesta a una petición previa de concesión de permisos.

9.9.3.2 MainActivity (Actividad principal)

- Variables
    - **DATA_RECEIVED:** cadena de texto privada estática y final que especifica la acción del Intent que se dispara cuando se recibe un dato.
    - **DeviceName:** cadena de texto privada, estática y final que especifica el nombre del dispositivo Bluetooth a buscar.
    - **LOCATION_PERMISSION_REQUESTED:** variable entera privada, estática y final que especifica la acción del Intent que se lanza cuando se piden los permisos de localización al usuario.
    - **SMS_SENT, SMS_DELIVERED:** cadenas de texto privadas, estáticas y finales utilizadas en los Intents de los SMS.
    - **AlarmSentMillis:** variable entera, pública, estática y final que contiene el tiempo expresado en milisegundos que habrán de pasar para enviar la alarma.
    - **tel:** cadena de texto estática, privada y final que contiene el número de teléfono al que se enviará la alerta.
    - **WaitMillis:** variable entera, privada, estática y final que contiene el tiempo expresado en milisegundos que ha de transcurrir desde que se establece la conexión con el microcontrolador y se le envía la configuración inicial.
    - **PresetPassword:** cadena de texto privada, estática y final que contiene el valor predeterminado de la contraseña.
    - **mGPSManager:** instancia privada de la clase GPSManager.
    - **mySelf:** instancia privada de Activity. Se inicializa al crearse la actividad para poder referirse a ella misma dentro de las subclases. Esto es debido a que en Java, una clase puede referenciarse a sí misma haciendo uso de this. Si al entrar en una subclase se quisiera hacer referencia a la clase padre no se podría hacer a través de this, ya que se referiría a la subclase.
    - **mySwitch:** instancia privada de la clase Switch. Más tarde esta instancia se referirá al botón tipo Switch de la actividad principal.

- **MainTxt, BpmTxt:** instancias privadas de la clase TextView. Durante la vida de la aplicación éstas variables se referirán al campo de texto estático principal de la actividad principal y al campo de texto estático que mostrará la frecuencia cardiaca, respectivamente.
- **SharedPrefs:** instancia privada de la clase SharedPreferences.
- **RequestThrsMenuItem, ResetMenuItem, ConnectMenuItem, ConfigureMenuItem:** instancias privadas de MenuItem. Se referirán a los objetos de menu de "Pedir umbrales de aviso", "Reiniciar SiAMP", "Conectar SiAMP" y "Configurar SiAMP". Son necesarios para desactivar estas opciones cuando no se puedan usar.
- **lthr, hthr, CountDownTimerSecs:** variables privadas y enteras. Respectivamente, almacenan el umbral inferior, el superior y el tiempo expresado en segundos que le queda al temporizador de envío de la alarma para ser enviado.
- **noti_id, noti_id2, noti_id3:** variables enteras, privadas y finales, que expresan la id de las notificaciones
- **InitializationStatus, AlertStatus, ConnectionStatus:** variables booleanas privadas. Respectivamente dan información sobre el estado de la inicialización, la alerta y la conexión del microcontrolador.
- **WaitForFirstResponse, WaitForSecondResponse:** variables booleanas privadas. Se ponen a 1 cuando se está esperando a la respuesta del primer y segundo umbral en el momento de la inicialización, respectivamente.
- **BtTurningOn:** variable booleana privada. Se pone a 1 brevemente durante el periodo de encendido del Bluetooth. Al incluir esta variable se evita preguntar dos veces por el encendido del Bluetooth.
- **builder:** instancia privada de la subclase Builder de la clase AlertDialog. Esta subclase facilita sustancialmente la creación de *pop-ups* de alerta.
- **alert:** instancia privada de AlertDialog. Usada para configurar los *pop-ups* a través de builder.
- **mReceiver:** instancia privada de BroadcastReceiver.
- **mmSocket:** instancia privada de BluetoothSocket.
- **mmDevice:** instancia privada de la clase BluetoothDevice. Es instanciada al conectarse al dispositivo.
- **mmInputStream:** instancia privada y estática de la clase InputStream. Es el stream de datos de entrada proporcionado por el socket de conexión.
- **mmOutputStream:** instancia privada y estática de la clase OutputStream. Análogamente, este es el stream de salida de la conexión.

- **workerThread:** instancia privada de Thread. Es el que se ocupa de leer del stream de datos de entrada.

- Métodos

  - **onCreate:** en el método onCreate de MainActivity se han programado todas las inicializaciones pertinentes: se asigna el valor de mySelf a la actividad, se instancia GPSManager, se asignan a cada una de las instancias de *widgets* el *widget* correspondiente, se crean los filtros de Intents y se registran al BroadcastReceiver. Se programa el comportamiento del switch a través de su listener. Adicionalmente, se cargan los valores almacenados de los umbrales en caso de haberlos y se prepara el *pop-up* de alerta. Si no hubiera umbrales almacenados, se dejan por defecto como umbral superior 180 y como umbral inferior 50.

  - **onCreateOptionsMenu:** en este método se asignan las *ids* de los ítems de menú que alterarán su estado durante la ejecución del programa a las variables pertinentes.

  - **onMenuItemSelected:** mediante una sentencia *switch* se programa el comportamiento de cada ítem de menú: pedir umbrales enviará la petición al microcontrolador en caso de que la conexión se haya establecido, la petición de conexión iniciará la conexión Bluetooth en caso de no estar conectada, configurar mostrará el diálogo para introducir la contraseña y reiniciar enviará al dispositivo un paquete tipo ALERTAOK.

  - **onPrepareOptionsMenu:** se configuran como habilitados o deshabilitados los ítems de menú arriba mencionados: la petición de umbrales está habilitada cuando la conexión está establecida, la petición de conexión cuando no lo está y la petición de reinicio cuando éste está en modo alerta.

  - **private void** SendSiAMP(**int** dato1, **int** dato2): crea un vector local de dos posiciones tipo byte y lo envía a través del stream de salida del socket de la conexión Bluetooth. El uso esperado de este método es que el primer dato sea la cabecera de datos y el segundo el dato en sí.

  - **private void** sendAlert(): este método se ocupa del proceso de envío de la alerta una vez dada por válida. Este método obtiene la latitud y la longitud de la última posición del GPS, una vez hecho esto, a través de la clase Geocoder se obtiene una localización física partiendo de las coordenadas. Se crea la notificación indicando que se ha detectado una alerta y se está avisando a las autoridades. Finalmente, se inicia el proceso de envío de SMS llamando a la función miembro sendSMS. En caso de no poder obtenerse, se muestra un mensaje de error en lugar de la localización.

  - **private void** sendSms(**String** num, **String** msg): este método se hace cargo del proceso completo de envío de SMS, tanto del envío en sí como de escuchar los Intents del sistema para obtener el estado del SMS (enviado, no enviado, leído). En todo momento se tiene al usuario informado del estado de estas operaciones ya sea mediante notificaciones descartables o mediante mensajes tipo *Toast*.

- **private void** alertPopUp(): función miembro que originalmente, cuando no se hacía uso de la clase Holder, se ocupaba del proceso de instanciar y programar el comportamiento del diálogo de alerta, el temporizador de cuenta atrás y las notificaciones. Una vez que se incluyó la clase Holder la instancia del cuadro de diálogo se hizo global para poder ser llamada en cualquier miembro de la clase debido a que, si se destruye la aplicación hay que volver a crear dicho cuadro de diálogo. Además, la gestión del temporizador de cuenta atrás es parte del trabajo de la clase Holder, por lo que lo único que hace esta función en la versión final de la aplicación es enviar la notificación y mostrar el cuadro de alerta.

- **onResume**: en este método se leen los valores de los umbrales almacenados en la clase SharedPreferences y se comparan con los almacenados en las variables globales. En caso de ser diferentes, se guardan los nuevos en las variables globales y se envían al microcontrolador. Esto ocurrirá cuando este método dispare después de dejar la actividad de configuración para volver a la actividad principal.

- **onSaveInstanceState**: se guardan en el Bundle el estado de inicialización del Arduino, el estado de la alerta, el contenido del campo de texto estático, el estado del switch y si éste está habilitado o no.

- **onRestoreInstanceState**: se recuperan las variables que se almacenaron en la función miembro **onSaveInstanceState**. Si la variable recuperada que representa el estado de la alerta es verdadera se muestra por pantalla el cuadro de diálogo.

- **onRequestPermissionResult:** si no se tenía el permiso de acceso a la localización anteriormente, se llama al método **getLocation** de GPSManager.

9.9.3.3 Settings (Actividad de configuración)

Esta clase está prácticamente inédita gracias a que el comportamiento de las actividades de configuración viene programado por defecto en el SDK y los ítems del menú de configuración están ligados a una entrada de la clase **SharedPreferences**. Lo único que se ha incluido respecto a la clase base de una actividad es que se ha creado una subclase que hereda de **PreferenceFragment** y en su método **onCreate** se llama a la función **addPreferencesFromResource**(**void** preferencesResId) que, pasándole la id del fichero XML que contiene los elementos del menú de preferencias, crea una pantalla automáticamente.

9.9.3.4 PasswordChange (Actividad de cambio de contraseña)

- Variables:
  - **button, backButton**: instancias privadas de la clase Button que se referirán a los botones "Confirmar" y "Atrás", respectivamente.

  - **editText1, editText2, editText3**: instancias privadas de la clase EditText, que se referirán a los campos de texto editables "Antigua contraseña", "Nueva contraseña" y "Confirmar contraseña"

  - **SharedPrefs**: instancia privada de la clase SharedPreferences.

