Universidad
de La Laguna

Escuela Superior de
Ingeniería y Tecnología
Sección de Ingeniería Informática

Trabajo de Fin de Grado

# Monitorización de variables fisiológicas mediante plataforma de bajo costo

*Monitoring of physiological variables through a low-cost platform*

Leonor Priego García

La Laguna, 2 de julio de 2015

D. **Juan Albino Méndez Pérez**, con N.I.F. 52824630-R, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

**C E R T I F I C A**

Que la presente memoria titulada:

*"Monitorización de variables fisiológicas mediante plataforma de bajo costo"*

ha sido realizada bajo su dirección por Dña. **Leonor Priego García**, con N.I.F. 54112657-M.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firma la presente en La Laguna a 2 de julio de 2015.

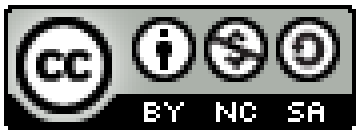
# Agradecimientos

A Juan Albino Méndez Pérez por haber sido mi tutor en el proyecto y haberme ayudado en todo lo posible.

A mi familia por estar siempre ahí apoyándome.

Y por último a todo el profesorado que me ha enseñado durante estos 4 años de carrera.

# Licencia



## Resumen

*El objetivo de este trabajo ha sido la monitorización de variables fisiológicas en dos aplicaciones, una móvil desarrollada mediante la tecnología Android y otra en PC desarrollada en Matlab. Las variables serán recogidas por diversos sensores conectados a una plataforma de bajo coste basada en Arduino y son enviadas mediante comunicación serial.*

*La aplicación realizada en Matlab será integrada en un software para la monitorización y control de anestesia ya desarrollado por el grupo de Ingeniería de Control de la Universidad de La Laguna.*

*Así mismo, la aplicación móvil podrá ser usada por cualquier persona que desee monitorizar en su hogar o en cualquier otro entorno sus propias variables fisiológicas*

*La aplicación desarrollada también permite almacenar en ficheros todas las variables recogidas para posteriores análisis y estudios. En el contexto de la línea de investigación que se desarrolla en el Grupo de Ingeniería de Control de la ULL, esto tiene mucha utilidad para obtener correlaciones entre estas variables y el estado analgésico (nivel de dolor) de los pacientes durante una intervención quirúrgica.*



**Palabras clave:** Arduino, Microcontrolador, Sensores, Monitorización, Variables fisiológicas.

## Abstract

*The objective of this project is the development of a solution to monitor the physiological variables in humans. The aim is to develop two applications, a mobile technology developed in Android and other developed in Matlab. These variables will be collected by several sensors connected to a low-cost platform such as Arduino and sent via serial communication.*

*The application made in Matlab will be integrated into a general software that the Control Engineering Group of the University of La Laguna has.*

*Also, the mobile application can be used by anyone who wants to control the state of his body at home or anywhere.*

*The developed application also allows save in files all the variables collected for further analysis and studies. In the context of the research taking place in the Control Engineering Group of the ULL, this is very useful for correlations between these variables and the state analgesic (pain level) of the patients during surgery .*



***Keywords:*** Arduino, Microcontroller, Sensors, Monitoring, Physiological variables.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1. Introducción

## 1.1 Motivación

La motivación principal para la realización de este Trabajo de Fin de Grado (TFG) ha sido la idea de diseñar y construir un sistema completo de monitorización de variables fisiológicas mediante una plataforma de bajo coste, poniendo en práctica los conocimientos adquiridos en las diversas materias de la titulación de Ingeniería Informática, especialidad en Ingeniería de Computadores.

Debido al fuerte carácter multidisciplinar de este proyecto, que representa la integración de varios campos como son: la creación de aplicaciones móviles con la tecnología Android, el uso de microcontroladores de bajo coste como Arduino y sus componentes, el uso de tecnologías de comunicación como Bluetooth y el desarrollo de una interfaz en el potente entorno de programación Matlab, es posible hacer una unión lo bastante atractiva como para decantarme para llevarlo a cabo.

En el desarrollo de proyectos como este es necesario tener en cuenta que el objetivo debe ser obtener un sistema suficientemente atractivo y fiable como para que las perspectivas de llegar a un producto comercializable sean realistas. Esta ha sido una de las motivaciones para decidir la implementación con un coste lo bastante reducido como para poder tener un valor añadido respecto a aparatos sofisticados dedicados a fines médicos.

Otra de las principales motivaciones para llevarlo a cabo ha sido su posible utilización dentro de los hogares, es decir, que las personas puedan monitorizar el estado de su cuerpo simplemente conectándose los sensores ellas mismas en sus casas. Puesto que no existe especificación de edad recomendada para el uso de esta plataforma, sino que, al contrario, lo puede usar desde un niño pequeño hasta una persona de edad adulta, se ha decidido desarrollar con una usabilidad adaptada a todo tipo de edades.

Por otra parte, esta plataforma también ha sido pensada para incorporarla a un proyecto de automatización de anestesia desarrollado por el Grupo de Ingeniería de Control de la Universidad de La Laguna en colaboración con un equipo de anestesistas del Hospital Universitario de Canarias. El objetivo del proyecto es el desarrollo de estrategias de control para la regulación del estado hipnótico (nivel de inconsciencia) y analgésico (nivel de dolor) en pacientes sometidos a anestesia general intravenosa. El sistema de control

diseñado persigue la regulación del nivel de inconsciencia y analgesia con rechazo de perturbaciones y optimización del proceso. Asimismo, se trabaja para el desarrollo de una herramienta que sirva de interfaz única y permita la monitorización del paciente, la comunicación con el sistema de infusión de fármacos y la gestión de toda la información generada en el proceso anestésico, liberando al anestesista de tareas rutinarias y permitiendo su concentración en otros puntos críticos que pudieran suponer una amenaza para la seguridad del paciente durante la intervención.

Por tanto, para servir como complemento al trabajo ya realizado, se busca obtener, registrar y representar varias de las variables fundamentales que pudieran estas relacionadas con la hipnosis y la analgesia del paciente en el proceso anestésico. Algunas de estas variable son el pulso, la cantidad de oxígeno en sangre, la temperatura corporal, la conductancia de la piel y el electrocardiograma (ECG). De esta forma, se pretende complementar el diseño de la interfaz realizada hasta el momento por el grupo de la ULL para añadir más información relativa al estado del paciente.

## 1.2 Antecedentes

Este trabajo se desarrolla dentro del grupo de investigación de Ingeniería de Control de la ULL. El grupo mantiene una línea activa relacionada con el diseño de sistemas automáticos de control aplicados a la anestesia. En particular, se ha abordado el control de la infusión de fármacos durante anestesia general para mejorar el rendimiento y seguridad del proceso.

Las variables de interés en el proceso anestésico son:

- Hipnosis: nivel de inconsciencia del paciente (relacionado con el recuerdo que tiene el paciente de la intervención).

- Analgesia: nivel de dolor que siente el paciente

- Relajación muscular: necesaria para garantizar el acceso a los órganos de forma segura.

El principal problema que surge cuando se aborda el control de este tipo de variables es que no son directamente medibles. Por ello, es necesario recurrir a medidas indirectas que estén correlacionadas con la variable de interés. En el caso de la hipnosis, se han propuesto diferentes medidas que permiten estimar el nivel de inconsciencia del paciente. De entre todas estas variables la más utilizada es el índice biespectral (BIS) que se extrae del electroencefalograma y varía entre 100 (despierto) y 0 (inactividad eléctrica). Durante la anestesia general el objetivo para esta variable se suele fijar en 50.

En el caso de la medida del nivel analgésico, la disponibilidad de variables que ofrezcan una estimación veraz del nivel de dolor que sufre el paciente es

más limitado. Actualmente, diferentes grupos de investigación en este campo están centrando su interés en la propuesta de un índice realmente eficiente que permita decidir qué dosis de fármaco hay que suministrar al paciente para garantizar un estado analgésico adecuado. Las variables que se están analizando para alcanzar este objetivo involucran mediciones de la frecuencia respiratoria, el electrocardiograma, electroencefalograma, conductancia de la piel, etc.

La línea abierta en el grupo de Ingeniería de Control de la ULL se centra actualmente en la analgesia y es, en esta línea donde tiene su origen la propuesta de este TFG. La idea es usar la nueva información que estaría disponible con la aplicación que se pretende desarrollar en este trabajo para modelar el proceso analgésico. Posteriormente se pretende desarrollar un sistema que permita el control automático de la infusión de fármaco analgésico.

Paralelamente se quiere iniciar con este trabajo una línea relacionada con la gestión personal de la información procedente de variables fisiológicas, tanto para uso médico como para aplicaciones en deporte u otras actividades.

## 1.3 Objetivos

El principal objetivo de este TFG se centra en realizar el diseño y construcción de una aplicación móvil en la que se monitoricen las variables recibidas a través de una plataforma Arduino mediante la tecnología Bluetooth. De esta manera, las personas se podrán monitorizar el estado de su cuerpo con una simple plataforma *low-cost* y una aplicación móvil en su *smartphone*.

La segunda parte de este trabajo de fin de grado se centra en complementar la aplicación de monitorización y control de anestesia desarrollada en el marco del proyecto de automatización de anestesia de la ULL, añadiéndole información de nuevas variables hasta ahora no registradas. Las nuevas variables son el pulso cardíaco, la saturación de oxígeno en sangre, la temperatura corporal, la conductancia de la piel y el electrocardiograma. Estas variables podrían ayudar a analizar el nivel de analgesia o dolor que sufre el paciente durante una intervención quirúrgica, dado que se parte de la hipótesis de que los estímulos dolorosos provocan cambios en la actividad del corazón, la temperatura y en la sudoración de la piel.

Por tanto, los objetivos específicos a desarrollar en este trabajo son:

- Recogida de variables fisiológicas usando una placa de desarrollo de Arduino por medio de diferentes sensores conectados al microcontrolador.

- Establecer una comunicación Bluetooth entre la aplicación móvil y un módulo Bluetooth conectado al microcontrolador Arduino, generando y extrayendo la información en el formato y protocolo adecuados.

- Creación de una aplicación móvil Android que permita recibir estas variables, monitorizarlas en la interfaz y registrarlas en un fichero para su posterior análisis.

- Establecer una comunicación a través de un puerto serial entre el ordenador y Arduino, generando y extrayendo la información en el formato y protocolo adecuados, para su posterior visualización en la interfaz.

- Creación de interfaz en PC en el entorno Matlab para la monitorización y registro en fichero de estas variables.

- Comprobación y validación de los resultados obtenidos.

- Incorporación de la nueva interfaz e información en la interfaz principal del proyecto de la ULL y comprobación del funcionamiento del conjunto.

- Diseño modular del proyecto, con arquitectura abierta tanto en software como en hardware, que permita modificaciones y ampliaciones favoreciendo la experimentación y el aprendizaje. Para esto, todo el código deberá estar comentado y existirá una guía documentada para posteriores desarrollos.

Desde el punto de vista del aprendizaje personal, la realización de este trabajo de fin de grado permitirá adquirir conocimientos sobre desarrollo de aplicaciones móviles con la tecnología Android, cubriendo así un área que no tuve oportunidad de desarrollar durante el grado. Además, ampliaré mi conocimiento sobre microcontroladores, la forma de comunicarse con diferentes equipos y el desarrollo de ambos. También es interesante, la experiencia de colaboración con un equipo de investigación, permitiendo tener un primer contacto en este ámbito.

## 1.4 Estructura del documento

El presente documento está estructurado en 9 capítulos en los que se abordan los distintos aspectos contemplados para la elaboración de este trabajo de fin de grado.

El capítulo 1 está dedicado a la introducción del proyecto, declarando los motivos que han llevado a realizar este TFG y los objetivos marcados a alcanzar con su realización.

El capítulo 2 muestra una breve introducción a la tecnología Arduino, se explica la placa Arduino y el dispositivo Bluetooth elegidos y el porqué de la elección de este microcontrolador y placa para llevar a cabo el proyecto.

El Capítulo 3 muestra la plataforma eHealth usada en el diseño del proyecto como son los sensores utilizados, la shield que controla estos

sensores y las librerías necesarias a incluir en el código Arduino para una satisfactoria recogida de la información de estos sensores.

En el capítulo 4 se procede a la explicación del desarrollo del código Arduino tanto para recoger los datos de los sensores como para enviarlos mediante comunicación serial, explicando ésta y el código de la interfaz en Matlab para la monitorización de las variables mediante gráficas y valores.

Por otro lado, el capítulo 5 se encarga de mostrar las pequeñas modificaciones y librerías añadidas en el código Arduino para poder enviar mediante Bluetooth estas variables recogidas. Así mismo, se explicará el protocolo Bluetooth seguido para la búsqueda de dispositivos Bluetooth, la conexión entre ambos y el envío y recepción de datos. También se comentarán las partes fundamentales del código de la aplicación móvil para la monitorización.

En el capítulo 6 se realiza una verificación de los resultados obtenidos en las aplicaciones.

Los capítulos 7 y 8 recogen las principales conclusiones derivadas de este trabajo fin de grado, así como las posibles líneas abiertas de investigación que se pudieran desarrollar en un futuro. El capítulo 7 está desarrollado en español, mientras que el 8 lo está en inglés.

Finalmente, el capítulo 9 recoge el desglose del presupuesto total del proyecto.

# Capítulo 2.
# Arduino

Arduino es una plataforma hardware de código abierto y licencia libre, basada en una placa con un microcontrolador y un entorno de desarrollo que implementa un lenguaje de programación Processing / Wiring. Es fácil de utilizar y ha sido diseñado para adaptarse a diferentes necesidades de todo tipo de público, tanto aficionados, desarrolladores como expertos en robótica. Por lo que se puede definir como una herramienta de creación de entornos interactivos destinados a proyectos multidisciplinares y multitecnología, como es el caso en este proyecto.

Los microcontroladores de las placas Arduino siguen la arquitectura Atmel AVR y desde 2012 empezó a usar también microcontroladores ARM de 32 bits (CortexM3). Los más usados son Atmega168, Atmega328, Atmega1280 y Atmega8 dado que son chips sencillos y de bajo coste con los que se puede crear cualquier tipo de diseño.

