_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

SISTEMA DE SENSORES INTEGRADO EN EQUIPACIÓN DE BOMBEROS

Víctor Luján Cuenca

Tutor: José Manuel Mossi García

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería de Telecomunicación

Curso 2016-17

Valencia, 24 de julio de 2017

Resumen

En el siguiente documento se desarrolla una parte teórica, en la que se realiza una investigación sobre los interfaces y protocolos del estándar MIPI para la comunicación entre dispositivos móviles con el fin de poder realizar una conexión directa entre unas gafas de realidad aumentada y un microcontrolador, y una parte práctica, donde se explica el funcionamiento de una serie de sensores para la evaluación de los parámetros que se han considerado de mayor importancia a la hora de determinar el grado de riesgo en el que se encuentra un bombero durante una operación.

Este proyecto forma parte de un conjunto mayor en el que se pretende dotar a los bomberos de diferentes accesorios para facilitarles su trabajo y minimizar los riesgos y desventajas que sufren al enfrentarse al fuego. Dicho sistema consta de un microcontrolador, unas gafas de realidad aumentada y una cámara térmica que ofrecen una imagen en función del calor que emiten los objetos sin que el usuario pierda la visión sobre la realidad física. Sobre esta base, se pretende mejorar la conexión entre el microcontrolador y las gafas y dotar al sistema de mayores prestaciones abasteciéndolo con sensores para hacerlo capaz de monitorizar la actividad del bombero en todo momento, así como avisarle de ciertas variables que se puedan detectar y medir, como por ejemplo la presencia de gases peligrosos.

Resum

Al següent document es desenvolupa una part teòrica, en la que es realitza una investigació sobre els interfaços i protocols de l'estandar MIPI per a la comunicació entre dispositius mòbils amb el fi de poder realitzar una conexió directa entre unes ulleres de realitat augmentada i un microcontrolador, ademés d'una part práctica, on s'explica el funcionament d'una serie de sensors per a l'evaluació dels paràmetres que s'han considerat de major importància a l'hora de determinar el grau de risc en el que es troba un bomber durant una operació.

Aquest projecte forma part d'un conjunt major en el que es pretén dotar als bombers de diferents accessoris per facilitar-los el seu treball i minimitzar els riscos i desavantatges que poden sofrir al enfrontar-se al foc. Aquest sistema consta d'un microcontrolador, unes ulleres de realitat augmentada i una càmara tèrmica que oferixen una imatge en funció del calor que desprenen els objectes sense que l'usuari perda la visió sobre la realitat física. Sobre aquesta base, es pretén millorar la conexió entre el microcontrolador i les ulleres i dotar al sistema de majors prestacions, abastint-lo amb sensors per a ferlo capaç de monitoritzar l'activitat del bomber en tot moment, així com avisar-lo de certes circumstàncies que es puguen detectar i mesurar mitjançant sensors, com per exemple la presència de gasos perillosos.

Abstract

In this document, it is explained a theorical part, where mobile devices connection interfaces and protocols from the MIPI standard are researched in order to achieve a direct connection between a microcontroller and the augmented reality glasses, and a practical one, where a group of sensors are explained and used to evaluate different parameters that have been considered important when determining the level of hazard to a firefighter when working on an operation.

This project becomes part of a bigger system that tries to provide firefighters a group of tools to make easier their job and minimize risks and disadvantages when fighting against fire. This system is formed by a microcontroller, a pair of augmented reality glasses and a thermal camera which provides images depending on items temperature without blocking user's vision about their physical world. Using this base, it is intended to improve connection between the microcontroller and the glasses and provide the system a better performance by adding sensors so firefighters' activity can be monitored in every moment. Furthermore, this system could alert about dangerous situations like toxic gases presence.

Índice

Capítulo 1. Introducción

El proyecto descrito en este texto caracteriza una pequeña sección de un proyecto más grande y complejo del cual se comentarán los aspectos más importantes. En concreto, la parte a desarrollar se compone de dos grandes bloques relacionados entre sí, del mismo modo que con las demás partes del conjunto, a través de su finalidad: el desarrollo de un sistema completo de apoyo y monitorización de las tareas que realizan los bomberos.

El proyecto completo incluye unas gafas de realidad aumentada, equipadas por el bombero, que son la base del desarrollo, permitiendo ofrecerle una visión térmica de su entorno sin una obstrucción de la visión normal. Sobre esa base, se pretende introducir una serie de mejoras y agregados que reduzcan el coste, aumenten la seguridad y la robustez del sistema y garanticen un valor añadido a los bomberos cuando se trate de una misión en la que el peligro es un factor que es necesario reducir.

Específicamente, el desarrollo que se relata en este escrito consiste en una parte teórica en la que se investigan los protocolos de la Alianza MIPI para la transmisión eficiente y de bajo consumo de señales en medios electrónicos. Más en concreto, en sistemas móviles y similares. Dicho bloque trata de comprender los protocolos y las distintas capas que MIPI ha diseñado con el fin aunar las tecnologías y conseguir un estándar común, para utilizarlos en la realización de una conexión directa cableada entre las gafas y el microcontrolador.

La segunda parte, de carácter más práctico, consiste en el acondicionamiento y programación de distintos sensores para ofrecer una monitorización de diferentes parámetros que se han considerado de interés para tratar de minimizar los riesgos a los que se exponen los trabajadores y disminuir los tiempos de reacción ante situaciones de peligro para estos.

1.1 Riesgos para los bomberos

Los bomberos ejercen una labor esencial para la seguridad pública. Su trabajo no solo pasa por extinguir incendios que pueden ser originados bajo distintas condiciones, sino que también deben encargarse de tareas como sacar a los heridos de automóviles y edificios en llamas bajo condiciones de alto riesgo, así como otras situaciones peligrosas. Durante una misión de rescate o extinción de incendios, un bombero se pone continuamente en riesgo. No sólo del fuego, sino también al exponerse a muchos otros peligros como gases o edificios que pueden llegar a colapsarse. Es por ello que resulta importante ser conscientes de estos riesgos que atañen a su seguridad.

Entre los distintos riesgos para la salud a los que se ven sometidos, el que parece más obvio es el riesgo de quemaduras. Las quemaduras pueden ser graves e incluso mortales, especialmente si un bombero se ve atrapado en un edificio en llamas. Por otra parte, la inhalación de humo es otro riesgo importante y puede fácilmente ahogar a un bombero cuyo equipo le falla. Un informe de la U.S. Fire Administration (Administración de Incendios de los Estados Unidos) señala que la combinación de contacto directo con las llamas y la inhalación de humo causa un 34% de todas las lesiones de los bomberos.

Asimismo, los bomberos también se encuentran en un riesgo significativamente mayor de desarrollar cáncer. Entre los distintos tipos de cáncer, los que se encuentran más estrechamente relacionados con este tipo de trabajo son el cáncer testicular, el cáncer de próstata, el mieloma múltiple y el linfoma no Hodgkin. Esto es debido a su exposición frecuente a productos químicos cancerígenos, incluyendo los gases de los escapes de motores como, por ejemplo, de coches, así como el gas de los camiones de bomberos. También pueden exponerse a la presencia de gases como benceno, el formaldehído, el estireno, el cloroformo o incluso el hollín. Estas sustancias estarían presentes en el aire mientras están combatiendo un incendio, así como también en la estación de bomberos. Estos gases pueden ser absorbidos a través de la piel o inhalándolos.

Frecuentemente, los bomberos deben entrar en edificios en llamas donde los pisos, techos y escaleras pueden colapsar bajo ellos dejándolos en un estado de vulnerabilidad. Si colapsa un piso, el bombero puede caer tres metros o más y la posibilidad de lesiones graves, incluyendo fracturas de huesos, es muy alta cuando se trata de alturas así. Además, generalmente, deben usar escaleras para llegar a las víctimas atrapadas o para dirigir sus mangueras a los puntos calientes, por lo que el riesgo de caer también está presente, y una caída de una escalera o plataforma alta a menudo causa una lesión lo suficientemente grave como para inhabilitar del trabajo a un bombero.

Por último, un riesgo que no se ve tan directamente, tiene que ver con el pesado equipo que los bomberos suelen usar para protegerse del calor y de las llamas como máscaras, chaquetas gruesas y tanques de oxígeno. Dicho equipo se suma a la cantidad de esfuerzo que un bombero tiene que realizar durante un incendio para avanzar entre obstáculos hasta las víctimas o a los puntos de calor para extinguir las llamas. A veces incluso deben manejar escaleras, mangueras, hachas y otros equipos para extinguir incendios y alcanzar otros puntos, por lo que, a menudo, tienen que soportar una gran cantidad de peso, así como de estrés durante muchas horas continuadas cuando combaten contra un incendio difícil. Según la U.S. Fire Administration, el agotamiento por el excesivo esfuerzo es la causa principal de las lesiones de los bomberos, representando el 25% de todas las muertes relacionadas con el fuego. (1)

Capítulo 2. Objetivos

El objetivo de este documento es el desarrollo y documentación de los dos bloques comentados en la introducción: El estudio de los protocolos del estándar MIPI y el acondicionamiento de una serie de sensores utilizados para monitorizar el entorno del bombero.

Respecto al estándar MIPI, los objetivos serán la comprensión de los diferentes conceptos referentes a la Alianza MIPI, así como las características principales y los protocolos de señalización utilizados con el fin de encontrar una forma viable de transmitir las señales desde un microcontrolador a unas gafas de realidad aumentada de forma cableada, obteniendo los beneficios de este estándar como pueden ser el bajo consumo o la resistencia ante las interferencias electromagnéticas.

En la parte práctica del proyecto, se abordan teóricamente un conjunto de sensores utilizados para medir la temperatura, la concentración de gas y la frecuencia cardíaca del usuario, así como para para detectar posibles caídas del portador y comunicarlo al equipo base. Se explicarán su funcionamiento y los protocolos de comunicación utilizados para, posteriormente, realizar la obtención de datos de forma interactiva a través del terminal, de modo que se lanzarán comandos y se interpretarán los resultados para alcanzar un valor numérico capaz de ser interpretado por el usuario como una cantidad medible de un factor de riesgo concreto. Finalmente, se avanzará hacia el desarrollo software en lenguaje C++ para realizar esas medidas de forma automática con una frecuencia relativamente alta y, en el caso de ser necesario, una interpretación automática por parte del microcontrolador.

Capítulo 3. El proyecto

3.1 Contexto

El proyecto sobre el que se ha trabajado se encuentra actualmente en desarrollo y su conjunto completo se divide en diferentes partes que se agrupan en un sistema electrónico que tiene como finalidad instalarse en el traje del bombero y ofrecer a sus usuarios una serie de herramientas para mejorar su seguridad y la eficacia en sus operaciones.

Actualmente, el sistema consiste en una cámara térmica capaz de generar imágenes mediante la captación del calor que desprenden los objetos y la asignación de un color más cálido o más frío según su temperatura. Esta cámara se instala en el casco y sus imágenes serán procesadas por un microcontrolador. En este caso, el microcontrolador es una Raspberry Pi que se encargará de recibir las imágenes de la cámara térmica y de enviarlas a unas gafas de realidad aumentada que llevará el bombero. Estas gafas se colocan como unas gafas normales y permiten, a través de unos microproyectores instalados en los laterales, proyectar imágenes digitales solapadas sobre la imagen real que el bombero ve en ese momento. Gracias a estas imágenes solapadas, el bombero puede ver lo que pasa a su alrededor y además tener una visión térmica sin tener que desviar la mirada ni su atención hacia una pantalla secundaria en situaciones de riesgo. A causa de la gran cantidad de humo, en ocasiones la visión que tienen los bomberos es nula y deben guiarse por el tacto. Es por este motivo que la visión que ofrece la cámara térmica es útil en zonas de mucho humo en los que la cámara permite localizar focos de calor o cuerpos humanos donde los ojos no pueden ver nada.

Figura 1. Ejemplo de gafas de realidad aumentada.

http://www.qrcodepress.com/augmented-reality-glasses-patent-filed-facebook-oculus-takes-next-step/8533677/

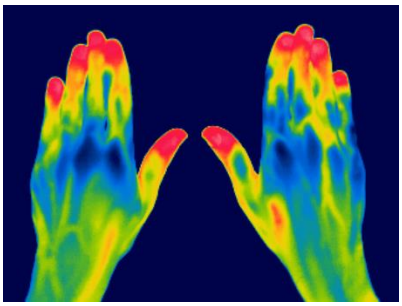


Figura 2. Ejemplo de visión con cámara térmica.

http://pressreleases.responsesource.com/news/76944/bath-s-royal-national-hospital-for-rheumatic-diseases-incorporates-latest/

La parte del proyecto abordada consiste en incorporar un conjunto de sensores en el traje del bombero para la monitorización de los distintos factores de riesgo que amenazan al trabajador durante las operaciones, de manera que el bombero sepa en todo momento las condiciones en las que se encuentra, como la temperatura o la concentración de gases nocivos a los que está expuesto, para minimizar dichos riesgos, así como la posible monitorización de dichos datos desde fuera para poder controlar la operación de una forma más exhaustiva.

3.2 Realidad aumentada

La realidad aumentada es el término utilizado para definir la visión de un entorno físico, del mundo real, a través de un dispositivo, es decir, se combinan los elementos físicos tangibles con elementos que proporcionan una información virtual solapada sobre el mundo físico, logrando de esta manera crear una *realidad aumentada* en tiempo real. En este caso, la información de la cámara térmica y de datos que se quieran mostrar al bombero en cada momento se muestran en tiempo real mientras este realiza su trabajo sin obstruir la visión de lo que ocurre en su entorno.

La realidad aumentada es diferente de la realidad virtual porque sobre la realidad material del usuario se añade información visual generada por la tecnología, por tanto, el usuario percibe una mezcla de las dos realidades al mismo tiempo. Por otra parte, en la realidad virtual, el usuario se aísla del mundo físico para "sumergirse" en un escenario o entorno totalmente virtual. En este caso, es necesario que la realidad física siga presente en todo momento y no se pierda detalle de esta, por lo que ver ambas imágenes a la vez es una gran ayuda para los bomberos.

Con la ayuda de la tecnología, la información sobre el mundo real en torno al usuario se convierte en interactiva y digital. Esta información sobre el entorno y los objetos puede ser almacenada y recuperada como una capa de información que se muestra al usuario sobre su realidad. (2)

Este tipo de arquitectura de procesado de visión extiende su uso a lo industrial, el mercado de consumo, medicina. Poco a poco se consolidará una transición desde unos procesadores de aplicación de propósito general a unas unidades de procesado cada vez más personalizadas y con mayor rendimiento, no solo teléfonos móviles.

Todavía hay una gran distancia que recorrer, tanto en crecimiento del mercado como en la optimización de los costes, lo cual permite que los dispositivos sean más accesibles por el consumidor y que este tipo de mercado tenga una adopción más amplia. Las especificaciones de la MIPI (Mobile Industry Processor Interface) Alliance de la cual se hablará más adelante, se han adoptado en un gran volumen del mercado móvil y están ocupando su lugar en otras aplicaciones, haciendo uso de las economías de escala: coste, consumo, simplicidad, escalabilidad, y atributos más amigables, así como del gran ecosistema de dispositivos validados en producción.

Con el crecimiento esperado en el campo de los sensores, cámaras e interfaces para displays y procesadores visuales de alto rendimiento computacional, parece bastante claro que los interfaces como el CSI (Camera Serial Interface), DSI (Display Serial Interface) e I3C (Improved Inter Integrated Circuit) continuarán siendo los interfaces dominantes en el mundo de los móviles y demás dispositivos. (3)

3.3 Componentes

La base del proyecto consta de tres componentes necesarios para su funcionamiento: Una cámara térmica, un microcontrolador y unas gafas de realidad aumentada.

La cantidad de sensores, y las funcionalidades que se añadan a posteriori pueden variar en función de las necesidades, sin embargo, estos tres elementos son fundamentales para el desarrollo del proyecto. Por supuesto, las marcas y modelos comentados en este texto pueden variar en función de costes y características, pero el conjunto siempre será el mismo.

3.3.1 Cámara térmica

La cámara térmica utilizada es el modelo Lepton de la marca FLIR. Este componente es un sensor infrarrojo de onda larga muy compacto con una resolución de 80x60 pixeles. Está desarrollado para equiparse a móviles y dispositivos de nueva generación para proporcionar imágenes térmicas. Esta cámara se coloca en la parte frontal del casco del bombero de manera que la cámara vea aproximadamente la misma parte de la escena que sus ojos para dar una visión térmica de lo que éste mira.

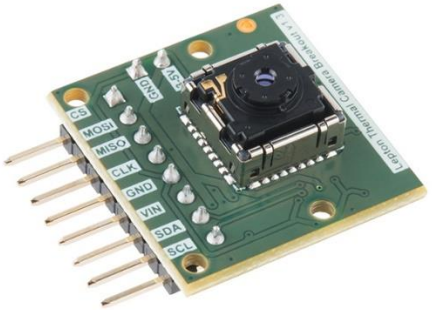


Figura 3. Cámara térmica Lepton.

https://www.sparkfun.com/products/13233

3.3.2 Raspberry Pi

Raspberry Pi es un ordenador de tamaño pequeño y coste reducido desarrollado en Reino Unido con un objetivo educativo. Sus características más importantes son su precio, así como el de sus componentes, y la posibilidad de utilizar software libre de código abierto. Es ideal para comenzar a desarrollar proyectos, pues tiene multitud de interfaces para realizar conexiones y el software generalmente es compartido por otros usuarios que ya se han enfrentado a él antes.

Figura 4. Raspberry Pi. *https://en.wikipedia.org/wiki/Raspberry_Pi*

3.3.3 Epson Noviero

Para la visualización de la realidad aumentada, actualmente ya existe una gran variedad de productos capaces de superponer imágenes y texto a la realidad que el usuario ve. Las gafas utilizadas son las Epson Moverio BT-100, las cuales, a diferencia de las Google Glasses, cubren toda la superficie del ojo con el cristal, por lo que no es necesario desviar la mirada de lo que se está haciendo. Dichas gafas, se controlan a través de un dispositivo táctil que se asemeja a un *Smartphone*, pero con tamaño más reducido, el cual se utiliza para navegar por los distintos menús del software de las Epson Moverio.

Figura 5. Gafas de realidad aumentada Epson Moverio BT-100.

http://www.appleros.net/nuevas-smartglasses-de-epson/

3.3.4 Proyecto a desarrollar

Actualmente, el funcionamiento del sistema es el siguiente: La cámara térmica controlada por la Raspberry Pi capta las imágenes térmicas de su alrededor ofreciendo una visión a través del humo, en lugares oscuros y en zonas con poca visión. Estos datos son enviados a la Raspberry que se encargará de transmitirlos a las gafas de realidad aumentada para que proyecten dichas imágenes sobre la realidad tangible que el usuario está viendo. Las gafas funcionan a través de tecnología WIFI, por lo que se equipa un dongle de WIFI en la Raspberry para ofrecer la conectividad entre las gafas y el microcontrolador.

Así pues, el proyecto que se documenta en esta memoria trata de ofrecer una serie de mejoras sobre la base comentada anteriormente. Por una parte, como las gafas están conectadas a un dispositivo controlador a través de un cable, y paralelamente se está utilizando una Raspberry Pi para el envío de las imágenes de la cámara térmica, parece lógico prescindir del dispositivo controlador y realizar todo el manejo de las gafas a través de la Raspberry, por lo que la primera parte del proyecto será el estudio de las posibilidades que existen para esta conexión, es decir, prescindir de la conexión inalámbrica, de menor fiabilidad, por una conexión cableada directa entre las gafas y la Raspberry Pi.

Por otra parte, se van a introducir sensores para la monitorización de la temperatura a la que está sometido el bombero, su frecuencia cardíaca para controlar posibles ataques de ansiedad, pánico o estrés, la concentración de gas, por si accede a un habitáculo en el que haya una fuga de algún gas contaminante o altamente inflamable y la detección de posibles caídas del bombero, haciendo que sea posible alertar a los compañeros de que ha habido algún problema.

Capítulo 4. Especificaciones MIPI

Como se ha comentado durante los capítulos anteriores, paralelamente a la incorporación de los sensores, se ha realizado una pequeña investigación en la que se busca optimizar la parte de comunicación entre la Raspberry Pi y las gafas.

La cámara térmica capta las imágenes que son procesadas por el microcontrolador y son enviadas a las gafas de realidad virtual. Sin embargo, esta conexión entre el microcontrolador y las gafas actualmente se realiza a través de una conexión WIFI, que es menos fiable que una conexión por cable. De esta forma, los datos se transmitirían de forma más rápida y segura, y sin riesgo de pérdida de paquetes. Las gafas Epson Moverio utilizan un cable para realizar la conexión a un dispositivo táctil que permite su manipulación y el desplazamiento por el menú gráfico. Así pues, la idea es averiguar cómo se realiza la comunicación entre las gafas y el dispositivo. De esta idea surgen diferentes preguntas. La primera es, ¿cómo podría realizarse esa conexión cableada entre la Raspberry Pi y las gafas? Aunque no es una pregunta trivial, si se diese con la respuesta de esa cuestión, aparecerían otras preguntas como: ¿cuál es el protocolo de comunicación que se utiliza?, ¿qué tiempos de margen tienen los datos para llegar al receptor?, ¿qué cabeceras se envían si se quiere enviar una única imagen?, etc.

4.1 Interfaces físicos

La Raspberry Pi cuenta con un conjunto de entradas y salidas. Aquí se comentan todos los conectores disponibles para la conexión de periféricos. Para conocer las posibilidades de entrada y salida de datos que hay al alcance, conviene estudiarlos brevemente. (4)

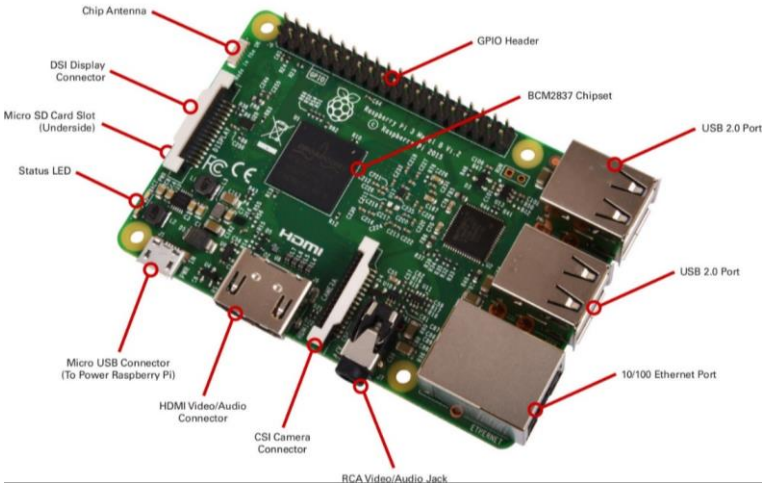


Figura 6. Interfaces físicos de la Raspberry Pi.

https://jeffskinnerbox.wordpress.com/2012/12/05/raspberry-pi-serial-communication/

4.1.1 General Purpose I/O

En primer lugar, la Raspberry dispone de los pines GPIO. Estos pines son el interfaz físico entre la Pi y el mundo exterior. En el ámbito más simple, estos pines digitales son como interruptores que pueden enviar y recibir señales de ON y de OFF. Además, contiene pines para realizar una conexión con componentes externos a través de UART, I2C, SPI o TTL. (5)

Todos estos protocolos mencionados anteriormente son protocolos serie, es decir, que las conexiones son multiplexadas en el tiempo (6):

- **Universal Asynchronous Receiver Transmitter** (UART). Son los pines TXD y RXD de la Raspberry. Es uno de los protocolos serie más usados. Utiliza una única línea para transmitir y otra para recibir. Normalmente se transmiten 8 bits de datos del siguiente modo: 1 bit de comienzo a nivel bajo, 8 bits de datos transmitidos y un bit de parada a nivel alto. El bit de comienzo a nivel bajo y el de parada a nivel alto tienen como propósito que siempre haya un flanco de bajada al comenzar una comunicación. No hay nivel de voltaje, se puede utilizar 3.3 o 5V. Los microcontroladores que se comuniquen por UART deben acordar la velocidad de transmisión, ya que solamente existe el flanco de bajada del bit de comienzo para realizar la sincronización. A esto se le llama comunicación asíncrona. Para comunicaciones a distancias más largas, los 5V dejan de ser fiables, por lo que se utiliza un voltaje mayor. Típicamente -12V para un 0 y +12V para un 1. Esta comunicación se llama RS-232. La dependencia del tiempo del UART es un gran hándicap, por lo que una solución es el USART, Universal Synchronous/Asynchronous Receiver-Transmitter, el cual, además de datos, contiene un reloj, por lo que se necesita más ancho de banda.

- **Serial Peripheral Interface** (SPI). Este protocolo envía una señal de reloj, y en cada flanco de subida envía un bit al esclavo a la vez que recibe un bit de éste. Las señales utilizadas, y por consecuencia el nombre de los pines asociados, son SCK para el reloj, MOSI para Master Out Slave In, y MISO para Master In Slave Out. Utilizando SS, Slave Select, el maestro puede controlar más de un esclavo en el bus.

- **Inter-Integrated Circuit** (I2C). Es el primer protocolo síncrono en el que se ve algo de "inteligencia". Mientras que los otros protocolos simplemente envían y reciben bits, I2C utiliza solamente 2 buses: el del reloj SCL y el de datos, SDA. Esto que significa que el maestro y el esclavo envían datos por el mismo bus. Dicho bus es controlado por el maestro, que es el que crea y transmite la señal de reloj.
  I2C no utiliza selección de esclavo como SPI, sino que cada dispositivo conectado tiene una dirección. El primer byte enviado por el maestro contiene 7 bits de dirección y un bit de escritura/lectura, indicando si el siguiente byte o bytes llegará del esclavo o del maestro. Después de cada byte, el receptor enviará un 0 indicando la recepción de dicho byte. Más adelante, se explicará esta tecnología con más detalle ya que se va a utilizar para realizar la mayoría de conexiones con los sensores.

- **Transistor Transistor Logic** (TTL). No es un protocolo, sino que es una tecnología más antigua utilizada para la lógica digital. No se abordará su funcionamiento en este documento.

4.1.1.1 PINOUT

Para tener un punto de referencia a la hora de las conexiones con los pines GPIO, aquí se muestra una imagen de la numeración de los distintos pines del modelo de Raspberry utilizado:

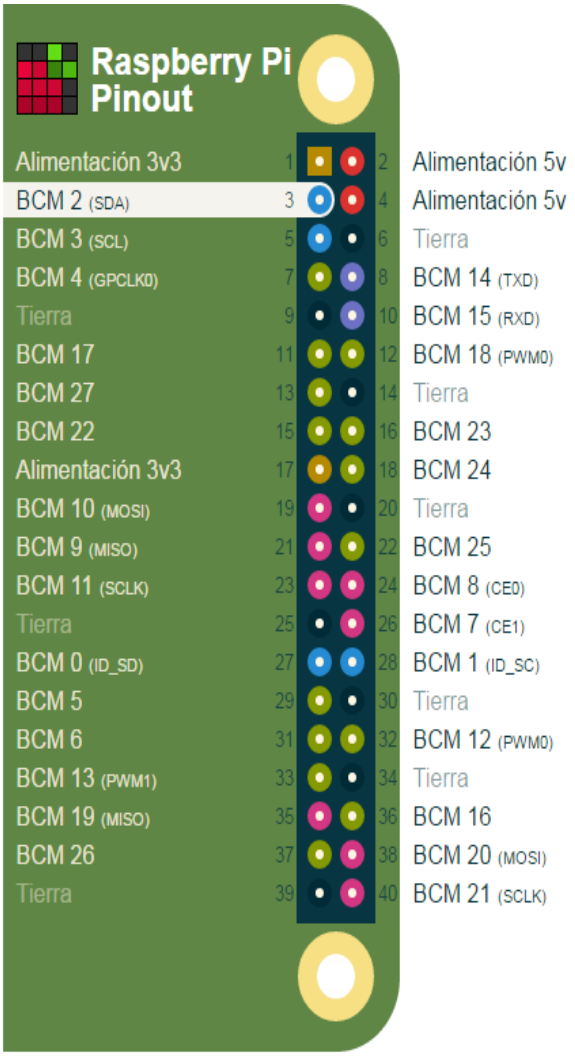


Figura 7. Esquema de los pines GPIO.

http://radioprog.ru/post/150

4.1.2 *Otros interfaces*

Además de los pines de propósito general, la Raspberry Pi cuenta con distintos interfaces, la mayoría habituales en cualquier ordenador:

- **Salida HDMI de audio y video**. Este interfaz soporta el formato 1080P HD y da la posibilidad de manejar cualquier ratio de display con gran calidad de audio. Esta característica hace a la Raspberry Pi una buena máquina base para diseñar un dispositivo para juegos o para multimedia.

- **Conector Jack de 3.5mm para sonido estéreo y video compuesto**. Ideal cuando no se dispone de un monitor HDMI. Es de aspecto cilíndrico y normalmente tiene de dos, tres, cuatro y, recientemente, hasta cinco contactos. Si cuenta con dos contactos, es para una conexión monofónica, si tiene tres o más es para una estereofónica o monofónica balanceada. La versión de tres contactos se conoce como TRS. La T viene de Tip, que significa la punta, y lleva la señal activa o el canal izquierdo. El anillo, Ring, porta la señal balanceada o canal derecho, y el cuerpo, Sleeve, es la tierra o masa.

