

Sistemas de Control y Monitorización WSN

Hernán Fernandez Lescano
Ingeniería Técnica Telecomunicaciones esp. Telemática

Consultor: Jordi Becares Ferres

3/1/2012

Sistemas de Control y Monitorización WSN.

En este trabajo final de carrera, se utilizarán los sistemas empotrados para crear un sistema de control y monitorización de una maquinaria por medio de una WSN (Wireless Sensor Network).

Un sistema empotrado, también denominado embebido, es un sistema concebido para realizar una o unas pocas tareas específicas, utilizando todos sus recursos para la realización de las mismas.

Para ello utilizaremos una clase de sistema empotrado denominado “**mota**”, el cual presenta unas características muy importantes como la de un tamaño y peso reducido, autonomía, comunicación inalámbrica y la posibilidad de integrar sensores de diversos tipos.

Estas características nos permiten realizar lecturas en sitios remotos o de difícil acceso, fácil instalación e integración con los sistemas a monitorizar.

Para programar la mota con la funcionalidad deseada se utilizan las herramientas TinyOs y NesC.

El principio de este trabajo se basa en que la mota o las motas, en el caso de mediciones distribuidas, realicen lecturas sobre una maquinaria.

En este proyecto en concreto, se realizará la lectura de las vueltas que da un eje en un intervalo de tiempo y también se leerá la temperatura de un motor.

La mota tomara las distintas lecturas, las procesará y las enviará a la estación base a intervalos de un tiempo predefinido.

La estación base es otro mota que se encuentra conectada a un puerto serie del terminal PC. La misma recibirá la señal enviada por la mota sensor a través de su radio y hará de puente entre el PC y la mota sensor.

En el PC Terminal, una interface nos mostrara por pantalla los datos obtenidos desde los sensores como así también, nos permitirá realizar algunas configuraciones sobre los parámetros de funcionamiento de la aplicación en los mismos.

Índice

	Página
1. Introducción	4
1.1 Justificación	4
1.2 Descripción	5
1.3 Objetivos del Trabajo	6
1.4 Enfoque	7
1.5 Planificación	7
1.6 Recursos Utilizados	11
1.7 Productos Obtenidos	11
1.8 Descripción de los capítulos siguientes	12
2. Antecedentes	13
2.1 Estado del arte	14
2.2 Estudio del mercado	16
2.3 TinyOs y NesC	17
3. Descripción Funcional	21
3.1 Sistema Total	24
3.2 PC	24
3.3 Mota	27
4. Descripción Detallada	29
4.1 Comunicación	29
4.2 PC	30
4.3 Mota	31
5. Viabilidad Técnica	37
6. Valoración Económica	39
7. Conclusiones	41
7.1 Conclusiones	41
7.2 Propuestas de Mejoras	42
7.3 Autoevaluación	42
8. Glosario	43
9. Bibliografía	44
10. Anexos	45
10.1 Manual de usuario	45
10.2 Ejecución y Compilación	46

Índice de Figuras.

Imagen 1. Mota COU24	5
Imagen 2. Esquema general del sistema	7
Imagen 3. Planificación Inicial del TFC	9
Imagen 4. Cronograma Final del TFC	11
Imagen 5. Esquema Básico de un sistema empotrado	14
Imagen 6. Volumen de ventas de microprocesadores	16
Imagen 7. Volumen de ingresos por ventas de microprocesadores	16
Imagen 8. Uso de los microcontroladores	17
Imagen 9. Pagina web de Libelium	18
Imagen 10. Diagrama en bloque de la aplicación.	19
Imagen 11. Mota instalada en la proximidad de un eje	20
Imagen 12. Mota instalada en la proximidad de un eje	21
Imagen 13. Mota instalada en la proximidad de un motor	21
Imagen 14. Recepción de mensajes en la aplicación MoteMonitor	22
Imagen 15. Diagrama de aplicación MoteMonitor	23
Imagen 16. Diagrama de aplicación SetTimer	23
Imagen 17. Diagrama de aplicación SendHello	24
Imagen 18. Diagrama de aplicación SendHello modo instalación	24
Imagen 19. Diagrama de aplicación CheckBat	24
Imagen 20. Módulos que se ejecutan en la mota	25
Imagen 21. Conexión entre dos componentes TinyOs	28
Imagen 22. Trama de un mensaje TinyOs	29
Imagen 23. Componentes de la mota COU24	31
Imagen 24. Conexiones componente TempReaderC	32
Imagen 25. Conexiones componente LectorHallC	33
Imagen 26. Conexiones componente BatReaderC	34
Imagen 27. Conexiones componente ButtonC	34
Imagen 28. Conexiones componente RangeCheckC	35
Imagen 29. Conexiones componente RemoteConfC	35
Imagen 30. Conexiones componente WatchDogC	36
Imagen 31. Extensión cobertura por nodos repetidores	37

Imagen 32. Periodo de recuperación de la inversión 40

Índice de Tablas.

Tabla 1. Estructuras utilizadas para datos por radio 29
Tabla 2. Interfaces y Clases provistas por net.tinyos.messages 30
Tabla 3. Métodos de la Clase MotelF 31
Tabla 4. Coste total de la implantación del sistema 39

1. Introducción.

En este trabajo final de carrera, se realiza la programación de una mota COU24 como la que se muestra en la imagen, por medio de las herramientas TinyOs y NesC, con el objetivo de que funcione como un sensor remoto de una interrupción de campo magnético y como sensor remoto de temperatura.



Imagen 1. Mota COU24

Dicho sensor, se comunicará de manera inalámbrica con un terminal en el cual también se programaron las interfaces correspondientes para llevar a cabo las interacciones con el mismo.

La utilidad de este sistema, será la de satisfacer la necesidad de una empresa genérica, de disponer de una herramienta económica y de fácil instalación para el mantenimiento y monitorización de su maquinaria, con la que se pueda realizar mediciones de velocidades de giro de ejes y temperatura de motores.

1.1 Justificación.

Este trabajo final de carrera se realiza con el objetivo de entrar en contacto con las tecnologías embebidas y familiarizarse con su entorno de programación y utilización, como así también con las comunicaciones inalámbricas.

Saber programar y comunicarse con una mota, la cual se compone de un procesador de uso específico junto con una radio, pone a merced de la creatividad del desarrollador un sinnúmero de aplicaciones basadas en WSN tanto industriales como domésticas.

La utilización de tecnología WSN en adquisición de datos del entorno, nos facilita y reduce el coste de la integración con el sistema a sensar gracias a su características físicas y comunicación inalámbrica.

1.2 Descripción.

El desarrollo del trabajo se basa en los requisitos funcionales de la empresa genérica que se detallaran a continuación, a los cuales se le adhieren requisitos propios con el objetivo de incrementar la calidad del producto final.

Requisitos de la empresa genérica:

Llevar a cabo monitorizaciones sobre ejes y motores, como así también realizar pruebas por parte del personal de mantenimiento, en las distintas maquinas del lugar.

Recopilación de los datos para ser analizados por personal técnico.

Herramienta de fácil portabilidad e instalación como así también un coste aceptable.

Para satisfacer estos requisitos, se utilizaran 2 motas COU24 por maquina a monitorizar. Una para la temperatura del motor, y otra para la velocidad de giro y temperatura ambiental.

En el caso que físicamente se pudiera medir la temperatura y velocidad del eje al mismo tiempo, la mota esta programada para tomar las dos lecturas simultáneamente.

Las motas que funcionarán como sensor remoto, disponen de un sensor de presencia de campo magnético y un sensor de temperatura con su debida programación.

Estas motas se comunicarán de forma inalámbrica con otra mota que hará de estación base para recopilar los datos suministrados y entregarlo al terminal de control, donde se encuentra la interfase de usuario destinada a visualizar los datos.

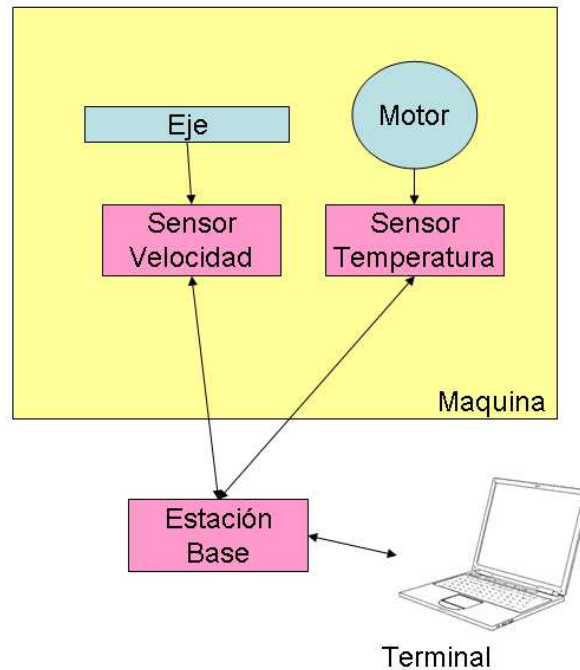


Imagen 2. Esquema general del sistema.

El reducido tamaño de la mota facilita la instalación. Un pequeño soporte aproxima la mota al motor para medir la temperatura en la carcasa. Otro soporte se utiliza también para aproximar la mota a un imán colocado en el eje a medir velocidad, el cual actuará sobre el sensor una vez por vuelta.

Como requisitos adicionales, se han creado alarmas para el nivel de batería, auto reinicio en caso de bloqueo y herramientas para la instalación y configuración.

1.3 Objetivos del Trabajo.

A continuación se detallan los objetivos llevados a cabo en el desarrollo del trabajo. Se destaca como máximo objetivo el hecho de haber realizado una programación modular de las funcionalidades que posee la mota, que luego son integradas en una aplicación de alto nivel. Gracias a este tipo de programación, los módulos pueden ser reutilizados para otras aplicaciones.

Programación de la mota:

- Lectura de los sensores: Obtener los datos de los sensores en el tiempo de muestreo establecido.
- Control de la batería: Leer y comprobar el nivel de la batería.
- Alarmas: Envío de alarmas de lecturas fuera de rango a terminal como visuales.

- Comunicación: Establecer la comunicación con la estación base.
- Configuración Remota: Recibir datos de configuración desde el terminal.
- Auto Reinicio: La mota se reinicia por si sola si dejara de funcionar el programa.

Programación del terminal:

- Creación de la interfase de usuario: Visualización de los datos obtenidos por la mota
- Herramientas de configuración e instalación: Aplicaciones para utilizar conjuntamente con la mota para comprobar coberturas y cambiar variables en la mota.

1.4 Enfoque

Para realizar el trabajo, se ha particionado el mismo en pequeñas tareas, a las cuales se les ha asignado un periodo temporal para su realización.

A su vez, para realizar las tareas, se ha tenido que llevar a cabo un estudio sobre los sistemas operativos, sobre las herramientas de programación y sobre el hardware utilizado.

Para ello se han consultado libros, realizado tutoriales que según la complejidad ha influido directamente sobre el tiempo de realización de las tareas.

Una vez finalizada las tareas de programación y realizados tests en laboratorio, se procede a implantar el sistema en la maquinaria y realizar las pruebas de campo pertinentes.

1.5 Planificación

Durante el desarrollo del trabajo, se han realizado diversas modificaciones respecto a la planificación inicial debido a las distintas dificultades que han ido apareciendo.

Para sortearlas se han tenido que modificar plazos, eliminar tareas opcionales y realizar distintas reestructuraciones.

A continuación se muestra la planificación inicial del trabajo.

