Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Gradu Amaierako Lana / Trabajo Fin de Grado
Ingenieritza Elektronikoko Gradua / Grado en Ingeniería Electrónica

Diseño y construcción de una plataforma móvil de sensores inalámbricos ambientales

Egilea/Autor/a:
Jesús María Gómez Ibáñez
Zuzendaria/Director/a:
Francisco Javier Echanobe Arias

© 2016 Jesús María Gómez Ibáñez

Leioa, 17 de agosto de 2016

Índice

Objeto y desarrollo

1. Objetivos

Este proyecto abordará el análisis de la plataforma de desarrollo Waspmote junto con el protocolo de comunicaciones inalámbricas XBee para su posterior utilización en el montaje de un robot con plataforma móvil capaz de enviar datos atmosféricos de forma inalámbrica para su posterior procesado en entorno de Matlab. Los objetivos fijados son los siguientes:

- Captación de mediadas desde la plataforma Waspmote
- Programación en Matlab de un protocolo de recepción de los datos
- Emisión y recepción inalámbricas de los datos.
- Programación de una plataforma móvil controlada por ordenador
- Control inalámbrico de la plataforma móvil.

2. Desarrollo

La estructura del trabajo está organizada del siguiente modo. El primer capítulo está dirigido a la introducción los sistemas de adquisición de datos, exponiendo su estado actual y el funcionamiento interno general.

En el segundo capítulo, se presentará la plataforma utilizada durante el desarrollo del proyecto detallando diferentes aspectos de su funcionamiento.

El tercer y cuarto capítulo están enfocados a presentar las características y los rangos de funcionamiento de los diferentes sensores y actuadores utilizados para la realización del proyecto.

La descripción del entorno de programación principal utilizado en el desarrollo del proyecto se realiza en el capítulo 5.

El desarrollo del proyecto se expone en el capítulo 6, en dicho capítulo se detalla la programación realizada para los diferentes módulos de la aplicación.

Por último, las principales conclusiones alcanzadas tras la realización del proyecto se exponen en el capítulo 6.

Capítulo 1. Preliminares

1.1. Introducción a las plataformas digitales de adquisición de datos

En la actualidad, la vertiginosa evolución de la electrónica y la microelectrónica, han desembocado en la automatización o monitorización de muchos de los procesos tanto de la vida cotidiana como del ámbito industrial.

Esto ha permitido el desarrollo de sistemas inteligentes que resuelven los más diversos problemas, son los llamados Sistemas de Adquisición de Datos.

La adquisición de datos, consiste en el muestreo de magnitudes del mundo real en forma de datos procesables por ordenador o por otros sistemas digitales. Este proceso se realiza transformando las señales eléctricas obtenidas mediante sensores en señales digitales, de esto se encarga un módulo de digitalización o tarjeta de adquisición de datos [1].

1.2. Sistemas embebidos

Un sistema embebido es un sistema de computación diseñado para realizar una o varias funciones específicas, frecuentemente en un sistema de computación en tiempo real.

Este tipo de sistemas, se ejecuta en una electrónica programable con unos recursos de hardware muy limitados, en muchos casos el sistema es alimentado por baterías por lo que en su programación debe tenerse en cuenta el consumo de energía.

1.3. Sistemas electrónicos digitales para captar medidas

Por lo general, el hardware para la adquisición de datos (DAQ) es una interface entre las señales analógicas y un ordenador (fig.1). Esta interface puede estar conectada al ordenador vía USB, puerto paralelo o a través de los slots PCI/PCI-e.

La utilización de estos sistemas en aplicaciones real time o en circuitos con señales analógicas de alta frecuencia, requiere sistemas DAQ de alta velocidad con tarjetas de procesado digital de señales (DSP) [2].

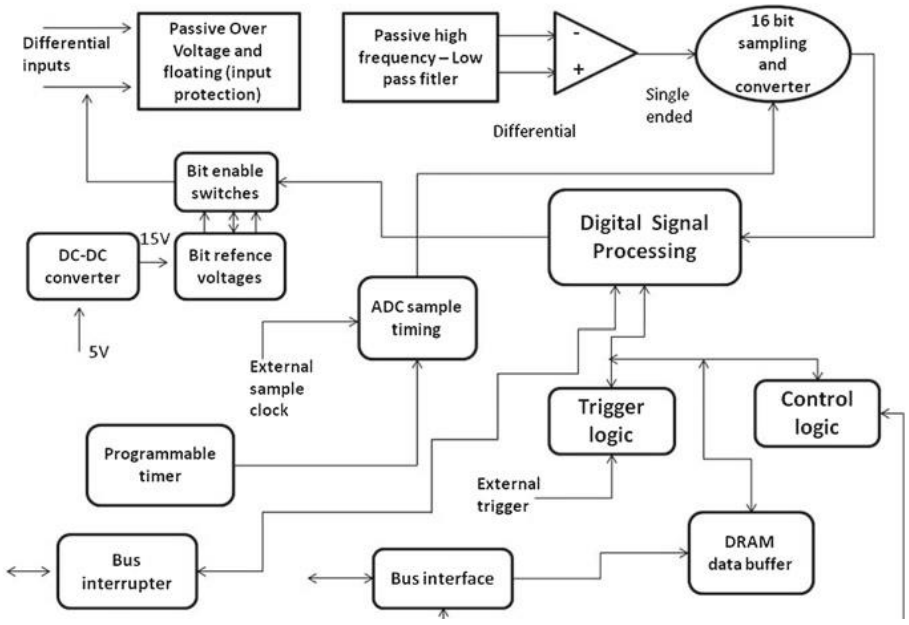


Figura 1. Esquema de funcionamiento de un DAQ

El componente principal de un DAQ es el convertidor A/D, que transforma las tensiones de las señales que se desea medir en señales digitales que pueden ser interpretadas por un ordenador. Un sistema DAQ actual se compone de los siguientes elementos:

- Multiplexor de entrada
- Amplificador de la señal de entrada
- Circuito *sample & hold*
- Convertidor A/D
- Memoria (DMA)
- Filtros
- Reloj
- Bus de datos
- Procesador digital de señales (DSP)
- Microprocesador o FPGA

Los ordenadores con canales DMA (fig.2) pueden transferir datos utilizando mucha menos CPU, esto afecta al rendimiento de los sistemas DAQ.

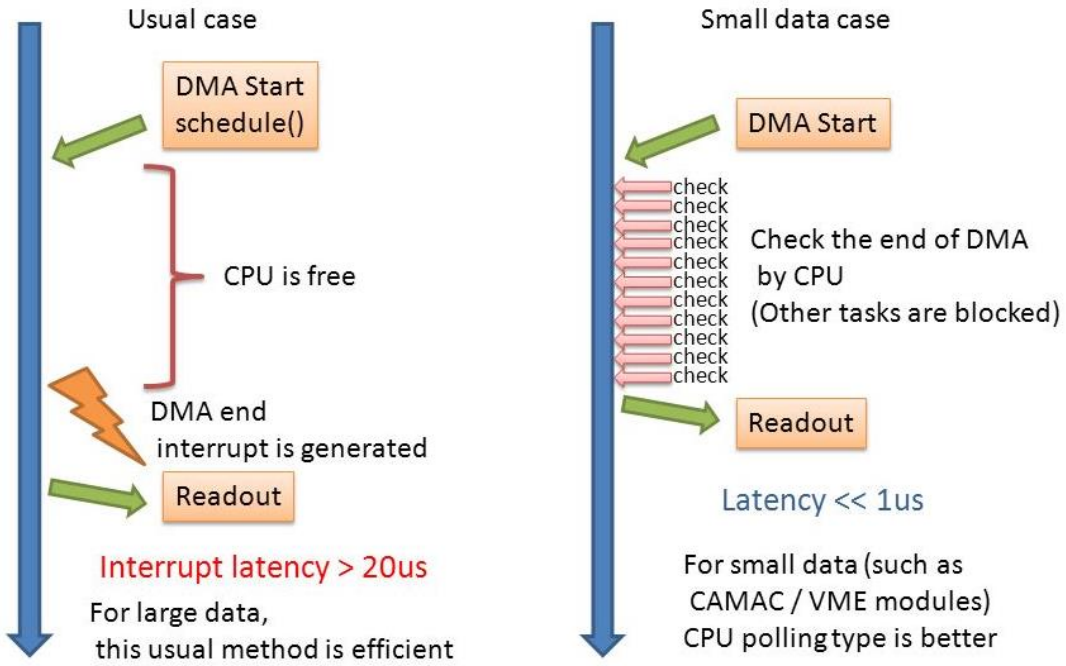


Figura 2. Transferencia DMA vs cola CPU

Capítulo 2. Descripción del Hardware

En este capítulo se describirán las características de la plataforma de desarrollo Waspmote junto con las diferentes tarjetas de expansión utilizadas en el proyecto.

2.1. Waspmote

2.1.1. Especificaciones

Waspmote se basa en una arquitectura modular [3]. Esto permite integrar únicamente los módulos necesarios en cada dispositivo siendo posible sustituir o ampliar las capacidades del sistema fácilmente según las necesidades del proyecto. El módulo base de *Waspmote* consta de las siguientes especificaciones:

- Microcontrolador: ATmega1281
- Frecuencia: 8MHz
- SRAM: 8Kb
- EEPROM: 4Kb
- FLASH: 128Kb
- SD Card: hasta 2Gb
- Peso: 20 gr
- Dimensiones: 73.5 x 51 x 13 mm
- Rango de temperatura: de -20 ºC a 65 ºC

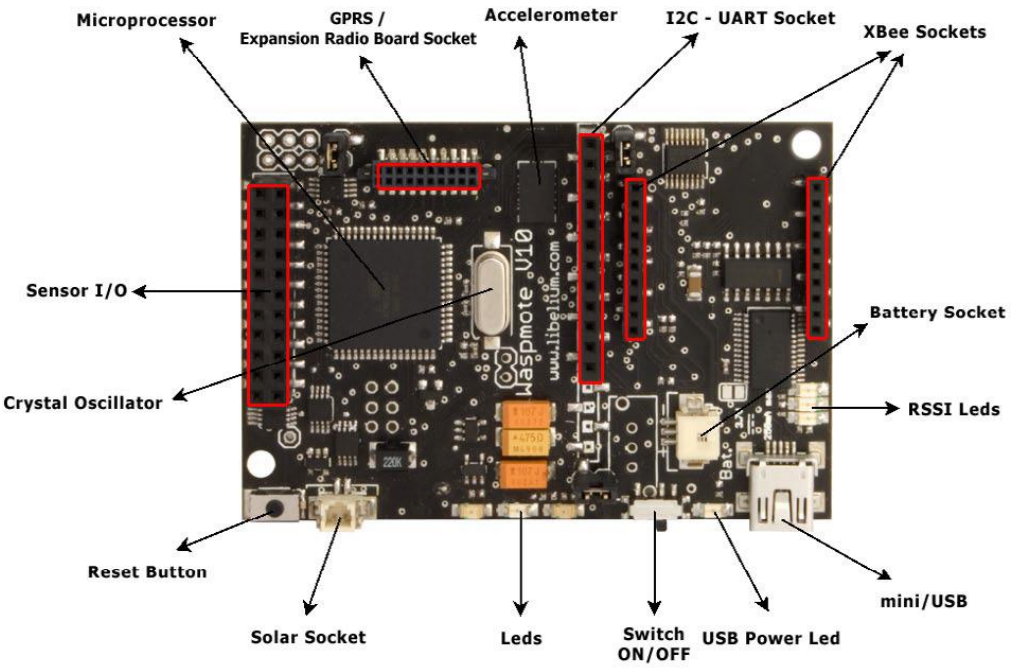


Figura 3. Tarjeta Waspmote (parte superior)

El módulo base de Waspmote (fig.3) tiene en su circuitería hasta tres osciladores para realizar instrucciones, generar alarmas y controlar los ciclos de reloj del sistema.

El reloj del sistema es un oscilador de cuarzo que trabaja a una frecuencia de 8MHz. Este oscilador es el encargado de que el microcontrolador ATmega1281 ejecute las instrucciones de bajo nivel.

El propio microcontrolador consta de un *watchdog* interno mejorado (WDT) que se encarga de contar de forma precisa los ciclos de reloj de un oscilador de 128KHz. Este dispositivo permite al microcontrolador despertar del estado de bajo consumo *sleep* o generar alarmas internas.

