eGRU - Indicador online del estado de los contenedores para residuos urbanos.

Estudiante
Miguel Ángel Fernández Cela
Ingeniería Técnica de Informática de Sistemas
Sistemas empotrados

Consultor
Jordi Bécares Ferrés

Profesores responsables de la asignatura
Pere Tuset Peiró
Xavi Vilajosana Guillen

13 de junio de 2016

A mi esposa Yolanda y a mi hija Elena.

Gracias por vuestro apoyo y por el tiempo que me regaláis.

Página intencionadamente en blanco.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Indicador online del estado de los contenedores para residuos urbanos*
Nombre del autor:	*Miguel Ángel Fernández Cela*
Nombre del consultor:	*Jordi Bécares Ferrés*
Nombre del PRA:	*Pere Tuset Peiró* *Xavi Vilajosana Guillen*
Fecha de entrega (mm/aaaa):	*06/2016*
Titulación o programa:	*Ingeniería Técnica de Informática de Sistemas*
Área del Trabajo Final:	*Sistemas empotrados*
Idioma del trabajo:	*Castellano*
Palabras clave:	*IoT, SmartCities, Gestión de residuos.*

Resumen del Trabajo:

El desarrollo de Internet ha cambiado definitivamente nuestra manera de comunicarnos, de informarnos, de aprender, de entretenernos, de comprar, de hacer negocios y de un largo etcétera de actividades humanas. Es lo que se ha dado en llamar la *Internet de las personas*.

En los últimos años se está produciendo lo que algunos consideran una evolución de una primera etapa de Internet más centrada en las personas hacia una Internet centrada en las cosas o *'IoT',* término derivado del acrónimo inglés *Internet Of Things*.

A medida que la técnica lo va permitiendo, todo tipo de máquinas y dispositivos se van conectando a la red. Sensores cada vez más reducidos, más baratos y con menor consumo de energía, unido al desarrollo de las redes de telefonía sin hilos, permiten la expansión de Internet hasta lugares que hasta ahora eran inaccesibles.

La *smart city* o ciudad inteligente es un buen ejemplo de lo que estas nuevas tecnologías pueden lograr. Proyectos que abarcan áreas muy diversas,

pretenden garantizar a los ciudadanos una mejor calidad de vida y crecimiento económico mediante una gestión más eficiente de los servicios y recursos.

El presente trabajo puede englobarse dentro de uno de los proyectos pertenecientes al último grupo descrito. En él se estudia el comportamiento de un sistema prototipo que permite monitorizar, online y en tiempo real, los parámetros clave para optimizar la gestión del vaciado de los contenedores urbanos de residuos.

Desarrollaremos una unidad sensora que colocada en los contenedores y a través de una red Wi-Fi e Internet enviará los datos a un servidor central. Utilizando un navegador web estándar, se podrá consultar el estado de dichos contenedores para poder tomar las decisiones de actuación más adecuadas.

Abstract:

The development of the Internet has changed, once and for all, our way of communicating, obtaining information, learning, being entertained, buying, doing business, and many other human activities. This has come to be known as the *Internet of people*.

In recent years we have seen the beginning of what some consider to be an evolution from an initial stage of the Internet more focused on people toward an Internet focused on things, or the *'IoT',* short for the *'Internet of Things*'.

As technology advances, all kinds of machines and devices are gradually being connected to the Internet. Increasingly small and cheap sensors, which consume less energy and the development of wireless telephone networks make it possible to extend the Internet to places which until now were inaccessible.

The *'smart city*' is a good example of what these new technologies can achieve. Projects which cover very different areas aim to offer the citizens better quality of life and economic growth through more efficient management of services and resources.

This work forms part of one of the projects belonging to the last group described. It studies the behaviour of a prototype system which can monitor,

online and in real time, the key parameters to be able to optimize the management of the emptying of urban waste containers.

We will develop a sensor unit which, when placed in the containers, will send the data to a central server via a Wi-Fi network and the Internet. On using a standard web browser, it will be possible to consult the state of these containers in order to be able to make decisions on the most appropriate action.

Página intencionadamente en blanco.

Índice de contenidos

Índice de contenidos

Índice de contenidos

Índice de contenidos

Índice de figuras

Índice de figuras

Índice de tablas

Página intencionadamente en blanco.

1. Introducción

1.1 Contexto y justificación del trabajo

El desarrollo del presente proyecto, surge a partir de que se detecta la necesidad que tienen las empresas dedicadas a la recogida de residuos urbanos de conocer, en tiempo real, el estado de los contenedores de dichos residuos para poder mejorar los costes y la calidad del servicio que prestan.

Para ello es necesario disponer en todo momento de información relacionada con parámetros clave como el nivel de llenado de los contenedores, su inclinación, la temperatura en su interior y el número de veces que han sido vaciados a lo largo de su vida útil.

La detección del nivel de llenado permite optimizar las rutas de recogida, de forma que cada contenedor se vacíe cuando realmente sea necesario, en función de dicho nivel y del tipo de residuo que contenga. De esta manera, por una parte baja el coste económico de la recogida y por otra se evita el desbordamiento con el consiguiente beneficio higiénico.

Una alarma generada a partir del ángulo de inclinación del contenedor, permite saber si éste ha perdido su verticalidad debido al viento, vandalismo u otras causas. Ello facilita una actuación temprana del servicio de recogida para restaurar su posición y limpiar el posible contenido vertido, como consecuencia de lo cual, el contenedor sigue siendo utilizable y se mejora la higiene de la zona.

La temperatura en el interior del contenedor, y sobre todo una alarma que permita saber si el contenido y/o el contenedor están ardiendo, posibilita la rápida intervención de los servicios municipales para evitar la propagación del fuego a otros contenedores contiguos, árboles, mobiliario urbano, edificios, etc.

Finalmente, es importante conocer el número de descargas realizadas de cada contenedor. Los periodos de mantenimiento vienen marcados por la cantidad de veces que se ha vaciado el contenedor, debido a que es el momento en el que sufre un mayor desajuste y estrés mecánico. El dato acumulado por cada contenedor permite optimizar el calendario de mantenimiento y en último término alargar su vida útil.

En este proyecto desarrollaremos un sistema experimental, orientado a satisfacer las necesidades de información en tiempo real descritas anteriormente. Al sistema lo hemos llamado **eGRU**, "**G**estión de **R**esiduos **U**rbanos" con la 'e' inicial haciendo referencia a su naturaleza 'online'.

El producto resultante, en su forma comercial, va dirigido a los responsables de planificación y mantenimiento de las empresas de recogida subcontratadas por los ayuntamientos, así como a los responsables de urbanismo de los propios ayuntamientos.

Adicionalmente, a partir de los datos que genera el sistema, los ayuntamientos podrían poner a disposición de los ciudadanos información útil para ellos como por ejemplo el nivel de llenado actual y la fecha-hora planificada para la recogida. Este servicio no forma parte del proyecto y se presenta aquí solamente como ejemplo de expansión del sistema a otros posibles usuarios.

1.2 Descripción

Cada contenedor debe incorporar necesariamente un conjunto de sensores destinados a obtener las magnitudes físicas que permitan generar los datos apuntados en el apartado anterior. Además, para traducir y gestionar la información proporcionada por los sensores, de forma que pueda ser útil, usaremos un microcontrolador. Finalmente, incorporaremos un módulo que permita enviar la información generada a un sistema remoto.

Al conjunto de elementos anteriores, en lo sucesivo le denominaremos *'unidad sensora'* y forma parte de lo que conocemos como un sistema empotrado.

En el sistema experimental que desarrollaremos, la comunicación entre la unidad sensora y el host se realizará a través de una red wifi doméstica e Internet. Como sistema remoto utilizaremos un servidor web compartido de una empresa que proporciona servicios de alojamiento. Será el encargado de recibir los datos que envíe la unidad sensora, de enviar a ésta la información relativa a los ajustes y de servir las peticiones que hagan los clientes mediante la interfaz que programaremos.

La interacción de los usuarios con el sistema, se llevará a cabo vía un navegador web estándar instalado en el dispositivo de acceso que se utilice.

Completaremos el trabajo con un estudio de viabilidad técnica y económica de un posible sistema comercial, basado en los resultados de las pruebas llevadas a cabo con el prototipo realizado.

1.3 Objetivos

El sistema desarrollado ha de ser capaz de:

- Monitorizar el nivel de llenado.
- Monitorizar la temperatura.
- Monitorizar la verticalidad.
- Monitorizar el número de descargas acumuladas por el contenedor.
- Disparar las alarmas correspondientes a las magnitudes controladas (nivel de llenado, temperatura, verticalidad y descargas acumuladas por el contenedor).
- Mostrar online y en tiempo real los datos anteriores.

- Permitir la edición online de los ajustes del sistema (niveles de disparo de las alarmas, intervalos de refresco de la información, etc.).
- Ofrecer la opción de enviar un correo electrónico, a las direcciones previstas para tal fin, cuando se active o desactive cualquiera de las alarmas anteriores.
- Conseguir el menor consumo posible de la unidad sensora, con el fin de optimizar el tiempo de autonomía de las pilas con las que se alimenta.

1.4 Enfoque

En este apartado vamos a indicar el método seguido para la elaboración del conjunto del proyecto y para el caso particular de las aplicaciones software desarrolladas como parte del mismo.

1.4.1 Enfoque del proyecto

Para la realización de este trabajo se ha seguido un proceso, generalmente conocido como ciclo de vida de un proyecto, basado en un patrón que incluye cinco fases:

1. *Iniciación.*

   Esta primera fase ha sido una etapa de aprendizaje y familiarización con los elementos hardware y las herramientas de desarrollo que vamos a utilizar.

   Paralelamente, junto con el director del proyecto, se ha definido el tema sobre el que versará el trabajo, a partir de que se detecta la demanda real en el mercado de un producto del tipo que desarrollaremos. Se han definido los objetivos (apartado 1.3) y se ha realizado un estudio previo de viabilidad técnica (capítulo 5) para determinar si es posible afrontar el problema dentro del marco delimitado por la asignatura, en cuanto a recursos físicos o hardware y lógicos o software que serán necesarios (apartado 1.7).

2. *Planificación.*

   Una vez superada la etapa anterior, se ha realizado un listado de tareas que nos ha permitido definir la planificación inicial mostrada en la Figura 1.1.

   Debido a que hemos tenido que ir adaptando y redefiniendo algunas tareas en la fase de ejecución, la planificación ha sido iterativa, obteniendo finalmente el resultado mostrado en el diagrama de la Figura 1.2.

3. *Ejecución.*

   En esta fase hemos puesto en marcha todo aquello que hemos planificado en la fase anterior. Ha sido la fase más larga de todas. Los capítulos 3, 4, 6 y 7, hacen referencia en su mayoría al trabajo desarrollado en este periodo.

4. *Control.*

   En paralelo con la fase de ejecución, se han ido analizando las desviaciones en el tiempo que se iban produciendo respecto de la planificación inicial y se han tomado las medidas necesarias en cada caso para tratar de minimizar dichas desviaciones.

   También hemos ido controlando, a medida que avanzábamos con la ejecución, que no nos desviábamos de los objetivos del proyecto.

5. *Finalización o cierre.*

   En nuestro caso, el cierre del proyecto lo marca la fecha prevista para la entrega final.

1.4.2 Enfoque de las aplicaciones

En las tareas correspondientes a la realización de software, se ha seguido el método de desarrollo en cascada debido a que los objetivos son claros y además conocemos de antemano los detalles de la solución en cuanto a requisitos y tecnología.

Siguiendo el método, hemos ido pasando progresivamente por cada una de las siguientes etapas:

1. *Requisitos.*

   Para elaborar la definición de requisitos nos hemos basado en los objetivos del proyecto (apartado 1.3).

2. *Análisis y diseño.*

   En esta etapa se ha definido cómo debe de comportarse el software, tanto desde el punto de vista funcional como técnico (capítulos 3 y 4).

   Se han definido las estructuras de los datos, la arquitectura o bloques de software necesarios y sus interdependencias, así como la interfaz de usuario en el caso del servidor.

   Hemos tenido en cuenta los detalles que puedan facilitar el mantenimiento posterior del sistema, durante la vida útil del producto.

3. *Implementación.*

   Hemos implementado el código según el análisis y diseño realizados en el punto anterior, hemos confeccionado el manual del usuario (anexo 11.1) y hemos compilado y generado los ejecutables.

4. *Pruebas.*

   Se han ido realizando pruebas parciales con la unidad sensora y con el servidor, primero de forma individual y a medida que se progresaba en el desarrollo, de forma conjunta. Finalmente se han realizado las pruebas definitivas del sistema completo.

1.5 Planificación del trabajo

1.5.1 Planificación inicial

Para confeccionar la planificación inicial hemos dividido el proyecto en cuatro fases temporales y cada una de ellas la hemos dividido, a su vez, en varias tareas y sub-tareas que nos permiten seguir el proceso con mayor nivel de detalle.

La primera fase, dedicada a preparar los recursos no disponibles inicialmente, ya se había completado antes de diseñar la planificación. Se indica como referencia (texto en color gris).

Fase 1: Preparación de recursos no disponibles.

T1.1: Pedir material hardware

T1.2: Instalar software

T1.3: Montar hardware

Fase 2: Versión previa.

Durante la segunda fase se realiza el análisis y diseño del software del sistema completo, se implementa y se prueba tanto en la unidad sensora como en el servidor.

También se hace un esqueleto de la memoria y se van consolidando en la misma los avances que se van produciendo en la realización del proyecto.

Esta fase contiene un hito importante, marcado por la fecha de entrega del código y de la memoria previa, para su seguimiento por parte del director del proyecto.

Fase 3: Versión final.

En este periodo se modifica el software y se revisa la memoria según las indicaciones hechas por el director del proyecto en la entrega de seguimiento, se realizan las pruebas definitivas de funcionamiento del sistema completo y se continúa elaborando la memoria del proyecto.

La fase termina con la entrega final del código.

Fase 4: Memoria y presentación.

Fase final dedicada a revisar y completar los últimos elementos entregables, que marcan los hitos finales y cierran el proyecto.

1.5.2 Planificación final

Aunque se ha intentado a lo largo del desarrollo del proyecto respetar los plazos marcados en la planificación inicial, no ha sido posible debido a dos hechos:

1– Para completar el código de la unidad sensora se han necesitado 19 días en contra de los 14 días planificados inicialmente.

2– De forma similar, el código de la parte del servidor ha requerido de 23 días frente a los 14 días previstos originalmente.

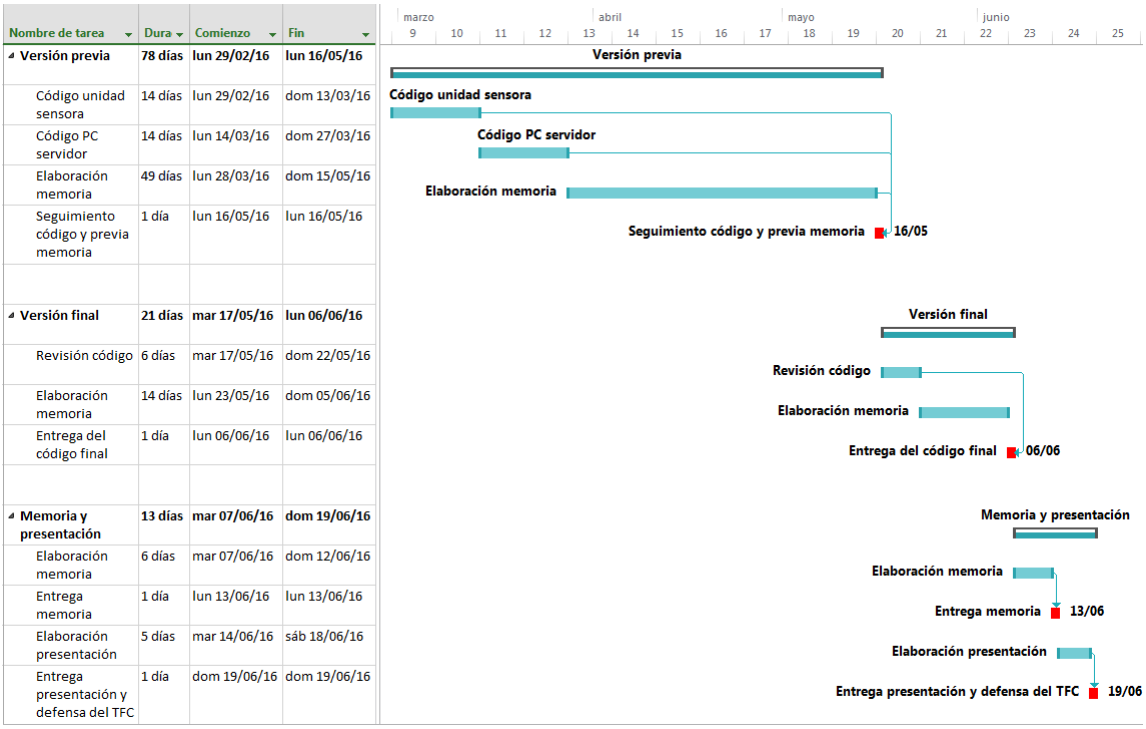
La diferencia tan abultada entre el tiempo programado y el real, denota claramente que la planificación inicial se hizo mal para estas dos etapas.

Tal y como habíamos previsto en el análisis de riesgos (apartado 1.6), ante el retraso acumulado, podíamos evitar programar en el servidor la parte dedicada al envío de los e-mails con la información de las alarmas.

Sin embargo, se observó que el desarrollo de dicha parte solo iba a variar el tiempo necesario en un día y se optó por incluirla también. De esta forma hemos dado prioridad a la consecución plena de los objetivos, frente al día adicional de retraso que ello ha supuesto.

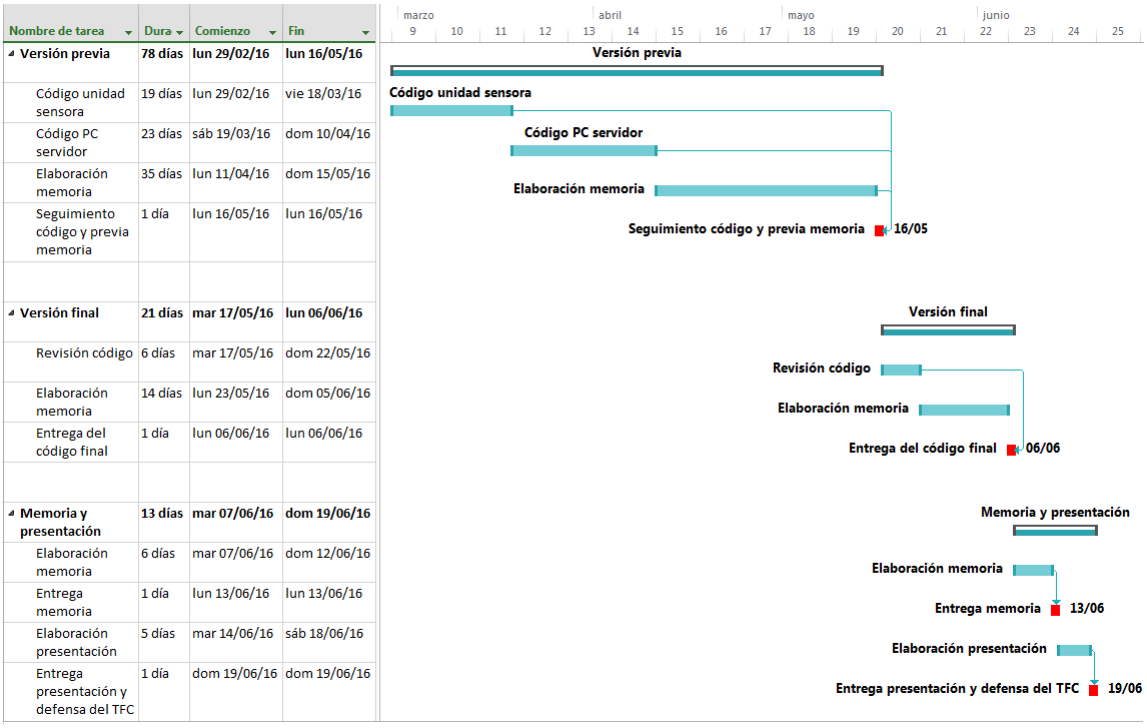
Hechas las aclaraciones anteriores, la única forma de terminar en plazo el proyecto, ha sido aumentar el promedio de horas diarias dedicadas a elaborar la presente memoria, que es la tarea sobre la que estaban repercutiendo los retrasos acumulados, como puede observarse en los cronogramas.

Las Figuras 1.1 y 1.2, muestran respectivamente el cronograma con la planificación inicial del proyecto y el cronograma final resultante.



Nombre de tarea	Dura	Comienzo	Fin
Versión previa	**78 días**	**lun 29/02/16**	**lun 16/05/16**
Código unidad sensora	14 días	lun 29/02/16	dom 13/03/16
Código PC servidor	14 días	lun 14/03/16	dom 27/03/16
Elaboración memoria	49 días	lun 28/03/16	dom 15/05/16
Seguimiento código y previa memoria	1 día	lun 16/05/16	lun 16/05/16
Versión final	**21 días**	**mar 17/05/16**	**lun 06/06/16**
Revisión código	6 días	mar 17/05/16	dom 22/05/16
Elaboración memoria	14 días	lun 23/05/16	dom 05/06/16
Entrega del código final	1 día	lun 06/06/16	lun 06/06/16
Memoria y presentación	**13 días**	**mar 07/06/16**	**dom 19/06/16**
Elaboración memoria	6 días	mar 07/06/16	dom 12/06/16
Entrega memoria	1 día	lun 13/06/16	lun 13/06/16
Elaboración presentación	5 días	mar 14/06/16	sáb 18/06/16
Entrega presentación y defensa del TFC	1 día	dom 19/06/16	dom 19/06/16

Figura 1.1: Cronograma inicial del proyecto.



