SIT-WSN: Sistema de Información de Tráfico mediante WSN

Antonio Suárez Reyes
Ingeniería Técnica de Telecomunicaciones. Especialidad Telemática

Director del proyecto: Sebastià Cortés Herms
7 de junio de 2012

Índice general

Índice de figuras

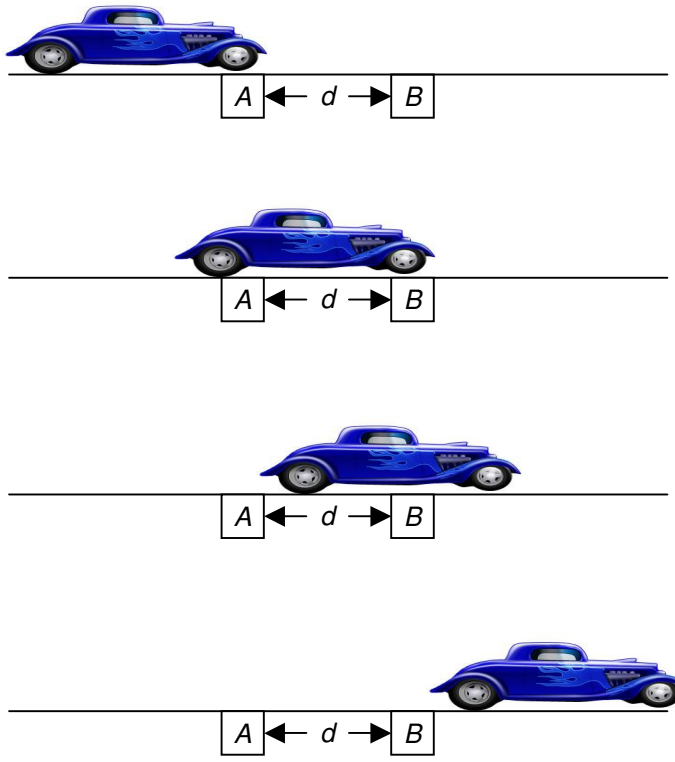
1. Introducción

A la hora de realizar el presente trabajo hemos perseguido un doble objetivo: de una parte, desarrollar los conocimientos y capacidades previstas en relación a los sistemas empotrados y, de otra, siguiendo las indicaciones de la wiki de la asignatura[1], ser ambiciosos en su consecución.

En este sentido, disponemos de dispositivos (motas) dotados de sensores de temperatura, luminosidad y efecto Hall que, a su vez, permiten conocer el estado de la tensión de alimentación y comunicarse entre ellos mediante tecnología ZigBee, formando una red inalámbrica de sensores (WSN).

De esta forma, situaremos dos motas a una distancia conocida bajo un carril translúcido por el que circulan objetos móviles. Con esta configuración calcularemos la velocidad de dichos objetos, su longitud así como la intensidad de tráfico existente en un periodo determinado.

Veamos gráficamente el principio de funcionamiento donde A y *B* señalan la posición de las motas y *d* la distancia entre las mismas.



En el instante $\mathbf{t_1}$, el móvil "eclipsa" la mota A.

En el instante $\mathbf{t_2}$, el móvil bloquea tanto la mota B como la A. Para ello exigiremos que *d* sea menor que la longitud mínima del objeto.

En el instante $\mathbf{t_3}$, la mota A ya no se encuentra oculta.

En el instante $\mathbf{t_4}$, se desbloquea finalmente la mota B.

Figura 1: Esquema de funcionamiento

Cada instante de tiempo se determinará considerando las variaciones en la luminosidad recibida.

Así, la velocidad del objeto vendrá dada por:

$$V = \frac{d}{t_2 - t_1} \quad (1)$$

[1] http://cv.uoc.edu/app/mediawiki14/

La longitud del objeto por:

$$L = V \cdot (t_3 - t_1) = \frac{d \cdot (t_3 - t_1)}{t_2 - t_1} \quad (2)$$

Y la intensidad de tráfico por:

$$I = \frac{n}{\Delta T} \quad (3)$$

Donde *n* es el número de objetos que han circulado en el periodo *ΔT*.

O bien, con una interpretación más propia de redes de telecomunicaciones, considerarla como una medida de la ocupación del carril mediante:

$$I = \frac{\sum_{i=1}^{n} t_{4i} - t_{1i}}{\Delta T} \quad (4)$$

Obsérvese que hemos representado interesadamente nuestro objeto mediante un vehículo para así destacar una de las aplicaciones prácticas donde podríamos determinar, además, si se respetan las distancias de seguridad.

Del mismo modo, la información relativa a la intensidad de tráfico sería especialmente útil para mejorar la planificación de la circulación, favoreciendo la toma de decisiones automáticas en relación a redirigir el tráfico en caso de saturación o la apertura de carriles bidireccionales en uno u otro sentido. Igualmente, y de forma paralela, podríamos reportar la temperatura en cada punto a otros efectos como aviso de temperaturas extremas así como por baja luminosidad.

De otra parte, excediendo las posibilidades actuales de nuestra mota, planteamos otra posible aplicación mediante el uso de etiquetas RFID. Así, si se generalizara la fabricación de vehículos incluyendo una etiqueta RFID con su número de bastidor a modo de “MAC” y añadiendo un lector RFID en la mota, ésta podría reportar la situación de cada vehículo a una central donde:

- Se detectarían vehículos en circulación sin seguro obligatorio, con ITV caducada, o robados.
- Situando las motas en distintos puntos de la ciudad, podríamos calcular si la velocidad media entre dichos puntos ha excedido de la máxima permitida.
- Se podría controlar la ruta que realizó/a un vehículo concreto.

No obstante, para evitar que al eliminar la etiqueta RFID de un vehículo éste “desaparezca” de nuestro alcance, las motas actuales detectarían su paso reportando dicho hecho y activando otras medidas complementarias como capturas de imágenes o similar.

Volviendo nuevamente a nuestro TFC, éste se limitará a la detección de objetos en circulación y cálculo de las magnitudes asociadas para lo que una de las motas estará conectada, a través del puerto USB, a un equipo informático que recibirá los instantes de tiempos recogidos y realizará el tratamiento de éstos acorde a lo indicado. Conforme a las herramientas de partida, dicha aplicación se realizará en Java, no descartando otras posibilidades.

En otra línea, destacar otra posible aplicación práctica en cintas transportadoras de entornos industriales.

1.1. Descripción detallada de la aplicación

Existe un silogismo ampliamente difundido según el cual:

> *Lo que no se puede **medir**, no se puede **controlar** y,*
> *lo que no se puede **controlar**, no se puede **mejorar**.*

En términos de ingeniería, concluiríamos, "... no se puede optimizar". Así, la adquisición de datos se erige como un elemento imprescindible para la optimización de procesos en general.

No obstante, la recogida de información puede conllevar, en ocasiones, un elevado coste económico, en particular si el origen de la misma se encuentra geográficamente disperso.

En este marco, una red de sensores inalámbricos de bajo coste de instalación y mantenimiento se antoja especialmente útil para cualquier aplicación de características similares.

Concretamente, nuestra aplicación se desarrolla en un contexto en el que existen elementos móviles cuya velocidad, longitud e intensidad de tráfico se quiere medir.

Sin pérdida de generalidad, nos centraremos en el tráfico de vehículos por carreteras, donde además reportaremos la temperatura y luminosidad a efectos informativos para el sistema y conductores así como la tensión de baterías para su mantenimiento.

1.1.1. Punto de partida y aportación del TFC

Tras analizar la wiki de la asignatura y la amplia documentación existente entre la que encontramos diversos TFCs de otros semestres, observamos funcionalidades básicas comunes que hacen las veces de punto de partida y soporte esencial del sistema. Así, tenemos:

- Obtención de medidas de luminosidad, temperatura y efecto Hall.
- Comunicación serie PC – mota.
- Comunicación mota – mota.

Se trata de los módulos fundamentales sobre los que se implementan aplicaciones de mayor complejidad, siendo habitual la presentación de dichos valores mediante una interfaz software desarrollada al efecto.

La aportación del presente TFC consiste, por tanto, en la singularidad de la aplicación a realizar que permite, mediante una configuración concreta del sistema, obtener velocidades, longitudes e intensidades de tráfico a partir de medidas de luminosidad, demostrando que la potencialidad de las motas utilizadas no se limita a la presentación de los valores tomados por sus sensores.

Figura 2: Funcionalidad del SIT-WSN

1.1.2. Aplicación real

En primer lugar representemos el esquema de comunicaciones que justifica el posterior desarrollo de aplicaciones.

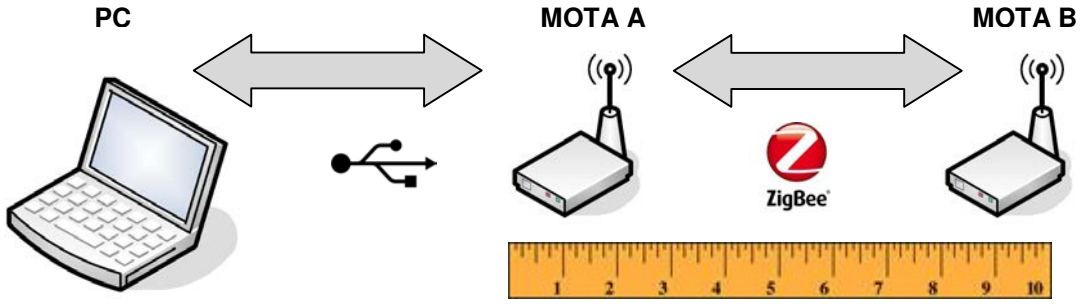


Figura 3: Esquema de comunicación del sistema

En cada dispositivo correrá una aplicación cuyas funciones representamos de forma sintetizada en el siguiente diagrama de bloques (léase de derecha a izquierda):

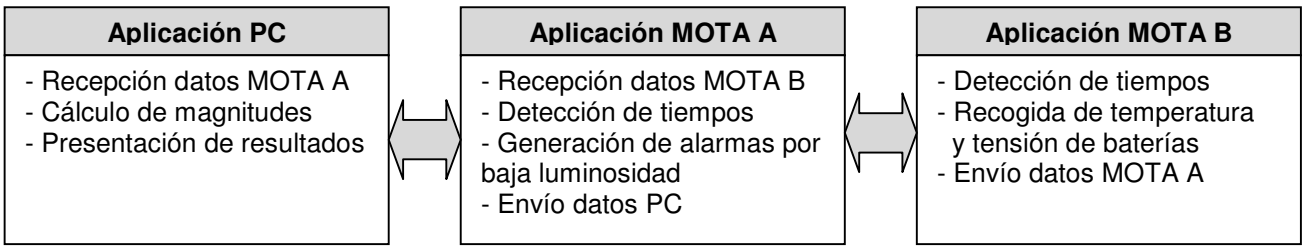


Figura 4: Diagrama de bloques del sistema

Donde los datos circularán conforme al flujo Aplicación PC ← Aplicación Mota A ← Aplicación Mota B, y las posibles órdenes de control, de ser necesarias, en sentido contrario.

A continuación ampliamos y justificamos cada función:

Aplicación Mota B

- Detecta cambios bruscos en el sensor de luminosidad registrando si se encuentra "libre" u "oculto".
- Recoge, periódicamente, el valor de la temperatura y tensión de baterías.
- Envía dichos datos a la MOTA A.

Aplicación Mota A

- Recibe datos de la MOTA B asignándole la referencia temporal.
- Detecta cambios bruscos en el sensor de luminosidad registrando si se encuentra "libre" u "oculto" así como el instante de tiempo en que sucede.
- Recoge el valor de luminosidad comparándola con patrones para detectar la falta de visibilidad y generar la correspondiente alarma.
- Envía dichos datos al PC.

Aplicación PC

- Recibe datos de la MOTA A señalizados con su procedencia (MOTA A o B) y los almacena.
- Realiza el cálculo de las magnitudes previstas: velocidad, longitud e intensidad de tráfico a la que es posible añadir sentido del tráfico mediante la comparación de tiempos.
- Informa y alerta, en función de los umbrales establecidos, sobre la temperatura, tensión de baterías y luminosidad.
- Presenta los resultados mediante una interfaz al efecto.

Obsérvese que en la distribución de tareas hemos procurado minimizar la carga de las aplicaciones que se ejecutan en las motas en detrimento de la aplicación del PC, limitando así el consumo de éstas.

Igualmente, para verificar la funcionalidad descrita hemos preparado el siguiente entorno de simulación donde se puede identificar la correspondencia de cada elemento con los anteriores esquemas.