- **Conector Ethernet RJ-45.** El RJ-45 es un interfaz físico comúnmente utilizado para conectar redes de computadoras con cableado estructurado. Posee ocho pines o conexiones eléctricas que normalmente se usan como extremos de cables de par trenzado.
  El tipo de cable puede ser directo o trenzado. El cable directo de red sirve para conectar dispositivos desiguales, como un computador y un Hub o Switch. En este caso, ambos extremos del cable deben tener la misma distribución. El cable cruzado es un cable que interconecta todas las señales de la salida en un conector con las de entrada en el otro conector, y viceversa, permitiendo a dos dispositivos electrónicos conectarse entre sí con una comunicación full dúplex. También permite una transmisión confiable vía una conexión Ethernet. Este interfaz puede utilizarse para realizar conexiones LAN como una alternativa a la conexión inalámbrica WIFI.

- **Conector para cámara CSI.** La Raspberry Pi cuenta con un interfaz serie de cámara de tipo 2 (CSI-2) diseñado por la MIPI Alliance. Este interfaz facilita la conexión de una pequeña cámara al procesador Broadcom BBCM2835 de la Raspberry proporcionando una conexión a través de un bus eléctrico entre los dos dispositivos.

  El propósito de este interfaz fue el de estandarizar el conectar módulos de cámaras a procesadores para la industria móvil. La versión CSI-2 es extremadamente popular y está presente en casi todos los teléfonos móviles y dispositivos a nuestro alrededor.

  Al incrementar la resolución de las imágenes, la necesidad de ancho de banda de transmisión desde la cámara al procesador también crece. La especificación para el CSI-2 de la Alianza MIPI soluciona la mayor parte de problemas que surgen al querer transmitir grandes cantidades de datos al procesador. (7)

- **Conector para display DSI.** Para conectar una pantalla pequeña o incluso una pantalla táctil directamente a la Raspberry Pi en proyectos que lo requieran.

  La Raspberry Pi también cuenta con un interfaz de display serie (DSI) para conectar un display de cristal líquido (LCD), utilizando un cable cinta de 15 pines. El procesador de la Raspberry inyecta los datos gráficos directamente al display a través de este conector. El conector DSI ofrece un interfaz rápido de alta resolución dedicado a enviar señales de video directamente desde la GPU a un display compatible.

Figura 8. Conector DSI de la Raspberry Pi.
https://www.petervis.com/Raspberry_PI/Raspberry_Pi_LCD/Raspberry_Pi_LCD_DSI_Display_Connector.html

Conectar un LCD a través de los pines GPIO sería fútil, pues su rendimiento no sería ni tan rápido ni tan bueno como la conexión serie que ofrece el DSI. Para conectar un LCD a este conector se necesitan tres cosas: el pinout del conector donde se muestren las conexiones eléctricas, un display compatible con DSI y el driver del display para la GPU.

Aquí se muestra la configuración de pines del DSI:

Pinout Conector DSI	
Pin	Función
1	GND
2	Línea de datos 1N
3	Línea de datos 1P
4	GND
5	CLK N
6	CLK P
7	GND
8	Línea de datos 0 N
9	Línea de datos 0 P
10	GND
11	-
12	-
13	GND
14	+3.3V
15	+3.3V

Cada conexión consiste en un par de pines que llevan la parte positiva y negativa de la señal. Esto se conoce como señalizado diferencial, y es un aspecto común en los interfaces de MIPI.

El voltaje utilizado normalmente es del orden de 200mV, que es extremadamente pequeño con el propósito de realizar el menor consumo de batería posible. El interfaz DSI permite la utilización de hasta 4 canales, sin embargo, aquí solamente están implementadas la pista 0 y 1 con un único reloj común dividido en dos, la parte positiva y la negativa. Como se explicará más adelante, todas las conexiones son unidireccionales desde el procesador al dispositivo, excepto la línea 0 que puede invertir su dirección, por lo que podría introducirse una pantalla táctil para enviar señales al procesador. (8), (9)

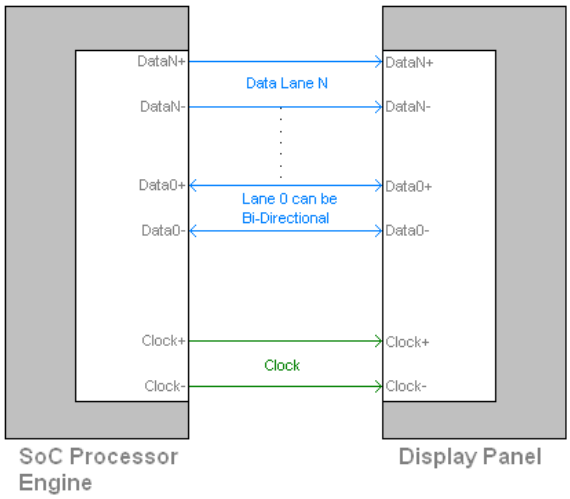


Figura 9. Esquema de conexión del procesador con el display a través de DSI.
https://www.petervis.com/Raspberry_PI/Raspberry_Pi_LCD/Raspberry_Pi_DSI_Connector.html

- **USB**. Además, cuenta con cuatro puertos USB, por lo que se puede conectar un amplio rango de dispositivos USB en un sistema basado en Linux.

Teniendo en cuenta estas posibilidades de conexión, se trata de encontrar la forma de interconectar la Raspberry Pi con las gafas, utilizando la Pi como si del dispositivo controlador táctil se tratara, enviando los bits necesarios para la interactuación con estas.

4.2 Conector DD2

En un primer acercamiento, la idea que parece más clara es la de cortar el cable que interconecta ambos dispositivos y realizar la conexión de las gafas con los distintos pines de la Pi. Una forma poco elegante pero útil en una primera instancia. Sin embargo, Tras analizar el conector y realizar varias búsquedas, se llega a la conclusión de que el conector de las gafas es de la serie DD2 de la marca JAE Electronics. (10)

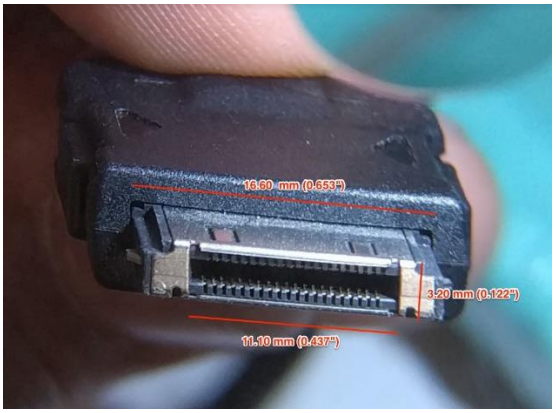


Figura 10. Conector DD2 utilizado por las Epson Moverio BT-100.
https://electronics.stackexchange.com/questions/227379/identifying-a-connector-for-an-ar-headset

Las series DD1 y DD2 son familias de conectores diseñadas para transmitir señales a alta velocidad a dispositivos móviles. El conector macho está disponible tanto como *traditional cable connecting plug*, como *cardle (dock) plug*, el cual puede ir directo a PCB. Dicho conector utiliza USB 2.0, mientras que otros conectores podrían utilizar HDMI o USB 3.0. (11)

General Specifications	
# of contacts	40 positions
Current	Signal: 0.5 A, Power: 1.0 A
Rated voltage	30 VACr.m.s.
Contact resistance (initial)	50 milliohm max.
Dielectric withstanding voltage	300 VACr.m.s. (1 minute)
Insulation resistance (initial)	1000 megohm min.
Durability mating cycles	10000 times
Operating temperature range	-25 deg. C to +75 deg. C

DD2 Series (40 positions)

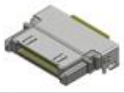
	Cable Plug	Cradle/Dock (Right angle)	Dock/Cradle (70°)	Receptacle
Picture				
Part Number	DD2P040MA1	DD2B040HA2	DD2B040VP4	DD2R040HP2

Figura 11. Conectores de la serie DD2. *http://www.jae.com/jccom/en/connectors/detail/DD2.*

El conector de la serie DD2 tiene en concreto 40 pines, de los cuales, hay unos más largos que son los de alimentación. Los demás pines los utilizarán para transmitir las distintas señales.

De este modo, habiendo identificado el conector, sería posible comprar un conector hembra al que conectar las gafas sin tener que cortar el cable.

Sin embargo, la información sobre los protocolos de transmisión, así como de las distintas señales enviadas, está cerrada al público por parte de Epson, por lo que no resultaría fácil descubrir dichas señales.

4.3 MIPI

Con la necesidad de una transmisión cada vez más rápida y de cantidades más grandes de datos, apareció la necesidad de unificar las tecnologías en un único estándar capaz de ofrecer altas tasas a bajo coste de consumo. Es aquí donde surge la alianza MIPI y sus interfaces CSI y DSI presentes en la Raspberry.

Es por eso por lo que, con gran certeza, la transmisión de los datos del dispositivo hardware a las gafas se realiza a través del interfaz DSI también disponible en la Raspberry. Esto abre una posibilidad de descubrir cómo se podría realizar la transmisión de las imágenes que capta la cámara térmica hacia las gafas sin necesidad de la utilización de la tecnología WIFI, por lo que se ha considerado oportuno asignar una parte de este trabajo a investigar sobre estos estándares,

protocolos e interfaces con el fin de hallar conclusiones sobre la comunicación que, al parecer, la gran mayoría de dispositivos realiza mediante estas especificaciones.

MIPI Alliance o sencillamente MIPI (Mobile Industry Processor Interface o Interfaz de Procesador de la Industria Móvil) es una organización global, de afiliación abierta, que desarrolla las especificaciones de interfaz para el ecosistema móvil incluyendo las industrias de su área de influencia y que fue fundada en 2003 por ARM, Intel, Nokia, Samsung, Texas Instruments y STMicroelectronics.

La organización cuenta con más de 250 miembros en todo el mundo, 12 grupos de trabajo activos y ha entregado más de 45 especificaciones dentro del ecosistema móvil en la última década. Entre sus miembros, incluye fabricantes de teléfonos móviles, fabricantes de equipos originales de dispositivos, empresas de semiconductores, proveedores de software, desarrolladores de procesos y de aplicaciones, proveedores de herramientas de prueba de IPs, y otras compañías fabricantes de equipos, de cámaras, y de tabletas, así como los fabricantes de portátiles. Aunque Seiko Epson Corp. no es un miembro de esta organización, posee una ID de fabricante. Esto significa que, aunque no tenga acciones en stock sobre la organización, sí que se encuentra registrada como fabricante que hace uso de sus estándares.

Los estándares MIPI son agnósticos a interfaces de aire o telecomunicaciones inalámbricas estándar. No tocan las interfaces vía radio ni las normas de telecomunicaciones inalámbricas dado que las especificaciones MIPI se refieren únicamente a los requisitos de la interfaz entre el procesador de aplicaciones y los periféricos como pudieran ser las características de señalización o protocolos. No se estandarizan procesadores de aplicaciones o periféricos concretos.

La misión de la MIPI Alliance es el desarrollo de la tecnología de las interfaces de bajo consumo de energía mediante el establecimiento, promoción y apoyo de los estándares de hardware y software de interfaces para el beneficio de las industrias móviles y su área de influencia. La organización promueve y fomenta activamente la adopción de estas especificaciones en toda la cadena del sector. Su alcance es llegar a desarrollar el más completo conjunto de especificaciones de las interfaces para los productos móviles y su área de influencia.

Las especificaciones MIPI proporcionan soluciones de interfaz para dispositivos móviles. A medida que el ecosistema móvil tradicional se ha ampliado para incluir las tabletas y los ordenadores portátiles, las especificaciones de la MIPI Alliance se han ido expandiendo hacia todo tipo de electrónica: tabletas, ordenadores, cámaras, electrónica industrial, realidad aumentada, incluso el mundo de la automoción y el de las tecnologías médicas. (12)

Los productos que utilizan las especificaciones MIPI tienen distintas características diferenciadoras, y al utilizar productos que comparten interfaces comunes de MIPI, la integración del sistema se hace menos compleja. Estas especificaciones pueden aplicarse para interconectar un gran abanico de componentes: desde el modem, antena, o el procesador de aplicaciones, a la cámara, el display, sensores y otros periféricos. Los fabricantes pueden utilizar dichas especificaciones para optimizar el rendimiento, simplificar el proceso de diseño, reducir costes de desarrollo, crear economías de escala para sus diseños y reducir los tiempos de mercado de sus productos. (13)

Actualmente todas las compañías de la industria móvil han adoptado al menos una de las especificaciones de la *MIPI Alliance*. Cualquier Smartphone utiliza como mínimo un interfaz MIPI y las mismas soluciones son utilizadas comúnmente para tabletas y portátiles. Las compañías en muchos mercados verticales están adoptando también estas especificaciones para presentes y futuros proyectos. Por ejemplo, los coches más modernos desarrollados por las industrias automovilísticas, o los dispositivos portables ofrecidos por compañías de deportes y healthcare ilustran un gran rango de dispositivos y ecosistemas de negocios que se benefician de estas especificaciones.

Cada especificación está optimizada para conseguir cada una de las siguientes características:

- Consumo bajo para alargar la vida de la batería.
- Un gran ancho de banda para soportar aplicaciones con muchas características y un uso intensivo de datos.
- Bajas interferencias electromagnéticas (EMI) para minimizar las interferencias entre radios y subsistemas del dispositivo. (13)

MIPI actualmente ha descrito tres de especificaciones para capas físicas de alta velocidad (PHY), la M-PHY, la C-PHY y la D-PHY, para soportar un gran rango de requerimientos de las aplicaciones en los dispositivos móviles. (14)

Estas especificaciones se ofrecen como interfaces individuales para que las compañías utilicen aquellas que se ajusten a sus necesidades particulares. Los vendedores pueden combinar estos interfaces con sus propias características de alto nivel para ofrecer un valor añadido a sus productos. (13) Además, como la conectividad móvil cuenta con gran facilidad para encontrar un hueco en otras industrias, las organizaciones externas a la Alianza ya están encontrando que pueden utilizar las especificaciones MIPI para sus propias aplicaciones también. (14)

4.3.1 Interfaces definidos por MIPI

4.3.1.1 Display Serial Interface

El Display Serial Interface (MIPI DSI) define un interfaz serie de alta velocidad entre un procesador host y un módulo display. El interfaz permite a los fabricantes integrar displays que alcanzan un alto rendimiento a la vez que tienen un bajo consumo y unas interferencias electromagnéticas bajas. Además, reduce el número de pines y se mantiene la compatibilidad entre distintos vendedores. Los diseñadores pueden utilizar el DSI MIPI para facilitar un buen renderizado de imágenes y videos y soportar una transmisión de contenido estereoscópico.

El DSI ha sido adoptado por una gran cantidad de smartphones y tablets, además de portátiles e híbridos portátil/tablet, así como en la industria automovilística y aplicaciones de realidad virtual/aumentada.

El DSI de MIPI opera en el D-PHY, una de las capas físicas antes comentadas y que se explicará más adelante, y utiliza un set de comandos definido en el MIPI DCS (MIPI Display Command Set). También incorpora el estándar DSC (Display Stream Compression) de VESA (Video Electronics Standards Association).

El Display Serial Interface tiene una versión más reciente llamada MIPI DSI-2, que ofrece soporte en la capa D-PHY y la C-PHY. (15)

En la capa física, DSI especifica un bus de señalizado diferencial punto a punto de gran velocidad. Este bus incluye una pista para el reloj de gran frecuencia y una o más pistas para la transmisión de datos. Cada pista se compone de dos líneas de transmisión, debido al señalizado diferencial. Todas las pistas viajan del host DSI al dispositivo DSI, excepto la primera pista de datos (Lane 0), la cual es capaz de invertir la dirección de transmisión del bus. Cuando se utiliza más de una línea, estas se utilizan en paralelo para transmitir datos, donde cada bit secuencial se transmite en la siguiente pista. Si se utilizan 4 pistas a la vez, se transmiten 4 bits simultáneamente. El link opera o bien en modo bajo consumo (LP), como en modo de alta velocidad (HS).

En modo bajo consumo se deshabilita el reloj de alta velocidad y la información de la señal de reloj está embebida en los datos. En este modo, la velocidad de transmisión de datos es insuficiente para hacer funcionar un display, pero se puede utilizar para enviar información sobre la configuración y comandos.

El modo alta velocidad, activa el reloj de alta velocidad (A frecuencias de decenas de Megahercios y hasta 1 Gigahercio), que actúa como el reloj para las otras pistas. La velocidad del reloj varía dependiendo de los requerimientos del display. Aún en este modo, esta transmisión está diseñada para reducir el consumo de batería gracias a la señalización de bajo voltaje y a la capacidad de transmisión en paralelo.

El protocolo de comunicaciones describe dos sets de instrucciones. El *Display Command Set* es un set de comandos comunes para controlar el dispositivo display y su formato se especifica por el estándar DSI. Define los registros que pueden ser direccionados y cuál es su operación. Incluye comandos básicos como *sleep*, *enable*, o invertir el display. El *Manufacturer Command Set* es un espacio para comandos específicos del dispositivo cuyas definiciones se dejan a libre elección del fabricante. También incluye comandos requeridos para la programación de una memoria no volátil, fijar registros específicos del dispositivo (como la corrección Gamma), o realizar otras acciones no descritas en el estándar DSI.

El formato de los paquetes en ambos sets se especifica por el estándar DSI. Existen los Short y Long Packets. Los paquetes cortos tienen un tamaño de 4 bytes, mientras que los grandes pueden ser de cualquier tamaño, hasta 2^{16} bytes. Los paquetes se componen por un DataID, el Word Count, el ECC (Error Correction Code) y el CRC (Payload and Checksum). Los comandos que requieren leer datos de vuelta activan un evento BTA, que permite al dispositivo responder a los datos solicitados. Un dispositivo no puede iniciar la transferencia, sino que solo puede responder a las peticiones de los hosts.

Los datos de la imagen en el bus son intercalados con señales para los intervalos de vaciado horizontal y vertical. Los datos son dibujados en el display en tiempo real y no se almacenan en el dispositivo. Esto permite al fabricante diseñar displays simples sin necesidad de añadir un buffer de memoria. Sin embargo, esto también implica que el dispositivo tiene que ser continuamente refrescado a una frecuencia de unos 30 o 60 frames por segundo, o la imagen se perderá. Los datos de la imagen se envían únicamente en modo HS, transmitiéndose los comandos durante los vaciados verticales. (16)

4.3.1.2 Camera Serial Interface

Los anchos de banda para los interfaces entre el procesador y el sensor de la cámara están siendo exprimidos hasta el límite a causa de la demanda de resoluciones de imágenes cada vez más altas, mayores tonalidades de color, y mayores cadencias de fotogramas. Más ancho de banda deja de ser suficiente para los diseñadores que tienen como objetivo un rendimiento para múltiples generaciones de productos. La alianza MIPI ofrece este interfaz como un estándar robusto, escalable, de bajo consumo, alta velocidad, y coste asequible que soporta un gran rango de soluciones en cuestión de imágenes para dispositivos móviles. Este interfaz resuelve el problema del ancho de banda tanto en la actualidad como a largo plazo. (17)

El estándar CSI (*Camera Serial Interface*) es una especificación de la MIPI Alliance que define el interfaz entre una cámara digital y un procesador host. Este, fue sucedido por el CSI-2, y más tarde por el estándar de interfaz más tardío, el CSI-3, que fue publicado en 2012. (18)

La mayoría de las cámaras utilizadas en productos de consumidores en alto volumen, como los smartphones y tables, utilizan sensores basados en MIPI. El interfaz más común utilizado para este tipo de sensores de imagen es el CSI-2, consistente en un único bus físico que contiene un reloj diferencial y de uno a 4 pistas diferenciales de datos. Este parece el esquema habitual como ya se ha venido observando en el DSI. Los interfaces utilizados son el D-PHY, para mantener la compatibilidad con el estándar CSI, y el C-PHY, que solamente requiere un mínimo de 3 pines en vez de 4. Será el desarrollador el que decida si utilizar una capa física o la otra, o incluso las dos, si así se quisiera. Lo que hace único este bus es que puede cambiar en el momento de señalizado diferencial a *single-ended*. Para el modo de alta velocidad (HS) se utiliza la señal diferencial, y para el modo de bajo consumo se utiliza el modo *single-ended*. Para un alto rendimiento se utilizará el modo HS.

El interfaz CSI-2 HS trabaja eléctricamente como un dispositivo SLVS (*Scalable-low-voltage-Signaling*) estándar con 200mV de voltaje en modo común. El reloj utiliza un sincronismo DDR y el número de pistas de datos para un interfaz puede variar de 1 a 4. Cada pista de datos transmite 8-bits en serie. A mayor resolución de imagen y más cadencia de *frames*, se necesitarán más pistas.

El límite práctico para un interfaz CSI-2 es menos de 1 Gbit/s, pero normalmente la carga no llega a los 700Mbits/s. Por ejemplo, una señal de video de alta definición de 1080p60 se transmitiría con cuatro líneas diferenciales, cada una a una velocidad aproximada de 500Mbits/s.

En la mayoría de aplicaciones, el sensor de imágenes CSI-2 estará configurado a máxima potencia para transmitir los fotogramas en modo HS continuamente. A este modo se le llama modo de libre reloj o modo de alta velocidad fija. (19)

4.3.1.3 Interfaces físicos (PHY)

Para la *MIPI Alliance*, la capa física, o PHY, es la clave para la solución de interconexión. Con esto, se ofrece una familia de tres capas físicas de alto rendimiento y coste optimizado:

- MIPI D-PHY
- MIPI M-PHY
- MIPI C-PHY

Las compañías pueden aplicar las especificaciones de estos interfaces físicos para soportar una gran variedad de capas de protocolos y de aplicaciones para smartphones, tablets, automóviles, weareables, etc.

El interfaz MIPI D-PHY se utiliza para interconectar cámaras y displays a un procesador de aplicación. El MIPI M-PHY soporta multimedia y comunicaciones chip a chip o chip a inter procesador. Por último, el MIPI C-PHY soporta cámaras y displays.

Como parte de la política de MIPI, todos los PHY alcanzan unos requerimientos muy rigurosos en cuanto a rendimiento, bajo consumo de operación y bajas interferencias electromagnéticas. (20)

Además, a diferencia de otros muchos interfaces, los interfaces PHY son capaces de operar en configuración simplex o dúplex, con una única línea de datos o múltiples, garantizando flexibilidad al fabricante. Además, el reloj es siempre unidireccional de maestro a esclavo, y está en fase y cuadratura con los datos.

La complejidad del diseño introduce algunos desafíos únicos a la hora de testear y validar los interfaces basados en D-PHY. El transmisor para las validaciones del D-PHY requiere un *jitter* reducido y un tiempo de análisis que se ajuste a las últimas especificaciones con tiempos mínimos a la hora del testeado. Con la inclusión de displays y cámaras de alta resolución con capacidad de captar o mostrar videos en alta definición, la cantidad de datos que se necesitan transmitir es enorme, por lo que son necesarios los protocolos de alto nivel CSI y DSI, los cuales usan estas capas físicas. (21)

4.3.1.3.1 M-PHY (v3.1, junio 2014)

M-PHY es una tecnología de interfaz serie con reloj embebido con grandes capacidades respecto al ancho de banda, específicamente desarrollado para rendimientos extremos y bajos requerimientos de consumo de aplicaciones móviles. Ha sido diseñado para la siguiente generación de interfaces punto a punto y redes componentes de gran velocidad utilizando arquitecturas simplex duales. M-PHY soporta siete protocolos diferentes, desde cámaras avanzadas hasta memorias de alta velocidad, donde una baja cantidad de pines, una escalabilidad de la pista, y una gran eficiencia de consumo son los requisitos fundamentales.

Transmitiendo en ráfagas largas o cortas, M-PHY se adapta a un gran rango de requerimientos mientras reduce el consumo. Además, opera sobre varios tipos de media, incluyendo interconexiones ópticas y dando soporte a convertidores de media. (22)

4.3.1.3.2 D-PHY (v1.2, septiembre 2014)

D-PHY es una tecnología de interfaz serie que utiliza, al igual que M-PHY, un señalizado diferencial para canales limitados en ancho de banda, con pistas de datos escalables y un reloj síncrono que ofrece inmunidad al ruido y una gran tolerancia al *jitter*. Además, soporta interfaces de consumo eficiente para aplicaciones de *streaming* como displays o cámaras por su flexibilidad, velocidad, su bajo consumo y coste. Se puede utilizar también en otros casos como el sistema de sensado de los automóviles.

Ofrece un comportamiento *half-duplex* para aplicaciones que se beneficien de una comunicación bidireccional a velocidades de transmisión de hasta 2.5Gbits por pista (22)

En concreto esta tecnología, conecta cámaras de megapíxeles y displays de gran resolución a un procesador de aplicación.

4.3.1.3.3 C-PHY (v1.0, octubre 2014)

C-PHY requiere pocos conductores y no necesita una línea de reloj separada. Además, proporciona flexibilidad para asignar pistas individuales en cualquier combinación a cualquier aplicación del procesador vía control de software. Debido a las similitudes eléctricas, C-PHY y D-PHY pueden ser implementados en los mismos pines del dispositivo, por eso el DSI permite utilizar una combinación de ambos si el diseñador así lo desea.

Este interfaz proporciona aproximadamente 2.28 bits por símbolo a través de grupos de tres líneas por pista. Esto permite velocidades de transmisión más altas a frecuencias de muestreo menores, lo que implica una reducción del consumo (22)

PHY Characteristics

Characteristic	*M-PHY v3.1*	*D-PHY v1.2*	*C-PHY v1.0*
Primary use case	Performance driven, bidirectional packet/ network oriented interface	Efficient unidirectional streaming interface, with low speed in-band reverse channel	Efficient unidirectional streaming interface, with low speed in-band reverse channel
HS clocking method	Embedded Clock	DDR Source-Sync Clock	Embedded Clock
Channel compensation	Equalization	Data skew control relative to clock	Encoding to reduce data toggle rate
Minimum configuration and pins	1 lane per direction, dual-simplex, 2 pins each (4 total)	1 lane plus clock, simplex, 4 pins	1 lane (trio), simplex, 3 pins
Maximum transmitter swing amplitude	SA: 250mV (peak) LA: 500mV (peak)	LP: 1300mV (peak) HS: 360mV (peak)	LP: 1300mV (peak) HS: 425mV (peak)
Data rate per lane (HS)	HS-G1: 1.25, 1.45 Gb/s HS-G2: 2.5, 2.9 Gb/s HS-G3: 5.0, 5.8 Gb/s (Line rates are 8b10b encoded)	80 Mbps to ~2.5 Gbps (aggregate)	80 Msym/s to 2.5 Gsym/s times 2.28 bits/sym, or max 5.7 Gbps (aggregate)
Data rate per lane (LS)	10kbps – 600 Mbps	< 10 Mbps	< 10 Mbps
Bandwidth per Port (3 or 4 lanes)	~ 4.0 – 18.6 Gb/s (aggregate BW)	Max ~10 Gbps per 4-lane port (aggregate)	Max ~ 17.1 Gbps per 3-lane port (aggregate)
Typical pins per Port (3 or 4 lanes)	10 (4 lanes TX, 1 lane RX)	10 (4 lanes, 1 lane clock)	9 (3 lanes)

Figura 12. Tabla resumen de las características de los enlaces físicos.
https://www.mipi.org/sites/default/files/PHY_Tech_Brief_20140916_0.pdf

4.3.2 Prueba con cámara y Pylibrary

Los protocolos de los interfaces de MIPI son de uso cerrado, y solamente están disponibles para los miembros de MIPI: Los fabricantes. Al no disponer de dicho protocolo, surge la posibilidad de utilizar los dispositivos de la Raspberry Pi, cuya conexión es la del interfaz DSI y CSI para tratar de obtener una fuente desde la que leer las señales que se envían o se reciben para realizar un primer acercamiento a estos estándares.

Sin embargo, para la utilización del LCD y de la PiCamera, existen librerías como la *pylibrary* que permiten su utilización, pero cuyo código es cerrado y propietario de Broadcom, por lo que, tras una búsqueda en foros y en las redes y la confirmación de tales sospechas, se abandona la idea de la utilización de los complementos de la Raspberry Pi para intentar obtener la transmisión de datos, pues la pylibrary utiliza métodos ya creados que permiten interactuar con la cámara son simples y sencillas llamadas a la clase.

4.3.3 Hallazgos en la red

Realizando la búsqueda de información que pudiera ser utilizada para la interconexión entre el conector DSI y las Epson Moverio, se encontraron varias páginas que pueden considerarse de interés.

Por una parte, se encontró un proyecto en el que, con una FPGA se realiza un adaptador de HDMI a DSI utilizando 4 pistas. Sin embargo, en este proyecto no se explica el proceso ni cómo se realiza el envío de los bits. (23) El resultado del proyecto se comercializa por 99$ y consiste en un circuito impreso que junto con la FPGA permiten transformar la señal de HDMI y enviarla a un LCD a través del protocolo DSI.

Por otra parte, se encontró un adaptador con entrada HDMI que se conecta directamente a las gafas Epson Moverio BT-200 pero cuyo precio roza los 1500$ además de tener un tamaño demasiado grande. (24)

Figura 13. Adaptador de HDMI a DSI para las Epson Moverio BT-100.

http://www.colorado-video.com/hi-def-moverio-interface/hi-def-moverio-interface.htm.

También se han encontrado multitud de foros en los que la gente pregunta sobre este tema, e incluso tiene la misma necesidad de interconectar las gafas a un controlador, sin embargo, no hay respuesta, pues el código está cerrado a usuarios.

