Sistema de protección personal mediante dispositivos interconectados

José Luis Fernández Piñero
Grado en Ingeniería en Sistemas de Telecomunicación
Arduino

Antoni Morell Pérez
Pere Tuset Peiró

2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Sistema de protección personal mediante dispositivos interconectados*
Nombre del autor:	*José Luis Fernández Piñero*
Nombre del consultor/a:	*Antoni Morell Pérez*
Nombre del PRA:	*Pere Tuset Peiró*
Fecha de entrega (mm/aaaa):	06/2019
Titulación:	*Grado en Ingeniería en Sistemas de Telecomunicación*
Área del Trabajo Final:	*Arduino*
Idioma del trabajo:	*Castellano*
Palabras clave	*Smart EPI, Industria 4.0, IoT*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

La seguridad laboral, además de un requisito legal, es una buena práctica que fomenta el compromiso de los empleados con la organización y promueve una cultura de excelencia que se traslada a otros ámbitos y que está relacionada con las empresas más exitosas.

Las nuevas tecnologías ofrecen posibilidades inexistentes hace solo unos años que ponen a disposición de las empresas dispositivos capaces de capturar datos en tiempo real y transmitirlos para su explotación, pero que van más allá e incluso posibilitan interactuar con el entorno físico.

El mercado ofrece sensores y equipos de protección personal para evitar o, al menos, reducir los riesgos a los que se enfrentan los empleados de las empresas y sus consecuencias. Sin embargo, todavía no son habituales los sistemas que combinen estos dispositivos entre sí y con los sistemas de información de las empresas.

Este proyecto propone la construcción de un prototipo de sistema de protección personal mediante sensores interconectados a dispositivos wearables aprovechando las oportunidades que ofrecen las tecnologías que posibilidad el denominado internet de las cosas (en inglés Internet of Things, abreviado como IoT).

Así mismo, se sugiere un plan de evolución para valorar la posibilidad de comercialización utilizando el enfoque iterativo de rápida presentación a los potenciales clientes denominado Lean Startup.

Abstract (in English, 250 words or less):

Occupational safety, in addition to being a legal requirement, is a good practice that encourages employees' commitment to the organization and promotes a culture of excellence that is transferred to other fields and is related to the most successful companies.

The new technologies offer possibilities that did not exist only a few years ago that make available to companies devices capable of capturing data in real time and transmitting them for exploitation, but that go beyond and even make it possible to interact with the physical environment.

The market offers sensors and personal protection equipment to avoid or, at least, reduce the risks faced by company employees and their consequences. However, systems that combine these devices with each other and with business information systems are not yet common.

This project proposes the construction of a prototype personal protection system using sensors interconnected to wearable devices taking advantage of the opportunities offered by the technologies that the so-called Internet of Things (IoT) offers.

It also suggests an evolution plan to assess the possibility of marketing using the iterative approach of quick presentation to potential customers known as Lean Startup.

1 Introducción

1.1 Motivación

La seguridad laboral, más allá del mero cumplimiento legal, es una palanca para fomentar la excelencia operativa en las organizaciones[1]. Este proyecto propone la utilización del concepto *Lean Startup* para construir un sistema distribuido que conforme un equipo de protección personal que pueda ser comercializado para su uso en aquellas empresas donde las condiciones del entorno son inherentemente peligrosas para sus empleados.

El sistema construido constituye lo que se podría denominar un equipo de protección personal inteligente (*Smart EPI*) que los trabajadores utilizarían tanto para aumentar su seguridad como para ofrecer información en tiempo real a la organización.

Los datos recopilados en tiempo real pueden ser combinados con otros para ofrecer información de mayor valor que apoye a la organización. Este enfoque permite a la empresa iniciar o reforzar su posicionamiento en la revolución industrial conocida por el nombre de “Industria 4.0”.

1.2 Concepto de equipo de protección inteligente

El artículo 2 del *Real Decreto 773/1997, de 30 de mayo, sobre disposiciones mínimas de seguridad y salud relativas a la utilización por los trabajadores de equipos de protección individual* [1] define equipo de protección individual (normalmente abreviado como EPI en español y PPE en inglés) como “[…] cualquier equipo destinado a ser llevado o sujetado por el trabajador para que le proteja de uno o varios riesgos que puedan amenazar su seguridad o su salud, así como cualquier complemento o accesorio destinado a tal fin”.

Un EPI *wearable* sería un equipo de protección que puede “llevarse puesto”. Si además incorpora nuevas tecnologías, como por ejemplo que las zonas vulnerables del cuerpo estén equipadas con sensores y actuadores que reaccionan e interactúan con el usuario y otros sistemas, este tipo de equipos de protección suelen denominarse “inteligentes” y englobarse bajo el concepto *Smart EPI*.

Los equipos de protección inteligentes de uso personal son en su mayoría *wearables* por el hecho inherente de ser de “uso personal”. Actualmente, en el mercado de los equipos de protección se pueden encontrar EPIs *wearable* como por ejemplo:

[1] En palabras de Paul O'Neill, CEO de ALCOA desde 1987 hasta 1999: “If you want to understand how Alcoa is doing, you need to look at our workplace safety figures.”

- Las gafas de protección que proyectan información en una esquina de la lente y que permiten al trabajador visualizar información relacionada con la tarea a realizar o la ubicación en la que se encuentra. En el mercado se pueden encontrar productos como los comercializados por Iristick [2].
- Los cascos para entornos ruidosos que reducen el ruido ambiente e integran sistemas de comunicación y que mejoran las condiciones de los trabajadores protegiendo su salud y aumentando su productividad. Más aún cuando los cascos incluyen protección ocular y en ese caso suman también las posibilidades de realidad aumentada ya mencionadas para las gafas de protección. En el mercado se pueden encontrar productos como los comercializados por DAQRI [3] o los fabricantes más relevantes para el periodo 2017-2021 [4].
- Los zapatos de seguridad para ambientes de trabajo pueden ofrecer múltiples características, como por ejemplo detectar que el trabajador se ha caído, conocer la ubicación del trabajador en el emplazamiento, realizar notificaciones de emergencia o capturar información respecto a cuestiones que pueden afectar a la salud (vibraciones, distancias recorridas, velocidad de desplazamiento, etc.). En el mercado se pueden encontrar productos como los comercializados por ZhorTech [5] e intellinium.io [6], ofreciendo esta última una solución muy innovadora ya que no solo comercializa zapatos de seguridad sino un dispositivo que se puede colocar a zapatos de seguridad tradicionales de otros fabricantes y así dotarlos de características propias de un *Smart EPI*.

Algunos de los ejemplos anteriores evidencian que la característica diferenciadora de los *Smart EPI* respecto a un EPI convencional es la interacción, tanto como con el usuario como con otros sistemas. Esta interconexión entre los *wearables* y otros sistemas se adentra en el concepto de IoT que posibilita la Industria 4.0 para mejorar la seguridad de los trabajadores.

1.3 Concepto de Industria 4.0

Se considera que hasta la fecha han ocurrido cuatro revoluciones industriales: la primera fue el uso intensivo de máquinas, la segunda el uso extensivo de la electricidad, la tercera el uso extensivo de ordenadores y la cuarta la inclusión de la inteligencia artificial.

El término Industria 4.0 [7][8] hace referencia a la cuarta revolución industrial que traerá (ya está trayendo) cambios significativos en la industria y la economía mundial.

Las características más relevantes de las fábricas inteligentes dentro de la Industria 4.0 incluyen la conectividad y la proactividad. La conectividad considera la interconexión de procesos, máquinas y personas que utilizan información en tiempo real para la toma de decisiones y la proactividad aprovecha dicha información para actuar de forma anticipada en lugar de limitarse a reaccionar.

La interconexión de máquinas y personas y el uso de la información para la toma de decisiones proactivas conecta de forma muy directa con el concepto de *Smart EPI wearable* ya mencionado. Aquí es cuando aparece su relación con Industria 4.0 e IoT.

Si se parte de un conjunto de sensores desplegados por los emplazamientos industriales e interconectados utilizando IoT se dispondrá de un buen número de datos en tiempo real sobre las condiciones del emplazamiento, las máquinas, la producción y las condiciones del entorno ambiental. Toda esa información recogida y procesada ofrecerá información útil para la toma de decisiones a nivel organizacional, pero también podrá ser explotada por los empleados que utilicen un *Smart EPI wearable* (por ejemplo ofreciendo información de realidad aumentada en sus gafas de protección con datos en tiempo real o haciendo que la ropa de trabajo vibre para alertar de que se está accediendo a una zona con acceso restringido).

1.4 Importancia de la seguridad laboral

Aumentar la seguridad de los trabajadores y reducir los accidentes laborales, además de ser una obligación ética para con la sociedad, es un requisito legal que también redunda en una mayor eficiencia de las organizaciones: mejorando la productividad y reduciendo los costes.

Los costes económicos de la accidentalidad son cuantitativamente muy relevantes y múltiples estudios lo atestiguan, ya que los datos económicos son argumentos contundentes para concienciar a las organizaciones en la necesidad de invertir en seguridad. Por ejemplo, la Agencia Europea para la Seguridad y la Salud en el Trabajo estimó en 2017 el coste anual a nivel mundial de los accidentes laborales en 476.000 millones de euros [9]. Así mismo, la Administración para la Seguridad y Salud de Estados Unidos de América (OSHA por su acrónimo inglés), ofrece múltiples estudios [10] al respecto incluyendo uno que estima en 250.000 millones de dólares los costes de la accidentalidad en Estados Unidos de América para el año 2007.

Las posibilidades de IoT para remplazar los EPI tradicionales por *Smart EPI* son múltiples y proporcionan ventajas significativas para mejorar la seguridad laboral. Además de los ejemplos enunciados previamente, surgen soluciones mucho más avanzadas [11] como:

- Localización y seguimiento de trabajadores en entornos peligrosos. Combinando para ello técnicas de localización en entornos abiertos, por ejemplo mediante GPS, con técnicas válidas en entornos cerrados, por ejemplo balizas por radiofrecuencia.
- Alertas por condiciones inseguras, especialmente las relativas al entorno y los riesgos invisibles como las concentraciones de gases tóxicos.
- Integración con permisos de trabajo sobre máquinas utilizando procedimientos de *lockout/tagout* [12] inteligente.
- Monitorización de la salud de los trabajadores (pulso, tensión arterial, etc.) para la prevención de enfermedades y la detección, lo más

temprana posible, de accidentes (por ejemplo desmayos, caídas, situaciones de estrés, etc.).

Una evidencia de la importancia de los EPIs inteligentes y sus posibilidades es que DuPont Sustainable Solutions (DSS), la división de DuPont dedicada a la consultoría de seguridad laboral reconocida como la primera empresa del sector [13][14], está estableciendo alianzas con empresas punteras en tecnología de EPIs inteligentes como GuardHat [15].

1.5 Objetivos y beneficios esperados

El objetivo principal es la construcción de un prototipo funcional de un sistema distribuido que interconecta uno o más sensores que capturan datos del entorno con dispositivos a los que se envía dicha información para ser procesada y utilizada para la mejora de la seguridad laboral.

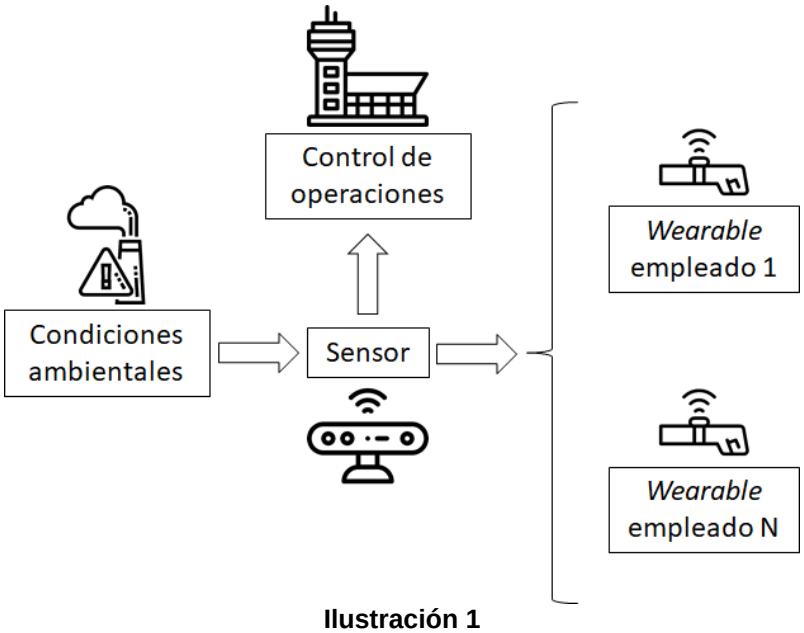

Ilustración 1

El sistema, tal y como muestra la Ilustración 1, estará constituido por un elemento sensor –que representará el conjunto de sensores que capturan las condiciones ambientales– y uno o más *Smart EPI* implementados en forma de dispositivo *wearable*, que facilite su utilización en entornos industriales sin ser una molestia para los trabajadores que lo utilicen.

Por ejemplo, si se consideran los sensores del fabricante Gray Wolf Sensing Solutions, se observa que cada empleado debería portar y manejar un sensor compuesto por una sonda y una interfaz de usuario que en total pesan aproximadamente 0,7 kg y que en la propia imagen de la hoja de especificaciones del producto (por ejemplo del modelo DirectSense®II [16]) el operario lleva la sonda sujeta a su cinturón y la interfaz con una cinta que la sujeta al cuello (para no tener que sujetarla con las manos en todo momento). Los precios varían en función del modelo elegido y sus características concretas, pero se mueven en un rango que va desde 4.000€ a 7.000€ [17].

En comparación, el sistema propuesto podría ofrecer al trabajador un *wearable* construido usando componentes del fabricante ESPRESSIF compatibles con el chip ESP32 [18][19] por lo que resultaría un dispositivo de reducidas dimensiones y peso (muy inferiores a las de cualquier teléfono móvil) que podría llevarse en un bolsillo. Incluso se podría transformar en *wearable* utilizando tecnología específica para incluirlo como parte de la ropa (por ejemplo se podría integrar en un chaleco reflectante). Para esto se podría construir basándose en la placa Arduino LilyPad más los componentes adicionales necesarios. Esta reducción de tamaño no implicaría un aumento de coste, ya que una placa ESP32 con conexión Bluetooth y batería de litio cuesta menos de 10€ y un Arduino LilyPad cuesta menos de 20€ (aunque en este caso habría que añadir otros componentes).

Con la solución propuesta incluso el tamaño y el peso del sensor ya no sería tan relevante porque el trabajador no debería cargar con él, ya que sería su *wearable* el que estuviese conectado inalámbricamente al sensor. Nuevamente los costes del sensor se reducirían porque sería compartido por todos los trabajadores con su *wearable* conectado a él e, incluso aunque no se comercializase en un formato integrable, podría construirse utilizando tecnología compatible. Por ejemplo un sensor de temperatura Texas Instruments LM35 cuesta menos de 2€ mientras que un sensor de CO2 Texas Instruments LMP91051 cuesta unos 100€.

En resumen, si se compara el ejemplo anterior con la solución propuesta, los beneficios que se obtendrán son:

1. Los empleados utilizarán un único dispositivo de tamaño y peso reducido e integrado en su ropa de trabajo, lo que aumentará la comodidad a la hora de realizar sus tareas y reducirá riesgos. Debe tenerse en cuenta que los equipos más ligeros requieren menos esfuerzo físico y previenen lesiones musculares, mientras que los equipos más pequeños evitan atrapamientos contra otros elementos del entorno.
2. Los costes de los sensores se reducen, ya que un conjunto de *Smart EPI wearables* utilizado por varios trabajadores estarán conectados al mismo elemento sensor del cual extraerán la información que él recoge.
3. Los datos capturados por el sensor también serán enviados al centro de control para su almacenamiento, procesado y explotación de forma integrada con la información recogida en otros sistemas o introducida por los propios empleados. Es importante considerar que los datos son considerados un activo con gran valor para las empresas ya que al explotarse con técnicas de *big data* se puede extraer de ellos información de mayor valor.

Es importante resaltar que incluso en el caso de que haya un único *wearable* conectado al sensor, la solución propuesta sigue aportando valor. Si se considera un escenario –cada vez más habitual– de *lone worker*, donde los trabajadores realizan actividades peligrosas de forma aislada y solitaria (por ejemplo la inspección de los aerogeneradores de un parque eólico), la conectividad entre el *Smart EPI* del trabajador y el centro de control ofrece posibilidades avanzadas para detectar accidentes (por ejemplo un botón de

alarma que el trabajador pueda pulsar, una alarma automática si el *Smart EPI* está inmóvil demasiado tiempo o si sufre una movimiento muy brusco, etc.) y reaccionar ante ellos.

En resumen, los objetivos que se pretenden conseguir son:

1. Ofrecer un prototipo de *Smart EPI wearable* que los trabajadores puedan transportar y utilizar sin interferir en su trabajo. Por lo que debe ser un **dispositivo de tamaño y peso reducido e integrado en su ropa de trabajo**. Como punto de partida se considerará que el tamaño y el peso es adecuado si no son superiores a un *smart phone* convencional y, preferentemente, que puedan llevarse sujetos a un cinturón, guardados en el bolsillo de la ropa de trabajo o sujeto por una correa como si fuese un reloj.
2. Ofrecer un dispositivo que:
    a. Aglutine uno o varios **sensores que capturen información respecto a las condiciones del entorno** que sea relevante para la seguridad de los trabajadores. La suposición inicial es que una o dos condiciones del entorno son las relevantes. Por ejemplo, temperatura y concentración de gases tóxicos para la realización de trabajos en espacios confinados, concentración de gases inflamables para la realización de trabajos con riesgo de incendio o humedad para la realización de trabajos con riesgo eléctrico. La disponiblidad de estos sensores se estudia posteriormente en el epígrafe dedicado al estado del arte respecto a sensores.
    b. Esté **conectado inalámbricamente** con los *Smart EPI* de los trabajadores. La suposición inicial es que la cobertura debe permitir operar el Smart EPI a una distancia superior a cinco metros en un espacio con obstáculos. No obstante, el requisito se afinará durante el plan de experimentación propuesto en el epígrafe 5.
    c. **Envíe información a los trabajadores** sobre las condiciones ambientales y los riesgos del entorno. La información debe enviarse en tiempo real o, como mucho, con un retraso inferior a un minuto entre la detección de la condición insegura y la notificación al trabajador. La tecnología de conexión inalámbrica que se elija tras el plan de experimentación debe ser capaz de satisfacer tanto este requisito como el anterior.
    d. **Envíe información al centro de control**. La información debe enviarse con una periodicidad prevista y que pueda ser configurable. Como punto de partida se aceptará un refresco con frecuencia inferior a diez minutos.

1.6 Requisitos adicionales para garantizar la viabilidad económica

El prototipo funcional tendrá viabilidad comercial si, además de cumplir los objetivos ya mencionados, se cumplen los siguientes requisitos:

1. **El coste de los *Smart EPI wearable* debe ser inferior al coste de dispositivos tradicionales que ya incluyan los sensores.** Para

facilitar esto, lo ideal es que el sensor del sistema sea realmente un conjunto de sensores, ya que de esta forma la posibilidad de encontrar un dispositivo tradicional que incluya todo el juego de sensores y que tenga menor coste disminuye. Este requisito se ha evidenciado en los epígrafes anteriores pero se ampliará aún más en los epígrafes posteriores dedicado al estado del arte y el análisis de los costes del prototipo.

2. **La conectividad inalámbrica entre el sensor y los *Smart EPI wearable* debe ser suficiente para el intercambio de datos necesario en entornos industriales reales**, que incluyen escenarios difíciles con interferencias electromagnéticas, espacios confinados, etc. Este requisito requiere ensayos en entornos reales de operación, por lo que el prototipo funcional construido debería ser sometido a pruebas, idealmente en colaboración con una o varias industrias que quieran participar en un piloto. En todo caso no se trata de un problema de ausencia de tecnología, más bien de elección de la tecnología de comunicación inalámbrica adecuada. También se ampliará en el epígrafe posterior dedicado al estado del arte.

En cualquier caso, el coste dispuesto a pagar por un producto o servicio siempre es inferior al valor que este aporta. Por tanto, una posibilidad existente para poder asumir mayores costes de fabricación es aumentar el valor que aporta la solución propuesta. En este sentido, la comunicación entre el sistema y el centro de operaciones abre un abanico casi infinito de evolución funcional para aportar mayor valor sin más que integrar con otros sistemas ya existentes. Por ejemplo para tener localizados los empleados dentro de la planta y denegar permisos de trabajos peligrosos que deban realizarse en la misma ubicación o zonas cercanas.

Por último, tanto los costes de construcción del prototipo funcional como los de la realización del piloto con los ensayos oportunos pueden ser objeto de inversión privada, pero también puede buscarse financiación pública. Por ejemplo a través del Ministerio de Educación, Ciencia y Universidades, la Cámara de Comercio de España o con organismos autonómicos como el Instituto de Desarrollo Económico del Principado de Asturias (IDEPA) que realiza campañas periódicas de financiación de proyectos de I+D+i en colaboración con el Fondo Europeo de Desarrollo Regional (FEDER).

2 Estado del arte

En los siguientes epígrafes se repasarán todos los factores a tener en cuenta para implementar una solución acorde a la arquitectura de la Ilustración 1.

2.1 Hardware Arduino

Una primera búsqueda en *Google* ofrece una explicación sencilla sobre qué es Arduino [20][21]:

- Arduino es una plataforma de desarrollo basada en una placa electrónica de hardware libre que incorpora un microcontrolador re-programable y una serie de pines hembra, los que permiten establecer conexiones entre el microcontrolador y los diferentes sensores y actuadores de una manera muy sencilla.
- Arduino es una compañía de fuente abierta y hardware abierto así como un proyecto y comunidad internacional que diseña y manufactura placas de desarrollo de hardware para construir dispositivos digitales y dispositivos interactivos que puedan detectar y controlar objetos del mundo real.

En el libro “Taller de Arduino: Un enfoque práctico para principiantes” [22] se detalla en la introducción una explicación más amplia sobre qué es Arduino:

- *Arduino es una plataforma electrónica abierta (open hardware) para la creación de prototipos basada en software y hardware libre.*
- *Arduino puede tomar información y datos del entorno a través de sus pines de entrada por medio de toda la gama de sensores que existen en el mercado. En base a ello, puede ser usada para controlar y actuar sobre todo aquello que le rodea; como por ejemplo luces, motores y otro tipo de actuadores.*

De lo anterior se deduce que la tecnología Arduino permite implementar dispositivos electrónicos totalmente funcionales para uso particular, por ejemplo soluciones *Do It Yourself* (DIY) para domótica doméstica, o prototipos para validar diseños que posteriormente puedan comercializarse. En este último caso la fabricación se realizaría con otro tipo de tecnología (por ejemplo las placas de cableado se sustituirían por placas de circuito impreso) orientadas a la reducción de tamaño, consumo de potencia y, sobre todo, costes de fabricación. En internet puede encontrarse multitud de recursos con ejemplos prácticos de proyectos implementados usando Arduino [23][24][25][26], muchos de los cuales tienen también un interés especial por su componente didáctica para complementar asignaturas teóricas de electrónica.