Gracias a sus pines se puede interactuar con casi infinita variedad de hardware externos como sensores (temperatura, pulsioxímetro, presión, etc), pantallas y diversos periféricos.

Respecto al entorno de desarrollo IDE (integratedDevelopmentEnvironment), es gratuito y fácil de adquirir, en la misma página web de Arduino [1] se puede descargar e instalar en cualquier ordenador (Windows, MAC OS X o GNU/Linux). Como se nombró con anterioridad, este entorno, junto con el framework de programación libre para microcontroladores Wiring, implementa el lenguaje de programación de alto nivel Processing, de código abierto y fácil uso, el cual permite heredar todas las funcionalidades de Java, convirtiéndose en una poderosa herramienta que permite realizar diversos proyectos y con bastante grado de complejidad.

Por otra parte, también es posible comunicar Arduino con infinidad de lenguajes de programación como PHP, Matlab, Python, Ruby, Perl, Java, etc, gracias a la transmisión serial que soportan estos lenguajes.

Uno de los motivos por los que está tan extendido el uso de Arduino es la posibilidad de ejecutar los programas en la placa sin necesidad de conectarlo a un ordenador, basta con conectarle una fuente de alimentación externa y podremos usarlo con total independencia a cables de conexión a PC mientras se haya programado previamente y su memoria almacene algún programa. La regulación de tensión necesaria la lleva a cabo el propio Arduino en la placa.

## 2.1 Arduino UNO

En el proyecto se utilizará la placa Arduino UNO-R3, basada en el microcontrolador Atmega328, la cual es la evolución de la placa Arduino Duemilanove, pues ya no usa el driver FTDI USB-to-serial sino que incorpora un nuevo chip (ATMega16U2) convertidor de USB a serie y tiene nuevo diseño que facilita la identificación de las entradas y salidas.

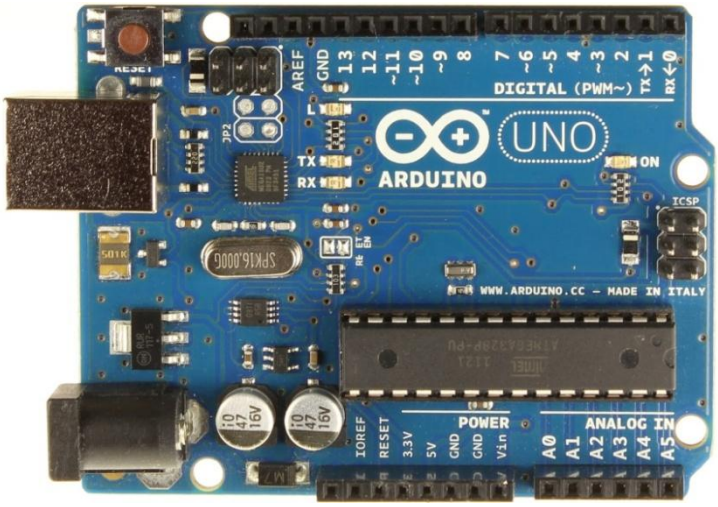



Figura 1. Placa de desarrollo Arduino UNO.

Las especificaciones de la placa Arduino UNO posee las siguientes características:

| | |
|---|---|
| Microcontrolador | ATmega328 |
| Voltaje Operativo | 5V |
| Voltaje Entrada (recomendado) | 7-12V |
| Voltaje Entrada (límite) | 6-20V |
| Pines E/S digitales | 14 (6 proveen PWM) |
| Pines Entrada analógica | 6 |
| Corriente por pin E/S | 40 mA |
| Corriente pin 3.3V | 50 mA |
| Memoria Flash | 32 KB |
| SRAM | 2 KB |
| EEPROM | 1 KB |
| Velocidad reloj | 16 MHz |
| Peso | 25 g |

| | |
|---|---|
| Dimensiones | 68.6 mm x 53.4 mm |

Tabla 1. Características de Arduino UNO.

Por otro lado, la comunicación se realiza vía serie con un ordenador, otro Arduino u otros microcontroladores con los pines 0(Rx) y 1(Tx) de la placa, que dispone de UART TTL (5V) de comunicación serie.

El Arduino UNO ha sido el modelo escogido para desarrollar el proyecto debido a su relación calidad-precio, dado que ofrece muy buenas prestaciones y su precio aproximado ronda los 20 €.

## 2.2 Dispositivo Bluetooth

Para realizar la comunicación Bluetooth entre el dispositivo Android y Arduino se ha elegido el modulo HC-06 [3], principalmente porque este dispositivo cumple perfectamente con los principios de bajo coste, ligereza y usabilidad en los que se basa este TFG. Este módulo bluetooth es de precio reducido, trabaja mediante conexión serie con Arduino y posee buenas características de transmisión y recepción con un alcance amplio. Es de pequeño tamaño, peso ligero, de bajo consumo de corriente y necesita tensión de alimentación de entre 3,6 y 5 V, por lo que es ideal para conectarlo a la fuente de alimentación de 5 V que provee Arduino.


Figura 2. Conector Bluetooth HC-06.

| | |
|---|---|
| Conector Bluetooth | HC-06 |
| Número de pines | 4 |
| Tamaño | 4.4 cm x 1.6 cm x 0.7 cm |

| | |
|---|---|
| Peso | 5 g |
| Voltaje de alimentación | 3,3 VDC - 6 VDC |
| Corriente de operación | < 40 mA |
| Corriente en modo sleep | < 1 mA |
| Ancho de banda ajustable | 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200. |
| Precio | 6 € |

Tabla 2. Características Conector Bluetooth HC-06.

Como se puede observar en la figura 2, dispone tan solo de 4 pines, por un lado VCC que irá conectado a los 5V del Arduino y GND que irá al pin GND de Arduino y por otro lado, el pin Tx y Rx encargados de la comunicación serial.

Hay dos formas distintas de conectar estos dos pines a la placa Arduino para conseguir una comunicación Bluetooth. La primera sería conectar el pin Tx del dispositivo HC-06 al pin Rx de Arduino y el pin Rx del dispositivo HC-06 al pin Tx de Arduino, de esta forma conseguimos una comunicación serial. Así mismo, la otra forma sería conectar los pines Rx y Tx del conector HC-06 a dos pines cualquiera de Arduino y usar la librería “Software Serial” que provee Arduino y que permite que cualquier par de pines puedan funcionar como un puerto serie.

La manera de conexión elegida ha sido la segunda debido a los inconvenientes que ofrece la conexión inversa de pines Rx y Tx con los del Arduino, que eliminan la posibilidad de utilizar el monitor del IDE de éste para depurar el programa. Además, para poder programar el Arduino mediante el sketch, el dispositivo Bluetooth no puede estar conectado a éstos, dado que daría errores de sincronización debido a que se necesitan para la conexión USB con el PC.

# Capítulo 3.

# E-Health Sensor Platform

En este capítulo se explica la plataforma sensorial e-Health, diseñada por CookingHacks para ayudar a investigadores, desarrolladores y artistas para medir datos de sensores biométricos con fines de experimentación, diversión y prueba. Además, es una alternativa de bajo coste en comparación con la venta de estos sensores en el mercado médico.

Figura 3. E-health Sensor Platform.

Como se puede ver en la figura 3, los distintos sensores irán conectados a una shield (e-Health shield) que puede integrarse tanto en Arduino como en Raspberry Pi.

Para este proyecto, se ha decidido la integración en Arduino por dos razones, en principio por cuestiones de comodidad y experiencia de uso y porque para la integración en Raspberry Pi es necesario un módulo que haga de puente entre ésta y la e-Health shield, aumentando el presupuesto global del proyecto. No obstante, la elección entre una u otra tecnología no conlleva pérdida de rendimiento ni eficiencia dentro de este proyecto.

## 3.1 E-Health Shield

E-Health Shield es el principal motor de esta plataforma dado que es la encargada de recoger la información provista por los distintos sensores gracias a la cantidad de pines específicos con los que cuenta.

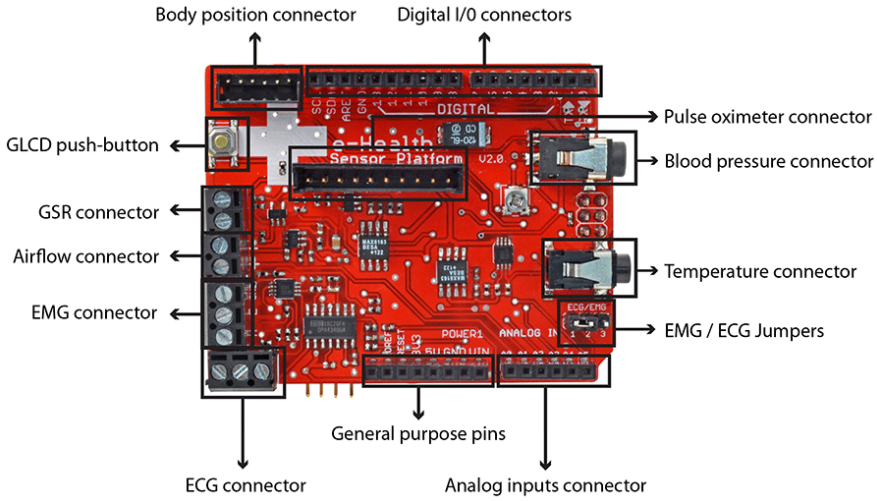



Figura 4. Parte delantera de la e-HealthShield.

Puede ser alimentada a través del Arduino por una fuente de alimentación externa o por el puerto USB del ordenador. No obstante, es posible que al tener conectados todos los sensores a la vez, tenga problemas a la hora de trabajar, por lo que sería conveniente alimentar el Arduino con una fuente de alimentación externa.

## 3.2 Sensores

Con respecto a los sensores, tan sólo se utilizarán cinco de ellos, dado que los demás no harán falta para este proyecto.

### 3.2.1 Pulso y oxígeno en Sangre

El pulso y el oxígeno en sangre componen la pulsioximetría, la cual es un método no invasivo que determina la cantidad (medida en porcentaje) de la saturación de oxígeno u oxígeno disuelto en la hemoglobina de la sangre dentro del sistema circulatorio gracias a métodos fotoeléctricos emitidos con el pulsioxímetro.

Figura 5. Pulsioxímetro.

El pulsioxímetro se coloca en zonas con buen flujo sanguíneo como pueden ser los dedos de la mano, del pie o incluso en el lóbulo de la oreja. Este emite dos longitudes de onda de luz roja (660 nm) e infrarroja (920 nm) que serán recogidas por un fotodetector diferenciando la hemoglobina oxigenada (oxihemoglobina) de la desoxigenada (desoxihemoglobina).

De este modo, este sensor puede ser útil en muchísimos casos en los que el paciente esté inestable, como puede ser en situaciones de emergencia, en operaciones, situaciones de nerviosismo o incluso en aviones sin presión.

Las medidas de oxígeno en sangre para un adulto sano están entre 95 y 99 %, las personas con algún problema hipóxico entre el 88 y 94 % y los valores superiores a 100 % pueden indicar envenenamiento por monóxido de carbono.

Así mismo, el ritmo cardíaco en un adulto en estado de reposo oscila entre 60 y 100 latidos por minuto. Por debajo de 60 puede ocultar una bradicardia, y por encima de 100 puede ser indicio de taquicardia.

Para su utilización, tan sólo se debe conectar el sensor a la shield, de la manera que se observa en la figura 6, insertar su dedo en el sensor y empezará a ver los valores en la pantalla del sensor.

Figura 6. Correcta conexión y utilización del pulsioxímetro.

### 3.2.2 Temperatura corporal.

La temperatura corporal permite evaluar la eficiencia de la regulación térmica o grado de calor de un organismo y es de vital importancia medirla dado que la mayoría de enfermedades están asociadas a un cambio en la temperatura.

Es distinta dependiendo de la zona del cuerpo en la que se mida, dado que no todas las zonas cuentan con la misma temperatura, sin embargo, sea cual sea la zona, la temperatura promedio debe encontrarse entre los 36,5 y 37,5 ºC.

| | |
|---|---|
| Hipotermia | <35,0 ° C (95,0 ° F) |

| | |
|---|---|
| Normal | 36,5 a 37,5 ° C (97,7-99,5 ° F) |
| Fiebre o hipertermia | > 37,5 a 38,3 ° C (99,5 a 100,9 ° F) |
| Hiperpirexia | > 40,0 a 41,5 ° C (104 a 106,7 ° F) |

Tabla 3. Rangos de temperatura en pacientes.

También depende de las condiciones de temperatura ambiental y de la intensidad de la actividad física realizada. Existen dos tipos de temperatura, una es la temperatura central (cerebro, sangre, vísceras, grandes vasos) que se mantiene siempre constante, y la temperatura periférica, que es la que se encuentra en la piel, extremidades, músculos, etc y varía dependiendo de diversos factores como el género de la persona, la hora del día, el consumo de alimentos, etc.

En la siguiente figura se puede ver el sensor utilizado en el proyecto, el cual, al igual que los demás, irá conectado y transmitirá información a la shield. La conexión se realizará como se puede observar en la misma. Se debería sujetar el sensor en la parte del cuerpo donde se quiera poner con cinta adhesiva. El contacto debería ser la parte metálica del sensor con la piel.

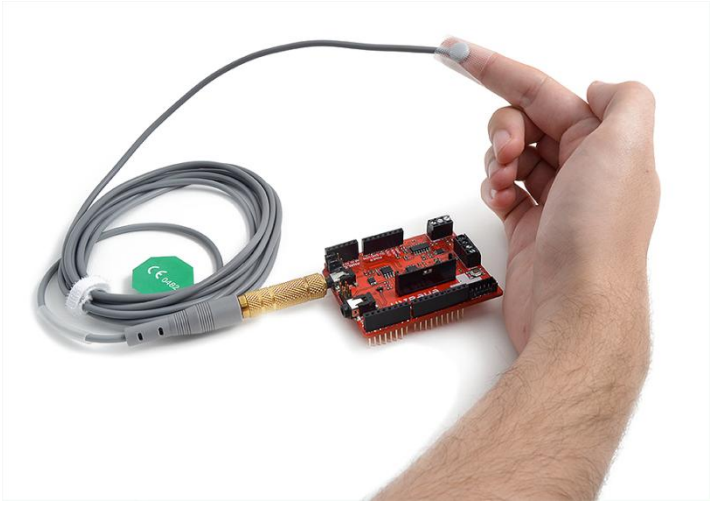


Figura 7. Conexión del sensor con la shield.

