Sistema de Alarma de Incendios basado en una red de sensores.

Pascual Martínez Pérez
Ingeniería Técnica Telecomunicaciones

Consultor: Jordi BécaresFerrés.

14-Junio-2012

Resumen.

Para todo ingeniero la máxima dificultad a la hora de hacer un diseño es la conjunción de varias tecnologías en un único producto. En nuestro caso particular como ingenieros de telecomunicaciones a la hora de diseñar un producto, debemos complementar el hardware con el software y viceversa. Ya que uno sin el otro no son razón de ser.

La motivación para escoger esta área ha sido la visión de nuestra vida cotidiana en la que estamos rodeados de sistemas embebidos. Desde un reproductor mp3, hasta un tablet, pasando por un televisor, una lavadora, un sistema de inyección…. En todos encontramos la pieza fundamental para su funcionamiento, un microcontrolador. También encontramos circuitos auxiliares, como puertos de entrada /salida, memoria, pantallas visualizadoras… y un software diseñado a medida y a la medida que lo hace funcionar.

En este TFC desarrollamos un sistema de detección de incendios formado por una red de sensores. Donde cada nodo de la red lo vemos como un sistema embebido. Estudiamos el dispositivo a nivel de hardware llamado mota e implementamos un software para que este sistema funcione como un sistema autónomo, en concreto para la detección de incendios. Hacemos que la unión de su hardware y software sea muy estrecha, esto implica que sea un reto su desarrollo.

Esto para mi ha supuesto para mí una labor de investigación y enriquecimiento personal con la cual he conseguido los resultados propuestos a la hora de escoger esta área.

Índice de Contenidos

Índice de Figuras.

Capítulo 1 Introducción.

1. Justificación

Las Redes de Sensores Inalámbricas son redes de pequeños dispositivos capaces de comunicarse de forma inalámbrica. Siendo una de las tecnologías más investigadas en la actualidad. Son de gran aplicación en variados escenarios, como el control de procesos industriales, seguridad en centros comerciales, edificios habitacionales, campos de cultivo, zonas propensas a desastres naturales, seguimiento de pacientes en zonas hospitalarias, seguridad en el transporte… Donde puede ser muy complicado la instalación y el mantenimiento de una red convencional médiate cableado. Siendo su bajos costos y reducido consumo de potencia, el completo ideal para este tipo de escenarios.

En la actualidad se apunta hacia una proliferación de redes de sensores inteligentes, capaces de manejar grandes cantidades de información, para que con sus despliegues podamos obtener datos de nuestro entorno en tiempo real y podamos explotar esta información dando lugar a sistemas como por ejemplo: prevención de desastres naturales, control de tareas domésticas, sistemas de control de tráfico, sistemas de seguridad…

Dentro del ámbito de la seguridad, se presenta este proyecto que pretende ser una aportación más hacia los sistemas de alarma. Desplegando una red de sensores con la capacidad de detección de incendios y envío de alarmas a una consola central para su registro y posterior tratamiento. Su ámbito de aplicación puede ser muy extenso podemos pensar en cualquier escenario donde sea necesaria la prevención de incendios. Desde un paraje de montaña, hasta un hospital, centro comercial etc…

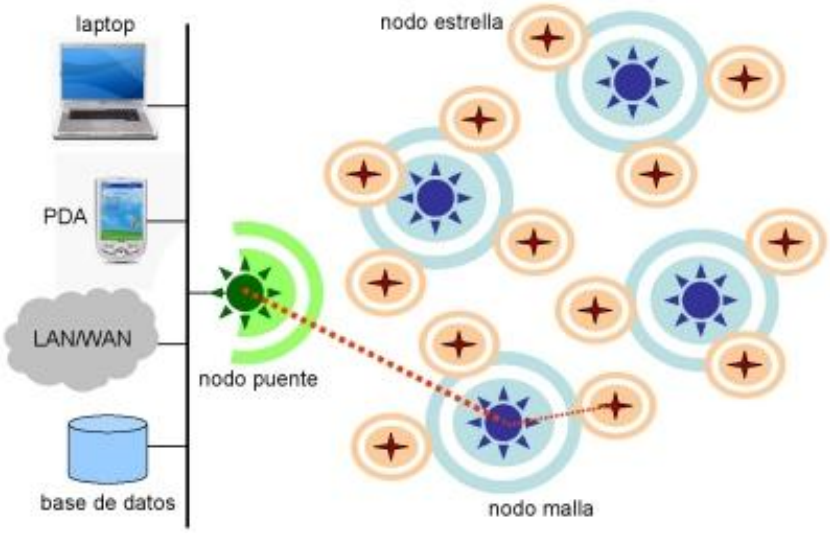


Ilustración 1 Red De Sensores

2. Descripción

El proyecto consiste en el diseño de una red de sensores para la detección de incendios.
En nuestro caso disponemos de dos sensores. Un sensor que actúa como receptor de parámetros del entorno (temperatura y luminosidad), llamado "mota remota". Y otro sensor llamado "mota estación base", que conectado a un ordenador mediante conexión usb, hace las funciones de repetidor de mensajes.

Este sensor mota "remota" mide la temperatura de su entorno y el estado de su batería, a intervalos regulares. Dicho sensor, también comprueba que la temperatura, no sea superior a una temperatura dada de referencia. Cuando la temperatura del entorno es superior a la de referencia, este activa una indicación luminosa en el sensor. Con el estado de la batería procedemos de igual forma, se mide regularmente su estado a intervalos regulares. Cuando el nivel de la batería sea inferior al de referencia, este lo indica de forma luminosa. También disponemos de un pulsador de emergencia que al pulsarlo, y nos lo indica visualmente.

Tanto los datos obtenidos del entorno, así como las alarmas de temperatura, batería y activación manual del pulsador, son enviados al sensor "mota estación base"vía radio. Este lo envía al ordenador mediante conexión usb. Allí, una aplicación recoge dichos datos. Y los visualiza de una forma sencilla.

También podemos interactuar con el sistema pudiendo cambiar los valores de referencia tanto de temperatura, como de nivel de batería del sensor, hacer un rearme del sistema (reset de alarmas), cambiar los ciclos de lectura tanto de la batería como los de temperatura. Todos estos cambios se realizan mediante el intercambio de mensajes entre la "mota remota" y la "mota estación base" y son generados por la aplicación consola. Finalmente, podemos ver todo lo que ocurre en la aplicación mediante una venta de log que registra el intercambio de mensajes de una forma amigable.

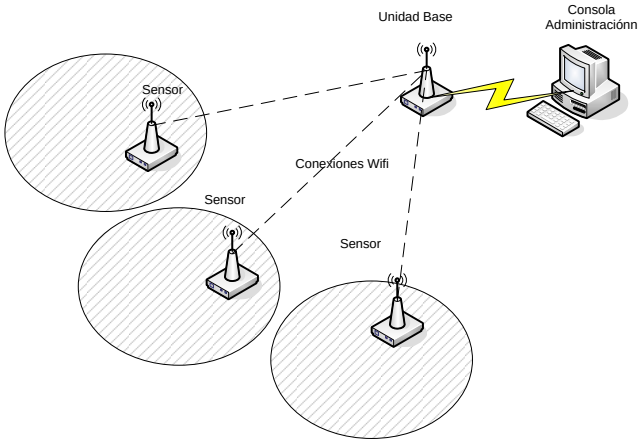


Ilustración 2 Descripción proyecto

3. Objetivos

Los diferentes objetivos se han logrado a medida que el proyecto iba avanzando y se iban adquiriendo nuevos conocimientos. Dentro del mismo se habían planteado los siguientes objetivos con respecto a los dos elementos mencionados anteriormente y a la aplicación de consola que nos permite manejar el sistema.

Objetivos Básicos

Mota Remota

- Medición de temperatura del entorno de forma periódica.
- Envío de la medida de temperatura a la estación base.
- Comprobación del estado de alarma
- Indicación visual y auditiva del estado de alarma
- Envió del estado de alarma a la estación base.
- Activación de forma manual de la situación de alarma.
- Comprobación periódica del estado de la batería.
- Envió del estado de la batería a la estación base
- Notificar estado de nivel bajo de batería.

Mota Estación base

- Recepción de alarmas de los sensores.
- Recepción de los mensajes de estado de los sensores.
- Recepción de los mensajes de condiciones de entorno
- Envío a consola de alarmas detectadas.
- Envío a consola del estado de los sensores
- Envió a consola de las condiciones del entorno detectadas por los sensores.
- Indicación del estado de alarma.

Consola Administración.

- Configurar parámetros aplicación, (temperatura de referencia, intervalo de comprobación...)
- Visualizar los mensajes de la estación base.
- Rearme del sistema.

Objetivos Complementarios

- La realización de un entorno a nivel de consola de administración (Gui, web ..) para que de una forma gráfica podamos comprobar el estado de los sensores , ver alarmas detectadas y configurar el sistema.

4. Requerimientos

Una vez definidos los objetivos a desarrollar los convertimos a requerimientos que realmente serán las funcionalidades que nos proporcionara la red de sensores.

Detectar incendios

- Detección por temperatura. Cuando la temperatura supera un umbral (TEMP_ALARM) el nodo debe enviar la alarma.
- El nivel de alarma debe poder configurar desde la aplicación del usuario.
- El sistema tomará un dato del sensor cada N segundos para comprobar el umbral. N debe ser configurable por el usuario

Notificar automáticamente la alarma de forma remota.

- Una vez se detectada la alarma esta se enviará de forma automática.
- La transmisión de datos será a través de una comunicación inalámbrica.

Notificar automáticamente la alarma de forma local.

- Se informará de la alarma de forma local, mediante una señal visual (led / s)

Sistema manual para activar la alarma

- El nodo debe proveer de un botón de emergencia para activar la alarma de forma manual

Transmisión de alarmas sin pérdidas

- No se puede perder ninguna alarma de incendio
- La comunicación se realizará con ACK

Reconocimiento de alarmas desde la aplicación de usuario

- La aplicación debe mostrar por pantalla cuando ha habido una alarma.
- Debe mostrar:
    - Tipo de alarma (manual o automática)
    - ¿Qué sensor ha detectado la alarma y qué temperatura detecta.
    - Hora en que se ha producido.
- El usuario debe poder confirmar la recepción de alarma

Monitorizar la batería de los nodos de la red

- Cada nodo enviará su batería de forma periódica cada L segundos
- debe ser configurable desde la aplicación
- Cuando el nodo detecte que tiene la batería critica, es decir por debajo de un umbral(BAT_LVL_ALARM), enviará una alarma notificando de su estado
- Este mensaje se utilizará para controlar que el nodo funciona correctamente. Si el PC hace más de L según que no recibe un mensaje del nodo, mostrará una alarma por pantalla.

Mostrar los diferentes estados del sistema en el nodo:

- Sensor apagado. De forma periódica puede encender un led, conforme el nodo puede funcionarcorrectamente.
- No alarma.
- Alarma detectada
- Alarma recibida en Estación Central
- Alarma reconocida
- Nivel batería bajo
- Fuera de cobertura

Sistema de protección de caídas.

- El nodo debe utilizar el Watchdog para evitar quedarse colgado.
- Debe recuperar la configuración de forma automática

Sistema de debug de la aplicación.

- El usuario debe poder ver las temperaturas de los diferentes sensores.
- El usuario puede solicitar recibir los datos cada M segundos. M debe ser configurable..

Prueba de cobertura

- Proporcionar un sistema para comprobar si donde se instalará la el nodo hay cobertura con elnodo base.

Sistema de activación

- El sensor no estará siempre encendido. Para iniciar el sensor, se hará uso del sensor de efecto Hall. Para encender el sensor tendrá que pasar un imán 2 veces seguidas por encima de éste. Para apagarlo se tendrá que pasar 4 veces seguidas por encima.

Interfaz de usuario

- Debe ser simple
- Autoinstalable y autoejecutable
- Script de instalación
- Contenga todas las librerías necesarias
- Instrucciones de instalación
- Contener un menú de ayuda (help con los pedidos)

5. Enfoque

Se ha acometido el desarrollo, dividiendo los objetivos y requerimientos a desarrollar en 3 fases.

En la fase 0 hemos instalado todo el sistema de desarrollo necesario para la creación de software. Solucionando todo los problemas relacionados con la generación de código.También ha sido la primera toma de contacto con las motas y los primeros programas de test.

En la fase 1, nos hemos centrado en el uso de la mota como sensor. Midiendo la temperatura del entorno, su nivel de batería, la detección de la situación de alarma yel envío de datos a la consola mediante comunicación serie (usb).

En la fase 2 hemos desarrollado la "mota remota" completamente, básicamente ha sido cambiar la comunicación serie por la comunicación via radio. Y la "mota estación base" que nos ha servido de repetidor entre la consola y la "mota remota".

En todas las fases se ha ido complementando con la creación de la aplicación consola, dándole funcionalidades a medida que avanzaba el proyecto.

La fase 3 corresponde con la generación de la documentación y presentación.

6. Planificación

La planificación inicial propuesta ha sido la siguiente:

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**TFC**	**77 días**	**mié 29/02/12**	**jue 14/06/12**
2	**Fase 0**	**15 días**	**mié 29/02/12**	**mar 20/03/12**
3	Propuesta de Trabajo	4 días	mié 29/02/12	lun 05/03/12
4	Plan de trabajo	11 días	mar 06/03/12	mar 20/03/12
5	Entrega PAC 1	1 día	mar 20/03/12	mar 20/03/12
6	Instalacion entorno Desarrollo	6 días	lun 05/03/12	lun 12/03/12
7	Pruebas con aplicaciones Ejemplo	6 días	mar 13/03/12	mar 20/03/12
8	**Fase1**	**25 días**	**mié 21/03/12**	**mar 24/04/12**
9	Comunicación de la estación base con la	5 días	mié 21/03/12	mar 27/03/12
10	Lectura de los sensores	5 días	mié 28/03/12	mar 03/04/12
11	Detección del estado de alarma	5 días	mié 04/04/12	mar 10/04/12
12	Envío de datos a la consola de administra	5 días	mié 11/04/12	mar 17/04/12
13	Visualización de datos en la consola de a	5 días	mié 18/04/12	mar 24/04/12
14	Entrega PAC2	1 día	mar 24/04/12	mar 24/04/12
15	**Fase2**	**20 días**	**mié 25/04/12**	**mar 22/05/12**
16	Comunicación del sensor remoto con la es	4 días	mié 25/04/12	lun 30/04/12
17	Lectura de los sensores remotos	4 días	mar 01/05/12	vie 04/05/12
18	Detección del estado de alarma	4 días	lun 07/05/12	jue 10/05/12
19	Envío de datos a la estación base	4 días	vie 11/05/12	mié 16/05/12
20	Visualización de datos en la consola de a	4 días	jue 17/05/12	mar 22/05/12
21	Entrega PAC 3	1 día	mar 22/05/12	mar 22/05/12
22	**Fase3**	**48 días**	**mar 10/04/12**	**jue 14/06/12**
23	Memoria	46 días	mar 10/04/12	mar 12/06/12
24	Presentacion	6 días	jue 07/06/12	jue 14/06/12
25	Entrega Final	1 día	jue 14/06/12	jue 14/06/12