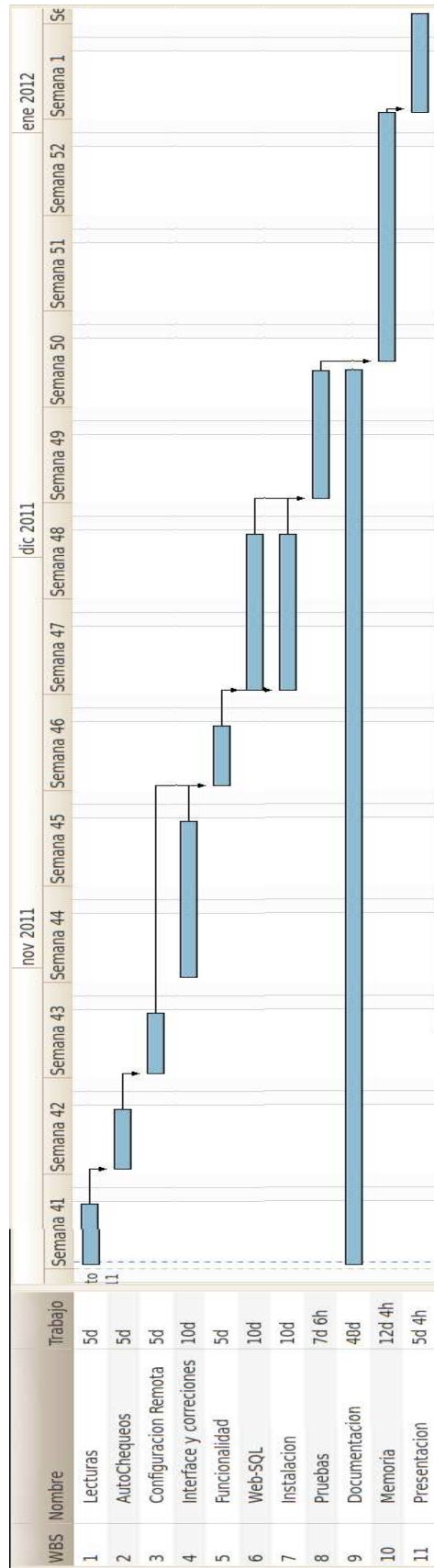


Imagen 3. Planificación Inicial del TFC

En la planificación inicial, podemos destacar las tareas **Lecturas, Autochequeos y Configuración Remota** .

En dichas tareas se ha realizado la programación individual e independiente de los módulos en los que esta dividido el software creado para la mota.

En la tarea **Interface** se desarrolla la aplicación que se ejecuta en el terminal de usuario, en donde se visualizará los datos recogidos por los sensores.

La tarea nombrada **Funcionalidad** consiste en integrar todos los módulos para crear la aplicación que da funcionalidad al sistema.

La tarea **Web-SQL** fue eliminada por su carácter optativo, la cual consistía en volcar los datos obtenidos por los sensores en una base de datos la cual podía ser luego consultada desde Internet.

En la tarea **Instalación**, se ha llevado a cabo la instalación del sistema operativo, el entorno para trabajar con las motas y las aplicaciones en la empresa donde se utilizara el sistema, como así también la instalación de los soportes en la maquinaria.

En la tarea **Pruebas**, se han llevado a cabo las pruebas funcionales en campo las cuales han sido superadas de manera exitosa.

Se puede observar que paralelamente a estas tareas, se ejecuta la tarea denominada **Documentación** donde se ha recopilado información, sobre todo lo acontecido durante el desarrollo del trabajo.

Finalmente, el desarrollo del trabajo ha seguido el siguiente esquema:

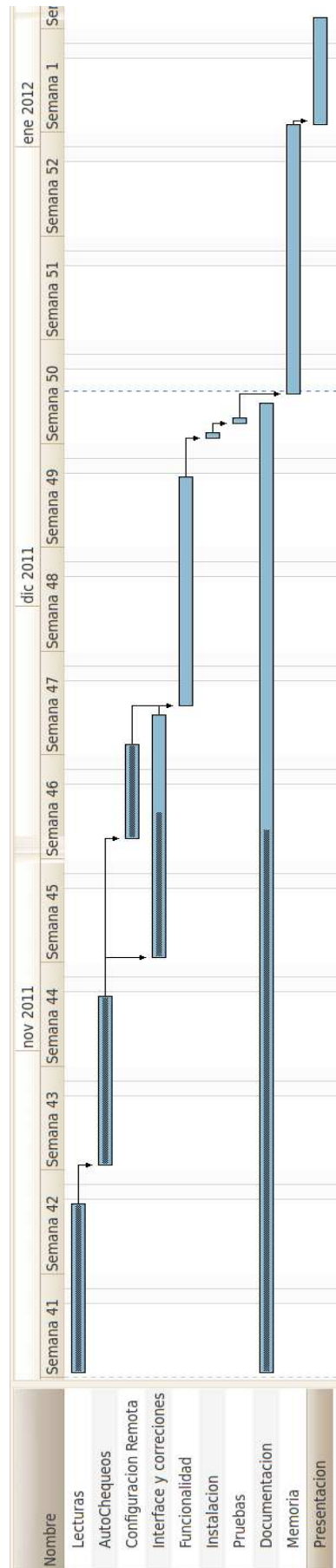


Imagen 4. Cronograma Final del TFC

Como se puede apreciar en el anterior esquema, todas las tareas de programación han requerido mas tiempo del planificado anteriormente, pero se han compensado, debido a la facilidad de instalación que posee el sistema y a que no ha presentado problema alguno durante las pruebas en campo.

1.6 Recursos Utilizados

Las herramientas utilizadas para el desarrollo del trabajo han sido las siguientes:

Programación de la mota:

- Entorno de programación Eclipse con extensiones para TinyOs y NesC
- Sistema Operativo TinyOs programado con lenguaje NesC.
- Aplicación **meshprog** para transferir la programación a la mota.
- Librerías suministradas por TinyOs para NesC.

Programación de la interface:

- Entorno de programación Eclipse con extensiones para Java.
- Lenguaje de programación Java
- Librerías suministradas por TinyOs para Java

Los terminales donde se han llevado a cabo la programación y las pruebas funcionan en bajo el sistema operativo Linux Ubuntu 10.01

Los detalles de la mota se explicarán en el capítulo 3.

1.7 Productos Obtenidos

Los productos obtenidos, son:

Mota Sensor: Sensor inalámbrico inteligente, capaz de enviar una alarma en caso de mal funcionamiento del sistema monitorizado, como así también alertar de un bajo nivel de batería y auto reiniciarse en caso de bloqueo. Permite la posibilidad de configurarse remotamente y dispone de herramientas para la instalación.

Mota Estación Base: Enlace entre el terminal de usuario y el sensor inalámbrico.

Conjunto de aplicaciones para el usuario: Aplicaciones que permiten al usuario recolectar los datos suministrados por la mota como también realizar configuraciones y testeos.

1.8 Descripción de los capítulos siguientes:

En siguiente capítulo, se hará una introducción a los sistemas embebidos o empotrados, para que se utilizan, con que tipos de tecnologías funciona y su importancia en los procesos industriales y en la vida cotidiana de las personas.

El capítulo 3 se da una introducción de cómo funciona el sistema.

El capítulo 4 detallara de manera completa y técnica todo lo referente al desarrollo y funcionamiento del proyecto realizado.

El capítulo 5 y 6 mostrara el análisis de la viabilidad económica y técnica de la implantación del sistema.

El capítulo 7 mostrara las conclusiones obtenidas de la realización del trabajo.

2. Antecedentes

A continuación, se hace una introducción a los sistemas empotrados o embebidos y luego se describirá su situación actual en tecnología y su posición en el mercado.

Sistemas Empotrados: Un sistema empotrado, también denominado embebido, es un sistema computacional, concebido para realizar exclusivamente una o pocas tareas específicas, concentrando todos sus recursos en las mismas.

Esto marca la diferencia conceptual con un sistema de uso general, en el cual sus recursos son compartidos en la realización de distintas tareas de diferentes índoles.

Ejemplo de esta diferencia conceptual los tenemos entre un ordenador y un teléfono móvil. Un ordenador además procesar cálculos, documentos, visualizar imágenes y almacenar información entre otras cosas, también se podría utilizar como un teléfono móvil, pero sería muy incomodo ir con él a cuestras por todos los sitios donde nos movamos.

Aquí es donde entran los sistemas empotrados en juego.

En un sistema empotrado, al utilizar todos los recursos para realizar unas pocas tareas, se puede reducir considerablemente la potencia, el tamaño físico, el consumo y la capacidad de un sistema general, aumentando incluso la eficiencia y rendimiento al realizar dichas tareas. Un teléfono móvil, es un sistema empotrado dedicado a establecer comunicaciones de voz de manera inalámbrica.

La arquitectura de un sistema específico no difiere mucho de uno general. Básicamente se puede agrupar en tres bloques como se muestra en la siguiente figura:

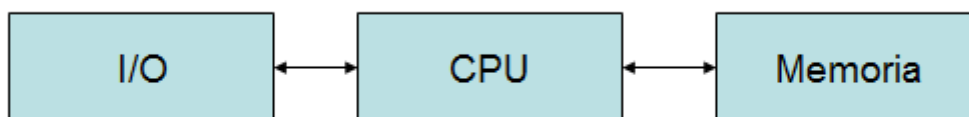


Imagen 5: Esquema básico de un sistema empotrado

CPU: Unidad Central de procesamiento. Se encarga de gestionar la ejecución del programa. Lee las instrucciones de la memoria y realiza las operaciones pertinentes a

partir de datos obtenidos de la memoria o de las interfaces de entradas de los mismos, volcando los resultados en la memoria o en la salida de datos.

En sistemas empotrados, generalmente se utilizan microprocesadores o microcontroladores según la complejidad de las tareas que se quieran realizar.

Mientras que un microprocesador, integra en un chip lo elemental para realizar funciones básicas, un microcontrolador puede integrar en un mismo chip, un reloj propio, convertidores analógicos digitales, puertos de comunicaciones, etc.

Un tipo de microcontrolador muy utilizado en sistemas empotrados es el DSP (Procesador de Señales Digitales) diseñado exclusivamente para procesos de tratamiento de señales como voz, música, video, etc.

Memoria: Dispone de una memoria tipo ROM donde almacenara el programa y constantes de manera no volátil, para que dichos datos no se pierdan al apagar el dispositivo. Actualmente se utiliza mucho la memoria ROM de tipo FLASH.

También posee una memoria tipo RAM, la cual tiene mayor velocidad de acceso que la ROM y se utiliza para trabajar con las variables del sistema y como memoria de trabajo de la CPU.

I/O Entradas y Salidas: Son los elementos necesarios con los que el sistema recibe datos de entrada o emite información sobre los acontecimientos o resultados de los procesos.

2.1 Estado del arte

Para hacer un hincapié en la importancia presente y futura de los sistemas empotrados, es interesante conocer los siguientes datos relativos al volumen de ventas de microcontroladores y microprocesadores utilizados en dichos sistemas en contraposición con los microprocesadores de uso general.