La tecnología *hardware* Arduino se comercializa bajo distintas variantes de placas, entre las que se pueden destacar:

- **Arduino UNO**, que es la placa Arduino básica que se usa para formación en institutos y universidades. Incorpora un chip ATmega328, memoria flash de 32 KB, 14 pines de entrada/salida digital y 6 pines de entrada/salida analógica (todos ellos en terminales hembra para facilitar las conexiones), un puerto USB para transmisión de datos y un puerto *jack* (cilíndrico) para alimentación.
- **Arduino Leonardo** es una evolución de Arduino UNO con chip ATmega32U4 y más recursos (por ejemplo 32 pines de entrada/salida entre digitales y analógicos y una memoria EEPROM de 1 KB).
- **Arduino Mega** es una evolución de Arduino UNO con chip ATmega2560 y todavía más recursos que Arduino Leonardo (por ejemplo 70 pines de entrada/salida entre digitales y analógicos y una memoria EEPROM de 4 KB).
- **Arduino Pro** es una evolución de la placa Arduino UNO ensamblada con componentes de superficie para reducir el tamaño y aumentar la robustez y calidad del diseño final. También incluye una conexión para alimentación por baterías LiPo. Existe una versión denominada Arduino Pro Mini de tamaño más reducido.
- **Arduino Micro** es una evolución de la placa Arduino Pro Mini que tiene un tamaño aún más reducido.
- **Arduino Nano** es una evolución de la placa Arduino Micro con un tamaño todavía más reducido.
- **Arduino Yun** combina el chip del modelo Leonardo y un módulo SOC (*System-On-a-Chip*) que ejecuta una distribución de Linux llamada Linino. El chip Arduino está conectado al módulo Linux, por lo que es muy fácil que se comuniquen entre ambos y delegar procesos pesados a la máquina Linux integrada en la placa.
- **Arduino Industrial** es una evolución de Arduino Yun con mayor potencia y recursos de conectividad.
- **Arduino LilyPad** es una placa diseñada para llevar en la ropa y textiles electrónicos. Puede ser cosida a la tela y, de manera similar, montarse con baterías, sensores y dispositivos de interacción con hilo conductor.
- **Arduino Fio** es una placa diseñada a partir de Arduino LilyPad pero incluyendo conectividad XBee y pudiendo alimentarse directamente desde una batería de litio y polímero (conocidas como baterías LiPo) como las usadas en teléfonos móviles.

Además de las placas anteriores existen variantes con diferencias menores entre características, especialmente en cuanto a puertos y conectividad inalámbrica integrada. Como el mercado de placas Arduino está en constante evolución, se recomienda consultar una tienda [27] en la que se puedan explorar todas las posibilidades disponibles.

Si bien cualquier placa Arduino acepta la conexión de componentes electrónicos para complementar su funcionalidad, resulta habitual recurrir a componentes que encapsulan cierta funcionalidad, como por ejemplo las que reciben el nombre de “shields” y son comercializadas listas para usar [28] o incluso soluciones más avanzadas como sistemas integrados en un *chip* denominados *System on a Chip* (SoC).

Estas “shields” se conectan a la placa Arduino fácilmente ya que están pensadas para ser apiladas usando los conectores necesarios para su funcionamiento a la vez que mantienen el acceso a los conectores no usados. Un ejemplo de esto sería el uso de una de estas “shields” para dotar a una placa Arduino UNO de conectividad inalámbrica Bluetooth.

2.2 Software para programación en Arduino

La comunidad Arduino contempla los conceptos *open source* tanto para el *hardware* como para el *software*, ofreciendo un entorno de desarrollo o IDE (acrónimo del término inglés *Integrated Development Environment*) en versión web y también en versión *desktop* instalable en un PC.

La versión desktop es muy sencilla y tras la instalación basta con conectar una placa Arduino a un puerto USB del PC para poder programarlo. Un programa para Arduino se denomina *sketch* (se podría traducir como boceto o borrador) y en su versión más sencilla incluye una función *setup*, que se invoca una vez al iniciar Arduino, y una función *loop* que se invoca en un bucle infinito hasta que Arduino se apague (tal y como se aprecia en la Ilustración 3).

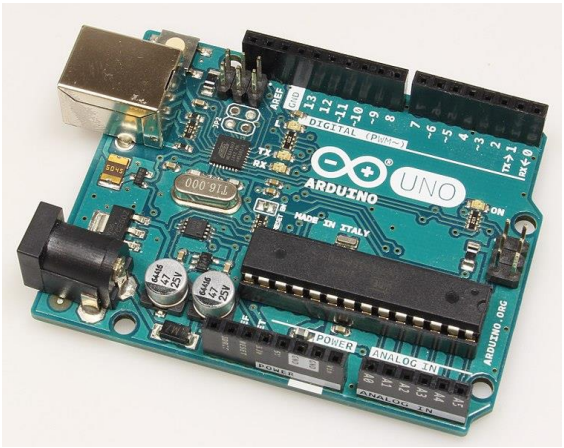

Ilustración 2[2]

Ilustración 3

La programación se hace en C/C++ utilizando una colección de librerías estándar u otras que se pueden conseguir de terceros que las ponen a disposición de los usuarios de Arduino con la filosofía de compartición de conocimiento inherentes a Arduino y su enfoque de *hardware* y *software open source*.

Las librerías de terceros, por ejemplo la librería u8g2 para manejar pantallas monocromas, pueden descargarse de repositorios de código en internet como GitHub [29] y luego ser incorporadas al IDe Arduino. No obstante, esto no suele ser necesario ya que el IDE de Arduino ofrece un sistema integrado para acceder a repositorios de librerías, descargarlas e instalarlas.

[2] Imagen de una placa Arduino UNO Cortesía de AdaFruit Industries y publicada en Flickr con licencia Attribution-NonCommercial-ShareAlike (https://creativecommons.org/licenses/by-nc-sa/2.0).

Además del IDE de Arduino, están disponibles otros también compatibles y con funcionalidades más avanzadas, por ejemplo Visual Micro (extensión para Visual Studio, el IDE comercializado por Microsoft).

Si bien el IDE de Arduino es suficiente para construir el código y transferirlo a las placas Arduino, no lo es para una gestión completa de la construcción de un prototipo. Además del código del *sketch* debe considerarse tanto los componentes electrónicos involucrados como su cableado y, para esto, es posible utilizar otros entornos *software*. Existen muchas opciones para diseñar circuitos, por ejemplo Autodesk Eagle, Proteus, KiCAD o Fritzing.

Fritzing resulta especialmente interesante porque es una iniciativa *open source* que ofrece una herramienta software para diseñar, documentar y, muy importante, crear esquemas de la placa de circuito impreso o PCB (acrónimo del término inglés *Printed Circuit Board*).

La obtención del esquema PCB es muy importante porque Arduino facilita la construcción de prototipos o proyectos electrónicos caseros, pero para uso profesional, especialmente para su comercialización, debe hacerse una implementación física del circuito con restricciones físicas (por ejemplo dimensiones) pero sobre todo económicas (buscando el menor coste). Esta construcción debe encargarse a empresas especializadas (por ejemplo PCBWay) a las que se les enviará el diseño PCB en un formato estándar.

Por último, mencionar que existen entornos de desarrollo para Arduino que reducen los conocimientos necesarios de programación C/C++ y que simplifican la implementación de diseños más elaborados al trabajar con una capa de abstracción muy superior basada en bloques e, incluso, la lógica de interacción entre ellos. Un ejemplo relevante de estos IDE es S4A, que permite realizar una programación puramente visual cuya resultado puede ser transferido a Arduino sin necesidad de escribir ningún fragmento de código C/C++.

La Ilustración 4 muestra un ejemplo de uso de Fritzing para diseñar un circuito que enciende un LED cuando se pulsa un botón y un ejemplo de la programación visual realizada con S4A[3] para ello.

[3] La imagen ha sido extraída de la documentación de S4A y dispone de licencia MIT (https://opensource.org/licenses/MIT).

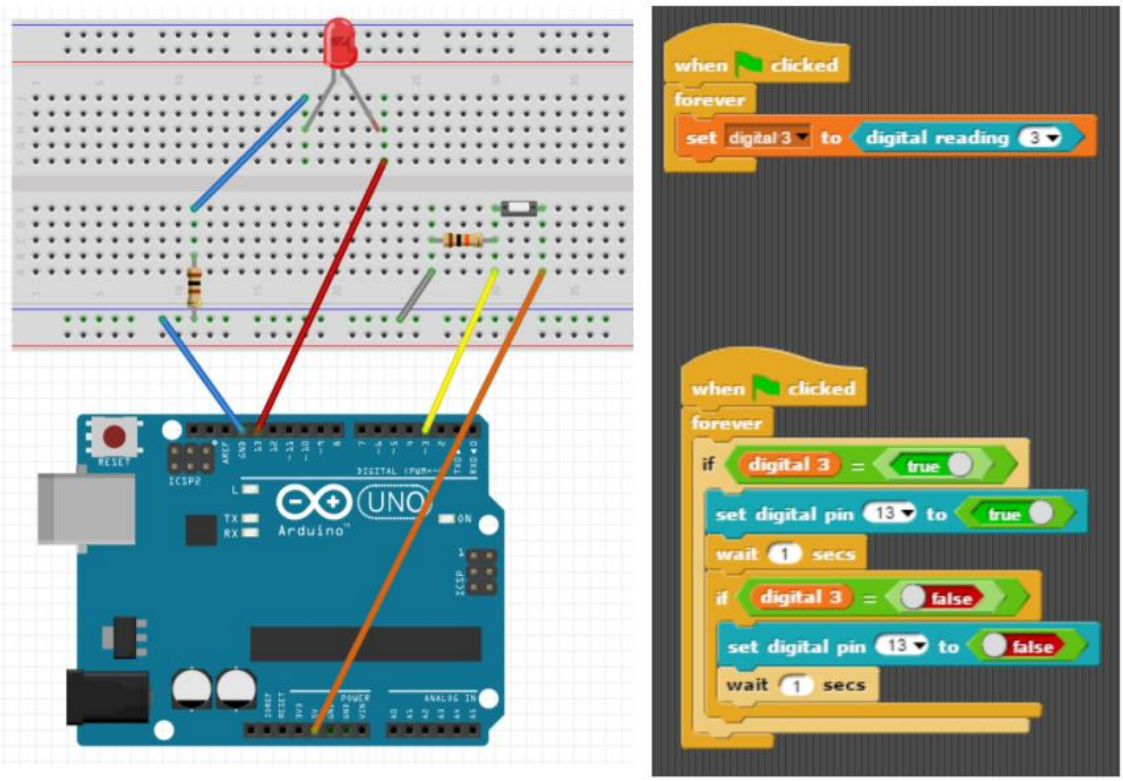

Ilustración 4

2.3 Software para integración de sistemas y explotación de datos

Arduino es una plataforma idónea para construcción de sistemas que capturan datos y los ponen a disposición del usuario para ser explotados. Esta idea está profundamente arraigada en el concepto de internet de las cosas (*Internet of Things*, IoT) donde todo está conectado con todo para compartir sus datos.

Si bien los datos capturados pueden ser compartidos usando cualquier mecanismo que lleve los datos a un repositorio donde puedan ser explotados por otros sistemas (por ejemplo una conexión WiFi que envíe datos en forma de fichero CSV a un FTP de un servidor que luego alimenta una base de datos Microsoft SQL Server), actualmente hay sistemas ya disponibles para utilizar sin tener que construirlos desde cero.

Un buen ejemplo es Xively, servicio online de almacenamiento de datos y explotación de los datos capturados que ha sido adquirido por Amazon e integrado en su plataforma de provisión de servicios en la nube Amazon Web Services (AWS) [30]. La Ilustración 5 muestra una visión de la arquitectura propuesta por Xively para su integración en aplicaciones que utilicen información de dispositivos IoT y que también pueden integrar con otros servicios de computación en la nube (*cloud computing*).

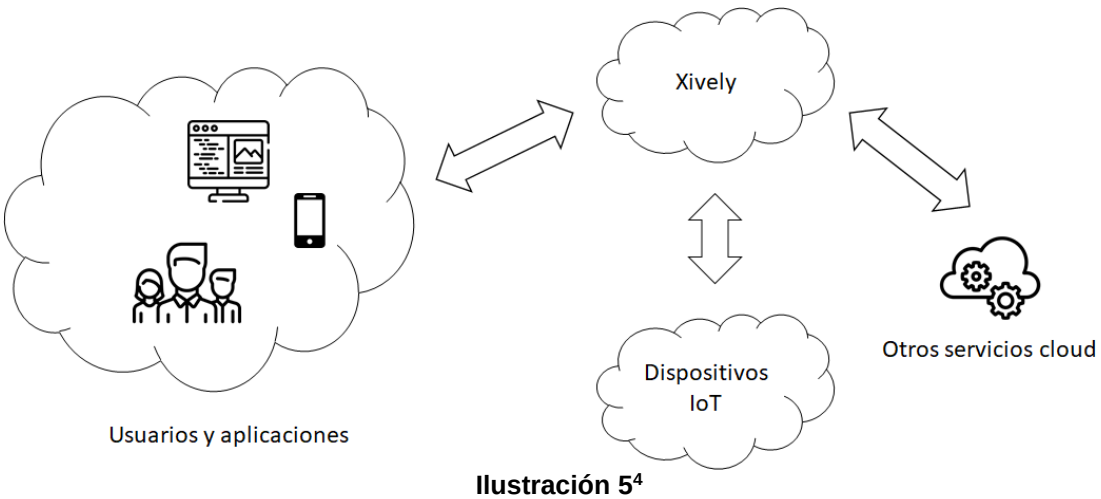

Ilustración 5[4]

Si bien Xively es un servicio muy relevante al estar respaldado por AWS, no es el único disponible en el mercado. Otros servicios de representación de datos capturados de forma visual son Plotly y ThingSpeak.

ThingSpeak es un servicio muy interesante porque ofrece licenciamiento gratuito para uso no comercial y está preparado para su integración con sistemas Arduino. Su utilización es muy sencilla porque se definen canales con campos que pueden leerse y actualizarse fácilmente mediante una sencilla API. Los canales representan contenedores de datos donde y los campos son los atributos de cada medida. Además, cada canal tiene atributos intrínsecos como, por ejemplo, la latitud y longitud para geolocalizarlo.

Toda esta información es explotable desde la interfaz web de ThingSpeak y desde sistemas de terceros que consumen los datos mediante la API disponible. Incluso existe una integración muy avanzada con MATLAB, el software para ingeniería de MathWorks, que de esta forma se posiciona en el sector IoT.

2.4 Sensores

Los sensores disponibles en el mercado son muy variados, por lo que se recomienda consultar una tienda [31] para explorar todas las posibilidades disponibles. No obstante, considerando el objetivo de este proyecto, se hará una preselección de sensores de condiciones ambientales que puedan ser de interés, haciendo hincapié en sus características más relevantes.

Es importante resaltar que un sensor para un equipo de protección no tiene los mismos requisitos que un sensor para la toma de mediciones en un proceso industrial. Por ejemplo, un sensor que mida temperatura para saber si es excesiva para trabajar en esas condiciones no necesita la misma precisión que un sensor que mida la temperatura en una fábrica de productos precocinados. Lo anterior permite considerar el uso de sensores más baratos, lo cual redunda en los requisitos y objetivos que se pretende alcanzar.

[4] Elaboración propia a partir de https://www.developerxively.com/docs/what-is-xively.

Calidad de aire y concentración de gases:

- Sensores para medir la concentración de distintos gases en el aire (CO, NO2, H2, NH3, CH4, etc.). Disponibles en el mercado por 30€ y que tienen la particularidad de utilizar canales de medición diferentes por lo que no todos los gases pueden medirse simultáneamente, aumentando el precio del sensor cuantos más canales disponga y, por tanto, más gases se puedan medir a la vez. Existen versiones más sencillas con un precio más reducido desde 4€.
- Sensores para detectar toxicidad del aire buscando la presencia de partículas en suspensión de más de 1 micrómetro de diámetro. Disponibles en el mercado por 15€.

Condiciones físicas del entorno:

- Sensores mixtos para medir presión, altura y temperatura. Disponibles en el mercado por 7€.
- Sensores específicos para medir temperatura ambiente (desde -55 hasta 125 grados centígrados). Disponibles en el mercado por 3€.
- Sensores para medir ruido. Disponibles en el mercado por 5€.
- Estaciones meteorológicas completas para medir la velocidad y dirección del viento así como la lluvia. Disponibles en el mercado por 69€.

2.5 Comunicación inalámbrica WiFi

Para dotar de comunicación inalámbrica a una placa Arduino y disponer de acceso a una red TCP/IP es posible utilizar alguna variante del protocolo IEEE 802.11 (WiFi), por ejemplo a través de una “shield” específica que ofrece conectividad mediante IEEE 802.11b/g. Sin embargo, esta solución ha caído en desuso y la *shield* ya no se comercializa porque es más sencillo disponer de conectividad WiFi mediante el módulo ESP8266 fabricado por Seed Studio [32].

El módulo ESP8266 [33] es un SOC (*System-On-a-Chip*) con un procesador dedicado y una antena integrada soldados a una placa de conexión de dimensiones muy reducidas. Otra ventaja del módulo ESP8266 es su reducido consumo de energía, especialmente interesante para dispositivos alimentados con baterías.

En la arquitectura de solución propuesta en la Ilustración 1, la comunicación WiFi puede utilizarse para interconectar los *wearables* con el elemento sensor y también para interconectar el elemento sensor con el centro de control. Sin embargo, en un caso de uso en el cual el elemento sensor se ubica en una zona con cobertura WiFi pero los trabajadores utilizan los *wearables* cerca del sensor pero en una zona sin cobertura WiFi, la conectividad entre el *wearable* y el sensor debe utilizar otra solución, como Bluetooth o Xbee.

2.6 Comunicación inalámbrica Bluetooth

Bluetooth es una especificación para comunicación inalámbrica pensada para interconectar dispositivos de uso personal y así crear pequeñas redes entre las que compartir datos. Ejemplos habituales son los dispositivos “manos libres” para telefonía en vehículos o los auriculares inalámbricos para telefonía móvil y reproductores de música.

La conexión mediante Bluetooth ofrece un alcance operativo que varía en función del dispositivo, categorizado en “clases” numeradas de la 1 a la 4.

Clase	Potencia máxima		Distancia máxima entre nodos (en metros)
	mW	dBm	
1	100	20	100
2	2,5	4	10
3	1	0	1
4	0,5	-3	0,5

La distancia máxima entre los dispositivos conectados puede ser una limitación, ya que la mayoría de los dispositivos comercializados son de clase 2 con un alcance máximo de 10 metros. Esto reduce el uso de Bluetooth, en la arquitectura de solución propuesta en la Ilustración 1, a interconectar los *wearables* con el elemento sensor y siempre que se esté en el rango de alcance, lo cual debe validarse experimentalmente en casos reales de uso.

En el mercado están disponibles módulos de clase 2 como el HC-05 [34] que permiten establecer redes para el intercambio de datos con distintas topologías, incluyendo punto a punto para emparejar dos dispositivos o difusión *broadcast*.

Si la conectividad mediante Bluetooth no es suficiente entonces debe utilizarse otra solución, como XBee.

2.7 Comunicación inalámbrica XBee

Los módulos XBee son una implementación de la especificación de comunicaciones inalámbricas Zigbee] comercializados por el fabricante Digi [35]. La funcionalidad y capacidad de los módulos XBee se incrementa en cada nueva generación (denominada “serie”) que Digi saca al mercado sin perder la característica fundamental de unas dimensiones sumamente reducidas.

Respecto a Bluetooth, hay dos diferencias fundamentales para considerar el uso de un módulo XBee:

- El alcance es muy superior. Por ejemplo, un módulo XBee de la serie 2 [36] tiene un alcance superior a los 100 metros transmitiendo con una potencia de 2 mW.

- El coste es muy superior. El módulo Bluetooth HC-05 tiene un coste de unos 5€ frente a los 44€ del módulo XBee de la serie 2 antes mencionado.

Por tanto, la elección de la solución para interconectar los *wearables* y el elemento sensor debe elegirse según el caso de uso, ya que si el alcance ofrecido por Bluetooth es suficiente y no es necesario utilizar XBee, el ahorro de costes es notable.

2.8 Alternativas hardware a Arduino

La placa ESP32 [18] del fabricante Espressif incluye un procesador mucho más veloz y potente que el incluido en las placas Arduino, mayor número de pines analógicos, conectividad WiFi y Bluetooth sin requerir componentes adicionales y todo ello en un tamaño mucho menor que las placas Arduino.

Incluso la programación puede hacerse de forma similar, ya que Espressif ha preparado lo necesario para utilizar el IDE oficial de Arduino para ser usado como entorno de programación de una placa ESP32. Para ello basta con descargar una librería con la información sobre la placa (puede hacerse desde el propio IDE de Arduino) y, entonces, elegir como placa el modelo ESP32.

Estas placas, en todas sus variantes [37], son una alternativa muy seria a las placas Arduino porque, además de mejorar las características *hardware*, su precio es mucho más reducido y se mantiene la compatibilidad con el ecosistema de productos compatibles con Arduino. A continuación se muestra una comparativa de las versiones más sencillas de ambos productos:

	ESP32-WROOM-32	Arduino UNO
Microprocesador	2 cores (32 bits)	1 (8 bits)
Frecuencia de reloj	40 MHz	16 MHz
Conectividad de red	WiFi (802.11 b/g/n) Bluetooth v4.2 B/EDR/BLE	No incluida
Memoria flash	4 MB	32 KB
Memoria SRAM	520 KB	2 KB
Pines analógicos	18 (resolución 12 bits)	6 (resolución 10 bits)
Pines digitales PWM	16	6
Dimensiones	18 x 25,50 mm	68,6 x 53,4 mm
Precio	Aprox. 4€	Aprox. 20€

2.9 Conclusiones

El sector de la seguridad laboral dentro del contexto de la Industria 4.0 evoluciona para ofrecer *Smart EPI* que los trabajadores puedan utilizar en su día a día de forma cómoda y sin molestias para realizar sus tareas pero

aportando nuevas ventajas tanto para facilitar su trabajo como para capturar información que la empresa pueda utilizar para mejorar sus procesos.

El mercado ya ofrece sensores y *wearables* que se apoyan en el concepto de IoT, pero sigue habiendo mucho espacio para la innovación ya que los casos de uso son casi ilimitados, lo que permite ofrecer soluciones novedosas que cubran necesidades reales que el mercado todavía no cubre.

Considerando las posibilidades tecnológicas identificadas y los objetivos, la tecnología que se propone utilizar para construir la solución descrita en la arquitectura se resume en la Ilustración 6 y consiste en:

- Conectividad WiFi entre el elemento sensor y el control de operaciones.
- Envío de datos capturados por el sensor a ThingSpeak para su posterior explotación.
- Conectividad Bluetooh o Xbee entre el elemento sensor y los *Smart EPI wearables*.
- Elemento sensor y *Smart EPI wearable* construido utilizando una placa Arduino o ESP32.