El siguiente elemento cronométrico de Waspmote es el RTC que informa al sistema del momento temporal en el que se encuentra. Esto, permite que se puedan programas acciones relativas en el tiempo o a intervalos absolutos.

El RTC seleccionado para la Waspmote es el DS3231SN de Maxim, con una frecuencia de 32.768 Hz (valor divisor del segundo). El DS3231SN es un oscilador a cristal con una precisión de ±2ppm. Además, en su interior consta de una circuitería que le permite realizar una compensación de las variaciones de oscilación debidas a la temperatura.

El proceso de recalibración del cristal de oscilación se realiza a través de los datos obtenidos por un sensor de temperatura interno que tiene el RTC.

El RTC es el encargado de despertar a Waspmote durante los modos de máximo ahorro energético, *Deep Sleep* e *Hibernate*. De esta forma se maximiza el nivel de la batería puesto que se obtienen unos consumos de 7µA.

Otro de los circuitos integrados en Waspmote base es el acelerómetro LIS3LV02DL de STMicroelectronics. Este circuito puede detectar cambios de dirección y caídas libres generando una interrupción diferente en cada caso en el microprocesador.

Las capacidades de Waspmote pueden verse ampliadas conectando diferentes sensores a los puertos de entrada y salida en función de la aplicación que se vaya a diseñar.

2.1.2. Conexiones I/O

Waspmote puede comunicarse con otros dispositivos externos a través de los diferentes puertos de entrada y salida que posee. Los dispositivos con los que puede intercambiar información pueden ser sensores, actuadores o componentes que cumplan las especificaciones de tensión o corriente de los puertos de Waspmote.

Además, Waspmote posee una serie de módulos propios que se conectan al módulo base de Waspmote permitiéndole tomar medidas ya sea de sensores, medidas referidas al sector agrónomo o medidas de radiación entre algunas de sus posibilidades.

El módulo base dispone de 7 entradas analógicas accesibles en el conector de sensores y 8 pines digitales configurables como entradas o salidas (fig.4).

Para las entradas analógicas el microcontrolador utiliza un conversor AD de aproximaciones sucesivas que utiliza una tensión de referencia para las entradas de 0V. El valor máximo de voltaje es de 3.3V. El conversor AD devolverá un valor comprendido entre 0 y 1023 siendo el 0 una tensión nula y 1023 la tensión máxima (3.3V).

SI se precisase un número mayor de entradas digitales el sistema puede transformar las analógicas de forma que podrían obtenerse un total de 14 conexiones digitales.

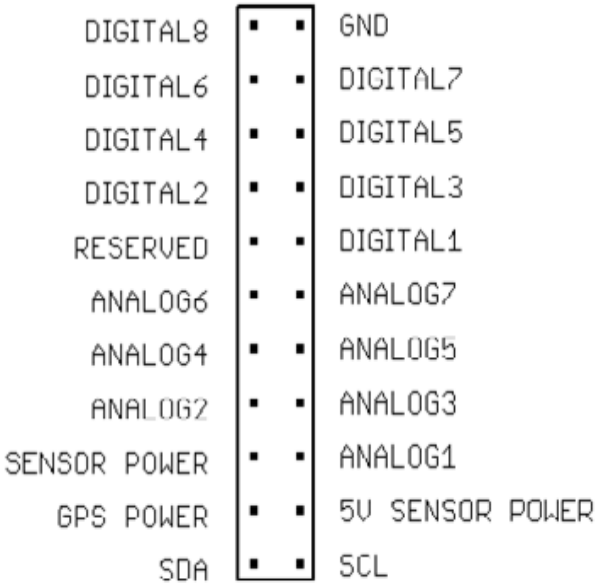


Figura 4. Pinout de la tarjeta Waspmote

2.2. Event board

2.2.1. Configuración

La *Event board* (fig.5) es una placa de eventos en la que los sensores están activos mientras Waspmote se encuentra en alguno de sus modos de bajo consumo. La placa de sensores utiliza un registro de desplazamiento, en el que indica que sensores han sido activados o han realizado una interrupción.

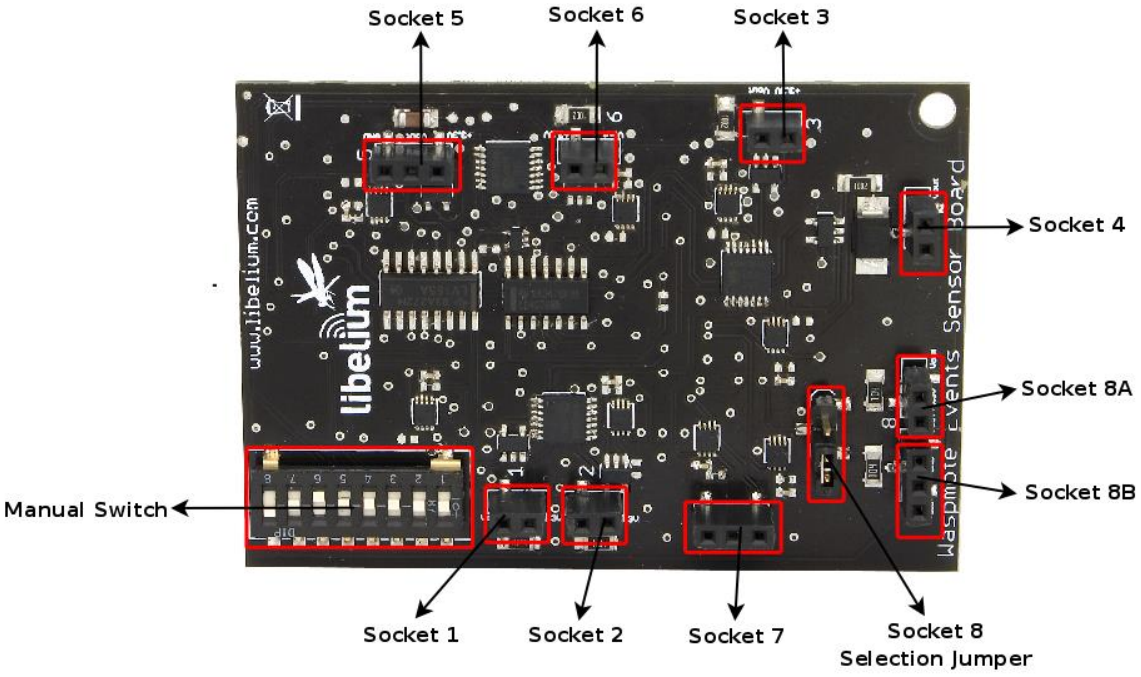


Figura 5. Event Board de Waspmote (parte superior)

2.2.2. Conexiones I/O

La *Event board* permite la conexión de hasta 8 sensores de manera simultánea. Las salidas de estos sensores, se comparan con un valor umbral y en el caso de superarlo, se combinan en una puerta lógica OR para ejecutar una interrupción en el registro de desplazamiento.

En la *Event board* además de las conexiones para los sensores, se dispone de una serie de interruptores manuales que nos permiten activar o desactivar el funcionamiento de cada uno de los sensores de manera individual.

Las tensiones de funcionamiento, como en el caso de Waspmote son de 3.3 V con una corriente máxima de continua de 200mA.

2.3. XBee 802.15.4 Networking board (Digimesh)

Waspmote tiene la posibilidad de conectar múltiples módulos de conexión inalámbrica a través de su puerto de conexión XBee. El módulo XBee 802.15.4 (fig.6) cumple con el estándar IEEE 802.15.4 que define la capa física y la capa de enlace.

La frecuencia utilizada por esta tarjeta se encuentra en la banda libre de los 2.4 GHz. Utiliza 16 canales de un ancho de banda de 5MHz cada uno.

Figura 6. Tarjeta XBee 802.15.4

Este módulo en concreto tiene la posibilidad de utilizar dos firmwares que otorgan características diferentes.

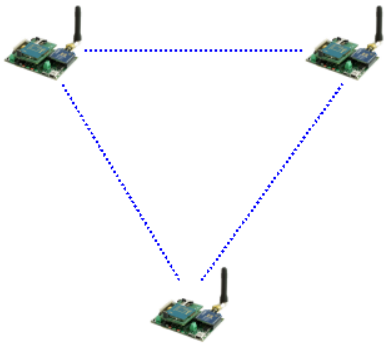


Figura 7. Configuración de red p2p

Por un lado, el firmware propio del módulo XBee 802.15.4 nos permite la configuración de redes *peer-to-peer* (fig.7) de esta forma se tienen dos tipos de dispositivos, coordinadores y dispositivos finales (end devices). En este caso los dispositivos finales se conectan a un coordinador que es el que genera la red. La red se genera fijando dos parámetros fundamentales, el PAN (Personal Area Network) y el canal. Los dispositivos finales pueden comunicarse entre sí utilizando un coordinador como intermediario [4].

Por otro lado, el firmware digimesh permite la configuración de dos topologías de red, una *peer-to-peer* como la del caso anterior en la que los dispositivos finales se interconectan a través de un coordinador, y una topología *mesh (fig.8)* que consiste en que todos los dispositivos finales pueden comunicarse entre sí de manera directa, tenemos dos modos de configuración, los routers que mediante algoritmos buscan la distancia más corta entre dispositivos, y los dispositivos finales [5].

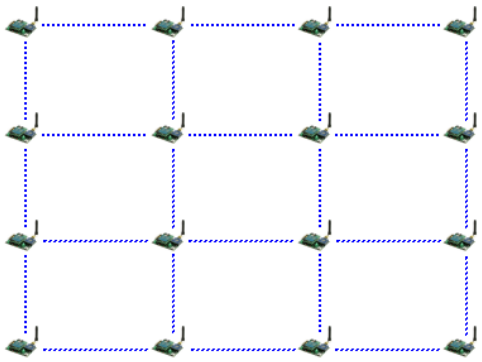


Figura 8. Configuración de red mesh

Para la configuración de las tarjetas se utiliza un Gateway (fig.9), que consiste en una interface usb a conexión XBee. Este dispositivo no solo permite la configuración de cada tarjeta, sino que además es posible transmitir y recibir tramas de todas de las otras tarjetas en la misma red.

Figura 9. Gateway XBee junto con tarjeta XBee

Capítulo 3. Sensores y actuadores

A continuación, se detallan los diferentes sensores y actuadores utilizados y sus características principales de funcionamiento.

3.1. Acelerómetro

Waspmote consta de un sensor que informa al microcontrolador de las variaciones de aceleración sufridas en los 3 ejes [6].

La integración de este sensor, permite la configuración de dos tipos diferentes de interrupciones, caída libre (free fall) y detección de cambio de dirección. Estas interrupciones permiten programar Waspmote para que ejecute diferentes rutinas en uno u otro caso.

El sensor seleccionado es un acelerómetro lineal de 3 ejes que transmite los datos a través del interfaz I2C. El acelerómetro consta de 2 rangos de medida diferentes, ±2g y ±6g. La resolución de ente circuito integrado es de 12 bits, permitiendo obtener hasta 4096 posibles valores.

El LIS3LV02DL permite configurar la tasa de refresco, esto nos permite controlar mejor el consumo de Waspmote ya que cuantas más medidas se obtengan mayor será su consumo. Por defecto el acelerómetro se actualiza 40 veces por segundo (40 Hz).

El rango de temperaturas del acelerómetro está comprendido entre los -40 ºC y los 85 ºC, además de la circuitería de medida, el integrado consta de una circuitería de test, de esta forma cuando las medidas no son validad es capaz de descartarlas.

3.2. Sensor de presión

El sensor de presión MPX4115A, es un sensor con compensación de temperatura que realiza una medida absoluta de la presión atmosférica [7].

El rango de presiones de operación se encuentra entre los 15kPa y los 115kPa, sin embargo, la presión máxima que soporta el sensor es de 400kPa.

El rango de temperaturas del MPX4115A está comprendido entre los -40 ºC y los 125 ºC, para medidas comprendidas entre los 0 ºC y los 85 ºC, el error máximo de medida es del 1,5%.

En la figura 10 puede verse representada su función de transferencia.

