UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

DEPARTAMENTO DE TECNOLOGIA ELECTRÓNICA

PROYECTO FIN DE CARRERA

INGENIERO TÉCNICO INDUSTRIAL
ESPECIALIDAD EN ELECTRÓNICA INDUSTRIAL

"DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO DE VIDEOPORTERO BASADO EN REDES INALÁMBRICAS DE SENSORES"

DIRECTORES: **D. Juan Suardíaz Muro**
D. Juan A. López Riquelme

AUTOR: **David Manuel Pérez Ortega**

Cartagena, Enero de 2012

AGRADECIMIENTOS

- A Juan Antonio y a Juan por la ayuda que me han prestado en la realización de este proyecto, incluso fuera del horario lectivo.
- A mi padre y a mi madre, que siempre han confiado en mí y han estado ahí cuando más los necesitaba.
- A mi hermano por recordarme que no hay que estar siempre trabajando y que hay que divertirse.
- A toda mi familia que han seguido mis estudios.

ÍNDICE

Capítulo 1

INTRODUCCIÓN

1.1 Planteamiento inicial del proyecto

El presente proyecto se centra en el estudio de las redes de sensores inalámbricas con el objetivo de desarrollar un prototipo demostrativo de un sistema de vídeo portero basado en esta tecnología. Con el fin de cumplir dicho objetivo, se realizará capturas del visitante a través de una cámara conectada a uno de los motes cuanto este pulse la llamada en el portero a la vivienda a visitar.

Esta entrada de imágenes serán retransmitidas al mote que actúa de base y que estará conectado a un ordenador, en el cual se visualizará dichas imágenes. Una vez visualizadas por el propietario del inmobiliario, este decidirá la apertura de la puerta a través de los mismos motes que retransmitieron las imágenes. Con esto se consigue el contacto audiovisual por parte del usuario con el visitante. En el caso de que el usuario no estuviera en casa, las imágenes tomadas del visitante serán almacenadas en el ordenador base para que posteriormente, sean estudiadas por el usuario a su llegada a la vivienda.

Como ya se ha comentado, se utilizarán como nodos de la red los motes. Además, se utilizará una cámara denominada CMUcam de tercera generación, específica para redes de sensores inalámbricos. También se precisará de una cerradura electrónica para la apertura del portal, así como de micrófonos y altavoces para la comunicación entre el propietario y el visitante.

Así pues, se irán describiendo a continuación todos los elementos necesarios para llevar a cabo este proyecto, desde todas las posibilidades que abarca el mercado en cuanto a motes y a nodos de redes de sensores inalámbricas o gestionar todos los circuitos de acondicionamiento necesarios para poder manejar la cámara o la apertura de la cerradura electrónica, así como de los pulsadores adicionales, todo ello a través del

nodo de la red conectado en el portero encargado de la monitorización y control del portal.

El presente proyecto se ha llevado a cabo utilizando el sistema operativo TinyOS, usando para el desarrollo del software un lenguaje de programación, variante del lenguaje C, llamado NesC, que permitirá programar los diferentes nodos de la red para conseguir el funcionamiento esperado.

1.2 Fases del proyecto

Para el correcto desarrollo del proyecto se han establecido las siguientes fases que se exponen a continuación:

1. Búsqueda y selección de una cámara de bajo coste que permita establecer una comunicación audiovisual adecuada.
2. Estudio del sistema operativo TinyOS, con el que se programará la red de sensores.
3. Desarrollo de un software cliente para los diferentes residentes de la comunidad.
4. Diseño y desarrollo de la electrónica de acondicionamiento entre la cámara y un nodo sensor de la red inalámbrica.
5. Diseño y desarrollo de la electrónica de acondicionamiento para una cerradura comercial.
6. Montaje de un prototipo demostrativo.
7. Pruebas y puesta a punto
8. Redacción del documento final de memoria de proyecto.

1.3 Desarrollo de la memoria

La memoria de este proyecto se divide en seis capítulos, en los que se describe, de forma detallada, todo los puntos presentados anteriormente. En este apartado se muestra un breve resumen de cada unos de los capítulos que conforman esta memoria, para su posterior desarrollo en cada uno de ellos:

- **Capítulo 1. Introducción**

En este capítulo se presentan los puntos y los objetivos para la realización y desarrollo del presente proyecto, así como las fases en las que está dividido.

- **Capítulo 2. Estado del Arte**

En este capítulo se describe el estado del arte de las redes de sensores inalámbricas, indicando sus características, los protocolos que utilizan, los componentes que lo forman, así como sus distintas aplicaciones en diferentes entornos. Así mismo, se explicará el concepto de nodos de sensores, su clasificación, atendiendo a la variedad de nodos existentes actualmente en el mercado, y sus modos de funcionamiento.

- **Capítulo 3. Descripción del Sistema**

En este capítulo se describirán todos los componentes que forman este proyecto, explicando todas sus características, sus usos, así como las posibilidades que ofrecen, y todo ello unido a su instalación y/o mantenimiento.

Además, en este capítulo se presenta una descripción del sistema operativo TinyOS, utilizado para la programación de los motes en este proyecto, y donde se desarrollará todas sus características, así como una descripción del lenguaje de programación utilizado.

- **Capítulo 4. Componentes Hardware**

En este capítulo se expone una descripción de todos los componentes y dispositivos que componen el hardware desarrollado en este proyecto, explicando todas sus características y los circuitos de acondicionamiento, los cuales son fundamentales para la conexión entre los diferentes componentes.

- **Capítulo 5. Descripción del Software**

Pero como es lógico, para el correcto funcionamiento del Hardware, este tiene que ir acompañado de una eficiente programación Software. Es por eso que en este capítulo, se describirá todos los programas y aplicaciones compilados en los motes, para poder conseguir el adecuado funcionamiento de este prototipo. También, se describirá la aplicación java desarrollada con el objetivo de controlar y gestionar los datos del mote principal, desde un ordenador que estará presente en cada vivienda, y que estará conectado a un mote que hará de estación base, lo que hará posible la comunicación con el mote principal.

- **Capítulo 6. Resultados, conclusiones y trabajos futuros**

Este capítulo es un breve resumen de todos los objetivos conseguidos para la culminación de este proyecto, así como de los resultados obtenidos, concluyendo todos los trabajos realizados y proponiéndose posibles proyectos futuros, trabajos o investigaciones relacionadas con las redes de sensores inalámbricas aplicadas a todo tipo de sistemas, en las cuales es necesaria una supervisión óptica del entorno.

Capítulo 2

ESTADO DEL ARTE

2.1 Redes de sensores Inalámbricas

Continuando con lo expuesto en el capítulo anterior, el principal objetivo de este proyecto es el diseño de un prototipo de videoportero mediante una red de sensores inalámbrica. A lo largo de este capítulo, se expondrá todo lo relacionado con las WSN (*Wireless Sensor NetWork*), sus características, las diferentes aplicaciones, así como las distintas funciones que puede desempeñar en el mercado. También se hará mención de todos los sistemas operativos disponibles para la programación de los nodos de las redes de sensores, entre los que se encuentra TinyOS que será el usado finalmente.

2.1.1 Definición

Las redes inalámbricas de sensores [1], denominados motes, se basan en una gran cantidad de pequeños dispositivos, capaces de recoger todo tipo de información de su entorno: temperatura, humedad, luz, movimiento…etc., a través de los sensores que llevan incorporados. Su reducido tamaño y la capacidad de transmitir sin cables, permiten un despliegue rápido y flexible de centenares de dispositivos.

Los últimos avances tecnológicos han hecho realidad el desarrollo de unos mecanismos distribuidos, diminutos (con él objetivo de que ocupen el menor espacio posible), baratos y de bajo consumo, que, además, son capaces tanto de procesar información localmente como de comunicarse de forma inalámbrica. La disponibilidad de micro sensores y comunicaciones inalámbricas permite desarrollar redes de sensores/actuadores para un amplio rango de aplicaciones.

Con todo esto, una red de sensores puede ser descrita como un grupo de motes que se coordinan para llevar a cabo una determinada aplicación. Al contrario que las redes tradicionales, las redes de sensores llevarán con más precisión sus tareas dependiendo de lo denso que sea el despliegue y lo coordinadas que estén.

Los motes integran sensores para realizar mediciones de luz, temperatura, presión, humedad, etc. Posee algunas limitaciones en cuanto se refiere a energía, potencia y memoria. Además hacen un uso intensivo de la radio para enviar/recibir y de la CPU para procesamiento.

Como están constituidos con un procesador, estos nodos son capaces de realizar cálculos y procesar los datos recibidos por parte de los sensores.

En definitiva, los elementos que constituyen la WSN son:

- ***Sensores*:** son los encargados de captar la información del entorno donde se encuentran, y convirtiéndola en señales eléctricas para su posterior procesamiento y análisis. Existen sensores de varios tipos (humedad, temperatura, acelerómetro, etc.).
- ***Los motes*:** recogen la información obtenida de los sensores y la retransmite a otros motes o a una estación base, además son capaces de actuar según estén programados para ciertas variaciones de los sensores.
- ***Pasarelas o Gateways*:** son elementos para la interconexión entre la red de sensores y una red TCP/IP.
- ***Estaciones base*:** es el mote final que se encuentra conectado a un ordenador para la visualización por parte del usuario de los datos recogidos.
- ***Red inalámbrica*:** típicamente basada en el estándar 802.15.4 *ZigBee.*

2.1.2 Características

Con el objetivo de desarrollar motes de coste mínimo, de pequeño tamaño, con un reducido consumo y que, además se comunican de forma inalámbrica, se ha creado redes basadas en la cooperación de los nodos, con una destacada mejora con respecto a las redes de sensores tradicionales.

A pesar de que la red pueda estar formada por nodos con una naturaleza y funciones muy diferentes, todos ellos tienen una serie de características comunes:

- Gran escala.

Una red puede estar continuamente en crecimiento, incluyendo en ella un centenar de nodos llegando hasta miles de nodos. Con el objetivo de controlar y analizar el fenómeno que se desea estudiar, esta red estará compuesta por una gran cantidad de sensores repartidos densamente alrededor de dicho fenómeno.

- Topología variable.

Los nodos serán colocados de forma arbitraria o estratégica y normalmente desconocida para el resto de los nodos. Esto es debido a que no es necesaria la localización preestablecida para permitir un despliegue aleatorio en terrenos inaccesibles u operaciones de alivio en desastres.

- Cooperación de nodos sensores.

Realizan un procesamiento local antes de transmitir los datos.

- Recursos limitados.

Los sensores disponen de recursos muy limitados debido a su bajo consumo.

- Funcionamiento autónomo.

Puede darse el caso que el nodo se le agoten las baterías, o funcione de forma inesperada al permanecer desatendido porque no se puede acceder a él físicamente.

- Comunicación.

Los nodos sensores usan comunicación por difusión, debido a que están densamente desplegados y los nodos vecinos están muy cerca unos de otros. Con la comunicación *multihop*, que es la comunicación entre dos nodos finales que se lleva a cabo a través de una serie de nodos intermedios, y cuya función es transmitir información desde un punto a otro, se consigue un menor consumo de potencia que la comunicación *single-hop*, que a diferencia de la anterior, no hay nodos intermedios. Además, los niveles de transmisión de potencia se mantienen muy bajos y existen menos problemas de propagación en comunicaciones inalámbricas a largas distancias.

2.1.3 Requisitos para una WSN

Para que una red pueda funcionar de acuerdo con las características descritas en el apartado anterior surgen una serie de requisitos no funcionales que una aplicación debe cumplir.

- Eficiencia energética.

  Este es uno de los puntos más importantes, ya que cuanto menor sea el consumo de un nodo mayor será el tiempo durante el cual pueda estar funcionando y realizando su tarea y, por lo tanto, mayor tiempo seguirá la red operativa. Para ello, las aplicaciones deberán de poseer la capacidad de bajar el consumo de potencia restringiendo el uso de la CPU y la radio FM. Esto se consigue desactivándolos cuando no se utilizan y, sobre todo, disminuyendo el número de mensajes que generan y retransmiten los nodos.

- Auto organización.

  Los nodos desplegados deben estar organizados de manera que permita establecer comunicaciones por las que mandar los datos que han obtenido. Los nodos necesitan conocer su lugar en esta topología, pero resulta inviable asignarlo manualmente a cada uno, debido al gran número de nodos. Por ello, es fundamental que sean capaces de formar la topología deseada sin ayuda del exterior de la red. Este proceso no sólo debe ejecutarse cuando la red comienza su funcionamiento, sino que debe permitir que, en cada momento, la red se adapte a los cambios que pueda haber en ella.

- Escalabilidad.

  Como las aplicaciones siguen en crecimiento durante el tiempo y el despliegue de la red es progresivo, es necesario que la red esté organizada de forma que permita su crecimiento sin que el rendimiento de la misma se vea afectado.

- Tolerancia a fallos.

Es importante reducir al máximo los fallos, ya que éstos pueden, bien por un mal funcionamiento de un nodo, ya sea por el estado de las baterías, un error de programación que no se ha tenido en cuenta, por las condiciones ambientales a las que está expuesto el nodo, el estado de la red no sea el adecuado, etc., lo que provoque el mal funcionamiento del conjunto de la red o deje de estar operativa.

- Tiempo real.

  Unos de los inconvenientes de estos dispositivos es que los datos llegan a su destino con cierto retraso. Pero algunos datos deben entregarse dentro de un intervalo de tiempo conocido. Pasado éste dejan de ser válidos, como puede pasar con datos que impliquen una reacción inmediata del sistema, o se pueden originar problemas serios. En caso de que una aplicación tenga estas restricciones, se deben tomar las medidas que garanticen la llegada a tiempo de los datos.

- Seguridad.

  Este aspecto es de considerable importancia a tener en cuenta, ya que al establecerse las comunicaciones inalámbricas por un medio fácilmente accesible puedan estar expuestas a personas ajenas a la red de sensores. Esto implica un riesgo potencial para los datos recolectados y para el funcionamiento de la red. Se deben establecer mecanismos que permitan tanto proteger los datos de estos intrusos, como protegerse de los datos que estos puedan inyectar en la red.

Según la aplicación que se diseñe, algunos de los anteriores requisitos cobran mayor importancia. Es necesario encontrar el peso que cada uno de ellos tiene en el diseño de la red, pues normalmente unos requisitos van en detrimento de otros. Por ejemplo, dotar a una red de propiedades de tiempo real podría implicar aumentar la frecuencia con la que se mandan mensajes con datos, lo cual repercutiría en un mayor consumo de potencia y un menor tiempo de vida de los nodos.