La búsqueda más cercana consistió en un archivo de Texas Instruments trata de desmitificar el DSI realizando distintas configuraciones y capturando el impacto que tiene sobre el protocolo DSI. Además, se incluyen consejos para el debugging del protocolo. Aquí se adjunta una imagen donde se hace el envío de un comando en modo LP (25):

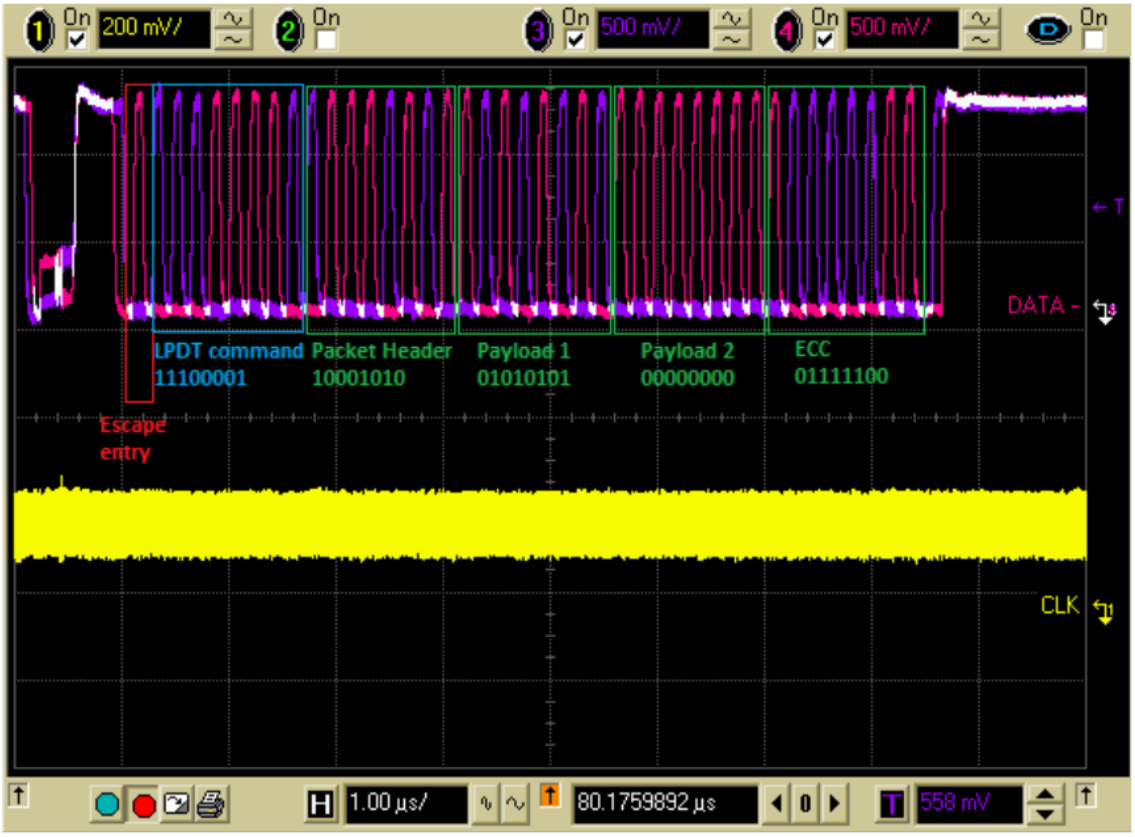


Figura 14. Señales obtenidas con osciloscopio al enviar un comando en modo LP
http://www.ti.com/lit/an/swpa225/swpa225.pdf

Como no se conoce la forma en que se interconectan los pines, ni qué significa la respuesta que se recibe, además de que se desconocen los comandos que son transmitidos en la conexión entre las gafas y el dispositivo táctil, no se puede considerar este documento como un avance definitivo, aunque es un paso importante.

4.3.4 Conclusiones sobre los interfaces de MIPI y el proyecto

En esta parte del proyecto se ha podido comprender la importancia de los estándares de la *MIPI Alliance* y se ha estrechado el cerco hacia una posible migración de una conexión WIFI a una conexión cableada. A pesar de ello, se alcanza un punto muerto en el momento que es necesario conocer al detalle el formato de transmisión de los paquetes desde el dispositivo táctil a las gafas. Para ello se requeriría de un osciloscopio, el conector DD2 comentado anteriormente y mucho tiempo de envío y recepción de señales para encontrar los patrones enviados, pues además de las imágenes, las gafas también reciben comandos e instrucciones desde el dispositivo táctil para enviarle órdenes como las de navegar por el menú, por ejemplo. Es por esto, que se ha decidido limitar el contenido de esta parte del proyecto para avanzar por otras vías como es la adquisición de datos a través de sensores para la monitorización del estado del bombero.

Capítulo 5. Monitorización de riesgos durante un rescate

La segunda parte del proyecto se basa en un desarrollo más práctico que le otorga cuerpo al trabajo. Esta parte consiste en la implementación de un software con capacidad de controlar distintos sensores a través de los diferentes puertos que tiene la Raspberry Pi para la monitorización del trabajo que realizan los bomberos y del peligro al que están expuestos en cada momento durante una operación de rescate.

5.1 Variables a evaluar

Como se ha comentado al comienzo del escrito, son varios los peligros a los que se expone el cuerpo de bomberos durante una operación de rescate o de extinción de incendios. Por ello, cabe en este proyecto tratar de minimizar esos riesgos mediante una monitorización de diferentes parámetros para que, en esas situaciones peligrosas, tanto el bombero como el mando, tenga la mayor cantidad de información posible sobre su entorno.

Para esto, se han valorado y escogido potenciales factores de riesgo que pueden ser medidos con sensores y representados como un número o porcentaje para determinar el grado de criticidad de la situación.

El objetivo es que los datos que estos sensores capturen puedan serle representados al bombero a través del cristal de las gafas para que sea consciente de su situación mientras aprovecha las ventajas de una visión térmica sin apartar el foco de visión de lo que se está haciendo, además de enviar esa información a otros compañeros que se encuentren tanto en la zona de actuación como fuera del recinto peligroso, pues un centro de mando podría querer monitorizar a los miembros desplegados y conocer su situación.

5.1.1 Temperatura

Como es obvio pensar, un bombero estará expuesto a altas temperaturas. El traje que visten les protege de las llamas y, en cierta medida, de las altas temperaturas, pero es interesante conocer la temperatura real de forma instrumental para conocer si la habitación en la que se encuentra el bombero contiene gases que se encuentran a una temperatura más alta de lo que cabría esperar. En cualquier caso, como dato representativo, la temperatura es el primer factor de riesgo que debería tenerse en cuenta, por lo que se colocará uno o varios sensores de temperatura, generalmente en partes altas del traje o en el casco. Cabe comentar que a temperaturas superiores a los 65º la circuitería no aguantaría y los circuitos se quemarían, pero que un ser humano tampoco podría aguantar dichas temperaturas. Los trajes les protegen de las llamas y las abrasiones, pero no mitigan la temperatura, por lo que no sería incompatible entremezclar las temperaturas que se alcanzan en los incendios con la electrónica.

Para calcular las temperaturas, el sensor a utilizar será un LM35, un dispositivo de circuito integrado de precisión para la medida de temperatura, cuya salida es linealmente proporcional a la temperatura Centígrada. Este sensor no requiere calibración externa y posee una precisión de ±¼°C a temperatura ambiente y ±¾°C sobre temperaturas más extremas. Es un dispositivo de bajo coste, baja impedancia de salida y un rango de operación de -55ºC a 150ºC.

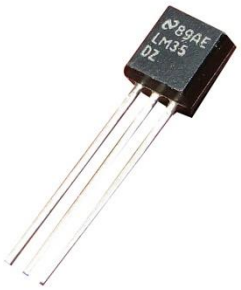


Figura 15. Sensor LM35.

https://electronilab.co/tienda/lm35-dz/

La característica más importante de este sensor es que ya está calibrado y ofrece una salida en grados Celsius de 10mV por grado, obteniéndose 1500mV a los 150ºC y -550mV a los -55ºC. (26)

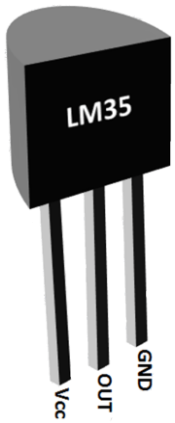


Figura 16. Esquema de los pines del sensor LM35.

https://geekytheory.com/internet-de-las-cosas-parte-1-leer-temperatura-con-arduino-y-lm35

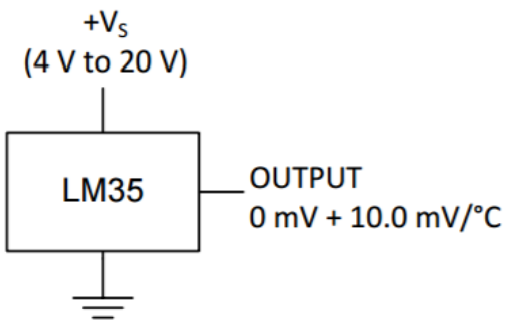


Figure 1. Basic Centigrade Temperature Sensor (+2°C to +150°C)

Figura 17. Esquema del sensor LM35 donde se indica el valor de salida que cabe esperar
http://www.ti.com/lit/ds/symlink/lm35.pdf

5.1.2 Pulso cardíaco

Durante una operación de rescate, el bombero está sometido a mucha presión y estrés, en lugares con temperaturas muy altas y en ocasiones se ven envueltos en situaciones peligrosas. Esto junto al esfuerzo físico que deben ejercer, hace que aumente el pulso cardíaco de los miembros del cuerpo. Estos factores hacen deseable tener un control de las pulsaciones por minuto de la persona que esté plena operación. Así pues, se utilizará un cinturón monitor de pulso cardíaco con tecnología Bluetooth para transmitir pulso cardíaco a la Raspberry Pi.

Para la medida del pulso, se ha optado por una banda pectoral que se utiliza para hacer deporte. Anteriormente, estas bandas solían consistir en una única banda de plástico de una pieza. Ahora, las marcas, cada vez más tienden a ofrecer una banda de tela a la que se conecta el transmisor por separado. Esto permite una mayor higiene y una ventaja a la hora de cambiar la cinta de tela o transmisor. Por ejemplo, en caso de mal funcionamiento de alguno de los dos componentes, no es necesario cambiar la cinta entera.

Básicamente, en el mercado, existen cuatro tipos de cintas de pulsómetros:

- Banda pulsómetro Bluetooth
- Banda pulsómetro ANT+
- Banda pulsómetro ANT+ y Bluetooth
- Banda pulsómetro propietaria

Bluetooth es un estándar de tecnología inalámbrica para el intercambio de datos a cortas distancias, utilizando ondas UHF en la banda ISM, de 2.4 GHz a 2.485 GHz. En concreto está diseñado para dispositivos móviles y fijos y para PANs (Personal Area Networks).

Bluetooth 4.0 + LE agrupa los protocolos del Bluetooth clásico, del Bluetooth de gran velocidad y del Bluetooth de bajo consumo. Bluetooth LE está enfocado a aplicaciones de salud, fitness, beacons y seguridad. Más adelante, se explicará más en detalle este apartado.

ANT es una tecnología de redes multicast de sensores inalámbricos propietaria, pero de libre acceso, diseñada por *ANT Wireless*. Dicha tecnología define un conjunto de protocolos de comunicación inalámbrica para 2.4GHz en la banda ISM, estableciendo una serie de normas para co-existencia, representación de datos, señalizado, autentificación y detección de errores. Es conceptualmente parecido al Bluetooth, pero orientado a la utilización de sensores.

Volviendo a los 4 tipos de bandas, las dos primeras están claras. Una banda de pulsómetro Bluetooth únicamente transmitirá el ritmo cardíaco a través de Bluetooth. Lo mismo ocurre para las cintas de pulsómetro ANT+. El tercer punto hace referencia a las cintas de pulsómetro que son compatibles tanto con ANT+ como con Bluetooth 4.0: las bandas de pulsómetro duales. que la característica principal de esta banda es que son capaces de emitir tanto en Bluetooth Smart como en ANT+. De este modo, son ideales para utilizarla con multitud de dispositivos sin tener que comprar varias. Por último, las bandas propietarias, utilizan su propio protocolo para la comunicación. (27)

La banda utilizada para la medición de las pulsaciones será la CooSpo H8, que utiliza Bluetooth 4.0 (BLE) (28)

Figura 18. Banda de frecuencia cardíaca CooSpo H8

https://www.amazon.com/dp/B06ZY242Q4

Resolución	**2 bpm**
Rango de pulsaciones	30 – 240 bpm
Transmisión	Compatible con Bluetooth4.0 (BLE)
Distancia de transmisión	Hasta 10m
Vida de la batería	12 meses

Como la banda transmite los paquetes de datos utilizando Bluetooth LE, es conveniente explorar más en profundidad esta tecnología. Más adelante ser realizará una descripción más detallada de esta tecnología para conocer su funcionamiento y la realización de la interacción entre la Raspberry y el sensor.

5.1.3 Detección de caídas o desvanecimientos

Al entrar en edificios en llamas, los bomberos encontrarán multitud de obstáculos, así como desprendimientos o colapsos que pueden dar lugar a caídas. Dichas caídas pueden ser críticas o no, pero será una buena idea detectar estas caídas y mandar una alerta a los demás compañeros para informar automáticamente de lo que ha sucedido y crear así una reacción rápida ante lo que podría ser un problema grave.

Para detectar una caída del bombero será necesario utilizar un acelerómetro, con el cual se alertará de una caída en el momento que la aceleración vertical ascienda de golpe.

El acelerómetro que se utilizará será el MPU-6050.

Este sensor contiene un acelerómetro MEMS además de un giróscopo MEMS en un único chip. Internamente, contiene un conversor Analógico/Digital de 16 bits por cada canal, pudiendo capturar los canales X, Y, y Z a la vez, utilizando un bus I2C para hacer de interfaz con el microcontrolador.

Figura 19. Sensor MPU-6050

http://www.electronoobs.com/eng_robotica_tut6_1.php

Leer valores sin procesar del acelerómetro y el giróscopo es relativamente fácil. Para ello, se debe desactivar el modo *Sleep* y leer los registros correspondientes a cada eje.

Sin embargo, el sensor contiene una memoria FIFO de 1024 bytes en el que pueden guardarse los datos para leerlos luego. Cuando se colocan datos en la FIFO, se activa una señal de interrupción para que el microcontrolador sepa que hay datos en la FIFO esperando a ser leídos.

Normalmente, el MPU-6050 actúa como esclavo, con los pines de SDA y SCL conectados al bus I2c. Sin embargo, además del bus I2C normal, el MPU-6050 tiene su propio controlador I2C para hacer de master de un segundo bus I2c. Para este escenario, se utilizarían los pines auxiliares AUX_DA y AUX_CL.

La complicación en el MPU-6050 viene con el DMP. El Digital Motion Processor, puede ser programado con firmware y es capaz de realizar cálculos complejos con los valores del sensor.

El DMP puede realizar cálculos de forma rápida en el chip, reduciendo la carga que se le aplica al microcontrolador. Además, puede realizar estos cálculos con los valores del sensor de otro chip conectado al segundo bus I2C. (29)

El giróscopo, permite conocer el ángulo de giro del sensor sobre sí mismo. Este giro se puede conocer a través de tres ángulos llamados Yaw, Pitch y Roll:

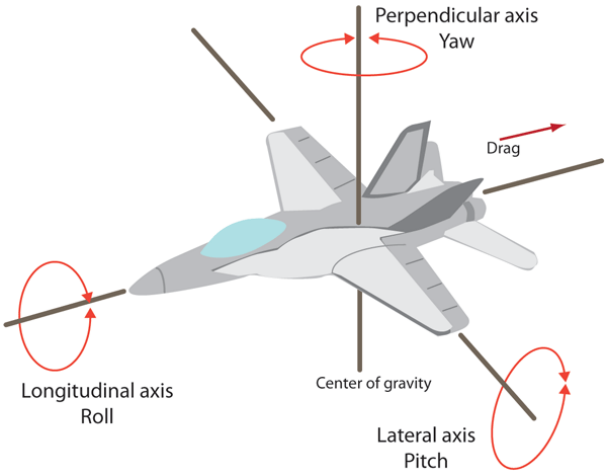


Figura 20. Esquema de los ángulos de navegación

http://www.prometec.net/usando-el-mpu6050/

Los ángulos de navegación son un tipo de ángulos de Euler usados para describir la orientación de un objeto en tres dimensiones. Estos ángulos, en aeronáutica, son equivalentes a tres maniobras consecutivas. Dado un sistema de tres ejes fijos en el aeroplano, llamados eje de guiñada (*yaw*), de cabeceo (*pitch*) y de alabeo (*roll*), existen tres rotaciones principales, normalmente llamadas igual que el eje sobre el que se producen, que permiten alcanzar el sistema del aeroplano desde el sistema de referencia.

- **Cabeceo/Pitch**: es una inclinación del morro del avión, o rotación respecto al eje formado de ala a ala.
- **Alabeo/Roll**: rotación respecto al eje morro-cola del avión.
- **Guiñada/Yaw**: rotación intrínseca alrededor del eje vertical perpendicular al movimiento avión.

Los tres ángulos corresponden a tres rotaciones intrínsecas, es decir, relativas al sistema móvil. Esto es útil, por ejemplo, en un contexto aeronáutico, cuando el piloto de un avión quiere describir una maniobra.

Los ángulos de navegación, llamados deriva (Ψ), inclinación (θ) y alabeo (ϕ), corresponden a los valores de estas tres rotaciones principales.

Aplicando la regla de la mano derecha, conocida de las clases de electromagnetismo, se coloca el pulgar apuntado en la dirección de la gravedad o z, y se ponen los dedos índice y anular a 90º entre sí y con el eje z. Utilizando uno de los dedos como eje de giro, se puede realizar la rotación de la mano para comprender los distintos movimientos de giro. Es así como el sensor determina los distintos ángulos utilizando las referencias primarias en estos tres ejes.

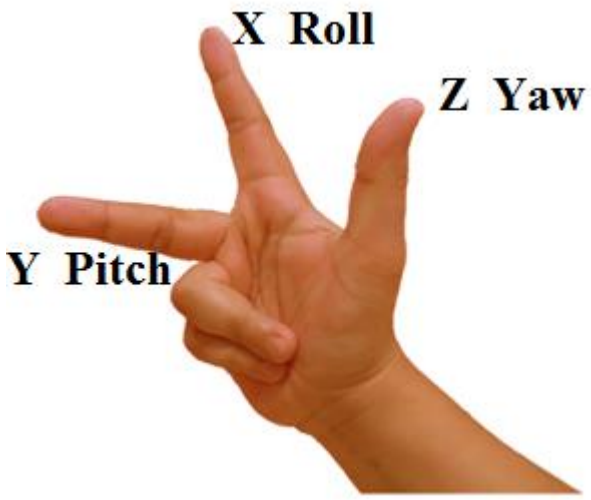


Figura 21. Ángulos de navegación obtenidos con la mano derecha

http://www.prometec.net/usando-el-mpu6050/

Para comprender el funcionamiento del sensor, cabe hablar un poco sobre física básica. El acelerómetro detectará siempre la gravedad independientemente de su posición, por lo que al comenzar el sensado, empezará a registrar valores hasta que se estabilice. Una vez estable, el sensor detectará el eje vertical porque es el que la gravedad le indica y por tanto obtiene una primera referencia del plano horizontal, que es el plano perpendicular a la gravedad. Colocando el giróscopo en horizontal, se puede observar que los ángulos de Pitch y de Roll deben tender más o menos a 0. Sin embargo, el giroscopio tiene una sensibilidad muy alta, y los valores cambian muy rápidamente. Eso significa que a la hora de determinar si ha tenido lugar una caída, el código tendrá que poder aceptar cambios en los valores dentro de un rango y detectar como caída un cambio muy radical de los valores en un corto instante de tiempo. Hay que tener en cuenta, que el bombero caerá en el eje en el que esté la gravedad y que podrá caer hacia adelante, hacia atrás o hacia los lados, por lo que hay que estudiar los diferentes escenarios para no detectar falsas caídas ni pasar por alto cuando ha ocurrido realmente una caída. (30)

5.1.4 Detección de gas

En una operación de rescate en un edificio, el bombero llevará puesta una máscara de oxígeno que le aísla del humo del entorno. Sin embargo, en ciertas ocasiones puede haber fugas de gas metano, o de gas natural, o grandes cantidades de CO2, que el bombero estaría inhalando inconscientemente.

Es por eso que sería conveniente utilizar un detector de gases nocivos que le indicara al bombero cómo de contaminado está el aire al que está expuesto o si hay grandes cantidades de algún gas inflamable en el habitáculo en el que se encuentra.

Existen varios sensores de gas de la serie MQ con distintas sensibilidades hacia ciertos componentes químicos. Es por eso, que utilizando varios de ellos a la vez, se pueden detectar los diferentes gases peligrosos que amenacen la salud o la vida del bombero.

Modelo	Gas objetivo
MQ-2	Gas combustible general
MQ-3	Alcohol
MQ-4	Gas Natural/Metano
MQ-5	LPG, Gas natural, gas carbón
MQ-6	LPG, Propano
MQ-7	Monóxido de carbono
MQ-8	Hidrógeno
MQ-9	CO y gas combustible
MQ-131	Ozono (O3)
MQ-135	Control de calidad del aire (NH3, Benceno, Alcohol, Humo)

Para un primer acercamiento, se utiliza el MQ-4, cuya sensibilidad es muy alta para el gas natural (CH4) y muy poca al alcohol o el humo, lo cual es importante, ya que, en un edificio en llamas, el humo será inevitable.

En la gráfica se muestra la sensibilidad del sensor a los distintos gases:

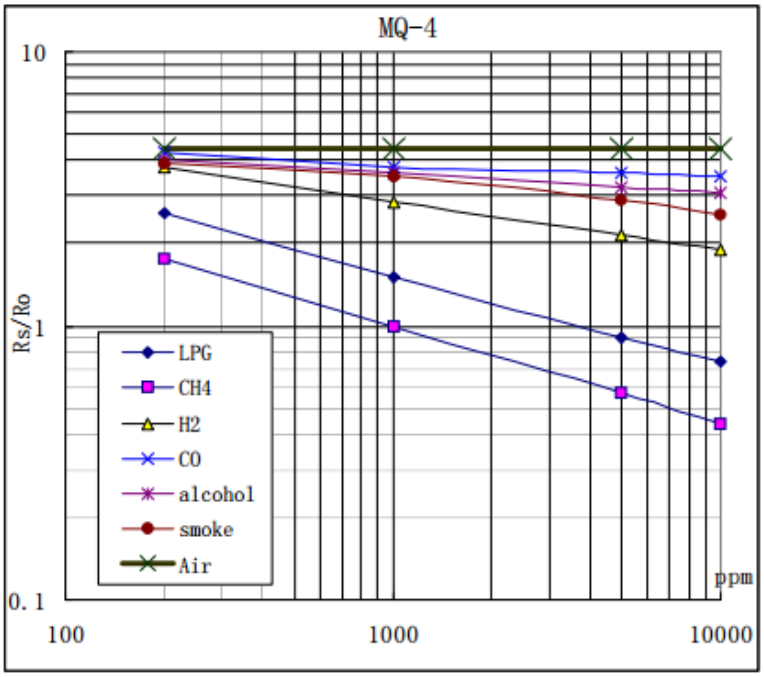


Figura 22. Gráfica donde se muestra la sensibilidad del sensor MQ-4 a distintos gases
https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-4.pdf

El material sensitivo del que está formado el sensor MQ-4, es SnO2, cuya impedancia aumenta en aire limpio y disminuye cuando la concentración de gas crece. Dicha conductividad se traduce en una mayor tensión. Con esa tensión se puede obtener el factor Rs/Ro y por tanto la concentración de gas en partes por millón.

La salida de este sensor es un único valor en el que se agrupan las distintas sensibilidades a los gases en un único dato que será la conductividad, pero otorgando un mayor peso al gas CH4.

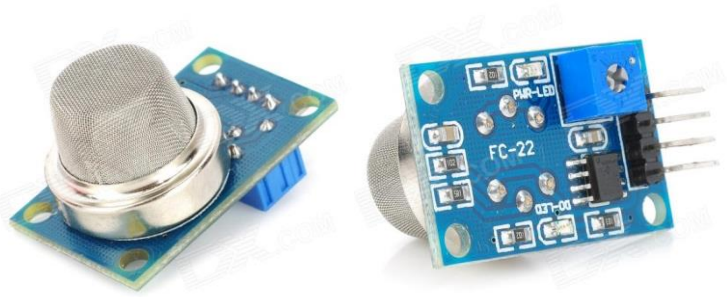


Figura 23. Sensor MQ-4 sobre placa FC-22

http://www.dnatechindia.com/MQ4-Gas-Sensor.html

El sensor está montado sobre una placa llamada FC-22. El sensor original consta de 3 pines: Alimentación, masa y salida analógica. El FC-22 agrega un pin adicional D0 que envía señales digitales a través de TTL, aunque no hay mucha información al respecto. La alimentación del sensor es de 5V.

En todas las fuentes de información halladas, se explica que antes de medir con precisión, el sensor debe calentarse durante al menos 20 segundos, y que la placa alcanza unas temperaturas altas cuando está en funcionamiento. (31)

SENSITVITY ADJUSTMENT

Resistance value of MQ-4 is difference to various kinds and various concentration gases. So,When using this components, sensitivity adjustment is very necessary. we recommend that you calibrate the detector for 5000ppm of CH_4 concentration in air and use value of Load resistance (R_L) about 20K Ω (10K Ω to 47K Ω).

When accurately measuring, the proper alarm point for the gas detector should be determined after considering the temperature and humidity influence.

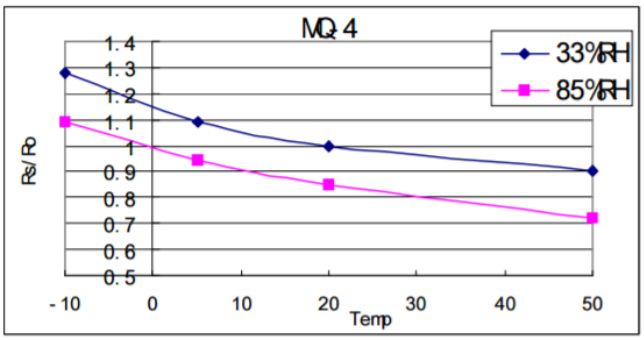


Fig.4 is shows the typical dependence of the MQ-4 on temperature and humidity.
Ro: sensor resistance at 1000ppm of CH_4 in air at 33%RH and 20 degree.
Rs: sensor resistance at 1000ppm of CH_4 in air at different temperatures and humidities.

Figura 24. Dependencia del sensor con la humedad

https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-4.pdf

5.2 Conversión de datos

Tanto el sensor de temperatura como el de gas, transmiten la señal de salida de forma analógica. Sin embargo, a diferencia de los microcontroladores Arduino, los pines GPIO de la Raspberry Pi son únicamente digitales, por lo que no se puede realizar una lectura de datos de forma directa.

Para medir las temperaturas a las que está expuesto el bombero y la concentración de gases nocivos que hay a su alrededor, es necesario utilizar un conversor ADC que transformará la señal analógica en una señal digital que se enviará en forma de bits y que la Raspberry leerá a través del pin de I2C, al igual que ocurre con el sensor de caídas, que ya tiene un conversor integrado y envía la señal directamente con este protocolo.

El conversor analógico-digital utilizado será el ADS115, que es el que se ha ido recomendando para la Raspberry Pi en multitud de foros, principalmente por su relación prestaciones/coste.

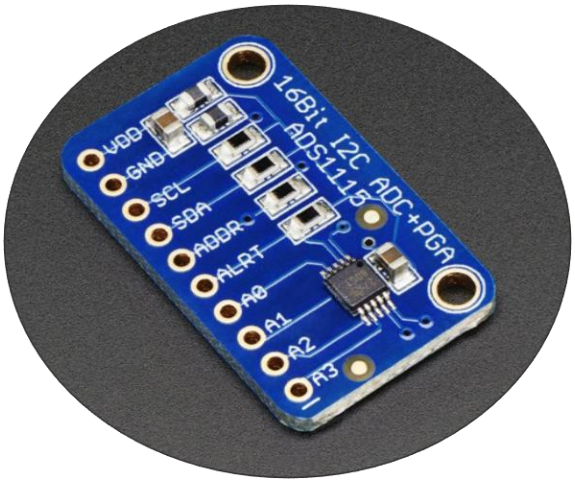


Figura 25. Conversor Digital/Analógico ADS1115

https://www.adafruit.com/product/1085

Este ADC proporciona 16 bits de precisión a una velocidad de 860 muestras/segundo a través de I2C. El chip se puede configurar como 4 canales de entrada single-end, es decir, comparando la tensión con masa, o 2 canales diferenciales.

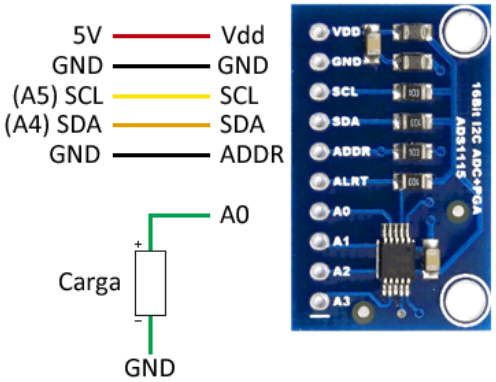


Figura 26. ADS1115 en modo single-end

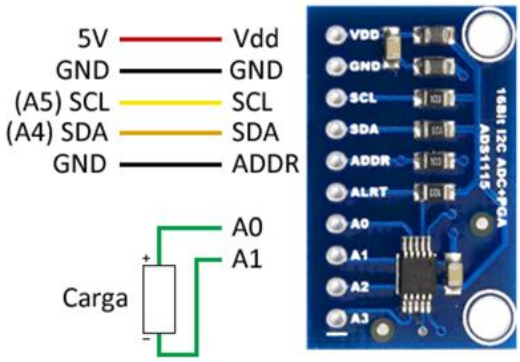


Figura 27. ADS1115 en modo diferencial

https://www.luisllamas.es/entrada-analogica-adc-de-16-bits-con-arduino-y-ads1115/

También incluye un amplificador de ganancia variable programable de hasta x16, para aumentar pequeñas señales diferenciales. Además, este ADC admite entradas desde 2V hasta 5V.