En el grafico siguiente se observa que el 80 % del mercado de microprocesadores y microcontroladores se destina a sistemas empotrados, mientras que el restante corresponde a sistemas de uso general

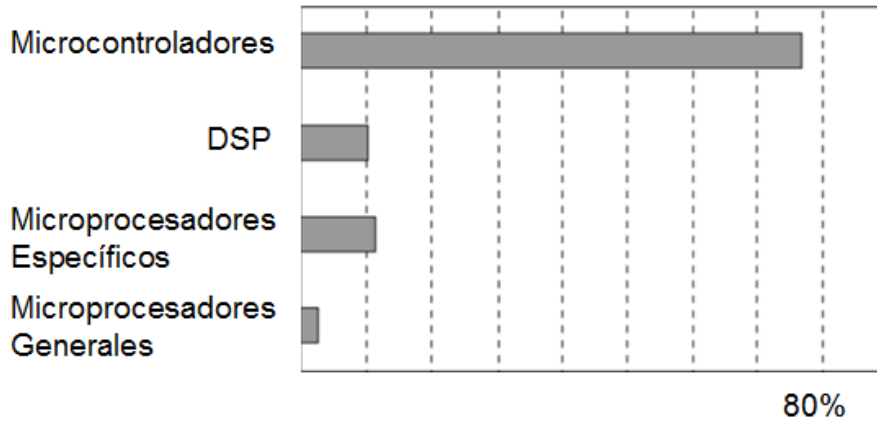


Imagen 6. Volumen de ventas de microprocesadores por categoría

Se observa también en la siguiente gráfica, que los ingresos obtenidos por las ventas de los volúmenes de procesadores anteriormente citados es del 50% respecto de los procesadores para sistemas de uso general.

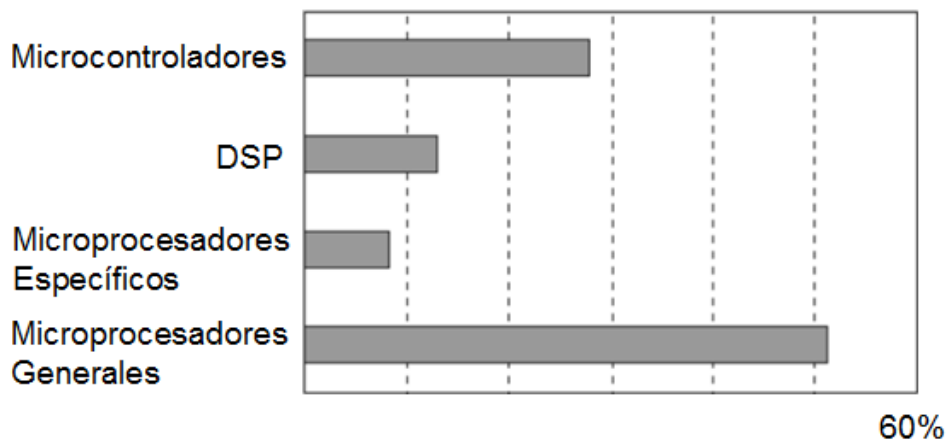


Imagen 7. Volumen de Ingresos por ventas por categorías

De estos datos se puede obtener la conclusión que los sistemas empotrados tienen un gran protagonismo en los dispositivos actuales y a su vez son productos de bajo costo en comparación con los sistemas de uso general.

A continuación una grafica que muestra el porcentaje de uso en los distintos sectores que se favorecen de esta tecnología.

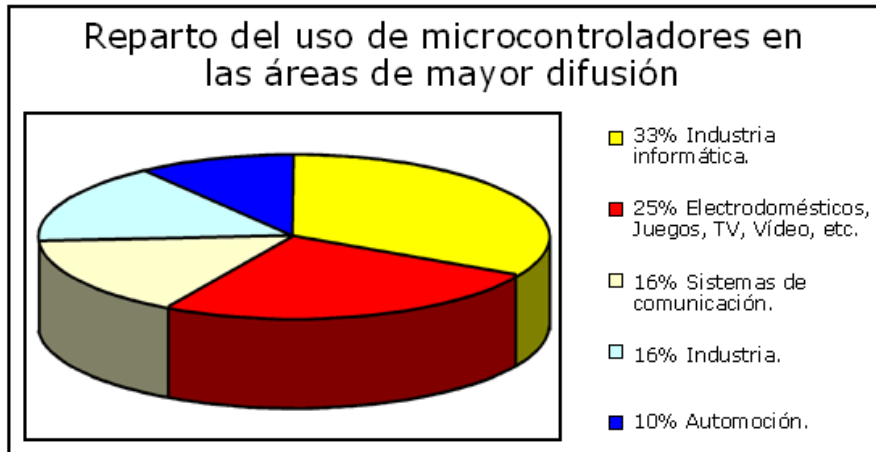


Imagen 8. Uso de los microcontroladores

2.2 Estudio del Mercado

Como se ha comentado anteriormente, las aplicaciones que se pueden diseñar con estos sistemas es innumerable, muchísimas empresas compiten en este mercado en todas sus ramas, como domótica, automoción, sistemas de control, recolección de datos, comunicaciones, etc.

Un ejemplo de empresa que comercializa productos muy similares a los utilizados en el proyecto es Libelium (www.libelium.com), la cual ofrece su gama WaspMote, con la posibilidad de configurar los sensores que deseemos de acuerdo a la utilidad que le queramos dar. La imagen 9 nos muestra uno de sus productos.

Otras empresas que se dedican al desarrollo de sistemas embebidos y microcontroladores son: Crossbow, Intel, Sun Microsystems, BTNode.



Descripción | Hardware | Sensores | OTA | Internet Gateway | Documentación | Comprar

The Sensor Device for Developers




FC CE

Mínimo Consumo

- ▶ Modo hibernate: **0,7uA**
- ▶ Modo Deep Sleep: 62uA
- ▶ Async sensor interruptions
- ▶ Sync timers interruptions

Máximo Rango

- ▶ **ZigBee** ▶ 2.4GHz - 7km
- ▶ 900MHz - 24km
- ▶ 868MHz - 40km
- ▶ Módulo GPRS Cuatribanda
- ▶ Módulo Bluetooth

Máxima Versatilidad

- ▶ Over the Air Programming (OTA)
- ▶ Módulo GPS
- ▶ Memoria SD: 2GB
- ▶ Panel Solar

Sensores

- ▶ Gases (CO, CO2, CH4..)
- ▶ Temperatura, nivel líquido
- ▶ Peso, presión, humedad
- ▶ Luminosidad, acelerómetro
- ▶ Humedad suelo, radiación solar

Open Source

- ▶ Open source API
- ▶ Open source Compiler
- ▶ Completa Documentación
- ▶ Códigos de ejemplo

Videos

- ▶ Accelerometer
- ▶ Gases Sensor Board
- ▶ Events Sensor Board
- ▶ Out of the Box

Imagen 9. Pagina web de Libelium. Resumen de su producto de la gama WaspMote

2.3 TinyOs y NesC

TinyOs es un sistema operativo liviano, especialmente diseñado para sensores inalámbricos de bajo consumo. Su diseño se focaliza en operaciones de bajo consumo.

Está diseñado para los microcontroladores que las motas poseen y ofrece sistemas y mecanismos para ahorro de energía.

TinyOs hace más fácil la creación de aplicaciones de redes de sensores. Provee importantes servicios y abstracciones como sensado, comunicación, almacenamiento y temporizadores y se puede ejecutar en más de una docena de plataformas genéricas.

A alto nivel, TinyOs provee tres elementos para hacer los sistemas y aplicaciones más fáciles:

- **Modelo de componentes:** Permite escribir pequeñas partes de código reutilizable para integrarlas en abstracciones mayores.
- **Ejecución concurrente:** Permite la ejecución de varios componentes al mismo tiempo.
- **APIs, librerías y servicios** que simplifican la escritura de nuevas aplicaciones y servicios.

NesC es un lenguaje de programación orientado a componentes que usan o proveen interfaces, es decir la unidad de trabajo es un elemento, que se comunican con otros e intercambian información a través de sus interfaces.

La sintaxis es similar al C, y la semántica presenta un modelo de eventos e implementación de interfaces.

Estructura de un componente:

- **Configuración:** Su función es la de configuración del componente.
- **Implementación:** También llamada “wiring”. Se vinculan los componentes utilizados por medio de sus interfaces. Un componente provee una interfaz y el otro la utiliza.
- **Modulo:** Contiene la programación de la funcionalidad del componente.

Ejemplo:

La siguiente aplicación enciende el led 0 (rojo).

El componente LedsC provee la interface Leds. A través de esta interface podemos acceder a los comandos que controlan los leds, como por ejemplo *led0on()*.

Entonces mi aplicación ha de utilizar el componente LedsC, como se muestra en la imagen a continuación.

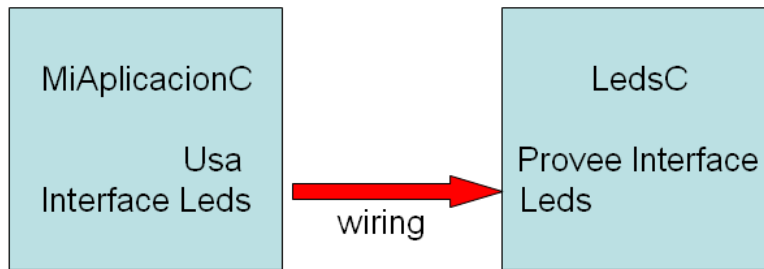


Imagen 10. Conexión entre dos componentes TinyOs.

La configuración e implementación de lo mostrado anteriormente sería algo así:

implementation{

```
//Declaro los componentes que utilizo.
```

```
Components MiAplicacionC,LedsC;
```

```
//Realizo las conexiones "wiring". MiAplicacionC usa la interface Leds del
```

```
//componente LedsC
```

```
MiAplicacionC.Leds -> LedsC.Leds
```

```
}
```

Una vez realizada la conexión, el componente MiAplicacionC ya puede utilizar el comando Led0On(), provisto por el componente LedsC a través de su interface.

3. Descripción Funcional

El sistema desarrollado consiste en realizar lecturas sobre una maquinaria en funcionamiento mediante sensores inalámbricos, obteniendo los resultados de dichas lecturas en tiempo real en un terminal remoto.

3.1 Sistema Total

Tal como se ha explicado anteriormente, los sistemas empotrados tienen la ventaja que al estar concebidos para realizar tareas limitadas de manera eficiente, su tamaño es reducido, brindando la posibilidad de situarlo dentro de una maquinaria sin alterar el funcionamiento de la misma, y dando la posibilidad de realizar mediciones en zonas de difícil acceso para el ser humano, ya sea por espacio o por seguridad. En el caso particular de este trabajo, por ambas razones ya que la maquinaria posee muchas partes móviles.

También el hecho de que la máquina posea muchas partes móviles, dificulta el cableado de los sensores, por lo que se hace uso de su radio, para establecer la comunicación con el terminal.

a) Diagrama general de la aplicación.