Esto lleva a buscar, para cada aplicación, un compromiso entre los requisitos anteriores que permita lograr un funcionamiento de la red adecuado para la misión que debe realizar.

2.1.4 Arquitectura de una WSN

Tomando como elementos de una red los nombrados anteriormente, podemos distinguir varios tipos de arquitecturas [2]:

2.1.4.1 Arquitectura Centralizada

Los nodos de una red que estudian un determinado fenómeno enviarán sus datos directamente al Gateway más cercano, dirigiendo el tráfico de forma concreta, tal como se muestra en la Figura 2.1.

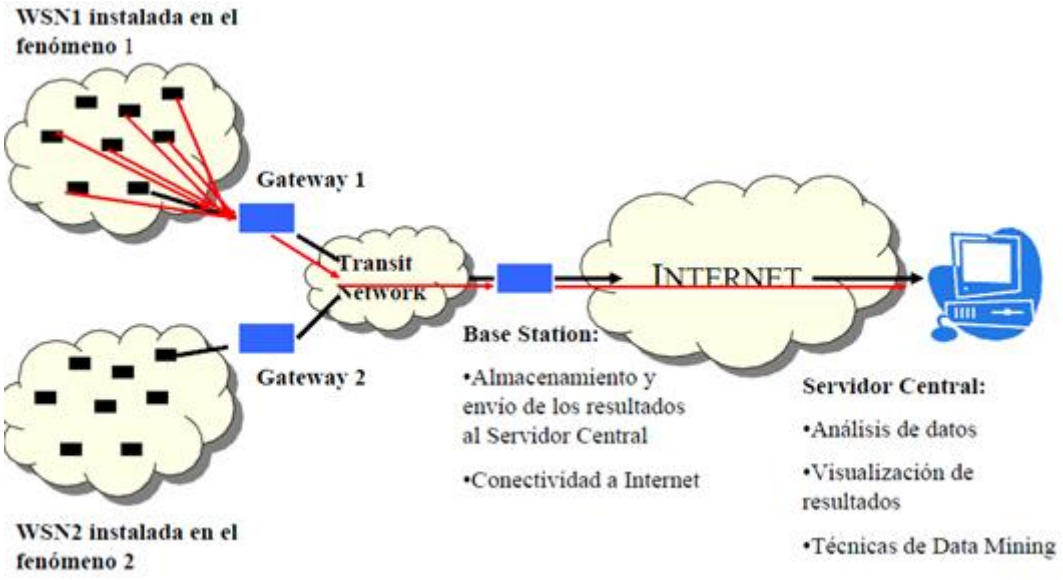


Figura 2. 1: Esquema de arquitectura centralizada

El ciclo de vida de un nodo consiste en despertarse, medir, transmitir y dormirse. No obstante, cada vez que transmita mensajes pasará, por el Gateway, creándose dos grandes problemas para la red:

- Cuello de botella en las pasarelas.
- Mayor consumo de energía por las comunicaciones, acortándose de ésta manera el tiempo de vida de la red.

2.1.4.2 Arquitectura Distribuida

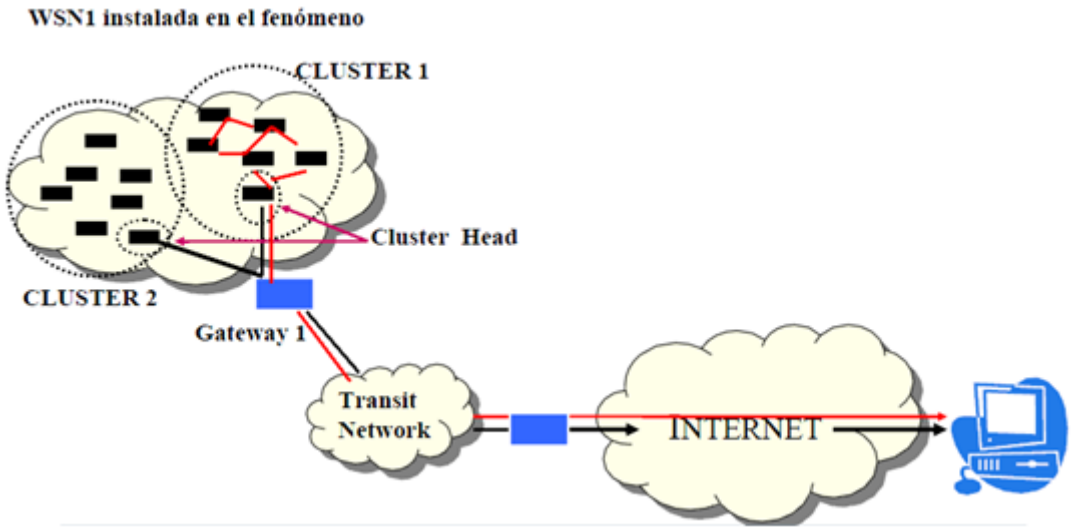


Figura 2. 2: Esquema de arquitectura distribuida

La Figura 2.2 anterior muestra cómo la comunicación entre nodos es fundamental, puesto que se basa en la cooperación de los mismos, garantizándose, de esta manera, un tipo de arquitectura con una computación distribuida, donde los nodos

sensores se encargan de ejecutar algoritmos distribuidos para obtener una única respuesta global que un nodo (clúster head) se encargará de comunicar a la estación base a través de las Gateway pertinentes. Además realiza un cálculo de agregados con un grano más grueso de los datos y unas respuestas más elaboradas.

2.1.5 Aplicaciones de las redes de sensores

Debido a las características y ventajas de las redes de sensores, tienen una amplia variedad de aplicaciones entre los que se encuentran:

- Monitorización de un hábitat (para determinar la población y comportamiento de animales y plantas).
- Monitorización del medio ambiente, observación del suelo o agua.
- Mantenimiento de ciertas condiciones físicas (temperatura, luz).
- Control de parámetros en la agricultura.
- Detección de incendios, terremotos o inundaciones.
- Sensorización de edificios "inteligentes".
- Control de tráfico.
- Asistencia militar o civil.
- Control de inventario.
- Control médico.
- Detección acústica.
- Cadenas de montaje, etc.

Debido a su ilimitada variedad de aplicaciones, las redes de sensores inalámbricas (WSN) tienen el potencial de revolucionar los complejos sistemas de control u observación, tal y como hoy se conciben. A continuación, se desarrolla algunas de las aplicaciones de las redes de sensores presentadas anteriormente:

- Aplicaciones de localización.

  Otra de las aplicaciones de las WSN, es la localización de personas u objetos dentro de un área cerrada. Por ejemplo, en un hospital puede resultar útil tener localizados a determinados enfermos para evitar que salgan de su zona de acceso o monitorizar al personal de la limpieza en aquellos centros de los que se disponga de este servicio, por ejemplo en hoteles, hospitales, universidades, etc.

- Aplicaciones militares.

  Las WSN pueden ser piezas en clave para sistemas militares que llevan acabo tareas como dar las órdenes y llevar el control de las comunicaciones, el procesamiento de los datos obtenidos, inteligencia, vigilancia, y reconocimientos de objetivos militares.

  Debido a su rápido y denso despliegue, su autoorganización y tolerancia a fallos las redes de sensores convierten en las respuesta para dar solución a este tipo de aplicaciones, ofreciendo una solución de bajo coste y fiable para éstas, ya que la pérdida de un nodo no pone en riesgo el éxito de las operaciones.

Ejemplos de esta aplicación de las WSN en el ámbito militar puede ser los siguientes:

- Monitorización de fuerzas aliadas, equipamientos y munición.
- Reconocimiento del terreno y fuerzas enemigas.
- Adquisición de blancos.
- Reconocimiento y determinación de ataques nucleares, biológicos y químicos.

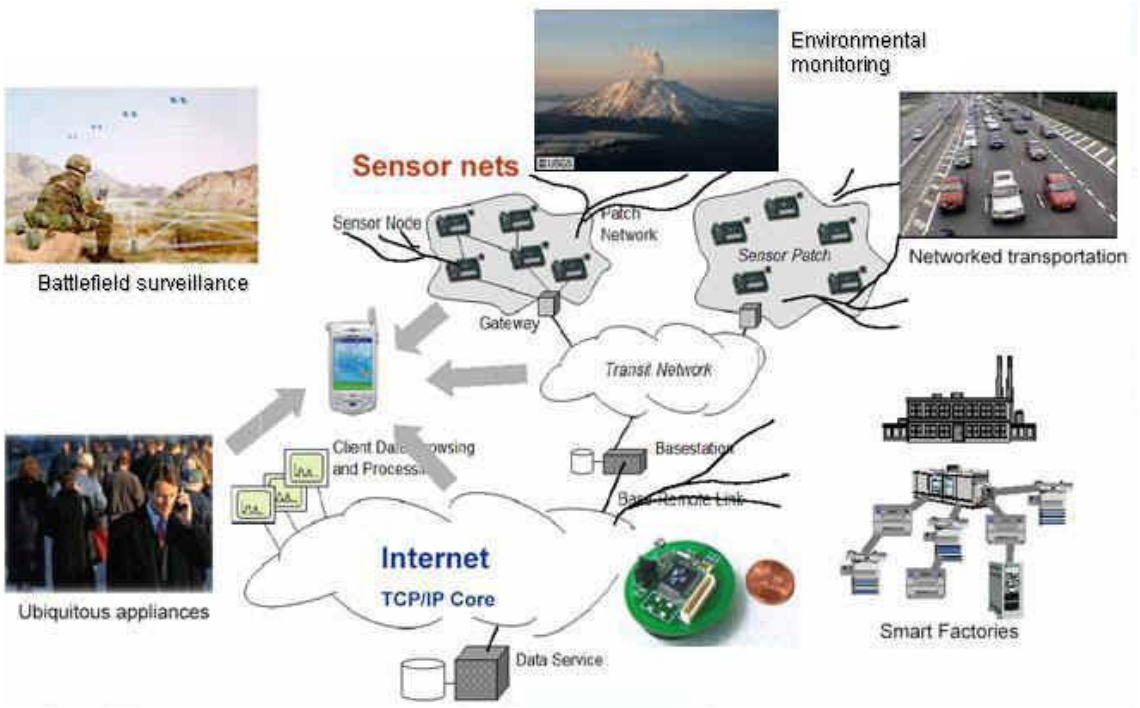


Figura 2.3: Aplicaciones de las WSN

- Aplicaciones medioambientales.

En este ámbito tenemos aplicaciones como el seguimiento de aves, animales e insectos; monitorización de condiciones ambientales que afectan al ganado y las cosechas; detección de incendios; investigación de meteorología o geofísica; detección de inundaciones, control de agricultura, microclimas, etc.

Un ejemplo de aplicación medioambiental podría ser el seguimiento del clima, controlando la humedad, la presión atmosférica, el estado de la iluminación con el objetivo de determinar la presencia de nubes o no, control visual directo de la presencia de dichas nubes, así como la capacidad de reconocer el tipo de nubes que son y si pueden derivar a tormenta, la dirección del viento y su velocidad, así como la temperatura en distintos puntos de la región cuyo clima se desee controlar. Todo esto, utilizando redes inalámbricas de sensores localizados a nivel del suelo, incluso con posible acceso a imágenes satelitales, integrados a un sistema de software y hardware específicos, utilizando para esto tecnología punta en sistemas de comunicación y teledetección satelital.

- Aplicaciones sanitarias.

A medida que va avanzando la tecnología, los chips van reduciendo su tamaño cada vez más, y creciendo su nivel de complejidad. Hasta ahora confinados en las máquinas, comienzan a adaptarse al cuerpo humano, con lo que es posible crear redes de sensores inalámbricos que permitan un control del estado de salud de las personas y sirvan para prevenir problemas y enfermedades. Con esta ayuda tecnológica, el hombre podría anticiparse a muchas patologías como los ataques al corazón.

Figura 2.4: Chips de reducido tamaño

Algunas de las aplicaciones de esta tecnología y que se está estudiando son: la creación de chip con historial médico, circuitos en el corazón, creación de ojos biónicos, tratamientos contra el cáncer, leer el pensamiento, manos biónicas y desarrollo de neuronas artificiales.

- Aplicaciones domóticas.

En este ámbito, los nodos sensores pueden ser introducidos en aparatos domésticos como aspiradoras, microondas, frigoríficos… Esto permite que sean manejados remotamente por los usuarios finales mediante una comunicación que se realizaría vía satélite o Internet.

Los sensores pueden integrarse en muebles y en todo tipo de aparatos del hogar. Éstos se comunican entre ellos y con servidores de la habitación. Todos ellos se integran y organizan con los dispositivos integrados existentes para autoorganizarse, autorregularse y autoadaptarse basándose en modelos de control. De esta forma se puede crear un entorno inteligente.

Otras aplicaciones comerciales, dentro del ámbito de la domótica serían:

- **Control ambiental en edificios de oficinas.**
  Normalmente la calefacción o el aire acondicionado se controlan desde una central, por lo que la temperatura dentro de cada habitación puede variar unos pocos grados debido a que la distribución de aire no está uniformemente distribuida y el control es central. Se podría desplegar una WSN para controlar el flujo de aire y la temperatura en diferentes partes de la habitación, así como detección de personas en el entorno para encendido o apagado de la calefacción. Esto podría ser una solución para ahorrar energía.

- **Gestión de inventarios.**
  Otra aplicación podría ser en un almacén en el que cada artículo incorporase un sensor, de modo que se puede conocer en todo momento su localización y número de artículos por categoría. Para dar de alta nuevos artículos se les incorporaría el sensor y se llevarían al almacén.

2.1.6 Componentes básicos de un nodo sensor

- Unidad sensora: Sensores y conversores analógico-digital (A/D) cuya función es la conversión de las señales analógicas, procedentes de los sensores, en digitales para el tratamiento posterior por el microprocesador. Los conversores A/D adaptan la señal para su procesamiento, llevándose a cabo tres etapas: muestreo, cuantificación y codificación.
- Procesador: Organiza y controla los procesos que permiten al nodo sensor colaborar con otros nodos para realizar las tareas asignadas. Normalmente el procesador está asociado a una pequeña unidad de almacenamiento.
  Aunque cada vez hay procesadores más pequeños y rápidos, las unidades de procesamiento y almacenamiento en los nodos poseen recursos muy limitados.