- **mySelf**: instancia privada de Activity. Igual que en el caso de MainActivity, esta instancia se inicializa para que haga referencia a la propia clase.
- **PresetPassword**: variable tipo cadena de texto, privada, estática y final que indica la contraseña inicial. Su valor es "aaaa".

- Métodos:
  - **onCreate:** se empieza por dar el valor de la instancia de la clase PasswordChange a mySelf, así como obtener el valor de SharedPreferences y asignar cada una de las variables de los *widgets* a su *widget* correspondiente. Por último, se instancia la subclase Button.onClickListener dos veces, una para cada botón.

10 CONCLUSIONES Y POSIBLES MEJORAS DEL PROYECTO

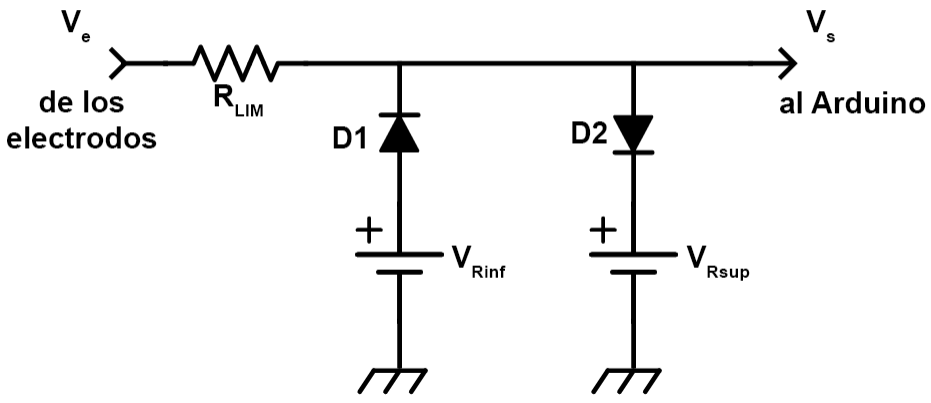
A continuación, se expone una lista de posibles mejoras del proyecto, en las que no se ha incurrido debido principalmente a la falta de recursos.

10.1 Implantación de un sistema de detección de los distintos episodios cardiacos a través del ECG

En el momento de finalización del proyecto, el sistema únicamente usa el electrocardiograma para tomar el pulso. Se plantea como posible ampliación la programación del microcontrolador para que analice el electrocardiograma y sea capaz de detectar e incluso predecir posibles episodios cardiacos. Esto no se ha implantado en el proyecto debido a que el microcontrolador no tenía más potencia de cálculo, y la obtención y filtrado de la señal cada 5 milésimas requiere prácticamente la totalidad de los recursos del mismo.

10.2 Adición de un sistema de protección eléctrica para el microcontrolador

Debido a que los pacientes que tengan implantado un ICD, en caso de sufrir una fibrilación o cualquier otro episodio cardiaco, recibirán una descarga, ésta se transmitirá a través de los electrodos al microcontrolador. Esto lógicamente puede dañar el circuito, por lo que se hace necesaria una protección eléctrica. Si el voltaje de la descarga sobre los electrodos no es demasiado elevado, se propone a tal fin un recortador de tensión situado entre los electrodos y el microcontrolador.



Figur: Circuito de limitación

El funcionamiento del circuito es simple y es el que sigue:

- Si $V_{Rsup} > V_e - V_{R_{LIM}} > V_{Rinf}$: las tensiones en los ánodos de D1 y D2 son menores que las de sus cátodos, luego ambos están cortados, consecuentemente, la tensión de salida es igual a la de entrada menos la tensión que cae en la resistencia de limitación, $V_s = V_e - V_{R_{LIM}}$. El circuito equivalente es el siguiente:

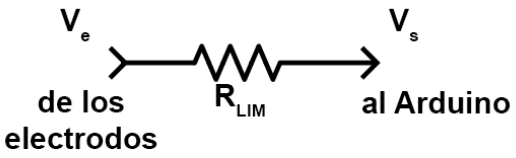


Figura 37: Circuito equivalente si la tensión está dentro de los rangos admitidos

- Si $V_e - V_{R_{LIM}} > V_{Rsup}$: La tensión en el ánodo de D2 es mayor a la del cátodo, por lo que entra en conducción, mientras que D1 sigue cortado. El circuito equivalente es el que sigue:

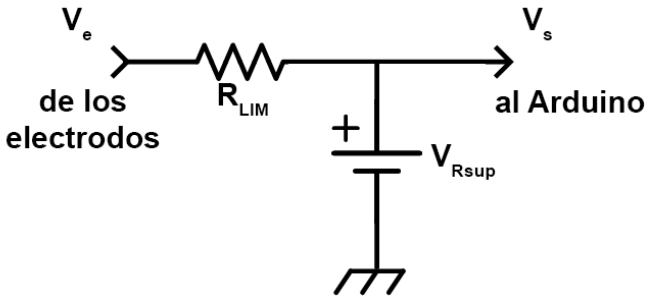


Figura 38: Circuito equivalente si la tensión es mayor que el máximo admitido

Como se puede apreciar, la tensión de salida es V_{Rsup}, y el exceso de tensión lo absorbe la resistencia de limitación

- Si $V_e - V_{R_{LIM}} < V_{Rinf}$: La tensión en el ánodo de D1 es mayor a la del cátodo, por lo que entra en conducción, mientras que D2 está cortado. El circuito equivalente es el que sigue:

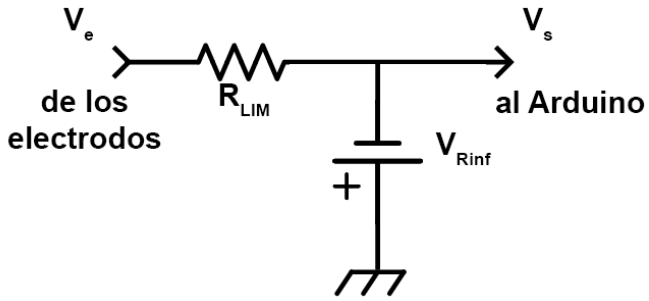


Figura 39: Circuito equivalente si la tensión es menor que el mínimo admitido

Se puede comprobar que la tensión de salida es V_{Rinf}, y al igual que en el caso anterior, el exceso de tensión es absorbido por la resistencia de limitación.

En el circuito mostrado arriba, la tensión de recorte superior V_{Rsup} y la tensión de recorte inferior V_{Rinf} deben ser las máximas y las mínimas que admite el microcontrolador como entrada analógica, en el caso del Arduino Uno, 5 y 0V, respectivamente. El valor de la resistencia de limitación, R_{LIM} ha de ser suficientemente grande como para que no disipe potencia en exceso (no se caliente demasiado) y suficientemente pequeño como para que no distorsione mucho la lectura de los electrodos. Para decidir el valor de la resistencia de limitación, se tendría que realizar un análisis de la entrada del microcontrolador cuando se implementa el circuito limitador y la potencia disipada en una resistencia:

$$P = \frac{V^2}{R}$$

10.3 Compactación del microcontrolador

Lógicamente, en el estado actual de prototipado, el sistema no está listo para el uso diario. Por lo tanto, sería necesario diseñar una placa que incluyera los elementos del sistema actual, compactándolos en un tamaño más pequeño, posiblemente en un dispositivo que se pudiera colocar en una cinta ajustada al pecho.

10.4 Conclusiones finales

Este proyecto puede sentar las bases para la monitorización continua mediante un método no invasivo. Actualmente tiene una serie de limitaciones, expuestas en este capítulo (funcionalidades limitadas, falta de protección eléctrica, estado de prototipo). Sin embargo, con la tecnología actual, estas limitaciones podrían ser eliminadas, obteniendo un dispositivo completamente funcional y de gran utilidad para pacientes enfermos del corazón.

De continuarse el desarrollo del proyecto, el camino a tomar probablemente debería ser la creación de un dispositivo que se pudiera conectar directamente al ICD, para evitar posibles inconveniencias con el sistema de toma de datos. De esta forma, podrían tenerse todas las funcionalidades en potencia de este proyecto, eliminando algunas de las inconveniencias intrínsecas. Asimismo, habría que desarrollar un protocolo de comunicación entre el ICD y el dispositivo, así como generar un método de conexión entre éste y el smartphone que no ponga en riesgo ni el funcionamiento del ICD, ni la salud del paciente. Al estar implantado en el cuerpo del paciente, el dispositivo tendría que cumplir las regulaciones que haya para elementos de esta índole, como pudieran ser la no toxicidad de los materiales o el completo aislamiento eléctrico.