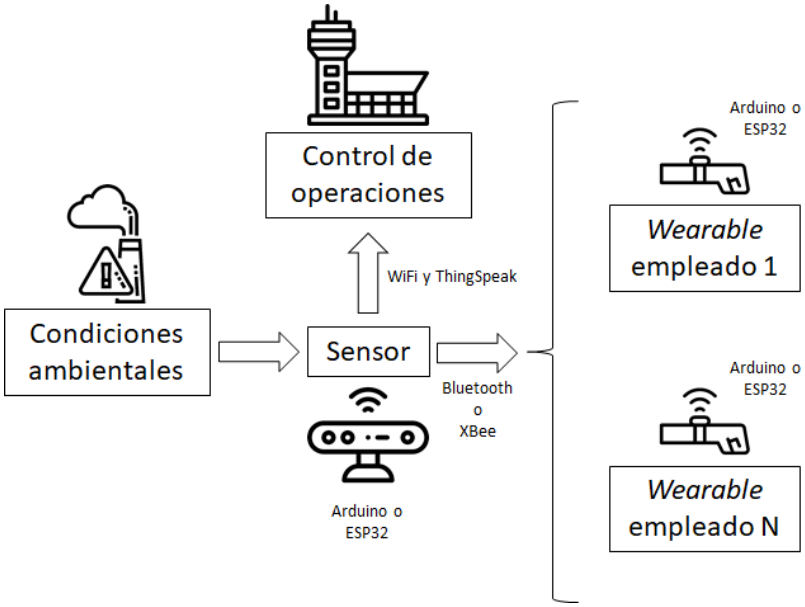

Ilustración 6

3 Proyecto

3.1 Metodología

El Trabajo Final de Grado debe conseguir unos entregables que son consecuencia tanto del propio propósito del Trabajo Final de Grado (por ejemplo una memoria) como de los objetivos planteados (por ejemplo un prototipo funcional).

La metodología que se utilizará para la realización del Trabajo Final de Grado pone el foco en los entregables y los condicionantes de plazos y fechas asociados a ellos. Por tanto, para garantizar que los entregables se construyan con la calidad y plazos requeridos, se requiere una planificación rigurosa que se acompañe de una gestión de riesgos que permita actuar proactivamente ante desviaciones que pongan en riesgo los entregables o el cumplimiento de plazos.

Así mismo, se incluirá un detalle de los costes de realización del prototipo funcional y una estimación del coste de un sistema comercializable para tratar de comprobar la viabilidad de la solución propuesta.

3.2 Entregables y planificación

Los entregables previstos son:

1. La memoria del Trabajo Final de Grado que incluye:
    a. El estudio del estado del arte requerido para adquirir el conocimiento y el contexto necesario para la construcción del prototipo funcional.
    b. La descripción de la solución técnica del prototipo funcional construido.
    c. Una reflexión crítica respecto a los objetivos alcanzados (y no alcanzados).
2. El prototipo funcional construido.
3. El material de apoyo necesario para presentar y defender el Trabajo Final de Grado ante los evaluadores.

La planificación considera una descomposición del proyecto en tareas con dependencias entre ellas y con limitaciones para su ejecución consecuencia de que el proyecto se realiza un único recurso: el propio autor del Trabajo Final de Grado.

Los elementos principales de la planificación son:

- Una fase de investigación para recopilar la información y conocimiento necesario para elaborar el estado del arte.
- Una fase de construcción del prototipo funcional.

- Una fase final de elaboración de las versiones finales de los entregables documentales, ya que versiones preliminares se irán construyendo a la par que las otras tareas son ejecutadas.
- Además se requieren unas tareas de gestión para la revisión y actualización de la planificación y los riesgos.

La planificación se detalla en un diagrama Gantt construido usando Microsoft Project, cuyo resumen de tareas y plazos se resumen en la Ilustración 7. Todas las tareas se han completado en las fechas previstas. No obstante, respecto a la planificación inicial, se han añadido nuevas tareas de seguimiento y actualización de la planificación teniendo en cuenta las fechas límite definitivas para los entregables documentales exigidos.

Task Name	Duration	Start	Finish
Inicio TFG	0 days	Mon 25/02/19	Mon 25/02/19
Gestión	**85 days**	**Mon 25/02/19**	**Fri 21/06/19**
Actualizar planificación	1 day	Mon 11/03/19	Mon 11/03/19
Actualizar planificación	1 day	Mon 25/03/19	Mon 25/03/19
Actualizar planificación	1 day	Mon 08/04/19	Mon 08/04/19
Actualizar planificación	1 day	Mon 22/04/19	Mon 22/04/19
Actualizar planificación	1 day	Mon 06/05/19	Mon 06/05/19
Actualizar planificación	1 day	Fri 17/05/19	Fri 17/05/19
Actualizar planificación	1 day	Fri 31/05/19	Fri 31/05/19
Actualizar planificación	1 day	Fri 07/06/19	Fri 07/06/19
Actualizar planificación	1 day	Fri 14/06/19	Fri 14/06/19
Investigación	**20 days**	**Mon 25/02/19**	**Sun 24/03/19**
Formación en Arduino	21 days	Mon 25/02/19	Sun 24/03/19
Estado del arte en wearables y EPIs	11 days	Mon 04/03/19	Sun 17/03/19
Prototipo	**45 days**	**Mon 18/03/19**	**Fri 17/05/19**
FASE 1: Elemento sensor simulado con captura y visualización de datos	3 days	Mon 18/03/19	Wed 20/03/19
FASE 2: Elemento sensor real con captura y visualización de datos	3 days	Thu 21/03/19	Sun 24/03/19
Fase 3: Elemento sensor con conexión inalámbrica al elemento wearable	5 days	Mon 25/03/19	Fri 29/03/19
Fase 4: Elemento sensor con conexión inalámbrica al elemento wearable con alerta	6 days	Sat 30/03/19	Fri 05/04/19
Fase 5: Elemento sensor con integración web para explotación de datos	7 days	Sat 06/04/19	Sun 14/04/19
FASE 6: Ampliaciones y mejoras (usabilidad, funcionalidad, etc.)	25 days	Mon 15/04/19	Fri 17/05/19
Entregables documentales	**85 days**	**Mon 25/02/19**	**Fri 21/06/19**
Elaboración borrador memoria	62 days	Mon 25/02/19	Tue 21/05/19
Elaboción versión final memoria	14 days	Wed 22/05/19	Sun 09/06/19
Elaboración de la presentación	6 days	Mon 10/06/19	Sun 16/06/19
Defensa TFG	5 days	Mon 17/06/19	Fri 21/06/19
Fin TFG	0 days	Fri 21/06/19	Fri 21/06/19

Ilustración 7

3.3 Riesgos

Si bien un análisis de riesgos puede realizarse de forma muy rigurosa, incluso tomando como referencia la norma ISO 31000, para el objeto de este proyecto se utilizará una aproximación sencilla donde se identificarán los riesgos y las contramedidas previstas.

Es importante recordar que dentro de las tareas se considera una revisión semanal de la planificación que se basa fundamentalmente en una revisión del grado de avance real sobre el previsto y en los riesgos ya identificados que se hayan materializado. Esa revisión semanal también incluye una actualización de los riesgos en base a nuevos escenarios que no se hubieran considerado previamente.

A continuación se muestran los riesgos identificados a fecha 27/05/2019, si bien ninguno de ellos llegó a materializarse.

Riesgos	Contramedidas
Dedicación insuficiente por motivos laborales que ponga en riesgo el cumplimiento de la planificación.	• Revisión de la planificación para adecuarla a la dedicación real (sin afectar a los objetivos previstos). A este efecto la planificación incorpora dos semanas de margen entre la finalización de la última tarea y la fecha final de presentación del TFG. • Solicitud de vacaciones durante la realización del TFG para aumentar la dedicación.
Incumplimiento de los objetivos previstos inicialmente.	• Eliminar del prototipo funcionalidades o características que puedan ser consideradas “extras” y que podrían pasar a formar parte de una evolución posterior.
Falta de un emplazamiento adecuado para la construcción del prototipo funcional.	• Solicitud de permiso para emplear o alquilar una sala propiedad de una empresa o de algún centro de formación.

4 Construcción del prototipo funcional

Para construir el prototipo funcional se ha optado por seguir una serie de pasos incrementales. Cada fase consiste en una evolución incremental sobre la anterior que deberá estar validada. Este enfoque dará robustez al proceso y permitirá avanzar poniendo el foco en las novedades que aporta cada fase respecto a la anterior.

La alternativa no es eficiente porque si se optase por construir directamente el sistema en su totalidad, cualquier funcionamiento erróneo obligaría a revisar todas las partes del sistema. Por el contrario, el enfoque incremental acota el diagnóstico de errores a los cambios incorporados en una fase respecto a la anterior ya validada.

Así mismo, se utilizará este enfoque incremental para ir evolucionando el código de los distintos *sketch* utilizados incorporando prácticas más avanzadas de ingeniería del *software* (por ejemplo uso de programación orientada a objetos, gestión de eventos mediante interrupciones *hardware*, etc.) y del conocimiento y experiencia adquiridos en las fases previas.

4.1 Fase 1: Simulación de captura de datos y visualización

El objetivo de esta fase es disponer de una primera versión del elemento sensor en el que los datos son recogidos y mostrados en una pantalla.

Se utilizará una placa Arduino UNO y una pantalla OLED. Al ser este el primer contacto con Arduino, no se utilizará un sensor real sino que se utilizará un potenciómetro del cual se leerá el valor de tensión. Este enfoque permite prescindir, por ahora, de la complejidad de leer e interpretar el dato que ofrecería un sensor real.

El *sketch* con el que se programará el Arduino UNO tendrá una función *setup* que inicializará Arduino y una función *loop* que se ejecutará indefinidamente con la lógica que figura en la Ilustración 8. El código del *sketch* se incluye en el epígrafe 8.1 del anexo.

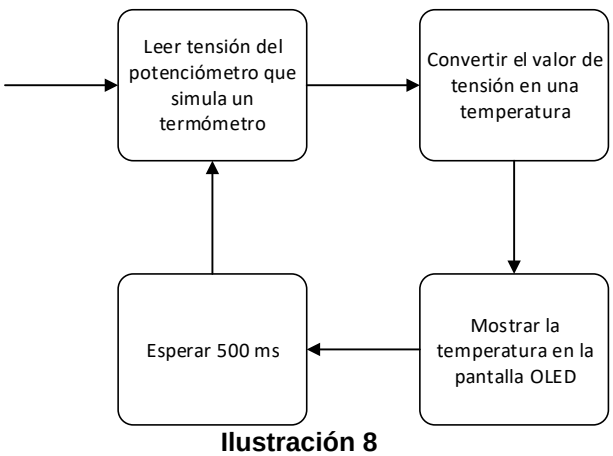

Ilustración 8

La escritura en la pantalla OLED se realiza utilizando la librería u8g2 [29] que permite abstraerse de los detalles exactos de la pantalla. La mayor dificultad para utilizar la pantalla OLED con esta librería es identificar correctamente qué puerto representa cada parámetro y así conectarlo correctamente al pin digital de Arduino.

4.2 Fase 2: Captura y visualización de datos de un sensor real

En esta fase se reemplaza el potenciómetro por un sensor real. El sensor elegido es un sensor para medición de temperatura TMP36 [38], ya que la validación de la medida así como provocar cambios en el entorno para que la medición varíe es sumamente sencillo en comparación con, por ejemplo, un sensor que mida concentración de gases tóxicos. Este sensor es muy económico y ofrece medidas con suficiente precisión (alrededor de 1º C) para su uso como parte de un EPI que proteja a trabajadores frente a temperaturas excesivas.

La modificación sobre la fase anterior es mínima ya que el sensor se comporta como un divisor de tensión, por lo que la lectura se realiza del mismo modo y tan solo hay que añadir la transformación entre el valor de tensión leído y la temperatura a la que corresponde.

Los sensores que operan como divisores de tensión son muy sencillos de incorporar ya que la sistemática es siempre la misma:

- Conectar el sensor a la tensión de referencia y a tierra.
- Conectar el sensor a un pin analógico de Arduino.
- Leer el valor del puerto analógico y escalarlo al rango de tensión.
- Aplicar la conversión de la tensión a la magnitud medida por el sensor.

En caso de necesitar utilizar varios sensores de forma simultánea, la única condición es tener suficientes pines analógicos disponibles, lo que debe tenerse presente a la hora de elegir el modelo de placa Arduino.

Considerando que la medición del sensor, en este caso una temperatura, se comprobará que esté dentro de un rango de valores que representen unas condiciones de trabajo seguras, es importante gestionar correctamente las mediciones que estén cerca de los valores frontera. Por ejemplo, si se quisiera medir la luz ambiente para comparar con un umbral a partir del cual encender o apagar iluminación artificial adicional, los valores que en cuestión de segundos sean inferiores y superiores al umbral provocarían el constante encendido y apagado de la iluminación, lo que podría ser muy molesto. Para evitar esta situación es necesario implementar un comparador con histéresis o aplicar un filtro pasa-bajo en el valor medido de forma que los cambios de alta frecuencia sean ignorados.

El *sketch* con el que se programará el Arduino UNO tendrá una función *setup* que inicializará Arduino y una función *loop* que se ejecutará indefinidamente

con la lógica que figura en la Ilustración 9. El código del *sketch* se incluye en el epígrafe 8.1 del anexo.

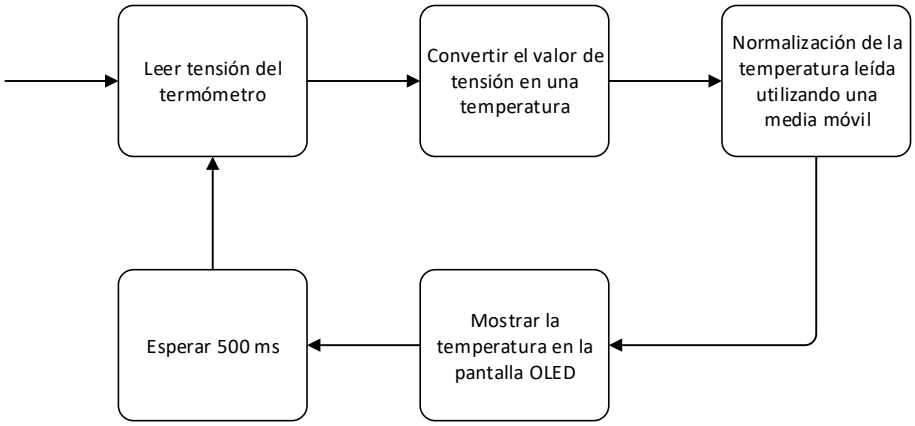

Ilustración 9

4.3 Fase 3: Sensor y wearable con conexión inalámbrica

El objetivo de esta fase es interconectar el elemento sensor y el elemento *wearable* de forma inalámbrica y validar la conectividad enviando datos de forma unidireccional entre el sensor y el *wearable*.

Para construir un *Smart EPI* que sea un dispositivo *wearable* interesa un tamaño mínimo y, por tanto, lo ideal sería utilizar una placa Arduino Nano, Lilypad o Fio. Sin embargo, con el objetivo de construir un prototipo con el menor coste posible se utilizará una placa Arduino UNO.

La conectividad entre el elemento sensor y el dispositivo *wearable* debería requerir poco consumo de energía y tener un alcance suficiente para que los trabajadores puedan estar alejados del elemento sensor. Nuevamente, con el objetivo de construir un prototipo con el menor coste posible se utilizará como conectividad inalámbrica la tecnología Bluetooth, ya que la utilización de XBee representa un coste muy superior.

Evoluciones posteriores, por ejemplo para la realización de un ensayo piloto en campo, requerirían la revisión del dispositivo *wearable* y la tecnología de conexión inalámbrica para reducir tamaños y aumentar alcances, aunque esto implicase mayores costes.

Para dotar de conectividad Bluetooth a las placas Arduino UNO se les conectará un módulo Bluetooth HC-05 el cual deberá configurarse previamente. Los detalles de esta configuración se incluyen en el epígrafe 8.3 del anexo.

En esta fase la construcción del prototipo se divide en dos partes: el elemento sensor y el elemento *Smart EPI wearable*.

El elemento sensor de esta fase es el resultado de tomar el prototipo de la fase anterior al que se le conecta el módulo Bluetooth HC-05 que previamente se ha configurado. Además, se modifica el *sketch* que ejecuta para que tras leer valor

de temperatura del sensor, lo envíe por un puerto serie *software* que actúa como interfaz Bluetooth, tal y como se muestra en la Ilustración 10.

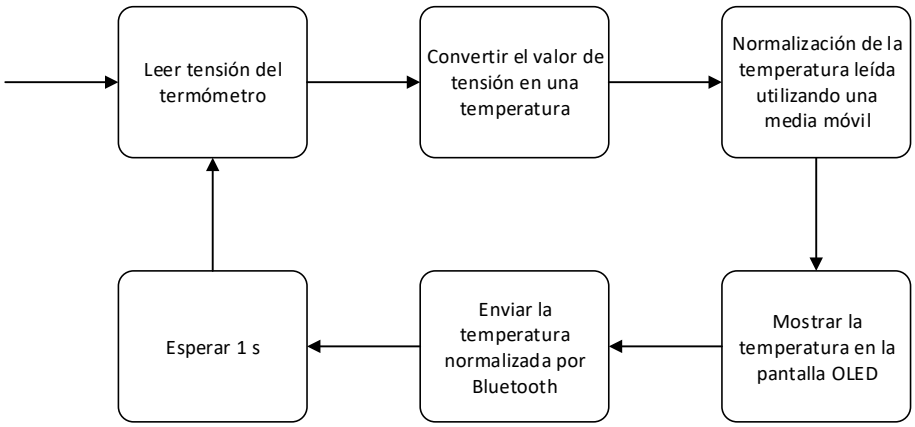

Ilustración 10

El *wearable* se construye utilizando una placa Arduino UNO a la que se le conecta el módulo Bluetooth HC-05 configurado previamente y que ejecutará un *sketch* que leerá los datos recibidos por el puerto serie software que actúa como interfaz Bluetooth. La lógica consiste en esperar a que se reciban datos por el puerto Bluetooth y, cuando se reciben, mostrarlos por el monitor serie utilizado para depuración, tal y como se muestra en la Ilustración 11.

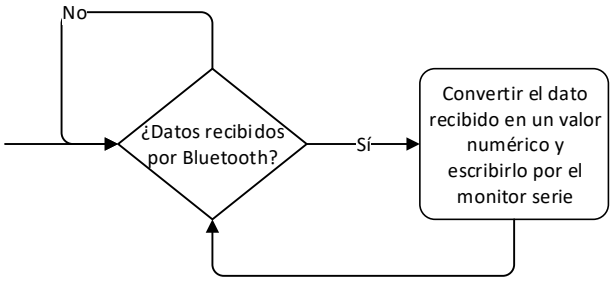

Ilustración 11

El resultado son dos *protoboards* conectadas cada una a una placa Arduino que representan el elemento sensor y el elemento *wearable* que, cuando se alimentan, la que representa el sensor captura el dato de temperatura y lo envía por Bluetooth para que la otra placa que representa el *wearable* muestre el valor y así confirmar que la conexión se ha establecido y los datos se intercambian.

Los *sketchs* que ejecutan el elemento sensor y el wearable se incluyen en los epígrafes 8.4 y 8.5 del anexo. Es especialmente interesante el código que representa el envío de los datos a través del puerto serie *software* que actúa como interfaz con el módulo Bluetooth, ya que se ha implementado un mecanismo de serialización de datos muy sencillo, en el que el valor entero se transforma en su representación textual y esta se envía añadiendo un salto de línea como terminador. Así, el código del elemento *wearable* que recibirá los datos – también por un puerto serie *software*- deberá leer caracteres hasta el salto de línea y realizar una conversión del texto al valor entero equivalente.

Para que las evoluciones de cada fase con respecto a la anterior sean incrementos pequeños fácilmente verificables, en este punto de la construcción del prototipo entregable final quedan varias cuestiones por tratar, como son:

#3.1	Confirmar que se puede ofrecer una conexión 1 a N donde un elemento sensor es capaz de enviar datos a N>1 elementos wearables. Esto requiere confirmar que la configuración Bluetooth elegida sea adecuada y modificarla en caso contrario.
#3.2	Establecer un protocolo más robusto de comunicación entre el sensor y los *wearables* para que se realice comunicación unidireccional sensor-*wearable* gestionando la serialización de los datos enviados.
#3.3	Evolucionar aún más el protocolo para gestionar comunicaciones bidireccionales y que los *wearables* puedan responder al sensor, lo que permitiría añadir mayor funcionalidad.

4.4 Fase 4: Sensor y wearable interconectados con gestión de alertas

El propósito de esta fase es evolucionar las dos partes del prototipo elaborado en la fase anterior. Lo primero es incorporar una interfaz de usuario para configurar en el elemento sensor el valor umbral a partir del cual las mediciones del sensor representan condiciones peligrosas que activen una alarma.

Para ello se utilizará nuevamente un potenciómetro, en este caso para representar una tensión que se convierta en una temperatura con un algoritmo idéntico al usado para el sensor de temperatura. De esta forma se dispondrá de dos temperaturas: la leída por el sensor y la que se establece a través del potenciómetro. Así el potenciómetro será la interfaz de usuario (obviamente muy simple) para configurar el elemento sensor al establecer la condición de disparo de la alarma.

La Ilustración 12 muestra el diagrama de conexiones representado utilizando Fritzing. En la Ilustración 13 se muestra el elemento sensor midiendo una temperatura de 17º C con una referencia de 13º C, por lo que se cumple la condición de disparo de la alarma al superar la temperatura ambiente el valor máximo de referencia establecido mediante el potenciómetro.