Todos los chips ADS tienen la dirección base de 7 bits para I2C 0x48 por defecto, pero utilizando el pin ADR pueden obtenerse hasta 4 direcciones distintas:

- 0x48 (1001000): Se obtiene conectando ADR a GND o dejando el pin al aire
- 0x49 (1001001): Conectar ADR a VDD
- 0x4A (1001010): Conectar ADR a SCA
- 0x4B (1001011): Conectar ADR a SCL (32)

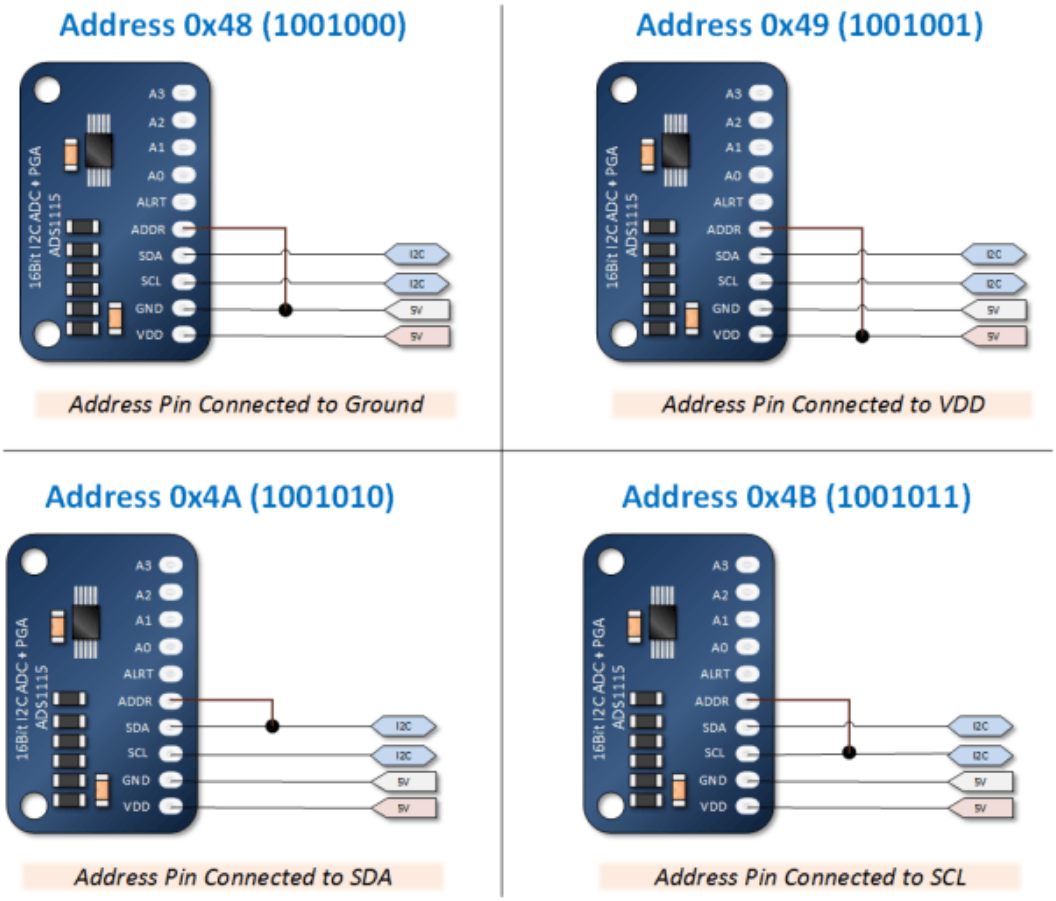


Figura 28. Conexiones del ADS1115 para obtener las diferentes direcciones

http://www.bristolwatch.com/rpi/ads1115.html

Para abordar la lectura de los datos que provienen del bus I2C, es conveniente profundizar un poco en esta tecnología para conocer la manera de leer los datos y de interpretarlos.

Más adelante, se procederá a explicar las distintas acciones y procedimientos empleados para la programación de la Raspberry Pi para la obtención de las señales, así como la interpretación de los datos recibidos.

5.3 Bluetooth Low Energy

El Bluetooth Low Energy (BLE), también conocido como Bluetooth Smart, es un subconjunto de bajo peso del Bluetooth clásico y se introdujo como parte de las especificaciones de Bluetooth 4.0. Aunque hay un pequeño solape con el Bluetooth clásico, BLE tiene un linaje totalmente distinto, y fue desarrollado por Nokia como parte de un proyecto llamado Wibree antes de ser adoptado por Bluetooth SIG.

Existe una gran cantidad de protocolos inalámbricos para ingenieros y diseñadores de productos, sin embargo, lo que hace BLE tan utilizado es la sencillez con la que se puede diseñar un dispositivo que pueda hablar con cualquier plataforma móvil moderna, además de consumir menos potencia, necesita menos tiempo y esfuerzo para conectar un par de dispositivos, aunque luego posee una menor velocidad de conexión.

La pila de protocolos de Bluetooth se divide en dos categorías: el controlador y el host. Cada categoría tiene sub-categorías, que realizan roles específicos. Las dos subcategorías a destacar son el Perfil de Acceso Genérico (GAP) y el Perfil de Atributo Genérico (GATT).

- GAP define la topología general de la pila de red de Bluetooth LE.
- GATT describe en detalle como los atributos (datos) son transmitidos una vez los dispositivos han establecido una conexión.

El Perfil de Acceso Genérico controla las conexiones y el intercambio de paquetes en Bluetooth. GAP es lo que hace a un dispositivo visible al mundo exterior y determina cómo dos dispositivos pueden, o no pueden, interactuar entre ellos.

GAP define varios roles para los dispositivos, pero los dos conceptos claves a tener en cuenta son los tipos de dispositivos Centrales o los Periféricos.

- Dispositivos Periféricos. Son pequeños, de bajo consumo y recursos limitados que pueden conectarse a un dispositivo central mucho más potente. Los dispositivos periféricos son dispositivos como el pulsómetro.
- Dispositivos Centrales. Normalmente son los teléfonos móviles o tablets que el usuario conecta y que poseen más potencia de procesado y memoria.

Hay dos mecanismos que un dispositivo BLE puede utilizar para comunicarse con el mundo exterior: mediante un broadcast o mediante el establecimiento de una conexión entre dos nodos. Estos mecanismos están sujetos a las directrices del Generic Access Profile. (33)

5.3.1 Conexión

Un dispositivo puede unirse a una red BLE adoptando los roles especificados en el Perfil de Acceso Genérico.

- Broadcasting: Estos roles no tienen que conectarse explícitamente a otro dispositivo para transmitir datos.
    - Broadcaster: Dispositivo que realiza un broadcast de paquetes públicos como pudiera ser el tiempo que ha estado pulsado un botón.
    - Observador: Dispositivos que escuchan a los paquetes de datos enviados por el broadcaster. No hay una conexión entre el transmisor y el receptor.

- Connecting: Estos roles deben realizar una conexión y una sincronización explícitas para la transmisión de datos. Estos roles son más comúnmente utilizados que los de broadcasting.

    - Periférico: Dispositivo que comunica su presencia para que los dispositivos centrales puedan establecer conexión. Una vez establecida dicha conexión, los periféricos dejan de transmitir datos a otros dispositivos centrales, y permanecen conectados al dispositivo que ha aceptado la petición de conexión. Los periféricos son dispositivos de bajo consumo porque solo tienen que enviar señales (beacons) periódicamente.

    - Central: Es el dispositivo que inicializa la conexión con el periférico comenzando por escuchar a los paquetes broadcast que este envía.

      Un dispositivo central se puede conectar a más de un dispositivo periférico.

      Cuando el central quiere conectarse, envía una petición de conexión al periférico. Este la acepta y se establece la conexión.

En este caso, la Raspberry actuará como un dispositivo central, y el pulsómetro como un dispositivo periférico.

5.3.2 Proceso de Avertising

Hay dos maneras de darse a conocer con GAP. El *Advertising Data Payload* y el *Scan Response Payload*.

Ambas cargas son idénticas y contienen 31 bytes de datos, pero solo el *payload* de *advertising* es obligatorio, ya que este será constantemente transmitido hacia otros dispositivos para permitir a los dispositivos centrales dentro del rango que conozcan que el dispositivo existe. El *Scan Response Payload* es opcional y lo demanda el dispositivo central, lo que permite a los diseñadores de dispositivos añadir algo de información, como pudiera ser el nombre del dispositivo.

Aquí se explica cómo funciona el proceso en el que un dispositivo se da a conocer y como el *Advertising Data Payload* y el *Scan Response Payload* funcionan.

El periférico tiene un determinado tiempo de *advertising*. Cada vez que este intervalo de tiempo vence, el dispositivo retransmitirá un paquete de broadcast a todos los dispositivos centrales dentro de su rango. Un intervalo de tiempo más grande ahorrará energía, pero dará la sensación que el dispositivo no responde si se da a conocer cada 2 segundos en vez de cada 20ms.

Si un dispositivo dentro del rango está interesado en el *Scan Response Payload* puede, opcionalmente, pedir al periférico que se lo envíe, con lo que el periférico responderá con información adicional.

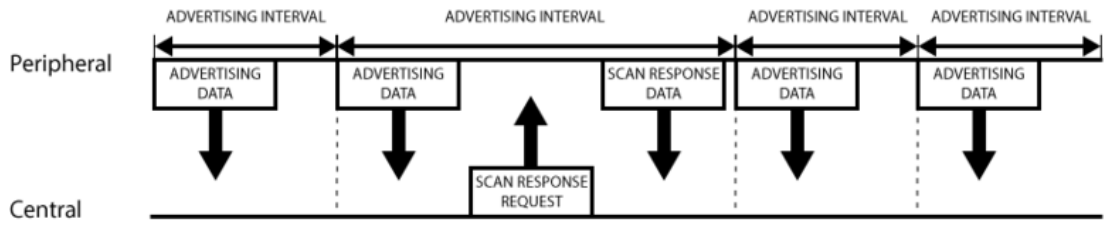


Figura 29. Esquema de comunicación entre el dispositivo periférico y el central

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap

5.3.3 Topología de redes broadcast

Mientras que la mayoría de periféricos se dan a conocer para que la conexión se establezca y así utilizar los servicios GATT (Lo que permite intercambiar información en ambas direcciones), hay situaciones en las que solamente se desean datos de broadcast.

El caso de uso principal es cuando se desea que un periférico envíe datos a más de un dispositivo a la vez. Esto solamente es posible mediante el broadcast, ya que, en modo conectado, los datos transmitidos solo pueden ser vistos por los dos dispositivos conectados.

Incluyendo una pequeña porción de datos configurables en los 31 bytes de los payloads de advertising o de respuesta, se puede configurar un dispositivo periférico BLE para enviar datos, que solamente irán en una dirección, a todos los dispositivos dentro del rango. Esto se conoce como Broadcasting en Bluetooth Low Energy.

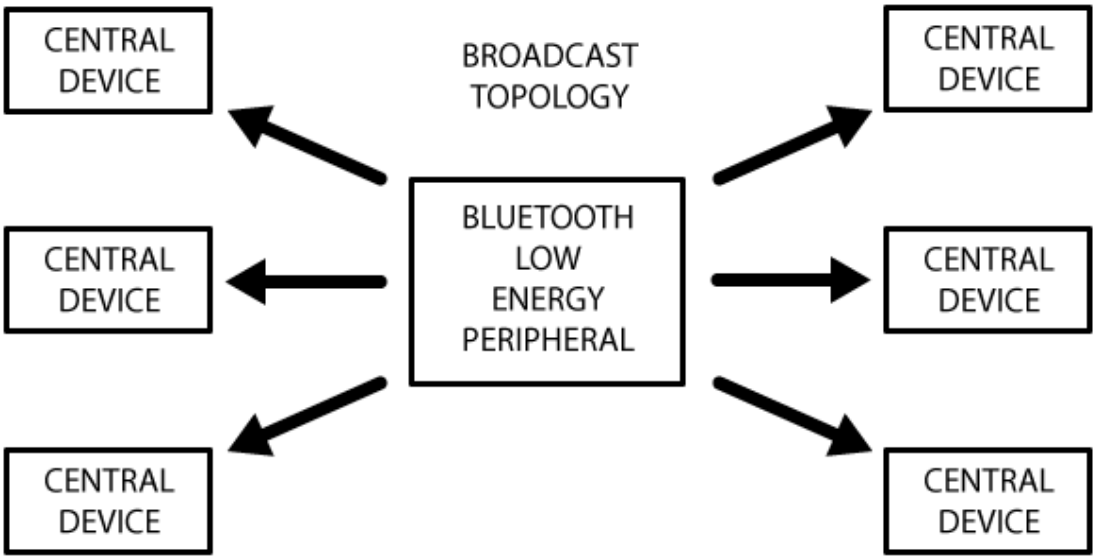


Figura 30. Esquema de la tipología broadcast

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap

Generalmente, cuando se establece una conexión entre un periférico y un dispositivo central, el proceso en el que el periférico se da a conocer con mensajes broadcast se detiene, y se dejan de enviar paquetes de advertising. Es en este momento cuando se utilizará los servicios y características de GATT para comunicarse en ambas direcciones.

5.3.4 Topología de conexión (GATT)

GATT es un acrónimo cuyo significado es Perfil de Atributos Genéricos (Generic Attribute Profile) que define la forma en que dos dispositivos Bluetooth Low Energy transmiten y reciben datos haciendo uso de un protocolo de datos genérico llamado Attribute Protocol (ATT), utilizado para almacenar Servicios, Características y datos relacionados en una simple tabla de consulta utilizando IDs de 16 bits para cada entrada de dicha tabla.

Los perfiles GATT ofrecen una estructura de datos jerarquizada para los dispositivos Bluetooth LE mientras que mantiene una interoperabilidad completa con otros dispositivos Bluetooth.

El nivel más alto de esta jerarquía es el perfil, el cual se compone por uno o más servicios necesarios para completar un caso de uso. Un servicio se compone de características o referencias a otros servicios. GATT agrupa estos servicios para encapsular el comportamiento de parte de un dispositivo, y describe un caso de uso, roles y comportamientos generales basados en la funcionalidad GATT.

Una característica consiste en un tipo, representado por una UUID (Universally Unique Identifier), un valor, un conjunto de propiedades que indican las operaciones que esa característica soporta, y un conjunto de permisos relacionados con la seguridad. También puede contener uno o más descriptores (metadatos o flags de configuración relacionados con la característica).

GATT entra en acción una vez la conexión está establecida entre dos dispositivos, por lo que antes se tiene que pasar por el proceso gobernado por el GAP.

Lo más importante a tener en cuenta con GATT es que las conexiones son exclusivas, es decir, que solo puede haber un periférico BLE conectado a un dispositivo central a la vez. En el momento que el periférico se conecte al central, dejará de mostrarse visible a otros dispositivos ni estos podrán establecer conexión con él hasta que la conexión existente se finalice.

Además, establecer una conexión es la única manera de permitir una comunicación en los dos sentidos, donde el dispositivo central puede enviar datos significativos al periférico y viceversa.

5.3.5 Topología de redes conectadas

En el diagrama se muestra cómo los dispositivos BLE trabajan cuando se encuentran en un estado de conexión. Un periférico solamente puede conectarse a un dispositivo central a la vez, pero los dispositivos centrales pueden estar conectados a varios periféricos.

Si se necesita intercambiar datos entre dos periféricos, se necesitará implantar un sistema de *mailbox* configurado para esa situación, donde todos los mensajes pasan a través de un dispositivo central.

Una vez se establece la conexión entre un periférico y un dispositivo central, la comunicación puede darse en los dos sentidos. Esta es la principal diferencia con la comunicación broadcast que solamente va en una dirección utilizando los datos de advertising de GAP.

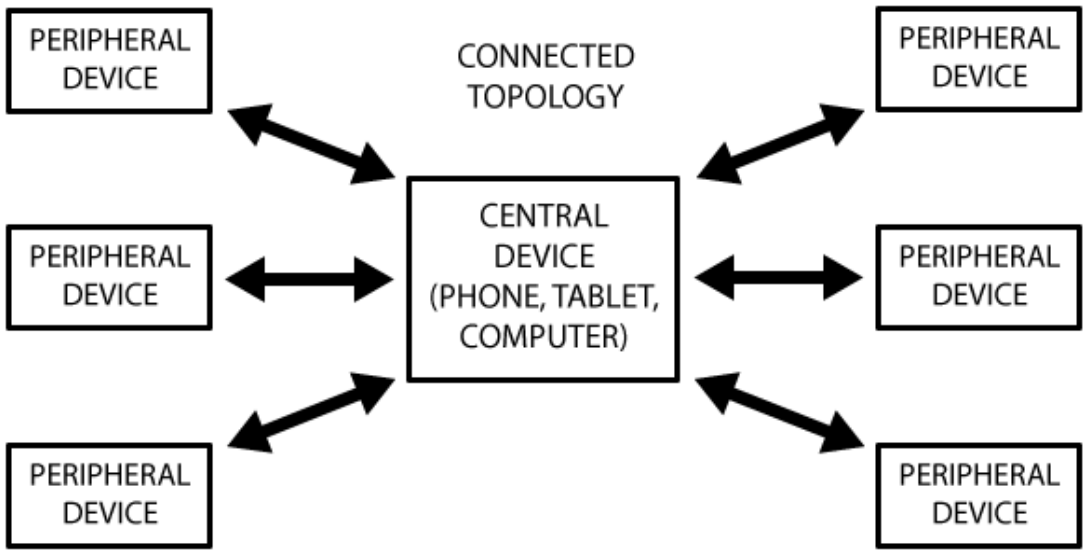


Figura 31. Esquema de la topología al establecer conexión

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap

5.3.5.1 Transacciones en GATT

Un concepto que es importante conocer en GATT, son las relaciones cliente/servidor.

El periférico se conoce como Servidor GATT, y contiene los datos de búsqueda ATT y las definiciones de los servicios y características, mientras que el Cliente GATT solamente envía peticiones a este servidor.

Todos los intercambios de información son demandados por el dispositivo maestro, el Cliente GATT, que recibe la respuesta del dispositivo esclavo, el Servidor GATT.

Cuando se establece una conexión, el periférico indicará un intervalo de conexión al dispositivo central, por lo que este intentará reconectar periódicamente pasado ese tiempo para cerciorarse de si hay datos nuevos disponibles. Es importante mantener en mente que este intervalo de tiempo es una sugerencia, y que el dispositivo central puede que no cumpla dicho tiempo por estar ocupado conectado a otro periférico o porque los recursos necesarios no están disponibles.

El siguiente diagrama muestra como es el proceso de intercambio de datos entre el Servidor GATT y el Cliente GATT, con el maestro iniciando cada transacción

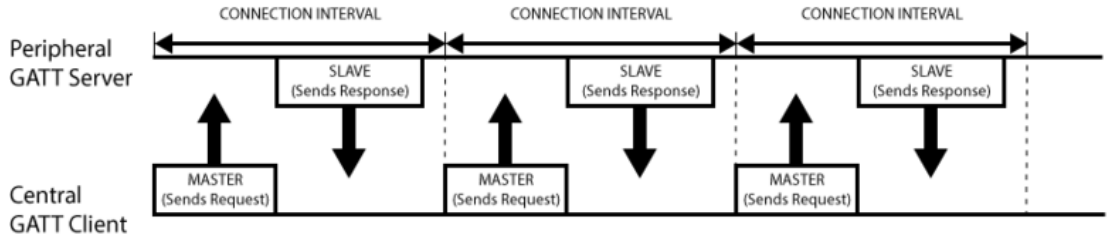


Figura 32. Esquema de comunicación entre el servidor y el cliente GATT

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

5.3.5.1.1 Servicios y Características

Las transacciones en BLE están basadas en objetos anidados de alto nivel llamados Perfiles, Servicios y Características.

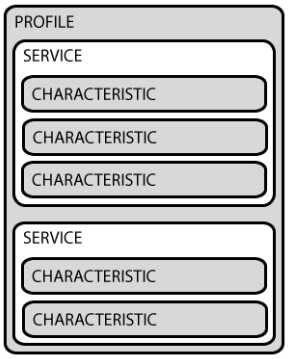


Figura 33. Jerarquía y niveles de GATT

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

5.3.5.1.2 Perfiles

Un perfil realmente no existe en un dispositivo periférico como tal. Es simplemente una colección de servicios predefinidos que han sido compilados o bien por Bluetooth SIG o por diseñadores de periféricos.

El perfil de frecuencia cardíaca, por ejemplo, combina el servicio de frecuencia cardíaca con el servicio de información del dispositivo.

Existe una gran variedad de perfiles y servicios basados en las especificaciones de GATT (34), los cuales se pueden consultar en formato XML. Existen perfiles como el Perfil de Presión Sanguínea, el Perfil de Velocidad y Cadencia de Pedaleo, el Perfil de Glucosa, el Perfil de Proximidad, etc. Todos estos perfiles suelen incluir servicios con el mismo nombre, además de incluir otros servicios.

5.3.5.1.3 Servicios

Los servicios se utilizan para fragmentar los datos en entidades lógicas que contienen agrupaciones de datos llamados características. Un servicio puede tener una o más características, y cada servicio se distingue de otros gracias a una ID numérica única llamada UUID, que puede ser de 16 bits para los servicios oficialmente adoptados de BLE o 128 bits para servicios personalizados.

Existe una gran cantidad de servicios oficialmente adoptados de BLE. Aquí se muestran varios ejemplos de cómo se presentan dichos servicios:

SpecificationName	SpecificationType	AssignedNumber	SpecificationLevel
Alert Notification Service	org.bluetooth.service.alert_notification	0x1811	Adopted
Automation IO	org.bluetooth.service.automation_io	0x1815	Adopted
Battery Service	org.bluetooth.service.battery_service	0x180F	Adopted
Blood Pressure	org.bluetooth.service.blood_pressure	0x1810	Adopted
Body Composition	org.bluetooth.service.body_composition	0x181B	Adopted
Bond Management	org.bluetooth.service.bond_management	0x181E	Adopted
Continuous Glucose Monitoring	org.bluetooth.service.continuous_glucose_monitoring	0x181F	Adopted
Current Time Service	org.bluetooth.service.current_time	0x1805	Adopted

Figura 34. Servicios de GATT adoptados por BLE

https://www.bluetooth.com/specifications/gatt/generic-attributes-overview

En esa misma tabla, se puede observar que, lógicamente, aparece el servicio de frecuencia cardíaca.

Heart Rate	org.bluetooth.service.heart_rate	0x180D	Adopted

Figura 35. El servicio Heart Rate tiene el UUID 0x180D

https://www.bluetooth.com/specifications/gatt/generic-attributes-overview

De esta tabla podemos obtener que este servicio ha adoptado oficialmente la UUID de 16 bits 0x180D. Entrando dentro de este servicio, se puede comprobar que incluye tres características, de las cuales, se indica que solamente es obligatoria la primera:

- Medida del pulso cardíaco. Se utiliza para enviar las medidas del pulso cardíaco
- Localización del sensor en el cuerpo. Se utiliza para describir la localización prevista del dispositivo.
- Punto de control de frecuencia cardíaca. Se utiliza para permitir al cliente escribir puntos de control al servidor para controlar su comportamiento.

5.3.5.1.4 Características

Como ya se ha comentado anteriormente, el nivel más bajo en el concepto de las transacciones GATT son las características, las cuales encapsulan un único valor.

Similarmente a los servicios, cada característica se distingue a sí misma con una UUID predefinida de 16 o 128 bits. Se puede utilizar una de las características estándar definidas por Bluetooth SIG, mediante las cuales se garantiza la interoperabilidad de los dispositivos, o se puede definir una propia que solamente entenderán los dispositivos preparados con tal propósito. (35), (36)

Aquí se muestran una serie de ejemplos de las características oficiales que aparecen en la web oficial:

SpecificationName	SpecificationType	AssignedNumber	SpecificationLevel
Aerobic Heart Rate Lower Limit	org.bluetooth.characteristic.aerobic_heart_rate_lower_limit	0x2A7E	Adopted
Aerobic Heart Rate Upper Limit	org.bluetooth.characteristic.aerobic_heart_rate_upper_limit	0x2A84	Adopted
Aerobic Threshold	org.bluetooth.characteristic.aerobic_threshold	0x2A7F	Adopted
Age	org.bluetooth.characteristic.age	0x2A80	Adopted
Aggregate	org.bluetooth.characteristic.aggregate	0x2A5A	Adopted
Alert Category ID	org.bluetooth.characteristic.alert_category_id	0x2A43	Adopted
Alert Category ID Bit Mask	org.bluetooth.characteristic.alert_category_id_bit_mask	0x2A42	Adopted
Alert Level	org.bluetooth.characteristic.alert_level	0x2A06	Adopted

Figura 36. Características de GATT

https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=239865

Cada Característica tiene un UUID asignado, que no hay que confundir con el UUID del Servicio.

De nuevo, si buscamos el pulso cardíaco, encontramos la siguiente definición:

Heart Rate Measurement	org.bluetooth.characteristic.heart_rate_measurement	0x2A37	Adopted

Figura 37. La característica Heart Rate Measurement tiene por UUID 0x2A37

https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=239865

La medida del pulso cardíaco, la cual es obligatoria, tiene como UUID el valor 0x2A37. Al consultar esta característica, se obtiene información sobre la estructura de datos que tienen los paquetes que se enviarán. Un paquete proveniente de esta característica empezará con un valor de 8 bits describiendo el formato en que se expresa la medida (si es UINT8, UINT16, etc.) y continúa incluyendo la medida de la frecuencia cardíaca en el formato especificado.

Las características son la manera principal de interactuar con el dispositivo Bluetooth Low Energy periférico, por lo que es necesario comprender el concepto. También se utilizan las características para enviar datos al periférico, ya que en algunas características se permite también escribir datos. Se podría implementar una interfaz tipo UART con un servicio UART y dos características: una para el canal de transmisión y otra para el de recepción, aplicando los privilegios de escritura y lectura pertinentes.

5.3.5.2 Especificación del Perfil de Pulso Cardíaco

El perfil define dos roles: El sensor de frecuencia cardíaca y el colector. El sensor de frecuencia cardíaca es el dispositivo que mide el pulso y otra información determinada, mientras que el colector es el dispositivo que recibe esta información desde el sensor.

- El sensor de frecuencia cardíaca será el servidor GATT
- El colector será el cliente GATT

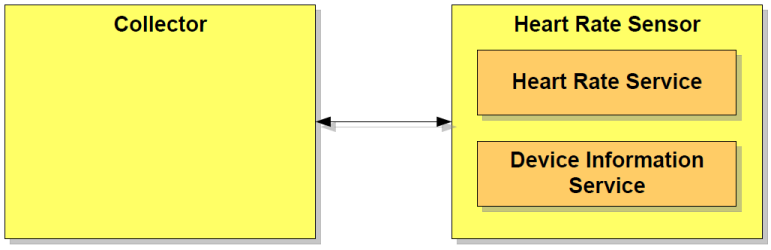


Figura 38. Esquema de los roles existentes en el sensor

http://saber.patagoniatec.com/3314-arduino-argentina-ptec-iic-i2c-muchos/.

En el esquema anterior, se ve representada la relación entre los servicios y los dos roles. En color amarillo se muestran los roles del perfil y en naranja los servicios. Un sensor de frecuencia cardíaca instancia únicamente un servicio de pulso cardíaco y un servicio de información de dispositivo.

El sensor de pulso cardíaco utilizará el rol GAP de periférico, mientras el colector usará el rol de central y el transporte de datos debe hacerse obligatoriamente sobre una conexión LE.

Cuando el dispositivo se está dando a conocer a través del GAP al colector, el sensor de frecuencia cardíaca incluye el UUID del servicio de frecuencia cardíaca, así como el nombre del dispositivo, por lo que el colector identificará al sensor incluso antes de establecer la conexión.

A través de GATT, el colector identificará los servicios y las características del sensor de frecuencia cardíaca y accediendo a ellas podrá comunicarse con el pulsómetro y leer los distintos datos de interés. (37)

5.4 I2C

I2C (Inter-Integrated Circuit) es un bus de comunicación muy utilizado para comunicar circuitos integrados. Uno de sus usos más comunes es la comunicación entre un microcontrolador y sensores periféricos. El I²C está diseñado como un bus maestro-esclavo. La transferencia de datos es siempre inicializada por un maestro, mientras que el esclavo reacciona a las señales que recibe de este. Es posible tener varios maestros mediante un modo multimaestro, en el que se pueden comunicar dos maestros entre sí, de modo que uno de ellos trabaja como esclavo. En este caso, el arbitraje (control de acceso en el bus) se rige por las especificaciones, de este modo los maestros pueden ir turnándose.

El bus I²C fue introducido en 1982 por Philips para la comunicación interna entre circuitos integrados como por ejemplo juegos de CD y televisiones. La primera especificación estandarizada 1.0 fue publicada en 1992 y fue introducida en la familia de microcontroladores MAB8400. Este bus sustituyó el estándar original de 100 kbps por un nuevo modo rápido con 400 kbps y expandió el espacio de direccionamiento a un modo de 10 bits, de tal manera que, en vez de los 112 nodos originales, era compatible con hasta 1136 nodos.

Con la versión 2.0 de 1998 llegó el modo de alta velocidad (Hs) con un máximo de 3,4 Mbps, aunque los requisitos de voltaje e intensidad de corriente fueron reducidos. La versión 3.0 de 2007 incluyó un nuevo modo denominado Fm+ (modo rápido mejorado) con una velocidad máxima de 1 Mbps que, al contrario que el modo Hs, utiliza el mismo protocolo que los modos de 100 y 400 kbps.

