Escuela Politécnica Superior

Monitorización remota de las condiciones ambientales de un entorno delimitado

Grado en Ingeniería en Sonido e Imagen en Telecomunicación

Trabajo Fin de Grado

Autor:
Joaquín Alavés Sempere

Tutor/es:
Ángel Grediaga Olivo

Junio 2015

Universitat d'Alacant
Universidad de Alicante

Índice

Índice de figuras

Índice de tablas

1. Motivación

El impacto de la calidad medioambiental en el entorno es un aspecto que debe ser tenido en cuenta en muchas ocasiones. A día de hoy, la protección del medio ambiente recibe una considerable atención y cada uno de los aspectos relacionados con esta protección requiere una monitorización de parámetros físicos o químicos. Cualquier programa de control medioambiental debe pasar por la determinación en tiempo real de los valores o concentraciones de las magnitudes a estudiar. El objetivo del presente proyecto será la construcción de un sistema de monitorización de las condiciones ambientales de un entorno centrándonos en unos parámetros concretos que podrán ir ampliándose con el tiempo sin necesidad de realizar cambios drásticos en el diseño. El propósito, por tanto, es obtener los datos recogidos por el sensor/es mediante la ejecución de un programa diseñado en el IDE de Waspmote y transferirlos de forma inalámbrica. Una vez recibidos se procederá a su tratamiento mediante un programa hecho en MatLab que presentará los resultados en una gráfica. En el siguiente diagrama se puede ver el desarrollo general del proyecto:

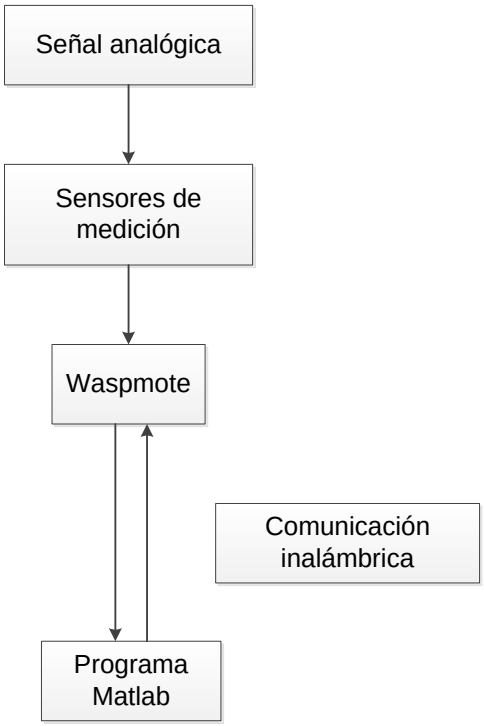

Fig. 1: Diagrama de bloques del proyecto

Las características de las redes de sensores inalámbricas (*Wireless Sensor Networks* o WSN), permite que los sensores que las forman sean desplegados fácilmente sobre el terreno. Esto, unido al bajo coste de cada uno de ellos, permite el emplazamiento de un gran número de dispositivos que actúan como nodos de la red, y hacen de ellas herramientas idóneas para realizar un control preventivo del medio ambiente y para la monitorización ambiental que en determinados momentos pueden utilizarse en estudios de campo [1].

En un futuro, disponer de herramientas de control preventivo que monitoricen en tiempo real las condiciones ambientales y que en un momento determinado el sistema sea capaz de generar una alarma en función de los valores obtenidos enviando la información al usuario a través de web o de una app, por ejemplo, o la posibilidad de realizar mapas de evolución de las variables estudiadas, supone la obtención de una información que estructurada y analizada de forma correcta puede permitirnos la estimación de altos niveles de toxicidad en el aire, prevención de incendios forestales, etc.

Con lo explicado, los objetivos del presente proyectos son:

- Realizar un estudio sobre las diferentes posibilidades de comunicación inalámbrica en función de la distancia requerida. Recabar información sobre los sistemas existentes y sus funciones.

- Estudiar los dispositivos que pueden formar parte del sistema y discutir las posibles mejoras.

- Diseño de los sensores autónomos para los parámetros a monitorizar.

- Diseño del sistema central y módulo de presentación de datos.

- Realización de un prototipo operativo. Reflexionar sobre el uso y las ventajas que se obtienen de este sistema.

2. Estado actual del sistema

Una WSN es un conjunto autónomo de nodos móviles, conectados por enlaces inalámbricos llamados nodos o motas[1] que no precisan de una infraestructura fija para operar. En un principio se idearon para operar en ambientes hostiles e irregulares, estando capacitadas para cambiar su topología para obtener nuevas formas y mantener la misma funcionalidad en la red. En general la información recogida es enviada a un elemento *sink*[2] de la red que actúa de pasarela, trasladando la información de la red al usuario.

En la figura 2 se puede ver un ejemplo de red de sensores inalámbricos, que generalmente consiste en una red de adquisición de datos y una red de distribución de datos, en la que un centro de control se encargará de su monitorización y control.

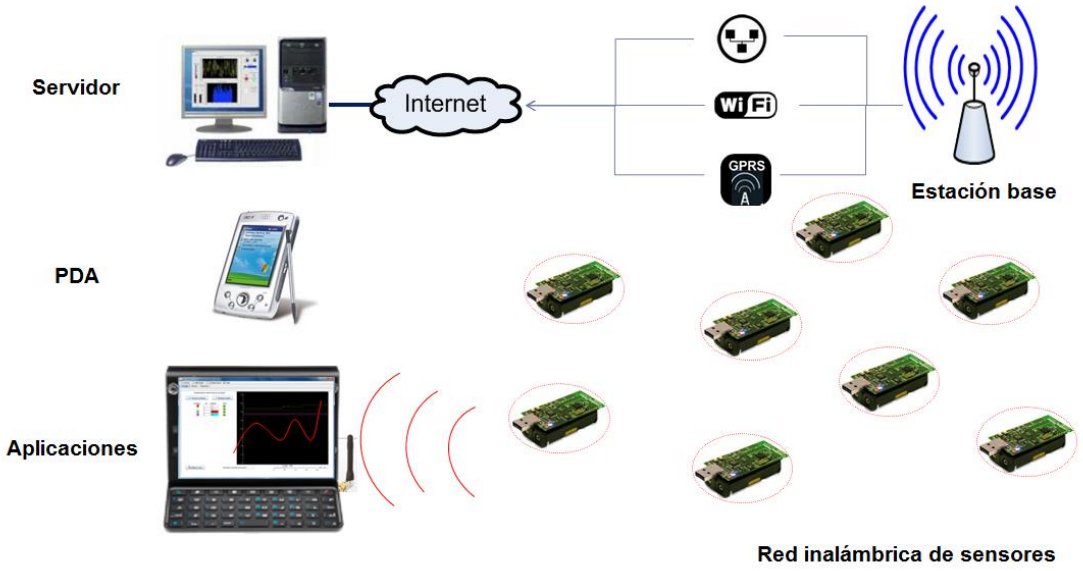

Fig. 2: Red de adquisición y distribución de datos.

Los elementos básicos en una red de sensores son los siguientes:

[1] motas: Elementos sensores autónomos
[2] *sink*: Sumidero

- **Nodo sensor**: Elemento autónomo capaz de obtener la información que se quiere medir (temperatura, humedad, CO_2, …) y transmitirla a un nodo central de coordinación. Consta de un microcontrolador, una fuente de alimentación (normalmente una batería), un transceptor (transmisor / receptor) RF y un sensor.

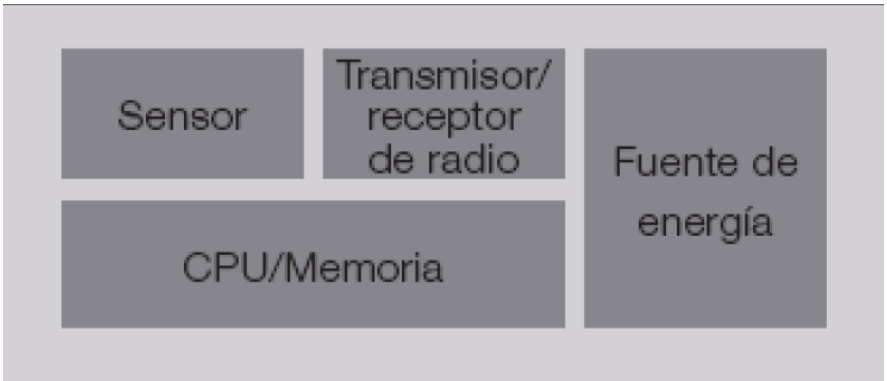

Fig. 3: Partes de un nodo sensor.

- **Router**: es el dispositivo que tiene la función de encaminar los paquetes por la red entre los nodos sensores y coordinador. Dispositivo que conecta los paquetes de red entre los nodos sensores y el nodo de coordinación.

- **Nodo coordinador**: Elemento que se encarga de constituir la red, gestionar las comunicaciones entre todos los elementos y de enviar la información obtenida de los nodos sensores al usuario.

Estos elementos se verán representados siguiendo un tipo determinado de topología, como se verá en el apartado 2.2.

2.1. Características de las WSN

Las redes de sensores tienen unas características propias, siendo algunas de ellas adaptaciones de las redes Ad-Hoc[3]:

- **Topología dinámica**: La facilidad para cambiar de forma en una red de sensores es un requisito importante, así como su capacidad para adaptarse a una topología en constante cambio y poder comunicar los nuevos datos adquiridos.

- **Fenómenos indeseados en los canales**: El canal radio es un canal muy variable que puede estar expuesto a frecuentes variaciones que perjudiquen la comunicación, como la atenuación o las interferencias que puede producir errores en los datos. Por esto, es importante realizar saltos en frecuencia para cambiar de un canal a otro de forma dinámica según se produzca algún efecto indeseado para preservar la fiabilidad de la red.

Es importante también que la red tenga tolerancia a errores producidos por la caída de un nodo, de forma que sea capaz de seguir funcionando a pesar de tener errores en su propio sistema.

- **Distribución del tráfico**: Hay que tener en cuenta el tipo de sensado al que se va a someter a la red. Si se tiene que realizar la sensorización de un parámetro ambiental, éste generará de forma periódica pequeños paquetes con datos que indicarán el estado del parámetro de estudio a una estación central de monitorización, lo que requiere un ancho de banda bajo, algo que se puede conseguir mediante la utilización de encabezados redundantes en los paquetes. Si se trata de detectar a un intruso en el campo de la seguridad, se generará un tráfico de detección en eventos con limitaciones en la transmisión a tiempo real.

- **Limitación en el hardware:** Para lograr un consumo ajustado hay que tener en cuenta el lugar de colocación de los nodos para el desarrollo de su soporte físico.

[3] Redes flexibles y sin infraestructura, en las que todos los nodos ofrecen servicios de encaminamiento para permitir la comunicación de nodos que no tienen conexión inalámbrica directa. Su principal característica es que todos sus dispositivos realizan también retransmisión de paquetes, algo normalmente asociado a routers, y que también funcionan como terminales finales.

Por ello, es fundamental que el hardware sea lo más sencillo posible, algo que limita la capacidad de procesado.

- **Consumo energético:** Es importante compaginar autonomía con capacidad de proceso, ya que actualmente las redes cuentan con una unidad de energía limitada, normalmente una batería por cada nodo sensor. Teniendo en cuenta que las redes en general suelen funcionar en ambientes agresivos en cuanto a condiciones ambientales se refiere y con muy poca supervisión humana (a veces nula), los nodos sensores tienen que estar provistos de un procesador y un transceptor con un consumo muy bajo, a lo que hay que añadir un software que también combine esta característica limitando aún más el consumo.

2.2. Topología de las WSN

La topología se basa en la configuración de los componentes hardware y en cómo los datos se transmiten a través de dicha configuración. Existen 3 tipos que pueden emplearse con tecnología Zigbee: estrella, malla o híbrida (estrella - malla).

Estrella: Es la topología más básica. Se trata de un sistema donde la información enviada sólo da un salto y donde todos los nodos sensores pueden ser muy simples, estando en comunicación directa con la puerta de enlace, normalmente una distancia de 30 a 100 metros.

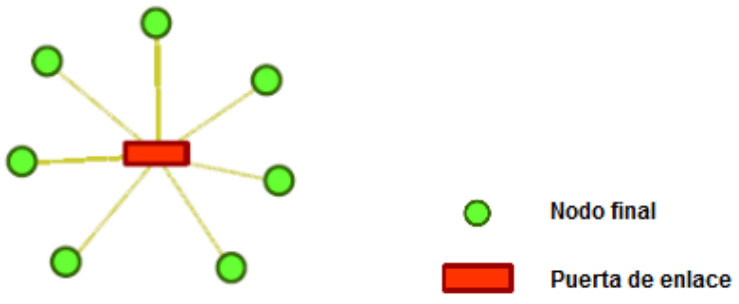

Fig. 4: Topología en estrella.

Malla: Es un sistema *multi - hop*[4] en el que todos los nodos son idénticos, además de routers. Cada nodo puede enviar y recibir información de otro nodo y de la puerta de enlace. Una diferencia con respecto a la topología en estrella es que en ésta los nodos pueden enviarse mensajes entre ellos, mientras que en estrella los nodos sólo pueden comunicarse con la puerta de enlace

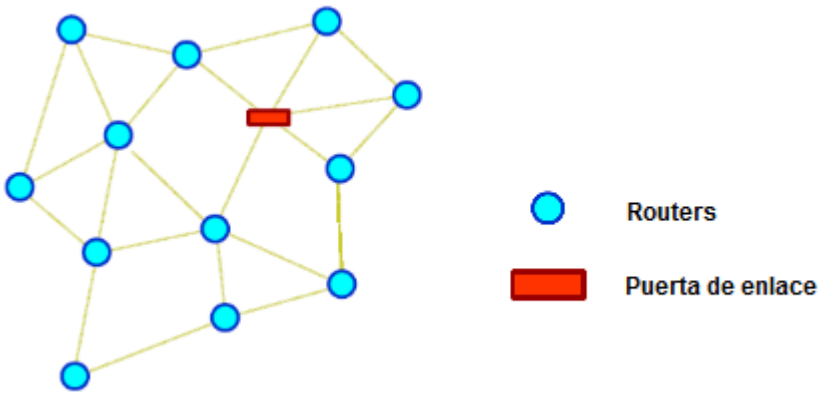

Fig. 5: Topología en malla.

Híbrida: Este sistema busca combinar las ventajas de los otros dos tipos: el bajo consumo y la simplicidad de la topología en estrella y la capacidad de cubrir una gran cantidad de terreno y de reorganizarse ante fallos de la topología en malla. La idea es crear una red en estrella alrededor de routers pertenecientes a una red en malla, lo que permite ampliar la red y corregir fallos en estos nodos. Los nodos finales se conectan con los routers cercanos logrando un ahorro de energía.

[4] multi –hop: También llamado multi – salto, este sistema hace que un nodo transmita a la estación base reenviando sus datos al nodo colindante más cercano, de forma que la información viaje salto a salto desde un nodo a otro desde la fuente al destino.

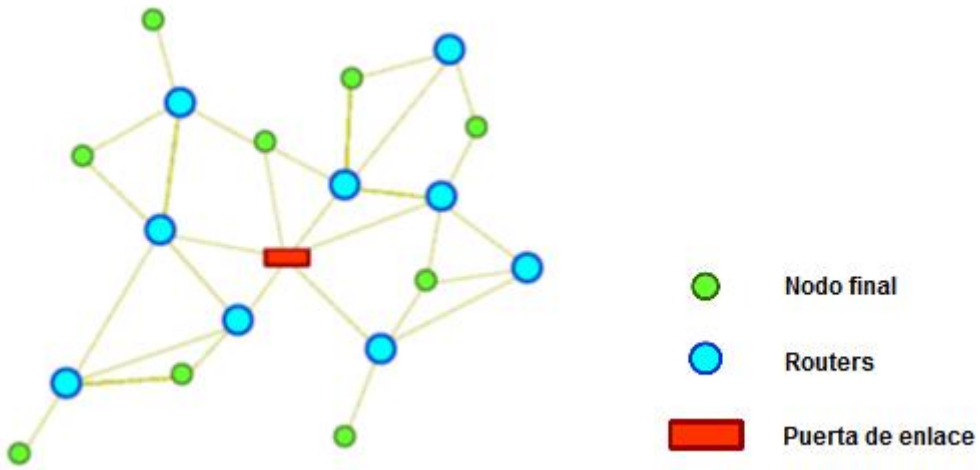

Fig. 6: Topología híbrida.

2.3. Aplicaciones

Dentro del campo de la monitorización de las condiciones ambientales del entorno, en la actualidad hay una gran cantidad de WSN implantadas en diferentes sectores, y la previsión en un futuro próximo es que su utilización crezca aún más. A continuación se detallan algunas de ellas:

Agricultura ecológica

La ecología investiga los procesos y patrones que interrelacionan a los seres vivos con su entorno, lo cual requiere de la observación durante días o meses de los cambios y evoluciones en dichos procesos. El poder registrar de modo simultáneo distintos parámetros en distintos lugares abre una puerta al uso de las WSN para realizar modelos y predicciones sobre medio ambiente y agricultura, gracias a su capacidad para recolectar gran cantidad de datos exactos dentro de una distribución espacio – temporal. Los sensores inalámbricos permiten disponer de cientos (incluso miles, dependiendo del área de medición) de dispositivos preparados para tomar datos de una manera no invasiva y a un bajo coste, garantizando la exactitud y veracidad de los datos suministrados de forma continua y a tiempo real.

Monitorización del entorno

Los nodos sensores pueden usarse para monitorizar datos como movimientos migratorios de animales y su comportamiento, o el estudio de las plantas en su hábitat natural con un impacto mínimo en la naturaleza. Los nodos también pueden monitorizar la calidad del aire, controlando e informando sobre indicios de contaminación medioambiental, incendios no controlados en bosques, u otros desastres naturales provocados (o no) por el hombre. La monitorización del entorno es una de las primeras aplicaciones de las WSN y, en este sentido, se deben tener en cuenta en estas aplicaciones el tiempo de vida de los sensores y la sincronización entre nodos sensores, para que dichos sensores duren un determinado período de tiempo y proporcionen los valores medidos precisando el instante de la toma de medida para realizar un historial de medidas significativo.

Existe un gran número de plataformas, como por ejemplo la llamada *Smart Citizen* [2] basada en Arduino, que permite conectar datos y cuyo objetivo es servir como nodo productivo para la generación de indicadores abiertos y herramientas distribuidas. Éste se basa en la geolocalización, en Internet y el hardware y software libres para la captura de datos y, en segundo plano, la producción de objetos. Permite conectar personas con su ciudad y entorno para crear relaciones más eficaces y optimizadas entre recursos, tecnología, comunidades, servicios y acontecimientos en el entorno urbano. El kit está equipado con sensores que miden la calidad del aire, temperatura, humedad, sonido y luz. La placa dispone de un cargador solar para conectar a paneles fotovoltaicos y una antena Wi-Fi para subir los datos recogidos por los sensores en tiempo real a plataformas online.

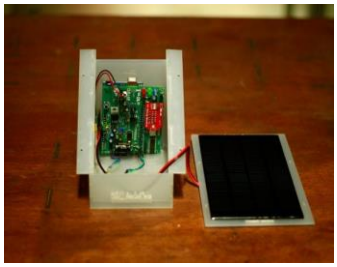

Fig. 7: Kit Smart Citizen

2.4. Fabricantes en el mercado

En cuanto a la fabricación de nodos sensores se puede encontrar un mercado muy diverso. Estos son algunos fabricantes:

- **CROSSBOW**: Empresa desarrolladora de plataformas hardware y software especializada en el mundo de los sensores, en especial en el de las redes inalámbricas. Entre sus productos se encuentran los nodos sensores MICA, que funcionan con sistema operativo TinyOS, MICA2 (868/916 MHz), MICAz (2.4 GHz) e IMOTE2, diseñado por Intel. En la figura 8 se pueden ver las partes de un sensor MICAz (a) y de la placa de sensores MTS 420 (b), enfocadas a la medición de parámetros ambientales:

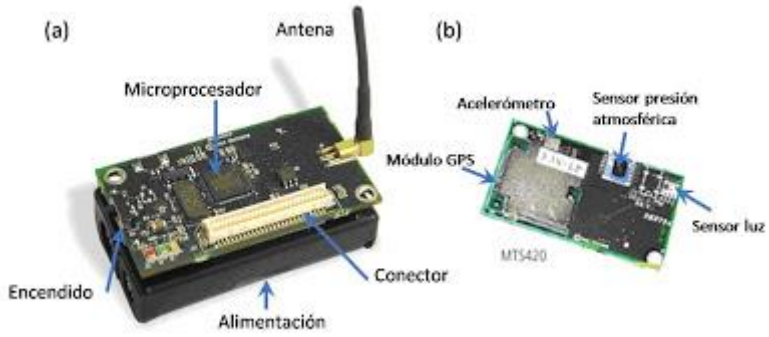

Fig. 8: Mota MICAz y placa MTS 420.

- **SENTILLA**: Adquirida por Ericsson, esta empresa (antes llamada MOTEIV) es la encargada de diseñar junto con la Universidad de Berkeley las plataformas Tmote Sky y Tmote Invent, preparadas para soportar TinyOS.

- **ARDUINO**: Plataforma de hardware libre para la creación de prototipos basada en una placa con un microcontrolador y un entorno de desarrollo flexible y fácil de usar, siendo por esto una de las más utilizadas por los usuarios. Si se combina con la tecnología Zigbee permite a la placa Arduino comunicarse de forma inalámbrica con otros nodos formando una red de sensores, proporcionando un sistema de red simple, fiable y flexible ya que otorga capacidad para ampliar la

red si así se desea. Una de las causas de su éxito comercial es su gran espectro de posibilidades de desarrollo, desde pequeños juguetes robóticos, pasando por implementaciones de monitorización ambiental, aplicaciones médicas, elementos musicales, etc.

- **RASPBERRY PI**: Plataforma similar a Arduino. A través de su GPIO se pueden conectar los mismos sensores y extras que con Arduino pero además se tiene la ventaja de que es un ordenador completo y barato. Es una buena elección para proyectos que requieran de una pantalla o interfaz gráfico, o que precisen de conexión a Internet. Posee capacidades acordes a la relación rendimiento / precio. Un ejemplo de ello es el proyecto AirPi, cuya función es la monitorización del clima y la calidad del aire.

Fig. 9: Placa Raspberry Pi.

- **BEAGLEBONE**: De mayor precio que RASPBERRY PI**,** está placa está diseñada para funcionar a un nivel mucho más alto y con mayor capacidad de proceso que Arduino gracias, por ejemplo, a sus convertidores analógicos / digitales con gran resolución útiles para aplicaciones de cierta exigencia. Puede ser ejecutada por Linux o Android.

- **LIBELIUM**: Empresa *startup* española de diseño y fabricación de hardware para la implementación de redes sensoriales inalámbricas, redes en malla y protocolos de comunicación para todo tipo de redes inalámbricas distribuidas. Algunos ejemplos de sus productos son:

  - Meshlium: Se trata de un router único que integra en un elemento las tecnologías Wifimesh (2.4GHz - 5GHz), ZigBee, GPRS, GPS y Bluetooth. Es un sistema capaz de detectar el dispositivo aunque este no esté conectado a la red inalámbrica. Su principal atractivo es poder conectar las redes de sensores a la nube.

  - **Waspmote**: Es un dispositivo de bajo consumo para diseñar redes de sensores inalámbricas cumpliendo con unos requerimientos bastantes específicos, y cuya función es ser desplegado en un escenario real, algo indispensable en un estudio de las condiciones ambientales.

La plataforma elegida para el desarrollo de nuestro proyecto es Waspmote, fundamentalmente por la versatilidad de componentes de que dispone (placas, sensores, etc.) y la buena relación calidad / precio. Un punto interesante es el entorno de programación de su compilador, ya que es muy similar al de Arduino, por lo que facilitará la labor a aquellos usuarios que nunca hayan utilizado el Waspmote y hayan trabajado en Arduino.