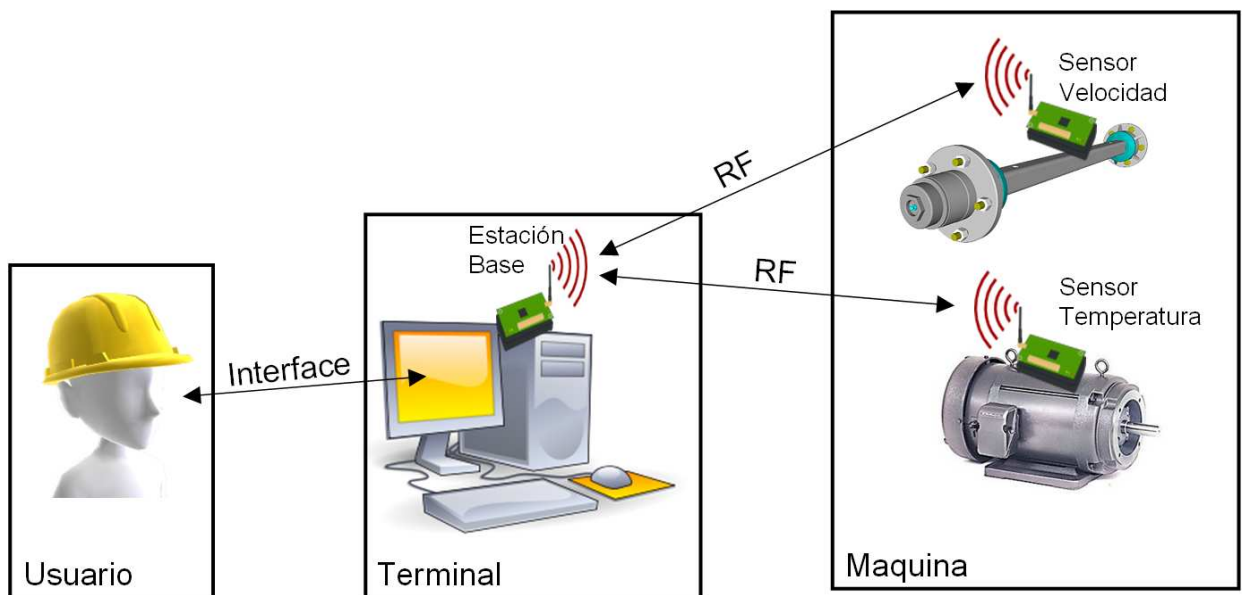


Imagen 11. Diagrama en bloques general de la aplicación

b) Red

La red, es uno de los elementos más importantes del sistema.

Las motas envían su información mediante radiofrecuencia utilizando el protocolo ZigBee a 2,4GHz y envían a la dirección BROADCAST. Esto significa que la información enviada es recibida por todas las radios que escuchen en la frecuencia 2,4Ghz.

c) Interacción de los componentes.

Las motas interactúan con el entorno a través de sus sensores y con el terminal a través de su radio.

A continuación se muestran imágenes de la mota instalada en el eje y en el motor.

Se puede observar también, el imán sujetado en el eje, para crear el campo magnético cuando este pasa por las proximidades del sensor.

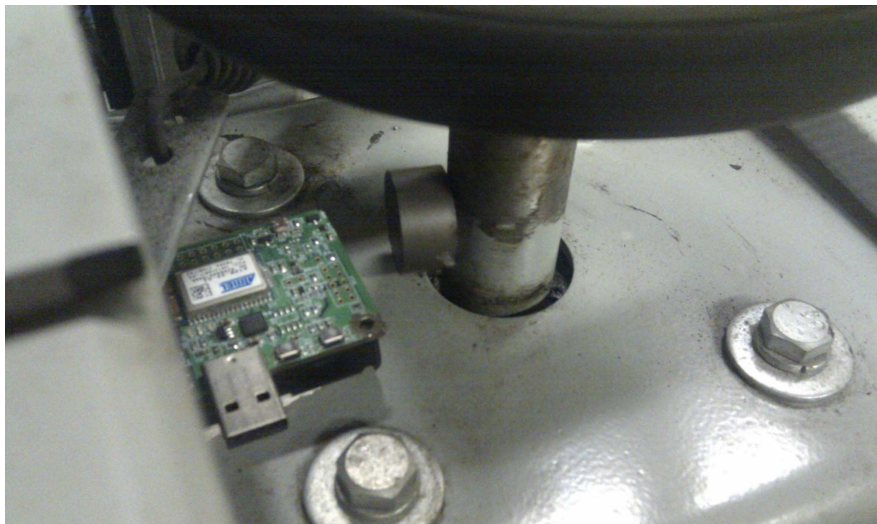


Imagen 12. Mota instalada en la proximidad de un eje.

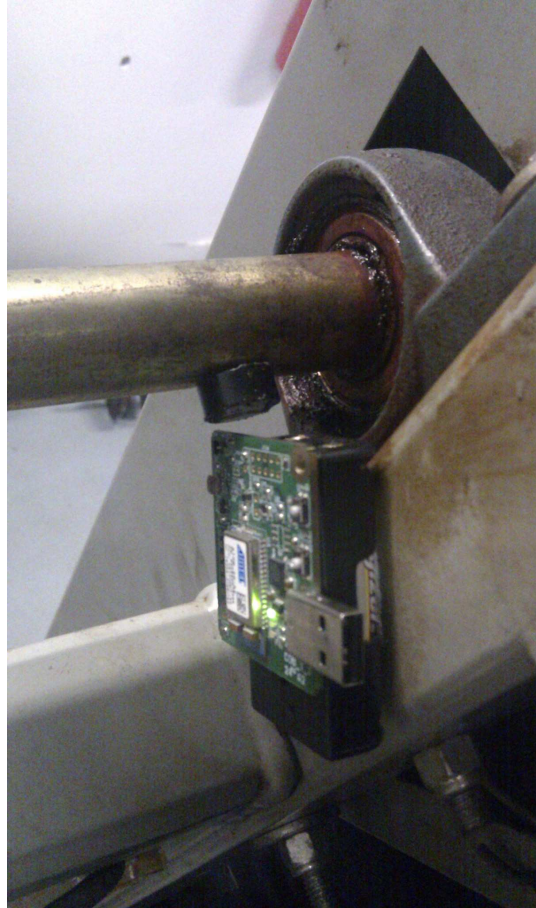


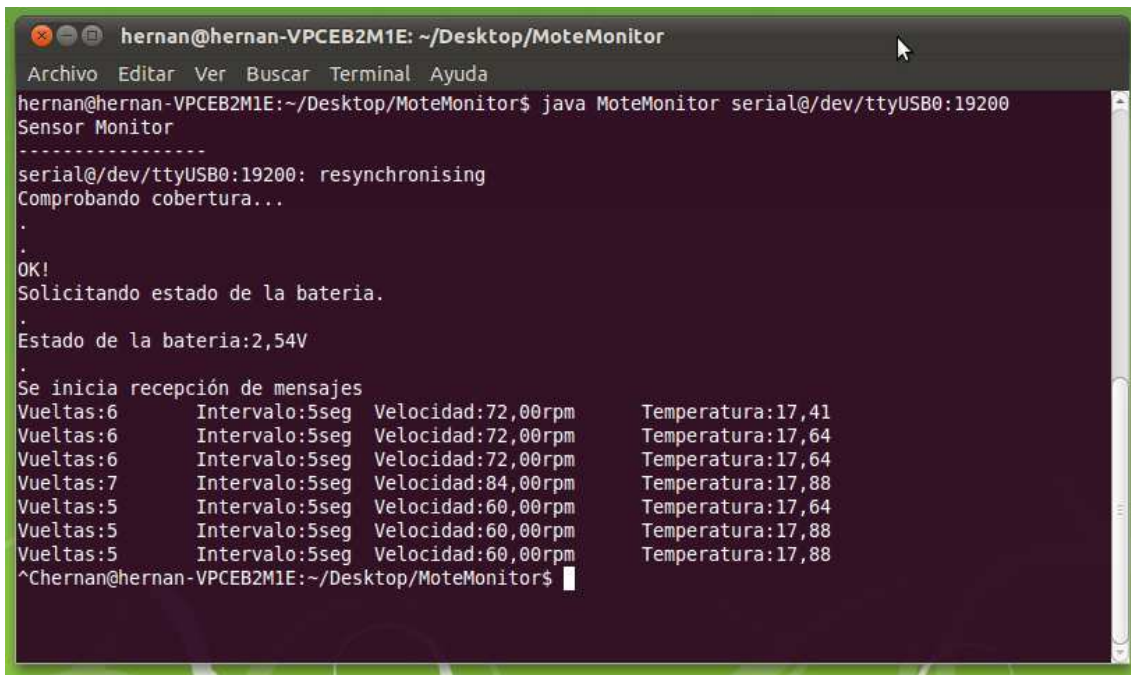
Imagen 13. Mota instalada en la proximidad de un eje.



Imagen 14. Mota instalada en contacto con la carcasa de un motor.

El terminal, interactúa con la mota a través de la radio suministrada por la estación base. Las aplicaciones programadas permiten recibir información de los sensores como enviar datos de configuración.

A continuación se muestra la ejecución del programa principal MoteMonitor, donde se vuelcan los datos recibidos desde las motas.



```
hernan@hernan-VPCEB2M1E: ~/Desktop/MoteMonitor
Archivo Editar Ver Buscar Terminal Ayuda
hernan@hernan-VPCEB2M1E:~/Desktop/MoteMonitor$ java MoteMonitor serial@/dev/ttyUSB0:19200
Sensor Monitor
-----
serial@/dev/ttyUSB0:19200: resynchronising
Comprobando cobertura...
.
.
OK!
Solicitando estado de la bateria.
.
Estado de la bateria:2,54V
.
Se inicia recepción de mensajes
Vueltas:6      Intervalo:5seg  Velocidad:72,00rpm  Temperatura:17,41
Vueltas:6      Intervalo:5seg  Velocidad:72,00rpm  Temperatura:17,64
Vueltas:6      Intervalo:5seg  Velocidad:72,00rpm  Temperatura:17,64
Vueltas:7      Intervalo:5seg  Velocidad:84,00rpm  Temperatura:17,88
Vueltas:5      Intervalo:5seg  Velocidad:60,00rpm  Temperatura:17,64
Vueltas:5      Intervalo:5seg  Velocidad:60,00rpm  Temperatura:17,88
Vueltas:5      Intervalo:5seg  Velocidad:60,00rpm  Temperatura:17,88
^Chernan@hernan-VPCEB2M1E:~/Desktop/MoteMonitor$
```

Imagen 15. Recepción de mensajes provenientes de las motas mediante la aplicación MoteMonitor

3.2 PC

Las aplicaciones que se ejecutan en el Terminal, se han programado utilizando el lenguaje Java, junto con las librerías ofrecidas por TinyOs.

A continuación se describe el funcionamiento de cada aplicación con un diagrama explicativo.

MoteMonitor: Esta es la aplicación principal, recibe los datos de los sensores y los imprime por pantalla, como así también activa el aviso de alarma al personal de mantenimiento y responde con el mensaje “Hola” en caso de recibirlo.

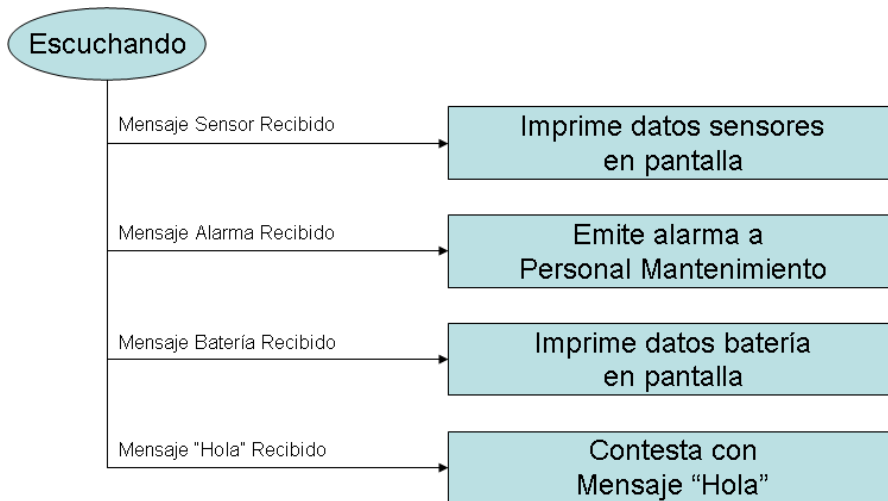


Imagen 16. Diagrama de la aplicación MoteMonitor

SetTimer: Esta función envía un mensaje a los sensores, con la nueva configuración de tiempo de muestreo.