Nombre de tarea	Dura	Comienzo	Fin
Versión previa	**78 días**	**lun 29/02/16**	**lun 16/05/16**
Código unidad sensora	19 días	lun 29/02/16	vie 18/03/16
Código PC servidor	23 días	sáb 19/03/16	dom 10/04/16
Elaboración memoria	35 días	lun 11/04/16	dom 15/05/16
Seguimiento código y previa memoria	1 día	lun 16/05/16	lun 16/05/16
Versión final	**21 días**	**mar 17/05/16**	**lun 06/06/16**
Revisión código	6 días	mar 17/05/16	dom 22/05/16
Elaboración memoria	14 días	lun 23/05/16	dom 05/06/16
Entrega del código final	1 día	lun 06/06/16	lun 06/06/16
Memoria y presentación	**13 días**	**mar 07/06/16**	**dom 19/06/16**
Elaboración memoria	6 días	mar 07/06/16	dom 12/06/16
Entrega memoria	1 día	lun 13/06/16	lun 13/06/16
Elaboración presentación	5 días	mar 14/06/16	sáb 18/06/16
Entrega presentación y defensa del TFC	1 día	dom 19/06/16	dom 19/06/16

Figura 1.2: Cronograma final del proyecto.

1.6 Análisis de riesgos

Para la finalización en plazo del presente trabajo, detectamos dos factores de riesgo que podrían influir negativamente:

1 – Para medir y optimizar los consumos de la unidad sensora en distintas circunstancias, es necesario hacer bastantes manipulaciones del hardware (desoldar leds, abrir y cerrar pistas, hacer mediciones en pines muy próximos, etc.). Estas operaciones podrían provocar un fallo definitivo de algún elemento que generaría el consiguiente retraso en el proyecto.

Hemos identificado los módulos 1 a 7 (ver Figuras 1.3 y 1.4), como posibles generadores de problemas y hemos optado por hacernos con una segunda unidad de cada uno de ellos.

También hemos decidido conectar dichos módulos a la placa base a través de zócalos de inserción, para disminuir el tiempo de sustitución en caso necesario.

Adicionalmente, nos hemos hecho con varias pilas de alimentación para facilitar un cambio instantáneo en caso de agotamiento de la pila activa.

2 – La poca experiencia con las herramientas de desarrollo del lado del servidor (Apache, mySQL y PHP) podrían alargar demasiado el tiempo dedicado a la programación de la funcionalidad del mismo.

Como medida de atenuación en caso de que surjan dificultades, reservamos la posibilidad de no completar la parte de envío de alarmas a través del correo electrónico. Esta tarea no es esencial para llevar a cabo el estudio que nos proponemos y figura entre los objetivos solamente para mejorar la percepción real de las posibilidades del sistema.

1.7 Recursos utilizados

La realización del proyecto ha obligado a utilizar una diversidad de recursos, de los cuales hacemos una breve introducción a continuación. Los vamos a dividir en tres grupos: hardware, software y el resto de recursos que incluiremos bajo el epígrafe “otros recursos”.

Un primer recurso importante, que no clasificaremos en los siguientes grupos por ser de carácter general, ha sido la Wiki de la asignatura. Se han hecho frecuentes consultas, principalmente en la fase inicial, sobre temas relacionados con el hardware, software y demás documentación a la que proporciona acceso.

Otro recurso constantemente utilizado ha sido todo el material bibliográfico procedente de libros y consultas en la Web al que hacemos referencia en el capítulo diez.

1.7.1 Hardware

En las dos imágenes siguientes (Figuras 1.3 y 1.4) se muestra la placa montada para hacer el estudio del prototipo, con los bloques principales enumerados y comentados a continuación.

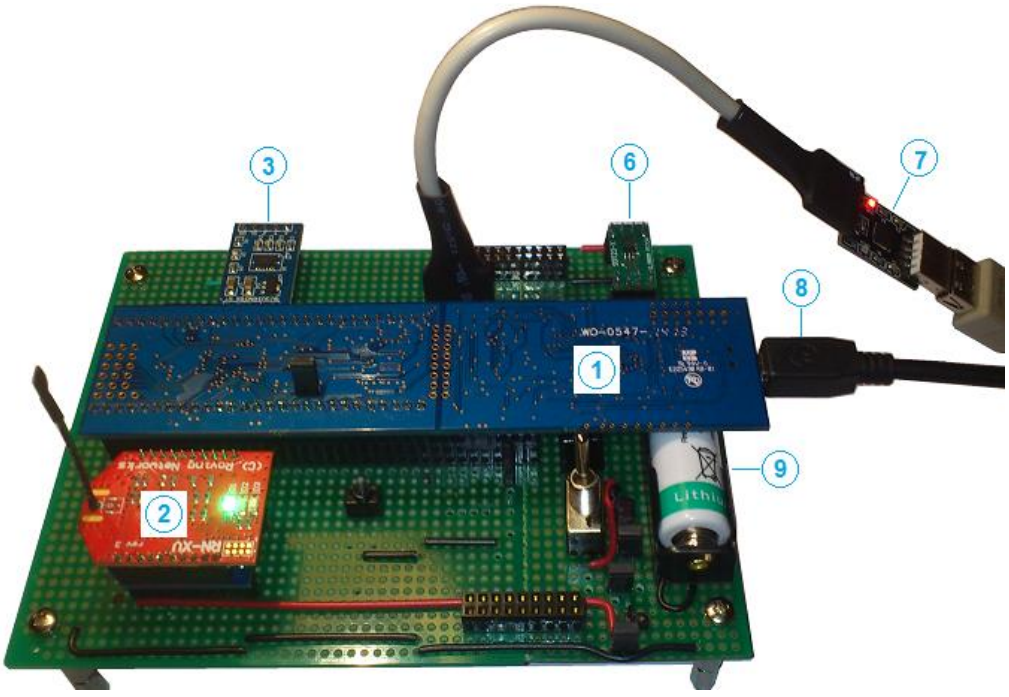


Figura 1.3: Placa prototipo, vista superior.

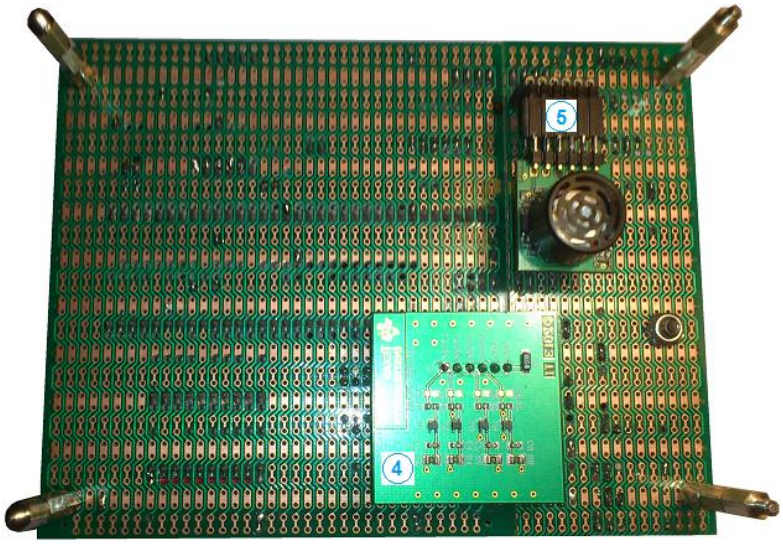


Figura 1.4: Placa prototipo, vista inferior.

1 – Placa de desarrollo del microcontrolador LPC1769 [W1]:

Fabricado por NXP Semiconductors, el microcontrolador LPC1769 está basado en la arquitectura ARM Cortex-M3 y puede trabajar hasta a 120MHz de frecuencia de CPU.

Incorpora 512 kB de memoria Flash, 64kB de memoria RAM, y un conjunto muy completo de periféricos, entre los cuales destacamos, por ser los que nosotros utilizamos: 52 GPIO, 1 ADC de 12 bits que multiplexa hasta ocho canales, 4 UARTs, 1 controlador DMA de ocho canales, y 1 RTC con alimentación independiente de la principal. Además, tiene cuatro modos de trabajo con bajo consumo.

En nuestro caso, el microcontrolador se encarga de la lectura de los sensores, del tratamiento de los datos leídos y de la gestión de la comunicación con el servidor a través del módulo Wifly. También se encarga de poner a todos los elementos de la unidad sensora, incluido a sí mismo, en modo de bajo consumo y de recuperarles de este modo en los periodos de trabajo activo.

2 – Módulo WiFly RN171 [W2]:

A través de su UART, se comunica con una de las UART del LPC1769. De esta manera el microcontrolador envía y recibe los datos desde el host vía la interfaz wifi del WiFly y su protocolo TCP/IP. En nuestro caso, utilizamos la red wifi y el modem-rúter domésticos para la salida a Internet.

3 – Acelerómetro MMA7361 [W3]:

A través de las salidas analógicas de la aceleración ‘g’ en cada uno de los tres ejes cartesianos y con los cálculos correspondientes, el microcontrolador conocerá los ángulos de inclinación del contenedor, permitiendo de esta manera saber si el contenedor está volcado o no.

4 – Sensor de temperatura LMT87 [W4]:

Utilizaremos la placa de evaluación de este IC para conectarla a nuestro sistema. Entrega una señal analógica proporcional a la temperatura que mediremos vía una entrada analógica del micro.

5 – Detector de distancia por ultrasonidos PmodMaxSonar [W5]:

Este sensor es el encargado de medir la distancia que queda disponible entre la parte superior del contenedor y los residuos que contiene.

Dispone de una señal digital de entrada que le indica cuándo hacer una medida. Las medidas realizadas las envía a través de su UART de solo transmisión.

6 – Interruptor de estado sólido FPF2104 [W7]:

Comandado por el micro, se encarga de dar o cortar alimentación al detector de distancia. De esta manera, el detector está alimentado en los periodos de trabajo activo y no tiene consumo en los periodos en los que la unidad sensora está en modo de ahorro de energía.

7 – Adaptador UART-USB CP2102 [W8]:

Es un conversor bidireccional de señal entre puertos UART y USB. Lo utilizamos para las conexiones entre las UARTs del microcontrolador o del módulo WiFly y el puerto USB del PC. Solo es necesario durante la fase de desarrollo y no forma parte, en sí mismo, del sistema.

8 – Interfaz de programación:

Conectado a un puerto USB del PC de desarrollo, permite programar y debugar el microcontrolador.

9 – Pila de alimentación:

Alimenta a todos los componentes de la unidad sensora con una tensión de 3.3 V y tiene una capacidad de 2600 mAh.

PC:

También dentro del capítulo de hardware, disponemos de un PC, asociado a la red wifi doméstica, con el que desarrollaremos y probaremos el sistema.

Funcionará a la vez como máquina de desarrollo y como PC de usuario para interactuar con el sistema, a través del navegador web que tiene instalado.

El PC incorpora el sistema operativo Windows 7. Todas las aplicaciones de desarrollo con las que trabajamos, y que describimos brevemente a continuación, se ejecutan sobre dicho sistema operativo.

1.7.2 Software

IDE LPCXpresso [W9]:

Entorno de programación integrado para las distintas familias de microcontroladores de *NXP Semiconductors*. Incluye compilador, ensamblador y enlazador para los lenguajes 'C', que es el que nosotros empleamos, y 'C++'. Versión utilizada: 7.6.2.

Apache [W10]:

Servidor web HTTP de código abierto. Le utilizamos para crear las páginas y servicios web en el servidor. Versión utilizada: 2.4.12.

MySQL [W11]:

También de código abierto, MySQL es un gestor de bases de datos relacional, multihilo y multiusuario. La usamos en el servidor para organizar y almacenar los datos persistentes. Versión utilizada: 5.6.23.

MySQL Workbench [W12]:

Es una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración, diseño, creación y mantenimiento para el sistema de base de datos MySQL. Versión utilizada: 6.3.

PHP [W13]:

Nos servimos de este este lenguaje de programación, de uso general y de código del lado del servidor, para gestionar el contenido dinámico en el servidor web. Versión utilizada: 5.6.6.

NetBeans [W14]:

Entorno integrado de desarrollo que permite trabajar con varios lenguajes de programación. En este caso lo utilizamos para agilizar la programación con el lenguaje PHP. Versión utilizada: 8.0.2.

FileZilla Client [W15]:

Cliente FTP de código abierto con el que subimos y bajamos archivos del servidor remoto. Versión utilizada: 3.18.0.

PuTTY [W16]:

Es un emulador de terminal con soporte para varios protocolos. Lo utilizamos durante la fase de desarrollo como sistema de test para enviar y recibir comandos desde el módulo WiFly, y también a modo de debugger durante la ejecución del programa del microcontrolador. Versión utilizada: 0.63.0.0.

A las aplicaciones anteriores, instaladas en el PC, podemos añadir los dos bloques software siguientes, utilizados como soporte en la programación de la unidad sensora:

Sistema operativo FreeRTOS [W17]:

Es un kernel o planificador en tiempo real sobre el cual se pueden construir aplicaciones que necesitan ejecutar tareas cuasi-simultáneamente en periodos estrictos de tiempo.

Gratuito y de código abierto, ocupa muy poco espacio en la memoria del programa y es ampliamente utilizado en sistemas empotrados, debido a la baja capacidad de procesamiento y memoria de los que estos últimos suelen disponer. Versión utilizada: 7.0.1.

Librería CMSISv2p00_LPC17xx:

La librería ha sido desarrollada por NXP para su familia de microcontroladores LPC17xx según el estándar marcado por ARM para los procesadores basados en la arquitectura Cortex-M3. Proporciona una capa de abstracción del hardware, facilitando el trabajo de interfaz entre el micro y los periféricos. Nos hemos apoyado en ella para hacer la programación del microcontrolador. Versión utilizada: 2.01.

1.7.3 Otros recursos

Además de los recursos citados, también hemos utilizado otros de carácter más genérico que hemos dedicado a tareas como el montaje de la placa de la unidad sensora, pruebas del hardware, mediciones de consumos, etc.

De entre todos ellos, destacamos los siguientes:

- Placas de montaje, cables, zócalos de conexión y herramientas de uso general.
- Soldador y estaño.
- Multímetro
- Osciloscopio

1.8 Productos obtenidos

El producto obtenido a la finalización del proyecto es un sistema experimental orientado a mejorar la gestión y capacidad de respuesta del servicio de recogida de residuos urbanos. Está compuesto por una unidad sensora o sistema empotrado autónomo y un host remoto.

La unidad sensora, una vez instalada en cada contenedor de residuos, se encarga de obtener las variables físicas necesarias, de tratarlas y de transmitirlas al host.

El host organiza los datos recibidos desde la unidad sensora y los pone a disposición del usuario para que, a través de un navegador web, pueda consultar la información y actuar en consecuencia.

1.9 Descripción del resto de capítulos

- Capítulo 2:

  Estado actual de la tecnología utilizada y estudio del mercado en cuanto a productos similares al desarrollado.

- Capítulo 3:

  Descripción funcional del conjunto. Decisiones de diseño adoptadas.

- Capítulo 4:

  Descripción técnica detallada de los apartados introducidos en el capítulo anterior.

- Capítulo 5:

  Estudio de viabilidad técnica del sistema desarrollado. Puntos fuertes y débiles.

- Capítulo 6:

  Valoración económica. Comparativa con productos similares.

- Capítulo 7:

  Estudio de la conversión del sistema desarrollado en un producto comercial. Modificaciones y pautas a seguir.

- Capítulo 8:

  Conclusiones, nivel de cumplimiento de los objetivos y autoevaluación.

- Capítulo 9:

  Glosario de términos utilizados.

- Capítulo 10:

  Bibliografía: textos consultados y referencias web.

- Capítulo 11:

  Anexos del proyecto.

1. Introducción

Página intencionadamente en blanco.

2.1 Estado del arte

En un mercado emergente como es el del *'Internet of Things'* (en adelante *'IoT'*), prácticamente cada día surgen nuevos elementos orientados a facilitar la construcción de dispositivos que den servicio a dicho mercado.

Desde el punto de vista del hardware, nuevos chips con mayor nivel de integración, menor consumo y más baratos, se suceden rápidamente en la escena actual.

En cuanto al software, protocolos que tienden a ser cada vez más livianos (simplicidad de recursos hardware y consumo de energía), así como más sólidos y seguros (solidez y seguridad del conjunto del sistema), compiten entre sí por hacerse hueco en este mercado.

Así las cosas, no es extraño que un dispositivo desarrollado solamente hace un año, quede obsoleto y se vea superado por otro que incorpore las últimas tecnologías que le permiten mejorar una o varias de las características hardware/software señaladas anteriormente, con una repercusión final positiva sobre el precio, consumo, solidez y / o seguridad.

2.1.1 Hardware

Debido a la necesidad de espacio y consumo reducidos, además de conectividad, en la mayoría de los casos los dispositivos utilizados en el mundo del *'IoT'* están soportados a nivel de hardware por lo que se conoce como un sistema empotrado.

Los sistemas empotrados se diseñan para cubrir necesidades específicas, pero todos parten de un conjunto de elementos o subsistemas con características comunes que mencionamos a continuación.

- Microcontrolador:

  Es el 'cerebro' del sistema. Se responsabiliza de hacer las operaciones principales encaminadas a realizar una tarea determinada y de dirigir el funcionamiento del resto de elementos que le rodean.

- Dispositivos de entrada o sensores:

  Son los encargados de capturar las magnitudes físicas y de ponerlas a disposición del microcontrolador para su procesado.

  Existen multitud de tipos de sensores de acuerdo a las magnitudes físicas que pueden capturar.

  Podemos poner como ejemplo, por ser ampliamente utilizados, los destinados a capturar el estado de un interruptor, el valor de la temperatura, de la distancia, de la intensidad de luz, la presencia de gases o los de tipo inercial que pueden leer valores de aceleración, velocidad angular o fuerza magnética.

- Dispositivos de salida o actuadores:

  Dan el formato adecuado a las magnitudes físicas de salida, una vez que éstas han sido establecidas por el micro como consecuencia de la tarea que está ejecutando.

  Ejemplos de ellos son los relés, displays, leds, electroválvulas, etc.

- Conectividad cableada:

  Puede dividirse en dos grupos, aquella que permite el intercambio de datos entre dispositivos internos del sistema y la que permite el intercambio de datos entre el sistema y el mundo exterior.

  En el primer grupo podemos situar, como ejemplo, las conexiones basadas en I2C, SPI o UART. En el segundo grupo estarían las conexiones basadas en RS232, RS485, USB, CAN o Ethernet.

- Conectividad sin hilos:

  Subsistema dedicado al intercambio de información entre el sistema empotrado y otros sistemas exteriores, sin necesidad de cableado entre ambos.

  Ejemplos de este tipo de conectividad son las redes que utilizan como base el NFC, Wifi, Bluetooth, Zig-Bee o las propias redes de telefonía sin hilos.

- Gestión de la energía:

  Todos los elementos anteriores necesitan energía para su funcionamiento. En el mundo del *'IoT'*, esta energía se suministra generalmente de forma autónoma, por ejemplo, mediante pilas, baterías, células fotovoltaicas e incluso recogiendo y almacenando la energía producida por las ondas de radiofrecuencia o la generada por las vibraciones procedentes del entorno del sistema.

  En cualquier caso, la gestión de la energía en el *'IoT'* es una faceta fundamental. Cuanto más se alargue en el tiempo la autonomía del sistema, menores serán los costes de reemplazo del elemento que le alimenta.

Existen en el mercado varios dispositivos que pueden ayudar en las fases de desarrollo de un sistema empotrado. Algunos de los más populares son los basados en Arduino, Raspberry Pi o BeagleBone.

Además, la mayoría de fabricantes de semiconductores, ofrecen sus propios elementos de desarrollo que permiten construir un sistema embebido y hacer pruebas basadas en la interconexión de diferentes bloques hardware. En el anexo 11.2, mostramos una lista con enlaces a algunos de los fabricantes más importantes de semiconductores en su actividad relacionada con el '*IoT*'.

Para la realización de este trabajo hemos optado por la construcción de un sistema empotrado de desarrollo, basado en el material hardware que se utiliza en la asignatura y añadiendo algunos elementos específicos para cubrir las necesidades del proyecto.

Entre los materiales propios de la asignatura se encuentran el microprocesador, el sensor de aceleración y el WiFly, que es el elemento que permite la conectividad sin hilos entre nuestro sistema y el servidor remoto. A los componentes anteriores, hemos añadido un sensor de temperatura y otro de distancia. Todos estos elementos ya los hemos descrito en el apartado 1.7.1, dedicado al hardware del proyecto.

El sensor de temperatura que hemos utilizado ya le conocíamos antes de la realización de este trabajo y le hemos elegido porque es uno de los sensores con menor consumo actualmente en el mercado (5.4 uA), además de tener una precisión (+/- 0.3ºC) y rango de temperatura (-50 a 150 ºC) que exceden las necesidades del desarrollo que nos proponemos.

Para la medición de la distancia, hemos comparado las características de dos sensores que son relativamente populares y fáciles de conseguir. El SRF01 fabricado por Devantech y el PmodMaxSonar fabricado por Digilent y basado en el LV-MaxSonar-EZ1 de MaxBotix. La Figura 2.1 muestra el aspecto de ambos sensores.

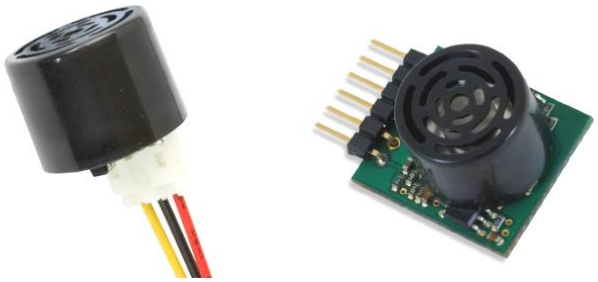


Figura 2.1: Sensores de distancia.
Devantech SRF01 (izquierda) y PmodMaxSonar (derecha).

Como se aprecia en la Tabla 2.1, los dos sensores tienen características similares excepto por lo que se refiere a la corriente de alimentación y el tiempo que tardan en hacer la primera medida. Estos dos factores son importantes debido a su influencia sobre el consumo.

	Vcc	Rango	Resolución	Comunicación	Ultrasonido	Consumo ranging / wait / sleep	Primera medida	Comp. Temp.
SRF01	3.3 - 5.5V	18 - 600 cm.	1 cm.	- UART con Tx/Rx de un solo pin	8 pulsos a 40 Khz.	25mA / 11mA / 55uA	aprox. 160 ms. (también después de wake-up)	No
MaxSonar-EZ1	3.3V	15 - 648 cm.	2.54 cm.	- Pulso + - UART Tx - Analógica - PWM	13 pulsos a 42 Khz.	2.6 mA media / No sleep	aprox. 350 ms	No

Tabla 2.1: Comparativa sensores de distancia SRF01 y MaxSonar-EZ1.

