Trabajo de Fin de Grado

Monitorización de variables de entorno mediante Spark

Monitoring of environment variables using Spark

Héctor Royo Concepción

La Laguna, 8 de septiembre de 2015

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43.826.207-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería informática y de sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"Monitorización de variables de entorno mediante Spark"

ha sido realizada bajo su dirección por D. **Héctor Royo Concepción**, con N.I.F. 42.195.789-N.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de septiembre de 2015.

Agradecimientos

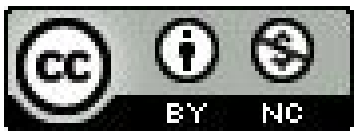
A Jesús Torres Jorge, mi tutor de TFG, por toda su ayuda

A mis padres, que me han apoyado durante todos estos años

A mis compañeros, "*los de aquí, y los de allá"*

Y a todo el profesorado del que he aprendido tanto

Licencia



Resumen

La “Inteligencia Ambiental” es un concepto nuevo y prometedor, que persigue mejorar los entornos en las que las personas nos desenvolvemos. A través de modelos automatizados, se buscan entornos reactivos a la presencia y necesidades de las personas.

En la actualidad, cada vez es más común que los objetos cotidianos se interconectan entre sí, a través del Internet de las Cosas. A través de la comunicación de los objetos, se puede obtener una mejoría en la calidad ambiental.

En este proyecto, se presenta un dispositivo capaz de detectar niveles de ruido y de luz en un entorno en tiempo real, y de actuar de forma consecuente a las medidas tomadas.

Palabras clave: *Inteligencia Ambiental, Internet de las Cosas*

Abstract

"Ambient Intelligence" is a new concept which is rising, and it seeks to improve the environments where we the people develop. By using automated models, it searches to get environments which are responsive to the presence and needs of the people.

Nowadays, it is becoming more and more common to see daily objects interconnected, using the Internet of Things (IoT). A good communication between this objects, can improve the quality of the environment.

This project presents a device capable of getting real time values of noise and luminosity, acting accordingly to data collected.

Keywords: *Ambiental Intelligence, Internet of Things*

Índice General

Índice de figuras

Capítulo 1. Introducción

1.1 Introducción General

En la actualidad, son muchos los entornos de trabajo en los que la calidad del entorno resulta crucial, y en los que factores como el ruido, la temperatura, la luminosidad o la calidad del aire pueden afectar la productividad. A través de sensores que midan las variables anteriormente citadas y sistema que actúen en consecuencia a los datos recogidos, un entorno puede mantener unos valores óptimos de forma automatizada.

Éste proyecto pretende realizar dicha tarea, centrándose en dos de los elementos anteriormente citados, el ruido y la luminosidad. Es por esto que resulta interesante para entornos como salas de estudio, bibliotecas o salas de ordenadores destinadas a estudiantes.

1.2 Proyecto

Este proyecto pretende aplicar la idea de la Inteligencia Ambiental a través de un ordenador de placa reducida (Single Board Computer, SBC), que, dotado de los sensores necesarios, sea capaz de monitorizar las variables de ruido y luz en un entorno.

Además del propio SBC, es necesario contar con un servidor, que servirá de interfaz para el usuario final. El servidor mantendrá un servicio web desde el que se pueden observar los valores captados por los distintos sensores. La comunicación entre servidor y el objeto dotado del SBC, se realiza siguiendo las pautas del Internet de las Cosas.

Se ha supuesto el caso del control de las dos variables antes citadas para una sóla habitación, reflejando los resultados cada cierto intervalo de tiempo en un sistema de monitorización web.

1.3 Internet de las Cosas e Inteligencia Ambiental

Internet de las Cosas, o IoT (del inglés, Internet of Things), es el nombre que se la ha dado a la interconexión de objetos cotidianos. Para que esto sea posible, los objetos deben estar dotados de Sistemas Empotrados, es decir, de sistemas electrónicos, de software, de sensores y de algún tipo de hardware de comunicación.

Gracias a estas redes de objetos se consiguen lo que se define como entornos dotados de Inteligencia Ambiental, o entornos inteligentes. Se caracterizan por ser entornos que a través de una serie de sistemas empotrados, perfectamente integrados e idealmente invisibles para el usuario, son capaces de conocer el contexto en el que desempeñan su función, para ofrecer soluciones anticipadas, a partir de las conclusiones que obtengan a partir de la información que manejan en cada momento.

En el futuro, el escenario ideal para la Inteligencia Ambiental es aquel donde no solo cada entorno cuenta con su infraestructura de IoT, sino que además todas estas redes de objetos están interconectadas, compartiendo su información y adaptándose mejor las necesidades de un público mayor. Esto obviamente levanta polémica, pues está presente el miedo a la pérdida de privacidad, algo que no debería ocurrir siempre que alguna de las grandes compañías no pueda acceder y analizar la información para su propio beneficio.

Capítulo 2.
Hardware

A la hora de elegir el hardware, primó la elección de un SBC, que enfocado al Internet de las Cosas, cumpla las necesidades de comunicación, tamaño y posibilidades. Se seleccionó el Particle Spark Core. Dotado de conexión Wi-Fi, este dispositivo es ideal para este proyecto, ya que permite una comunicación con el servidor, que está situado en un ordenador personal, de manera inalámbrica, sin necesidad una instalación cableada.

Además del SBC, se seleccionaron distintos sensores que gracias a su pequeño tamaño, pueden ser integrados en un entorno, sin resultar invasivos para el usuario.

2.1 Particle Spark Core

Lanzado en 2013 por Particle (anteriormente Spark), este kit de desarrollo permite crear prototipos con capacidades de conexión vía Wi-Fi gracias a su módulo CC3000.

Figura 2.1 Spark Core

El Spark Core se comercializa como hardware libre. Sin embargo, su uso está enfocado a la utilización del servicio de nube de Particle. En un uso típico de éste SBC, es necesario una conexión a internet, por lo que lo primero es conectar el Spark Core a Wi-Fi, reclamarlo a través de la nube, y después, realizar la programación a través del Web IDE. Una vez programado, la ejecución realizada por el SBC se comunica de nuevo con la Spark Cloud. Es un uso similar al del Electric Imp, otro SBC utilizado anteriormente en proyectos similares, que también restringe su uso al obligar a usar un servicio de nube.

Sin embargo, por la naturaleza del proyecto, utilizar una nube centralizada ajena a nosotros, que nos exija una conexión a internet constante para obtener los resultados no es interesante. Es por ello que se configuró una nube local como se explica en el apartado de software, algo que si bien es posible gracias a código abierto

de Particle, no resulta para nada trivial. Esta configuración local permite programar, ejecutar y obtener los resultados a través de una red Wi-Fi pero sin ser necesaria una conexión a internet.

En cuanto a las posibilidades de conexión del Spark Core, se cuentan con 8 pines PWM, alimentación de 3.3V y 3.3V con bajo nivel de ruido y pines SDA y SCL para la comunicación con dispositivos I2C, como es el caso del sensor de luz.

2.2 Micrófonos

En algunas zonas de trabajo como puede ser una biblioteca o una oficina, el ruido es un problema que puede afectar directamente al correcto funcionamiento de estos lugares de trabajo. Habitualmente, especialmente en bibliotecas, es el propio personal el que se encarga de detectar y actuar en caso de ruido. Sin embargo, la tendencia a aumentar la superficie de estas áreas de trabajo, hacen difícil que el personal pueda ser consciente del ruido existente en cada punto del entorno de trabajo.

Se plantea una monitorización de ruido a través de micrófonos. Los micrófonos son capaces de captar el sonido de su alrededor y transformarlos en información útil.