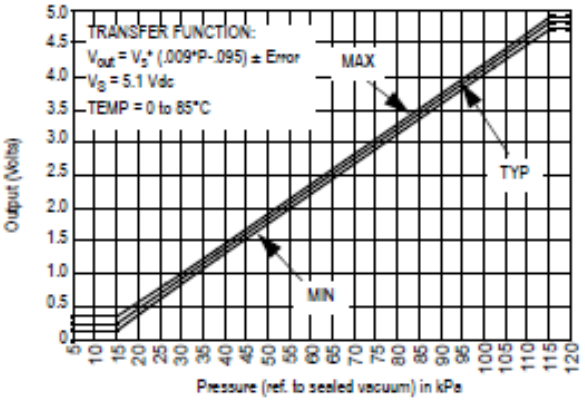


Figura 10. Gráfico de la función de transferencia del sensor de presión

3.3. Sensor de luminosidad

El sensor de luminosidad consiste en una LDR (fig.11) o fotorresistencia, la conductividad de ésta varía en función de la intensidad de luz recibida. Esta LDR tiene una resistencia en oscuridad de 20 MΩ y una resistencia en iluminación comprendida entre los 5 kΩ y los 20kΩ. [8]

El rango espectral medible se encuentra entre 400 nm y 700 nm que coincide con el espectro de la luz visible por el ojo humano.

El rango de temperaturas para la obtención de valores validos de medida está entre los -30 ºC y los 75 ºC.

Figura 11. Sensor LDR

3.4. Sensor de humedad

El sensor de humedad elegido es el 808H5V5 cuya repuesta es lineal con respecto a la humedad relativa [9].

El rango de medida se sitúa entre el 0 y el 100% de la humedad relativa (RH) con una precisión ≤ ±4% RH para medidas comprendidas entre el 30% ~ 80% RH (a 25 ºC) y ≤ ±6% RH para medidas comprendidas entre el 0% ~ 100%.

El sensor posee una alta estabilidad (<1% RH/año) y su tiempo de respuesta es < 15s.

El rango de temperaturas del sensor 808H5V5 se sitúa entre los -40 ºC y los 85 ºC.

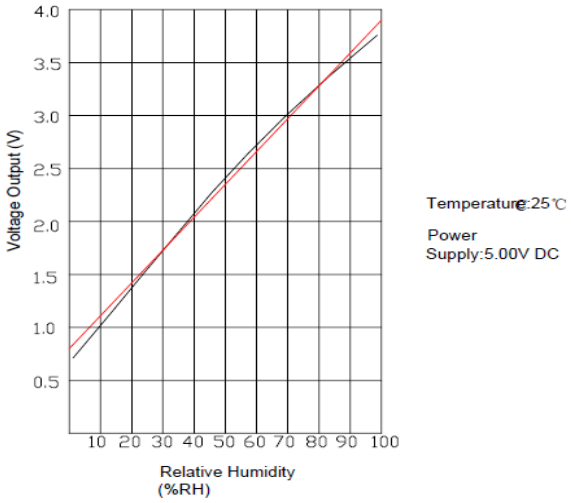
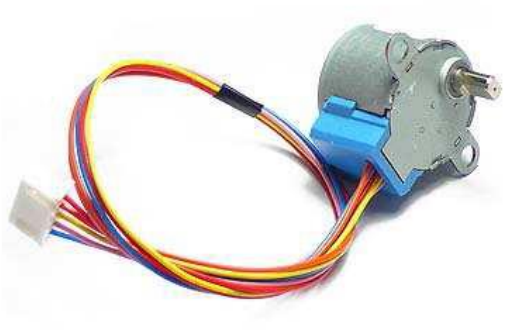


Figura 12. Gráfico de la función de transferencia del sensor de humedad

En la figura 12 está representada la función de transferencia del sensor de humedad, puede observarse que es prácticamente lineal.

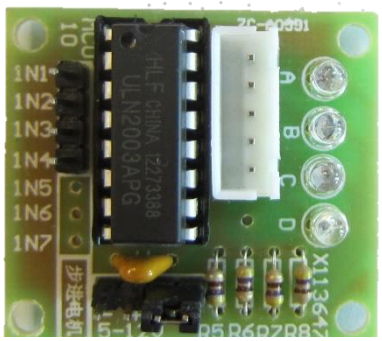
3.5. Stepper direccional



El stepper seleccionado para la dirección de la base es un motor unipolar modelo 28BYJ-48 (fig.13). El stepper 28BYJ posee una relación de engranajes de 1/64 dando como resultado que por cada paso avance 5,625/64 grados. La frecuencia de operación máxima es de 1000 Hz con una alimentación comprendida entre los 5 y los 12 V [10].

Figura 13. Stepper modelo 28BYJ-48

El par que ejerce el motor es de 34.3 mN.m con un consumo de aproximadamente 55mA.



Para su funcionamiento se ha utilizado un controlador ULN2003A (fig.14) que es un array de transistores Darlington que posee una conexión para el motor y unos pines para conectar a Waspmote. El controlador ULN2003A consiste en siete parejas de transistores NPN que dan salidas de alta tensión [11]. Cada entrada de este dispositivo tiene un diodo zener en serie con una resistencia para controlar la entrada de corriente.

Figura 14. Controlador ULN2003A

3.6. Stepper de avance

El stepper utilizado para que la base avance es un motor bipolar modelo STP-42D0001 (fig.15). El número de grados que avanza por cada paso es de 1.8º con una precisión de ±5%.

La temperatura de operación del motor se encuentra en el rango de 0 ºC a 50 ºC con una humedad de hasta el 90%.

La alimentación del motor para su correcto funcionamiento es de 12 V con una frecuencia máxima de hasta 1000 Hz.

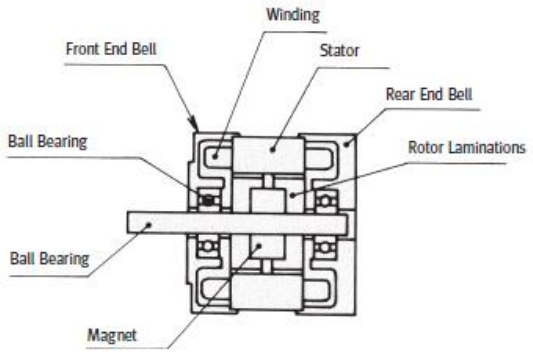


Figura 15. Stepper bipolar modelo STP-42D0001

Este stepper permite múltiples métodos de excitación (fig.16), es posible excitar una única fase, las dos fases a la vez o una fase y posteriormente la otra. Cada una de las posibles configuraciones afecta al consumo y al par del motor.

El controlador utilizado para el stepper de avance es el L9110S (fig.17) con una tensión de alimentación comprendida en el rango de 2.5V a 12V. El controlador L9110S es compatible con los niveles lógicos TTL.

La temperatura de funcionamiento del integrado está en el rango de 0 ºC a 80 ºC.

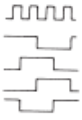
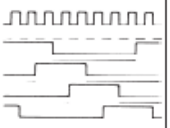
		Excitation Method		
		Single Phase	Dual Phase	1-2 Phase
Switching sequence	Pulse phase A phase B phase A phase B			
Features		Hold & running torque reduced by 39% Increased efficiency. Poor step accuracy.	High torque Good step accuracy.	Poor step accuracy. Good resonance characteristics. Higher pulse rates. Half stepping

Figura 16. Modos de excitación del stepper

Figura 17. Controlador L9110S

Capítulo 4. Waspmote IDE

A lo largo de este capítulo se explicará tanto la instalación del entorno de desarrollo de Waspmote como la estructura del programa una vez instalado.

4.1. Instalación del programa

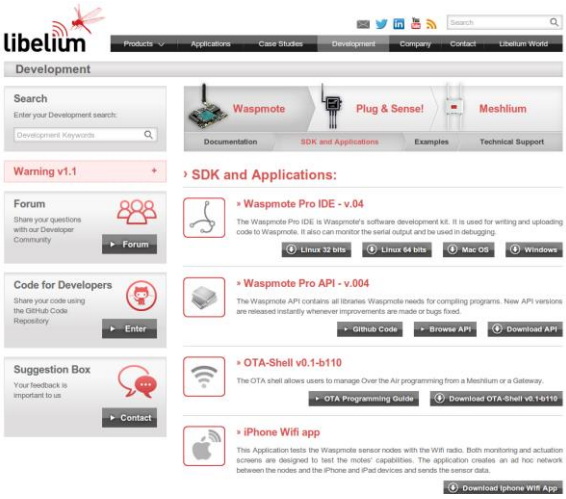


Figura 18. Pagina web de Libelium

A partir de Windows 7 la instalación de los drivers se realiza automáticamente. En versiones anteriores de Windows, descargar la última versión del driver de la página: http://www.ftdichip.com/Drivers/VCP.htm.

Abrir el 'Asistente para añadir hardware nuevo' y seleccionar la ubicación del driver descargado. El asistente buscará en la carpeta e instalará el driver más adecuado. Tras finalizar el asistente comprobar en el apartado de 'Dispositivos e impresoras' (fig.19) que se ha instalado correctamente. Debiera aparecer un nuevo dispositivo llamado FT232R USB UART.

La instalación del entorno de programación de *Waspmote* se realiza a través de un archivo descargable desde la página web de *Libelium (fig.18)*. El programa está disponible para los tres sistemas operativos principales, Linux, Mac OS y MS Windows [12].

Tras la descarga del archivo se descomprime su contenido en una carpeta. Antes de iniciar el programa, hay que conectar la tarjeta *Waspmote* para que la instalación de los drivers se realice correctamente.

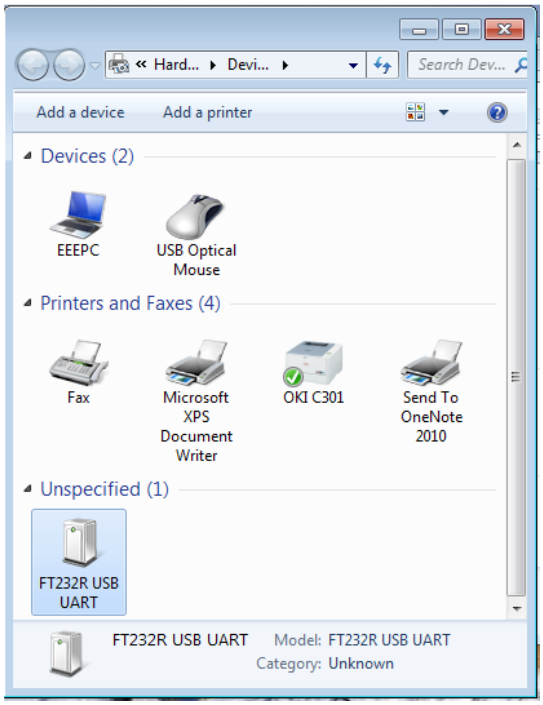


Figura 19. Dispositivos e impresoras de Windows

4.2. Descripción del entorno

El entorno de desarrollo de *Waspmote*, está basado en el entorno de desarrollo libre de Arduino. *Waspmote* IDE (fig.20) trae de serie numerosos ejemplos de programación de los diferentes módulos que posee [12].

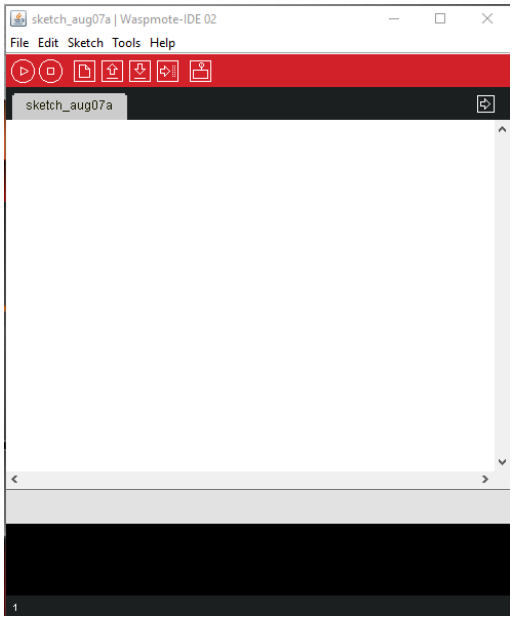


Figura 20. Ventana de Waspmote IDE

El entorno de programación además provee de las librerías necesarias para la programación de sus placas de expansión, así como de librerías para la programación de sus puertos de serie o leds.

Existe la posibilidad de añadir nuevas librerías al entorno para lo cual habría que copiarlas en la carpeta *hardware\cores\Waspmote-api-v.033* dentro la carpeta raíz del programa, además de añadir el nombre de archivo de la librería en el archivo *WaspClasses.h* dentro de la misma carpeta.