El EZ1 no tiene posibilidad de pasar a modo *'sleep'* como el SRF01. No obstante, incluso un consumo constante en modo *'sleep'* de 55uA puede ser elevado, si como pretendemos, la autonomía de la alimentación ha de extenderse a lo largo de diez años.

En vista de lo anterior, una primera decisión que tomamos fue la de cortar totalmente la alimentación al sensor durante los periodos de espera. En este caso, la ventaja que podría suponer el modo de trabajo *'sleep'* del SRF01, deja de ser significativa.

El consumo medio del SRF01 es de 11.1 mA durante 160 ms. y el del EZ1 de 2.6 mA durante 350 ms. Con un par de sencillas multiplicaciones vemos que el sensor EZ1 es el que tendrá un menor consumo de energía y es lo que finalmente ha hecho que nos decantásemos por el mismo [1].

2.1.2 Conectividad y protocolos

La necesidad de adaptar la Internet tradicional a un entorno más restringido, en cuanto a recursos hardware y disponibilidad energética, fuerza a que protocolos muy elaborados y con mucha sobrecarga de tramas sobre la información útil vayan dejando paso a protocolos más ligeros que permiten un mejor aprovechamiento del ancho de banda disponible, utilizando hardware más sencillo y con consumos de energía más bajos.

Partiendo de una versión simplificada de cuatro capas del modelo OSI, podemos hacer una primera aproximación no exhaustiva de la equivalencia entre la tecnología Web y el Internet de las cosas (Figura 2.2).

2.1.2.1 Capa de Enlace

IEEE 802.15.4:

Es un estándar que define el nivel físico (PHY) y el control de acceso al medio (MAC), para redes inalámbricas que requieran comunicación a corta distancia, con potencia reducida y con bajas tasas de transmisión de datos.

Ofrece dos opciones para el nivel físico:

- La banda de 2.4 GHz, disponible a nivel mundial, con una velocidad máxima de transmisión de 250 Kb/s y 16 canales.
- La banda de 868 MHz (Europa) con un canal o la de 915 MHz (EEUU) con diez canales y con velocidades de transmisión máximas de 20 y de 40 Kb/s respectivamente.

[1] - Al hacer la comparación de consumos centrada exclusivamente en los sensores, no tuvimos en cuenta que durante el tiempo de lectura de la distancia también está activa la MPU del micro y eso hace variar el resultado. Ver los cálculos finales en el apartado 4.2.1.5.

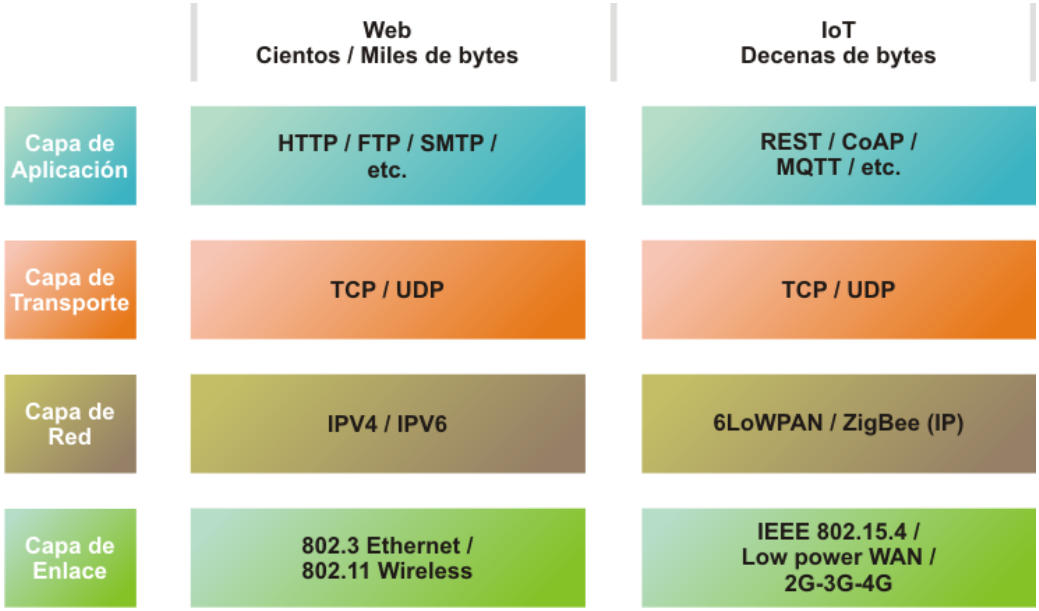


Figura 2.2: Comparativa entre las tecnologías Web e IoT.

Los dispositivos típicos (1mW) cubren un rango de 10 a 20 m. Con un incremento moderado en la potencia de transmisión, pueden llegar en campo abierto a los 80-100 m.

La capa MAC incluye funciones de búsqueda a través de los canales permitidos en la banda, mientras que la PHY contiene varias funciones de bajo nivel, tales como la detección de los niveles de energía recibidos, indicadores de calidad en el enlace, así como de conmutación de canales, lo que permite la asignación dinámica de canales en función de la calidad de la conexión.

Soporta varios tipos de seguridad: solo encriptación (AES-CTR), solo autentificación (AES-CBC-MAC) y encriptación más autentificación (AES-CCM).

Low power WAN:

En el mundo del *'IoT'*, en muchos casos se necesita conectividad entre dispositivos colocados en lugares alejados unos de otros y en entornos de difícil acceso o inhóspito, en los que no siempre se encuentra disponible una red local para el intercambio de datos.

Una posible solución es una red WAN con cobertura en un área extensa, orientada a bajas tasas de transmisión de datos y con bajo consumo.

Este tipo de redes trabajan en las bandas de 868 MHz (Europa) y de 915 MHz (EEUU). Según la potencia de emisión tienen consumos de entre 20 y 70 mA, logrando alcances de varios Km.

Ejemplos de este tipo de redes son las que proponen la compañía francesa Sigfox y la alianza internacional LoRaWAN.

Redes de telefonía móvil:

Para cubrir las mismas necesidades que en el caso anterior, puede ser necesaria una red con cobertura global, como la que proporcionan las redes de telefonía móvil, con un 95% de cobertura 2G a escala mundial.

La mayoría de empresas de este sector, están tomando conciencia de la importancia de la 'IoT' para sus negocios y están adaptando progresivamente su infraestructura a este tipo de servicio.

2.1.2.2 Capa de red:

A los protocolos IPV4 e IPV6, se unen otros dos protocolos que por sus características resultan más apropiados para su aplicación en el 'IoT': "6LoWPAN" y "ZigBee". Ambos están basados en el estándar IEEE 802.15.4 como capa de enlace.

6LoWPAN:

Acrónimo de "IPv6 over Low-Power Wireless Personal Area Networks", es un protocolo estándar y abierto, definido por el IETF [W20] en la norma RFC 6282.

Transporta eficientemente paquetes IPV6 (1280 bytes máximo) dentro de tramas más pequeñas (127 bytes), que son las definidas en IEEE 802.15.4. De esta manera cada nodo puede tener una dirección individual IP enrutable.

El uso de infraestructura basada en IP, le dan la ventaja de muchos años de desarrollo de esta tecnología, facilitando el uso de estándares abiertos y la interoperabilidad entre redes.

En la Figura 2.3 pueden identificarse los tres componentes principales de una red 6LoWPAN:

- El '*Edge Router*', o enrutador de borde, es el encargado del intercambio de datos entre los nodos de la red 6LoWPAN e Internet (u otra red IPV6), también gestiona el intercambio de datos entre los nodos locales y mantiene la subred de radio. Se comunica de forma nativa con otras redes a través de routers IP.

- Los *'Routers*' o nodos que tienen la capacidad de dirigir paquetes de datos destinados a otros nodos dentro de la red.

- Los '*Hosts*' o nodos finales que no tienen la posibilidad de enrutamiento. Son los únicos elementos que pueden pasar a estado 'sleep', solicitando datos a los '*Routers*' en los momentos en los que están activos.

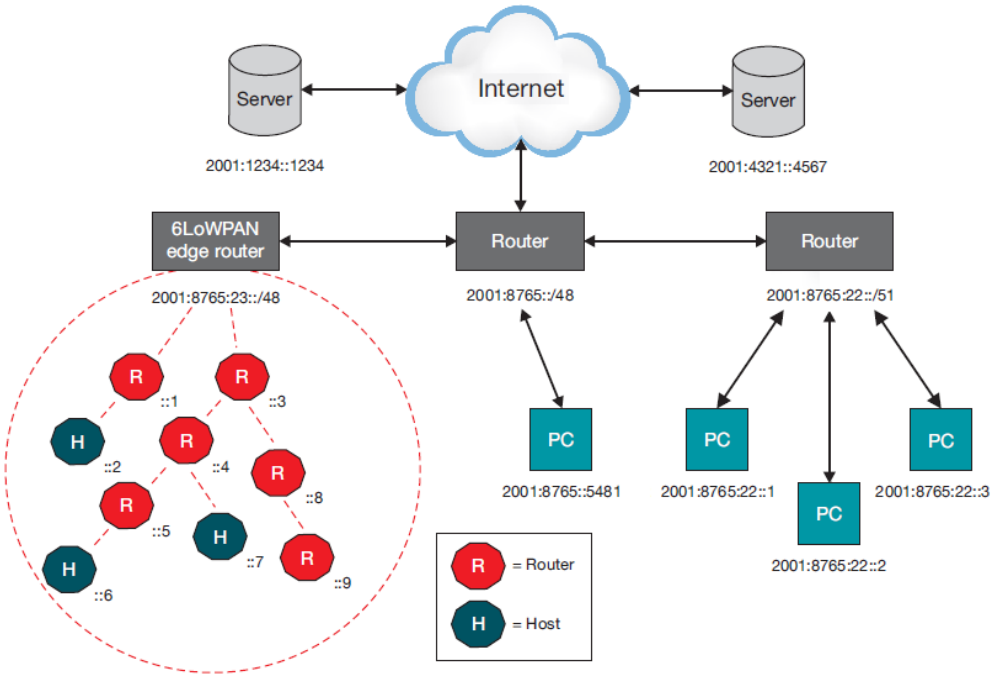


Figura 2.3: Ejemplo de una red IPV6 con una red 6LoWPAN.

ZigBee:

Es el nombre de un conjunto de protocolos mantenidos por la organización “ZigBee Alliance”, formada por empresas, universidades y emprendedores de todo el mundo [W21]. Pese a tener abiertas las especificaciones de los protocolos, para poder utilizarlos de forma comercial es necesario certificarlos y ser miembro de pago de la organización.

El protocolo principal se denomina “ZigBee PRO” del cual derivan otra serie de protocolos o perfiles dirigidos a aplicaciones en áreas específicas como la de la automatización del hogar, la salud o la iluminación, entre otras. Actualmente está en fase de prueba una reunificación de todos estos protocolos dentro de una especificación denominada “ZigBee 3.0”.

Atendiendo al rol que tienen en la red, existen tres tipos de nodos, todos ellos con las capacidades básicas de lectura de sensores y/o control de los actuadores asociados:

- ‘*Coordinator*’: Es el tipo de nodo más completo y ha de existir uno por cada red. Es el encargado de controlar la formación y la seguridad de la red.
- ‘*Router*’: También llamado ‘*Full function device*’. Actúa como repetidor intermedio, pasando datos entre distintos nodos. Está pensado para extender el alcance de la red.
- ‘*End device*’: También denominado ‘*Reduced function device* ‘. Puede conectar con su nodo padre (coordinador o rúter) pero no puede redirigir información entre otros nodos.

Es posible la formación de redes en malla, árbol y estrella. En la Figura 2.4 se muestra un ejemplo de cada topología.

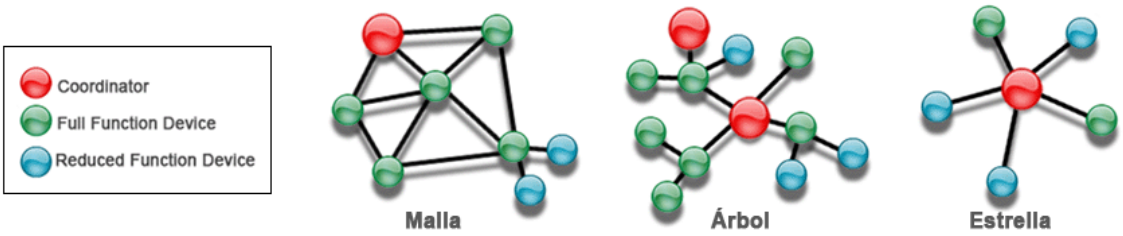


Figura 2.4: Ejemplos de redes ZigBee en malla, árbol y estrella.

Para reducir en lo posible el consumo de energía, las redes ZigBee pueden trabajar en modo '*beacon*' o baliza. El coordinador, una vez formada la red, asigna unos slots de tiempo a cada participante durante los cuales tendrán que estar activos para ejecutar las tareas de intercambio de datos, además de sus propias tareas de control.

De esta manera, cada nodo, incluido el propio coordinador, pueden pasar a modo '*sleep*' y despertar en función del periodo asignado a su slot, o en el caso del coordinador, despertar con el primer slot asignado y pasar de nuevo a modo de bajo consumo cuando se hayan finalizado las tareas de comunicación con todos los nodos.

2.1.2.3 Capa de transporte:

En esta capa, siguen predominando los dos protocolos principales de transporte en la Internet tradicional.

TCP: Orientado a la conexión, con mayor sobrecarga de trama ('*pay-load*') pero con aseguramiento de corrección de errores y validación de la entrega.

UDP: No orientado a la conexión, más ligero y rápido, pero con posibilidad de pérdida de datos. Algunos protocolos basados en él, como veremos a continuación (CoAP), tratan de minimizar este riesgo en las transferencias de los datos más significativos, utilizando confirmaciones.

2.1.2.4 Capa de aplicación:

Aquí encontramos gran variedad de protocolos. Todos ellos tratan de adecuarse de alguna manera al trabajo con nodos y redes de bajos recursos.

REST (Representational State Transfer):

El término se utiliza en la actualidad para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los mismos.

Más que un protocolo estandarizado, puede considerarse como un marco de trabajo para comunicar entre sí aplicaciones Web.

Utiliza URLs para identificar a los objetos (.../temperatura/1234), verbos HTTP para especificar acciones sobre los mismos (GET, PUT, POST, DELETE, …) y formatos estándares (XML, JSON, CBOR, etc.) para la representación de los propios objetos.

Existe un gran número de aplicaciones en la web y varios portales que ofrecen su interfaz para desarrolladores basada en 'REST'. Algunos ejemplos los encontramos en Amazon, Yahoo!, Facebook, Twitter o Google.

CoAP (Constrained Application Protocol)

Standard abierto desarrollado por el grupo CoRE perteneciente al IETF (Internet Engineering Task Force [W20]). Corre sobre UDP y puede interoperar con HTTP a través de proxies sencillos.

Sigue el modelo cliente/servidor y es fundamentalmente un protocolo del tipo '*one-to-one'*.

Al estar soportado sobre UDP, existe la posibilidad inherente de pérdida de datos. El protocolo trata de mitigarlo trabajando con dos tipos de mensajes, confirmables y no confirmables. Los mensajes con información más relevante necesitan ser confirmados por el receptor con un paquete ACK. Los mensajes no confirmables, simplemente se envían y no se espera el reconocimiento por parte del receptor (*'fire and forget'*) [W18].

MQTT (Message Queue Telemetry Transport)

También de código abierto, originalmente desarrollado por IBM y actualmente soportado por una comunidad de desarrolladores independientes [W22].

Trabaja sobre TCP y su propósito principal es la telemetría o monitorización remota de datos.

El protocolo sigue los modelos cliente/servidor y publicador/subscriptor. Cada sensor es un cliente que conecta exclusivamente con el servidor, también llamado *'broker'*. Sin embargo, cada cliente puede subscribirse a información proporcionada por otros clientes que le será remitida por el *'broker'*. De esta manera, se convierte en un protocolo de tipo *'many-to-many'*, donde los mensajes entre nodos se pasan a través del *'broker'* central.

DDS (Data Distribution Service) [W23]:

Protocolo propietario centrado en los datos ('*data centric'*), controla las relaciones entre ellos y su distribución.

Las distintas aplicaciones se comunican publicando y suscribiéndose a los temas definidos por su nombre y en las cuales están interesadas. Publicadores y subscriptores pueden pertenecer

a distintos dominios, sin que existan transferencias de datos entre los mismos, facilitando de esta manera la seguridad.

En la Tabla 2.2 se hace una comparación entre las principales características de los protocolos mencionado anteriormente.

Protocolo	*Transporte*	*Mensajes*	*2G,3G,4G*	*Consumo de energia*	*Recursos memoria*	*Seguridad*
HTTP / REST	TCP	Petición / Respuesta	Excelente	Adecuado	10Ks RAM/Flash	Baja - Opcional
CoAP	UDP	Petición / Respuesta	Excelente	Excelente	10Ks RAM/Flash	Media - Opcional
MQTT	TCP	Publicador / Subscriptor Petición / Respuesta	Excelente	Bueno	10Ks RAM/Flash	Media - Opcional
DDS	UDP	Publicador / Subscriptor Petición / Respuesta	Adecuado	Pobre	100Ks RAM >>> Flash	Alta - Opcional

Tabla 2.2: Comparativa entre algunos protocolos de la capa de aplicación (fuente: Cisco).

Ya hemos señalado anteriormente que para la realización de este proyecto nos hemos basado en los materiales que se utilizan en la asignatura, entre otros, el módulo WiFly. La conexión entre el sistema empotrado que hemos construido y el host remoto, se hace a través de una red Wifi local que proporciona el acceso a Internet.

En cuanto a protocolos en la capa de aplicación, teniendo en cuenta que estamos desarrollando un sistema prototipo, hemos optado por implementar un protocolo de comunicación entre la unidad sensora y el host que nos permita probar y contrastar diversas opciones en vez de ceñirnos a un estándar puro.

Nos hemos basado en el estilo de arquitectura ‘REST’, pero hemos añadido la posibilidad de que el host pueda enviar una respuesta de confirmación de las tramas recibidas y hemos incluido para todos los mensajes un CRC que permite comprobar si el intercambio de datos es correcto. Estas dos últimas funciones son más propias del protocolo CoAP.

El diseño anterior nos proporciona agilidad en las pruebas, siendo bastante trivial una adaptación posterior a los estándares ‘REST’ en un entorno de producción real.

En el capítulo cuatro se explican todos los detalles relativos al protocolo y se ponen varios ejemplos de tipo práctico.

2.2 Estudio de mercado

Existen algunos productos similares al que desarrollamos en este trabajo. Sin embargo, su escasa implantación en el mercado, hace suponer que adolecen de una o más de las características siguientes:

- Precio elevado.
- Falta de fiabilidad.
- Poca duración de la batería o duración real no acorde con lo anunciado.

La mayoría de ellos tiene varios años de vida y posiblemente las faltas anteriores sean consecuencia de la tecnología de la que se disponía en el momento en el que fueron diseñados.

En la Tabla 2.3, mostramos una comparación entre algunos de los sistemas que conocemos, con las características que anuncian, y el nuestro:

	Medida de nivel	**Medidas adicionales**	**Conectividad**	**Duración anunciada de la pila**	**Precio**
Distromel	Si	Temperatura	GPRS	4 años @ 1 medición y 1 envío diarios	?
Urbiotica	Si	Temperatura	GPRS/3G	10 años @ 1 medición cada 20 minutos 1 envío cada 8 horas	?
Srubo	Si	Temperatura	GPRS/3G	Baterias + panel solar	?
Enevo	Si	Temperatura Verticalidad	GPRS/3G	10 años @ ?	Para 10 años y 1000 unidades 12€/mes unidad
Objetivos de nuestro sistema	Si	Temperatura Verticalidad Descargas	Wi-Fi	Capítulo 4 *	Capítulo 7

* Objetivo: 10 años @
1 vigilancia de alarmas de temperatura y verticalidad por minuto,
1 vigilancia de alarma de llenado y envío de datos al host cada 15 minutos.

Tabla 2.3: Comparativa entre productos similares y el nuestro

2. Antecedentes

Página intencionadamente en blanco.

3.1 Sistema total

Como ya hemos expuesto en el resumen inicial, en este proyecto desarrollamos un sistema prototipo que permite monitorizar online y en tiempo real un conjunto de datos, con el objetivo de mejorar la gestión de los servicios de recogida de contenedores urbanos de residuos.

En la Figura 3.1 se muestra un ejemplo con todos los elementos que participarían en una situación real. En este trabajo nos centramos en los componentes clave, redondeados en azul, para disponer de un sistema mínimo y funcional que nos permita estudiar su comportamiento.

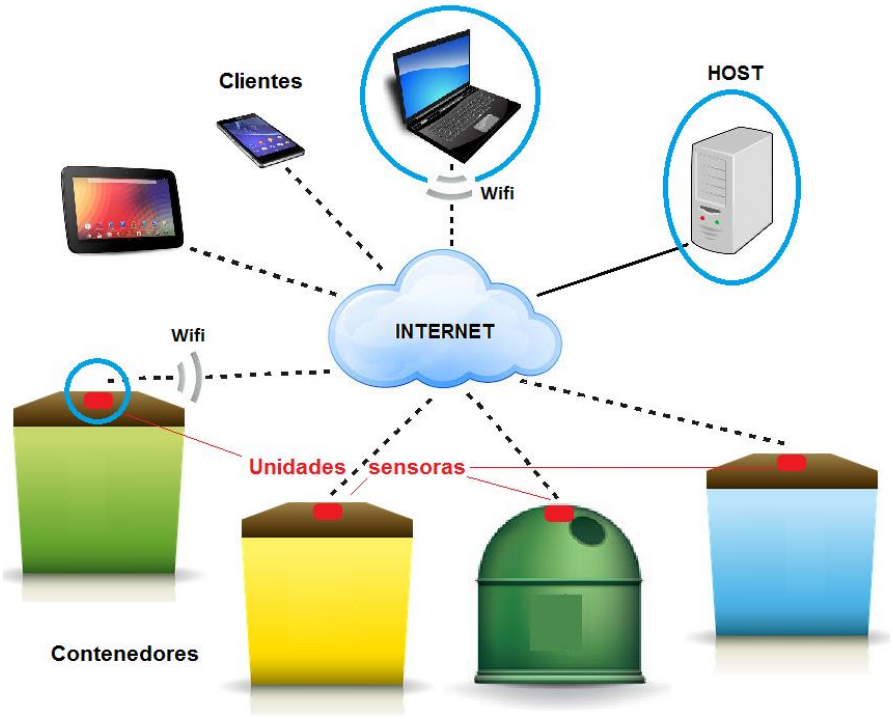


Figura 3.1: Sistema total.