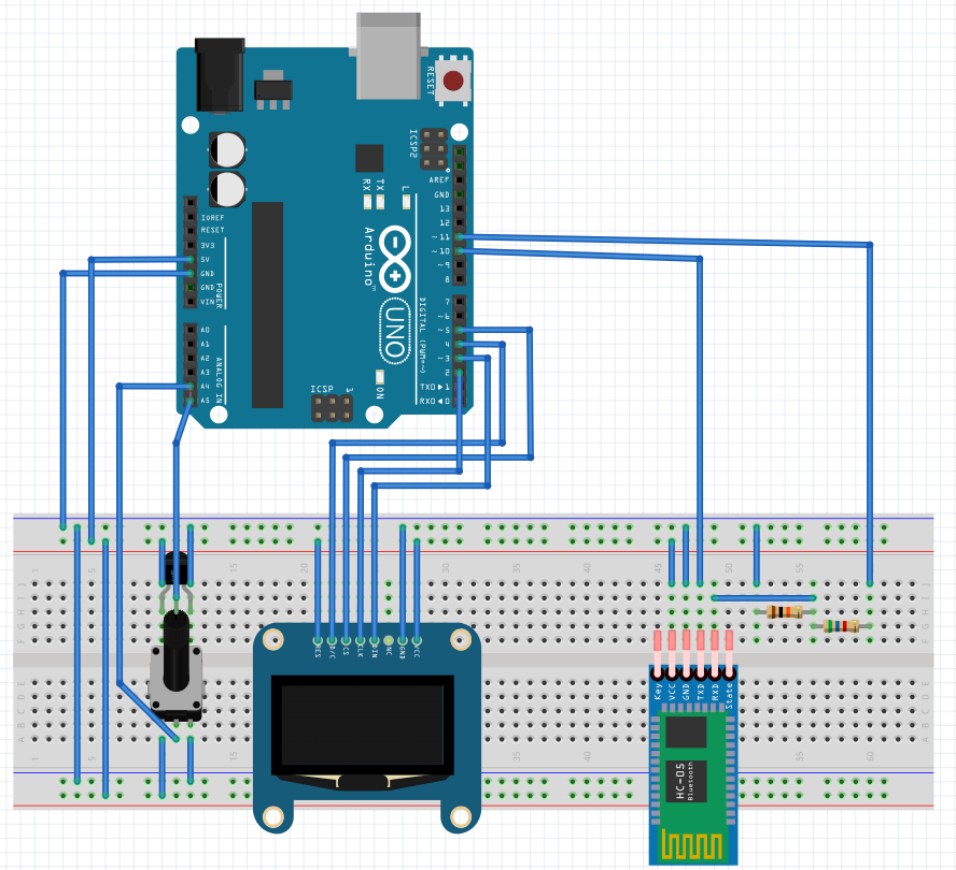

Ilustración 12

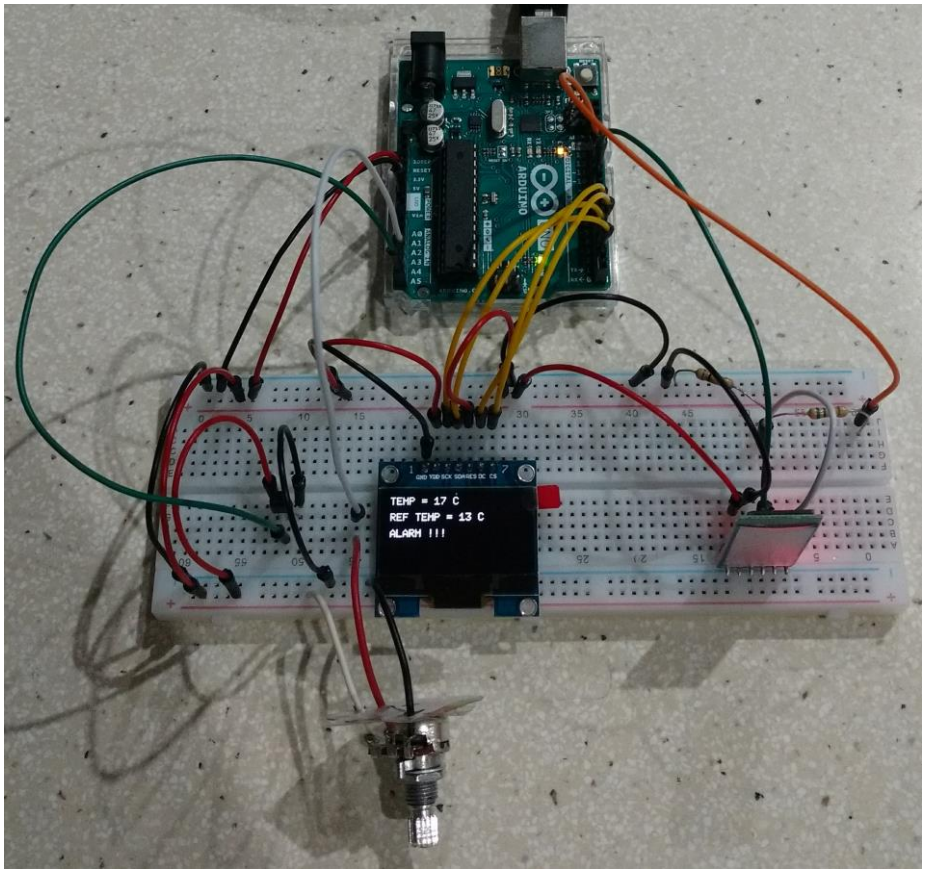

Ilustración 13

La lógica del *sketch* que ejecuta el elemento sensor, cuyo código se incluye en el epígrafe 8.6 del anexo, se describe en el siguiente diagrama, donde se aprecia que, para no generar tráfico innecesario, los valores leídos sólo son enviados por Bluetooth cuando hay variación. Este evento es el mismo que se utiliza para redibujar la pantalla OLED, ya que esta preserva su contenido hasta que llega un nuevo comando de dibujo.

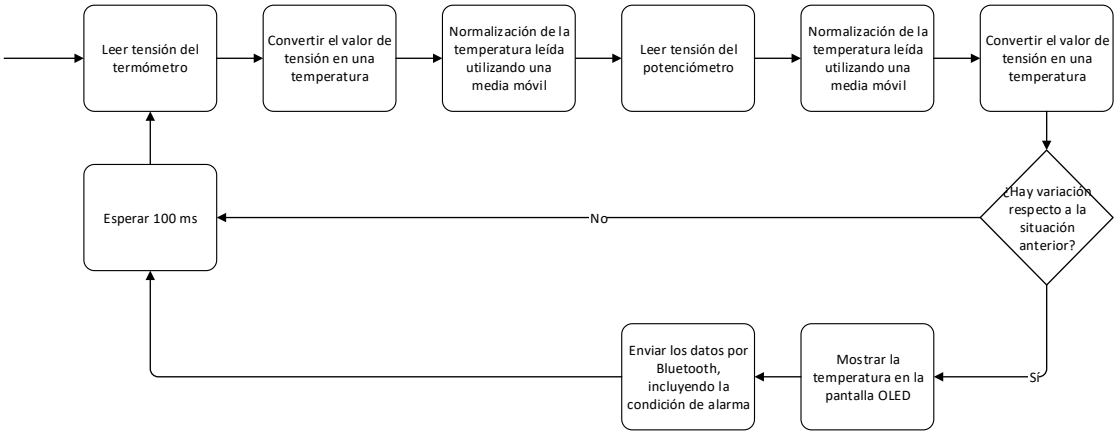

Ilustración 14

Por último, se modificará el elemento *wearable* para que reciba las notificaciones de alarma y muestre esta situación a través de una interfaz de usuario. Para ello se conectará un LED, un zumbador y un botón al *wearable* de forma que el LED estará encendido mientras el sensor envíe la situación de alarma y el zumbador vibrará mientras el LED esté encendido y no se pulse el botón.

A continuación se muestra el diagrama con la lógica del *sketch* que ejecuta el *wearable* y que se incluye en el epígrafe 8.7 del anexo.

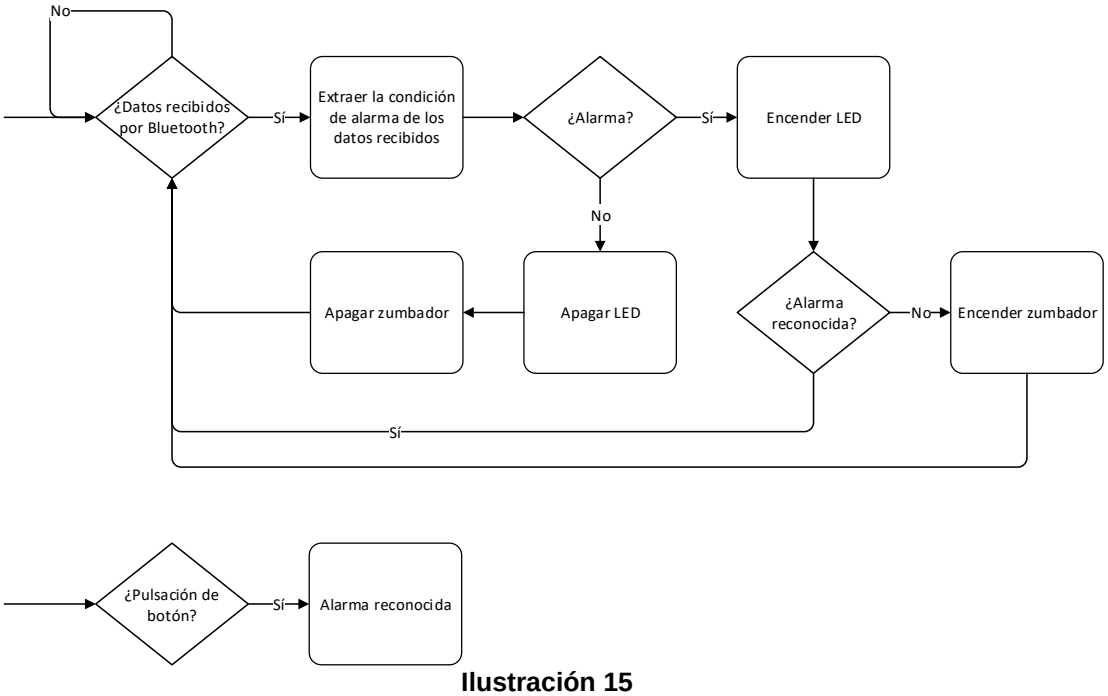

Ilustración 15

El LED, el zumbador y el botón representan la interfaz de usuario (muy simple) para el elemento *wearable* que permiten al empleado que lo utiliza conocer si hay una condición de alarma activada y aceptar esa alarma para que aunque se mantengan las condiciones, la notificación más activa a través del zumbador, no sea excesivamente molesta.

La Ilustración 16 muestra el diagrama de conexiones representado utilizando Fritzing mientras que la Ilustración 17 se muestra el resultado cuando la condición de alarma está activada.

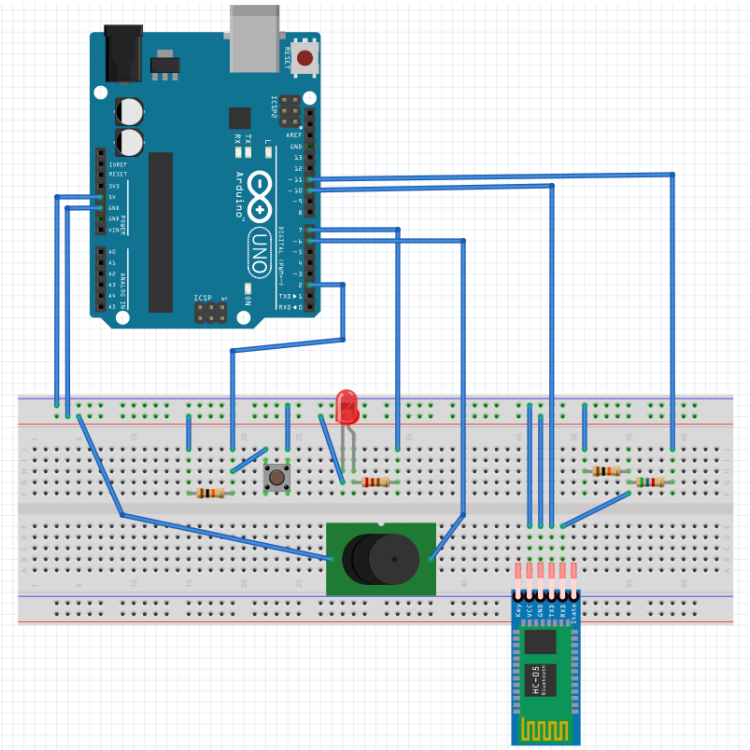

Ilustración 16

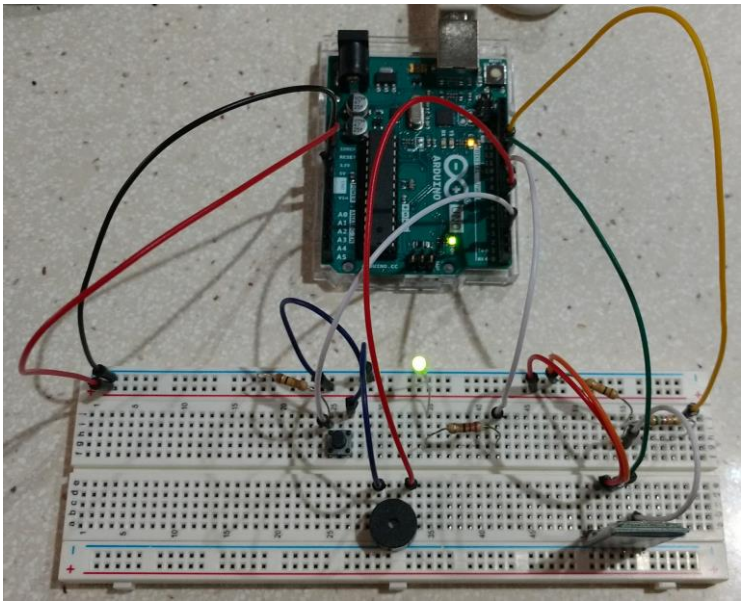

Ilustración 17

Por último, en este punto de la construcción del prototipo entregable final se identifican las siguientes cuestiones por tratar:

#4.1	Modificar el elemento sensor para incorporar varios sensores distintos adecuados al escenario del caso de uso real.
#4.2	Dotar de una interfaz de usuario de configuración adecuada para establecer las condiciones de alarma de los distintos sensores utilizados.
#4.3	Dotar de una interfaz de usuario al elemento *wearable* adecuada al escenario del caso de uso real.

4.5 Fase 5: Integración del sensor con un sistema de terceros

El propósito de esta fase es evolucionar el elemento sensor del prototipo elaborado en la fase anterior para integrarlo con un sistema de terceros al que enviar los datos recopilados. Esto demostrará la posibilidad de interconectar el *Smart EPI* con el centro de control y así permitir la incorporación de mayor valor al permitir la explotación de los datos capturados en combinación con otra información ya existente en otros sistemas dentro del concepto de IoT e Industria 4.0.

El sistema de terceros elegido es ThingSpeak, por sencillez, pero sin pérdida de generalidad. En él se configuró un canal que representa el sensor con un único campo para almacenar las medidas de temperatura que recoge. El canal se habilitó para uso público y así recibir datos de forma anónima siempre que se conozca la clave para el uso de la API. Este enfoque permite que una simple petición GET con los parámetros adecuados actualice la información registrada en ThingSpeak.

```
GET /update?api_key=<CLAVE API>&field1=<TEMPERATURA>&headers=false HTTP/1.1
Host: <IP DEL MÓDULO ESP8266>
Connection: close
Accept: */*
```

Ilustración 18

La integración con ThingSpeak requiere acceso a internet desde el elemento sensor, lo que se resuelve dotándole de conectividad WiFi. Para ello a la placa Arduino UNO que conforma el elemento sensor se le conectará un módulo ESP8266-01 convenientemente configurado mediante comandos AT, tal y como se detalla en el epígrafe 8.8 del anexo.

De este modo la placa Arduino UNO dispone de conectividad WiFi a través del módulo ESP8266-01, con el que interactuará a través de un puerto serie *software* en el cual podrá escribir para enviar los comandos AT necesarios para realizar una petición GET y leer la respuesta a dichos comandos. Los detalles de la conexión del módulo ESP8266-01 a la placa Arduino UNO se explican en el epígrafe 8.9 del anexo.

La lógica del *sketch* que ejecuta el elemento sensor, cuyo código se incluye en el epígrafe 8.10 del anexo, se describe en el diagrama que se muestra a continuación, que difiere de la fase anterior en que se envían los datos periódicamente al repositorio web.

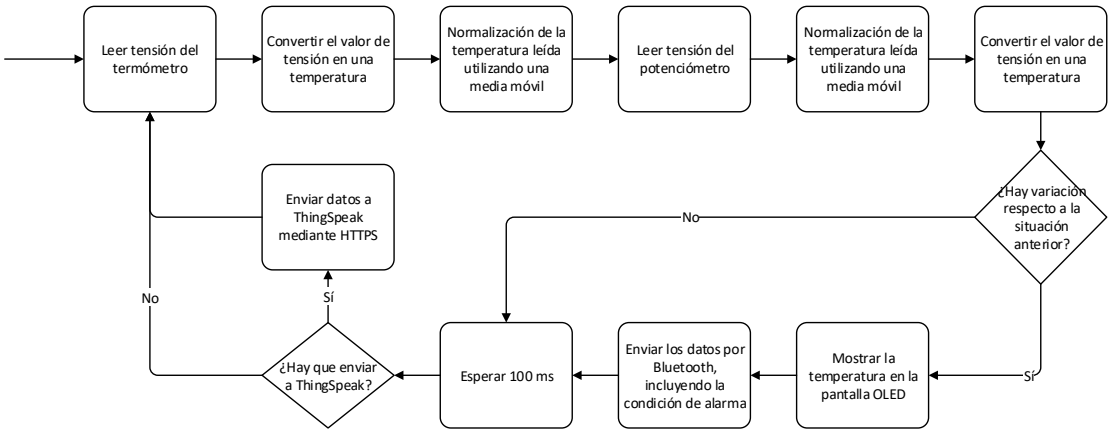

Ilustración 19

El resultado de la integración con ThingSpeak, enviando las mediciones de temperatura cada dos minutos, ofrece un resultado plenamente satisfactorio, tal y como se aprecia en la Ilustración 20.

Por otra parte, una vez que los datos están en la plataforma ThingSpeak, integrarlos en otros sistemas es muy sencillo. A modo de ejemplo, en la Ilustración 21 se muestra uno de los gráficos construidos sobre el canal siendo visualizado en el navegador web de un *Smartphone*.

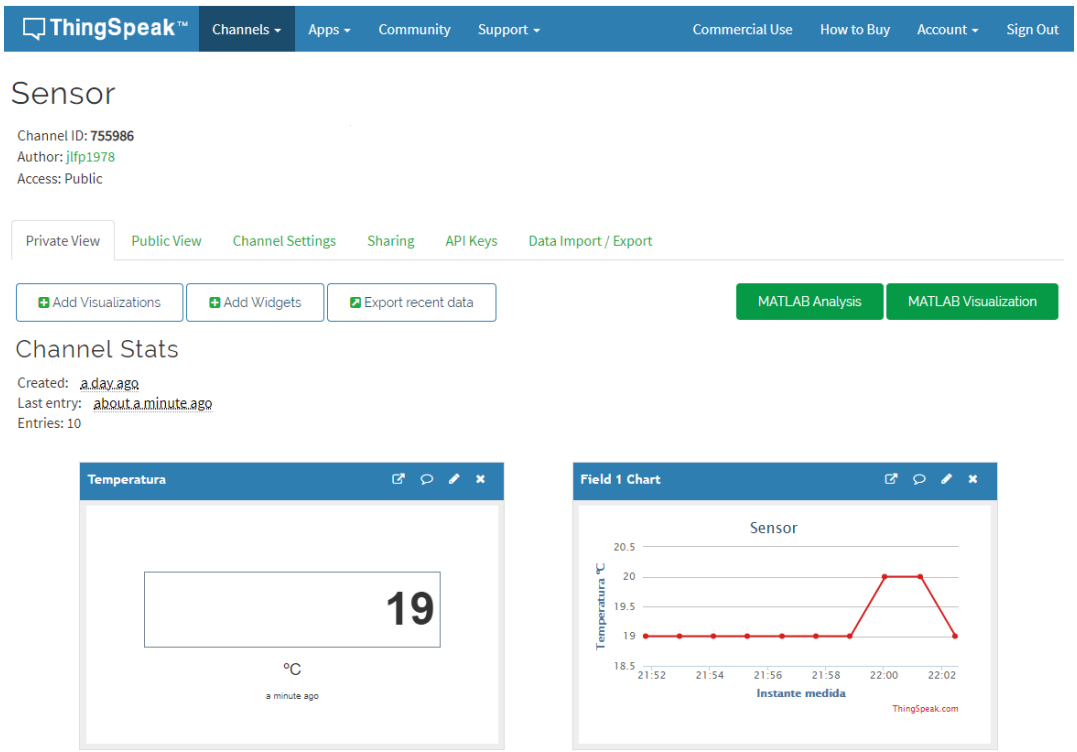

Ilustración 20

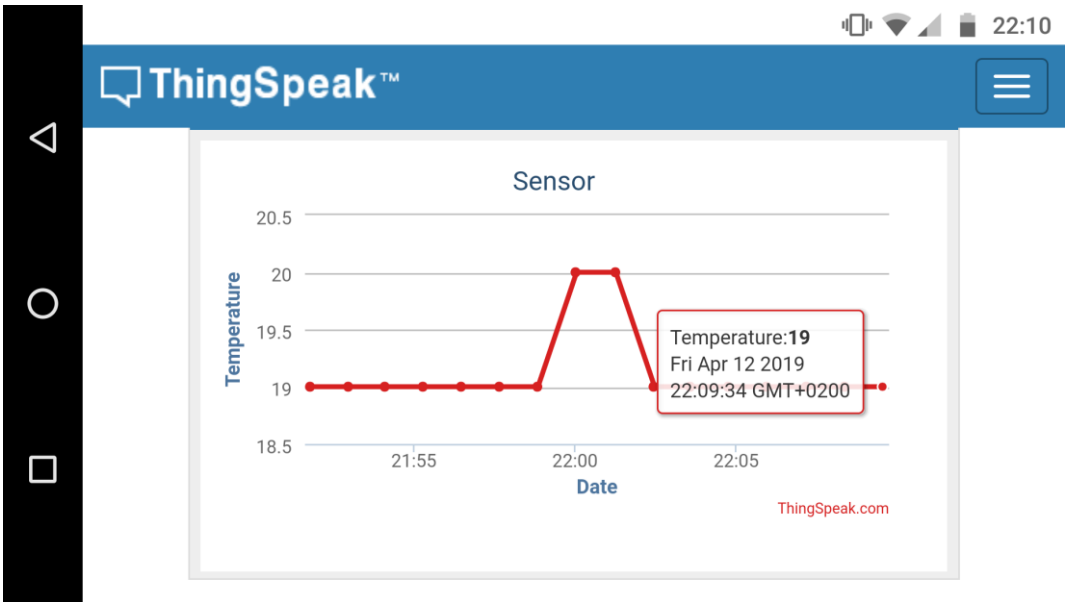

Ilustración 21

En este punto de la construcción del prototipo entregable final quedan varias cuestiones por tratar, como son:

#5.1	Dotar de una interfaz de usuario de configuración adecuada para establecer la conexión WiFi y los demás parámetros de funcionamiento.
#5.2	Incorporar un mecanismo de control de errores en las comunicaciones WiFi.
#5.3	Considerar reemplazar la placa Arduino UNO por el propio módulo ESP8266. El módulo ESP8266 es un SoC (*System on Chip*) similar al módulo ESP32. Sus características son más limitadas que el módulo ESP32, pero aun así son muy superiores a Arduino UNO.
#5.4	Explorar opciones de explotación de datos más avanzadas usando las funcionalidades propias de ThingSpeak o mediante la utilización de otras herramientas que interactúen con los datos almacenados en ThingSpeak.