La obtención del software es gratuita y se realiza desde la propia página de *Libelium*. En este proyecto se ha utilizado la versión v.02 ya que es la última compatible con el modelo de hardware V1.1.

A continuación, se procede a explicar las diferentes partes del entorno y su funcionamiento. En primer lugar, tenemos los menús de selección (fig.21). El primer menú, *file*, nos permite abrir o guardar proyectos, además contiene un menú desplegable donde se ven todos los proyectos de ejemplo. El menú *Sketch* contiene las opciones de compilación del proyecto abierto. Por último, *Tools* contiene las opciones de configuración relativas a las librerías y al hardware. Desde este menú puede configurarse la lista de librerías que deben usarse en el proceso de compilación, además es el menú en el que se selecciona el puerto de la *Waspmote* que se desea programar. Los menús *Edit* y *Help* no requieren explicación ya que son iguales a los presentes en el resto de programas.

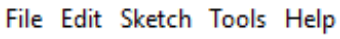


Figura 22. Menús de selección

Figura 21. Barra de accesos rápidos de Waspmote IDE

Bajo los menús de selección, se encuentra una barra de accesos rápidos (fig.22). Esta barra contiene 7 botones que realizan las funciones siguientes, de izquierda a derecha:

- Compila el proyecto en busca de errores.
- Detiene la compilación en curso del proyecto.
- Crea un nuevo proyecto.
- Abrir un proyecto existente.
- Guardar el proyecto actual.
- Compila el proyecto actual y lo carga en la Waspmote.
- Abre el puerto monitor serie en el puerto seleccionado dentro del menú Tools.

Bajo la barra de accesos rápidos se encuentran las pestañas con los diferentes proyectos abiertos (fig.23). Seguidamente está la zona del proyecto.

Figura 23. Pestañas de código en Waspmote IDE

Por último, se encuentra la consola (fig.24), donde aparecerán los errores de compilación o en el caso de una compilación satisfactoria un aviso en blanco con los bits cargados.

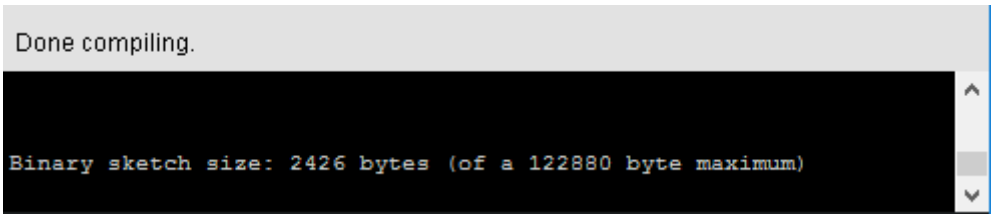


Figura 24. Consola de Waspmote IDE

Los proyectos escritos utilizando el IDE se llaman sketches. Estos sketches se escriben en el editor de textos. El IDE guarda estos sketches con extensión '.pde'.

Capítulo 5. Aplicación desarrollada

Una vez se han analizado tanto el software como el hardware que se va a utilizar, en este capítulo se llevará a cabo el desarrollo del sistema para la monitorización de variables ambientales.

El objetivo principal es la captación de medidas de temperatura, presión, humedad, luminosidad y presencia las cuales son enviadas de manera inalámbrica al ordenador. Desde el ordenador se realiza un procesado de los datos a través de Matlab realizando una presentación gráfica de los mismos. Así mismo, el sistema de monitorización está montado sobre una plataforma móvil que es controlada desde el ordenador.

El funcionamiento de la aplicación desarrollada se subdivide en dos esquemas independientes que pueden verse en la figura 25 y la figura 26.

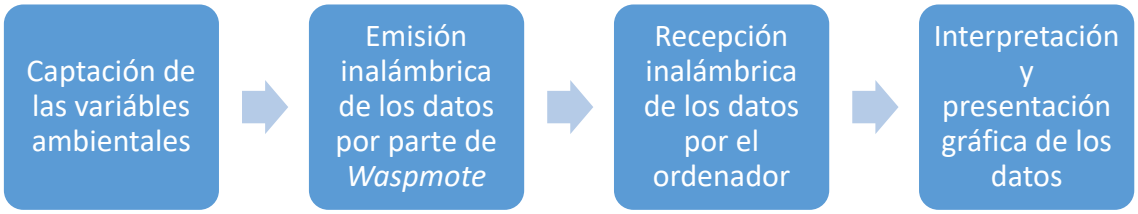


Figura 25. Esquema de funcionamiento de la captación y procesado de medidas

Esto permite desarrollar de manera independiente ambos módulos facilitando el diseño y el debugging.

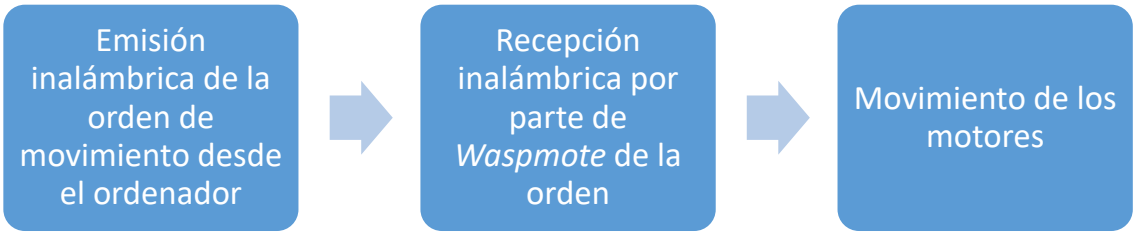


Figura 26. Esquema de funcionamiento de la plataforma móvil

5.1. Programación de una red de sensores inalámbrica

5.1.1. Programación de los sensores embebidos

La programación de los sensores embebidos es un proceso sencillo gracias a las librerías que acompañan al IDE de desarrollo. En el proyecto se han utilizado los siguientes sensores embebidos en la plataforma *Waspmote*:

- Sensor de temperatura propio del RTC
- Sensor acelerómetro

Ambos sensores se activan inicialmente en la función **setup()** y se configuran dentro de la misma. El código utilizado para ello es el siguiente:

```
void setup()
{
  RTC.ON(); //Inicializamos el RTC
  RTC.clearAlarmFlag(); //Reseteo de las flags del RTC
  RTC.setAlarm1("15",RTC_OFFSET,RTC_ALM1_MODE5); //Creamos una alarma que
se active cada 30 segundos
  ACC.ON(); //Inicialización del acelerómetro
  ACC.setFF(); //Hacemos que la interrupción del acelerómetro se active
cuando la placa este en caída libre

  led0_OFF(); //Inicialización del LED 0
  led1_OFF(); //Inicialización del LED 1

}
```

Figura 27. Código de inicialización de la tarjeta de sensores

En la figura 27 puede observarse como se enciende el reloj del RTC y se elimina la interrupción del RTC dentro del array de interrupciones, esto es necesario ya que si no se realiza únicamente se detectará una interrupción y esta permanecerá activa siempre dentro del array. A continuación, lo que se configura es una alarma para que realice una determinada acción cada 15 segundos, dicha acción se programa posteriormente; en este caso será la lectura de todos los sensores de la plataforma.

El acelerómetro es el siguiente sensor en inicializarse, en este caso se ha configurado para detectar interrupciones de caída libre. En el caso del acelerómetro, la inicialización de la interrupción dentro del array se realiza únicamente si se detecta una interrupción debida al acelerómetro.

Por último, se inicializan los LED de *Waspmote* de tal manera que permanezcan apagados hasta que alguna otra función los active.

5.1.2. Programación de los sensores de la Event board

La programación de los sensores conectados a la *Event board* se beneficia de una librería existente gracias a lo cual las medidas se realizan conociendo cuales son los pines correspondientes a cada sensor. En este caso, hay cuatro sensores conectados a la *Event board*, y su lectura se realiza del modo presentado en la figura 28.

```
char* humedad()
{
  dtostrf(SensorEvent.readValue(SENS_SOCKET5),5,3,Humedad);
  return Humedad ;
}
```

Figura 28. Código de ejemplo para la medida de un sensor

Cada sensor posee una función propia que devuelve un array de caracteres. Esto es así ya que los datos que se envían a través del puerto de comunicación serie son cadenas de caracteres. Para la conversión del tipo de datos 'double' a char* se utiliza la función **dtostrf()** cuyos parámetros son el valor que se quiere convertir, el número de dígitos totales, el número de dígitos decimales y la variable en la que guardar el array de caracteres.

En la figura 28 puede verse que el sensor de humedad está conectado al pin 5 de la *Event board* y que el valor de la medida se almacena en la variable HUMEDAD en forma de array de datos.

```
void sensores()
{
  if(intFlag & ACC_INT){
    USB.begin();
    USB.println("Interrupcion del acelerometro, CAIDA LIBRE!");
    led1_ON();
    delay(5000);
    ACC.clearAlarmFlag(); //Reseteo de las flags del Acelerómetro
    RTC.clearAlarmFlag(); //Reseteo de las flags del RTC
    ACC.setFF(); //Hay que reactivar la interrupción de caída libre.
    led1_OFF();
    USB.flush();
    USB.close();
    delay(500);
  }
  else if(intFlag & RTC_INT){
    SensorEvent.setBoardMode(SENS_ON);
    led0_ON();
    delay(6000); //Tiempo de estabilización del sensor PIR
    RTC.clearAlarmFlag(); //Reseteo de las flags del RTC
    temperatura();
    humedad();
    luminosidad();
    presion();
    pir();
    sprintf(StrData, "#%s; %s; %s; %s; %s\n", Temperatura, Humedad,
Luminosidad, Presion, Pir );
    SensorEvent.setBoardMode(SENS_OFF);
    wifiConfig();
    delay(2000);
    led0_OFF();
    led1_OFF();
  }
}
```

Figura 29. Protocolo de almacenamiento y emisión de medidas

En la figura 29 se presenta la programación de todos los sensores conectados. Las medidas de los sensores se realizan cada 15 segundos, siempre que no se haya dado una interrupción debida a caída libre. Para que no se sobrepase el consumo permitido por *Waspmote*, es necesario activar la alimentación y desactivarla en cada medida, ya que si se mantiene activa la *Event board* a la vez que la tarjeta XBee, se produce un sobreconsumo que interrumpe el correcto funcionamiento de Waspmote resultando en continuos reinicios del sistema.

Para la correcta transmisión de los datos, se diseña un protocolo de transmisión en el que se incluyen los datos de todos los sensores en un orden concreto. Para detectar cual es el comienzo de la línea se introduce el carácter '#' y para detectar cada uno de los datos, estos se formatean de tal modo que lleven ';' al final. Una vez se tienen los datos correctamente formateados se desactiva la alimentación de los sensores y se envían a través de la tarjeta XBee.

5.2. Programación de la conexión inalámbrica

5.2.1. Módulos necesarios para la conexión

Para la conexión de la Waspmote dedicada a la toma de medidas con el ordenador, y la conexión de la segunda Waspmote dedicada al control de los steppers al ordenador, es necesaria la utilización de al menos tres tarjetas XBee 802.15.4 junto con un *Gateway*.

Las tres tarjetas han de programarse previo uso en el proyecto a través de la aplicación XCTU para que sea posible la recepción y envío de paquetes de datos.