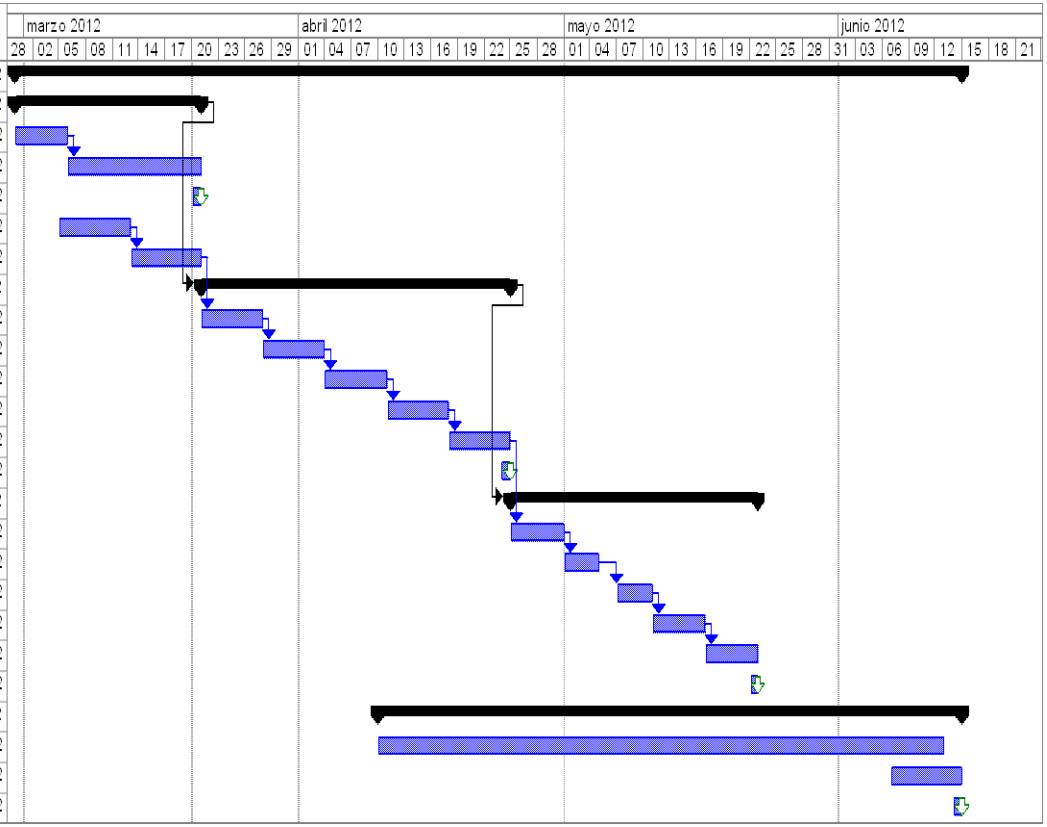


Ilustración 3 Planificación Inicial

La Fase más problemática ha sido la fase 0. Todo lo concerniente a la instalación del entorno de desarrollo. Debido a los diversos errores de configuración, de versiones de software. Hasta incluso en la Fase 1 tuvimos que volver a reinstalar nuevamente el sistema operativo y preparar el ordenador con el entorno de desarrollo.

Durante la Fase 1 ha sido prácticamente la toma de contacto con la mota y ver como trabajaban las aplicaciones de test. Para una vez comprendido, aplicarlo al funcionamiento completo desde la creación de los componentes mediante nesC compilarlos para TinyOs y transferirlo a las motas. En esta fase nos hemos centrado en aprender los diversos elementos de la mota y como se acceden a ellos, así como la comunicación usb. Finalmente hemos desarrollado la primera entrega de código. Un código funcional, pero todo desarrollado en un solo componente.

La Fase 2, ha consistido en aplicar lo aprendido en la Fase 1 y aplicar la descomposición en componentes cada uno especializado en una tarea específica y como se comunican esos componentes mediante interfaces e implementado los métodos para que se pueda realizar esa comunicación. En esta fase también, hemos desarrollado la comunicación vía radio. Finalmente se ha desarrollado una aplicación funcional, pero incompleta en algunos aspectos, y con alguna carencia de requisitos.

La Fase 3, ha consistido en la generación de documentación.

Finalmente veo que con el código obtenido en esta fase a pesar de ser la fase 3, tendría que haberlo conseguido al final de la fase1. Así podría haber testeado mejor la aplicación y terminar algún requisito que me falta.

Cronograma final del proyecto. La mayor diferencia con respecto al planteado inicialmente son los cambios refertes a la fase 2.

Id	Nombre de tarea	Duración	Comienzo	Fin
1	TFC	77 días	mié 29/02/12	jue 14/06/12
2	Fase 0	15 días	mié 29/02/12	mar 20/03/12
3	Propuesta de Trabajo	4 días	mié 29/02/12	lun 05/03/12
4	Plan de trabajo	11 días	mar 06/03/12	mar 20/03/12
5	Entrega PAC 1	1 día	mar 20/03/12	mar 20/03/12
6	Instalacion entorno Desarrollo	6 días	lun 05/03/12	lun 12/03/12
7	Pruebas con aplicaciones Ejemplo	6 días	mar 13/03/12	mar 20/03/12
8	Fase1	25 días	mié 21/03/12	mar 24/04/12
9	Comunicación de la estación base con la	5 días	mié 21/03/12	mar 27/03/12
10	Lectura de los sensores	5 días	mié 28/03/12	mar 03/04/12
11	Detección del estado de alarma	5 días	mié 04/04/12	mar 10/04/12
12	Envío de datos a la consola de administra	5 días	mié 11/04/12	mar 17/04/12
13	Visualización de datos en la consola de a	5 días	mié 18/04/12	mar 24/04/12
14	Entrega PAC2	1 día	mar 24/04/12	mar 24/04/12
15	Fase2	20 días	mié 25/04/12	mar 22/05/12
16	Comunicación del sensor remoto con la es	4 días	mié 25/04/12	lun 30/04/12
17	Crear Componentes	4 días	mar 01/05/12	vie 04/05/12
18	Comunicacion esntre componenetes	4 días	lun 07/05/12	jue 10/05/12
19	Aplicación consola	4 días	vie 11/05/12	mié 16/05/12
20	Comunicación con Consola	4 días	jue 17/05/12	mar 22/05/12
21	Entrega PAC 3	1 día	mar 22/05/12	mar 22/05/12
22	Fase3	48 días	mar 10/04/12	jue 14/06/12
23	Memoria	46 días	mar 10/04/12	mar 12/06/12
24	Presentacion	6 días	jue 07/06/12	jue 14/06/12
25	Entrega Final	1 día	jue 14/06/12	jue 14/06/12

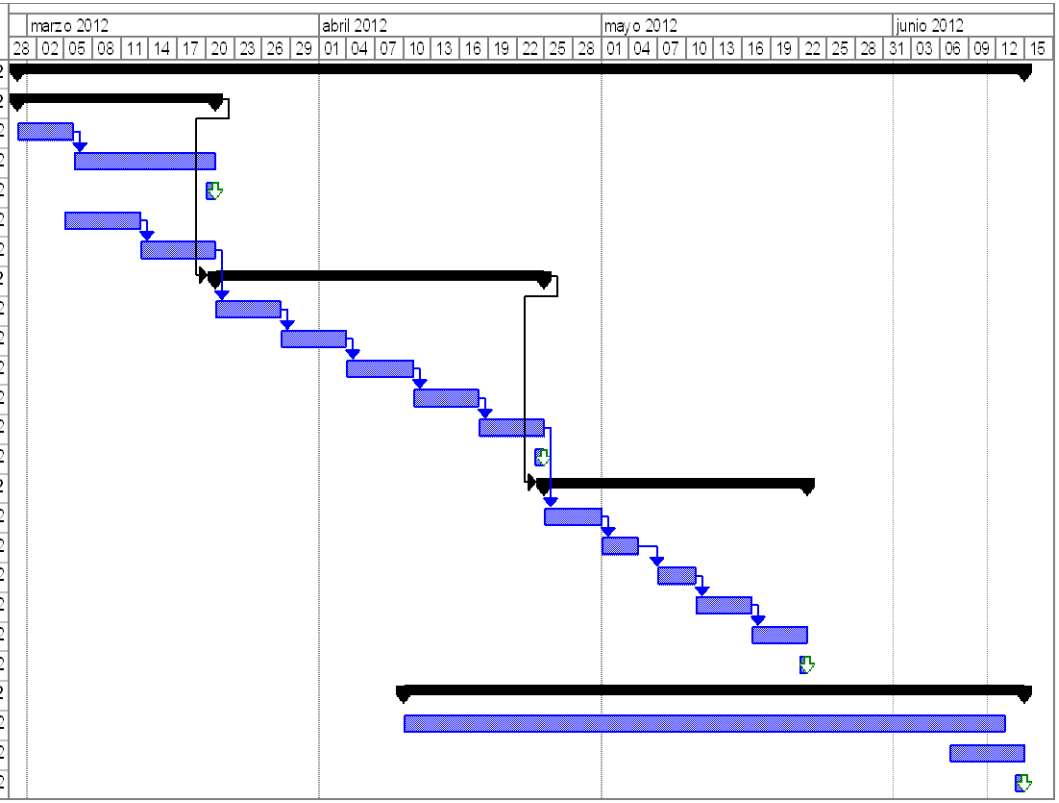


Ilustración 4 Cronograma final

7. Recursos Empleados

Disponemos principalmente de dos tipos de recursos para la realización del proyecto, Hardware y software.

Hardware.

- Dos motas modelos ATZB-24-A2, basadas en microcontroladorAtmel.
- Ordenador para el desarrollo, basado en tecnología intelCore 2 Duo y con 2 GB de Ram.

Software.

Todo el software necesario para el desarrollo residirá en este ordenador.

- Entorno de desarrollo netBeans.
- Java 1.6
- Librerias de TinyOS
- Utilidad meshprog para programar las motas mediante usb.
- Sistema Operativo desarrollo Ubuntu 10.04

8. Productos Obtenidos

Del desarrollo del proyecto hemos obtenido tres productos. La aplicación consola un entorno gráfico, que nos permite registrar y monitorizar a las “motas remotas” que actúan como sensores. Esta aplicación consola está implementada en java, se ejecuta en un ordenador. Los otros productos son los referentes a la “motas”. Uno de ellos es el código fuente en nesC de la “mota remota” que actúa como sensor midiendo los parámetros del entorno. El otro es el código fuente es de la mota que hace de “puente” entre el ordenador y la mota remota, llamada “mota estación base”, también desarrollado en nesC.

Capítulo 2 Antecedentes.

1. Estudio del arte

Las Redes Inalámbricas de Sensores (Wireless Sensor Networks) representan hoy en día una de las tecnologías más novedosas. Existe un amplio abanico de aplicaciones en las que se está investigando, pero las posibilidades que nos ofrecen este tipo de redes es enorme. Contamos no sólo con todas las aplicaciones que hasta el momento se pensaban para una red de sensores tradicionales, sino también, con aplicaciones donde antes era impensable operar por problemas de cableado, consumo, etc. Por ejemplo: control de microclimas, controles médicos, controles de presión, edificios inteligentes, aplicaciones militares.

Ilustración 5 Aplicaciones Redes Sensores

Las redes inalámbricas de sensores son sistemas embebidos hardware/software basados en estándares WLAN (Wireless Local Area Network), tipo IEEE 802.15.4, una versión reducida del estándar de comunicaciones WIFI de 2,4 GHz, o en redes propietarias. Un sistema operativo muy utilizado entre los diseñadores de WSN es el TinyOS por presentar un código fuente libre. Seguramente, el mayor reto que presentan estas redes es cómo conseguir programar, de forma eficiente y fiable, cientos de pequeños nodos que pueden coexistir en un sistema. Por ejemplo, Intel ha construido una familia de dispositivos de redes de sensores inalámbricos de bajo consumo integrando componentes comerciales en una plataforma denominada “mote”. Estos “motes” se han usado tanto para evaluar algoritmos como para monitorización ambiental y rastreo de objetos.

En esencia, un mote o nodo de una red de sensores consiste físicamente de un sistema embebido hardware/software compuesto por un microprocesador, un dispositivo de almacenamiento de datos, sensores, convertidores analógico-digital, un transmisor de radio, controladores y una fuente de energía.

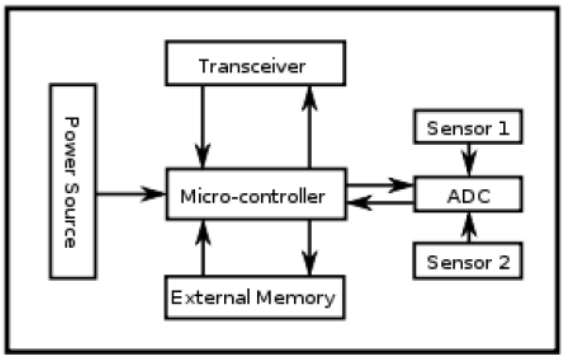


Ilustración 6 Arquitectura Mota

Los avances relacionados con las nuevas generaciones de sensores, es que estos últimos más son inteligentes y con un grado de conectividad muy elevado, permitiendo interacciones múltiples. Una característica más de los sensores inalámbricas tienen que cumplir tiene que ver con el consumo de energía. En los últimos años el consumo de energía ocupa un lugar especial en el proceso de diseño. En el caso de este tipo de redes compuestas de varias decenas de sistemas autónomos, el consumo de energía es clave para su desarrollo y aceptación en el mercado.

Antes de diseñar e instalar cualquier sistema de sensores, es necesario entender de forma detallada el entorno físico. Los sensores pueden utilizarse para monitorizar condiciones y movimientos de animales salvajes o de plantas en su propio hábitat, donde se requiere una alteración mínima. También pueden monitorizar la calidad del aire, la polución del entorno, fuegos incontrolados, o cualquier otro desastre producido por el hombre. Adicionalmente, los sensores también pueden monitorizar posibles desastres biológicos o químicos y avisar con antelación. La monitorización de terremotos es otra posible área. Un ejemplo a muy gran escala es el Sistema para la vigilancia del Amazonas (SIVAM) que proporciona monitorización ambiental, monitorización de tráfico de drogas y control del tráfico aéreo en la cuenca del Amazonas.

Los algoritmos de enrutamiento y reconfiguración de este tipo de redes tienen que ser lo suficientemente flexibles como para poder desplegar una red compuesta por varias decenas de nodos. Además, este tipo de redes se caracteriza por su estructura dinámica: algunos nodos pueden desaparecer, bien porque sean retirados o bien por que exista un aislamiento radioeléctrico temporal, mientras que se pueden incorporar otros a la red. Este tipo de operación no sólo no debe llevarse a cabo de manera que no perturbe la operación de la red, sino que también debe de asegurarse que todo nodo conectado a la red esté autorizado a formar parte de la red y que la información a transmitir viaje debidamente protegida. Las redes de sensores se diseñan para transmitir los datos desde un matriz de sensores a un servidor que sirve de centralizador. No usan necesariamente un único camino para enviar los datos a través de la red. Los elementos del sistema tomarán decisiones sobre qué datos pasar para minimizar la energía consumida y maximizar el contenido de la información

Las redes de sensores van a expandir de forma significativa la red Internet existente hacia los espacios físicos. El procesamiento de datos, almacenamiento, transporte, consultas, así como la interacción entre TCP/IP y las redes de sensores presentan interesantes retos en la investigación, que deberán ser tratados desde una perspectiva multidisciplinar. Los últimos avances en tecnologías de redes inalámbricas y miniaturización hacen posible monitorizar de forma realista el entorno natural. En el avance tecnológico de las redes de sensores se ven involucradas tres áreas diferentes de investigación: la percepción, las comunicaciones y la computación

2. Estudio de mercado

La compañía Crossbow Technology (15) es el fabricante de las plataformas Imote2, Mica2, Mica2dot, Micaz, Telosb e IRIS. Esta empresa nació de un grupo de investigadores de la UCB (University of California, Berkley). Los dispositivos Micaz, Mica2 y Mica2dot (15) son unos de los modelos más comunes para desarrollar redes de sensores y conforman la tercera generación de esta familia de nodos. El Mica2 es el nodo principal del cual derivan los otros dos.