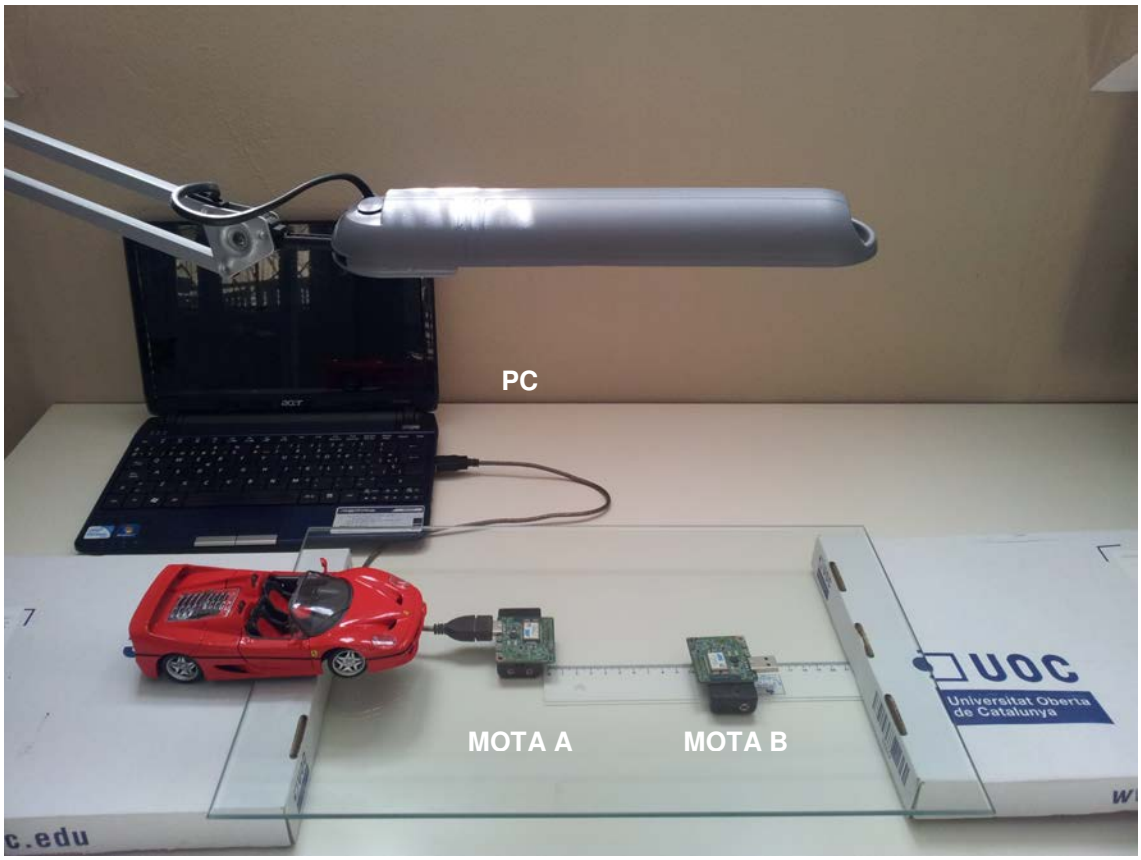


Figura 5: Entorno de simulación

En él hemos situado los sensores de luminosidad a una distancia de 10 cm, que es inferior a la longitud del vehículo de prueba, de 17,5 cm, conforme señalamos en apartados anteriores.

En el extremo superior disponemos de un flexo que incrementará la luminosidad. No obstante, realizaremos pruebas con luz natural en distintas franjas horarias para confirmar la validez de la aplicación en entornos no controlados.

A continuación mostramos el detalle de la ubicación de las motas.

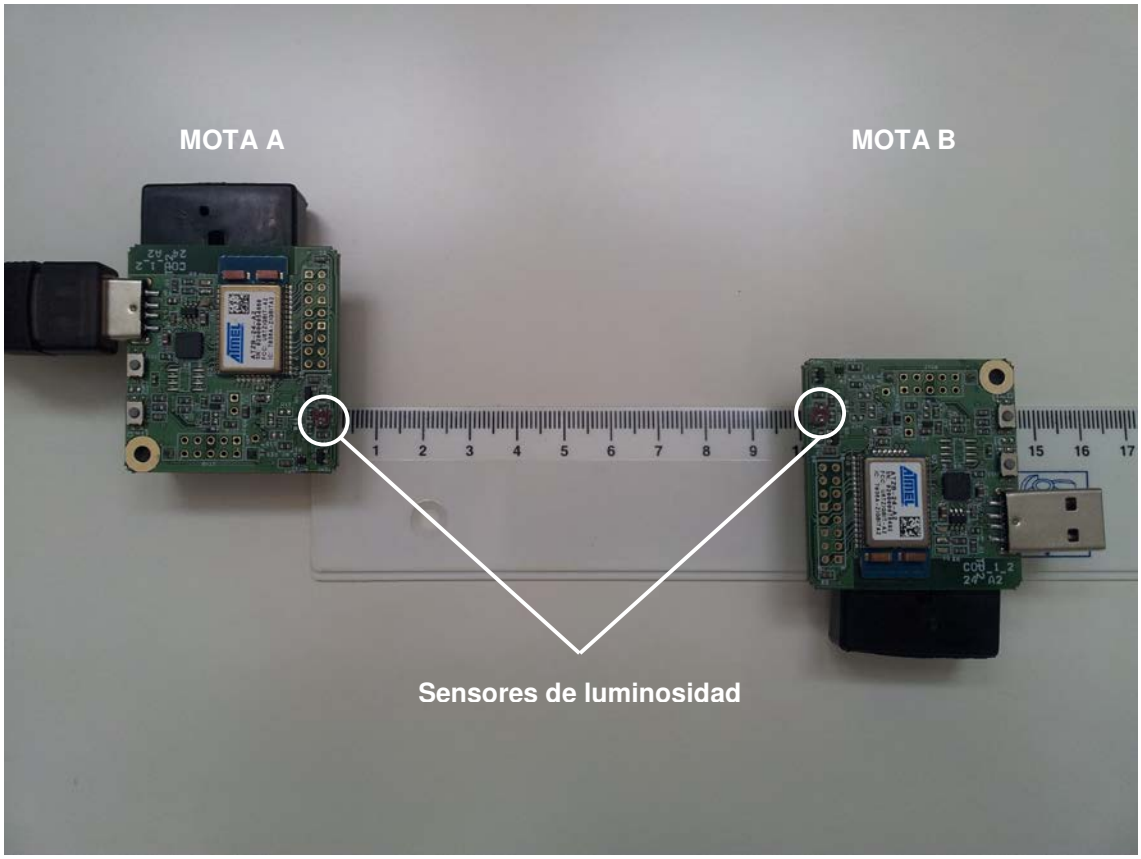


Figura 6: Detalle de ubicación de motas

1.2. Casos de uso de la aplicación

SIT-WSN es un Sistema de Información cuyos datos se reportarían a otras aplicaciones de la Dirección General de Tráfico (DGT) a los efectos oportunos. Así:

- En caso de detectar excesos de velocidad o longitudes propias de camiones en vías donde se excluyen estos vehículos, la información sería tratada por el sistema gestor de propuestas de sanción.

- De otra parte, la información relativa a la intensidad de tráfico se trasladaría al sistema de control para proceder a la apertura de carriles bidireccionales y, al igual que las alertas por temperaturas extremas y / o baja visibilidad, a informar a los conductores mediante paneles electrónicos proponiendo rutas alternativas y medidas de prevención.

- Del mismo modo, las alertas provocadas por batería baja, datos incoherentes o la ausencia prolongada de los mismos, se derivarían a operadores de mantenimiento para la oportuna revisión del sistema.

Según lo indicado, y con el objeto de evitar un crecimiento de potenciales actores y ofrecer un diagrama de casos de uso intuitivo acorde a las funcionalidades básicas previstas, consideramos inicialmente tres actores: DGT, operario de mantenimiento y conductor.

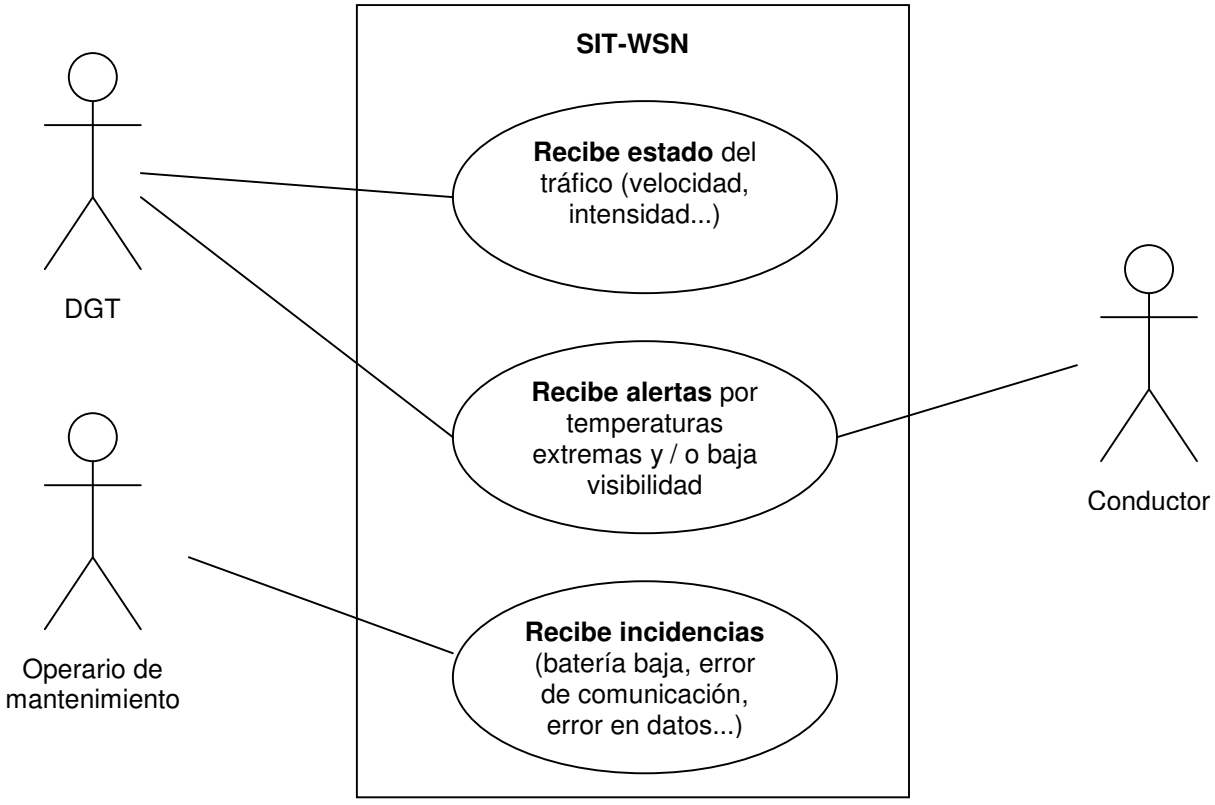


Figura 7: Casos de uso inicial de SIT-WSN

No obstante, con independencia de los posibles usos de la información proporcionada (gestión de sanciones, información a conductores, actuaciones de mantenimiento), la información generada por el tráfico de vehículos únicamente fluye hacia la DGT, por lo que si ampliamos la descripción de la funcionalidad propia de SIT-WSN realizada en el apartado 1.1, tenemos:

- Cada vez que un Conductor circule sobre las motas, el sistema almacenará el registro temporal de este hecho.

- Periódicamente el sistema actualizará la información sobre la temperatura y tensión de baterías.

- Cuando la DGT solicite, bajo demanda, un informe de los datos recogidos (incluidos errores), el sistema lo ofrecerá.

Nótese por tanto que los únicos actores reales de nuestro sistema son:

- La DGT que utiliza directamente el sistema recibiendo la información que éste genera.

- El Conductor cuya circulación provee de datos al sistema.

De hecho, la recogida de datos sobre la temperatura y tensión de baterías es un proceso automático del sistema donde no participa ningún actor.