3. Estudio de las posibilidades

3.1. Elección del método

3.1.1. Comunicación inalámbrica. Estándares de comunicación

Las transmisiones inalámbricas son una herramienta eficaz que permite la transferencia de voz, vídeo y datos sin necesidad de cableado. Esta transferencia de información se logra a través de la emisión de ondas de radio obteniendo dos ventajas fundamentales: movilidad y flexibilidad del sistema en general. Desde hace unos años, las redes de sensores han sufrido un importante desarrollo. Su principal activo es su bajo coste y la capacidad para ser distribuidas en grandes cantidades. Debido entre otras cosas a su gran espectro de actuación, es interesante estandarizar algunos de los elementos de las redes de sensores para facilitar la operación entre productos de distintos fabricantes. A día de hoy, realmente no existe un estándar de facto que las aúne ya que es difícil englobarlas todas, en primer lugar porque las tecnologías siempre están en constante cambio y desarrollo, y también por la enorme amplitud de las aplicaciones de las redes de sensores.

3.1.1.1. Infrarrojos

Los infrarrojos son ondas electromagnéticas que se propagan en línea recta que pueden ser interrumpidas por cuerpos opacos. Su utilización no precisa de licencias administrativas y no se ve afectado por interferencias radioeléctricas externas, pudiendo alcanzar distancias de hasta 200 metros entre emisor y receptor.

El principio de funcionamiento se basa en un transmisor que envía un haz de luz infrarroja hacia receptor. La transmisión de luz se codifica y decodifica en el envío y recepción en un protocolo de red existente. Este tipo de comunicación es útil, por ejemplo, para enviar datos a un robot desde un sensor, establecer y detectar balizas en un entorno, o para que una persona cambie de canal utilizando un aparato convencional de control remoto, como el de la TV.

3.1.1.2. GSM (Groupe Special Mobile)

Este es el estándar conocido como “2G”. En Europa, el estándar GSM usa las bandas de frecuencia de 900MHz y 1800 MHz. Permite un rendimiento máximo de 9,6 kbps, que proporciona transmisiones de voz y de datos digitales de volumen bajo, como mensajes de texto (**SMS**, *Servicio de mensajes cortos*) o mensajes multimedia (**MMS**, *Servicio de mensajes multimedia*). La estructura de la red se puede ver en la figura 10:

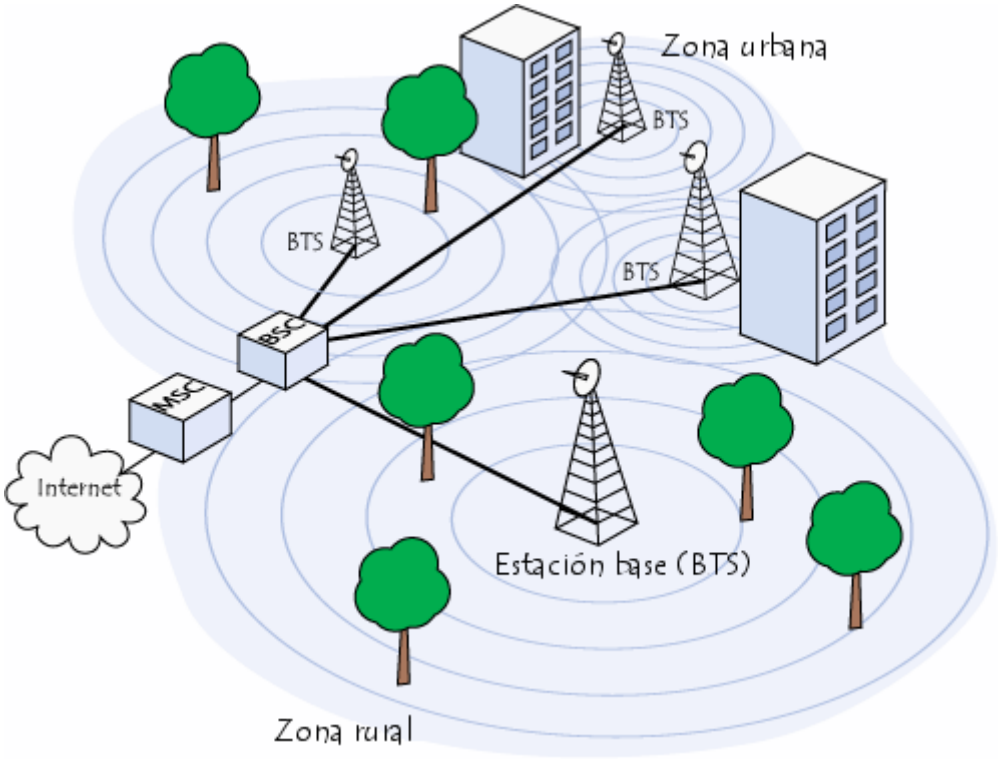

Fig. 10: Sistema GSM

En la que podemos destacar:

- **BTS (Estación Base Transceptora):** Se encuentran en en centro de cada celda de la red celular.

- **MSC (Mobile Switching Center ó Centro de servicio de conmutación móvil):** Realiza las funciones de telefonía de conmutación del sistema, controla las llamadas desde y hacia otro teléfono y maneja los sistemas de datos. También realiza funciones como la interconexión de red, la señalización por canal común, etc.

- **BSC** (**Base Station Controller ó Controlador de estación base**): Otorga todas las funciones de control y enlaces físicos entre el MSC y el BTS. Se trata de una conmutación de alta capacidad que ofrece funciones como los datos de configuración de la celda y el control de la frecuencia de radio (RF) de los niveles de potencia en las estaciones de transceptor de base. Un número determinado de BSCs son atendidos por un MSC.

3.1.1.3. GPRS (General Packet Radio Service)

Tecnología que comparte el rango de frecuencias de la red GSM utilizando una transmisión de datos por medio de "paquetes". La conmutación de dichos paquetes es un procedimiento más adecuado para transmitir datos que en GSM, donde los datos se habían transmitido mediante conmutación de circuitos, procedimiento más adecuado para la transmisión de voz. Las principales ventajas con respecto al estándar GSM son que los canales se comparten entre los diferentes usuarios y que se obtiene mayor velocidad y mejor eficiencia de la red. Por todo esto se dice que el estándar GPRS es de 2.5 G.

3.1.1.4. Bluetooth (IEEE 802.15.1)

Especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que permite la transmisión de datos y voz entre dos dispositivos por medio de una frecuencia (2.4 GHz), prescindiendo así de cables para su interconexión. Su uso resulta interesante cuando se tienen dos o más dispositivos en un área reducida sin grandes necesidades de ancho de banda. Existen 3 clases de Bluetooth en función de su alcance:

- Clase 1: 100 m aproximadamente.
- Clase 2: 10 m.
- Clase 3: 1 m.

3.1.1.5. Wi-Fi (IEEE 802.11b)

Tiene una velocidad máxima de transmisión de 11 Mbps y emplea el mismo método de acceso definido en el estándar original CSMA/CA. Cabe destacar que a nivel práctico la velocidad máxima de transmisión con este estándar es de aproximadamente 5,9 Mbits sobre TCP y 7,1 Mbit/s sobre UDP, debido al espacio ocupado por la codificación de dicho protocolo. Este estándar funciona en la banda de 2,4 GHz y se deben conectar preferentemente utilizando el estándar 802.11g, dando lugar a Wi-Fi (802.11b/g). Esto sirve para evitar la degradación de las celdas al utilizar los mecanismos de seguridad en toda la red inalámbrica.

3.1.1.6. ZigBee (IEEE 802.15.4)

En este estándar se definen los niveles de red básicos para dar servicio a un tipo específico de red WPAN centrada en la habilitación de comunicación entre dispositivos ubicuos con baja complejidad, pequeño consumo de energía, conectividad inalámbrica de baja velocidad de datos entre dispositivos (a diferencia de estándares más orientados directamente a los usuarios medios, como Wi-Fi) y de bajo coste. La tasa de datos bruta debe ser lo suficientemente alta (200 kbps como máximo) para satisfacer un conjunto de necesidades simples, siendo adaptable a las necesidades de sensado y automatización para comunicaciones inalámbricas (10 kbps o menos). Se enfatiza el bajo coste de comunicación entre nodos cercanos (con o sin infraestructura) para favorecer aún más el bajo consumo. En la tabla 1 se muestra una comparativa entre varias tecnologías inalámbricas:

Prestaciones	Wi-Fi *802.11*	Bluetooth 802.15.1	ZigBee 802.15.4
Frecuencia de radio (GHz)	2.4	2.4	0.868; 0.915; 2.4
Consumo de corriente (mA)	30	65-170	30
Nº de nodos por master	7	32	65000
Velocidad de transmisión RF (Kbps)	54000	1000-3000	250
Potencia de transmisión (mW)	40-200	1-100	1-2
Rango de trabajo (m)	30-100	1-100	1-100
Duración de la batería	100 – 1000 días	1 semana	12 y 48 horas
Tipo de datos	Pequeños paquetes de datos	Audio, gráficos, películas, ficheros	Vídeo, audio, gráficos, películas, ficheros
Características más importantes	Velocidad y flexibilidad	Costes y perfiles de aplicación	Fiabilidad, bajo consumo y bajo coste
Complejidad	Complejo	Muy complejo	Sencillo

Tabla 1: Comparativa de tecnologías de comunicación inalámbrica

Tecnología escogida en el proyecto

Entre todas las posibilidades a nuestro alcance, se ha elegido una red ZigBee frente a otras tecnologías inalámbricas gracias a su capacidad para permitir una jerarquía de asociaciones entre los distintos elementos de la red en lugar de simples asociaciones maestro – esclavo, lo que posibilita extender la red en términos de superficie, empleando distintos tipos de nodos en cada punto. De este modo, un elemento conectado al nodo coordinador de la red puede actuar a su vez como enrutador, y permitir que otros dispositivos se conecten a él, mejorando los niveles de asociación.

3.2. Plataforma de desarrollo

La placa Waspmote Pro 1.2 es una plataforma de código abierto que se encarga de recoger la información ambiental y procesarla. Su principal cualidad es proporcionar un consumo mínimo con el máximo rendimiento y capacidades. Waspmote (el nombre comercial de la placa) cuenta con un consumo de 0.7μA en el modo de hibernación y siete modelos diferentes de radios de comunicación que pueden ser elegidos en función de:

- **Frecuencia**: 2.4GHz, 900MHz, 868MHz
- **Protocolo**: 802.15.4, ZigBee
- **Potencia**: 1mW, 100mW

Fig. 11: Waspmote PRO 1.2

Su alta sensibilidad en recepción (RX) y su potencia en transmisión (TX) permiten los siguientes alcances:

- 7km – 2,4 GHz
- 24Km a 900 MHz
- 40 Km a 868 MHz

Esto nos va a permitir monitorizar casi cualquier lugar. La plataforma está basada en una arquitectura modular, esto significa que podemos implementar módulos adicionales en función de nuestras necesidades, como módulos GPS, GPRS o tarjetas SD. Esta filosofía modular permite la conexión al Waspmote de diversas placas de sensores de gases (CO, CO_2, SH_2, NH_3…), peso, luminosidad, etapas de amplificación, agricultura y por supuesto Smart Cities, una de las razones por la que se ha elegido dicha placa. Lleva incorporado también un acelerómetro de 3 ejes útil para para obtener la máxima precisión y estabilidad en ambos rangos (+-2g, +-6 g) y controlar en tiempo real cualquier tipo de movimiento o vibración.

Se alimenta con una batería de litio que puede recargarse a través de un conector preparado para paneles solares. Esta opción es especialmente interesante para los despliegues en entornos naturales como bosques e incluso ciudades.

Para trabajar con la plataforma, se dispone de un entorno de programación, el API de Waspmote y el compilador, ambos de código abierto.

Microcontrolador	ATmega1281
Frecuencia	14.7456 MHz
SRAM	8KB
EEPROM	4KB
FLASH	128KB
SD Card	2GB
Peso	20 gramos
Dimensiones	73.5 x 51 x 13 mm
Rango de temperatura	[-10ºC, +65ºC]

Tabla 2: Especificaciones del Waspmote Pro 1.2

3.2.1. Características generales

A continuación se pueden ver las principales características de la placa, así como los diferentes puertos y tomas que contiene.

3.2.1.1. Diagrama de bloques del Waspmote

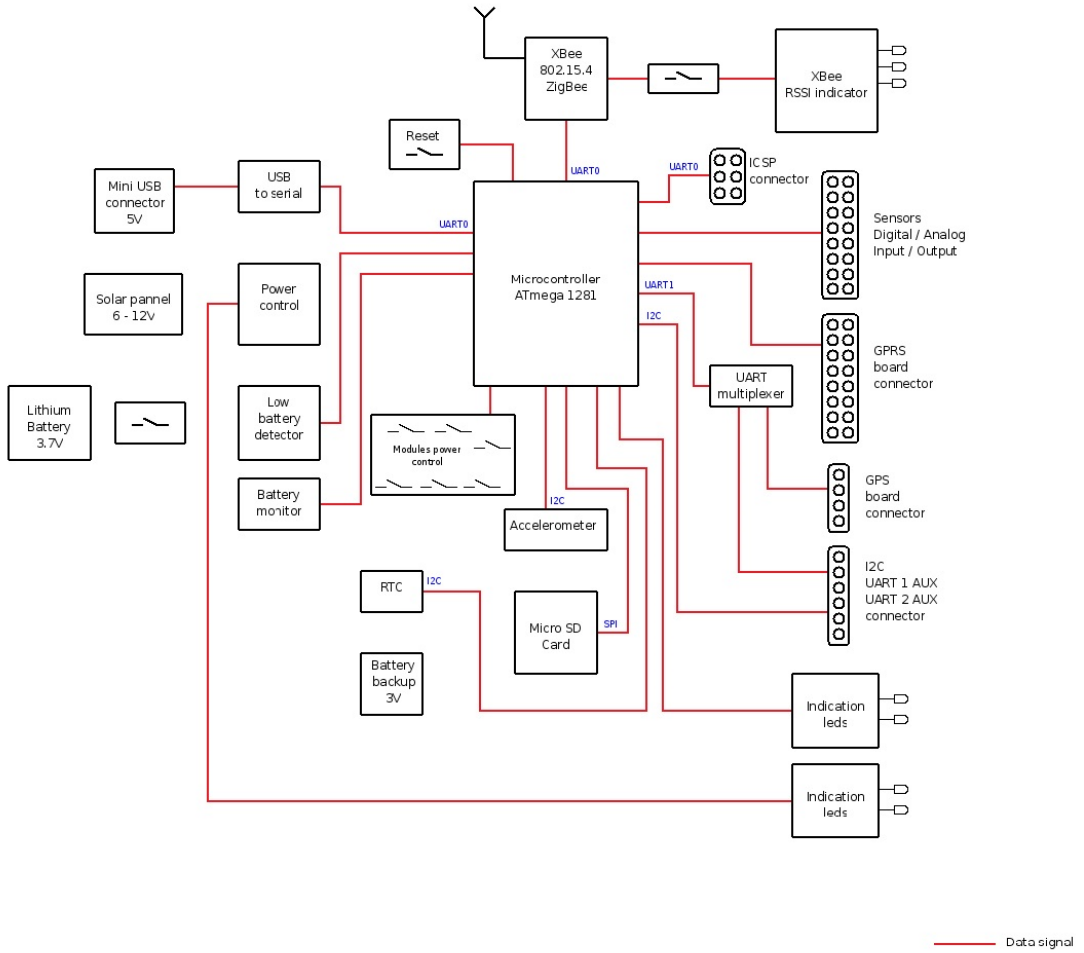

Fig. 12: Diagrama de bloques y señales del Waspmote

3.2.1.2. E/S

Waspmote puede comunicarse con otros dispositivos externos a través de los diferentes puertos de entrada / salida:

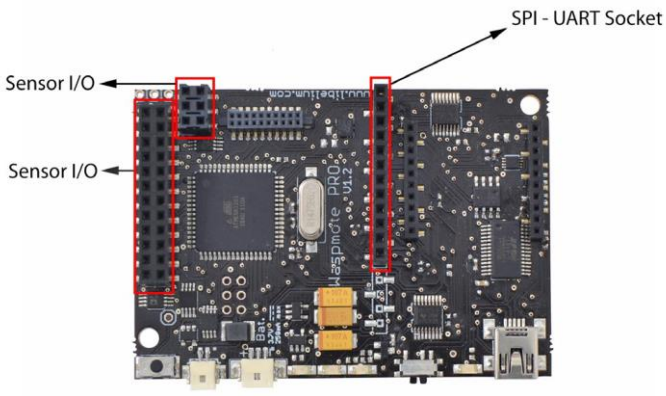

Fig. 13: Conectores de E/S

En la figura 14 se pueden ver los 3 puertos principales con los identificadores correspondientes:

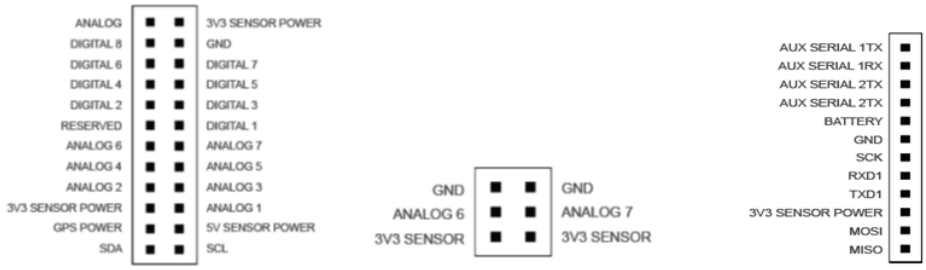

Fig. 14: Descripción de los pines de conexión de E/S

3.2.1.3. Entradas analógicas

Waspmote tiene 7 entradas analógicas accesibles para la conexión de sensores. Cada entrada se encuentra directamente conectada al microcontrolador, que utiliza un convertidor analógico / digital (ADC) de aproximación sucesiva de 10 bits. La tensión de referencia en las entradas es de 0V (GND), y el valor máximo de tensión de entrada es 3.3V, que se corresponde con la tensión general de alimentación del microcontrolador.

Para la obtención de valores de entrada, la función `analogRead(entrada analógica)`, el parámetro de entrada de la función será el nombre de la entrada que será

leído (“ANALOG1, ANALOG2…”).El valor obtenido de esta función será un número entero entre 0 y 1.023, donde 0 corresponde a 0 V y 1023 a 3,3 V.

Los pines de entrada analógicas también se pueden utilizar como pines de entrada / salida digitales. En este caso, se debe tener en cuenta la siguiente lista de correspondencia para los nombres de los pines:

- ANALOG1 => 14
- ANALOG2 => 15
- ANALOG3 => 16
- ANALOG4 => 17
- ANALOG5 => 18
- ANALOG6 => 19
- ANALOG7 => 20

Y con la instrucción se accede al valor:

```
{
      valor = analogRead(ANALOG1);
}
```

3.2.1.4. Entradas digitales

Waspmote tiene pines digitales que se pueden configurar como entrada o salida en función de las necesidades de la aplicación. Los valores de tensión correspondientes a los diferentes valores digitales serían:

- 0V for logic 0
- 3.3V for logic 1

3.2.1.5. PWM

El pin DIGITAL1 también se puede utilizar como salida PWM (modulación por ancho de pulso) con la que una señal analógica puede ser "simulada". En realidad, se trata de una onda cuadrada entre 0 V y 3,3 V para la que la proporción de tiempo cuando la señal es alta puede cambiar (es decir, su ciclo de trabajo) de 0% a 100%, simulando una tensión de 0 V (0%) a 3,3 V (100%). Posee una resolución de 8 bits, por lo que se pueden configurar hasta 255 valores entre 0-100%.

3.2.1.6. UART

Hay dos UARTs: UART0 y UART1. Además, hay varios puertos que pueden ser conectados a estas UARTs a través de dos multiplexores diferentes, uno para cada una.

La UART0 está compartida por el puerto USB y el *Socket0*. Esta toma se utiliza para los módulos XBee, módulos RFID, módulos Bluetooth, módulos Wi-fi, etc. El multiplexor en este UART controla la señal de datos que por defecto siempre se cambia a *Socket0*. Cuando el USB necesita enviar información a través de la UART0, el multiplexor se conecta momentáneamente al puerto USB y retrocede de nuevo a *Socket0* después de la impresión.

La UART1 está compartida por cuatro puertos: *Socket1*, *Socket* GPS, y Auxiliar1 y Auxiliar2. Es posible seleccionar en el mismo programa cual de los cuatro puertos se conecta a UART1 en el microcontrolador. La configuración del multiplexor UART1 se lleva a cabo mediante las siguientes instrucciones:

```
{
      Utils.setMuxAux1(); // Establece la toma Auxiliar1
      Utils.setMuxAux2(); // Establece la toma Auxiliar2
      Utils.setMuxGPS(); // Establece la toma GPS
      Utils.setMuxSocket1(); // Establece la Socket1
}
```

3.2.1.7. I2C

El bus de comunicación I2C también se utiliza en Waspmote donde tres dispositivos estén conectados en paralelo: el acelerómetro, el RTC y el potenciómetro digital (digipot) que configura el nivel de umbral de alarma de batería baja. En todos los casos, el microcontrolador actúa como master (maestro), mientras que los otros dispositivos conectados al bus son slaves (esclavos).

3.2.1.8. SPI

El puerto SPI se usa en el microcontrolador para la comunicación con la tarjeta micro-SD. Todas las operaciones que utilizan el bus son realizadas por la biblioteca específica. El puerto SPI está disponible también en el conector SPI/UART.

3.2.1.9. USB

Este puerto se utiliza para la comunicación con un ordenador o dispositivos USB compatibles. Esta comunicación permite la carga del programa del microcontrolador en otros dispositivos. Para la comunicación USB, se utiliza la toma UART0. El chip FT232RL lleva a cabo la conversión al estándar USB.

3.2.1.10. LEDs

Hay 4 indicadores LED en la placa:

- **Indicador LED de carga de la batería**: Un LED rojo indica que hay una batería conectada en Waspmote que está siendo cargada. Una vez que la batería está completamente cargada, el LED se apagará automáticamente.
- **Indicador LED USB**: Cuando el LED está encendido indica que el cable USB está conectado correctamente al Waspmote. Al retirar el cable USB el LED se apagará automáticamente.
- **LED 0**: Se identifica mediante un LED verde. Es totalmente programable por el usuario mediante código de programación. Además, indica el reseteo del dispositivo mediante un parpadeo.
- **LED 1**: Se identifica mediante un LED rojo. De forma análoga al anterior es totalmente programable por el usuario desde el código de programa.