5.4.1 Funcionamiento

El bus I2C cuenta con dos líneas. La línea de datos SDA (Serial Data) y la línea de reloj SCL (Serial Clock). Al igual que en SPI, a cada flanco de SCL se captura un bit de SDA, aunque la forma de transmitir la información es diferente. El Bus I²C trabaja con lógica positiva, esto quiere decir que un nivel alto en la línea de datos corresponde a un 1 lógico, el nivel bajo a un 0. Además, hay que tener en cuenta que la línea SDA solo puede cambiar de valor en caso de que la línea SCL este a 0. Esto es así porque el 0 se consigue forzando la línea a esa tensión, pero, por el contrario, el 1 se consigue con pull-up, por lo que, en caso de que un rol transmita un 0 y otro un 1, en la línea solo se verá reflejado el 0. Por lo tanto, se define como el valor de reposo del bus como el 1, ya que si alguien quiere empezar a comunicar siempre podrá modificar el estado del bus y los demás se darán cuenta. El nivel alto debe ser de al menos 0,7 x VDD y el nivel bajo no debe ser más de 0,3 x VDD.

Este protocolo utiliza la misma línea de datos tanto para enviar los datos como para recibirlos, por lo que es necesario un control de acceso al bus y un direccionamiento de cada elemento.

La señal de reloj siempre es generada por el maestro. Para cada modo especificado, está predeterminado respectivamente un pulso de reloj máximo permitido. En general, también pueden ser utilizadas señales de reloj más lentas, siempre y cuando sean compatibles con la interfaz del maestro Si el esclavo necesita más tiempo que el dictado por el reloj del maestro puede mantener entre la transferencia de bytes individuales la señal de reloj en nivel bajo o low (clock-stretching) para frenar de este modo al maestro.

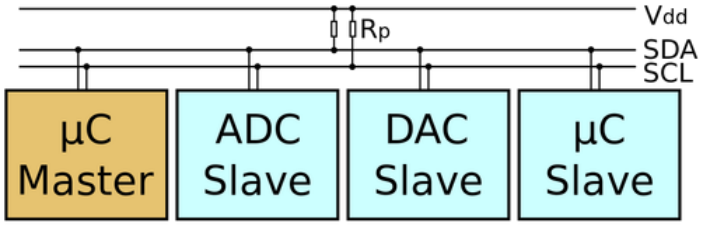


Figura 39. Esquema de funcionamiento de I2C

http://www.uco.es/~el1mofer/Docs/IntPerif/Bus%20I2C.pdf.

Los datos (bits individuales) sólo son válidos si su nivel lógico no cambia durante una fase de reloj alta. Las excepciones son el inicio, la parada, y la señal de inicio repetida o reset.

- La señal de arranque es un flanco descendente en SDA mientras SCL se encuentra a nivel alto.
- La señal de parada es un flanco ascendente en SDA mientras SCL está a nivel alto.
- La señal de reset se comporta de igual manera que la señal de inicio.

Una unidad de datos consta de 8 bits (los cuales puede ser interpretados como un valor o como una dirección, dependiendo del protocolo) y un bit de ACK. Este bit de confirmación (Acknowledge) es señalizado por un esclavo como NACK (not acknowledge) con un nivel alto, durante un nivel bajo en SDA y el noveno nivel alto de SCL (que sigue siendo generado por el maestro). El esclavo debe poner un nivel bajo en SDA antes de que SCL cambie a nivel alto, de lo contrario otros participantes podrían interpretar esto como una señal de arranque.

Cada uno de los circuitos integrados con capacidad de soportar un I²C tiene una dirección predeterminada por el fabricante, de la cual los últimos tres bits (subdirección) pueden ser fijados por tres pines de control. En este caso, pueden funcionar en un I²C hasta 8 circuitos integrados. Si no es así, los circuitos integrales (que precisan ser idénticos) deben ser controlados por varios buses I²C separados.

Debido a la escasez de direcciones, se introdujo más tarde un direccionamiento de 10 bits. Es compatible con el estándar de 7 bits mediante el uso de 4 de las 16 direcciones reservadas. Ambos modos de direccionamiento pueden utilizarse simultáneamente, lo que permite hasta 1136 nodos en un único bus. (38)

Otro aspecto a tener en cuenta en I2C es que el maestro o maestros, en caso de haber más de uno, es el único que puede controlar la línea de SCL. Esto implica que solo un maestro puede iniciar una transmisión por lo que un Slave tendrá que esperar a que un maestro le pregunte por un dato para poder enviarlo.

5.4.2 Estados del bus

El bus I2C puede encontrarse en distintos estados, que la norma define como los siguientes (39):

- **Libre**: Este estado se caracteriza por encontrase las líneas SDA y SCL a 1, sin que se esté realizando ningún tipo de transferencia.

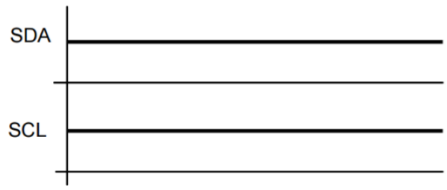


Figura 40. Bus en estado Libre

http://www.uco.es/~el1mofer/Docs/IntPerif/Bus%20I2C.pdf

- **Inicio**: Se produce una condición de inicio cuando un maestro inicia una transacción. El primer bit es un bit especial ya que, como hemos dicho antes, la línea SDA no puede cambiar a menos que SCL este a 0. Este bit rompe dicha norma y provoca un cambio de 1 a 0 cuando SCL está a nivel alto. A partir de que se dé una condición de inicio se considerará que el bus está ocupado y ningún otro maestro deberá intentar generar su condición de inicio.

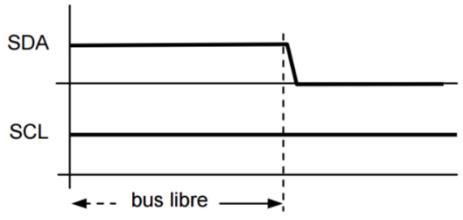


Figura 41. Bus en estado de Inicio

http://www.uco.es/~el1mofer/Docs/IntPerif/Bus%20I2C.pdf

• **Cambio**: Se produce una condición de cambio cuando, estando a baja la línea SCL, la línea SDA puede cambiar de estado. En la transferencia de datos por un bus I2C éste es el único instante en el que el sistema emisor (que podrá ser tanto un maestro como un esclavo) podrá poner en la línea SDA cada bit del carácter a transmitir.

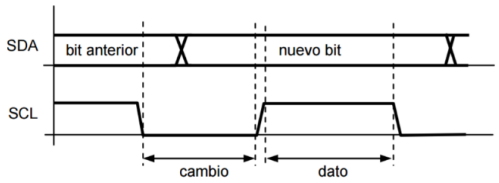


Figura 42. Bus en estado de cambio

http://www.uco.es/~el1mofer/Docs/IntPerif/Bus%20I2C.pdf

- **Dato:** Este estado es el que, una vez iniciada una transacción, queda definido por la fase alta de la señal de sincronía SCL. En este estado se considera que el dato emitido es válido, y no se admite que pueda cambiar. Recuérdese que, ya iniciada una transferencia, el único instante en que la línea de datos puede cambiar es en el estado de cambio.

  Los bits transmitidos, por norma general, serán los siguientes:

  1. Address: El primer byte enviado empieza con 7 bits de dirección, el cual indica a quien enviamos o solicitamos el dato.
  2. R/W (Read/Write): El siguiente bit indica si vamos a realizar una operación de lectura o escritura.
  3. ACK: Este bit está presente al final de cada byte que enviamos y nos permite asegurarnos que el byte ha llegado a su destino. De este modo el que transmisor pone la línea a nivel alto al finalizar el envío. Si el receptor ha recibido el mensaje forzará ese bit a 0. De esta manera se confirma que ha llegado el byte y la transmisión puede continuar.
  4. 1º Byte de datos: Este es el primer byte de datos propiamente dicho ya que los valores del anterior vienen prefijados por el protocolo. Aquí se transmitirán los datos provenientes del sensor o los datos que el microcontrolador escriba en el sensor, por ejemplo, para configurarlo. Después del byte de datos se espera otro ACK del receptor.
  5. Se repite el paso mientras la conexión continúe.

  • **Parada**. Se produce una condición de fin o parada cuando, estando la línea SCL a alta, se produce un cambio de baja a alta en la línea SDA. Obsérvese que esto es una violación de la condición de dato, y es precisamente por esto por lo que se utiliza para que un maestro pueda indicar al esclavo que se finaliza la transferencia. Tras la condición de parada se entra automáticamente en el estado de bus libre

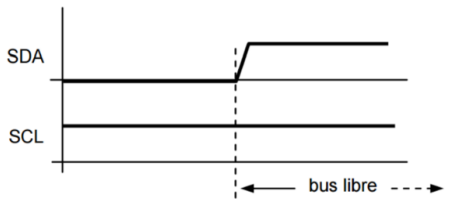


Figura 43. Bus en estado de parada

http://www.uco.es/~el1mofer/Docs/IntPerif/Bus%20I2C.pdf

En la siguiente imagen, se muestra el desarrollo temporal que tendrían los distintos bits transmitidos en la comunicación:

Figura 44. Sucesión de bits en el tiempo

http://www.uco.es/~el1mofer/Docs/IntPerif/Bus%20I2C.pdf

Lo explicado anteriormente es una trama típica de I2C, aunque en realidad puede haber algunas variantes, ya que en caso de que el mismo Master quiera seguir comunicándose no es necesario terminar la transmisión con un Stop y luego volver a empezar una transmisión. El protocolo permite que en medio de una trama se realice otro Start por lo que se tendrá que volver a enviar la dirección y el R/W, así que se puede cambiar el modo de comunicación y pasar a una lectura o a otra escritura y acceder a otro sensor o a otro registro del mismo sensor.

Una de las propiedades del I²C es el hecho de que un microcontrolador puede controlar toda una red de circuitos integrados con sólo dos I/O-Pins (Input/Output) y un software muy simple. Los buses de este tipo surgieron ya que una proporción significativa del precio de un circuito integrado y la placa de circuito depende del tamaño de la carcasa y del número de pines. Una carcasa grande tiene más pines, necesita más espacio en la placa de circuito y tiene más conexiones que podrían fallar. Todo esto aumenta los costes de desarrollo, producción y pruebas.

Aunque es más lento que los sistemas de bus más nuevos, I²C es beneficioso (debido al bajo coste) para los sistemas periféricos que no necesitan ser rápidos. A menudo es usado para la transmisión de datos de control y configuración, por ejemplo, para control de volumen, conversor de señal analógica-digital o digital-analógica con baja tasa de frecuencia de muestreo, relojes a tiempo real, pequeños espacios de memoria o conmutadores bidireccionales y multiplexores. Incluso los sensores electrónicos integran con frecuencia un convertidor analógico-digital con un I²C.

El protocolo I²C tuvo gran importancia en el pasado en el área de las tarjetas chip. La tarjeta sanitaria utilizada en Alemania hasta finales del 2014 era una tarjeta I²C, es decir, debajo de la superficie dorada del chip, se encontraba una simple I²C-EEPROM, que podía ser leída y escrita por el lector de tarjetas a través de un protocolo de bus I²C.

5.5 Proceso de obtención de datos

Los sensores irán fijados al traje de algún modo o los llevará incorporado el bombero como puede ser el caso de la banda de pulsómetro. Dichos sensores deben ofrecer fiabilidad, robustez y rapidez en la obtención y envío de los factores a evaluar, pues en estas circunstancias, el tiempo y la fiabilidad de la información es vital.

Contrastando las cuatro variables a evaluar, el dato más importante y que mayor frecuencia debe tener sería la detección de caídas. Seguido de la pulsación del bombero, la concentración de gas y la temperatura.

Se deben garantizar unos tiempos mínimos y una conexión entre el microcontrolador y el sensor constante e ininterrumpida. Los datos deberán procesarse para que puedan ser interpretados por el usuario o por la propia Raspberry Pi en el caso de que se deba lanzar cierta señal de alerta cuando se alcance un determinado valor en alguna de las variables.

5.5.1 Pulsómetro CooSpo H8

Como se ha comentado anteriormente, el pulsómetro utiliza la tecnología Bluetooth para enviar la información relativa a las pulsaciones por minuto del usuario. Es por eso que, antes que nada, hay que acondicionar la Raspberry Pi para poder utilizar Bluetooth.

Utilizando un dongle USB de Bluetooth 4.0 dotamos a la Pi de conexión a través de esta tecnología, pero para su utilización se necesita instalar una serie de librerías. En primer lugar, instalamos la última versión de BlueZ, un stack Bluetooth para Linux que trabaja con Raspbian, y que permite configurar el dongle.

Con esta librería instalada, se puede activar la conexión Bluetooth y detectar los dispositivos cercanos. Primero, hay que encender el dongle a través del comando hciconfig:

- *hciconfig hci0 up* → activará el Bluetooth en la Raspberry.
- *hciconfig hci0 down* → lo desactivará.

Para realizar una captura de los mensajes de advertising de alrededor y localizar los dispositivos cercanos, habrá que utilizar *lescan*, de la herramienta hcitool (40):

```
pi@raspberrypi:~ sudo hciconfig hci0 up
```

Figura 45. Comando para iniciar el bluetooth en la Raspberry

```
pi@raspberrypi:~ $ sudo hcitool -i hci0 lescan
LE Scan ...
D0:5F:B8:41:16:2F heart rate sensor
D0:5F:B8:41:16:2F heart rate sensor
D0:5F:B8:41:16:2F heart rate sensor
D0:5F:B8:41:16:2F heart rate sensor
```

Figura 46. Detección del sensor a través del dongle Bluetooth

Como se puede observar en la imagen, cuando se ordena al dongle Bluetooth escuchar los mensajes broadcast que hay a su alrededor, comienzan a aparecer muchos mensajes de advertising del pulsómetro. Estos mensajes, como se ha explicado anteriormente, constan de dos partes: El Advertising Data Payload y el Scan Response Payload. El primero, la dirección Bluetooth de la banda de frecuencia cardíaca, cuyo valor es D0:5F:B8:41:16:2F, y es la que se utilizará para realizar la conexión. El segundo, es la información adicional que el fabricante coloca a su producto, en este caso es información para detectar de forma más sencilla de qué dispositivo se trata: *heart rate sensor*.

Para realizar la conexión al dispositivo, es necesaria otra librería llamada gatttool. Tras instalarla, se puede utilizar para enviar señales entre ambos dispositivos. Con el comando connect y escribiendo la dirección Bluetooth, se puede establecer la conexión (41)

```
pi@raspberrypi:~ $ sudo gatttool -b D0:5F:B8:41:16:2F -I
[D0:5F:B8:41:16:2F][LE]>
```

Figura 47. Conexión al sensor a través de la herramienta Gatttool

El –b es para indicar que el valor que va a continuación será la dirección física del dispositivo, mientras que –I indica que es modo interactivo, es decir, se ofrecerá una especie de interfaz en el que el usuario puede enviar comandos al dispositivo y recibir respuestas, realizando una interacción más dinámica con este. El modo no interactivo se utilizaría en programas que llamen a esta librería, los cuales no van a actuar en base a la respuesta recibida, sino que lanzarán el comando deseado junto a las configuraciones deseadas todo en una misma línea en la que se indicará también la dirección del dispositivo por cada vez que se desee enviar un comando.

Una vez el dispositivo ha apuntado a esa dirección, nos aparecerá una cabecera en el terminal indicando la dirección a la que apunta la conexión y un indicativo de que se está utilizando Bluetooth de bajo consumo BLE.

```
[D0:5F:B8:41:16:2F][LE]> connect
Attempting to connect to D0:5F:B8:41:16:2F
Connection successful
[D0:5F:B8:41:16:2F][LE]>
```

Figura 48. Establecimiento de la conexión con el sensor

Al realizar la conexión, en la cabecera aparecerá como que ambos dispositivos han establecido una conexión, por lo que ya se podrían enviar y recibir señales de la banda de pulsómetro.

Como se ha explicado anteriormente sobre el protocolo GATT, un dispositivo contiene en primer lugar un perfil, que no es más que una agrupación de servicios. Estos a su vez contienen características con propiedades, valores, y descriptores.

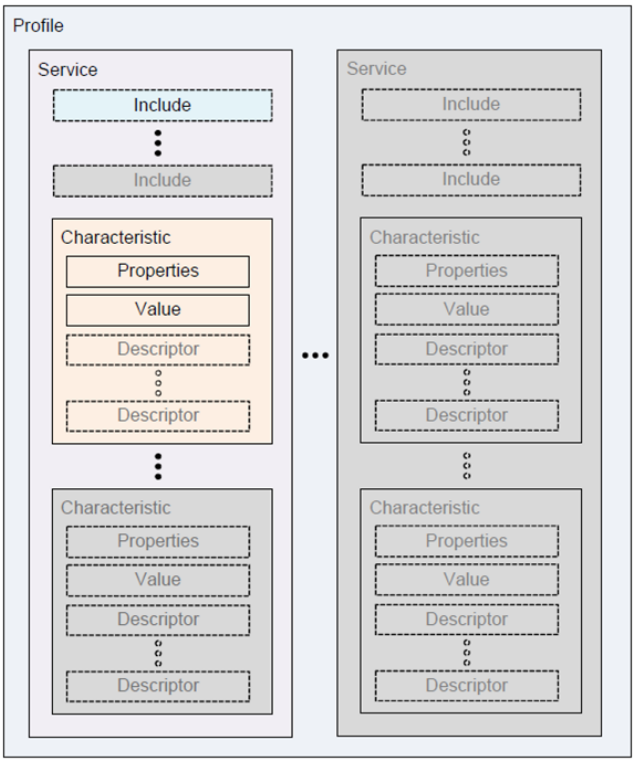


Figura 49. Esquema de niveles de GATT

https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

Al utilizar el comando *help*, se listan por pantalla los distintos comandos compatibles con la herramienta gatttool. Entre ellos, se pueden encontrar las funciones que permiten interactuar con el perfil GATT, así como una breve descripción de estas:

```
GATT commands
  --primary                             Primary Service Discovery
  --characteristics                     Characteristics Discovery
  --char-read                           Characteristics Value/Descriptor Read
  --char-write                          Characteristics Value Write Without Response (Write Command)
  --char-write-req                      Characteristics Value Write (Write Request)
  --char-desc                           Characteristics Descriptor Discovery
  --listen                              Listen for notifications and indications
```

Figura 50. Comandos de la herramienta gatttool

Entre los distintos comandos, el punto de partida, después de realizar la conexión, sería el comando *primary*, que descubre los servicios primarios que ofrece el dispositivo (el perfil).

Al haber ejecutado la herramienta gatttool en modo interactivo, no es necesario escribir de nuevo el comando con sus argumentos, sino que simplemente escribiendo *primary*, se ejecuta el comando. Al hacerlo, por la pantalla del terminal aparecen los distintos servicios que ofrece el dispositivo, identificados por la UUID, así como las direcciones inicial (attr handle) y final (end grp handle) que son las que tendremos que leer para obtener los datos de un servicio concreto (42).

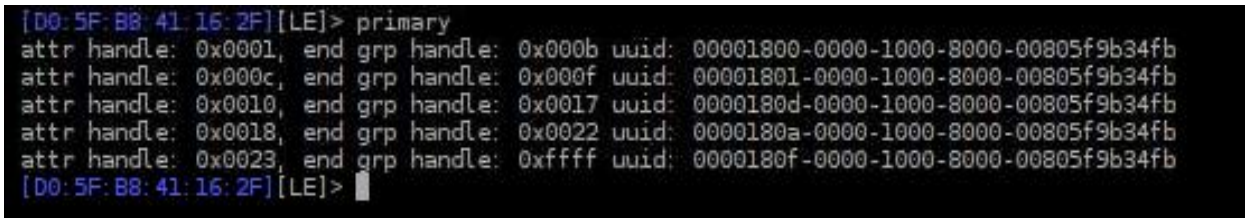


Figura 51. Servicios ofrecidos por el sensor

Para este dispositivo concreto, los servicios que se ofrecen son los siguientes:

- 0x1800 ► Direcciones Genéricas
- 0x1801 ► Atributos Genéricos
- **0x180d** ► Pulso cardíaco
- 0x180a ► Servicio de información de dispositivo
- 0x180f ► Servicio de batería

Por lo tanto, el perfil de Heart Rate, ofrece como servicios información sobre el dispositivo, la medida del pulso cardíaco y un servicio de comprobación de la batería.

5.5.1.1 Obtención del pulso cardíaco

Para descubrir las características de un servicio, hay que utilizar el comando *characteristics*. A este comando se le pueden pasar como argumentos la dirección inicial y la final. Como para el pulso cardíaco (UUID=0x180D), las direcciones utilizadas son de la 0x0010 a la 0x0017, entre estas direcciones se encontrarán sus características, y por tanto también las propiedades, valores y descriptores asociados a cada una.

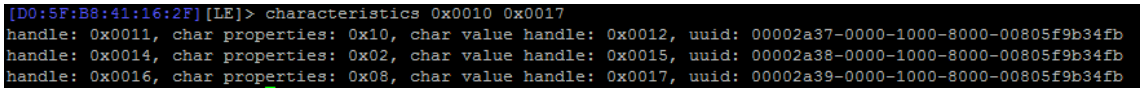


Figura 52. Características ofrecidas por el sensor relacionadas con el servicio Heart Rate

Si no se le pasaran argumentos al comando *characteristics*, se mostraría una lista de todas las características que aparezcan en todas las direcciones de memoria de todos los servicios juntos.

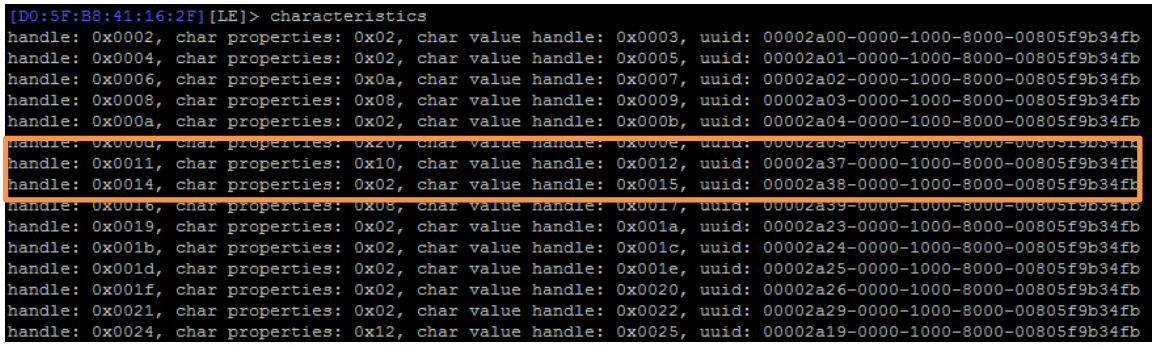


Figura 53. Conjunto de características ofrecidas por el sensor. Encuadradas aparecen las que tienen que ver con el servicio Heart Rate

Volviendo a las características del Servicio de pulso cardíaco, el resultado de la búsqueda habían sido tres entradas, cada una con su propia UUID: 0x2A37, 0x2A38 y 0x2A39.

Consultando la tabla de características de GATT, se consigue dar nombre a estas características y se observa que, efectivamente, son las propias del servicio, como se había comentado durante la descripción del funcionamiento de la tecnología Bluetooth Low Energy y de la especificación del pulso cardíaco en GATT.

Heart Rate Measurement	org.bluetooth.characteristic.heart_rate_measurement	0x2A37	Adopted
Body Sensor Location	org.bluetooth.characteristic.body_sensor_location	0x2A38	Adopted
Heart Rate Control Point	org.bluetooth.characteristic.heart_rate_control_point	0x2A39	Adopted

Figura 54. UUIDs de las características del servicio Heart Rate

https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=239865.

Con las direcciones obtenidas anteriormente y el comando *char-desc*, se puede descubrir el interior que cada uno de estos servicios tiene.

Para la medida del pulso cardíaco, con UUID 0x2A37, el handle tiene la posición 0x0011, las propiedades la 0x0010, y el handle del valor la posición 0x012.

Como esta característica empieza en la posición 0x011 y la siguiente en la 0x014, lo lógico será leer los descriptores que hay en ese rango de direcciones:

```
[D0:5F:B8:41:16:2F][LE]> char-desc 0x010 0x013
handle: 0x0010, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0011, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0012, uuid: 00002a37-0000-1000-8000-00805f9b34fb
handle: 0x0013, uuid: 00002902-0000-1000-8000-00805f9b34fb
```

Figura 55. Descriptores para la característica Heart Rate Measurement

Esta información se divide entre declaraciones, descriptores y la característica.

Las dos primeras entradas, situadas en las direcciones de memoria 0x0010 y 0x0011 con UUID 0x2800 y 0x2803 respectivamente, son la declaración del servicio primario y de la característica, lo cual está presente en todas las características de las que se lean los descriptores.

La dirección 0x0012 tiene como UUID la de la característica, la cual es la que gatttool lee cuando se hace la búsqueda de características.

Por último, la dirección 0x0013 es la información del descriptor.

Client Characteristic Configuration	org.bluetooth.descriptor.gatt.client_characteristic_configuration	0x2902	Adopted

Figura 56. UUID del descriptor para leer el valor de frecuencia cardíaca

Es, por tanto, un campo con la configuración de la característica del dispositivo. Este descriptor tiene un formato de 16bits, pero su máximo valor es 3, estando los demás bits reservados para futuros usos.

La característica tiene, por tanto, dos posibles campos a modificar, la habilitación/deshabilitación de las notificaciones y la habilitación/deshabilitación de las indicaciones.

Con el comando *char-write-req* se puede realizar una petición de escritura y una espera de la respuesta recibida. Así pues, activando las notificaciones y esperando la respuesta, se deberían recibir datos de pulso cardíaco:

```
[D0:5F:B8:41:16:2F][LE]> char-write-cmd 0x0013 0100
Notification handle = 0x0012 value: 00 4e
Notification handle = 0x0012 value: 00 4e
Notification handle = 0x0012 value: 00 4e
Notification handle = 0x0012 value: 00 4e
Notification handle = 0x0012 value: 00 4e
Notification handle = 0x0012 value: 00 50
Notification handle = 0x0012 value: 00 51
Notification handle = 0x0012 value: 00 53
```

Figura 57. Notificaciones del sensor indicando el pulso cardíaco en hexadecimal

Al activar las notificaciones, automáticamente comienzan a recibirse paquetes a través del handler 0x0012 (característica de pulso cardíaco).

Esta información está en hexadecimal, por lo que habría que convertir los valores a decimal y así obtener el valor del pulso cardíaco del portador de la banda.

5.5.1.2 Obtención del estado de la batería

Viendo los servicios que ofrece el dispositivo, también sería una buena práctica registrar en alguna parte el estado de la batería restante por si hubiese que reponerla próximamente. Su tiempo de vida aproximado es de un año, por lo que, con una debida atención al porcentaje restante antes de cada operación, la probabilidad de que la batería se agote en medio de un ejercicio es bastante escasa.

Como se ha visto en el punto anterior, la UUID del servicio de la batería es la 0x0180f, y sus características van desde la dirección 0x0023 hasta el final de la memoria.

```
[D0:5F:B8:41:16:2F][LE]> characteristics 0x0023 0x00ff
handle: 0x0024, char properties: 0x12, char value handle: 0x0025, uuid: 00002a19-0000-1000-8000-00805f9b34fb
```

Figura 58. Perfil para la lectura de batería

En este caso, únicamente se encuentra una característica con UUID 2A19. Al consultar la tabla de GATT dicha UUID, se obtiene que el nombre de la característica es *Battery Level*.

Leyendo los descriptores desde la posición 0x0023 hasta el final, se leerán los descriptores relacionados con el nivel de batería. Es aquí donde se aprecia que la estructura es idéntica a la de la característica del pulso cardíaco, y, en consecuencia, a la de todas las características en general: se obtienen la característica, las declaraciones y el descriptor.

```
[D0:5F:B8:41:16:2F][LE]> char-desc 0x0023 0x00ff
handle: 0x0023, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0024, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0025, uuid: 00002a19-0000-1000-8000-00805f9b34fb
handle: 0x0026, uuid: 00002902-0000-1000-8000-00805f9b34fb
```

Figura 59. Características del servicio de nivel de batería

Solamente hay que activar las notificaciones y para recibir con cierta frecuencia, bastante menor que para el pulso cardíaco, el valor de la batería.

```
[D0:5F:B8:41:16:2F][LE]> char-write-req 0x0026 0100
Notification handle = 0x0025 value: 5a
Characteristic value was written successfully
```

Figura 60. Escritura del valor en el sensor y obtención del nivel de batería

Obteniéndose el valor hexadecimal 5A, que en decimal significa un 90% de batería.

5.5.2 Acelerómetro MPU 6050

El acelerómetro irá ubicado a la espalda del traje del bombero, con el eje Y perpendicular al suelo y paralelo a la dirección de su espalda. Esto nos permitirá conocer la posición en la que se encuentra la persona en cada momento y detectar si ha sufrido una caída o si, simplemente, está agachado.

Este acelerómetro nos da una gran cantidad de datos sobre la dirección de la gravedad respecto de los tres ejes que sufre el acelerómetro y de la inclinación a la que se encuentra en cada momento.