### 3.2.3 Respuesta galvánica de la piel.

El sensor galvánico se encarga de medir la conductancia eléctrica de la piel o también conocida como la respuesta galvánica de la piel, que varía con su nivel de humedad.

Figura 8. Sensor galvánico.

La conductancia de la piel es una indicación de la excitación psicológica o fisiológica como el miedo, la ira o los sentimientos; estos cambios en la conductancia de la piel dependen de ciertos tipos de glándulas sudoríparas que son abundantes en los dedos y en las manos. Estas glándulas sudoríparas son controladas por el sistema nervioso, por lo tanto, si la rama simpática del sistema nervioso autónomo está muy excitada, la actividad de las glándulas del sudor también aumentará, incrementando así la conductancia de la piel. De esta manera, la conductancia de la piel se puede utilizar como una medida de las respuestas emocionales.

La resistencia galvánica de la piel se refiere a la resistencia eléctrica entre dos electrodos grabada cuando una corriente muy débil se hace pasar de manera constante entre ellos. Los electrodos se colocan normalmente sobre una pulgada de distancia, y la resistencia registrada varía de acuerdo con el estado emocional del sujeto. Esto se lleva a cabo mediante la conexión de los electrodos a un amplificador de tensión. Del mismo modo, esta tensión varía con el estado emocional del sujeto.

La medición de la conductancia de la piel es uno de los componentes de los dispositivos de polígrafo y se utiliza en la investigación científica de excitación emocional o fisiológica.

El sensor irá conectado a la shield como muestra la siguiente figura. Como se puede observar, tiene dos contactos y funciona como un ohmímetro midiendo la resistencia entre dos puntos, así que se deben colocar cada uno de los dos dedos en los contactos metálicos y se deben sujetar con los velcros que vienen incorporados.

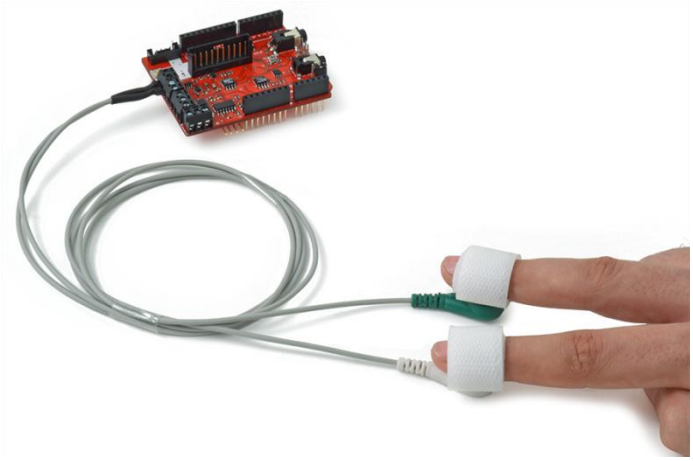


Figura 9. Conexión del sensor galvánico a la shield y acople a los dedos.

### 3.2.4 Electrocardiograma.

Un electrocardiograma (ECG) es una prueba física ampliamente utilizada para valorar la condición del corazón en forma no invasiva. Dicha prueba se usa para evaluar el estado del sistema de conducción del corazón, el del músculo, y también, de forma indirecta, la condición de este órgano como bomba y la aparición de ritmos patológicos causados por daño al tejido de conducción de las señales eléctricas u otros trastornos no-cardíacos.

Figura 10. Sensor medidor del electrocardiograma (ECG).

Con este sensor se podría medir o detectar la orientación del corazón en la cavidad torácica, la hipertrofia, daños en las diversas partes del músculo del corazón, evidencias de deterioro del flujo sanguíneo de forma aguda al músculo del corazón, patrones de actividad eléctrica anormal que pueden predisponer al paciente a alteraciones del ritmo cardíaco anormal o simplemente el mecanismo de ritmo del corazón.

El sensor irá conectado a la shield como muestra la siguiente figura. Hay que tener en cuenta que cada uno de los sensores o parches tiene un polo,

por lo que deberán ir en el lugar correcto que indiquen las serigrafías existentes en la shield.


Figura 11. Conexión del sensor medidor del ECG a la shield.

De tal manera, cada uno de los parches tendrá un apósito que se retirará y se colocará en diversas zonas del cuerpo dependiendo de la polaridad del parche. El parche con el receptor rojo (positivo) irá en el pectoral derecho, el que tiene el receptor negro (negativo) irá en la parte izquierda del abdomen y el blanco (neutro) en el pectoral izquierdo.

## 3.3 Librerías

En este caso, tenemos la ventaja de que la *e-Health Sensor Platform* ya viene preparada con una librería específica escrita en lenguaje C++, la cual permite la lectura y transmisión de la información de los sensores. Es de software libre y bastante fácil de entender y usar.

Se puede descargar desde la página principal de *CookingHacks*, en la sección de *e-Health Sensor Platform*, mediante un fichero .zip, en el que se incluye tanto la librería de control de los sensores *"eHealth"* como la librería *"PinChageInt"* que permite lanzar una interrupción cuando cambia de estado de los pines. Esta última librería tan sólo es necesaria si se va a utilizar el pulsioxímetro dado que necesita de interrupciones.

# Capítulo 4. Monitorización de variables en PC

En este apartado se explicará el desarrollo llevado a cabo tanto en la plataforma Arduino como en el entorno Matlab para la recogida de mediciones de variables en Arduino, envío de éstas mediante comunicación serial con el PC y posterior monitorización en una interfaz realizada con Matlab.

## 4.1 Desarrollo en Arduino

En este TFG, el microcontrolador Arduino será el responsable tanto de empaquetar las mediciones de las variables recogidas por la *shield* gracias a los sensores y enviarlas mediante una comunicación serial.

En principio, y una vez conectados los sensores a la *shield* como se describió en la sección 3.2, se recogerán los datos gracias a las diversas funciones que provee la librería *eHealth*. Se utilizarán tan sólo las correspondientes a los sensores que se usen en la plataforma.

```
#include <eHealth.h>

eHealth.getBPM(); //Ritmo cardiaco
eHealth.getOxygenSaturation(); //Cantidad de oxígeno en sangre
eHealth.getTemperature(); //Temperatura
eHealth.getSkinConductance(); //Conductancia de la piel
eHealth.getECG(); //Electrocardiograma
```

Figura 12. Funciones que provee la librería eHealth para recoger los datos de los sensores.

Una vez se recogen los diferentes datos de los sensores, los cuales corresponden a las variables fisiológicas del paciente, se envían por el puerto serial. No obstante, para no enviarlas constantemente sin un periodo establecido, se configuran una serie de interrupciones. Al principio se había pensado hacerlo de la forma fácil, es decir, con la función *delay* a la que ya estábamos acostumbrados. Sin embargo, el problema de esta función es que congela al Arduino el tiempo especificado y ocasionaría conflictos, dado que no se podrían recibir los datos de los sensores mientras el Arduino estuviera parado. Por ello, se descartó esta opción y en su defecto se usó una opción alternativa, los *Timers* o interrupciones programadas. Estos timers hacen lo mismo que las interrupciones hardware, pero en lugar de dispararse cuando

se cumple un cierto proceso hardware en uno de los pines, se dispara cuando ha transcurrido un tiempo específico, previamente programado.

Para esto se usó la librería *Timer* de Arduino, la cual permite configurar diferentes timers a diferentes periodos de tiempo.

```
#include <Timer.h>

Timer t;
Timer t2;

void setup() {
  t.every(2500, funcion1);
  t2.every(250, funcion2);
}

void loop() {
  t.update();
  t2.update();
}

void funcion1(){}
void funcion2(){}
```

Figura 13. Configuración de timers en Arduino.

Como se puede observar en la figura 13, a cada timer se le indica mediante la función *every* un periodo en milisegundos y una función que se ejecutará cada vez que se cumpla ese tiempo, asimismo, en el bucle principal de Arduino hay que actualizar los timers, es decir, configurarlos de nuevo para poder seguir trabajando con ellos.

Las variables se enviarán por el puerto serial cada 2500 ms, sin embargo, como en Matlab se va a graficar el electrocardiograma, se necesitará un muestreo mayor dentro de esos 2,5 segundos que permita construir la gráfica con más detalle. Es por esto que se ha decidido configurar otro timer, con periodo de 250 ms con el que se ejecute la función de recoger los datos del sensor del electrocardiograma (ECG) y se vayan metiendo en una cola, que se enviará una vez se ejecute el primer *timer* de enviar variables.

Crear la cola es tan sencillo como en cualquier otro lenguaje, simplemente es añadir al *Sketch* la librería *QueueList* de Arduino y declararla, indicándole el tipo de valores que se van a almacenar.

Por tanto, los datos se almacenarán en un paquete que contará con una cabecera específica que sirva para identificar el inicio del paquete, así mismo, la cabecera irá seguida de los datos de los sensores separados por un carácter separador para que luego en la recepción sea más sencillo separarlos utilizando la función *split.* Para indicar el fin de un paquete le añadiremos al final un carácter de retorno de carro o “\n”.

| | | | | | | |
|---|---|---|---|---|---|---|
| Cabecera | BPM | SPO2 | Temperatura | Conductancia | Cola de datos del ECG | Carácter de fin |

Figura 14. Modelo de paquete a enviar por Arduino.

## 4.2 Programación orientada a objetos en Matlab

Con vistas a integrar el resultado de este trabajo de fin de grado en el software general y para respetar los requisitos de organización y modularidad se ha seguido el modelo de programación orientada a objetos con el lenguaje de programación Matlab.

En un principio, dado que Matlab no incorporó la orientación a objetos hasta la versión 2010, el proyecto tiene partes de código que sigue con el modelo de programación clásico de Matlab. No obstante, se ha optado por modernizar de alguna manera el código y adaptarlo a versiones más recientes, pasándolo a un modelo orientado a objetos en el que poder trabajar fácilmente con clases y con sus atributos mediante métodos *getter* y *setter*, puesto que el modelo clásico resulta un poco rudimentario y debería poseer bastante lógica para acceder a estos datos.

Toda clase en Matlab está formada por dos tipos de elementos:

Atributos: son las variables que almacenan el estado de un objeto particular de esa clase. Tan sólo se puede acceder a ellas desde un método de la propia clase.

Métodos: son las únicas funciones que están autorizadas a acceder directamente a los atributos de un objeto. Se construyen como funciones comunes de Matlab, siendo el primer parámetro a recibir un objeto de la clase.

Para la implementación de cada clase, se creará un directorio cuyo nombre debe coincidir con el de ésta, anteponiendo el carácter “@” como prefijo. Dentro de éste, se colocarán todos los ficheros de Matlab (.m) con los métodos. Adicionalmente, y al igual que en la mayoría de lenguajes de programación, el constructor se define como una función con el mismo nombre de la clase, y en él se especificarán los campos que conformarán los atributos del objeto.

## 4.3 Problema de soporte de la programación multihilo

Uno de los problemas fundamentales en Matlab, y en este proyecto en concreto, ha sido la inexistencia de programación multihilo en este entorno y lenguaje de desarrollo. Pues para la ejecución y desarrollo de este proyecto, existe una alta necesidad de utilizar hilos tanto para leer del puerto serial las

variables como para graficarlas y mostrarlas en los campos de texto correspondientes.

En su defecto, Matlab posee Timers que, al igual que los usados anteriormente en el capítulo de desarrollo en Arduino, se programan con un cierto periodo y con la función que se tiene que invocar cada vez que se cumple este tiempo. No obstante, la limitación que existe es que los timers son bloqueantes, es decir, mientras se está ejecutando la función que ha sido llamada por esa interrupción, el resto del sistema se encuentra bloqueado hasta que termine.

No obstante, los timer de Matlab no se caracterizan por una precisión rigurosa, por lo que su uso podría comprometer el funcionamiento general del programa. Además, en vistas a poder integrar posteriormente nuestro programa en el software original, podría acarrear problemas con otros timer, impidiendo ejecuciones simultáneas.

Es por esto, que estas funciones deberían de ser lo bastante simples y rápidas como para no bloquear la aplicación durante bastante tiempo.

```
interfaz.pintarEcgTimer = timer ('Name', 'timerPintarECG',...
            'Period', 1,...
            'ExecutionMode', 'fixedSpacing',...
            'BusyMode', 'drop');


interfaz.pintarVariablesTimer = timer ('Name', 'timerPintarVariables',...
            'Period', 3,...
            'ExecutionMode', 'fixedSpacing',...
            'BusyMode', 'drop',...
            'TimerFcn', @(o, e) pintarVariables(eHealth));

start(interfaz.variablesTimer);
```

Figura 15. Extracto de código de interfaz.m que muestra la declaración y uso de timers.

## 4.4 Desarrollo de la aplicación en el PC

Basándonos en los conceptos introducidos en el apartado 4.2, se implementarán cuatro clases que nos permitirán almacenar los datos del paciente, el puerto serial, el estado de éste, las variables leídas y todo lo que compone la interfaz como los timers, campos de texto o botones.

Las clases en Matlab suelen heredar de la clase “*handle*” para que cada vez que se use la clase no se cree una nueva instancia de ésta sino que se pueda modificar. Esto viene a raíz de solucionar el problema de la inmutabilidad de clases en Matlab con el cual no se podían modificar los

parámetros de una función sino que se hacía de manera local, sin embargo, no permanecían tras la ejecución de la función. Por lo que, la solución a esto era devolver como resultado de la función la clase modificada.

- @Interfaz: Clase encargada de ejecutar la aplicación con todos sus componentes gráficos, en ella se encuentran los timers, los botones, los campos de texto, el diálogo principal y la gráfica.

- @Variables: Clase encargada de almacenar las variables leídas para su posterior utilización en cualquier parte del programa.

- @PuertoSerial: Clase encargada de definir el objeto serial tanto como los valores para la comunicación entre el ordenador y el Arduino como los baudios, número de bits de datos, bits de parada, paridad, velocidad de transmisión, control de flujo y el estado del puerto.