En la figura 15 se pueden ver principales componentes de la placa con sus respectivos identificadores:

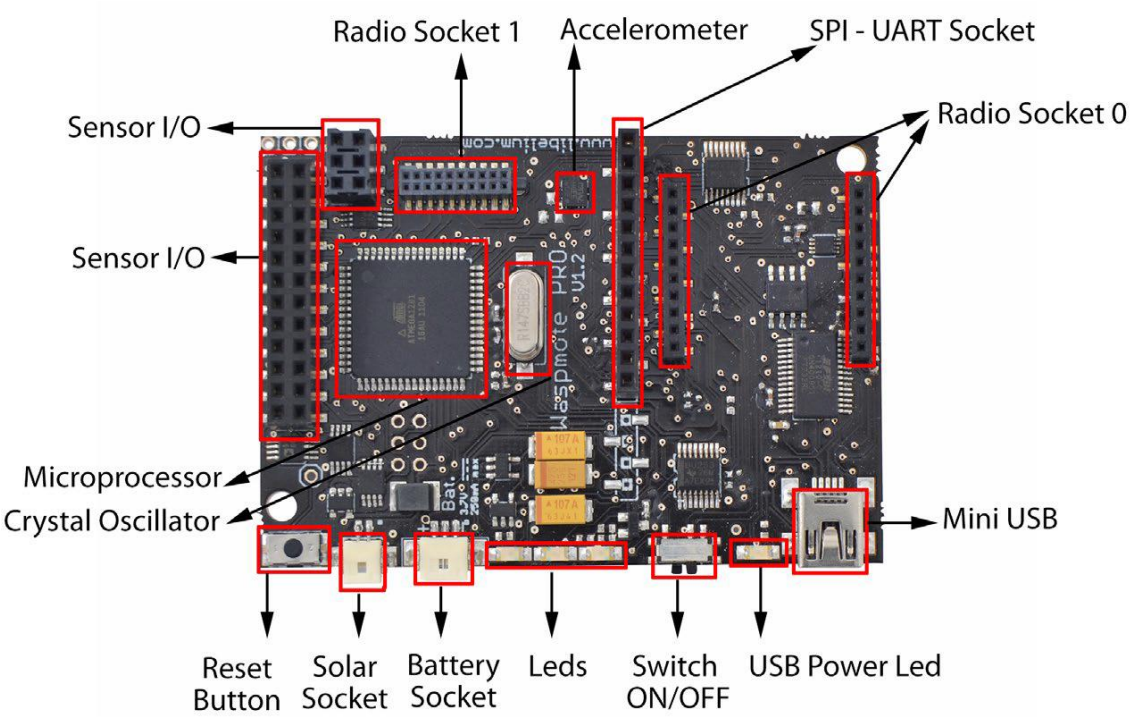

Fig. 15: Parte superior del Waspmote PRO 1.2

3.2.2. Microcontrolador ATmega1281

3.2.2.1. Información general

El ATmega1281 es un microcontrolador CMOS de baja potencia de 8 bits basado en la arquitectura RISC mejorada de AVR. Mediante la ejecución de potentes instrucciones en un solo ciclo de reloj, el ATmega1281 logra rendimientos que se acercan a 1 MIPS[5] a 1MHz por MHz, lo que permite optimizar el consumo de energía en comparación con la velocidad de procesamiento. A continuación se pueden ver las características genéricas del ATmega1281:

- Dispositivo: ATmega1281
- FLASH: 128KB
- EEPROM: 4KB
- RAM: 8KB
- Pines E/S de propósito general: 54
- Canales PWM de resolución 16 bits: 6
- Puertos USARTs serie: 2
- Canales ADC: 8

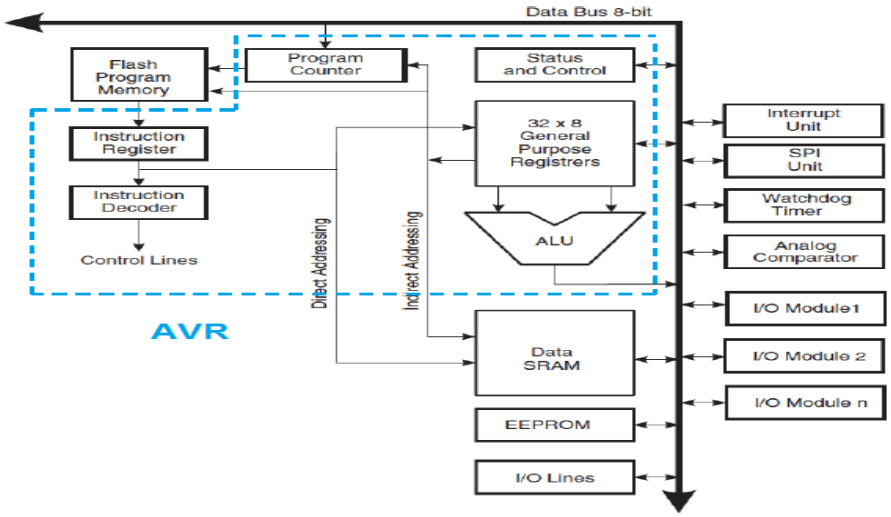

Fig. 16: Diagrama de bloques del microprocesador AVR

[5] MIPS: Millones de Instrucciones Por Segundo.

Para maximizar las prestaciones, el AVR está dotado de arquitectura Harvard (con buses y memorias separadas para programas y datos). Mientras se está ejecutando una instrucción, se realiza la fase de búsqueda en memoria de la siguiente. Este concepto permite que se ejecute una instrucción en cada ciclo de reloj.

3.2.2.2. Interrupciones

Las interrupciones son señales recibidas por el microcontrolador que indican que debe detener la tarea que está haciendo en ese instante para asistir a un evento que acaba de ocurrir. El control de las interrupciones libera al microcontrolador de tener que ocuparse del sensado todo el tiempo. También hace que los sensores adviertan al Waspmote cuando se alcanza un determinado umbral. Las interrupciones pueden ser de dos tipos: síncronas o asíncronas.

Interrupciones síncronas: Están programadas por los timers, y permiten programar cuando se desean que se puedan activar.

Interrupciones asíncronas: Éstas no están programadas por lo que no se sabe cuándo van a ser activadas.

El diagrama de funcionamiento con sus diversas interrupciones puede verse en la figura 17:

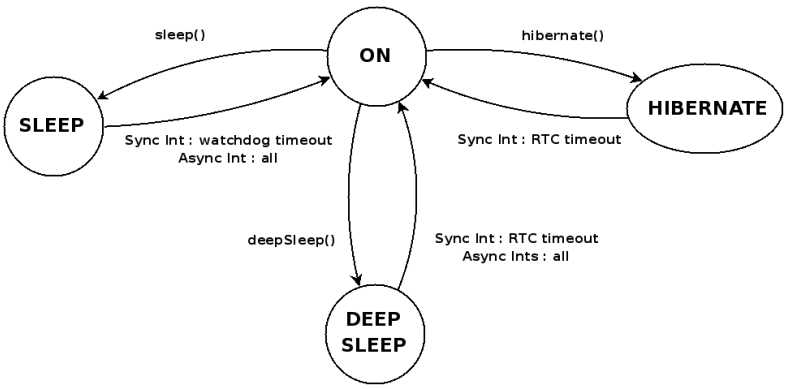

Fig. 17: Diagrama de modos de funcionamiento

En la tabla 3 se puede ver de forma sintetizada las características de los modos:

	Consumo	µC	Ciclo	Interrupciones admitidas
ON	15mA	ON	-	Síncronas y asíncronas
SLEEP	55µA	ON	32ms – 8s	Síncronas (Watchdog) y asíncronas
DEEP SLEEP	55µA	ON	1s - min/horas/días	Síncronas (RTC) y asíncronas
HIBERNATE	0.06µA	OFF	1s - min/horas/días	Síncronas (RTC)

Tabla 3: Modos de operación del Waspmote

3.2.2.3. ADC - Convertidor Analógico / Digital

Es un convertidor AD de 10 bits de aproximación sucesiva. Está conectado a un multiplexor analógico de 8 canales que permite 16 entradas de tensión de terminación única construidas a partir de los pines del puerto F y K. Las entradas de un solo extremo de tensión se refieren a 0 V (GND).

Descripción de los registros

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Fig. 18: Detalle de los registros ADCH y ADCL (ADLAR = 0)

Cuando se realiza una conversión AD, el resultado se encuentra en estos dos registros. Si se usan canales diferenciales, el resultado se presenta en forma de complemento a dos.

Al leer el registro ADCL, el registro de datos ADC no se actualiza hasta que se lea el ADCH. Por tanto, si el resultado se deja ajustado y no se requiere mayor precisión de 8

bits (7 bits + bit de signo para los canales de entrada diferencial), es suficiente para leer ADCH. De lo contrario, ADCL debe leerse primero, y luego el ADCH.

El bit ADLAR en ADMUX, y los bits MUXn en ADMUX afectan a la forma de leer el resultado de los registros. Si ADLAR = 1, el resultado se ajusta izquierda y si ADLAR = 0 (por defecto), el resultado se ajusta a la derecha. No se activará en este caso.

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 19: Detalle del registro ADCSRA

Bit 4 – ADIF: ADC Interrupt Flag: Este bit se ajusta cuando una conversión ADC se completa y los registros de datos se actualizan. ADIF se borra por hardware al ejecutar el vector de manejo de interrupciones correspondiente. De forma alternativa, ADIF se borra escribiendo un uno lógico en el flag. Hay que tener en cuenta que si se hace una lectura / escritura / modificación en ADCSRA, se puede desactivar una interrupción pendiente. Por tanto, para realizar la conversión debe estar a nivel bajo.

Resultado de la conversión ADC (ADC9:0:)

Después de terminar la conversión (ADIF a nivel alto), el resultado de la conversión se puede encontrar en los registros de resultados ADCL y ADCH. Para la conversión unilateral (sin el uso de canales diferenciales), el resultado es:

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}} \quad (1)$$

donde V_{IN} es la tensión en el pin de entrada seleccionado y V_{REF} la tensión de referencia. 0x000 representa la masa analógica, y 0x3FF representa la tensión de referencia menos un LSB. Si se utilizan canales diferenciales, el resultado es:

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot 512}{V_{REF}} \quad (2)$$

donde V_{POS} es la tensión en el pin de entrada positivo, V_{NEG} la tensión en el pin de entrada negativo, y V_{REF} la tensión de referencia. El resultado se presenta en forma de complemento a dos, a partir de 0x200 (-512d) hasta 0x1FF (+ 511D).

Para más información acerca del ATmega1281 puede consultarse el *datasheet* [3].

3.2.3. Módulo *XBee* 802.15.4 PRO

Este módulo cumple con el estándar IEEE 802.15.4, que define el nivel físico y el nivel de enlace (capa MAC). Además, añade ciertas funcionalidades [4] a los aportados por la norma, tales como:

- Detección de nodos: Cierta información ha se añade a las cabeceras de los paquetes para que puedan encontrar otros nodos en la misma red. Permite enviar un mensaje de descubrimiento de un nodo, de modo que el resto de nodos de la red responden indicando sus datos (identificador de nodo, @MAC, @16 bits, RSSI).
- Detección de paquetes duplicados: Esta funcionalidad no está establecida en la norma y se añade por los módulos XBee.

El cifrado se proporciona a través del algoritmo AES 128b, específicamente a través del tipo AES-CTR, En este caso, el campo *Frame Counter* tiene un identificador único y cifra toda la información contenida en el campo *Payload*, que es el lugar dentro del marco 802.15.4 donde los datos a se almacenan.

La forma en que se han desarrollado las bibliotecas para la programación del módulo de activación de cifrado es tan simple como ejecutar la función de inicialización y darle una clave para utilizar en el proceso de cifrado.

Fig. 20: XBee 802.15.4 PRO

Características del XBee 802.15.4 PRO

- Frecuencia: 2,40 - 2,48GHz
- Potencia de transmisión: 63.1mW
- Sensibilidad: -100dBm
- Nº de canales: 13
- Distancia: 7000m

La frecuencia utilizada es la banda libre de 2,4 GHz, utilizando 16 canales con un ancho de banda de 5 MHz por canal, de los cuales el XBee PRO soporta del canal 12 (0x0C) al canal 23 (0x17), como se muestra en la figura 21:

Fig. 21: Canales de frecuencias en la banda de 2,4 GHz

3.2.4. Tarjeta de memoria SD

Es una tarjeta micro-SD (*Secure Digital*) que se utiliza específicamente para reducir espacio en la placa al mínimo. Waspmote utiliza el sistema de archivos FAT16 y puede soportar tarjetas de hasta 2 GB. Se puede acceder a dicha información en varios sistemas operativos (Linux, Windows o Mac OS). Al utilizar la biblioteca SD del Waspmote se ignoran automáticamente aquellos bloques que sean defectuosos. Sin embargo, cuando se utiliza un estándar *OTA*[6] como ZigBee, esos bloques de la SD no pueden ser evitados, por lo que la ejecución podría bloquearse. Para evitar esto emplearemos la tarjeta SD oficial distribuida por Libelium:

[6] OTA: Over-the-air programming

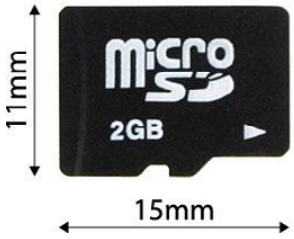

Fig. 22: Tarjeta Micro-SD

Para comunicarse con el módulo SD se usa el bus SPI, que incluye líneas para el reloj, los datos de entrada y salida de datos, y un pin de selección. La tarjeta SD se alimenta a través de un pin digital del microcontrolador, y por tanto no es necesario el uso de un interruptor para cortar la alimentación, poniendo un pin a valor bajo es suficiente para establecer el consumo de la SD a 0μA.

Una aproximación de la capacidad de información que se puede almacenar en una tarjeta de 2GB, se puede comprobar dividiendo su tamaño (2GB) por la media de lo que un *frame* de sensado en Waspmote ocupa normalmente (aprox. 100 Bytes), lo cual da 20 millones de medidas, siendo el límite 256 archivos por directorio y hasta 256 subdirectorios en cada directorio. No hay límite en el número de niveles anidados.

En este proyecto se podría programar un código para guardar los datos de forma continuada y, cuando el usuario lo precise, pedirlos por medio del interfaz de Matlab. Otra opción sería la de sacar la SD del Waspmote e introducirla en el ordenador para extraer los datos almacenados directamente. Como se puede ver, hay varias alternativas pero al final se ha optado por la opción de guardar los datos y mandarlos periódicamente sin necesidad de que el usuario lo pida, debido a la limitación de comunicación entre la placa y el módulo USB, que es en una única dirección.

3.3. Elección de los parámetros

Hay una gran cantidad de parámetros que son susceptibles de ser analizados en el entorno, como por ejemplo la medición de los gases que se encuentran en la atmósfera. Para comenzar a realizar el estudio se ha empleado un sensor de temperatura:

3.3.1. Sensor de temperatura (MCP9700A)

El MCP9700A de Waspmote es un sensor analógico que recibe un valor de temperatura y lo convierte en una tensión analógica proporcional, donde la equivalencia para 0° C es de 500mV, y el rango de medida a la salida sería de - 40º C a 125º C (equivalencia en tensión: 100 mV a 1.75 V)

Rango de medida	[-40ºC ,+125ºC]
Tensión de salida a 0ºC	500mV
Sensibilidad	10mV/ºC
Precisión	±2ºC (entre 0ºC ~ +70ºC), ±4ºC (entre -40 ~ +125ºC)
Tensión de alimentación	2.3 ~ 5.5V
Tiempo de respuesta	1.65 segundos (Respuesta del 63% desde 30 hasta + 125 ° C).
Consumo típico	6µA
Consumo máximo	12µA

Tabla 4: Especificaciones del MCP9700A

Fig. 23: Imagen del sensor de temperatura MCP9700A

En la figura 24 puede verse la conexión de los pines empleada sabiendo que:

- Pin 1: V_{cc}
- Pin 2: V_{out}
- Pin 3: GND

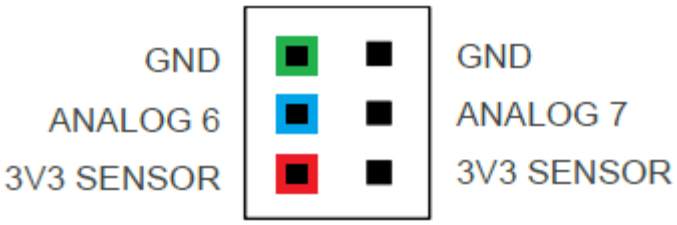

Fig. 24: Conexionado de los puertos para el MCP9700A.

3.3.2. Sensor de luz (LDR)

Un sensor LDR (*Light Dependent Resistor*) es una resistencia que varía su valor en función de la luz que recibe. Por esto se puede deducir que cuanta más luz reciba, menor será su resistencia:

Fig. 25: Imagen del sensor LDR

Este sensor necesita una modificación antes de conectarlo al Waspmote, en este caso un divisor de tensión. La conexión realizada es la siguiente:

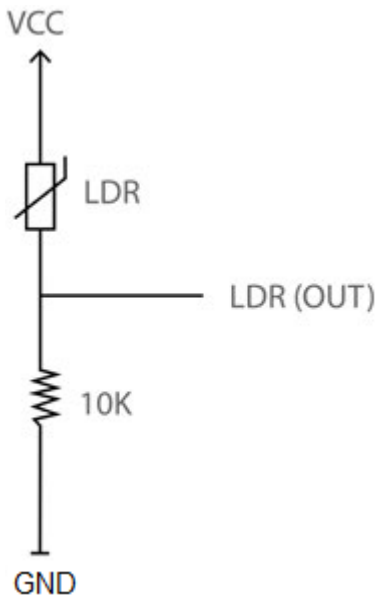

Fig. 26: Divisor de tensión para el LDR

4. Herramientas de desarrollo

En la actualidad el estudio de las redes de sensores inalámbricas se encuentra en continuo desarrollo y, en muchas ocasiones, es necesario antes de su implementación utilizar algún simulador que permita evaluar los resultados obtenidos por la red, así como poder realizar cambios en los parámetros de los sensores para estudiar los resultados y medir también el rendimiento y el consumo de potencia antes de probarla en la plataforma real. El compilador empleado en el proyecto ha sido el IDE propio de Waspmote [5].

4.1. IDE para Waspmote

Es un entorno de desarrollo ideado para trabajar en Waspmote prácticamente idéntico al compilador de Arduino, ya que tiene el mismo estilo y funcionamiento, además de incluir todas las librerías del API necesarias para la compilación de los programas.
En la figura 27 se pueden ver las partes de un programa tipo:

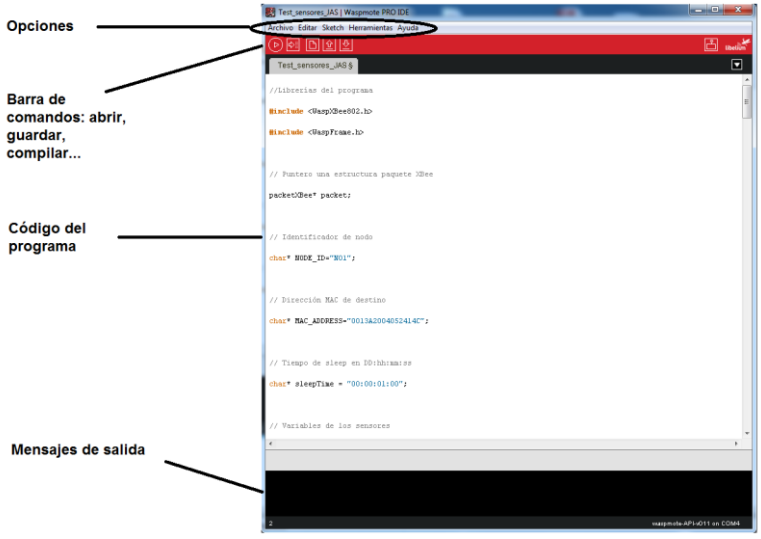

Fig. 27: Partes del IDE de Waspmote

Opciones: Menú en el que se pueden configurar parámetros generales como la selección del puerto serie, la placa, etc.

Menú de botones: Permite verificar, abrir, cargar o guardar en la placa el código del programa.

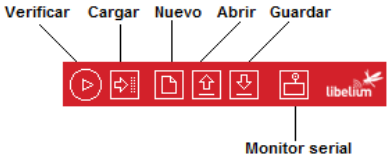

Fig. 28: Botones del menú del IDE

Código del programa: Es el código que se carga en Waspmote.

Mensajes de salida: En estos mensajes se muestran los posibles errores de compilación o carga ocurridos, y si el código es correcto, los mensajes de compilación satisfactoria.

En la figura 29 se puede ver un ejemplo de selección de tarjeta y puerto serial. En la pestaña "Herramientas/Tarjeta" se debe seleccionar la placa Waspmote (waspmote-API-v011 en este caso), mientras que en la opción de abajo, "Puerto Serial", se selecciona el puerto USB en el que se ha conectado la placa de Waspmote (COM4). El puerto no tiene por qué coincidir en distintos ordenadores (de hecho, lo normal es que no coincida) por lo que es recomendable verificar dicho puerto antes de compilar el programa:

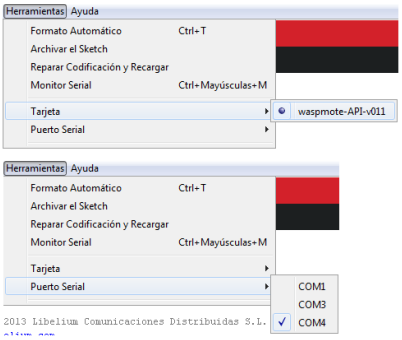

Fig. 29: Selección de la tarjeta y del puerto serial en el IDE

Para cargar el programa se debe pulsar el botón "Cargar" con la placa conectada al ordenador y comenzará la compilación, en la que Waspmote se reiniciará

automáticamente y comenzará la carga. El IDE mostrará una barra de carga y un mensaje cuando la carga esté completa, o mostrará un error, indicando con mensajes de color rojo los fallos en el código. A la hora de guardar, los programas completos (llamados sketches) tienen extensión *.pde.

Librerías

Las librerías proporcionan funcionalidad adicional para el uso de programas. Para utilizar una librería en un sketch, seleccione desde el menú "Sketch/Importar Librería". Esto insertará una o más sentencias `#include` en la parte superior del sketch. Algunas bibliotecas se incluyen con la API de Waspmote para facilitar la programación de aplicaciones, y otras se pueden descargar de diversas fuentes. Dicha API, desarrollada en C/C++, engloba todos los módulos que se integran en Waspmote además del manejo de otras funcionalidades, como las interrupciones o los diferentes estados energéticos.

Arquitectura del sistema

Como se ha visto en el apartado 3.2.2, la arquitectura del Waspmote se basa en el microcontrolador ATmega1281 de ATMEL. Esta unidad de procesamiento comienza a ejecutar el *bootloader* [7], que es responsable de cargar en la memoria los programas y las bibliotecas compiladas previamente almacenados en la memoria flash, por lo que el programa principal que se ha creado puede empezar a ejecutarse. Cuando Waspmote se conecta e inicia el *bootloader*, hay un tiempo de espera (62.5ms) antes de comenzar la primera instrucción para iniciar la carga de nuevas versiones de programas compilados.

La estructura de los códigos se divide en 2 partes básicas: *setup* y *loop*. Ambas partes del código tienen un comportamiento secuencial, siguiendo la ejecución de las instrucciones el orden establecido.

setup: Es la primera parte del código, que sólo se ejecuta una vez al iniciar el código. En esta parte es recomendable incluir la inicialización de los módulos que se van a utilizar, así como la parte del código que sólo es importante cuando se inicia el Waspmote.

[7] bootloader: gestor de arranque

loop: Se ejecuta continuamente, formando un bucle infinito. Debido al comportamiento de esta parte del código, es recomendable el uso de interrupciones para realizar acciones con Waspmote.

Una técnica de programación común para ahorrar energía se basa en el bloqueo del programa (ya sea mantener el micro despierto o dormido, en casos particulares) hasta que algunas de las interrupciones disponibles en Waspmote muestren que se ha producido un evento. De esta manera, cuando se detecta una interrupción de la función asociada almacenada previamente en un vector de interrupción, se ejecutará el programa. Para ser capaz de detectar la captura de interrupciones durante la ejecución del código, se han creado una serie de flags para indicar el evento que ha generado la interrupción. Un ejemplo de bloqueo de bucle, en que aparece y se trata la interrupción:

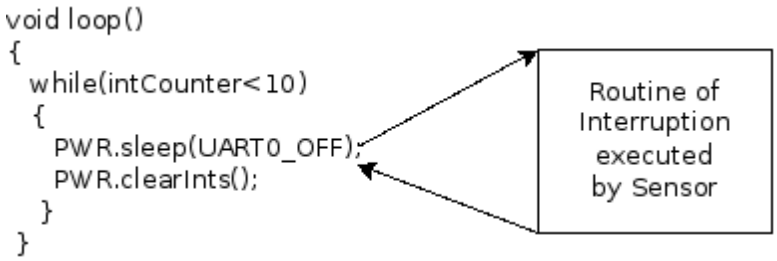

Fig. 30: Bloqueo del bucle

Cuando Waspmote se enciende o se reinicia, el código comienza de nuevo a partir de la función de configuración y luego la función de bucle. Por defecto, los valores de variables declaradas en el código y modificadas en ejecución se perderán cuando se produzca un reset o no haya batería. Para almacenar de forma permanente los valores, es necesario utilizar la memoria EEPROM del microcontrolador (4 KB). Las direcciones EEPROM de 0 a 1023 son utilizadas por Waspmote para guardar los datos importantes, por lo que no deben ser escritos sobre ellas. Por este motivo las direcciones de almacenamiento disponibles van de 1024 a 4095. Otra opción es utilizar de la capacidad de la tarjeta SD (2 GB).