La cuestión #5.3 es muy relevante porque se podría programar el módulo ESP8266 para ejecutar un *sketch* similar al utilizado en Arduino, con lo que se reaprovecharía gran parte del trabajo ya realizado, pero con la ventaja de tener acceso a la conexión WiFi de forma directa. Así se evitaría el uso de un puerto *software* serie y el manejo de comandos AT, lo que simplifica el acceso y uso de la conexión WiFi al disponer de abstracciones de mayor nivel, como las ofrecidas por la librería ESP8266WiFi [39] (por ejemplo para disponer de un cliente HTTP completo).

4.6 Fase 6: Ampliaciones y mejoras

En esta fase se abordarán algunas de cuestiones que quedaron pendientes en fases anteriores. Para ello el punto de partida será abandonar la tecnología Arduino y pasar a utilizar M5Stack [40] que es un kit de desarrollo basado en ESP32 compatible con Arduino.

La posibilidad de usar M5Stack no se había identificado en la etapa de estudio del estado del arte, pero surgió durante la construcción de las distintas fases previas del prototipo. Su utilización ofrece múltiples ventajas para abordar cuestiones pendientes de las fases anteriores.

Las cuestiones pendientes que se resuelven en esta fase son:

#3.1	Para garantizar la conexión 1 a N entre el elemento sensor y los distintos elementos *wearables* se replanteará la conectividad Bluetooth haciendo que el elemento sensor actúe de maestro y los *wearables* de esclavos, de esta forma un número variable de *wearables* podrán conectarse al mismo maestro.
#4.2	La interfaz de configuración pasa de un potenciómetro a modo de dial para establecer el valor de referencia a unos botones. Así mismo se mejora la usabilidad explotando la integración entre la pantalla y los botones que ofrece el encapsulado M5Stack.
#4.3	La interfaz de usuario construida con un LED y un botón se sustituye por una pantalla con botones para interactuar con el elemento *wearable*.
#5.2	La gestión de errores en las comunicaciones WiFi se resuelve al reemplazar la conexión WiFi mediante comandos AT por el uso de librerías que da un control mayor a la vez que más sencillo de la conexión WiFi y, especialmente, del cliente HTTP.
#5.3	El posible reemplazo de Arduino por ESP8266 se sustituye por el uso de ESP32 y bajo el empaquetado de M5Stack.

El elemento sensor de las fases anteriores basado en Arduino y otros componentes se reemplaza por un M5Stack que incorpora todos los componentes necesarios salvo los sensores físicos.

La lógica del *sketch* que ejecuta el elemento sensor, cuyo código se incluye en el epígrafe 8.11 del anexo, se describe en el siguiente diagrama.

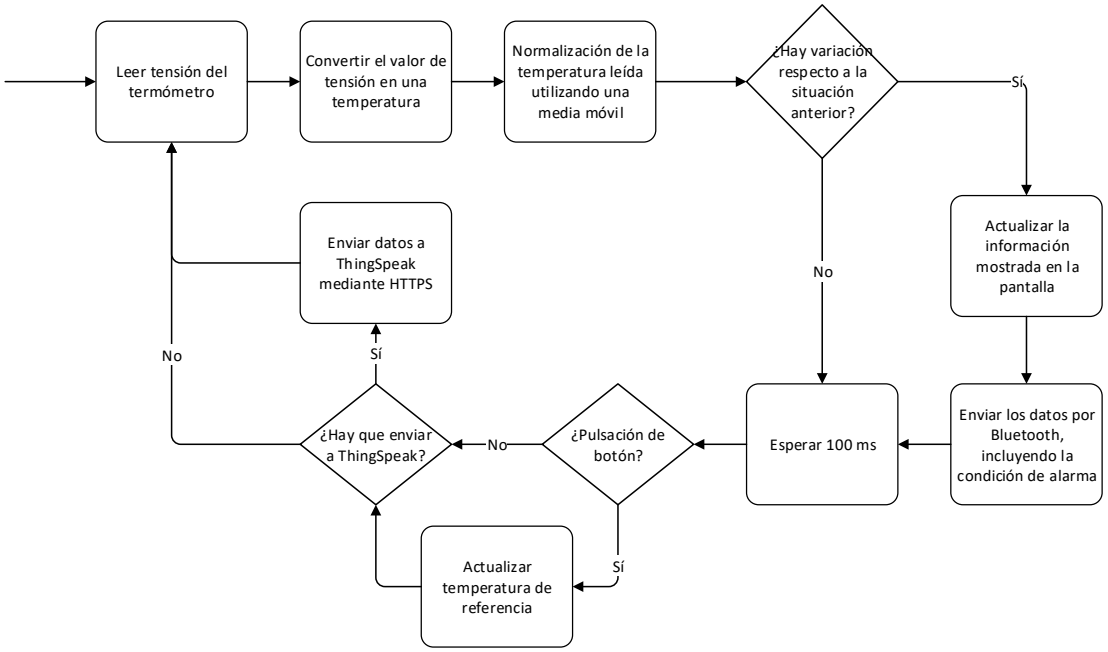

Ilustración 22

La diferencia principal en la lógica del *sketch* respecto a la fase anterior es que al utilizar los botones del M5Stack como parte de la interfaz para el usuario, se prescinde de la lectura de la temperatura de referencia desde una tensión que requería normalización mediante una media móvil. En su lugar se incorpora la lógica para gestionar la pulsación de los botones.

Sin embargo, pese a que la lógica del *sketch* apenas ha cambiado, sí se ha realizado cambios significativos en el código:

- Respecto a la lectura de la temperatura, las lecturas analógicas con ESP32 no requieren transformación ya que el valor obtenido es directamente la tensión.
- No se utilizan puertos serie *software* para la conexión WiFi ni Bluetooth. En su lugar se utilizan librerías disponibles que aportan funciones y objetos que abstraen enormemente la complejidad de las comunicaciones inalámbricas.
- Se configura la conectividad Bluetooth para que actúe como maestro que publica un servicio con una característica que se puede leer y que los cambios en ella son notificados a los esclavos conectados. Por sencillez se ha utilizado una única característica que contiene el dato serializado que ya se manejaba en la fase anterior.

Desde el punto de vista de los componentes hardware hay muchos cambios porque prácticamente todos desaparecen, pero se mantiene el termómetro como sensor de temperatura. No obstante, debe tenerse en cuenta que:

- Las pines analógicos de entrada no soportan más de 3,3 V, por lo que el termómetro se alimenta con esa tensión y no con 5 V como en fases anteriores.
- Esta reducción de la tensión máxima no es una merma en la precisión ya que el conversor analógico-digital del ESP32 tiene 12 bits de resolución en lugar de los 10 que ofrece Arduino UNO.
- El conversor analógico-digital es muy sensible y, para evitar ruido, se conectó un condensador de 10 nF que estabiliza las medidas.

En la Ilustración 23 se aprecia el resultado con la pantalla que muestra la temperatura actual, el valor de referencia y los botones para subir y bajar el valor de la temperatura de referencia. El único elemento *hardware* ajeno al empaquetado M5Stack es el sensor (junto con un condensador para estabilizar las medidas) y el cableado necesario.

El elemento *wearable* de las fases anteriores basado en Arduino y otros componentes se reemplaza por un M5Stack que incorpora todos los componentes necesarios.

En la Ilustración 24 se aprecia el elemento sensor y el elemento *wearable*, cada uno mostrando información distinta en pantalla. El sensor muestra la temperatura medida y la de referencia que provocan una condición de error. El

wearable muestra la temperatura medida, la condición de error y emitirá un sonido hasta que la alarma cese o se reconozca pulsando el botón.

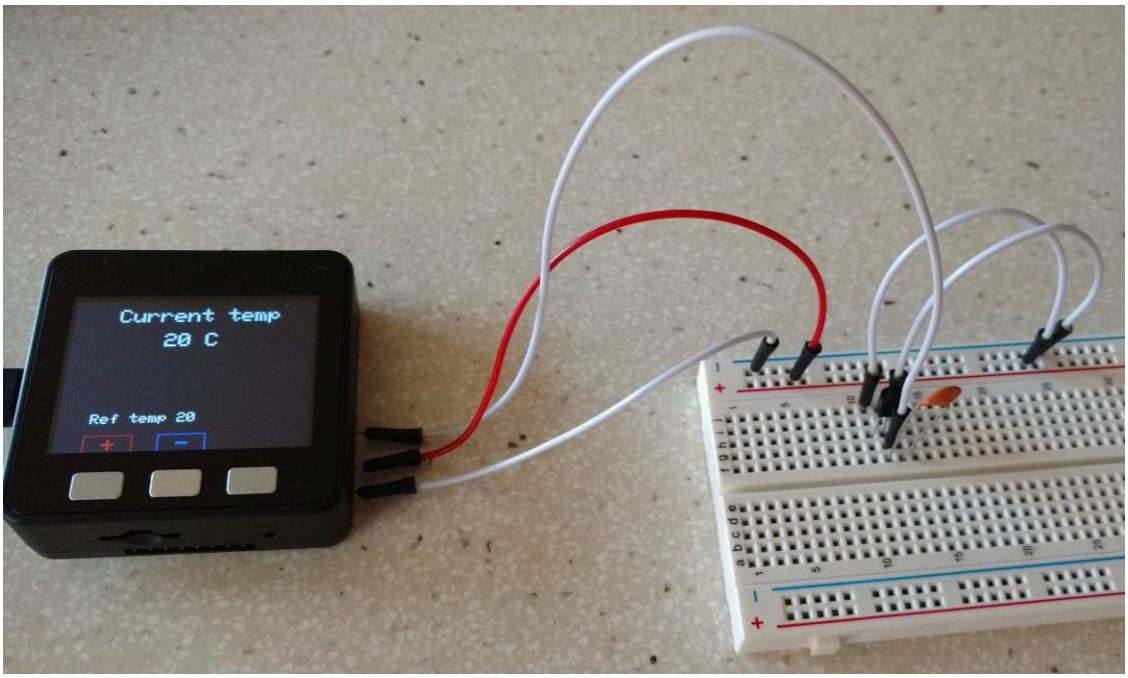

Ilustración 23

Ilustración 24

La lógica del *sketch* que ejecuta el elemento *wearable*, cuyo código se incluye en el epígrafe del anexo, se describe en el diagrama que se muestra a continuación.

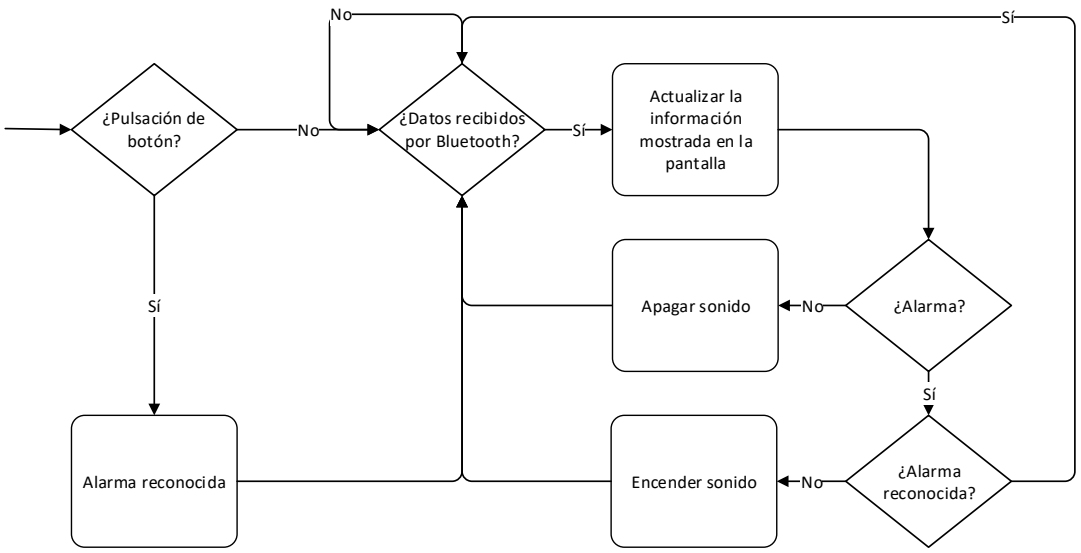

Ilustración 25

La diferencia principal en la lógica del *sketch* respecto a la fase anterior es que el reconocimiento de la alarma se hace nuevamente como parte del bucle principal en lugar de utilizar interrupciones.

Otros cambios significativos son la inclusión de la pantalla en la interfaz de usuario y que no se utilizan puertos serie *software* para la conexión *Bluetooth*. En su lugar se utilizan librerías disponibles que aportan funciones y objetos que abstraen enormemente la complejidad de las comunicaciones inalámbricas. También hay que recordar que el *wearable* configura su conectividad Bluetooth como esclavo para conectar al elemento sensor que actúa como maestro.

En este punto de la construcción del prototipo entregable final quedan varias cuestiones por tratar, como son:

#6.1	Disponer de un empaquetado para el elemento sensor apto para su uso real o, al menos, en los ensayos en campo.
#6.2	Explorar el uso de interrupciones para gestionar la interfaz de usuario, especialmente para reaccionar ante la pulsación de botones.
#6.3	Explorar el uso del concepto de característica como forma de que el elemento sensor publique distintas informaciones (por ejemplo de distintos sensores físicos) en lugar de un único chorro de datos serializados. Esto simplificaría la lectura de información e incluso que los *wearables* pudiesen configurarse para suscribirse solo a algunas características.
#6.4	Incorporar un control de errores robusto para la comunicación inalámbrica entre el elemento sensor y el Smart EPI *wearable*, que garantice que pérdidas puntuales de conectividad son detectadas y gestionadas.

Las demás cuestiones pendientes de las fases previas, junto con las identificadas en esta fase, se analizarán posteriormente como tareas que se podrían abordar en una continuación de este trabajo.

5 De prototipo a producto

5.1 Metodología para la validación como producto comercializable

Para evolucionar el prototipo obtenido hasta alcanzar un producto comercializable se sugiere utilizar el enfoque *lean startup* [41], que presupone que los clientes solo pagan por aquello que aporta valor y que son los propios clientes los que saben si el producto o servicio les aporta valor. Por tanto, este enfoque defiende que cuanto primero se muestre a los clientes cuál es la propuesta del producto o servicio a comercializar, primero se sabrá si les entrega un valor por el que estén dispuestos a pagar.

Para poder ofrecer un producto o servicio a los clientes lo antes posible se maneja el concepto de "producto mínimo viable" (en inglés *Minimum Viable Product*, más conocido por su acrónimo MVP). El enfoque *lean* sugiere que se puede enunciar el producto mínimo pero será el mercado, los clientes, quienes digan si es viable. Por tanto, debe entregarse un MVP tan pronto como sea posible, recoger *feedback* de los clientes, modificar el producto y hacer una nueva iteración del proceso. Este ciclo se repetirá hasta que el producto aporte el valor por el que los clientes estén dispuestos a pagar.

La realización de un conjunto de experimentos con casos reales cubre dos propósitos:

1. Validar técnicamente la versión más avanzada del sistema (en este caso en la fase 6) y obtener información sobre las modificaciones que serían necesarias.
2. Validar el aporte de valor a los clientes y capturar nuevos requisitos que no se habían contemplado inicialmente o bien modificar los establecidos como punto de partida cuando se hayan demostrado erróneos.

El primero de los propósitos tiene interés desde el punto de vista de la ingeniería, pero el segundo lo tiene desde el punto de vista de la comercialización.

Considerando lo anterior, se propone el siguiente plan de experimentación (que sería una continuación del proyecto desde el punto actual):

1. Identificar un conjunto de empresas que deseen participar en los experimentos. Las pequeñas empresas pueden ser más accesibles debido a la cercanía pero las empresas más grandes son las que más recursos dedican a innovación, por lo que debería tratar de involucrar a grandes empresas. Para ello se puede utilizar convocatorias de proyectos de innovación que publican directamente o a través de fundaciones o, incluso, en colaboración con universidades.
2. Para cada empresa dispuesta a participar, proponer el ensayo sobre al menos tres casos de uso real distintos. De esta forma, si se involucra a cinco empresas, se realizarán 15 experimentos.

3. Para cada experimento, asumir que son necesarias tres iteraciones para ajustar la solución. Esto implica 45 ensayos.
4. Realización de un informe final con el resultado para presentar a las empresas participantes.

En este punto, antes de alcanzar unas conclusiones sobre la viabilidad comercial, se recopilan los costes de la construcción del prototipo y se estiman los costes de la realización de los experimentos.

5.2 Costes de la construcción del prototipo

A continuación se detallan los costes de los materiales empleados en la construcción de los dispositivos de cada fase. Los costes del cableado, resistencias, LED, pulsadores, etc. no se incluyen por ser despreciables frente a los costes de los demás componentes utilizados.

Fase	Costes			
1	Elemento	Cantidad	Precio (sin IVA) por unidad	Importe (sin IVA)
	Arduino UNO	1	19,95 €	19,95 €
	Potenciómetro	1	1,40 €	1,40 €
	Pantalla OLED SH1106 1,3"	1	7,50 €	7,50 €
	Proboard	1	1,29 €	1,29 €
			Importe TOTAL (sin IVA)	**30,14 €**
2	Elemento	Cantidad	Precio (sin IVA) por unidad	Importe (sin IVA)
	Arduino UNO	1	19,95 €	19,95 €
	Sensor TMP36	1	1,90 €	1,90 €
	Pantalla OLED SH1106 1,3"	1	7,50 €	7,50 €
	Proboard	1	1,29 €	1,29 €
			Importe TOTAL (sin IVA)	**30,64 €**
3	Elemento	Cantidad	Precio (sin IVA) por unidad	Importe (sin IVA)
	Arduino UNO	2	19,95 €	39,90 €
	Sensor TMP36	1	1,90 €	1,90 €
	Módulo HC-05	2	5,20 €	10,40 €
	Pantalla OLED SH1106 1,3"	1	7,50 €	7,50 €
	Proboard	2	1,29 €	2,58 €
			Importe TOTAL (sin IVA)	**62,28 €**
4	Elemento	Cantidad	Precio (sin IVA) por unidad	Importe (sin IVA)
	Arduino UNO	2	19,95 €	39,90 €
	Sensor TMP36	1	1,90 €	1,90 €
	Zumbador	1	0,95 €	0,95 €
	Potenciómetro	1	1,40 €	1,40 €
	Módulo HC-05	2	5,20 €	10,40 €
	Pantalla OLED SH1106 1,3"	1	7,50 €	7,50 €
	Proboard	2	1,29 €	2,58 €
			Importe TOTAL (sin IVA)	**64,63 €**

5 (configuración del ESP8266)	Elemento	Cantidad	Precio (sin IVA) por unidad	Importe (sin IVA)
	Mini breadboard	1	1,90 €	1,90 €
	Adaptador USB -> TTL	1	8,26 €	8,26 €
			Importe TOTAL (sin IVA)	**10,16 €**
5	Elemento	Cantidad	Precio (sin IVA) por unidad	Importe (sin IVA)
	Arduino UNO	2	19,95 €	39,90 €
	Sensor TMP36	1	1,90 €	1,90 €
	Zumbador	1	0,95 €	0,95 €
	Potenciómetro	1	1,40 €	1,40 €
	Módulo HC-05	2	5,20 €	10,40 €
	Pantalla OLED SH1106 1,3"	1	7,50 €	7,50 €
	Mini breadboard	1	1,90 €	1,90 €
	Conversor 5 V -> 3,3 V	1	4,95 €	4,95 €
	Adaptador USB -> TTL	1	8,25 €	8,25 €
	Módulo WiFi ESP8266	1	7,99 €	7,99 €
	Proboard	2	1,29 €	2,58 €
			Importe TOTAL (sin IVA)	**87,72 €**
6	Elemento	Cantidad	Precio (sin IVA) por unidad	Importe (sin IVA)
	M5 Stack	2	24,95 €	49,90 €
	Sensor TMP36	1	1,90 €	1,90 €
	Mini breadboard	1	1,90 €	1,90 €
			Importe TOTAL (sin IVA)	**53,70 €**

Además de los costes de los materiales es necesario considerar el tiempo empleado en la construcción de los distintos prototipos, que podría convertirse en un importe económico asignado un precio por hora, tal y como se muestra a continuación.

Elemento	Cantidad	Precio (sin IVA) por unidad	Importe (sin IVA)
Horas experimentación	43	40,00 €	1.720,00 €
Horas construcción	23	40,00 €	920,00 €
		Importe TOTAL (sin IVA)	**2.640,00 €**

Si se comparan las cifras anteriores es obvio que el coste de los materiales es despreciable frente al coste de las horas dedicadas. No obstante, considerando tanto los materiales como el tiempo dedicado a la construcción de los prototipos, la cifra para alcanzar este punto del proyecto (sin considerar el tiempo dedicado a la elaboración de la memoria, la presentación y la defensa), es de **aproximadamente 2.900€**.

5.3 Costes de la realización de los experimentos

A continuación se presenta una estimación de los costes del plan de experimentación asumiendo un coste de personal de 40€/h (el mismo aplicado a las horas de construcción del prototipo).

Concepto	Cantidad	Coste por unidad	Coste total
Búsqueda de las cinco empresas colaboradoras (horas)	160	30,00 €	4.800,00 €
Visita presencial a las empresas (vuelo + taxi + comida)	5	185,00 €	925,00 €
1ª ronda 15 ensayos (horas)	600	30,00 €	18.000,00 €
2ª ronda 15 ensayos (horas)	450	30,00 €	13.500,00 €
3ª ronda 15 ensayos (horas)	300	30,00 €	9.000,00 €
Visita presencial a las empresas (vuelo + taxi + comida)	5	185,00 €	925,00 €
		Importe TOTAL	**47.150,00 €**

Los costes que la participación en los experimentos provocarían en las empresas involucradas no se han considerado que se supone que los asumen como parte de su compromiso con la innovación (y la imagen positiva frente al mercado y la sociedad que eso significa) o porque se consiga algún tipo de acuerdo de colaboración si el sistema final resulta comercializable (por ejemplo un parte de los beneficios).

5.4 Conclusiones sobre la viabilidad económica

Para poder disponer de un producto que comercializar primero debe validarse el interés del mercado y su capacidad para satisfacer una necesidad. Por tanto, el primer paso es disponer de la financiación para completar la fase de experimentación.

Tras esa fase de experimentación se abrirían tres opciones:

1. Si todo fuese exitoso, se dispondría de un prototipo evolucionado del producto definitivo que permitiría hacer un diseño comercial y calcular los costes de fabricación.
2. Si los experimentos y los recursos asociados a esa fase son insuficientes para disponer del prototipo definitivo pero la aplicación real se confirma entonces se habrán recopilado los requisitos que todavía faltarían por incorporar y debería hacerse otro ciclo de experimentación (que requeriría nueva financiación).
3. Si los experimentos evidencian la no aplicación real del prototipo porque pese a lo previsto no satisface una necesidad real entonces deberá descartarse la comercialización del producto.

La metodología *lean startup* sugiere que la innovación es una iteración continua para adecuar tu producto o servicio al mercado hasta que te quedas sin presupuesto para evolucionarlo más. Si en ese punto se puede comercializar entonces se confirma la oportunidad de negocio pero si no se puede entonces al menos la innovación no será un pozo sin fondo de recursos ya que se ha establecido, a priori, un punto en el que se deja de iterar.