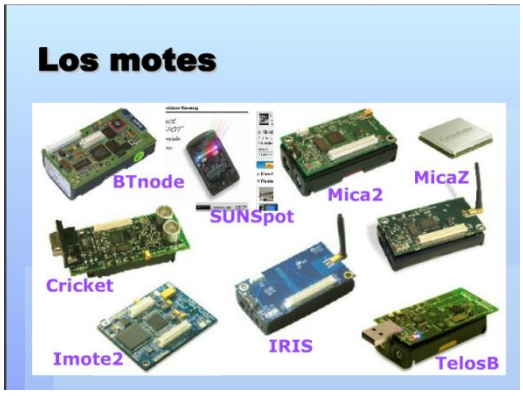


Ilustración 7 Diferentes Tipos de Motas

Imote (Intel Mote) (15) es la plataforma desarrollada por Intel Corporation en colaboración con la Universidad de Berkley, UCB, para redes de sensores. El objetivo era conseguir una plataforma que ofreciera mejores prestaciones que las existentes con el objetivo de abordar aspectos como la medida de vibraciones o aceleración para usos industriales que comportan una cantidad de información mayor que el caso de las medidas convencionales de humedad, temperatura, presión entre otras. Por este motivo sus prestaciones de memoria y procesado son muy superiores a las ofrecidas por otros dispositivos como los Mica, Telosb o similares.

La plataforma IRIS(15) comercializada también por Crossbow Technology incorpora un chip radio que cumple el IEEE 802.15.4 y permite un alcance superior al de los nodos Mica que puede llegar a ser de hasta 500 metros en situaciones de visión directa.

Adjuntamos un sumario gráfico de características.

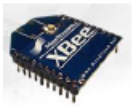
CRITERION	MOTE MICA2	MICAZ	IMOTE2	XBEE 802.15.4 OEM RF Modules v1.xAx	XBEE-PRO 802.15.4 OEM RF Modules v1.xAx	XBEE ZigBee OEM RF Modules v8.x17 Beta	XBEE-PRO ZigBee OEM RF Modules v8.x17 Beta
Photo							
Standard	Proprietary of Xbow	IEEE 802.15.4 (CC2420)	Zigbee	IEEE 802.15.4	IEEE 802.15.4	ZigBee (beta)	ZigBee (beta)
	0	1	3	1	1	2	2
Logical Topology	Star Mesh Hybrid	Star Mesh	Star Tree	Star Tree	Star Tree	Mesh, Star	Mesh, Star
	2.25	2.25	1.25	1.25	1.25	2.25	2.25
Max. number of nodes	max. 65,533 nodes	max. 65,533 nodes	max.65,533 nodes	max. 65,533 nodes	max. 65,533 nodes	46,656 nodes, due to limitations in the network structure	46,656 nodes, due to limitations in the network structure
	2	2	2	2	2	1	1
Price	107.99 €	90.16 €	275.73 €	13.39 €	22.54 €	13.39 €	22.54 €
	0	1	0	2	2	2	2
Maximum data load per packet	248 bytes	102 bytes	100 bytes	100 bytes	100 bytes	72 bytes	72 bytes
	3	2	2	2	2	1	1
Interfaces / Buses	Analog Inputs, Digital I/O, I2C, SPI, UART	Analog Inputs, Digital I/O, I2C, SPI, UART	Opios 2x, SPI 3x, UART, I2C, SDIO, USB host, USB client,AC´97,Camera	Analog Inputs (7), Digital I/O (9), UART	Analog Inputs (7), Digital I/O (9), UART	UART	UART
	3	3	3	3	3	1	1
Security	Identifiers for motes, remote border security, occupancy monitoring and acknowledgement protocol…[Cullinan05]	802.15.4 Default Security Protocol MAC encryption (AES-128).	802.15.4 Default Security Protocol MAC encryption (AES-128).	128-bit AES encryption	128-bit AES encryption	128-bit AES encryption	128-bit AES encryption
	2	3	3	3	3	3	3
RF Data Rate	38.4 kbps	250 kbps	250 kbps	250 kbps	250 kbps	250 kbps	250 kbps
	0	3	3	3	3	3	3
TOTAL	*12,25*	*17,25*	*17,25*	*17,25*	*17,25*	*15,25*	*15,25*

Ilustración 8 Características generales motas

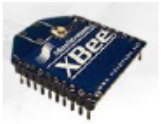
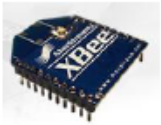
CRITERION	MOTE MICA2	MICAZ	IMOTE2	XBEE (802.15.4)	XBEE-PRO (802.15.4)	XBEE (Zigbee)	XBEE-PRO (Zigbee)
Photo							
Transmit Power (dBm)	5, 0, -10, -20	0, -1, -3, -5, -7, -10, -15, -25	0, -1, -3, -5, -7, -10, -15, -25	-10, -6, -4, -2, 0	10, 12, 14, 16, 1	-10, -6, -4, -2, 0	10, 12, 14, 16, 18
Operating Current	(@ 3.0 V & 5 dBm) Tx: 27 mA Rx/Idle: 10 mA CPU current: 8 mA	(@3.0 V & 0 dBm) Tx: 17.4 mA Rx/Idle: 19.7 mA CPU current: 8 mA	(@4.5 V & 0 dBm) Tx: 44 mA Rx/Idle: 31 mA CPU current: 31 mA @13 MHz. It is possible to scale it till 416 MHz.	(@3.3 V & 0 dBm) Transmit: 45 mA Receive/Idle: 50 mA Processor current: 6.5 mA	(@ 3.3 V & 18 dBm) Tx: 215 mA Rx/Idle: 55 mA CPU current: 6.5 mA	(@ 3.3 V & 0 dBm) Tx:: 45 mA Rx/Idle: 50 mA CPU current: 6.5 mA	(@ 3.3 V & 18 dBm) Tx: 215 mA Rx/Idle: 55 mA CPU current: 6.5 mA
Sleep Current	(@ 3.0 V) < 1 µA CPU current: < 15 µA	(@ 3.0 V) 1 µA CPU current < 15 µA	(@ 4.5 V) at 13 Mhz 390 µA CPU current: 290 µA	(@ 3.0 V) < 1 µA CPU current < 70 µA	(@ 3.0 V) < 1 µA CPU current < 70 µA	(@ 3.0 V) < 1 µA CPU current < 70 µA	(@ 3.0 V) < 1 µA CPU current < 70 µA
Receiver sensitivity	-98 dBm	-94 dBm	-94 dBm	-92 dBm	-100 dBm	-92 dBm	-100 dBm
Battery lifetime average power consumption[2]	3.244 years 0.0665 mAh.	2.34 years 0.0922 mAh	0.457 years 0.4721 mAh	1.563 years 0.1381 mAh	1.26 years 0.1712 mAh	1.563 years 0.1381 mAh	1.26 years 0.1712 mAh.
	3	2	0	1	1	1	1
Supply Voltage	3.0 V	3.0 V	4.5 V	3.4 V	3.4 V	3.4 V	3.4 V
	2	2	0	1	1	1	1
Indoor Distance Range	Up to 150 m with line of sight	Indoor/Urban: up to 30 m	30 m with line of sight	Indoor/Urban: up to 30 m	Indoor/Urban: up to 100 m	Indoor/Urban: up to 30 m	Indoor/Urban: up to 100 m
	2	0	0	0	2	0	2
Outdoor Distance Range	Up to 150 m with line of sight	Outdoor: Up to 100 m	30 m with line of sight	Outdoor: Up to 100 m	Outdoor: Up to 1.5 km	Outdoor: Up to 100 m	Outdoor: Up to 1.5 km
	2	1	0	1	3	1	3
TOTAL	9,00	5,00	0,00	3,00	7,00	3,00	7,00

Ilustración 9 Requisitos Energía

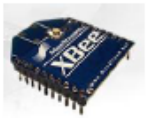
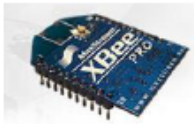
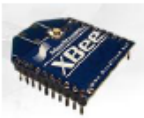
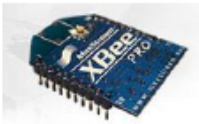
CRITERION		MOTE MICA2	MICAZ	IMOTE2	XBEE (802.15.4)	XBEE-PRO (802.15.4)	XBEE (Zigbee)	XBEE-PRO (Zigbee)
Photo								
Learning curve	Standard Language (2)	nesC language	nesC language	c# micro framework	C	C	C	C
	Standard Environment (1)	linux-like	linux-like	Visual Studio	CodeWarrior	CodeWarrior	CodeWarrior	CodeWarrior
Programming environment	Programming IDE (1.5)	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Emulator (0.75)	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Debugging utilities (0.50)	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Simulator (0.25)	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Language capabilities	Simple I/O access to the hardware (1)	Yes	Yes	Yes	No	No	No	No
	Large set of external libraries (1)	Yes	Yes	No	No	No	No	No
	Automatic memory management (0.5)	No	No	Yes	No	No	No	No
	Error management with exceptions (0.25)	No	No	Yes	No	No	No	No
	Object oriented programming paradigm (0.25)	No	No	Yes	No	No	No	No
TOTAL		6	6	8	5	5	5	5

Ilustración 10 Características de Programación

3. Motas Utilizadas

Las motas utilizadas en este proyecto son un sistema embebido completo donde disponemos de diversos tipos de sensores, temperatura, luminosidad, efecto Hall, un compartimento para dos pilas AA de 1,5 voltios, un conector USB y el núcleo principal está formado por un módulo ATZB-24-A2, que contiene el microcontrolador 'Atmega 1281' más el transmisor 'AT86RF230'. Presentamos a continuación una fotografía de la mota llamada COU24, con sus partes indicadas:

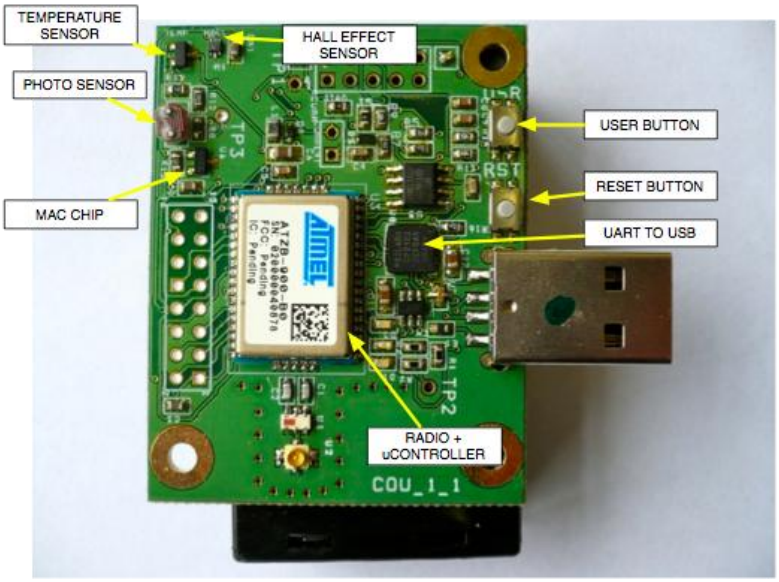


Ilustración 11 Mota Utilizada

El módulo principal de la mota contiene el microcontrolador y el transmisor. Diagrama de bloques de este módulo principal y sus características, que se ha consultado en el 'Datasheet':

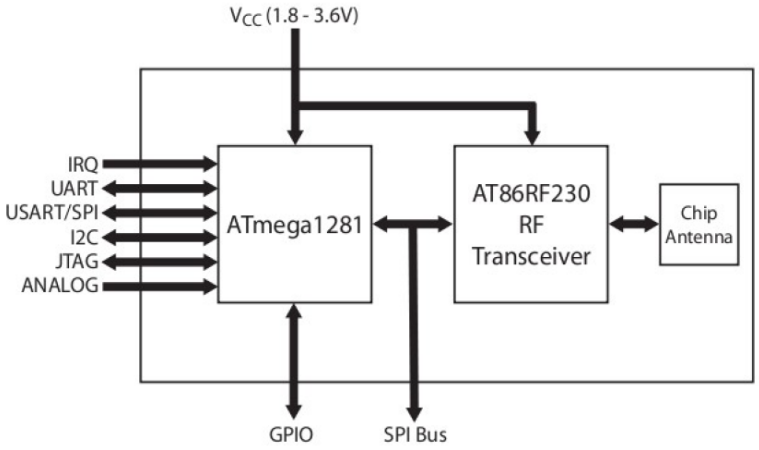


Ilustración 12 Diagrama Bloques Núcleo Mota

Microcontrolador

El Atmega 1281 es un microcontrolador de bajo consumo, de tipo CMOS de 8 bits basado en la arquitectura RISC del fabricante 'Atmel', familia con núcleo AVR. Posee una memoria Flash de 128 Kb, una EEPROM de 4 Kb, una memoria SRAM (de 8 Kb. Con 32 registros de propósito general, 6 Timers / counters y 4 transceptores universales de comunicación serie (USART). Una interfaz de comunicación serie de dos cables un puerto serie SPI un interfaz de test JTAG, que cumple los requisitos del estándar IEEE 1149.1 ', que se utiliza para la programación y depuración donde chip 'de los microcontroladores. También contiene un conversor analógico digital de 10 bits con entradas únicas, diferentes y con un paso programable.

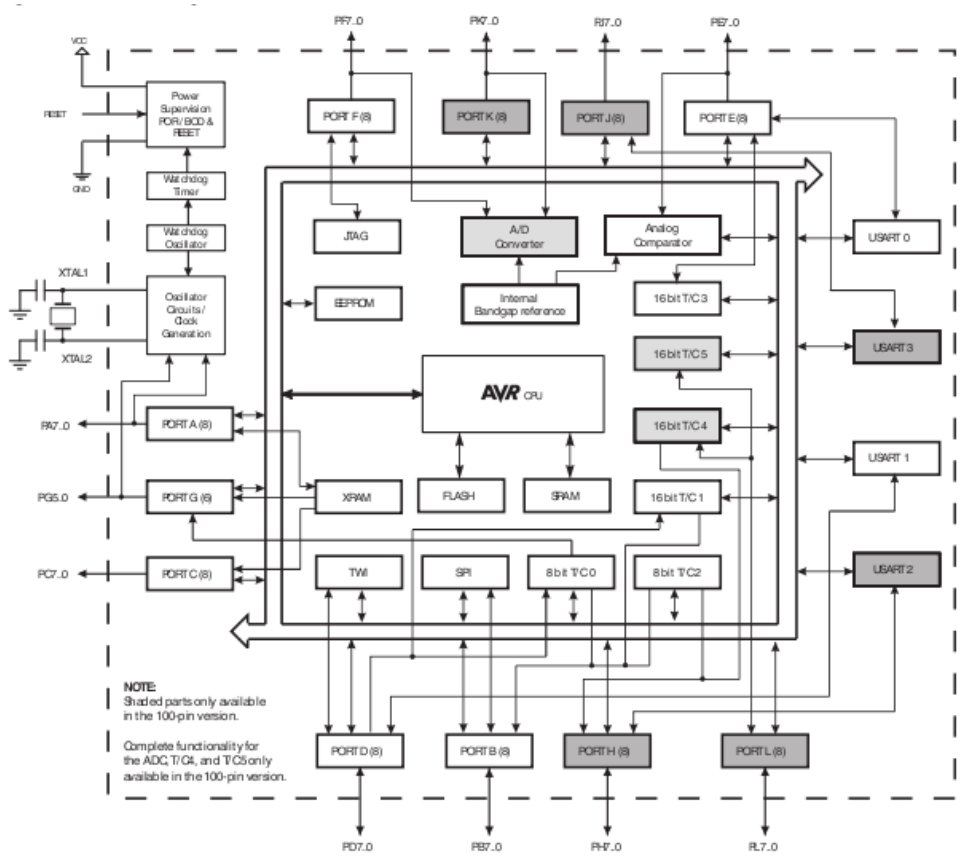


Ilustración 13 Diagrama Bloques Microcontrolador

Transceptor

El transceptor AT86RF230, es una radio transceiver de bajo consumo con alta sensibilidad (-101 dBm), y una distancia de de aproximadamente 30 metros. Trabaja en la banda libre de radiofrecuencia ISM a 2.4 Ghz. También decir que tiene 16 canales de comunicación y opera con el protocolo estándar de comunicación IEEE 802.15.4., que pertenece a la especificación de un conjunto de protocolos "ZigBee".