4.2. Comunicación entre la placa Waspmote y el módulo XBee

El proyecto se ha diseñado siguiendo las siguientes consideraciones: se dispone de *n* sensores (temperatura, luz, etc.), una placa Waspmote en la que se carga un código con las órdenes necesarias y transmitir los datos a un PC por vía inalámbrica para ser tratados por una interfaz en Matlab. Para ello es de vital importancia conocer que tipo de *frames*[8] se pueden utilizar en la comunicación entre ellos, y hay de dos tipos: ASCII o Binario. Se ha optado por usar el tipo ASCII porque será más interesante trabajar con este formato cuando la información tenga que ser procesada por Matlab. El frame está pensado para facilitar la comprensión de los datos a enviar, y en él se pueden ver dos partes diferentes: la cabecera (*header*), siempre con la misma estructura, y el *payload* (carga útil), donde se incluyen los valores de los sensores:

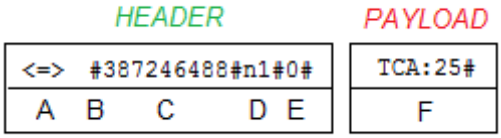

Fig. 31: Detalle de un frame ASCII

Partes de la cabecera

- ***Start Delimiter* [3 Bytes] (A)**: Necesario para identificar el inicio de cada frame.

- ***Separator* [1 Byte] (B)**: El carácter '#' funciona como separador. Se pone antes y después de cada campo de la trama.

- ***Serial ID* [10 Bytes] (C)**: Es un campo de como máximo 10 bytes, que identifica cada dispositivo Waspmote único. Sólo es de lectura y no puede ser modificado.

- ***Waspmote ID* [0-16Bytes] (D)**: Se trata de una cadena (string) definida por el usuario que puede identificar cada Waspmote dentro de la red del usuario. Si no se le da ninguno, el campo permanece vacío entre separadores: "##".

[8] frame: Técnica simplificada basada en conmutación de paquetes capaz de transmitir tramas especialmente útil para el envío de grandes cantidades de datos.

- ***Frame sequence* [1-3Bytes] (E)**: Este campo indica el número de frame de la secuencia. Este contador es de 8 bits, por lo que va de 0 a 255. Como es un frame ASCII, el número se convierte en una cadena con el fin de ser entendido. Esta es la razón por la que la longitud de este campo varía entre uno y tres bytes. Cada vez que el contador alcanza el máximo (255) se repone a 0. Este número se utiliza con el fin de detectar la pérdida de frames.

Partes del *payload*

En esta parte se incluye la información recogida por los sensores. Pueden ser de 3 tipos:

- **Datos simples (F)**: El campo del sensor está compuesto por un único valor, como se puede apreciar en la figura 30.

- **Datos complejos**: No se utilizan en este proyecto ya que son específicos del acelerómetro y del GPS.

- **Datos especiales**: Diseñado para incluir la fecha y la hora en formato especial (yy-mm-dd).

Campos de sensores

Existe un gran número de sensores a utilizar en esta plataforma, por lo que es necesario un sistema de identificación para poder emplearlos correctamente:

- **Referencia**: Es el número de referencia de sensor dada por Libelium a cada uno en el catálogo de sensores.

- **Sensor TAG**: Define las constantes necesarias para agregar cada sensor al frame mediante la función addSensor().

- **Identificador del sensor (ID)**: Cada campo de sensor tiene su propio ID. Dependiendo del TAG del sensor elegido, se establecerá un ID diferente como

identificador del sensor. Los frames ASCII utilizan un *label* (etiqueta) como identificador.

- **Número de campos**: Define el número de campos distintos que presenta un valor del sensor. La mayoría de los sensores sólo necesitan un único campo, aunque se dan algunas excepciones como el módulo GPS, que necesita 2 campos para realizar mediciones de latitud y longitud.

- **Tipo y tamaño**: Indica el tipo de variable que tiene que utilizarse para cada sensor. Las posibilidades son: *uint8_t* (1 Byte*), int* (2 Bytes), *float* (4 Bytes), *unsigned long* (4 Bytes), *string* (tamaño variable). Los frames ASCII no tienen limitaciones cuando se añaden campos para nuevos sensores para facilitar al usuario la inserción de nuevos datos.

- **Precisión Decimal por Defecto (*Default Decimal Precision*)**: Especifica para cada sensor el número de decimales utilizados en los frames ASCII al utilizar variables de tipo *float*.

- **Unidades**: Fija las unidades utilizadas para cada sensor.

En la siguiente tabla se puede ver cada una de las partes que integran el campo de un sensor, en este caso el de temperatura. Si es necesario añadir más campos de datos se puede consultar en la referencia [6] y aplicarla del mismo modo que éste. En el punto 5.5 se describe esta situación:

Sensor	Referencia	Sensor TAG	Sensor ID(ASCII)	Nº de campos	Tipo	Tamaño	Default Decimal Precision	Unidades
Temperatura (Celsius)	9203	SENSOR_TCA	TCA	1	float	4	2	ºC

Tabla 5: Estructura del campo de tipo temperatura.

4.3. Funcionamiento del programa. Diagrama de bloques del código

El código implementado en el IDE de Waspmote se ha realizado siguiendo las siguientes especificaciones:

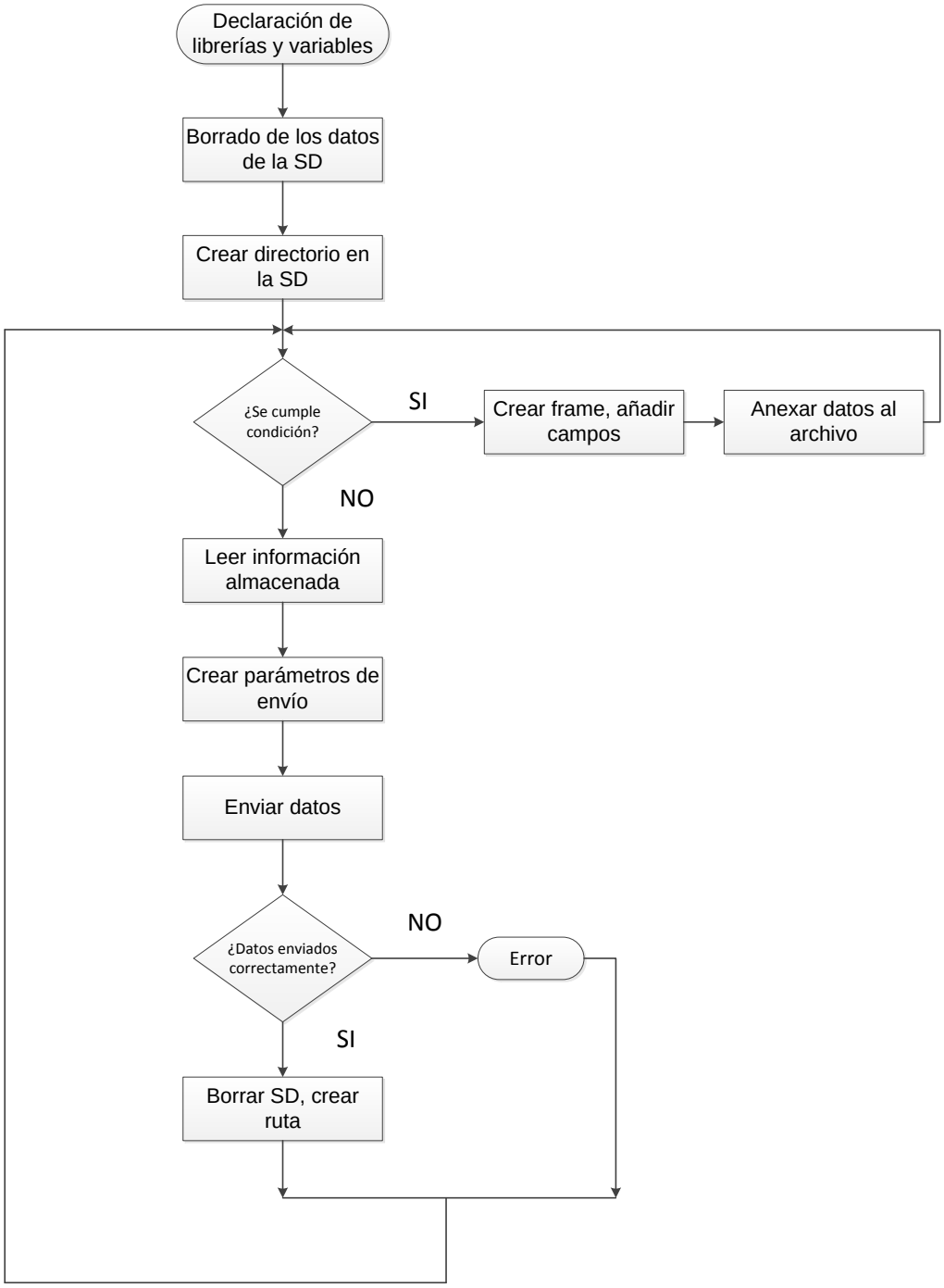

Fig. 32: Diagrama de bloques del programa.

Descripción

Se declaran las variables y librerías necesarias para el funcionamiento del programa.

EN EL *SETUP*: Se enciende el RTC y la tarjeta SD. Se borran los datos que hubiera en el directorio especificado, para a continuación crear uno nuevo. De esta manera se reinicia al inicio del programa y se libera espacio para las próximas medidas.

EN EL *LOOP*: Dentro de un bucle cuya condición está supeditada al tiempo que se haya decidido al inicio del programa (10 minutos, por ejemplo), se realiza la obtención de las mediciones realizadas por los sensores, así como su incorporación dentro del frame en los campos habilitados a tal efecto. Con el frame completo se procede a su guardado en la tarjeta SD, añadiendo una línea tras otra con cada iteración del bucle. Entre cada medida hay un pequeño retardo que será utilizado en el interfaz a la hora de calcular las muestras que se deseen, por ejemplo, si se quieren 50 muestras y se desea que el programa obtenga valores durante 30 minutos se necesita un *delay* de 36 segundos. Esto es algo que es necesario decidir al principio de la ejecución del programa, ya que una vez cargado el código en la placa, si hay que cambiar estos valores se tendría que volver a cargar el programa nuevamente.

Fuera del bucle, el programa analizará el contenido de la tarjeta SD y enviará al módulo XBee, en orden de medida, cada uno de los frames almacenados. Cuando se termine de enviar, se limpiará de nuevo la SD para optimizar el espacio de almacenamiento disponible, y se reiniciará la ejecución. En la figura 33 se puede ver un ejemplo de la ejecución del programa a través del Monitor Serial en un caso sencillo, enviar 2 muestras cada 10 segundos:

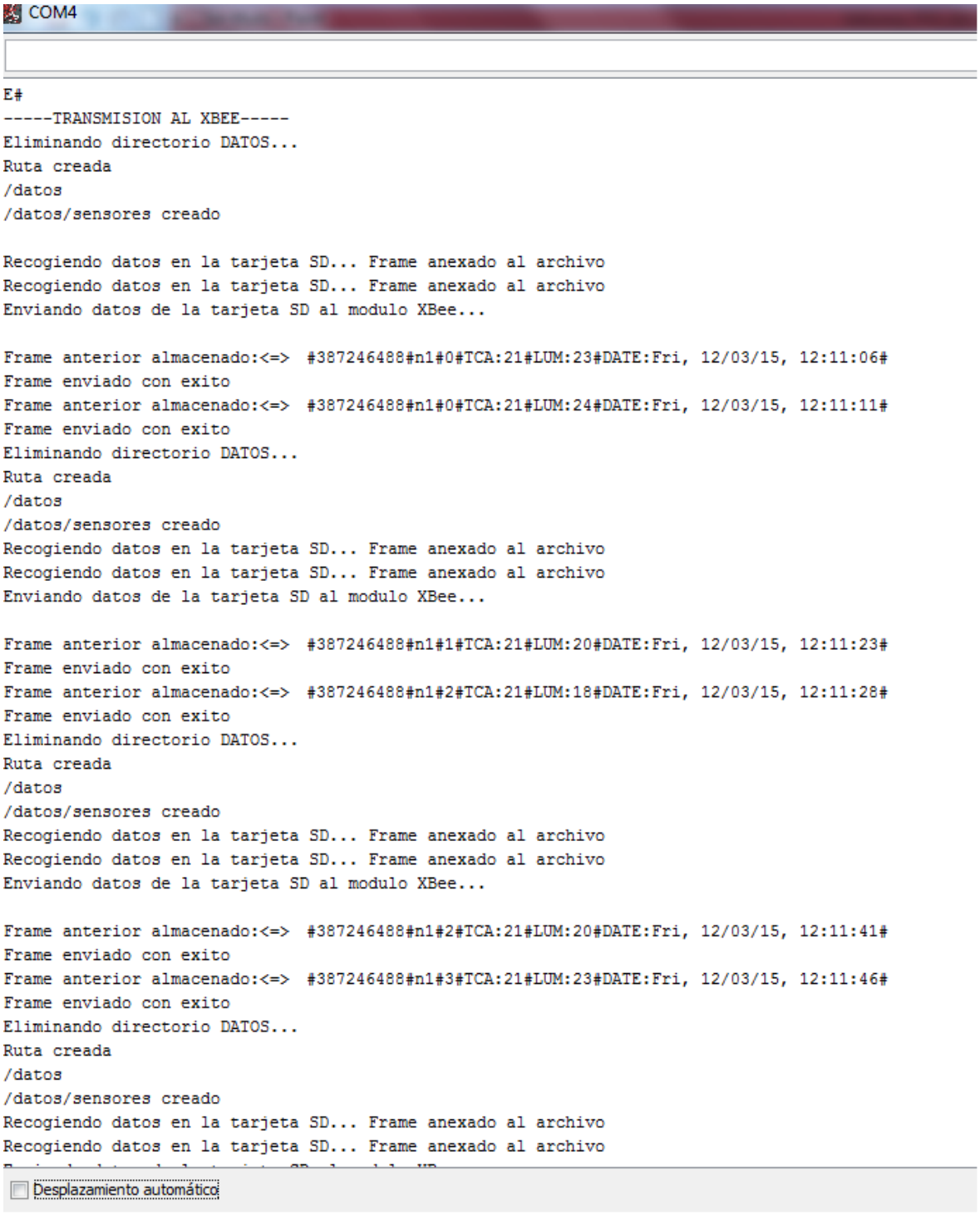

Fig. 33: Captura de la ejecución del programa

5. Aplicación desarrollada en Matlab GUIDE

5.1. Introducción

Para la recepción, tratamiento y visualización de los datos procedentes de Waspmote se ha realizado un programa en Matlab. Se ha elegido este entorno y no otro (como Labview, Visual Basic, Visual C++…) por la versatilidad, sencillez y eficacia de programación que ofrece debido al gran número de funciones ya implementadas que hay en esta plataforma. El motivo más importante por el que finalmente se ha decidido utilizar Matlab que proporciona una visualización de los resultados obtenidos rápida y efectiva y, además, permite la creación de un entorno gráfico que puede controlar los distintos parámetros del sistema, obteniendo una simulación de resultados óptima para el usuario. Para ello se va a emplear el entorno de desarrollo GUIDE de Matlab.

5.2. Aplicación de usuario

Mediante la aplicación el usuario puede arrancar en un archivo *.exe* las tareas desde esa pantalla. Para realizar el programa con Matlab GUIDE lo primero que hay que hacer es acceder al propio GUIDE, escribiendo `>> guide` en la ventana de comandos de Matlab. Ahora aparece una pantalla que presenta la opción de cargar un proyecto anterior o comenzar uno nuevo, con varias posibilidades: una plantilla en blanco o tres opciones prediseñadas para hacer más sencillo el camino al usuario. En este caso se ha optado por la plantilla en blanco (*Blank GUI*):

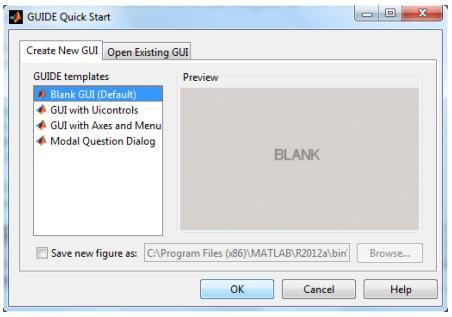

Fig. 34: Pantalla de inicio del GUI

Una vez seleccionada la opción se abre una pantalla que se divide en varias partes: el área de diseño y la paleta de componentes y herramientas como Alinear objetos, Editor de menú, Editor de orden de etiqueta, Editor de la barra de herramientas, Editor del M-file, Propiedades, Navegador de objetos y Grabar y ejecutar. Por defecto los componentes aparecen sin etiqueta, para identificarlos como se ve en la figura 35 se va a *File >> Preferences >> Show names in component palette:*

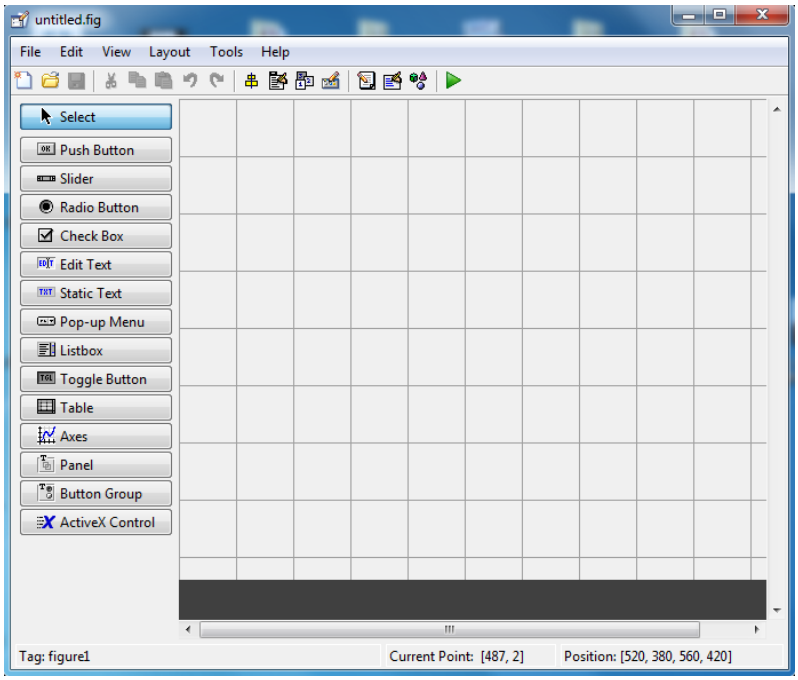

Fig. 35: Entorno de diseño del GUI con componentes etiquetados.

El método para programar con GUIDE se encuentra entre la POO[9] y la programación en Matlab, constando de dos archivos: un fichero `.m` con el código responsable de la ejecución del programa y otro `.fig` que contiene los elementos gráficos. Aparte de esto, los valores de las propiedades de los elementos (como su valor, color, posición, si es o no un *string*…) y los valores de las variables transitorias del programa se guardan en una estructura concreta, y se accede a ellos por medio de un único identificador idéntico para todos ellos. Un ejemplo de asignación podría ser:

```
handles.output = hObject;
```

[9] POO: Programación Orientada a Objetos.

handles es el identificador a los datos de la aplicación. Esta definición de identificador se guarda con la instrucción *guidata,* que es la función que guarda las propiedades y las variables de los elementos en la estructura de datos de la aplicación:

```
guidata(hObject, handles);
```

De forma genérica, en cada subrutina se debe escribir en la última línea dicha sentencia, ya que nos asegura que cualquier asignación o cambio en las propiedades o en las variables quede correctamente almacenado. Por ejemplo, se tiene una operación dentro de una subrutina que da como resultado una variable *dato*; para que pueda ser utilizada por otra subrutina o desde el mismo programa se debe guardar así:

```
handles.dato = dato;
guidata(hObject,handles);
```

En la primera línea se crea la variable *dato* en la estructura de datos apuntada por *handles*, y en la segunda se graba el valor de dicha variable.

5.3. Elementos del GUIDE

En la siguiente tabla se muestra una descripción de los componentes de que se dispone:

Elemento	Descripción
Push Button	Invoca una acción al pulsarlo
Slider	Representa valores en un determinado rango
Radio Button	Indica una opción que puede seleccionarse
Check Box	Indica el estado de una opción
Edit Text	Permite editar texto
Static Text	Muestra un *string* de texto
Pop-up Menu	Proporciona una lista de opciones
Listbox	Muestra una lista en formato deslizable
Toggle Button	Sólo permite dos acciones: “on”, “off”
Table	Inserta una tabla

Axes	Inserta un cuadro de tipo gráfica
Panel	Aúna varios botones en un grupo
Button Group	Panel de uso exclusivo para radio buttons y toggle buttons
ActiveX control	Abre controles ActiveX en GUIDE

Tabla 6: Elementos del GUIDE

5.4. Propiedades de los componentes

Cada elemento del GUIDE tiene un conjunto de opciones al que se puede acceder haciendo clic derecho sobre el objeto. Aquí podemos destacar dos de las opciones:

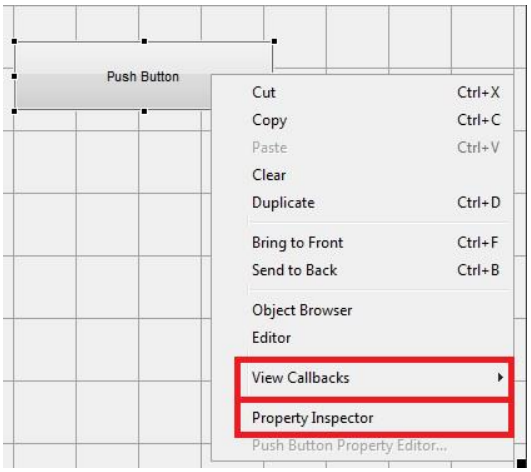

Fig. 36: Propiedades del elemento.

Property Inspector: Reúne los valores de las diferentes propiedades del elemento en cuestión, desde el texto que le da nombre hasta el color de fondo.

View Callbacks: El manejo de datos entre los elementos de la aplicación y el archivo .m está unido a través de las subrutinas *Callback*. Las hay de diversos tipos dependiendo de la función que desempeñen. Cuando se ejecuta una de estas opciones, Matlab abre el archivo .m asociado y automáticamente nos coloca en la parte del programa correspondiente a la subrutina seleccionada. En ella el usuario podrá escribir aquello que pretenda hacer el elemento, por ejemplo, cuando se apriete un botón que se dibuje una gráfica.

5.5. Diagrama de bloques de la aplicación

El funcionamiento de la aplicación puede verse en la figura 37, en la que se resumen los diferentes botones y procesos asociados en su ejecución. Previamente, se advierte al usuario de la necesidad de insertar un número de muestras en un cuadro dispuesto a tal efecto. Si no se introduce un valor correcto se mostrará un mensaje de error. En el diagrama se pueden ver los cuatro botones disponibles:

ADQUIRIR DATOS: Abre la comunicación entre el puerto serie al que está conectado el módulo XBee y el interfaz. Si la comunicación entre ambos dispositivos es incorrecta saldrá “ERROR” en la ventana de comandos de Matlab, y si es correcta se reciben las tramas deseadas, de las que se procederá a extraer la información. Cuando termine este proceso se mostrará por pantalla un mensaje informando sobre el número de muestras que se han recibido finalmente (por si ha habido pérdidas) y el tiempo de ejecución empleado. En el punto 5.6 se explica en detalle el método de adquisición.

REPRESENTAR DATOS: Muestra en figuras separadas las gráficas con los datos extraídos de los sensores.

REINICIAR: Resetea los valores de las muestras y limpia los elementos con imágenes para volver a realizar medidas.

EXTRAER DATOS: Incorpora los datos a un fichero de texto (.txt) con la fecha y hora de su medida. Los nuevos datos se añaden a continuación de los ya existentes.