Para abordar la tarea, hemos dividido el sistema en cuatro bloques:

- Unidad sensora.
- Host o servidor del sistema.
- Redes de comunicación.
- Clientes.

En los apartados siguientes describimos los bloques mencionados.

3.2 Unidad sensora

La unidad sensora, Figura 3.2, va instalada en la parte superior o lateral de cada contenedor y es un sistema empotrado que recoge los valores de las variables físicas con la ayuda de los

correspondientes sensores, los procesa y los envía al host remoto a través de las redes de comunicación establecidas entre ambos. El host también envía datos, básicamente valores de ajuste, a la unidad sensora a través de los mismos canales de comunicación.

Las magnitudes físicas que obtiene la unidad sensora son la inclinación del contenedor a través de un sensor de aceleración, la temperatura en su interior por medio de un sensor de temperatura y el nivel de llenado con la ayuda de un medidor de distancia por ultrasonidos.

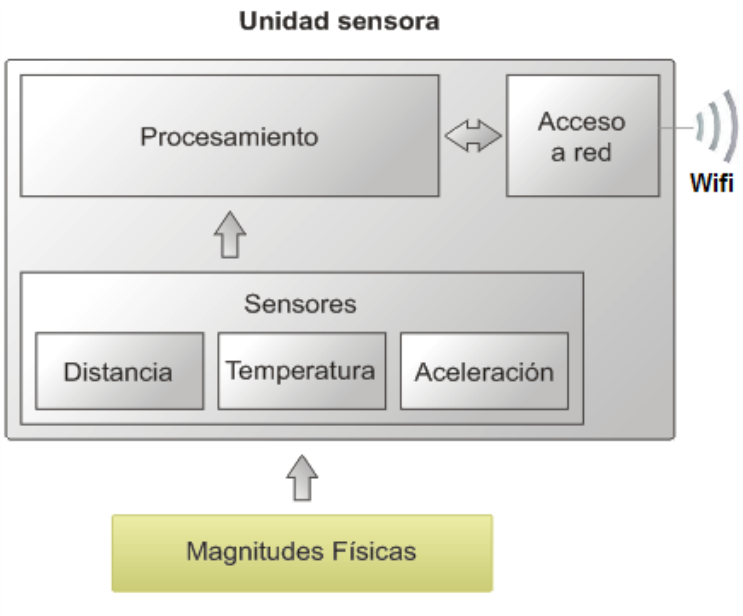


Figura 3.2: Diagrama general de la unidad sensora.

Para cumplir con los objetivos previstos, en la unidad sensora hemos de conseguir:

- Obtener el ángulo de inclinación del contenedor con respecto al plano horizontal de gravedad y generar la alarma correspondiente.
- Obtener la temperatura y generar la alarma correspondiente.
- Obtener el nivel de llenado.
- Detectar la acción de descarga.
- Enviar los datos anteriores al control central o host, recibir desde el mismo los parámetros de ajuste y aplicarlos.
- Trabajar con el menor consumo posible para optimizar el tiempo de autonomía de las pilas con las que se alimenta.

Con relación al consumo en la unidad sensora, hay dos elementos que destacan muy por encima del resto: el sensor de distancia y el WiFly. La forma en la que se utilice cualquiera de ellos, influirá de forma decisiva en el consumo final.

Al microcontrolador lo mantendremos en modo de bajo consumo, junto con el resto de componentes, en el periodo que transcurre entre lecturas de los sensores.

En cuanto a la fiabilidad, además de las medidas generales, tendremos que tomar medidas particulares para el momento del vaciado de los contenedores.

Existen distintos tipos de contenedores atendiendo al diseño, forma, capacidad, etc. No obstante, todos ellos se descargan o bien por su parte superior, en cuyo caso el contenedor se voltea, o bien por su parte inferior, en cuyo caso el contenedor se descarga verticalmente. El tiempo que se tarda en realizar esta operación varía con el tipo de contenedor, pero nunca es superior a los dos minutos.

En cualquiera de las dos situaciones anteriores, tendremos que detectar que el contenedor se ha vaciado para contabilizarlo. Además, tendremos que evitar validar valores de nivel o de verticalidad que, aunque sean reales, no son útiles ya que se deben al movimiento que sufre el contenedor durante la maniobra de descarga.

Una vez expuestas las consideraciones generales, vamos a centrarnos en cada una de las acciones que hemos de llevar a cabo en la unidad sensora y en la resolución de los problemas que se nos plantean.

3.2.1 Ángulo de inclinación:

Utilizaremos un sensor de aceleración o acelerómetro para medir la aceleración estática producida por la fuerza de la gravedad que nos permitirá decidir si el contenedor está o no volcado.

Esta será la primera lectura a hacer ya que de ella depende la obtención o no del nivel de llenado. Mientras el contenedor haya perdido su verticalidad, no obtendremos el nivel de llenado debido a que en esta situación no es significativo dicho valor o bien porque se está descargando o bien porque el contenedor se ha volcado, en cuyo caso la medida del nivel no sería de confianza por los posibles residuos vertidos.

Cuando un contenedor se vuelca, dada su construcción y peso, lo hace totalmente sobre una de sus caras laterales o sobre una arista en el caso de los contenedores de tipo iglú. Por otra parte, siempre se tiende a emplazarles en una zona lo más plana posible para evitar su deslizamiento.

Con los datos anteriores podemos considerar que un contenedor está volcado si su ángulo de inclinación es mayor de 45º en cualquier dirección respecto del plano horizontal de la gravedad. De esta manera contamos con un margen suficiente para el caso de los contenedores cuya posición estable no coincida con dicho plano.

3. Descripción funcional

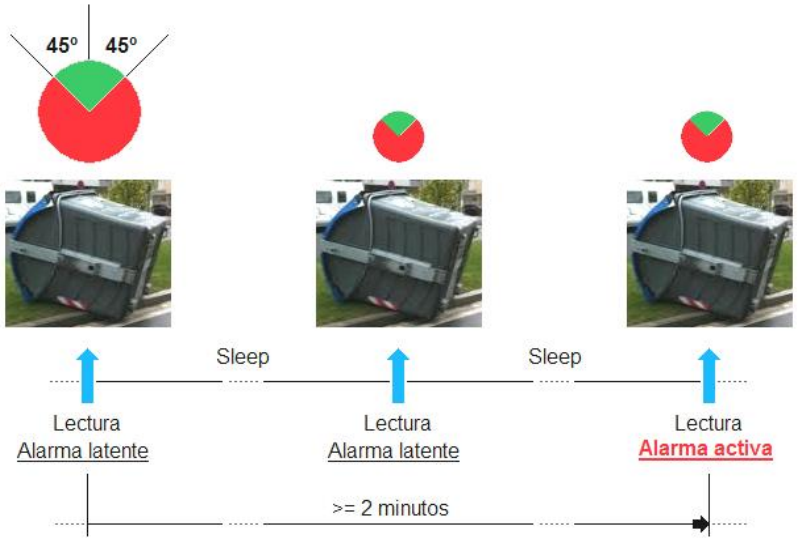


Figura 3.3: Elaboración de la alarma por verticalidad.

Tanto el periodo de lectura del sensor como el valor límite de alarma (45º), son fijos sin posibilidad de ajuste por parte del usuario. En el capítulo siguiente justificaremos el periodo de lectura.

Para evitar falsas alarmas en el momento de la descarga del contenedor, siempre esperaremos el tiempo máximo de descarga (dos minutos) antes de validar la alarma por pérdida de verticalidad.

En la Figura 3.3 se muestra de forma gráfica la elaboración de la alarma de verticalidad.

3.2.2 Alarma por temperatura:

Con la medición del valor de la temperatura, lo que pretendemos fundamentalmente es disparar una alarma en el caso de que el contenedor o su contenido estén ardiendo.

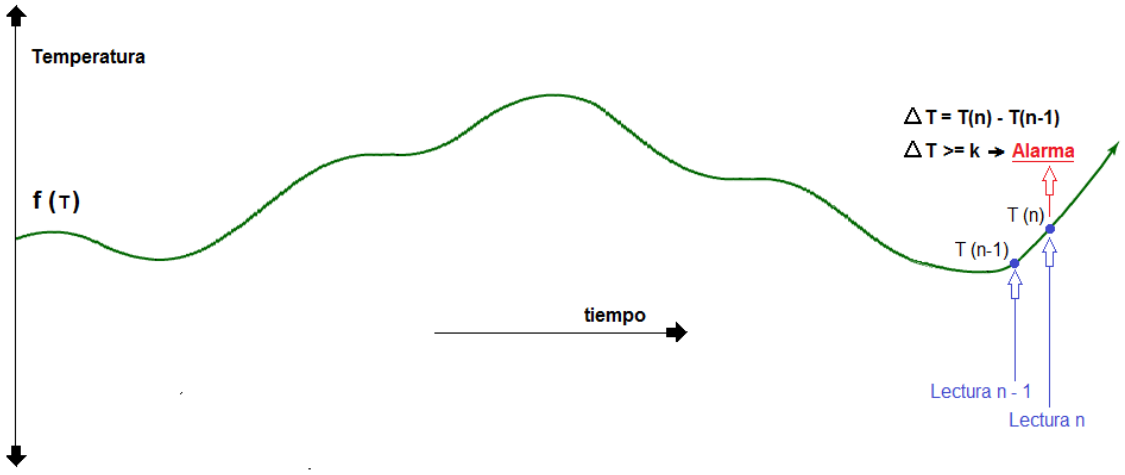


Figura 3.4: Elaboración de la alarma por temperatura.

Considerando que los contenedores pueden estar situados en zonas con condiciones climáticas muy diferentes y que además puede haber grandes diferencias de temperatura entre el día y la noche, para generar la alarma no compararemos valores absolutos de temperatura, sino que lo haremos calculando la velocidad con la que ésta varía en el tiempo (ver Figura 3.4).

Como en el caso del sensor de verticalidad, el periodo de lectura de la temperatura y el umbral para el cálculo de la alarma (k) están fijadas en el sistema, sin posibilidad de ajuste por parte del usuario. En el capítulo cuatro, explicitaremos los valores.

3.2.3 Nivel de llenado:

En la Figura 3.5, observamos que la unidad sensora puede ir colocada en la parte superior (A) o lateral (B) del contenedor en función de que éste sea de descarga inferior o superior respectivamente.

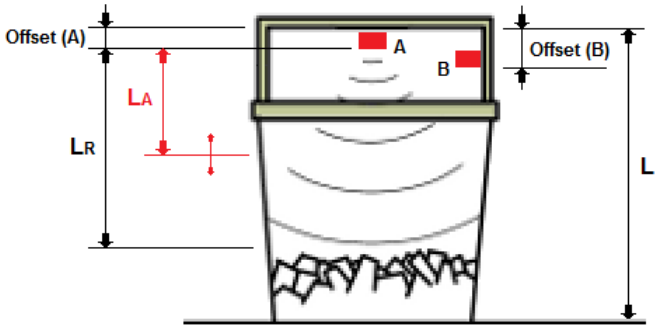


Figura 3.5: Nivel de llenado (L_R), nivel de alarma (L_A) y offset.

Para determinar el nivel de llenado (L_R), utilizamos el medidor de distancia por ultrasonidos que, dada su ubicación, indicará la distancia que queda libre desde el sensor hasta el nivel que marcan los residuos depositados. Esto, unido a que conocemos la altura total del contenedor (L), nos permitirá calcular el porcentaje de llenado.

La altura total (L) y el pequeño offset que pueda introducir la unidad sensora son variables asociadas al modelo de contenedor y están fijadas en la base de datos.

El periodo de medición del nivel y el nivel de disparo de la alarma (L_A) son parámetros programables por el usuario y asociados también al modelo de contenedor (ver Figura 3.6).

3.2.4 Detección de descarga:

Para detectar el momento de la descarga del contenedor, compararemos siempre la última medida de nivel con la medida obtenida en la lectura inmediatamente anterior, y si la diferencia está por debajo de una longitud pre-programada en el sistema, siempre que no exista alarma latente o activa de verticalidad, señalizaremos la descarga.

El número de descargas necesarias para disparar la alarma de mantenimiento es programable por el usuario (ver Figura 3.6).

3.3 Host

El host o servidor central se encarga de recibir y procesar los datos enviados por la unidad sensora y de enviar a la misma las variables de ajuste a través de los canales de comunicación mencionados en el apartado siguiente. También se ocupa de servir las peticiones que hacen los usuarios mediante la interfaz de usuario y del envío de los e-mails con las alarmas.

Para cumplir con los objetivos previstos, en el host tenemos que conseguir:

- Monitorizar el nivel de llenado, la temperatura y la verticalidad a partir de los datos enviados por la unidad sensora.
- Calcular y monitorizar el número de descargas acumuladas por el contenedor desde el último mantenimiento.
- Permitir que el usuario programe el periodo para petición de datos de ajuste desde la unidad sensora y el periodo de lectura del nivel.
- Enviar los datos anteriores a la unidad sensora.
- Permitir que el usuario programe los valores límite para generar la alarma de nivel y la alarma por descargas acumuladas.
- Calcular y generar las alarmas de nivel y de descargas acumuladas, utilizando los datos enviados por la unidad sensora y los valores límite programados.
- Ofrecer la opción de enviar un correo electrónico a las direcciones previstas para tal fin, cuando se active o desactive cualquier alarma correspondiente al nivel, temperatura, verticalidad o descargas acumuladas.

Para materializar los requisitos anteriores, nos hemos servido de los siguientes recursos:

- Servidor HTTP (Apache) para enviar al cliente Web del usuario las páginas con las que éste interactuará.
- Base de datos (mySql) para almacenar los datos persistentes. En el anexo 11.3 puede verse el diagrama EER de la misma.
- Servidor SMTP para el envío de los e-mails informativos con el estado de las alarmas.
- Hemos desarrollado una aplicación en el servidor con los lenguajes PHP y HTML para generar las páginas dinámicas que permiten la interacción con el usuario, mostrando y recogiendo los datos de interés del sistema. La aplicación se encarga también de la comunicación con la unidad sensora y del control de la lógica de las variables con las que trabajamos.

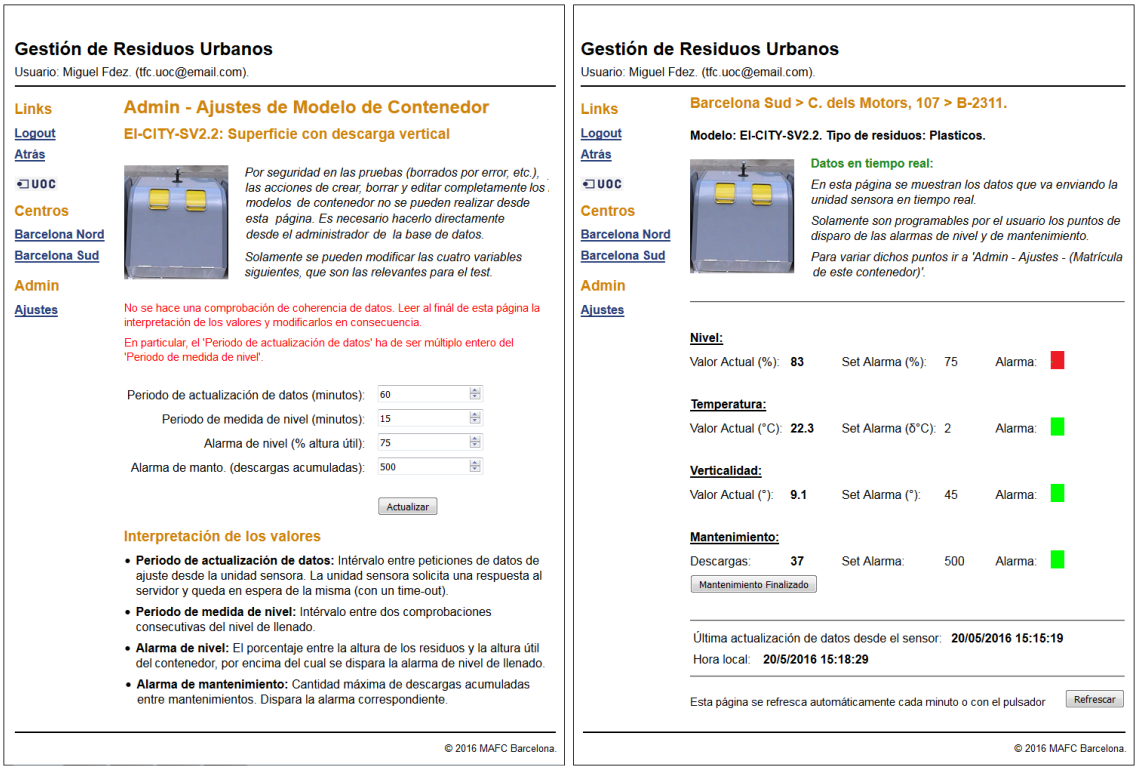


Figura 3.6: Entrada de ajustes de sistema (izquierda) y datos en tiempo real (derecha).

Como ejemplo de la interfaz desarrollada, en la Figura 3.6, a la izquierda, se muestra el formulario que permite introducir los ajustes de usuario en el sistema en función del modelo de contenedor y a la derecha el que permite visualizar los valores actuales de un contenedor determinado.

En la propia imagen pueden leerse algunas aclaraciones sobre los datos manejados que no describiremos aquí para no repetirnos.

Otros formularios implementados en el host son el de entrada de usuario y los de selección de centro de operaciones o de contenedor contra el que se quiere trabajar.

Por otra parte, en la Figura 3.7 se observa un ejemplo de e-mail enviado a los usuarios por cambio de estado de alarmas en un contenedor.

Pueden verse reflejados dos cambios de estado, el de la alarma de nivel que ha pasado a estar por encima del umbral previsto y el de verticalidad que ha pasado a estado normal después de una alarma previa.

Estos e-mails, se envían solamente a aquellos usuarios que estén dados de alta en la base de datos del sistema como destinatarios de los mismos.

3. Descripción funcional

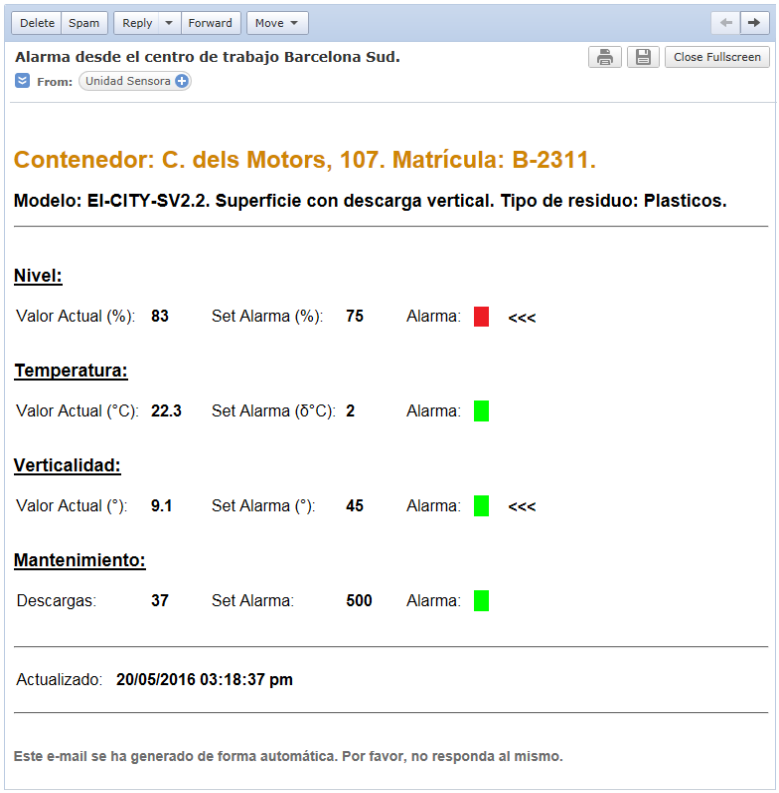


Figura 3.7: Notificación de alarmas enviada al usuario.

3.4 Redes de comunicación

Las redes de comunicación transportan los datos entre el host, la unidad sensora y los clientes.

En nuestro caso utilizamos dos tipos de redes; Wifi para la conexión de la unidad sensora y de los clientes al modem-router inalámbrico local e Internet para la conexión entre ambos elementos y el servidor.

3.5 Clientes

Los clientes del sistema son los encargados de facilitar la interacción humana con el mismo. A través de ellos, los usuarios envían y obtienen la información con el objetivo último de mejorar la gestión de la recogida de residuos.

En las pruebas, nosotros utilizamos un PC portátil con el sistema operativo Windows 7 Profesional 64 bits, Service Pack 1 y el navegador Mozilla Firefox, versión 46.0.1.

4.1 Introducción

En este capítulo vamos a exponer con detalle el sistema experimental realizado y los datos obtenidos del mismo, para lo cual le dividiremos en tres apartados destinados a la descripción del hardware, del software y del protocolo para comunicación entre la unidad sensora y el host.

Aunque no está explícitamente incluido entre los objetivos del proyecto, damos por sobreentendido que a lo largo de las etapas de diseño y de implementación, se han de tener en cuenta las premisas que conduzcan a lograr un sistema fácil de mantener y altamente fiable.

En algunos casos, estos requisitos se oponen al objetivo de conseguir un bajo consumo en la unidad sensora y por lo tanto hemos tenido que llegar a un buen compromiso entre ambos condicionantes.

4.2 Hardware

Para la descripción del hardware vamos a apoyarnos en el diagrama de bloques del sistema mostrado en la Figura 4.1. y en el diagrama de bloques hardware de la unidad sensora de la Figura 4.2.