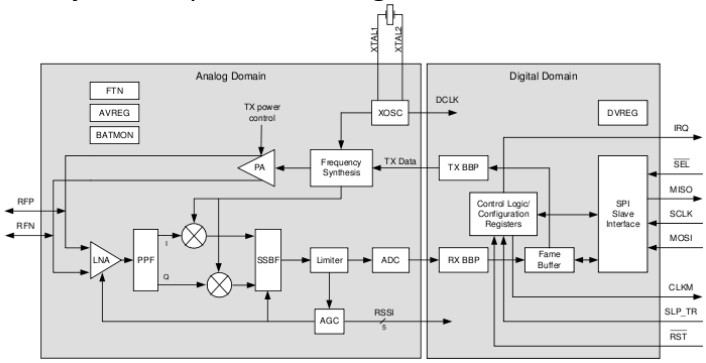


Ilustración 14 Diagrama Bloques Transceptor

En las motas hay integrados tres sensores, el de temperatura, luminosidad y el de efecto Hall. En este proyecto utilizamos sólo el sensor de temperatura.

Sensor Temperatura.

Sensor de temperatura MCP9700A: Su funcionamiento se basa en una resistencia que varía con la temperatura. Convierte la lectura de temperatura en una señal de voltaje analógico en mili voltios. El rango de temperaturas detectable es de -40 º C a +150 º C y la precisión de ± 2 º C en el rango de 0 º C a +70 º C. No requiere ningún circuito añadido y la salida Vout de voltaje del sensor se puede conectar directamente a la entrada ADC (Analogic Digital Converter) del microcontrolador 'Atmega 1281'. El coeficiente de temperatura esta ajustado para proveer 1 º C / bit de resolución, con un convertidor de analógico a digital de 8 bits y con un voltaje de referencia de 2.5 voltios.

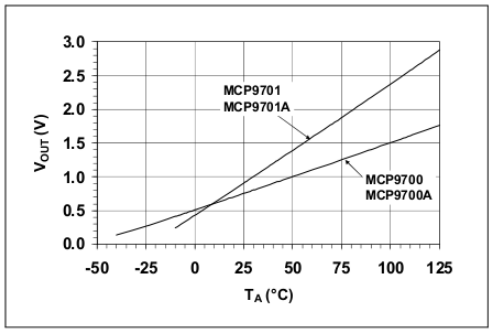


Ilustración 15 Gráfica Sensor Temperatura

Protocolo de comunicación "ZigBee"

Es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica, para la utilización de radiodifusión digital de bajo consumo, basada en el estándar de comunicación IEEE 802.15.4. de redes inalámbricas de área personal. Su objetivo son las aplicaciones de sistemas embebidos, que requieren comunicaciones seguras con baja tasa de transferencia de envío de datos y ahorro de energía de las baterías. Las características más importantes son:

- Bajo consumo energético.
- Topología de red en malla.
- Fácil integración, posibilidad de fabricar nodos con pocos componentes electrónicos.

Se definen tres tipos de dispositivos "ZigBee", dependen de la función que hacen en la red:

- Coordinador "ZigBee": Es el dispositivo más completo y debe haber uno en la red. Se encarga de controlar la red y los caminos que deben seguir los dispositivos paraconectarse.
- Router "ZigBee": Se encarga de la interconexión de dispositivos dentro de la topología de la red. Ofrece un nivel de aplicación para la ejecución de código de usuario.
- Dispositivo final: Posee la funcionalidad de comunicarse con el nodo padre (nodo coordinador o router), pero no envía información a otros nodos.

TinyOS

El sistema operativo TinyOS posee el lenguaje de programación 'NESC', que se utiliza para implementar las aplicaciones que serán ejecutadas en las motas. Es un lenguaje de programación que combina dos aspectos de la orientación a objetos que son la orientación a componentes y la orientación a eventos. La orientación a componentes provee los componentes primitivos, que proporciona el propio sistema operativo y los componentes complejos que son externos. Estos componentes proporcionan unas interfaces que el programador utiliza para crear sus propios componentes, este proceso se denomina 'WIRING'. En cuanto a la orientación a eventos, la programación no se secuencial sino basada en la ejecución de un evento, es decir, el código se ejecutará cuando se ejecute el evento.

Capítulo 3 Descripción funcional.

El proyecto consiste en la realización de un sistema de detección de incendios basada en una red de sensores vía radio (wifi). Un ejemplo de aplicación podría ser el de la figura adjunta. Este ejemplo podría ser una zona despoblada que necesitamos supervisión, como un bosque para detectar incendios o un hotel donde no se puede hacer cableado.

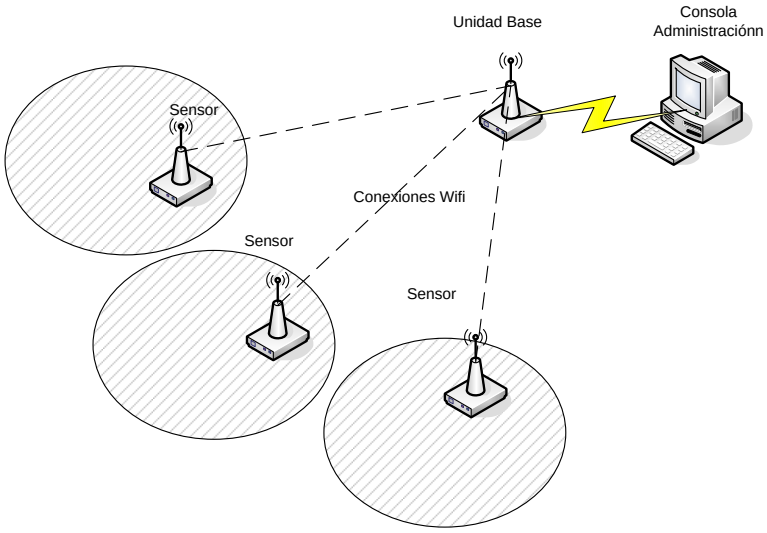


Ilustración 16 Escenario de Aplicación

Dentro del escenario de aplicación, disponemos de sensores que actúan como receptores del entorno (temperatura y luminosidad) y de una unidad base que será la encargada de supervisar activamente a los sensores. Además de un ordenador que conectado a la unidad base nos indicara en forma de mensajes el estado de los sensores e información asociada al funcionamiento del sistema.

Cada sensor comprobará la temperatura de su entorno a intervalos regulares. Este comprobará que dicha temperatura, no sea superior a una temperatura dada de referencia. Cuando la temperatura del entorno sea superior a la temperatura de referencia, este activará una indicación sonora y luminosa en el sensor. También enviará una "alarma" de temperatura a una unidad base que es la receptora de alarmas. Existirá en el sensor un pulsador para que de forma manual se pueda activar la alarma.

El propio sensor comprobará el estado de su batería enviando una alarma a la estación base si este nivel es bajo.

La unidad base notificara igualmente la situación del estado de alarma mediante una indicación luminosa y auditiva. Esta indicación, se producirá si desde cualquier sensor se recibe la situación de alarma. Así mismo se indicará con un mensaje en un ordenador que actuara de consola de administración de que se ha producido una alarma e indicara el sensor que la ha detectado.

Desde la unidad base se comprobara el funcionamiento correcto de los sensores interrogándolos periódicamente. Comprobando las condiciones medioambientales del entorno y el correcto funcionamiento del sensor.

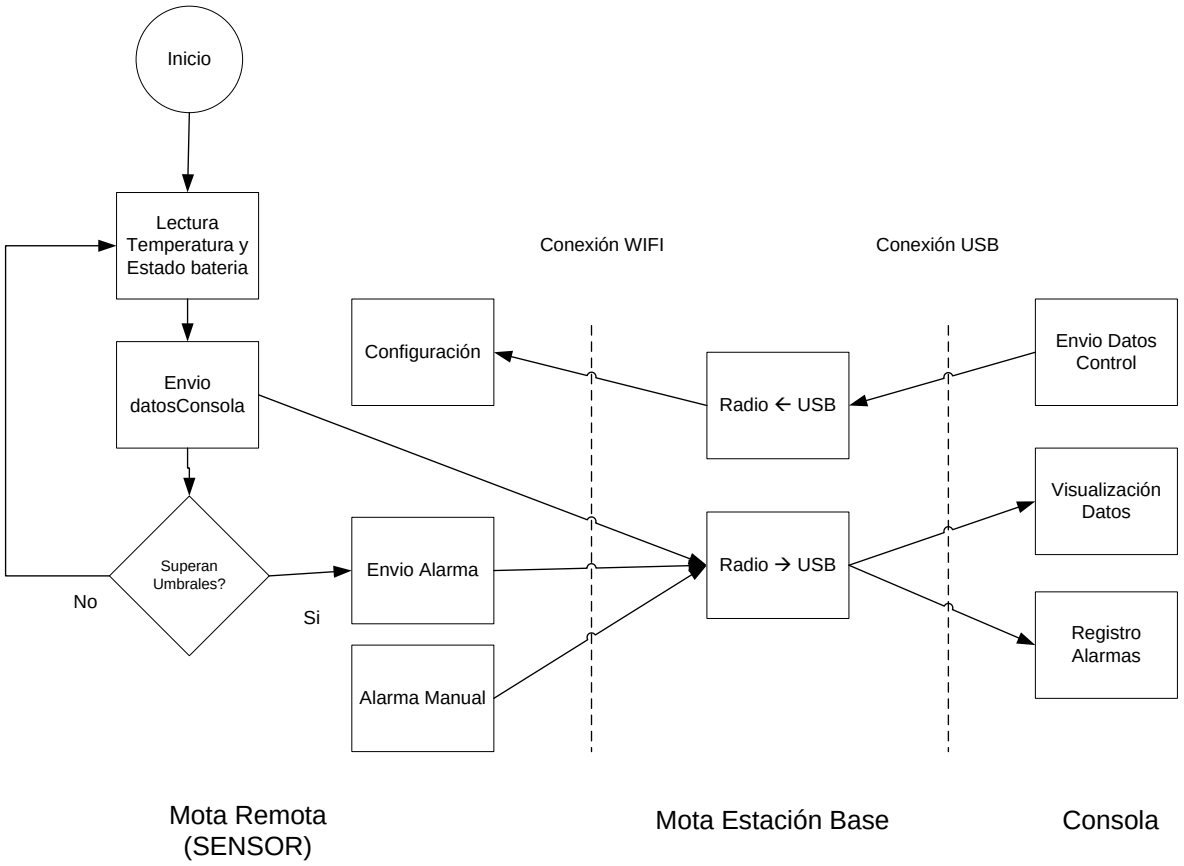


Ilustración 17 Diagrama Funcional de Bloques

Capítulo 4 Descripción detallada.

1. *Mota Remota (Sensor)*

La aplicación está dividida en componentes. Cada uno desempeña la funcionalidad para la cual se ha programado. La comunicación entre los componentes, se hace mediante comandos, y estos comandos son públicos para el resto de módulos mediante interfaces que hacen lo hacen visible. Con esto conseguimos las características de una programación orientada a objetos, como es encapsulación, aislamiento reutilización y sobre todo a la hora de modificar el código, es más sencillo. Aunque añada complejidad en la estructura.
En primer lugar vamos a describir los diferentes componentes de la aplicación que ira instalada en la mota remota para después explicar básicamente su funcionamiento.

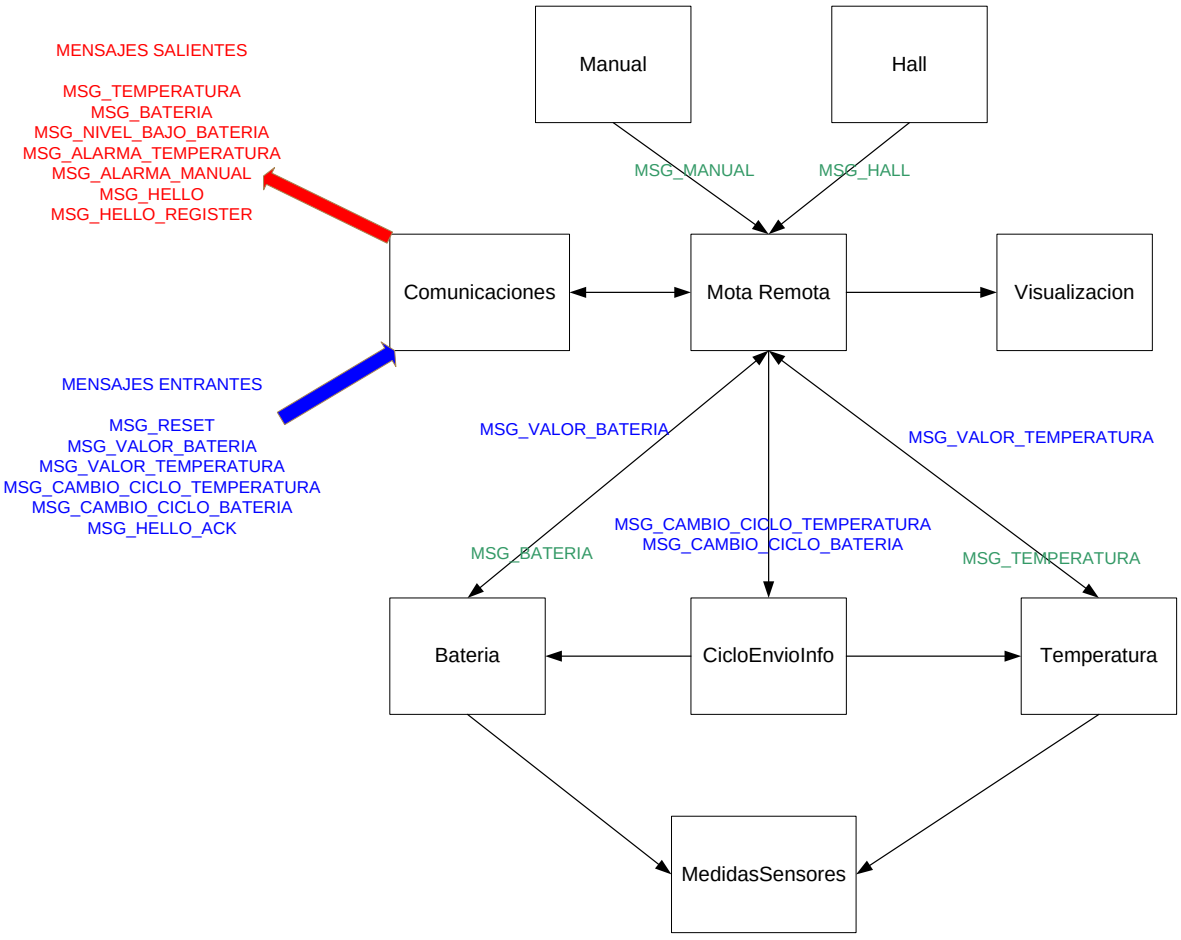


Ilustración 18 Componentes y mensajes mota Remota

Componentes Mota Remota

Mota Remota: Es el corazón del sistema. Alberga una máquina de estados la cual va cambiando de estado en función de qué tipo de mensaje recibo del resto de módulos y las acciones asociadas a ese tipo de mensaje.