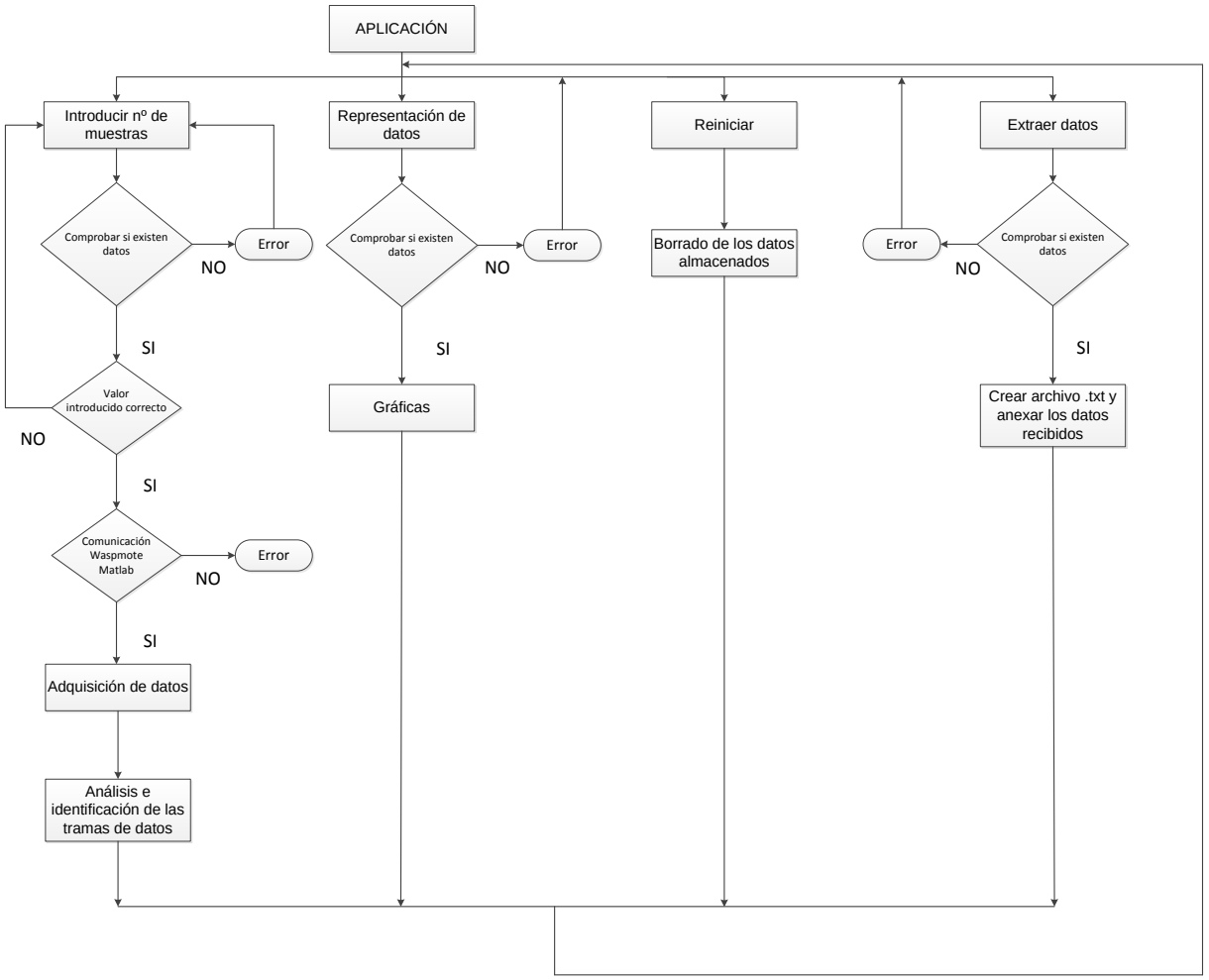

Fig. 37: Diagrama de bloques de la aplicación en MATLAB.

5.6. Adquisición de datos

La obtención de los datos a través de Matlab consta de dos procesos:

5.6.1. Recepción de datos

Al inicio el programa comprueba si el puerto *Puerto seleccionado* está disponible. Si no lo está (porque no existe o está siendo utilizado por otro programa) se avisa al usuario, indicando a su vez cuáles son los que sí están disponibles. Si es correcto se procede a su apertura:

```
delete(instrfind({'Port'},{'Puerto seleccionado'}));
puerto = serial('Puerto seleccionado','BaudRate',115200);
```

Es muy importante indicar en este punto del código el tamaño de la cantidad de información que se va a recibir en el buffer, ya que si se recibe más de la que tiene prefijada (por defecto 512 bytes) habrá error en la medida. Matlab permite saber cuál es el tamaño de una trama, así que previamente se realiza una ejecución a modo de ejemplo pidiendo 1 muestra. En el *Workspace* de Matlab se puede ver:

Fig. 38: Tamaño de la trama

La variable *valor* indica que se ha recibido un vector columna de 80 elementos, por lo que ése es el tamaño de cada trama. Si se añadiesen más campos de medida este elemento tendría que recalcularse. Se aplica dicho valor al buffer:

```
puerto.InputBufferSize = N_muestras * 80;
```

Una vez definido esto se procede a abrir el puerto. Como se ha visto en la figura 38 los datos se leen con la instrucción `fread`, y se guarda el resultado en *valor* con formato numérico. A continuación se guarda en otra variable (*entrada*) la conversión de dichos valores a sus correspondientes caracteres ASCII. Esto es necesario para la siguiente parte del proceso.

5.6.2. Tratamiento e identificación de información

Para realizar esta tarea se ha escrito un pseudocódigo que describe cómo se obtiene la temperatura:

```
Subproceso Adquirir_Temperatura

Leer tam = tamaño(valor);
aux = 1;

Para i = 1 hasta tam
    % Valor de 3 dígitos
    Si valor(i) = 67 Y valor(i+1) = 65 Y valor(i+2)= 58 Y valor(i+6)= 35

        temp1(aux)  = entrada(i+5);
        temp2(aux)  = entrada(i+4);
        temp3(aux)  = entrada(i+3);
        aux = aux+1;

    % Valor de 2 dígitos
  ó Si valor(i) = 67 Y valor(i+1) = 65 Y valor(i+2)= 58 Y valor(i+5)= 35
        temp1(aux)  = entrada(i+4);
        temp2(aux)  = entrada(i+3);
        temp3(aux)  = '0';
        aux = aux+1;

    Fin Si
Fin Para
```

```
% Unir caracteres y convertir a número

temp_final = temp3 + temp2 + temp1;

temp_final = pasar_string_a_numero(temp_final);
```

Los números 67, 65, 58 y 35 son los equivalentes a C, A, : y # en caracteres ASCII, de forma que se guardan sólo los elementos comprendidos entre esos caracteres. Para sacar la luminosidad y la fecha y hora de cada medida se ha seguido el mismo sistema. Las casillas verdes son los elementos a guardar:

LUMINOSIDAD						
ASCII	M	:	Num	Num	Num	#
DEC	77	58	---	---	---	35

DÍA					
ASCII	,	espacio	Num	Num	/
DEC	44	32	---	---	47

MES				
ASCII	/	Num	Num	/
DEC	47	---	---	47

AÑO				
ASCII	/	Num	Num	,
DEC	47	---	---	44

HORAS				
ASCII	espacio	Num	Num	:
DEC	32	---	---	58

MINUTOS				
ASCII	:	Num	Num	:
DEC	58	---	---	58

SEGUNDOS							
ASCII	:	Min	Min	:	Num	Num	#
DEC	58	---	---	58	---	---	35

Tabla 7: Patrones de búsqueda

En el apartado de pruebas experimentales se podrá ver durante el transcurso de este proceso un *.gif* que se ha incluido en el interfaz para simbolizar el envío de información satisfactorio de la placa al Matlab, y una barra de progreso. Ambos se cerrarán automáticamente cuando el proceso termine.

5.7. Montaje del proyecto

Este montaje está enfocado a realizar pruebas para comprobar el funcionamiento de todos los elementos:

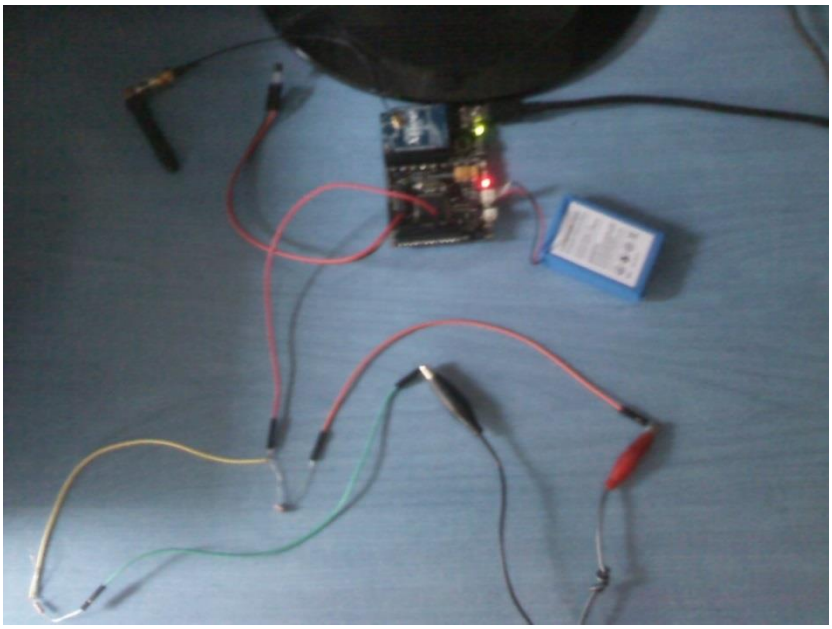

Fig. 39: Elementos del proyecto

Elementos usados (en la imagen):

- Adaptador de red AC/DC con pinzas de cocodrilo (configurado a 4.5V).
- Sensor de temperatura MCP9700A.
- Sensor de luz.
- Resistencia de 10 kΩ.
- Placa Waspmote Pro v 1.2.
- Batería de litio recargable.
- Cables eléctricos.
- Cable USB.

Se necesita también conectar el módulo USB en el PC para recibir los datos.

6. Pruebas experimentales

Obtener 50 muestras de temperatura y luz durante 20 minutos.
Distancia entre Placa – Módulo USB: 17 metros.

1. Se calcula la cantidad de *delay* necesario entre muestras, para este caso 24 segundos y se carga el programa en la placa. Se inicia Matlab y se introduce el número de muestras:

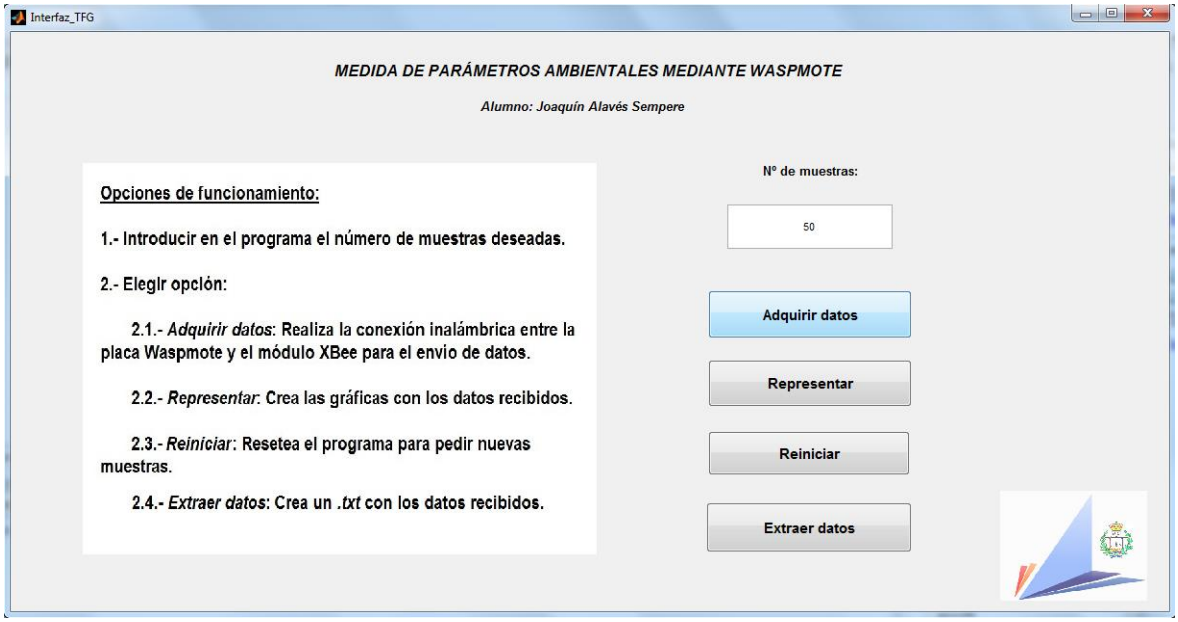

Fig. 40: Paso 1.

2. Al pulsar el botón “Adquirir datos”, se pide confirmar que se desea establecer comunicación. Al hacerlo empezará la obtención de los datos. Durante este proceso aparece en el cuadro izquierdo una imagen .GIF activa sólo durante dicho período, así como una barra de progreso:

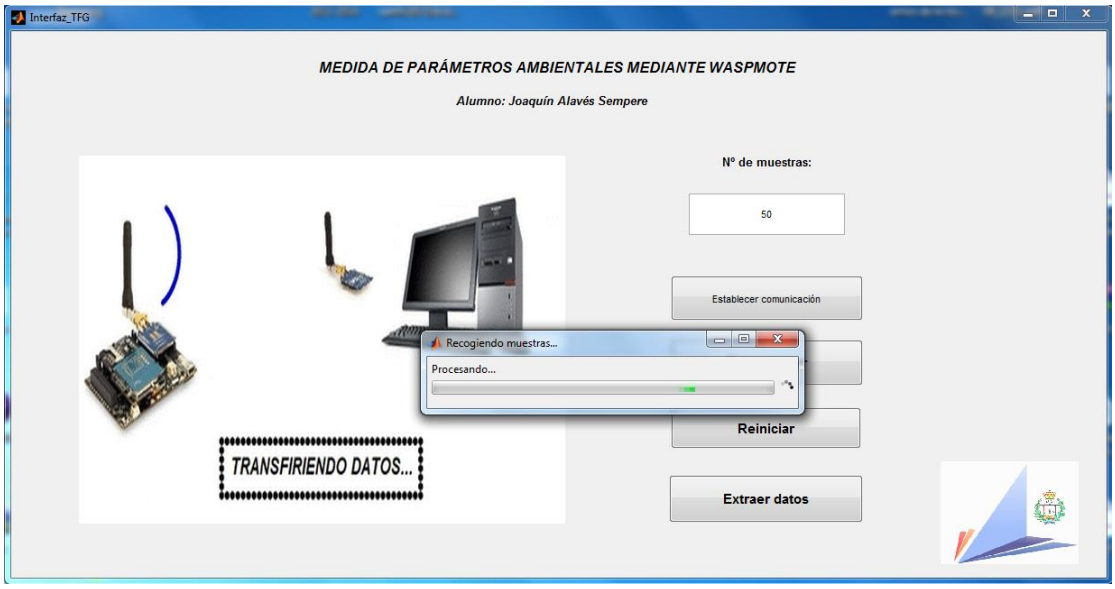

Fig. 41: Paso 2.

3. Al terminar, se avisa al usuario mediante un mensaje con las muestras obtenidas y el tiempo empleado para ello. Se han obtenido 32 muestras en vez de las 50 deseadas, y eso es debido a que el *Timeout* puesto en Matlab no ha sido lo suficientemente grande:

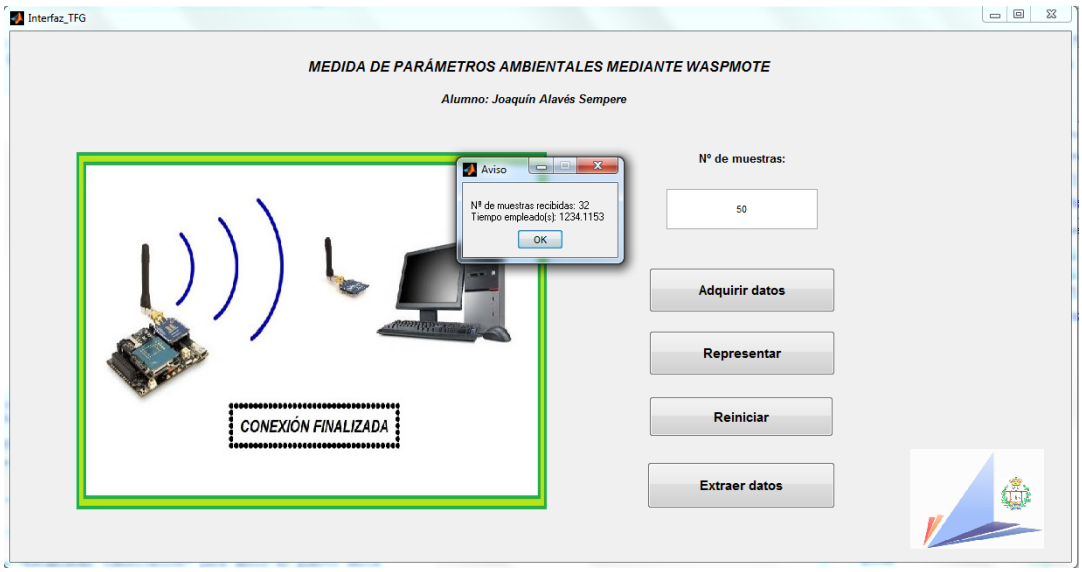

Fig. 42: Paso 3.

4. Al pulsar el botón “Representar” se nos muestran las gráficas en dos figuras con los valores recogidos:

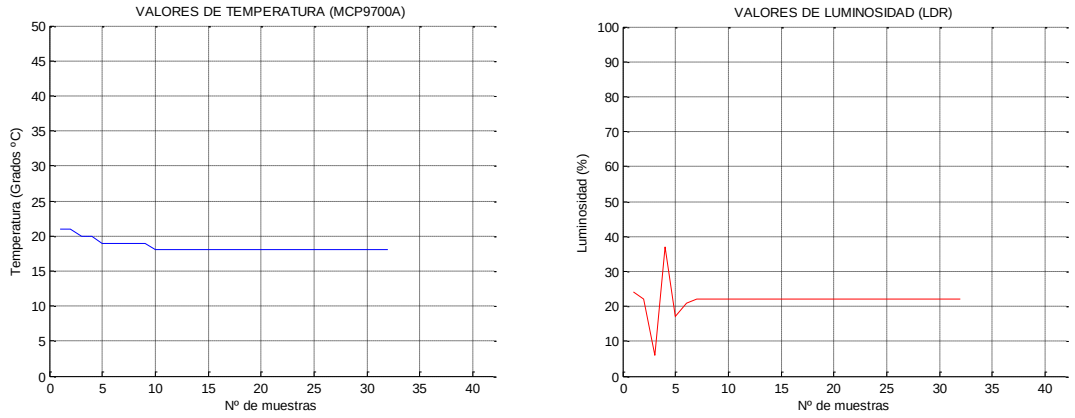

Fig. 43: Paso 4.

5. Finalmente, se pueden extraer los datos en un fichero de texto indicando cada dato junto a la fecha y hora de su obtención. Al pulsar dicho botón aparecerá el mensaje “Datos guardados con éxito” y se creará en la carpeta en la que esté alojado el interfaz un .txt:

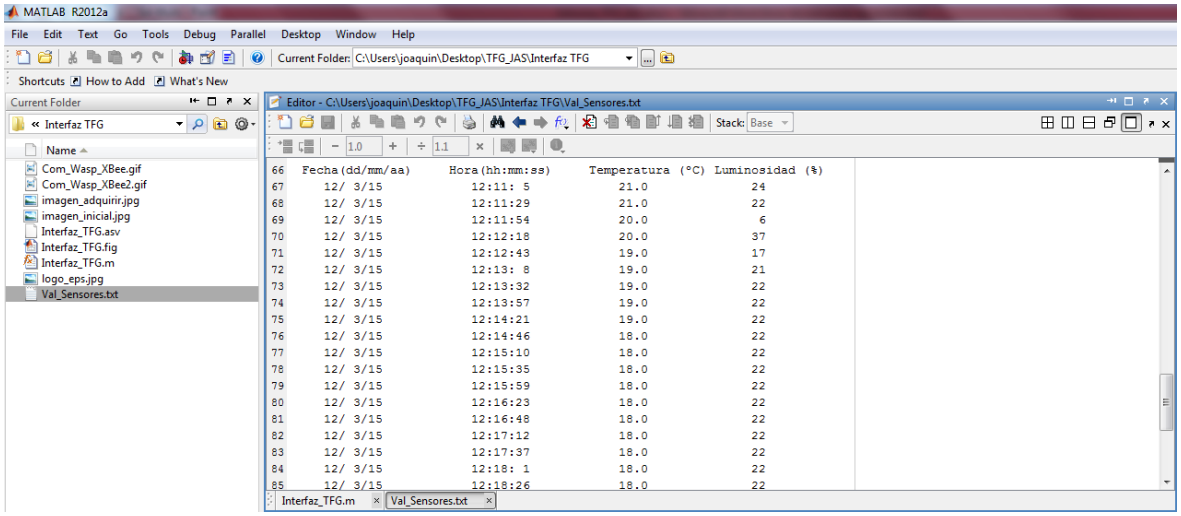

Fig. 44: Paso 5.

El paso 3 refleja el problema principal que hay que mejorar: se hace necesario encontrar un método que, a partir de las muestras, calcule un determinado valor temporal

que permita abarcar la totalidad de las medidas a realizar. En el siguiente punto se detalla cómo hacer dicho cálculo.

6.1. Estimación del *Timeout*

Para poder realizar la aproximación de este parámetro es necesario medir la ejecución del programa en la placa de Waspmote. Para ello se ha optado por poner un tiempo corto pero lo suficientemente grande como para poder medirlo, en este caso:

Tiempo de ejecución: 30 segundos
Retardo entre muestras (*Delay)*: 5 segundos
Nº de muestras: 6

Viendo una captura del Monitor Serial, se puede dividir el programa en las siguientes partes:

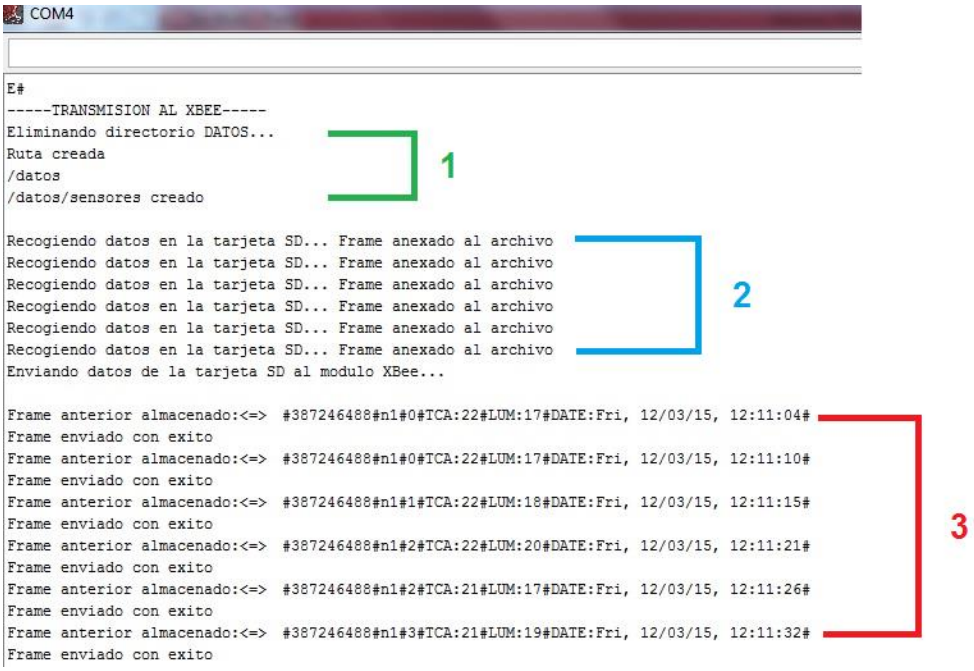

Fig. 45: Partes de la ejecución.