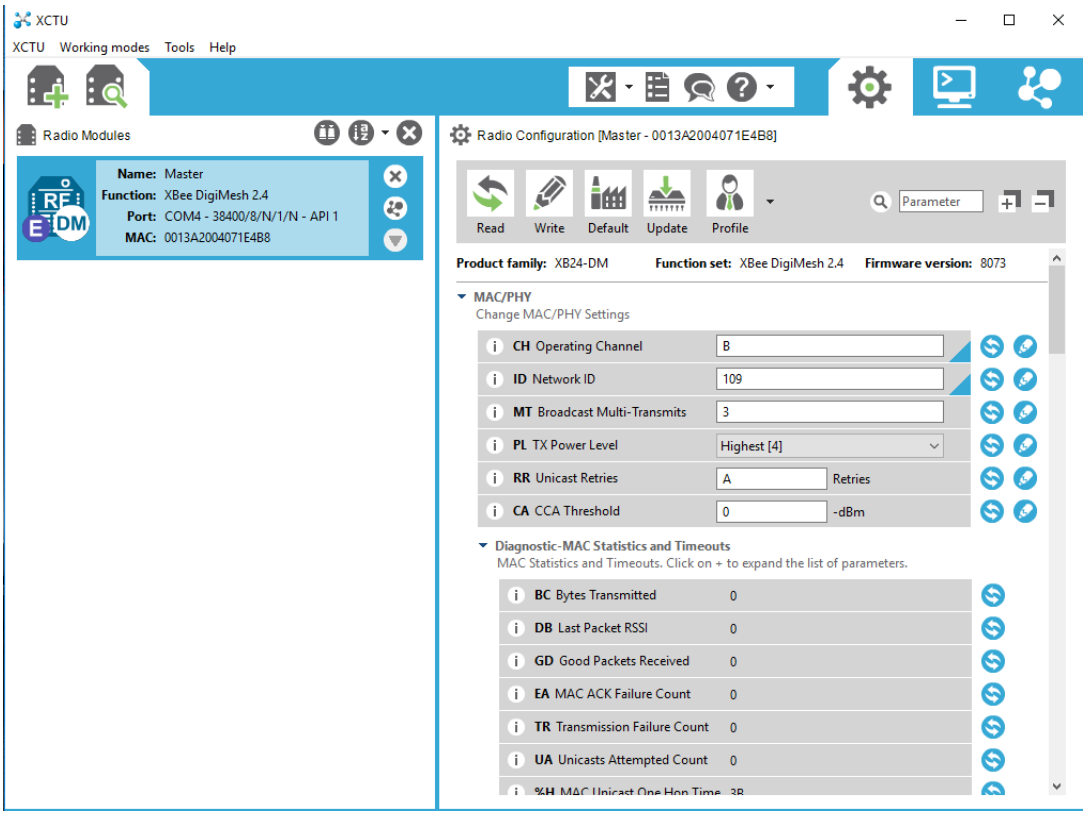


Figura 30. Programa XCTU para XBee

5.2.2. Implementación de una red XBee

Para la implementación de una red XBee se ha utilizado el firmware digimesh ya que permite redes en las que cada dispositivo puede comunicarse con todos los dispositivos en lo que se denomina una red mesh, evitando de este modo el funcionamiento maestro-esclavo.

Cada tarjeta se ha configurado como dispositivo final (end device) ya que la otra opción, router (standard router), no es necesaria en este caso siendo la red tan pequeña. La velocidad de funcionamiento (BaudRate) ha de configurarse en 38400 Baudios para su correcto funcionamiento con *Waspmote*.

Una vez se ha configurado el modo de operación de cada tarjeta y su velocidad, es necesario configurar la red. Para ello hay que fijar en cada tarjeta los siguientes parámetros:

- Canal de operación (CH)
- ID de red (ID)
- Encriptación y clave (EE y KY)
- API Activo (AP)

La API no es un parámetro propio de la configuración de la red, sino que es un parámetro del formato que se da a las tramas enviadas y recibidas. La encriptación no es obligatoria y por evitar problemas a la hora de conexión no se ha utilizado en el desarrollo del proyecto, si bien su uso se recomienda en los manuales [5].

La configuración de red utilizada ha sido la siguiente:

Operating Channel	Network ID	Routing/Messaging Mode	Encryption Enable	API Enable
B	109	End Device	Disabled	API Mode Without Escapes

Tabla 1. Configuración de la red XBee

Una vez se han configurado estos parámetros a través del programa XCTU, el envío de paquetes se programa desde el Waspmote IDE.

```
xbeeDM.init(DIGIMESH,FREQ2_4G,NORMAL);
```

Figura 31. Código de inicialización del módulo XBee

```
void wifiConfig()
{
  xbee802.ON();
  xbeeDM.send("0013A2004071E4B8", StrData);
  xbeeDM.OFF();
}
```

Figura 32. Código de configuración del envío de paquetes de datos

En la figura 31 incluye la línea que configura Waspmote para el envío y recepción de paquetes a través del módulo XBee digimesh, esta línea de código se coloca dentro de la función **setup()**. Los parámetros que se le pasan a dicha función son el nombre del módulo que se utiliza, la frecuencia en la que opera dicho módulo y por último la versión del módulo (Normal o Pro).

En la figura 32 puede verse la función utilizada en la Waspmote encargada de los sensores para enviar los datos al ordenador.

5.3. Plataforma móvil

La plataforma móvil consiste en una placa de plástico ABS (fig.33) sobre la que se han fijado dos motores stepper.

Uno de los motores se encarga de la tracción de la base por medio de una serie de engranajes conectados a un eje con ruedas, mientras que el otro motor se encarga de controlar el ángulo de giro de una tercera rueda para controlar la dirección del movimiento (fig.34).

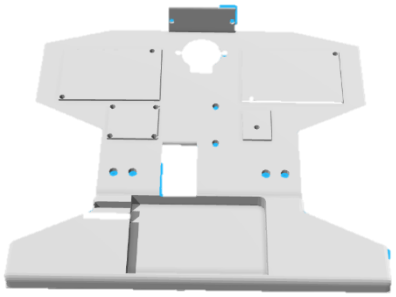


Figura 33. Chasis de la plataforma móvil

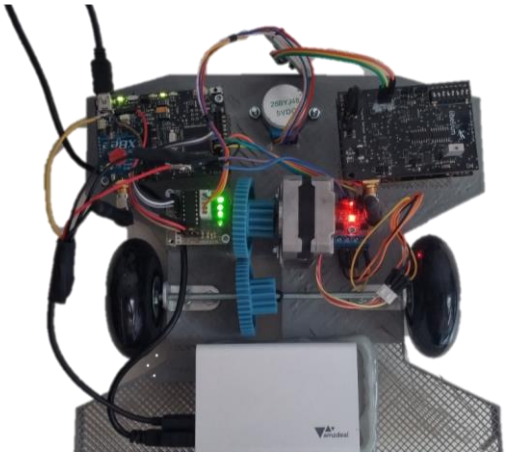


Figura 34. Sistema de captación de medidas sobre plataforma móvil

Para la programación de estos motores se utilizan 4 pines de la *Waspmote* en cada caso además de la alimentación. Para el giro de un motor stepper es necesario excitar el devanado con pulsos de una frecuencia determinada por el fabricante en un orden concreto.

En conexionado de los devanados dentro del stepper varía según el modelo utilizado (fig.35). En este caso, el stepper modelo STP-42D0001 tiene dos devanados independientes mientras que el stepper modelo 28BYJ conecta ambas bobinas.

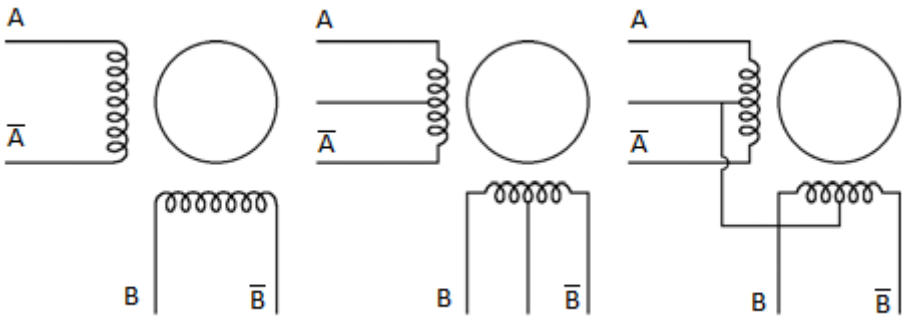


Figura 35. Tipos de devanado dentro de un stepper

La excitación secuencial de dichas bobinas genera unos campos magnéticos que hacen que el motor se mueva. Existen varios modos de excitación posibles en función de si se busca el máximo par motor, la máxima precisión por paso, o un compromiso entre ambas. Para el motor encargado de la tracción se buscaba el máximo par motor, por lo que se han excitado ambas fases simultáneamente (Tabla2).

	A	$\overline{A}$	B	$\overline{B}$
Paso 1	1	0	1	0
Paso 2	0	1	1	0
Paso 3	0	1	0	1
Paso 4	1	0	0	1

Tabla 2. Excitación de las fases del stepper de tracción

Por otro lado, el stepper encargado de la dirección se ha programado de tal forma que permita dar medios pasos, de esta forma se pierde precisión por paso, pero el movimiento es menos brusco (Tabla 3).

	A	$\overline{A}$	B	$\overline{B}$
Paso 0.5	1	0	0	0
Paso 1	1	1	0	0
Paso 1.5	0	1	0	0
Paso 2	0	1	1	0
Paso 2.5	0	0	1	0
Paso 3	0	0	1	1
Paso 3.5	0	0	0	1
Paso 4	1	0	0	1

Tabla 3. Excitación de las fases del stepper de movimiento

A continuación, se presenta la función utilizada para el control de un stepper.

```
void step( int steps_to_move)
{
  int steps_left = abs(steps_to_move); //nº de pasos que hay que dar.
  if(steps_to_move > 0){direction = 0;}
  if(steps_to_move < 0){direction = 1;}

  while(steps_left > 0)
  {
       digitalWrite(DIGITAL8, Paso[step_number][0]);
       digitalWrite(DIGITAL7, Paso[step_number][1]);
       digitalWrite(DIGITAL6, Paso[step_number][2]);
       digitalWrite(DIGITAL5, Paso[step_number][3]);

       if(direction == 1)
       {
           step_number++;
       }else
       {
           step_number--;
       }
       //decremento de los pasos que quedan
       steps_left--;
       if (steps_left != 0)
       {
         step_number=(step_number+4)%4;
       }
       delay(1);
  }
}
```

Figura 36. Código de la función de control del stepper

En la figura 36, los modos de excitación se han almacenado en un array **Paso** de tal forma que en cada iteración se escribe el contenido correspondiente del array en los pines del motor, haciendo que este retroceda o avance consecuentemente.

5.4. MatLab

5.4.1. Descripción del entorno

El entorno de desarrollo de Matlab permite está enfocado al desarrollo de aplicaciones científicas. La plataforma de Matlab esta optimizada para resolver problemas de ingeniería y científicos. A continuación, vamos a describir el entorno de desarrollo de Matlab junto con algunas de sus funciones [13].

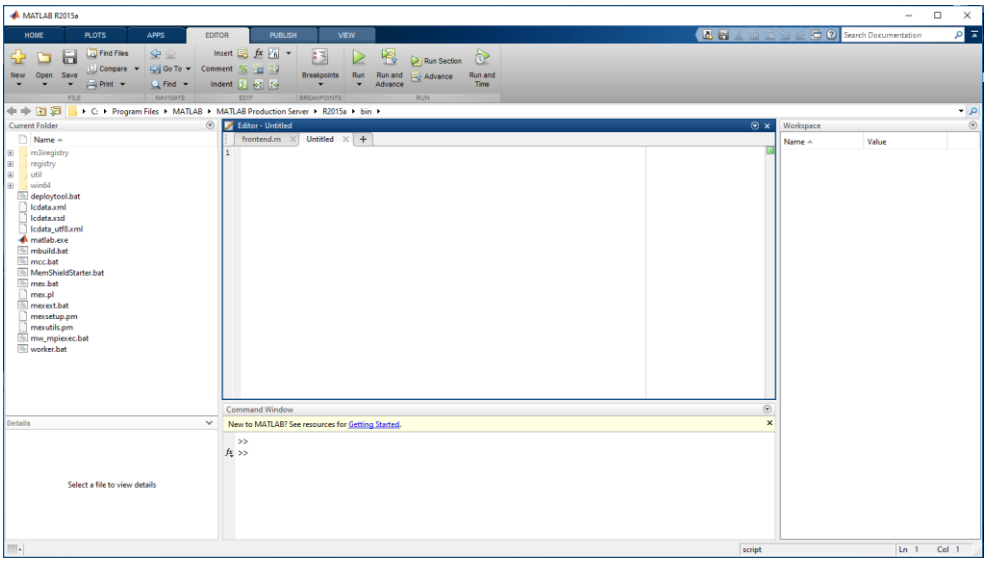


Figura 37. Entorno de programación de Matlab

Matlab es un lenguaje de alto nivel enfocado al cálculo científico. Posee herramientas que permiten la visualización gráfica de los datos introducidos.

En la figura 37 pueden observarse las diferentes partes del entorno que se describirán a continuación. En el lado izquierdo de la pantalla se encuentra la carpeta raíz (fig.38), es decir, la carpeta desde la cual se ejecutará el programa que se está desarrollando. Desde este menú pueden verse las diferentes funciones o clases que se han programado y que se encuentran dentro del mismo directorio.