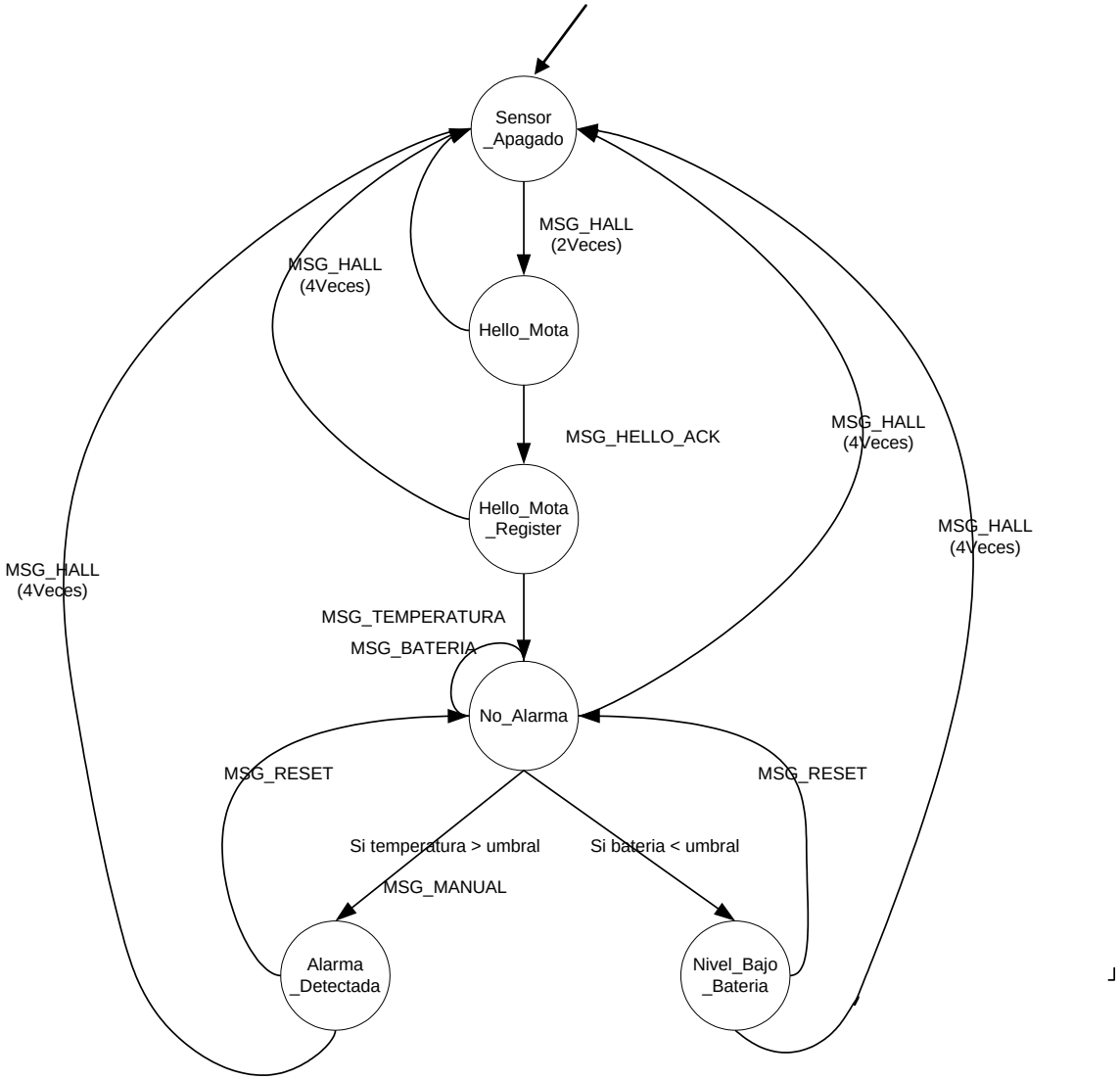


Ilustración 19 Maquina Estados Mota Remota

A este modulo se accede mediante un interfaz llamado MotaRemota:

```
interface MotaRemota
{
        command error_t  queueMessage(nx_uint16_t messageType, nx_uint16_t data);
        event void messageSent(error_t rt);

}
```

Mediante este interfaz los diversos componentes, de la mota van dejando los mensajes para su posterior procesado por parte de este componente. El procesado se hace en un método con estructura swtich /case identificando el tipo de mensaje y realizando las acciones asociadas. Por ejemplo:

Al identificar que el componente de temperatura nos envía el valor de la temperatura detectada por el sensor, se comprueba que no supere el umbral, si es superior, cambiamos de estado al estado a alarma detectada y posteriormente mediante el componente comunicaciones, enviaremos un mensaje de alarma de temperatura.

```
case MSG_TEMPERATURA:

        tipoMensaje=messageType;
        datos=data;
        if (data>temperaturaAlarma )
        {
                        atomic pEstat=ALARMA_DETECTADA;
                        tipoMensaje=MSG_ALARMA_TEMPERATURA;

        }
        break;
```

El componente puede responde con el evento de recepción correcta de mensaje.

Hall: Es el modulo que se encarga de la lectura del sensor de efecto hall y dejar su información en el componente MotaRemota mediante la interfaz implementada.

Manual: Es el modulo que se encarga de la lectura del pulsador y dejar su información en el componente MotaRemota mediante la interfaz implementada

Visualización: Tenemos un componente especializado en la visualización mediante el encendido y apagado de los leds de la mota visualizando los estados de MotaRemota. De igual modo su comunicación es mediante una interfaz llamada Visualizacion. Básicamente este componente, consiste en una estructura swtich /case identificando el estado en el que se encuentra la MotaRemota y visualizándolo.

Cada vez que se produce un cambio de estado en MotaRemota y según este, se encienden unos leds u otros. En el caso del sensor apagado, hacen un parpadeo los tres leds.

```
switch(data)
        {
                case SENSOR_APAGADO:
                        //Enciando y apago ledss
                        call Leds.set(LEDS_LED0 | LEDS_LED1 | LEDS_LED2);

                        call Timer.startOneShot(100);
                        break;

                case NO_ALARMA:
                        //call Leds.led2Toggle();
                        call Leds.set(LEDS_LED2);

                        break;
…
```

MedidasSensores. Es el componente encargado de la lectura de los sensores de la mota y guardar sus valores en unas variables, que son lo único visible al exterior. La lectura de los sensores se hace mediante un temporizador que no se puede variar su ciclo de lectura. Este componente mediante su interfaz deja visibles los valores de los sensores.

```
interface MedidasSensores
{
        command nx_uint16_t getTemperatura();
        command nx_uint16_t getBateria();
        command nx_uint16_t getPhoto();
}
```

También está basado en un maquina de estados que cíclicamente va leyendo los valores de temperatura, nivel de batería y lectura del sensor de luminosidad, aunque este último, no se usa.

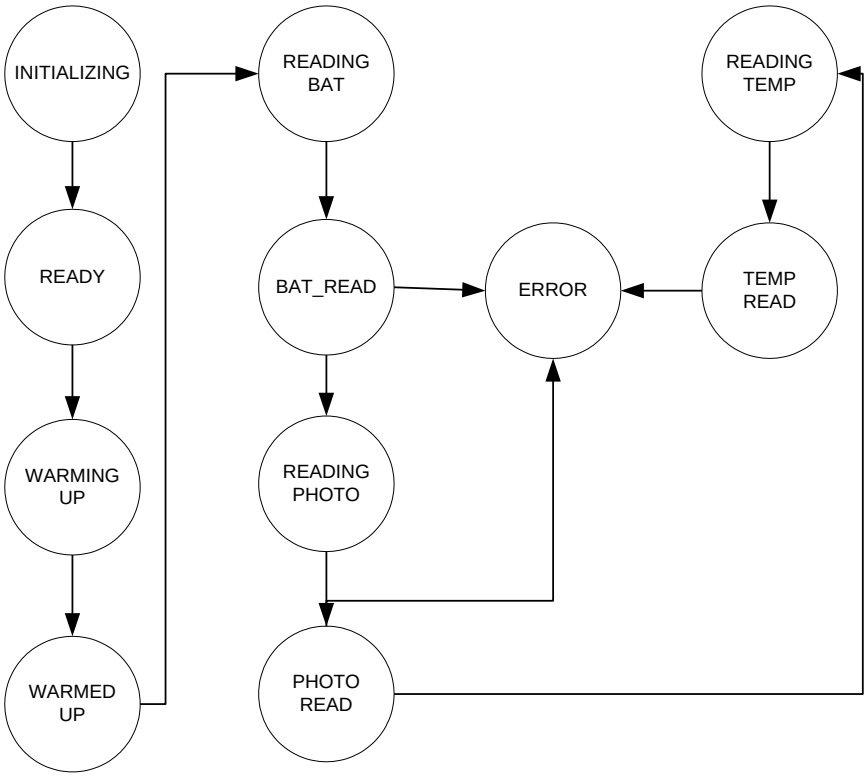


Ilustración 20 Maquina Estados Medidas Sensores

Temperatura y Batería. Son componentes que hacen de puente entre Sensores y MotaRemota. Estos dos módulos leen la temperatura y el nivel de batería del componente sensores que son variables públicas. Dejan su valor independientemente en el componente MotaRemota mediante sendos mensajes través de la implementación de la interface MotaRemota. Esto lo hacen mediante un temporizador que es programable su ciclo de lectura por parte del usuario mediante la interfaz CicloEnvioInfo. La interfaz CicloEnvioInfo es visible al exterior como componentes independientes, mediante:

```
TemperaturaP
…
//Interfaz para cambio ciclo envio
provides interface CicloEnvioInfo as TemperaturaCicloEnvio;

BateriaP
//provee interfaz Ciclo envio referencia a el mismo
provides interface CicloEnvioInfo as BateriaCicloEnvio;
```

CicloEnvioInfo. Es una interfaz pública que implementan los componentes Temperatura y Batería, que a su vez exportan la funcionalidad que les brinda la interfaz para poder cambiar los ciclos de lectura de temperatura y batería por parte del usuario.

```
interface CicloEnvioInfo
{
        command void setCicloEnvio(nx_uint8_t segundos);
}
```

Comunicaciones. Este componente es nuestro experto en comunicaciones. Todo lo que recibe, lo envía a Mota Remota y todo lo que recibe de MotaRemota lo envía al exterior. Lo podemos configurar mediante radio o usb cambiando solo tres líneas de su código. También dispone de una maquina de estados.

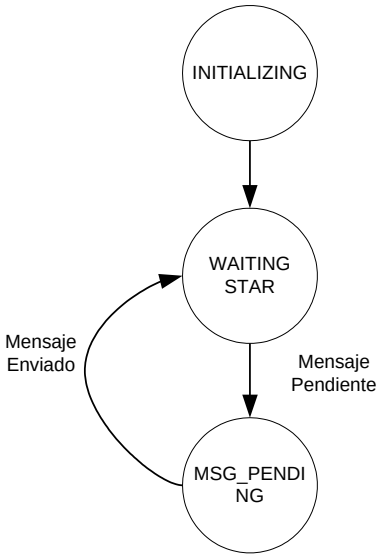


Ilustración 21 Maquina Estados Comunicaciones

Para la comunicación con el componente MotaRemota, usamos una cola que nos brinda el componente Queue<tramaMsg>. Mediante una interfaz llamada Comunicaicones.

```
interface Comunicaciones
{
        command error_t queueMessage(nx_uint16_t messageType, nx_uint16_t data, nx_uint8_t moteId);
        event void messageSent(error_t rt);
}
```

Alli el componenete MotaRemota encola el mensaje y Cambiamos el estado a MSG_PENDING indicando que algo para enviar. Comprobamos la cola y mientras haya mensajes los seguiremos enviando hasta dejar la cola vacía.

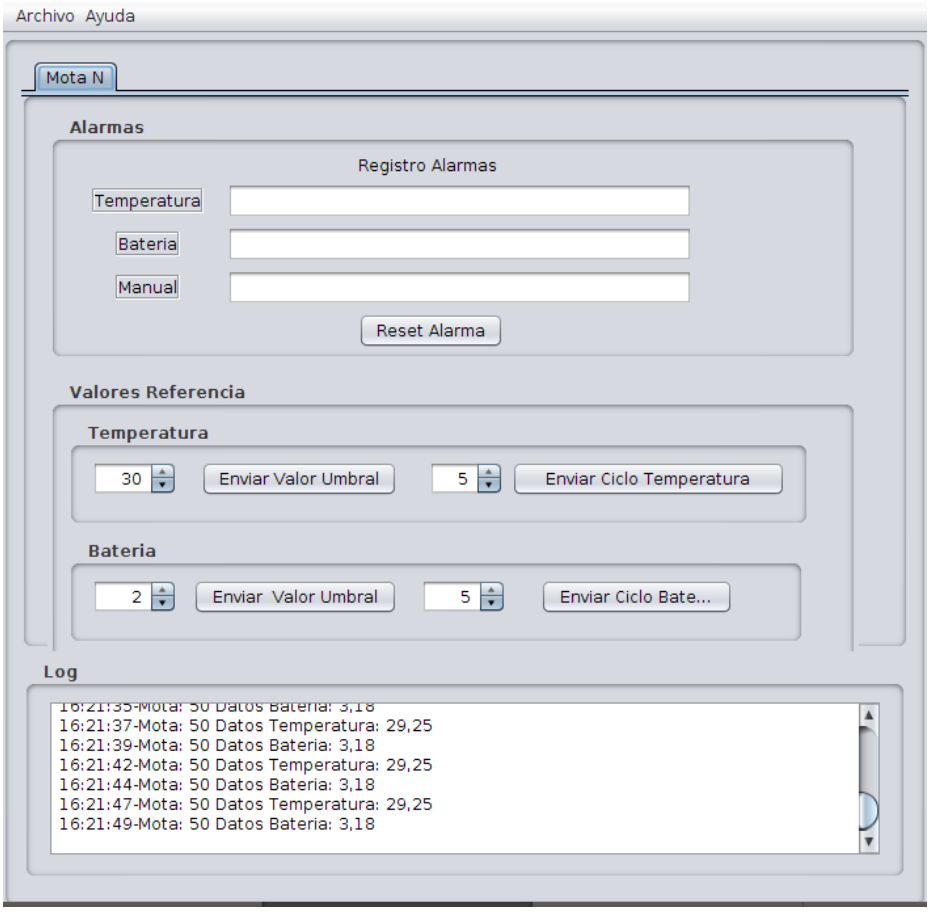
Mota Estación Base

Para la mota Estación base hemos usado el módulo BaseStation de los ejemplos.

BaseStation. Este componente tiene la funcionalidad de repetidor. Consiste en dos colas una para todo lo que recibe de la mota remota lo redirecciona por usb y la otra para todo lo que le envía la aplicación consola, lo reenvía vía radio a las motas remotas.

Consola

Es una aplicación hecha en java, que nos permite ver que ocurre en el sistema, visualizando los mensajes recibidos de la motas, registrando las alarmas y nos permite cambiar la configuración de las motas en concreto los valores de detección de umbral y cambios de ciclo de lectura tanto de temperatura como de batería. La veremos con mayor detalle en el manual de usuario.



En principio, la aplicación consola es solo para una mota, si hubiese mas motas, habría una pestaña para cada mota.

2. Intercambio de mensajes

Definimos una estructura de mensajes válida para la comunicación entre las motas y la aplicación consola. Esta estructura es la que utilizamos para todos los intercambios de mensajes. Los diferenciaremos por el campo tipoMsg. La estructura es la siguiente:

```
//mesaje de envio de datos del sistema
typedefnx_structtramaMsg {
        nx_uint8_tmoteId;
        nx_uint16_tdatos;
        nx_uint16_ttipoMsg;
        nx_uint16_tsecuencia;
        nx_uint8_tmacAddr[8];

} tramaMsg;
```

moteId: indica el identificador asociado a la mota.
datos: El valor asociado al tipo de mensaje. Por ejemplo en un mensaje tipo MSG_TEMPERATURA, indica el valor leído de la temperatura. En un mensaje de tipo MSG_MANUAL, no se usa.
tipoMsg: El tipo de mensaje entre la mota y la consola
secuencia: Campo no usado. Estaba previsto para el saludo inicial a tres vías para la obtencio del identificador de mota y para el reconocimiento de alarmas por parte de la consola.
macAddr: La dirección mac de la mota.