Se han obtenido los siguientes valores en las mediciones de los tiempos:

1º Parte: Tiempo empleado en el borrado y creación de directorios en la SD = 1.11 s
2º Parte: Recogida y guardado de los datos medidos en la tarjeta SD = 33.90 s
3º Parte: Envío de los *frames* con los datos al módulo XBee = 3.48 s

Total: 38.49 s

Con estos valores se puede realizar una estimación genérica del tiempo que se debe emplear en Matlab:

$$valor_{Timeout} = t_{EjecuciónBCD} + (t_{delay} * nº\ muestras\ ) + t_{EnvioMuestras} \quad (3)$$

Donde $t_{EjecuciónBCD}$ representa la 1ºparte (cuyo valor será 1.11 + (33.90 - 30) = 5.01 s), $(t_{delay} * nº\ muestras\ )$ el tiempo de ejecución (30 s) y $t_{EnvioMuestras}$ la 3º parte dividida por el número de muestras (el envío de una muestra sería $\frac{3.24\ s}{6\ muestras} = 0.58\ s/muestra$).

Para que esto funcione, hay que tener en cuenta que para la toma de datos se necesita que el programa se ejecute al menos una vez, según el ejemplo descrito esto es aproximadamente 39 s. En este caso serviría si cuando se realiza la conexión con Matlab coincidiese con el inicio de la ejecución del programa de Waspmote, pero no tiene por qué ser así, lo que significa que no se obtendrían todas las muestras.

Por ello, se ha optado por la solución de multiplicar por 2 dicho tiempo (el equivalente a dos ejecuciones del código de Waspmote) para las muestras menores o iguales a 6, pero considerando 6 el número de muestras en todos los casos. En la siguiente tabla se refrenda la ventaja de utilizar este método:

Nº muestras	T_ejecución experimental (s)	T_ejecución teórico (s)
1	37.22	2 * 38.49
2	37.76	
3	39.57	
4	39.26	
5	39.89	
6	40.52	

Tabla 8: Comparativa de tiempos (muestras <= 6).

Para los valores mayores que 6 se utiliza en la fórmula el número de muestras que se introduzca, multiplicando todo por un parámetro *p* producto de la división entre el número introducido y las muestras de Waspmote, que simboliza el número de ejecuciones que necesita la placa para enviar esas muestras:

Nº muestras	T_ejecución experimental (s)
7	74.78
8	75.01
9	75.67
10	76.91
17	151.61
30	189.69
78	523.29

Tabla 9: Comparativa de tiempos (muestras > 6).

La conclusión que se extrae de las tablas es que, ante la imposibilidad de poder comunicar Matlab con la placa (y viceversa) por el hecho de ser una comunicación modo símplex, no se puede seguir un patrón para localizar dónde se encuentra la ejecución en cada momento. La multiplicación por 2 y por *p* permite aumentar la precisión temporal y mejorar la eficiencia del programa, ya que las muestras se reciben en todos los casos.

Programa en Waspmote diseñado para medir 5 muestras cada 15 minutos.

Obtener 10 muestras de temperatura y luz.

Lugar: Habitación bien iluminada.

Delay necesario:

$$\frac{15\ minutos}{5\ muestras} = \frac{900000\ ms}{5\ muestras} \approx 180000\ ms/muestra.$$

Recepción de las muestras:

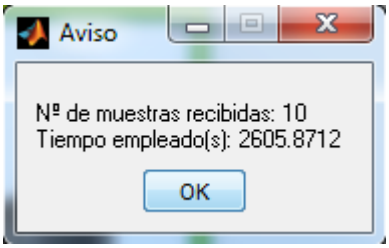

Fig. 46: Muestras recibidas

Representación:

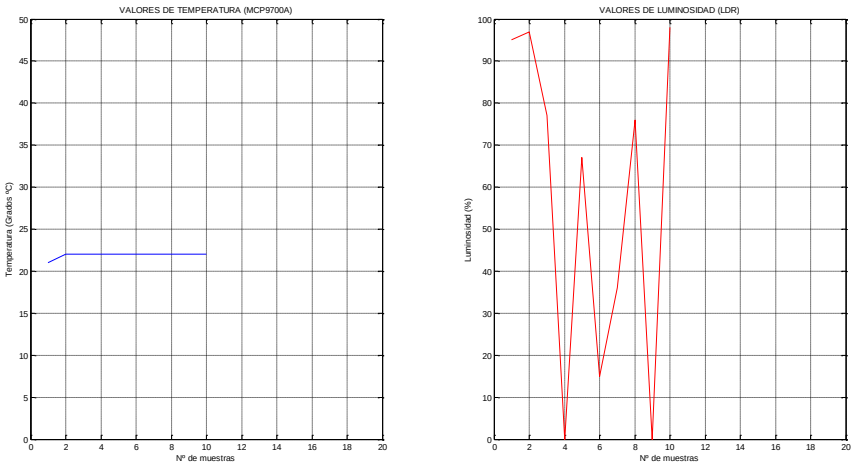

Fig. 47: Gráficas obtenidas

Obtener 27 muestras de temperatura y luz.

Recepción de las muestras:

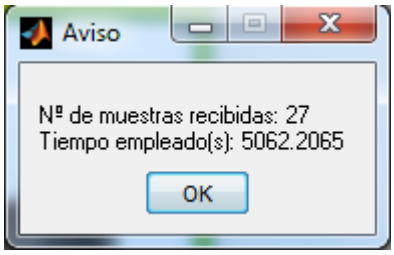

Fig. 48: Muestras recibidas

Representación:

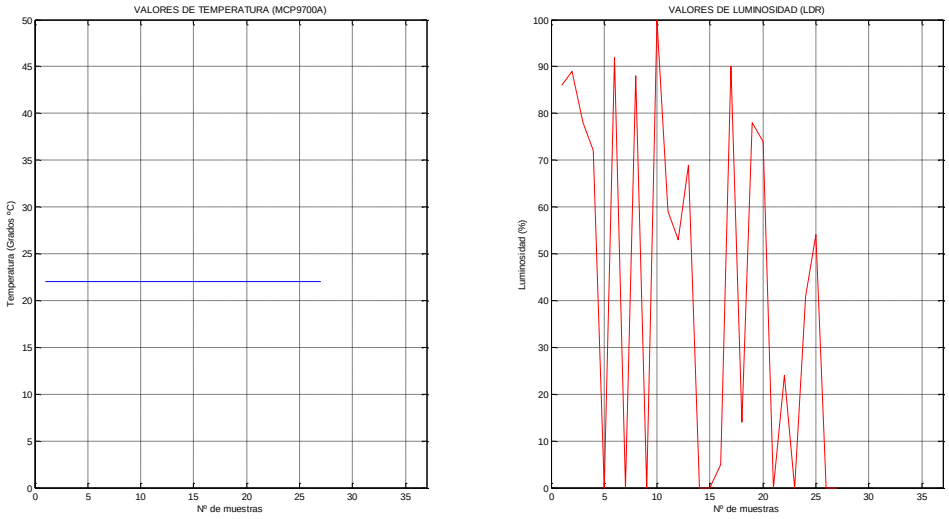

Fig. 49: Gráficas obtenidas

Obtener 60 muestras de temperatura y luz.

Recepción de las muestras:

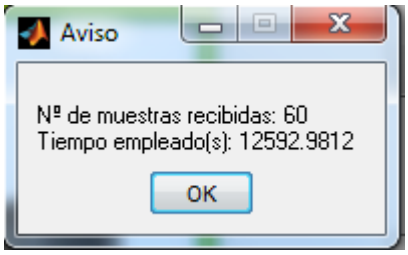

Fig. 50: Muestras recibidas

Representación:

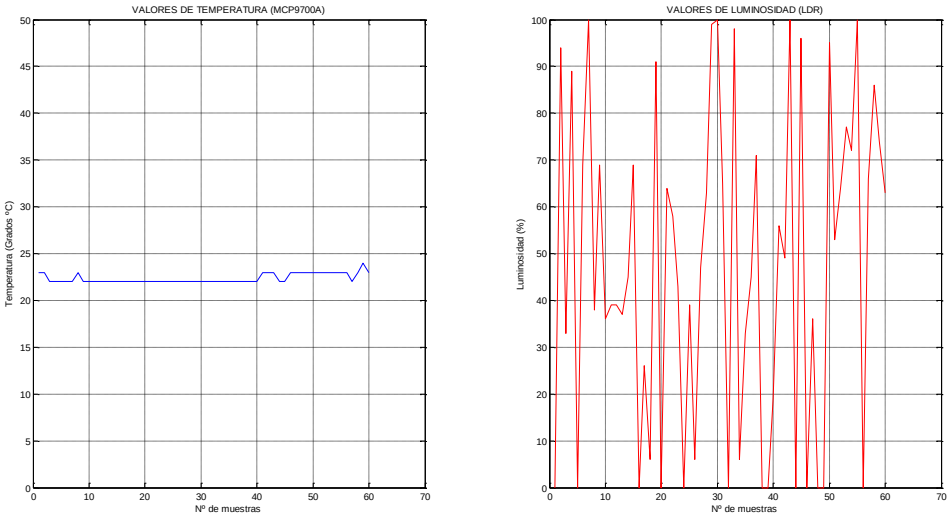

Fig. 51: Gráficas obtenidas

Programa en Waspmote diseñado para medir 1 muestra cada 10 minutos.

Obtener 3 muestras de temperatura y luz.

Delay necesario: 600000 ms

.

Recepción de las muestras:

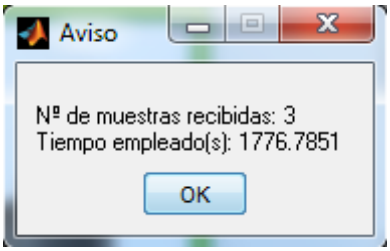

Fig. 52: Muestras recibidas

Representación:

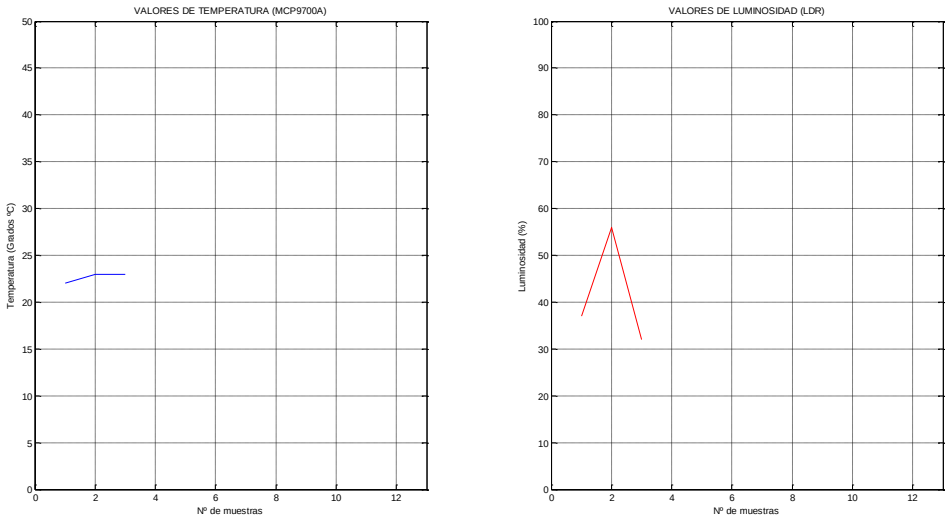

Fig. 53: Gráficas obtenidas

Obtener muestras de temperatura y luz durante 2 horas (12 muestras).

Recepción de las muestras:

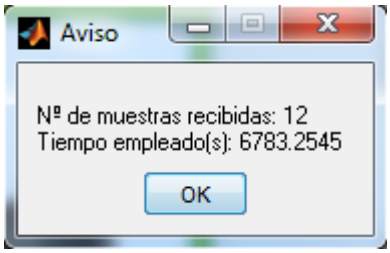

Fig. 54: Muestras recibidas

Representación:

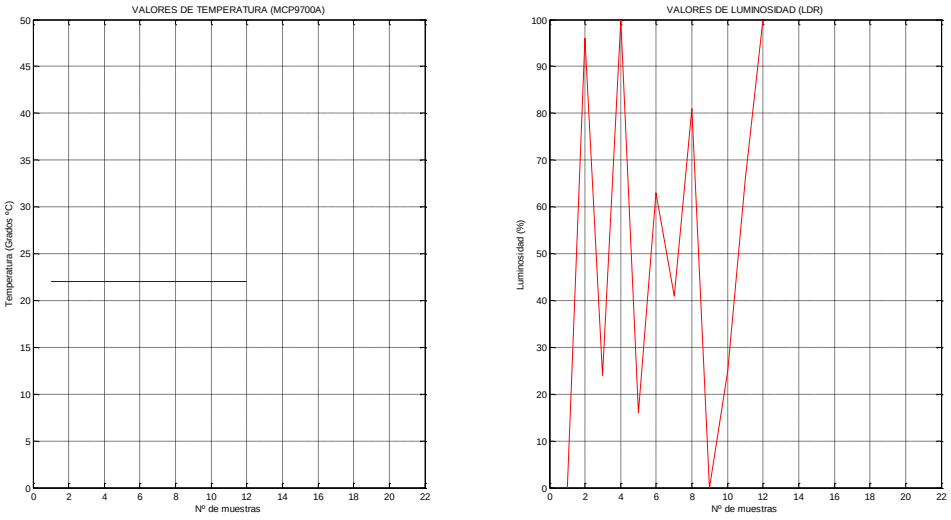

Fig. 55: Gráficas obtenidas

Obtener muestras de temperatura y luz durante 5 horas (30 muestras).

Recepción de las muestras:

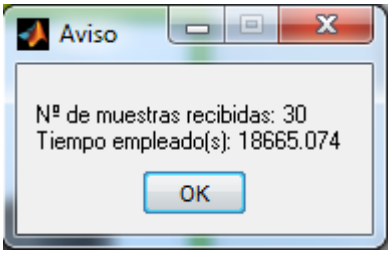

Fig. 56: Muestras recibidas

Representación:

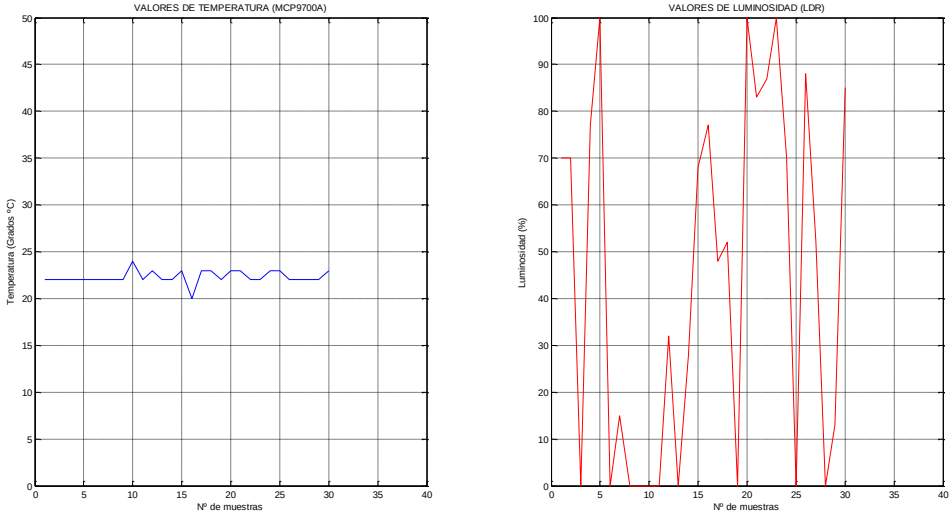

Fig. 57: Gráficas obtenidas

6.2. Medición en entornos remotos. Uso de interrupciones.

El programa realizado en Waspmote que se ha descrito en este proyecto ha sido diseñado para su uso en un entorno doméstico, en el que es posible la carga directa de la batería, y que por tanto no tiene problemas de consumo. No obstante, si el dispositivo tiene que alimentarse únicamente de la batería es recomendable el uso de interrupciones (véase apartado 3.2.2.2). Para ofrecer esta alternativa al usuario y tomando como punto de partida dicho programa, se puede considerar la utilización de una interrupción de tipo *Deep Sleep* o incluso *Hibernate* ya que, como se ha visto, el RTC permite despertar al microcontrolador de un estado de bajo consumo mediante interrupciones. En el siguiente punto se va a realizar un estudio basado en los modos propios de funcionamiento de Waspmote y del consumo que éstos necesitan.

6.2.1. Consumo de la batería en los distintos modos

Sabiendo que la capacidad nominal de la batería es 2300 mAh y si sólo se empleara uno de estos modos ininterrumpidamente se tendrían los siguientes valores de duración:

Modo	Consumo	Duración
ON	15 mA	153,33 horas ≡ 6,38 días
S / DS	55 µA.	4,77 años
HIBERNATE	0.06 µA	4.375,95 años

Tabla 10: Duración de la batería.

De la tabla se pueden extraer que, efectivamente, el uso de los modos de bajo consumo alarga ostensiblemente la capacidad de la batería siendo especialmente

significativo el modo hibernación. No obstante, la placa deberá despertarse en algún momento para realizar la tarea que se le haya programado. Tomando como ejemplo el programa diseñado en este proyecto (6 muestras cada 5 segundos, ejecución de 39 s de duración) y considerando un tiempo de hibernación de una hora, se puede calcular el consumo total del programa:

Ejecución (Modo ON): 1,059 x 10^{-6} A ≡ 1,059 μA

Hibernación: 1,565 x 10^{-15} A ≡ 1,565 fA

Con los cálculos obtenidos se puede considerar que el consumo en hibernación es despreciable en comparación al de ejecución, y por tanto la placa sólo consumirá recursos durante esos 39 s. Con todo lo expuesto se puede averiguar cual sería la duración de la batería para este programa funcionando de forma continua:

$$Duración\ de\ la\ batería\ \approx \frac{2300\ x\ 10^{-3} Ah}{1{,}059\ x\ 10^{-6}\ A} = 2{,}17x\ 10^{6}\ h\ = 247{,}93\ años \quad (4)$$

Como se puede ver, de una duración de apenas 6,38 días (tabla 10) se ha pasado a casi 248 años. Por esto el uso de interrupciones es muy útil para este tipo de programas, multiplicando el rendimiento de la batería de forma excepcional. En los siguientes apartados se pueden ver dos ejemplos de interrupciones empleadas.

6.2.2. Modo *Deep Sleep*

Se realiza la misma operación dentro del bucle while que se encarga de almacenar las medidas en la SD, pero prescindiendo de la función `millis()` para que el bucle no se base en el tiempo propio de la ejecución y sí en el de las interrupciones. Se mantiene un valor de *Timeout* y se añade un contador que se va actualizando con cada medida, y al salir del bucle se envían los datos recogidos. Puede verse en la siguiente figura un ejemplo de uso con el interfaz:

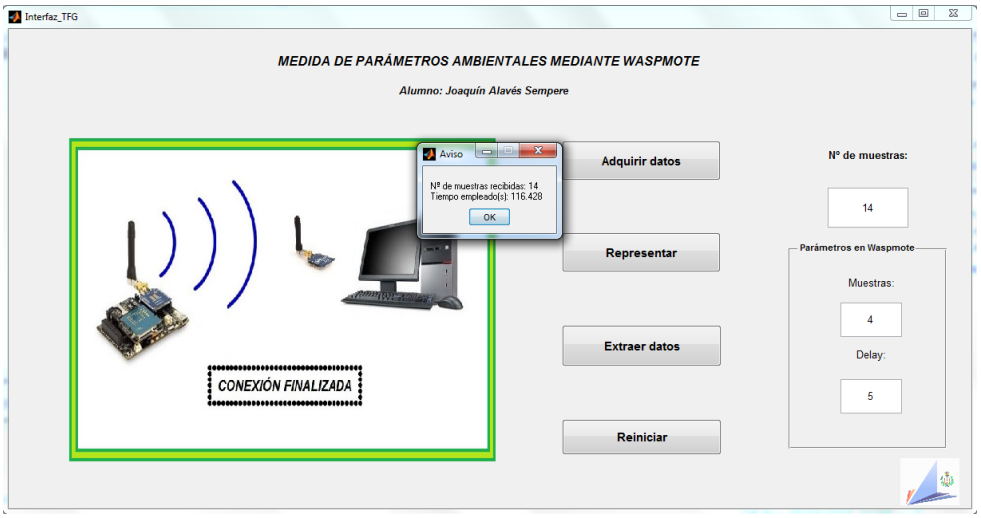

Fig. 58: Funcionamiento con interrupción *Deep Sleep*.

Como se puede comprobar el funcionamiento es correcto.

6.2.3. Modo *Hibernate*

En este modo hay que considerar que, según la teoría, la hibernación hace que se detenga el programa principal, el microcontrolador y todos los módulos de Waspmote, quedando completamente desconectados, y para volver a activar el dispositivo la única manera es mediante una alarma previamente programada en el RTC, que se alimenta a través de una batería auxiliar de la que consume 0,7µA. Esto significa que si se utiliza este modo en un script, las alarmas RTC sólo se podrán utilizar para establecer el despertar del modo de hibernación. Cuando el switch de hibernación esté abierto, cualquier alarma RTC entrante mientras se ejecuta el código podría causar colisiones internas, por lo que la alarma RTC se supone que ocurrirá cuando el Waspmote está hibernando.

Esta situación implica que todas las variables que se almacenen se borrarán de una ejecución a otra del loop y, por ello, la utilización de este modo comporta una forma distinta de medir, siendo la más evidente la de realizar una única fase de medidas y aprovechar las características de bajo consumo del modo. Para la activación de dicho modo se deben tener en cuenta los siguientes pasos:

1. Conectar la batería y encender el botón.
2. Encender el Waspmote.
3. Esperar a que el led rojo parpadee (esto se ejecuta dentro de la función `ifHibernate`).
4. Cuando se apague, activar el interruptor hibernate (Fig. 59).

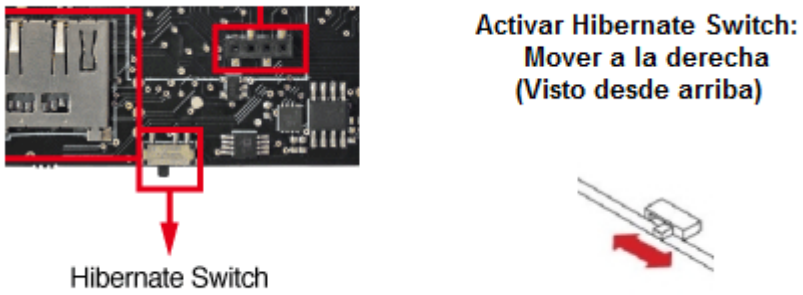

Fig. 59: Activar modo hibernación.