Para su utilización, los pines VCC, GND, SCL y SDA del acelerómetro deben conectarse a los pines de igual nombre de la Raspberry como se muestra en la siguiente figura:

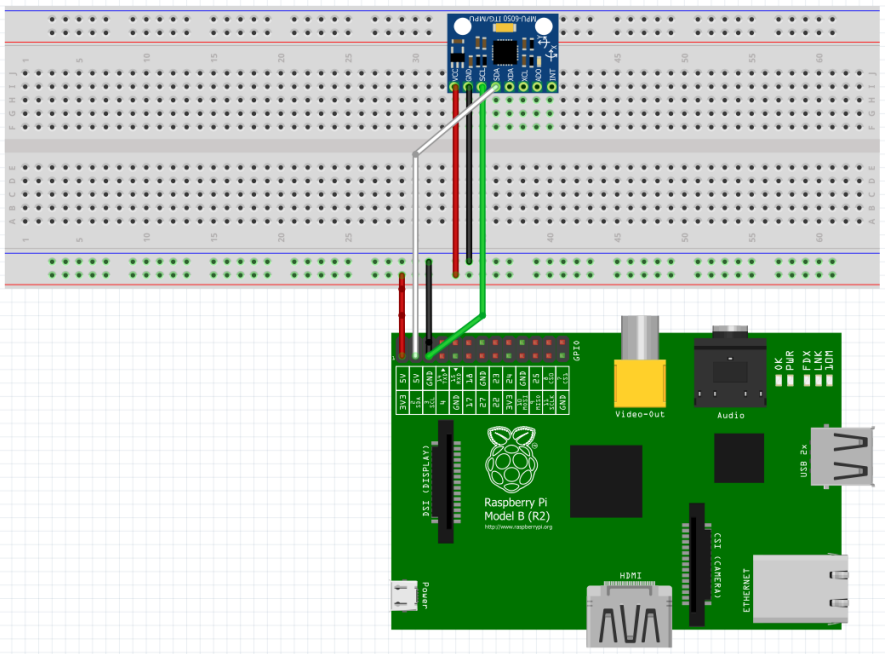


Figura 61. Conexión entre el MPU-6050 con la Raspberry Pi

Al igual que con el Bluetooth, para utilizar el bus I2C de forma rápida y sencilla, se deben instalar las librerías correspondientes que permiten detectar las direcciones en las que hay dispositivos conectados y leer o escribir en estas direcciones.

En primer lugar, tenemos que instalar los drivers necesarios para utilizar I2C. Para eso tenemos que editar el fichero modules:

```
sudo vi /etc/modules
```

En el caso de que no se encontraran ya, es necesario añadir las siguientes líneas, guardar el archivo y reiniciar la Raspberry:

```
i2c-bcm2708
i2c-dev
```

Como medida adicional, es un buen ejercicio cerciorarse que en la lista negra de la Raspberry no se ha denegado el uso del i2c. Para ello hay que acceder al archivo *raspi-blacklist.conf*

```
sudo vi /etc/modprobe.d/raspi-blacklist.conf
```

Si el protocolo i2c aparece en este archivo, con comentar mediante # dicha línea, bastaría. Si no aparece, es que está disponible.

```
#blacklist i2c-bcm2708
```

Una vez se ha conectado el acelerómetro a los pines de la Raspberry Pi, y ejecutados los pasos previos, hay que instalar las herramientas de I2C como se ha comentado anteriormente:

```
sudo apt-get install i2c-tools
```

Ahora ya es posible averiguar cuál es la dirección a la que está conectado el acelerómetro a través del comando i2cdetect.

`sudo i2cdetect -y 0` Para una placa de revisión 1

`sudo i2cdetect -y 1` Para una placa de revisión 2

En el caso de este proyecto, la placa era de revisión 2, lanzar *i2cdetect –y 0* en este caso, hará que no se encuentre ningún directorio. (43)

```
pi@raspberrypi:~/Downloads/bcm2835-1.52 $ sudo i2cdetect -y 0
Error: Could not open file `/dev/i2c-0' or `/dev/i2c/0': No such file or directo
ry
pi@raspberrypi:~/Downloads/bcm2835-1.52 $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figura 62. Tabla de direcciones de I2C

Este comando devuelve una tabla con el número de la dirección en la que ha detectado un dispositivo. Es así como I2C permite la conexión a varios dispositivos a través de un mismo pin.

El dispositivo cuenta con una gran cantidad de registros con propiedades de lectura o escritura para controlar, además del acelerómetro, las funcionalidades del giróscopo, la temperatura adquirida por el termómetro, registros para la configuración y utilización del DMP, el acceso a la FIFO…

En concreto, las direcciones para la lectura en crudo de los datos de aceleración, los registros son las siguientes:

- ACC_X_H → 0x3B
- ACC_X_L → 0x3C
- ACC_Y_H → 0x3D
- ACC_Y_L → 0x3E
- ACC_Z_H → 0x3F
- ACC_Z_L → 0x40

Siendo X, Y, Z los ejes de dirección de la aceleración y H y L indicativos del MSB (Most Significant Bit) y el LSB (Less Significant Bit).

Los datos leídos llegan en hexadecimal y complemento a 2, por lo que, para un eje concreto, hay que concatenar el registro con el bit más significativo con el del bit menos significativo y pasarlo a decimal con signo.

Para ello hay que coger el bit más significativo, si es un 0, hay que transformar el valor a un número decimal. Si es un 1, hay que invertir todos los bits, sumarle uno y pasarlo a número decimal. Al número obtenido hay que añadirle el signo negativo.

También existen registros de test con los que el MPU realiza un *autotest* de los distintos sensores para que los usuarios puedan testear eléctrica y mecánicamente los componentes del chip. Además, también cuenta con los registros lógicos para la configuración de los sensores y la FIFO, así como para la lectura de extraída o almacenada por estos. Por último, hay registros también para la configuración del I2C, bien para indicar que se va a utilizar un modo multimaestro, como para la configuración de transferencia de los distintos esclavos.

Para la lectura en crudo de la aceleración, primero habría que despertar al MPU para que comenzara a sensar datos. Para ello existen los registros de PWR_MGMT_1 y PWR_MGMT_2, situados en las direcciones 0x6B y 0x6C.

El primer registro, contiene un bit para el modo sleep, que por defecto está a 0. Cambiándolo a 1, el sensor acelerómetro despertará y comenzará a guardar datos en los otros registros. También se puede indicar el ciclo de reloj con el que se quiere muestrear, si se desea sensar la temperatura o no e incluso resetear los registros del dispositivo a su estado original. El segundo registro de power management se utiliza para configurar el acelerómetro en modo bajo consumo.

Así pues, para leer los valores de aceleración en X, por ejemplo, hay que utilizar el comando *i2cget*. Para ello, como se ha explicado anteriormente, hay que indicar la revisión de la placa en la que se está trabajando, en este caso revisión 2, que se indica con *–y 1*. Inmediatamente después, hay que indicar la dirección I2C en la que se encuentra el dispositivo acelerómetro (Recordemos que puede haber más de un dispositivo conectado al mismo bus I2C). En este caso es la dirección 0x68. Por último, hay que indicar el registro del dispositivo que se quiere leer. Como se ha comentado, los registros a leer serán el 0x3B y el 0x3C. (44)

```
pi@raspberrypi:~ $ sudo i2cget -y 1 0x68 0x3B
0x00
pi@raspberrypi:~ $ sudo i2cget -y 1 0x68 0x3C
0x00
```

Figura 63. Lectura de la salida del sensor sin haber ordenado al ADC que convierta

Al iniciar el MPU6050 e intentar leer los registros ACC_X_H y ACC_X_L, obtenemos como resultado que no hay valores almacenados en el dispositivo. Esto es porque para evitar un consumo innecesario, el MPU6050 está en modo sleep. Es por ello que hay que despertarlo escribiendo en la posición 0x6B.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE _RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

Figura 64. Distribución de los bits para el registro 0x6B

https://cdn-shop.adafruit.com/datasheets/ads1115.pdf

El bit 7 es para reiniciar el dispositivo, por lo que debe dejarse a 0. El bit 6 debe ponerse a 0 para sacar al sensor del modo sleep. Si se activa el ciclo, el MPU6050 saldrá de modo sleep para leer los valores capturados por los sensores y volverá a dormirse a una frecuencia indicada por los bits de LP_WAKE_CTRL (registro 0x6C, PWR_MGMT_2).

Los bits 0, 1 y 2, indican la fuente del reloj que, por defecto, es el oscilador interno de 8MHz.

Por tanto, hay que sacar al dispositivo del modo sleep y se pondrá en modo cíclico para que lea datos constantemente. Esto significará poner los 8 bits de este registro como 0010 0000, o lo que es igual 0x20:

```
pi@raspberrypi:~ $ sudo i2cset -y 1 0x68 0x6B 0x20
pi@raspberrypi:~ $ sudo i2cget -y 1 0x68 0x3B
0xf7
pi@raspberrypi:~ $ sudo i2cget -y 1 0x68 0x3C
0x48
```

Figura 65. Escritura en el registro para sacar el ADC del modo sleep y posterior lectura

Leer los datos del sensor es relativamente sencillo, sin embargo, es evidente que en este ejemplo no se está aprovechando el potencial del MPU, puesto que no se utiliza la memoria FIFO, ni el procesador DMP.

Dado que las posibilidades y la cantidad de registros son extensos, se optó por buscar una librería capaz de sacarle el máximo rendimiento al dispositivo y a los sensores que este tiene. Buscando por la red, se localizó una librería en el que se adapta el driver utilizado para Arduino a lenguaje C++ para ser utilizado en la Raspberry Pi. (45)

Esta librería cuenta con tres demos:

- Demo RAW: aquí se leen los datos directamente como llegan, tanto desde el sensor acelerómetro como del giróscopo. Esta sería la manera más sencilla de obtenerlos, aunque la menos precisa.

```
ypr  -150.52  -10.30   74.52    areal    239   4197   -565
ypr  -150.14  -10.67   74.26    areal   -733   3833   -775
ypr  -150.98  -10.93   74.25    areal   -386   4344   -535
ypr  -150.83  -11.50   73.90    areal   -838   3681   -711
ypr  -149.15  -12.75   72.87    areal   -739   3802   -760
ypr  -147.08  -14.17   71.55    areal   -791   3873   -787
ypr  -145.08  -15.44   70.35    areal   -774   3853   -781
ypr  -144.73  -15.81   70.06    areal   -819   3880   -788
```

Figura 66. Valores obtenidos al lanzar la demo RAW

- Demo DMP: Este programa utiliza el DMP y la FIFO, lo cual lo hace más complejo pero los resultados obtenidos son más fiables y libra al microprocesador de la tarea de tener que procesar los datos ya que el dispositivo lo hace internamente.

- Demo 3D. Aquí, el autor de la librería hace una combinación de C++ y C para la representación de un objeto tridimensional que cambia su posición y rota al mismo tiempo que lo hace el sensor. Esta demo tiene un gran impacto visual para realizar demostraciones de su funcionamiento y, aunque no resulte útil como tal para la detección de las caídas, es conveniente echarle un vistazo cuando se trabaja con el acelerómetro por primera vez.

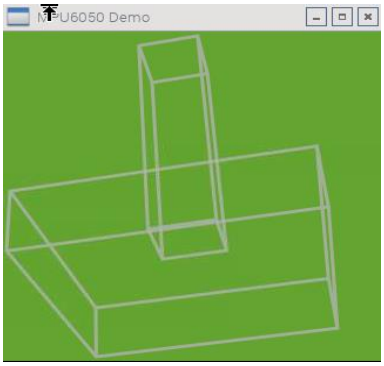


Figura 67. Objeto 3D que aparece al utilizar el demo 3D y que representa el movimiento del sensor

Así pues, descargando la librería y guardando todos los archivos en una misma carpeta, se puede editar el archivo *Makefile* para que compile el código personalizado, el cual se creará a partir de la demo del DMP, la segunda, por ser un término intermedio que no utiliza el visualizado 3D que es innecesario para la aplicación en cuestión, y por utilizar el procesador y la FIFO que proporcionarán estabilidad y precisión a los datos obtenidos.

Así pues, se realiza una copia del archivo dmp_demo.cpp y se guarda con otro nombre. Este nombre será el que haya que insertar en el Makefile para que compile todos los archivos linkados al mismo, donde se define, por ejemplo, la clase *mpu*, que tiene los métodos como leer la fifo, leer el giróscopo, etc…

Si se desea, puede editarse directamente el archivo de la librería, lo cual permite no tener que editar el archivo Makefile, el cual es más tedioso de comprender.

Una vez se obtiene acceso al código y se puede manipular como se desee, el primer paso será modificarlo para que guarde en un archivo *txt* los datos procedentes del acelerómetro y procesados por el DMP, así como la hora en que se han registrado esos datos. La hora se obtiene desde el terminal de Linux, y se añade a los valores obtenidos en los tres ejes concatenados y separados por “;”. Haciendo esto, se obtiene un archivo que puede ser abierto en Excel, permitiendo separar las columnas por el símbolo “;” dando lugar a las columnas X, Y, Z y una marca de tiempo. Así, se pueden dibujar en el eje Y de una gráfica las aceleraciones en los tres sentidos, y en el eje X el tiempo, con lo que se puede controlar la variación a lo largo del tiempo.

5.5.3 Conversor Analógico/Digital ADS1115

Antes de proceder a la obtención de datos de los sensores restantes, cabe hacer hincapié en el funcionamiento del ADC, el cual es el interfaz entre las salidas analógicas de los sensores, y las entradas digitales de la Rasperry Pi.

La salida del ADS1115 es lo que se conoce como un entero con signo. Esto significa que, de los 16 bits utilizados para formar cada palabra, el bit de mayor peso se utilizará para indicar si es un numero positivo o negativo, por lo que el valor de salida realmente solo serán 15 bits, lo que significa que hay 32.768 posibles valores de salida, de 0 a 32.767.

El valor de un bit estará determinado por el amplificador de ganancia programable (PGA), y en modo por defecto, la salida es ±6.144V. Dividiendo los 6.144V por la cantidad de posibilidades, se consigue un factor de 0.1875mV por bit.

Este rango de tensiones puede ser confuso y parecer que pueden leerse valores tan elevados de tensión, pero no se puede. Hay que notar la diferencia entre el rango de la PGA y el máximo voltaje medible. El rango programado determina el valor de un bit (o valor de escala), mientras que el máximo rango medible define qué entrada analógica se puede manejar de forma segura.

El máximo voltaje medible se establece por el voltaje que se aplique al chip, y su valor es 0,3 voltios mayor al voltaje aplicado. Superar este valor dañaría el chip. (32), (46)

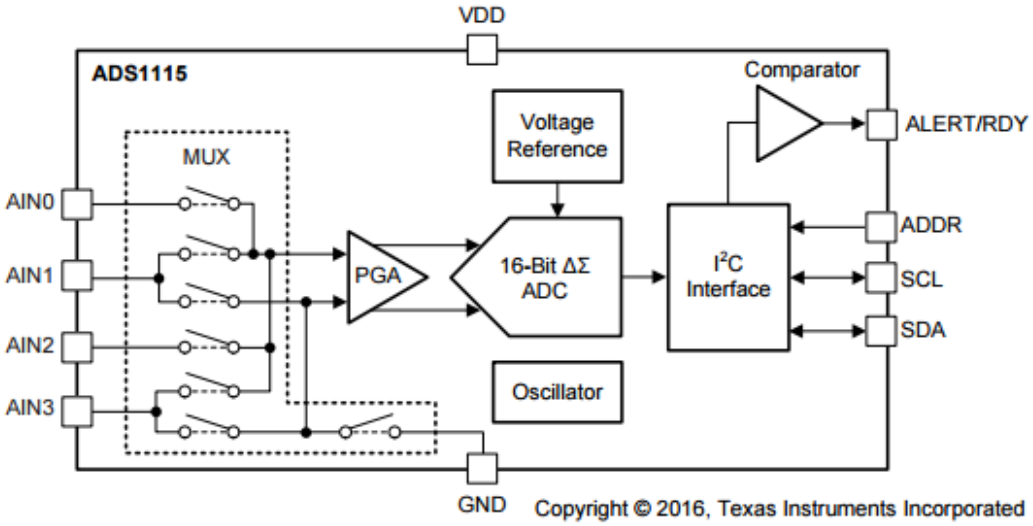


Figura 68. Esquema del ADS1115

https://cdn-shop.adafruit.com/datasheets/ads1115.pdf

Arriba se muestra el esquemático del ADS1115, en el que pueden verse sus 4 pines de entrada multiplexados a un amplificador llamado PGA. La salida del amplificador va al conversor ADC y la salida al interfaz I2C y de ahí se realizará la conexión a la Raspberry (47).

El ADC tiene distintos parámetros a configurar:

- Muestras por segundo: Este convertidor es capaz de transformar de 8 a 860 muestras por segundo. Este parámetro en realidad asigna el tiempo que se va a tardar en convertir cada muestra, no por la frecuencia de muestreo. Si se introduce un valor de 200, cada conversión tardará 1/200 segundos. Esto significa que, para menores frecuencias de muestreo, la conversión se realizará durante más tiempo, siendo el resultado la media de todas las entradas obtenidas durante este periodo.

- Canal de entrada: El dispositivo tiene cuatro pines de entrada, de A0 a A3. Se puede configurar cuál de estas cuatro entradas debe leer el dispositivo al realizar la conversión. El dispositivo acepta señales single-ended o diferenciales.
- PGA (Amplificador de ganancia programable): Antes de que el voltaje que entra por los pines llegue al ADC, pasa por el amplificador. La ganancia de este amplificador puede ser modificada, lo cual permite medir voltajes más bajos con resolución aumentada a costa de un rango de medida menor.

FSR	LSB SIZE	CONFIGURACIÓN
±6.144V	187.5 µV	000
±4.096V	125 µV	001
±2.048V	62.5 µV	010
±1.024V	31.25 µV	011
±0.512V	15.625 µV	101
±0.256V	7.8125 µV	≥110

- Conversión continua: El dispositivo tiene dos modos de conversión. En el modo por defecto, la conversión se inicializa por parte de la Raspberry Pi. El resultado se almacena en un registro del ADS1115 que esta leerá. El segundo modo es el modo de conversión continua. En este modo, en cuanto una conversión se realiza, comienza una nueva. Los resultados se almacenan en el registro de conversión sobrescribiendo el anterior. La Raspberry puede leer cuando quiera el registro, aunque si lo hace más rápido que la frecuencia de muestreo del ADC, leerá el mismo valor dos veces.
- Comparador: Se puede configurar el ADS1115 para que trabaje como un comparador tradicional o uno de ventana. Cuando el comparador detecta un valor que excede el límite indicado, pone a nivel alto el pin RDY/ALERT.

Para configurar los distintos parámetros comentados, hay que acceder a los registros de configuración del ADS1115.

Los registros pueden ser de lectura, de escritura, o ambos. Normalmente se utilizan para configurar o para mostrar información sobre algún aspecto del hardware. El ADS1115 únicamente tiene cuatro registros accesibles por el interfaz I2C. El registro de conversión contiene el resultado de la última conversión. El registro de configuración se utiliza para cambiar los modos de operación del dispositivo y para preguntarle su estado. Por último, los registros *Lo_tresh* y *Hi_tresh*, sirven para establecer los márgenes inferior y superior del comparador. (47)

El registro de conversión se encuentra en la posición 0x0, y tiene un tamaño de 16 bits. El resultado de la última conversión se almacena en este registro en formato binario en complemento a 2, enviándose primero el MSB y luego el LSB.

El registro de configuración, situado en la dirección 0x1, controla toda la configuración del conversor y sus bits están distribuidos de la siguiente forma:

Bit [15]: Al poner a 1 este bit, el ADS1115 realizará una conversión. Al leer este registro, si este bit es un 0, significa que se está realizando una conversión, si es 1, el dispositivo está idle sin realizar ninguna conversión.

Bits [14:12]: Estos bits configuran la entrada del multiplexor. Por defecto, la entrada positiva del PGA está conectada a AIN0 y la negativa a AIN1. Seleccionaremos 100, que es la comparación de AIN0 con GND.

Bits [11:9]:_ Se utilizan para controlar la ganancia del amplificador. Por defecto está situado en 010, FSR = ±2.048 V.

Bit [8]: Modo de operación del dispositivo. 0 para modo continuo, y 1 para modo single-shot.

Bits [7:5]: Velocidad de transmisión de datos. Por defecto está a 100, 128 SPS.

Bit [4] Elige entre el modo tradicional de comparación(0) o el modo de comparador por ventana(1)

Bit [3]: Activa/desactiva el modo comparador. Por defecto está a nivel bajo, desactivado.

Bit [2]: Este bit decide si el pin ALERT/RDY permanece después de haber sido activado o si se devuelve a 0 una vez los valores vuelvan a estar dentro de los márgenes indicados.

Bit [0:1]: Estos bits realizan dos funciones. Si están a 11, se desactiva el comparador y el pin ALERT/RDY se pone en alta impedancia. Si se coloca en otro valor, las funciones del pin ALERT/RDY se activan y su valor determina el número de conversiones que deben exceder los márgenes antes de activar el pin de ALERT/RDY. (47)

Los registros de para los márgenes del comparador, no serán objeto de este documento.

Para la finalidad que tendrá el ADC, una configuración posible sería:

Bit	**[15]**	**[14:12]**	**[11:9]**	**[8]**	**[7:5]**	**[4]**	**[3]**	**[2]**	**[1:0]**
Valor	1	100	000	0	100	0	0	0	11

Este valor en hexadecimal es 0xC083 con lo que escribiendo este valor en el registro 0x1 del ADS1115, obtendremos una conversión continua con un FSR de ±6.144V y la velocidad de conversión por defecto.

Con las librerías instaladas del punto 5.5.2 para la utilización del interfaz I2C, el proceso a realizar será el mismo. Primero hay que detectar la dirección del bus en la que se encuentra el dispositivo. (48)

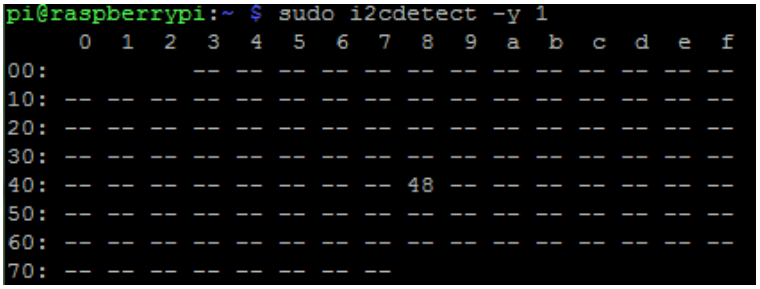


Figura 69. Detección del dispositivo conectado a I2C en la dirección 0x48

Esta vez la dirección I2C del dispositivo conectado es la 0x48. Nótese que para los sensores que vienen a continuación, el sensor de gas y de temperatura, al utilizar los comandos de I2C no se está interactuando con el propio sensor, sino con el ADS1115, el convertidor analógico digital.

Para esto, primero hay que realizar el set para configurar el ADC como se ha explicado anteriormente. Si se tratara de leer el registro de conversión sin haber configurado nada, ni siquiera se habría ordenado al ADS1115 que realice una conversión, por lo que el valor leído sería 0.

Por tanto, se asigna el valor 0xC083 en el registro 0x1 del dispositivo ubicado en la dirección 0x48. En la comunicación con el ADS1115 el LSB se debe enviar primero, por lo que el valor a escribir será 0x83C0:

```
pi@raspberrypi:~ $ sudo i2cset -y 1 0x48 1 0x83C0 w
```

Figura 70. Configuración del ADS1115 a través de I2C

La letra w se utiliza para indicar que se está escribiendo una palabra de 16 bits.

```
pi@raspberrypi:~ $ i2cset
Usage: i2cset [-f] [-y] [-m MASK] [-r] I2CBUS CHIP-ADDRESS DATA-ADDRESS [VALUE]
... [MODE]
  I2CBUS is an integer or an I2C bus name
  ADDRESS is an integer (0x03 - 0x77)
  MODE is one of:
    c (byte, no value)
    b (byte data, default)
    w (word data)
    i (I2C block data)
    s (SMBus block data)
    Append p for SMBus PEC
```

Figura 71. Caracteres utilizados para indicar la longitud de las palabras

Esto hará que de forma continua se lea el valor de tensión en el pin A0 y se compare con GND. Este voltaje se convertirá en bits con un FSR de ±6.144V, lo que significa que cada bit recibido a la salida del ADC serán 187.5 µV, por lo que al leer la dirección 0x0 del ADC, habrá que multiplicar el número de bits recibidos por 187.5 x 10^{-6}

Para leer este valor, el comando a utilizar será el *get*:

```
pi@raspberrypi:~ $ sudo i2cget -y 1 0x48 0 w
```

Figura 72. Lectura del registro de almacenamiento del ADC

5.5.4 Sensor de gas MQ-4

Para la detección de gas, es necesaria la combinación de distintos sensores para captar un rango más amplio de gases nocivos e incrementar la veracidad de los datos, ya que un único sensor puede detectar más de un gas a la vez y, en algunos casos, dichos gases se repiten. En este caso, se ha realizado la prueba con un único sensor, aunque incrementar el número de sensores no es complicado, ya que, al tener ya en el sistema un conversor ADC de varios canales de entrada cuyo protocolo de salida es el I2C, es posible conectar varios sensores en una misma Rasperry Pi.

El proceso de obtención de datos en este sensor prácticamente ha sido explicado en el punto anterior. Pues los datos a leer serán los del convertidor ADS1115, aunque sea otro sensor el que escriba en su registro.

El sensor debe conectarse al convertidor analógico-digital, y la salida de este a los pines de I2C. Esto se hace conectando el sensor de gas a VCC (5V), a GND y la salida A0 al pin A0 del ADC.

El conversor irá conectado a una alimentación de 3.3V, a GND y los pines SCA y SCL a los pines SCA y SCL de las Raspberry.

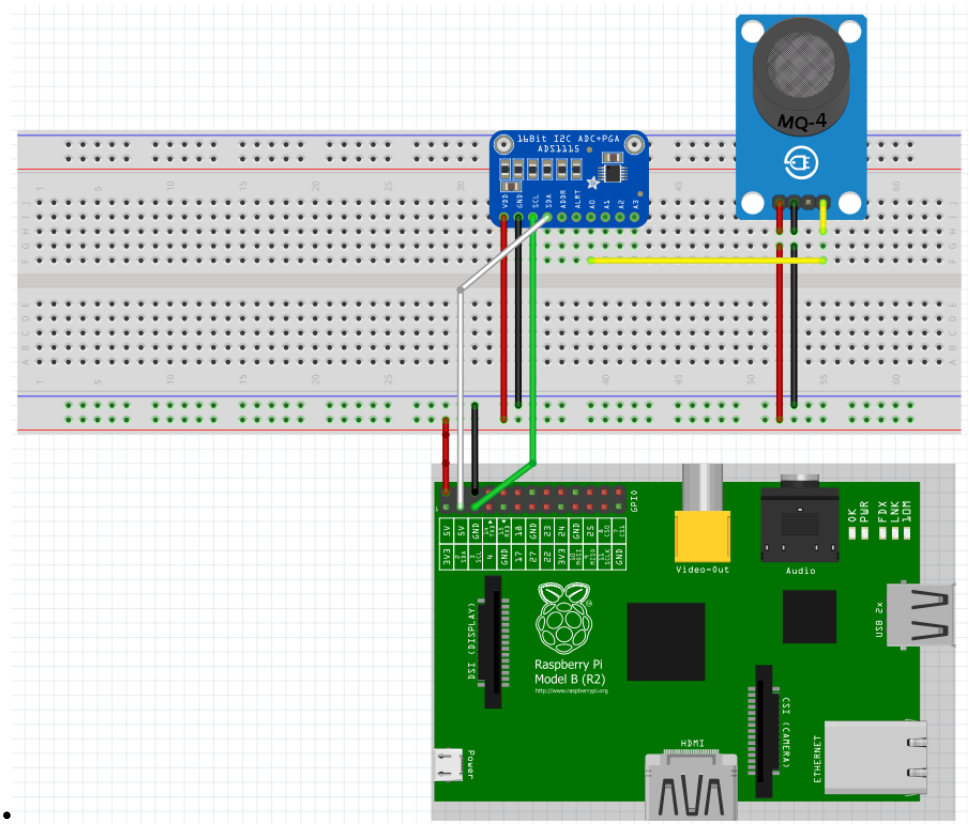


Figura 73**. Sensor MQ-4 conectado al ADS1115 y a las Raspberry Pi**

Tras calentarse durante unos 20 segundos, el sensor ofrecerá una tensión de salida mayor a medida que aumente la concentración de gas.

El circuito equivalente de este sensor son dos impedancias: Rs y RL. RL es la carga, y cuenta con una resistencia nominal de 20kΩ, mientras que Rs depende del propio sensor de gas y tiene una resistencia nominal que varía de 10k a 60k, dependiendo de las condiciones de temperatura ambiente, humedad y concentración de gas.

Según informa el distribuidor, se debe calibrar el sensor primero para una concentración de 5000ppm y así obtener el valor nominal de Rs. Con una resistencia nominal acotada, habría que hallar la ecuación de la recta para cada tipo de gas según la impedancia del sensor Rs en cada momento, la cual sería capaz hallarla sabiendo el voltaje de salida tras pasar por el sensor.

Sin embargo, al no disponer de una forma de medir y estabilizar la concentración de gas, es complicado obtener un valor de resistencia que ofrezca luego valores coherentes en partes por millón.

El valor de bits obtenido debe multiplicarse por 187.5µV por bit y, a través de esa tensión hay que calcular el valor de la resistencia Rs y con ella la concentración de gas en ppm.

5.5.5 Sensor de temperatura LM35

El sensor LM35 tiene tres pines. VCC debe ser una tensión entre 4 y 20V, por lo que se alimentará a 5V. El segundo pin, irá a la entrada A0 (A1 si A0 ya está ocupada) y, por último, el tercer pin a GND.

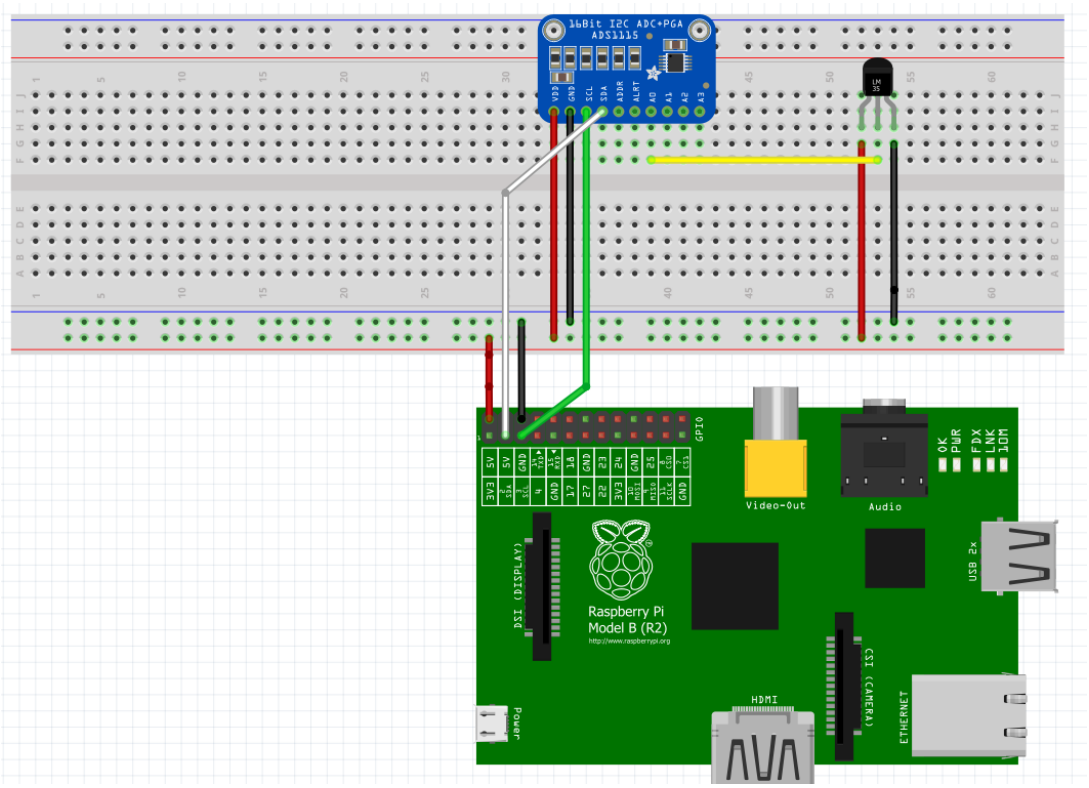


Figura 73. Conexión del LM35 con el ADS1115 y las Raspberry Pi

El sensor LM35 ofrece 10mV/ºC, por lo que es bastante sencillo de obtener la temperatura medida con el ADC.