11 ANEXOS

11.1 Códigos

11.1.1 Arduino

```
#include <compat/deprecated.h>
#include <FlexiTimer2.h>

#define AVISO 0
#define CONTINUO 1
#define ALERTA 2
#define INICIALIZACION 3
#define LEDPIN 7
#define BTNPIN 4
#define SAMPFREQ 200
#define TIMERVAL (1000/(SAMPFREQ))
#define BPM '#'
#define ALERT '!'
#define HTHR '^'
#define LTHR 'v'
#define RESPONSE '~'
#define MODOCONT 'A'
#define MODOAVISO 'B'
#define CAMBIOHTHR 'C'
#define CAMBIOLTHR 'D'
#define PEDIRTHR 'E'
#define ALERTOK 'F'

//Declaración de variables globales

int ch = 0, t = 0, led_st = LOW, sum_bpm, lthr=0, hthr=0;
double yk1 = 0, yk2 = 0, yk = 0, win[10], sum, xk1 = 0, xk2 = 0, xk = 0, yabs;
char flg = 1, cont = 0, modo = INICIALIZACION, k, bpm, bpm_avg, dato[2], bpm_win[15], i = 0, primero = 1, j;
long blink1 = 0, blink2 = 0, del_alerta = 100, del_init = 500, intervalo;
```

```
 //Prototipos de funciones

 void send_bpm(int);
 void send_alert(void);
 void send_thr(int, int);
 void send_response(void);
 void timer_fcn(void);

 //Funciones principales

 void setup() {
   FlexiTimer2::set(TIMERVAL,    timer_fcn);    //Configuración    del
temporizador
   FlexiTimer2::start();
   Serial.begin(38400);
   pinMode(LEDPIN, OUTPUT); //Configuración del pin del LED como pin
de salida
   pinMode(BTNPIN, INPUT); //Configuración del pin del botón como pin
de entrada
   for (j = 0; j < 15; j++) bpm_win[j] = 0; //Inicialización de los
vectores
   for (j = 0; j < 10; j++) win[j] = 0;
 }

 void loop() {
     if (Serial.available()) { //Si existen datos para leer por el
puerto serie
     Serial.readBytes(dato, 2); //Almacenar en el vector dato
     switch (dato[0]) {
       case MODOCONT: //Cambio a modo continuo
         if (modo != ALERTA && modo != INICIALIZACION) { //Si no está
en modo alerta o de inicialización
           digitalWrite(LEDPIN, HIGH); //Encencer el LED
           modo = CONTINUO;
         }
         break;
```

```
        case MODOAVISO: //Cambio a modo de aviso en alerta
          if (modo != ALERTA && modo != INICIALIZACION) {
            digitalWrite(LEDPIN, LOW); //Apagar el LED
            modo = AVISO;
          }
          break;
        case CAMBIOHTHR: //Cambio de umbral superior
          if (modo != ALERTA) {
            hthr = dato[1]&0xFF; //Tomar el segundo byte del mensaje y almacenarlo en hthr (umbral alto)
            send_response(); //Notificar el éxito
            if (modo == INICIALIZACION && lthr > 0) { //Si estuviera en inicialización y el otro umbral estuviera ya configurado, pasar al modo de aviso
              modo = AVISO;
            }
          }
          break;
        case CAMBIOLTHR: //Cambio de umbral inferior
          if (modo != ALERTA) {
            lthr = dato[1]&0xFF; //Tomar el segundo byte del mensaje y almacenarlo en lthr (umbral bajo)
            send_response(); //Notificar el éxito
            if (modo == INICIALIZACION && hthr > 0) { //Si estuviera en inicialización y el otro umbral estuviera ya configurado, pasar al modo de aviso
              modo = AVISO;
            }
          }
          break;
        case PEDIRTHR: //Petición de ambos umbrales
          if (modo!=ALERTA)send_thr(hthr,lthr); //Si no está en modo de alerta, enviar los umbrales
          break;
        case ALERTOK:
          modo = AVISO; //Volver a modo de aviso
          break;
      }
    }
```

```
    if (digitalRead(BTNPIN) == HIGH) { //Comprobación de que se pulsa el botón
      while (digitalRead(BTNPIN) == HIGH); //Comprobación de que se suelta el botón
      send_alert();
    }
  }

  //Declaración de las funciones

  void timer_fcn() {
    if (modo != ALERTA && modo != INICIALIZACION) { //Si está en uno de los modos de funcionamiento normal del sistema
      xk = analogRead(ch); //Leer la señal
      //yk=.1242*yk1-0.0035*yk2+0.1189*(xk1 - xk2); //Aplicar el filtro paso banda
      yk = 1.277 * yk1 - 0.466 * yk2 + 0.3189 * (xk - xk1);
      yabs = abs(yk); //Valor absoluto de la señal
      sum = 0;
      for (k = 1; k < 10; k++) { //Sumar los últimos 10 valores
        win[k - 1] = win[k];
        sum += win[k - 1];
      }
      win[9] = yabs;
      sum += yabs;
      sum = sum / 10; //Realizar la media móvil
      if (sum > 100 && flg == 1) { //Si se detecta un pulso
        cont = 0;
        flg = 0; //No permitir la detección de un pulso hasta que se flg vuelva a estar a 1
        bpm = (char)((SAMPFREQ * 60) / t); //Cálculo de la frecuencia cardiaca
        bpm_win[i] = bpm; //Almacenar en el vector para realizar la media móvil de las frecuencias cardíacas
        i++;
        if (i == 15) { //Si se han contado 15 pulsos
          i = 0; //Reiniciar el contador
```

```
        if (primero == 1)primero = 0; //Poner primero a 0 para
indicar que el vector está lleno
       }
       if (primero == 0) {
         sum_bpm = 0;
         for (j = 0; j < 15; j++) sum_bpm += bpm_win[j]; //Sumar los
elementos del vector de frecuencias cardiacas
         bpm_avg = sum_bpm / 15; //Calcular la frecuencia media
         if (modo == CONTINUO)send_bpm(bpm_avg); //Mandar la media
       }
       t = 0;
     }
     if (cont == 20) {          //Contar 20 tiempos de muestreo hasta
permitir el siguiente pulso
       cont = 0;
       flg = 1;
     }
     else if (cont < 20 && flg == 0) { //Si se permite la detección de
otro pulso y el contador es menor a 20
       cont++;
     }
     yk2 = yk1; //Actualización de las variables que intervienen en el
filtrado
     yk1 = yk;
     xk2 = xk1;
     xk1 = xk;
     t++;
     if ((bpm_avg < lthr || bpm_avg > hthr) && modo != ALERTA && modo
!= INICIALIZACION && primero == 0) { //Si la frecuencia sobrepasa los
límites inferiores o superiores y no se está en ningún modo de
funcionamiento anómalo
       send_alert();
     }
   }
   //Parpadeo del LED

   blink2 = millis(); //Tomar el valor de los microsegundos que han
pasado desde que se inició el microcontrolador
```

```
    if (modo == ALERTA) { //Definición del intervalo de parpadeo en
función del estado
      intervalo = del_alerta;
    }
    else if (modo == INICIALIZACION) {
      intervalo = del_init;
    }
    else {
      intervalo = -1;
    }
    if (modo == ALERTA || modo == INICIALIZACION) { //Si se está en un
modo en el que el LED esté parpadeando
      if (blink2 - blink1 > intervalo) { //Si ha transcurrido el
intervalo
        blink1 = blink2;
        if (led_st == LOW) { //Cambiar el estado del LED
          led_st = HIGH;
        }
        else {
          led_st = LOW;
        }
        digitalWrite(LEDPIN, led_st);
      }
    }
  }


  void send_bpm(int bpm) {
    String msg = String(BPM);
    msg+=String(bpm,DEC);
    msg+='\n';
    Serial.print(msg);
  }

  void send_alert() {
    modo = ALERTA; //Paso a modo de alerta
    String msg=String(ALERT);
```

```
  msg+='\n';
  Serial.print(msg);
}

void send_thr(int _hthr, int _lthr) {
  String msg= String(HTHR);
  msg+=_hthr;
  msg+=LTHR;
  msg+=_lthr;
  msg+='\n';
  Serial.print(msg);
}


void send_response() {
  String msg = String(RESPONSE);
  msg+='\n';
  Serial.print(msg);
}
```

11.1.2 Android

11.1.2.1 Java

11.1.2.1.1 GPSManager.java

```
package com.andcotjim.siampv3;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;

public class GPSManager extends Service implements LocationListener {

    private boolean isGPSEnabled = false;
```

```
    private boolean isNetworkEnabled = false;
    private Location mLocation;
    private double latitude;
    private double longitude;
    private LocationManager mLocationManager;
    boolean canGetLocation = false;
    private Context mContext;

    public GPSManager(Context context) {
        mContext = context;
        getLocation();
    }

    public Location getLocation() {
        try {
            mLocationManager = (LocationManager)
mContext.getSystemService(LOCATION_SERVICE);
            isGPSEnabled =
mLocationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
            isNetworkEnabled =
mLocationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
            if (!isGPSEnabled && !isNetworkEnabled) {
                // No hacer nada
            } else {
                // Tomar la posición del proveedor de red
                this.canGetLocation = true;
                if (isNetworkEnabled) {
                    if (mLocationManager != null) {
                        try {
                            mLocation =
mLocationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
                            if (mLocation != null) {
                                latitude = mLocation.getLatitude();
//Extraer la latitud y la longitud
                                longitude = mLocation.getLongitude();
                            }
                        } catch (SecurityException e){
                            e.printStackTrace();
                        }
                    }
                }
```

```
                // Tomar la posicion del GPS
                if (isGPSEnabled) {
                    if (mLocation == null) {
                        if (mLocationManager != null) {
                            try {
                                mLocation = mLocationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
                                if (mLocation != null) {
                                    latitude = mLocation.getLatitude(); //Extraer la latitud y la longitud
                                    longitude = mLocation.getLongitude();
                                }
                            } catch (SecurityException e){
                                e.printStackTrace();
                            }
                        }
                    }
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

        return mLocation;
    }

    public double getLatitude() {
        if (mLocation != null) { //Si la posición no es nula, devolver la latitud
            latitude = mLocation.getLatitude();
        }
        return latitude;
    }

    public double getLongitude(){
        if (mLocation != null){//Si la posición no es nula, devolver la longitud
            longitude = mLocation.getLongitude();
        }
        return longitude;
    }
```

```
    @Override
    public void onLocationChanged(Location location) {
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras)
{
    }

    @Override
    public void onProviderEnabled(String provider) {
    }

    @Override
    public void onProviderDisabled(String provider) {
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

}
```