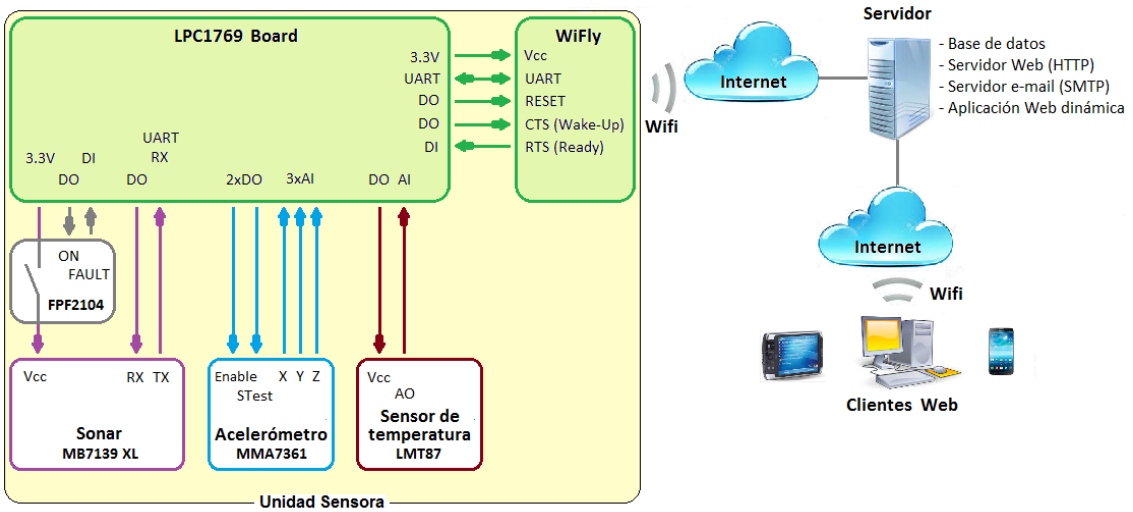


Figura 4.1: Diagrama de bloques del sistema.

4.2.1 Unidad sensora

Como ya hemos introducido en el capítulo anterior, la unidad sensora se encarga de recoger los valores de las variables físicas con la ayuda de los correspondientes sensores, de procesarlos y de enviarlos al host.

4. Descripción detallada

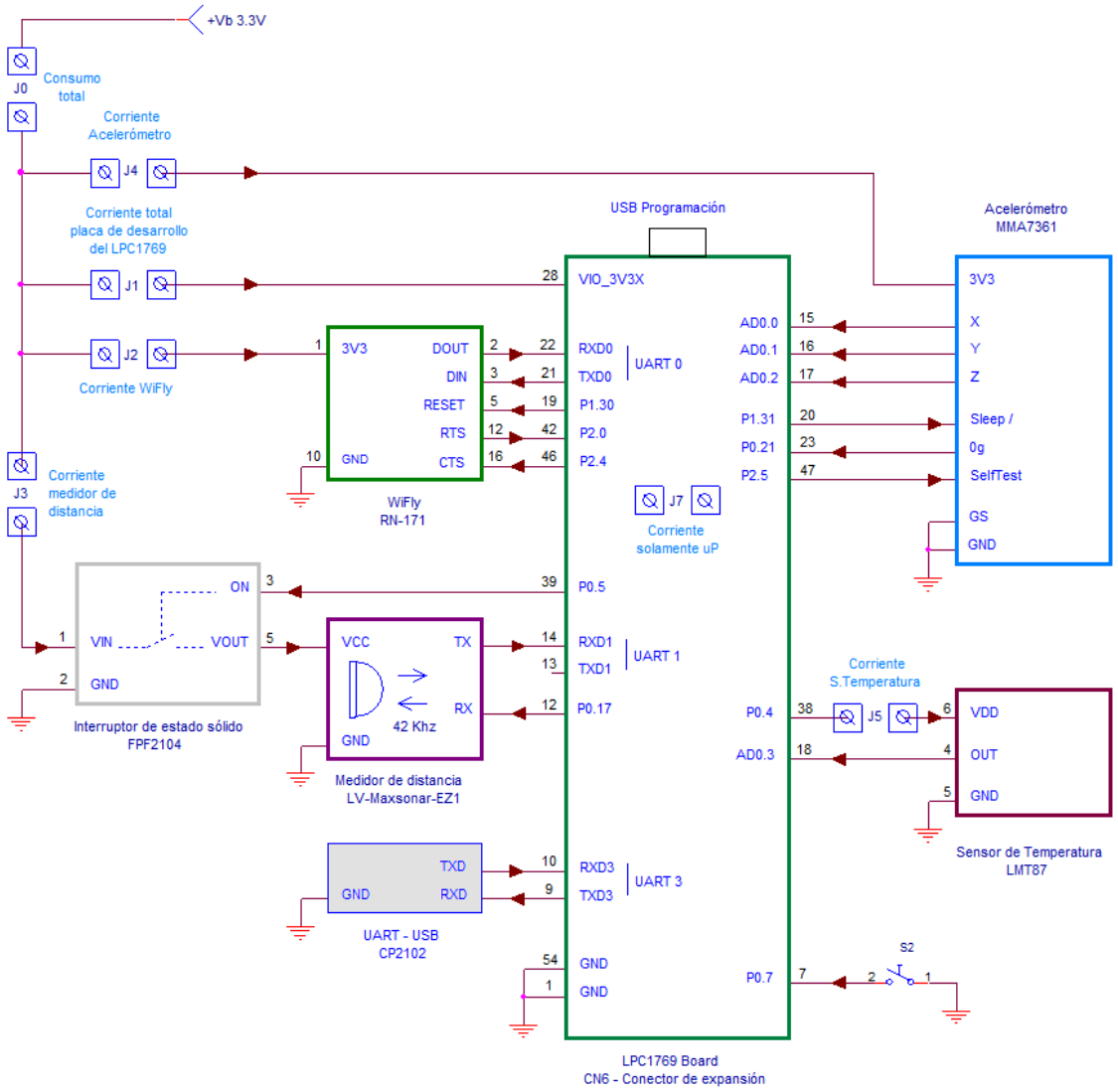


Figura 4.2: Diagrama de bloques hardware de la unidad sensora.

A continuación, detallamos cada uno de los componentes de la unidad sensora e indicamos los datos de consumo obtenidos por cada bloque. Para hacer dichas medidas, hemos suprimido elementos superfluos como los diodos led.

En la Figura 4.2, además de los bloques hardware, puede apreciarse la disposición de los puentes montados en la placa base de la unidad sensora, que nos han permitido medir la corriente consumida por los diferentes elementos en distintas circunstancias.

Con el interruptor S2, hemos ganado en agilidad durante las pruebas. Cuando está actuado, el tiempo del sistema en modo *'sleep'* se divide por seis respecto al programado, de modo que cada minuto de espera se convierte en solamente diez segundos.

4.2.1.1 LPC1769

Ejecuta el programa de la unidad sensora, gestiona el control de todos los elementos hardware y se comunica con el servidor a través del módulo WiFly.

Alterna entre los estados de trabajo y de bajo consumo en los periodos transcurridos entre lecturas de datos. Como estado de bajo consumo, utilizamos el modo 'deep power-down', en el que solamente queda activa la alimentación para el módulo de reloj de tiempo real.

De entre todos los periféricos de los que dispone, utilizamos el conversor analógico-digital para la lectura de los sensores de temperatura y verticalidad, dos UART para la lectura del sensor de distancia y comunicación interna con el módulo WiFly y el GPIO para el control y lectura de estados del resto de módulos hardware.

También utilizamos su reloj de tiempo real con un doble propósito. Por un lado, es el encargado de hacer el 'wake-up' del micro cada vez que expira el tiempo programado y por otro lado, utilizamos su memoria remanente de 20 bytes para mantener los datos que han de conservarse entre ciclos de trabajo.

La media de consumo en modo activo es de aproximadamente 56 mA y en modo 'deep power-down' de 1uA. El periodo de actividad depende de que haya que tomar datos del sensor de distancia o no y de si hay que comunicarse con el host. Para la medida y procesado de la temperatura y de la verticalidad, el periodo activo es de 6 ms, la lectura de nivel añade otros 350 ms y el envío o recepción de datos 12 segundos más como media.

4.2.1.2 WiFly

A través de su UART, se comunica con una de las UART del LPC1769. De esta manera el microcontrolador envía y recibe los datos desde el host vía la interfaz wifi del WiFly y su protocolo TCP/IP. En nuestro caso esta comunicación se produce a través de una red wifi doméstica.

El paso del módulo a modo durmiente lo activa el micro, con el comando apropiado, a través de la UART que conecta a ambos y la señal para despertarle se la envía a través de una salida digital. Además, también a través de una salida digital, el micro envía al WiFly la señal de 'reset' y recibe desde éste la señal de 'ready' por una entrada digital.

El módulo tiene un consumo de 4 uA. en modo 'sleep', de 35 mA. durante la recepción y de 185 mA. durante la transmisión a su máxima potencia de 12 dBm. Durante las pruebas realizadas, el consumo medio ha sido de 43 mA. durante aproximadamente 12 seg. que es la suma del tiempo que tarda en asociarse a la red wifi más el tiempo que tarda en conectar con el servidor y enviarle o recibir los datos.

4.2.1.3 Sensor de temperatura

La salida analógica del sensor, correspondiente a la temperatura medida, se conecta a una entrada analógica del micro para leer su valor. Dado el bajo consumo del sensor, le alimentamos directamente desde una salida digital del micro tal y como propone el fabricante en su hoja de datos.

Con la medición del valor de la temperatura, lo que pretendemos es generar una alarma que se active si el contenedor está ardiendo. Dicha alarma la detectaremos a través de la velocidad de subida de la temperatura en el tiempo, o lo que es lo mismo, basados en la diferencia de temperatura medida en dos ciclos consecutivos.

Alarma por temperatura = $(k > (T_i - T_{i-1}))$, siendo *'T'* la temperatura, *'i'* el ciclo en el que se lee el valor de la temperatura y *'k'* una constante que fijaremos como umbral de alarma.

Como ya comentamos en el apartado de justificación de este proyecto, con esta alarma se trata no tanto de evitar que el contenedor que está ardiendo quede inservible, cuanto de evitar que el fuego se propague a los elementos adyacentes (otros contenedores, vegetación, mobiliario urbano, etc.). Con esta consideración, tenemos que llegar a un buen compromiso entre el consumo, el intervalo de lectura de la temperatura, $I_T = i - (i - 1)$, y el valor de *'k'*, que marca el umbral de disparo de la alarma.

Como consecuencia de una serie de pruebas prácticas hemos fijado el periodo de lectura en un minuto y el umbral de alarma en dos grados centígrados por minuto. Una vez activada la alarma, la desactivaremos si la temperatura baja de 50ºC.

El consumo medido en el sensor es de aproximadamente 7 uA durante 3 ms.

4.2.1.4 Acelerómetro

Es el encargado de medir el ángulo de inclinación del contenedor con respecto al plano de gravedad terrestre. Las salidas analógicas del acelerómetro correspondientes a los ejes X, Y, Z van a parar a tres entradas analógicas del micro. La habilitación la hacemos directamente desde una salida digital.

Para bajar el tiempo de cómputo al mínimo posible y dado que en nuestro caso es irrelevante la dirección de la inclinación, si tomamos como referencia solamente el valor de la gravedad en el eje Z, el problema se reduce a una comparación entre el valor que nos entrega el acelerómetro para dicho eje y el seno del ángulo que utilizamos como umbral de alarma, en nuestro caso el seno de 45º.

En la figura 4.3 podemos observar gráficamente la situación descrita tomando como referencia el plano XZ y girando sobre el eje Y. El comportamiento es exactamente el mismo si tomamos como referencia el plano YZ y lo giramos sobre el eje X.

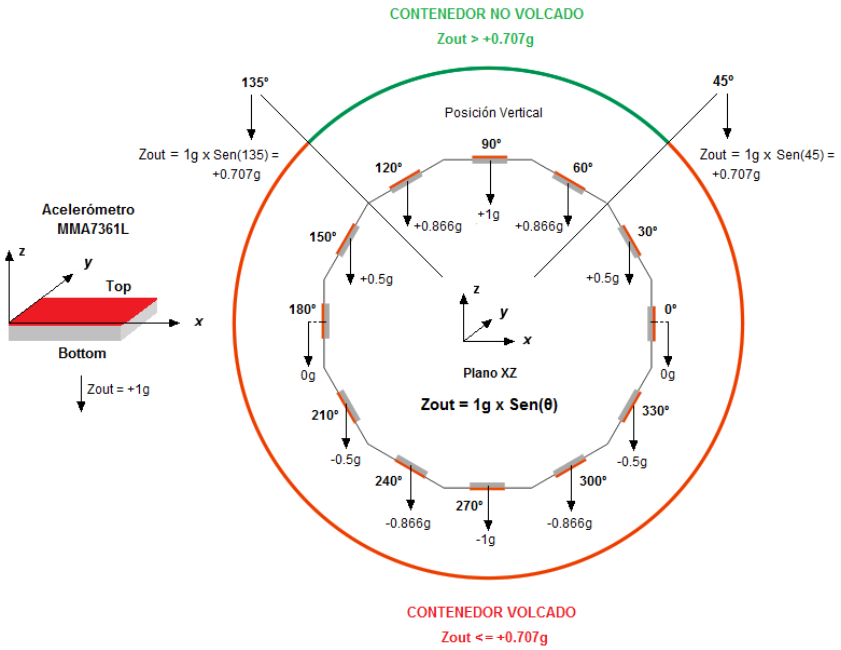


Figura 4.3: Comportamiento del acelerómetro en el plano XZ.

El intervalo de lectura del valor actual para vigilancia de la alarma será fijo, sin posibilidad de ajuste por parte del usuario ya que tal como lo hemos planteado, es un parámetro que no depende de ninguna variable externa. De esta manera no se incrementa el trabajo de mantenimiento del sistema sin necesidad.

Debido a que el periodo de lectura no es crítico, lo definiremos como múltiplo del intervalo de lectura de la temperatura, que ya hemos fijado en un minuto (ver apartado anterior), para aprovechar la misma maniobra que activa al sistema sensor:

$I_V = I_T * n, (n \in \mathbb{N} \ \& \ n > 0)$

Siendo I_V el periodo de lectura de la verticalidad, I_T el periodo de lectura de la temperatura y n un número natural mayor que 0.

Todas las pruebas las hemos hecho con n = 1, resultando que cada vez que medimos la temperatura, también medimos la verticalidad.

En cuanto al consumo, en las medidas realizadas, teniendo en cuenta solamente la salida del eje Z, la corriente de alimentación alcanza aproximadamente los 500 uA durante 3 ms y 4uA cuando el acelerómetro está deshabilitado.

4.2.1.5 Sensor de distancia por ultrasonidos

El medidor de distancia por ultrasonidos (LV-EZ1) permite al microcontrolador conocer la longitud que hay entre el sensor y el nivel que alcanzan los residuos depositados (ver figura

3.5). Su resolución es de 2,54 cm y el rango de medida va desde los 25 cm. hasta los 645 cm. Para objetos más cercanos de 25 cm., el sensor indica 25 cm.

El micro, a través del interruptor de estado sólido (FPF2104), conecta la alimentación solamente durante los ciclos activos en los que hay que medir el nivel, manteniéndola desconectada el resto del tiempo.

Para obtener el valor de la distancia, el micro envía un pulso al sensor que desencadena el proceso de lectura. Una vez finalizado, el sensor envía la longitud leída a través de su UART de solo transmisión a la UART del micro. El valor siempre lo envía con el formato "Rxxx\r", donde 'xxx' es la distancia en pulgadas con un valor máximo de 255 y '\r' es el carácter ASCII correspondiente al retorno de carro (13_d).

Cuando hicimos la comparación de consumos entre los sensores LV-EZ1, que es el que elegimos para este proyecto, y el SRF01 (ver final del apartado 2.1.1), no tuvimos en cuenta que durante el tiempo que tardan en hacer una lectura, la MPU también está activa y por lo tanto consumiendo energía.

El SRF01 tiene un consumo medio de 11.1 mA durante 160 ms. y el EZ1 de 2.6 mA. durante 350 ms. Si calculamos el consumo de energía en la primera lectura, que es la que nosotros utilizamos, tendremos que:

*SRF01: 3.3v * 11.1mA * 160ms * 10^{-3} = 5.86 mW*segundo.*
*EZ1: 3.3v * 2.6mA * 350ms * 10^{-3} = 3.00 mW*segundo.*

De los resultados anteriores, podemos deducir que el EZ1 consume casi la mitad que el SRF01. Sin embargo, al sumar los 58 mA que consume el micro (apartado 4.2.1.1) obtenemos:

*SRF01: 5.86 + (3.3v * 58mA * 160ms * 10^{-3}) = 36.49 mW*segundo.*
*EZ1: 3.00 + (3.3v * 58mA * 350ms * 10^{-3}) = 70.00 mW*segundo.*

En este caso, que es el real, el sensor que consume prácticamente la mitad es ahora el SRF01. El cambio de resultado es debido a la elevada relación entre el consumo del micro y el de los sensores y al tiempo que tardan en hacer la lectura, que en el caso del SRF01 es prácticamente la mitad que en el otro sensor.

Como ya habíamos comprado dos unidades del EZ1 y funcionalmente ambos sensores son prácticamente idénticos, decidimos utilizarlo en el proyecto. Sin embargo, para los cálculos de autonomía de la unidad sensora que haremos en el apartado siguiente, utilizaremos los datos teóricos del sensor SRF01, que nos aproximarán más a la realidad de lo que se puede conseguir en una supuesta producción comercial.

El consumo del IC (FPF2104) que hemos utilizado para conmutar la alimentación del sensor de distancia es de 10 nA en estado 'Off' y de 1 uA en estado 'On'. Despreciaremos el consumo en estado 'On', pero tendremos en cuenta los 10 nA para el estado Off.

4.2.1.6 Consumo y autonomía

Lo primero que vamos a hacer es fijar los parámetros bajo los cuales calcularemos la autonomía de la alimentación en la unidad sensora, basados en los datos de los apartados anteriores y en algunos supuestos:

A. Intervalo de medida y elaboración del estado de las alarmas de temperatura y verticalidad: 1 minuto.

B. Intervalo de medida del nivel y envío al host de los datos de nivel, temperatura, verticalidad y en su caso descarga: 15 minutos.

C. Intervalo de petición de datos de ajuste al host: 60 minutos.

D. Supondremos una alarma por temperatura o verticalidad diaria, cuya consecuencia será el envío adicional de los flancos de comienzo y fin al host.

Vamos a calcular el consumo diario a partir de la suma de los consumos de cada uno de los tramos enumerados anteriormente más el intervalo en modo *'sleep'*. La unidad común que utilizaremos es la de mAs (miliamperios segundo).

*A: (micro + sensor temperatura + acelerómetro) * 24 * 60 =*
*((56mA * 6ms * 10^{-3}) + (7uA * 10^{-3} * 3ms * 10^{-3}) + (0.5mA * 3ms * 10^{-3})) * 1440 = 486 mAs*

*B: (micro + nivel + WiFly) * 24 * 4 =*
*((56mA * 12.16s) + (11.1mA * 160ms * 10^{-3}) + (43mA * 12s)) * 96 = 115079 mAs*

C: Teniendo en cuenta la mínima diferencia de tiempo que existe entre solamente transmitir o transmitir y recibir los datos, respecto del tiempo que el WiFly tarda en asociarse a la red Wifi más la conexión con el servidor, podemos considerar que este caso ya está contabilizado en el caso B y por lo tanto no añade consumo.
Como consecuencia de la observación anterior, podemos concluir que la unidad sensora podría recibir automáticamente los datos de ajuste cada vez que envía datos de proceso al host, sin que ello suponga prácticamente penalización en el consumo.

*D: (micro + WiFly) * 2 = ((56mA * 12s) + (43mA * 12s)) * 2 = 2376 mAs*

A los consumos anteriores hay que sumarles el consumo durante el tiempo en modo *'sleep'* de la unidad sensora:

*SLEEP: (micro + WiFly + acelerómetro + FPF2104) * (86400s – 8s(A) – 1167s(B) – 24s(D)) =*
*(1uA + 4uA + 4uA + 0.01uA) * 10^{-3} * 85201 s = 768 mAs*

Sumando los consumos parciales, tenemos que el consumo total diario con los parámetros asignados es de 118709 mAs/día = **32.98 mAh/día** (miliamperios * hora / día).

Con la pila que nosotros utilizamos en la placa de pruebas, que es de 3.3v / 2600 mAh, tamaño "AA" (50 mm largo x 14.2 mm diámetro), y contando con una auto-descarga del 10% tendríamos una autonomía de 2340 mAh / 32.98 mAh/día ≈ **71 días**.

Basándonos en los mismos parámetros, pero haciendo un solo envío al día de datos al host y sin contar con la alarma diaria, tendremos que el consumo será de 2453 mAs/día = 0.682 mAh/día. Con la misma pila y auto-descarga anterior, alcanzaremos una autonomía de 3431 días = 9.4 años, periodo que ya se asemeja más al anunciado para otros productos similares al nuestro (ver apartado 2.2).

Con una pila de 3600 mAh de capacidad, de dimensiones algo mayores que la anterior (50 mm largo x 17 mm diámetro), y aplicando el último supuesto de un solo envío diario, obtendríamos una autonomía de (3600 - 360) / 0.682 = 4750 días = 13 años.

Si queremos mantener los parámetros propuestos inicialmente, para una autonomía de 10 años necesitaremos una capacidad de 32.98 mAh/día * 365 * 10 ≈ 120 Ah a 3.3v, que es ya una batería de dimensiones y precio considerables. Otra alternativa es hacer algunas modificaciones en el diseño, tal y como propondremos en el capítulo siete.

En la Tabla 4.1 se resumen los resultados anteriores en función de las condiciones de trabajo.

Capacidad	Periodo lectura temperatura y verticalidad	Periodo lectura nivel y envío datos al host	Alarmas diarias	Autonomia @ 10% autodescarga
2600 mAh	1 mínuto	4 por hora	1	71 días
120 Ah	1 minuto	4 por hora	1	10 años
2600 mAh	1 minuto	1 por día	0	9,4 años
3600 mAh	1 minuto	1 por día	0	13 años

Tabla 4.1: Autonomía de la alimentación de la unidad sensora bajo distintos supuestos.

4.2.2 Host

El host del sistema es un servidor compartido contratado a una empresa que presta servicios de hosting y del que desconocemos los detalles del hardware.

4.2.3 Clientes

Los clientes Web pueden acceder al servidor con cualquier dispositivo que tenga instalado un navegador estándar y acceso a internet. Los equipos pueden ser de tipo PC, Tablet, Teléfono móvil, etc.