- Transceptor: Conecta el nodo a la red, realizando las operaciones de transmisión y recepción de mensajes.

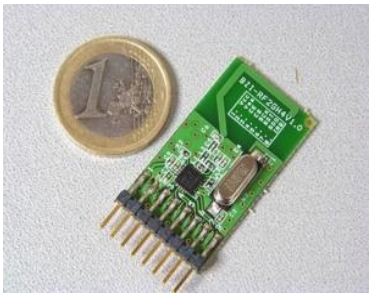


Figura 2.5: Transceptor RF

Los transceptores pueden ser dispositivos ópticos activos o pasivos o el uso de radiofrecuencia. Radiofrecuencia requiere modulación, filtro paso banda, filtrado, demodulación y circuito multiplexor, lo que puede hacelos demasiado complejos y caros.

No obstante se prefieren transceptores RF en la mayoría de los proyectos, porque los paquetes transportados son pequeños y las tasas de transferencia también.

- Alimentación: Se obtiene a partir de las baterías, aunque puede estar ayudado de un generador.

Todo lo descrito anteriormente debe caber en un módulo del tamaño de una caja de cerillas y, a veces, en hasta de tamaño mucho más pequeño, un centímetro cúbico, para poder suspenderlo en el aire.

2.1.7 Clasificación de los sensores

- Atendiendo al fundamento físico.

Este tipo de sensores atienden a una propiedad física cuya variación produce excitación:

- Resistivos: Miden variaciones de la resistencia.
- Capacitivos: Miden variaciones de la capacidad.
- Inductivos: Miden variaciones de la inductancia electromagnética o magnitud de flujo magnético.
- Generadores de tensión o intensidad: La magnitud física provoca la generación de tensión o intensidad en el dispositivo, sin necesidad de alimentación externa.

- Atendiendo a la alimentación.

  - Activos: Ellos mismos generan una tensión o corriente, no requiriendo, por tanto, una alimentación externa. Se basan en diferentes efectos: termoeléctrico, piezoeléctrico…
  - Pasivos: Requieren de una alimentación o excitación externa para generar una señal.

- Atendiendo a la salida.

  - Analógicos: La salida del transductor es un nivel de tensión o intensidad que varía de forma continua con la variable a medir dentro del rango de medida del transductor. Suelen emplearse valores normalizados a 0-10V y 4-20mA.
  - Digitales: La salida está codificada mediante un código binario o en forma de pulsos. Son codificaciones habituales la binaria, Gray o el BCD.
  - Todo-Nada: Se consideran un caso particular de los digitales. La salida sólo presenta dos estados: activa o no activa. Es el caso de dispositivos tales como los detectores de presencia.

- Atendiendo a la magnitud física a medir: posición, velocidad, aceleración, temperatura, fuerza, nivel, presión, etc.

Por otro lado, podemos establecer una serie de factores que evalúan y catalogan también los tipos de nodos sensores. Estos factores serían: energía, flexibilidad, robustez, seguridad, comunicación, computación, sincronización, tamaño y coste.

2.2 Protocolos de una WSN

A continuación se presenta varios protocolos de comunicación disponibles actualmente:

2.2.1 WI-FI

El protocolo Wi-Fi [3] es similar a la red Ethernet tradicional y, por tanto, se necesita una configuración previa para establecer la comunicación. También se le denomina Wi-Fi al "Ethernet sin cables", para dar una gran idea de las ventajas e

inconvenientes que tiene respecto a otras alternativas. La ventaja de una red Wi-Fi es que permite conexiones mucho más rápidas y rangos de distancias mayores, aparte de mejores mecanismos de seguridad, además permiten a los dispositivos una mayor movilidad al no necesitar cables para transferencia de datos.
No obstante, una de sus desventajas es que es muy fácil acceder a estas redes por parte de personas ajenas si la red no está bien configurada o bien protegida, lo que es necesaria la seguridad.
El rango de frecuencia utilizada es la misma que la que usa el Bluetooth, la diferencia es que tiene una potencia de salida mayor que lleva a conexiones más sólidas. Cabe aclarar que esta tecnología no es compatible con otros tipos de conexiones sin cables como Bluetooth, GPRS, UMTS, etc.
Existen varios tipos de dispositivos que se pueden clasificar en dos grupos:

- Dispositivos de distribución o red, entre los que cabe destacar los *routers*, repetidores o puntos de acceso.
- Dispositivos terminales, que suelen ser las tarjetas receptoras empleadas en la comunicación con el ordenador. Pueden ser las tarjetas PCI o bien tratarse de USB externos.

2.2.2 Bluetooth

El protocolo Bluetooth [4] fue diseñado con el objetivo de reemplazar la tecnología con cables, usando una conexión de radio muy segura y de corto alcance. Permite la conexión entre dispositivos a un bajo costo y a cortas distancias. El Bluetooth tiene un alcance normalmente de 10 metros, ya que establece una conexión utilizando el mínimo consumo posible de energía de las baterías. Sin embargo, es posible establecer un alcance mucho mayor, similar al de las redes Wi-Fi, con el inconveniente de un aumento de consumo de energía mucho mayor. Es recomendable que no exista nada físico entre los elementos que se conectan mediante este protocolo para mejorar la comunicación. La banda de radiofrecuencia utilizada sería la ISM (*Industrial, Scientific and Medical*) de los 2,4GHz.

Existen varios tipos de clases de dispositivos que usan el Bluetooth:

- **Clase 1.** Esos dispositivos tienen una potencia máxima de 100 mW o 20 dBm y cuyo rango sería de aproximadamente unos 100 metros.
- **Clase 2.** Serían aquellos dispositivos con una potencia máxima de 2,5 mW o 4 dBm y con un alcance aproximado de unos 20 m.
- **Clase 3.** Son los de mínimo consumo (1mW) y mínimo alcance (1m aproximadamente).

Actualmente existen diferentes versiones de bluetooth, cuya diferencia es su ancho de banda, donde es posible encontrar la Versión 1.2 de 1Mbits/s, Versión 2.0+EDR de 3Mbits/s y la versión 3.0+HS de 24 Mbits/s.
La tecnología Bluetooth se utiliza en productos como módems, teléfonos móviles, auriculares, control remoto como el de los mandos de algunas videoconsolas, y en todo tipo de gadgets.

2.2.3 IEEE 802.15.4

IEEE 802.15.4 [5] es un estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con bajas tasas de transmisión de datos (LRWPAN, *Low Rate Wireless Personal Area Network*).
También es la base sobre la que se define la especificación de *ZigBee*, cuyo propósito es ofrecer una solución completa para este tipo de redes construyendo los niveles superiores de la pila de protocolos que el estándar IEEE 802.15.4 no cubre.

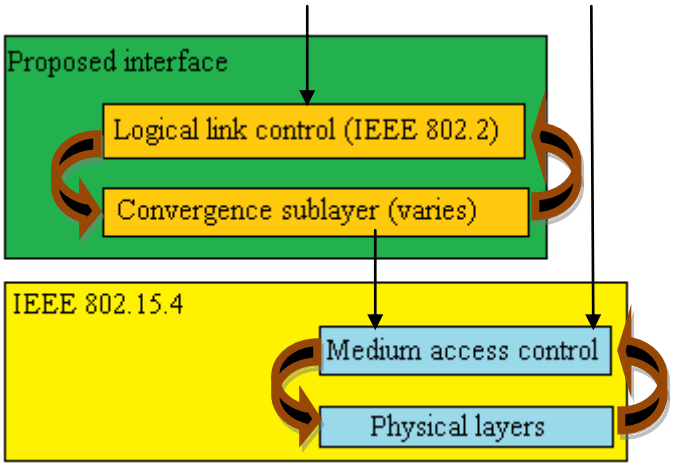


Figura 2. 6: Pila de protocolo IEEE 802.15.4

El propósito del estándar es definir los niveles de red básicos para dar servicio a un tipo específico de red inalámbrica de área personal WPAN, (*Wireless Personal Area Network*) centrada en la habilitación de comunicación entre dispositivos ubicuos con bajo coste y velocidad (en contraste con esfuerzos más orientados directamente a los usuarios medios, como WiFi). Se enfatiza el bajo coste de comunicación con nodos cercanos y sin infraestructura o con muy poca, para favorecer aún más el bajo consumo.
En su forma básica se concibe un área de comunicación de 10 metros con una tasa de transferencia de 250 kbps. Se pueden lograr tasas aún menores con la consiguiente reducción de consumo de energía. Como se ha indicado, la característica fundamental de 802.15.4 entre las WPAN's es la obtención de costes de fabricación excepcionalmente bajos por medio de la sencillez tecnológica, sin perjuicio de la generalidad o la adaptabilidad.

Entre los aspectos más importantes, se encuentra la adecuación de su uso para tiempo real por medio de slots de tiempo garantizados, evitación de colisiones por CSMA/CA (*Carrier Sense, Multiple Access, Collision Avoidance*) [6] y soporte integrado a las comunicaciones seguras. También se incluyen funciones de control del consumo de energía como calidad del enlace y detección de energía.
Un dispositivo que utilice el 802.15.4 puede transmitir en las siguientes bandas de frecuencia:

- En Europa entre 868 y 868.8 MHz, permitiendo hasta tres canales.
- En América del Norte entre 902 y 928 MHz, extendido a treinta canales.
- 2400-2483.5 MHz de uso para todo el mundo, con la posibilidad de hasta dieciséis canales.

El *control de acceso al medio* (MAC, *Medium Access Control*) transmite tramas MAC usando para ello el canal físico. Además del servicio de datos, ofrece un interfaz de control y regula el acceso al canal físico y al balizado de la red. También controla la

validación de las tramas y las asociaciones entre nodos y garantiza slots de tiempo. Por último, ofrece puntos de enganche para servicios seguros.
El estándar no define niveles superiores ni subcapas de interoperabilidad. Existen extensiones, como la especificación de ZigBee, que complementan al estándar en la propuesta de soluciones completas.

2.2.4 ZigBee

2.2.4.1 Introducción

ZigBee [7] es el nombre de la especificación de un conjunto de protocolos de comunicación de alto nivel para su utilización con radios digitales de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal WPAN. Conlleva el objetivo de realizar las aplicaciones en las que es necesaria una comunicación segura con una baja tasa de envío de los datos y la información y a la vez, que la vida útil de las baterías sea lo más larga posible.
Es muy utilizado en las aplicaciones de domótica por sus tres características:

- Bajo consumo
- Topología en red de malla
- Fácil integración, lo que permite crear nodos con muy poca electrónica.

ZigBee utiliza la banda ISM (*Industrial, Scientific and Medical*). En concreto 868MHz para Europa y 915MHz para Estados Unidos, además de 2,4GHz para todo el mundo, para usos industriales o científicos, aunque generalmente se usará sobre todo la última de todas.

Los niveles Físico / Red / Aplicación del protocolo Zigbee son:

- Protocolos de enrutamiento.
- Mantenimiento de la topología.
- Descubrimiento de vecinos.
- Preservar la energía de los nodos (*sleep*).
- Tolerancia a fallos.
- Escalabilidad.
- Protocolos MAC.

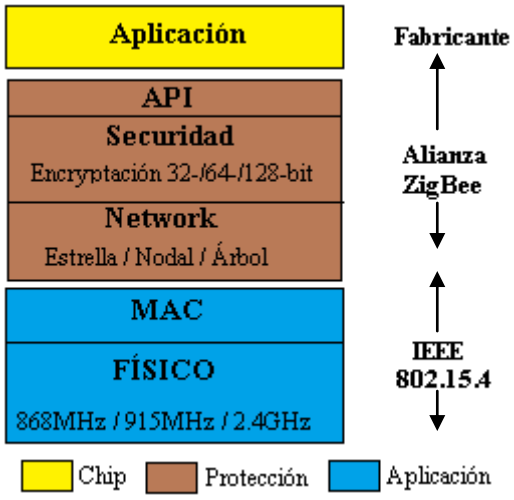


Figura 2. 7: Capas del protocolo Zigbee

2.2.4.2 ZigBee vs. Bluetooth

ZigBee es muy similar a Bluetooth pero con algunas diferencias [8]:
Una red ZigBee puede constar de un máximo de 64000 nodos, mientras que una red de Bluetooth sólo puede constar como máximo, de 8 nodos.

Menor consumo eléctrico que el ya de por sí bajo de Bluetooth. ZigBee tiene un consumo de 30 mA transmitiendo y de 3 uA en reposo, frente a los 40 mA transmitiendo y 0.2 mA en reposo que tiene el Bluetooth.

Según para qué se necesite se usa uno u otro. Se tiene en cuenta, por ejemplo, que para transmisiones de pocos datos con el Zigbee es suficiente ya que su velocidad es de hasta 250 kbps, mientras que si se necesitar enviar más cantidad de datos es necesario el, Bluetooth, puesto que puede llegar su versión más actual a una velocidad de hasta 3Mbps. Debido a las velocidades de cada uno, Bluetooth se usa para aplicaciones de telefonía móvil o infomática (transferencia de archivos multimedia), mientras que el ZigBee se suele utilizar en la domótica (transferencia de datos de poco tamaño).

2.3 Plataformas Hardware Comerciales

Los motes llevan integrados sensores para medir diferentes aspectos ambientales como la luz, la temperatura o la humedad. También utilizan la radio de forma intensiva para enviar y recibir datos y para procesamiento del microcontrolador. Son la base principal de las redes de sensores inalámbricas, puesto que cada mote se corresponde con cada uno de los nodos que forman dicha red.

Antes de explicar las diferentes características de los motes comerciales, es necesario hacer hincapié en una de las principales limitaciones que tienen los motes que es el consumo de energía. Por ello se ha desarrollado una estrategia que provoca un ahorro de energía permitiendo una mayor duración de los motes.

Dicha estrategia, radica en conseguir que, durante su funcionamiento, el mote evolucione a través de los siguientes estados:

1. ***Sleep***: como su nombre indica, el mote está en estado de hibernación. Es preciso mantener el mote la mayor parte del tiempo posible en este estado, con el fin de que el consumo sea mínimo y permitir así una mayor duración de las baterías.
2. ***Wakeup***: es el estado de cambio en el que el nodo se despierta y pasa a estado activo. Dicho cambio se produce cuando el sensor recibe algún cambio, estímulo o interrupción programada dentro de sus funciones de detección y análisis. Uno de los objetivos es minimizar este tiempo para pasar rápidamente al estado de trabajo.
3. ***Active***: es el estado activo del mote, donde realiza todo el proceso de adquisición y transmisión de datos. Por supuesto, este tiempo debe ser mínimo para volver cuanto antes al estado sleep, ya que el consumo será el mayor de los tres que se dan en cada fase.