11.1.2.1.2 MainActivity.java

```
package com.andcotjim.siampv3;

import android.Manifest;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.BroadcastReceiver;
import android.content.Context;
```

```
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.location.Address;
import android.location.Geocoder;
import android.media.RingtoneManager;
import android.os.Bundle;
import android.os.CountDownTimer;
import android.os.Handler;
import android.preference.PreferenceManager;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.telephony.SmsManager;
import android.text.InputType;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.Switch;
import android.widget.TextView;
import android.widget.Toast;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.GregorianCalendar;
import java.util.List;
import java.util.Set;
import java.util.UUID;

public class MainActivity extends AppCompatActivity {
    //Variable estáticas
    public static final String DATA_RECEIVED = "DATA_RECEIVED";
    private static final String DeviceName = "HC-05",
PresetPassword="aaaa";
```

```
    private static final int noti_id = 30891, noti_id2 = 999;
    private static final int WaitMillis = 3000; //Tiempo de espera en
milisegundos entre que se conecta con SiAMP y se le manda la
configuración inicial
    private static final int LOCATION_PERMISSION_REQUESTED=630;
    public static final int AlarmSentMillis = 15000; //Tiempo en
milisegundos del timer para tomar la alarma como falsa
    private static final String SMS_SENT = "SMS_SENT", SMS_DELIVERED
= "SMS_DELIVERED";
    private static final String tel =""; //Número al que enviar el
SMS

    //Instancias de clases
    private                                                    MenuItem
RequestThrsMenuItem,ResetMenuItem,ConnectMenuItem, ConfigureMenuItem;
    private TextView BpmTxt, MainTxt;
    private Switch mySwitch;
    private SharedPreferences SharedPrefs;
    private Activity mySelf;
    BluetoothAdapter mBluetoothAdapter;
    BluetoothSocket mmSocket;
    BluetoothDevice mmDevice;
    OutputStream mmOutputStream;
    InputStream mmInputStream;
    Thread workerThread;
    private AlertDialog.Builder builder;
    private AlertDialog alert;
    private GPSManager mGPSManager;
    private CountDownTimer timer;

    //Otras variables
    byte[] readBuffer;
    int readBufferPosition;
    private int lthr, hthr;
    volatile boolean stopWorker;
```

```
    private boolean ConnectionStatus = false, AlertStatus=false,
InitializationStatus=false,                WaitForFirstResponse=false,
WaitForSecondResponse=false, BtTurningOn=false;

    //Receptor de Intents
    private BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if   (action.equals(BluetoothDevice.ACTION_ACL_CONNECTED)
&& mmDevice.getName().equals(DeviceName)) {
                //Dispositivo bluetooth conectado y el nombre
coincide con el esperado
                NotificationManager      notificationManager      =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE);
                notificationManager.cancel(noti_id);
                ConnectionStatus = true;
                InitializationStatus = false;
                MainTxt.setText(R.string.idle);
                Toast.makeText(mySelf,           R.string.connected,
Toast.LENGTH_SHORT).show();
            }else                                                  if
(action.equals(BluetoothDevice.ACTION_ACL_DISCONNECTED)) {
                //Dispositivo bluetooth desconectado
                Notification n = new Notification.Builder(mySelf)

.setContentTitle(getResources().getString(R.string.siamp_not_connected
))

.setContentText(getResources().getString(R.string.please_connect))

.setTicker(getResources().getString(R.string.siamp_not_connected))
                        .setSmallIcon(R.mipmap.ic_notification)

.setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICAT
ION))
                        .setOngoing(true)
                        .setPriority(Notification.PRIORITY_MAX)
```

```
.setContentIntent(PendingIntent.getActivity(mySelf,       0,       new
Intent(mySelf, MainActivity.class), 0))
                        .build();
                NotificationManager       notificationManager       =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE);
                notificationManager.notify(noti_id, n);
                ConnectionStatus = false;
                InitializationStatus = false;
                MainTxt.setText(R.string.siamp_not_connected);
                mySwitch.setEnabled(false);
            }else
if(action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)){
                //Estado del adaptador Bluetooth cambiado
                BtTurningOn=true;
                try {
                    FindSiAMP();
                    SetupBTConnection();
                }catch (IOException e){}
            }else if(action.equals(DATA_RECEIVED)) {
                //Llegada de datos
                String data = intent.getStringExtra("data");
                String  header  =  data.substring(0,1);  //Se  toma  el
primer caracter del mensaje
                if (header.equals(Protocol.RESPONSE)){
                    //Si es una respuesta
                    if (WaitForFirstResponse) {
                        WaitForFirstResponse = false;
                        WaitForSecondResponse = true;
                        SendSiAMP(Protocol.CAMBIOLTHR, lthr & 0xFF);
                    }else if(WaitForSecondResponse){
                        WaitForSecondResponse=false;
                        InitializationStatus=true;
                        mySwitch.setEnabled(true);
                        Toast.makeText(mySelf,
R.string.initializing_success, Toast.LENGTH_SHORT).show();
                    }else{
```

```
                        Toast.makeText(mySelf, R.string.thrs_success,
Toast.LENGTH_SHORT).show();
                    }
                }else if (header.equals(Protocol.ALERT)){
                    //Si es una alerta
                    alertPopUp();
                }else if (header.equals(Protocol.BPM)){
                    //Si es una medida de frecuencia cardiaca
                    BpmTxt.setText(data.substring(1));
                }else if (header.equals(Protocol.HTHR)) {
                    //Si son los umbrales
                    String[] _thrs = data.split(Protocol.LTHR);
                    _thrs[0] = _thrs[0].substring(1);
                    AlertDialog.Builder      alert      =      new
AlertDialog.Builder(mySelf); //Instanciar AlertDialog y configurar el
cuadro de alerta
                    alert.setPositiveButton(R.string.ok, null);
                    alert.setTitle(R.string.alert_thrs);

alert.setMessage(getResources().getString(R.string.high)    +"    "    +
String.valueOf(_thrs[0])          +          "          "          +
getResources().getString(R.string.low)       +       "       "       +
String.valueOf(_thrs[1]));
                    alert.show();
                }
            }
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        //Gestor de notificaciones
        final     NotificationManager     notificationManager     =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE);
```

```
        //Gestor de GPS
        mGPSManager = new GPSManager(this);

        mySelf = this;

        //Instanciar los widgets
        mySwitch = (Switch) findViewById(R.id.switch1);
        BpmTxt = (TextView) findViewById(R.id.bpm_txt);
        MainTxt = (TextView) findViewById(R.id.main_txt);

        //Leer de SharedPreferences los valores de los umbrales
        SharedPrefs =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext())
;
        lthr = Integer.parseInt(SharedPrefs.getString("lthr", "50"));
        hthr     =     Integer.parseInt(SharedPrefs.getString("hthr",
"180"));

        //Configurar el filtro de intents
        IntentFilter fil1 = new IntentFilter(DATA_RECEIVED);
        IntentFilter              fil2              =              new
IntentFilter(BluetoothDevice.ACTION_ACL_CONNECTED);
        IntentFilter              fil3              =              new
IntentFilter(BluetoothDevice.ACTION_ACL_DISCONNECTED);
        IntentFilter              fil4              =              new
IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
        this.registerReceiver(mReceiver,fil1);
        this.registerReceiver(mReceiver,fil2);
        this.registerReceiver(mReceiver,fil3);
        this.registerReceiver(mReceiver,fil4);

        //Listener para el switch
        mySwitch.setEnabled(false);
        mySwitch.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            @Override
```

```
            public void onCheckedChanged(CompoundButton
compoundButton, boolean b) {
                //b=valor al que se cambia
                if (b) {
                    SendSiAMP(Protocol.MODOCONT, 0); //Paso a modo
continuo
                    MainTxt.setText("");
                } else {
                    SendSiAMP(Protocol.MODOAVISO, 0); //Paso a modo
de aviso en caso de alerta
                    MainTxt.setText(R.string.idle);
                    BpmTxt.setText("");
                }
            }
        });
        BpmTxt.setText("");
        MainTxt.setText(R.string.siamp_not_connected);

        // Creación de la alerta
        builder = new AlertDialog.Builder(mySelf); //Instanciación
del constructor de cuadros de alerta
        alert = builder.create();
        //Botón de envío de la alerta
        alert.setButton(DialogInterface.BUTTON_POSITIVE,
getResources().getString(R.string.possible_alert_button_send),     new
DialogInterface.OnClickListener(){
            @Override
            public void onClick(DialogInterface dialogInterface, int
whichButton){
                sendAlert(); //Iniciar el proceso de envío de alerta
                notificationManager.cancel(noti_id2);   //Cancelación
de la notificación de posible alerta
                timer.cancel();
            }
        });
        //Botón de cancelación de envío de la alerta
        alert.setButton(DialogInterface.BUTTON_NEGATIVE,
getResources().getString(R.string.possible_alert_button_do_not_send),n
ew DialogInterface.OnClickListener(){
```

```
            @Override
            public void onClick(DialogInterface dialogInterface, int
whichButton){
                SendSiAMP(Protocol.ALERTAOK,0); //Enviar ALERTOK
                AlertStatus = false;
                notificationManager.cancel(noti_id2);  //Cancelar  la
notifiación de posible alerta
                timer.cancel();
                alert.cancel(); //Cerrar el cuadro de diálogo
            }
        });
        alert.setCanceledOnTouchOutside(false);   //Evitar   que   se
cancele el cuadro de alerta tocando fuera de él

alert.setTitle(getResources().getString(R.string.possible_alert_detect
ed_title)); //Configurar título y contenido del cuadro de alerta

alert.setMessage(getResources().getString(R.string.possible_alert_dete
cted)+"        "        +        String.valueOf(AlarmSentMillis/1000)+"
"+getResources().getString(R.string.seconds));

        if
(ContextCompat.checkSelfPermission(this,Manifest.permission.ACCESS_FIN
E_LOCATION)!= PackageManager.PERMISSION_GRANTED) {
            if
(ActivityCompat.shouldShowRequestPermissionRationale(this,Manifest.per
mission.ACCESS_FINE_LOCATION)) {
            } else {
                ActivityCompat.requestPermissions(this,new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},LOCATION_PERMISSION
_REQUESTED);
            }
        }

        timer = new CountDownTimer(AlarmSentMillis,1000) {
            @Override
            public void onTick(long millisUntilFinished) {
```

```
alert.setMessage(getResources().getString(R.string.possible_alert_dete
cted)+"       "       +       String.valueOf(millisUntilFinished/1000)+"
"+getResources().getString(R.string.seconds));
                }

                @Override
                public void onFinish() {
                    NotificationManager       notificationManager       =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE); //Quitar
la notificación de posible alerta
                    notificationManager.cancel(noti_id2);
                    alert.cancel(); //Quitar el cuadro de alerta
                    sendAlert(); //Enviar el proceso de envío de alerta
                }
            };

            Notification n = new Notification.Builder(mySelf)

.setContentTitle(getResources().getString(R.string.siamp_not_connected
))

.setContentText(getResources().getString(R.string.please_connect))

.setTicker(getResources().getString(R.string.siamp_not_connected))
                    .setSmallIcon(R.mipmap.ic_notification)

.setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICAT
ION))
                    .setOngoing(true)
                    .setPriority(Notification.PRIORITY_MAX)
                    .setContentIntent(PendingIntent.getActivity(mySelf,
0, new Intent(mySelf, MainActivity.class), 0))
                    .build();
            notificationManager.notify(noti_id, n);
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
```

```
        // Inflate the menu; this adds items to the action bar if it
is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        //Asignar cada ítem de menú a la variable que le representa
        RequestThrsMenuItem = menu.findItem(R.id.request_thrs);
        ResetMenuItem = menu.findItem(R.id.reset);
        ConnectMenuItem = menu.findItem(R.id.force_connect);
        ConfigureMenuItem = menu.findItem(R.id.configure);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        //Crear la alerta
        final     AlertDialog.Builder     LocalBuilder     =     new
AlertDialog.Builder(mySelf);
        final AlertDialog LocalAlert = LocalBuilder.create();
        //Crear un campo de texto editable
        final EditText input = new EditText(this);
        switch (item.getItemId()) { //Detectar qué ítem de menú se ha
pulsado
            case R.id.request_thrs: //Pedir umbrales
                SendSiAMP(Protocol.PEDIRTHR,0);
                break;
            case R.id.force_connect: //Conectar
                //Iniciar conexión
                try{
                    FindSiAMP();
                    SetupBTConnection();
                }catch (IOException ex) { }
                break;
            case R.id.configure: //Configurar
                LocalAlert.setTitle(R.string.password);  //Configurar
el diálogo de introducción de contraseña
                LocalAlert.setView(input);
                input.setInputType(InputType.TYPE_CLASS_TEXT        |
InputType.TYPE_TEXT_VARIATION_PASSWORD);
```

```
                //Listener del botón positivo
                LocalAlert.setButton(DialogInterface.BUTTON_POSITIVE,
getResources().getString(R.string.ok), new
DialogInterface.OnClickListener(){
                    @Override
                    public void onClick(DialogInterface
dialogInterface, int whichButton){
                        final String pw =
SharedPrefs.getString("password",PresetPassword);//Obtener la
contraseña
                        if (pw.equals(input.getText().toString())){
//Si es igual a la introducida
                            Intent i = new Intent(mySelf,
SettingsActivity.class); //Abrir la actividad de configuración
                            startActivity(i);
                        }else{
                            LocalAlert.cancel(); //Cerrrar el cuadro
de diálogo

Toast.makeText(mySelf,R.string.wrong_password,
Toast.LENGTH_SHORT).show();//Notificar al usuario
                        }
                    }
                });
                //Listener del botón negativo
                LocalAlert.setButton(DialogInterface.BUTTON_NEGATIVE,
getResources().getString(R.string.cancel), new
DialogInterface.OnClickListener(){
                    @Override
                    public void onClick(DialogInterface
dialogInterface, int whichButton){
                        LocalAlert.cancel(); //Cerrrar la alerta
                    }
                });
                LocalAlert.show();//Mostrar la alerta
                break;
            case R.id.reset: //Reiniciar
                SendSiAMP(Protocol.ALERTAOK,0);//Enviar señal de
ALERTAOK
```

```
                AlertStatus = false;
                break;
            }
            return super.onOptionsItemSelected(item);
        }

        @Override
        public boolean onPrepareOptionsMenu(Menu menu) {
            //Habilitar o deshabilitar las opciones del menú en función
del estado de SiAMP

RequestThrsMenuItem.setEnabled(ConnectionStatus&InitializationStatus);
            ResetMenuItem.setEnabled(AlertStatus);