Así, el diagrama de casos de uso final viene dado por:

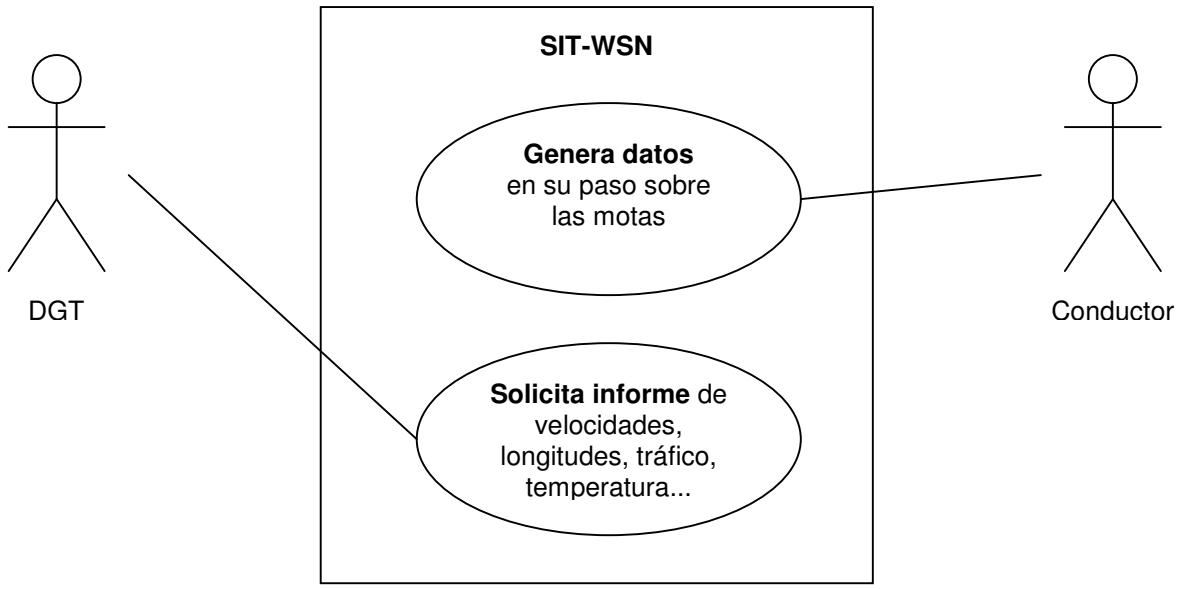


Figura 8: Casos de uso final de SIT-WSN

En las siguientes tablas se especifican ambos casos de uso donde, conforme señalan la pre / post condición, son independientes ya que el sistema genera los datos con independencia de que se soliciten o no informes al respecto.

Del mismo modo, la DGT puede demandar un informe que contendrá los datos almacenados hasta el momento. Si no existen datos, el informe estará en blanco.

<table>
<tr><td>Nombre</td><td>01 - Genera datos</td></tr>
<tr><td>Descripción</td><td>El conductor genera datos en su paso sobre las motas.</td></tr>
<tr><td>Actores</td><td>Conductor.</td></tr>
<tr><td>Precondición</td><td>Ninguna.</td></tr>
<tr><td>Secuencia principal</td><td>
<table>
<tr><td>01</td><td>El Conductor cubre la primera* mota.</td></tr>
<tr><td>02</td><td>El sistema registra este hecho así como el instante en que sucede</td></tr>
<tr><td>03</td><td>El Conductor cubre la segunda* mota.</td></tr>
<tr><td>04</td><td>El sistema registra este hecho así como el instante en que sucede</td></tr>
<tr><td>05</td><td>El Conductor deja al descubierto la primera* mota.</td></tr>
<tr><td>06</td><td>El sistema registra este hecho así como el instante en que sucede</td></tr>
<tr><td>07</td><td>El Conductor deja al descubierto la segunda* mota.</td></tr>
<tr><td>08</td><td>El sistema registra este hecho así como el instante en que sucede</td></tr>
</table>
* Entiéndase primera y segunda mota conforme al sentido de circulación.
</td></tr>
<tr><td>Errores alternativas</td><td>
<table>
<tr><td>03 ↔ 05</td><td>Aunque por construcción la distancia entre motas es inferior a la longitud del vehículo, el sistema registrará, de forma transparente, situaciones en las que se intercambien temporalmente los pasos 03 y 05.</td></tr>
<tr><td>No
(03 y 07)</td><td>Si como consecuencia de un cambio de carril justo en el momento en que atraviesa las motas, no se cubre y descubre la segunda de ellas:
• Si el siguiente vehículo circula en el mismo sentido, el sistema actualizará los registros 02 y 06 conforme a la nueva transición.
• Si el siguiente vehículo circula en sentido contrario, al cubrir en primera instancia la segunda mota (desde el punto de vista del caso anterior), el sistema no puede diferenciar este hecho de una circulación “normal” del vehículo anterior. Si bien, la disparidad de velocidades y longitudes denotará este hecho.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>Ninguna.</td></tr>
<tr><td>Notas</td><td>No.</td></tr>
</table>

Figura 9: Especificación del caso de uso “Genera datos”

<table>
<tr><td>Nombre</td><td>02 - Solicita informe</td></tr>
<tr><td>Descripción</td><td>La DGT solicita informe sobre velocidades, longitudes, tráfico, temperatura, tensión de baterías y luminosidad.</td></tr>
<tr><td>Actores</td><td>DGT.</td></tr>
<tr><td>Precondición</td><td>Ninguna.</td></tr>
<tr><td>Secuencia principal</td><td>
<table>
<tr><td>01</td><td>La DGT demanda el informe.</td></tr>
<tr><td>02</td><td>El sistema ofrece el informe registrados hasta el momento.</td></tr>
</table>
</td></tr>
<tr><td>Errores alternativas</td><td>No.</td></tr>
<tr><td>Postcondición</td><td>Ninguna.</td></tr>
<tr><td>Notas</td><td>No.</td></tr>
</table>

Figura 10: Especificación del caso de uso “Solicita informe”

1.3. Tecnología a usar

Haciendo referencia nuevamente a la wiki de la asignatura y a la amplia bibliografía existente, en este apartado relacionaremos sin mayor desarrollo las herramientas software y hardware que vamos a utilizar.

El detalle de los conocimientos concretos necesarios para cada funcionalidad se describirá al efecto. De esta forma, siguiendo las directrices del documento "Trabajo final de carrera", evitamos el incremento innecesario de información técnica que no aporta un mayor valor añadido.

1.3.1. Software

	Mota	PC
Sistema operativo	TinyOS programado en NesC	Linux Ubuntu 11.04
Entorno de programación	Eclipse	Eclipse
Librerías	TinyOS para NesC	TinyOS para Java
Lenguaje de programación	NesC	Java
Otros	Aplicación meshprog para transferir los desarrollos a la mota	
	Microsoft Excel 2007	

Figura 11: Software utilizado

1.3.2. Hardware

- 1 Equipo portátil Acer Aspire 1810TZ
- 2 motas COU_1_2 [2]

Figura 12: Detalle mota COU_1_2

[2] http://cv.uoc.edu/app/mediawiki14/wiki/Hardware_COU_1_2

1.4. Objetivos del proyecto

El presente TFC persigue diversos objetivos fundamentales:

- Servir de nexo de unión de los conocimientos adquiridos en las distintas disciplinas cursadas poniendo en práctica competencias necesarias en el desarrollo de la profesión como, especialmente, la gestión, documentación y presentación de proyectos.

- Concretamente dentro del área seleccionada, profundizar en los sistemas empotrados, ahondando en el sistema operativo TinyOS y el lenguaje de programación NesC.

- Desarrollar la capacidad creativa mediante la elección de una aplicación concreta orientada a su posible explotación comercial.

- Mejorar la capacidad de optimización en entornos con fuertes restricciones que obligan a disponer de un elevado conocimiento del hardware para ajustar los desarrollos al efecto.

- Demostrar el potencial de las WSN aplicado, en nuestro caso, a la adquisición de datos para el sistema de información de tráfico, que encaja en el concepto de las *Smartcities*.

- Maximizar las posibilidades prácticas de las motas COU_1_2 obteniendo magnitudes que, en principio, no derivan directamente de la luminosidad, temperatura o efecto Hall.

1.5. Planificación del proyecto

A continuación presentamos la relación de tareas inicial para la consecución del proyecto:

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Propuesta de proyecto**	**10 días?**	**lun 27/02/12**	**vie 09/03/12**	
2	Lectura de documentación	7 días	lun 27/02/12	mar 06/03/12	
3	Concreción de la idea	2 días	mié 07/03/12	jue 08/03/12	2
4	Redacción de la propuesta	1 día?	vie 09/03/12	vie 09/03/12	3
5	**Planificación del proyecto**	**8 días?**	**lun 12/03/12**	**mié 21/03/12**	**4**
6	Lectura de documentación	3 días?	lun 12/03/12	mié 14/03/12	
7	Análisis de funcionalidades	3 días?	jue 15/03/12	lun 19/03/12	6
8	Redacción de la planificación	2 días?	mar 20/03/12	mié 21/03/12	7
9	**Diseño del proyecto y primera entrega de código**	**24 días?**	**jue 22/03/12**	**mar 24/04/12**	**8**
10	Lectura de documentación	5 días?	jue 22/03/12	mié 28/03/12	
11	Instalación del entorno de trabajo	6 días?	jue 22/03/12	jue 29/03/12	
12	Diseño de aplicaciones	16 días?	vie 30/03/12	vie 20/04/12	11
13	Implementación aplicación PC y mota A	16 días?	vie 30/03/12	vie 20/04/12	11
14	Test aplicación PC y mota A	16 días?	vie 30/03/12	vie 20/04/12	11
15	Redacción del diseño y primera entrega de código	2 días?	lun 23/04/12	mar 24/04/12	14
16	**Segunda entrega del código fuente**	**20 días?**	**mié 25/04/12**	**mar 22/05/12**	**15**
17	Lectura de documentación	5 días?	mié 25/04/12	mar 01/05/12	
18	Diseño de aplicaciones	9 días?	mié 02/05/12	lun 14/05/12	17
19	Implementación aplicación mota B	9 días?	mié 02/05/12	lun 14/05/12	17
20	Test aplicación mota B	9 días?	mié 02/05/12	lun 14/05/12	17
21	Test aplicación completa	4 días?	mar 15/05/12	vie 18/05/12	20
22	Redacción de segunda entrega de código	2 días?	lun 21/05/12	mar 22/05/12	21
23	**Entrega de la memoria final del proyecto**	**12 días?**	**mié 23/05/12**	**jue 07/06/12**	**22**
24	Redacción de la memoria final del proyecto	12 días?	mié 23/05/12	jue 07/06/12	
25	**Presentación final del proyecto**	**5 días?**	**vie 08/06/12**	**jue 14/06/12**	**24**
26	Realización de la presentación final del proyecto	5 días?	vie 08/06/12	jue 14/06/12	

Figura 13: Relación de tareas

En ella destacamos que, aunque hemos utilizado un calendario laboral con fines de semanas de descanso, considerando el carácter docente del presente TFC, podemos hacer uso de los mismos, lo que nos aporta un “margen extra” en la planificación. En este mismo sentido, no hemos señalado los días festivos.

De otra parte, presentamos de forma simultánea las tareas de diseño, implementación y test, aludiendo a su carácter iterativo dentro del proceso, habiendo reservado un periodo adicional para el test de la aplicación completa.

En la siguiente figura presentamos el diagrama de Gantt dado por las presentes tareas.