En la figura 2.8 se muestra una gráfica que representa los estados de un nodo sensor:

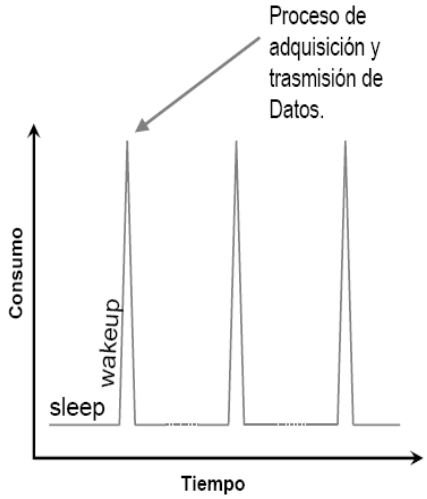


Figura 2. 8: Estados de un nodo sensor

En las siguientes tablas 2.1 y 2.2 se muestran la distribución del consumo de energía durante la recepción y transmisión de datos respectivamente:

Componentes	Desglose de trabajo durante la recepción de paquetes	Porcentaje del uso de la CPU	Energía consumida (nJ/bit)
AM	0.05%	0.02%	0.33
Packet	1.12%	0.51%	4.58
Radio Handler	26.87%	12.16%	182.38
Radio decode thread	5.48%	2.48%	37.2
RFM	66.48%	30.08%	451.17
Radio Reception	-	-	1350
Idle	-	54.75%	-
Total	100.00%	100.00%	2028.66

Tabla 2.1: Distribución del consumo de energía en la recepción de datos

Componentes	Desglose de trabajo durante la recepción de paquetes	Porcentaje del uso de la CPU	Energía consumida (nJ/bit)
AM	0.03%	0.01%	0.18
Packet	3.33%	1.59%	23.89
Radio Handler	35.32%	16.90%	253.55
Radio decode thread	4.53%	2.17%	32.52
RFM	56.80%	27.18%	407.17
Radio Reception	-	-	1800
Idle	-	52.14%	-
Total	100.00%	100.00%	4317.89

Tabla 2.2: Distribución del consumo de energía en la transmisión de datos

Existen distintas plataformas hardware en el mercado, algunas de ellas son las siguientes:

2.3.1 Intel Mote 2

Intel Mote surge de la colaboración entre los laboratorios de Intel en Berkeley [9], la Universidad de Berkeley y otros centros de investigación del resto de USA. Los motes de Intel son pequeños, autónomos, alimentados por medio de baterías y con capacidad de comunicación por radio, de forma que son capaces de compartir información con otros y organizarse automáticamente dentro de una red *AdHoc*.

Uno de los principales objetivos del grupo de trabajo de los Intel Motes es colaborar con la comunidad investigadora en la exploración de las potenciales aplicaciones de los Motes y de las redes de sensores. Con este objetivo en mente, se han diseñado los Motes con una total compatibilidad con el sistema operativo TinyOS.

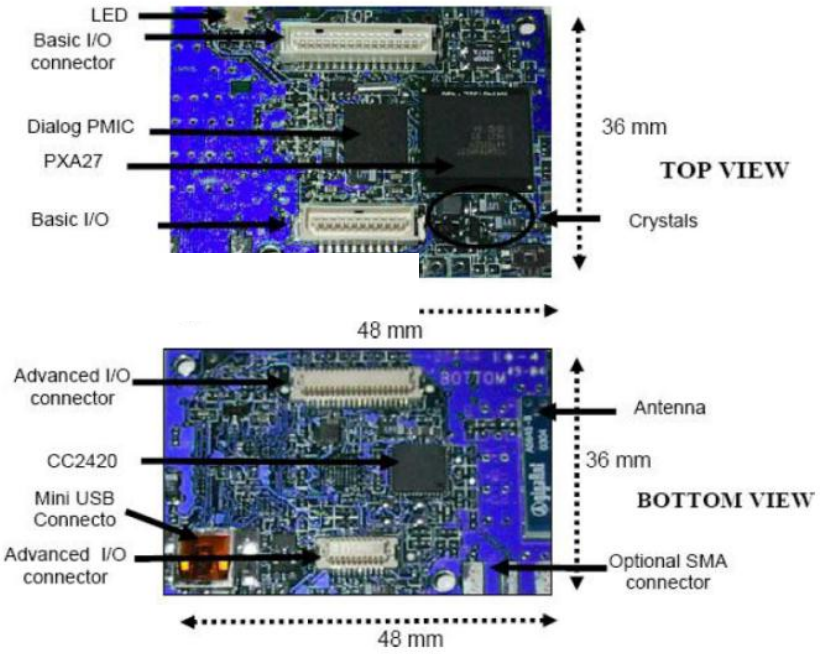


Figura 2. 9: Intel mote 2

Las investigaciones de mejora se están centrando en tres aspectos:

- El trabajo con consumos de energía ultra bajos.
- Integración del sistema.
- Reconfiguración hardware.

Intel Mote 2 es una plataforma avanzada para la creación de una red de sensores *wireless*. La plataforma está construida alrededor del procesador de bajo consumo XScale PXA27x con comunicación radio 802.15.4 sobre la placa principal y una antena de 2.4GHz. Contiene 2 interfaces de sensores básicos en uno de los lados de la placa central y otros 2 interfaces avanzados para sensores en el otro lado.

El Intel Mote 2 es una plataforma modular y escalable. La placa principal contiene el procesador y el módulo radio y se le pueden agregar diferentes módulos sensores, dependiendo de la aplicación final de la red de sensores. Dicho procesador, cuando se habilita en el modo bajo consumo, puede funcionar a bajo voltaje (0.85V) y una frecuencia también baja (13MHz), mientras que su valor máximo de operación es de 416MHz. También integra una memoria SRAM de 256KB dividida en 4 bancos de

64KB y diferentes opciones de E/S, lo que lo hace extremadamente flexible para soportar diferentes sensores A/D, opciones de radio. Estas E/S incluyen comunicación mediante I^2C, 3 puertos serie síncronos, 3 puertos UART, E/S digitales, un cliente USB. Además, el propio procesador incluye diversos temporizadores (*timers*) y un reloj en tiempo real.

El Intel Mote 2 ofrece diversas opciones para su alimentación: desde aprovechar los 5V que proporciona el puerto USB hasta el uso de baterías recargables.

Entre sus características destaca las siguientes:

- El conector SMA externo es opcional.
- Permite comunicación con dispositivos USB.
- Gran cantidad de E/S estándar, 3XUART, 2XSPI, I^2C, SDIO, GPIOs.
- Algunas de sus aplicaciones son: procesamiento de imágenes digitales y monitorización y análisis industrial.
- Es una plataforma avanzada con diseño modular.
- Dispone de conectores de interfaz en la parte superior y en ambos lados inferiores.

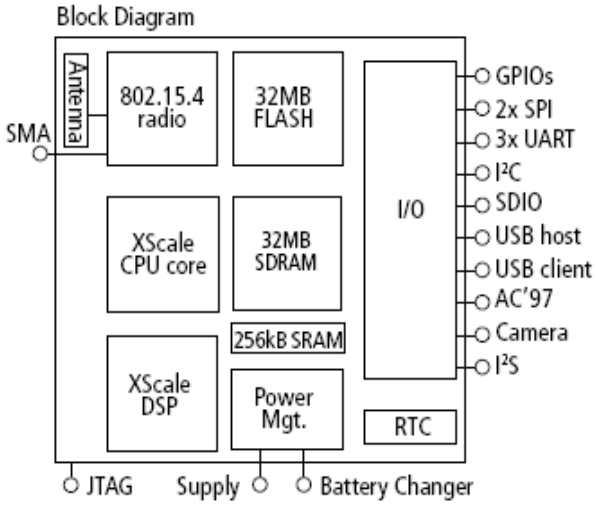


Figura 2.10: Diagrama de bloques de Imote2.

- Conectores Superiores: Proporcionan un juego estándar de señales de E/S básicas.
- Conectores Inferiores: Proporcionan interfaces adicionales de alta velocidad.
- El procesador tiene varios modos de funcionamiento en bajo consumo: "sleep" y "deep sleep."
- El procesador cuenta con numerosos temporizadores, entre ellos un reloj en tiempo real.
- Usa el transceptor de radio CC2420 IEEE 802.15.4.
- Velocidad de transmisión de datos a 250 Kb/s con 16 canales a 2.4 GHz.

Alimentación	
Batería de la placa	3x AAA
Voltaje del USB	5.0 V
Voltaje de la Batería	3.2 – 4.5 V

Cargador de batería Li-Ion	
Mecánica	
Dimensiones de la placa del Imote2	36mm x 48mm x 9mm
Peso	12g

Tabla 2.3: Otras características

-

Batería de la placa	IBB2400CA
Baterías	3x AAA
Máxima Corriente	500mA
Medidas	52mm x 43mm x 18mm
Peso con las 3 AAA baterías	51g
Peso sin las baterías	12g

Tabla 2.4: Características de la batería del Imote2

2.3.2 Familia de los MICAz

A continuación se explicará la familia de los mote MICAz, donde se expondrá sus características técnicas, así como los componentes que los forman:

2.3.2.1 MICAz

La plataforma de sensores MICAz [10] está comercializada por Crossbow. Son una de las últimas generaciones de motes que trabaja en la banda de frecuencias de 2400 MHz a 2483.5 MHz.

Figura 1.2: MICAz

En la figura 2.11 se muestra el diagrama de bloques del MICAz:

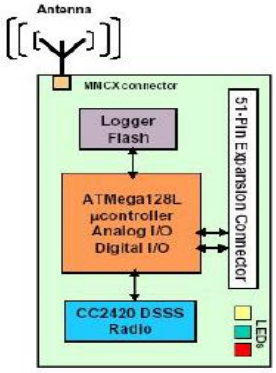


Figura 2.11: Diagrama de Bloques

La familia MICAz usa el Chipcon CC2420, que cumple con la normativa IEEE 802.15.4 y tiene un transceptor de radio frecuencia Zigbee integrado con un micro controlador Atmega 128L. Dispone de 51 pines de I/O y una memoria tipo flash.

Para la programación de estos motes se emplea la *Placa de desarrollo MIB510*. Esta placa actúa como interfaz entre el PC y los motes a través del puerto serie, permitiendo la programación de los dispositivos acoplados.

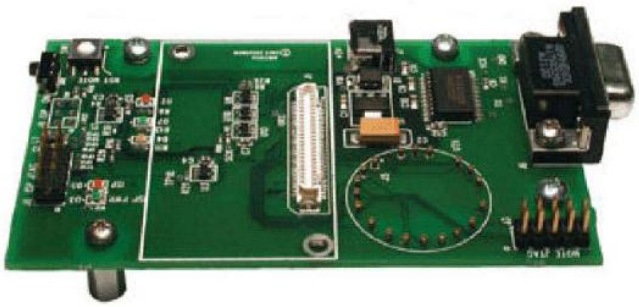


Figura 2.12: Placa de desarrollo MIB510

La placa MIB510 tiene un procesador (ISP, Procesador de la Señal de la Imagen) Atmega16L, mediante el cual se programan los motes. El código se descarga al ISP a través del puerto serie RS-232, y es el ISP el que programa el código en el mote. Esta placa tiene conectores para motes MICAz, MICA2 y MICA2DOT.

2.3.2.2 Mica2

Los motes Mica2 son los módulos de tercera generación de motes que se usan para redes de sensores inalámbricas de baja potencia. Mejoran las características del Mica original, de la siguiente forma:

- Diseñado específicamente para redes de sensores integradas.
- Distintas frecuencias de transmisión con amplio rango.
- Más de un año de batería mediante los modos *sleep*.
- Permiten reprogramación inalámbrica a distancia.
- Amplia variedad de placas de sensores compatibles: luz, temperatura, presión, aceleración, acústica, etc.

Figura 2.13: Mica2

Las distintas aplicaciones en las que se utilizan estos motes son, principalmente, las WSN, la seguridad y vigilancia, la monitorización ambiental o las redes inalámbricas de gran escala.

Dependiendo de su frecuencia de uso, los Mica2 se dividen en tres modelos: MPR400 (915MHz), MPR410 (433MHz), y MPR420 (315MHz). Estos motes usan el Chipcon CC1000, con radio modulada en FSK. Todos los modelos usan un potente microcontrolador Atmega128L y una radio de frecuencia sintonizable en un rango amplio.

El diagrama de bloques del mote Mica2 se muestra en la figura 2.14:

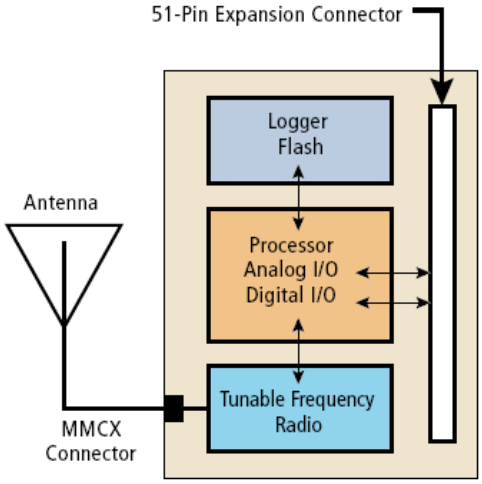


Figura 2.14: Diagrama de Bloques

A continuación se muestran algunas características más sobre las plataformas Mica2 MPR400CB**:**

Rendimiento del Procesador		
Programa Memoria Flash	128K bytes	
Medición (serial) Flash	512K bytes	>100,000 Medidas
Configuración EEPROM	4K bytes	
Comunicaciones Serial	UART	Transmisión de niveles 0-3V
Conversión Analógico-Digital	10 bit ADC	8 canales, entrada 0-3 V
Otras interfaces	DIO, I^2C, SPI	
Consumo de corriente	8mA	Modo Activo
	< 1uA	Modo Hibernación

Tabla 2.3: Características del Procesador.