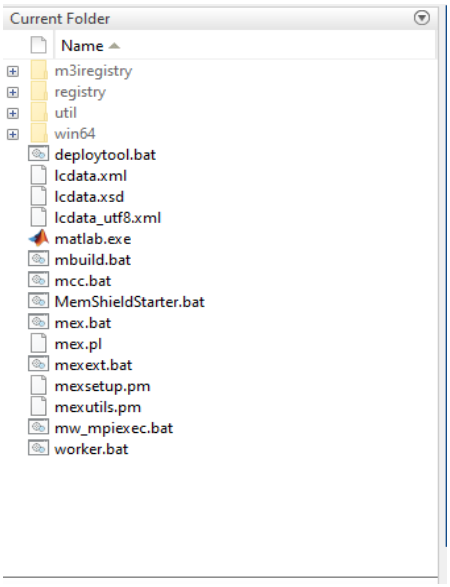


Figura 38. Directorio actual del

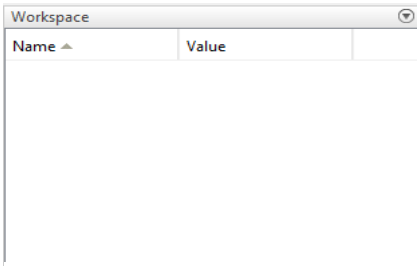


Figura 39. Espacio de trabajo

En el lado contrario, se encuentra el workspace (fig.39), en este espacio aparecen las variables que crea el programa en uso, o las variables que se escriben en la línea de comandos.

Entre ambas columnas, se encuentran el editor de textos y la línea de comandos (fig.40). En el editor aparecerán las palabras reservadas de Matlab en un color distinto, además según se vayan escribiendo funciones, Matlab dará a opción de ocultar el cuerpo de la función para facilitar la lectura del código.

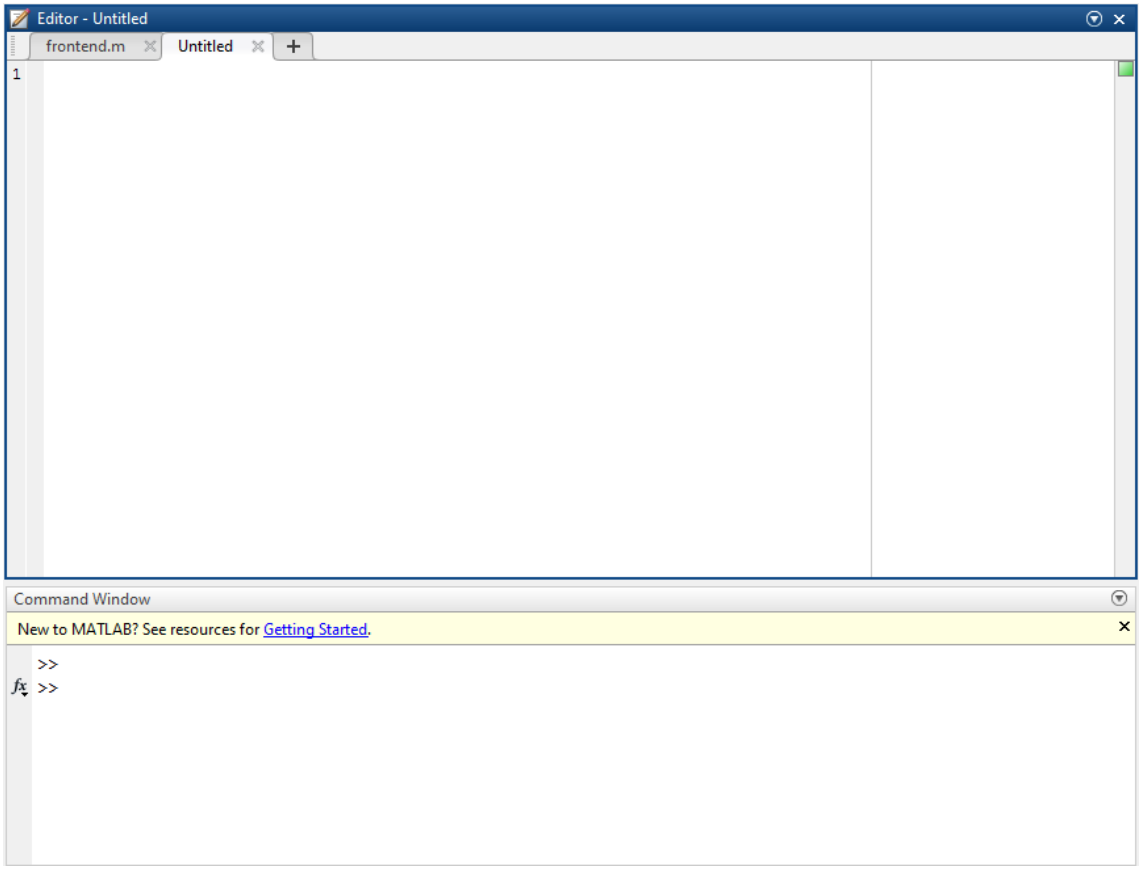


Figura 40. Editor de texto (arriba) y línea de comandos (abajo)

Por otro lado, es posible ejecutar funciones y cálculos en Matlab sin necesidad de crear un archivo, para ello basta escribir en la línea de comandos la función que se desea ejecutar.

Matlab permite a partir de Simulink la programación visual de elementos (fig.41). Utilizando esta característica, es posible programar procesos de control, sin embargo, esto va más allá de los objetivos de este proyecto y no se entrará en detalles.

Por último, es posible generar un archivo ejecutable del programa realizado una vez se ha finalizado su desarrollo.

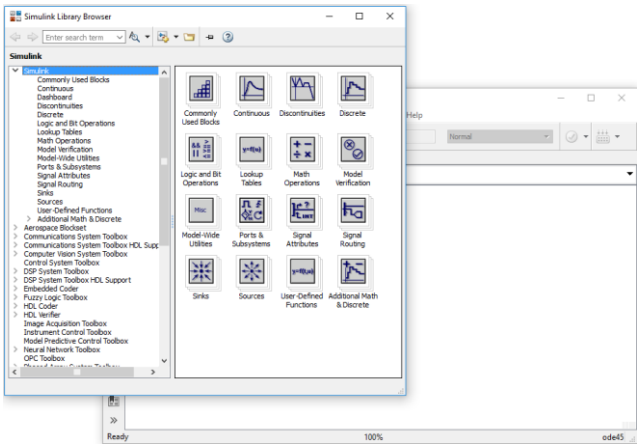


Figura 41. Entorno Simulink

5.4.2. Programación de la conexión SERIE

La programación de una función que controle la conexión del puerto de serie, se realiza comprobando el sistema operativo que corre el ordenador anfitrión. Esto se debe a que algunas de las funciones utilizadas durante la programación no son compatibles con sistemas basados en Unix o Linux, por lo tanto, si no se cumple la condición del sistema el programa directamente lanza una ventana de aviso y se cierra.

```
function Conectar_Callback(hObject, eventdata, handles)
    % hObject    handle to Conectar (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % guidata(hObject, handles);
    handles = guidata(hfigure);
    if ispc
         disp('Dispositivo Windows')
         handles.s=serial('COM4');
         set(handles.s,'BaudRate',38400);
         handles.s.Terminator = 'CR';
         fopen(handles.s);
         handles.err=0;
         handles.conected=1;
    else
        msgbox('Incompatible con este sistema')
        handles.err=1;
        handles.conected=0;
        delete(handles.figure1);

    end
    guidata(hfigure, handles);
```

Figura 42. Código de conexión serie

En la figura 42 se observa la existencia de una estructura con nombre ***handles,*** esta es necesaria para el correcto funcionamiento de la interfaz gráfica/controlador que se creará con Matlab ya que contiene gran parte de las variables que se utilizan. La función presente en el código 4 tras fijar el puerto y la velocidad de la conexión, abre la conexión y la mantiene abierta hasta el cierre del programa.

5.4.3. Procesamiento y presentación de los datos

```
function update_display(hObject,eventdata,hfigure)
    % Timer timer1 callback, called each time timer iterates.
    handles=guidata(hfigure);
    popup_sel_index = get(handles.popupmenu1, 'Value');
    if handles.i == 0
        handles.plot=plot(0);
        set(handles.axes1,'YGrid','on');
        handles.title=title('');
    end
    if handles.conected
% --- Funcion de lectura del puerto de serie.
        if ~handles.err && handles.conected
            handles.t=[handles.t 10*handles.i];
            handles.i=handles.i + 1;
            dataArray=zeros(1,30);
            a=fread(handles.s,1);
            while a~=35 %35 es la # en ascii
                a=fread(handles.s,1);
            end
            i=1;
            while (a~=10) %10 es LF en ascii
                dataArray(i)=a;
                i=i+1;
                a=fread(handles.s,1);
            end
            dataArray(i)=0;
            dataArray=char(dataArray);

            handles.data=str2double(dataArray);
            handles.tempbak{end + 1}=dataArray(2:3);
            handles.presbak{end + 1}=dataArray(6:10);
            handles.humedbak{end + 1}=dataArray(13:17);
            handles.lumnbak{end + 1}=dataArray(20:24);
            handles.presenbak{end + 1}=dataArray(27:29);

            handles.temp=str2double(handles.tempbak);
            handles.pres=str2double(handles.presbak);
            handles.humed=str2double(handles.humedbak);
            handles.lumn=str2double(handles.lumnbak);
            handles.presen=str2double(handles.presenbak);
        end
        switch popup_sel_index
            case 1
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Temperatura');
                    set(handles.plot,'LineWidth',2.0);
                    set(handles.plot,'Color','y');
                    handles.before=1;
                end
                set(handles.plot,'Ydata',handles.temp);
                set(handles.plot,'Xdata',handles.t);
            case 2
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Presión');
                    set(handles.plot,'LineWidth',2.0);
                    set(handles.plot,'Color','k');
                    handles.before=2;
                end
                set(handles.plot,'Ydata',handles.pres);
                set(handles.plot,'Xdata',handles.t);
```

```
            case 3
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Humedad');
                    set(handles.plot,'LineWidth',2.0);
                    set(handles.plot,'Color','b');
                    handles.before=3;
                end
                set(handles.plot,'Ydata',handles.humed);
                set(handles.plot,'Xdata',handles.t);
            case 4
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Luminosidad');
                    set(handles.plot,'LineWidth',2.0);
                    set(handles.plot,'Color','c');
                    handles.before=4;
                end
                set(handles.plot,'Ydata',handles.lumn);
                set(handles.plot,'Xdata',handles.t);
            case 5
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Presencia');
                    set(handles.plot,'LineWidth',2.0);
                    set(handles.plot,'Color','g');
                    handles.before=5;
                end
                set(handles.plot,'Ydata',handles.presen);
                set(handles.plot,'Xdata',handles.t);
        end
    end
    guidata(hfigure,handles);
```

Figura 43. Protocolo de procesamiento de datos

El procesado de los datos, requiere la utilización del protocolo desarrollado en *Waspmote* para obtener unos datos inteligibles. Debido a que los datos se transmiten de forma inalámbrica, la lectura de bytes a través del puerto de serie da como resultado además de los datos que se envían, bytes espurios que hay que obviar.

Como puede verse en la figura 43, el primer paso es la búsqueda del carácter '#'. Esto es así, ya que, tras comprobar experimentalmente que entre los bytes espurios nunca se formaba el carácter '#', se estableció como carácter de inicio de los datos válidos.

Una vez se ha encontrado el inicio, los datos se almacenan sin organizar en un array de datos. El almacenamiento se realiza hasta que se encuentra un carácter 'LF'. Cuando se tienen los datos en un único array lo que se hace es separar los datos respecto a las características ambientales que se quieren medir y representar, de esta forma se obtiene un array de tipo 'double' que puede representarse en un plot.

Tras almacenar el array, dependiendo de la variable atmosférica que se quiera representar, se fija el título y color de la gráfica y se representan todos los puntos del array de datos frente al tiempo.