- @DatosPaciente: Clase encargada de recoger los datos del paciente a monitorizar para su posterior utilización en cualquier parte del programa.

Además de estas cuatro clases, se crean también una serie de funciones que permiten llevar a cabo la implementación de este proyecto. En los próximos epígrafes se procede al análisis de la implementación.

### 4.4.1 Diseño de la interfaz de usuario

Con vistas a dotar al usuario final de una forma de acceso a la funcionalidad implementada, se opta por el diseño de una interfaz gráfica. La base de diseño de la misma estará orientada a generar una interfaz sencilla que facilite su uso de manera intuitiva. No obstante, no se empleará demasiado tiempo en la parte gráfica de la interfaz dado que el proyecto general ya cuenta con una y tan sólo tendríamos que añadirle pequeñas partes de la nuestra.

Para el diseño se podría emplear la herramienta Guide proporcionada por Matlab, sin embargo, se ha optado por un diseño de la interfaz a través de programación de código.

Como en todo proyecto, se creó principalmente un mockup que reflejara cómo debía ser la interfaz de la aplicación:

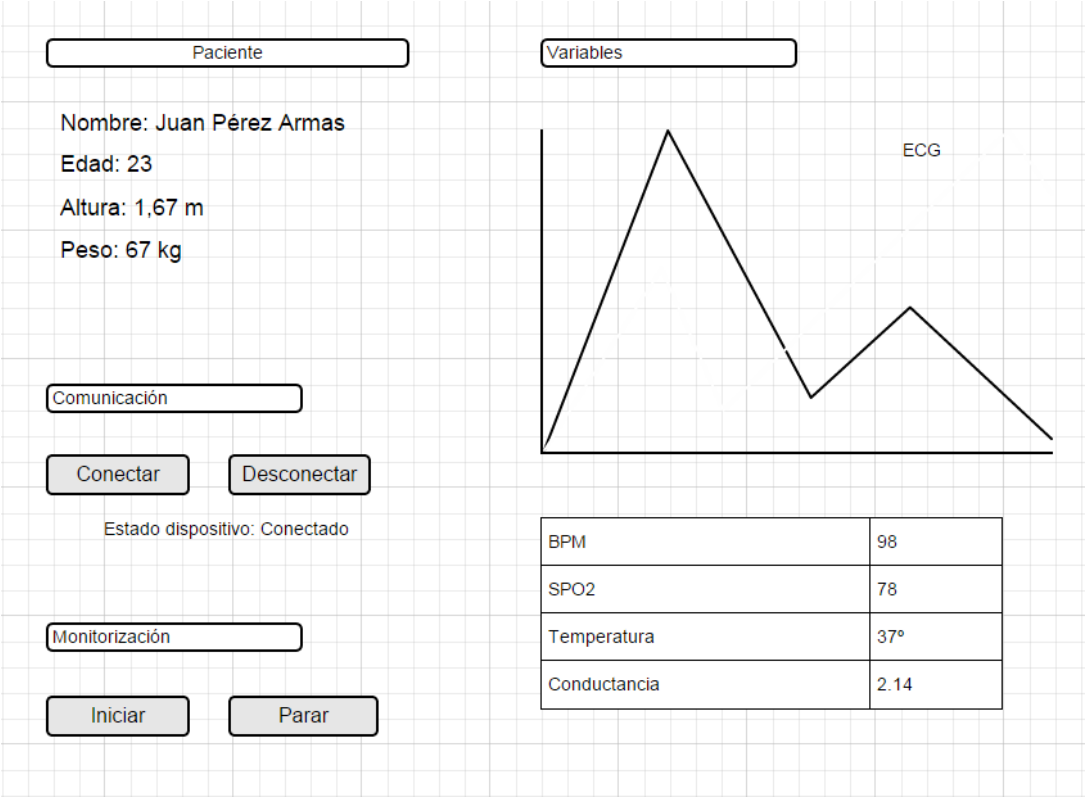



Figura 16. Mockup de la interfaz gráfica de la aplicación.

El diseño se dividirá en tres zonas:

Zona de datos del paciente: en esta zona se recoge la información relativa a los datos del paciente a monitorizar (nombre, edad, peso, sexo y altura).

Zona destinada a la comunicación con el dispositivo Arduino: en esta zona se podrá establecer la conexión con el dispositivo y dar la orden de comienzo del proceso.

Zona de visualización de variables: zona en la que se representarán las variables, el ECG mediante una gráfica y las demás variables mediante valores dentro de campos de texto.

El aspecto final es el recogido en la figura 39.

### 4.4.2 Datos del paciente

Para registrar los datos del paciente al que vamos a monitorizar se ha implementado un diálogo emergente, como el que vemos en la figura 17, que solicite los datos al usuario una vez se inicia la aplicación.

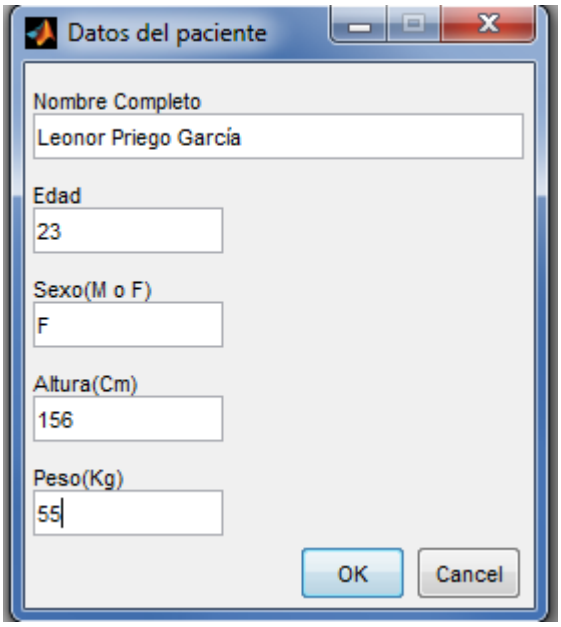



Figura 17. Diálogo emergente que recoge los datos del paciente a monitorizar.

Estos datos se almacenan en variables que posteriormente se volcarán en un fichero destinado a almacenar los datos recogidos en el proceso de monitorización. La cabecera de cada fichero incluirá los datos del paciente.

```
%%FICHERO
nombreFichero = strcat(interfaz.nombre, '.txt');
interfaz.fichero=fopen(nombreFichero,'w');
fprintf(interfaz.fichero,'Nombre:  %s \t', datosPaciente.nombre);
fprintf(interfaz.fichero,'Edad:  %d \t', datosPaciente.edad);
fprintf(interfaz.fichero,'Sexo:  %s \t', datosPaciente.sexo);
fprintf(interfaz.fichero,'Altura:  %d \t', datosPaciente.altura);
fprintf(interfaz.fichero,'Peso:  %d \t', datosPaciente.peso);
```

Figura 18. Extracto de código de interfaz.m para la creación de los ficheros de texto.

### 4.4.3 Comunicación con Arduino

Antes del comienzo del proceso, es necesario que el usuario conecte el dispositivo con el ordenador. En caso de no llevarse a cabo, aparecerá un mensaje de error que alertará de esta situación.

Para poder establecer la conexión con el dispositivo Arduino hay que pulsar el botón “Conectar”, el cual estará controlado mediante un *callback* que llamará a la función conectar.m que será la encargada de leer los datos del puerto y establecer la conexión. Una vez se haya establecido la conexión, la etiqueta de estado de conexión pasará a mostrar “Dispositivo Conectado”.

```
function conectar_pushbutton_callback(hObject,~,puertoSerial)
    conectar(puertoSerial);
    if(puertoSerial.conectado == 1)
       set (puertoSerial.estadoDispcsitivo, 'String', 'Dispositivo Conectado');
    end
end
```

Figura 19. Extracto de código de la función conectar_pushbutton_callback.m para la conexión del dispositivo.

Del mismo modo, para desconectar hay que pulsar el botón “Desconectar” y el *callback* asociado llamará a la función desconectar.m, que desconectará el puerto, pondrá su estado a desconectado y cerrará el fichero que recoja los datos de la monitorización.

Una vez el monitor esté conectado se podrá iniciar el proceso de monitorización pulsando el botón “Inicio”, que mediante el *callback* asociado iniciará los *timers* correspondientes para leer del puerto y representar tanto en la gráfica como en los campos de texto las variables correspondientes.

```
function inicio_pushbutton_callback(hObject,~,interfaz, variables)
    start(interfaz.escaneoTimer);
    start(interfaz.variablesTimer);
    set(interfaz.mostrarEcgTimer, 'TimerFcn', @(o, e) pintaECG(variables.ECG,p));
    start(interfaz.mostrarEcgTimer);
```

Figura 20. Extracto de código de la función inicio_pushbutton_callback.m para el inicio de la monitorización.

En el caso de querer parar el proceso de monitorización, tan sólo habrá que pulsar el botón “Parar”, cuyo *callback* será el correspondiente de parar los *timers*.

#### 4.4.3.1 Configuración de la comunicación serial

La configuración del puerto serie tanto para la transmisión como para la recepción viene descrita en la siguiente tabla:

| | |
|---|---|
| Velocidad de transmisión | 9600 |
| Bits de datos | 8 |
| Bits de parada | 1 |
| Paridad | Sin paridad |

| | |
|---|---|
| Control de flujo | No |

Tabla 4. Configuración del puerto serie.

### 4.4.4 Lectura y representación de variables

Una vez se ha establecido la conexión entre dispositivos y se ha iniciado la transmisión de datos, se lee del puerto serial mediante la función *fscanf* que nos permite Matlab, a la cual tan sólo hay que pasarle por parámetro el puerto serial del que va a leer y el tipo de dato en el que queremos que nos los almacene. En este caso, guardaremos el paquete como un *string* y llamaremos a la función descomponer.m que separará el paquete en campos mediante el método *split* y el carácter de separación que hayamos indicado a la hora de construir el paquete en el código Arduino. Una vez los haya separado se almacenan como variables de la clase @Variables.m para su posterior representación.

```
function descomponer(datos,variables)
      datos = regexp(datos, '\*', 'split'); %separar datos en campos

      if(strcmp(datos(1),'Cabecera'))
          variables.pulso = datos(2);
          variables.oxigeno = datos(3);
          variables.temperatura = datos(4);
          variables.conductancia = datos(5);

          %Lectura de la cola de valores del ECG
          i=6;j=1;
          while (i < numel(datos))
              variables.ECG(j) = str2double(datos(i));
              j = j+1;
              i=i+1;
          end
      end
end
```

Figura 21. Extracto de código de la función descomponer.m que descompone el paquete en campos.

El timer para la lectura del puerto serial se ha establecido en 3 segundos, para poder leer a medida que el Arduino envíe los datos cada 2.5 segundos. De tal manera, el timer establecido para representar las variables será cada 3,5 segundos. Estos timers con no muy poco periodo de interrupción son ideales para no influir ni estorbar a otros timers en la ejecución del resto del programa, además, son suficientes para la aplicación de procesos de control como para la visualización.

#### 4.4.4.1 Representación de variables en campos de texto

Como se acaba de comentar, la función encargada de representar las variables en campos de texto “mostrarVariables.m” es llamada por un timer que se ejecuta cada 3,5 segundos, tras almacenar los datos leídos en variables. Como argumentos de la función se le pasan la clase Variables.m, los *textEdit* creados en la interfaz que recogerán los valores a modificar y el fichero en el que se almacenarán los datos recogidos, que en este caso lo hemos englobado en la clase Interfaz.m.

```
function mostrarVariables(variables, interfaz)
    set (interfaz.variablesLista(1), 'String', variables.pulso);
    set (interfaz.variablesLista(2), 'String', variables.oxigeno);
    set (interfaz.variablesLista(3), 'String', variables.temperatura);
    set (interfaz.variablesLista(4), 'String', variables.conductancia);
```

Figura 22. Extracto de código de la función mostrarVariables.m que permite mostrar las variables en los campos de texto.

En la figura 22 se puede observar como la función set permite cambiar el valor de las propiedades de los objetos gráficos de la interfaz. En este caso, tan sólo se mostrarán en campos de texto, las variables pulso, oxígeno, temperatura y conductancia; el electrocardiograma se representará mediante una gráfica, cuyo proceso será explicado en el apartado siguiente.

#### 4.4.4.2 Representación gráfica del Electrocardiograma

Al igual que en el apartado anterior, la función mostrarVariables.m también se encargará de graficar el ECG, sin embargo, la peculiaridad de este caso reside en que hay que crear la gráfica (*plot* como se le reconoce en Matlab) con sus ejes en otra función que tan sólo se invoque una vez y pasársela por parámetros a la función antes de iniciar timer, dado que si se hiciese directamente en mostrarVariables.m, cada tres segundos se crearía una nueva gráfica y no es lo que se busca.

Por lo que creamos la gráfica en el *callback* que controla el botón “Inicio”.

```
positionVector2 = [0.4, 0.35, 0.55, 0.6];    % position of second subplot
a = axes('Position',positionVector2, 'xlimmode', 'manual')
p = plot(0,NaN,'-r', 'parent', a, 'tag', 'grafica')
grid on;

start(eHealth.variablesTimer);
```

Figura 23. Extracto de código de la función inicio_push_button.m que permite crear la gráfica.

Para mostrar los datos del ECG de la cola recibida del Arduino, creamos un algoritmo que recorra el eje x en función de cada una de las muestras del ECG en el tiempo, es decir, como sabemos que cada 3 segundos nos llega una cola con 9 muestras del ECG, hacemos la división aritmética de 3 entre 9 para saber cada cuanto tiempo se representa un valor. Por tanto, cada 0,33 segundos representaríamos una muestra del ECG de la cola.

```
if(numel(interfaz.ejexGrafica)==1)
    interfaz.ejexGrafica(1)=0;
else
    interfaz.ejexGrafica(1)=interfaz.ejexGrafica(end)+(3/9);
end
for i=2:numel(variables.ECG)
    interfaz.ejexGrafica(i)=interfaz.ejexGrafica(i-1)+(3/9);
end

x = get (p, 'xdata');
y = get (p, 'ydata');

p = findobj ('tag', 'grafica')
set(p,'ydata',[y variables.ECG],'xdata',[x interfaz.ejexGrafica]);
get(p,'xdata')
```

Figura 24. Extracto de código de la función mostrarVariables.m que permite representar las muestras del ECG en la gráfica.