            ConnectMenuItem.setEnabled(!ConnectionStatus);
            ConfigureMenuItem.setEnabled(ConnectionStatus);
            return super.onPrepareOptionsMenu(menu);
        }


        @Override
        protected void onResume() {
            super.onResume();
            //Comprobar que ninguno de los dos umbrales excede 255
            final SharedPreferences.Editor editor = SharedPrefs.edit();
            if            (Integer.parseInt(SharedPrefs.getString("lthr",
"50"))>255){
                editor.putString("lthr", "255");
                editor.apply();
            }
            if            (Integer.parseInt(SharedPrefs.getString("hthr",
"50"))>255){
                editor.putString("hthr", "255");
                editor.apply();
            }
            //Comprobar que ninguno de los dos umbrales es menor de 0
            if (Integer.parseInt(SharedPrefs.getString("lthr", "50"))<0){
                editor.putString("lthr", "0");
```

```
            editor.apply();
        }
        if (Integer.parseInt(SharedPrefs.getString("hthr", "50"))<0){
            editor.putString("hthr", "0");
            editor.apply();
        }
        if
(!String.valueOf(lthr).equals(SharedPrefs.getString("lthr","50"))){
//Si ha cambiado el umbral inferior
            lthr                                                    =
Integer.parseInt(SharedPrefs.getString("lthr","50"));  //Guardarlo  en
la variable global
            SendSiAMP(Protocol.CAMBIOLTHR,lthr);    //Enviarlo    al
microcontrolador
        }
        if
(!String.valueOf(hthr).equals(SharedPrefs.getString("hthr","180"))){
//Si ha cambiado el umbral superior
            hthr                                                    =
Integer.parseInt(SharedPrefs.getString("hthr","180")); //Guardarlo  en
la variable global
            SendSiAMP(Protocol.CAMBIOHTHR,hthr);    //Enviarlo    al
microcontrolador
        }
    }

    @Override
    public  void  onRequestPermissionsResult(int  requestCode,String
permissions[], int[] grantResults) {
        switch (requestCode) {
            case LOCATION_PERMISSION_REQUESTED: {
                if  (grantResults.length  >  0  &&  grantResults[0]  ==
PackageManager.PERMISSION_GRANTED) {
                    //Si   se   han   obtenido   los   permisos   de
localización, pedir localización
                    mGPSManager.getLocation();
                } else {

                }
```

```
                return;
            }
        }
    }

    void FindSiAMP(){
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if(mBluetoothAdapter == null){
        }

        if(!mBluetoothAdapter.isEnabled() && !BtTurningOn){
            Intent enableBluetooth = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBluetooth, 0);
        }
        Set<BluetoothDevice> pairedDevices =
mBluetoothAdapter.getBondedDevices();
        if(pairedDevices.size() > 0){
            for(BluetoothDevice device : pairedDevices){
                if(device.getName().equals(DeviceName)){
                    mmDevice = device;
                    BtTurningOn=false;
                    break;
                }
            }
        }
    }

    void SetupBTConnection() throws IOException{
        if (BluetoothAdapter.getDefaultAdapter().isEnabled()) {
            UUID uuid = UUID.fromString("0d03c278-1467-11e7-93ae-
92361f002671");
            mmSocket =
mmDevice.createRfcommSocketToServiceRecord(uuid);
            mmSocket.connect();
            mmOutputStream = mmSocket.getOutputStream();
            mmInputStream = mmSocket.getInputStream();
```

```
            StartListening();
            NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);//Cancelar la notificación en caso de haberla
            notificationManager.cancel(noti_id);
            new CountDownTimer(WaitMillis, WaitMillis) { //Configurar el timer para esperar hasta enviar la configuración inicial
                public void onTick(long millisUntilFinished) {
                }

                public void onFinish() {
                    Toast.makeText(mySelf, R.string.initializing, Toast.LENGTH_SHORT).show(); //Enviar la configuración inicial
                    SendSiAMP(Protocol.CAMBIOHTHR, hthr & 0xFF);
                    WaitForFirstResponse = true;
                }
            }.start(); //Ejecutar el timer
        }
    }

    void StartListening(){
        final Handler handler = new Handler();
        final byte delimiter = 10;

        stopWorker = false;
        readBufferPosition = 0;
        readBuffer = new byte[1024];
        workerThread = new Thread(new Runnable(){
            public void run(){
                while(!Thread.currentThread().isInterrupted() && !stopWorker){
                    try{
                        int bytesAvailable = mmInputStream.available();
                        if(bytesAvailable > 0){
                            byte[] packetBytes = new byte[bytesAvailable];
```

```
                                mmInputStream.read(packetBytes);
                                for(int i=0;i<bytesAvailable;i++){
                                    byte b = packetBytes[i];
                                    if(b == delimiter){
                                        byte[]    encodedBytes    =    new
byte[readBufferPosition];
                                        System.arraycopy(readBuffer,   0,
encodedBytes, 0, encodedBytes.length);
                                        final    String    data    =    new
String(encodedBytes, "US-ASCII");
                                        readBufferPosition = 0;
                                        handler.post(new Runnable(){
                                            public void run(){
                                                Intent     i     =     new
Intent(DATA_RECEIVED);
                                                i.putExtra("data",data);
                                                sendBroadcast(i);
                                            }
                                        });
                                    }else{
                                        readBuffer[readBufferPosition++]
= b;
                                    }
                                }
                            }
                        }
                        catch (IOException ex){
                            stopWorker = true;
                        }
                    }
                }
            });

            workerThread.start();
        }

        private void SendSiAMP(int dato1, int dato2){
```

```
        byte[] msg = {(byte) dato1, (byte) dato2}; //Creación del
paquete de datos
        try {
            mmOutputStream.write(msg);
        }catch (IOException e){}
    }
    private void alertPopUp(){
        //Creación de la notificación de la alerta
        Notification n = new Notification.Builder(mySelf)

.setContentTitle(getResources().getString(R.string.possible_alert_dete
cted_title))

.setContentText(getResources().getString(R.string.possible_alert_detec
ted))

.setTicker(getResources().getString(R.string.possible_alert_detected_t
itle))
                .setSmallIcon(R.mipmap.ic_notification)

.setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICAT
ION))
                .setLights(0xFF9900, 100, 100)
                .setContentIntent(PendingIntent.getActivity(mySelf,
0, new Intent(mySelf, MainActivity.class), 0))
                .build();
        if (!AlertStatus){ //Si no hay alarma
            AlertStatus = true;
            final    NotificationManager    notificationManager    =
(NotificationManager)  getSystemService(NOTIFICATION_SERVICE);  //Envío
de la notificaión
            notificationManager.notify(noti_id2, n);
            timer.start();
            alert.show(); //Mostrar el cuadro de diálogo
        }
    }
    private void sendAlert() {
        String add = getResources().getString(R.string.no_location);
//Mensaje por defecto
```

```
        //Tomar la fecha y la hora del sistema
        GregorianCalendar calendar = new GregorianCalendar();
        String datestr = calendar.get(GregorianCalendar.DAY_OF_MONTH)
+    "/"    +    calendar.get(GregorianCalendar.MONTH)    +    "/"    +
calendar.get(GregorianCalendar.YEAR);
        String hourstr = calendar.get(GregorianCalendar.HOUR_OF_DAY)
+ ":" + calendar.get(GregorianCalendar.MINUTE);
        double  latitude  =  mGPSManager.getLatitude();  //Tomar  las
coordenadas
        double longitude = mGPSManager.getLongitude();
        Geocoder  geocoder  =  new  Geocoder(mySelf);  //Instanciar  el
geocoder para hacer un geocoding inverso
        try {
            List<Address> list = null; //Instanciar una lista de
direcciones
            Address address = null; //Instanciar una dirección
            list = geocoder.getFromLocation(latitude, longitude, 1);
//Tomar la lista de direcciones proporcionadas por el geocoder
            address = list.get(0); //Tomar la primera dirección

MainTxt.setText(getResources().getString(R.string.alert_1)
//Introducir la dirección en el campo de texto estático
                    + " " + datestr + " "
                    + getResources().getString(R.string.alert_2) + "
"
                    +       hourstr       +       "       "       +
getResources().getString(R.string.approx_address) + "\n"
                    +    address.getAddressLine(0)    +    "    "    +
address.getLocality() + " " + address.getSubAdminArea()
                    + " " + address.getCountryName());
            add    =    address.getAddressLine(0)    +    "    "    +
address.getLocality() + " " + address.getSubAdminArea(); //Crear la
cadena de texto con la dirección
        } catch (IOException ex) {
        } catch (IndexOutOfBoundsException ex){
            MainTxt.setText(R.string.localization_error);   //Mensaje
de error en el campo de texto estático
```

```
        }
        Notification not = new Notification.Builder(this)
//Construcción y envío de la notificación de envío de alerta

.setContentTitle(getResources().getString(R.string.alert))

.setContentText(getResources().getString(R.string.alert_detected))

.setTicker(getResources().getString(R.string.app_name) + ": " +
getResources().getString(R.string.alert))
                .setSmallIcon(R.mipmap.ic_notification)

.setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICAT
ION))
                .build();
        NotificationManager notificationManager =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        notificationManager.notify(0, not);
        try {
            sendSms(tel, SharedPrefs.getString("name", "[Sin
nombre]") + R.string.sms + add); //Envío de SMS
        } catch (Exception e) {
            Toast.makeText(this,
getResources().getString(R.string.sms_generic_error),
Toast.LENGTH_SHORT).show();//Notificación en caso de error
        }
    }
    private void sendSms(String num, String msg) {
        PendingIntent PIntentSent = PendingIntent.getBroadcast(this,
0, new Intent(SMS_SENT), 0); //Declaración de los PendingIntent
        PendingIntent PIntentDelivered =
PendingIntent.getBroadcast(this, 0, new Intent(SMS_DELIVERED), 0);
        registerReceiver(new BroadcastReceiver() {
//BroadcastReceiver local que comprueba el envío de SMS
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                String msg = null;
                switch (getResultCode()) {
```

```
                    case Activity.RESULT_OK: //Se ha enviado
correctamente
                        msg =
getResources().getString(R.string.sms_sent);
                        break;
                    case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
//Error genérico
                        msg =
getResources().getString(R.string.sms_generic_error);
                        break;
                    case SmsManager.RESULT_ERROR_NO_SERVICE: //No hay
cobertura
                        msg =
getResources().getString(R.string.sms_service_unavailable);
                        break;
                    case SmsManager.RESULT_ERROR_RADIO_OFF: //Red
deshabilitada
                        msg =
getResources().getString(R.string.sms_radio_off);
                        break;
                    case SmsManager.RESULT_ERROR_NULL_PDU:
//Protocolo no válido
                        msg =
getResources().getString(R.string.sms_null_pdu);
                        break;
                }
                Notification not = new Notification.Builder(mySelf)
//Construcción y envío de la notificación

.setContentTitle(getResources().getString(R.string.sms_noti_title))
                        .setContentText(msg)

.setTicker(getResources().getString(R.string.app_name) + ": "
+getResources().getString(R.string.sms_noti_title))
                        .setSmallIcon(R.mipmap.ic_notification)
```

```
.setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION))
                        .build();
                NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
                notificationManager.notify(0, not);
            }
        }, new IntentFilter(SMS_SENT));
        registerReceiver(new BroadcastReceiver() { //BroadcastReceiver local que comprueba la llegada del mensaje
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                String msg = null;
                switch (getResultCode()) {
                    case Activity.RESULT_OK: //Ha sido recibido
                        msg = getResources().getString(R.string.sms_delivered);
                        break;
                    case Activity.RESULT_CANCELED: //No ha sido recibido
                        msg = getResources().getString(R.string.sms_not_delivered);
                        break;
                }
                Notification not = new Notification.Builder(mySelf) //Creación y envío de la notificación

.setContentTitle(getResources().getString(R.string.sms_noti_title))
                        .setContentText(msg)

.setTicker(getResources().getString(R.string.app_name) + ": " +getResources().getString(R.string.sms_noti_title))
                        .setSmallIcon(R.mipmap.ic_notification)

.setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION))
                        .build();
```

```
                NotificationManager notificationManager =
(NotificationManager) getSystemService(NOTIFICATION_SERVICE);
                notificationManager.notify(0, not);
            }
        }, new IntentFilter(SMS_DELIVERED));
        SmsManager smsmanager = SmsManager.getDefault(); //Tomar el
gestor de SMS del sistema
        smsmanager.sendTextMessage(num, null, msg, PIntentSent,
PIntentDelivered); //Enviar el SMS
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(mReceiver);
        stopWorker = true;
        try {
            mmOutputStream.close();
            mmInputStream.close();
            mmSocket.close();
        } catch (IOException e) {
        }
    }
    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        //Asignar el Switch y el campo de texto a las variables
locales para acceder a su contenido
        TextView tv = (TextView) findViewById(R.id.main_txt);
        Switch sw = (Switch) findViewById(R.id.switch1);
        //Guardar el estado de ejecución de la aplicación en el
Bundle
        outState.putBoolean("InitializationStatus",
InitializationStatus);
        outState.putBoolean("AlertStatus", AlertStatus);
        outState.putCharSequence("Text",tv.getText());
        outState.putBoolean("SwitchIsEnabled",sw.isEnabled());
```

```
        outState.putBoolean("SwitchIsChecked",sw.isChecked());
    }

    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState)
{
        super.onRestoreInstanceState(savedInstanceState);
        //Asignar el Switch y el campo de texto a las variables
locales para modificar sus datos
        TextView tv = (TextView) findViewById(R.id.main_txt);
        Switch sw = (Switch) findViewById(R.id.switch1);
        //Tomar los datos guardados en el Bundle
        AlertStatus = savedInstanceState.getBoolean("AlertStatus");
        InitializationStatus                                    =
savedInstanceState.getBoolean("InitializationStatus");
        //Poner el texto en el campo de texto estático
        tv.setText(savedInstanceState.getCharSequence("Text"));
        //Configurar el Switch para dejarlo en el estado en el que se
encontraba antes de destruir el proceso

sw.setEnabled(savedInstanceState.getBoolean("SwitchIsEnabled"));

sw.setChecked(savedInstanceState.getBoolean("SwitchIsChecked"));
        //Si había una alerta, mostrar el cuadro de diálogo
        if (AlertStatus) {
            alert.show();
        }
    }
 }
```