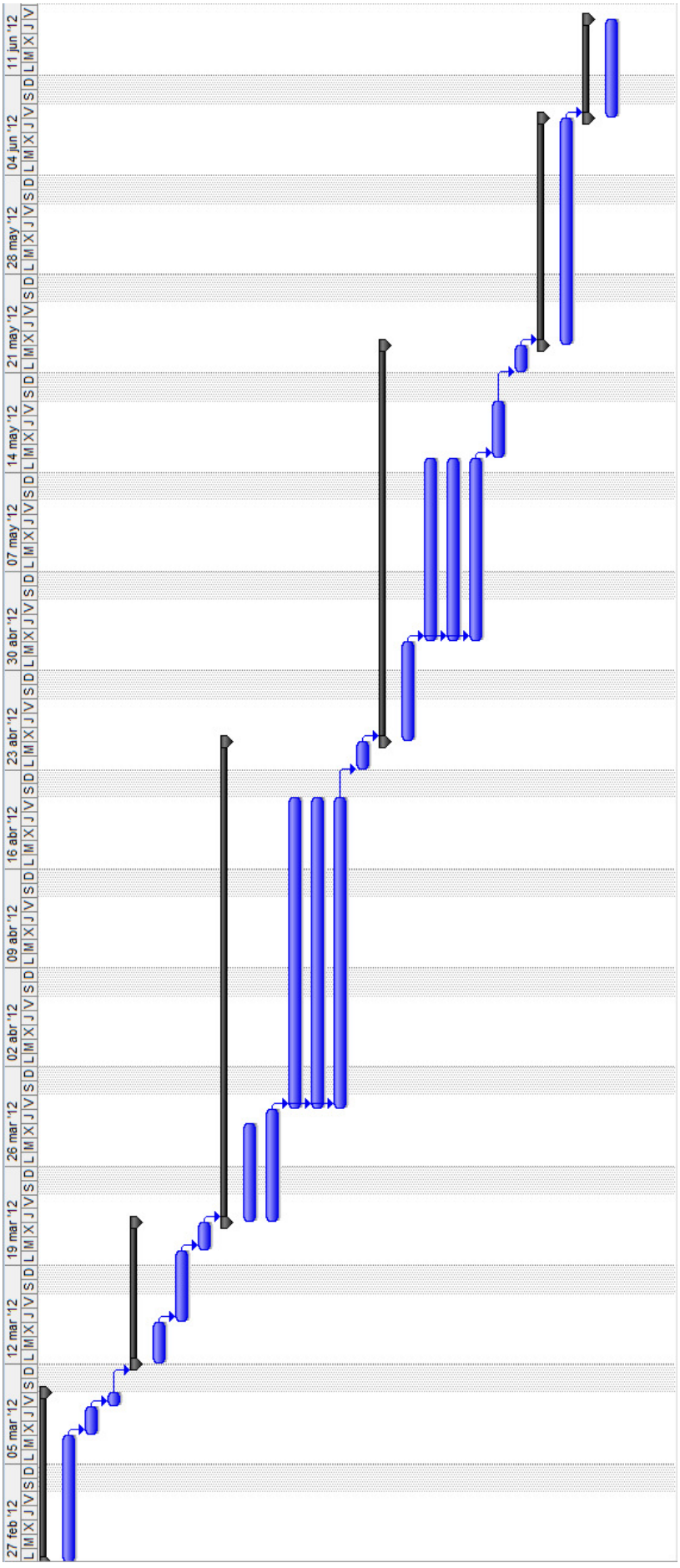


Figura 14: Diagrama de Gantt de la planificación

1.5.1. Seguimiento de la planificación del proyecto

La planificación del proyecto ha supuesto, en sí, un ítem fundamental para su consecución del mismo, permitiendo un mayor control sobre posibles desfases así como una más rápida actuación correctiva al respecto.

Aunque la mayor parte de tareas se han completado en plazo, debido a la particularidad de la funcionalidad principal propuesta (cálculo de velocidades, longitudes, etc.) que requiere de la medida conjunta de ambas motas, la tarea 9, relativa al diseño del proyecto y primera entrega del código presentó un retraso de 8 días debido a la necesidad de programar conjuntamente ambas motas.

Es por ello que, considerando los constantes cambios en las aplicaciones, se retrasó intencionadamente la realización de diagramas UML, componentes, etc. al resultar más costoso documentar parcialmente aspectos posteriormente modificados, que conseguir la funcionalidad y documentarla con una visión más holística.

2. Producto obtenido

A continuación se relacionan los apartados que describen el producto obtenido.

2.1. Diagramas de componentes

En los siguientes apartados abordaremos el diagrama UML de componentes de la aplicación que se ejecuta en el PC y el de componentes TinyOS y sus relaciones.

2.1.1. Diagrama UML de componentes de la aplicación PC

Conforme detallamos en el apartado dedicado al diseño, la aplicación que se ejecuta en el PC es una modificación de la clase *PrintfClient* a la que hemos denominado *PrintfSitWsn*. Es por ello que su diagrama UML se limita al representado en el siguiente gráfico.

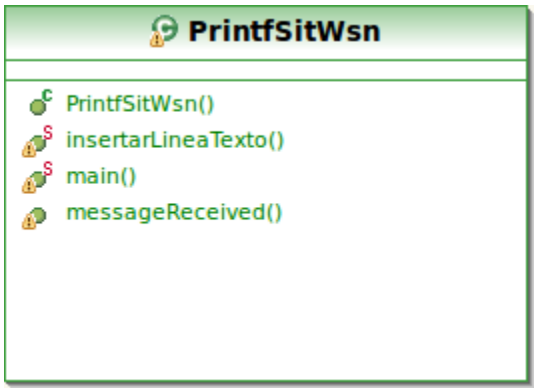


Figura 15: Diagrama UML de la aplicación PC

Aunque por su simplicidad no es necesario, hemos hecho uso del plugin de Eclipse eUML2 (www.soyatec.com/euml2/screenshots) que permite generar diagramas UML a partir de su código.

2.1.2. Diagrama de componentes TinyOS

A continuación mostramos el diagrama de componentes TinyOS de la aplicación que se ejecuta en la Mota A que, conforme a la nomenclatura utilizada, corresponde a la que se encuentra conectada al PC mediante USB.

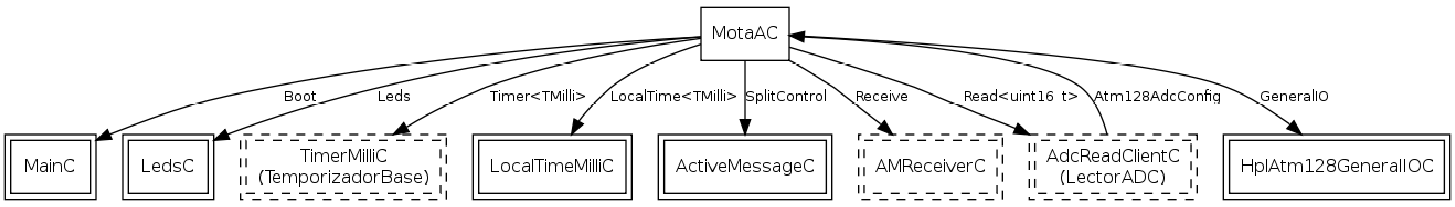


Figura 16: Diagrama de componentes TinyOS de la aplicación MotaAApp

Hemos obtenido dicho diagrama, así como la documentación asociada al mismo, mediante el comando:

```
make cou24 docs
```

Del mismo modo, generamos el diagrama de componentes TinyOS de la aplicación que se ejecuta en la Mota B.

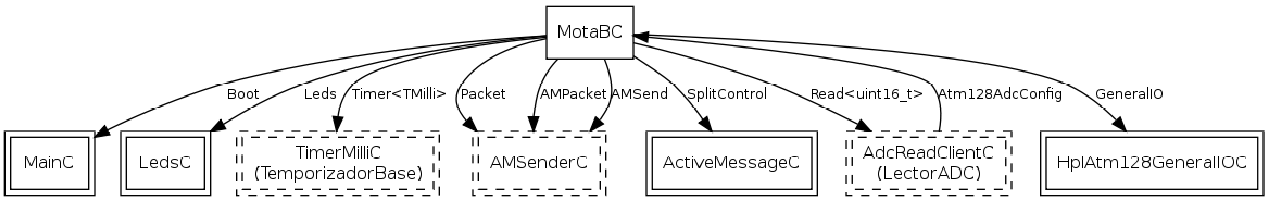


Figura 17: Diagrama de componentes TinyOS de la aplicación MotaBApp

2.2. Explicación del diseño

A diferencia de otros proyectos del ámbito domótico donde las magnitudes a medir varían lentamente, la funcionalidad propuesta confiere al **factor tiempo** un carácter determinante obligando a que el sistema realice un muestreo constante que condiciona el diseño del mismo.

Así, la velocidad máxima de los vehículos determina la frecuencia mínima de muestreo de la variable controlada para evitar que éstos puedan pasar "inadvertidos".

Además, cuanto mayor es dicha frecuencia, mayor es el consumo energético, cuyo ajuste es, igualmente, uno de los objetivos fundamentales en el diseño de cualquier sistema empotrado.

Nos encontramos, por tanto, ante la necesidad de una solución de compromiso que impondrá limitaciones en la funcionalidad perseguida implementada mediante la tecnología descrita.

Así, las motas se erigen como meras sondas remotas que únicamente colectan y reportan datos a la aplicación que se ejecuta en el PC que, a su vez, los almacena para su posterior tratamiento e interpretación por los clientes, descargando así al sistema de actividades que podrían suponer una merma en el rendimiento y, con éste, de la funcionalidad.

Todo ello confirma el sentido del flujo de datos descrito en el apartado 1.1.2: Aplicación PC ← Aplicación Mota A ← Aplicación Mota B, eliminando cualquier posible orden de control en sentido contrario cuya atención limite, aún más, la frecuencia de muestreo.

De esta forma, el diseño se atiene al previsto en el planteamiento inicial.

De otra parte, el valor de luminosidad se encuentra en el rango [0, 1023] que, aún con luz constante en un entorno controlado, presenta una variación de unas 100 unidades entorno al punto de equilibrio.

Las pruebas realizadas con objetos móviles conforme a la figura 5, nos muestran un decremento mínimo del valor de luminosidad de 325 unidades cuando éstos se encuentran sobre la mota, retornando aproximadamente al valor inicial tras dejar la mota descubierta.

Así, hemos definido un **umbral** que, a modo de histéresis, evitará falsos positivos. Cuando dicho umbral es superado, entendemos que el objeto ha entrado o abandonado la zona de control, procediéndose a reportar la información al efecto.

La información llega a la aplicación PC en una cadena con formato CSV cuyo significado exponemos sobre la siguiente captura real:

Id mota	Tipo	Número de periodos	Valor magnitud
2	1	10435	0
2	1	10484	1
1	1	10518	0
1	1	10541	1
1	1	13737	0
1	1	13769	1
2	1	13776	0
2	1	13796	1
2	1	16885	1
1	1	19788	0

Figura 18: Ejemplo de datos reportados a la aplicación PC

De esta forma:

- El campo “Id mota” puede ser 1 (mota A) ò 2 (mota B).
- El valor del campo “Número de periodos” / 1024 [s] establece la referencia temporal necesaria para los cálculos.

Los campos “Tipo” y “Valor” presentan los siguientes posibles valores.

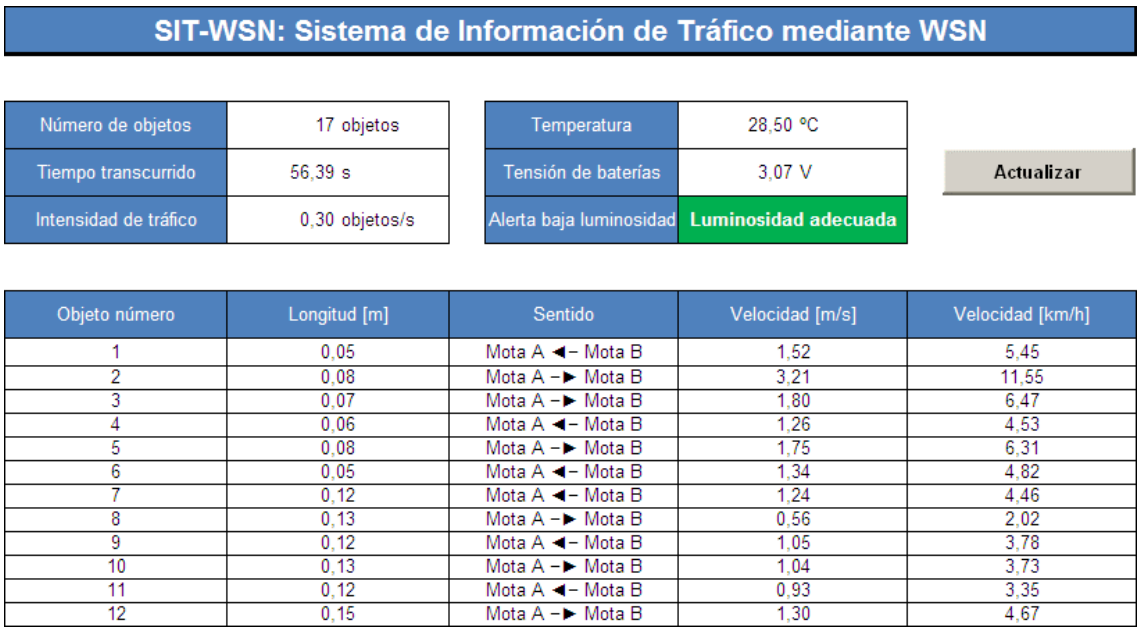
Tipo	Valor
1: Variaciones de luminosidad	0: El objeto se encuentra sobre la mota
	1: El objeto ha abandonado la mota
2: Tensión de batería	[0, 1023] que determina la tensión de batería
3: Temperatura	[0, 1023] que determina la temperatura
4: Indicador de baja luminosidad	0: No aporta mayor significado
99: Error en la lectura de sensores / comunicaciones	0: No aporta mayor significado

Figura 19: Significado de los campos “Tipo” y “Valor”

Los objetivos del proyecto establecidos en el apartado 1.4 se centran en el ámbito de la programación en nesC sobre TinyOS de las motas COU_1_2, por lo que la aplicación PC se limita a almacenar los datos recibidos en un archivo con formato CSV (Comma Separated Values) que es un estándar de facto para el intercambio de datos cuya adquisición y tratamiento puede llevarse a cabo tanto desde aplicaciones ofimáticas como por cualquier otra herramienta desarrollada al efecto.