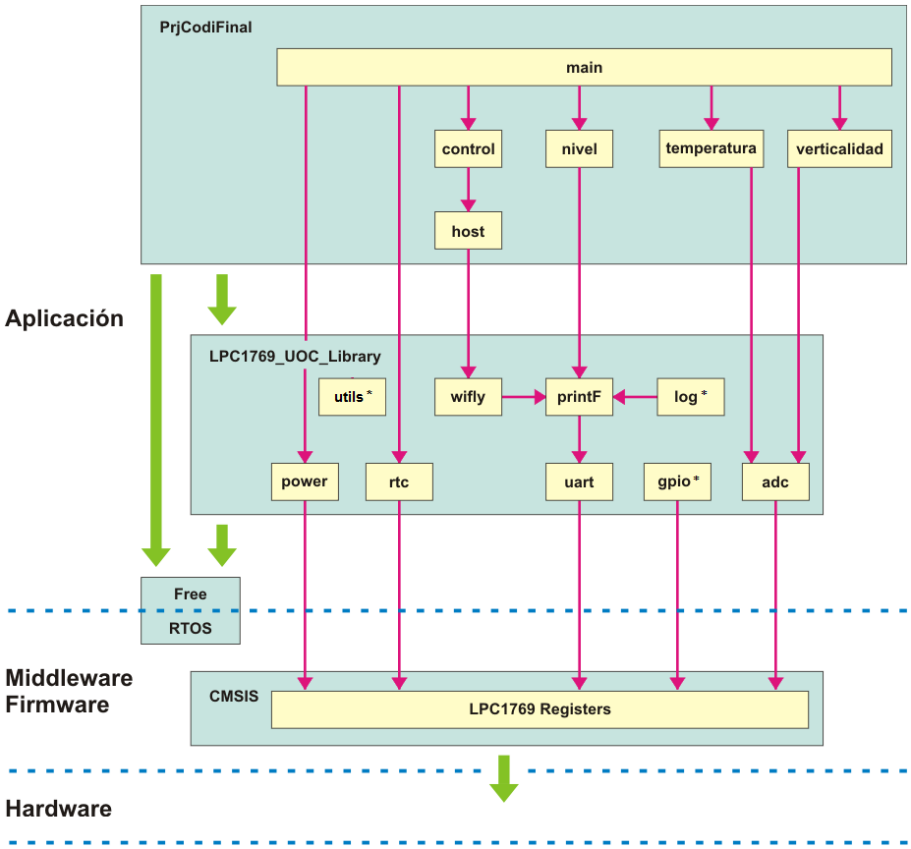
4.3 Software

4.3.1 Unidad sensora

Para la programación del microcontrolador de la unidad sensora, se han utilizado varios bloques de módulos:

- La librería '*CMSISv2p00_LPC17xx'* desarrollada por NXP para la familia de microcontroladores LPC17xx.
- El sistema operativo en tiempo real '*FreeRTOS*'.
- La librería desarrollada durante la preparación de este trabajo '*LPC1769_UOC_Library*', con los módulos necesarios para hacer una abstracción del hardware.
- El programa principal de control '*PrjCodiFinal*'.

En la Figura 4.4 pueden observarse los bloques anteriores, ampliados con los módulos correspondientes al programa y a la librería.



***** El módulo '*gpio'* es utilizado por otros módulos que necesitan leer y/o escribir en las entradas-salidas digitales del micro. Los módulos 'utils' y '*log'* son utilizados por prácticamente el resto de módulos. En los tres casos se han suprimido las flechas de uso en el diagrama para mejorar la legibilidad del mismo.

Figura 4.4: Diagrama de bloques software de la unidad sensora.

4. Descripción detallada

En la librería, a los módulos '*wifly'*, '*printF'*, '*log'* y '*uart'* desarrollados durante el curso, hemos añadido los módulos '*adc'* y '*gpio'* para la lectura y escritura de los puertos analógicos y digitales respectivamente del micro.

Además, se ha creado también el módulo '*power'* que permite pasar al micro a cualquiera de los modos de bajo consumo posibles y el módulo '*rtc'* (real timer control) que tiene dos cometidos; por una parte, gestionar la programación del tiempo entre *wake-ups* del micro, y por otra organizar la memoria del reloj de tiempo real, que es la única que queda soportada por la alimentación en los modos de bajo consumo, permitiendo salvar los datos que necesitan ser persistentes entre *wake-ups*.

Los datos que salvamos son los siguientes:

- La temperatura leída en el último ciclo activo, necesaria para compararla con el valor obtenido en el ciclo actual y de esta forma poder elaborar la alarma (ver apartados 3.2.2 y 4.2.1.3).
- El estado de la alarma por temperatura. Al compararla con el estado en el ciclo actual, permite conocer si se ha producido un flanco positivo o negativo, en cuyo caso tendremos que notificárselo al host.
- El número consecutivo de ciclos con alarma por verticalidad sin consolidar. Nos permite consolidar la alarma como activa solamente después del número de ciclos previstos para ello y evitar de esta forma falsas alarmas debido a la descarga del contenedor (ver apartado 3.2.1).
- El estado consolidado de la alarma por pérdida de verticalidad. Como en el caso de la temperatura, nos permite detectar los flancos en los cambios de estado y, en su caso, notificarlo al host.
- El nivel de los residuos. Utilizado para detectar diferencias negativas entre valores leídos en ciclos consecutivos y determinar de esta forma si se ha producido o no una descarga del contenedor.
- Los últimos valores de ajuste recibidos desde el host; el de intervalo de petición al host de los propios datos de ajuste y el de intervalo de lectura del nivel de residuos.
- Los tiempos (ciclos) acumulados desde la última petición de datos al host y desde la última medida de nivel. Junto a los datos de ajuste del punto anterior, permiten saber cuándo hacer una nueva petición al host y cuándo hacer una nueva medida del nivel de residuos.

Además de los módulos de la librería comentados, en el programa principal se han creado los módulos que se encargan a más alto nivel de la gestión y el control de la lógica de la unidad sensora.

Para entender el cometido de estos módulos, podemos relacionarlos con las actividades mostradas en el diagrama de flujo de la Figura 4.5:

- El módulo *'main'* se corresponde en el diagrama de flujo con la zona etiquetada como *'Secuencia de inicio'*.
- Los módulos *'nivel'*, *'temperatura'*, *'verticalidad'* y una parte del módulo *'control'* se corresponden con la zona *'Lectura y tratamiento de datos'*.
- Finalmente, la otra parte del módulo *'control'* y el módulo *'host'* se corresponden con la zona que hemos denominado '*Comunicación con el host'.*

Diagrama de flujo

La figura 4.5 muestra el diagrama de flujo correspondiente a la unidad sensora. Está dividido en cuatro zonas que repasamos a continuación.

- *Secuencia de inicio:*

  El arranque del micro puede deberse a la conexión de la alimentación o a un *wake-up* posterior a un periodo en modo *sleep*. Para saber a partir de cuál de las dos situaciones se ha producido el inicio, consultamos un bit especial del reloj de tiempo real (en lo sucesivo RTC) que indica si ha perdido la alimentación durante el periodo en *sleep* o no. En función de que sea o no el primer ciclo después de una pérdida de alimentación, actuamos de distinta forma.

  En la lectura del diagrama, se ha de tener en cuenta que el periodo para actualización de datos de ajuste es siempre múltiplo entero del periodo de lectura de nivel, por lo tanto, el refresco de los datos de ajuste implica siempre una lectura del nivel de residuos. A su vez, una lectura del nivel de residuos, implica el envío al host de los datos de proceso.

  En todos los ciclos se lee el valor de la verticalidad, de la temperatura y se elabora el estado de sus respectivas alarmas.

- *Lectura y tratamiento de datos:*

  En esta zona se leen las variables físicas y se gestiona toda la lógica del programa. Trabajan tres '*tasks'* de tipo productor, cada una de ellas dedicadas a una variable, y una '*task'* de tipo consumidor encargada de recoger los datos que generan las '*tasks'* productoras.

  Aquí hemos de advertir que la '*task'* dedicada a la lectura y gestión del nivel de residuos, se crea y trabaja con independencia de que toque leer el nivel o no. Si no toca leer el nivel, lo notifica a la '*task'* consumidora e inmediatamente pasa al estado de suspensión.

4. Descripción detallada

Cada '*task*', una vez ha leído el valor de la variable física con la que trabaja, se encarga de deshabilitar o cortar la alimentación del sensor correspondiente.

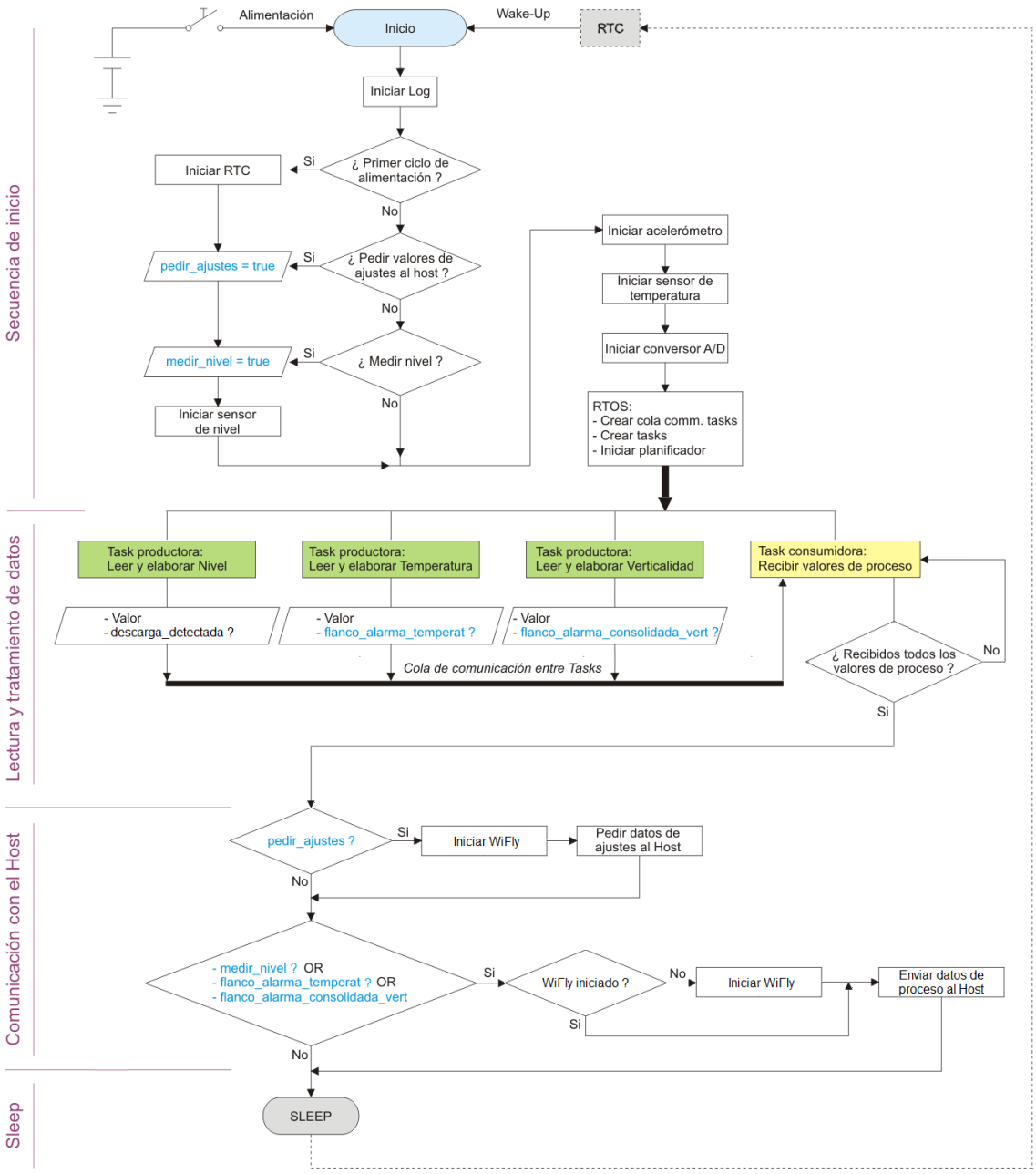


Figura 4.5: Diagrama de flujo de la unidad sensora.

Para el paso de valores a través de la cola, hemos creado una estructura de datos común a las tres variables físicas de interés, con independencia de la naturaleza de cada una. La mostramos a continuación:

```
typedef struct  {
    uint8_t id;             // Identificador del tipo de dato (nivel, temperatura o verticalidad).
    int16_t value;          //  Valor del dato.
    bool_t alarm;           // ¿ Alarma activa ?
    bool_t alarmFlange;     // ¿ Cambio del estado de alarma en el ciclo actual (flanco) ?
} sensorData_t;
```

- *Comunicación con el host:*

  Si es necesario en el ciclo actual, se inicia el WiFly, se conecta con el servidor y se pide y/o se envía la información correspondiente. En el apartado 4.4, dedicado al protocolo de comunicación, se amplía la información relativa a esta fase.

- *Sleep:*

  Finalmente, todos los elementos de la unidad sensora, incluido el propio microprocesador, pasan al estado de bajo consumo en el que se mantendrán durante el periodo de tiempo programado en el RTC. Pasado dicho periodo, el RTC despertará al micro y comenzará un nuevo ciclo.

4.3.2 Host

Para describir la aplicación desarrollada en el host, nos serviremos del diagrama de casos de uso mostrado en la Figura 4.6.

Su cometido fundamental es el de recibir los datos de proceso que envía la unidad sensora, tratarlos y ponerlos a disposición de los usuarios para su explotación. También se encarga de enviar a la unidad sensora las variables de ajuste del sistema y de gestionar la interfaz de usuario que permite la comunicación hombre-máquina del sistema.

En el apartado 3.3 se pueden revisar los recursos con los que trabajamos en el servidor, a los cuales nos referiremos ocasionalmente en los siguientes párrafos.

La administración de la base de datos no está incluida en la interfaz desarrollada. Se realiza con un gestor gráfico de bases de datos (MySQL Workbench).

Tanto la administración de la base de datos como los usuarios Web, necesitan autentificarse para poder trabajar con en el sistema. La unidad sensora trabaja sin autentificación.

A partir de los datos de nivel y de descarga del contenedor recibidos desde la unidad sensora, la aplicación genera las alarmas correspondientes, comparándolos contra los valores introducidos por el usuario en la base de datos en el proceso de edición de ajustes.

El estado las alarmas anteriores, junto al estado de las alarmas de temperatura y verticalidad, recibidas directamente desde la unidad sensora, permiten tomar la decisión de envío de los e-mails de alarmas a los usuarios que estén dados de alta con este fin en la base de datos.

4. Descripción detallada

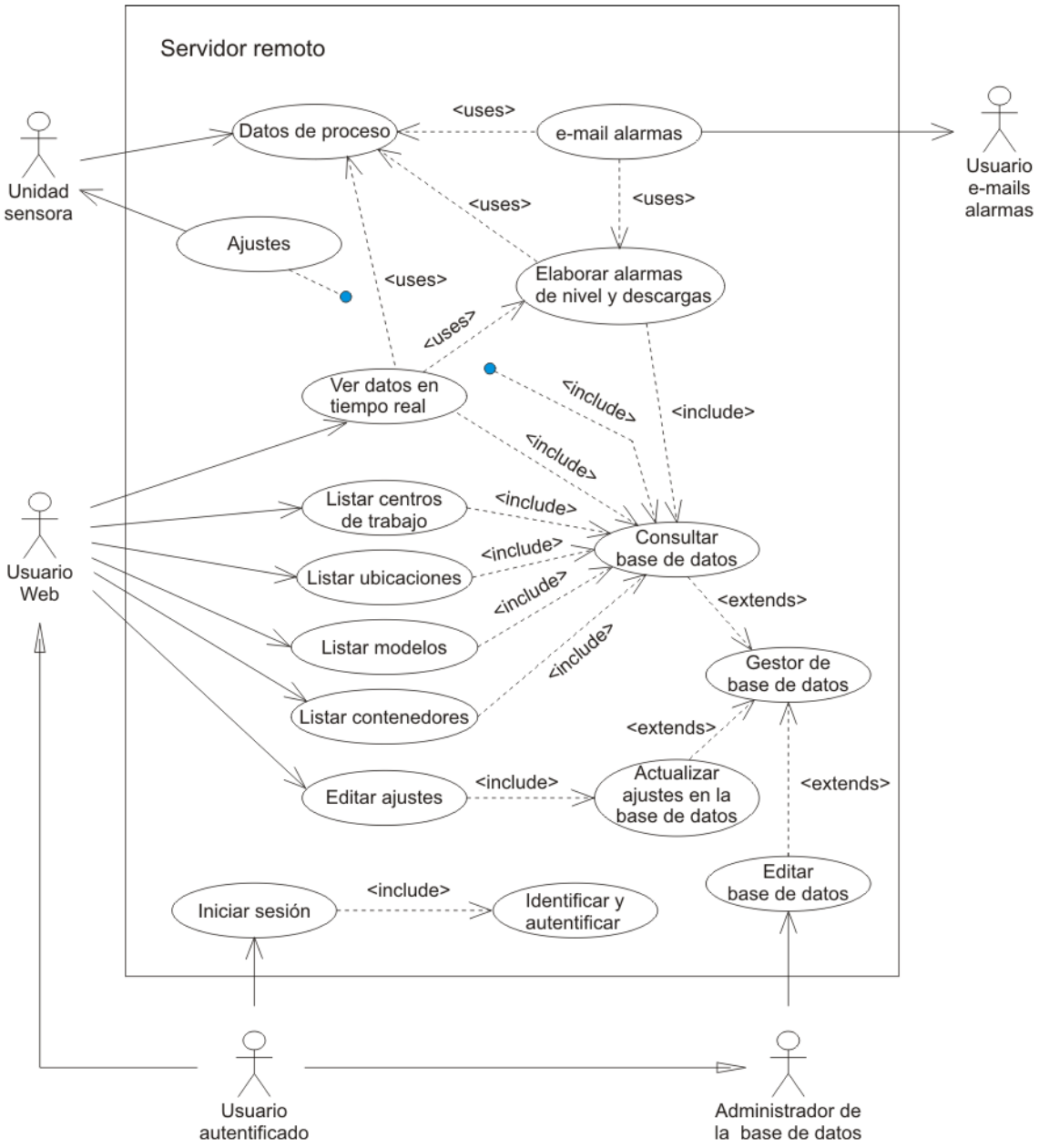


Figura 4.6: Diagrama de casos de uso de la aplicación del servidor.

Los datos de ajuste que puede editar el usuario Web son:

- El periodo de actualización de datos de ajuste en la unidad sensora.
- El periodo de medida del nivel de los residuos.
- El porcentaje de llenado del contenedor para disparar la alarma correspondiente.
- La cantidad de descargas acumuladas para disparar la alarma de mantenimiento.

Los dos primeros datos se incluyen en el envío de ajustes a la unidad sensora.

Una vez finalizado el mantenimiento del contenedor, desde la propia interfaz de usuario se puede poner a cero la cantidad de descargas acumuladas para comenzar la contabilización de un nuevo periodo de mantenimiento.

Los datos de proceso que se muestran en el formulario de tiempo real son:

- El nivel de llenado, la temperatura, la verticalidad y la cantidad de descargas acumuladas actualmente.
- El estado de las alarmas correspondientes a las cuatro variables anteriores.

4.4 Protocolo de comunicación

Como ya adelantamos en el capítulo dos, hemos implementado un protocolo para la comunicación entre la unidad sensora y el host basado en el estilo de arquitectura 'REST' (apartado 2.1.2.4), en el que cada variable está definida por un identificador. Hemos añadido la posibilidad de que el host, a petición de la unidad sensora, pueda enviar una respuesta de confirmación de las tramas recibidas y hemos incluido para todos los mensajes un CRC que permite comprobar si el intercambio de datos es correcto.

Algunas características generales del protocolo que utilizamos son:

- Todas las comunicaciones entre el host y la unidad sensora, las inicia siempre la unidad sensora para minimizar los tiempos de espera por parte de la misma y así mejorar su consumo.
- Para el cálculo de los CRCs, utilizamos el polinomio CRC-16 XModem 1021_h. La ecuación generadora es: $x^{16} + x^{12} + x^5 + 1$, con valor inicial o semilla igual a 0000_h.
- Trabajamos con direcciones IPV4 codificadas en 32 bits según se muestra en el ejemplo de la Figura 4.7.

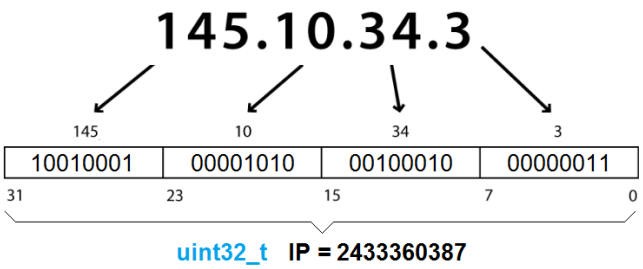


Figura 4.7: Codificación de direcciones IPV4.

- Utilizamos el carácter '*&*' como separador para los pares identificador-valor de las distintas variables.

- La URL de destino de los mensajes originados en la unidad sensora, comienza con una parte común para cualquier tipo de mensaje que incluye el nombre del host, la estructura de directorios que contienen el fichero encargado de gestionar el mensaje, el nombre del propio fichero y el identificador de la variable que define el tipo de mensaje. En la Figura 4.8 se muestra la parte común de la URL.

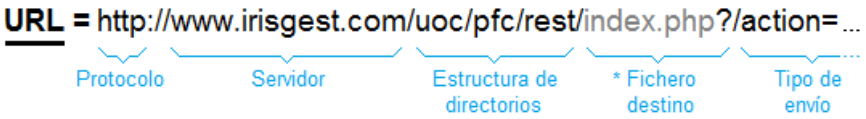


Figura 4.8: Inicio de la URL de destino.

Puede suprimirse el fichero de destino debido a que su nombre *'index.php'* coincide con el nombre del destino por defecto en ausencia de una referencia al mismo.

- Como se aprecia en la Figura 4.9, hay dos situaciones en las que se produce el intercambio de información:
    - Petición de los valores de ajuste desde la unidad sensora con la correspondiente respuesta desde el host.
    - Envío de los datos de proceso desde la unidad sensora al host. El envío puede incluir la petición de respuesta de confirmación por parte del host o no.