El objetivo de variar el tiempo de muestreo es obtener una mayor precisión en función de la velocidad de giro de un eje. Cuanto mas rápido gire el eje, menor tiempo de muestreo se habría de aplicar.

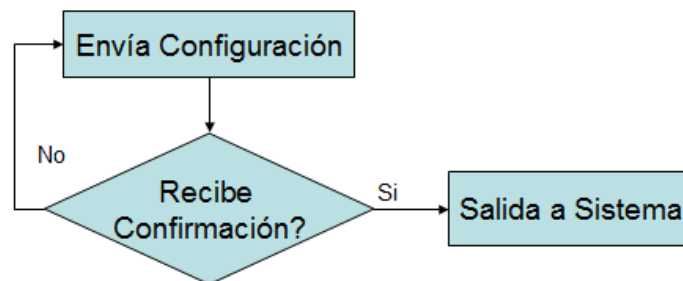


Imagen 17. Diagrama de la aplicación SetTimer.

SendHello: Esta aplicación envía un mensaje Hello hasta que su recepción sea exitosa. Utilizando el parámetro “-i”, se ejecuta el modo instalación, el cual realiza un intercambio continuo de mensajes “Hello” con la mota, hasta que la misma comunique la finalización de la instalación.

En modo normal:

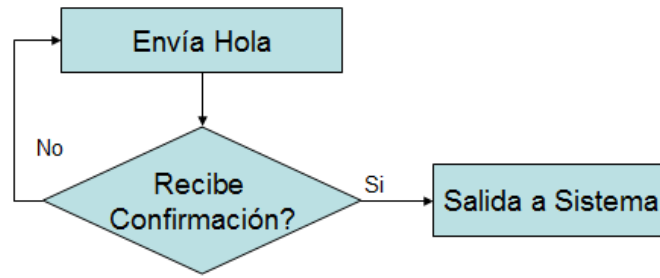


Imagen 18. Diagrama de la aplicación SendHello.

En modo instalación:

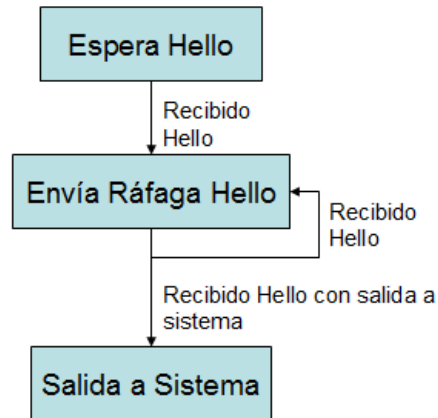


Imagen 19. Diagrama de la aplicación SendHello modo Instalación.

CheckBat: Solicita el estado de la batería.

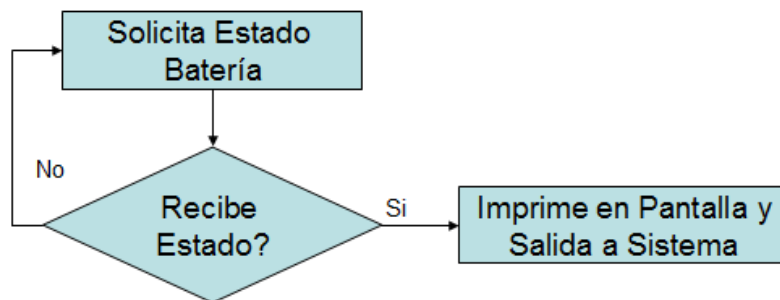


Imagen 20. Diagrama de la aplicación CheckBat.

Todas las aplicaciones son de tipo consola de comandos, no se han implementado funciones graficas.

3.3 Mota

Para la programación de la mota se ha utilizado el lenguaje NesC para operar sobre el sistema operativo TinyOs.

Se ha adoptado para dicha programación una estructura de tipo modular, donde cada módulo realiza una tarea independiente e interaccionan a través de un módulo principal que hace de núcleo del sistema denominado TFCCoreP, tal como se muestra en la siguiente figura:

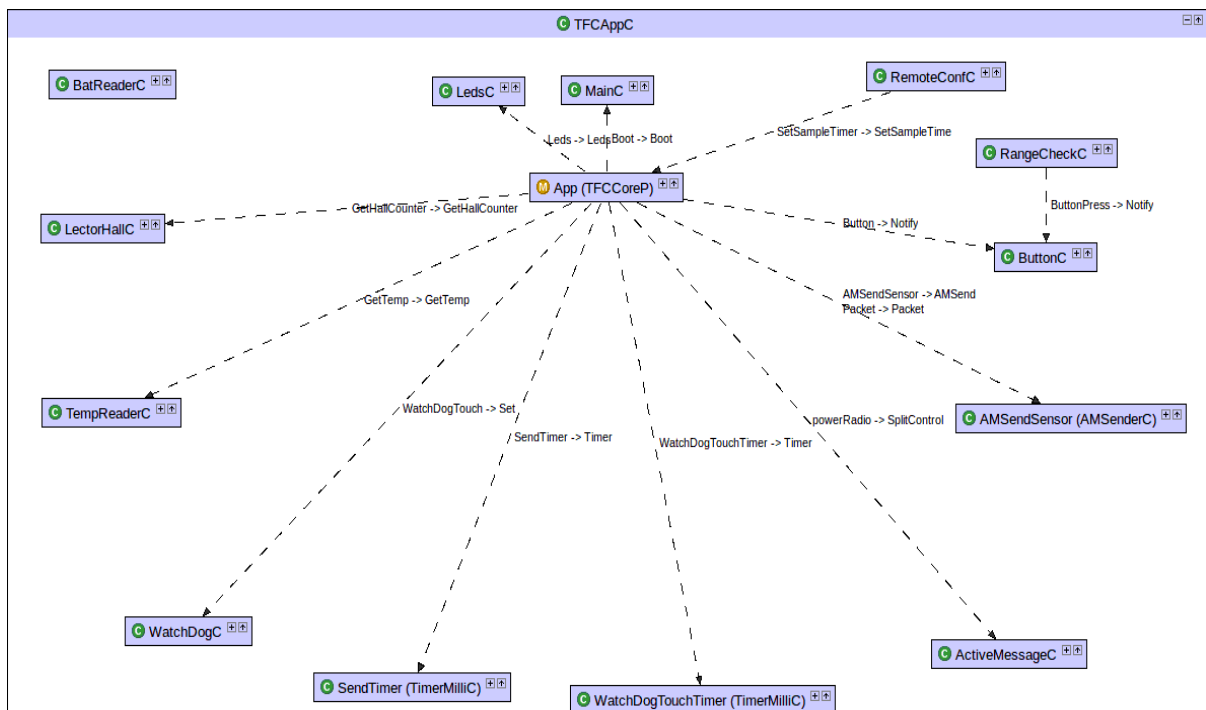


Imagen 21. Módulos de la aplicación que se ejecuta en la mota.

A continuación se da una breve explicación de cada modulo:

TFCAppC: Modulo contenedor de toda la aplicación, vincula los módulos individuales con el módulo TFCCoreP.

TFCCoreP: Modulo que gestiona y organiza las tareas e intercambios de información entre módulos.

MainC: Modulo requerido para señalar a la aplicación que la mota ha booteado correctamente.

LedsC: Modulo que controla los leds.

WatchDogC: Modulo que reinicia la mota en caso de bloqueo de la aplicación.

TimerMilliC: Modulo que controla los temporizadores.

ActiveMessageC: Modulo para controlar la radio y provee herramientas para el uso de la misma como encapsular información en paquetes para poder ser transmitidos.

AMSenderC: Modulo para enviar información a través de la radio.

ButtonC: Modulo que controla el botón de usuario.

RemoteConfC: Modulo que recibe las configuraciones remotas y realiza las modificaciones pertinentes.

BatReaderC: Modulo que lee la batería para enviar el estado por radio. También envía señales de alarma en caso de que detecte un nivel bajo de batería y enciende el led rojo como notificación visual.

TempReaderC: Modulo que lee el sensor de temperatura. Envía señal de alarma en caso que la temperatura exceda un limite definido como máximo y enciende el led rojo como notificación visual.

LectorHallC: Modulo que cuenta las veces que un campo magnético actúa sobre el sensor.

RangeCheckC: Modulo que se utiliza para comprobar la cobertura por medio de mensajes "Hello".

Antes de realizar una explicación detallada del sistema, se hace en el siguiente capítulo, una introducción a las herramientas TinyOs y NesC.

4. Descripción Detallada

En este capítulo se explicará detalladamente los 3 componentes más importantes del sistema. La Mota, la interface y la comunicación.

Empezaremos por la comunicación.

4.1 Comunicación.

La comunicación se lleva a cabo por medio radioeléctrico utilizando el protocolo de comunicación ZigBee, el mismo se encuentra especificado en el estándar IEEE 802.15.4.

El sistema operativo TinyOs, nos proporciona herramientas para poder encapsular y desencapsular mensajes dentro de este protocolo para poder ser propagado por un canal radioeléctrico.

A la información proveniente de la capa de aplicación, se la encapsula en la capa ActiveMessage, acompañada de un identificador denominado AM Type, el cual se utiliza para diferenciar el tipo de mensaje que se está utilizando, facilitando la tarea de trabajar con distintos mensajes independientes en una misma aplicación.

La trama de un mensaje tiene la siguiente estructura:

Dirección Destino	Dirección Origen	Longitud Mensaje	Grupo	AM Type	Payload
2 Bytes	2 Bytes	1 Byte	1 Byte	1 Byte	Hasta 28 Bytes

Imagen 22. Trama de un mensaje TinyOs.

A continuación se detalla las distintas estructuras definidas para el intercambio de mensajes con su respectivo AM Type y las variables y tipos de datos que utilizan.

Estructura	AM Type	Datos	Tipo de datos	Descripción	Tamaño
<i>HelloMsg</i>	41	seqNum	nx_uint8_t	numero de secuencia	8 bits
<i>SensorMsg</i>	42	hallcounter	nx_uint16_t	conteo campo magnético	48 bits
		interval	nx_uint16_t	tiempo muestreo	
		temp	nx_uint16_t	temperatura leída	
<i>BatMsg</i>	43	Bat	nx_uint16_t	nivel de batería	16 bits
<i>ConfigMsg</i>	44	sampleTime	nx_uint16_t	tiempo muestreo	16 bits
<i>AlarmMsg</i>	45	alarmCode	nx_uint8_t	Código de Alarma	8 bits

Tabla 1. Estructuras utilizadas para el intercambio de información a través de la radio.

Estas estructuras quedan definidas en el fichero “Msgs.h” del código desarrollado. Existe una herramienta llamada MIG (Message Interface Generator) que nos crea clases para trabajar en Java y poder acceder a la estructura de los mensajes en el terminal de usuario a partir del fichero mencionado.

4.2 PC

Tal como se ha comentado en el punto anterior, la herramienta MIG nos generara las clases necesarias para poder acceder a los datos de los mensajes.

Las mismas son:

HelloMsg.class, SensorMsg.class, BatMsg.class ConfigMsg.class y AlarmMsg.class.

Para poder utilizarlas basta con importarlas en el código Java que se este desarrollando tal como se ha hecho en este proyecto.

Una vez que sabemos como recoger los datos del mensaje, veremos ahora las herramientas que nos brinda TinyOs para poder capturar los mensajes provenientes de la Radio.