Así, atendiendo igualmente al objetivo de integración de sistemas, dejando en evidencia la facilidad para trabajar con este tipo de formatos y considerando la práctica empresarial habitual de utilizar hojas de cálculo para la manipulación de datos en formato tabla, haremos uso de la herramienta Microsoft Excel para, mediante su tratamiento automatizado, mostrar el análisis de los datos adquiridos mediante las motas que pueden encontrarse a kilómetros de distancia.

En este sentido, se facilita el archivo “SitWsn.xlsm” que requiere de Microsoft Excel 2007 donde se ha programado mediante VBA la interfaz para la adquisición y tratamiento de los datos contenidos en “reporte.csv”. Para su uso es necesario, por tanto, habilitar macros.



Objeto número	Longitud [m]	Sentido	Velocidad [m/s]	Velocidad [km/h]
1	0,05	Mota A ◄– Mota B	1,52	5,45
2	0,08	Mota A –► Mota B	3,21	11,55
3	0,07	Mota A –► Mota B	1,80	6,47
4	0,06	Mota A ◄– Mota B	1,26	4,53
5	0,08	Mota A –► Mota B	1,75	6,31
6	0,05	Mota A ◄– Mota B	1,34	4,82
7	0,12	Mota A ◄– Mota B	1,24	4,46
8	0,13	Mota A –► Mota B	0,56	2,02
9	0,12	Mota A ◄– Mota B	1,05	3,78
10	0,13	Mota A –► Mota B	1,04	3,73
11	0,12	Mota A ◄– Mota B	0,93	3,35
12	0,15	Mota A –► Mota B	1,30	4,67

Figura 20: Interfaz gráfica de SitWsn.xlsm

En la figura 20 ofrecemos la interfaz gráfica donde se muestra la información reportada al usuario.

Finalmente, hemos de aclarar que el uso de los leds en ambas motas corresponde a la representación de los tres bits menos significativos de un contador que se incrementa con cada transición (entrada o salida) de vehículos.

Se trata, lógicamente, de una utilidad demostrativa que, con objeto de minimizar el consumo, no tendría cabida en la aplicación definitiva.

2.3. Código generado

El presente documento se entrega comprimido junto a dos carpetas “PC” y “MotaA” que contienen respectivamente la aplicación Java que se ejecuta en el PC y la aplicación escrita en nesC que se ejecuta en la Mota A.

2.3.1. Aplicación PC

En modo depuración, podemos compilar y ejecutar el código de la aplicación PC haciendo uso del IDE Eclipse cuyo GUI representamos en la siguiente figura.

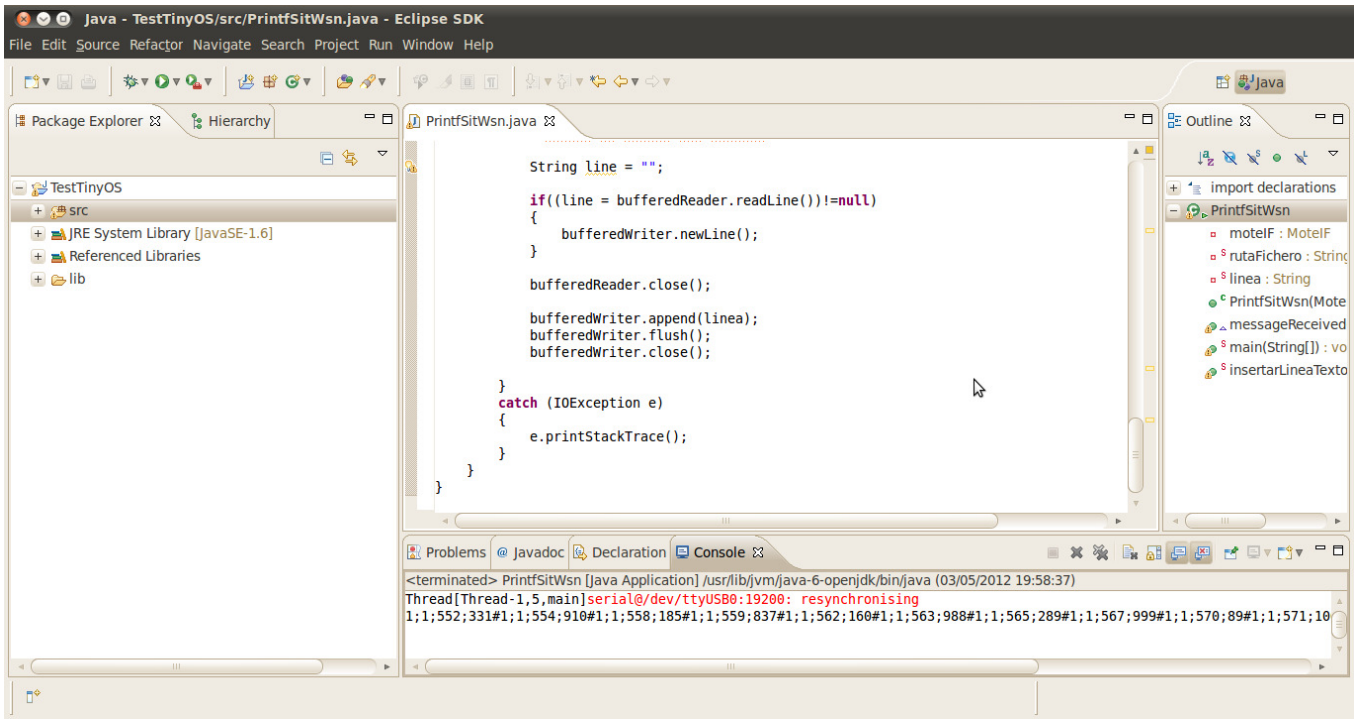


Figura 21: GUI de IDE de Eclipse

No obstante, definiendo correctamente la variable de entorno CLASSPATH incluyendo “tinyos.jar”, podemos compilar el código, en modo comando, mediante “javac PrintfSitWsn.java” y ejecutarlo con “java PrintfSitWsn”.

2.3.2. Aplicación Mota A y Mota B

A continuación, incluimos el detalle de las instrucciones (destacadas en color azul) y feedback recibido para compilar y cargar la aplicación en la Mota A cuyo descriptor, de acuerdo al comando “motelist”, es “/dev/ttyUSB0”. (/dev/ttyUSB1 para la Mota B).

Igualmente, señalamos que, de forma paralela mediante dichas instrucciones, estamos asignando a dicha mota el identificador 1 (2 para la Mota B).

```
Setting up for TinyOS 2.1.1

asuarez@AP:~/Escritorio/MotaA$ make cou24 install,1
mkdir -p build/cou24
    compiling MotaAAppC to a cou24 binary
ncc -o build/cou24/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow
-Wnesc-all -target=cou24 -fnesc-cfile=build/cou24/app.c -board= -
DDEFINED_TOS_AM_GROUP=0x22 --param max-inline-insns-single=100000 -
I/opt/tinyos-2.x/tos/lib/printf -DIDENT_APPNAME=\"MotaAAppC\" -
DIDENT_USERNAME=\"asuarez\" -DIDENT_HOSTNAME=\"AP\" -
DIDENT_USERHASH=0x71b77448L -DIDENT_TIMESTAMP=0x4fa2c6abL -
DIDENT_UIDHASH=0x440e8d5dL -fnesc-dump=wiring -fnesc-
dump='interfaces(!abstract())' -fnesc-
dump='referenced(interfacedefs, components)' -fnesc-
dumpfile=build/cou24/wiring-check.xml MotaAAppC.nc -lm
    compiled MotaAAppC to build/cou24/main.exe
          18518 bytes in ROM
           1002 bytes in RAM
avr-objcopy --output-target=srec build/cou24/main.exe
build/cou24/main.srec
avr-objcopy --output-target=ihex build/cou24/main.exe
build/cou24/main.ihex
    writing TOS image
tos-set-symbols build/cou24/main.srec build/cou24/main.srec.out-1
TOS_NODE_ID=1 ActiveMessageAddressC__addr=1
    installing cou24 binary using dapa
avrdude -cdapa               -U hfuse:w:0x99:m -pm1281 -U
efuse:w:0xff:m -C/etc/avrdude/avrdude.conf  -U
flash:w:build/cou24/main.srec.out-1:a
avrdude: can't open device "/dev/parport0": No such file or
directory
avrdude: failed to open parallel port "/dev/parport0"

make: *** [program] Error 1

asuarez@AP:~/Escritorio/MotaA$ meshprog -t/dev/ttyUSB0 -
f./build/cou24/main.srec.out-1
using tty /dev/ttyUSB0
reading from file ./build/cou24/main.srec.out-1
sending ./build/cou24/main.srec.out-1 -> /dev/ttyUSB0

Opened file ./build/cou24/main.srec.out-1
Opened device /dev/ttyUSB0
..............,[i••&]
Starting transmission.
finished!
```

Figura 22: Compilación y carga de la aplicación de la mota A

Con el objetivo de que la carga se realice con éxito, es necesario realizar un reset a la mota para que, mediante el boot loader, ésta admita la nueva programación. Posteriormente, iniciará su ejecución.

2.3.3. Casos de uso implementados

El código entregado atiende los casos de uso previstos. Así, permite:

- Detectar y reportar a la aplicación PC variaciones superiores a un determinado umbral del valor de luminosidad muestreado cada TIEMPO_MUESTREO / 1024 segundos tanto en la mota A como B.

- Reportar periódicamente a la aplicación PC los valores de tensión de baterías y temperatura en el rango [0, 1023].

- Reportar alertas por baja luminosidad generadas cuando ésta desciende por debajo del UMBRAL_BAJA_LUMINOSIDAD en el TIEMPO_DETECCION_BAJA_LUMINOSIDAD.

- Almacenar en un documento .csv dicha información para su posterior procesamiento y presentación.

- Ofrecer al usuario una herramienta simple con la obtener y visualizar los resultados obtenidos.

Para un correcto cálculo de velocidades y distancias, SitWsn.xlsm dispone de una constante "distancia" cuyo valor es 0,1 m (10 cm) que es la distancia a la que se encuentran las motas.

2.4. Arquitectura TinyOS

La arquitectura TinyOS surge, conforme se simboliza en la figura 23, de la necesidad de adaptar la arquitectura de sistemas operativos tradicionales a los limitados recursos de los dispositivos donde se ejecuta TinyOS.

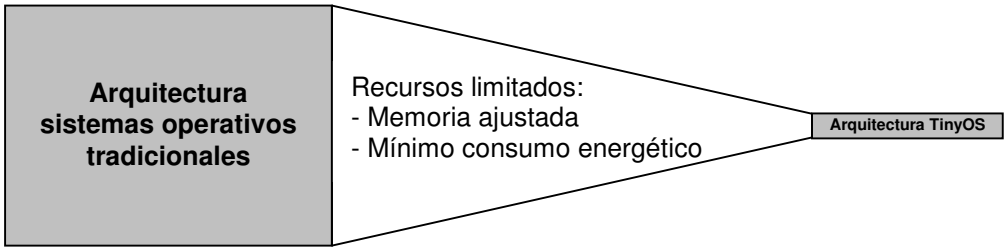


Figura 23: Origen de la arquitectura TinyOS

Se trata, por tanto, de una arquitectura aligerada de aquellos elementos prescindibles. Así:

- No existe un núcleo del sistema operativo que administre el hardware, en su lugar se realiza una manipulación directa del mismo.
- No se lleva a cabo la administración de procesos ya que, en cada momento, sólo habrá un proceso en ejecución.
- No se trabaja con memoria virtual, sino con un único espacio lineal de direcciones físicas.
- No existe asignación dinámica de memoria ya que dicha asignación se realiza en tiempo de compilación.
- No se generan señales vía software o excepciones, trabajándose en su lugar con llamadas a funciones.

Con todo ello conseguimos el objetivo de minimizar el tamaño de la memoria y la sobrecarga del sistema, lo que nos lleva, a su vez, al esquema habitual de la arquitectura TinyOS.