Se ha seleccionado el modelo MAX4466, un micrófono de ganancia ajustable que opera con voltajes de 2.4V a 5.5V, lo que lo hace compatible con el Spark Core. El micrófono devuelve un voltaje como salida, que como se ve en el apartado de software se puede utilizar para calcular el valor de sonido en decibelios, la unidad relativa de la acústica.

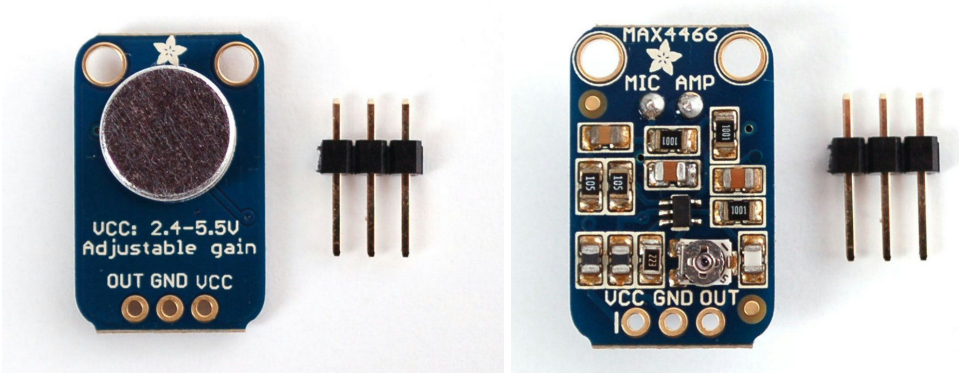


Figura 2.2 Microfono MAX4466

A la hora de trabajar con varios de estos micrófonos, su mejor virtud puede convertirse en su peor defecto: Al ajustarse la ganancia de forma física a través de un potenciómetro, es bastante probable los micrófonos tengan un ajuste levemente diferente, que se refleja en una diferencia en el nivel de sonido medido por cada uno ellos bajo las mismas circunstancias.

Se han utilizado 4 micrófonos, con el planteamiento más lógico para una habitación cuadrada, situándose uno en cada esquina.

2.3 Sensor de luz

La justificación del sensor de luz, además de por el motivo obvio de mejorar las condiciones de visión para las personas en su puesto de trabajo, viene dada por utilizar sistemas de ahorro de energía. Un sensor de luz, combinado con un sistema de iluminación, podría ser capaz de detectar el nivel de luz actual y reaccionar, por ejemplo aumentando la cantidad de luz artificial cuando comienza a anochecer, o disminuyendo la intensidad de luz artificial en caso contrario.

Para este proyecto, se ha seleccionado el sensor de luz digital TSL2561. Dotado de un interfaz I2C, es posible utilizar hasta 3 de estos dispositivos en un único SBC, lo que permitiría cubrir 3 áreas diferentes.

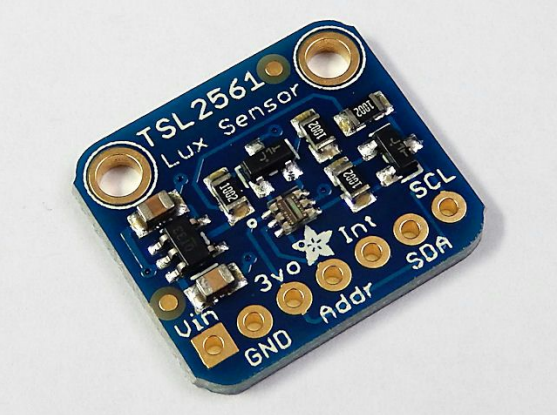


Figura 2.3: Sensor de Luz TSL2561

En la configuración de este proyecto se ha utilizado un único sensor, que deberá ir situado en el centro de la habitación y será el encargado de obtener los valores de luminosidad.

2.4 Otros dispositivos

Además de Spark Core, los micrófonos y el sensor de luz, se han utilizado otros dispositivos que a continuación serán descritos brevemente:

2.4.1 Batería/Cargador

Batería de 6600mAh con voltaje de salida de 3.7V. Utilizada como fuente de alimentación del montaje realizado. Se utilizó un cargador basado en el controlador de carga MCP73833. Este cargador, alimentado vía USB mini-B, permite cargar la batería seleccionada. Como se explica en el apartado de software, se utilizaron dos modos de trabajo distintos, basados en el consumo. En el capítulo de resultados se detalla la autonomía obtenida.

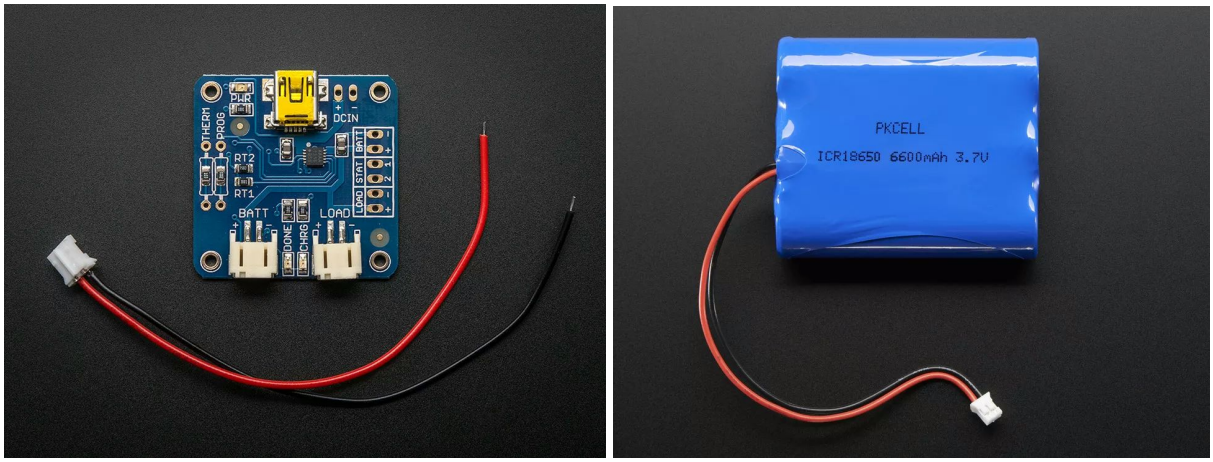


Figura 2.4.1: Batería y cargador

2.4.2 Barra de Leds

Bi-Color(Red/Green) 24 Bar BarGraph: Se utilizó como primera interfaz. Consistente en 24 leds de dos colores cada uno, esta barra de led es compatible con el protocolo I2C. Este dispositivo sirve para mostrar de forma gráfica el nivel de ruido en tiempo real. Sin embargo, para alargar la vida de la batería y por incompatibilidad con el sensor de luz que usa el mismo protocolo, se excluyó del montaje final.