Multi-Canal Radio		
Frequencia Central	868/916 MHZ	Bandas ISM (*Industrial, Scientific and Medical*)
Número de canales	4 / 50	Programable, específico según el país
Ratio de los datos	38.4 Kbaud	Codificador Manchester
Alimentación RF	-20 a +5 dBm	Programación típica
Sensibilidad de recepción	-98 dBm	Típica analogía RSSI para ch 0 AD
Rango al aire libre	500 ft	¼ onda dipolar, línea de visión
Consumo de corriente	27 mA	Transmisión con alimentación máxima
	10 mA	Recibiendo
	< 1uA	Hibernación

Tabla 2.4: Características de Canal de Radio.

Electromecánico		
Batería	2x AA baterías	Pack adjunto
Alimentación externa	2.7 – 3.3 V	Provisto de conector
Interfaz usuario	3 LEDs	Programable por el usuario
Tamaño (in)	2.25 x 1.25 x 0.25	Excluido el pack de la batería
(mm)	58 x 32 x 7	Excluido el pack de la batería
Peso (oz)	0.7	Excluida las baterías
(gramos)	18	Excluidas las baterías
Conector de expansión	51 pins	Para todas las principales señales I/O

Tabla 2.5: Otras características.

Para la programación de estos motes se emplea, al igual que con las plataformas MICAz, la placa de desarrollo MIB510 y también la MIB520.

2.3.2.3 Mica2Dot

Los Mica2Dot son un tipo de motes diseñados especialmente para aplicaciones donde el tamaño físico es fundamental. Al igual que los Mica2 hay tres modelos dependiendo de su frecuencia: MPR500 (915MHz), MPR510 (433MHz), y MPR520 (315MHz). El resto de características son similares a las de los Mica2, lo más relevante es la forma física y el reducido tamaño que poseen, tal y como podemos ver en la figura 2.15:

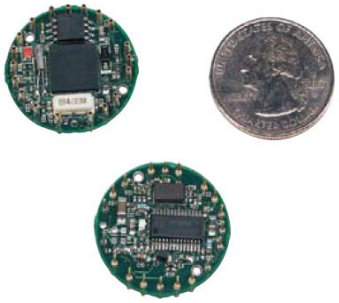


Figura 2.15: Mica2Dot

Está basado en el microcontrolador Atmel ATmega128L de baja potencia y opera con TinyOS desde su memoria flash interna. Tiene 18 pines de expansión para conectar 6 entradas analógicas, I/O digital y una interfaz de comunicaciones serie o UART.

Sus principales características son:

- Transmisión multicanal 868 / 916 MHz, 433 MHz o 315 MHz de rango extendido.
- Permite programación remota inalámbrica.
- Compatible con el mote Mica2.
- Dispone de un sensor de temperatura, monitor de la batería y LED.

Procesador/Radio de la placa	**MPR500CA**	**MPR510CA**	**MPR520CA**
Rendimiento del Procesador			

Programa de la Memoria Flash	128K bytes	128K bytes	128K bytes
Medida (Serial) Flash	512K bytes	512K bytes	512K bytes
Configuración EEPROM	4K bytes	4K bytes	4K bytes
Comunicación Serial	UART	UART	UART
Conversión analógica a digital	10 bit ADC	10 bit ADC	10 bit ADC
Otras interfaces	DIO	DIO	DIO
Consumo de corriente	8mA	8mA	8mA
	< 15 uA	< 15 uA	< 15 uA

Tabla2.6: Propiedades de los distintos tipos de procesadores.

El diagrama de bloques de este dispositivo se muestra en la figura 2.16:

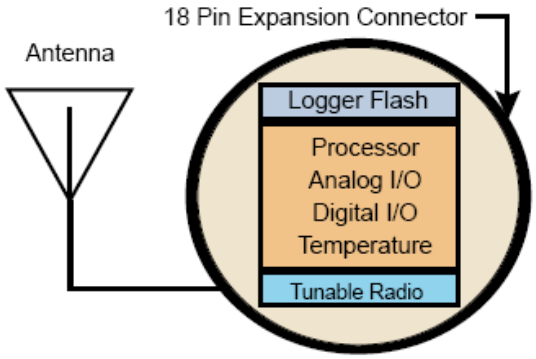


Figura 2.16: Diagrama de Bloques.

A continuación, algunas características técnicas sobre dichos motes:

	MPR500CA	**MPR510CA**	**MPR520CA**	
Multi-Canal Radio	SÍ	SÍ	SÍ	
Frequencia Central	868/916 MHZ	433 MHZ	315MHZ	Bandas ISM (*Industrial, Scientific and Medical*)
Número de canales	4 / 50	4	5	Programable, específico según el país
Ratio de los datos	38.4 Kbaud	38.4 Kbaud	38.4 Kbaud	Codificador Manchester
Alimentación RF	-20 a +5 dBm	-20 a +5 dBm	-20 a +5 dBm	Programación típica
Sensibilidad de recepción	-98 dBm	-101 dBm	-101 dBm	Típica analogía RSSI para ch 0 AD
Rango al aire libre	500 ft	1000 ft	1000 ft	¼ onda dipolar, línea de visión
Consumo de corriente	27 mA	25 mA	25 mA	Transmisión con alimentación máxima
	10 mA	8 mA	8 mA	Recibiendo
	< 1uA	< 1uA	< 1uA	Hibernación

Tabla 2.7: Propiedades del canal de radio en los distintos tipos

	MPR500CA	**MPR510CA**	**MPR520CA**	
Electromecánico				

Batería	3V Botón pila	3V Botón pila	3V Botón pila	
Alimentación externa	2.7 – 3.3 V	2.7 – 3.3 V	2.7 – 3.3 V	Provisto de conector
Interfaz usuario	1 LED	1 LED	1 LED	Programable por el usuario
Tamaño (in)	1.0 x 0.25	1.0 x 0.25	1.0 x 0.25	Excluido el pack de la batería
(mm)	25 x 6	25 x 6	25 x 6	Excluido el pack de la batería
Peso (oz)	0.11	0.11	0.11	Excluida las baterías
(gramos)	3	3	3	Excluidas las baterías
Conector de expansión	18 pins	18 pins	18 pins	Para todas las principales señales I/O

Tabla 2.8: Otras características.

2.3.3 Tmote Sky

Tmote Sky es una plataforma también comercializada por Crossbow, la cual se usa para aplicaciones en redes de sensores de muy bajo consumo y alta recopilación de datos. Algunas de sus características son:

- Lleva integrados tanto los sensores como la radio, antena y microcontrolador. Los coeficientes para estos sensores se guardan en la EEPROM. La precisión del sensor es de 0.5º C.
- Cuenta con dos sensores, el primero censa la radiación fotosintéticamente activa, el segundo censa la radiación solar total.
- Fácilmente programable.
- Las operaciones son realizadas gracias al *microcontrolador MSP430* [12] *F1611*, de 8MHz (10Kb de RAM, 48Kb de Flash). Este procesador RISC de 16 bits consume muy poca batería, pero para reducir al máximo este consumo permanece en estado sleep durante la mayoría del tiempo, se despierta para la adquisición, procesado y transmisión de datos y vuele a estado sleep.
- Interactúa con otros dispositivos IEEE 802.15.4.
- Utiliza un controlador USP de FTDI para comunicarse con el ordenador y maneja una radio de *Chipcon CC2420* de 250Kbps y 2.4GHz, la cual es un estándar IEEE 802.15.4 que provee una fiable comunicación inalámbrica para WSNs.
- La radio tiene la posibilidad de enviar datos a muy alta frecuencia. El microcontrolador se comunica con la antena a través del puerto SPI. La antena interna *"Invertid-F micro strip*" es una antena seudo-omnidireccional que puede alcanzar los 50 metros dentro de un edificio y hasta los 125 en el exterior (debido a la ausencia de paredes).

En la siguiente figura 2.17 se puede observar los distintos componentes que forma el TmoteSky:

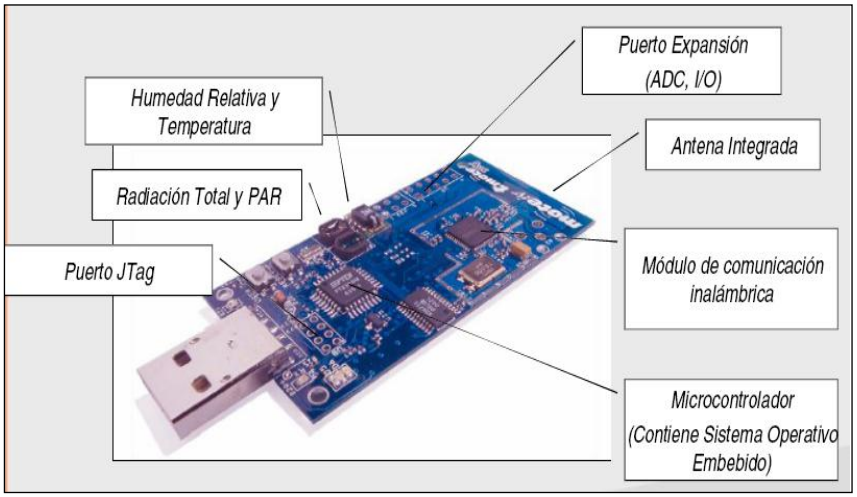


Figura 2.17: Tmote Sky.

En resumen, las características generales del Tmote Sky son las siguientes:

- Dispone de una controladora DMA integrada.
- Muy bajo consumo.
- Rápido en salir del estado *sleep*.
- Hardware para encriptación y autenticación de la capa de enlace.
- Programación y recogida de datos por USB.
- Ayuda de TinyOS: enrutamiento de malla e implementación de las comunicaciones.

2.3.3.1 Telosb

Los motes Telosb [11] (TPR2400) reúnen todo lo esencial para estudios de laboratorio usando una plataforma simple, incluyendo la capacidad de programación por USB, una antena integrada con un sistema de radio IEEE 802.15.4, un procesador de bajo consumo con una memoria extendida y un conjunto de sensores.

Figura 2.18: Telosb

Las características generales de los Telosb son:

- Transmisión RF de acuerdo con la norma IEEE 802.15.4/ZigBee.
- Banda de frecuencias desde 2.4 a 2.4835 GHz.

- Velocidad de transferencia de datos de 250 kbps.
- Antena integrada.
- Micro-controlador MSP430 a 8MHz con 10kB de RAM.
- Bajo consumo.
- Memoria flash externa, de 1 Mb para almacenamiento de datos.
- Programación y toma de datos vía USB.
- Conjunto de sensores de luz, temperatura y humedad.
- Soporta TinyOS para implementación y comunicación de redes.

Su correspondiente diagrama de bloques lo podemos ver a continuación (Figura 2.19):

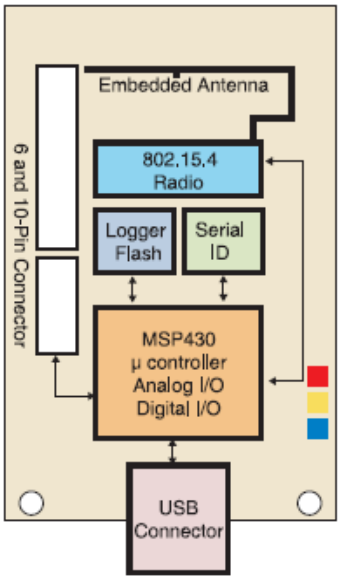


Figura 2.19: Diagrama de Bloques

Esta plataforma consigue un bajo consumo de potencia permitiendo una larga vida a las baterías además de tener un tiempo mínimo en el estado de *wakeup*, otro de los objetivos dentro de las estrategias de bajo consumo.

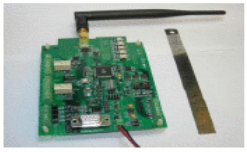
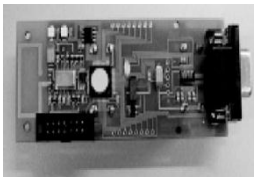
Los Telosb se alimentan de 2 baterías dipo AA, aunque si es conectado mediante el puerto USB para programación o comunicación, la alimentación la proporciona el ordenador.

También se proporciona la capacidad de añadir dispositivos adicionales. Los dos conectores de expansión de los que dispone pueden ser configurados para controlar sensores analógicos, periféricos digitales y displays LCD.

Con todas estas características, el mote Telosb no sólo proporciona más facilidad para programación, más flexibilidad y más presentación, sino que es el que menos consumo de potencia ofrece, permitiendo alargar la vida de los nodos considerablemente y siendo ésta su principal baza frente a los otros dispositivos.