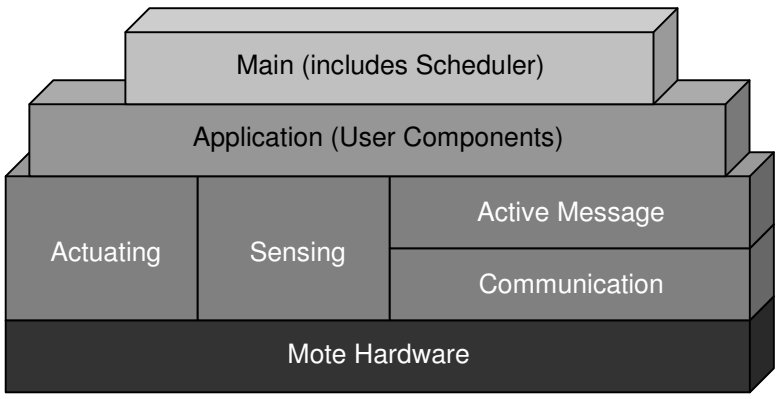


Figura 24: Arquitectura TinyOS simplificada

En él hemos utilizado las denominaciones inglesas para evitar traducciones inexactas y poco adecuadas.

Del mismo modo, se incluye explícitamente a modo de referencia dónde se ubicaría el hardware de la mota, cuya cercanía con las aplicaciones de usuario puede sugerir una fuerte dependencia que, sin embargo, TinyOS consigue evitar.

De hecho, TinyOS se caracteriza, además de por la ya referida limitación de recursos y del consumo energético, por facilitar un entorno cuasi-transparente del hardware sobre el que se ejecuta.

De esta forma, TinyOS permite dos niveles de planificación:

- Eventos
    - Actúan de forma similar a las interrupciones, teniendo prioridad sobre las tareas, lo que les permite cierta garantía en términos de tiempos de ejecución.
    - Se utilizan en procesos pequeños tales como transiciones de estados.
    - Pueden anticiparse a otros eventos.
    - Son independientes de la planificación FIFO aplicada a las tareas.
- Tareas
    - Menos prioritarias que los eventos por lo que su uso no está recomendado para procesos con fuertes requerimientos temporales.
    - Orientadas a una mayor carga de procesamiento.
    - No pueden anticiparse a otras tareas, lo que las dota de un carácter atómico respecto de las mimas.
    - Su gestión se realiza en una pila FIFO con un número limitado de tareas pendientes. Cuando ésta está vacía, el planificador limita la actividad de la CPU al reloj, minimizando así el consumo.

El uso adecuado de ambos niveles permite alcanzar un alto rendimiento en aplicaciones con concurrencia intensiva.

Otro de los elementos fundamentales que configuran TinyOS es su programación en NesC, un dialecto del C que agrupa el código en componentes que se comunican a través de interfaces. De ahí que se utilicen términos como arquitectura basada en componentes o modelo de programación orientado a componentes.

A su vez, estos componentes pueden ser:

- Módulos (nombreC.nc) que implementan interfaces con funciones: comandos y eventos.
    - Los comandos son especificados a alto nivel e implementados a bajo nivel. Realizan peticiones que no bloquean al componente de nivel inferior.
    - Los eventos, comparables a funciones *callback*, son especificados a bajo nivel e implementados a alto nivel.
- Configuraciones (nombreAppC.nc) que conectan interfaces entre sí (denominado *wiring* y representado mediante fechas que unen quienes utilizan la interfaz con quienes las proveen).

Lo indicado queda sintetizado en el siguiente gráfico.

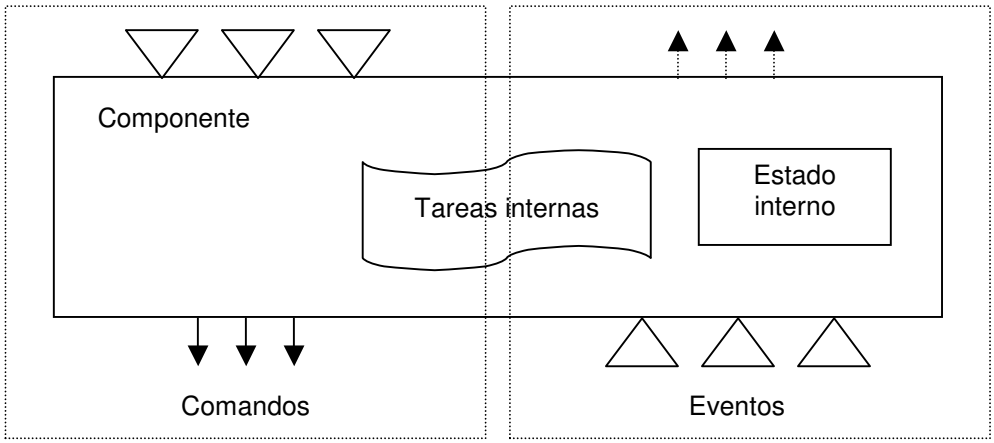


Figura 25: Modelo de componente TinyOS y su interacción

Véase igualmente las figuras 16 y 17 como ejemplo de *wiring* referido a nuestra aplicación.

Por último, con el objeto de tener una visión global de la arquitectura, incluimos el gráfico de componentes de una aplicación genérica.

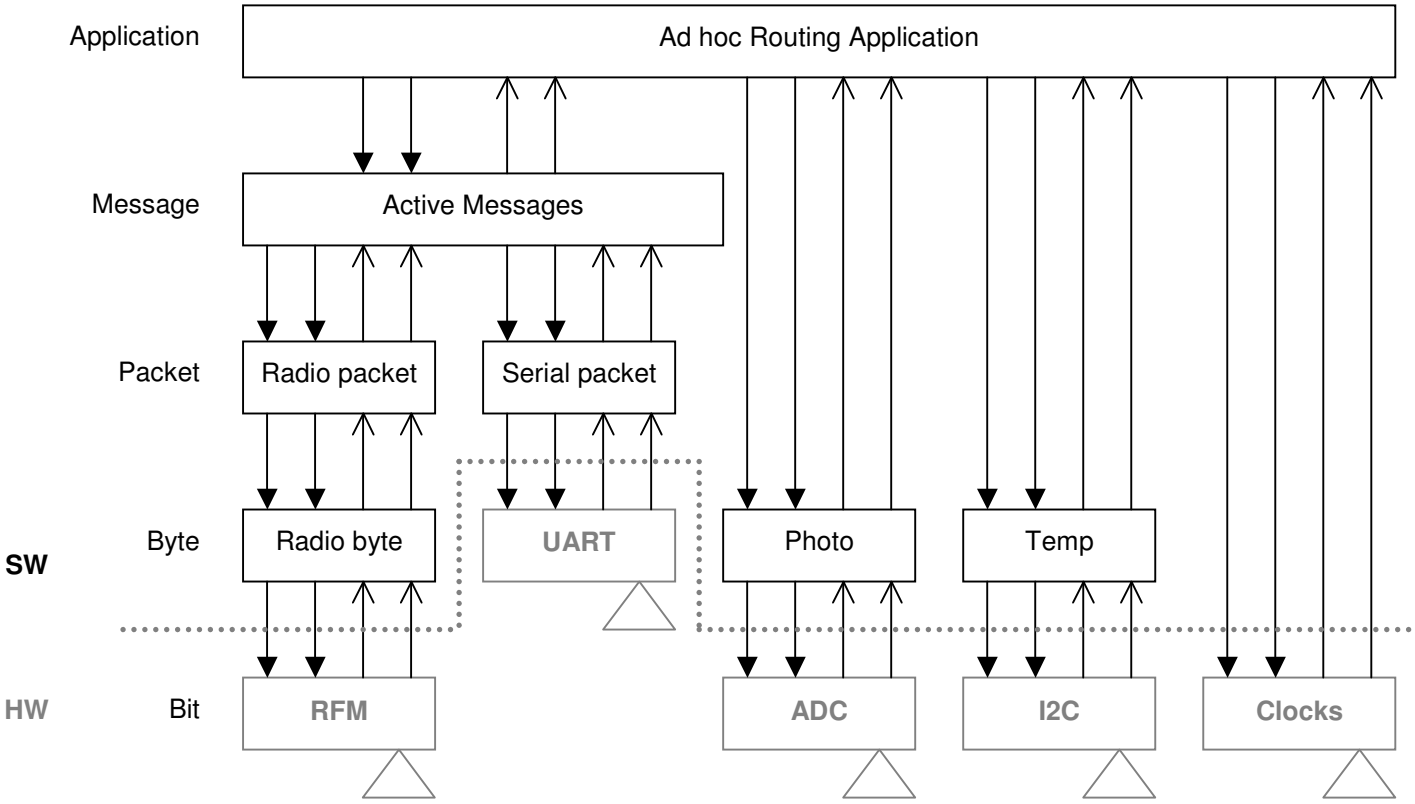


Figura 26: Gráfico de componentes de una aplicación genérica

2.4.1. Adaptación del diseño: Base de tiempos

En este punto queremos enfatizar cómo hemos adaptado nuestra aplicación a la arquitectura TinyOS que lógicamente ha condicionado el diseño de la misma.

Las funcionalidades propuestas exigen fuertes restricciones temporales así como el mayor sincronismo posible por lo que, conforme se ha detallado en el apartado anterior, han sido implementadas mediante eventos.

De otra parte, la comparación de valores temporales adquiridos desde múltiples sistemas requiere garantizar que éstos utilizan la misma base de tiempos – el mismo reloj – o, al menos, que existe una sincronización previa que simula fehacientemente la unicidad de referencias.

La resolución de esta necesidad, esencial para la aplicación propuesta, ha obligado a adoptar distintos enfoques que han ido evolucionando en función de las incidencias detectadas alrededor de dos cuestiones fundamentales que detallamos en los siguientes apartados.

2.4.1.1. Reloj del sistema

Nuestras aplicaciones disponen de un temporizador base (componente TimerMilliC) cuyo periodo permite muestrear los valores de los distintos sensores.

Así, en primer lugar, con el objetivo de minimizar el número de temporizadores necesarios para optimizar el rendimiento del sistema, creamos una variable contador que se incrementaba con cada disparo del mismo.

No obstante, las distintas simulaciones pusieron de manifiesto que la evolución de dicho contador no era lineal con el tiempo sino que dependía fuertemente de la carga del sistema proporcional al tráfico de vehículos.

De este modo, pasamos a implementar el reloj local mediante el componente LocalTimeMilliC cuyo comportamiento sí se ajustaba al previsto.

2.4.1.2. Sincronización de relojes

Para sincronizar los relojes locales de ambas motas, creamos inicialmente el siguiente protocolo:

Mota	Acción
A	Arranca
B	Arranca
B	Obtiene RelojLocal(B)
B	Envía RelojLocal(B) a Mota A
A	Recibe RelojLocal(B)
A	Calcula diferencia := RelojLocal(A) - RelojLocal(B)
...	
B	Envía ValorSensor, RelojLocal(B) a Mota A
A	Recibe ValorSensor, RelojLocal(B)
A	Asigna ValorSensor, RelojLocal(B) + diferencia

Figura 27: Algoritmo de sincronización previsto inicialmente

Así, la mota A recibía, a modo de timeStamp, el reloj de la mota B cuya diferencia con el suyo propio repercutía para el resto de medidas.

Sin embargo, volvemos a comprobar que dicha diferencia no permanece constante haciendo que determinados sucesos parezcan adelantarse al momento en que realmente ocurren. De hecho, conforme al comando get de la interface LocalTime, el temporizador puede llegar a detenerse cuando el procesador entra en modo de bajo consumo.

De otra parte, la opción de realizar una sincronización previa a cada dato enviado resulta subóptimo debido a las exigencias del sistema en términos energéticos y de tiempos de respuesta.

2.4.1.3. Solución implementada

Ante las dificultades para ajustar la funcionalidad prevista a las limitaciones del sistema, decidimos realizar el siguiente experimento:

- Creamos un único reloj local en la mota A que asignará referencias temporales a sus medidas y a las que recibe de la mota B.

- Situamos muy próximos los sensores de luminosidad de ambas motas y, sobre éstos, una lámpara que encendemos y apagamos alternativamente de forma que las medidas puedan entenderse simultáneas.

- El análisis estadístico de una muestra elevada de las diferencias temporales entre la mota A y B, nos permite aproximar la distribución de dicha variable aleatoria mediante una normal de media 18 y desviación típica 10 (donde un valor de 1024 corresponde a 1 segundo).

Los resultados de las pruebas realizadas mediante esta solución no sólo son coherentes conforme a lo esperado, sino que, además, acotan y minimizan el error a centésimas de segundo que, hasta el momento, se situaba en el orden de varias décimas de segundo, lo que resultaba incompatible con la funcionalidad deseada.

Obsérvese que, de forma añadida, eliminamos el reloj local de la mota B, minimizando los recursos necesarios de ésta que revertimos en la captación de los valores periódicos de la tensión de baterías y temperatura, requeridos para los otros casos de uso.