En Matlab es bastante difícil representar valores frente al tiempo cuando realmente no se sabe el tiempo en el que llegan, dado que cada tres segundos llegan varias muestras. Por ello ha habido que realizar un algoritmo que lo permita, así mismo, se deberían guardar los valores anteriores de ambos ejes para que la representación no fuese sólo del instante de esos tres segundos, sino de la gráfica completa desde que empezó la monitorización.

En el extracto de código no se han incluido los comentarios por ahorrar espacio en el documento, sin embargo, en el Apéndice A se incluye la dirección de un repositorio en el que se puede consultar el código con sus comentarios.

# Capítulo 5. Monitorización de variables en aplicación móvil

En este apartado se explica el desarrollo llevado a cabo tanto en la plataforma Arduino como en Android para la recogida de mediciones de variables en Arduino, envío de éstas mediante comunicación serial con el dispositivo móvil y posterior monitorización en una aplicación móvil realizada con el lenguaje Android y el entorno Android Studio.

## 5.1 Desarrollo en Arduino

El código desarrollado en Arduino para la monitorización en Matlab como en un dispositivo Android es prácticamente el mismo, pero en este caso se utilizará un dispositivo Bluetooth para comunicar Arduino con el dispositivo móvil. Por ello, es necesario conectar el dispositivo Bluetooth expuesto en el apartado 2.3 de la manera que se muestra en el diagrama de conexión del Apéndice B.

Una vez conectado y configurado, se debe programar el envío de las variables que se desean transmitir.

El hardware de Arduino tiene soporte para las comunicaciones serie en los pines 0 (RX) y 1 (TX), dado que el soporte serie nativo se da mediante un circuito integrado llamado UART que permite al chip Atmega recibir comunicaciones serie incluso mientras trabaja en otras tareas, siempre y cuando haya espacio en el buffer serie de 64 bytes.

No obstante, para permitir la comunicación serie de los pines RX y TX con el PC, se utilizará la librería *SoftwareSerial* de Arduino, la cual ha sido desarrollada para permitir la comunicación serie a través de otros pines digitales de Arduino, utilizando software para replicar la funcionalidad. Como no está soportada por hardware, esta librería tiene algunas limitaciones, como por ejemplo, que sólo funciona a velocidades de hasta 9600 baudios y no permite dos comunicaciones simultáneas si se configuran múltiples puertos software serial.

```
#include <PinChangeInt.h>
#include <SoftwareSerial.h>
#include <eHealth.h>

SoftwareSerial Bluetooth(2, 3); // RX | TX

void setup(){
  Bluetooth.flush();
  delay(500);
  Bluetooth.begin(9600);
}

void loop(){
  if(Bluetooth.isListening()){
    Bluetooth.print(cabecera);
    Bluetooth.print(eHealth.getBPM());
    Bluetooth.print(delimitador);
```

Figura 25. Extracto de código de EnvioVarArduino.ino que permite la comunicación Bluetooth.

En la figura 25 se muestra un extracto de código que permite configurar un segundo puerto serie en otro par de pines cualesquiera (en Arduino Mega, Leonardo y Micro no todos los pines lo permiten, pero en Arduino Uno sí). Asimismo, para configurar la velocidad para la comunicación serial se utiliza la función *begin* y para enviar valores, la función *print*.

La estructura del paquete será exactamente igual que la comentada en el apartado 4.1.

### 5.1.1 Problema de superposición de puertos de interrupciones

Durante el desarrollo de la mejora en el código Arduino para añadirle la comunicación Bluetooth hubo problemas debido a la superposición de pines que permiten interrupciones en las librerías *PinChangeInt* y *SoftwareSerial* dado que ambos usan los mismos puertos para las interrupciones. Es un problema bastante comentado en Internet, pero con muy pocas soluciones, por lo que en un principio se ha optado por una solución fácil. Dado que el módulo Bluetooth HC-06 no necesita recibir ningún tipo de dato, es decir, no necesita leer datos del serial, sino solamente enviar, no hace falta el uso de interrupciones. No obstante, en la librería *PinChangeInt* sí que las necesita para el funcionamiento de nuestro código, por lo que se ha optado por modificar la librería *SoftwareSerial* y comentar las líneas que permiten interrupciones en esos puertos.

También hay que tener en cuenta que la librería *E-Health* utiliza los pines comprendidos entre el 6 y el 13 para el uso del pulsioxímetro, por lo que no se podrán usar para crear el puerto *SoftwareSerial*.

## 5.2 Desarrollo en Android

En un principio se habían barajado diversas plataformas en las que desarrollar la aplicación como Phonegap, Cordova, Android e IOS; no obstante, se ha decidido crear una aplicación nativa para poder sacarle el máximo partido al hardware, en este caso Bluetooth. El desarrollo para IOS para este proyecto no se ha contemplado dado que no se dispone de un ordenador Mac para poder programar la aplicación, por lo que finalmente se decide utilizar el lenguaje Android y el entorno Android Studio.

La aplicación móvil se va a encargar de solicitar conexión mediante tecnología Bluetooth al dispositivo Bluetooth conectado a Arduino; una vez se conecte recibirá los distintos paquetes de muestras recibidas de los sensores y las monitorizará en diferentes campos de texto.

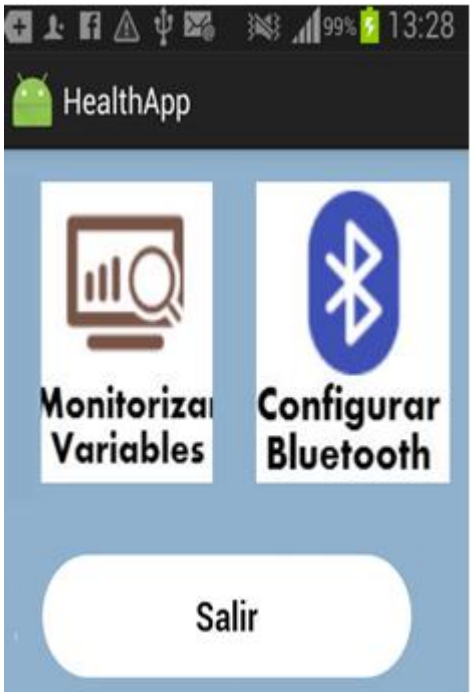



Figura 26. Interfaz principal de la aplicación móvil.

En los siguientes apartados se explicará por temas el desarrollo implementado.

### 5.2.1 Activación y desactivación del Bluetooth

El primer paso para iniciar la aplicación será comprobar si el Bluetooth está activado o no en el dispositivo y, en caso de que no sea así, solicitar al usuario permiso para activarlo.

En primer lugar, para hacer uso del Bluetooth será requisito indispensable añadir los permisos adecuados para ello en el fichero *AndroidManifest.xml*, archivo de configuración situado en la raíz de nuestras aplicaciones donde podremos aplicar las configuraciones gráficas de la aplicación.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Figura 27. Extracto de código de AndroidManifest.xml que permite añadir los permisos para el uso del Bluetooth.

Pese a que la tecnología Bluetooth está ampliamente extendida, es de obligado cumplimiento la comprobación de que el servicio esté disponible en el dispositivo. Para realizar esta operación haremos uso de la clase *android.bluetooth.BluetoothAdapter*. Declararemos una referencia como atributo dentro de nuestra *Activity* y el método *comprobarEstadoBluetooth* se encargará de detectar si el dispositivo puede hacer uso del Bluetooth y si éste se encuentra activo o no.

```
// Obtenemos el adaptador Bluetooth. Si es NULL, significara que el
// dispositivo no posee Bluetooth, por lo que mostramos dialogo de alerta y salimos.
bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if(bluetoothAdapter == null) {
    final AlertDialog.Builder dialog = new AlertDialog.Builder(this);
    dialog.setTitle("Bluetooth no disponible");
```

Figura 28. Extracto de código de la función comprobarEstadoBluetooth que permite detectar si el dispositivo posee Bluetooth.

Tras la comprobación de que sí posee Bluetooth, si está desactivado, se le enviará una petición al usuario para activarlo, pues, además de ser una buena práctica, al usuario generalmente le gusta ser consciente de lo que activa y desactiva en su dispositivo, por lo que es aconsejable pedirle permiso para ello. Esta petición requiere de un *Intent* (que permite invocar componentes o llamar a aplicaciones externas a la nuestra) que solicite el permiso para activarlo y de la lectura de la respuesta (el usuario puede cancelar la activación).

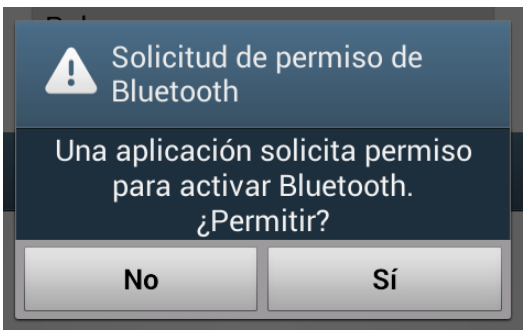



Figura 29. Solicitud de permiso para activar el Bluetooth.

Para recibir una respuesta de una actividad en la actividad actual, ésta se lanzará mediante el método *startActivityForResult*, en el que se especificará el *Intent* que queremos lanzar y un entero que servirá de clave para filtrar la respuesta en el método *onActivityResult*, que será el que se ejecutará al volver de la actividad y contendrá el resultado de la operación.

```
// Lanzamos el Intent que mostrara la interfaz de activacion del
// Bluetooth. La respuesta de este Intent se manejara en el metodo
// onActivityResult
Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
```

Figura 30. Extracto de código que permite lanzar el *Intent* de activación del Bluetooth.

Una vez activado el Bluetooth, se elegirá entre conectarse al dispositivo Bluetooth por defecto en la aplicación (que será el último dispositivo al que se ha conectado) y comenzar a monitorizar las variables, o bien hacer una búsqueda de dispositivos y elegir un dispositivo de la lista de dispositivos encontrados.

### 5.2.2 Descubrir dispositivos Bluetooth

Para que se realice una conexión Bluetooth es necesario configurar uno de los dispositivos como servidor para recibir conexiones entrantes, y otro como cliente que solicite la conexión al dispositivo servidor, y para esto deberá conocer la dirección física del dispositivo a conectar.

En este caso, el dispositivo Bluetooth HC-06 será el servidor, y el dispositivo móvil el cliente. No debemos preocuparnos de contemplar más de una conexión entrante en el dispositivo Bluetooth HC-06 dado que éste tan sólo se puede configurar como esclavo (a diferencia del Bluetooth HC-05 que puede configurarse también como maestro), por lo que sólo permite una conexión simultánea.

Para conocer la dirección de un dispositivo sería necesario realizar un proceso denominado descubrimiento o *discovery*, dado que las direcciones no se conocen a priori. Para realizar el proceso de *discovery*, añadiremos al *layout* de la aplicación un botón con el texto "Buscar", que se encargue de activar el proceso de descubrimiento. También declararemos un *ArrayList* de dispositivos Bluetooth (*BluetoothDevice*) para almacenar aquellos dispositivos que se vayan descubriendo.

Una vez se pulse el botón (codificando el método *onClick* de éste), se comprobará si existe un proceso de descubrimiento en curso, en cuyo caso cancelamos y lo volvemos a iniciar.

```
if(listaDispositivos != null){
    listaDispositivos.clear();
}
// Comprobamos si existe un descubrimiento en curso. En caso afirmativo, se cancela.
if(bluetoothAdapter.isDiscovering())
    bluetoothAdapter.cancelDiscovery();

// Iniciamos la busqueda de dispositivos
if(bluetoothAdapter.startDiscovery()) {
    Toast.makeText(this, "Iniciando busqueda de dispositivos", Toast.LENGTH_SHORT).show();
```

Figura 31. Extracto de código de la función buscarDispositivos que permite iniciar la búsqueda de dispositivos.

Instanciamos un *BroadcastReceiver* que se encargue de detectar dos tipos de acciones:

- *BluetoothDevice.ACTION_FOUND*: que se active cada vez que se descubra un nuevo dispositivo. Servirá para añadirlo al array.
- *BluetoothAdapter.ACTION_DISCOVERY_FINISHED*: que se lance cuando finalice el proceso de *discovery*.

```
// Cada vez que se descubra un nuevo dispositivo por Bluetooth, se ejecutara
if (BluetoothDevice.ACTION_FOUND.equals(action))
{
    if(listaDispositivos == null)
        listaDispositivos = new ArrayList<>();

    // Extraemos el dispositivo del intent mediante la clave BluetoothDevice.EXTRA_DEVICE
    BluetoothDevice dispositivo = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
    listaDispositivos.add(dispositivo);
    Log.v("Dispositivo encontrado: ", descripcionDispositivo);
}
// Cuando finalice la busqueda, se ejecutara
else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action))
{
    lvDispositivos = (ListView)findViewById(R.id.lvDispositivos);

    arrayAdapter = new AdaptadorDispositivosBluetooth(getBaseContext(), android.R.layout.sim
    lvDispositivos.setAdapter(arrayAdapter);
```

Figura 32. Extracto de código de la funcionbroadcastReceiver que permite realizar acciones durante el proceso de descubrimiento.

Cuando finalice el proceso de descubrimiento se creará una *ListView* que muestre los dispositivos, será necesario crear un adaptador para que los dispositivos se muestren correctamente en el elemento. La clase se llamará

*AdaptadorDispositivosBluetooth* que se encargará de adaptar los dispositivos en los listView generando dinámicamente un objeto interfaz o *view* personalizado. Heredará de *ArrayAdapter* y contendrá dos atributos, la lista de dispositivos y el contexto. Así mismo, mostrará los dos *textview* superpuestos, el superior y de mayor tamaño será el nombre del dispositivo y el menor e inferior será la dirección física de éste.

El resultado del desarrollo durante este apartado se puede observar en la figura 33.