2.3.4 Otros motes comerciales

Node	Picture	CPU	Memory	I/O Sensors	Radio
CSIRO Fleck		Atmega128L, 8MHz	512K external memory	Temperature, Light, Screw terminal for 4X digital i/o and 2X analog	Nordic 903

NICTOR				2 Serial (RS-232) Ports, 4 Digital Inputs, 2 Digital Outputs, 2 AnalogInputs	2.4GHz
BTnode		Atmel ATmega128L(AVR RISC 8 MHz @ 8 MIPS)	64+180 Kbyte SRAM, 128 Kbyte Flash ROM, 4 Kbyte EEPROM	UART, SPI, I2C, GPIO, ADC, Clock, Timer, LEDs Standard Molex 1.25mm Wire-to-Board and Hirose DF17 Board-to-Board connectors	Chipcon CC1000 operating in ISM Band 433-915 MHz)
EYES		MSP 430F149 (5 MHz @ 16 Bit)	60 Kbytes of program memory , 2 Kbytes of data memory, 4 Kbyte EEPROM	UART, AD and I/O, JTAG interface and sensor board with compass, accelerometer, temperature sensor, light sensor, pressure sensor, microphone nad push button lines	RFM TR1001 hybrid radio transceiver
Rockwell Wins-Hidra Nodes	 	SrongARM 1100 (133MHz)	4MB Flash 1MB SRAM	seismic(geophone), acoustic, magnetometer, accelerometer, temperature and pressure	Connexant's RDSSS9M (100Kbps)
Sensoria WINS NG 2.0		SH-4 processor (167 Mhz)		GPS and imaging	Dual 2.4 Ghz FH

Sensoria WINS 3.0		Intel PXA255 (scalable from 100 to 400 MHz)	64MB SDRAM 32MB Flash	GPS, USB(2 host ports, 1 device port), RS-232 serial(5 generic + 1 Linux console),Audio in/out(1 stereo input, GPS, USB(2 host ports, 1 device port),PCMCIA/CardBus(1 external slot),	Dual embedded 802.11b modules

Tabla 2.9: Otros nodos comerciales

2.4 Sistemas operativos para MOTES

Para la programación de motes se dispone de diversos sistemas operativos de los cuales aquí se enumeran los más conocidos:

- **Bertha** (*pushpin computing platform*) Una plataforma de software diseñada e implementada para modelar, testear y desplegar una red de sensores distribuida de muchos nodos idénticos.
- **Nut/OS**: Es un pequeño sistema operativo para aplicaciones en tiempo real, que trabaja con CPUs de 8 bits.
- **Contiki**: Es un Sistema Operativo de libre distribución para usar en un limitado tipo de computadoras, desde los 8 bits a sistemas embebidos en microcontroladores, incluidas motas de redes inalámbricas.
- **CORMOS**: A *Communication Oriented Runtime System for Sensor Networks*, específico para redes de sensores inalámbricos como su nombre indica.
- **eCos**: (*embedded Configurable operating system*) es un sistema operativo gratuito, en tiempo real, diseñado para aplicaciones y sistemas embebidos que sólo necesitan un proceso. Se pueden configurar muchas opciones y puede ser personalizado para cumplir cualquier requisito, ofreciendo la mejor ejecución en tiempo real y minimizando las necesidades de hardware.
- **EYESOS**: se define como un entorno para escritorio basado en Web, permite monitorizar y acceder a un sistema remoto mediante un sencillo buscador.
- **MagnetOS:** es un sistema operativo distribuido para redes de sensores o adhoc, cuyo objetivo es ejecutar aplicaciones de red que requieran bajo consumo de energía, adaptativas y fáciles de implementar.
- **MANTIS** (*MultimodAl NeTworks In-situ Sensors*).
- **TinyOS:** Sistema Operativo desarrollado por la universidad de Berkeley, y es el que se ha elegido para realizar este proyecto. Más adelante se explicará con más detalle.
- **t-Kernel**: es un sistema operativo que acepta las aplicaciones como imágenes de ejecutables en instrucciones básicas. Por ello, no importará si está escrito en C++ o lenguaje ensamblador.
- **LiteOS**: Sistema operativo desarrollado en principio para calculadoras, pero que ha sido también utilizado para redes de sensores.
- **Protothreads**: específicamente diseñado para la programación concurrente, provee hilos de dos bytes como base de funcionamiento.
- **SNACK**: facilita el diseño de componentes para redes de sensores inalámbricas, sobre todo cuando la información o cálculo a manejar es muy voluminoso,

complicado con nesc, este lenguaje hace su programación más eficiente. Luego es un buen sustituto de nesc para crear librerías de alto nivel a combinar con las aplicaciones más eficientes.
- **C@t**: iniciales que hincan computación en un punto del espacio en el tiempo (*Computation at a point in space (@) Time*).
- **DCL:** Lenguaje de composición distribuido (*Distributed Compositional Language*).
- **GalsC:** diseñado para ser usado en TinyGALS, es un lenguaje programado mediante el modelo orientado a tarea, fácil de depurar, permite concurrencia y es compatible con los módulos nesc de TinyOS.
- **SQTL** (*Sensor Query and Tasking Language*): como su nombre indica es una interesante herramienta para realizar consultas sobre redes de motes.

A continuación, se ofrece información adicional sobre los que se han considerado más adecuados dentro del contexto del proyecto:

2.4.1. Contiki

Es un sistema operativo de código abierto [13], multitarea, portable, desarrollado para su uso en ordenadores que van desde los 8 bits, a sistemas integrados sobre microcontroladores, incluidos nodos de redes de sensores. El nombre de Contiki, proviene de la famosa balsa empleada por el explorador y escritor noruego Thor Heyerdahl's en su viaje a través del océano Pacífico.

A pesar de la multitarea incorporada y la pila TCP/IP incluida, Contiki sólo necesita unos pocos kilobytes de código y unos cientos de bytes de RAM. Un sistema completo con una interfaz gráfica de usuario (GUI) necesita alrededor de 30 kB de memoria RAM.

El núcleo básico y casi todas sus funciones fueron desarrollados por Adam Dunkels, perteneciente al Swedish Institute of Computer Science.

Contiki está desarrollado para sistemas embebidos con escasa memoria. Una configuración típica en Contiki requiere para su funcionamiento 2 kB de RAM y 40 kB de ROM.

Contiki está compuesto por un núcleo, el cual impulsa eventos y hace uso de protohilos, sobre el cual los programas son cargados y descargados dinámicamente. Los procesos de Contiki, usan *protothreads*, se trata de mecanismos de poco consumo que proveen y permiten una programación lineal o de tipo hilo.

Cabe destacar que al necesitar poco recursos para su funcionamiento, Contiki se ejecuta en una gran variedad de plataformas que van desde los microcontralodores integrados como el MSP430 y la AVR, hasta en viejos equipos domésticos.

Contiki usa la comunicación de tipo paso de mensajes a través de eventos, así como una interfaz gráfica opcional, ya sea directamente del subsistema gráfico de apoyo a nivel local o de los terminales conectados con red virtual de VNC o Telnet.

Una instalación completa de Contiki incluye las siguientes características:

- Kernel multitarea.
- Multitarea preferente opcional por aplicación.
- Protohilos de ejecución.

- Conectividad TCP/IP de red, incluyendo IPv6.
- Sistema de ventanas y GUI.
- Red de visualización remota utilizando Visual Network Computing.
- Un navegador web (el más pequeño del mundo).
- Incluye servidor web personal.
- Cliente telnet sencillo.
- Salvapantallas.

2.4.2. PalOS

Este sistema operativo ha sido desarrollado por la UCLA (Universidad de California). Durante la ejecución del programa, cada tarea registra una tarea de eventos en la programación del sistema. Si la tarea 1 desea comunicarse con la tarea 2, esta realiza una petición (*post*) de un evento en la cola de eventos de la tarea 2, usando una funcionalidad del *Scheduler* (organizador o programador) del sistema, para que luego la tarea 2 reciba ese evento al preguntar al *Scheduler* si tiene algún evento para él. Es imprescindible el uso de un "timer" para que maneje la periodicidad con que una tarea registra eventos, esto es recomendable para un correcto funcionamiento de esta estructura de software.

Este "timer" posee tres colas:

1. Cola Nexo, encargada de interactuar con las demás tareas (recibe el envío de otras tareas).

2. Cola Delta, en la cual se ordenan los distintos eventos dependiendo del tiempo de expiración.

3. Eventos Expirados, donde se van colocando para su posterior ejecución.

2.4.3 TinyOS

El sistema operativo utilizado en este proyecto es TinyOS. El diseño de TinyOS está basado en responder a las características y necesidades de las redes de sensores, tales como reducido tamaño de memoria, bajo consumo de energía, operaciones de concurrencia intensiva (simultaneidad en la ejecución de múltiples tareas interactivas). Además se encuentra optimizado en términos de uso de memoria y eficiencia de energía.

El diseño del Kernel (núcleo) de TinyOS está basado en una estructura de dos niveles de planificación.

- Eventos: Pensados para realizar un proceso pequeño (por ejemplo cuando el contador del timer se interrumpe, o atender las interrupciones de un conversor análogo-digital). Además pueden interrumpir las tareas que se están ejecutando.

- Tareas: Las tareas están pensadas para hacer una cantidad mayor de procesamiento y no son críticas en tiempo. Las tareas se ejecutan en su totalidad, pero la solicitud de iniciar una tarea, y el término de ella son funciones separadas.

Con este diseño permitimos que los eventos (que son rápidamente ejecutables), puedan ser realizados inmediatamente, pudiendo interrumpir a las tareas (que tienen mayor carga computacional en comparación a los eventos).

TinyOS y NesC, que es el lenguaje de programación utilizado por los motes en el presente proyecto, se encuentran profundamente relacionados. Es por eso, que, a continuación, se desarrollará y se explicará las características fundamentales de este lenguaje.

2.5 Sistema Operativo TinyOS

2.5.1 Descripción

Las redes de sensores inalámbricas están compuestas por nodos de pequeño tamaño y que poseen limitaciones tanto de memoria como de procesador. Además sufren el inconveniente del consumo de potencia, lo que supone un problema en cuanto a la vida útil de las mismas. Es por ello por lo que es imprescindible mejorar tanto el hardware como el software para lidiar estas limitaciones e inconvenientes.

De ahí nace TinyOS [14] (*Tiny Micro-Threading Operating System*), un sistema operativo que fue desarrollado por la Universidad de Berkeley, cuya característica principal reside en que al ser modular resulta ideal para instalarse en sistemas con restricciones de memoria, como es el caso de los motes. Es por ello que TinyOS se ha convertido en la base de la programación de los mismos.

2.5.2 Características de TinyOS

- Es un sistema operativo dirigido por eventos.
- Su arquitectura está basada en componentes, ofreciendo un marco para el desarrollo de aplicaciones orientadas a componentes.
- Su lenguaje de programación utilizado, nesC, es una extensión de C que permite la definición de componentes y la interconexión de los mismos. Este lenguaje no tiene gestión de procesos, en su lugar tiene dos hilos de ejecución de eventos (tareas y eventos).
- Se trata de un sistema operativo liviano y puede ser ejecutado en dispositivos con recursos muy limitados ya que, tanto el espacio que ocupa en memoria flash, como la RAM que consume son mínimos.
- Gestiona la potencia consumida, cuya función es la de desactivar los recursos hardware que no están siendo usados en ese momento. Permite un modo de bajo consumo de potencia, mientras no se estén transmitiendo o recibiendo datos, del mismo modo se crean estados de actividad o latencia y periodos en los que periódicamente la radio se apaga. Además gestiona los tiempos de paso de un estado a otro.
- Permite comunicación *broadcast* y encaminamiento *multi-hop* entre otros. En el caso del encaminamiento, la aplicación puede elegir el algoritmo que desee, pero dicho algoritmo debe cumplir una arquitectura de componentes que define el

sistema operativo. Esto se lleva a cabo para que las aplicaciones puedan cambiar fácilmente el algoritmo que usan por otro.
- Permite la programación en red.
- Usa componentes para el cifrado de datos.
- Puede ser instalado tanto en plataformas Linux como Windows.

2.5.3 Dinámica general de TinyOS.

Las dos fuentes de concurrencia en TinyOS son las **tareas** y los **eventos**. Los programas están dirigidos por tareas, y estas pueden ser interrumpidas por eventos.

En la siguiente figura 2.20 se muestra el modo de funcionamiento de TinyOS. Se emplean los términos **tareas**, **eventos** y **comandos**, a continuación se explica cada uno de ellos.

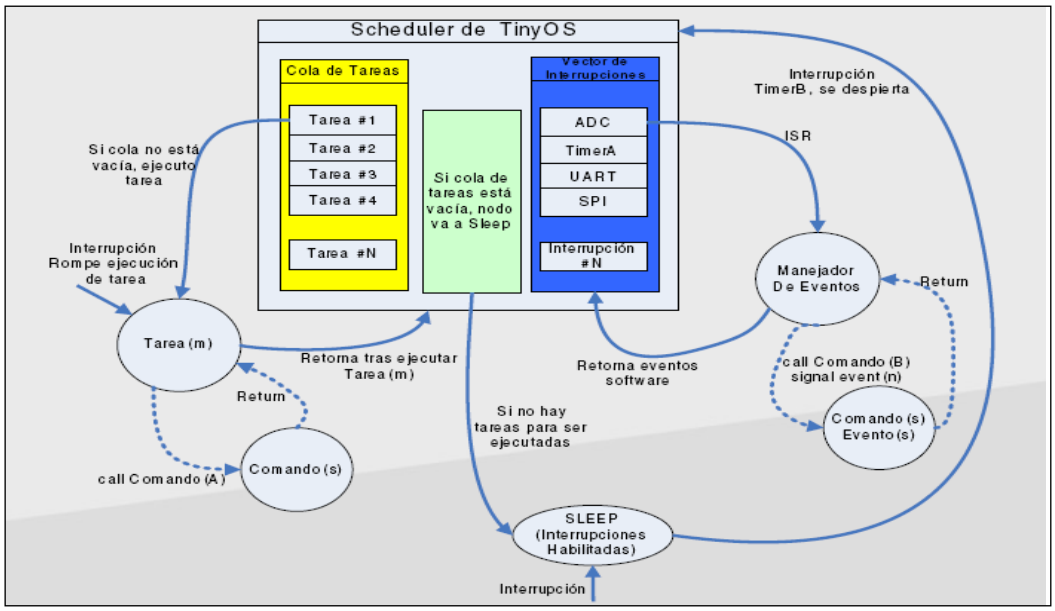


Figura 2.20: Modo de funcionamiento de TinyOS.

- Los **comandos** son llamadas a componentes de capas inferiores. Al llamar a un comando, éste dentro de su componente llama a otros componentes.
- Los **eventos** son llamadas a componentes de capas superiores. Un componente avisa a otro de más alto nivel, de que se ha producido una interrupción. Los eventos tiene mayor prioridad que cualquier tarea, así si sucede un evento mientras se está ejecutando una tarea, se ejecutaría el evento y dejaría de ejecutarse dicha tarea.
- Las **tareas** son fracciones de código que se ejecutan de forma asíncrona, siempre y cuando la CPU no tenga que ejecutar ningún evento, ya que, como se ha comentando en el párrafo anterior, pueden ser interrumpidas por eventos. Las tareas se van almacenando en una cola de tipo FIFO. Si la cola está vacía, el procesador entrará en estado "*sleep*" hasta que un nuevo evento haga despertar al procesador.

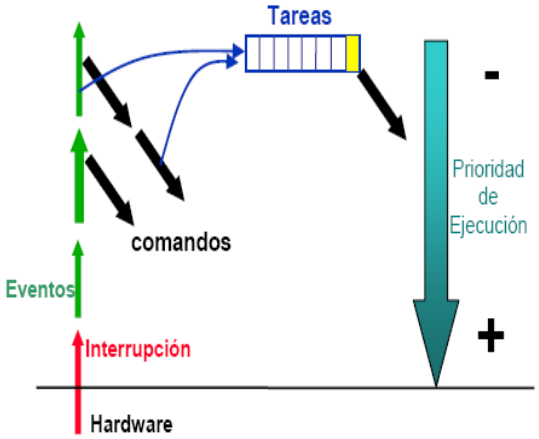


Figura 2.21: Prioridades de ejecución.