11.1.2.1.3 PasswordChange.java

```
 package com.andcotjim.siampv3;

 import android.app.Activity;
 import android.app.AlertDialog;
 import android.content.DialogInterface;
```

```
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;


public class PasswordChange extends Activity {
    private Button button, backButton; //Variables para los botones
    private EditText editText1, editText2, editText3; //Variables
para los campos de texto editable
    private SharedPreferences SharedPrefs; //Instancia de
SharedPreferences
    private Activity mySelf;
    private static final String PresetPassword = "aaaa"; //Contraseña
predefinida
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_password_change);
//Configurar el XML que contiene el layout
        mySelf = this;
        SharedPrefs =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext())
;
        final SharedPreferences.Editor editor = SharedPrefs.edit();
        editText1 = (EditText) findViewById(R.id.editText); //Asignar
a cada variable de campo
        editText2 = (EditText) findViewById(R.id.editText2); //de
texto editable su widget
        editText3 = (EditText) findViewById(R.id.editText3);
        button = (Button) findViewById(R.id.button); //Asignar los
botones
        backButton = (Button) findViewById(R.id.button2);
```

```
        button.setOnClickListener(new Button.OnClickListener() {
//Listener para el botón de confirmación
            private String OldPass;
            private AlertDialog.Builder alert;
            @Override
            public void onClick(View view) {
                alert = new AlertDialog.Builder(mySelf);
                String NewPass1=editText2.getText().toString();
//Tomar los textos introducidos
                String NewPass2=editText3.getText().toString();
                OldPass = SharedPrefs.getString("password",
PresetPassword);
                if (OldPass.equals(editText1.getText().toString())){
//Comprobar que la contraseña actual es correcta
                    if (NewPass1.equals(NewPass2)){ //Comprobar que
las contraseñas nuevas coincidan
                        if(NewPass1.length()>=4){ //Comprobar que la
contraseña tiene al menos 4 caracteres

editor.putString("password",NewPass1);//Actualizar el registro de la
contraseña
                            editor.commit();
                            Toast.makeText(mySelf,
R.string.password_change_success, Toast.LENGTH_SHORT).show();
//Notificar al usuario
                            mySelf.finish(); //Cerrrar la actividad
                        }else{ //Si la contraseña es demasiado corta,
avisar al usuario
                            alert.setTitle(R.string.error);

alert.setMessage(R.string.password_too_short);
                            alert.setNegativeButton(R.string.ok,new
DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface
dialogInterface, int i) {
                                }
                            });
                            alert.show();
```

```
                        }
                    }else{ //Si las contraseñas no coinciden, avisar
al usuario
                        alert.setTitle(R.string.error);

alert.setMessage(R.string.passwords_dont_match);
                        alert.setNegativeButton(R.string.ok,new
DialogInterface.OnClickListener() {
                            @Override
                            public    void    onClick(DialogInterface
dialogInterface, int i) {
                            }
                        });
                        alert.show();
                    }
                }else{ //Si la contraseña introducida no coincide con
la actual, avisar al usuario
                    alert.setTitle(R.string.error);

alert.setMessage(R.string.wrong_password_explanation);
                    alert.setNegativeButton(R.string.ok,new
DialogInterface.OnClickListener() {
                        @Override
                        public      void      onClick(DialogInterface
dialogInterface, int i) {
                        }
                    });
                    alert.show();
                }
            }
        });
        //Listener para el botón de cancelación
        backButton.setOnClickListener(new Button.OnClickListener() {
            @Override
            public void onClick(View view) {
                mySelf.finish(); //Cerrar la actividad
            }
```

```
        });
    }

    @Override
    public boolean onMenuItemSelected(int featureId, MenuItem item) {
        return super.onMenuItemSelected(featureId, item);
    }
}
```