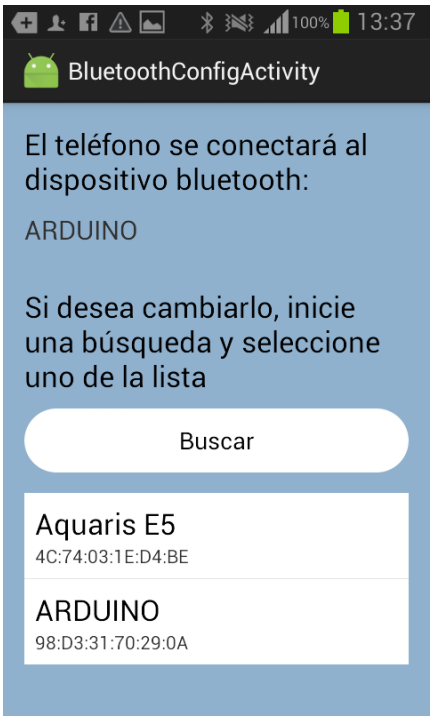



Figura 33. Interfaz del resultado del proceso de descubrimiento.

Sin embargo, cabe decir que el proceso de descubrimiento consume muchísimos recursos, por lo que será bastante útil el contenido del siguiente apartado.

### 5.2.3 Lectura y escritura de dispositivo en fichero

Para simplificar la utilización de la aplicación, y a modo de ahorro de recursos, se ha decidido guardar en un fichero el último dispositivo al que se ha conectado. Por ello, el usuario no tiene que iniciar un proceso de descubrimiento cada vez que inicie la aplicación, sino que directamente se podrá conectar al dispositivo por defecto guardado en ésta.

Este dispositivo se configurará en un fichero dentro de la memoria del teléfono. Hay tres lugares donde poder almacenarlo: en la memoria interna del teléfono, la tarjeta SD (si se dispone de ésta) o en la aplicación en forma de recurso.

No obstante, en la aplicación en forma de recurso no se puede almacenar porque, aunque es útil en muchos casos, sólo se podrá usar cuando no necesitemos realizar modificaciones del mismo, ya que por defecto, tendremos limitado el acceso a sólo lectura. La aplicación necesita guardar el dispositivo en el fichero, por lo que no es posible almacenarlo en este tipo de memoria.

De tal manera, no podemos decidir almacenar el fichero en la memoria SD dado que no todos los dispositivos disponen de este tipo de memoria, por lo que se almacenará en la memoria interna del teléfono, lo cual es un proceso realmente sencillo.

Android proporciona para ello el método *openFileOutput* que recibe como parámetros el nombre del fichero y el modo de acceso con el que queremos abrir el fichero. Este modo de acceso puede variar entre *MODE_PRIVATE* para acceso privado desde nuestra aplicación (crea el fichero o lo sobrescribe si ya existe), *MODE_APPEND* para añadir datos a un fichero ya existente, *MODE_WORLD_READABLE* para permitir a otras aplicaciones leer el fichero, o *MODE_WORLD_WRITABLE* para permitir a otras aplicaciones escribir sobre el fichero. Para esta aplicación se ha decidido establecer el modo privado, dado que no vamos a añadir datos al fichero existente, sino sobrescribirlo. Además, los dos últimos modos no deberían utilizarse dada su peligrosidad, de hecho, han sido declarados como obsoletos (*deprecated*) en la API 17.

Este método devuelve una referencia al *stream* de salida asociado al fichero (en forma de objeto *FileOutputStream*). Tras eso, convertiremos este *stream* a un *OutputStreamWriter* para escribir el dispositivo en el fichero.

```
OutputStreamWriter fileOut= new OutputStreamWriter(
        openFileOutput(nombreFichero , Context.MODE_PRIVATE));
fileOut.write(dispositivo.toString());
```

Figura 34. Extracto de código de la función guardarDispositivoEnFichero que permite escribir el dispositivo en un fichero txt.

Cabe recordar que, cuando almacenamos ficheros en la memoria interna, debemos tener en cuenta las limitaciones de espacio que tienen muchos dispositivos, por lo que no se debería abusar de este espacio utilizando ficheros de gran tamaño.

En Android, este fichero se guardará por defecto en la ruta:

/data/data/<paquete.java>/files/<nombre_fichero>

y será de acceso privado desde fuera de la aplicación, es decir, no se podrá encontrar accediendo desde el gestor de archivos ni desde otra aplicación.

Por otro lado, al elegir directamente la opción de monitorizar, se conectará al dispositivo guardado en el fichero, por lo que deberá leerlo de éste.

El proceso de lectura es igual de fácil que el de escritura, pero utilizando la clase *BufferedReader* y un *stream* de lectura.

```
BufferedReader fin =
        new BufferedReader(
                new InputStreamReader(
                        openFileInput(nombreFichero)));

String direccionDispositivoRemoto = fin.readLine();
fin.close();
```

Figura 35. Extracto de código de la función leerDispositivoDeFichero que permite leer el dispositivo de un fichero txt.

### 5.2.4 Conexión de dispositivos

Para intercambiar información por medio del protocolo Bluetooth, se ha necesitado estudiar algunas nociones básicas de la arquitectura cliente-servidor. Sin embargo, no ha sido complicado dado que este protocolo no se diferencia gran cosa de otros protocolos como TCP, y este sí ha sido bastante estudiado a lo largo de la carrera.

El funcionamiento es básicamente el siguiente. Cada dispositivo tiene tres capas o tres componentes: servidor, cliente y conexión, y cada uno estará gestionado por un hilo independiente.

Para empezar, el servidor crea un socket para escuchar y se bloquea atendiendo solicitudes de conexión hasta que una conexión entrante sea recibida, mientras el cliente instancia un dispositivo Bluetooth a partir de su dirección MAC y abre un nuevo socket de tipo *BluetoothSocket* a partir del dispositivo que obtuvo previamente, y a través del socket realiza una solicitud de conexión.

El servidor acepta la conexión, se lo notifica al cliente y abre un socket como resultado de aceptar la conexión, el cliente la recibe y sale de la espera ocupada. Tras esto, tanto el cliente como el servidor obtienen los flujos de entrada y salida de su respectivo socket y el hilo de la conexión empieza a sondear el flujo de entrada esperando obtener datos. Si obtiene los datos, los envía a la interfaz mediante un *handler*.

Se comenzará codificando el hilo encargado de establecer la conexión. Crearemos una clase específica que contendrá los tres hilos, de modo que la funcionalidad entre interfaz de usuario y lógica de la aplicación se mantenga separada. En esta clase se recogerán los datos recibidos por el flujo de comunicación entre dispositivos, se creará un paquete y se enviará a la interfaz de usuario mediante un *handler*.

Un *handler* tiene dos funciones principales: programar mensajes que serán ejecutados en algún momento en el futuro o encolar una acción que será ejecutada en un hilo distinto al actual, que es el caso que nos ocupa. Por lo tanto, será el método que utilizaremos para comunicar el hilo encargado de la interfaz de usuario y el hilo encargado de la conexión.

El *handler* responderá a los eventos de lectura notificados por el método run en el hilo de conexión. Esto es gracias a los métodos *handleMessage* y *obtainMessage* del handler. El método *handleMessage* será el encargado de recibir el mensaje cuando otro hilo llame al método *obtainMessage* con los datos leídos del buffer de entrada, para poder visualizarlos en los *textview* correspondientes a la monitorización.

### 5.2.5 Monitorización de las variables en los TextView

Como último punto del desarrollo de la aplicación Android, se creará una *Activity* para representar los datos recibidos y poder monitorizar las variables fisiológicas.

Para esto, se establecerán diferentes *textView* a lo largo de la pantalla, tal y como se muestra en la figura 36, jugando con las posiciones y tamaños de éstos en el *layout*.

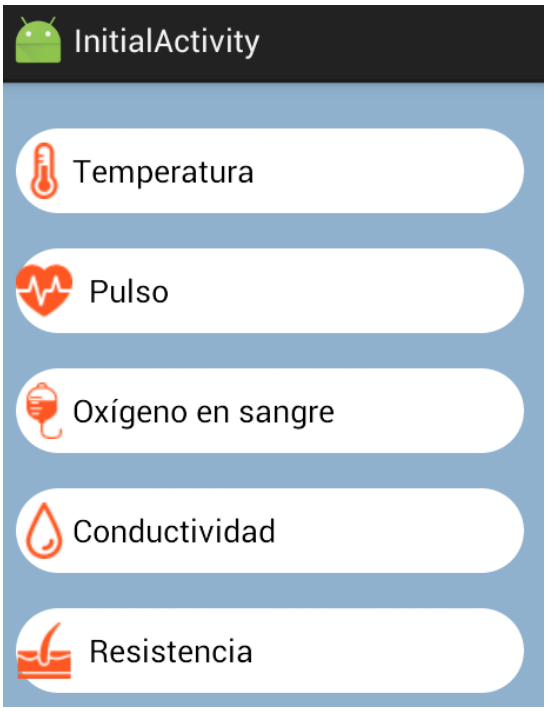



Figura 36. Interfaz de monitorización de variables.

Cada vez que el *handler* sea invocado, recibiremos en la interfaz los datos, estos se separarán por campos, al igual que en la aplicación hecha en Matlab, y se mostrarán al lado de su correspondiente etiqueta superpuesta en los *textview*.

El resultado de la monitorización se puede ver en la siguiente figura.


Figura 37. Monitorización de variables en la aplicación móvil.

## 5.3 Problemas encontrados

El principal problema o, más bien la principal dificultad, ha sido el aprendizaje del desarrollo con el lenguaje Android. Para esto, se han seguido diferentes tutoriales y leído bastante documentación.

No obstante, otro de los problemas encontrados ha sido la configuración y envío de datos mediante el dispositivo Bluetooth, dado que venía mal configurado de fábrica y había que adivinar la velocidad a la que venía configurado para poder reconfigurarlo a gusto personalizado. Tras varios intentos y muchísima lectura de documentación, foros y análisis, no se consiguió dar con la velocidad adecuada, por lo que hubo que resetear el dispositivo y configurarlo de nuevo.

Por otro lado, al leer los datos recibidos en el dispositivo móvil, no parecían tener formato, es decir, a veces llegaba el paquete entero, a veces llegaba a medias, etc. El problema resultó ser que el hilo tarda bastante en procesar el handler y en pasarle los datos, por lo que a veces se llamaba al handler sin haber terminado de procesar los datos. La solución fue ponerle un tiempo de espera de medio segundo más y los datos se procesaban correctamente. Además, esta modificación no afecta negativamente al rendimiento del sistema.

# Capítulo 6.
# Validación de la plataforma de captura y de las aplicaciones

Una vez realizado el código, se procede a comprobar si los resultados obtenidos cumplen con los requisitos establecidos inicialmente. Se comienza por validar el resultado final de la aplicación de Matlab, a continuación se valida el funcionamiento de la aplicación móvil y por último si las variables mostradas son correctas y se corresponden con las observadas.

## 6.1 Validación de la aplicación de monitorización en PC

En un principio se valida la interfaz de usuario. Comprobaremos que se conecte al dispositivo (en caso contrario da un error), que se monitoricen correctamente las variables enviadas y que se desconecte el dispositivo sin fallos.

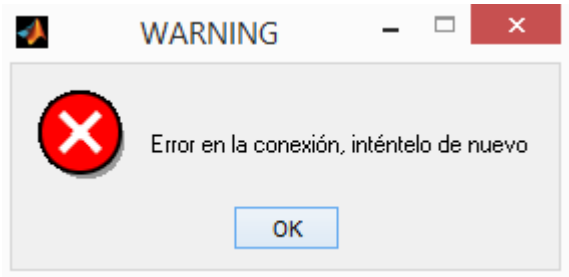



Figura 38. Cuadro de diálogo de error a la hora de conectar el dispositivo.

La monitorización correcta de los datos se refleja en la figura 39. Una vez finalizada la misma, se procede al parado y a la desconexión del dispositivo con los botones habilitados para tal fin. El resultado se desarrolla sin incidentes, por lo que se da por válida la interfaz de usuario.

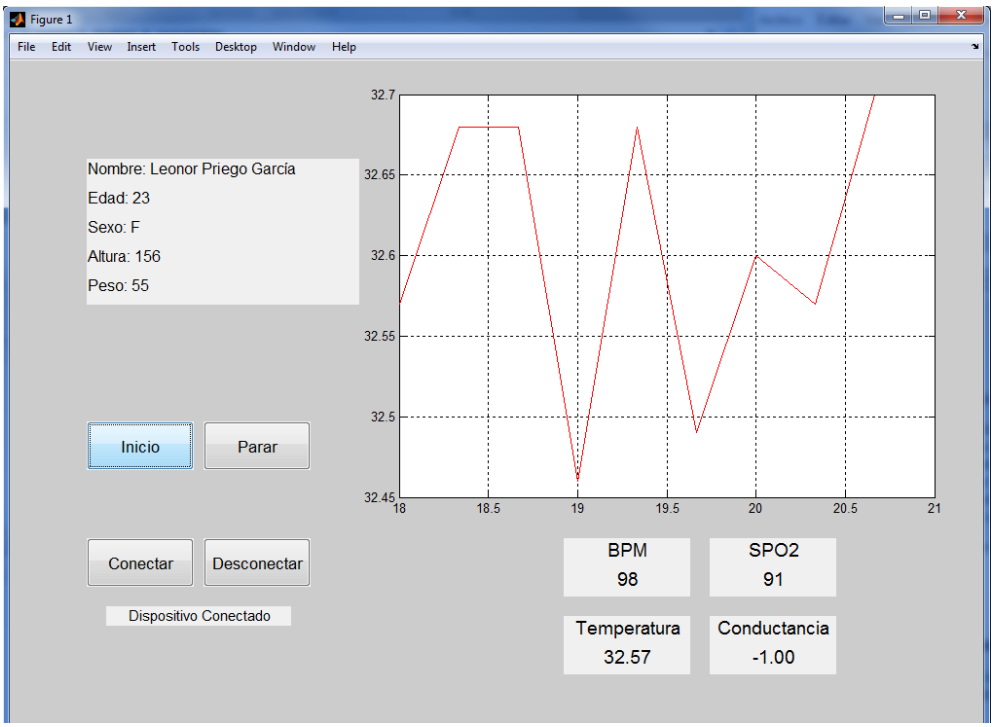



Figura 39. Interfaz gráfica de la aplicación en Matlab.