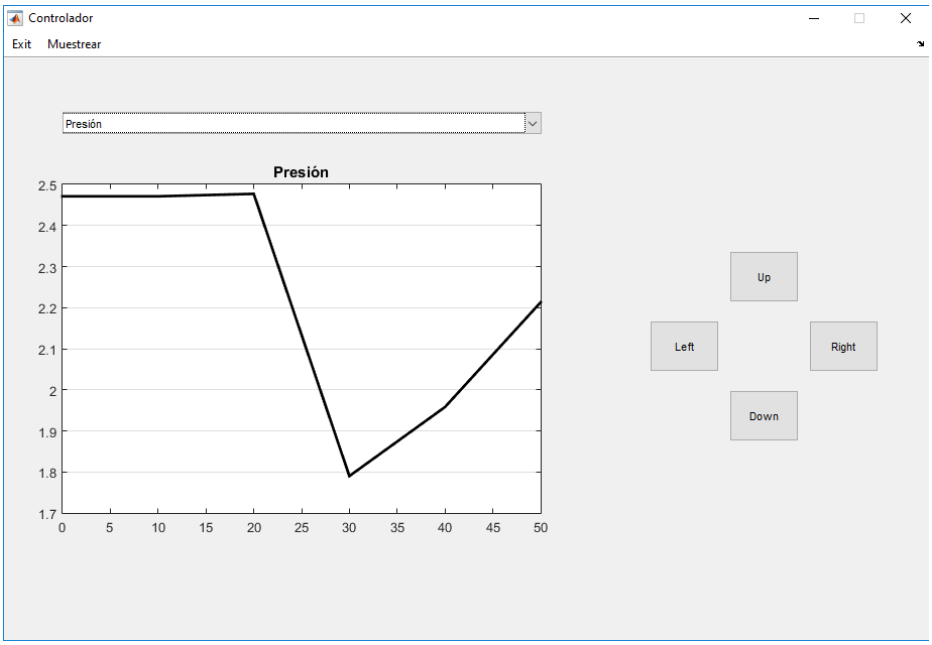


Figura 44. Interfaz gráfica/Controlador del sistema

En la figura 44 puede observarse la interfaz gráfica/controlador desarrollado en Matlab, a un lado se encuentra un desplegable para seleccionar la gráfica que se representara en los ejes. La escala del eje Y se autoajusta a cada array de datos, mientras que la escala del eje X es siempre la misma ya que depende del intervalo de muestreo (10 segundos). En el lado derecho de la interfaz se encuentran los botones destinados al control de los motores de la plataforma móvil. Por último, los dos botones *Exit* y *Muestrear son necesarios para controlar la desconexión del puerto de serie y el control del timer de muestreo respectivamente.*

5.5. Resultados

Para comprobar el correcto funcionamiento del sistema se ha puesto en marcha todo el sistema simultáneamente, es decir, se han activado tanto la *Waspmote* dedicada a la medida de valores atmosféricos como la *Waspmote* dedicada al control de los motores. Si bien el sistema ha funcionado a través de conexiones cableadas, el resultado no ha sido el mismo al convertir dichas conexiones en inalámbricas.

La captación de datos es completamente operativa y prueba de ello es que los datos se representan de manera correcta en el entorno diseñado en Matlab. Sin embargo, la recepción de ordenes enviadas desde el ordenador para el control de los motores no ha funcionado, posiblemente debido a una mala configuración en la tarjeta XBee.

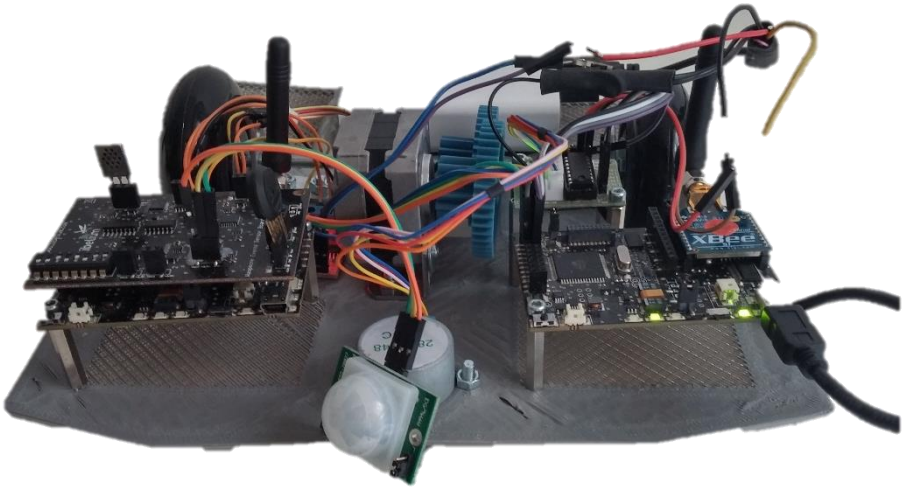


Figura 45. Frontal del sistema de captación de medidas con plataforma móvil

Capítulo 6. Conclusiones

De los resultados obtenidos durante el desarrollo del proyecto puede concluirse que:

Se ha logrado desarrollar un sistema basado en la plataforma Waspmote con la capacidad para captar medidas ambientales. Se ha diseñado un módulo secundario que permite la movilidad del sistema sobre una base, aunque no es controlable de manera inalámbrica.

Se ha realizado un diseño de la base en un programa de diseño asistido por ordenador (CAD) para su posterior impresión con la finalidad de darle un acabado profesional.

Respecto al procesado y presentación de los datos obtenidos ha sido de gran utilidad el entorno de Matlab ya que, sin las funciones propias del lenguaje, la creación de un entorno GUI para un usuario sin experiencia hubiera requerido mucho más tiempo.

Tras las pruebas realizadas en el laboratorio se confirma el cumplimiento de la mayoría de los objetivos exceptuando el control inalámbrico de la base. A pesar de ello queda margen de mejora y como futuro trabajo se pretende que no sea necesaria la presencia de un usuario. Para ello es preciso conocer la posición de la base y deberá desarrollarse un método preciso de localización, por ejemplo, basado en gps.

Bibliografía

[1] «Wikipedia,» 19 febrero 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Adquisición_de_datos.

[2] M. D. P. Emilio, Embedded systems design for high-speed data acquisition and control, Springer, 2015.

[3] Libelium Comunicaciones Distribuidas, Waspmote Guía Técnica, 2012.

[4] Libelium Comunicaciones Distribuidas, Waspmote 802.15.4 Networking Guide, 2012.

[5] Libelium Comunicaciones Distribuidas, Waspmote Digimesh Networking Guide, 2012.

[6] STMicroelectronics, «MEMS inertial sensor LIS3LV02DL,» 2008.

[7] Freescale Semiconductor, «Integrated Silicon Pressure Sensor MPX4115A Datasheet,» 2006.

[8] Libelium Comunicaciones Distribuidas, Eventos Guía Técnica, 2012.

[9] Sencera Co., «808H5V5 humidity transmitter datasheet».

[10] Kiatronic, «28BYJ-48-5V Stepper Motor Datasheet,» Cherrywood.

[11] Prometec, «http://www.prometec.net,» 18 Agosto 2015. [En línea]. Available: http://www.prometec.net/motor-28byj-48/.

[12] Libelium Comunicaciones Distribuidas, «Waspmote IDE User Guide,» 2014.

[13] The MathWorks, «MatLab,» 2016. [En línea]. Available: http://es.mathworks.com/products/matlab/features.html. [Último acceso: 18 Agosto 2016].

Anexos

Código Sensores

```
char StrData[25];
char Temperatura[3];
char Luminosidad[6];
char Presion[6];
char Humedad[6];
char Pir[2];

void setup()
{
  RTC.ON(); //Inicializamos el RTC
  RTC.clearAlarmFlag(); //Reseteo de las flags del RTC
  RTC.setAlarm1("15",RTC_OFFSET,RTC_ALM1_MODE5); //Creamos una
alarma que se active cada 30 segundos
  ACC.ON(); //Inicialización del acelerómetro
  ACC.setFF(); //Hacemos que la interrupción del acelerómetro se
active cuando la placa este en caída libre

  xbeeDM.init(DIGIMESH,FREQ2_4G,NORMAL); //Inicialización de la
libreria XBee digimesh

  led0_OFF(); //Inicializacion del LED 0
  led1_OFF(); //Inicializacion del LED 1

}

void loop()
{
  sensores();
}

void led0_ON() //función de encendido del LED 0
{
  Utils.setLED(LED0, LED_ON);
}

void led1_ON() //función de encendido del LED 1
{
  Utils.setLED(LED1, LED_ON);
}

void led0_OFF() //función de apagado del LED 0
{
  Utils.setLED(LED0, LED_OFF);
}

void led1_OFF() //función de apagado del LED 1
{
  Utils.setLED(LED1, LED_OFF);
}

char* temperatura()
{
  float temperatura=RTC.getTemperature();
  dtostrf(temperatura,2,0,Temperatura);
  return Temperatura;
```

```
}

char* luminosidad()
{
  dtostrf(SensorEvent.readValue(SENS_SOCKET3),5,3,Luminosidad);
  return Luminosidad ;
}

char* humedad()
{
  dtostrf(SensorEvent.readValue(SENS_SOCKET5),5,3,Humedad);
  return Humedad ;
}

char* presion()
{
  dtostrf(SensorEvent.readValue(SENS_SOCKET8),5,3,Presion);
  return Presion;
}

char* pir()
{
  dtostrf(SensorEvent.readValue(SENS_SOCKET7),1,0,Pir);
  return Pir;
}

void sensores()
{
  if(intFlag & ACC_INT){
    USB.begin();
    USB.println("Interrupcion del acelerometro, CAIDA LIBRE!");
    led1_ON();
    delay(5000); //Tiempo de estabilización del PIR
    ACC.clearAlarmFlag(); //Reseteo de las flags del ACC
    RTC.clearAlarmFlag(); //Reseteo de las flags del RTC
    ACC.setFF(); //Hay que reactivar la interrupción de caída
libre.
    led1_OFF();
    USB.flush();
    USB.close();
    delay(500);
  }
  else if(intFlag & RTC_INT){
    SensorEvent.setBoardMode(SENS_ON);
    led0_ON();
    delay(8000);
    USB.begin();
    RTC.clearAlarmFlag(); //Reseteo de las flags del RTC
    temperatura();
    humedad();
    luminosidad();
    presion();
    pir();
    sprintf(StrData, "#%s; %s; %s; %s; %s\n", Temperatura,
Humedad, Luminosidad, Presion, Pir );
    SensorEvent.setBoardMode(SENS_OFF);
    USB.println(StrData);
    wifiConfig();
    led0_OFF();
    led1_OFF();
    USB.flush();
```

```
    USB.close();
  }
}
void wifiConfig()
{

  xbee802.ON();
  led1_ON();
  xbeeDM.send("0013A2004071E4B8", StrData);
  xbeeDM.OFF();
  led1_OFF();
}
```