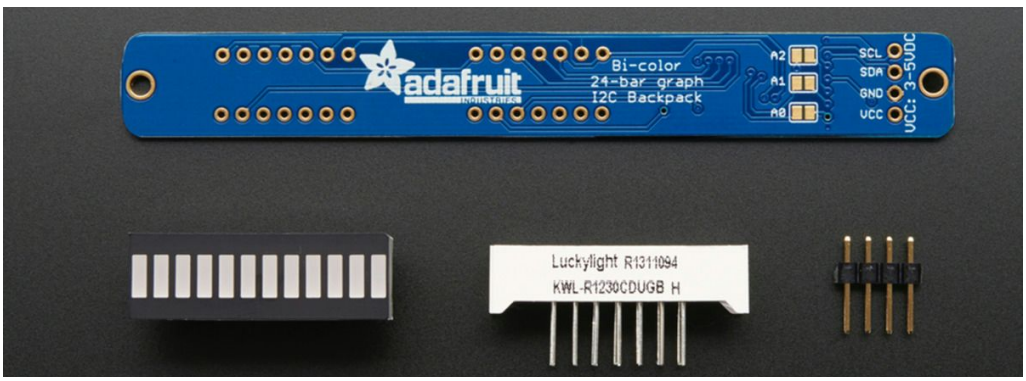


Figura 2.4.1: Barra de LEDs

2.5 Esquema de conexión

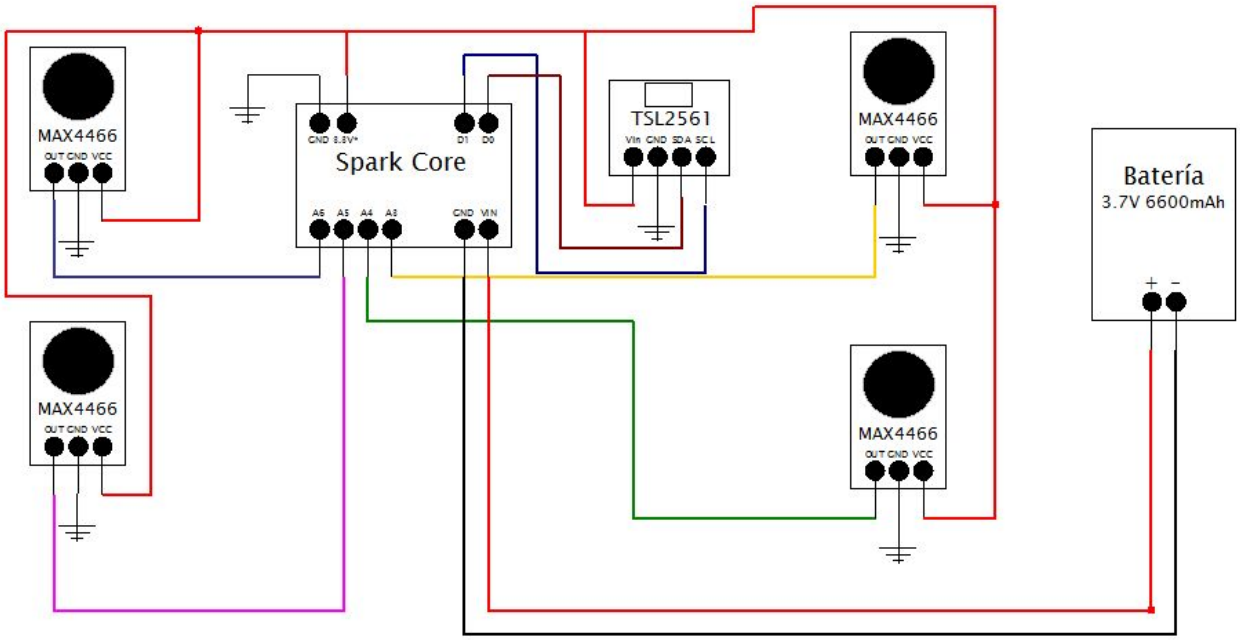


Figura 2.5 Esquema de Conexión

En el esquema de conexión que se presentan, sólo aparecen las conexiones que se han utilizado. El Spark Core cuenta con muchas conexiones más, lo que le permitirá añadir nuevos sensores, ya sean del mismo tipo o distintos. Destacar el uso de los pines D1 y D0 como pines para el protocolo I2C, que como se observa, se comunica con los pines SCL y SDA del sensor de luz correspondientemente.

Para los micrófonos, se conecta su salida a los pines A6,A5,A4 y A3 del SBC, que serán configurados como pines de entrada analógica. La alimentación de los micrófono y del sensor de luz, se hace a través del pin de 3.3V*, que es una salida del voltaje indicado que además aplica un filtro para evitar ruidos.

En cuanto a la alimentación del Spark, se utiliza la batería de 3.7V. Dado que el Spark Core admite voltajes en el rangos 3.6V - 6V, no existe ningún tipo de incompatibilidad. Además, gracias a su gran capacidad, 6600mAh, proporciona una gran autonomía, tal y como se detalla más adelante.

Capítulo 3.
Software

Para que poder obtener los resultados de los sensores seleccionados, es necesario contar una programación tanto de cara al Spark Core como al servidor. En el servidor además de almacenar, se transforman los datos de manera que la información se muestre forma útil para el usuario.

3.1 Programación del Spark Core

La programación del Spark Core está condicionada por el tipo de sensores que se seleccionó y por la comunicación que se ha elegido. En lugar de transmitir los valores según se capturan, se planteó enviar valores sólo una vez por cada periodo de tiempo. Sin embargo, finalmente se optó por enviar los valores medios de cada sensor por cada periodo.

Se plantearon dos modos de trabajo, definidos por la duración del periodo. Por un lado, se seleccionó un periodo corto, de tan solo 5 segundos, que obliga al Spark Core a estar conectado continuamente, para recibir las peticiones del servidor. Este modo ofrece resultados continuos y precisos, pero exige un mayor consumo. Por otra parte, se usó un periodo más largo, de 5 minutos, que permite al Spark Core reposar, desconectar su módulo WiFi, y sólo conectarlo cuando esté listo para enviar información. Los datos son menos precisos, pero se gana mucho en autonomía.

3.2 Configuración de la Nube Local

El Particle Spark Core es comercializado como hardware libre, lo que permite realizar variaciones sobre su uso inicial. Aunque generalmente una aplicación que haga uso de este dispositivo se apoya en el servicio en la nube Spark Cloud, es posible configurar una nube de forma local gracias a los aportes de la creciente comunidad de usuarios de Particle.

Para conseguir realizar una aplicación en una nube local, son necesarios:

- Servidor Spark, que sustituirá a Spark Cloud en la comunicación con el Core.
- Particle Spark Core, el dispositivo que ejecuta la aplicación.
- Red de Interconexión, puesto que aunque se configure de forma local, el Spark debe realizar la comunicación vía WiFi.

Adicionalmente, se han de configurar los compiladores y los medios para grabar la programación en el Core.

Los pasos que se siguieron para la configuración de la nube local fueron los siguientes.

3.2.1 Servidor

El servidor se configuró en un ordenador personal bajo el sistema operativo Ubuntu 12.04 LTS. Los archivos para instalar el servidor están disponibles en el repositorio https://github.com/spark/spark-server/.

3.2.2 Spark Core

El dispositivo, por defecto, apunta hacia la Spark Cloud. Es necesario que pase a apuntar al servidor local, y para ello, ambos deben encontrarse en la misma red. Se utilizo un router ADB PDGA4001n con cifrado WPA-PSK. Aunque técnicamente es posible realizar la conexión con los cifrados WEP o WPA2, en la práctica no consiguió ningún resultado positivo. Una vez conectados a esta red común Servidor y Core, es necesario indicarle al dispositivo la IP/dominio del servidor y además la clave pública del servidor. Si la configuración es correcta, el servidor detectará el Core.

3.2.3 Compilación / Firmware

Utilizando Spark Cloud, es en el propio Web IDE dónde se escribe el código, se compila y se transmite al Core automáticamente. Sin embargo, con la configuración local, esto ya no es posible, pues el Spark ya no se comunica con la nube de Particle. Es necesario contar con los medios para programar, compilar y grabar el firmware de forma independiente.

Para compilar, es necesario descargar Spark Core Firmware, los ficheros para hacer compatible el compilador de GCC con el Spark. La programación del Spark Core se realiza en un lenguaje basado en Wiring, similar al utilizado por Arduino.

Una vez compilados los ficheros compatibles con el Spark Core, es necesario grabar dicho Firmware al Core. Este proceso se realiza mediante conexión USB utilizando DFU-Utils (Device Firmware Upgrade Utilities).