## 6.2 Validación de la aplicación móvil

Por consiguiente, se continúa validando la aplicación móvil. Se comprueba que busque los dispositivos correctamente, que guarde el dispositivo del fichero y lo lea de éste, que se conecte a los dispositivos y envíe datos adecuadamente.

La lectura del fichero y búsqueda de dispositivos se realiza de forma satisfactoria, así se puede comprobar en la figura 33. De tal manera, la lectura, conexión y monitorización se observan en la figura 40, dado que si muestra datos es que se ha conectado previamente al dispositivo que ha leído del fichero de configuración. Por lo tanto, se realiza correctamente.


Figura 40. Muestra de valores recibidos en la aplicación Android.

## 6.3 Validación de muestra de datos correctos

En este apartado se decide la fiabilidad de la plataforma puesto que se verifica que las variables mostradas en ambas aplicaciones sean las correctas. Para esto, hacemos dos comprobaciones.

En primer lugar, constatamos si los datos mostrados en la plataforma son los recibidos por los sensores. Para ello, se realiza una prueba que consiste en colocar el sensor de temperatura en diferentes zonas del cuerpo, frotando algunas para conseguir mayor temperatura, y visualizar en una gráfica si realmente se muestra lo llevado a cabo.

En principio, el sensor no estaba pegado a ninguna parte del cuerpo, por lo que captaba la temperatura ambiente (que rondaba los 32,7ºC), tras esto se colocó cerca de la axila, que es donde se suele medir la temperatura corporal y se pudo observar cómo aumentó la temperatura casi 5ºC más. Después se colocó en diferentes zonas más frías del cuerpo para que disminuyera y para terminar, frotamos la yema del dedo para alcanzar una temperatura bastante superior, pudiendo comprobar en la gráfica la fiabilidad de esta plataforma dado que realmente mostraba los cambios de temperatura que iban sucediendo.

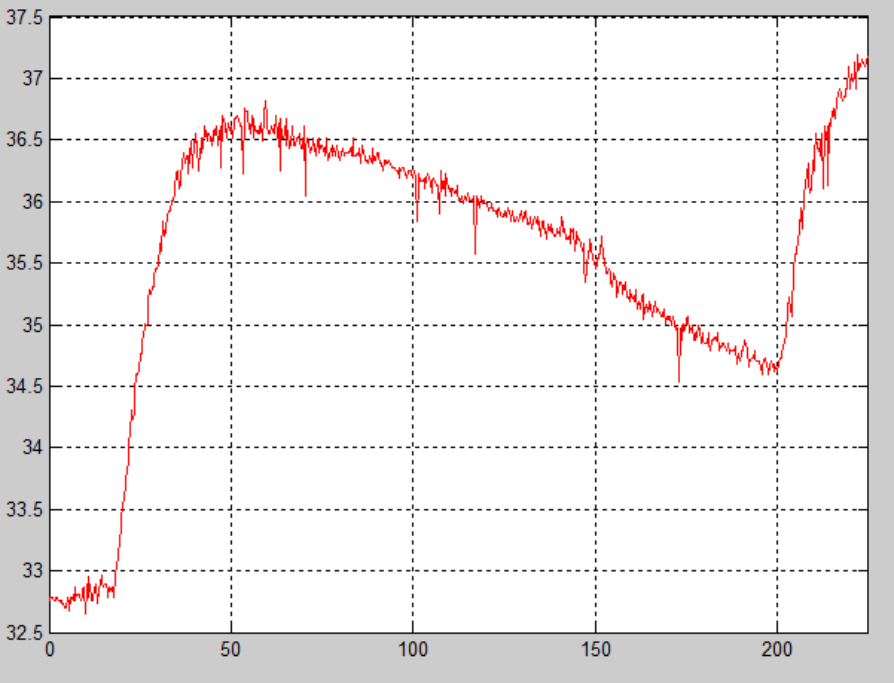



Figura 41. Gráfica que permite comprobar la evolución de la temperatura en el experimento.

Por otra parte, también se comprobó la fiabilidad de esta plataforma con el pulsioxímetro. Es una buena herramienta para verificar si realmente se muestran los valores que envía el sensor, puesto que dispone de una pantalla que muestra los valores obtenidos. La fiabilidad de las plataformas

se puede observar en las figuras siguientes, dado que en éstas se muestran los mismos valores que los capturados por el sensor.

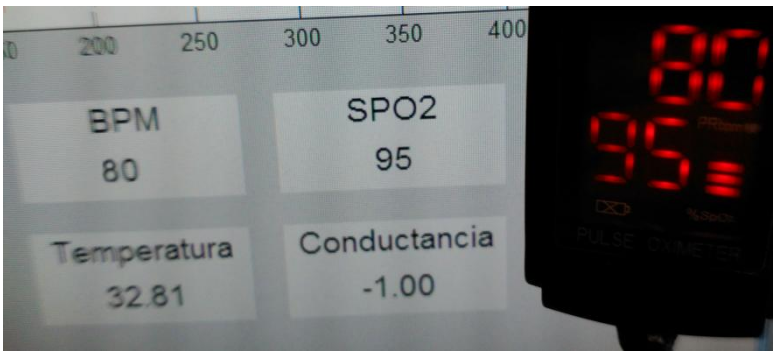



Figura 42. Comparación de valores mostrados en la aplicación Matlab y en el pulsioxímetro.

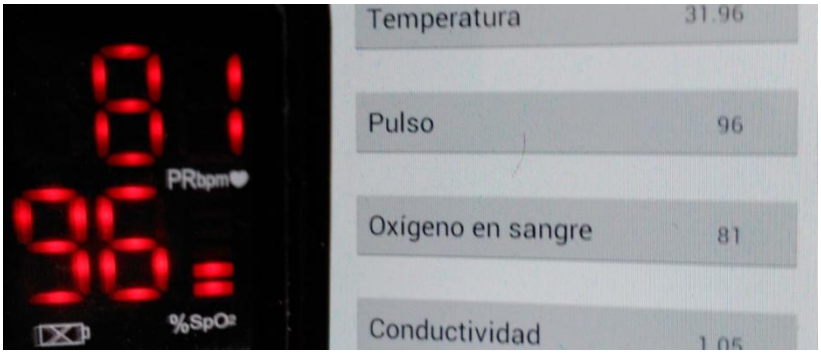



Figura 43. Comparación de valores mostrados en la aplicación móvil y en el pulsioxímetro.

# Capítulo 7.
# Conclusiones y líneas futuras

Gracias al largo alcance que ha tenido el movimiento MAKER, las plataformas abiertas como la de Arduino y los avances tecnológicos, es bastante fácil y asequible adquirir los conocimientos y componentes necesarios para crear e implementar sistemas de bajo coste como el descrito en esta memoria.

En este trabajo se ha desarrollado una aplicación en Matlab para supervisar la actividad de ciertas variables fisiológicas durante el proceso anestésico. Así mismo, como segundo objetivo se propuso utilizar la plataforma *low-cost* creada para que las personas pudieran monitorizarse ellas mismas desde cualquier lugar. Por lo que para este fin se desarrolló una aplicación móvil que mostrara las variables recibidas mediante Bluetooth.

Comparando los resultados pudimos comprobar y concluir que los software implementados son herramientas fiables que pueden ser útiles tanto en el proceso de monitoreo anestésico como para poder llevar un control rutinario de variables fisiológicas en humanos. Además, se podrán guardar los datos de las variables procesadas en ficheros para posteriores investigaciones y análisis que indiquen la relación entre el dolor sufrido por los pacientes y las muestras obtenidas.

Para finalizar, hacemos balance de los objetivos que se plantearon al inicio del trabajo de fin de grado y se ha de concluir afirmando que se han alcanzado las metas propuestas en ambas fases del proyecto.

## 7.1 Líneas futuras

Como líneas futuras se puede proponer una mejora que suponga la comunicación de la plataforma con un servidor que almacene los ficheros de todos los pacientes como registro de historial. Asimismo, también se podría añadir a la aplicación móvil comunicación GSM para permitir enviar un mensaje a un familiar de la persona que se está monitorizando para advertirle de un posible malestar de ésta.

También se podrían estudiar los valores de las variables para obtener una relación entre el dolor y la actividad del paciente durante la intervención. El resultado de este estudio podría ser útil para poner en práctica un mecanismo de control para suministrar analgésicos durante el proceso quirúrgico.

# Capítulo 8.
# Summary and Conclusions

Thanks to long-range MAKER movement has had, open platforms such as Arduino, in this case, and technological advances is quite easy and affordable to acquire the knowledge and components needed to create and implement low cost systems as described in this report.

The main objective of this TFG is the development of an application in Matlab to monitor the activity of certain physiological variables during the anesthetic process. Also, as a second objective it was proposed to use a low-cost platform created to that people could themselves be monitored in anywhere. So, for this purpose a mobile application that show the variables received through Bluetooth was developed.

Comparing the results we can check and conclude that the implemented software are quite reliable tools that can be useful for monitoring anesthetic process and to take a routine check. In addition, data can be saved in files for further research and analysis that indicate the relationship between pain suffered by patients and samples obtained.

Finally, we can check the proposed objectives at the start of the project and must conclude saying that we have achieved the goals set in both phases of the project.

## 8.1 Future researches

As future researches we could propose an improvement involving communication with a server platform that stores files of all patients as history log. Furthermore, we could add a GSM mobile communication to the mobile application to enable send a message to a relative of the person which is being monitored to warn of a possible upset.

Also, we could study values of variables to obtain a relation between pain and patient activity during surgery. The result of this study could be useful to implement a mechanism of control to supply analgesic during the surgery process.

# Capítulo 9. Presupuesto

Por un lado, una de las principales intenciones del proyecto era construir una plataforma de bajo coste que pudiese estar al alcance de cualquier persona que desee monitorizarse y tener un control del estado de su cuerpo. Si analizamos el presupuesto total de este primer sistema, se ha logrado mantener entre los límites teóricos que se habían fijado desde un primer momento, donde se impuso no superar la barrera de los 250€.

Por otro lado, el segundo objetivo ha sido desarrollar una aplicación móvil que permita la monitorización de las variables recibidas mediante Bluetooth. En este caso, no ha habido que comprar ningún material, por lo que no se especificará nada en el presupuesto con respecto a esta parte del desarrollo.

En el siguiente apartado, se desglosará el presupuesto total del sistema.

## 9.1 Presupuesto Total

| Artículos | Coste |
|---|---|
| Arduino UNO [Link] | 20€ |
| Módulo Bluetooth HC-06 [Link] | 9,31€ |
| E-Health Platform Shield [Link] | 75 € |
| Sensor Temperatura Corporal [Link] | 20€ |
| Pulsioxímetro [Link] | 55€ |
| Sensor galvánico [Link] | 30€ |
| Sensor Electrocardiograma [Link] | 35€ |
| TOTAL | 244,31€ |

Tabla 5. Presupuesto total.