Paquete net.tinyos.message

La siguiente tabla describe el paquete con las clases e interfaces que provee:

Interfaces	
MessageListener	Escucha mensajes tinyos.
Clases	
Message	
MoteIF	MoteIF ofrece una interface Java a nivel aplicación para recibir y enviar mensajes a una mota a través de un puerto serie o alguna otra forma de conectividad.
Receiver	Recibe mensajes tinyos.
Sender	Envia mensajes tinyos.
SerialPacket	

Tabla 2. Interfaces y Clases provistas por el paquete net.tinyos.messages

En este proyecto se ha utilizado la clase MoteIF y la interface MessageListener.

A continuación un resumen de los métodos que posee la clase.

Métodos de la clase MoteIF	
Void	deregisterListener (Message m, MessageListener l) Anula la escucha para un determinado tipo de mensaje.
PhoenixSource	getSource ()
Void	registerListener (Message m, MessageListener l) Registra una escucha para un determinado tipo de mensaje.
Void	send (int moteId, Message m) Envía mensaje a mota.

Tabla 3. Métodos de la Clase MoteIF.

El método **registerListener** configura la escucha para el determinado tipo de mensaje y la interface MessageListener nos avisa por medio de su método **messageReceived** de la recepción de un mensaje.

También se utiliza el método **send** para enviar los mensajes a la mota remota.

De esta manera, se programa la comunicación entre la mota y el PC.

4.3 Mota

En la siguiente imagen, se pueden observar la distribución de los componentes físicos en la mota. Los sensores del entorno se encuentran en la periferia para una mejor recepción.

Se usara el sensor de temperatura y el de presencia de campo magnético.

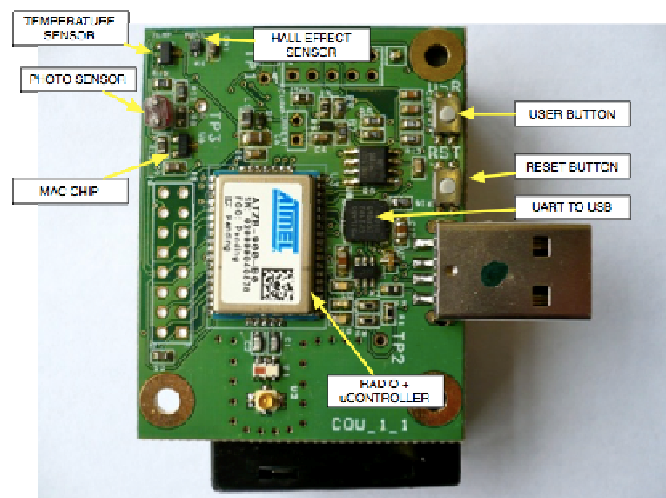


Imagen 23. Componentes de la mota COU24

El microprocesador es un Atmel Atmega1281 de 8 bits y a continuación mostraremos la estructura de los módulos programados para la interacción con sus componentes y con el entorno mediante las herramientas suministradas por TinyOs.

Modulo TempReaderC:

Se utiliza para leer la temperatura. Provee la interface *GetTemp* de tipo *Read*, la cual se utiliza para obtener el valor de las lecturas realizadas por el modulo.

Para obtener las temperaturas, se utiliza el componente *HplAtm128GeneralIO* que provee interfaces del tipo *GeneralIO* para encender los sensores y el componente *AdcReadClientC* para realizar las lecturas a través de la su interface *Read*.

Utiliza un Temporizador para esperar a que los sensores se estabilicen antes de realizar la lectura.

A continuación se muestra un esquema de las conexiones entre módulos.

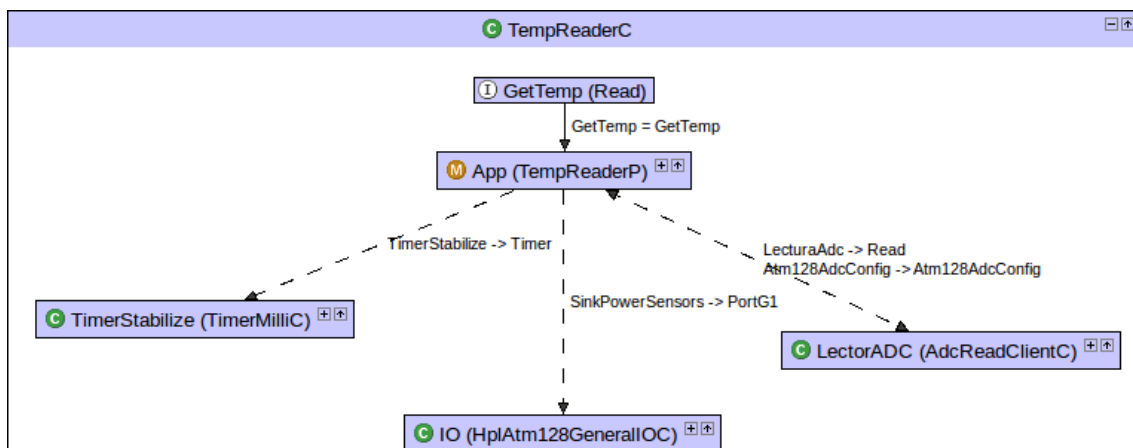


Imagen 24. Conexiones entre componentes del modulo TempReaderC.

Modulo LectorHallC:

Se utiliza para contar las vueltas del eje. Provee la interface *GetHallCounter* de tipo *Get*, la cual se utiliza para obtener el valor de la variable que cuenta las interrupciones provenientes de la interface *HlpAtm128Interrupt*, las cuales se emiten cada vez que se detecta un campo magnético próximo al sensor mediante el evento *fired()*.

Una vez leída la variable se reinicia el contador.

Utiliza el módulo *MainC*, por medio de la interface *Boot*. Cuando el dispositivo bootea, se recibe el evento *booted()*, donde se configuraran las condiciones iniciales del módulo, como los pines que utilizara para leer los datos del sensor magnético.

También utiliza el módulo *LedsC* para alternar el led verde cada vez que se detecta la existencia de campo magnético.

El módulo *HplAtm128GeneralIOC* se utiliza para encender los sensores a través de la interface *GeneralIO*

El *TimerMilliC* se utiliza para dar tiempo a que los sensores se estabilicen antes de realizar la lectura.

A continuación se muestra un esquema de las conexiones entre módulos.

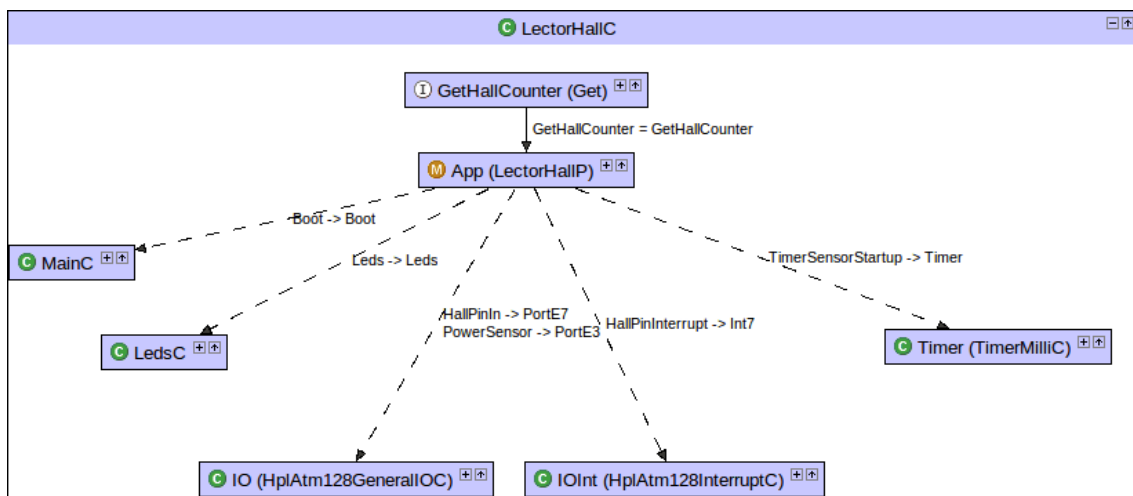


Imagen 25. Conexiones entre componentes del módulo LectorHallC

Modulo BatReaderC:

Se utiliza para obtener el estado de la batería. Es un módulo autónomo, es decir, no está controlado por la aplicación principal.

Tal como se ha visto en los anteriores módulos, se utiliza *HplAtm128GeneralIOC* y *AdcReadClientC* para realizar el encendido y lectura de los sensores.

Utiliza temporizadores *TimerMilliC* para tiempo de muestreo y *LedsC* para manipular los leds.

También utiliza la radio. Para ello hace uso de los módulos *AMSenderC* con su comando *send* para enviar, *AMReceiverC* con su notificación de recepción de mensajes y el módulo *ActiveMessageC*, para encender la radio por medio de la interface *SplitControl*.

A continuación se muestra un esquema de las conexiones entre módulos.

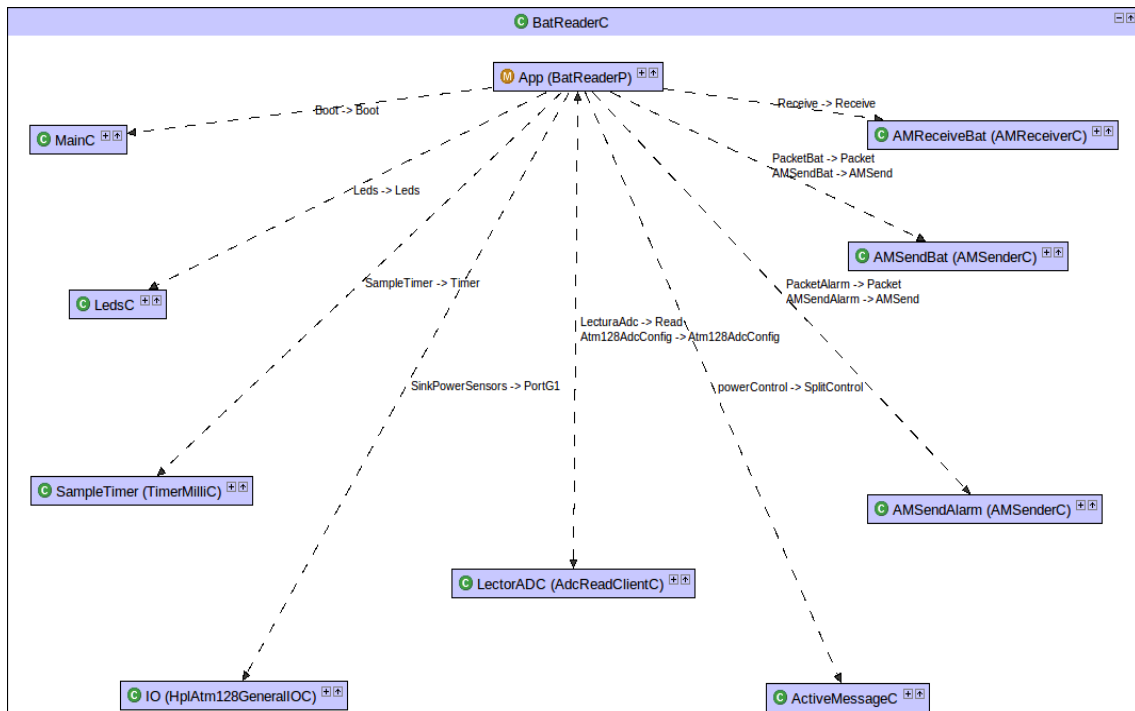


Imagen 26. Conexiones entre componentes del módulo BatReaderC.