3.2.4 Ejecución de un programa en la nube local

Una vez realizada toda la configuración anterior y con un programa ya grabado en el Spark Core, se pueden obtener resultados, llamar a funciones, monitorear

variables, etc, mediante la línea de comandos Spark-Cli. La línea de comandos consultará a la nube local, y esta se comunicará con el dispositivo conectado vía WiFi.

En este proyecto sólo se ha trabajado con un sólo Core, pero la configuración es compatible con varios. Sólo es necesario identificarlos en el servidor, gracias al ID único de cada dispositivo.

3.3 Servidor LAMP

Para que los usuarios puedan interactuar con la información recopilada, hace falta contar con una interfaz que proporcione un uso fácil e intuitivo. Se tomó la decisión de hacerlo a través de un servicio Web, que se apoye en un servidor LAMP. Un servidor LAMP es una infraestructura que sobre un sistema operativo Linux, proporciona un servidor web Apache, un gestor de base de datos MySQL y aplicaciones en el lenguaje de programación PHP.

Se ha configurado un Panel de Control que permite acceder a toda la información que ha sido captada por sensores, y que, además, realiza transformaciones sobre ellas para, por ejemplo, mostrar cómo ha ido cambiando los niveles de ruido en las últimas horas.

3.3.1 Panel de Control:

Monitorización

El panel de control consta de dos secciones fundamentales. En la primera sección. Monitorización, se muestran los últimos valores de ruido y luminosidad, captados por los micrófonos y sensor de luz en las unidades universales correspondientes, decibelios y lux correspondientemente.

Además de los últimos valores numéricos obtenidos, se muestran gráficas con los últimos 50 valores de cada sensor. Dado que cada medida se realiza cada 5 segundos, la información que se muestra en cada momento corresponde a las últimas 4 horas. Crear estas gráficas se ha utilizado una combinación de Javascript, para actualizar las gráficas cada 5 segundos, y PHP, más concretamente la librería enfocada a la creación de gráficas JpGraph.

Adicionalmente, dado que el prototipo no consta de sensores de temperatura o humedad, se muestra los valores para estas variables proporcionados por la Agencia Estatal de Meteorología, para la localización dónde se ubique el dispositivo.

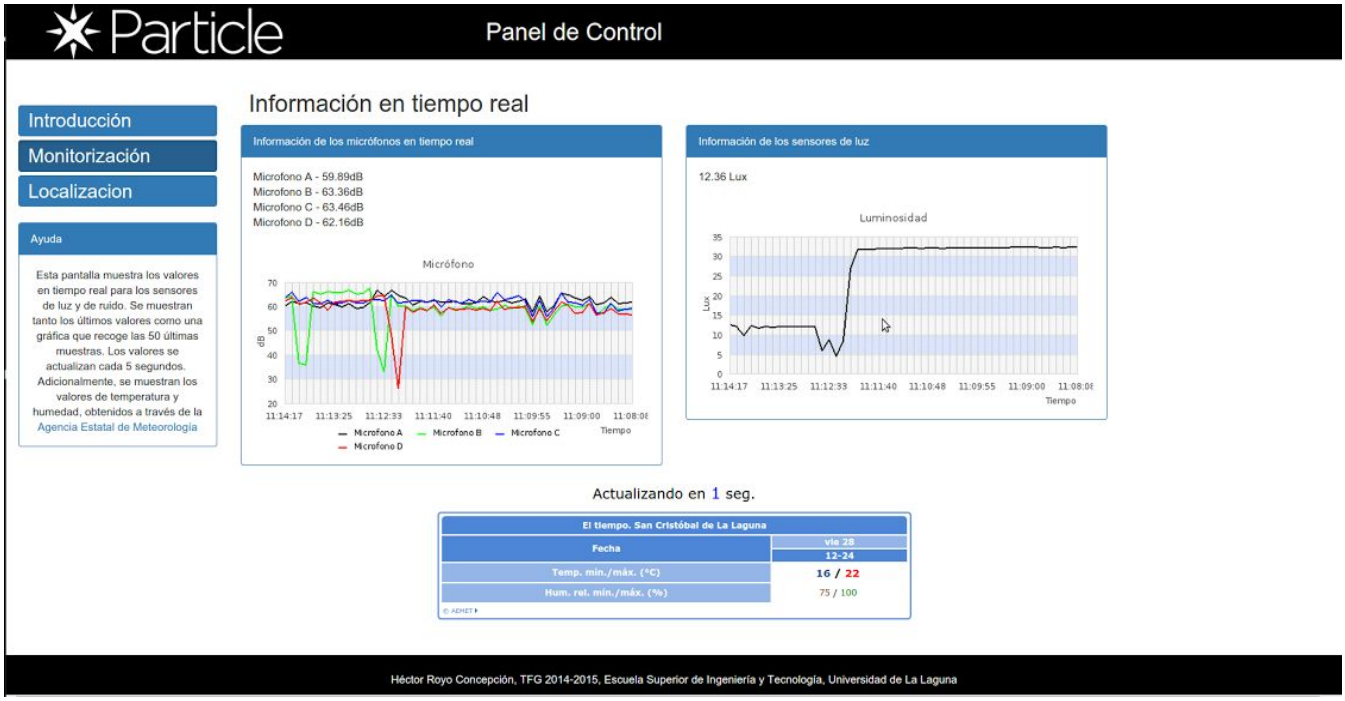


Figura 3.3.1.1 Panel de Monitorización

Localización

Este apartado realiza una aproximación a una aplicación real que se podría realizar con las mediciones de ruido. Se ha supuesto una habitación cuadrada, en la que se ha situado un micrófono en cada esquina. Gracias a los valores recogidos por cada micrófono, se puede realizar una cuadrícula de la sala que estime, gracias a un código de colores, en qué zonas se encuentra el ruido.

Esta aplicación ejecuta un algoritmo que estima el sonido para cada punto de la habitación a partir del sonido real captado por cada micrófono, y el de los puntos más contiguos a dicho punto. En el panel de Ayuda situado a la izquierda existe una leyenda que asocia cada color a un rango de decibelios.