El estudio de la viabilidad económica tiene un último escenario que es la comercialización real del producto si se confirma que el prototipo es adecuado. En ese caso el coste de la fase 6 ofrece un valor de 25€ para el prototipo del elemento sensor y del elemento *wearable*. Considerando que los prototipos siempre son más caros que los elementos construidos en tiradas de producción para comercializar, se puede utilizar la cifra de 25€ como una cota superior del coste del producto.

Este importe es similar al de un par de botas de seguridad de las más sencillas utilizadas en entornos laborales peligrosos y que son reemplazadas anualmente [42]. Por lo que si el precio del EPI inteligente definitivo es inferior a esa cifra, es razonable pensar que el producto puede comercializarse a un coste que resulte interesante para las empresas que buscan la excelencia en seguridad y que van más allá del mero cumplimiento legal.

6 Conclusiones

6.1 Lecciones aprendidas

La realización de este proyecto ha proporcionado conocimiento y experiencia en el manejo de Arduino y la tecnología relacionada. Como consecuencia se han identificado cuatro lecciones aprendidas que se consideran especialmente valiosas y que se resumen a continuación.

Las características del *hardware* deben ser tenidas especialmente en cuenta o, en caso contrario, además de que un diseño teóricamente correcto no funcione en el mundo real, es posible que incluso se dañen los componentes utilizados.

En el mundo del desarrollo de *software* las características del *hardware* subyacente son muy lejanas y están ocultas por múltiples capas de abstracción que simplifican su uso. Sin embargo, en los proyectos como este en los que el *hardware* es un elemento clave, los detalles exactos del *hardware* y sus características son especialmente relevantes.

Por ejemplo, la alimentación de la solución para disponer de conexión WiFi requiere una tensión que la placa Arduino puede ofrecer, sin embargo la corriente que la placa Arduino ofrece es insuficiente.

El código debe construirse pensando exclusivamente en el uso que se le va a dar en lugar de dotarlo de funcionalidades no necesarias pensando en una posible reutilización posterior.

En el mundo del desarrollo de *software* el *hardware* subyacente suele estar sobredimensionado. Esto es debido a que el coste del *hardware* suele ser inferior al coste del desarrollo del *software* y, por tanto, es preferible simplificar el desarrollo del *software* ya que el coste global del proyecto se reduce. Sin embargo, en los proyectos como este donde el *hardware* tiene restricciones muy específicas (especialmente en cuanto a tamaño y consumo de energía) no es posible contar con un sobredimensionamiento de la capacidad de cómputo, por lo que el código debe ser más eficiente aunque sea a costa de sacrificar modularidad.

Por ejemplo, en el primer experimento de uso de la pantalla OLED monocroma, se programó una función que escribía líneas de longitud arbitraria porque gestionaba el posicionamiento del cursor troceando las líneas aprovechando los espacios en blanco entre las palabras y maximizando el texto en cada línea de la pantalla OLED. Pero al ejecutar este código el resultado no fue el esperado porque, pese a que la depuración del código ofrecía los resultados de posicionamiento del texto esperados, la velocidad de procesamiento de Arduino y la de refresco de la pantalla OLED eran inferiores a lo que el algoritmo requería.

La calidad de los materiales simplifica el trabajo porque ofrece seguridad en los resultados.

Esta lección se hizo evidente durante la construcción de la segunda etapa del prototipo. Inicialmente el dato de la temperatura era incorrecto y el foco para corregirlo se puso en el código para convertir la tensión en temperatura. Sin embargo, tras revisar el código y las hojas de especificaciones del sensor repetidas veces en busca de un error, no quedó más remedio que revisar el circuito hasta que se detectó -usando un polímetro- que la conexión del sensor a tierra era incorrecta porque el cable estaba estropeado y ese pin no estaba a 0 V.

La experimentación es necesaria para contrastar el comportamiento real frente al teórico de los componentes y materiales utilizados.

Los componentes para uso no profesional no tienen el nivel de calidad al que se está acostumbrado al trabajar con productos comerciales. La documentación no siempre está disponible y actualizada e incluso el funcionamiento puede tener comportamientos no previstos.

Por ejemplo, el elemento sensor construido con M5Stack emite un pequeño sonido que no es molesto pero se escucha perfectamente en ambientes silenciosos y resulta estar provocado por interferencias entre el altavoz y las lecturas analógicas.

6.2 Líneas de trabajo futuras

Las cuestiones pendientes, que recopilamos a continuación, son posibles líneas de trabajo futuro que podrían usar este trabajo como punto de partida.

#3.2	Establecer un protocolo más robusto de comunicación entre el sensor y los *wearables* para que se realice comunicación unidireccional sensor-*wearable* gestionando la serialización de los datos enviados.
#3.3	Evolucionar aún más el protocolo para gestionar comunicaciones bidireccionales y que los *wearables* puedan responder al sensor, lo que permitiría añadir mayor funcionalidad.
#4.1	Modificar el elemento sensor para incorporar varios sensores distintos adecuados al escenario del caso de uso real.
#5.1	Dotar de una interfaz de usuario de configuración adecuada para establecer la conexión WiFi y los demás parámetros de funcionamiento.
#5.4	Explorar opciones de explotación de datos más avanzadas usando las funcionalidades propias de ThingSpeak o mediante la utilización de otras herramientas que interactúen con los datos almacenados en ThingSpeak.
#6.1	Disponer de un empaquetado para el elemento sensor apto para su uso real o, al menos, en los ensayos en campo.
#6.2	Explorar el uso de interrupciones para gestionar la interfaz de usuario, especialmente para reaccionar ante la pulsación de botones.

#6.3	Explorar el uso del concepto de característica como forma de que el elemento sensor publique distintas informaciones (por ejemplo de distintos sensores físicos) en lugar de un único chorro de datos serializados. Esto simplificaría la lectura de información e incluso que los *wearables* pudiesen configurarse para suscribirse solo a algunas características.
#6.4	Incorporar un control de errores robusto para la comunicación inalámbrica entre el elemento sensor y el Smart EPI *wearable*, que garantice que pérdidas puntuales de conectividad son detectadas y gestionadas.

Para la cuestión pendiente #5.1 relativa a la incorporación de una interfaz de configuración, es posible dar una recomendación de cómo abordarla.

Partiendo del hecho de que ESP32 puede actuar como *host* que se conecta a una red WiFi existente pero también puede actuar como un punto de acceso que levanta una red WiFi a la que pueden conectarse otros dispositivos, la solución para ofrecer una interfaz de configuración consiste en:

1. En condiciones normales, el ESP32 que está dentro del M5Stack arrancará en el modo normal de funcionamiento utilizando la configuración que se habrá guardado en un almacén no volátil (por ejemplo una tarjeta de memoria compatible con el zócalo que incluye).
2. Una combinación de botones, por ejemplo mantener pulsado un botón durante más de dos segundos, hará que reinicie en lo que se podría denominar “modo configuración”.
3. Al iniciar en “modo configuración” se levantará una red WiFi a la que se podrá conectar otro dispositivo (por ejemplo un *Smartphone*) y ejecutará un servidor web que ofrecerá una página con el formulario de configuración y la posibilidad de guardar esa nueva configuración. Cuando se guardase la configuración en el almacén no volátil, el ESP32 arrancaría de nuevo pero ahora en el modo normal de operación.

Esta solución propuesta es técnicamente factible y no requiere el desarrollo de ningún otro elemento, ya que el navegador *web* de un *Smartphone* evita la necesidad de, por ejemplo, tener que desarrollar una aplicación para dispositivos móviles.

Además, si los casos de uso reales –que se evidenciarán durante la realización de los experimentos– confirman que las organizaciones interesadas en un sistema como este ya disponen de sensores desplegados que envían datos al centro de control (es decir, son organizaciones con un grado de madurez relevante en el uso de IoT), entonces se podría utilizar una versión distinta de la arquitectura de la solución. Tal y como se muestra en la Ilustración 26, si los sensores están desplegados y recogen y envían esa información en tiempo real al centro de control utilizando conectividad inalámbrica, por ejemplo WiFi, la misma conectividad podría ser usada para enviar información a los *Smart EPI wearables*.

En esta situación, respecto a la arquitectura original, se elimina el elemento sensor ya que se aprovechan los sensores ya desplegados y, por tanto, las cuestiones pendientes #3.2, #4.1, #6.1, #6.3 y #6.4 ya no aplican y pueden ignorarse.

Además, en ese escenario la cuestión pendiente #5.4 integraría la #3.3 y ambas podrían solventarse si el sistema de control de operaciones ofrece una interfaz (API, *application programming interface*) para interactuar con él. Esta interfaz podría construirse como un conjunto de servicios web que el *Smart EPI wearable* utilizaría de forma bidireccional.

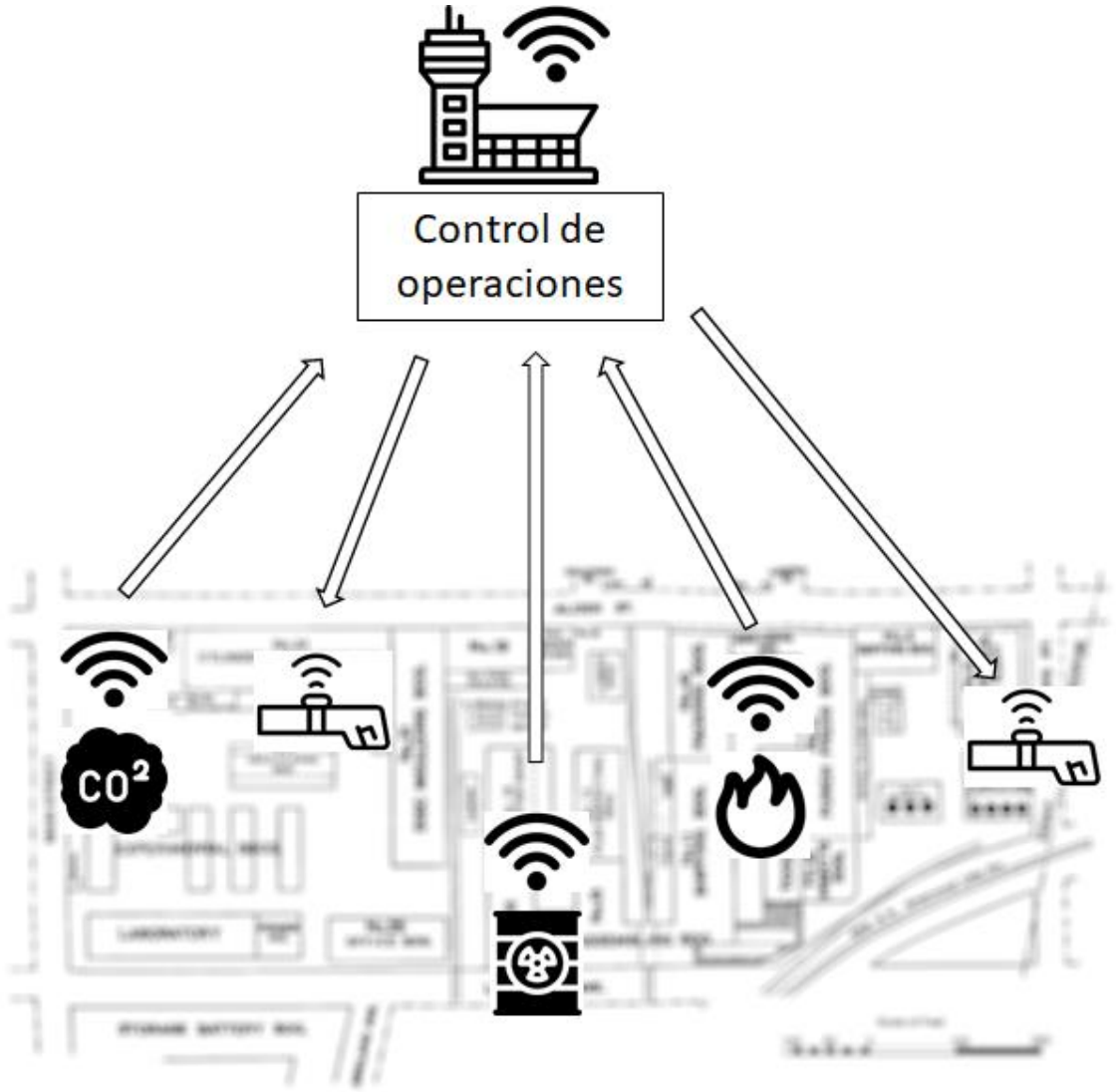

Ilustración 26

6.3 Evaluación del cumplimiento de los objetivos

Respecto a los objetivos establecidos en el epígrafe 1.5, podemos considerar que se ha conseguido un prototipo de *Smart EPI* que los trabajadores podrían transportar y utilizar sin interferir en su trabajo. No obstante, si bien el *Smart EPI* tiene dimensiones y peso reducido pudiendo llevarse, por ejemplo, en el bolsillo de un pantalón o de una chaqueta, no es realmente un *wearable*.

Este objetivo puede considerarse sólo parcialmente alcanzado pero debe tenerse en cuenta que durante la construcción del prototipo se descartó el uso de un *hardware* más adecuado para ser utilizado como *wearable* para reducir los costes de construcción del prototipo en el ámbito de un proyecto académico. En el caso de avanzar hacia la comercialización e iniciar el plan de ensayos propuesto, debería revisarse si el *Smart EPI* construido usando M5Stack es adecuado o bien si hay que transformarlo en un auténtico *wearable* utilizando otra tecnología identificada en el estudio del estado del arte.

El requisito de que el dispositivo pudiese manejar uno o varios sensores se considera cubierto si bien en el prototipo sólo se utilizó un sensor por simplicidad y ahorro de costes, ya que las conclusiones alcanzadas son generalizables si se manejan varios sensores simultáneamente. Solo en el caso de pretender manejar un número muy alto de sensores que ofrezcan múltiples datos que hubiese que gestionar debería analizarse si la solución basada en M5Stack sigue siendo la más adecuada o podría interesar retomar la implementación del elemento sensor utilizando una placa Arduino, aunque en este caso sin duda no sería Arduino UNO y sería mejor utilizar una versión que facilite la conectividad inalámbrica.

Respecto a los requisitos de conectividad inalámbrica, se ha logrado conectar en tiempo real el elemento sensor con un número variable de *Smart EPI*. Considerando el uso de *Bluetooth Low Energy* (BLE) que incorporan los ESP32 de los dispositivos M5Stack, ocho sería el número máximo de trabajadores cuyos *Smart EPI* podrían estar conectados al mismo elemento sensor. Esta cifra parece adecuada si bien debería validarse durante el plan de experimentos para la comercialización. Este detalle, así como que el alcance efectivo de la tecnología *Bluetooth* sea suficiente, es uno de los puntos a validar durante los ensayos en campo. No obstante, en durante la construcción del prototipo se comprobó que la conectividad Bluetooth supera los cinco metros en el interior de una vivienda.

También la conectividad inalámbrica entre el elemento sensor y el centro de control, representado en este caso por la plataforma ThingSpeak, se ha conseguido exitosamente con un envío periódico con frecuencia inferior a 10 minutos (durante la construcción del prototipo se probó con una frecuencia de envío de un minuto).

Además de los objetivos técnicos, se requería que el coste de los *Smart EPI wearables* fuese suficientemente reducido como para ofrecer un valor superior a su coste. Tras el análisis realizado en el epígrafe 5 sobre cómo pasar de un prototipo a un producto comercializable, se considera que este objetivo también se puede considerar cumplido.

7 Bibliografía

[1] Real Decreto 773/1997, de 30 de mayo, sobre disposiciones mínimas de seguridad y salud relativas a la utilización por los trabajadores de equipos de protección individual. «BOE» núm. 140, de 12 de junio de 1997.

[2] Catálogo de productos IRISTICK. Recuperado el 3 de junio de 2019 de https://iristick.com/products.

[3] Catálogo de productos DAQRI. Recuperado el 3 de junio de 2019 de https://daqri.com/products.

[4] Business Wire (2017). *Top 10 Vendors in the Smart Helmets Market from 2017 to 2021*. Recuperado el 3 de junio de 2019 de https://www.businesswire.com/news/home/20171005005900/en/Top-10-Vendors-Smart-Helmets-Market-2017.

[5] Catálogo de productos ZHORTECH. Recuperado el 3 de junio de 2019 de https://zhortech.com/safety.

[6] Catálogo de productos INTELLINIUM. Recuperado el 3 de junio de 2019 de https://intellinium.io/smart-connected-safety-shoes.

[7] Forbes (2018). *What is Industry 4.0? Here's A Super Easy Explanation For Anyone*. Recuperado el 3 de junio de 2019 de https://www.forbes.com/sites/bernardmarr/2018/09/02/what-is-industry-4-0-heres-a-super-easy-explanation-for-anyone.

[8] Deloitte (2015). *Industry 4.0: Challenges and solutions for the digital transformation and user of exponential technologies.* Recuperado el 3 de junio de 2019 de https://www2.deloitte.com/content/dam/Deloitte/ch/Documents/manufacturing/ch-en-manufacturing-industry-4-0-24102014.pdf.

[9] European Agency for Safety and Health at Work (2017). *Work-related accidents and injuries cost EU €476 billion a year according to new global estimates*. Recuperado el 3 de junio de 2019 de https://osha.europa.eu/en/about-eu-osha/press-room/eu-osha-presents-new-figures-costs-poor-workplace-safety-and-health-world.

[10] Occupational Safety and Health Administration (sin fecha). *Business Case for Safety and Health*. Recuperado el 3 de junio de 2019 de https://www.osha.gov/dcsp/products/topics/businesscase/costs.html.

[11] EHS Today (2017). *PPE and the Internet of Things*. Recuperado el 3 de junio de 2019 de https://www.ehstoday.com/eye-face-head/ppe-and-internet-things.

[12] Catálogo de productos NOKE. Recuperado el 3 de junio de 2019 de https://noke.com/products.

[13] Dupont Sustainable Solutions (sin fecha). *#1 in Brand Awareness and Preference Among EHS Consulting Companies*. Recuperado el 3 de junio de 2019 de http://www.dupont.com/products-and-services/consulting-services-process-technologies/articles/ehs-leaders-survey-verdantix.html.

[14] Verdantix (2019). *Green Quadrant EHS Software*.

[15] Cision (2018). *DuPont Sustainable Solutions Announces Partnership with Guardhat to Advance Digital and Data Analytics Strategy*.

Recuperado el 3 de junio de 2019 de https://www.prweb.com/releases/dupont_sustainable_solutions_announces_partnership_with_guardhat_to_advance_digital_and_data_analytics_strategy/prweb15865132.htm.

[16] Hoja de producto de GRAY WOLF DIRECT SENSE II (2018). Recuperado el 3 de junio de 2019 de https://graywolfsensing.com/wp-content/pdf/GrayWolf-DirectSenseII-Brochure-Nov18.pdf.

[17] Catálogo de productos GRAY WOLF en MIP. Recuperado el 3 de junio de 2019 de http://www.mip.fi/en/tuotteet/verkkokauppa/valmistajat/manufacturer/graywolf.

[18] Hoja de producto de ESPRESSIF ESP32 SoC (sin fecha). Recuperado el 3 de junio de 2019 de https://www.espressif.com/en/products/hardware/esp32/overview.

[19] ESP32 (2019). En Wikipedia. Recuperado el 3 de junio de 2019 de https://en.wikipedia.org/wiki/ESP32.

[20] MCI Electronics (sin fecha). *¿Qué es Arduino?* Recuperado el 3 de junio de 2019 de https://arduino.cl/que-es-arduino.

[21] Arduino (2019). En Wikipedia. Recuperado el 3 de junio de 2019 de https://es.wikipedia.org/wiki/Arduino.

[22] Tojeiro Calaz, G., (2014). *Taller de Arduino: Un enfoque práctico para principiantes*. Editorial MARCOMBO.

[23] How to Mechatronics (sin fecha). *20 ARDUINO PROJECTS WITH DIY INSTRUCTIONS*. Recuperado el 3 de junio de 2019 de https://howtomechatronics.com/arduino-projects.

[24] Catálogo de proyectos en la plataforma Arduino Create (sin fecha). Recuperado el 3 de junio de 2019 de https://create.arduino.cc/projecthub.

[25] Catálogo de proyectos Arduino en la plataforma Instructables (sin fecha). Recuperado el 3 de junio de 2019 de https://www.instructables.com/id/Arduino-Projects.

[26] Catálogo de proyectos Arduino en la plataforma Maker Pro (sin fecha). Recuperado el 3 de junio de 2019 de https://maker.pro/arduino/projects.

[27] Catálogo de productos Arduino en Bricogeek (sin fecha). Recuperado el 3 de junio de 2019 de https://tienda.bricogeek.com/51-arduino-original.

[28] Catálogo de *shields* Arduino en Bricogeek (sin fecha). Recuperado el 3 de junio de 2019 de https://tienda.bricogeek.com/53-shields-arduino.

[29] Librería U8g2 en GitHub (sin fecha). Recuperado el 3 de junio de 2019 de https://github.com/olikraus/u8g2/wiki.

[30] Catálogo de productos Amazon Web Services (sin fecha). Recuperado el 3 de junio de 2019 de https://aws.amazon.com/es/products.

[31] Catálogo de productos en Arduino Store (sin fecha). Recuperado el 3 de junio de 2019 de https://store.arduino.cc/components/components-sensors.

[32] Luis Llamas, Ingeniería Informática y Diseño (2017). *CONECTAR ARDUINO POR WIFI CON EL MÓDULO ESP8266 ESP01*. Recuperado el 3 de junio de 2019 de https://www.luisllamas.es/arduino-wifi-esp8266-esp01.

[33] Hoja de producto de ESPRESSIF 8266 SoC (sin fecha). Recuperado el 3 de junio de 2019 de http://www.farnell.com/datasheets/2661567.pdf?_ga=2.66950573.2034435700.1552324093-1610923507.1552324093.

[34] Hoja de producto Módulo Bluetooth HC-05 en Bricogeek (sin fecha). Recuperado el 3 de junio de 2019 de https://tienda.bricogeek.com/modulos-bluetooth/800-modulo-bluetooth-hc-05.html.

[35] Catálogo de productos Xbee (sin fecha). Recuperado el 3 de junio de 2019 de https://www.digi.com/xbee.

[36] Hoja de producto de XBee 2 MW antena de alambre Malla en Amazon (sin fecha). Recuperado el 3 de junio de 2019 de https://www.amazon.es/XBee-antena-alambre-Malla-zigbee/dp/B01FB0O16W/ref=sr_1_1_sspa.

[37] Catálogo de productos ESP32 en ESPRESSIF (sin fecha). Recuperado el 3 de junio de 2019 de https://www.espressif.com/en/products/hardware/modules.

[38] Hoja de producto de TMP35 de Analog Devices (sin fecha). Recuperado el 3 de junio de 2019 de https://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Temp/TMP35_36_37.pdf.

[39] Librería esp8266wifi en Read the Docs (2017). Recuperado el 3 de junio de 2019 de https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html.