Código Motores

```
int step_number;
int direction;
//es un stepper de 1,8° por step
int number_of_steps = 200;
int Paso [ 4 ][ 4 ] =
   {  {1, 0, 1, 0},
      {0, 1, 1, 0},
      {0, 1, 0, 1},
      {1, 0, 0, 1}
   };
int step_number2;
//es un stepper de 5.625°/64 grados por step
int number_of_steps2 = 4096;
int Paso2 [ 8 ][ 4 ] =
   {  {1, 0, 0, 0},
      {1, 1, 0, 0},
      {0, 1, 0, 0},
      {0, 1, 1, 0},
      {0, 0, 1, 0},
      {0, 0, 1, 1},
      {0, 0, 0, 1},
      {1, 0, 0, 1}
   };

char data[100];


void setup()
{
  RTC.clearAlarmFlag(); //Reseteo de las flags del RTC
  RTC.ON(); //Inicializamos el RTC

  xbeeDM.init(DIGIMESH,FREQ2_4G,NORMAL); //Inicialización de la
librería XBee 802.15.4

  led0_OFF();   //Inicialización del LED 0
  led1_OFF();   //Inicialización del LED 1

  //Configuración de los pines
  digitalWrite(SENS_PW_5V,LOW);
  pinMode(DIGITAL1, OUTPUT); //Stepper bipolar de dirección CH
B-IB
  pinMode(DIGITAL2, OUTPUT); //Stepper bipolar de dirección CH
B-IA
  pinMode(DIGITAL3, OUTPUT); //Stepper bipolar de dirección CH
A-IB
```

```
  pinMode(DIGITAL4, OUTPUT); //Stepper bipolar de dirección CH
A-IA

  pinMode(DIGITAL5, OUTPUT); //Stepper bipolar de movimiento CH
B-IB
  pinMode(DIGITAL6, OUTPUT); //Stepper bipolar de movimiento CH
B-IA
  pinMode(DIGITAL7, OUTPUT); //Stepper bipolar de movimiento CH
A-IB
  pinMode(DIGITAL8, OUTPUT); //Stepper bipolar de movimiento CH
A-IA

}
void loop()
{
    xbeeDM.ON();
    xbeeDM.treatData();
    delay(5000);
    led1_ON();
    if( xbeeDM.pos>0 )
    {
        for(int f=0;f<xbeeDM.packet_finished[xbeeDM.pos-1]-
>data_length;f++)
        {
            data[f]=char(xbeeDM.packet_finished[xbeeDM.pos-1]-
>data[f]);
        }

        free(xbeeDM.packet_finished[xbeeDM.pos-1]);
        xbeeDM.packet_finished[xbeeDM.pos-1]=NULL;
    }
    xbeeDM.OFF();

    if ("w" == data)
    {
      step(200);
    }
    if ("s" == data)
    {
      step(-200);
    }
    if ("a" == data)
    {
      direccion(0);
      step(200);
      direccion(1);
    }
    if ("d" == data)
    {
      direccion(1);
      step(200);
      direccion(0);
    }
}
void step( int steps_to_move)
{
  int steps_left = abs(steps_to_move); //n° de steps que hay que
dar.
  if(steps_to_move > 0){direction = 0;}
  if(steps_to_move < 0){direction = 1;}
```

```
  while(steps_left > 0)
  {
       digitalWrite(DIGITAL8, Paso[step_number][0]);
       digitalWrite(DIGITAL7, Paso[step_number][1]);
       digitalWrite(DIGITAL6, Paso[step_number][2]);
       digitalWrite(DIGITAL5, Paso[step_number][3]);

       if(direction == 1)
       {
           step_number++;
       }else
       {
           step_number--;
       }
       //decremento de los steps que quedan
       steps_left--;
       if (steps_left != 0)
       {
         step_number=(step_number+4)%4;
       }
       delay(1);
  }
}

void direccion(int direction)
{
   //direccion 1: Derecha
   //direccion 0: Izquierda
   int steps_left2 = 4096/6;
   while(steps_left2 > 0){
     digitalWrite(DIGITAL4, Paso2[step_number2][ 0]);
     digitalWrite(DIGITAL3, Paso2[step_number2][ 1]);
     digitalWrite(DIGITAL2, Paso2[step_number2][ 2]);
     digitalWrite(DIGITAL1, Paso2[step_number2][ 3]);

     delay(1);
     if(direction == 1)
     {
       step_number2++;
     }else
     {
       step_number2--;
     }
     //decremento de los steps que quedan
     steps_left2--;
     if (steps_left2 != 0)
     {
       step_number2= ( step_number2 +8)%8;
     }
   }
}
void led0_ON() //función de encendido del LED 0
{
  Utils.setLED(LED0, LED_ON);
}
void led1_ON() //función de encendido del LED 1
{
  Utils.setLED(LED1, LED_ON);
}
void led0_OFF() //función de apagado del LED 0
{
```

```
  Utils.setLED(LED0, LED_OFF);
}
void led1_OFF() //función de apagado del LED 1
{
  Utils.setLED(LED1, LED_OFF);
}
```

Código Matlab

```
function varargout = frontend(varargin)
    % FRONTEND MATLAB code for frontend.fig
    %      FRONTEND, by itself, creates a new FRONTEND or raises
the existing
    %      singleton*.
    %
    %      H = FRONTEND returns the handle to a new FRONTEND or
the handle to
    %      the existing singleton*.
    %
    %      FRONTEND('CALLBACK',hObject,eventData,handles,...)
calls the local
    %      function named CALLBACK in FRONTEND.M with the given
input arguments.
    %
    %      FRONTEND('Property','Value',...) creates a new
FRONTEND or raises the
    %      existing singleton*.  Starting from the left,
property value pairs are
    %      applied to the GUI before frontend_OpeningFcn gets
called.  An
    %      unrecognized property name or invalid value makes
property application
    %      stop.  All inputs are passed to frontend_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
frontend

    % Last Modified by GUIDE v2.5 02-Jun-2016 19:47:27

    % Begin initialization code - DO NOT EDIT
    gui_Singleton = 1;
    gui_State = struct('gui_Name',       mfilename, ...
                       'gui_Singleton',  gui_Singleton, ...
                       'gui_OpeningFcn', @frontend_OpeningFcn,
...
                       'gui_OutputFcn',  @frontend_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 frontend is made visible.

function frontend_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 frontend (see
VARARGIN)

    % Choose default command line output for frontend
    handles.output = hObject;
    handles.before = 0;
    handles.s=0;
    handles.t=[];
    handles.i=0;
    handles.err=0;
    handles.conected=0;
    handles.temp=[];
    handles.pres=[];
    handles.humed=[];
    handles.lumn=[];
    handles.presenbak=[];
    handles.tempbak=[];
    handles.presbak=[];
    handles.humedbak=[];
    handles.lumnbak=[];
    handles.presenbak=[];
    handles.datos=[];

    set(handles.figure1,'Name','Controlador');
    if ispc
        disp('Dispositivo Windows')
        handles.s=serial('COM4');
        set(handles.s,'BaudRate',38400);
        handles.s.Terminator = 'CR';
        fopen(handles.s);
        handles.err=0;
        handles.conected=1;
    else
        disp('Incompatible con este sistema')
        handles.err=1;
        handles.conected=0;

    end
    % This sets up the initial plot - only do when we are
invisible
    % so window can get raised using frontend.
    if strcmp(get(hObject,'Visible'),'off')
        axes(handles.axes1);
        handles.plot=plot(0);
```

```
        title('Selecciona gráfica');
    end
    % Create a timer object to fire at 1/10 sec intervals
    % Specify function handles for its start and run callbacks
    handles.timer = timer(...
        'ExecutionMode', 'fixedRate', ...       % Run timer
repeatedly
        'Period', 10, ...                        % Initial
period is 1 sec.
        'TimerFcn', {@update_display,hObject}); % Specify
callback function
    % Initialize slider and its readout text field
    % Update handles structure
    guidata(hObject, handles);

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



% --- Outputs from this function are returned to the command
line.

function varargout = frontend_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;

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, ~, handles)
    % hObject    handle to popupmenu1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see
GUIDATA)
    % Hints: contents = get(hObject,'String') returns popupmenu1
contents as cell array
    %        contents{get(hObject,'Value')} returns selected
item from popupmenu1
    %axes(handles.axes1);
    %cla;
    popup_sel_index = get(handles.popupmenu1, 'Value');
    handles=guidata(hObject);

    guidata(hObject, handles);

% --- Executes during object creation, after setting all
properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to popupmenu1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
```

```
    % handles    empty - handles not created until after all
CreateFcns called
    % Hint: popupmenu 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
    set(hObject, 'String', {'Temperatura', 'Presión', 'Humedad',
'Luminosidad', 'Presencia'});


% --- Executes on button press in Up.
function Up_Callback(hObject, eventdata, handles)
    % hObject    handle to Up (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see
GUIDATA)
    guidata(hObject, handles);
    if ~handles.err && handles.conected
        hexString = dec2bin(119);
        fwrite(handles.s,hexString);
    else
        msgbox('No es plosible escribir en el puerto de serie');
    end

% --- Executes on button press in Down.
function Down_Callback(hObject, eventdata, handles)
    % hObject    handle to Down (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see
GUIDATA)
    guidata(hObject, handles);
    if ~handles.err && handles.conected
        hexString = dec2bin(115);
        fwrite(handles.s,hexString);
    else
        msgbox('No es plosible escribir en el puerto de serie');
    end

% --- Executes on button press in Left.
function Left_Callback(hObject, eventdata, handles)
    % hObject    handle to Left (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structuif ~handles.errre with handles and user
data (see GUIDATA)
    guidata(hObject, handles);
    if ~handles.err && handles.conected
        hexString = dec2bin(97);
        fwrite(handles.s,hexString);
    else
        msgbox('No es plosible escribir en el puerto de serie');
    end

% --- Executes on button press in Right.
function Right_Callback(hObject, eventdata, handles)
    % hObject    handle to Right (see GCBO)
```

```
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see
GUIDATA)
    guidata(hObject, handles);
    if ~handles.err && handles.conected
        hexString = dec2bin(100);
        fwrite(handles.s,char(hexString));
    else
        msgbox('No es plosible escribir en el puerto de serie');
    end

% --- Función de lectura del puerto de serie.


function update_display(hObject,eventdata,hfigure)
    % Timer timer1 callback, called each time timer iterates.
    handles=guidata(hfigure);
    popup_sel_index = get(handles.popupmenu1, 'Value');
    if handles.i == 0
        handles.plot=plot(0);
        set(handles.axes1,'YGrid','on');
        handles.title=title('');
    end
    if handles.conected
% --- Función de lectura del puerto de serie.
        if ~handles.err && handles.conected
            handles.t=[handles.t 10*handles.i];
            handles.i=handles.i + 1;
            dataArray=zeros(1,30);
            a=fread(handles.s,1);
            while a~=35 %35 es la # en ascii
                a=fread(handles.s,1);
            end
            i=1;
            while (a~=10) %10 es LF en ascii
                dataArray(i)=a;
                i=i+1;
                a=fread(handles.s,1);
            end
            dataArray(i)=0;
            dataArray=char(dataArray);

            handles.data=str2double(dataArray);
            handles.tempbak{end + 1}=dataArray(2:3);
            handles.presbak{end + 1}=dataArray(6:10);
            handles.humedbak{end + 1}=dataArray(13:17);
            handles.lumnbak{end + 1}=dataArray(20:24);
            handles.presenbak{end + 1}=dataArray(27:29);

            handles.temp=str2double(handles.tempbak);
            handles.pres=str2double(handles.presbak);
            handles.humed=str2double(handles.humedbak);
            handles.lumn=str2double(handles.lumnbak);
            handles.presen=str2double(handles.presenbak);
        end
        switch popup_sel_index
            case 1
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Temperatura');
                    set(handles.plot,'LineWidth',2.0);
```

```
                    set(handles.plot,'Color','y');
                    handles.before=1;
                end
                set(handles.plot,'Ydata',handles.temp);
                set(handles.plot,'Xdata',handles.t);
            case 2
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Presión');
                    set(handles.plot,'LineWidth',2.0);
                    set(handles.plot,'Color','k');
                    handles.before=2;
                end
                set(handles.plot,'Ydata',handles.pres);
                set(handles.plot,'Xdata',handles.t);
            case 3
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Humedad');
                    set(handles.plot,'LineWidth',2.0);
                    set(handles.plot,'Color','b');
                    handles.before=3;
                end
                set(handles.plot,'Ydata',handles.humed);
                set(handles.plot,'Xdata',handles.t);
            case 4
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Luminosidad');
                    set(handles.plot,'LineWidth',2.0);
                    set(handles.plot,'Color','c');
                    handles.before=4;
                end
                set(handles.plot,'Ydata',handles.lumn);
                set(handles.plot,'Xdata',handles.t);
            case 5
                if handles.before~=popup_sel_index
                    title(handles.axes1,'Presencia');
                    set(handles.plot,'LineWidth',2.0);
                    set(handles.plot,'Color','g');
                    handles.before=5;
                end
                set(handles.plot,'Ydata',handles.presen);
                set(handles.plot,'Xdata',handles.t);
        end
    end
    guidata(hfigure,handles);

% --------------------------------------------------------------------
------
function Exit_Callback(hObject, eventdata, handles)
    % hObject    handle to Exit (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see
GUIDATA)
    selection = questdlg(['Close ' get(handles.figure1,'Name')
'?'],...
                         ['Close ' get(handles.figure1,'Name')
'...'],...
                         'Yes','No','Yes');
    if strcmp(selection,'No')
        return;
    end
```

```
    if strcmp(get(handles.timer, 'Running'), 'on')
        stop(handles.timer);
    end
    if handles.conected
        delete(handles.s);
    end
    delete(handles.figure1)


% --------------------------------------------------------------------
------
function Muestrear_Callback(hObject, eventdata, handles)
    % hObject    handle to Muestrear (see GCBO)
    % eventdata  reserved - to be defined in a future version of
MATLAB
    % handles    structure with handles and user data (see
GUIDATA)
    if strcmp(get(handles.timer, 'Running'), 'off')
        start(handles.timer);
          end
```