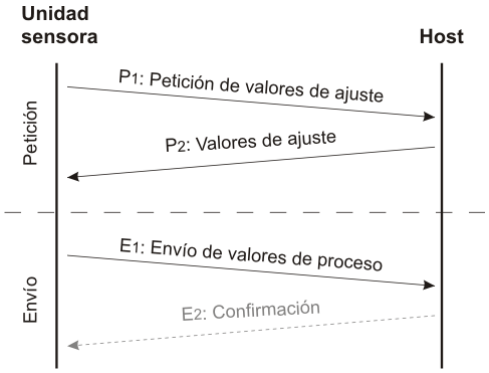


Figura 4.9: Tipos de mensajes.

En las páginas siguientes, detallamos cada una de los escenarios anteriores.

4.4.1 Petición-respuesta de valores de ajuste:

La unidad sensora envía una petición de los valores de ajuste al servidor en cualquiera de las dos circunstancias siguientes:

- Es alimentada por primera vez y necesita conocer los datos de ajuste para poder trabajar. En este caso, hasta que no reciba los valores de ajuste desde el host, no comienza a enviar datos de proceso.
- Ha transcurrido el intervalo de tiempo previsto para el refresco de los valores de ajuste. La unidad sensora continúa enviando datos de proceso basados en los datos de ajuste de los que dispone actualmente.

En ambos casos, si no se obtiene la respuesta desde el servidor después de un tiempo ajustado en el programa, la petición se repite cada minuto.

En la Figura 4.10 se muestra la estructura del mensaje de petición y en la Tabla 4.2 se hace un resumen de los distintos campos.

El bloque etiquetado como '*URL*' (con fondo rosa), hace referencia a la parte inicial común de la URL de destino (ver Figura 4.8). La variable *'action'* indica que el mensaje es de tipo '*request*' o petición. Se incluye la dirección IP de la unidad sensora para que el host pueda identificarla como emisora del mensaje y a continuación se incluyen el/los identificadores de las variables objeto de la petición. El mensaje finaliza con el CRC calculado.

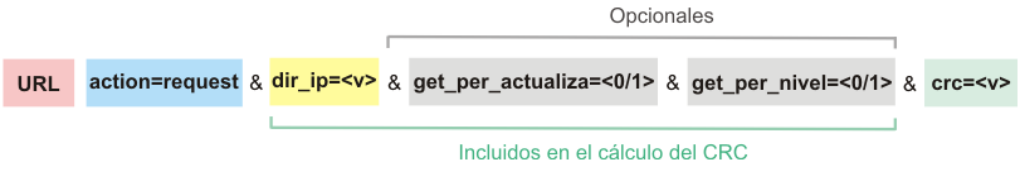


Figura 4.10: Petición de valores de ajuste desde la unidad sensora al host.

Clave	Valor	Inclusión	Formato
action	request	obligatoria	---
dir_ip	<v>	obligatoria	Dirección IP de la unidad sensora codificada en 4 bytes.
get_per_actualiza	<0/1>	opcional	Si clave presente y valor = 1, se está solicitando al host el periodo de actualización de ajustes.
get_per_nivel	<0/1>	opcional	Si clave presente y valor = 1, se está solicitando al host el valor del periodo de lectura de nivel.
crc	<v>	obligatoria	Calculado desde 'dir_ip' (incluido) hasta el byte anterior (incluido) a la clave 'crc'.

Tabla 4.2: Resumen del formato de petición desde la unidad sensora.

La respuesta desde el host incluye explícitamente la dirección IP de la unidad sensora, el valor de la/s variables solicitadas en la petición y el CRC calculado sobre los campos del mensaje enviado. En la Figura 4.11 y en la Tabla 4.3 se visualiza y resume el formato de la respuesta.

4. Descripción detallada

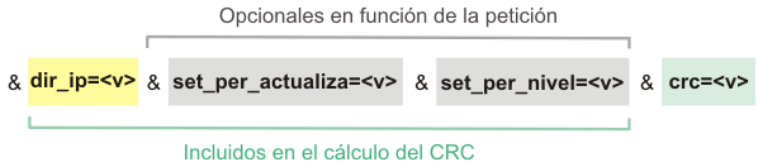


Figura 4.11: Respuesta del host a la petición de valores de ajuste.

Clave	Valor	Inclusión	Formato
dir_ip	<v>	obligatoria	Dirección IP de la unidad sensora codificada en 4 bytes.
set_per_actualiza	<v>	si solicitado en la petición	Periodo de actualización de ajustes en minutos.
set_per_nivel	<v>	si solicitado en la petición	Periodo de lectura de nivel en minutos.
crc	<v>	obligatoria	Calculado desde 'dir_ip' (incluido) hasta el byte anterior (incluido) a la clave 'crc'.

Tabla 4.3: Resumen del formato de respuesta del host.

Un ejemplo de petición de los parámetros de ajuste podría ser:

"http://irisgest.com/uoc/pfc/rest/?action=request&dir_ip=3232235812&get_per_actualiza=1&get_per_nivel=1&crc=47686".

En la petición se está solicitando el valor del periodo de actualización de datos y el de lectura de nivel. Si todo va bien, la respuesta del host será del estilo:

"&dir_ip=192.168.1.36&set_per_actualiza=60&set_per_nivel=15&crc=52376".

En la respuesta se observa que el periodo para petición de valores de ajuste es de 60 minutos y el de lectura de nivel de 15 minutos.

Si el host detecta un error de CRC, la respuesta será: "&crc=???".

Si la dirección IP de la unidad sensora incluida en la petición, no está dada de alta en la base de datos del servidor, la respuesta será: "&id=???".

4.4.2 Envío-confirmación de los valores de proceso:

A la parte común descrita anteriormente (Figura 4.8) y representada en la Figura 4.12 por el cuadro inicial con fondo rosa, se le unen los pares clave/valor de las variables que completan el mensaje.

En este caso, la variable 'action' contiene el valor 'send', indicando con ello al servidor que éste es un envío de los valores de proceso.

A continuación, se concatena la dirección IP de la unidad sensora (cuadro con fondo amarillo) para que el host pueda identificarla como emisora del mensaje.

Las variables de proceso propiamente dichas son las cinco que vienen a continuación, nivel, temperatura, verticalidad, alarmas y vaciado del contenedor.

Finalmente, la solicitud de respuesta de confirmación y el CRC finalizan el mensaje.

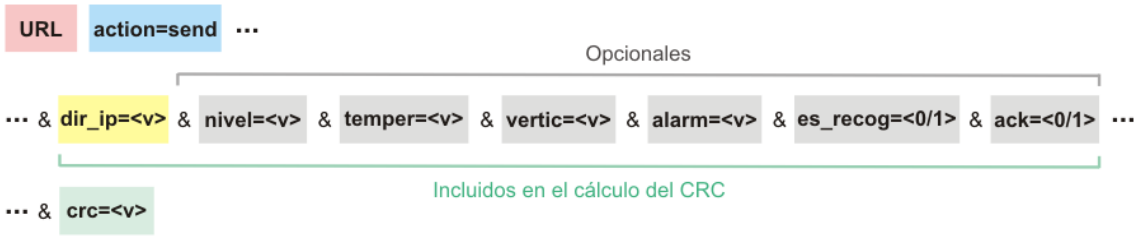


Figura 4.12: Envío de valores de proceso desde la unidad sensora al host.

Clave	Valor	Inclusión	Formato
action	send	obligatoria	---
dir_ip	<v>	obligatoria	Dirección IP de la unidad sensora codificada en 4 bytes.
nivel	<v>	opcional	Valor absoluto de la distancia a los residuos en cm. Coma fija, 1 decimal.
temper	<v>	opcional	Valor absoluto de la temperatura en grados centígrados. Coma fija, 1 decimal.
vertic	<v>	opcional	Valor absoluto de la inclinación en grados respecto de la gravedad terrestre. Coma fija, 1 decimal.
alarm	<v>	opcional	Valor: 1 Byte codificado por bits: bits 7-2: 0 (reservados). bit 1: Alarma de verticalidad, 1 = alarma. bit 0: Alarma de temperatura, 1 = alarma.
es_recog	<0/1>	opcional	Si clave presente y valor = 1, se acaba de descargar el contenedor.
ack	<0/1>	opcional	Si clave presente y valor = 1, el envío requiere respuesta de reconocimiento desde el host.
crc	<v>	obligatoria	Calculado desde 'dir_ip' (incluido) hasta el byte anterior (incluido) a la clave 'crc'.

Tabla 4.4: Resumen del formato de envío de los valores de proceso desde la unidad sensora.

Si la unida sensora solicita la confirmación de los valores enviados (ack=1), el host responderá con un mensaje del tipo que se muestra en la Figura 4.13. Incluye la dirección IP de la unidad sensora y el mismo CRC que se envió junto a los datos.

& dir_ip=<v> & crc=<v>

Figura 4.13: Respuesta del host a un envío de valores de proceso que incluye petición de confirmación.

Un ejemplo de envío al host de las variables de proceso podría ser:

"http://www.irisgest.com/uoc/pfc/rest/?action=send&dir_ip=3232235812&nivel=773&temper=223&vertic=91&alarm=0&es_recog=0&ack=1&crc=39452".

En este caso, el valor del nivel es de 77.3 cm., la temperatura es de 22.3ºC, el contenedor está inclinado 9.1º en alguna dirección respecto de la gravedad terrestre, no hay alarmas, no hay descarga y se está solicitando una respuesta de confirmación al host.

Como el envío incluye una petición de confirmación, el host responderá con:

"&dir_ip=192.168.1.36&crc=39452".

La dirección IP es la del contenedor decodificada y el CRC el mismo que se envió, que a su vez ha sido comprobado por el servidor. Todo ha ido bien.

Si el host hubiese detectado un error de CRC, la respuesta sería: "&crc=???".

Si la dirección IP de la unidad sensora incluida en el envío, no hubiese estado dada de alta en la base de datos del servidor, la respuesta sería: "&id=???".

5. Viabilidad técnica

A lo largo de todas las pruebas realizadas, no ha surgido ningún problema con el sistema. A nivel de hardware, pese a todas las manipulaciones hechas durante el montaje y medida de consumos, no ha dado ningún fallo y se ha comportado de forma robusta. Lo mismo puede decirse del software, que una vez se consiguió que funcionase libre de errores y según los objetivos marcados, se ha comportado de forma estable tanto en la unidad sensora como en el servidor.

A continuación, enumeramos los principales puntos fuertes y débiles del sistema:

- Puntos fuertes:
    - Pese a ser un prototipo, a nivel funcional el sistema ha demostrado ser capaz de cumplir los objetivos marcados inicialmente, que fueron definidos según las necesidades reales a cubrir.
    - El protocolo implementado es tolerante a fallos gracias a los mecanismos de confirmación de mensajes y comprobación de CRCs. Tanto la unidad sensora como el servidor responden adecuadamente ante la aparición de errores en los datos transmitidos.
    - No utilizamos tecnologías experimentales o excesivamente complicadas. Tanto el hardware como el software están basados en elementos ampliamente utilizados en el trabajo real de campo. Sensores de temperatura, de distancia por ultrasonidos, de gravedad o herramientas como FreeRTOS, Apache o MySql ya han demostrado durante mucho tiempo su robustez y fiabilidad.
- Puntos débiles:
    - La capacidad necesaria de la pila que alimenta a la unidad sensora, hace inviable el poder trabajar con los intervalos de muestreo de datos propuestos inicialmente. Por otra parte, el poder mantener dichos intervalos, nos daría una clara ventaja competitiva sobre productos similares al nuestro (ver Tabla 2.3).
    - En el caso de una aplicación real del sistema que estamos estudiando, en el que los contenedores generalmente estarán situados en lugares sin acceso a una red Wifi, hacen que este medio de comunicación entre la unidad sensora y el host no sea apropiado en la práctica.
    - La utilización de un servidor compartido, hace que no tengamos plena libertad para la elección de las aplicaciones con las que trabajamos, ni para realizar en ellas los ajustes que permitan adaptar su comportamiento a las condiciones óptimas para nuestro sistema.

En el capítulo siguiente, proponemos algunas modificaciones en el sistema experimental, orientadas a eliminar los puntos débiles, manteniendo los puntos fuertes anteriores.

5. Viabilidad técnica

Página intencionadamente en blanco.

6. Sistema comercial

Si queremos convertir el sistema experimental en un producto comercial, lo primero que tendremos que hacer será eliminar los puntos débiles o, mejor aún, convertirlos en puntos fuertes.

Al final del capítulo anterior hemos enumerado los puntos débiles del sistema y a continuación describiremos las modificaciones realizadas.

Respecto a la utilización del servidor compartido que hemos hecho durante el desarrollo del prototipo, la solución es inmediata: utilizar un servidor dedicado.

Aunque resulte más caro, utilizar un servidor dedicado solamente a soportar nuestro sistema eliminaría las carencias que tenemos con el servidor compartido. Además, es una medida que tendríamos que tomar necesariamente en cuanto el número de empresas y de contenedores empezase a aumentar drásticamente. El servidor dedicado, permitirá escalar mucho mejor al conjunto del sistema.

6.1 Arquitectura de red

Para resolver los puntos referentes al consumo y a la red de comunicación, necesitamos centrarnos en un tipo de red disponible en cualquier emplazamiento físico que requiera del menor consumo posible en las unidades sensoras y que proporcione acceso a la red global Internet.

Ya hemos visto que la red Wifi, además de no estar siempre disponible, tiene un consumo elevado de energía eléctrica.

Otro tipo de redes que nos pueden permitir el acceso directo de cada unidad sensora a Internet son las de tipo ‘Low power Wan’ y las de telefonía móvil (ver apartado 2.1.2.1). En este caso sí que ofrecen disponibilidad global, sobre todo las redes de telefonía móvil, pero siguen adoleciendo de un consumo alto de energía.

En vista de lo anterior, vamos a cambiar la estrategia de conectividad. En las unidades sensoras, trabajaremos con una red local de bajo consumo y la conexión de dicha red con Internet la haremos a través de una pasarela a una red con disponibilidad global.

La red local la basaremos en la tecnología ZigBee (ver apartado 2.1.2.2) debido al bajo consumo de sus transceptores y al mínimo tiempo que los nodos necesitan permanecer en estado activo para intercambiar los datos. Además, es una tecnología madura y de bajo coste.

Como pasarela a Internet, utilizaremos las redes de telefonía sin hilos que disponen de una cobertura del 95% a nivel mundial. Aquí sí que el consumo es similar al de un nodo asociado a una red Wifi, pero con dos ventajas definitivas que enumeramos a continuación:

1 – Al no ser necesario montar el hardware directamente sobre los contenedores, se puede fijar sobre elementos adyacentes como paredes, farolas, mobiliario urbano, etc. y ello permite que se pueda alimentar directamente desde la red eléctrica o a través de algún medio alternativo, como por ejemplo una célula fotovoltaica.

2 – El relativamente alto consumo que tendrá, será repercutido sobre el conjunto de contenedores situados en la isla y no será multiplicado por cada uno de ellos, como en el caso del sistema que hemos desarrollado.

En la Figura 6.1 se muestran gráficamente los elementos participantes en la nueva arquitectura.

Hemos de hacer notar que los contenedores se sitúan siempre próximos a algún punto de alumbrado, lo que nos garantizada la disponibilidad de la red eléctrica municipal, con independencia de que se decida utilizar otra fuente de energía como la fotovoltaica.

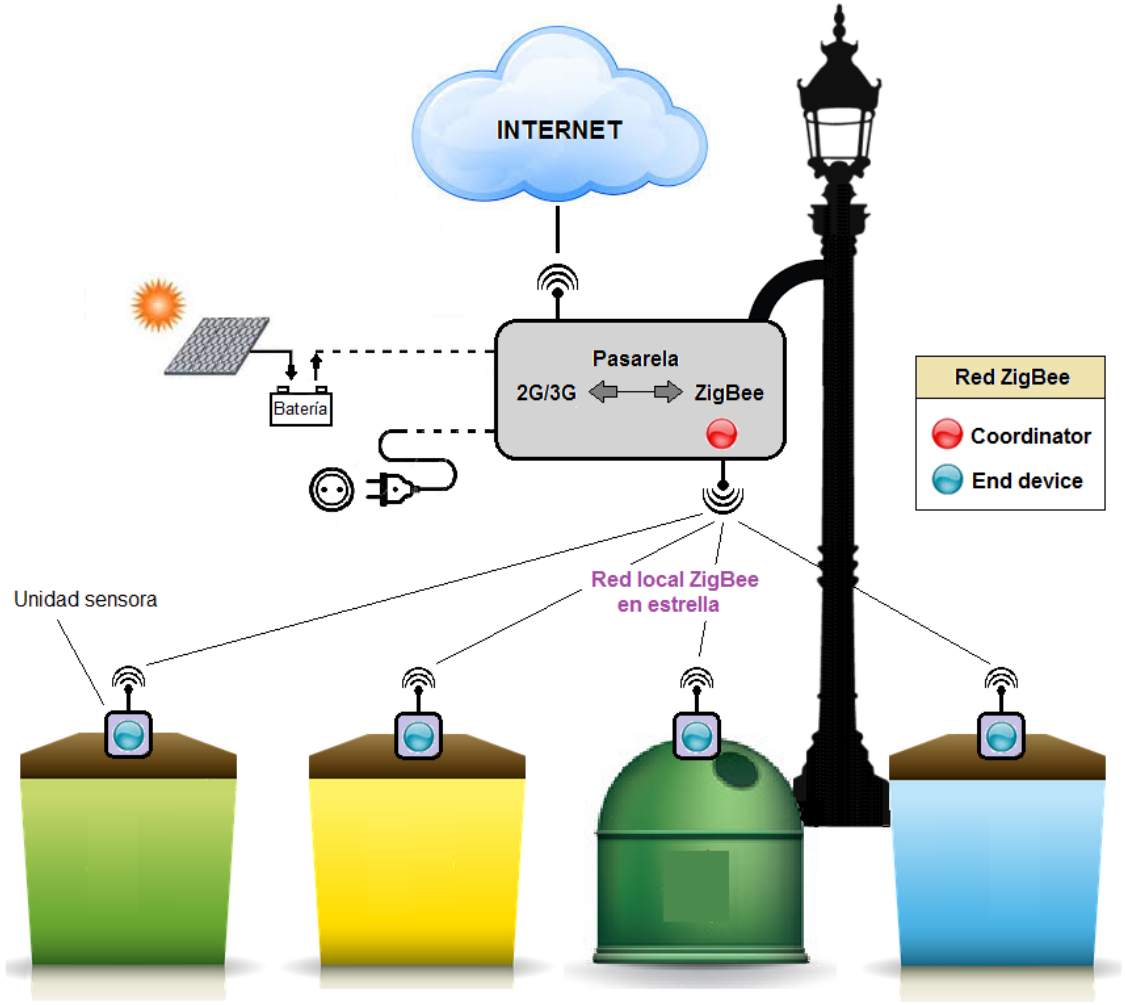


Figura 6.1: Arquitectura de redes del sistema comercial.

6.2 Unidad sensora

Una vez definida la nueva arquitectura de red, vamos a seguir intentando bajar el consumo de la unidad sensora y de paso bajar el precio y el tamaño de la misma. Estos dos últimos factores también son importantes en todo dispositivo hardware, aunque no hayamos hecho referencia explícita a ellos.

6.2.1 Revisión del hardware

Para centrar los temas de este apartado y del siguiente, dedicado a calcular la nueva autonomía de la unidad sensora después de las modificaciones que estamos haciendo, vamos a presentar un elemento hardware dedicado a reemplazar al microprocesador y al WiFly utilizados en del desarrollo original.

Es un dispositivo SoC, acrónimo inglés de '*System on Chip*', que incorpora en un solo IC una CPU para la aplicación con sus correspondientes periféricos más el *'stack'* de comunicación, en este caso ZigBee, otra CPU que dirige la circuitería de RF y finalmente, un controlador para diversos tipos de sensores. Externamente solo es necesario añadir el circuito de antena. Se trata del circuito integrado CC2630 de Texas Instruments [W19].

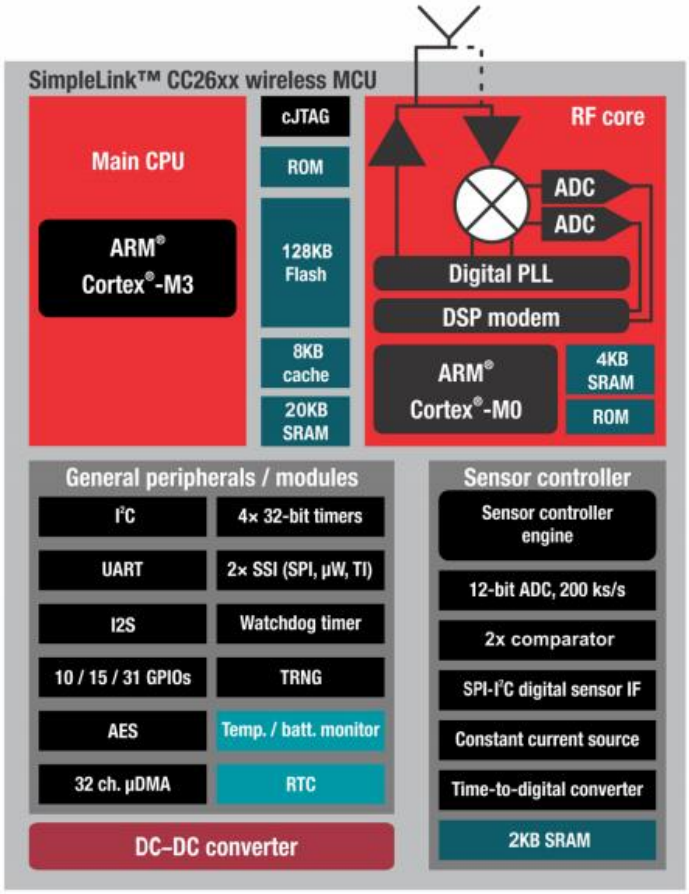


Figura 6.2: Diagrama de bloques del CC2630.

6. Sistema comercial

Hemos de insistir en que no es el único *'chip'* del mercado con características similares y que le citamos aquí solamente para tener una referencia clara en cuanto a posibilidades y a valores de consumo. De lo que si estamos convencidos es de que un producto comercial orientado a dar soluciones dentro del escenario que hemos propuesto en este proyecto, ha de pasar por componentes similares a este, con un alto nivel de integración, bajo consumo y precio moderado teniendo en cuenta la cantidad de componentes que incorpora (3,95€ para 1000 unidades).

En la Figura 6.2 se muestra el diagrama de bloques del CC2630, en el que pueden observarse sus características principales.