Para la detección del tipo de mensaje conlleva la realización de un "proceso" que según el tipo de mensajes nos realice las acciones asociadas a ese tipo de mensaje. Se traduce a nivel de código en una estructura de tipo swtich(tipoMensaje) / case (MensajeTemperatura)….

Mensajes internos

Tipo mensaje	Dirección Componentes	Significado	Campo Datos
MSG_MANUAL	Manual → Mota Remota	Indica que se ha activado la alarma de forma manual	No usado
MSG _TEMPERATURA	Temperatura → Mota Remota	Indica el valor de la temperatura	Valor leído temperatura
MSG_BATERIA	Bateria→ Mota Remota	Indica el valor de la batería	Valor leído nivel batería
MSG_HALL	Hall → Mota Remota	Indica que se ha activado el sensor de efecto hall	No usado

Mensajes entrantes

Tipo mensaje	Dirección	Significado	Campo datos
MSG_TEMPERATURA	Mota remota → Consola	Indica el valor medido de la temperatura	Valor leído temperatura
MSG_ALARMA_TEMPERATURA	Mota remota → Consola	Indica alarma detectada en temperatura y el valor medido.	Valor leído temperatura alarma
MSG_BATERIA	Mota remota → Consola	Indica el valor medido del nivel de batería	Valor leído batería
MSG_NIVEL_BAJO_BATERIA	Mota remota → Consola	Indica alarma detectada en nivel de batería y el valor medido.	Valor leído nivel batería alarma
MSG_ALARMA_MANUAL	Mota remota → Consola	Indica pulsación manual de alarma	No usado
MSG_HELLO	Mota remota → Consola	Mensaje inicial para obtener un identificador de mota	No usado
MSG_HELLO_REGISTER	Mota remota → Consola	Mensaje final que indica recepción del identificador mota	No usado

Mensajes salientes

Tipo mensaje	Dirección	Significado	Campo datos
MSG_RESET	Consola → Mota Remota	Reset de alarmas en la mota	No usado
MSG_VALOR_TEMPERATURA	Consola → Mota Remota	Cambia el valor del umbral de temperatura	El nuevo valor del umbral de detección de temperatura
MSG_CAMBIO_CICLO_TEMPERATURA	Consola → Mota Remota	Cambia el ciclo de lectura de temperatura	El nuevo valor del ciclo de lectura de temperatura
MSG_VALOR_BATERIA	Consola → Mota Remota	Cambia el valor del umbral de batería	El nuevo valor del umbral de detección de nivel de batería
MSG_CAMBIO_CICLO_BATERA	Consola → Mota Remota	Cambia el ciclo de lectura de batería	El nuevo valor del ciclo de lectura de nivel de batería
MSG_HELLO_ACK	Consola → Mota Remota	Notificación de la recepción de petición de identificador de mota	El valor del identificador de la mota.

3. Funcionamiento detallado del sistema

Inicialmente la mota remota está apagada (estado SENSOR_APAGADO), indicando su estado mediante el parpadeo de los tres leds. Esperando a la activación mediante el sensor de efecto hall. Este envía a la Mota Remota mensajes MSG_HALL cuando es activado por un imán. Cuando recibe 2 mensajes de este tipo se produce la activación del sensor. Cuando recibamos 4 mensajes MSG_HALL el sensor se apaga y pasa nuevamente al estado SENSOR_APAGADO.

Una vez activado el sensor la mota intenta obtener un identificador de mota que pide mediante un saludo a tres vías. Pasa del estado SENSOR_APAGADO a HELLO_MOTA en este estado envía mensajes MSG_HELLO cada segundo, hasta obtener de la consola un MSG_HELLO_ACK. Este mensaje le hace cambiar de estado a HELLO_MOTA_REGISTER y finalmente la mota remota envía un MSG_HELLO_REGISTER. Con este proceso obtengo un id mota.

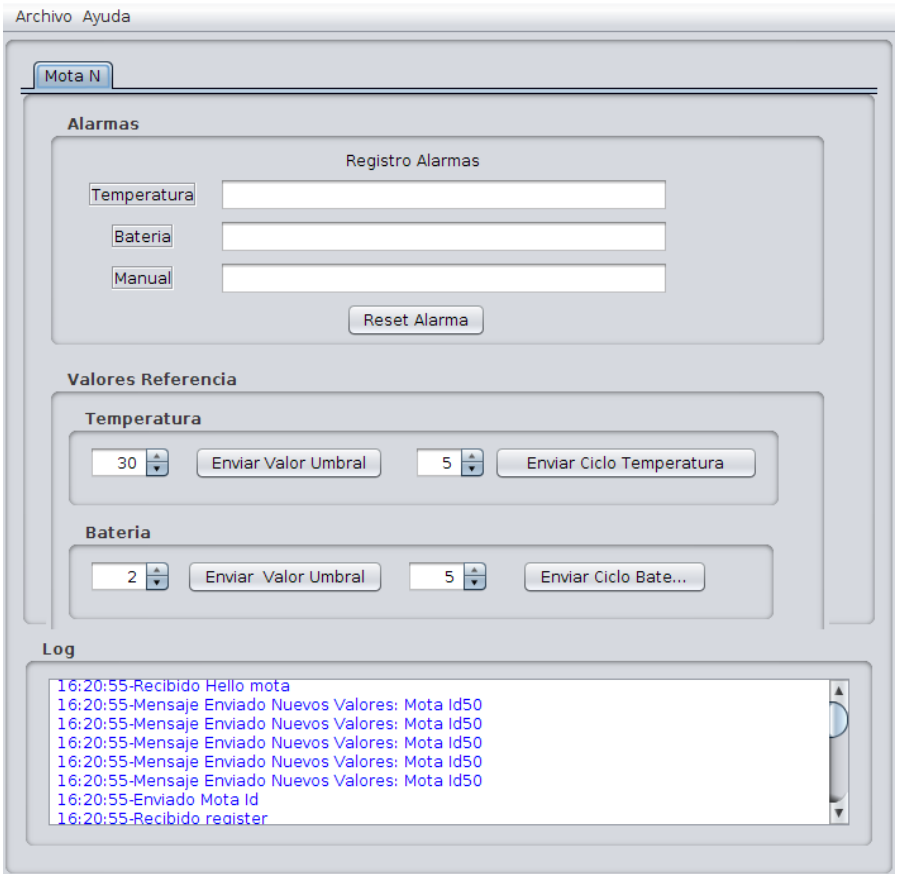


Ilustración 22 Dialogo inicial obtener Identificador mota

Una vez obtenido el identificador, el componente sensores lee los parámetros del entorno y los deja en unas variables. Esto es cíclico mediante temporizador y este valor de ciclo no es accesible para el usuario. Seguidamente Los componentes Temperatura y Batería leen esos valores de los sensores y los envía a MotaRemota mediante mensajes MSG_TEMPERATURA y MSG_BATERIA. Esto se hace cíclicamente y este valor si lo puede cambiar el usuario. Los mensajes son enviados a la consola para su monitorización.

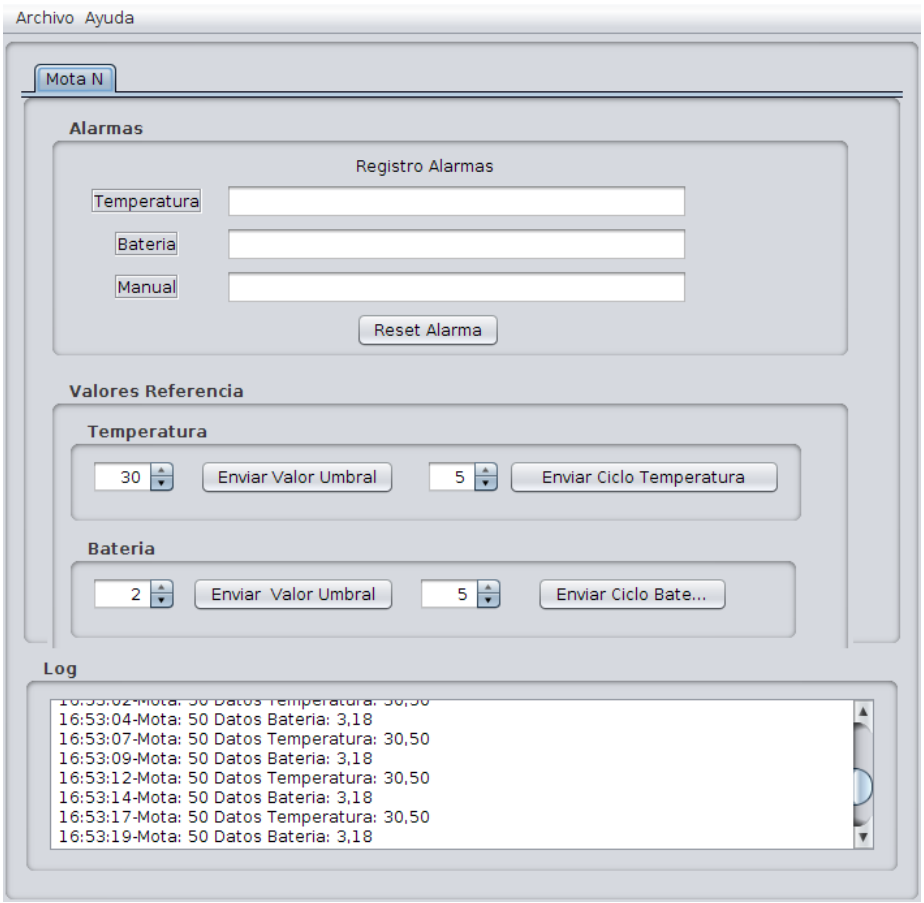


Ilustración 23 Mensajes de temperatura y batería

MotaRemota comprueba los valores de Temperatura y batería con sus umbrales de referencia y si no están fuera de márgenes, los reenvía al exterior sin modificarlos como mensajes MSG_TEMPERATURA Y MSG_BATERIA. Si están fuera de los umbrales programados, hace cambiar de estado a MotaRemota, hacia ALARMA_DETECTADA y aquí modifica los mensajes y los convierte en MSG_ALARMA_TEMPERATURA o MSG_NIVEL_BAJO_BATERIA. Identificado un estado de alarma. Estas alarmas quedan visualizadas y registras en la aplicación consola.

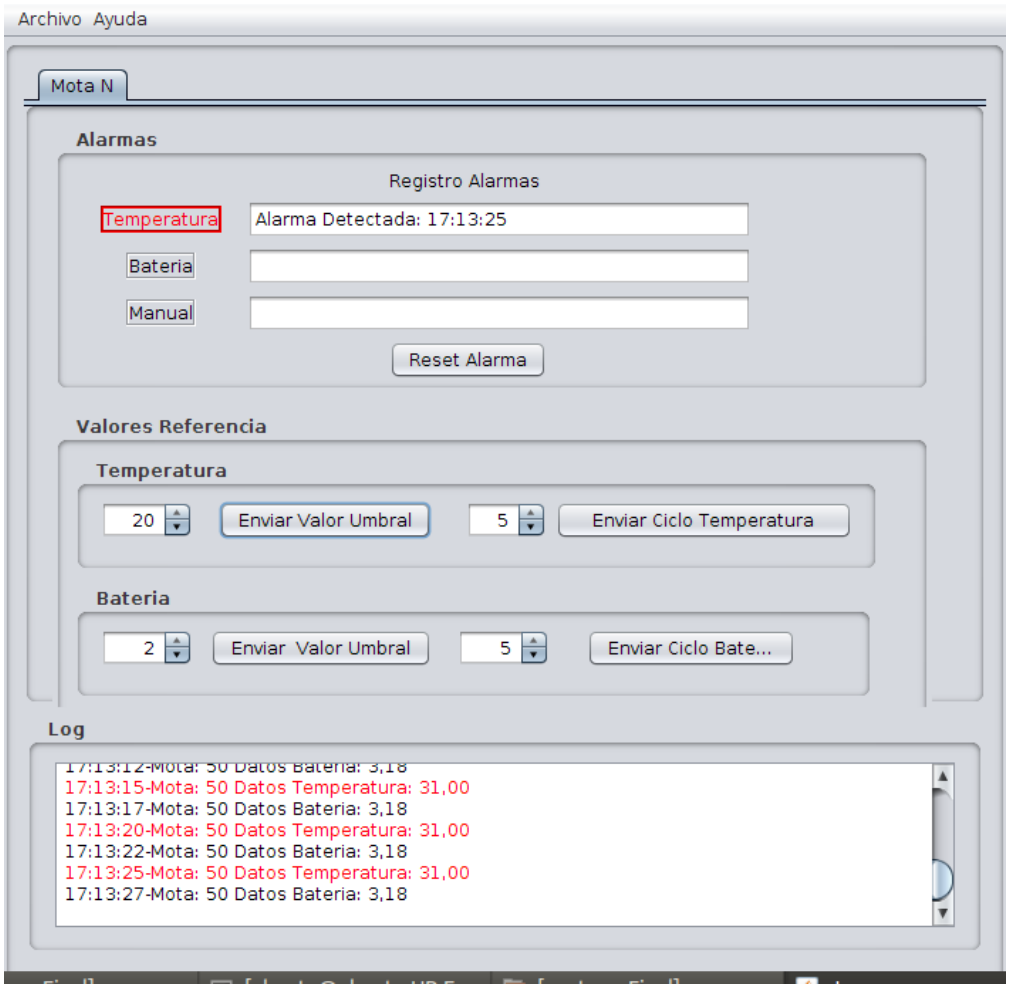


Ilustración 24 Alarma De Temperatura Detectada

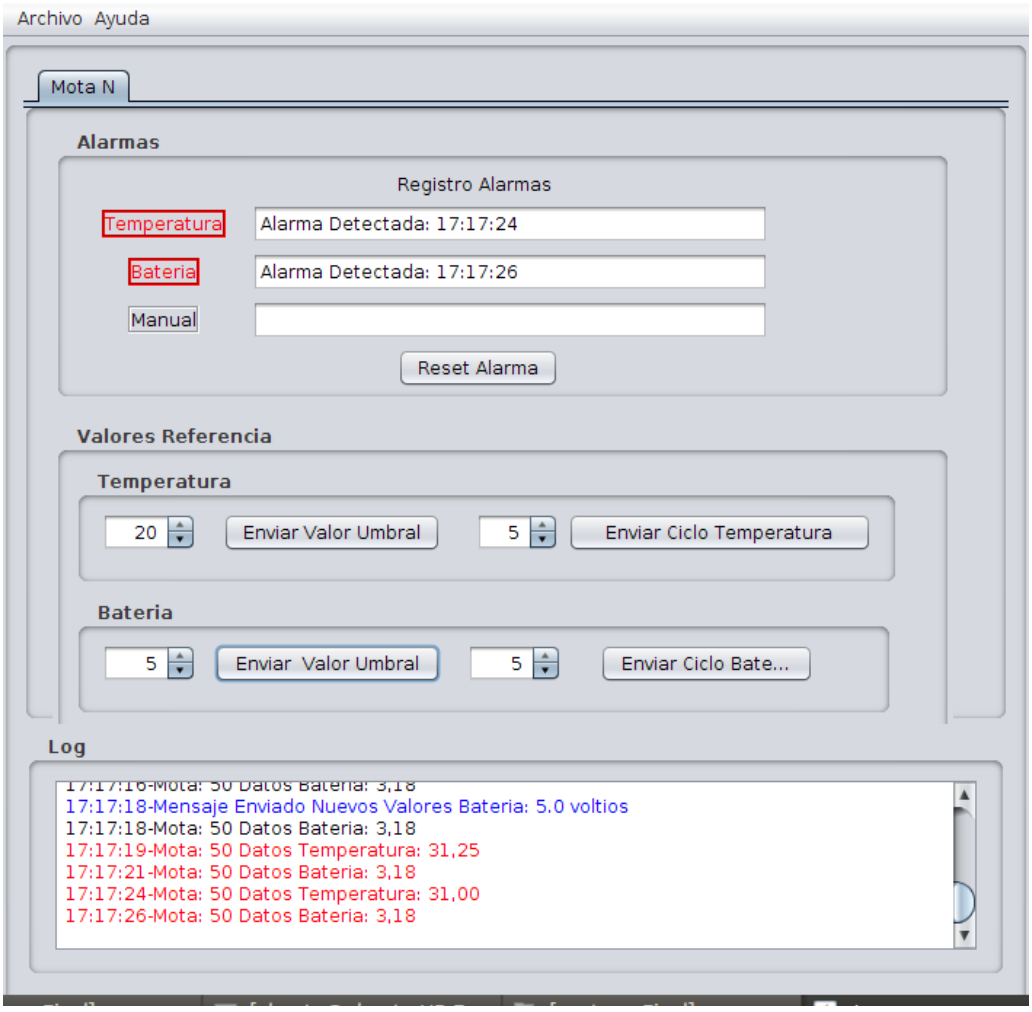


Ilustración 25 Alarma De Bateria Detectada

También podemos en cualquier momento pulsar el pulsador de alarma manual esto provoca un mensajes MSG_ALARMA_MANUAL notificándolo enviando un solo mensaje.