[40] Catálogo de productos M5Stack (sin fecha). Recuperado el 3 de junio de 2019 de https://m5stack.com/collections/m5-core.

[41] El blog de retos para ser directivo, EAE Business School (2017). *Aprendizaje, experimentación e iteración, la metodología lean startup*. Recuperado el 3 de junio de 2019 de https://retos-directivos.eae.es/aprendizaje-experimentacion-e-iteracion-la-metodologia-lean-startup.

[42] Catálogo de productos en Vestuario Laboral (sin fecha). Recuperado el 3 de junio de 2019 de https://www.vestuariolaboral.com/botas-de-seguridad-c-371.

8 Anexos

Por simplicidad, en el código de los distintos *sketch* que se muestran en este anexo no se incluyen los comentarios del código ni el código que se repite entre las distintas fases del proyecto. No obstante, el código completo está publicado en GitHub https://github.com/jfernandezpine/TFG_Arduino.

8.1 Sketch fase 1

```
#include <U8g2lib.h>

#define PIN_ANALOGICO_MEDIDA 5
#define DIGITAL_PIN_SCK 2
#define DIGITAL_PIN_SDA 3
#define DIGITAL_PIN_DC 4
#define DIGITAL_PIN_CS 5

U8G2_SH1106_128X64_NONAME_1_4W_SW_SPI u8g2 (U8G2_R0, DIGITAL_PIN_SCK,
DIGITAL_PIN_SDA, DIGITAL_PIN_CS, DIGITAL_PIN_DC);

void setup() {
  pinMode(PIN_ANALOGICO_MEDIDA, INPUT);
  u8g2.setFont(u8g2_font_6x10_mf);
  u8g2.setFontRefHeightExtendedText();
  u8g2.setFontPosTop();
  u8g2.begin();
}

void ShowSensorValue(float value){
  u8g2.firstPage();
  do {

      u8g2.setCursor(0, 0);
      u8g2.print("SENSOR VALUE IS");
      u8g2.setCursor(0, 15);
      u8g2.print(String(value).c_str());
      u8g2.print(" V");
  } while(u8g2.nextPage());
}

void loop() {
  int rawValue = analogRead(PIN_ANALOGICO_MEDIDA);
  float value_mV = map(rawValue, 0, 1023, 0, 5000);
  float value_V = value_mV / 1000.0;
  ShowSensorValue(value_V);
  delay(500);
}
```

8.2 Sketch fase 2

```
#include <U8g2lib.h>

#define PIN_ANALOGICO_MEDIDA 5
#define DIGITAL_PIN_SCK 2
#define DIGITAL_PIN_SDA 3
```

```
#define DIGITAL_PIN_DC 4
#define DIGITAL_PIN_CS 5

#define VS_mV 5000

U8G2_SH1106_128X64_NONAME_1_4W_SW_SPI u8g2(U8G2_R0, DIGITAL_PIN_SCK,
DIGITAL_PIN_SDA, DIGITAL_PIN_CS, DIGITAL_PIN_DC);

void setup() {
  pinMode(PIN_ANALOGICO_MEDIDA, INPUT);
  u8g2.setFont(u8g2_font_6x10_mf);
  u8g2.setFontRefHeightExtendedText();
  u8g2.setFontPosTop();
  u8g2.begin();
}

void ShowSensorValue(int value){
  u8g2.firstPage();
  do {

      u8g2.setCursor(0, 0);
      u8g2.print("TEMPERATURE IS");
      u8g2.setCursor(0, 15);
      u8g2.print(String(value).c_str());
      u8g2.print(" C");
  } while(u8g2.nextPage());
}

float CurrentSensorValue(){
  int rawValue = analogRead(PIN_ANALOGICO_MEDIDA);
  float value_mV = map(rawValue, 0, 1023, 0, VS_mV);
  double value_C = (value_mV - 500) / 10;
  return value_C;
}

int AvgSensorValue(){
  static float avg [3];
  static int n = 0;

  if (n == 0)
  {
    avg[0] = CurrentSensorValue();
    n++;
    return round(avg[0]);
  }

  if (n == 1)
  {
    avg[1] = CurrentSensorValue();
    n++;
    return round((avg[0] + avg[1]) / 2);
  }

  if (n == 2)
  {
    avg[2] = CurrentSensorValue();
    n++;
    return round((avg[0] + avg[1] + avg[2]) / 3);
  }
```

```
  if (n >= 3)
  {
    avg[0] = avg[1];
    avg[1] = avg[2];
    avg[2] = CurrentSensorValue();
    return round((avg[0] + avg[1] + avg[2]) / 3);
  }
}

void loop() {

  ShowSensorValue(AvgSensorValue());
  delay(500);
}
```

8.3 Configuración Bluetooth fase 3

Los comandos AT utilizados para configurar para el módulo Bluetooth HC-05 del elemento sensor son:

AT+NAME=Sensor	Asignación de nombre “Sensor”
AT+PSWD=0000	Asignación de pin “0000”
AT+ROLE=0	Asignación de modo de operación esclavo
AT+UART=9600,0,0	Asignación de velocidad 9600 baudios, 1 bit de parada y sin bit de paridad.
AT+ADDR?	Consulta de dirección (la respuesta fue **+ADDR:18:e4:34cdec**)

Los comandos AT utilizados para configurar para el módulo Bluetooth HC-05 del elemento *wearable* son:

AT+NAME=Wearable1	Asignación de nombre “Wearable1”
AT+PSWD=0000	Asignación de pin “0000”
AT+ROLE=1	Asignación de modo de operación maestro
AT+UART=9600,0,0	Asignación de velocidad 9600 baudios, 1 bit de parada y sin bit de paridad.
AT+CMODE=0	Se restringe la conexión a un dispositivo cuya dirección se indicará explícitamente.
AT+BIND=18,e4,34cdec	Dirección del dispositivo al que se conectará.

La configuración anterior permitirá que el *wearable*, actuando como maestro, se conecte automáticamente al sensor, que actuará como esclavo.

La configuración del módulo Bluetooth HC-05 se realizará mediante el envío de comandos AT por el puerto serie, tal y como se aprecia en la Ilustración 27, utilizando el circuito descrito en la Ilustración 28 y un *sketch* que permite usar el monitor serie del IDE de Arduino para enviar comandos AT al módulo HC-05.

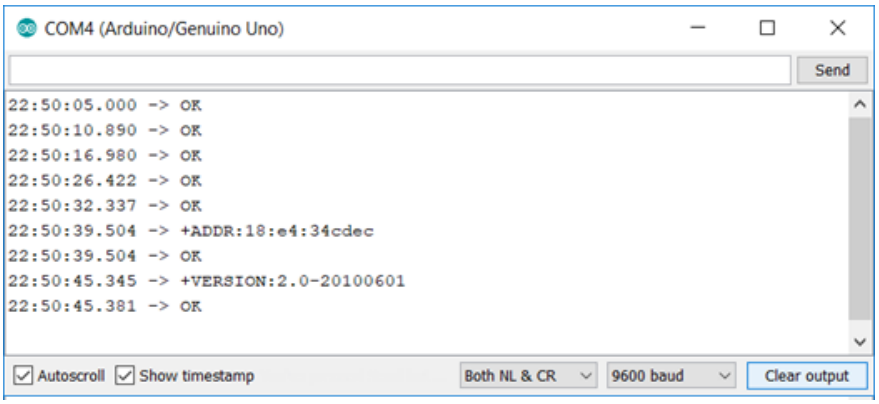

Ilustración 27

```
#define PIN_RX 10
#define PIN_TX 11

#include <SoftwareSerial.h>
SoftwareSerial BT(PIN_RX,PIN_TX);

void setup()
{
  BT.begin(9600);
  Serial.begin(9600);
}

void loop()
{
  if(BT.available())
    Serial.write(BT.read());

  if(Serial.available())
     BT.write(Serial.read());
}
```

Una vez que el *sketch* anterior se transfiere a la placa Arduino UNO, esta se ha conectado al módulo Bluetooth HC-05 y ambos están encendidos, se pulsará el botón que el módulo HC-05 incluye y que entonces pone el módulo en el modo AT 1 para aceptar comandos. En ese momento los comandos se escribirán en el monitor serie del IDE de Arduino y el módulo HC-05 los procesará respondiendo a ellos. Los cambios de configuración quedarán guardados en el módulo aunque se corte la alimentación.

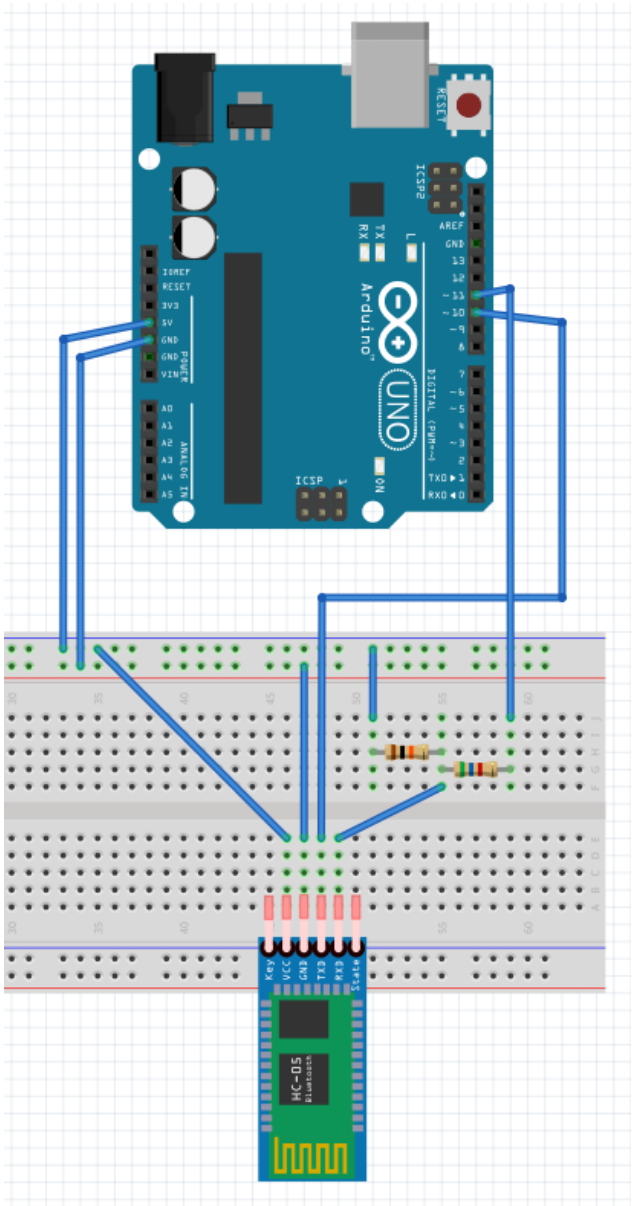

Ilustración 28

1) El módulo HC-05 se alimenta con la tensión de 5 V de la placa Arduino y se conectan dos pines a las conexiones RX y TX. El pin que va a la conexión RX del módulo Bluetooth utiliza un divisor de tensión para que los niveles altos del pin digital Arduino (que son 5 V) se transformen en 3,3 V que es lo que manejan los conectores de datos del módulo Bluetooth.

2) El divisor de tensión se consigue utilizando dos resistencias, una de $5{,}6\ k\Omega$ y otra de $10\ k\Omega$. De esta forma, alimentando el divisor de tensión con los 5 V de la salida del pin digital y conectando la resistencia de $10\ k\Omega$ a masa se entregan al módulo HC-05 $5\ V \cdot 10\ k\Omega/(10\ k\Omega + 5{,}6\ k\Omega) = 3{,}2\ V$.

3) El pin de transmisión del módulo Bluetooth se conecta directamente al pin digital de la placa Arduino porque de ahí se leerán valores que, aunque lleguen a 3,3 V, son interpretados como un nivel alto por la placa Arduino.

4) Es importante resaltar que los pines etiquetados como RX y TX en el módulo Bluetooth HC-05 son configurados como pines de escritura y recepción en el código del *sketch*. Esta inversión de lógica se debe a que cuando Arduino "envía" datos por el puerto serie, el módulo Bluetooth los "recibe". Del mismo modo, cuando el módulo Bluetooth "envía" datos, la placa Arduino los "recibe" por el puerto serie.

8.4 Sketch sensor fase 3

```
#include <U8g2lib.h>
#include <SoftwareSerial.h>

#define PIN_RX 10 // pin Arduino a conectar con el pin TX del HC-05
#define PIN_TX 11 // pin Arduino a conectar con el pin RX del HC-05

#define BAUD 9600

SoftwareSerial BT(PIN_RX,PIN_TX);
```

```
#define PIN_ANALOGICO_MEDIDA 5
#define VS_mV 5000
#define DIGITAL_PIN_SCK 2
#define DIGITAL_PIN_SDA 3
#define DIGITAL_PIN_DC 4
#define DIGITAL_PIN_CS 5

U8G2_SH1106_128X64_NONAME_1_4W_SW_SPI u8g2(U8G2_R0, DIGITAL_PIN_SCK,
DIGITAL_PIN_SDA, DIGITAL_PIN_CS, DIGITAL_PIN_DC);

void setup() {
  pinMode(PIN_ANALOGICO_MEDIDA, INPUT);
  u8g2.setFont(u8g2_font_6x10_mf);
  u8g2.setFontRefHeightExtendedText();
  u8g2.setFontPosTop();
  u8g2.begin();
  BT.begin(BAUD);
}

void loop() {
  int val = AvgSensorValue();
  ShowSensorValue(val);
  String data = String(val);
  data += '\n';
  BT.write(data.c_str());
  delay(500);
}
```

8.5 Sketch wearable fase 3

```
#define PIN_RX 10 // pin Arduino a conectar con el pin TX del HC-05
#define PIN_TX 11 // pin Arduino a conectar con el pin RX del HC-05

#define BAUD 9600

#include <SoftwareSerial.h>
SoftwareSerial BT(PIN_RX,PIN_TX);

String data;

void setup()
{
  BT.begin(BAUD);
}

void loop()
{
  if(BT.available())
  {
    char c = BT.read();
    if (c == '\n')
    {
      int val = atoi(data.c_str());
      data = "";
    }
    else
      data += c;
  }
}
```

8.6 Sketch sensor fase 4

Es importante resaltar que se ha modificado la forma de calcular la media móvil pasando de una implementación para una media móvil de tres valores a una media móvil genérica basada en una abstracción en capas, donde primero se construye un objeto que representa un *buffer* circular y este se usa para construir un objeto que representa medias móviles de un número configurable de elementos.

```
#include <U8g2lib.h>
#include <SoftwareSerial.h>

#include "MovingAverage.h"
#include "SensorData.h"

#define PIN_RX 10 // pin Arduino a conectar con el pin TX del HC-05
#define PIN_TX 11 // pin Arduino a conectar con el pin RX del HC-05

#define BAUD 9600

SoftwareSerial BT(PIN_RX,PIN_TX);

#define PIN_ANALOGICO_MEDIDA 5
#define VS_mV 5000
#define PIN_ANALOGICO_POTENCIOMETRO 4
#define DIGITAL_PIN_SCK 2
#define DIGITAL_PIN_SDA 3
#define DIGITAL_PIN_DC 4
#define DIGITAL_PIN_CS 5

U8G2_SH1106_128X64_NONAME_1_4W_SW_SPI u8g2(U8G2_R0, DIGITAL_PIN_SCK,
DIGITAL_PIN_SDA, DIGITAL_PIN_CS, DIGITAL_PIN_DC);

void setup() {
  pinMode(PIN_ANALOGICO_MEDIDA, INPUT);
  pinMode(PIN_ANALOGICO_POTENCIOMETRO, INPUT);

  u8g2.setFont(u8g2_font_6x10_mf);
  u8g2.setFontRefHeightExtendedText();
  u8g2.setFontPosTop();
  u8g2.begin();

  BT.begin(BAUD);
}

float Volt2Temp(float value_mV) {
  return (value_mV - 500) / 10;
}

float CurrentAnalogValue(int pin) {
  int rawValue = analogRead(pin);
  float value_mV = map(rawValue, 0, 1023, 0, VS_mV);
  float value_C = Volt2Temp(value_mV);
  return value_C;
}

void Display(int valSensor, int valPot, bool alarm) {
```

```
  u8g2.firstPage();
  do {
      u8g2.setCursor(0, 0);
      u8g2.print("TEMP = ");
      u8g2.print(String(valSensor).c_str());
      u8g2.print(" C");
      u8g2.setCursor(0, 15);
      u8g2.print("REF TEMP = ");
      u8g2.print(String(valPot).c_str());
      u8g2.print(" C");
      u8g2.setCursor(0, 30);
      if (alarm)
        u8g2.print("ALARM !!!");
  } while(u8g2.nextPage());
}

String DoSerial (int valSensor, int valPot, bool alarm) {
  String str(valSensor);
  str += "#";
  str += (alarm)? "1" : "0";
  str += '\n';
  return str;
}
void loop() {
  static SensorData sensor;
  static MovingAverage avgSensor(30);
  static MovingAverage avgPot(10);

  int valSensor =
    avgSensor.Add(CurrentAnalogValue(PIN_ANALOGICO_MEDIDA));
  int valPot =
    avgPot.Add(CurrentAnalogValue(PIN_ANALOGICO_POTENCIOMETRO));

  SensorData current;
  current.SetSensorValue(valSensor);
  current.SetPotValue(valPot);

  if (current != sensor)
  {
    Display(current.GetSensorValue(), current.GetPotValue(), current.GetAlarm());
    String data = DoSerial(current.GetSensorValue(), current.GetPotValue(),
      current.GetAlarm());
    BT.write(data.c_str());
    sensor = current;
  }

  delay(100);
}
```

Fichero CircularBuffer.h

```
template <class T>
class CircularBuffer {
  public:
  CircularBuffer (const int size);
  ~CircularBuffer ();
  void Add (const T& value);
  int NumberOfElements () const;
  T& operator[] (int index) const;
```

```
  private:
  int maxSize;
  int firstPos;
  int numberOfElements;
  T* buffer;
};

template <class T>
CircularBuffer<T>::CircularBuffer (const int size) {
  maxSize = size;
  firstPos = numberOfElements = 0;
  buffer = new T[maxSize];
}

template <class T>
CircularBuffer<T>::~CircularBuffer () {
    delete buffer;
}

template <class T>
void CircularBuffer<T>::Add (const T& value) {
  int firstFreePos = (firstPos + numberOfElements) % maxSize;
  buffer[firstFreePos] = value;
  if (numberOfElements >= maxSize)
    firstPos++;
  else
    numberOfElements++;
}

template <class T>
int CircularBuffer<T>::NumberOfElements () const {
  return numberOfElements;
}

template <class T>
T& CircularBuffer<T>::operator[] (int index) const {
  int pos = (firstPos + index) % maxSize;
  return buffer[pos];
}
```

Fichero MovingAverage.h

```
#include "CircularBuffer.h"

class MovingAverage {
  public:
  MovingAverage (const int size);
  float Add (const float newValue);

  private:
  CircularBuffer<float> avg;
};

MovingAverage::MovingAverage (const int size) :avg(size) {
}

float MovingAverage::Add (const float newValue) {
  avg.Add(newValue);
```

```
  float sum = 0;
  for (int i = 0; i < avg.NumberOfElements(); i++)
    sum += avg[i];

  return sum / avg.NumberOfElements();
}
```

Fichero SensorData.h

```
class SensorData {
  public:
  SensorData (int s, int p);
  void SetSensorValue (int value);
  int GetSensorValue () const;
  void SetPotValue (int value);
  int GetPotValue () const;
  bool GetAlarm () const;

  private:
  int valSensor;
  int valPot;
};

SensorData::SensorData (int s = 0, int p = 0) {
  SetSensorValue(s);
  SetPotValue(p);
}

void SensorData::SetSensorValue (int value) {
  valSensor = value;
}

int SensorData::GetSensorValue () const {
  return valSensor;
}

void SensorData::SetPotValue (int value) {
  valPot = value;
}

int SensorData::GetPotValue () const {
  return valPot;
}

bool SensorData::GetAlarm () const {
  return (valSensor > valPot);
}

bool operator== (const SensorData& left, const SensorData& right) {
  if (left.GetSensorValue() != right.GetSensorValue())
    return false;
  if (left.GetPotValue() != right.GetPotValue())
    return false;
  if (left.GetAlarm() != right.GetAlarm())
    return false;

  return true;
}
```

```
bool operator!= (const SensorData& left, const SensorData& right) {
  return !(left == right);
}
```

8.7 Sketch wearable fase 4

En esta evolución del *sketch* que ejecuta el *wearable* son especialmente reseñables dos cuestiones que representan una posible limitación *hardware*:

1. La gestión de la pulsación del botón se realiza mediante interrupciones. Este aspecto debe tenerse en cuenta a la hora de elegir el modelo Arduino ya que el número de interrupciones disponibles está limitado.
2. Para la generación de la seña que se utilizará para hacer vibrar el zumbador y que genere un sonido se utiliza un pin digital capaz de manejar PWM (*Pulse Width Modulation*). Este aspecto debe tenerse en cuenta a la hora de elegir el modelo Arduino ya que el número de pines con soporte para PWM está limitado

```
#define PIN_LED 7
#define PIN_NOISE 6
#define NOISE_FREQ 1000
#define PIN_BUTTON 2

#define PIN_RX 10 // pin Arduino a conectar con el pin TX del HC-05
#define PIN_TX 11 // pin Arduino a conectar con el pin RX del HC-05

#define BAUD 9600

#include <SoftwareSerial.h>
SoftwareSerial BT(PIN_RX,PIN_TX);
String data;
volatile bool AlarmAck = false;

void TurnOffNoise() {
  AlarmAck = true;
  noTone(PIN_NOISE);
}

void setup()
{
  pinMode(PIN_LED, OUTPUT);
  BT.begin(BAUD);
  pinMode(PIN_BUTTON, INPUT);
  attachInterrupt(digitalPinToInterrupt(PIN_BUTTON), TurnOffNoise, RISING);
}

void UnDoSerial (String str, int& valSensor, int& valPot, bool& alarm) {
  String aux = str.substring(0, str.indexOf('#'));
  valSensor = aux.toInt();
  aux = str.substring(str.indexOf('#') + 1, str.length());
  alarm = aux.toInt();
}

void loop()
{
  if(BT.available())
```

```
 {
   char c = BT.read();

   if (c == '\n')
   {
     int valSensor;
     int valPot;
     bool alarm;
     UnDoSerial(data, valSensor, valPot, alarm);

     if (alarm)
     {
       digitalWrite(PIN_LED, HIGH);
       if (AlarmAck)
         noTone(PIN_NOISE);
       else
         tone(PIN_NOISE, NOISE_FREQ);
     }
     else
     {
       digitalWrite(PIN_LED, LOW);
       noTone(PIN_NOISE);
       AlarmAck = false;
     }

     data = "";
   }
   else
     data += c;
 }
}
```

8.8 Configuración módulo ESP8266-01 para la fase 5

Para configurar el módulo ESP8266-01 enviándole comandos AT es necesario conectarlo a un PC usando un adaptador Serie USB a TTL con Chip FTDI FT232RL[5]. Para ello se preparó una *protoboard* de tamaño reducido cableada oportunamente para poder pinchar el módulo ESP8266-01 de forma cómoda y conectarlo así al puerto USB del PC, tal y como se muestra a continuación.