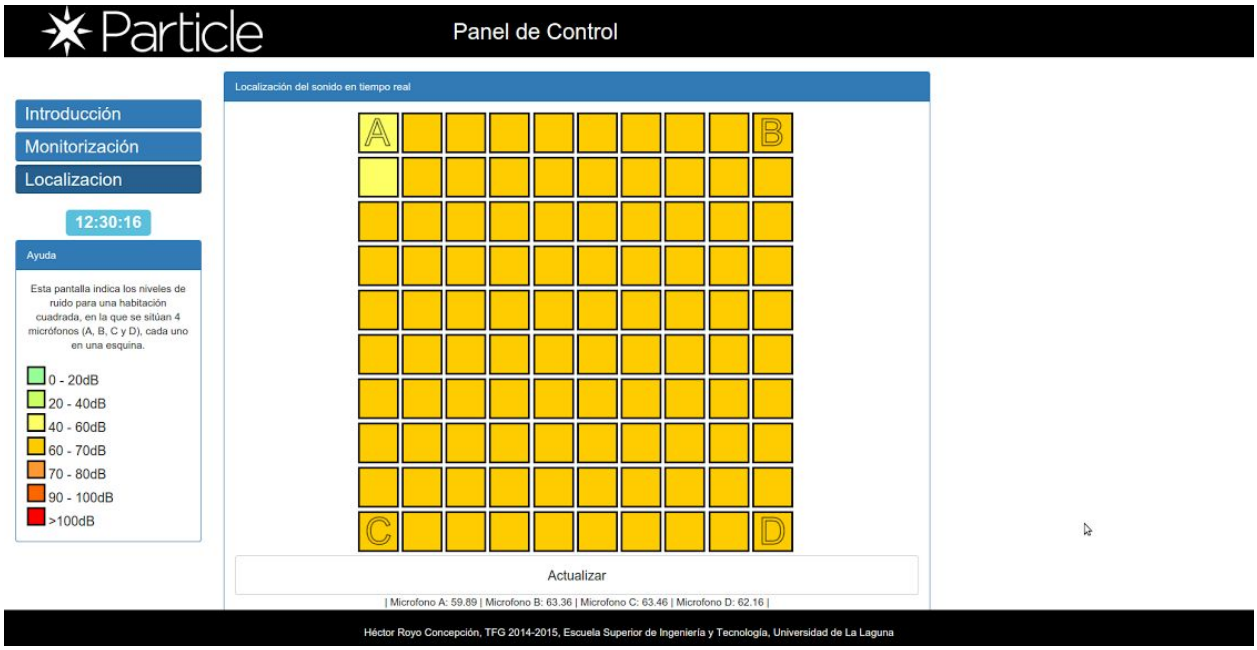


Figura 3.3.1.2 Panel de Localización

3.3.2 MySQL

Para que las aplicaciones anteriores sean posibles, es necesario contar tanto con los valores actuales de ruido/luminosidad, como valores anteriores. Es por ello que se utiliza una base de datos mysql que almacena los valores de las variables del Spark Core, junto con la fecha y hora asociada.

Como se citó anteriormente, existen dos modos de trabajo, según el periodo seleccionado, que pueden ser 5 segundos o 5 minutos.

3.3.2 Modo de trabajo con periodo de 5 segundos

La idea de este modo es que el Particle Spark core recoja los datos de luz y sonido durante 5 segundos, y que pasado este tiempo, calcule los valores medios. En el servidor, se ejecuta un Script, que cada 5 segundos, consulta al Spark por los valores medios y los inserta en la base de datos. Esto quiere decir que la comunicación es iniciada por el servidor, y que el Spark debe de ser capaz de responderle en el instante que se le solicite.

Esto repercute en los resultados de dos maneras: Por una parte, se obtienen resultados fiables y precisos, prácticamente en tiempo real. De cara a la página web, se pueden observar mucho mejor los resultados, puesto que todos los valores se actualizan cada 5 segundos. Por otra parte, el consumo es mucho mayor. El Spark Core debe mantener su conexión constantemente, puesto que en 5 segundos no es capaz de cerrar la conexión y abrirla tan sólo cuando tenga el resultado. Esto tiene un gran impacto si se utiliza la batería, ya que ve su autonomía muy menguada.

Es por todo esto que este modo está pensado para sistemas críticos, en los que sea necesaria esta gran cantidad de información, y en los que no existan limitaciones en cuanto a la alimentación.

3.3.3 Modo de trabajo con periodo de 5 minutos

En este caso, los valores se recogen durante un periodo de 5 minutos, y se calcula el valor medio, igual que en el caso anterior. Sin embargo, existen novedades, ya que ahora el Spark Core es capaz de desconectarse de la red WiFi, y conectarse cuando cuente con los datos para iniciar la comunicación con el servidor.

Esto significa que durante 5 minutos, el Spark Core trabajará sin conexión, de manera autónoma. Pasado este tiempo, se calcularán los valores medios de ruido y luminosidad, y además, se encenderá el módulo WiFi y realizará la conexión con la nube local. Una vez están los valores calculados y la conexión establecida, el Spark realiza una petición de HTTP GET a un script PHP que será el encargado de actualizar la base de datos e insertar la luminosidad y ruido captado en los últimos 5 minutos.

Esto cuenta con sus ventajas e inconvenientes. Por un lado, se consigue un aumento de la batería muy considerable, dado que el módulo WiFi es el elemento que más consume, y pasa de estar conectado continuamente a sólo unos segundos por

cada 5 minutos. En contra, se pierde calidad en los datos, ya que el valor es mucho menos representativo, y no se cuenta con el flujo de datos continuo del caso anterior.

Para solucionar la calidad en los datos, se ha propuesto un sistema de alarma. Este sistema, hace que el Spark no sólo se conecte y envíe datos cada 5 minutos, sino que, si en cualquier momento se captura un nivel de ruido superior a un umbral, establecido en 65dB, también sea capaz de conectarse e informar de la anomalía.

Con esta modificación, el consumo vendrá ligado a los niveles de ruido. En un entorno con sonidos estridentes, las conexiones serán más frecuentes, y la autonomía se verá afectada. Sin embargo, al estar enfocado a entornos tipo bibliotecas u oficinas, serán más frecuentes los períodos de bajo nivel de ruido que alto, por lo que seguirá mostrando una autonomía mucho mayor que en el caso del periodo de 5 segundos.

Capítulo 4.
Resultados

Una vez detallados el sistema hardware y software que se ha utilizado y desarrollado, podemos concretar en los resultados que este proyecto proporciona.

4.1 Ruido

En cualquiera de los modos anteriormente citados (periodo de 5 segundos o 5 minutos), el nivel de ruido ambiental se obtiene como el valor en decibelios que es capaz de captar cada uno de los micrófonos. Sin embargo, el valor que proporcionan los micrófonos no es directamente el nivel de decibelios, sino que se trata de un valor analógico que oscila entre los valores 0 - 2048 - 4096, siendo 2048 en mínimo valor, 0 el máximo valor negativo y 4096 el máximo valor positivo.

Para obtener el valor en decibelios, lo primero que se hizo fue transformar estas lecturas analógicas a voltaje real. Esto se consigue obteniendo la amplitud de pico a pico (diferencia mínimo, máximo) durante un periodo de tiempo. En este proyecto se ha utilizado un periodo de 50ms. Esto significa que cada medición en realidad es la diferencia del valor máximo y mínimo obtenido en el periodo establecido.

El último paso para obtener el nivel en decibelios fue utilizar la fórmula $10 \cdot \log_{10}(V_1^2 \ / \ V_0^2)$, que usa los valores de voltaje máximo proporcionado (V_1 = 3.3V) y el leído (V_0). Los resultados obtenidos así también dependiendo del nivel de ganancia de cada uno de los micrófonos. Para realizar una calibración, se realizó un montaje con todos los micrófonos uno al lado del otro, y a la misma distancia de una fuente de sonido única. Además, se utilizó como apoyo una aplicación para teléfonos móviles que proporcionaba un valor aproximado del nivel de ruido que se captaba en la misma zona.

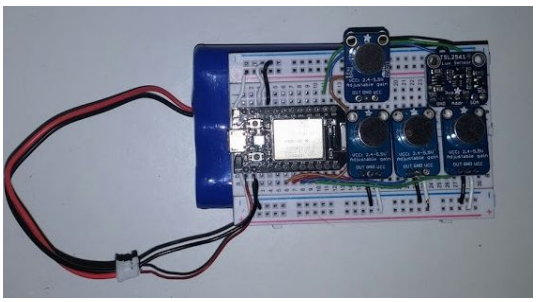


Figura 4.1.1 Montaje de calibración de micrófonos

Los valores de sonido ambiental, ya en la unidad correcta, se recopilan durante el periodo de trabajo establecido, y se calcula el valor medio. Este valor se introduce en una base de datos MYSQL, junto con una serie de atributos. Éstos son la fecha y la hora en la que se introduce el sonido, el micrófono que ha captado dicho sonido y un identificador.

Con los valores de la base de datos, se realizan las funcionalidades del panel de control, como es por ejemplo el sistema de monitorización gráfica. Se utiliza la librería JPGraph de PHP para crear gráficas que muestran las últimas 50 mediciones de cada micrófono.