A continuación, hacemos un resumen de los parámetros sacados de su hoja de datos, centrándonos exclusivamente en aquellos que tienen influencia sobre el comportamiento de nuestro sistema a nivel de tiempos y/o de consumo.

- ARM Cortex M3 a 48 Mhz, ZigBee speed 250kb/s:
    - Consumo en modo activo: 1.45mA + 31uA/Mhz = 2.94mA
    - Consumo en modo Stand-By (con retención de SRAM y Wake-Up por RTC): 1 uA
    - Ready desde Wake-Up: 151us
    - Corriente de transmisión @3.3v / 5 dbm: 9.1mA
    - Corriente de recepción @3.3v: 5.9mA

En la Figura 6.3 mostramos como quedaría el nuevo diagrama de bloques hardware de la unidad sensora. Puede notarse la simplificación respecto al diagrama original de la Figura 4.2.

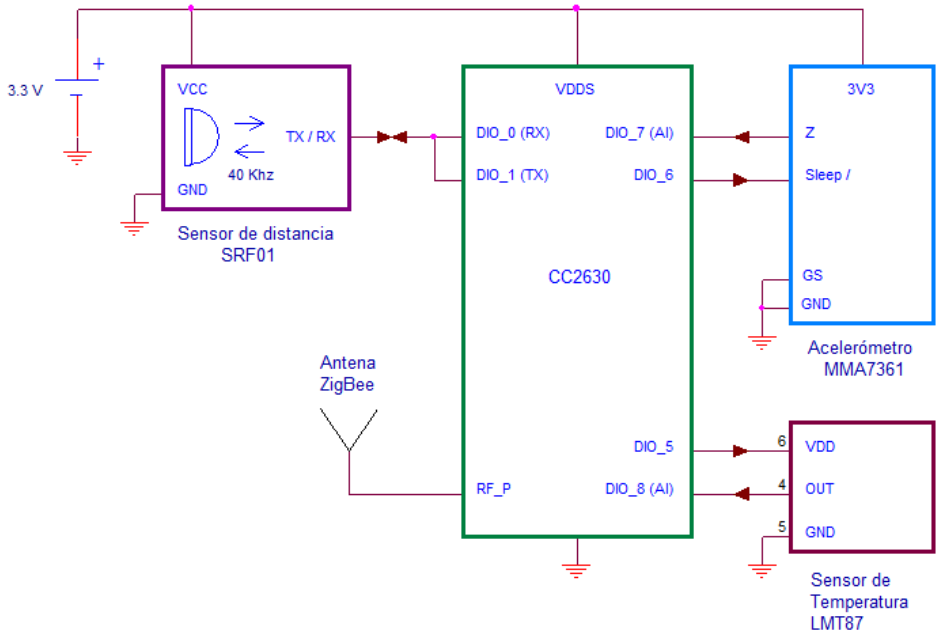


Figura 6.3: Diagrama modificado de bloques hardware de la unidad sensora.

6.2.2 Revisión del consumo y autonomía

Para esta revisión, nos basaremos en los mismos parámetros y supuestos utilizados para el cálculo original (apartado 4.2.1.6).

Consideraremos que la duración de la secuencia activa en un envío o recepción de datos entre el dispositivo final (unidad sensora) y el coordinador(pasarela) es:

16.9 ms + (número de bytes a transmitir o recibir / (velocidad de transmisión en bytes / ms))

Si fijamos una media de 50 bytes por cada transmisión o recepción y sabiendo que la velocidad es 250 kb/ s = 31.25 bytes/ms, tenemos que una secuencia de Tx o Rx durará:

16.9 ms + (50 / 31.25) = 18.5 ms

Recordamos que partimos del cálculo del consumo diario y que la unidad que utilizamos es la de mAs.

A. Intervalo de medida y elaboración del estado de las alarmas de temperatura y verticalidad: 1 minuto.

*(micro + sensor de temperatura + acelerómetro) * 24 * 60 =*
*((2.94mA * (0.151 + 6.0) ms * 10^{-3}) + (7uA * 10^{-3} * 3ms * 10^{-3}) + (0.5mA * 3ms * 10^{-3})) * 1440 = 28.24 mAs*

B. Intervalo de medida del nivel y envío al coordinador de los datos de nivel, temperatura, verticalidad y en su caso descarga: 15 minutos.

*(micro + nivel + ZigBee(Tx)) * 24 * 4 =*
*((2.94mA * (0.151 + 160 + 18.5) ms * 10^{-3}) + (11.1mA * 160ms * 10^{-3}) + (9.1mA * 18.5ms * 10^{-3})) * 96 = 237.1 mAs*

C. Intervalo de recepción de datos de ajuste: 60 minutos.

*(micro + ZigBee(Rx)) * 24 =*
*((2.94mA * (0.151 + 18.5) ms * 10^{-3}) + (5.9mA * 18.5ms * 10^{-3})) * 24 = 3.94 mAs*

D. Supondremos una alarma por temperatura o verticalidad diaria, cuya consecuencia será el envío adicional de los flancos de comienzo y fin al coordinador.

*(micro + ZigBee(Rx)) * 2 =*
*((2.94mA * (0.151 + 18.5) ms * 10^{-3}) + (5.9mA * 18.5ms * 10^{-3})) * 2 = 0.33 mAs*

A los consumos anteriores hay que sumarles el consumo durante el tiempo en modo *'sleep'* de la unidad sensora (consideramos el día entero en modo 'sleep'):

*SLEEP: ((micro(sleep) + acelerómetro(sleep)) * 10^{-3}) * 86400s = (1uA + 4uA) * 10^{-3} * 86400 s = 432 mAs*

6. Sistema comercial

Sumando los consumos parciales, tenemos que el consumo total diario con los parámetros asignados es de 702 mAs/día = **0.195 mAh/día** (miliamperios * hora / día).

Con la pila que nosotros utilizamos en la placa de pruebas, que es de 3.3v / 2600 mAh, tamaño "AA" (50 mm largo x 14.2 mm diámetro), y contando con una auto-descarga del 10% tendríamos una autonomía de 2340 mAh / 0.195 mAh/día ≈ **32.8 años**.

Con el cambio del microprocesador y de la red Wifi por la red ZigBee, hemos multiplicado por 169 la autonomía de la unidad sensora (ver apartado 4.2.1.6).

Ahora sí podemos decir que manteniendo los intervalos de muestreo de datos y los periodos de comunicación con el servidor propuestos inicialmente, la unidad sensora tiene una autonomía de 32.8 años con una pila de 3.3v / 2600 mAh.

La mejora tan espectacular se debe a que:

- Hemos pasado de utilizar un microcontrolador que consumía 56 mA de media a otro que consume 2.9 mA.
- El WiFly consumia 43 mA de media, mientras que ahora, el dispositivo final o 'end device' de la red ZigBee consume 9.1 mA en Tx y 5.9 mA en Rx.
- El tiempo medio de conexión a la red wifi más el servidor ha pasado de los 12 segundos anteriores a los aproximadamente 18.5 ms que tardan en intercambiar los datos el *'end device*' y el '*coordinator*' de la red ZigBee.

No obstante, para reducir el consumo, hemos tenido que añadir al sistema una pasarela ZigBee - 2G/3G, necesaria por otra parte para solucionar la falta de acceso a una red wifi en condiciones reales de trabajo.

7.1 Costes fijos

En la Tabla 7.1 se muestra una relación con los costes fijos del proyecto. Están incluidos los costes de diseño, de certificación del producto y de alojamiento en el servidor.

Estos precios, repercuten en el coste final una sola vez a lo largo de la vida del producto, con independencia de la cantidad total de unidades sensoras fabricadas. La única excepción es el coste de alojamiento en el servidor compartido, que es de periodicidad anual.

Concepto	Cantidad	Precio unitario (€)	Precio total (€)
Diseño del esquema (horas)	12	40	**480**
Diseño de la PCB (horas)	20	40	**800**
Pantalla de montaje	1	160	**160**
Gastos iniciales de fabricación de la PCB (incluye 10 PCBs de prueba)	1	460	**460**
Programación aplicación unidad sensora (horas) (incluye pruebas)	200	40	**8000**
Programación aplicación servidor (horas) (incluye diseño de la base de datos y pruebas)	300	40	**12000**
Certificado CE (incluye certificado de compatibilidad electromagnética y pruebas)	1	3800	**3800**
Certificado IP67 (incluye pruebas)	1	1200	**1200**
Coste anual servidor compartido	1	180	**180**
		Total costes fijos:	**27080,00**

Tabla 7.1: Relación de costes fijos.

7.2 Costes variables

La Tabla 7.2 muestra la relación de costes variables. Repercuten sobre el precio final del producto, en función de la cantidad de unidades sensoras a fabricar. En este caso, estamos suponiendo que vamos a hacer una fabricación de 3000 unidades.

La tabla es auto explicativa y no insistiremos en los datos que contiene.

7. Valoración económica

Concepto	Cantidad	Precio unitario (€) *	Precio total (€)
PCB	3000	4,2	**12600**
Montaje de componentes	3000	2,6	**7800**
Pila 3.3v / 2600 mA	3000	2,6	**7800**
Placa LPC1769	3000	16,4	**49200**
WiFly RN171	3000	22,06	**66180**
Acelerómetro MMA7361	3000	1,43	**4290**
Sensor de temperatura LMT87	3000	0,224	**672**
Detector de distancia por ultrasonidos SRF01	3000	16,99	**50970**
Interruptor de estado sólido FPF2104	3000	0,392	**1176**
Caja envolvente	3000	8,3	**24900**
Componentes varios (condensadores y resistencias)	3000	2	**6000**

Total costes variables:	**231588,00**
Total coste fabricación:	**258668,00**
Coste unitario:	**86,22**
Cantidad Unidades Sensoras a fabricar:	**3000**

Tabla 7.2: Relación de costes variables y costes finales.

7.3 Costes finales

En la Tabla 7.2, también se muestran los costes finales, en función de las unidades sensoras a fabricar.

El coste total de fabricación es la suma de los costes fijos (mostrados en la Tabla 7.1) y de los costes variables.

El coste unitario es el cociente entre el coste total de fabricación y las unidades fabricadas. En este caso, en el que estamos suponiendo una fabricación de 3000 unidades sensoras, el coste final por unidad sería de 86,22 €.

8.1 Conclusiones

Durante la realización de este proyecto, hemos consolidado a nivel práctico los conocimientos que ya poseíamos en las diversas áreas con las que hemos trabajado. Además, hemos entrado en contacto con tecnologías con las que no teníamos ninguna experiencia previa.

Hemos afianzado conocimientos en el campo de los sistemas operativos, de la programación en lenguaje 'C' o de la utilización de entornos integrados de desarrollo, por poner algunos ejemplos.

Entre las herramientas con las que no habíamos trabajado anteriormente, podemos citar al servidor 'Apache', el gestor de base de datos 'MySQL', o los lenguajes de programación PHP y HTML. Este proyecto nos ha permitido entrar en contacto con ellas y adquirir conocimientos sobre su utilización.

Finalmente, pero no menos importante, la propia naturaleza de un proyecto final de carrera, nos ha obligado a adquirir conocimientos y experiencia en la utilización de herramientas de planificación, elaboración de diagramas, redacción de documentos técnicos y presentaciones.

8.2 Autoevaluación

El sistema experimental desarrollado cumple los objetivos que nos propusimos inicialmente (ver apartado 1.3), que a su vez están orientados a solucionar las necesidades que tiene el mercado al que va dirigido.

Hemos procurado ceñirnos todo lo posible a la planificación que hicimos al comenzar el proyecto. Durante la realización del mismo, han surgido algunos factores que han provocado algunas desviaciones sobre dicha planificación (ver apartado 1.5.2). No obstante, hemos podido cumplir con el plazo de entrega final.

La parte más complicada del proyecto se la podemos adjudicar a la programación de la aplicación del servidor, debido a la falta de experiencia que teníamos con las herramientas utilizadas.

Otra etapa compleja ha sido la dedicada a la comunicación entre la unidad sensora y el host. Nos ha obligado a realizar bastantes pruebas y modificaciones hasta conseguir que el WiFly conectase con el servidor remoto, o para ser más exactos, que el servidor admitiese la conexión del WiFly, debido a las políticas de seguridad que tiene establecidas.

Como consecuencia de todo ello, hemos tenido que dedicar al proyecto más tiempo del que habíamos previsto inicialmente, para poder cumplir con el plazo de entrega final. No obstante, como recompensa, hemos visto cumplidos todos los objetivos iniciales, que por otra parte eran bastante ambiciosos.

8. Conclusiones

8.3 Líneas de trabajo futuro

Hemos detectado dos problemas que dificultarían la implantación real del sistema desarrollado: el consumo excesivo de la unidad sensora y la falta de disponibilidad de redes wifi en el entorno físico de trabajo.

En el capítulo seis, dedicado a la conversión del sistema experimental en un sistema comercial, ya hemos propuesto algunas líneas de trabajo que permiten dar solución a los problemas anteriores.

Además, se debe de abordar el tema de la seguridad en las comunicaciones, que no hemos tratado en este trabajo, y hay que mejorar la interfaz de usuario del servidor. En particular, ha de permitir que los usuarios del sistema puedan gestionar directamente la base de datos.

9. Glosario

ADC: Conversor analógico-digital.

Capacidad de una pila: Indica la cantidad de carga eléctrica que almacena. Su unidad es el Ah (amperio hora).

CPU: Unidad central de proceso. Es la parte del microcontrolador dedicada a procesar los datos.

CRC: Sistema de detección de errores usado frecuentemente en redes de comunicaciones y en dispositivos de almacenamiento para detectar cambios accidentales en los datos.

GPIO: Entradas-salidas de propósito general.

IDE: Acrónimo del inglés '**I**ntegrated **D**evelopment **E**nvironment', es una aplicación informática que proporciona servicios integrales para facilitar el desarrollo del software.

Interfaz: Conexión, física o lógica, entre una computadora y un usuario, un dispositivo periférico o un enlace de comunicaciones.

IoT: Internet **o**f **T**hings o Internet de las cosas. El término hace referencia a la interconexión de objetos cotidianos con Internet.

Isla: En el contexto en el que se desarrolla este trabajo, es el lugar físico donde se encuentran colocados uno o más contenedores.

MEMS: Son dispositivos electromecánicos de tamaño muy reducido, utilizados como sensores para la lectura de algunas magnitudes físicas. El acelerómetro utilizado en este proyecto pertenece a este tipo de dispositivos.

RTC: Reloj de tiempo real.

RTOS: Procedente del término inglés '**R**eal **T**ime **O**perating **S**ystem', es un sistema software diseñado para ejecutar aplicaciones en tiempo real.

Sistema empotrado: Es un sistema de computación concebido para cubrir unas necesidades específicas de automatización.

UART: Receptor-transmisor asíncrono universal.

Unidad sensora: En el contexto de este proyecto, es el dispositivo que se coloca en los contenedores para la lectura de las variables físicas de interés y su envío al servidor.

URL: Es una secuencia de caracteres que sigue un estándar y que permite denominar recursos en el entorno de Internet para que puedan ser localizados.

Wifi: Es una tecnología de comunicación inalámbrica destinada a conectar entre sí equipos electrónicos.

10. Bibliografía

10.1 Textos

[1] Using the FreeRTOS Real Time Kernel. NXP LPC17xx Edition.
Richard Barry. Real Time Engineers Ltd. 2011.

[2] The FreeRTOS Reference Manual. API Functions and configuration Options.
Real Time Engineers Ltd. 2015.

[3] Introducción a la ingeniería del software.
Jordi Pradell Miquel y Jose Raya Martos. UOC PID_00171152.

[4] Diseño de bases de datos - Módulo 5.
Dolors Costal Costa. UOC - P05/75002/00537.

[5] Redacció de textos científico tècnics.
Nita Sáenz Higueras y Rut Vidal Oltra. UOC P08/19018/00445

[6] Presentació de documents i elaboració de presentacions.
Roser Beneito Montagut. UOC P08/19018/00446.

[7] The Internet of Things: Key Applications and Protocols, 2nd edition.
Oliver Hersent, David Boswarthick, Omar Elloumi. ISBN: 978-1-119-99435-0.

[8] Murach's PHP and MySQL (2nd Edition).
Joel Murach and Ray Harris. Mike Murach & Associates. ISBN 978-1-890774-79-0.

[9] Murach's HTML5 and CSS3 (3rd Edition).
Anne Boehm and Zak Ruvalcaba. Mike Murach & Associates. ISBN 978-1-890774-83-7.

10.2 Referencias Web

Nota: Las fechas corresponden a la primera consulta realizada. En la mayoría de los casos se han hecho consultas posteriores.

[W1] Placa de desarrollo del LPC1769: http://www.nxp.com/products/software-and-tools/hardware-development-tools/lpcxpresso-boards/lpcxpresso-board-for-lpc1769:OM13000
Fecha: 16/03/2015

[W2] Módulo WiFly: https://www.sparkfun.com/products/10822
Fecha: 10/03/2015

[W3] Acelerómetro MMA7361: http://www.nxp.com/products/sensors/sensor-toolbox/1.5g-6g-3-axis-analog-output-acceleration-sensor:MMA7361LC
Fecha: 22/03/2015

10. Bibliografía

[W4] Sensor de temperatura LMT87: http://www.ti.com/product/LMT87
Fecha: 22/03/2015

[W5] Detector de distancia por ultrasonidos PmodMaxSonar (EZ1):
http://www.maxbotix.com/Ultrasonic_Sensors/MB1014.htm
Fecha: 22/03/2015

[W6] Detector de distancia por ultrasonidos SRF01:
https://www.robot-electronics.co.uk/htm/srf01tech.htm
Fecha: 22/03/2015

[W7] Interruptor de estado sólido FPF2104: https://www.fairchildsemi.com/products/power-management/advanced-load-switches/advanced-current-limited-load-switches/FPF2104.html
Fecha: 22/03/2015

[W8] Adaptador UART-USB CP2102:
https://www.silabs.com/Support%20Documents/TechnicalDocs/CP2102-9.pdf
Fecha: 10/03/2015

[W9] LPCXpresso: https://www.lpcware.com/lpcxpresso
Fecha: 02/03/2015

[W10] Apache HTTP Server Documentation: https://httpd.apache.org/docs/2.4/en/
Fecha: 14/04/2015

[W11] MySQL Reference Manual: http://dev.mysql.com/doc/
Fecha: 14/04/2015

[W12] MySQL Workbench: https://www.mysql.com/products/workbench/
Fecha: 20/05/2015

[W13] Manual de programación PHP: http://php.net/manual/es/
Fecha: 14/04/2015

[W14] NetBeans: https://netbeans.org/
Fecha: 18/05/2015

[W15] Cliente FTP FileZilla: https://filezilla-project.org/
Fecha: 15/10/2015

[W16] Emulador de terminal Putty: http://www.putty.org/
Fecha: 10/03/2015

[W17] Sistema Operativo FreeRTOS: http://www.freertos.org/
Fecha: 10/03/2015

10. Bibliografía

[W18] Fire and forget:
http://docs.oracle.com/cd/E17904_01/doc.1111/e17363/chapter05.htm#FPCON248
Fecha: 10/05/2016

[W19] Texas Instruments CC2630: http://www.ti.com/product/CC2630
Fecha: 20/05/2016

[W20] IETF - Internet Engineering Task Force: https://www.ietf.org/
Fecha: 20/12/2015

[W21] ZigBee Alliance: http://www.zigbee.org/
Fecha: 22/12/2015

[W22] MQTT Protocol: http://mqtt.org/
Fecha: 22/12/2015

[W23] DDS Protocol: http://portals.omg.org/dds/
Fecha: 22/12/2015

Información general:

Wikipedia: https://es.wikipedia.org/wiki/Wikipedia:Portada
Fecha: 01/03/2015

Residuos urbanos:
https://es.wikipedia.org/wiki/Residuos_s%C3%B3lidos_urbanos_en_Espa%C3%B1a
Fecha: 08/03/2015

Tipos de pilas: https://es.wikipedia.org/wiki/Anexo:Tipos_de_pila
Fecha: 10/05/2015

Cálculo de vida de una pila: http://oregonembedded.com/batterycalc.htm
Fecha: 02/06/2016

Protocolos '*IoT*':

Eclipse: https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php
Fecha: 12/12/2015

'Embedded Computing: http://embedded-computing.com/articles/internet-things-requirements-protocols/ Fecha: 15/12/2015

InfoWorld: http://www.infoworld.com/article/2972143/internet-of-things/real-time-protocols-for-iot-apps.html Fecha:15/12/2015

Electronic Design: http://electronicdesign.com/iot/understanding-protocols-behind-internet-things Fecha: 15/12/2015

11.1 Manual de usuario de la Web y del email receptor de los mensajes de alarma.

Para acceder a la Web, escribir en la barra de direcciones del navegador:

http://www.irisgest.com/uoc/pfc/

Desde la pantalla inicial seleccionar la opción 'Login' y una vez en la pantalla de 'Login', introducir:

E-Mail: tfc.uoc@email.com
Password: pa55word

Aparecerá otra pantalla con el listado de los centros de trabajo y ubicaciones de las islas de contenedores pertenecientes al centro seleccionado.

Para ver los datos en tiempo real, hay que seleccionar como centro de trabajo 'Barcelona Sud' y como ubicación 'C. dels Motors, 107'.

Se abrirá una pantalla con el listado de los contenedores colocados en la ubicación seleccionada (en este caso solo uno). A continuación, seleccionar el contenedor, y se abrirá una pantalla mostrando sus datos en tiempo real.

Para hacer los ajustes de trabajo, en la pantalla con los datos de tiempo real anterior, seleccionar 'Ajustes' y aparecerá otra pantalla con el listado de modelos de contenedor (en este caso un solo modelo).

Seleccionando el modelo de contenedor, accederemos al formulario que nos permitirá editar los parámetros de ajuste.

Para acceder al email que recibe los mensajes de alarma, ir a la dirección siguiente:

http://www.mail.com/int/

Los datos de acceso son los mismos que para la Web.

Una vez dentro, se podrán consultar los emails con los mensajes de alarma enviados por la unidad sensora.

11.2 Enlaces a algunos fabricantes de semiconductores en su actividad relacionada con el '*IoT*'.

Texas Instruments, ST Microelectronics, Microchip, Intel, Toshiba, Qualcomm, Freescale, Infineon (seguridad), Analog Devices, NXP, Marvell, Samsung, ON Semiconductor, Atmel.

11. Anexos

11.3 Diagrama EER de la base de datos desarrollada en el servidor.