Tomando como ejemplo el caso del punto 6.1 la ejecución duraba aproximadamente 39 s. En este ejemplo se han puesto a modo de prueba 10 s de hibernación:

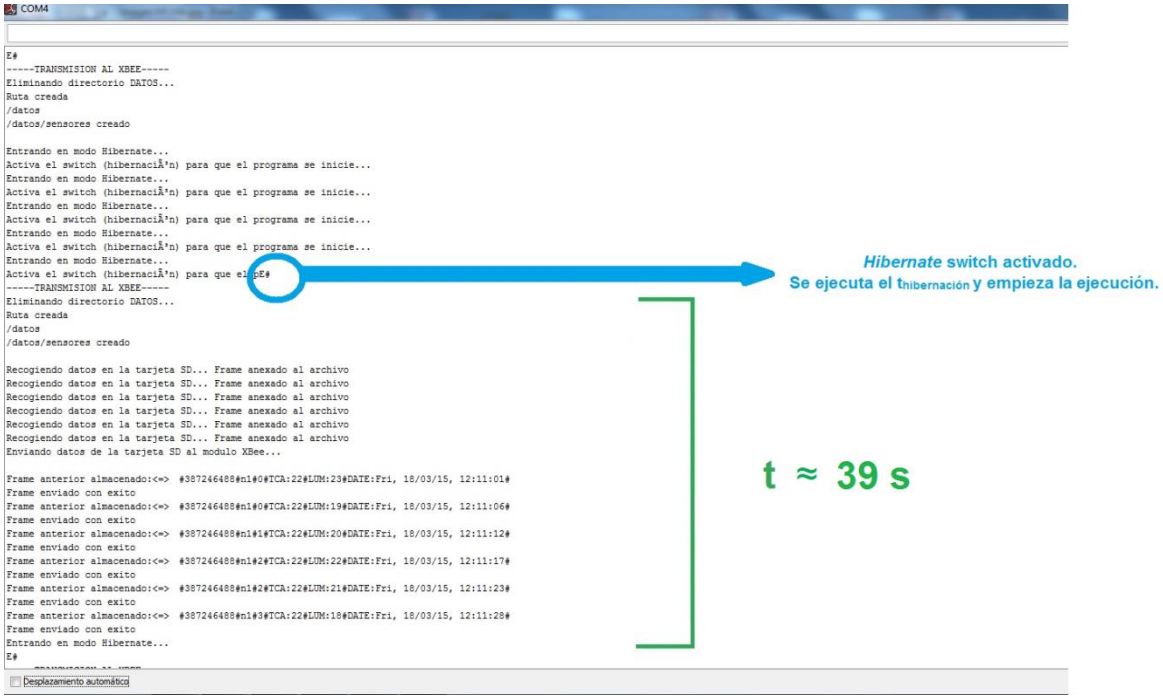

Fig. 60: Ejecución con interrupción *Hibernate*

Se han obtenido los siguientes valores en las mediciones para los siguientes tiempos de hibernación:

$t_{hibernación}$ (s)	t_1 (s)	t_2 (s)	$(t_1 - t_2) - t_{hibernación}$ (s)
5	44.24	38.52	0.72
10	49.46	38.56	0.9
15	53.65	37.97	0.68

Tabla 11: Comparativa de tiempos (hibernación).

, donde t_1 es el tiempo de ejecución con hibernación y t_2 el tiempo de ejecución sin hibernación. La cuarta columna es un tiempo que se pierde en la ejecución y que proviene del accionamiento manual del switch para activar la hibernación. Por ello, se va a incluir una opción en la interfaz que permita introducir un valor de interrupción si es necesario, además de añadir dicho valor a la ecuación de estimación de *Timeout*.

6.3. Determinación de la distancia máxima de recepción de señal

La distancia a la que se puede enviar datos según la documentación de Waspmote es de unos 7 km. En este punto se va a realizar una prueba de transmisión para comprobar si el programa funciona en este tipo de situaciones. Cabe destacar que no es necesario probar a 7 km exactos, ya que no es algo relevante para este proyecto. Una distancia interesante podría ser 1 km más o menos, y se ha tomado como lugar de pruebas la Universidad de Alicante:

Fig. 61: Lugar de transmisión (izquierda) y de recepción (derecha).

Ahora se mide la distancia entre ambos puntos:

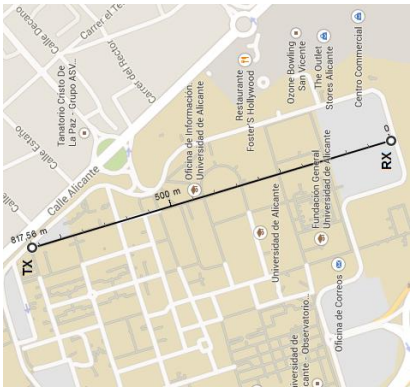

Fig. 62: Distancia entre TX- RX

Como se ve en la imagen, la distancia entre estos dos puntos es 818 m aproximadamente. Ahora ejecutamos la interfaz y pedimos 6 muestras:

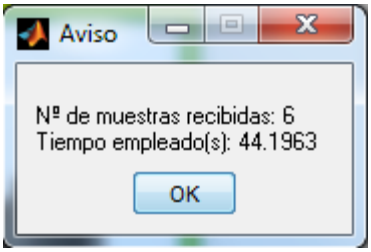

Fig. 63: Muestras recibidas

Las muestras se han recibido correctamente. A efectos prácticos, si se quisiera realizar una instalación que abarcase toda la Universidad, sólo se precisarían unos 4 o 5 dispositivos para ello, lo cual da una idea de la eficacia del programa no sólo a nivel técnico sino también económico.

7. Problemas resueltos

El principal escollo a resolver al realizar las pruebas es el hecho de que los parámetros en Waspmote pueden variar, y con ello, se han de cambiar también en el interfaz. Por ello se ha incluido un panel con tres cuadros en los que se podrá meter los valores de las muestras, el retardo y el tiempo de hibernación empleados en Waspmote, pudiendo introducirlos directamente en el interfaz y agilizando el proceso sin tener que ir al código cada vez a cambiar los números:

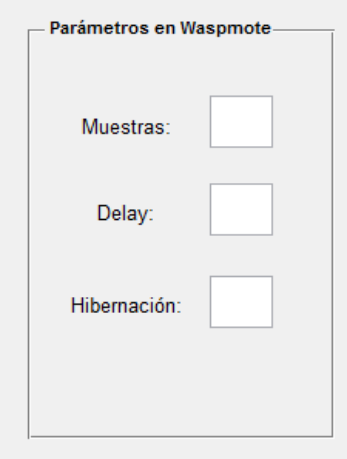

Fig. 64: Cuadro de parámetros en Waspmote.

Otra situación incómoda para el usuario puede ser el hecho de que las gráficas creadas mediante el botón “Representar” salgan en otra figura independiente. Por ello se ha implementado la inclusión de la gráfica en el elemento `axes` del interfaz. En la figura 65 se ve un ejemplo para temperatura y luz:

Fig. 65: Interfaz con gráfica incorporada.

Por último se ha considerado el incluir dentro de la adquisición de datos un sistema de aviso a fin de alertar al usuario ante cualquier situación de carácter anómalo, como por ejemplo picos de temperatura indeseados:

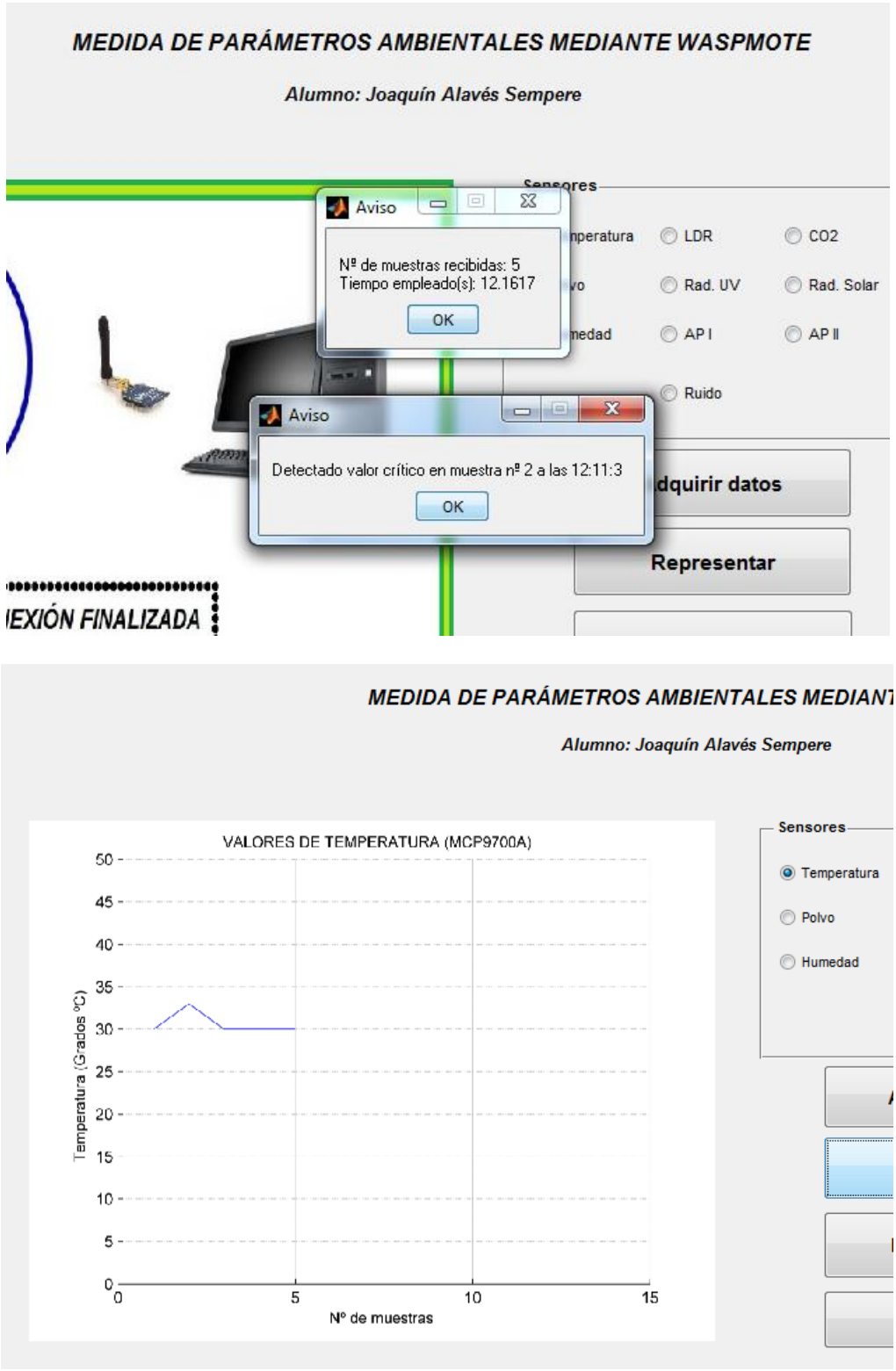

Fig. 66: Detección de valores críticos.

8. Trabajos futuros

Un factor sobre el que trabajar es el hecho de que el programa no puede detenerse a la hora de adquirir datos, debido fundamentalmente a que la comunicación es unidireccional. Se intentó incluir un botón para detenerlo desde el propio interfaz pero la estructura de tipo `try/catch` empleada no lo permite, ya que se ejecuta de forma indefinida hasta que el *Timeout* asignado se termina. De cara a futuras ampliaciones, se puede añadir un nodo que actúe de enlace entre el módulo y la placa, cosa que en principio no era objeto de este proyecto, consiguiendo así la bidireccionalidad que se precisa.

Una utilidad también interesante sería la de realizar una aplicación para dispositivos móviles (Android, iOS…) y con ello poder acceder en cualquier momento a los datos en tiempo real, creando alarmas para avisar de datos incoherentes o especialmente significativos, y pudiendo modificar parámetros desde el propio teléfono. Esta idea es especialmente interesante si se contempla un sistema integrado por varios nodos que cubra un área determinada y de la que se precisen datos de forma continua, como por ejemplo un barrio de una ciudad con altos índices de contaminación, una zona de difícil acceso, medidas en alta mar durante períodos largos, etc.:

Fig. 67: Nodos distribuidos por una ciudad.

La idea sobre la que se ha trabajado en este proyecto se ha centrado en la temperatura y en la luminosidad, pero en el programa se ha tenido en cuenta la futura inclusión de sensores en el mismo. Para ello se ha dejado programado en el interfaz la estructura base para integrar nuevos sensores y aumentar las prestaciones, y se ha colocado a tal efecto una lista de botones para cuando sea necesario su uso:

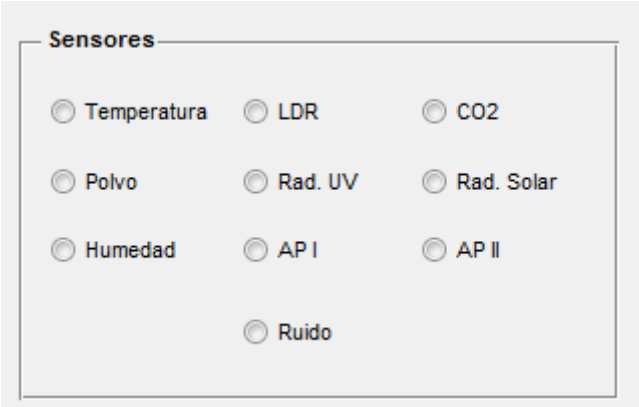

Fig. 68: Cuadro de sensores.

Una de las ventajas que otorga este panel es que al hacer clic sobre cada botón se podrá visualizar la gráfica con los datos correspondientes a cada sensor al apretar el botón de representación, como se ha visto en la figura 65. También se avisará al usuario si los sensores no están disponibles mediante un mensaje de aviso:

Fig. 69: Mensaje de aviso (ausencia se sensor).

La elección de estos sensores se ha tomado sin seguir un criterio específico, sólo se han elegido una serie de ellos de entre los disponibles para Waspmote, en este caso:

- CO2
- Polvo (partículas en suspensión)
- Radiación Ultravioleta
- Radiación Solar
- Humedad
- AP I (Contaminantes del aire I): C_4H_{10} (Butano), CH_3CH_2OH (Etanol), H_2, CO, CH_4 (Metano).
- AP II (Contaminantes del aire II): $C_6H_5CH_3$ (Tolueno), H_2S, CH_3CH_2OH, NH_3 (Amoníaco), H_2.
- Ruido

Podría darse la circunstancia de que se necesite medir un parámetro mediante un sensor que no esté en el catálogo. Es posible contactar con Waspmote y preguntar si es viable integrar este nuevo sensor para nuestro programa, lo cual abre el campo de aplicaciones de forma sustancial.

9. Conclusiones

Una vez terminado el proyecto, es necesario revisar si se han cumplido los objetivos fijados al principio del mismo. En este sentido se ha realizado un estudio investigando los distintos estándares de comunicación inalámbrica que se podrían utilizar en el proyecto en función de diversos parámetros como la potencia de transmisión, el rango de trabajo, etc., y la justificación de la opción escogida. Del mismo modo se ha recopilado información sobre las distintas plataformas sobre las que desarrollar el trabajo incluyendo aquellos elementos que pudiesen formar parte del sistema, siendo finalmente Waspmote la elegida.

Se ha llevado a cabo la realización de un programa que permita un comportamiento totalmente autónomo del sistema mediante el IDE de Waspmote para el envío de datos, en el que se ha primado ante todo la eficiencia en el consumo a través de interrupciones, en concreto la hibernación, y se ha programado pensando siempre en la posibilidad de que se puedan incorporar nuevos parámetros que puedan ser monitorizados, y que puedan ser incluidos a posteriori sin necesidad de alterar el diseño básico del programa. Los datos medidos, además, se guardan en la tarjeta SD antes de ser enviados.

Empleando MATLAB se ha diseñado una interfaz de apariencia ordenada e intuitiva que permita al usuario establecer una comunicación entre la placa y el módulo USB para el envío de información, ver las representaciones de los datos recibidos, así como extraer la información en otros formatos para poder visualizarlos en distintas plataformas. Se han habilitado también opciones que otorgan flexibilidad de cara a la incorporación de más sensores en el futuro con sólo apretar un botón.

A nivel práctico se ha realizado un prototipo que se ha utilizado para documentar con éxito las pruebas experimentales que constan en el informe. Se ha modificado el comportamiento del programa sobre la marcha en virtud de los resultados y problemas que iban apareciendo, discutiendo sobre cómo resolverlos en cada caso.

Por último, se ha planteado una serie de propuestas enfocadas a futuros trabajos que podrían desempeñarse utilizando como base este proyecto, y que no formaban parte de los objetivos de éste en el inicio.

Con todo lo expuesto, se puede considerar que se han cumplido todos los objetivos, siendo el resultado un dispositivo autónomo, eficiente energéticamente y que es capaz de medir los parámetros asignados previamente de forma precisa, que era lo que se pretendía.

10. Bibliografía

[1] Vicent Gallart, Santiago Felici-Castell, Manuel Delamo , Andrew Foster. "*Evaluation of a Real, Low Cost, Urban WSN Deployment for Accurate Environmental Monitoring*", Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference.

[2] Proyecto Smart Citizen: https://goteo.org/project/smart-citizen-sensores-ciudadanos (12/Septiembre/ 2014)

[3] *Datasheet* del Atmega1281: https://www.google.es/?gws_rd=ssl#q=Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet

(15/Septiembre/2014)

Documentación Libelium:

[4] Documento de características de los distintos módulos de Waspmote: http://www.libelium.com/downloads/documentation/waspmote_technical_guide.pdf

[5] Manual de usuario del IDE de Waspmote: http://www.libelium.com/downloads/documentation/waspmote_ide_user_guide.pdf

[6] Manual de uso de frames: http://www.libelium.com/uploads/2013/02/data_frame_guide.pdf

Manual de sensores:

http://www.libelium.com/v11-files/documentation/waspmote/waspmote-utilities-programming_guide.pdf

Manual de la tarjeta SD:

http://www.libelium.com/v11-files/documentation/waspmote/waspmote-sdcard-programming_guide.pdf

Manual del módulo XBee:

http://www.libelium.com/v11-files/documentation/waspmote/waspmote-802.15.4-networking_guide.pdf

Comunicación inalámbrica: http://redes-moviles-e-inalambricas.wikispaces.com/GSM

(7/Octubre/2014)

Tabla código ASCII: http://userscontent2.emaze.com/images/efdb5eda-4797-4da6-a0ff-db43f1bb4ed0/6920ff79-e383-4a25-aa74-50b00501c796.png

(9/Enero/2014)

Información Waspmote: https://www.cooking-hacks.com/forum/

(11/Diciembre/2015)

Información MATLAB: http://www.lawebdelprogramador.com/foros/Matlab/

(23/Febrero/2015)

Información de interés sobre aplicaciones y/o trabajos futuros:

Simon János, István Matijevics. "*Simulation and Implementation of Mobile Measuring Robot Navigation Algorithms in Controlled Microclimatic Environment Using WSN*", Intelligent Systems and Informatics (SISY), 2011 IEEE 9th International Symposium

Zhi-qin Liu, Xiu-feng Jiang, Hai-hao Wang, Kai Zhang. "*An On-line Monitor System on Off Gases of Vehicles with WSN's Design Based on Zigbee Technology*", 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing.

11. ANEXOS

11.1. ANEXO I: Código Waspmote

```
/*
 *  ------   ENVÍO DE DATOS AL XBEE   --------
 */


// Declaración de librerías
#include <WaspFrame.h>
#include <WaspXBee802.h>


// Formato del paquete de datos que será enviado
packetXBee* packet;


// Declaración de variables para enviar
int temperatura;
int luz;
char* tiempo;


// Carpeta y archivo en el que se guardarán los datos
char* path="/datos";
```

```
char* filename="/datos/sensores";


// Buffer para escribir datos en la SD

char toWrite[200];


// Variables para leer líneas de archivo

uint8_t frameSD[MAX_FRAME+1];

uint16_t lengthSD;

int32_t numLines;

int startLine;

int endLine;


// Variable para confirmar que el funcionamiento es correcto

uint8_t sd_answer;


// Almacena valores temporales (en ms)

unsigned long previo;

unsigned long t_ejec;


// Define un valor de timeout y de retardo (en ms)

unsigned long TIMEOUT = 30000;

unsigned long retardo = 5000;
```

```
// Tiempo de hibernación (en s)
char hib_time[12]="00:00:00:15";

void setup()
{

  // Indica si se viene de un reset normal o de uno de hibernación
  PWR.ifHibernate();

  USB.ON();
  USB.println(F("-----TRANSMISION AL XBEE-----"));

  RTC.ON();
  // Establece el tiempo [yy:mm:dd:dow:hh:mm:ss]
  RTC.setTime("18:03:15:06:12:11:00");

  // Iniciar XBee
  xbee802.ON();

  // Enciende la SD
```

```
SD.ON();


// Borra todo lo que haya en la ruta
sd_answer = SD.rmRfDir("DATOS");


if( sd_answer == true )
{
  USB.println(F("Eliminando directorio DATOS..."));
}
else
{
  USB.println(F("Error de borrado"));
}


// Crea una nueva ruta
sd_answer = SD.mkdir(path);


if( sd_answer == 1 )
{
  USB.println(F("Ruta creada"));
  USB.println(path);
}
```

```
else
{
  USB.println(F("Error en  comando mkdir"));
}


// Crea el archivo para almacenar los datos
sd_answer = SD.create(filename);

if( sd_answer == 1 )
{
  USB.println(F("/datos/sensores creado"));
}
else
{
  USB.println(F("/datos/sensores no creado"));
}

// Indica el valor de tiempo en este momento
previo = millis();

USB.println();
```

```
}

void loop()
{

  // Se ejecuta si se capta una interrupción de Hibernación
  if( intFlag & HIB_INT )
  {

    previo = millis();



    //  Se accede al bucle while durante el TIMEOUT
    while( (millis() - previo) < TIMEOUT )
    {

      USB.print(F("Recogiendo datos en la tarjeta SD... "));

    ////////////////////////////////////////
    // 1. Creación del frame de datos
    ////////////////////////////////////////
```

```
// Crea un frame nuevo (de tipo ASCII)
 frame.createFrame();

// Se guarda el valor medido en cada variable
 temperatura = Utils.readTemperature();
 luz = Utils.readLight();
 tiempo = RTC.getTime();

 // Se añaden los campos al frame
 frame.addSensor(SENSOR_TCA,temperatura);
 frame.addSensor(SENSOR_LUM,luz);
 frame.addSensor(SENSOR_DATE,tiempo);

 // Reserva memoria para el buffer
 memset(toWrite, 0x00, sizeof(toWrite) );

 Utils.hex2str( frame.buffer, toWrite, frame.length);

 ///////////////////////////////////
 // 2. Anexar datos al archivo
 ///////////////////////////////////
```

```
    sd_answer = SD.appendln(filename, toWrite);


    if( sd_answer == 1 )
    {
      USB.println(F("Frame anexado al archivo"));
    }
    else
    {
      USB.println(F("Anexion fallida"));
    }


    // Retardo entre cada medida
    delay(retardo);



      // Comprobar desbordamiento
      if( millis()<previo )  previo = millis();
    }


    USB.println(F("Enviando datos de la tarjeta SD al modulo
XBee..."));
```

```
USB.println();

////////////////////////////////////////////////
// 3. Leer información almacenada
////////////////////////////////////////////////

// Saca el número de líneas del archivo
numLines = SD.numln(filename);

startLine = numLines-1;
endLine = numLines;

for( int i=0; i<endLine ; i++ )
{
  // Lee la linea 'i' del archivo -> SD.buffer
  SD.catln( filename, i, 1);

  // Inicializar frameSD
  memset(frameSD, 0x00, sizeof(frameSD) );

  lengthSD = Utils.str2hex(SD.buffer, frameSD );
```

```
      USB.print(F("Frame anterior almacenado:"));
      for(int j=0; j<lengthSD; j++)
      {

        USB.print(frameSD[j]);

      }

    USB.println();

    /*************************************************
    * A partir de aquí, 'frameSD' y 'lengthSD' pueden
    * usarse como 'frame.buffer' y 'frame.length' para
    * enviar informacion a través del XBee
    *************************************************/

    // Creación del paquete de datos
    packet=(packetXBee*) calloc(1,sizeof(packetXBee)); // Reservar
memoria
    packet->mode=BROADCAST;
```

```
    // Se introducen los parámetros a enviar
    xbee802.setDestinationParams(packet,"000000000000FFFF",
frameSD, lengthSD);




    // Envío de datos
    xbee802.sendXBee(packet);


    // Comprobación de envío correcto
    if( !xbee802.error_TX ) USB.println(F("Frame enviado con
exito"));
    else USB.println(F("Error"));


    // Liberar memoria
    free(packet);
    packet=NULL;


    }


  // Limpia el flag de interrupción
  intFlag &= ~(HIB_INT);


  }
```

```
  // Saca por pantalla el tiempo empleado en la ejecución
  t_ejec = millis();
  USB.print(F("Tiempo:"));
  USB.print(t_ejec);
  USB.println();

  USB.println(F("Entrando en modo Hibernate..."));


  // Pone el Waspmote en modo Hibernación durante n segundos
  PWR.hibernate(hib_time,RTC_OFFSET,RTC_ALM1_MODE2);

  // Cuando la hibernación está inactiva se lee este mensaje
  USB.println(F("Activa el switch (hibernación) para que el
programa se inicie..."));

}
```