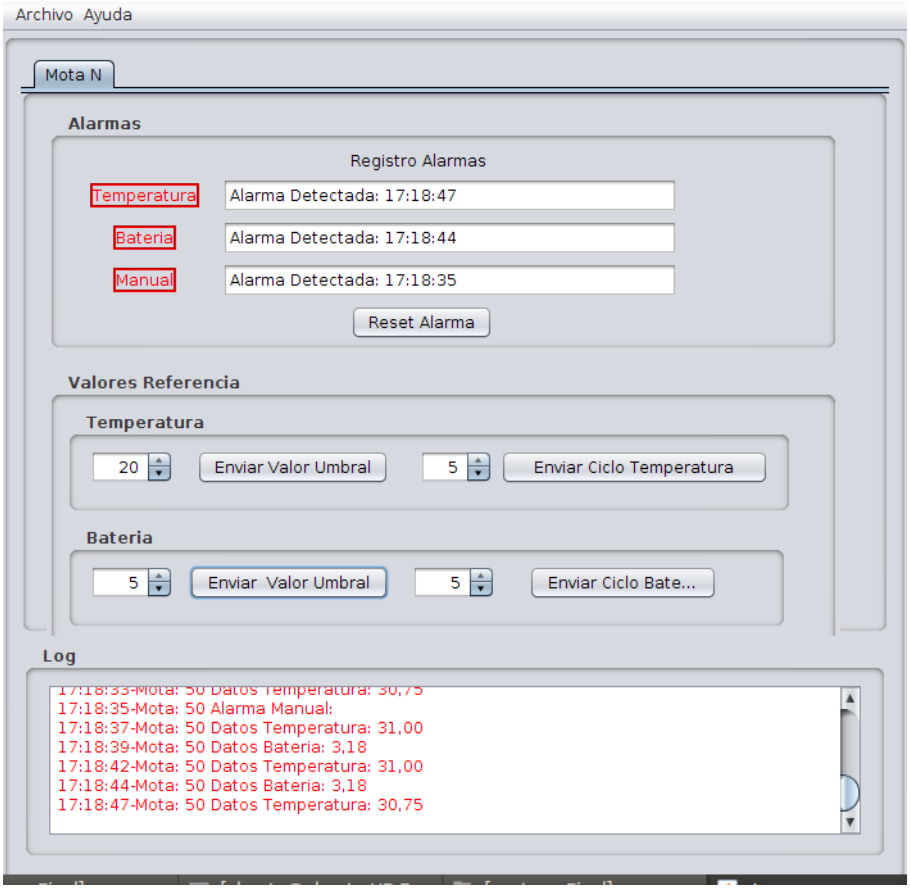


Ilustración 26 Alarma Manual Registrada

Mediante la aplicación de consola, podemos configura el sistema para, cambiar los valores de detección de alarmas por parte de la mota remota enviando mensajes MSG_VALOR_TEMPERATURA o MSG_VALOR_BATERIA. También podemos variar los ciclos de lectura de los sensores mediante mensajes MSG_CAMBIO_CICLO_TEMPERATURA y MSG_CAMBIO_CICLO_BATERIA. Este tipo de mensajes cambian el valor de variables dentro de la mota. Además el mensaje MSG_RESET borra las alarmas de la mota. La hace cambiar de estado a NO_ALARMA.

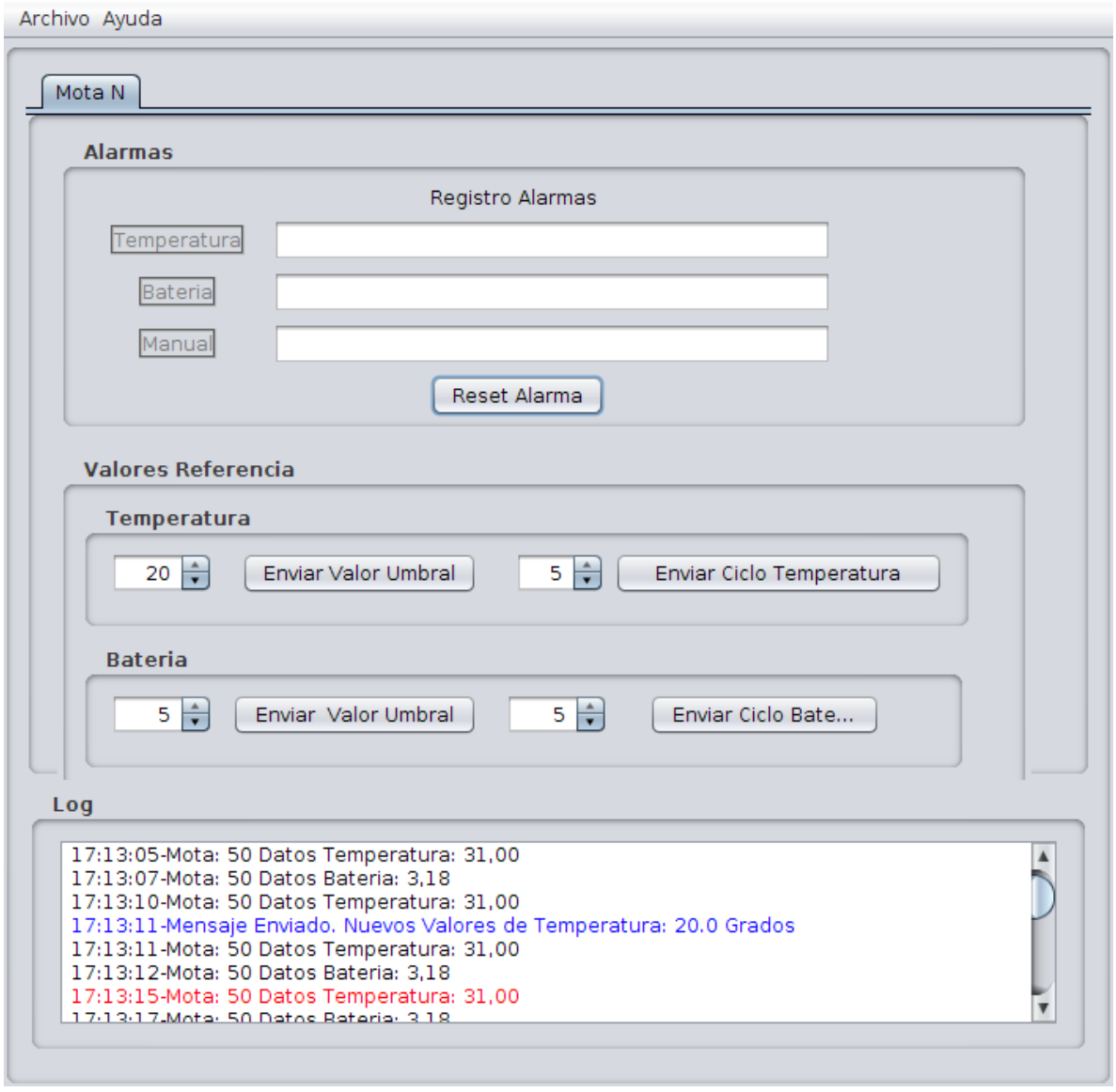


Ilustración 27 Cambio Umbral Temperatura

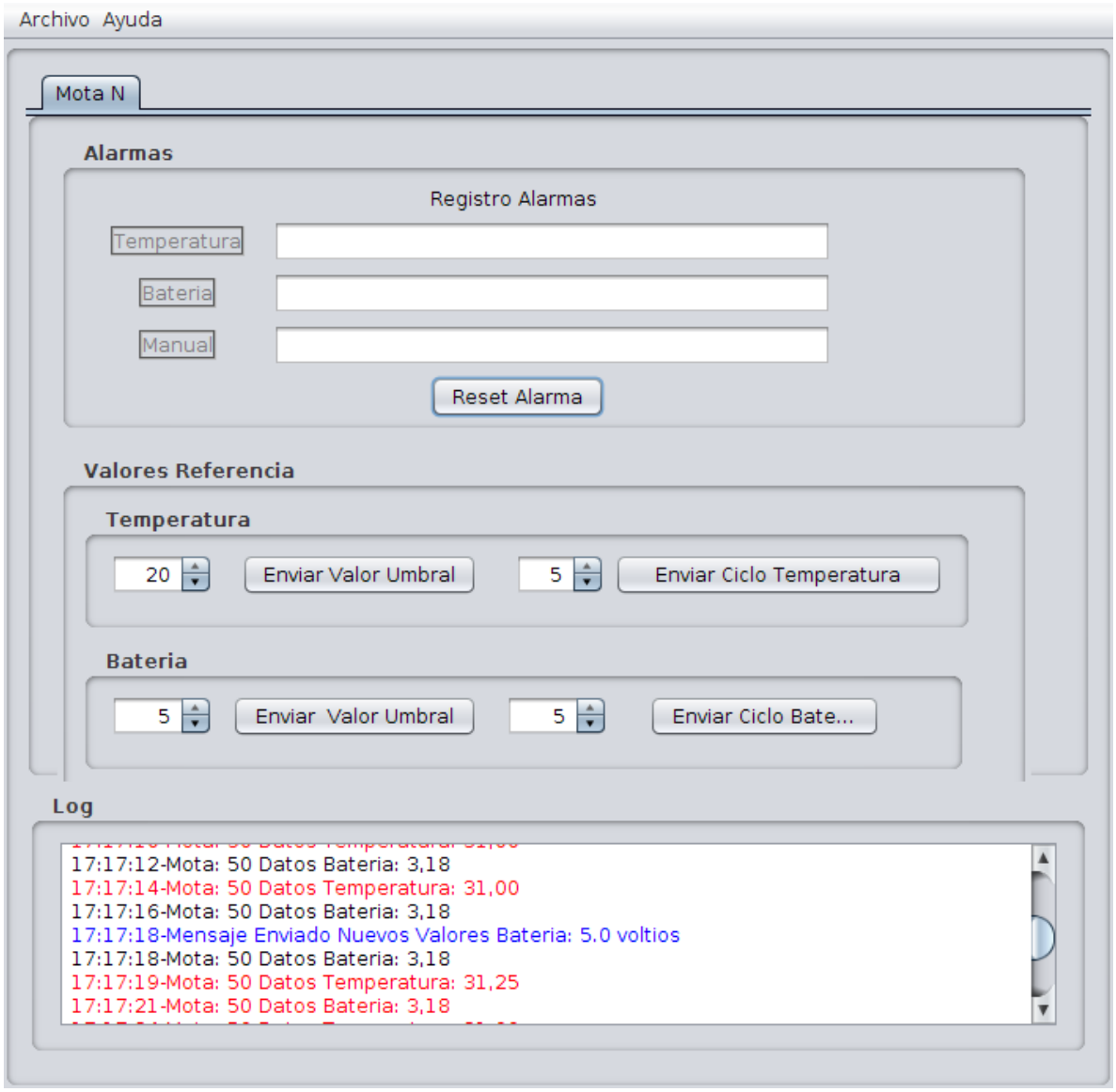


Ilustración 28 Cambio Umbral Batería

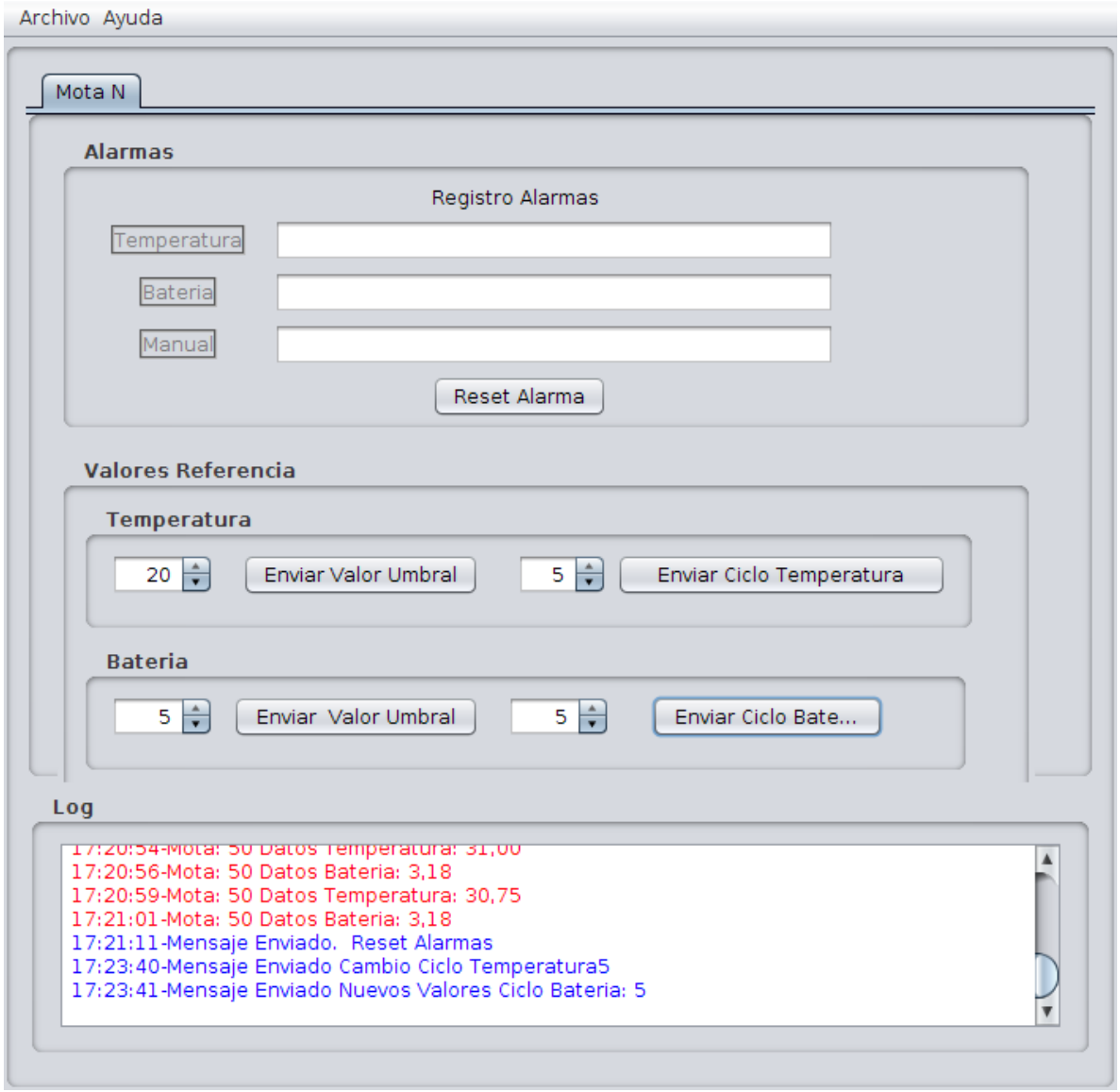


Ilustración 29 Cambios temporización y reset alarmas

Mientras por otro lado el componente visualización, nos permite ver el estado de la mota mediante el encendido de los leds. Un led verde indica funcionamiento correcto. Un led rojo y verde indica pulsación de alarma manual o alarma temperatura. Un led ambar y verde indica nivel bajo de batería.