```
pi@raspberrypi:~ $ i2cget -y 1 0x48 0x0 w
0x4705
```

Figura 74. Lectura del valor del sensor LM35 a través del ADC

Utilizando el comando i2cget se obtiene un número de bits. Para calcular el voltaje correspondiente, se deberá multiplicar el número de bits obtenido por el voltaje por bit que se obtiene del FSR. Además, cabe tener en cuenta que el LSB va primero, por lo que, en la imagen superior, el valor hexadecimal obtenido en verdad es 0x0547, es decir, una cantidad convertida de 1351 bits. Estos bits multiplicados por el voltaje por bit, 187.5µV da como resultado 0.2533V. Teniendo en cuenta que el voltaje de salida del sensor son 0.01mV por voltio, el resultado son 25.33º, con un error de ±¼°C.

Capítulo 6. Código

En este apartado se realizará la explicación de los programas que se han desarrollado para controlar los sensores. En el anexo de este documento se adjuntarán el programa controlador el pulsómetro, el del sensor de gas y el programa principal de control del giróscopo, ya que el del sensor de temperatura es muy parecido al segundo y se repiten muchas partes del código. Por otra parte, código de lectura del giróscopo únicamente incluye unas modificaciones que se han realizado sobre un software de código abierto que incluye varios módulos, por lo que solamente se adjunta el programa principal por no alcanzar una extensión desorbitada con los códigos, sin embargo, es necesario la compilación de todos ellos cuando se quiera lanzar el programa.

El código que controla cada sensor se ha desarrollado por separado y se ha unificado sobre el programa de lectura del giróscopo por mayor sencillez de adaptación, ya que este es muy complejo y es el único que incluye más módulos necesarios para su compilación. Así pues, en los anexos se dispondrá de los códigos por separado, por sencillez a la hora de realizar explicaciones y probar los sensores. Sin embargo, las funciones desarrolladas en los programas suelen ser reutilizadas para todos los sensores en general.

Todos los códigos han sido desarrollados en C++ por coherencia con el software utilizado para controlar las gafas Epson Moverio, aunque, si se quisiera, no habría ningún problema en utilizar otros lenguajes como Python para controlar los sensores si se han comprendido las características básicas explicadas anteriormente.

6.1 Pulsómetro.cpp

El software que se encarga de leer el pulsómetro ha sido desarrollado basándose en la herramienta *gatttool*. Tras consultar multitud de fuentes y ver que en todas se utiliza esta herramienta para la lectura e interacción con dispositivos bluetooth Low Energy se decide realizar la llamada desde el código a dicha herramienta mediante el modo *no interactivo*, es decir, lanzar de golpe todas las variables y funciones en una única llamada, sin esperar a una interacción con el sensor.

El código utiliza la herramienta mencionada para acceder a las características del dispositivo que permiten obtener el ritmo cardíaco del usuario y el estado de la batería. En cada iteración, se muestra por consola el valor del pulso y se escribe en un fichero donde se indicará la hora en que se hizo el registro y el valor obtenido en ese momento. Así, además de poder informar en tiempo real al usuario de su ritmo cardíaco, se puede tener un histórico de las pulsaciones del bombero durante toda la operación.

En primer lugar, se inicializa el dispositivo bluetooth de la Raspberry y se guarda la fecha en el fichero de texto. Después durante un bucle infinito se leerán y registrarán los valores del pulso cardíaco. En la primera iteración se llamará a la función *Lee_Batería*, que accede a la característica que proporciona información sobre el estado de la batería. Esta función realiza una llamada a otra llamada *GetStdoutFromCommand*, que se encarga de lanzar un comando por

consola y registrar la respuesta. Con esto, se pueden obtener los valores que el sensor devuelve al preguntar tanto por la batería como por el ritmo cardíaco.

En concreto, el comando que se lanza es:

```
sudo gatttool -i hci0 -b D0:5F:B8:41:16:2F --char-read -a 0x025
```

Dicho comando accede a la herramienta gatttool, en la dirección física especificada y lee el carácter situado en el registro 0x025, que como se ha explicado en apartados anteriores, es la dirección del registro donde se ubica el valor de porcentaje de batería restante.

Al lanzar este comando, si todo ha ido correctamente, la respuesta obtenida será “Characteristic value/descriptor:” seguida del valor en hexadecimal del valor de carga de la batería. Si es así, el siguiente paso es localizar la posición del string en la que se encuentra el valor y aislarla de los demás caracteres del string.

Con el valor hexadecimal almacenado en un string, se realiza la llamada a la función HextoDec, que convierte el string hexadecimal en un string decimal después de realizar una serie de conversiones en base al tipo std::stringstream, convirtiendo un string en un simple flujo de bits y representando esos bits como un entero, que será transformado a decimal y convertido de nuevo a string.

Tras realizar la lectura y transformación con éxito, se guardará este valor en el fichero asociado al pulsómetro y se dispondrá entonces a leer el valor del pulso registrado por el sensor.

Antes de cada lectura, se obtendrá el valor de la hora actual en la que el sistema operativo se encuentra:

```
system("date +%T");
```

Con +%T, se quita el día, mes y año, dejando únicamente horas minutos y segundos.

Tras registrar la hora en el fichero, se lanza el siguiente comando:

```
system("sudo timeout 1.5 gatttool -i hci0 -b D0:5F:B8:41:16:2F --char-write-req -a 0x013 -n 0100 --listen > lastlog.txt");
```

Al lanzar *char-write-req*, gatttool escribe un valor en el registro y espera la respuesta del dispositivo. Al contrario que en la batería, el registro no se guarda directamente, sino que hay que especificarlo.

Una vez escrito, el dispositivo activará sus notificaciones y comenzará a registrar y enviar los valores del pulsómetro. Es por eso que, para no perder el control de la Raspberry Pi, hay que lanzar el comando con un timeout que detenga la lectura. El contenido de la respuesta se guardará en un fichero de texto que contiene únicamente la última respuesta recibida, para que si se apaga la Raspberry a mitad de ejecución se pueda tener guardado el último registro, además del histórico ya guardado. Esto se hizo en un inicio por si la Raspberry se apagaba y se perdían los datos, pero luego se comprobó que es lo suficientemente robusto como para no perder el histórico si hay un corte a mitad de ejecución.

Tras obtener la respuesta del dispositivo, se realiza el mismo procedimiento que para la batería, localizando el valor de las pulsaciones, aislándolo y transformando el valor hexadecimal en decimal. Finalmente, este valor se registrará en el fichero de texto.

Como aporte de robustez y seguridad, si la Raspberry recibe como respuesta un string distinto al esperado, se coloca un contador que resetea el Bluetooth del dispositivo para volver a establecer la conexión.

El resultado de este código se ha testeado de la siguiente manera: Con la banda pulsométrica colocada, se ha combinado el reposo con el ejercicio anaeróbico de forma periódica, con variaciones en ritmo e intensidad. Además, se quiso probar la capacidad de recuperación del programa dejando un largo tiempo la Raspberry alejada, incapaz de conectarse a ningún dispositivo Bluetooth y regresando más tarde para comprobar si la conexión regresa.

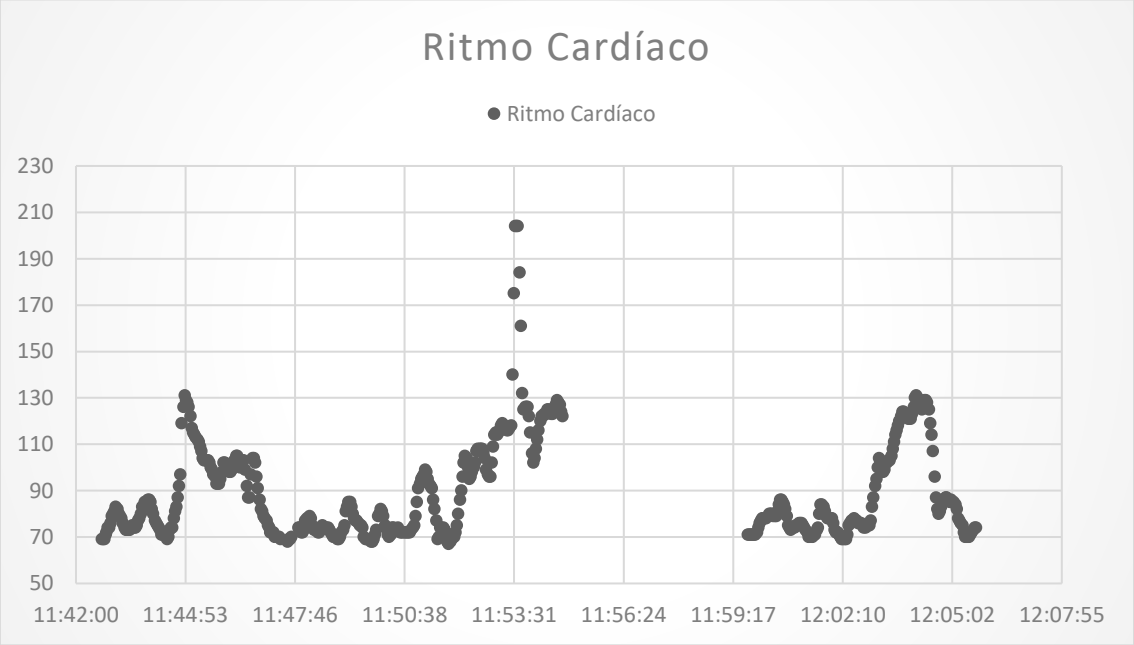


Figura 75. Gráfica del pulso cardíaco variando distintos factores

En los periodos de reposo, el ritmo cardíaco se encuentra estable en torno a las 60-80 pulsaciones por minuto, mientras que en determinado momento alcanza un valor de 200. Además, durante unos 5 minutos se abandona la Raspberry Pi y cuando ambos dispositivos vuelven a estar próximos, se reestablece la conexión automáticamente.

6.2 Gas_sensor.cpp y Temperature_sensor.cpp

Al igual que con el pulsómetro, los sensores de gas y de temperatura registran los datos que son enviados a la Raspberry Pi y almacenados en un archivo de texto junto a la hora exacta para poder representarlos posteriormente en una gráfica.

Como en estos dos casos el procedimiento es muy similar se explicarán en conjunto. El primer paso es la creación del fichero, donde se colocarán los datos en forma de columnas separadas por puntos ya que así resulta muy sencillo abrirlos con Excel y representarlos. Después, se inicializa el conversor analógico digital como se ha explicado en apartados anteriores. Esto se hace con la siguiente línea:

```
system("i2cset -y 1 0x48 1 0x83C0 w");
```

Como se está realizando una conversión continua de la entrada del sensor conectado en la dirección I2C 0x48, en el bucle solamente habrá que realizar la lectura de dichos datos.

Para ambos sensores se utiliza la función LecturaADC, que lee el registro en el que se guarda el resultado de la conversión analógica/digital y lanza otra función que permite su interpretación. La función LecturaADC comienza enviando el siguiente comando:

```
    result = GetStdoutFromCommand("i2cget -y 1 0x48 0 w");
```

Mediante esta línea, se guarda en la variable result el resultado de la conversión de los datos del sensor.

El valor obtenido son dos dígitos de dos octetos cada uno. El primer par de octetos es el MSB y el segundo el LSB por lo que hay que dividirlos en dos variables distintas y recomponer el string con el LSB primero y el MSB después.

Con el string reordenado, se utiliza la función HextoDec_C2, que realiza la conversión de hexadecimal a decimal, pero en complemento a 2, es decir, teniendo en cuenta el signo dependiendo del valor del bit más grande. Con esto se obtiene el valor decimal del número de bits correspondientes al voltaje de entrada del ADC, por lo que para obtener el propio voltaje hay que multiplicar el número de bits por el voltaje por bit, que dependerá del FSR explicado anteriormente. Con el valor del voltaje obtenido, ya se puede introducir en el archivo creado al principio.

El LM35 ofrece 10mV por ºC con un offset de 0V, por lo que únicamente con la tensión es sencillo obtener el valor de temperatura en el que se encuentra el bombero multiplicando el valor obtenido por 100.

Aquí se muestra el resultado de medir durante un largo periodo la temperatura ambiente. Al principio se toca el sensor para aumentar su temperatura. Luego se deja en reposo para observar su evolución.

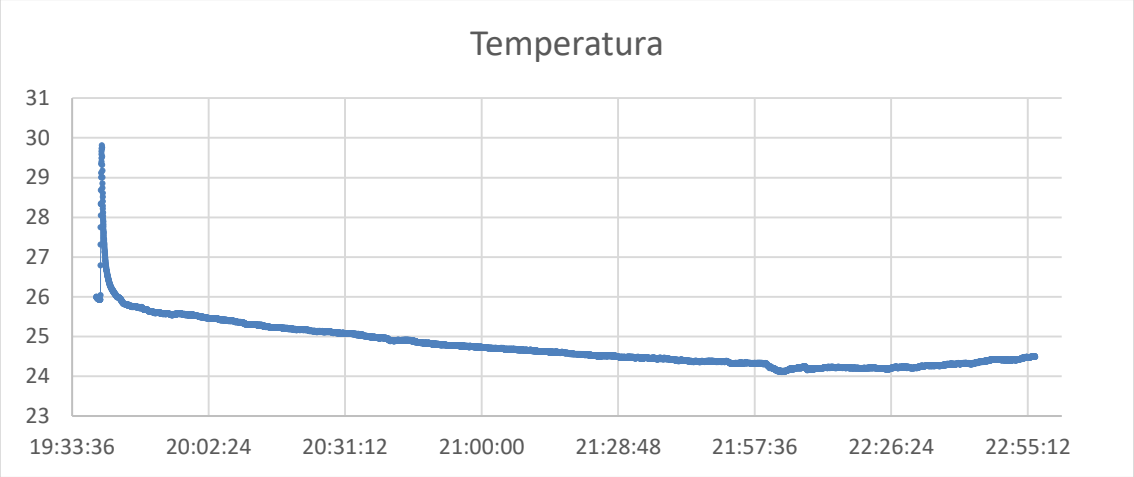


Figura 76. Respuesta del sensor de temperatura durante varias horas. Al principio se toca el sensor para que la temperatura cambie

También se muestra el resultado por pantalla que ofrece el sensor de temperatura cuando está realizando las medidas. Se añade el valor hexadecimal y el decimal para que se observen los valores que se obtienen antes de llegar al valor final de temperatura:

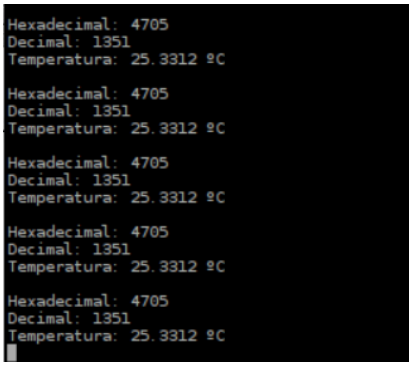


Figura 77. Mensajes mostrados por consola al medir la temperatura

Para el sensor de gas, se han realizado pruebas y con una concentración mínima de gas, el voltaje a la salida del sensor es de en torno a 0.1V, mientras que cuando se aplica gas este asciende hasta 1.5V.

El resultado puede comprobarse en la siguiente imagen, donde se ve el sensor de gas en reposo y al cual se le aplica con un encendedor una ráfaga de gas. El sensor reacciona rápidamente y obtiene el nivel de gas, mientras que para el retorno al estado inicial su respuesta es más suave.

En la parte derecha de la gráfica se coloca el sensor dentro de una caja que mantendrá el gas. Aquí se puede observar como los niveles de concentración del gas se reparten por la caja y como se mantiene más estable el nivel de concentración a lo largo del tiempo. Una segunda ráfaga de gas aumenta la concentración sobre el nivel ya existente.

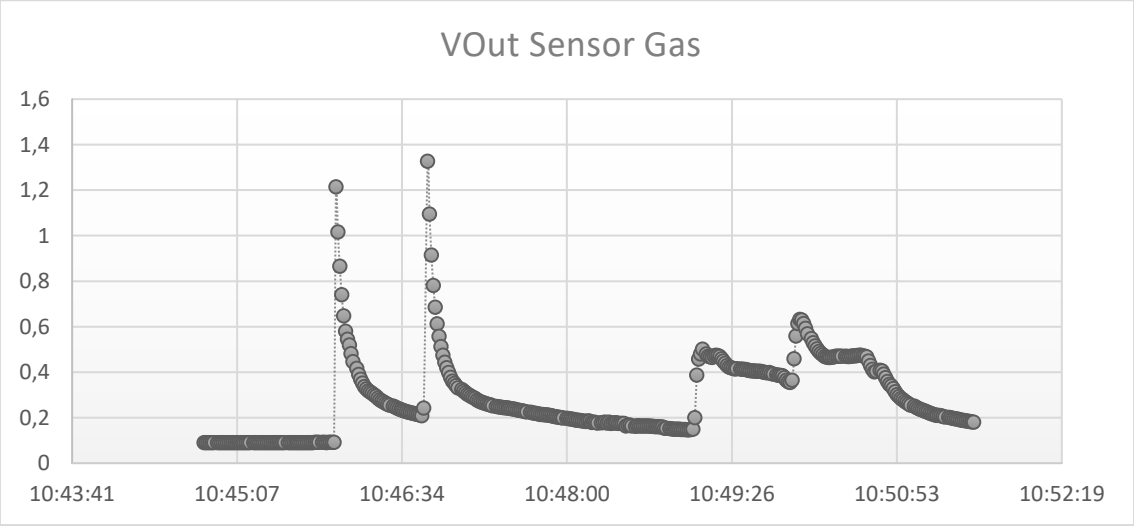


Figura 78. Gráfica del sensor de gas. Primero se realizan dos ráfagas de gas directamente. Luego se introduce en una caja y se llena con gas.

El resultado mostrado por consola es igual que el del sensor de temperatura, mostrando el voltaje a la salida.

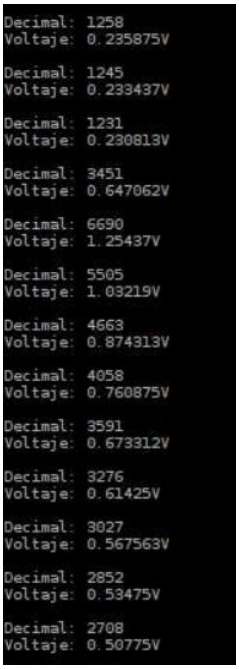


Figura 79. Lectura por consola de los valores que proporciona el sensor de gas

6.3 Acelerómetro_dmp.cpp

Para realizar las medidas de aceleración, se recurre al módulo de código abierto que contiene tres demos (49).

La demo que se utilizará será la intermedia entre los datos conforme llegan del acelerómetro y el ejemplo en 3D. Esta demo utiliza distintos submódulos que se deben incluir al realizar la compilación. El programa principal del que cuelgan los demás módulos se llama demo_dmp.cpp, el cual se copia y se renombra a acelerómetro_dmp.cpp. Este archivo se mantiene igual, pero se añade en la parte de inicialización un trozo de código en el que se inicializa el fichero donde se guardarán los datos. En el bucle en el que se leen los valores del DMP, se ha modificado el código para extraer el string correspondiente al valor hexadecimal leído para transformarlo a un valor decimal y colocar los datos de la aceleración en X, Y y Z, así como una marca de tiempo, separados por puntos y coma. Del mismo modo que en los otros sensores, el resultado puede ser abierto como un archivo de Excel y puede ser representado fácilmente.

Es en este bucle donde también se añaden los códigos de lectura del pulsómetro, del sensor de gas y del sensor de temperatura, reutilizando las funciones de conversión de hexadecimal a decimal, o la de envío de comando y obtención del resultado.

Tras ejecutar el programa y agitar el acelerómetro, se pueden observar los cambios en las aceleraciones en función del tiempo:

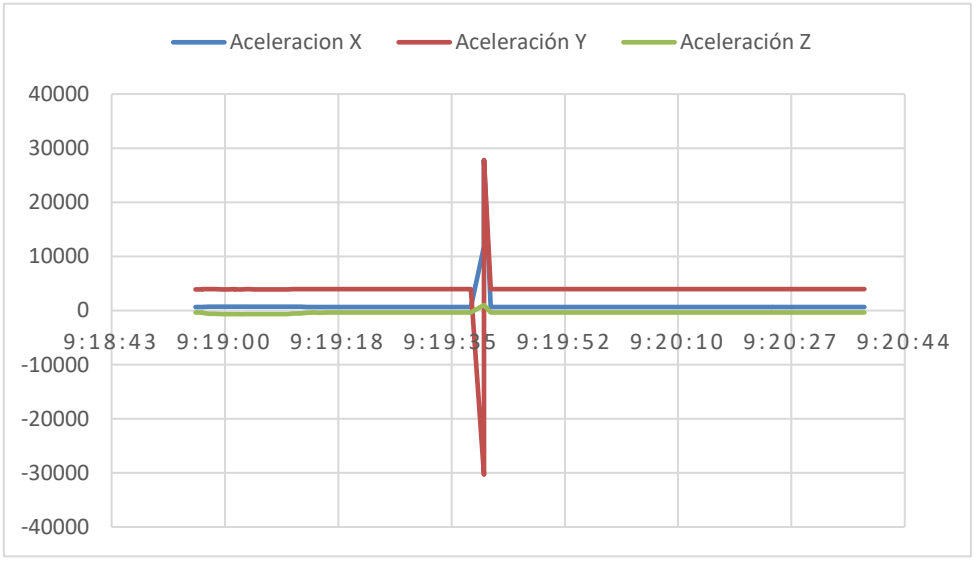


Figura 80. Variación de la aceleración tras mover bruscamente el sensor

Donde se puede observar que toda la aceleración ha sido en el eje Y, mientras que el eje Z no se ha movido prácticamente.

Así pues, desglosando los tres ejes en tres gráficas independientes, puede observarse mejor la aceleración:

Figura 81. Desglose de las aceleraciones en los 3 ejes

El acelerómetro irá pegado a la espalda del bombero con el eje Y en la perpendicular al plano de tierra. Así pues, para cada iteración en la que se lea el valor de la gravedad se guardará en un registro el valor anterior. La aceleración en el eje Y variará constantemente su valor, pues el bombero estará en movimiento se podrá agachar incluso tumbarse en el suelo. Es por eso, que para detectar una caída se hará cuando se detecte un cambio muy brusco de la aceleración, en 5000 unidades o más:

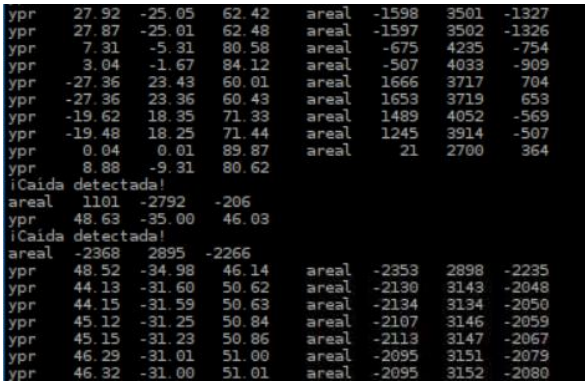
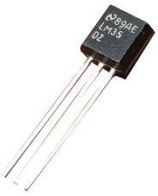
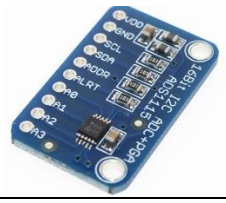


Figura 82.Detección de una caída

Capítulo 7. Presupuesto

En el siguiente presupuesto se va a detallar el coste de los materiales utilizados:

	**Raspberry Pi B Rev. 2 (50)**	**33.11 €**
	Gafas Epson Moverio BT-100 (51)	616 €
	Cámara Térmica FLIR (52)	147.54€
	Sensor de temperatura LM35 (53)	1.34 €
	Sensor de gas MQ-4 (54)	2.68 €

	Acelerómetro + Giróscopo MPU6050 (55)	5.25 €
	Banda pulsométrica Coospo h8 (56)	22.86 €
	Dongle Bluetooth 4.0 (57)	10.51 €
	Conversor Analógico/Digital ADS1115 (58)	3.74 €
	Placa ProtoBoard (59)	3.10€
	Total:	**846.13 €**

Capítulo 8. Conclusiones

El sistema que se está desarrollando incluye muchos subsistemas que aportan un valor añadido y una serie de ayudas al bombero que lo equipe. Por la naturaleza del entorno hostil en el que se encontrará un bombero durante una operación, es importante dotarles de medios como este para minimizar los riesgos a los que se exponen y tener un mayor control de la situación a través de sensores que permiten evaluar el entorno y de las gafas de realidad aumentada que juegan un papel clave a la hora de ofrecer comodidad en el acceso instantáneo y constante a estos datos. Además, por el mismo motivo, la seguridad y la robustez debe ser un punto clave a controlar, pues la obtención de datos debe hacerse con un bajo periodo y se debe garantizar que todas las señales, desde los datos capturados por la cámara térmica hasta los sensores equipados en el traje, se transmitan desde el microcontrolador a las gafas independientemente del entorno y las condiciones electromagnéticas en las que se encuentre el bombero. Es por eso por lo que se ha investigado el protocolo MIPI, que incluye, entre otros, el CSI, un protocolo para el envío de señales entre un microcontrolador y una cámara y el DSI, el protocolo para el envío entre el microcontrolador y un display.

El estándar MIPI está presente en todos los dispositivos móviles de nueva generación, tablets, ordenadores y cada vez más dispositivos. Se ha localizado a Epson Moverio, fabricante de las gafas utilizadas, como empresa registrada en la Alianza MIPI, por lo que se confirma las gafas utilizan este protocolo. Aunque no se ha conseguido realizar la conexión cableada entre la Raspberry Pi y las gafas Epson Moverio BT-100, se ha investigado sobre el funcionamiento de las distintas capas físicas que posee este estándar y de cómo las señales se envían a través de líneas cuyo número puede variar de 1 hasta 4 y que permiten al dispositivo trabajar en modo Low Energy o High Performance gracias a las capas físicas M-PHY, C-PHY y D-PHY, por lo que se allanado el camino hacia un nuevo proyecto en el que se trate de descifrar cómo se realiza la comunicación entre el dispositivo táctil de las gafas y las propias gafas para sustituir el dispositivo por un microcontrolador que tenga acceso tanto a los sensores como al display.

Por otra parte, se ha avanzado en la construcción del prototipo de sistema para incorporar al traje de bomberos y dotar de la capacidad de sensar el entorno del usuario para utilizar esos datos y proyectarlos en las gafas de tal manera que el bombero sea consciente en todo momento de la temperatura a la que está expuesto, de si hay una alta concentración de gases peligrosos o muy inflamables en el aire, su ritmo cardíaco e indicar y alertar a los demás compañeros de que cierto bombero ha sufrido una caída.

Estos sensores cuentan con distintos protocolos para la transmisión de datos, en concreto el Bluetooth Low Energy y el I2C, por lo que se ha realizado una explicación detallada y sencilla del funcionamiento de estos interfaces de comunicación, así como el funcionamiento y las características de cada uno de los diferentes sensores que componen el sistema.

En resumen, se ha desarrollado una pequeña parte de un proyecto más grande y complejo que permitirá a los bomberos ofrecer mayor seguridad y sortear algunas de las dificultades a las que se ven sometidos en operaciones de rescate y permitirá una monitorización de su estado y el de su entorno, además de dar lugar a una plataforma sobre la que introducir más sensores y más herramientas a medida que vaya surgiendo una necesidad.

Capítulo 9. Referencias

1. **Cindy Quarters, traducido por Francisco Roca.** Los riesgos de ser un bombero. [En línea] http://pyme.lavoztx.com/los-riesgos-de-ser-un-bombero-10903.html.

2. **Wikipedia.** Realidad Aumentada. [En línea] https://es.wikipedia.org/wiki/Realidad_aumentada.

3. **Saar, Hezi.** MIPI inside Augmented Reality Headsets. [En línea] 2016 de Agosto de 25. https://blogs.synopsys.com/onthemove/2016/08/mipi-inside-augmented-reality-headsets/.

4. **balearicdynamics.** Essential Raspberry Pi Peripherals #1: GPIO and Other Connectors. [En línea] https://www.element14.com/community/community/raspberry-pi/raspberry-pi-accessories/blog/2015/08/26/essential-raspberry-pi-peripherals-1-gpio-and-other-stuff.

5. **Raspberry Pi.** AN INTRODUCTION TO GPIO AND PHYSICAL COMPUTING ON THE RASPBERRY PI. [En línea] https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/.

6. **stevenvh.** Electrical Engineering (Foro). [En línea] https://electronics.stackexchange.com/questions/37814/usart-uart-rs232-usb-spi-i2c-ttl-etc-what-are-all-of-these-and-how-do-th.

7. **Vis, Peter J.** Raspberry Pi CSI Camera Interface. [En línea] http://www.petervis.com/Raspberry_PI/Raspberry_Pi_CSI/Raspberry_Pi_CSI_Camera_Interface.html.

8. —. Raspberry Pi LCD DSI Display Connector. [En línea] http://www.petervis.com/Raspberry_PI/Raspberry_Pi_LCD/Raspberry_Pi_LCD_DSI_Display_Connector.html.

9. —. Raspberry Pi DSI Connector Spec. [En línea] http://www.petervis.com/Raspberry_PI/Raspberry_Pi_LCD/Raspberry_Pi_DSI_Connector.html.

10. **ooterness.** Identifying a connector for an AR headset (Foro). [En línea] https://www.reddit.com/r/AskElectronics/comments/4e1fdh/identifying_a_connector_for_an_ar_headset/.

11. **JAE.** DD2 Series. [En línea] http://www.jae.com/jccom/en/connectors/detail/DD2.

12. **Wikipedia.** Mipi Alliance. [En línea] https://es.wikipedia.org/wiki/MIPI_Alliance.

13. **MIPI Alliance.** MIPI Alliance Specifications. [En línea] https://mipi.org/specifications.

14. **Peter Lefkin, Rick Wietfeldt.** Understanding MIPI Alliance Interface Specifications. [En línea] 1 de Abril de 2014. http://www.electronicdesign.com/communications/understanding-mipi-alliance-interface-specifications.

15. **MIPI Alliance.** MIPI Display Serial Interface (MIPI DSI). [En línea] https://mipi.org/specifications/dsi.

16. **Wikipedia.** Display Serial Interface. [En línea] https://en.wikipedia.org/wiki/Display_Serial_Interface.

17. **MIPI Alliance.** Evolving CSI-2 Specification. [En línea] https://www.mipi.org/sites/default/files/MIPI_CSI-2_Specification_Brief.pdf.

18. **Wikipedia.** Camera Serial Interface. [En línea] https://en.wikipedia.org/wiki/Camera_Serial_Interface.

19. **Marena, Ted.** Camera Serial Interface (CSI-2) sensors in embedded designs. [En línea] 8. http://www.electronicproducts.com/Digital_ICs/Communications_Interface/Camera_Serial_Interface_CSI-2_sensors_in_embedded_designs.aspx.

20. **MIPI Alliance.** Physical Layer. [En línea] https://mipi.org/specifications/physical-layer.

21. **Tektronix.** Understanding and Performing MIPI. [En línea] 61W_25772_0_HR.pdf.

22. **MIPI Alliance.** Physical Layers: M-PHY, D-PHY, C-PHY. [En línea] https://www.mipi.org/sites/default/files/PHY_Tech_Brief_20140916_0.pdf.

23. **twl.** MIPI DSI Display Shield/HDMI Adapter. [En línea] https://hackaday.io/project/364-mipi-dsi-display-shieldhdmi-adapter.

24. **Colorado Video.** High Definition DVI (HDMI) to Moverio Interface. [En línea] http://www.colorado-video.com/hi-def-moverio-interface/hi-def-moverio-interface.htm.

25. **Texas Instruments.** Demystify DSI I/F. [En línea] http://www.ti.com/lit/an/swpa225/swpa225.pdf.

26. —. LM35 Precision Centigrade Temperature Sensors. [En línea] http://www.ti.com/lit/ds/symlink/lm35.pdf.

27. **Chollo deportes.** Banda Pulsómetro ANT y Bluetooth. [En línea] http://www.chollodeportes.com/banda-pulsometro-ant-bluetooth-barata-21e/.

28. **Coospo.** Heart Rate Sensor H8. [En línea] http://coospo.com/h8.html.

29. **Playground Arduino.** MPU-6050 Accelerometer + Gyro. [En línea] http://playground.arduino.cc/Main/MPU-6050.

30. **Prometec.** USANDO EL MPU6050. [En línea] http://www.prometec.net/usando-el-mpu6050/.

31. **Hanwei Electronics.** TECHNICAL DATA MQ-4 GAS SENSOR. [En línea] https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-4.pdf.

32. **Henry's Bench.** Arduino ADS1115 Module Getting Started Tutorial. [En línea] http://henrysbench.capnfatz.com/henrys-bench/arduino-voltage-measurements/arduino-ads1115-module-getting-started-tutorial/.

33. **Punch Through.** How GAP and GATT Work. [En línea] https://punchthrough.com/bean/docs/guides/everything-else/how-gap-and-gatt-work/.

34. **Bluetooth SIG.** GATT Specifications. [En línea] https://www.bluetooth.com/specifications/gatt.

35. **Adafruit.** GATT. [En línea] https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

36. **Bluetooth SIG.** GATT Overview. [En línea] https://www.bluetooth.com/specifications/gatt/generic-attributes-overview.

37. —. Heart Rate Profile. [En línea] https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=239865.

38. **PatagoniaTec.** Conectar 3 Arduino por I2C. [En línea] http://saber.patagoniatec.com/3314-arduino-argentina-ptec-iic-i2c-muchos/.

39. **Fernández, Antonio Moreno.** El Bus I2C. [En línea] http://www.uco.es/~el1mofer/Docs/IntPerif/Bus%20I2C.pdf.

40. **RepRage.** How to connect the Raspberry Pi to a Bluetooth heart rate monitor. [En línea] https://reprage.com/post/how-to-connect-the-raspberry-pi-to-a-bluetooth-heart-rate-monitor.

41. **Sinha, Pratik.** Using gatttool in a manual/non-interactive mode to read BLE devices. [En línea] http://www.humbug.in/2014/using-gatttool-manualnon-interactive-mode-read-ble-devices/.

42. **Firszt, Przemo.** Bluetooth BLE, gatttool and (almost) all those numbers .... explained. [En línea] http://blog.firszt.eu/index.php?post/2015/09/13/bt.

43. **Birkett, Andrew.** Interfacing Raspberry Pi and MPU-6050. [En línea] http://blog.bitify.co.uk/2013/11/interfacing-raspberry-pi-and-mpu-6050.html.

44. **Debra.** MPU-6050 Redux: DMP Data Fusion vs. Complementary Filter. [En línea] http://www.geekmomprojects.com/mpu-6050-redux-dmp-data-fusion-vs-complementary-filter/#.

45. **Hirst, Richard.** PiBits . [En línea] https://github.com/richardghirst/PiBits.

46. **Adam.** Using the I2C interface. [En línea] http://www.raspberry-projects.com/pi/programming-in-c/i2c/using-the-i2c-interface.

47. **Texas Instruments.** ADS111x Ultra-Small, Low-Power, I2C-Compatible, 860-SPS, 16-Bit ADCs. [En línea] http://www.ti.com/lit/ds/symlink/ads1114.pdf.

48. **OpenLabTools.** RPi and I2C Analog-Digital Converter. [En línea] http://openlabtools.eng.cam.ac.uk/Resources/Datalog/RPi_ADS1115/.

49. **Hirst, Richard.** Git Hub - PiBits. [En línea] https://github.com/richardghirst/PiBits/tree/master/MPU6050-Pi-Demo.

50. **Enlace de compra Raspberry PI. [En línea] http://uk.rs-online.com/web/p/processor-microcontroller-development-kits/832-6274/.**

51. ***Enlace de compra Epson Moverio BT-100.*** **[En línea]** https://www.amazon.in/Epson-Moverio-BT-100-Smart-Glasses/dp/B00KLFS4GG/257-7537992-9866546?_encoding=UTF8&%2AVersion%2A=1&%2Aentries%2A=0&portal-device-attributes=desktop.

52. Enlace de compra Cámara FLIR. [En línea] https://www.digikey.es/product-detail/es/flir/500-0643-00/500-0643-00-ND/5215151.

53. Enlace de compra LM35. [En línea] https://www.amazon.es/Bobury-Precisi%C3%B3n-Temperatura-Cent%C3%ADgrado-Impedancia/dp/B06XK6JD2J/ref=sr_1_2/257-5615063-0071641?ie<=UTF8&qid=1498538969&sr=8-2&keywords=lm35.

54. Enlace de compra MQ-4. [En línea] http://www.dx.com/es/p/fc-22-a-mq-2-combustible-gas-sensor-module-for-liquefied-gas-propane-hydrogen-blue-silver-193674#.WVHnhGjyg2w.

55. Enlace de compra MPU6050. [En línea] https://store.invensense.com/ProductDetail/MPU6000-InvenSense-Inc/420595/.

56. Enlace de compra Cospoo h8. [En línea] https://www.amazon.com/Wireless-Bluetooth-Monitor-Tracker-Android/dp/B06ZY242Q4.

57. Enlade de compra Dongle. [En línea] https://www.adafruit.com/product/1327.

58. Enlace de compra ADS1115. [En línea] https://www.banggood.com/CJMCU-ADS1115-16Bit-ADC-Development-Board-Module-p-986645.html.

59. Enlace de compra ProtoBoard. [En línea] https://www.amazon.es/dp/B00L7Z7H9Y?psc=1.

Capítulo 10. Anexo Figuras

Capítulo 11. Anexos

11.1 Pulsometro.cpp