En conclusión, en TinyOS el código se ejecuta de dos formas: **Asíncrona** (en respuesta a interrupciones) y **Síncrona** (para la ejecución de tareas), esto quiere decir que las tareas se ejecutan en su totalidad pero no tienen prioridad sobre otras tareas o eventos. En cambio, los eventos disponen de una prioridad mayor para ser ejecutados y pueden interrumpir otros eventos o tareas, con el objetivo de cumplir de la mejor forma los requerimientos de tiempo real. Se recomiendan que las tareas sean cortas y, en el caso de necesitar un procesamiento mayor, dividirlo en múltiples tareas.

2.5.4 Inconvenientes de TinyOS.

- Ya que sólo existe un hilo para la ejecución de las tareas y el planificador de estas es una cola FIFO, no es posible tener garantías de tiempo real ni se ofrece gestión de prioridades. Lo único que TinyOS hace al respecto, es permitir que los nuevos eventos desalojen a las tareas que puedan ejecutarse.
- No dispone de mecanismos fiables de sincronización de datos entre nodos ni protección de memoria. Sin embargo esto lleva consigo que requiera menos recursos para su ejecución.
- No proporciona conectividad con grandes infraestructuras de servicios como Internet, pero es posible el uso de aplicaciones que sirvan de puente entre las redes de sensores y otras redes como las basadas en TCP/IP o IPv6.
- Es posible la reprogramación en red, no obstante se debe reprogramar todo el código que se ejecuta en el mote.

2.5.5 Lenguaje NesC

Se trata de un lenguaje de programación [15] que presenta grandes ventajas para el desarrollo de aplicaciones en sistemas de pocos recursos, específicamente en las redes de sensores inalámbricas, por lo que está en constante desarrollo.

Como se comentó anteriormente en el capítulo 2, NesC es una extensión del lenguaje de programación C. Se diseñó para plasmar los conceptos de estructuración y modelos de ejecución de TinyOS. Está diseñado bajo la expectación de que el código será generado por compiladores de programas completos.

A continuación se indican algunos conceptos necesarios para entender la mecánica de NesC y TinyOS.

- Los programas son realizados mediante **componentes**, que son "conectados" a través de **interfaces**, para formar el programa completo.
- Los archivos en NesC se clasifican en tres tipos: Interfaces, módulos y configuraciones.

Las **interfaces** desempeñan un papel muy importante, ya que son utilizadas para llevar a cabo gran variedad de acciones. De las interfaces cabe destacar que:

- Son el único punto de acceso al componente. Son bidireccionales.
- Los hilos de control pueden pasar a un componente a través de ellas. Estos hilos pueden ser de ejecución de tareas o de interrupciones hardware.
- A los componentes, se les especifica el comportamiento mediante las interfaces.
- Pueden ser "proporcionadas" o "usadas" por los componentes. Las interfaces proporcionadas, son para representar la funcionalidad que el componente proporciona al usuario y las usadas, para representar la funcionalidad que el componente necesita para llevar a cabo su trabajo.
- Disponen de un conjunto de funciones para ser implementadas por el proveedor de la interfaz (los métodos) y de otro conjunto para ser implementadas por el usuario de la interfaz (los eventos). Esto permite que una interfaz simple represente una interacción compleja entre componentes.
- Para llamar a los métodos de una interfaz, se deben implementar los eventos de esa interfaz.

2.5.5.1 Componentes

Las aplicaciones realizadas en TinyOS son construidas mediante componentes unidos para formar un ejecutable, con lo cual son unos de los elementos más importantes. Por ello, a partir de lo comentado en el apartado anterior se hará hincapié en los distintos tipos existentes y sus estructuras.

Existe dos tipos de componentes en NesC: los módulos y las configuraciones.

- Los módulos son los que proporcionan el código de la aplicación, implementando una o más interfaces.
- Las configuraciones se usan para ensamblar los componentes unos con otros, conectando las interfaces que usan unos componentes a las interfaces que les proporcionan otros. Esto es lo que se llama "*wiring*" (conexionado). Cada aplicación NesC se describe por una configuración de alto nivel que conecta todos los componentes que contiene.

Cada componente está formado por:

- Manejador de comandos.
- Manejador de eventos.
- Un "frame" (bloque que proporciona el contexto en el cual se encuentra el programa y se almacenan las variables) de tamaño fijo y estáticamente asignado, en el cual se representa el estado interno del componente.
- Un bloque con tareas simples.

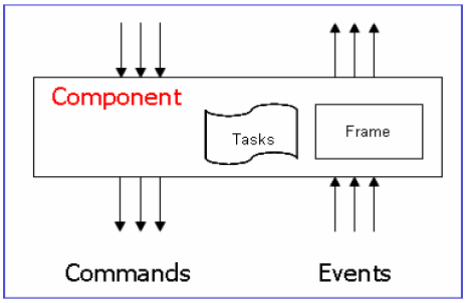


Figura 2.22: Estructura de un componente.

Una aplicación como mínimo va a estar formada por los archivos mostrados en la figura 2.23:

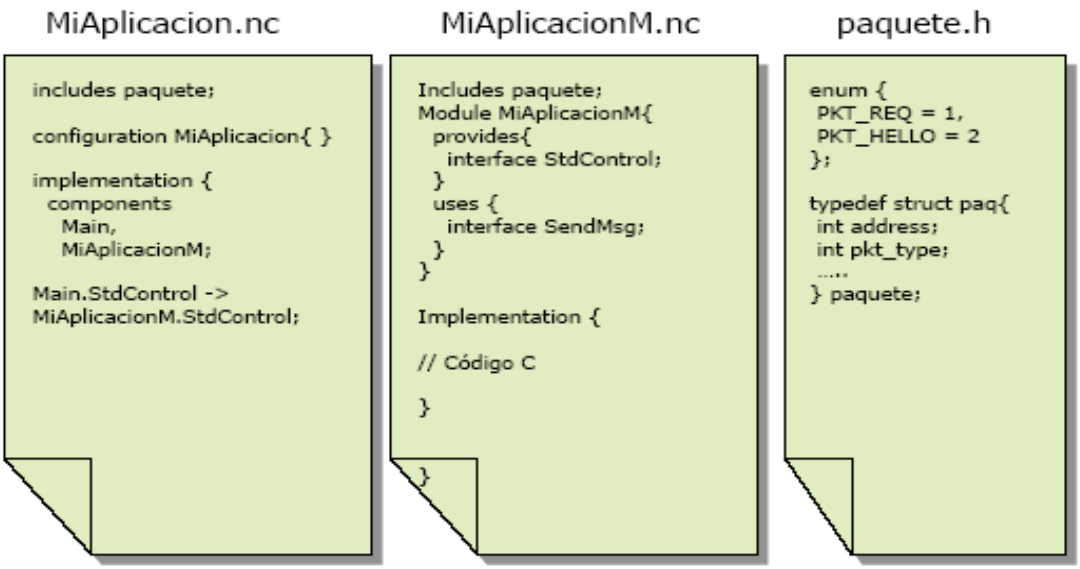


Figura 2.23: Ficheros mínimos de una aplicación.

- Fichero "MiAplicacion.nc"

  Se trata del fichero de configuración del componente, en el cual su código se distingue los siguientes dos apartados:

  - *Configuración*. En general estará vacía, sólo contendrá algo si se pretende crear un componente, no mediante su implementación de código directa (en Module), sino ensamblando otros componentes ya creados.
  - *Implementación*. Define las conexiones que hay entre los diferentes componentes que utilizan la aplicación.

- Fichero "MiAplicaciónM.nc"

Es el módulo del componente, en el cuál se implementa su comportamiento. En él se definen las interfaces que se proporcionan y que se usan y, por supuesto, contiene la implementación de la aplicación en sí.

Además, pueden existir más ficheros para una misma aplicación. Esto puede ocurrir si se incluyen librerías en las que se definen diferentes estructuras necesarias para la aplicación. Un ejemplo de esto es el archivo *paquete.h* de la figura 2.23, anteriormente expuesta.

Algunos ejemplos de componentes básicos son:

- Main → Es el componente principal en una aplicación TinyOS.
- Temporización → Cabe destacar dos componentes: SingleTimerb (implementa un solo interfaz TimerC) y TimerC (implementa múltiples timers que usan la identificación del timer como parámetro).
- Leds→ *LedsC*: Componente realiza el encendido y apagado de los LEDs del mote.
- Comunicaciones Radio→ *GenericComm.sendMsg* es para el envío de mensajes por radio y *GenericComm.receiveMsg* para recibir mensajes por radio.
- Sensores→Telos: *HamamatsuC.TSR, HamamatsuC.PAR, HumidityC.Humidity, HumidityC.Temperature.*
  Para poder obtener y modificar la potencia de transmisión de radio.

  CC2420RadioC.CC2420Control para motes Telosb.

  CC1000RadioC.CC1000Control para motes mica2.

- Almacenamiento de datos en flash→ Componente *Logger*.
- Reprogramación en red→ *Xnp* y *Deluge*.
- Criptografía→ Subsistema *TinySec*.

2.5.6 Herramientas de TinyOS.

A continuación se muestra una serie de herramientas que proporciona TinyOS para facilitar su correcta programación y utilización:

- **Listen**: Esta herramienta cumple la función de imprimir los datos en bruto de cada uno de los paquetes recibidos por el puerto serie.
- **Serial Forwarder**: Es una aplicación java que permite recibir paquetes por el puerto serie del PC y reenviarlos por otros puertos del mismo. Esto hace posible que otros programas puedan comunicarse con la red de sensores, a través de un nodo conectado al PC, que hace de pasarela.
- **NesDot**: Permite generar documentación automáticamente a partir del código fuente de un programa. Para cada fichero fuente, genera un fichero HTML con un gráfico que describe el conexionado de los componentes del fichero a través de sus interfaces y una descripción textual sacada de los comentarios del fichero fuente.
- **Surge View:** Esta aplicación java viene incluida en los paquetes de instalación de TinyOS. Permite la monitorización de una red y analizar su funcionamiento. Sus características incluyen exploración y configuración automática de la red,

visionado de su topología, almacenamiento y visualización de estadísticas de la red como rendimiento, calidad de los enlaces, etc. y una herramienta gráfica para el visionado de datos.

- **Motelist**: Proporciona el puerto COM para la comunicación entre el PC y el mote conectado a él.
- **TinyViz**: Es un simulador de eventos discretos para TinyOS. TOSSIM (TinyOS SIMulator), permite una simulación bastante completa de una red. Entre sus posibilidades se encuentran, simular las transmisiones de datos a nivel a de bit, lectura de los sensores…
- **TinyDB**: Es un sistema de procesado de consultas. Se encarga de extraer información de una red de sensores, transformando la red en una tabla de una base de datos distribuida. El lenguaje de programación utilizado es TinySQL, es una extensión del lenguaje de consultas SQL.
- **Bombilla/ Maté**: Maté es la máquina virtual de TinyOS, Bombilla es el conjunto de componentes que se sitúan sobre el sistema operativo para ejecutar los scripts de Maté.
- **MIG** (*Message Interface Generator*): Es una herramienta utilizada para generar automáticamente, las clases de Java correspondientes a los tipos de mensaje activos que utiliza la aplicación del mote. MIG lee en nesC, las definiciones struct para los tipos de mensajes en la aplicación del mote y genera una clase Java por cada tipo de mensaje. Esta herramienta en el presente proyecto juega un papel muy importante ya que ha permitido la creación de la aplicación que ha hecho posible la interactuación con los motes. A continuación se explicará esta herramienta con más detalle.

2.5.7 La herramienta MIG (Message Interface Generator) en NesC

La herramienta MIG se utiliza en todas aquellas aplicaciones de TinyOS en la que exista una comunicación entre el mote y el PC. MIG (*message interface generator*) es un instrumento para generar el código que procesa mensajes TinyOS y su definición se incluye en el Makefile.

Sinopsis:

*Mig tool [any ncc option] [**-o** output-file] [**-java-classname=**full-class-name] [**-java-extends=**class-name] msg-format-file message-type*

Descripción:

El argumento full-class-name especifica el nombre completo de la clase java que se desea generar y especifica el archivo nesC de cabecera de la aplicación. "*Message type"* especifica el nombre de la estructura de datos C que se desea procesar contenida dentro del fichero .h. El tipo de C debe definirse con una estructura o unión *"message-type"* en un archivo .h mencionado en el "include" de la aplicación nesC. Si el archivo .h que define el tipo de mensaje no depende de ningún otro archivo, se puede especificar el archivo .h directamente como "msg format file".
Si se encuentra una constante enumerada llamada AM mesage type, entonces se asume que el valor de esta constante es el tipo de mensaje activo para message type.

Actualmente sólo hay una herramienta (tool), **java**, que genera una clase java que codifica y descodifica mensajes. Las opciones **–java-*** son específicas para esta herramienta. Se describen a continuación:

-target=known-tinyos-platform: especifica la arquitectura del dispositivo en el que funcionará la aplicación en NesC que genera y recibe mensajes. Esta es una de las opciones del compilador ([any ncc option] como se especifica en la sinopsis). Por ejemplo, si se compila la aplicación para Tossim (tos simulator) se especificaría –target=pc. En el caso de este tfc "-target=$(PLATFORM)" hace referencia a la plataforma Telosb.

También hay otros parámetros opcionales que no ha sido necesario utilizarlos pero que se describen aquí a modo de información:

-o output file: especifica el fichero en el cual se debe depositar el código generado.
-I dir: especifica un directorio de búsqueda adicional para los componentes nesC.

2.5.8 La herramienta Java

En este proyecto, el programa, ejecutado en el ordenador que hace de base estación, se ha realizado en Java, porque TinyOS da soporte completo para el intercambio de datos entre una aplicación java, que es ejecutada en el PC y una aplicación NesC que se ejecuta en el mote.

2.6 Lenguaje de programación Java

Java es un lenguaje de programación orientado a objetos y desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo es muy parecido al de C y C++, y su principal característica es que un programa java se puede ejecutar en múltiples sistemas operativos (p. ej. Windows, Linux…etc) y plataformas (como por ejemplo, ordenadores, teléfonos móviles…etc.) sin necesidad de compilar la aplicación cada vez que se usa.

2.6.1 Conceptos fundamentales de Java

La estructura Java está formada principalmente por los siguientes elementos:

• **Objeto:** es la entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos). Se corresponde con los objetos reales del entorno donde está ubicado, o a objetos internos del sistema (del programa). Es una instancia a una clase.