[5] https://www.amazon.es/DSD-TECH-Adaptador-FT232RL-Compatible/dp/B07BBPX8B8

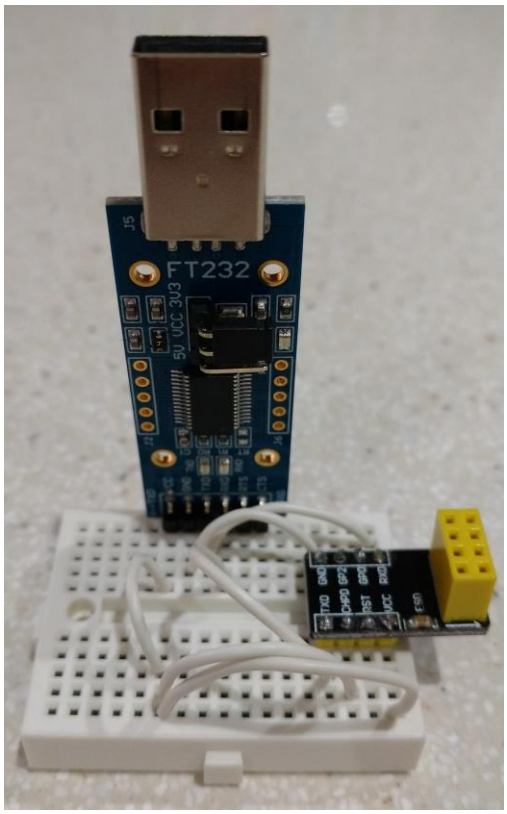

Ilustración 29

Los pines TX y RX del módulo ESP8266-01 deben conectarse a los pines RX y TX del adaptador. Esta inversión es necesaria porque lo que el PC envíe por el puerto serie deberá recibirlo el módulo ESP8266-01 y viceversa.
Los pines CHPD y VCC deben conectarse a la alimentación que debe ser de 3,3 V. No se requiere conversión de voltaje porque el módulo adaptador permite elegir la tensión entre 5 V y 3,3 V.
Los pines GND del adaptador y el módulo ESP8266-01 deben conectarse entre sí.
Los pines GPIO0 y GPIO2 deben dejarse sin conectar.
El pin RST debe dejarse sin conectar pero para iniciar el módulo hay que conectarlo momentáneamente a GND. Por sencillez no se incorporó un pulsador para esto último. Para iniciar el módulo cortocircuitaba GND y RST con un cable que no se muestra en la Ilustración 29.

Una vez conectado al PC el módulo ESP8266-01, se configuró enviando comandos AT a través del *software* Realterm[6], que emula un terminal serie configurado a 115200 baudios, tal y como requiere el módulo. Los comandos AT utilizados para configurar la conectividad WiFi del módulo ESP8266-01 son:

AT+GMR	Comprobación de versión de *firmware*.
AT+CWMODE=1	Activar el modo *host* (por defecto se comporta como un punto de acceso).
AT+CWJAP="SSID","PASSWORD"	Conexión a la red WiFi con SSID y PASSWORD.
AT+CIFSR	Comprobación de asignación de dirección IP por el servicio DHCP de la red WiFi a la que se conectó.
AT+PING="IP"	Comprobación de conexión a internet a través de la red WiFi (así se valida la asignación de puerta de enlace y de servidor DNS si en lugar de una dirección IP se prueba con una dirección DNS).

Es interesante resalta qur utilizar el monitor serie de la placa Arduino tal y como se hizo en la fase 3 para configurar el módulo Bluetooth no es posible. El motivo es que Arduino UNO no maneja correctamente puertos serie a 115200 baudios y reconfigurar el módulo ESP8266-01 a una velocidad inferior (por ejemplo 9600 baudios) lo inutilizaba.

[6] https://realterm.sourceforge.io

8.9 Conexión del módulo ESP8266 a Arduino UNO para la fase 5

El módulo ESP8266 debe conectarse a las tensiones de alimentación y tierra de forma similar a como se describió su conexión al adaptador USB TTL.

Sin embargo, aunque la placa Arduino ofrece alimentación a 3,3 V, esta salida no ofrece la corriente que requiere el módulo ESP8266 para operar, especialmente al iniciarse y cuando se transmiten datos vía WiFi. Por este motivo debe utilizarse una fuente de alimentación externa capaz de dar 3,3 V y más de 200 mA.

Para disponer de una fuente de alimentación con estas características se eligió una fuente referenciada en proyectos implementados con Arduino publicados en internet. Esta fuente ofrecía, en teoría, salidas configurables a 5 V y 3,3 V suministrando y 800 mA de corriente. Sin embargo, tras probarla se comprobó con un multímetro que si se configuraba para dar 3,3 V se obtenían casi 4 V, lo que supera el límite de los 3,6 V a partir de los cuales el módulo ESP8266 puede dañarse.

Para solventar este problema se configuró la fuente de tensión para dar 5 V y se incorporó un conversor de 5 V a 3,3 V que garantiza los 800 mA de corriente. No se utilizó un divisor de tensión pasivo mediante resistencias para garantizar que la corriente requerida estaba disponible.

Sí se utilizó un divisor de tensión pasivo para conectar el pin de RX del módulo ESP8266 al pin de la placa Arduino usado como TX. Esto se hizo para garantizar que en el pin RX del módulo ESP8266 también se utilizan 3,3 V en lugar de 5 V. Como ya pasó al conectar los módulos Bluetooth, el pin TX del módulo ESP8266 se conectó al pin de la placa Arduino usado como RX directamente porque la tensión de 3,3 V es reconocida por Arduino como un nivel alto en las entradas digitales.

En la Ilustración 30 puede verse el resultado de la modificación sobre el prototipo tal y como estaba al terminar la fase anterior.

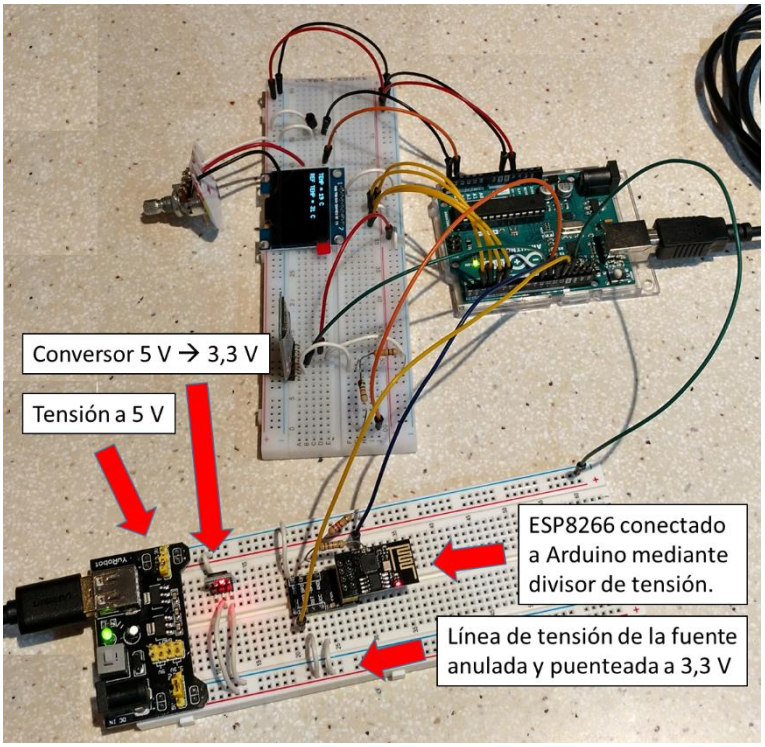

Ilustración 30

8.10 Sketch sensor fase 5

```
#include <U8g2lib.h>
#include <SoftwareSerial.h>
#include "MovingAverage.h"
#include "SensorData.h"

#define MY_IP "192.168.0.8"
#define THINGSPEAK_KEY "N7VK389S52SRWNTV"
#define THINGSPEAK_SENDING_FREQ_MS 60000

#define PIN_RX 10 // pin Arduino a conectar con el pin TX del HC-05
#define PIN_TX 11 // pin Arduino a conectar con el pin RX del HC-05
#define BT_BAUD 9600
SoftwareSerial BT(PIN_RX,PIN_TX);

#define PIN_RX 13 // pin Arduino a conectar con el pin TX del ESP8266
#define PIN_TX 12 // pin Arduino a conectar con el pin RX del ESP8266
#define WIFI_BAUD 115200
SoftwareSerial esp8266(PIN_RX, PIN_TX);

#define PIN_ANALOGICO_MEDIDA 5
#define VS_mV 5000
#define PIN_ANALOGICO_POTENCIOMETRO 4
#define DIGITAL_PIN_SCK 2
#define DIGITAL_PIN_SDA 3
#define DIGITAL_PIN_DC 4
#define DIGITAL_PIN_CS 5

U8G2_SH1106_128X64_NONAME_1_4W_SW_SPI u8g2(U8G2_R0,
  DIGITAL_PIN_SCK, DIGITAL_PIN_SDA, DIGITAL_PIN_CS,
  DIGITAL_PIN_DC);
```

```
void setup() {
  pinMode(PIN_ANALOGICO_MEDIDA, INPUT);
  pinMode(PIN_ANALOGICO_POTENCIOMETRO, INPUT);

  u8g2.setFont(u8g2_font_6x10_mf);
  u8g2.setFontRefHeightExtendedText();
  u8g2.setFontPosTop();
  u8g2.begin();

  BT.begin(BT_BAUD);
  esp8266.begin(WIFI_BAUD);
}

void SendTemperature2ThingSpeakDotCom (int temp) {
    esp8266.listen();

    String cmd;
    cmd = "AT+CIPMODE=0\r\n";
    esp8266.print(cmd.c_str());
    delay(500);
    cmd = "AT+CIPMUX=0\r\n";
    esp8266.print(cmd.c_str());
    delay(500);
    cmd = "AT+CIPSSLSIZE=4096\r\n";
    esp8266.print(cmd.c_str());
    delay(500);
    cmd = "AT+CIPSTART=\"SSL\",\"api.thingspeak.com\",443\r\n";
    esp8266.print(cmd.c_str());
    delay(1000);

    String httpGet = "GET /update?api_key=";
    httpGet += THINGSPEAK_KEY;
    httpGet += "&field1=";
    httpGet += String(temp);
    httpGet += "&headers=false HTTP/1.1\r\nHost: ";
    httpGet += MY_IP;
    httpGet += "\r\nConnection: close\r\nAccept: */*\r\n\r\n";

    cmd = "AT+CIPSEND=" + String(httpGet.length()) + "\r\n";
    esp8266.print(cmd.c_str());
    delay(500);
    esp8266.println(httpGet.c_str());
    delay(1000);
    cmd = "AT+CIPCLOSE\r\n";
    esp8266.print(cmd.c_str());
    delay(1000);

    while (esp8266.available())
      esp8266.read();

    BT.listen();
}

void loop() {

  static SensorData sensor;
  static MovingAverage avgSensor(30);
  static MovingAverage avgPot(10);

  int valSensor =
```

```
    avgSensor.Add(CurrentAnalogValue(PIN_ANALOGICO_MEDIDA));
  int valPot =
    avgPot.Add(CurrentAnalogValue(PIN_ANALOGICO_POTENCIOMETRO));
  SensorData current(valSensor, valPot);

  if (current != sensor)
  {
    DisplaySensor(current.GetSensorValue(),
        current.GetPotValue(),current.GetAlarm());
    String data = DoSerial(current.GetSensorValue(),
        current.GetPotValue(), current.GetAlarm());
    BT.write(data.c_str());
    sensor = current;
  }

  const unsigned int iter_delay_ms = 100;
  delay(iter_delay_ms);

  static unsigned int numIter = 0;
  if (numIter >= THINGSPEAK_SENDING_FREQ_MS / iter_delay_ms)
  {
    SendTemperature2ThingSpeakDotCom(sensor.GetSensorValue());
    numIter = 0;
  }
  else
    numIter++;
}
```

8.11 Sketch sensor fase 6

```
#include <M5Stack.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <WiFiClientSecure.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <BLE2902.h>

#include "MovingAverage.h"
#include "SensorData.h"

const int DEFAULT_REF_TEMP = 18;

const char* WIFI_SSID = "XXXXXXXXXXX";
const char* WIFI_PWD = "XXXXXXXXXXX ";
const unsigned long WIFI_MS_TIMEOUT = 10*1000; // 10 segundos

#define THINGSPEAK_SENDING_FREQ_MS 60000
const char* THINGSPEAK_KEY = "N7VK389S52SRWNTV";

const char* BLE_SERVICE_UUID =
  "19686c12-2faa-486e-9b1c-a1c286cfc540";
const char* BLE_CHARACTERISTIC_UUID =
  "d053c0e2-36cf-432f-9ea5-681ef9d5995f";

const int TMP_PIN = 36;

void PaintBaseLCD() {
```

```
  M5.Lcd.clear();

  M5.Lcd.setTextSize(3);
  M5.Lcd.setCursor(58, 215);
  M5.Lcd.setTextColor(RED);
  M5.Lcd.printf("+");
  M5.Lcd.drawRect(35, 212, 60, 30, RED);

  M5.Lcd.setTextSize(3);
  M5.Lcd.setCursor(150, 215);
  M5.Lcd.setTextColor(BLUE);
  M5.Lcd.printf("-");
  M5.Lcd.drawRect(127, 212, 60, 30, BLUE);
}

void UpdateLCD (const SensorData& s) {
    PaintBaseLCD();
    M5.Lcd.setTextColor(WHITE);
    M5.Lcd.setTextSize(3);
    M5.Lcd.setCursor(58, 15);
    M5.Lcd.printf("Current temp");
    M5.Lcd.setCursor(120, 55);
    M5.Lcd.printf(String(s.GetSensorValue()).c_str());
    M5.Lcd.printf(" C");
    if (s.GetAlarm())
    {
      M5.Lcd.setTextColor(YELLOW);
      M5.Lcd.setCursor(85, 95);
      M5.Lcd.printf("ALARM!!!");
    }
    M5.Lcd.setTextColor(WHITE);
    M5.Lcd.setCursor(40, 180);
    M5.Lcd.setTextSize(2);
    String text = "Ref temp ";
    text += String(s.GetPotValue());
    M5.Lcd.printf(text.c_str());
    M5.update();
}

void Sent2ThingSpeaks(const SensorData& s) {
  String request = "https://api.thingspeak.com/update?api_key=";
  request += THINGSPEAK_KEY;
  request += "&field1=";
  request += String(s.GetSensorValue());

  HTTPClient https;
  https.begin(request);
  int ret = https.GET();
  if (ret == HTTP_CODE_OK || ret == HTTP_CODE_MOVED_PERMANENTLY)
    String response = https.getString();
}

BLECharacteristic* pCharacteristic = 0;

void setup() {
  M5.begin();
  dacWrite(25, 0);

  WiFi.begin(WIFI_SSID, WIFI_PWD);
  unsigned long t = millis();
```

```
  while (WiFi.status() != WL_CONNECTED)
  {
    if (millis() - t > WIFI_MS_TIMEOUT)
      break;
    DisplayDebug("Connecting WiFi");
    delay(500);
  }

  String text;
  if (WiFi.status() == WL_CONNECTED)
  {
    text = "WiFi connected!\n\nAddress is ";
    text += WiFi.localIP().toString();
  }
  else
  {
    text = "WiFi connection failed!";
  }
  DisplayDebug(text);
  delay(2000);

  BLEDevice::init("Sensor");
  BLEServer* pServer = BLEDevice::createServer();
  BLEService* pService = pServer->createService(BLE_SERVICE_UUID);
  pCharacteristic = pService->createCharacteristic(
    BLE_CHARACTERISTIC_UUID,
    BLECharacteristic::PROPERTY_READ   |
    BLECharacteristic::PROPERTY_WRITE  |
    BLECharacteristic::PROPERTY_NOTIFY |
    BLECharacteristic::PROPERTY_INDICATE
    );
  pCharacteristic->addDescriptor(new BLE2902());
  pService->start();
  BLEAdvertising* pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(BLE_SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x06);
  pAdvertising->setMinPreferred(0x12);
  BLEDevice::startAdvertising();
}

void loop() {

  static SensorData sensor(0, DEFAULT_REF_TEMP);
  static MovingAverage avgSensor(100);

  int valSensor = avgSensor.Add(Volt2Temp(analogRead(TMP_PIN)));
  SensorData current(valSensor, sensor.GetPotValue());

  if ((current != sensor) || M5.BtnC.wasPressed())
  {
    sensor = current;

    UpdateLCD(sensor);
    String data = DoSerial(current.GetSensorValue(), current.GetPotValue(),
      current.GetAlarm());
    pCharacteristic->setValue(data.c_str());
    pCharacteristic->notify();
  }
```

```
  if (M5.BtnA.wasPressed())
  {
    sensor.SetPotValue(sensor.GetPotValue() + 1);

    String data = DoSerial(sensor.GetSensorValue(), sensor.GetPotValue(), sensor.GetAlarm());
    pCharacteristic->setValue(data.c_str());
    pCharacteristic->notify();

    UpdateLCD(sensor);
  }
  if (M5.BtnB.wasPressed())
  {
    sensor.SetPotValue(sensor.GetPotValue() - 1);

    String data = DoSerial(sensor.GetSensorValue(), sensor.GetPotValue(), sensor.GetAlarm());
    pCharacteristic->setValue(data.c_str());
    pCharacteristic->notify();

    UpdateLCD(sensor);
  }
  if (M5.BtnC.pressedFor(2000))
    M5.powerOFF();
  M5.update();

  const unsigned int iter_delay_ms = 100;
  delay(iter_delay_ms);

  static unsigned int numIter = 0;
  if (numIter >= THINGSPEAK_SENDING_FREQ_MS / iter_delay_ms)
  {
    Sent2ThingSpeaks(sensor);
    numIter = 0;
  }
  else
    numIter++;
}

void DisplayDebug (String str) {
  M5.Lcd.clear();
  M5.Lcd.setCursor(0, 0);
  M5.Lcd.setTextSize(2);
  M5.Lcd.printf(str.c_str());
  M5.update();
}
```

8.12 Sketch wearable fase 6

```
#include <M5Stack.h>
#include "BLEDevice.h"

const char* BLE_SERVICE_UUID =
  "19686c12-2faa-486e-9b1c-a1c286cfc540";
const char* BLE_CHARACTERISTIC_UUID =
  "d053c0e2-36cf-432f-9ea5-681ef9d5995f";
const unsigned long BLE_S_TIMEOUT = 10; // 10 segundos

String BLE_Buffer;
bool AlarmAck = false;
bool InAlarmStatus = false;
```

```
static boolean doConnect = false;
static boolean connected = false;
static boolean doScan = false;
static BLERemoteCharacteristic* pRemoteCharacteristic;
static BLEAdvertisedDevice* myDevice;

static void notifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t*
pData, size_t length, bool isNotify) {
    BLE_Buffer = (char*) pData;
}

class MyClientCallback : public BLEClientCallbacks
{
  void onConnect(BLEClient* pclient) {
    }
  void onDisconnect(BLEClient* pclient) {
    connected = false;
  }
};

bool connectToServer() {
  BLEClient* pClient = BLEDevice::createClient();
  pClient->setClientCallbacks(new MyClientCallback());
  pClient->connect(myDevice);
  BLERemoteService* pRemoteService = pClient->
    getService(BLEUUID(BLE_SERVICE_UUID));
  if (pRemoteService == nullptr)
  {
    pClient->disconnect();
    return false;
  }
  pRemoteCharacteristic = pRemoteService->
    getCharacteristic(BLEUUID(BLE_CHARACTERISTIC_UUID));
  if (pRemoteCharacteristic == nullptr)
  {
    pClient->disconnect();
    return false;
  }
  if(pRemoteCharacteristic->canNotify())
    pRemoteCharacteristic->registerForNotify(notifyCallback);
  connected = true;
}

class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks
{
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    if (advertisedDevice.haveServiceUUID() &&
      advertisedDevice.isAdvertisingService(BLEUUID(BLE_SERVICE_UUID)))
    {
      BLEDevice::getScan()->stop();
      myDevice = new BLEAdvertisedDevice(advertisedDevice);
      doConnect = true;
      doScan = true;
    }
  }
};

void UpdateLCD (int valSensor, bool alarm) {
  M5.Lcd.clear();
```

```
  if (alarm && !AlarmAck)
  {
    M5.Lcd.setTextSize(3);
    M5.Lcd.setCursor(58, 215);
    M5.Lcd.setTextColor(RED);
    M5.Lcd.printf("!");
    M5.Lcd.drawRect(35, 212, 60, 30, RED);
  }

  M5.Lcd.setTextColor(WHITE);
  M5.Lcd.setTextSize(3);
  M5.Lcd.setCursor(58, 15);
  M5.Lcd.printf("Current temp");
  M5.Lcd.setCursor(120, 55);
  M5.Lcd.printf(String(valSensor).c_str());
  M5.Lcd.printf(" C");

  if (alarm)
  {
    M5.Lcd.setTextColor(YELLOW);
    M5.Lcd.setCursor(85, 95);
    M5.Lcd.printf("ALARM!!!");
  }

  M5.update();
}

void setup() {

  M5.begin();
  M5.Speaker.begin();

  BLEDevice::init("Wearable1");
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setInterval(100);
  pBLEScan->setWindow(99);
  pBLEScan->setActiveScan(true);
  pBLEScan->start(BLE_S_TIMEOUT, false);
}

static bool firstIteration = true;

void loop() {

  if (doConnect)
  {
    connectToServer();
    doConnect = false;
  }

  if (firstIteration)
  {
    if(pRemoteCharacteristic->canRead())
    {
      std::string value = pRemoteCharacteristic->readValue();
      BLE_Buffer = value.c_str();
    }
    firstIteration = false;
```

```
  }

  int valSensor;
  int valPot;
  bool alarm;
  bool DisplayUpdateNeeded = false;

  if (BLE_Buffer != "")
  {
    UnDoSerial(BLE_Buffer, valSensor, valPot, alarm);
    DisplayUpdateNeeded = true;
    BLE_Buffer = "";

    if (alarm && !InAlarmStatus)
    {
      InAlarmStatus = true;
      AlarmAck = false;
      M5.Speaker.beep();
    }
    else if (!alarm && InAlarmStatus)
    {
      InAlarmStatus = false;
      AlarmAck = false;
      M5.Speaker.mute();
    }
  }

  if (InAlarmStatus && !AlarmAck && M5.BtnA.wasPressed())
  {
    AlarmAck = true;
    DisplayUpdateNeeded = true;
    M5.Speaker.mute();
  }

  if (InAlarmStatus && !AlarmAck)
    M5.Speaker.beep();

  if (DisplayUpdateNeeded)
    UpdateLCD(valSensor, alarm);

  if (M5.BtnC.pressedFor(2000))
    M5.powerOFF();

  M5.update();

  delay(100);
}
```