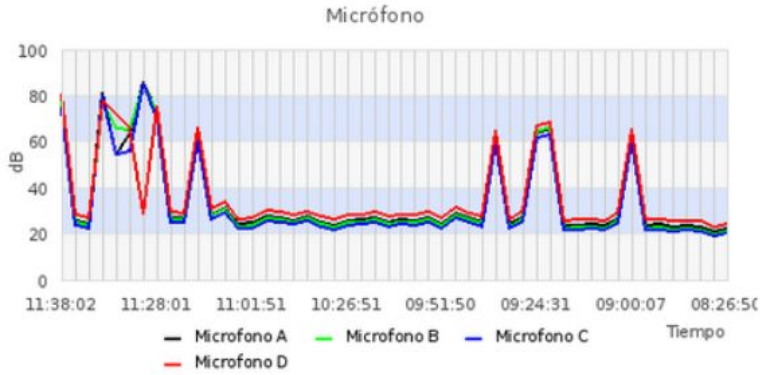


Figura 4.1.2 Gráfica de sonido

Otra de las funcionalidades que se plantea es la de la localización del sonido. Se supone una habitación cuadrada con un micrófono en cada esquina, y a partir del sonido que capta cada micrófono, se dibuja un mapa de distribución de ruido. En este supuesto, la fuente de ruido es única.

4.2 Luz

Para obtener los valores de luminosidad a partir del sensor no es necesario realizar tantos pasos como sucede con el sonido. El sensor utilizado es un dispositivo digital, que cuenta con su librería que permite establecer los parámetros deseados y obtener los valores de luminosidad sin mayor problema.

Al igual que con el ruido, se captura durante un periodo de tiempo, se calcula el valor medio y se envía a la base de datos, junto a los valores de identificador, fecha y hora. No se inserta el dispositivo que ha realizado la captura, porque en este montaje es único.

La funcionalidad que se ha desarrollado para la variable de la luminosidad ha sido también la de generar gráficas con JPGraph, que permiten observar cómo han variado los niveles de luminosidad a lo largo del tiempo.

Figura 4.2 Gráfica de luminosidad

4.3 Batería

Dado que se busca un sistema autónomo, la duración de la batería es un asunto crucial. Cada uno de los esquemas de trabajo planteados tienen metas diferentes. Si bien el modo de trabajo con periodo de 5 segundos proporciona resultados fiables, cercanos al tiempo real, el modo con periodo de 5 minutos ofrece una autonomía mucho mayor.

Estos datos quedan contrastados mediante pruebas reales. Se ha hecho funcionar cada uno de los modos con la batería al máximo de su carga hasta que se ha descargado, midiendo el tiempo de autonomía para cada caso.

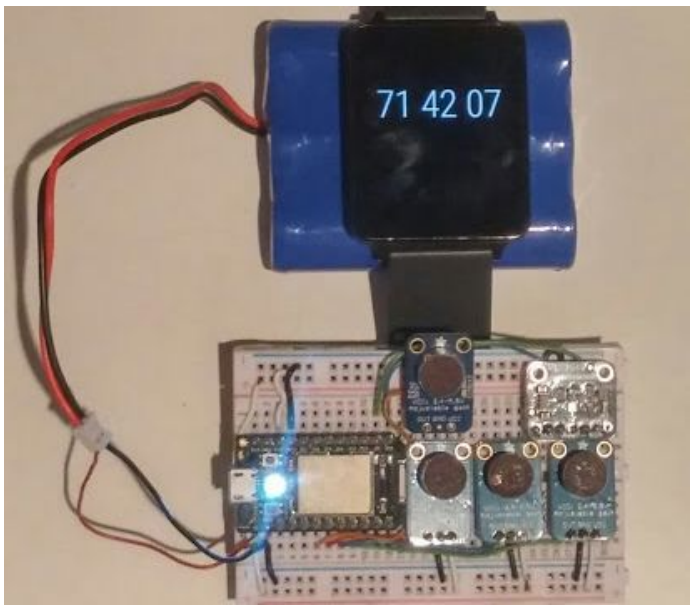


Figura 4.3.1 Cronometrado de la duración de la batería

En la siguiente tabla se recogen las mediciones realizadas:

Modo de trabajo	Autonomía
Periodo de 5 segundos	35 horas
Periodo de 5 minutos	<80 horas

Tabla 4.3.2. Tabla de autonomía por modo de trabajo

4.4 Resultados excluidos del montaje final

Durante el desarrollo del proyecto, existieron distintas opciones, desechadas del montaje final. Es el caso de la barra de leds. Pese a que se consiguió que funcionara correctamente, mostrando el nivel de sonido que captaba uno o varios micrófonos, no se pudo hacer funcionar de forma simultánea al sensor de luz.

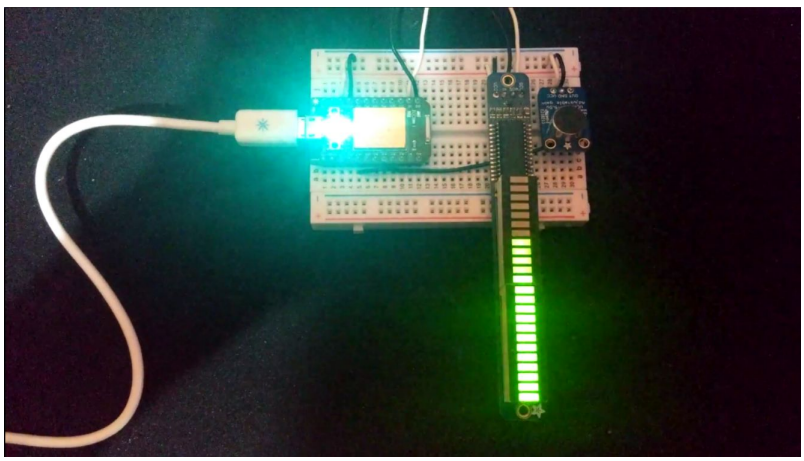


Figura 4.4 Funcionamiento de la barra de leds junto a un micrófono

Tanto el sensor de luz como la barra de leds hacen uso del protocolo I2C, como de unas librerías propias para poder operar con ellas desde Arduino. Modificando las dependencias de estas librerías para que fueran compatibles con Spark, se intentó asignar direcciones distintas a los dispositivos, pero no se consiguió su funcionamiento simultáneo.

Capítulo 5.
Conclusiones y líneas futuras

Este proyecto permite ver una aplicación de la inteligencia ambiental para un caso real. Utilizando un ordenador de placa reducida y diferentes sensores, es posible monitorizar la calidad de un ambiente de trabajo. Y las posibilidades no acaban ahí, ya que sería posible añadir al montaje actual mayor cantidad y variedad de sensores.

En un futuro, se podría mejorar la autonomía del dispositivo, lo que permitirá añadir nuevos dispositivos, con el display led con el que se cuenta pero que se desecha. El protocolo de comunicación entre Spark y servidor se podría ampliar, de manera que se pudieran conectar varios dispositivos. Así, se podrían tener varios dispositivos distribuidos por distintas zonas de un área de trabajo, y que todos se comunicaran con el mismo servidor centralizado.

Capítulo 6.
Summary and Conclusions

The project presented is a system that can measure the light and noise levels for a work environment. It relies in the Particle Spark Core, a source-board computer that uses several sensor to know how is the current status of the environment.

6.1 Hardware

The main part of the hardware is the Particle Spark Core which was previously cited. It is a SBC and it uses a WiFi module for communication. The Spark core is focused on the use of the Spark Cloud, a web service provided by the manufacturer.

There are two kinds of sensors that are attached to this computer for this project: Microphones and Light sensors. Microphones are able to capture the noise in the environment, and the light sensor can measure the light intensity. There are four of this microphones, which allow to know what area of the room where they are located is more or less noisy.