```
#include <iostream>
#include <cstdlib>
#include <time.h>
#include <unistd.h> //para el sleep
#include <string>
using std::string;
#include <stdexcept>
#include <stdio.h>
#include <sstream>

string GetStdoutFromCommand(string cmd);
string HextoDec(string strHex);
bool Lee_Bateria();
int busycont = 1;
int main()
{
    bool bConnected = false; //Para conectar la primera vez que se
inicia el programa
    bool bRead = false; //Para cada vez que se lea el pulso
    system("echo 1. Inicializando bluetooth...");
    usleep(5);
    system("sudo hciconfig hci0 up"); //encendemos bluetooth
    std::string result = "";


    while( true)
    {
        while (bConnected == false) //Colocaremos fecha y batería una
vez únicamente
        {
            system("date > lastlog.txt"); //Introducimos la fecha en
el txt de logs
            system("date"); //Y la mostramos por consola
            usleep(1000000);
            bConnected = Lee_Bateria();

        }
        //Se ha introducido la fecha de hoy y la batería
correctamente. Pasamos a leer pulsaciones
        while (bRead == false)
        {
            system("date +%T");
            //Ponemos timeout al comando para que no lea
infinitamente, ya que al enviar el comando, el terminal se queda
leyendo respuestas y perdemos el control
            system("sudo timeout 1.5 gatttool -i hci0 -b
D0:5F:B8:41:16:2F --char-write-req -a 0x013 -n 0100 --listen >
lastlog.txt"); //El resultado del comando se guarda en lastlog
            result = GetStdoutFromCommand("cat lastlog.txt"); //Leemos
lastLog y lo guardamos en result
            usleep(5000);

            if(result.find("Characteristic value was written
successfully") != std::string::npos  ) { //Si se ha leido
correctamente la pulsacion
                usleep(5000);
                system("mv lastlog.txt logs");
```

```
                int pos = result.find("value: 00"); //Buscamos la
posición en la que se nos indica el valor de las pulsaciones

                result.erase(0,pos + 10);
                std::string pulso = HextoDec (result);
                system("date +%T >>
$HOME/Desktop/logs/historial.txt");
                system(("echo Pulsaciones/min: " + pulso + " >>
$HOME/Desktop/logs/historial.txt").c_str());
                bRead = true;
                if (busycont > 1) {
                busycont -= 1;
            }

            } else
            { //Si no se ha conseguido leer se repite la acción. Si
hemos encontrado el dispositivo ocupado, esperamos cada vez un poco
más para darle tiempo
                if(result.find("busy") != std::string::npos  ||
result.find("") != std::string::npos)
                {
                    usleep(500000 * busycont);
                    system("echo busy or nothing");
                    busycont +=1;
                    result = "";
                    //Reiniciamos el bluetooth
                        system("sudo hciconfig hci0 down");
//encendemos bluetooth
                        usleep(2000 * busycont );
                        system("sudo hciconfig hci0 up"); //encendemos
bluetoot

                }
                //si no se ha leído bien el pulsometro, se guarda en
el historial igualmente
                system("mv lastlog.txt logs");
                usleep(5000000);
                if (busycont > 20)
                {
                        system("sudo hciconfig hci0 down");
                        usleep(5000000);
                        bRead = false;
                        bConnected = true;
                        busycont = 0;
                        system("sudo hciconfig hci0 up");
                }
            }
            result = "";
        }
      bRead = false;
    }

    return 0;
}
string GetStdoutFromCommand(string cmd) { //Lanza un comando y
devuelve el resultado
    string data;
    FILE * stream;
    const int max_buffer = 256;
    char buffer[max_buffer];
    cmd.append(" 2>&1");
```

```
    stream = popen(cmd.c_str(), "r");
    if (stream) {
        while (!feof(stream))
            if (fgets(buffer, max_buffer, stream) != NULL)
data.append(buffer);
                pclose(stream);
    }
    return data;
}

string HextoDec (string strHex) { //Transforma un string hexadecimal a
un string decimal

        int iDec;
    std::stringstream ssHex;
    std::stringstream ssDec;

    ssHex << std::dec << strHex;
    ssHex >> std::hex >> iDec;
    ssDec << iDec;
    std::string strDec = ssDec.str();
    system(("echo Hexadecimal: " + strHex).c_str());
    system(("echo Decimal: " + strDec).c_str());
    return strDec;
}

bool Lee_Bateria() { //Lee el registro de la batería del dispositivo
        std::string strHexBattery = "null";
        strHexBattery = GetStdoutFromCommand("sudo gatttool -i hci0 -b
D0:5F:B8:41:16:2F --char-read -a 0x025"); //Leemos el estado de la
batería
            if(strHexBattery.find("Characteristic value/descriptor:")
!= std::string::npos  )  //Si se ha leído correctamente el valor
            {
                strHexBattery.erase(0, strHexBattery.length()-
4);//borramos todo menos los dos últimos caracteres, teniendo en
cuenta que hay un /n invisible
                strHexBattery.erase(2, strHexBattery.length());
                string strDecBattery = HextoDec(strHexBattery);
                system(("echo Bateria =" + strDecBattery + "%>>
lastlog.txt").c_str());

                //Muevo el archivo a la carpeta logs y copio su
contenido a historial
                system("cp lastlog.txt logs");
                system("cat $HOME/Desktop/logs/lastlog.txt >>
$HOME/Desktop/logs/historial.txt");
                return true; //Si se ha leido correctamente, salimos
del loop
            }
            else
            {
                system("echo No se ha podido conectar. Reintentando
lectura de bateria");
                usleep(1000000);
                return false;
            }
    }
```

11.2 Gas_Sensor.cpp

```
#include <string>
using std::string;
#include <stdexcept>
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <stdexcept>
#include <stdio.h>
#include <sstream>
#include <time.h>
#include <unistd.h> //para el sleep
#include <cstdint>
#include <iostream>
#include <stdlib.h>
#include <cmath>
#include <bitset>
#include <algorithm>
string GetStdoutFromCommand(string cmd);
string HextoDec (string strHex);
string HextoDec_C2 (string strHex);
std::string LecturaADC();
void Conversion(float voltaje);
void FileWrite(std::string voltaje);
std::string result_H;
std::string result_L;
std::string acc_X;
std::string acc_Y;
std::string acc_Z;

using std::cout;
using std::endl;
using std::cin;
using std::hex;
using std::dec;
using std::bitset;

std::ofstream outfile ("logs/concentracion_gas.txt");

int main()
{
    //Creo el fichero de texto donde guardaré los datos
    outfile << "Voltaje;Tiempo" << std::endl;
    system("i2cset -y 1 0x48 1 0x83C0 w");
    for (;;) {
std::string voltaje;
voltaje = LecturaADC(); //Leo la salida del ADC
FileWrite (voltaje);
    }
}
string GetStdoutFromCommand(string cmd) {

    string data;
    FILE * stream;
    const int max_buffer = 256;
    char buffer[max_buffer];
    cmd.append(" 2>&1");
```

```
    stream = popen(cmd.c_str(), "r");
    if (stream) {
        while (!feof(stream))
            if (fgets(buffer, max_buffer, stream) != NULL)
data.append(buffer);
                pclose(stream);
    }
    return data;
}
string HextoDec (string strHex) {


        int iDec;
    std::stringstream ssHex;
    std::stringstream ssDec;

    ssHex << std::dec << strHex;
    ssHex >> std::hex >> iDec;
    ssDec << iDec;
    std::string strDec = ssDec.str();
    //system(("echo Hexadecimal: " + strHex).c_str());
    //system(("echo Decimal: " + strDec).c_str());
    return strDec;
}

std::string HextoDec_C2 (string strHex) {

    strHex = strHex;
    //signed short iDec;
//signed int iHex;
 int iDec;
 int iHex;
 int signo;
    std::stringstream ssHex;
    std::stringstream ssDec;
    std::stringstream ssSig;
    std::stringstream ssMid;

    std::string strDec = "0";
    std::string strsign = "";

    ssHex << std::hex << strHex;
    ssHex >> std::hex >> iHex; //integer hexadecimal

    signo = (iHex >> 15) ;

    if (signo == 1) {
    iHex = ~iHex + 1;
iHex = iHex & 0x0000FFFF;
//iHex = iHex.erase(0,4);
    ssDec << std::hex << signo;

    strDec = ssDec.str();

    //ssDec << std::hex << Signo;
    //std::string strsign = ssDec.str();



    ssSig << std::dec << iHex;
    ssSig >> std::hex >> iDec;
```

```
    ssMid << std::hex << iDec; //Tengo el valor. Me falta el signo
    strsign = ssMid.str();

    strsign = "-" + strsign;
} else

{
    ssSig << std::dec << iHex;
    ssSig >> std::hex >> iDec;


    ssMid << std::hex << iDec; //Tengo el valor. Me falta el signo
    strsign = ssMid.str();

}

    //strDec = strDec.erase(0,4);
    //system(("echo Hexadecimal: " + strHex).c_str());
    //system(("echo Decimal: " + strDec).c_str());
    //system(("echo Signoh: " + strsign).c_str());
    return strsign;
}

std::string LecturaADC (){
    std::string result;
    std::string LSB;
    std::string MSB;

    result = GetStdoutFromCommand("i2cget -y 1 0x48 0 w");
    result = result.erase(0,2);
    LSB = result.at(2);
    LSB = LSB + result.at(3);
    //  std::cout << "LSB: ";
//  std::cout << LSB;
    //std::cout << "\n";

    MSB = result.at(0);
    MSB = MSB + result.at(1);
    //  std::cout << "MSB: ";
    ////std::cout << MSB;
    std::cout << "\n";

//      std::cout << "El valor leído era: ";
//std::cout << result;
//  std::cout << "\n";
    result = LSB + MSB;
    std::string valor = HextoDec_C2 (result);
        std::cout << "Decimal: ";
    std::cout << valor;
    std::cout << "\n";


    int valorDec;
    std::stringstream ssDec;
    ssDec << std::dec << valor;
    ssDec >> std::dec >> valorDec;
```

```
    float voltaje = valorDec * 187.5 / 1000000; //Paso a voltios
multiplicando el numero de bits(valorDEC) por 187.5uV por bit

    std::stringstream ssVoltaje;
    std::string strVoltaje;

    ssVoltaje << voltaje;
    strVoltaje = ssVoltaje.str();
    std::cout << "Voltaje: ";
    std::cout << strVoltaje;
    std::cout << "V\n";

    Conversion(voltaje);
    usleep(1000000);
    return strVoltaje;
}

void FileWrite (std::string voltaje)
{
    std::string fila;
    std::string fecha = GetStdoutFromCommand("echo `date +%T`';'");
        std::replace(voltaje.begin(), voltaje.end(), '.', ',');
               fila = voltaje + ";" + fecha;
           outfile << fila << std::flush;
}

void Conversion(float voltaje) {
        float RS = ((5 * 20000) / voltaje ) - 20000;

    std::stringstream ssRS;
    std::string strRS;

    //ssRS << RS;
    //strRS = ssRS.str();
    //std::cout << "RS: ";
    //std::cout << strRS;
    //std::cout << " Ohm\n";

    float relacion = RS/650000;
    float concentracionLPG = 28485.7 - 11314.3*relacion;
    std::stringstream ssLPG;
    std::string strLPG;

    //ssLPG << concentracionLPG;
    //strLPG = ssLPG.str();
    //std::cout << "LPG: ";
    //std::cout << strLPG;
    //std::cout << " ppm\n";

    float concentracionCH4 = 26600.0 -14666.7*relacion;

    std::stringstream ssCH4;
    std::string strCH4;

    //ssCH4 << concentracionCH4;
    //strCH4 = ssCH4.str();
    //std::cout << "CH4: ";
    //std::cout << strCH4;
    //std::cout << " ppm\n";

}
```

11.3 Acelerómetro_dmp.cpp

```
#include <string>
using std::string;
#include <stdexcept>
#include <stdio.h>
#include <sstream>
#include <iostream>
#include <cstdlib>
#include <stdexcept>
#include <stdio.h>
#include <sstream>
#include <time.h>
#include <unistd.h> //para el sleep
#include <cstdint>
#include <iostream>
#include <stdlib.h>
#include <cmath>
#include <bitset>

string GetStdoutFromCommand(string cmd);
string HextoDec (string strHex);
string HextoDec_C2 (string strHex);

std::string result_H;
std::string result_L;
std::string acc_X;
std::string acc_Y;
std::string acc_Z;

using std::cout;
using std::endl;
using std::cin;
using std::hex;
using std::dec;
using std::bitset;
bool wile = true;
int cont = 0;
//uint16_t x2=0;
int main()
{
    system("sudo i2cset -y 1 0x68 0x6b 0x20");//Pongo a 1 el modo
cíclico y apago el modo sleep 0x20
    system ("echo Acelerando > acc.txt");
    while(wile == true){
    //X
    result_H = GetStdoutFromCommand("sudo i2cget -y 1 0x68 0x3B");
    result_L = GetStdoutFromCommand("sudo i2cget -y 1 0x68 0x3C");

    result_H = result_H.erase(0,2);//eeeeeeeeee
    result_H = result_H.erase(result_H.length()-1);
    result_L = result_L.erase(0,2);
    result_L = result_L.erase(result_L.length()-1);

    acc_X = result_H+result_L;


    //uint16_t x2=0;
   //acc_X>>x2;
   // int diff=0x0000-x2;
```

```
    //cout<<"The number you have entered is: "<<dec<<diff<<endl;


    //system(("echo Resultado_H: " + result_H).c_str());
    //system(("echo Resultado_L: " + result_L).c_str());
    //system(("echo Resultado: " + acc_X).c_str());

    std::string acc_X_Dec = HextoDec_C2(acc_X);
    system(("echo Resultado_DecX: " + acc_X_Dec).c_str());


        //Y
    result_H = GetStdoutFromCommand("sudo i2cget -y 1 0x68 0x3D");
    result_L = GetStdoutFromCommand("sudo i2cget -y 1 0x68 0x3E");

    result_H = result_H.erase(0,2);//eeeeeeeeee
    result_H = result_H.erase(result_H.length()-1);
    result_L = result_L.erase(0,2);
    result_L = result_L.erase(result_L.length()-1);

    acc_Y = result_H+result_L;

    //system(("echo Resultado_H: " + result_H).c_str());
    //system(("echo Resultado_L: " + result_L).c_str());
    //system(("echo Resultado: " + acc_X).c_str());

    std::string acc_Y_Dec = HextoDec_C2(acc_Y);
    system(("echo Resultado_DecY: " + acc_Y_Dec).c_str());



        //Z
    result_H = GetStdoutFromCommand("sudo i2cget -y 1 0x68 0x3F");
    result_L = GetStdoutFromCommand("sudo i2cget -y 1 0x68 0x40");

    result_H = result_H.erase(0,2);//eeeeeeeeee
    result_H = result_H.erase(result_H.length()-1);
    result_L = result_L.erase(0,2);
    result_L = result_L.erase(result_L.length()-1);

    acc_Z = result_H+result_L;

    //system(("echo Resultado_H: " + result_H).c_str());
    //system(("echo Resultado_L: " + result_L).c_str());
    //system(("echo Resultado: " + acc_X).c_str());

    std::string acc_Z_Dec = HextoDec_C2(acc_Z);
    system(("echo Resultado_DecZ: " + acc_Z_Dec).c_str());

        int X = stoi(acc_X_Dec);
        int Y = stoi(acc_Y_Dec);
        int Z = stoi(acc_Z_Dec);

        if (Z < -15000 ) {
            system("echo Caída en Z!!!");
            system("echo Caída en Z!!! >> acc.txt");
                system(("echo PResultado_DecZ " + acc_Z_Dec + " >>
acc.txt").c_str());

        }
        if (Y < -15000 ) {
```

```
            system("echo Caída en Y!!!");
            system("echo Caída en Y!!! >> acc.txt");
                system(("echo PResultado_DecY " + acc_Y_Dec + " >>
acc.txt").c_str());
        }
        if (X <-15000 ) {
            system("echo Caída en X!!!");
            system("echo Caída en X!!! >> acc.txt");
                system(("echo PResultado_DecX" + acc_X_Dec + " >>
acc.txt").c_str());
        }

        if (Z > 15000 ) {
            system("echo Subida en Z!!!");
            system("echo Subida en Z!!! >> acc.txt");
                system(("echo Resultado_DecZ " + acc_Z_Dec + " >>
acc.txt").c_str());
        }
        if (Y > 15000 ) {
            system("echo Subida en Y!!!");
                system("echo Subida en Y!!! >> acc.txt");
                system(("echo Resultado_DecY " + acc_Y_Dec + " >>
acc.txt").c_str());
        }
        if (X > 15000 ) {
            system("echo Subida en X!!!");
            system("echo Subida en X!!! >> acc.txt");
                system(("echo Resultado_DecX" + acc_X_Dec + " >>
acc.txt").c_str());
        }
        system("echo");
        //cont++;
        //if (cont > 500) {
            //wile = false;
        }
    //usleep(1000000);
    //wile = false;
}
}

string GetStdoutFromCommand(string cmd) {

    string data;
    FILE * stream;
    const int max_buffer = 256;
    char buffer[max_buffer];
    cmd.append(" 2>&1");

    stream = popen(cmd.c_str(), "r");
    if (stream) {
        while (!feof(stream))
            if (fgets(buffer, max_buffer, stream) != NULL)
data.append(buffer);
                pclose(stream);
    }
    return data;
}
string HextoDec (string strHex) {


        int iDec;
```

```
    std::stringstream ssHex;
    std::stringstream ssDec;

    ssHex << std::dec << strHex;
    ssHex >> std::hex >> iDec;
    ssDec << iDec;
    std::string strDec = ssDec.str();
    //system(("echo Hexadecimal: " + strHex).c_str());
    //system(("echo Decimal: " + strDec).c_str());
    return strDec;
}

string HextoDec_C2 (string strHex) {

    strHex = strHex;
    //signed short iDec;
//signed int iHex;
 int iDec;
 int iHex;
 int signo;
    std::stringstream ssHex;
    std::stringstream ssDec;
    std::stringstream ssSig;
    std::stringstream ssMid;

    std::string strDec = "0";
    std::string strsign = "";

    ssHex << std::hex << strHex;
    ssHex >> std::hex >> iHex; //integer hexadecimal

    signo = (iHex >> 15) ;

    if (signo == 1) {
    iHex = ~iHex + 1;
iHex = iHex & 0x0000FFFF;
//iHex = iHex.erase(0,4);
    ssDec << std::hex << signo;

    strDec = ssDec.str();

    //ssDec << std::hex << Signo;
    //std::string strsign = ssDec.str();



    ssSig << std::dec << iHex;
    ssSig >> std::hex >> iDec;



    ssMid << std::hex << iDec; //Tengo el puto valor. Me falta el
signo
    strsign = ssMid.str();

    strsign = "-" + strsign;
} else

{
    ssSig << std::dec << iHex;
    ssSig >> std::hex >> iDec;
```

```
    ssMid << std::hex << iDec; //Tengo el puto valor. Me falta el
signo
    strsign = ssMid.str();

}

    //strDec = strDec.erase(0,4);
    //system(("echo Hexadecimal: " + strHex).c_str());
    //system(("echo Decimal: " + strDec).c_str());
    //system(("echo Signoh: " + strsign).c_str());
    return strsign;
}
```