11.2. ANEXO II: Código Interfaz MATLAB

```
function varargout = Interfaz_TFG(varargin)
% INTERFAZ_TFG MATLAB code for Interfaz_TFG.fig
%      INTERFAZ_TFG, by itself, creates a new INTERFAZ_TFG or
raises the existing singleton*.
%
%      H = INTERFAZ_TFG returns the handle to a new INTERFAZ_TFG
or the handle to the existing singleton*.
%
%      INTERFAZ_TFG('CALLBACK',hObject,eventData,handles,...)
calls the local function named CALLBACK in INTERFAZ_TFG.M with the
given input arguments.
%
%      INTERFAZ_TFG('Property','Value',...) creates a new
INTERFAZ_TFG or raises the existing singleton*. Starting from the
left, property value pairs are applied to the GUI before
Interfaz_TFG_OpeningFcn gets called. An unrecognized property name
or invalid value makes property application stop. All inputs are
passed to Interfaz_TFG_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES


% Edit the above text to modify the response to help Interfaz_TFG


% Last Modified by GUIDE v2.5 23-Apr-2015 17:39:03


% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Interfaz_TFG_OpeningFcn, ...
                   'gui_OutputFcn',  @Interfaz_TFG_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Interfaz_TFG is made visible.
function Interfaz_TFG_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Interfaz_TFG (see VARARGIN)

%--------------------------------------------%
%          INICIAR LOGOS E IMÁGENES          %
%--------------------------------------------%

axes(handles.axes2)
handles.imagen = imread('logo_eps','jpg');
imagesc(handles.imagen)
axis off;

axes(handles.axes3)
handles.imagen = imread('imagen_inicial','jpg');
imagesc(handles.imagen)
axis off;


% Inicializar variables
handles.muestras = 0;
handles.muestras_Waspmote = 0;
handles.delay = 0;
handles.hiber = 0;

% Choose default command line output for Interfaz_TFG
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Interfaz_TFG wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Interfaz_TFG_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


    %--------------------------------------------------------%
    %                    BOTÓN REPRESENTAR                   %
    %--------------------------------------------------------%


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Muestra un mensaje de error si no hay muestras.
if (handles.muestras == 0)

    errordlg('No hay muestras que representar. Introduzca primero
las muestras y pulse "Adquirir datos"','Error');

else

% ---------------------------------------------------------------

% Cambia la imagen principal.
cla(handles.axes3)

axes(handles.axes3)
handles.imagen = imread('imagen_inicial','jpg');
imagesc(handles.imagen)
axis off;

% ---------------------------------------------------------------

% Representar temperatura y luz.

Temp = handles.sensor_temp;
Lum = handles.sensor_luz;
[S,L] = size(Temp);

figure;
y = 1:1:S;
hold on
```

```
subplot(1,2,1);
plot(y,Temp);
grid on
xlim([0 S+10])
ylim([0 50])
title('VALORES DE TEMPERATURA (MCP9700A)');
xlabel('N° de muestras');
ylabel('Temperatura (Grados °C)');

subplot(1,2,2);
plot(y,Lum,'r');
grid on
xlim([0 S+10])
ylim([0 100])
title('VALORES DE LUMINOSIDAD (LDR)');
xlabel('N° de muestras');
ylabel('Luminosidad (%)');


end

    %--------------------------------------------------------%
    %              CUADRO INTRODUCIR MUESTRAS                %
    %--------------------------------------------------------%

function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of
edit2 as a double

muestras = str2double(get(hObject, 'String'));

% Muestra un mensaje de error si el caracter introducido es
inválido.
if isnan(muestras)
    set(hObject, 'String', 0);
    errordlg('Carácter inválido. Introduzca un número','Error');
end

% Guarda el nuevo valor de las muestras
handles.muestras = muestras;
guidata(hObject,handles)
```

```
% --- Executes during object creation, after setting all
properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


    %-------------------------------------------------------------%
    %                     BOTÓN REINICIAR                         %
    %-------------------------------------------------------------%


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Cambia la imagen principal.
cla(handles.axes3)

axes(handles.axes3)
handles.imagen = imread('imagen_inicial','jpg');
imagesc(handles.imagen)
axis off;

% Si el valor es distinto de cero se pone a cero
if(handles.muestras ~= 0 )

    handles.muestras = 0;
    guidata(hObject,handles)

elseif(handles.handles.muestras_Waspmote  ~= 0 )

    handles.muestras_Waspmote = 0;
    guidata(hObject,handles)

elseif(handles.delay ~= 0)
```

```
    handles.delay = 0;
    guidata(hObject,handles)

elseif(handles.hiber ~= 0)

    handles.hiber = 0;
    guidata(hObject,handles)
end

% Deja los cuadros en blanco
limpiar = ' ';
set(handles.edit2,'string',limpiar);
set(handles.edit3,'string',limpiar);
set(handles.edit4,'string',limpiar);
set(handles.edit5,'string',limpiar);


    %--------------------------------------------------------%
    %                    BOTÓN ADQUIRIR                      %
    %--------------------------------------------------------%


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)



% Muestra un mensaje de error si no hay muestras.
if (handles.muestras == 0 || handles.muestras_Waspmote == 0)

    errordlg('Introduzca primero las muestras y pulse "Adquirir
datos"','Error');

elseif(handles.delay == 0)

    errordlg('Introduzca el delay y pulse "Adquirir
datos"','Error');

elseif(handles.hiber == 0)

    errordlg('Introduzca el tiempo de hibernación y pulse
"Adquirir datos"','Error');

else

    clc;
```

```
    % Inicia el reloj para medir el tiempo de ejecución.
    tic

    % --------------------------------------------------------------

    cla(handles.axes3)
    axes(handles.axes3)
    set(handles.axes3,'visible','off');


    N_muestras = handles.muestras;

    % ------------------------------------------------------------ %
    % ***************** ESTIMAR VALOR DE TIMEOUT ***************** %
    % ------------------------------------------------------------ %

    % Parámetros en Waspmote
    M_Wasp = handles.muestras_Waspmote;
    delay_Wasp = handles.delay;
    t_hib = handles.hiber;

    % Se calcula el Timeout a partir de las muestras recibidas,
valores de control y el delay
    % introducido en el código del Waspmote.
    t_BCD = 5.01;
    t_EnvioM = 0.58 * M_Wasp;
    valor_Timeout1 = 2 * (t_BCD +(M_Wasp * delay_Wasp) + t_EnvioM
+ t_hib);


    if(N_muestras <= M_Wasp)

        valor_Timeout = valor_Timeout1;

    else

        veces = ceil( N_muestras/M_Wasp);
        str = sprintf('%d',veces);
        disp(str);

        t_EnvioM = 0.58 * N_muestras;
        valor_Timeout = veces * (t_BCD +(N_muestras * delay_Wasp)
+ t_EnvioM + t_hib);

    end


    % Selecciona el puerto sobre el que se realizará la
comunicación.
```

```
    delete(instrfind({'Port'},{'COM___'}));
    pserial=serial('COM___','BaudRate',115200);

    % Reserva espacio para el buffer de entrada (en bytes). Por
defecto es 512
    % bytes.

    pserial.InputBufferSize = N_muestras * 80;
    get(pserial,{'InputBufferSize','BytesAvailable'})

    % ------------------------------------------------------- %
    %  ******************  METER .GIF  ********************** %
    % ------------------------------------------------------- %

    %  Botón para pedir comunicación
    h = uicontrol('Position',[700 228 200 50],'String','Establecer
comunicación',...
                  'Callback','uiresume(gcbf)');
    je = javax.swing.JLabel('<html><img
src="file:/C:\Users\joaquin\Desktop\TFG_JAS\Interfaz
TFG\Com_Wasp_XBee2.gif"/></html>');
    [hj, hc] =  javacomponent(je,[76 61 539 408], gcf);
    % hj.requestFocus;
    set(hc, 'Units','norm');
    disp('Pulse "Establecer comunicación" para abrir el puerto
serie');



    %  Espera a que se pulse el botón para abrir la comunicación
    uiwait(gcf);
    disp('Puerto serie abierto');

    % ------------------------------------------------------------

    %  Abre el puerto serie
    set(pserial,'Timeout',valor_Timeout);
    fopen(pserial);



    try

        % Vuelve inactivos los botones y los cuadros
         set(handles.pushbutton3,'Enable','off')

set(handles.pushbutton3,'backgroundcolor',get(Interfaz_TFG,'color'
))
         set(handles.pushbutton4,'Enable','off')
```

```
set(handles.pushbutton4,'backgroundcolor',get(Interfaz_TFG,'color'
))
         set(handles.pushbutton6,'Enable','off')

set(handles.pushbutton6,'backgroundcolor',get(Interfaz_TFG,'color'
))

% --------------------------------------------------------

%         % Mete una barra de progreso
%
%         barra_prog =
com.mathworks.mlwidgets.dialog.ProgressBarDialog.createProgressBar
('Recogiendo muestras...', []);
%         barra_prog.setValue(0.75);                          %
default = 0
%         barra_prog.setProgressStatusLabel('Procesando...');  %
default = 'Please Wait'
%         barra_prog.setSpinnerVisible(true);                 %
default = true
%         barra_prog.setCircularProgressBar(true);           %
default = false  (true means an indeterminate (looping) progress
bar)
%         barra_prog.setCancelButtonVisible(false);           %
default = true
%         barra_prog.setVisible(true);                        %
default = false
%         % set(hc, 'pos', [50 45 351 301])

% --------------------------------------------------------

        % Lee y almacena en "valor" lo que le llega del puerto
serie.
        valor = fread(pserial);

        % Convierte a caracteres los valores enviados. Para saber
si transmision
        % correcta
        entrada = char(valor);

        [N,M] = size(valor);


% ------------------------------------------------------------ %
%  *********** IDENTIFICAR VALORES DE TEMPERATURA *********** %
% ------------------------------------------------------------ %

        aux = 1;
```

```
        for k = 1:N

            if(valor(k,1) == 67 && valor(k+1,1) == 65 &&
valor(k+2,1) == 58 && valor(k+6,1) == 35)

                temp_1(aux,1) = entrada(k+5,1);
                temp_2(aux,1) = entrada(k+4,1);
                temp_3(aux,1) = entrada(k+3,1);
                aux = aux + 1;

            elseif(valor(k,1) == 67 && valor(k+1,1) == 65 &&
valor(k+2,1) == 58 && valor(k+5,1) == 35)

                temp_1(aux,1) = entrada(k+4,1);
                temp_2(aux,1) = entrada(k+3,1);
                temp_3(aux,1) = '0';
                aux = aux + 1;

            end
        end



        % Unir los valores en una sola variable
        temp_final = strcat(temp_3,temp_2,temp_1);
        sensor_temp = str2num(temp_final);

        % Guardar valor para poder usarlo en otra funcion.
        handles.sensor_temp = sensor_temp;
        guidata(hObject, handles);


% ------------------------------------------------------------ %
%  ********** IDENTIFICAR VALORES DE LUMINOSIDAD ********** %
% ------------------------------------------------------------ %

        aux = 1;

        for k = 1:N

            if(valor(k,1) == 77 && valor(k+1,1) == 58 &&
valor(k+5,1) == 35)

                luz_1(aux,1) = entrada(k+4,1);
                luz_2(aux,1) = entrada(k+3,1);
                luz_3(aux,1) = entrada(k+2,1);
                aux = aux + 1;

            elseif(valor(k,1) == 77 && valor(k+1,1) == 58 &&
valor(k+4,1) == 35)
```

```
                luz_1(aux,1) = entrada(k+3,1);
                luz_2(aux,1) = entrada(k+2,1);
                luz_3(aux,1) = '0';
                aux = aux + 1;

            elseif(valor(k,1) == 77 && valor(k+1,1) == 58 &&
valor(k+3,1) == 35)

                luz_1(aux,1) = entrada(k+2,1);
                luz_2(aux,1) = '0';
                luz_3(aux,1) = '0';
                aux = aux + 1;

            end
        end


        luz_final = strcat(luz_3,luz_2,luz_1);
        sensor_luz = str2num(luz_final);


        handles.sensor_luz = sensor_luz;
        guidata(hObject, handles);

% ------------------------------------------------------------ %
%  *********** IDENTIFICAR VALORES FECHA Y HORA *********** %
% ------------------------------------------------------------ %

        aux = 1;



        for k = 1:N

% ---------------------------- DIA -----------------------------
            if(valor(k,1) == 44 && valor(k+1,1) == 32 &&
valor(k+4,1) == 47)

                dia_1(aux,1) = entrada(k+2,1);
                dia_2(aux,1) = entrada(k+3,1);
                aux = aux + 1;
            end

% ---------------------------- MES -----------------------------
            if(valor(k,1) == 47 && valor(k+3,1) == 47)

                mes_1(aux,1) = entrada(k+1,1);
                mes_2(aux,1) = entrada(k+2,1);
                aux = aux + 1;
```

```
            end

% ----------------------------- ANYO -------------------------------
            if(valor(k,1) == 47 && valor(k+3,1) == 44)

                anyo_1(aux,1) = entrada(k+1,1);
                anyo_2(aux,1) = entrada(k+2,1);
                aux = aux + 1;

            end

% ---------------------------- HORAS -----------------------------
            if(valor(k,1) == 32 && valor(k+3,1) == 58)

                 hora_1(aux,1) = entrada(k+1,1);
                 hora_2(aux,1) = entrada(k+2,1);
                 aux = aux + 1;
            end

% --------------------------- MINUTOS ---------------------------
            if(valor(k,1) == 58 && valor(k+3,1) == 58)

                 min_1(aux,1) = entrada(k+1,1);
                 min_2(aux,1) = entrada(k+2,1);
                 aux = aux + 1;
            end

% ------------------------- SEGUNDOS -------------------------
            if(valor(k,1) == 58 && valor(k+3,1) == 58 &&
valor(k+6,1) == 35)

                seg_1(aux,1) = entrada(k+4,1);
                seg_2(aux,1) = entrada(k+5,1);
                aux = aux + 1;

            end
        end



        % Unir los valores en una sola variable

        dia = strcat(dia_1,dia_2);
        dia_final = str2num(dia);

        handles.dia = dia_final;
        guidata(hObject, handles)

        mes = strcat(mes_1,mes_2);
        mes_final = str2num(mes);
```

```
        handles.mes = mes_final;
        guidata(hObject, handles)

        anyo = strcat(anyo_1,anyo_2);
        anyo_final = str2num(anyo);

        handles.anyo = anyo_final;
        guidata(hObject, handles)

        hora = strcat(hora_1,hora_2);
        hora_final = str2num(hora);

        handles.hora = hora_final;
        guidata(hObject, handles)

        min = strcat(min_1,min_2);
        min_final = str2num(min);

        handles.min = min_final;
        guidata(hObject, handles)

        seg = strcat(seg_1,seg_2);
        seg_final = str2num(seg);

        handles.seg = seg_final;
        guidata(hObject, handles)

% ------------------------------------------------------------------

        [W,H] = size(sensor_temp);

        % Se guarda el tiempo transcurrido en la ejecución del
programa
        handles.t_ejecucion = toc;
        guidata(hObject, handles)

        % Cerrar GIF, eliminar botón "Establecer comunicación" y
barra de progreso
        delete(hc)
        delete(h)
%         barra_prog.setVisible(false);

        % ------------------------------------------------------

        % Limpia la imagen del eje seleccionado
        cla(handles.axes3)

        axes(handles.axes3)
        handles.imagen = imread('imagen_adquirir','jpg');
```

```
        imagesc(handles.imagen)
        axis off;

        % ------------------------------------------------------

        % Reinicia los botones inactivos
        set(handles.pushbutton3,'Enable','on')
        set(handles.pushbutton4,'Enable','on')
        set(handles.pushbutton6,'Enable','on')

        % ------------------------------------------------------

        % Mensaje que informa sobre las muestras recibidas y el
tiempo empleado
        % para ello.
        msgbox({['N° de muestras recibidas: ',num2str(W)],['Tiempo
empleado(s): ', num2str(handles.t_ejecucion)]}, 'Aviso');


% ---------------------------------------------------------- %
%  ************ DETECCIÓN DE VALORES CRÍTICOS ************  %
% ---------------------------------------------------------- %

        umbral = 45;
        pos = find(sensor_temp >= umbral);

        [R,C] = size(pos);

        %  Si se han encontrado valores críticos saca un mensaje
de aviso
        if (R ~= 0)

            for s = 1:R

                msgbox({['Detectado valor crítico en muestra n°
',num2str(pos(s,1)),' a las
',num2str(hora_final(pos(s,1),1)),':',num2str(min_final(pos(s,1),1
)),':',num2str(seg_final(pos(s,1),1))]}, 'Aviso');

            end

        end


    catch me

        display('ERROR')
        fclose(pserial);
        delete(pserial);
        clear all;
```

```
    end

end

    %--------------------------------------------------------%
    %                  BOTÓN EXTRAER DATOS                   %
    %--------------------------------------------------------%

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Muestra un mensaje de error si no hay muestras.
if (handles.muestras == 0)

    errordlg('No hay datos. Introduzca primero las muestras y
pulse "Adquirir datos"','Error');

else

% Pasar datos a un .txt con formato de tabla.

    fichero = fopen('Val_Sensores.txt','a');
    fprintf(fichero,'%8s %18s %20s
%5s\n','Fecha(dd/mm/aa)','Hora(hh:mm:ss)','Temperatura
(°C)','Luminosidad (%)');

    Temper = handles.sensor_temp;
    Lumin = handles.sensor_luz;
    Dia = handles.dia;
    Mes = handles.mes;
    Anyo = handles.anyo;
    Hora = handles.hora;
    Minut = handles.min;
    Segun = handles.seg;

[S,L] = size(Temper);

    for b = 1:S
```

```
        fprintf(fichero,'%5.0f/%2.0f/%2.0f %13.0f:%2.0f:%2.0f
%15.1f %15.0f\n', Dia(b), Mes(b), Anyo(b), Hora(b), Minut(b),
Segun(b), Temper(b), Lumin(b));

    end

fclose(fichero);

% Mensaje de confirmación de guardado de datos
msgbox('Datos guardados con éxito','Aviso');

end

 %--------------------------------------------------------%
 %           CUADRO INTRODUCIR MUESTRAS (WASPMOTE)         %
 %--------------------------------------------------------%

function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

muestras_Waspmote = str2double(get(hObject, 'String'));

% Muestra un mensaje de error si el caracter introducido es
inválido.
if isnan(muestras_Waspmote)
    set(hObject, 'String', 0);
    errordlg('Carácter inválido. Introduzca un número','Error');
end


handles.muestras_Waspmote = muestras_Waspmote;
guidata(hObject,handles)


% --- Executes during object creation, after setting all
properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


%--------------------------------------------------------%
%            CUADRO INTRODUCIR DELAY (WASPMOTE)          %
%--------------------------------------------------------%


function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

delay = str2double(get(hObject, 'String'));

% Muestra un mensaje de error si el caracter introducido es
inválido.
if isnan(delay)
    set(hObject, 'String', 0);
    errordlg('Carácter inválido. Introduzca un número','Error');
end


handles.delay = delay;
guidata(hObject,handles)


% --- Executes during object creation, after setting all
properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%--------------------------------------------------------%
%        CUADRO INTRODUCIR HIBERNACIÓN (WASPMOTE)        %
%--------------------------------------------------------%
```

```
function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

hiber = str2double(get(hObject, 'String'));

% Muestra un mensaje de error si el caracter introducido es
inválido.
if isnan(hiber)
    set(hObject, 'String', 0);
    errordlg('Carácter inválido. Introduzca un número','Error');
end


handles.hiber = hiber;
guidata(hObject,handles)


% --- Executes during object creation, after setting all
properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%-----------------------------------------------------------%
%                 RADIO BUTTON TEMPERATURA                  %
%-----------------------------------------------------------%


% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of radiobutton1


%-------------------------------------------------------------%
%                                                             %
%          RADIO BUTTON LDR (Sensor no habilitado)            %
%                                                             %
%-------------------------------------------------------------%


% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2

if (get(hObject,'Value') == get(hObject,'Max'))

    warndlg('No se ha detectado sensor de luminosidad','Error');

end
```

11.3. ANEXO III: Código inclusión de gráficas en el interfaz

```
% ------------------------------------------------------------ %
%                                                              %
%                  INCLUIR GRÁFICA EN EL INTERFAZ              %
%                                                              %
% ------------------------------------------------------------ %


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

global opcion;

% Muestra un mensaje de error si no hay muestras.
if (handles.muestras == 0)

    errordlg('No hay muestras que representar. Introduzca primero
las muestras y pulse "Adquirir datos"','Error');

else

    switch opcion

        case 1


            Temp = handles.sensor_temp;
            [S,L] = size(Temp);

            h.figure = figure;
            clf
            position = get(gcf,'Position');
            set(h.figure,'Color','w',...
            'PaperPositionMode', 'auto', ...
            'Units','in','Position',[position(1:2) 6 2.5 ],...
            'PaperPosition',[0.25 0.25 6 2.5])


            y = 1:1:S;
            hold on
            plot(y,Temp);
            grid on
            xlim([0 S+10])
```

```
            ylim([0 50])
            title('VALORES DE TEMPERATURA (MCP9700A)');
            xlabel('N° de muestras');
            ylabel('Temperatura (Grados °C)');

            % Exporta el plot a imagen .jpg
            myStyle = hgexport('factorystyle');
            myStyle.Format = 'jpeg';
            myStyle.Width = 539;
            myStyle.Height = 408;
            myStyle.Resolution = 900;
            myStyle.Units = 'pixels';
            myStyle.FixedFontSize = 1;

hgexport(h.figure,'graficaTemp.jpeg',myStyle,'Format','jpeg')


            % Cambia la imagen principal.
            cla(handles.axes3)

            axes(handles.axes3)
            handles.imagen = imread('graficaTemp','jpeg');
            im_base = imread('imagen_inicial','jpg');
            [numrows,numcols] = size(im_base);
            handles.imagen = imresize(handles.imagen, [numrows
numcols]);
            imagesc(handles.imagen)
            axis off;

        case 2

            Lum = handles.sensor_luz;
            [S,L] = size(Lum);

            h.fig = figure;
            clf
            position = get(gcf,'Position');
            set(h.fig,'Color','w',...
            'PaperPositionMode', 'auto', ...
            'Units','in','Position',[position(1:2) 6 2.5 ],...
            'PaperPosition',[0.25 0.25 6 2.5])

            y = 1:1:S;
            plot(y,Lum,'r');
            grid on
            xlim([0 S+10])
            ylim([0 100])
            title('VALORES DE LUMINOSIDAD (LDR)');
            xlabel('N° de muestras');
            ylabel('Luminosidad (%)');
```

```
            % Exporta el plot a imagen .jpg
            myStyle = hgexport('factorystyle');
            myStyle.Format = 'jpeg';
            myStyle.Width = 539;
            myStyle.Height = 408;
            myStyle.Resolution = 900;
            myStyle.Units = 'pixels';
            myStyle.FixedFontSize = 1;

hgexport(h.fig,'graficaLuz.jpeg',myStyle,'Format','jpeg')


            % Cambia la imagen principal.
            cla(handles.axes3)

            axes(handles.axes3)
            handles.imagen = imread('graficaLuz','jpeg');
            im_base = imread('imagen_inicial','jpg');
            [numrows,numcols] = size(im_base);
            handles.imagen = imresize(handles.imagen, [numrows
numcols]);
            imagesc(handles.imagen)
            axis off;

        otherwise

            errordlg('Seleccione un sensor','Error');
    end

end


%-------------------------------------------------------------%
%                 RADIO BUTTON TEMPERATURA                    %
%-------------------------------------------------------------%


% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1

global opcion;
if (get(hObject,'Value') == get(hObject,'Max'))
```

```
    opcion = 1;

end


%--------------------------------------------------------------%
%                     RADIO BUTTON LDR                         %
%--------------------------------------------------------------%


% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2


global opcion;
if (get(hObject,'Value') == get(hObject,'Max'))

    opcion = 2;

end
```