Capítulo 5 Viabilidad técnica.

Este proyecto consiste en la experimentación e implantación de un sistema basado en una red de sensores con solo un sensor y una estación base repetidora.. A pesar de estar en un entorno puramente académico de experimentación. Pienso que puede ser llevado a la práctica finalizando los requisitos mínimos marcados. Sobre todo lo referente a la confirmación de alarmas por parte de la aplicación consola y dotar de supervisión a los nodos mediante la comprobación periódica de los mensajes de nivel batería. Y supervisan de errores en las motas mediante estados de error.

Capítulo 6 Conclusiones.

La realización del proyecto ha sido muy favorable tanto en cuanto al aprendizaje de "cómo usar" un sistema embebido. No sin estar exento de errores. Primeramente hay que comprender como funciona globalmente todo el proceso y pasar después a los detalles. Una vez visto el proceso completo edición, modificación y compilación código fuente, el resto ha sido "sencillo".

He llegado a desarrollar un producto dentro del ámbito académico pero con una cierta viabilidad técnica. A pesar de no haber cumplido al 100% con los requisitos propuestos creo haber conseguido un alto grado de funcionalidad. Sobre todo estoy muy contento en la segunda fase del proyecto donde he conseguido separar el código en distintos componentes, dejando solo visibles las partes que me han interesado mediante interfaces, así como la implementación de un interfaz común por parte de dos componentes exportándola como componentes distintos.

Finalmente he logrado una visión de que es un sistema embebido, su uso actual en redes de sensores viendo también lo importante que es su desarrollo para captar datos del entorno. Mediante los cuales y con técnicas de minería de datos, extrapolaciones… podamos predecir un comportamiento de un sistema complejo.

Capítulo 7 Glosario.

Sistema embebido. Sistema hardware con todas las características propias de un "ordenador" que puede funcionar de manera autónoma.

Mota. Sistema embebido

Mota Sensor. Sistema embebido que incorpora sensores para recoger datos del entorno.

Redes de Sensores Inalámbricas son redes de pequeños dispositivos embebidos (sensores) capaces de comunicarse de forma inalámbrica.

IEEE. Instituto de ingeniería eléctrica y electrónica, donde estandarizan normativas.

IEEE 802.15.4. Una versión reducida del estándar de comunicaciones WIFI de 2,4 GHz.

TinyOS sistema operativo open source para programación de las motas.

ISM. Banda libre de radio frecuencia (industrial, científica, medica)

RISC: (Reduced Instruction Set Computer), Tecnologia usada en microprocesadores con un conjunto reducido de instrucciones

USB : (Universal Serial Bus), connector serie universal.

MAC : (Medium Acces Control Layer), capa de control de acceso al medio

JTAG : (Joint Test Action Group),standard 'IEEE 1149.1' , se utiliza para la programación y depuración de los microcontroladores.

SPI : (Serial Peripheral Interface), Estándar de comunicación para la transferencia e información entre circuitos integrados

Capítulo 8 Bibliografía

National Instruments Developer Zone: *Qué es una red de sensores inalámbrica (WSN)?*
http://zone.ni.com/devzone/cda/tut/p/id/9507

Berkeley, University of California: *The tinyOS help archives*
https://www.millennium.berkeley.edu/pipermail/tinyos-help

Introduccion Redes Inalambricas
Introducción a las **Redes Inalámbricas** de **Sensores**

Aplicaciones Motas en la Agricultira
http://diec.unizar.es/~imr/personal/docs/ProyPM0272006.pdf

Redes De sensores y plataformas
http://cdn.intechopen.com/pdfs/18324/InTech-ireless_sensor_network_at_a_glance.pdf

Comparativa motas
http://www.tecnologico.deusto.es/projects/smartmotes/files/D2.1_SmartMotes_WSN_ComparativeAnalysis_v1.8.pdf

Redes de Sensores
http://www.sensor-networks.org/

Pagina sistemas embebidos UOC
http://cv.uoc.edu/app/mediawiki14/wiki/P%C3%A0gina_principal

TinyOS
http://tinyos.net/

Errores relacionados con el entorno programación y las motas
http://cv.uoc.edu/app/mediawiki14/wiki/Known_problems
http://amrra.net/2010/11/09/solved-tinyos-error-tos-install-jni-31-unexpected-operator/
http://sourceforge.net/tracker/index.php?func=detail&aid=3122329&group_id=56288&atid=480036
http://www.george-smart.co.uk/wiki/TinyOS

Capítulo 9 Anexos

Creación del entorno de ejecución.

Para la ejecución del código del sistema, se necesita un sistema Linux preferiblemente Ubuntu.

Dicho sistema debe contener la instalación de tinyOS, en nuestro caso la versión tinyOS-2.1.1 Además debe ser modificado para poder compilar en el dispositivo especifico "cou24". También necesitaremos el programa meshprog, para transferir el código a las motas. Las instrucciones para preparar el entorno completo las podemos encontrar en http://cv.uoc.edu/app/mediawiki14/wiki/Inici24.

Es necesario igualmente, tener instalado una versión de java sdk, para poder ejecutar la consola de administración. Podemos encontrar instrucciones de instalación en http://www.guia-ubuntu.org/index.php?title=Java

No es necesario tener un entorno de desarrollo para ejecutar la consola de administración. Pero si muy apropiado si se desea modificar o añadir funcionalidades a la consola o a las motas. En nuestro caso recomendamos netbeans 7.1 ya que lleva integrado el plugin para el desarrollo de entornos graficos. Podemos encontrar información al respecto en http://www.guia-ubuntu.org/index.php?title=NetBeans. También podemos instalar el plugin para tinyOS.

Códigos del sistema.

Se adjunta con el documento, una carpeta donde se pueden encontrar los programas necesarios para el sistema:

Ruta carpeta	Contenido
Codigos\AplicacionConsola	Ejecutable de la consola de administración.
Codigos\CodigosFuente\AplicacionConsola	Código fuente de la consola de administración
Codigos\CodigosFuente\CodigosMota	Código fuente de las mota
Codigos\CodigosFuente\CodigosMota\MotaRemota	Codigo Fuente de la que hace de mota remota
Codigos\CodigosFuente\CodigosMota\BaseStation	Codigo Fuente de la que hace de mota estación base

Ejecución del sistema.

Para poner en funcionamiento el sistema debemos realizar los siguientes pasos.

Preparación del entorno descrito en el apartado anterior.

Ingresar en la carpeta Codigos\CodigosFuente\CodigoMota y compilar el código de la mota mediante el comando “make cou24”.

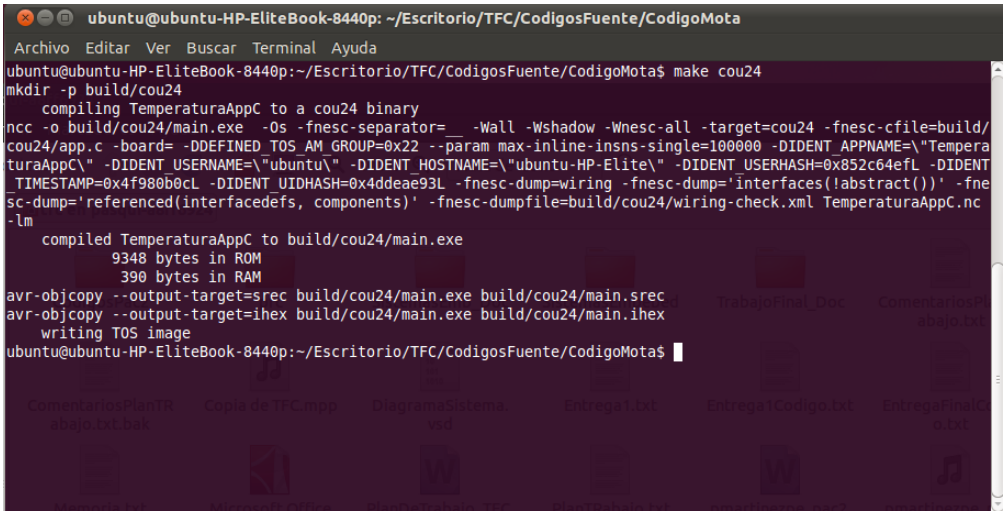


Ilustración 30 Compilación de codigo motas

Transferir el fichero generado a la mota. Dentro de la misma carpeta, donde hemos compilado el código, nos habrá creado una carpeta llamada build. Dentro de esta carpeta, tenemos una carpeta llamada cou24, que corresponde con nuestro dispositivo. Y dentro de ella, tenemos el código compilado, listo para transferir a la mota. El fichero a transferir es el main.srec. Es código binario para nuestra mota. Para transferir el comando a la mota lo haremos desde la carpeta “raíz” (Codigos\CodigosFuente\CodigoMota) mediante el comando “meshprog –t/dev/ttyUSB –f./build/cou24/main.srec”. Una vez ejecutado el comando, hay que pulsar el botón de reset en la mota.

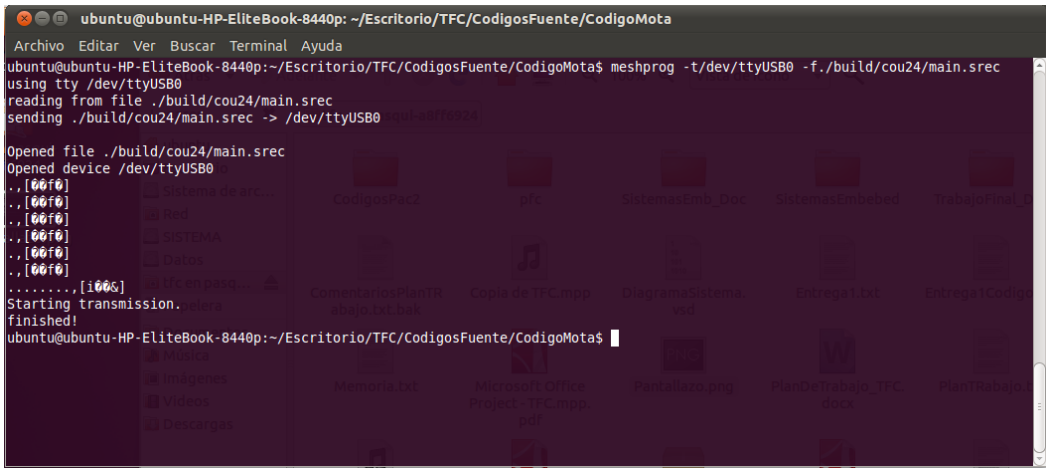
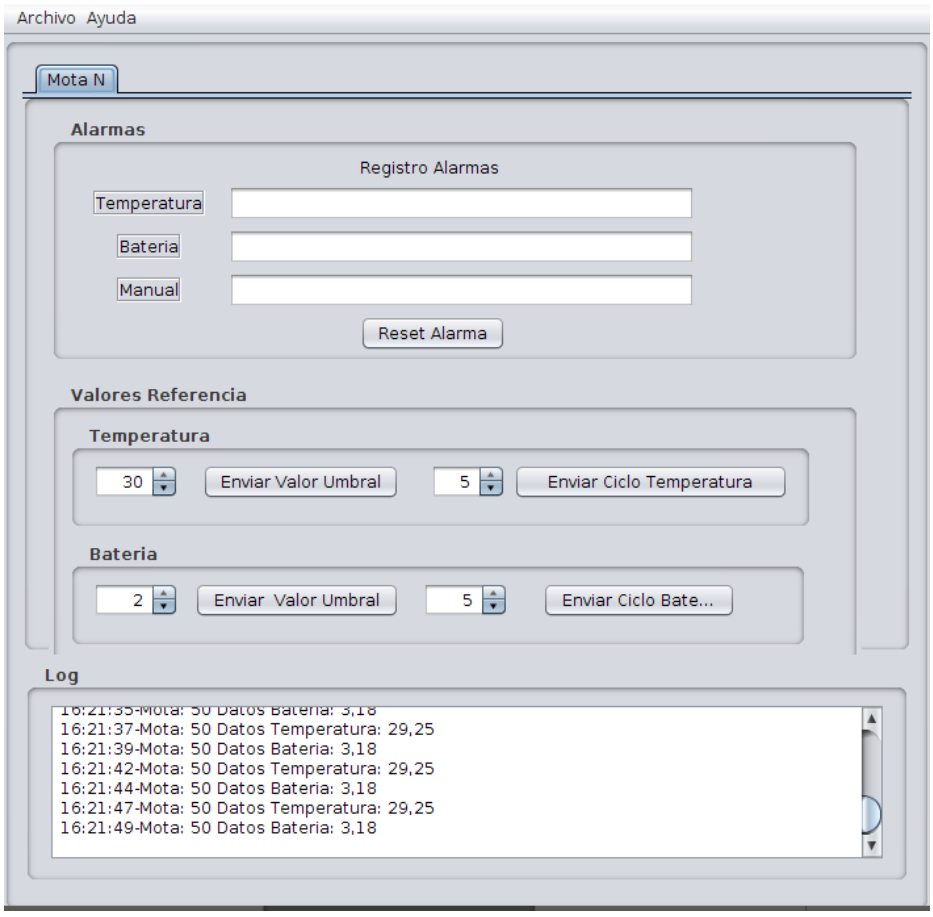


Ilustración 31 Transferencia de programa a la mota

Una vez transferido el código a la mota, los tres leds, se ponen a parpadear, Funcionamiento correcto. Para conectar el sistema, pasar dos veces un iman por el sensor de efecto hall.

Repetiremos el mismo proceso escrito para la estación base, pero esta vez usaremos el código de la carpeta BaseStation.

Ahora vamos a ejecutar la consola de administración. Nos cambiamos de carpeta a Codigos\AplicacionConsola, vamos a ejecutar la aplicación de consola para ver los mensajes que nos envía la mota y poder interactuar con ella. Lo hacemos con el comando java –jar –puerto <puerto donde está conectada la mota:velocidad>. En mi caso es el "serial@/dev/ttyUSB:19200". Para saber en qué puerto está conectada la mota podemos usar el comando "motelist". Finalmente el comando queda como:
"java –jar –puerto serial@/dev/ttyUSB:19200".



Modificación en código fuente de la aplicación

Para modificar el código fuente de la aplicación, podemos hacerlo desde un entorno o desde un editor de textos. He preferido netbeans.

- Creo un proyecto nuevo de netbeans.
- Añado al proyecto la clase principal que nos unirá las piezas del sistema.
- Añado al proyecto las clases java que he generado en el apartado anterior con el jmakefile o el mig.
- Hay que hacer referencia en el classpath de java a la librería tinyos.jar, que se puede encontrar en la carpeta AplicacionConsola\lib del código proporcionado