There is also a battery, which is used as a power supply for both the SBC and the sensors. Since the Spark Core has a WiFi module, and we are supplying it with a battery, it can be placed anywhere in a room, as long as it is a spot where the WiFi network can reach.

The SBC communicates with a server, installed on a PC, which can receive the information from the Spark and transform it to useful data through different software.

6.2 Software

It was developed to kinds of software: The software used by the Spark Core and the software used by the server.

The Spark Core was programmed to capture the noise and light during a period of time. At the end of each period, the SBC has to send it to the server, which will operate with that data to offer different results.

In the server, there are a software used to replace the manufacturer cloud service with a local cloud service, allowing us to work with the Spark Core without an internet connection. Also there is a LAMP server, a Web server which relies on a Linux OS, and uses MySQL as database management and PHP for different applications. Data sent by the SBC is stored in the MySQL database and transformed by PHP applications.

6.3 Results

The data from the SBC is processed for both the server and the Spark Core. The Spark Core handles the data transformation, from the raw readings of the sensors to the right units, dB and lux.

Once this data is transformed and sent to the server, it is stored in the database. PHP applications can work with this data. For example, the web server can create a graph showing how light and noise has changed for a period of time, or it can show, for a square room with one microphone in each corner, where the sound comes from.

6.4 Conclusion

This project allow to see a real application of ambient intelligence in a real case. Using a source board compute and several kind of sensors, it is possible to monitor the quality of a work environment. And the possibilities are much bigger, since there is the possibility of adding more sensors, that can measure other variables like temperature, or air quality.

Capítulo 7. Presupuesto

En este capítulo, se recogen el coste de cada componente utilizado en este proyecto. Los precios han sido obtenidos a partir del distribuidor Adafruit.

7.1 Presupuesto por componente y total

Componente - Descripción	Unidades	Precio / Ud	Precio total
SBC - Spark Core	1	39.95$	39,95$
Microfono - MAX4466	4	6.95$	27,80$
Sensor de luz - TSL2561	1	5.95$	5.95$
Batería Lithium Ion Battery Pack - 3.7V 6600mAh	1	29.50$	29.50$
Cargador - USB LiIon/LiPoly charger - v1.2	1	12.50$	12.50$
		TOTAL	**86,20$**

Tabla 7.1. Tabla de presupuesto por componente y total

Apéndice A - Código del Spark Core

A.1. Periodo de 5 minutos

```
/*****************************************************************************
 *
 * Application.cpp
 *
*****************************************************************************
 * Héctor Royo Concepción
 * 04/09/2015
 * Sistema de monitorización de ruido con 4 micrófonos y un sensor de luz. En esta versión, se envía la información
 * al servidor cada 5 minutos mediante el método HTTP-GET
 *****************************************************************************/

#include "application.h"
#include "math.h"
#include "SFE_TSL2561.h"
#include <string.h>

//Variables para la comunicación HTTP
TCPClient client;
byte server[]={192,168,1,183};

//Modo de funcionamiento del Spark Core, para control energético
SYSTEM_MODE(SEMI_AUTOMATIC);

//Variables para la medida del ruido
const int sampleWindow = 50;
unsigned int sample[6];
double volts[6];
double db[6];
unsigned long t_delay = 300000;
unsigned long t_post = 0;

double db_sumA = 0.00; double db_sumB = 0.00; double db_sumC = 0.00; double db_sumD = 0.00;

double out_db = 0.00;
double out_dbA = 0.00;  double out_dbB = 0.00;
double out_dbC = 0.00;  double out_dbD = 0.00;

//Asignación de Pins
int speakerA = A3;  int speakerB = A4;
int speakerC = A5;  int speakerD = A6;


//Variables para la medida de la luminosidad

unsigned int data1 = 0;  unsigned int data2 = 0;
double parcial = 0.00;
```

```
unsigned int ms = 0;
double lux = 0.00;
double out_lux = 0.00;

double lux_cont = 0.00;

unsigned int out_light = 0;

//Objeto para el sensor de luz
SFE_TSL2561 lsensor;

// Variables comunes
double cont = 0.00;

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

  Spark.variable("dbA",&out_dbA, DOUBLE);
  Spark.variable("dbB",&out_dbB, DOUBLE);
  Spark.variable("dbC",&out_dbC, DOUBLE);
  Spark.variable("dbD",&out_dbD, DOUBLE);

  Spark.variable("lux",&out_lux,DOUBLE);

  pinMode(speakerA, INPUT);
  pinMode(speakerB, INPUT);
  pinMode(speakerC, INPUT);
  pinMode(speakerD, INPUT);

      lsensor.begin();
      lsensor.setTiming(0,2,ms);
      lsensor.setPowerUp();
}

// Función para calcular el nivel de luminosidad
void getLux(double d){
      if ( (d > 0.00) && (d< 0.125) ){
         lux = (0.0304 - (0.0272 * d))*data1;
  } else if ( (d> 0.125) && (d< 0.250) ){
         lux = (0.0325 - (0.0440 * d))*data1;
  } else if ( (d > 0.250) && (d < 0.375) ){
         lux = (0.0351 - (0.0544 * d))*data1;
  } else if ( (d > 0.375) && (d < 0.50) ){
         lux = (0.0381 - (0.0624 * d))*data1;
  }  else if ( (d > 0.50) && (d < 0.61) ){
         lux = (0.0224 - (0.031 * d))*data1;
  }  else if ( (d > 0.61) && (d < 0.80) ){
         lux = (0.0128 - (0.0153 * d))*data1;
  } else if ( (d > 0.80) && (d < 1.30)){
         lux = (0.00146 - (0.00112 * d))*data1;
  }   else if ((d > 1.30)){
         lux = 0;
  }
 }
 //Bucle principal
void loop()
{
```

```
  unsigned long startMillis= millis();  // Control del tiempo

  unsigned int peakToPeak[6];  // Amplitud de sonido de pico a pico


  //Valores máximos y minimos de sonido
  unsigned int signalMax[6];    unsigned int signalMin[6];
  for (int i = 1; i < 6; i++)
  {
        signalMin[i] = 4096;
        signalMax[i] = 0;
  }

  // Captura de sonido
  while ((millis() - startMillis) < sampleWindow)
  {
    sample[1] = analogRead(speakerA);
    sample[2] = analogRead(speakerB);
    sample[3] = analogRead(speakerC);
    sample[4] = analogRead(speakerD);

    for (int i = 1; i < 6; i++)
    {
     if (sample[i] < 4096)
     {
       if (sample[i] > signalMax[i])
       {
         signalMax[i] = sample[i];
       }
       else if (sample[i] < signalMin[i])
       {
         signalMin[i] = sample[i];
       }
     }
    }
  }
  for(int i = 1; i < 6; i++)
  {
      peakToPeak[i] = signalMax[i] - signalMin[i];  // max - min = peak-peak amplitude
   volts[i] = (peakToPeak[i] * 3.3) / 4096;  // convert to volts
   db[i] = 115 - 20*log10((3.3*3.3)/(volts[i]*volts[i]));
  }

  db_sumA += db[1];  db_sumB += db[2]; db_sumC += db[3]; db_sumD += db[4];

  //Captura de luminosidad y cálculo de lux
  lsensor.getData(data1,data2);

  parcial = (data2*1.00000)/(data1*1.00000);
  getLux(parcial);
  lux_cont += lux;

cont++;

  if (startMillis > t_post + t_delay){
        // Conexión del módulo WiFi del Spark Core y conexión al servidor
        if(!WiFi.ready()){
              WiFi.on();
              WiFi.connect();
```

```
                Spark.connect();
                while(!WiFi.ready()){
                }
        }
        t_post = startMillis;
        // Cálculo de valores medios
        out_dbA = db_sumA/cont;  out_dbB = db_sumB/cont;  out_dbC = db_sumC/cont;
        out_dbD = db_sumD/cont;

        out_db = (out_dbA + out_dbB + out_dbC + out_dbD)/4;
        out_lux = lux_cont/cont;

        cont = 0.00;
        db_sumA = db_sumB = db_sumC = db_sumD = lux_cont = 0.00;
        //Comunicación HTTP
        if (client.connect(server, 80)){
                client.println("GET /spark/sparksays.php?noiseA=" + String(out_dbA) + "&noiseB=" +
String(out_dbB) + "&noiseC=" + String(out_dbC) + "&noiseD=" + String(out_dbD) + "&lux="+ String(out_lux) +"
HTTP/1.0");
                client.println("Content-Length: 0");
                client.println();
                }
                delay(1000);
    }
     else if(db[1] > 65 || db[2] > 65 || db[3] > 65 || db[4] > 65){ // Si se da algÃºn valor de ruido excepcionalmente alto,
se envÃa como alarma.
        // Conexión del modulo WiFi del Spark Core y conexión al servidor
        if(!WiFi.ready()){
                WiFi.on();
                WiFi.connect();
                Spark.connect();
                while(!WiFi.ready()){
                }
        }
        //Comunicación HTTP
        if (client.connect(server, 80)){
                client.println("GET /spark/sparksays.php?noiseA=" + String(db[1]) + "&noiseB=" + String(db[2]) +
"&noiseC=" + String(db[3]) + "&noiseD=" + String(db[4]) + "&lux="+ String(lux)+" HTTP/1.0");
                client.println("Content-Length: 0");
                client.println();
                }
                delay(1000);
    }
    else{
        // Desconexión del módulo wifi
        WiFi.off();
    }
  }
```