2.5. Arquitectura Cliente - Servidor

Aunque el apartado 1.1.2 refleja con suficiente claridad la arquitectura cliente – servidor del sistema, queremos aprovechar este apartado para hacer hincapié en el esfuerzo de integración de tecnologías realizado.

Así, un esquema más completo incorporaría otro PC en el que se ejecutaría SitWsn.xlsm que accedería al reporte.csv, compartido mediante la herramienta samba.

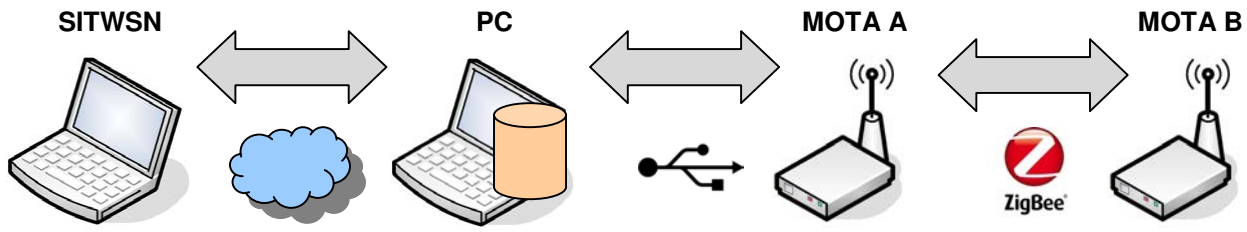


Figura 28: Arquitectura Cliente – Servidor

Con una visión global del conjunto, podemos imaginar una red de sensores WSN que reportarían las transiciones de los vehículos a una central de datos a la que accederían otros equipos o sistemas para su consulta y tratamiento oportuno.

3. Conclusiones

El presente trabajo de fin de carrera nos ha facilitado profundizar en los conceptos fundamentales de los sistemas empotrados.

En concreto, la aplicación propuesta ha permitido intuir, a modo de frontera de posibilidades de producción, el compromiso existente entre la funcionalidad deseada, en nuestro caso denotada por la frecuencia de muestreo del vial, y las limitaciones técnicas encabezadas por el consumo energético mínimo de este tipo de sistemas.

A su vez, conforme señalamos en la introducción, hemos perseguido, en todo momento, ser ambiciosos en los objetivos y en la orientación comercial de nuestra aplicación.

En este sentido, a continuación se relacionan posibles mejoras al proyecto que, con un criterio esencialmente comercial, en ocasiones exceden del campo técnico abordado.

Del mismo modo, ofrecemos un último apartado donde relacionamos aquellos aspectos que creemos necesarios para facilitar la implementación real de la aplicación.

3.1. Propuestas de mejoras

Con el objeto de optimizar aún más si cabe el rendimiento y posibilidades de nuestra aplicación proponemos las siguientes mejoras:

- Incluir en la mota un lector RFID en modo de funcionamiento pasivo donde, si se generalizara la fabricación de vehículos añadiendo una etiqueta RFID, alimentada mediante batería, con su número de bastidor a modo de “MAC” permitiría:
    - Detectar, mediante el acceso a la base de datos de la DGT, vehículos en circulación sin seguro obligatorio, con ITV caducada, o robados.
    - Calcular si la velocidad media de un vehículo que ha circulado por distintas ubicaciones de nuestra red de motas ha excedido de la máxima permitida.
    - Controlar y reproducir la ruta que realizó/a un vehículo concreto.
- Estudiar la viabilidad de implementar la funcionalidad descrita de etiquetas RFID mediante la tecnología ZigBee, ya presente en la mota.
- Incluir en la mota un circuito de bajo consumo que, en conjunción con el sensor de luminosidad, implemente el efecto umbral, realizado actualmente mediante código, generando una interrupción, con lo que se conseguiría una menor carga del sistema que revertiría en un consumo más ajustado y una optimización de los tiempos de ejecución que, a su vez, mejoraría la precisión aumentando su rango de uso.
- Modificar la aplicación PC para que almacene los registros en una base de datos creada al efecto y que, en conjunción con el resto sistemas de la DGT, permitiera la implementación de múltiples funcionalidades de valor añadido.

3.2. Implementación real de la aplicación

El presente apartado tiene como objetivo fundamental relacionar aquellos aspectos necesarios para permitir la implementación real de la aplicación que ha sido la piedra angular sobre la que hemos diseñado la misma.

- Nuestro sistema va a instalarse en las diferentes vías de la red de transporte por carretera. El entorno donde se ubicará toma especial importancia por cuanto:
    - Su adecuación debe ser económicamente viable.
    - Debe permitir la funcionalidad prevista, para lo que igualmente requiere ser translúcido y facilitar la propagación del calor sin que, por efecto invernadero, éste se intensifique.
    - Su protección IP debe ser tal que resista a la presión de vehículos así como presentar la suficiente cobertura frente lluvias que imposibilite filtraciones.
    - Debe garantizar su rápida instalación y mantenimiento, evitando, a su vez, accesos no autorizados de terceras personas que puedan robar o alterar su funcionamiento comprometiendo la validez y utilidad de las medidas.
- Un vez concretado, han de realizarse pruebas en dicho entorno real que pongan de manifiesto:
    - La adecuación de éste conforme a los puntos reseñados.
    - Dificultades y modificaciones necesarias en el diseño para adaptarse al mismo.
- Igualmente, es necesario realizar un estudio exhaustivo para que el sistema pueda ser aceptado en un hipotético juicio como prueba pericial, para lo que:

Debe establecerse la precisión mínima y, con ésta, el error máximo en términos de velocidad, documentándose convenientemente y superando las pruebas, al efecto, del INTA.

Anexo: Manual del usuario

Con el objetivo de realizar un manual sintético que no redunde en los conceptos ya tratados en apartados anteriores, supondremos que el usuario dispone de unos conocimientos básicos acerca de los mismos.

De esta forma, abordaremos cómo compilar y utilizar cada elemento software que compone nuestra aplicación. Éstos se entregan separados en cuatro carpetas: MotaA, MotaB, PC y SitWsn.

Debe tenerse en cuenta al efecto el software y hardware del que se dispone conforme a los apartados 1.3.1 y 1.3.2. Así, tenemos:

Aplicación Mota A:

- Compilación:
    - Desde el terminal de gnome, situados en la carpeta MotaA, ejecutamos la instrucción:

```
make cou24 install,1
```

    - Además de las indicaciones mostradas por pantalla acerca de la correcta consecución de la operación conforme a la figura 22, este comando genera una estructura de carpetas, que cuelgan de MotaA, así como un conjunto de archivos cuyos nombres podemos visualizar mediante la instrucción:

```
ls ./build/cou24
```

    - Si la compilación se ha realizado satisfactoriamente, debemos obtener el siguiente listado:

```
app.c            main.exe        main.ihex  main.srec.out-1  wiring-check.xml
ident_flags.txt  main.exe.out-1  main.srec  tos_image.xml
```

- Carga en la mota A:
    - Conectamos la mota A en uno de los puertos USB de nuestro equipo.
    - Para confirmar la referencia asignada al dispositivo debemos ejecutar, desde el terminal de gnome, la instrucción:

```
motelist
```

    - La respuesta de la misma debe ser similar a la siguiente, donde observamos que podemos acceder a la mota A a través de /dev/ttyUSB0

```
Reference  Device           Description
---------- ---------------- ----------------------------------------------
0001       /dev/ttyUSB0     Silicon Labs CP2102 USB to UART Bridge Controller
```

    - Situados nuevamente en la carpeta MotaA, desde el terminal de gnome, procedemos a la carga de la aplicación propiamente en la mota A. Para ello ejecutamos:

```
meshprog -t/dev/ttyUSB0 -f./build/cou24/main.srec.out-1
```

- Nótese que debemos utilizar, dentro del primer argumento, la referencia obtenida de la instrucción anterior.

- Inmediatamente a continuación reseteamos la mota presionando el botón de reset que es el situado más próximo al conector USB de la mota, con lo que en el terminal de gnome debemos visualizar:

```
using tty /dev/ttyUSB0
reading from file ./build/cou24/main.srec.out-1
sending ./build/cou24/main.srec.out-1 -> /dev/ttyUSB0

Opened file ./build/cou24/main.srec.out-1
Opened device /dev/ttyUSB0
.........................,[i••&]
Starting transmission.
finished!
```

- Ejecución:

  - Una vez cargada, la ejecución de la aplicación es automática, sólo requiriendo que la mota esté conectada a un puerto USB del equipo donde se ejecute la aplicación PC para su alimentación y comunicación de datos.

Aplicación Mota B:

- Compilación:

  - Desde el terminal de gnome, situados en la carpeta MotaB, ejecutamos la instrucción:

```
make cou24 install,2
```

  - Además de las indicaciones mostradas por pantalla acerca de la correcta consecución de la operación conforme a la figura 22, este comando genera una estructura de carpetas, que cuelgan de MotaB, así como un conjunto de archivos cuyos nombres podemos visualizar mediante la instrucción:

```
ls ./build/cou24
```

  - Si la compilación se ha realizado satisfactoriamente, debemos obtener el siguiente listado:

```
app.c           main.exe        main.ihex  main.srec.out-2  wiring-check.xml
ident_flags.txt  main.exe.out-2  main.srec  tos_image.xml
```

- Carga en la mota B:

  - Aunque no es imprescindible, para mayor uniformidad, no desconectaremos la mota A del equipo. De esta forma, conectamos la mota B en otro puerto USB.

  - Para confirmar la referencia asignada al dispositivo debemos ejecutar, desde el terminal de gnome, la instrucción:

```
motelist
```

  - La respuesta de la misma debe ser similar a la siguiente, donde observamos que podemos acceder a la mota B a través de /dev/ttyUSB1 ya que la anterior referencia corresponde a la mota A.

```
Reference  Device           Description
---------- ---------------- ----------------------------------------------
0001       /dev/ttyUSB0     Silicon Labs CP2102 USB to UART Bridge Controller
0001       /dev/ttyUSB1     Silicon Labs CP2102 USB to UART Bridge Controller
```

- Situados nuevamente en la carpeta MotaB, desde el terminal de gnome, procedemos a la carga de la aplicación propiamente en la mota B. Para ello ejecutamos:

```
meshprog -t/dev/ttyUSB1 -f./build/cou24/main.srec.out-2
```

- Nótese que debemos utilizar, dentro del primer argumento, la referencia obtenida de la instrucción anterior.

- Inmediatamente a continuación reseteamos la mota presionando el botón de reset que es el situado más próximo al conector USB de la mota, con lo que en el terminal de gnome debemos visualizar:

```
using tty /dev/ttyUSB1
reading from file ./build/cou24/main.srec.out-2
sending ./build/cou24/main.srec.out-2 -> /dev/ttyUSB1

Opened file ./build/cou24/main.srec.out-2
Opened device /dev/ttyUSB1
.........................,[i••&]
Starting transmission.
finished!
```

- Ejecución:

  - Una vez cargada, la ejecución de la aplicación es automática, sólo requiriendo su alimentación mediante baterías o la conexión a un puerto USB de cualquier equipo encendido.

Aplicación PC

- Compilación:

  - Con el objetivo de facilitar su uso, considerando que la aplicación será utilizada siempre desde un mismo equipo con una configuración conocida, ésta se ha diseñado de forma que no requiera parámetros para su ejecución. A estos efectos se exige que la mota A se encuentre conectada vía USB en /dev/ttyUSB0 y la existencia en el PC de la estructura de carpetas implícita en la variable rutaFichero.

  - De esta forma, previo a su compilación, debemos verificar estos extremos y, en su caso, realizar las modificaciones oportunas sobre la variable rutaFichero (línea 17) y source (línea 56) del archivo PrintfSitWsn.java. Estos valores, ajustados al equipo donde y para el que se ha diseñado, son:

```
private static String rutaFichero = "/home/asuarez/Escritorio/Carpeta
compartida/reporte.csv";

String source = "serial@/dev/ttyUSB0:19200";
```

  - De otra parte, aunque el conjunto de archivos y directorios situados en la carpeta PC se encuentran preparados para su compilación y ejecución en Eclipse, también podemos acceder a la subcarpeta “src” y compilar la misma en modo comando mediante:

```
javac PrintfSitWsn.java
```

- A estos efectos, conforme se indicó en el apartado 2.3.1, es necesario que la variable de entorno CLASSPATH incluya “tinyos.jar” que se encuentra en la subcarpeta “lib”.