Modulo ButtonC:

Este modulo configura los pines donde esta conectado el botón de usuario. Utiliza la interfase *GeneralIO* del componente *HplAtm128GeneralIOC* para encender el sensor y la interface *HplAtm128Interrupt* informa de que el botón ha sido apretado por medio de una interrupción.

Cuando sucede esto, el modulo, envía una señal de notificación a la aplicación superior por medio de la interface que provee, la cual es de tipo *Notify*.

A continuación se muestra un esquema de las conexiones entre módulos.

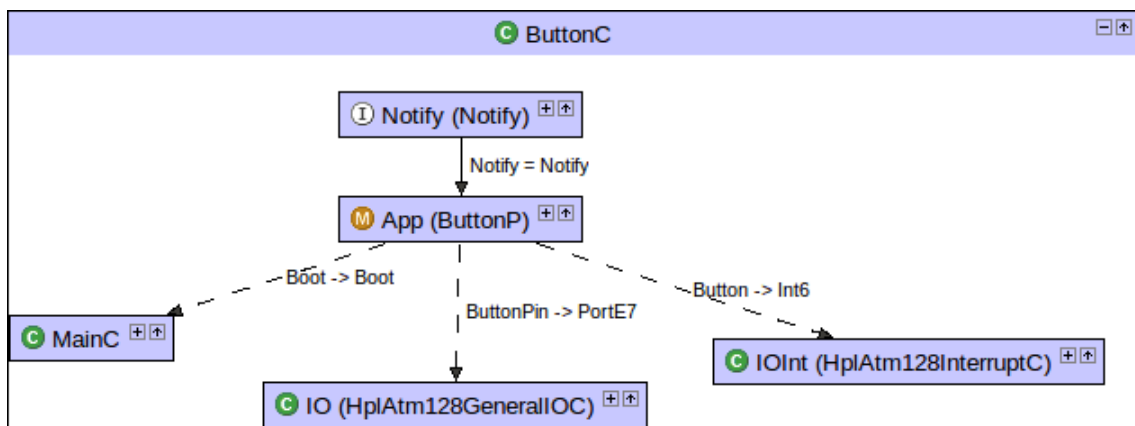


Imagen 27. Conexiones entre componentes del módulo ButtonC.

Modulo RangeCheck:

Este modulo se utiliza para intercambiar mensajes de tipo “Hello”.

Utiliza el componente *MainC* para obtener la señal de que el sistema a booteado por medio de la interface *Boot*. De esta manera se enciende la radio en el inicio por medio del componente *ActiveMessageC*.

Utiliza la radio para enviar recibir mensajes “Hello” a través de las interfaces *Send* y *Receive* de los componentes *AMSenderC* y *AMReceiverC* .

También utiliza un temporizador de tipo *TimerMilliC*, para temporizar el envío de los mensajes.

A continuación se muestra un esquema de las conexiones entre módulos.

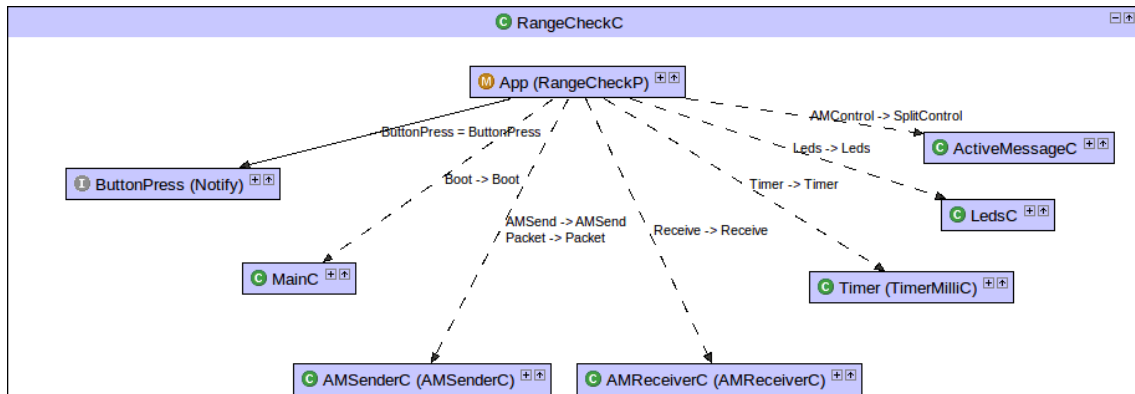


Imagen 28. Conexiones entre componentes del modulo RangeCheckC.

Modulo RemoteConfC:

Este modulo recibe a través de la radio valores de configuración que son enviados desde el terminal.

Para setear estos valores utiliza la interfaz *Set*

A continuación se muestra un esquema de las conexiones entre módulos.

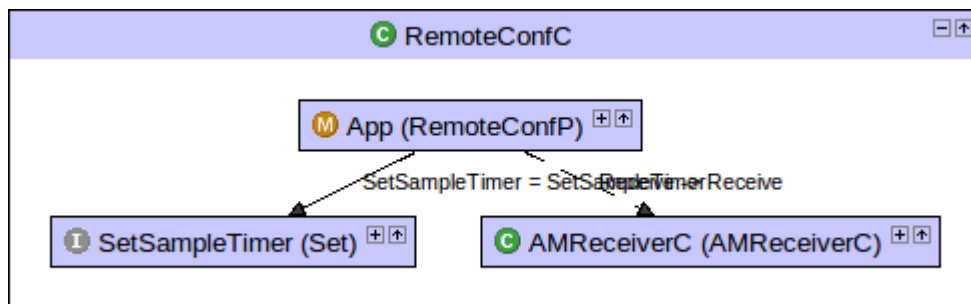


Imagen 29. Conexiones de los componentes del modulo RemoteConfC.

Modulo WatchDogC

Este módulo se encarga de reiniciar el sistema en caso de que la aplicación se bloquee. Funciona con un temporizador a bajo nivel que ha de ser reseteado por la aplicación para que no se ejecute el reinicio.

Por el contrario, si la aplicación se bloquea, no se resetea el temporizador y por lo tanto ejecuta el reinicio.

Para resetear el temporizador, la aplicación hace uso de la interface *Set*.

Utiliza el componente *MainC* para obtener el punto de partida en el booteo y activar el temporizador.

A continuación se muestra un esquema de las conexiones entre módulos.

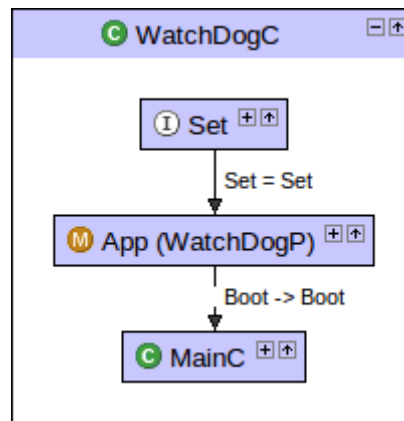


Imagen 30. Conexiones de los componentes del módulo WatchDogC.

5. Viabilidad Técnica.

La versatilidad ofrecida por el reducido tamaño de la mota, la comunicación inalámbrica y su bajo consumo eléctrico, hace que la implantación de un sistema basado en estas tecnologías sea técnicamente viable.

Como puntos fuertes se destacan la facilidad de instalación física. Un simple soporte permite colocar el dispositivo prácticamente en cualquier sitio, dados su reducido tamaño y peso.

Su comunicación inalámbrica puede ser extendida mas allá del rango de cobertura de la estación base por medio de una repetición de los mensajes tal como se muestra en la siguiente figura.

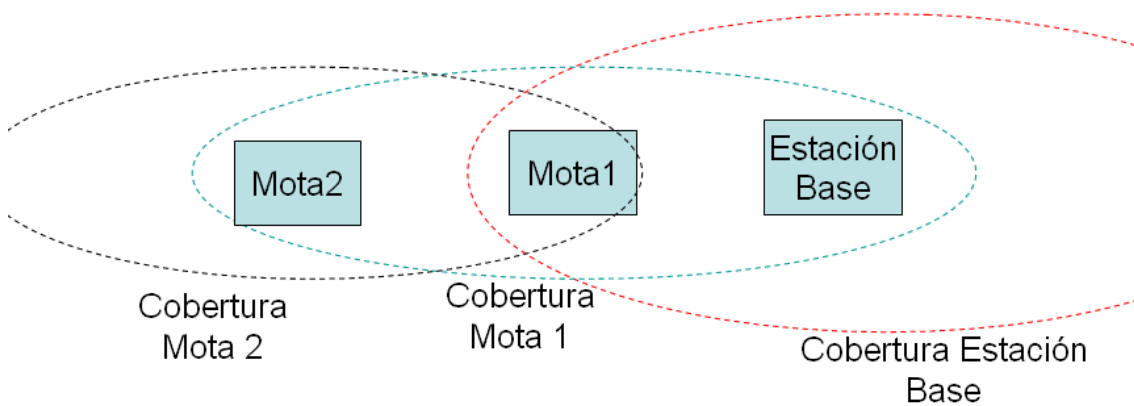


Imagen 31. Extensión de cobertura de la estación base por medio de nodos repetidores.

La mota2 no puede comunicarse directamente con la estación base porque se encuentra fuera de su área de cobertura, sin embargo la mota1 se encuentra dentro de la zona de alcance de la estación base y de la mota2 simultáneamente, dotándose de la posibilidad funcionar como nodo repetidor, al mismo tiempo de realizar su función específica en el sistema.

También se destaca la posibilidad de intercambiar parámetros de configuración de manera remota, de esta manera, se pueden realizar ajustes sobre algunas variables para tener un sistema mas preciso.

Como puntos débiles, la modificación o actualización del sistema no puede hacerse de forma remota, lo que obliga a tener que desinstalar físicamente todos los nodos para conectarlos al terminal de programación. No obstante actualmente existen

dispositivos, de mayor coste obviamente, que ofrecen la posibilidad de programación remota.

6. Valoración Económica.

A continuación se realizara una valoración económica en el supuesto que la empresa que solicita el trabajo posee 24 maquinas a monitorizar.

El precio de cada mota se estima en 50€. Se necesitarán dos por maquina.

Se recomienda disponer de 2 terminales PC con idéntico funcionalidad, pero uno de ellos sea un portátil. Esto permitirá al personal de mantenimiento, obtener las lecturas in situ si se esta trabajando sobre la maquinaria, con lo que no tendrá que realizar desplazamientos hasta el terminal de monitorización. El coste por terminal se estima en 800€.

Tal como se ha comentado anteriormente, el coste de instalación es muy reducido debido a la facilidad de esta tarea.

Estimaremos la instalación de las motas en la maquinaria mas realización de pruebas en dos días de trabajo, con dos técnicos en plantilla de la empresa, con un coste aproximado de 240€.

El desarrollo del software, para un programador con experiencia en estos sistemas es abrumadamente inferior al que se ha requerido en este trabajo.

Podemos estimar el tiempo de desarrollo en 40 horas a un precio de 20€ por hora con un coste de 400€.

El mantenimiento de las motas es prácticamente despreciable.

Entonces se resume el costo total de la implantación en la siguiente tabla.

<i>Descripción</i>	<i>Cantidad</i>	<i>Precio Unidad</i>	<i>Total</i>
Terminal PC	2	800 €	1.600 €
Motas	50	50 €	2.500 €
Desarrollo	40	20 €	800 €
Instalación	2	120 €	240 €
Coste Total			5.140 €

Tabla 4. Coste Total de la Implantación del sistema

Como se puede ver, la implantación de sistemas empotrados presenta un coste reducido.

Este sistema facilitara las tareas de mantenimiento, reduciendo el tiempo establecido para su realización, como así también detectará y alertará sobre malfuncionamientos,

para que se pueda actuar a consecuencia y realizar los ajustes necesarios para prevenir el deterioro de la maquinaria.