A.2. Periodo de 5 segundos

```
/*******************************************************************************
 *
 * Application.cpp
 *
*******************************************************************************
 * Héctor Royo Concepción
 * 04/09/2015
* Sistema de monitorización de ruido con 4 micrófonos y un sensor de luz. En esta versión,se actualiza la
* información cada 5 segundos, para ser solicitada por el servidor.
 *******************************************************************************/
#include "application.h"
#include "math.h"
#include "SFE_TSL2561.h"

//Variables para la medida del ruido
const int sampleWindow = 50;
unsigned int sample[6];
double volts[6];
double db[6];
unsigned long t_delay = 300000;
unsigned long t_post = 0;

double db_sumA = 0.00; double db_sumB = 0.00; double db_sumC = 0.00; double db_sumD = 0.00;

double out_db = 0.00;
double out_dbA = 0.00;  double out_dbB = 0.00;
double out_dbC = 0.00;  double out_dbD = 0.00;

//Asignación de Pins
int speakerA = A3;  int speakerB = A4;
int speakerC = A5;  int speakerD = A6;


//Variables para la medida de la luminosidad

unsigned int data1 = 0;  unsigned int data2 = 0;
double parcial = 0.00;
unsigned int ms = 0;
double lux = 0.00;
double out_lux = 0.00;

double lux_cont = 0.00;

unsigned int out_light = 0;

//Objeto para el sensor de luz
SFE_TSL2561 lsensor;

// Variables comunes
double cont = 0.00;

void setup()
{
  Serial.begin(9600);
```

```
  Spark.variable("dbA",&out_dbA, DOUBLE);
  Spark.variable("dbB",&out_dbB, DOUBLE);
  Spark.variable("dbC",&out_dbC, DOUBLE);
  Spark.variable("dbD",&out_dbD, DOUBLE);

  Spark.variable("lux",&out_lux,DOUBLE);

  pinMode(speakerA, INPUT);
  pinMode(speakerB, INPUT);
  pinMode(speakerC, INPUT);
  pinMode(speakerD, INPUT);

      lsensor.begin();
      lsensor.setTiming(0,2,ms);
      lsensor.setPowerUp();
}

// Función para calcular el nivel de luminosidad
void getLux(double d){
      if ( (d > 0.00) && (d< 0.125) ){
        lux = (0.0304 - (0.0272 * d))*data1;
  } else if ( (d> 0.125) && (d< 0.250) ){
        lux = (0.0325 - (0.0440 * d))*data1;
  } else if ( (d > 0.250) && (d < 0.375) ){
        lux = (0.0351 - (0.0544 * d))*data1;
  } else if ( (d > 0.375) && (d < 0.50) ){
        lux = (0.0381 - (0.0624 * d))*data1;
  }  else if ( (d > 0.50) && (d < 0.61) ){
        lux = (0.0224 - (0.031 * d))*data1;
  }  else if ( (d > 0.61) && (d < 0.80) ){
        lux = (0.0128 - (0.0153 * d))*data1;
  } else if ( (d > 0.80) && (d < 1.30)){
        lux = (0.00146 - (0.00112 * d))*data1;
  }   else if ((d > 1.30)){
        lux = 0;
  }
 }
 //Bucle principal
void loop()
{

  unsigned long startMillis= millis();  // Control del tiempo

  unsigned int peakToPeak[6];  // Amplitud de sonido de pico a pico


  //Valores máximos y minimos de sonido
  unsigned int signalMax[6];    unsigned int signalMin[6];
  for (int i = 1; i < 6; i++)
  {
      signalMin[i] = 4096;
      signalMax[i] = 0;
  }

  // Captura de sonido
  while ((millis() - startMillis) < sampleWindow)
  {
    sample[1] = analogRead(speakerA);
    sample[2] = analogRead(speakerB);
```

```
    sample[3] = analogRead(speakerC);
    sample[4] = analogRead(speakerD);

    for (int i = 1; i < 6; i++)
    {
     if (sample[i] < 4096)
     {
       if (sample[i] > signalMax[i])
       {
         signalMax[i] = sample[i];
       }
       else if (sample[i] < signalMin[i])
       {
         signalMin[i] = sample[i];
       }
     }
    }
  }
  for(int i = 1; i < 6; i++)
  {
      peakToPeak[i] = signalMax[i] - signalMin[i];  // max - min = peak-peak amplitude
   volts[i] = (peakToPeak[i] * 3.3) / 4096;  // convert to volts
   db[i] = 115 - 20*log10((3.3*3.3)/(volts[i]*volts[i]));
  }

  db_sumA += db[1];  db_sumB += db[2]; db_sumC += db[3]; db_sumD += db[4];

  //Captura de luminosidad y cálculo de lux
  lsensor.getData(data1,data2);

  parcial = (data2*1.00000)/(data1*1.00000);
  getLux(parcial);
  lux_cont += lux;

cont++;

  if (startMillis > t_post + t_delay){

        t_post = startMillis;
        // Cálculo de valores medios
        out_dbA = db_sumA/cont;  out_dbB = db_sumB/cont;  out_dbC = db_sumC/cont;
        out_dbD = db_sumD/cont;

        out_db = (out_dbA + out_dbB + out_dbC + out_dbD)/4;
        out_lux = lux_cont/cont;

        cont = 0.00;
        db_sumA = db_sumB = db_sumC = db_sumD = lux_cont = 0.00;
  }
}
```

Bibliografía

[1] Particle. https://www.particle.io/.

[2] Adafruit. https://www.adafruit.com/.

[3] LAMP Server. https://es.wikipedia.org/wiki/LAMP/.

[4] spark-cli. https://github.com/spark/particle-cli/.

[5] spark-server. https://github.com/spark/spark-server/.

[6] HTTP POST, GET. http://www.w3schools.com/tags/ref_httpmethods.asp/.

[7] MAX4466. http://www.adafruit.com/datasheets/MAX4465-MAX4469.pdf/.

[8] TSL2561. https://www.adafruit.com/datasheets/TSL256x.pdf/.