- Si la compilación se ha realizado correctamente, habrá generado un archivo de nombre “PrintfSitWsn.class”

- Ejecución:

  - En el mismo entorno tras su compilación, podemos ejecutarla mediante la instrucción:

```
java PrintfSitWsn.class
```

  - En caso correcto, en la pantalla visualizaremos:

```
Thread[Thread-1,5,main]serial@/dev/ttyUSB0:19200: resynchronising
```

  - A continuación se recibirán los registros derivados de la circulación de vehículos que, a efectos informativos, se mostrarán por pantalla en el mismo formato que se almacenan en el reseñado archivo reporte.csv

Aplicación SitWsn

- Compilación:

  - No requiere compilación ya que consiste en un archivo de Microsoft Excel 2007 que procesará, *offline*, la información contenida en el archivo reporte.csv generado por la aplicación PC.

- Ejecución:

  - Abrimos el archivo SitWsn.xlsm y procedemos a habilitar el uso de macros siguiendo las instrucciones que la misma aplicación provee.

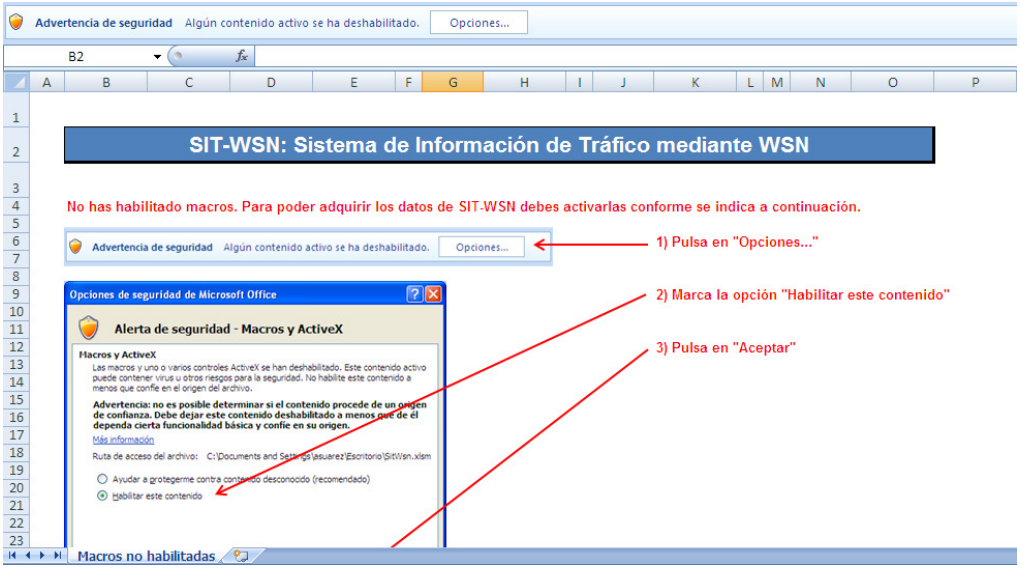


Figura 29: Ayuda contextual de SitWsn.xlsm

- Tras ello, la aplicación muestra el siguiente aspecto.

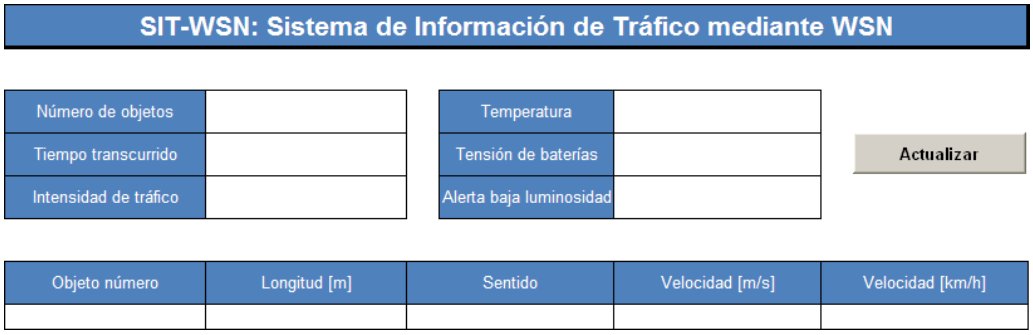


Figura 30: Pantalla principal de SitWsn.xlsm

- La descripción de cada campo identifica suficientemente su contenido que, no obstante, aclararemos sobre un informe concreto. Continuamos pulsando en el botón “Actualizar” que abre un cuadro de diálogo donde seleccionar el archivo reporte.csv

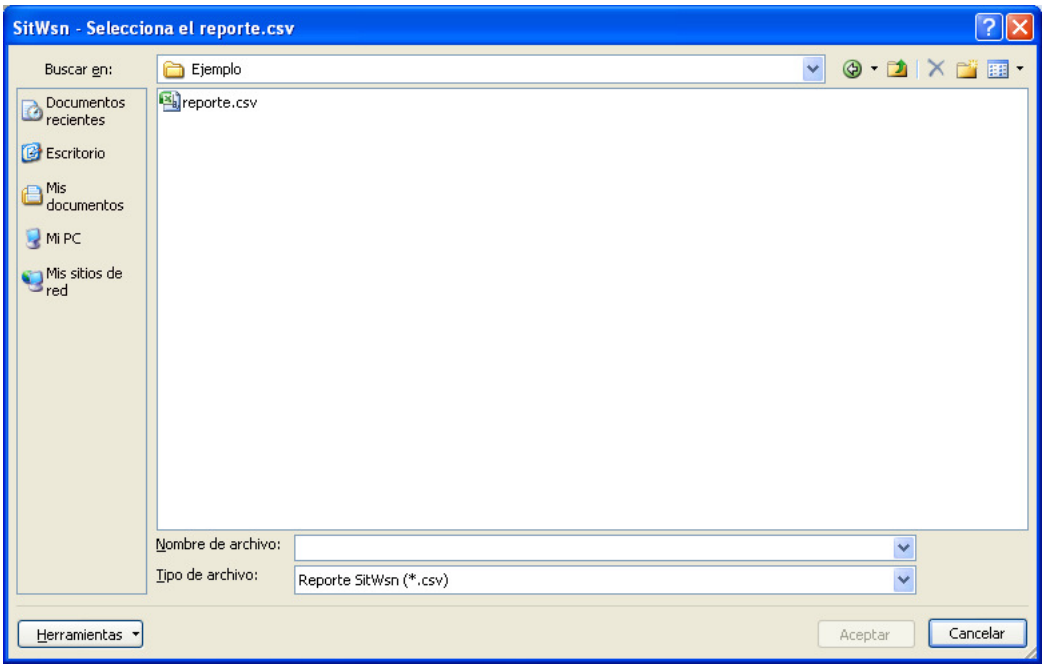


Figura 31: Formulario de selección del reporte.csv

- Tras seleccionarlo y pulsar en aceptar, el sistema lo procesa y nos indica el número de errores detectados entre los que se incluyen las incidencias en comunicaciones y transiciones doblemente señalizadas.

Figura 32: Información del número de errores

- El informe que genera un reporte.csv concreto es el siguiente:

SIT-WSN: Sistema de Información de Tráfico mediante WSN

Número de objetos	17 objetos
Tiempo transcurrido	56,39 s
Intensidad de tráfico	0,30 objetos/s

Temperatura	28,50 ºC
Tensión de baterías	3,07 V
Alerta baja luminosidad	Luminosidad adecuada

Actualizar

Objeto número	Longitud [m]	Sentido	Velocidad [m/s]	Velocidad [km/h]
1	0,05	Mota A ◄– Mota B	1,52	5,45
2	0,08	Mota A –► Mota B	3,21	11,55
3	0,07	Mota A –► Mota B	1,80	6,47
4	0,06	Mota A ◄– Mota B	1,26	4,53
5	0,08	Mota A –► Mota B	1,75	6,31
6	0,05	Mota A ◄– Mota B	1,34	4,82
7	0,12	Mota A ◄– Mota B	1,24	4,46
8	0,13	Mota A –► Mota B	0,56	2,02
9	0,12	Mota A ◄– Mota B	1,05	3,78
10	0,13	Mota A –► Mota B	1,04	3,73
11	0,12	Mota A ◄– Mota B	0,93	3,35
12	0,15	Mota A –► Mota B	1,30	4,67
13	0,11	Mota A ◄– Mota B	1,13	4,05
14	0,15	Mota A –► Mota B	1,26	4,53
15	0,11	Mota A ◄– Mota B	1,10	3,95
16	0,15	Mota A –► Mota B	1,40	5,03
17	-0,22	Mota A ◄– Mota B	-1,11	-3,98

Figura 33: Informe SitWsn de un reporte.csv concreto

- El informe nos indica el número de objetos que han circulado durante el tiempo transcurrido [segundos] y, a partir de éstos, la intensidad de tráfico [objetos / segundos].

- También nos ofrece la temperatura [ºC], la tensión de baterías [V] y una alarma que nos informa si la luminosidad es adecuada o no.

- Asimismo, se genera una alarma por temperatura extrema si ésta es inferior a 0ºC ò superior a 40 ºC. De otra parte, se genera una alarma por tensión baja de baterías si es inferior a 2,60 V. Éstos valores se han fijado de acuerdo a las exigencias del sistema y a la información contenida en los datasheets de cada sensor.

- Si seleccionamos la celda donde se indica el valor de la temperatura o la tensión de baterías se nos habilita un menú contextual que nos informa al efecto.

Figura 34: Alarmas y menú contextual

- Por último, la alarma por baja luminosidad exige que el valor de ésta (considerando su rango ADC [0, 1023]) sea inferior a 550 durante 8 segundos.

- En relación a los registros de cada objeto, los 6 primeros consisten en un objeto de 0,065 m. A partir del registro 7, utilizamos otro objeto de 0,11 m. Las diferencias respecto de los valores de longitudes obtenidas se deben a los errores de precisión derivados de la variación del valor de luminosidad, de la frecuencia de muestreo y de la diferencia de sincronismo.

- De hecho, el registro 17 presenta un valor negativo motivado por un valor extremo en la variable aleatoria diferencia de sincronismo.

- Si volvemos a pulsar en el botón “Actualizar”, asume el mismo origen y, si éste se ha modificado, actualiza los cambios.

- De hecho, en el entorno de desarrollo la aplicación PC actualiza el reporte.csv en una carpeta compartida mediante samba que, para el usuario “asuarez”, el sistema mapea convenientemente facilitando un efecto de actualización online.

Bibliografía

[1] Vilajosana, Xavier. (2012, 25 de febrero). “Arp@:Embedded Systems Lab@Home”. UOC. Estudios de Informática, Multimedia y Telecomunicación. [página web en línea]. [Fecha de consulta: 27 de febrero de 2012].
<http://cv.uoc.edu/app/mediawiki14/>

[2] VV.AA. (2011, 1 de agosto). “TinyOS Home Page”. TinyOS Working Groups. [página web en línea]. [Fecha de consulta: 27 de febrero de 2012]
<http://tinyos.net/>

[3] Atmel Corporation. (2012). “8-bit Atmel Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash”. Atmel Corporation. [documento en línea]. 2549O–AVR–05/12. [Fecha de consulta: 15 de marzo de 2012].
<http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf>

[4] Advanced Photonix, Inc. (2011, 17 de octubre). “CdS Photoconductive Photocells”. Advanced Photonix, Inc. [documento en línea]. PDV-P9003-1. [Fecha de consulta: 15 de marzo de 2012].
<http://www.advancedphotonix.com/ap_products/pdfs/PDV-P9003-1.pdf>

[5] Microchip Technology Inc. (2009, 26 de marzo). “Low-Power Linear Active Thermistor™ ICs”. Microchip Technology Inc. [documento en línea]. DS21942E. [Fecha de consulta: 15 de marzo de 2012].
<http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf>

[6] Levis, Philip; Gay, David. (2009, 16 de julio). “TinyOS Programming”. [documento en línea]. [Fecha de consulta: 27 de febrero de 2012].
<http://csl.stanford.edu/~pal/pubs/tos-programming-web.pdf>

[7] Ulloa Suárez, José Francisco. (2007, 28 de mayo). “Estudio y Análisis de las Características de TinyOS”. [documento en línea]. [Fecha de consulta: 20 de mayo de 2012].
<http://alumnos.elo.utfsm.cl/~julloa/titulo/download/UnoSeccionDos.doc>

[8] Rellermeyer, J. S. (2004). “TinyOS - Distributed Systems Seminar WS 04/05”. [documento en línea]. [Fecha de consulta: 20 de mayo de 2012].
<http://people.inf.ethz.ch/rjan/publications/TinyOS_Slides.pdf>

[9] He, Tian. (2001, 9 de septiembre). “Get Start with TinyOS From technique perspective”. [documento en línea]. [Fecha de consulta: 20 de mayo de 2012].
<http://www.cs.virginia.edu/~cl7v/cs851-talks/tinyos_tian.ppt>