Todo esto llevara a reducir el coste de mantenimiento. Dado el alto coste de las piezas de la maquinaria, se puede estimar que el presupuesto de mantenimiento se reduce en 250€ euros mensuales.

Esto llevara a un periodo de recuperación de la inversión de 20 meses. Esto nos da una valoración económica positiva y que la inversión es rentable.

A continuación mostramos un gráfico con el periodo de recuperación.

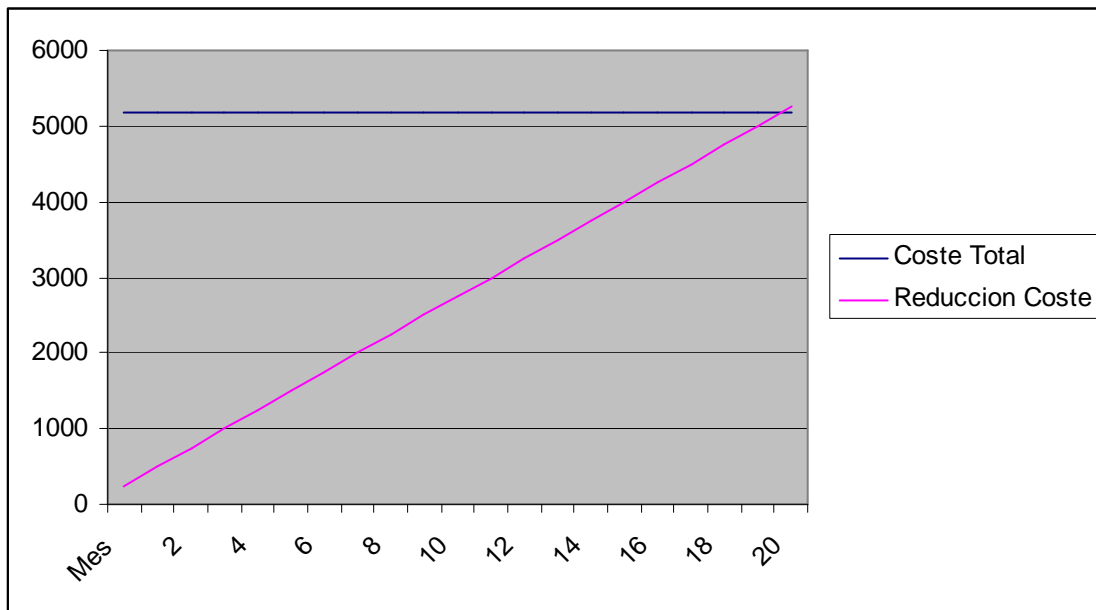


Imagen 32. Periodo de recuperación de la inversión

Otra valoración positiva que podemos hacer, es que pasado el periodo de recuperación, la empresa dispondrá de mayor tecnología en su patrimonio y dispondrá de los sistemas empotrados para reutilizarlos cuando quiera, en su amplio espectro de aplicaciones.

7. Conclusiones

Se han conseguido todos los objetivos principales.

El principal objetivo cumplido, fue el de desarrollar una aplicación para la mota que pudiera recolectar datos de una maquinaria y transmitirlos a la estación base, y visualizar estos datos en el terminal por medio de la interface correspondiente la cual era otro de los objetivos que se han conseguido.

Como objetivos adicionales, se ha dotado a la mota de diversas funcionalidades como la de hacerla mas robusta frente a posibles bloqueos, envío de alarmas de malfuncionamiento de la maquinaria y ayudas de instalación como comprobaciones de cobertura.

No se han podido realizar los objetivos opcionales como el volcado de datos a una base de datos y su visualización a través Internet por falta de tiempo, pero los mismos no presentan mayores dificultades ya que existen potentes herramientas para ello.

En este trabajo final de carrera, se obtuvieron los conocimientos necesarios para desarrollar aplicaciones para sistemas WSN. Se ha entrado en contacto con esta tecnología para experimentar sus posibilidades y limitaciones.

Se ha conseguido desarrollar un proyecto basado en estos sistemas y ponerlo en funcionamiento demostrando todos los conocimientos adquiridos.

7.1 Conclusiones

- Los sistemas empotrados son óptimos para realizar tareas específicas, con mayor eficiencia y menor coste que un sistema de uso general.
- Las motas son un tipo de sistema empotrado, que gracias a su autonomía, comunicación inalámbrica y la posibilidad de integrar sensores de todo tipo nos ofrece innumerables utilidades. También nos permite instalarlo fácilmente donde se quiera adquirir datos del entorno de manera remota.
- TinyOs y NesC son herramientas para desarrollar aplicaciones sobre las motas. Se obtuvieron los conocimientos necesarios sobre estas herramientas para programar la mota y hacerlo de manera modular y eficiente, con la posibilidad de poder reutilizar el código en otras aplicaciones.

- Los proyectos basados en WSN tienen un altísimo porcentaje de ser viables técnica y económicamente.

7.2 Propuestas de Mejora.

- Mejorar el consumo. La mota envía señales periódicamente, por lo tanto hace mediciones y uso de la radio de manera constante, incluso cuando la maquinaria no está en uso. Esto representa un consumo de energía innecesario. El cuidado de la batería es muy importante para la eficiencia del sistema.

Por lo tanto, se propone como mejora, que la mota no envíe mensajes ni realice lecturas cuando la maquinaria no se encuentra en uso. Como así también, apagar los sensores entre lectura y lectura.

- Mejorar la interface de usuario. Esto escapa un poco a los sistemas empotrados y se acerca más a la informática, pero sería interesante disponer de una interface gráfica para ser más amena las lecturas y la interacción con las motas.

7.3 Autoevaluación.

La autoevaluación es totalmente positiva. Se ha realizado un estudio profundo de cómo programar las motas, adquiriendo los conocimientos para llevar a cabo el desarrollo de cualquier tipo de aplicación.

Se valora positivamente la configuración de los entornos de programación para trabajar con los lenguajes de programación de las motas.

Se ha trabajado con las motas a nivel físico, realizando pruebas en campo y funcionamiento real, aportándonos la experiencia para realizar la instalación de estos sistemas.

8. Glosario.

Broadcast: Un mensaje enviado a BroadCast, es escuchado por todos los nodos de la red.

Eclipse: Es un entorno de desarrollo integrado multiplataforma (IDE). Las IDE son herramientas que utilizan los programadores para facilitar la escritura de código.

Standard IEEE: Estándar definido por el instituto de Ingenieros en Eléctrica y Electrónica.

Linux Ubuntu 10.01: Una de las distribuciones del sistema operativo Linux, compatible con las herramientas suministradas por la versión de TinyOs utilizadas en este proyecto.

Java: Lenguaje de programación orientado a objetos.

Meshprog: Aplicación para transferir el programa que ejecuta la mota desde el PC.

NesC: Lenguaje de programación orientado a eventos, utilizado para programar la funcionalidad de la mota.

Protocolo: conjunto de estándares que controlan la secuencia de mensajes que ocurren durante una comunicación entre componentes de una red.

TinyOs: Sistema operativo concebido para funcionar sobre motas, suministrando herramientas para su programación y utilización.

WSN: Wireless Sensor Network. Red formada por sensores inalámbricos.

ZigBee: Estándar de comunicación inalámbrico, utilizado mayoritariamente por equipos de bajo consumo como las motas utilizadas en este trabajo.

9. Bibliografía.

Web oficial de TinyOS, Tutoriales y Documentos TEPS y toda la documentación de los componentes e interfaces.

<http://www.tinyos.net>

Wiki UOC y datasheets

<http://cv.uoc.edu/app/mediawiki14/>

TinyOS programming – Cambridge - de Philip Levis

Teach Yourself Java 6 in 21 Days – Sams – Rogers Cadenhead

NesC 1.1 manual de referencia

<http://nesc.sourceforge.net/papers/nesc-ref.pdf>

Wikipedia Sistemas embebidos

http://es.wikipedia.org/wiki/Sistema_embebido

Material Didáctico UOC: Sistemes encastrats.

Material Didáctico UOC: Sistemes electrònics digitals.

Wireless Sensor Network Research Group.

<http://www.sensor-networks.org/>

10. Anexos.

10.1 Manual de usuario.

El producto final consta de las motas con el programa previamente instalado, y un fichero comprimido llamado MoteMonitor que contiene las aplicaciones para ejecutar en el terminal de usuario.

a) Instalación del software

- Descomprimir el fichero MoteMonitor en el directorio de deseado.
- Ejecutar ./instalar

Nota: Hay que asegurarse que la variable de entorno CLASSPATH contenga el directorio “/.”, de lo contrario, la instalación no podrá llevarse a cabo.

b) Instalación de las motas.

Ejecutar en el terminal la aplicación SendHello en modo instalación.

Para realizarlo con el prompt en situado en el directorio MoteMonitor escribir:

```
>java SendHello fuente_de_datos -i
```

Donde *fuente_de_datos* es el dispositivo donde esta dirigido el flujo de datos proveniente de la estación base y tiene el siguiente formato: serial@/dev/ttyX:19200 donde 19200 es el baudrate y ttyX es el fichero de dispositivo ej: ttyUSB0.

Luego se presiona el botón de usuario de la mota, para ponerla en modalidad instalación. Se encenderá el led rojo. El led verde comenzara a titilar siempre que la mota se encuentre en cobertura. De esta manera presentamos la mota en el sitio donde se fijara y comprobamos si se encuentra en el área de cobertura.

Ya podemos fijar la mota. Apretamos nuevamente el botón de usuario para quitar la mota del modo instalación.

Tal como se muestran en las imágenes 12 y 13, para medir la velocidad de un eje, la mota se debe situar en la proximidad de un campo magnético generado por un imán el cual estará fijado al eje en cuestión.

Para medir la temperatura, se deberá aproximar la mota lo máximo posible con la carcasa del motor para obtener una lectura más certera.

Una vez fijada la mota en la maquinaria, ya se puede empezar a visualizar lecturas por medio de la aplicación MoteMonitor.

10.2 Ejecución y Compilación.

A continuación se detalla la manera de ejecutar el kit de aplicaciones suministrado.

MoteMonitor: Aplicación que recibe la información de los sensores. Se ejecuta de la siguiente manera:

```
> java MoteMonitor fuentes_de_datos
```

CheckBat: Aplicación que solicita a la mota el estado de la batería

```
> java CheckBat fuentes_de_datos
```

SetTimer: Envía el nuevo tiempo de muestreo deseado al que funcionara la mota.

```
> java SetTimer fuentes_de_datos tiempo(ms)
```

SendHello: Envía mensaje “hello” a la mota, para comprobar cobertura

```
> java SendHello fuentes_de_datos
```

Donde *fuentes_de_datos* es el dispositivo donde esta dirigido el flujo de datos proveniente de la estación base y tiene el siguiente formato: `serial@/dev/ttyX:19200` donde 19200 es el baudrate y `ttyX` es el fichero de dispositivo ej: `ttyUSB0`.

Reinstalación del software de la mota.

En caso que la mota perdiera el software, para reinstalarlo se ha de ejecutar el siguiente comando.

```
> meshprog -t /dev/ttyX -f ./build/sensor/main.srec en caso del sensor
```

```
> meshprog -t /dev/ttyX -f ./build/baseStation/main.srec en caso de la BaseStation
```

Donde `ttyX` es el fichero de dispositivo ej: `ttyUSB0`

Luego se ha de apretar el botón de reset de la mota, para que se inicie la transferencia del programa.