11.1.2.1.4 Protocol.java

```
package com.andcotjim.siampv3;

public class Protocol {
    public static final int MODOCONT = 'A';
    public static final int MODOAVISO = 'B';
    public static final int CAMBIOHTHR = 'C';
    public static final int CAMBIOLTHR = 'D';
    public static final int PEDIRTHR = 'E';
    public static final int ALERTAOK = 'F';
    public static final String RESPONSE ="~";
    public static final String ALERT = "!";
    public static final String HTHR = "^";
    public static final String LTHR = "v";
    public static final String BPM = "#";
}
```

11.1.2.1.5 SettingsActivity.java

```
package com.andcotjim.siampv3;

import android.app.Activity;
import android.os.Bundle;
import android.preference.PreferenceFragment;
import android.view.MenuItem;
```

```
  public class SettingsActivity extends Activity {
      @Override
      protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          //Creación del fragmento de configuración

getFragmentManager().beginTransaction().replace(android.R.id.content,
new SettingsFragment()).commit();
      }
      public static class SettingsFragment extends PreferenceFragment {
          @Override
          public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              //Añadir las preferencias desde el archivo XML
              addPreferencesFromResource(R.xml.preferences);
          }
      }

      @Override
      public boolean onMenuItemSelected(int featureId, MenuItem item) {
          return super.onMenuItemSelected(featureId, item);
      }
  }
```

11.1.2.2 XML

11.1.2.2.1 /layout/activity_main.xml

```
  <?xml version="1.0" encoding="utf-8"?>
  <android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    tools:context="com.andcotjim.siampv3.MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </android.support.design.widget.AppBarLayout>

    <include layout="@layout/content_main" />

</android.support.design.widget.CoordinatorLayout>
```

11.1.2.2.2 /layout/content_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >

    <TextView
        android:id="@+id/main_txt"
        android:textSize="9pt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        android:layout_marginTop="35dp"
```

```
        android:layout_below="@+id/switch1"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <Switch
        android:id="@+id/switch1"
        android:textSize="10pt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="79dp"
        android:text="@string/cont_mode"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:id="@+id/bpm_txt"
        android:textSize="10pt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/switch1"
        android:layout_marginLeft="22dp"
        android:layout_marginStart="22dp"
        android:layout_toEndOf="@+id/switch1"
        android:layout_toRightOf="@+id/switch1"
        android:text="TextView" />

  </RelativeLayout>
```

11.1.2.2.3 /layout/activity_password_change.xml

```
  <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
        android:layout_height="match_parent"
        tools:context="com.andcotjim.siampv3.PasswordChange"
        android:orientation="vertical">
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"

android:textAppearance="?android:attr/textAppearanceMedium"
                android:text="@string/old_password"
                android:id="@+id/textView"/>

            <EditText
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:inputType="textPassword"
                android:ems="10"
                android:id="@+id/editText" />

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"

android:textAppearance="?android:attr/textAppearanceMedium"
                android:text="@string/new_password"
                android:id="@+id/textView2" />

            <EditText
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:inputType="textPassword"
                android:ems="10"
                android:id="@+id/editText2" />

            <TextView
                android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"

android:textAppearance="?android:attr/textAppearanceMedium"
            android:text="@string/confirm_password"
            android:id="@+id/textView3" />

        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:inputType="textPassword"
            android:ems="10"
            android:id="@+id/editText3" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/confirm_changes"
            android:id="@+id/button"
            android:layout_gravity="center_horizontal" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/back"
            android:id="@+id/button2"
            android:layout_gravity="center_horizontal" />
    </LinearLayout>

 </LinearLayout>
```

11.1.2.2.4 /menu/menu_main.xml

```
 <menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.andcotjim.siampv3.MainActivity">
    <item android:id="@+id/force_connect"
```

```
        android:title="@string/force_connect"
        android:orderInCategory="1"/>
    <item android:id="@+id/request_thrs"
        android:title="@string/request_thrs"
        android:orderInCategory="2"/>
    <item android:id="@+id/configure"
        android:title="@string/configure"
        android:orderInCategory="3"/>
    <item android:id="@+id/reset"
        android:title="@string/reset"
        android:orderInCategory="4"/>
</menu>
```

11.1.2.2.5 /values/dimens.xml

```
<resources>
    <dimen name="fab_margin">16dp</dimen>
</resources>
```

11.1.2.2.6 /values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">SiAMP</string>
    <string name="connect">Conectar</string>
    <string name="searching_device">Buscando SiAMP</string>
    <string name="connected">SiAMP conectado</string>
    <string name="disconnected">SiAMP desconectado</string>
    <string name="alert_1">Alerta el </string>
    <string name="alert_2"> a las </string>
    <string name="ok">Aceptar</string>
    <string name="cancel">Cancelar</string>
    <string name="alert_thrs">Umbrales de aviso</string>
    <string name="high">Superior: </string>
```

```
    <string name="low">Inferior: </string>
    <string name="cont_mode">Modo continuo</string>
    <string name="alert_mode">Modo de alerta</string>
    <string name="idle">Modo de aviso en alerta. Sin alertas.</string>
    <string name="new_high_thr">Nuevo umbral de aviso superior</string>
    <string name="new_low_thr">Nuevo umbral de aviso inferior</string>
    <string name="connection_error">ERROR: dispositivo no conectado.</string>
    <string name="request_thrs">Pedir umbrales de aviso</string>
    <string name="approx_address">Dirección aproximada: </string>
    <string name="force_connect">Conectar SiAMP</string>
    <string name="alert_detected">Alerta detectada. Avisando a los servicios de emergencia.</string>
    <string name="alert">¡Alerta detectada!</string>
    <string name="sms_sent">SMS enviado con éxito.</string>
    <string name="sms_generic_error">Ha ocurrido un error con el envío del SMS.</string>
    <string name="sms_service_unavailable">No se ha podido enviar el SMS: no hay cobertura.</string>
    <string name="sms_radio_off">No se ha podido enviar el SMS: la red está deshabilitada.</string>
    <string name="sms_null_pdu">Error al enviar SMS: PDU nulo</string>
    <string name="sms_noti_title">Aviso de SMS</string>
    <string name="sms_delivered">SMS recibido con éxito.</string>
    <string name="sms_not_delivered">SMS no recibido</string>
    <string name="siamp_not_connected">SiAMP no conectado.</string>
    <string name="please_connect">Por favor, inicie la aplicación SiAMP.</string>
    <string name="possible_alert_detected_title">¡Posible alerta detectada!</string>
    <string name="possible_alert_detected">Se ha detectado una posible alerta. Si es una falsa alarma pulse "No enviar", en caso
```