# Apéndice A. Repositorios de código

## A.1 Repositorio Código Arduino

Arduino - Android

Arduino - Matlab

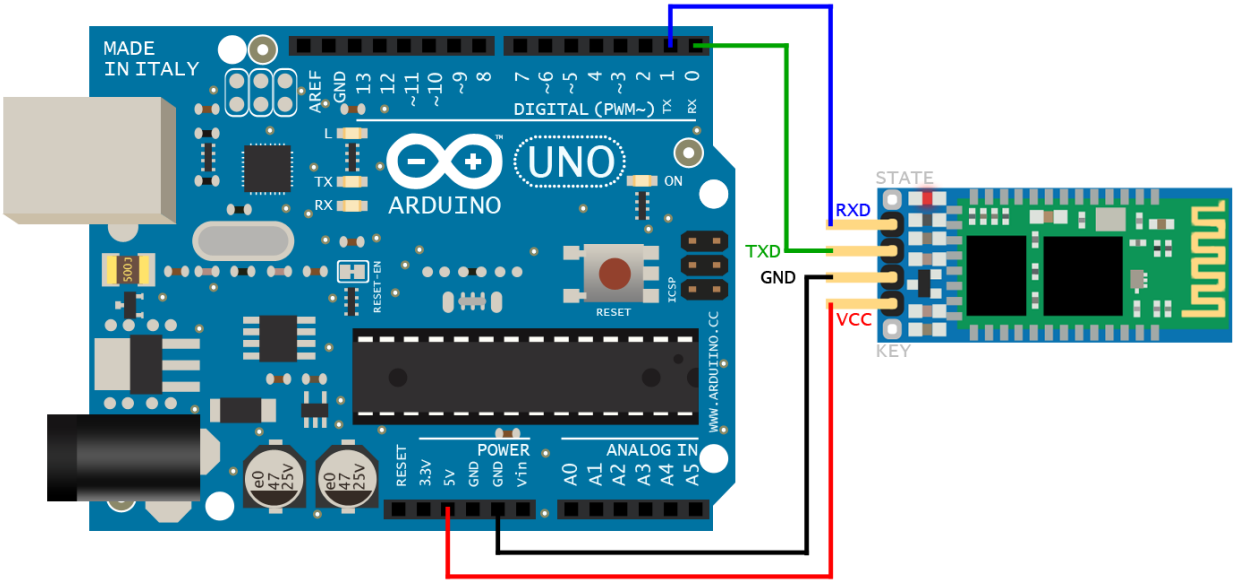
## A.2 Repositorio Código Matlab

https://github.com/Lprigara/e-health/tree/master/Matlab

## A.3 Repositorio Código Android

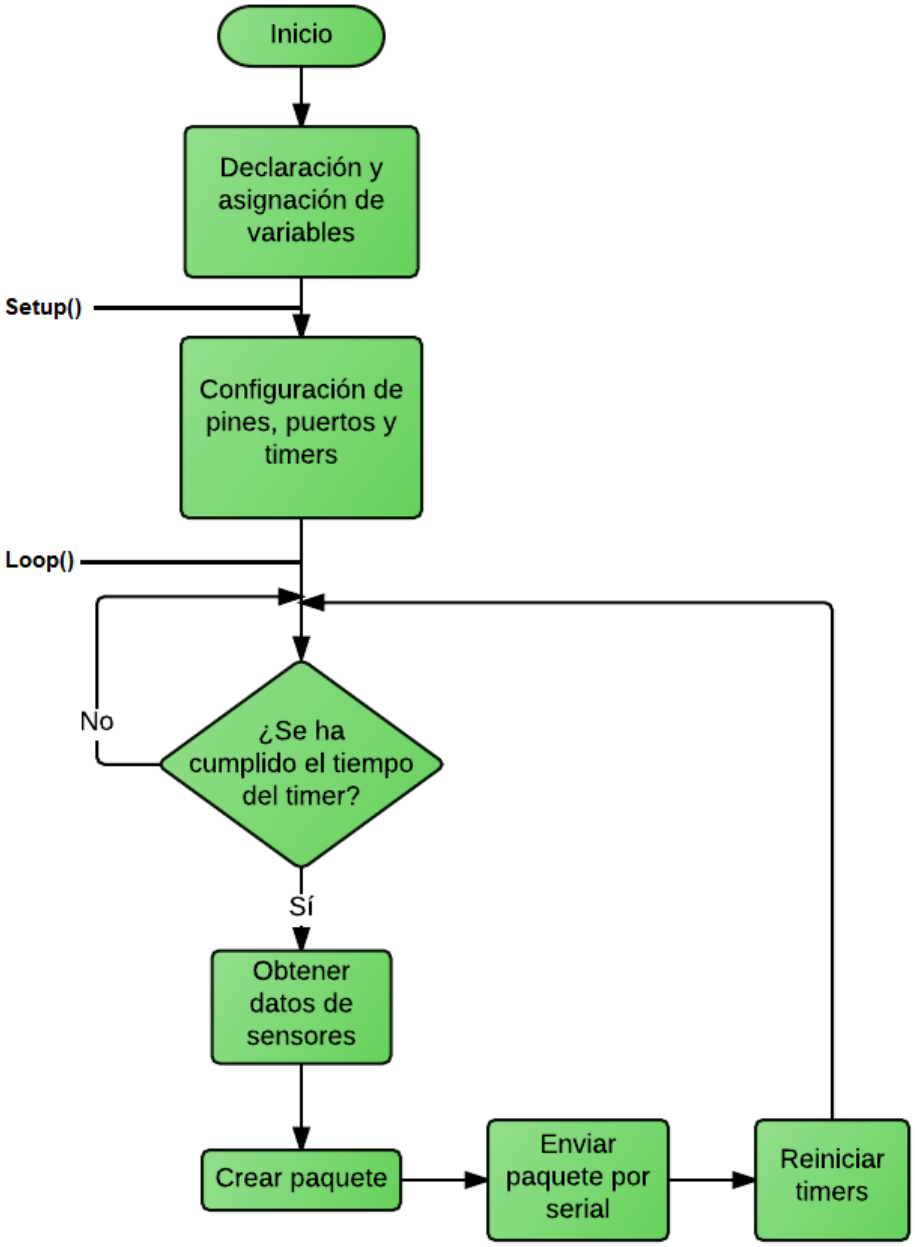
https://github.com/Lprigara/HealthApp

# Apéndice B.
# Esquema de conexión Arduino - Módulo Bluetooth HC-06

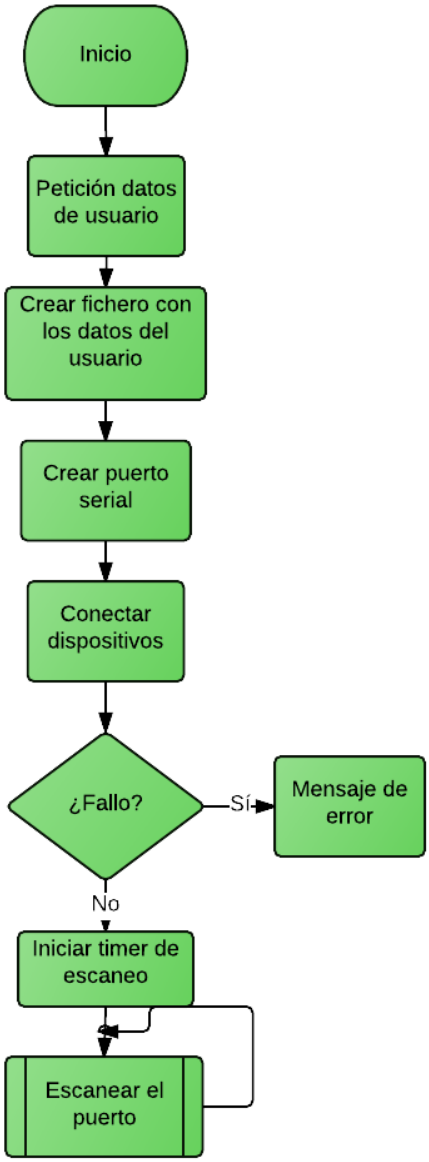


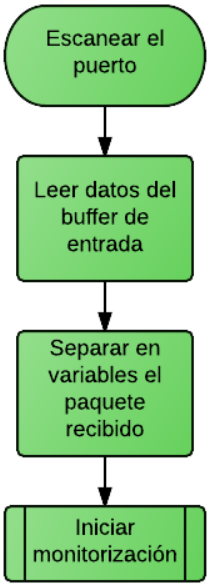
# Apéndice C.
# Diagramas de flujo

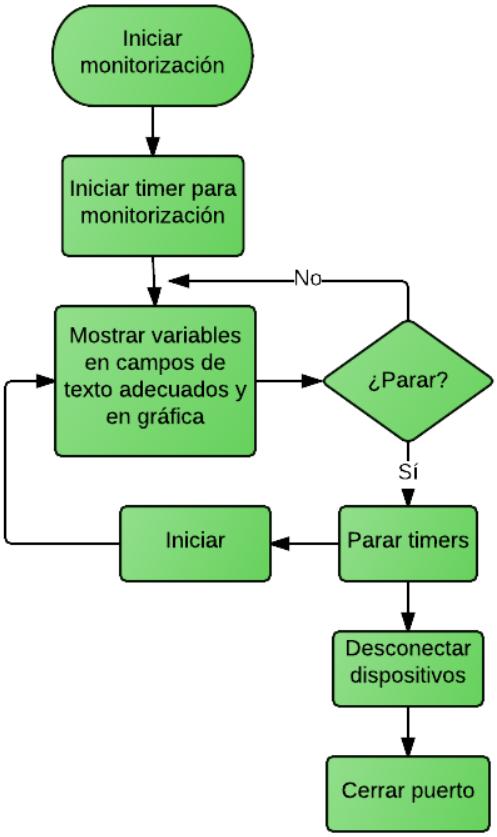
## C.1 Diagrama de flujo desarrollo en Arduino



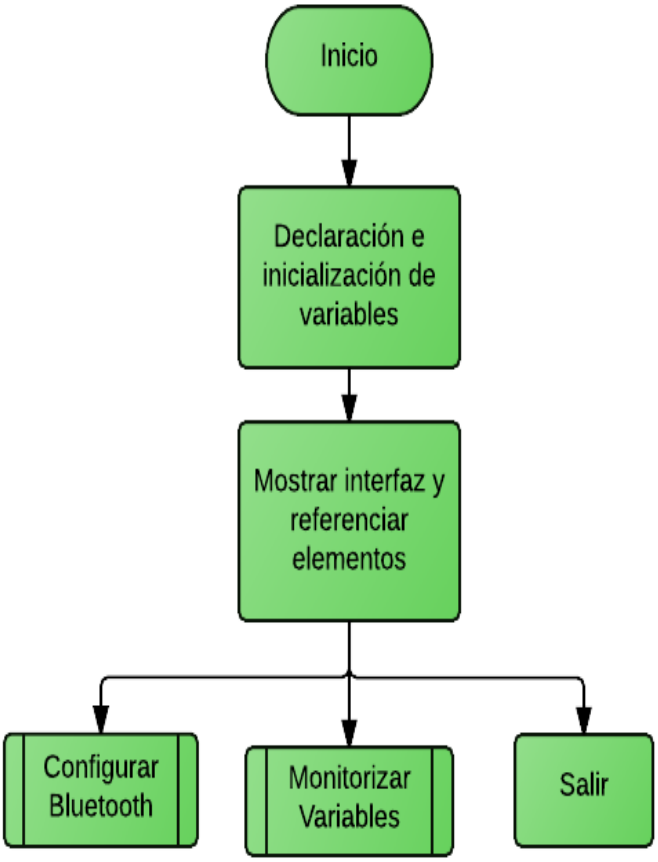
## C.2 Diagrama de flujo desarrollo en Matlab

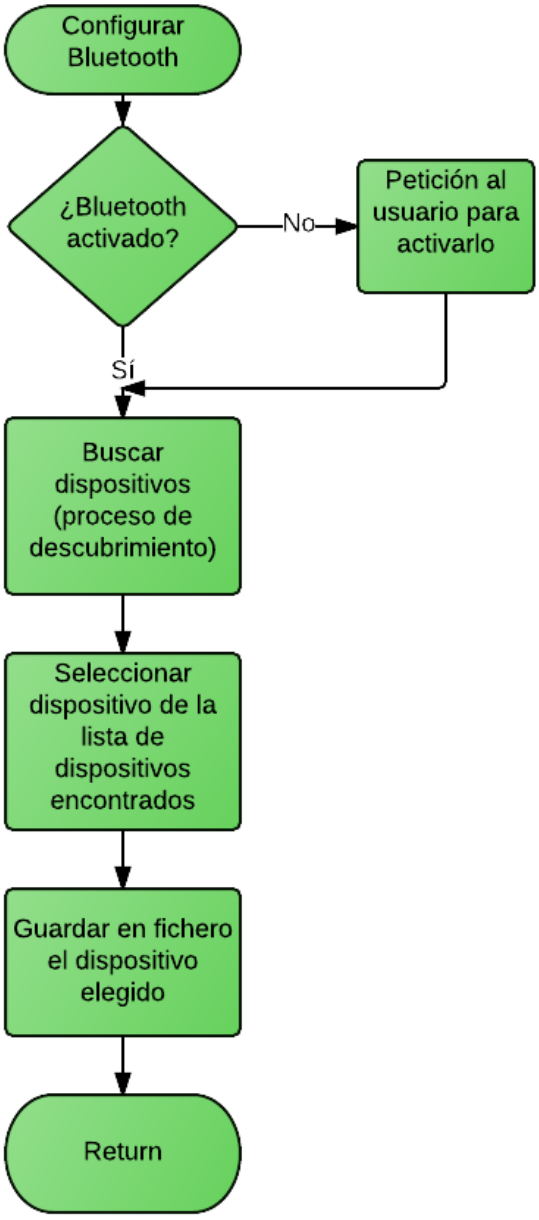


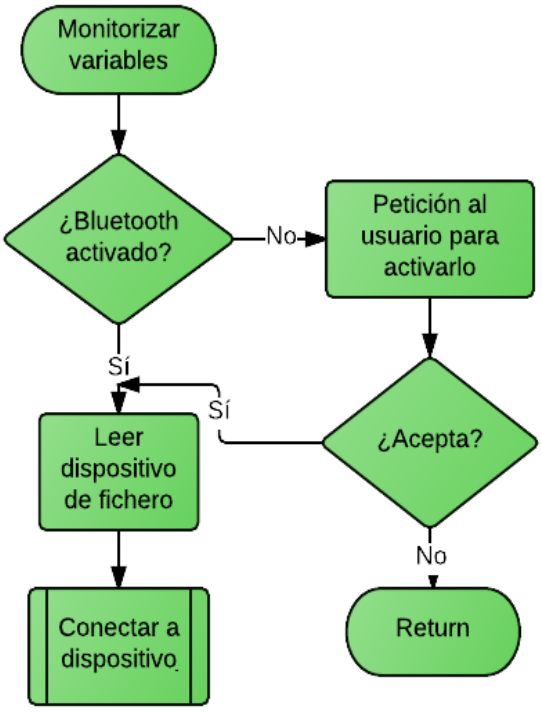



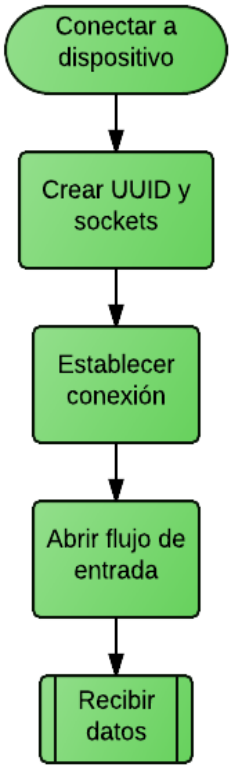



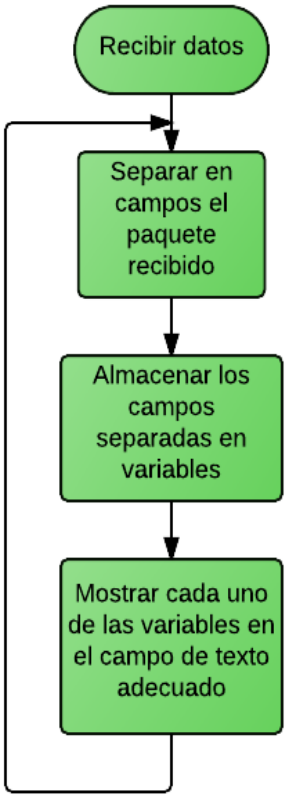
## C.3 Diagrama de flujo desarrollo en Android




Monitorizar variables
¿Bluetooth activado?
No
Petición al usuario para activarlo
Sí
Sí
Leer dispositivo de fichero
¿Acepta?
No
Conectar a dispositivo
Return


Conectar a dispositivo
Crear UUID y sockets
Establecer conexión
Abrir flujo de entrada
Recibir datos


Recibir datos
Separar en campos el paquete recibido
Almacenar los campos separadas en variables
Mostrar cada uno de las variables en el campo de texto adecuado

# Bibliografía


[1] Arduino [Link].

[2] Arduino UNO [Link].

[3] Bluetooth HC-06 [Link].

[4] Comunicación serial [Link].

[5] Conexión Arduino - Bluetooth [Link].

[6] Matlab [Link].

[7] Gráficas en Matlab [Link].

[8] Ciclo de vida aplicación Android [Link].

[9] Android [Link].

[10] Pulsioximetría [Link]

[11] Electrocardiograma [Link].

[12] Conductancia [Link].