contrario pulse "Enviar". La alarma se enviará automáticamente en </string>

<string name="**seconds**"> **segundos.**</string>

<string name="**possible_alert_button_send**">**Enviar**</string>

<string name="**possible_alert_button_do_not_send**">**No enviar**</string>

<string name="**initializing**">**Inicializando**…</string>

<string name="**initializing_success**">**SiAMP inicializado.**</string>

<string name="**thrs_success**">**Umbral enviado**</string>

<string name="**not_connected**">**SiAMP no conectado**</string>

<string name="**localization_error**">**Error en la localización, comprobar estado del GPS.**</string>

<string name="**configure**">**Configurar SiAMP**</string>

<string name="**title_activity_configure**">**Configuración**</string>

<string name="**action_settings**">**Settings**</string>

<string name="**password**">**Introducir contraseña**</string>

<string name="**wrong_password**">**Contraseña errónea**</string>

<string name="**title_activity_settings**">**Configuración**</string>

<string name="**thrs_change**">**Cambio de umbrales**</string>

<string name="**patient_name**">**Nombre del paciente**</string>

<string name="**personal_data**">**Datos personales**</string>

<string name="**password_title**">**Contraseña**</string>

<string name="**password_change**">**Cambiar la contraseña**</string>

<string name="**title_activity_password_change**">**Cambiar la contraseña**</string>

<string name="**old_password**">**Contraseña actual**</string>

<string name="**new_password**">**Nueva contraseña**</string>

<string name="**confirm_password**">**Confirmar contraseña**</string>

<string name="**confirm_changes**">**Confirmar cambios**</string>

<string name="**wrong_password_explanation**">**La contraseña actual no es correcta.**</string>

<string name="**error**">**¡Error!**</string>

<string name="**passwords_dont_match**">**Las nuevas contraseñas no coinciden**</string>

<string name="**password_too_short**">**La contraseña es demasiado corta, debe tener al menos 4 caracteres.**</string>

<string name="**password_change_success**">**Contraseña cambiada con éxito.**</string>

<string name="**back**">**Atrás**</string>

```
    <string name="sms"> puede necesitar asistencia en </string>
    <string name="no_location">Sin dirección aproximada</string>
    <string name="reset">Reiniciar SiAMP</string>
    <string     name="insert_patient_name">Inserte     nombre     del
paciente</string>
    <string  name="insert_new_hthr">Inserte  nuevo  umbral  de  aviso
superior</string>
    <string  name="insert_new_lthr">Inserte  nuevo  umbral  de  aviso
inferior</string>

    <string       name="navigation_drawer_open">Open       navigation
drawer</string>
    <string      name="navigation_drawer_close">Close      navigation
drawer</string>
  </resources>
```

11.1.2.2.7 /values/styles.xml

```
  <resources>

    <!-- Base application theme. -->
    <style                                              name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>
```

```
    <style                                  name="AppTheme.AppBarOverlay"
parent="ThemeOverlay.AppCompat.Dark.ActionBar" />

    <style                                   name="AppTheme.PopupOverlay"
parent="ThemeOverlay.AppCompat.Light" />

  </resources>
```

11.1.2.2.8 /xml/preferences.xml

```
  <?xml version="1.0" encoding="utf-8"?>

  <PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android">
      <PreferenceCategory
          android:title="@string/thrs_change">
          <EditTextPreference
              android:key="hthr"
              android:inputType="number"
              android:dialogMessage="@string/insert_new_hthr"
              android:title="@string/new_high_thr"/>
          <EditTextPreference
              android:key="lthr"
              android:inputType="number"
              android:dialogMessage="@string/insert_new_lthr"
              android:title="@string/new_low_thr"/>
      </PreferenceCategory>
      <PreferenceCategory
          android:title="@string/personal_data">
          <EditTextPreference
              android:key="name"
              android:dialogMessage="@string/insert_patient_name"
              android:title="@string/patient_name"/>
      </PreferenceCategory>
      <PreferenceCategory
          android:title="@string/password_title">
          <PreferenceScreen
              android:title="@string/password_change">
```

```
            <intent android:targetPackage="com.andcotjim.siampv3"

android:targetClass="com.andcotjim.siampv3.PasswordChange"/>
         </PreferenceScreen>
     </PreferenceCategory>
  </PreferenceScreen>
```

11.1.2.2.9 AndroidManifest.xml

```
  <?xml version="1.0" encoding="utf-8"?>
  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.andcotjim.siampv3">

      <uses-permission android:name="android.permission.BLUETOOTH" />
      <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
      <uses-permission android:name="android.permission.INTERNET" />
      <uses-permission  android:name="android.permission.ACCESS_FINE_LOCATION"
/>
      <uses-permission android:name="android.permission.SEND_SMS" />

      <application
          android:allowBackup="true"
          android:icon="@mipmap/ic_launcher"
          android:label="@string/app_name"
          android:supportsRtl="true"
          android:theme="@style/AppTheme">
          <activity
              android:name=".MainActivity"
              android:label="@string/app_name"
              android:theme="@style/AppTheme.NoActionBar">
              <intent-filter>
                  <action android:name="android.intent.action.MAIN" />
                  <category android:name="android.intent.category.LAUNCHER" />
              </intent-filter>
          </activity>
          <activity
              android:name=".SettingsActivity"
              android:label="@string/title_activity_configure"
```

```
            android:theme="@style/AppTheme.NoActionBar">
        </activity>
        <activity
            android:name=".PasswordChange"
            android:label="@string/title_activity_password_change"
            android:theme="@style/AppTheme.NoActionBar">
        </activity>
    </application>

</manifest>
```

REFERENCIAS

[1] S. Rodriquez, «Defibrillator | Flickr,» [En línea]. Available: https://www.flickr.com/photos/n28ive1/431939091/.

[2] Blausen.com staff, «Medical gallery of Blausen Medical 2014,» 2014. [En línea]. Available: https://en.wikiversity.org/wiki/WikiJournal_of_Medicine/Medical_gallery_of_Blausen_Medical_2014.

[3] «Wikipedia - Implantable Cardioverter Defibrillator,» [En línea]. Available: https://en.wikipedia.org/wiki/Implantable_cardioverter-defibrillator.

[4] Hazmat2, «ECG with annotations,» [En línea]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/3/34/EKG_Complex_en.svg/581px-EKG_Complex_en.svg.png.

[5] «Wikipedia - Electrocardiography,» [En línea]. Available: https://en.wikipedia.org/wiki/Electrocardiography.

[6] John, «Story and History of Development of Arduino - Circuits Today,» 26 Marzo 2014. [En línea]. Available: http://www.circuitstoday.com/story-and-history-of-development-of-arduino.

[7] Android, «Dashboards - Android Developers,» [En línea]. Available: https://developer.android.com/about/dashboards/index.html.

[8] «Wikipedia - Eclipse (software),» [En línea]. Available: https://en.wikipedia.org/wiki/Eclipse_(software).

[9] «Wikipedia - Android Studio,» [En línea]. Available: https://en.wikipedia.org/wiki/Android_Studio.

[10] «Wikipedia - Bluetooth,» [En línea]. Available: https://en.wikipedia.org/wiki/Bluetooth.

[11] J. C. J. R. a. L. J. J. D. A. M. J. Theuns, «Home monitoring in ICD therapy: future perspectives,» *Europace,* nº 5, p. 139–142, (2003).

[12] J. Daněk, «CARDIOLOGY ECG – basic information,» Marzo 2010. [En línea]. Available: http://noel.feld.cvut.cz/vyu/x31let/Lectures/05_EKGDanek.pdf.

[13] «Scilab Ninja - Module 7: Continuous to Discrete Conversion Methods,» [En línea]. Available: http://scilab.ninja/study-modules/scilab-control-engineering-basics/module-7-continuous-to-discrete-conversion-methods/.

[14] Arduino, «Arduino Uno REV3,» [En línea]. Available: https://store-usa.arduino.cc/products/a000066.

[15] Olimex, «SHIELD EKG-EMG,» [En línea]. Available: https://www.olimex.com/Products/Duino/Shields/SHIELD-EKG-EMG/.

[16] Micro4You, «HC-05,» [En línea]. Available: http://www.micro4you.com/store/images/source/Bluetooth_Module_bb.png.

[17] Olimex, «SHIELD EKG-EMG-PA,» [En línea]. Available: https://www.olimex.com/Products/Duino/Shields/SHIELD-EKG-EMG-PA/open-source-hardware.

[18] Android, «The Activity Lifecycle | Android developers,» [En línea]. Available: https://developer.android.com/guide/components/images/activity_lifecycle.png.

[19] F. Cejas, «Android Location Providers – gps, network, passive,» [En línea]. Available: https://developerlife.com/2010/10/20/gps/.

GLOSARIO