• **Clase:** define las propiedades y el comportamiento de un tipo concreto de objeto. Puede haber varios objetos que pertenecen a una misma clase, de la misma forma que pueden haber varias variables del mismo tipo (por ejemplo del tipo número entero).

• **Método:** es el algoritmo asociado a un objeto (o a una clase de objetos).

Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" para interactuar con otro objeto del sistema.

• **Evento**: es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento, a la reacción que puede producir un objeto, es decir la acción que genera.

Como ya se comentó, la herramienta MIG genera una clase java para codificar y decodificar un mensaje que procede del mote (paquete tinyos), basándose en la infraestructura net.tinyos.message.

Para cada campo de la estructura de tipo de mensaje, existen los siguientes métodos:

• get_fname: consigue el valor del campo fname perteneciente a la estructura de datos que contiene el paquete tinyos algunos de los cuales se enumeran a continuación.

• set_*fname*: pone el valor en un campo.

• offsetBits_*fname*: retorna el bit de offset de un campo en un tipo de mensaje.

• size_*fname*: retorna el tamaño en bytes de un campo (no sirve para vectores) (ausente si el campo es un bit de campo).

• isSigned_*fname*: retorna cierto si el campo es un tipo de valor con signo.

• isArray_*fname*: retorna cierto si el campo es un vector.

Las estructuras empotradas dentro de la estructura de mensaje se expanden, usando "_" para separar el nombre de la estructura y sus campos. Por ejemplo si lo que se tiene como campo fname es en realidad una estructura de datos, y de la que se desea obtener el campo x, el método que realizaría esta tarea sería get_*fname_x*.

También existe un número de métodos para campos que son vectores:

• getElement_*fname*: consigue un elemento de un vector.

• setElement_*fname*: pone un elemento en un vector.

• elementSize_*fname*: retorna el tamaño en bytes de los elementos de un vector.

• elementSizeBits_*fname*: retorna el tamaño en bits de los elementos de un vector.

• numDimensions_*fname*: retorna la dimensión de un vector.

• numElements_*fname*: retorna el número de elementos de un vector para una dimensión dada (la dimensión más a la izquierda se nombra con 0) – la dimensión es opcional para vectores unidimensionales.

• totalSize_*fname*: retorna el tamaño en bytes de un vector (ausente si el vector es de tamaño variable).

• totalSizeBits_*fname*: retorna el tamaño en bits de un vector (ausente si el vector es de tamaño variable).

La herramienta java acepta las siguientes opciones:

-java-classname=full-class-name

Esta opción se requiere para especificar el package y nombre de la clase generada.
"full-class-name" se debe escribir como package.
"nombreDeLaClase" donde el package hace referencia a un conjunto de clases java que se agrupan en un directorio. Esta agrupación ayuda a organizar clases java

-java-extends=class-name:

Especifica de qué clase heredará la clase generada por MIG. La clase de la que hereda por defecto es net.tinyos.message.

Capítulo 3

DESCRIPCIÓN DEL SISTEMA

3.1 Introducción

En el capítulo anterior, se ha descrito todo lo relacionado con los motes, su funcionamiento, su gran variedad disponible en el mercado, sus aplicaciones, sus capacidades operativas, así como los sistemas operativos empleados, todo esto con el fin de conocer y comprender todo lo referente a los motes.

Es por ello por lo que en este capítulo se centrará en explicar el sistema operativo utilizado en este proyecto, el software y el hardware empleado, así como sus características técnicas, su instalación y funcionamiento. En definitiva, todos sus componentes, como de los programas necesarios para su programación.

3.2 Telosb

3.2.1 Introducción

Como se comentó en el capítulo 2, los motes Telosb son dispositivos que funcionan como nodos en las redes inalámbricas de sensores; siendo de muy baja potencia, y muy utilizado en aplicaciones de monitorización rápida de prototipos. Puede operar con otros dispositivos ya que han sido diseñados con estándares como USB e IEEE 802.15.4.

El Telosb es adecuado en módulos con sensores de bajo coste destinados a aumentar la robustez y el tamaño del conjunto del dispositivo. Este mote lleva integrado un sensor de temperatura, uno de humedad y otro de luz (radiación solar total y radiación fotosintéticamente activa) y es muy flexible a la hora de conectarse con otros sensores externos o periféricos, lo que le hace tener una gran variedad de aplicaciones.

3.2.2 Descripción del módulo

En las siguientes figuras 3.1 y la figura 3.2 se presenta los distintos sensores integrados del Telosb, así como la disposición de la radio y su antena, y el lugar que ocupa el microcontrolador:

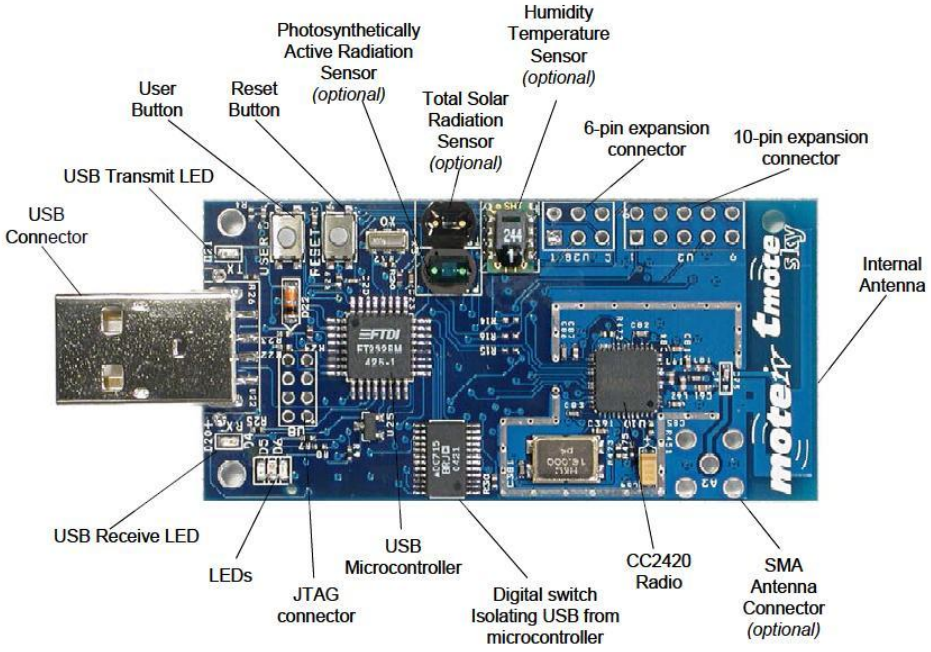


Figura 3. 1: Parte frontal del Telosb.

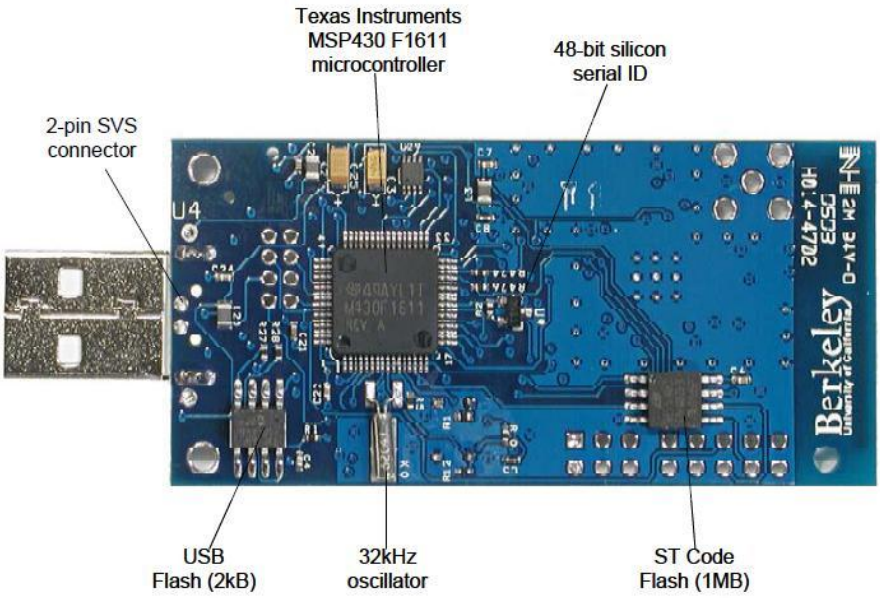


Figura 3. 2: Parte trasera del Telosb.

- Alimentación:

  El Telosb es alimentado mediante dos baterías tipo AA. Ha sido diseñado para que las baterías sumistren un voltaje en el rango de 2,1 a 3,6V, aunque es necesario para programar el microcontrolador una tensión superior a los 2,7V. Cuando el Telosb se encuentre conectado por el puerto USB, se alimenta a una tensión de 3V transformados ya que a través de un puerto USB posee un voltaje de 5V, por lo que el mote lleva un regulador integrado. Cuando el mote esté conectado al ordenador por el puerto USB no es necesario el uso de las baterías.

- Condiciones típicas de trabajo:

	MÍNIMO	NOMINAL	MÁXIMO	UNIDAD
Alimentación	2,1		3,6	V
Alimentación durante la programación	2,7		3,6	V
Temperatura ambiente de trabajo	-40		85	ºC
Consumo corriente: MCU on, Radio RX		21,8	23	mA
Consumo corriente: MCU on, Radio TX		19,5	21	mA
Consumo corriente: MCU on, Radio off		1800	2400	µA
Consumo corriente: MCU libre, Radio off		54,5	1200	µA
Consumo corriente: MCU standby		5,1	21	µA

Tabla 3. 1: Condiciones de trabajo de un Telosb.

3.2.3 Microprocesador

El bajo consumo de energía del Telosb se debe al microcontrolador MSP430 de ultra bajo consumo, con 10 Kb de RAM, 48 Kb de memoria flash y 128 bytes de almacenamiento de información.

Este procesador es de 16-bit RISC, y posee unas características de actividad muy baja y, gracias a la función de *sleep*, hace posible la utilización del Telosb durante varios años con sólo un par de baterías AA. Aunque para que esto sea posible ha de estar programado de forma que el consumo sea lo mínimo posible, y que el paso de modo *sleep* a estar activo sea muy rápido, contando además con un tiempo de actividad de duración suficiente.

El módulo MSP430 posee un oscilador interno controlado digitalmente (DCO) que puede alcanzar una frecuencia de trabajo hasta los 8 MHz. Se puede encender desde el modo *sleep* en 6 µs; sin embargo el valor típico de encendido a temperatura ambiente es de 292 ns. Cuando el DCO está apagado, el MSP430 opera con un cristal de reloj a

32,768 kHz. A pesar de los cambios de frecuencia del DCO con la tensión y la temperatura, puede ser calibrado mediante el uso de un oscilador de 32 KHz.

Además del DCO, el MSP430 tiene 8 puertos externos ADC y 8 puertos internos ADC. Los puertos internos del ADC pueden ser usados para leer el termistor interno o controlar el voltaje de la batería. Dispone también de una gran variedad de periféricos disponibles, incluyendo SPI, ART, puertos I / O digitales, *watchdog*.

Las características de la familia de MSP430 se presentan en detalle en la Guía del usuario de Texas Instrument [16].

- Características típicas de trabajo:

	MÍNIMO	NOMINAL	MÁXIMO	UNIDAD
Alimentación durante ejecución	1,8		3,6	V
Alimentación durante programación	2,7		3,6	V
Temperatura ambiente	-40		85	ºC
Baja frecuencia del cristal		32,768		KHz
Corriente activa a Vcc = 3V, 1MHz		500	600	µA
Corriente en *sleep*		2,6	3	µA
Tiempo activación modo *wake up*			6	µs

Tabla 3. 2: Características del microprocesador del Telosb.

- Comunicación con el PC

  El Telosb hace uso del controlador de USB para comunicarse con el ordenador que actúa como host. Para que el mote pueda conectarse con el PC. Éste debe tener instalados los drivers correspondientes del nuevo hardware. Estos drivers son proporcionados por el FTDI, y compatibles para todos los sistemas operativos, además de que se pueden descargar de internet de forma gratuita.
  Para comprobar que el PC ha detectado el mote, así como el numero de puerto al que está conectado este, es necesario abrir el Administrador de dispositivos, al cual se puede acceder a él desde Equipo, botón derecho Propiedades. En la figura 3.3 se observa una captura de pantalla en la cual se puede ver que el mote ha sido detectado:

  En esta figura, se puede observar como el ordenador asocia una conexión con el mote por el puerto COM5. Se pueden conectar varios motes al mismo tiempo, a los cuales se les asignará un puerto COM diferente.

  Otra forma de averiguar si el mote ha sido detectado es introduciendo en el terminal el comando siguiente:

  $ motelist

  Si, por el contrario, lo que se hace es programar y cargar en el mote un código, en el terminal se debe escribir el siguiente comando:

  $ make telosb install,n bsl,p

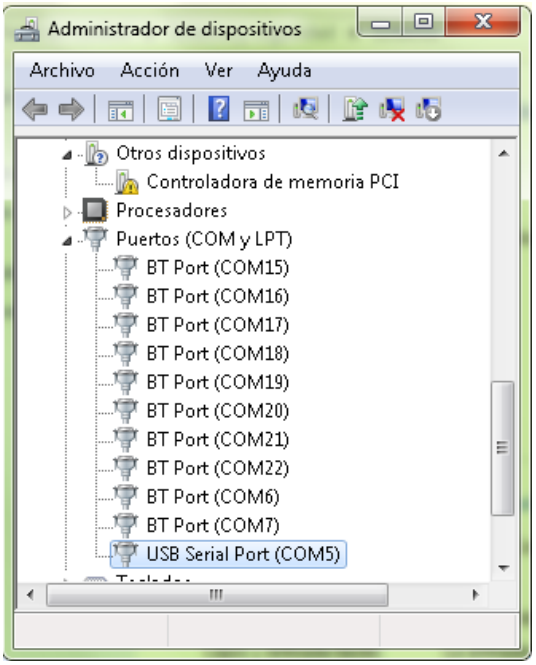


Figura 3. 3: Detección del Telosb en el Administrador de Dispositivos de Windows.

Donde "n" es la dirección asignada al nodo y "p" el puerto COM donde está conectado el mote, el cual se le resta una unidad. Es decir, que si por ejemplo el mote está conectado al puerto COM5, "p" es igual a 5-1=4.

$ make telosb install,1 bsl,4

- Diagrama de bloques:

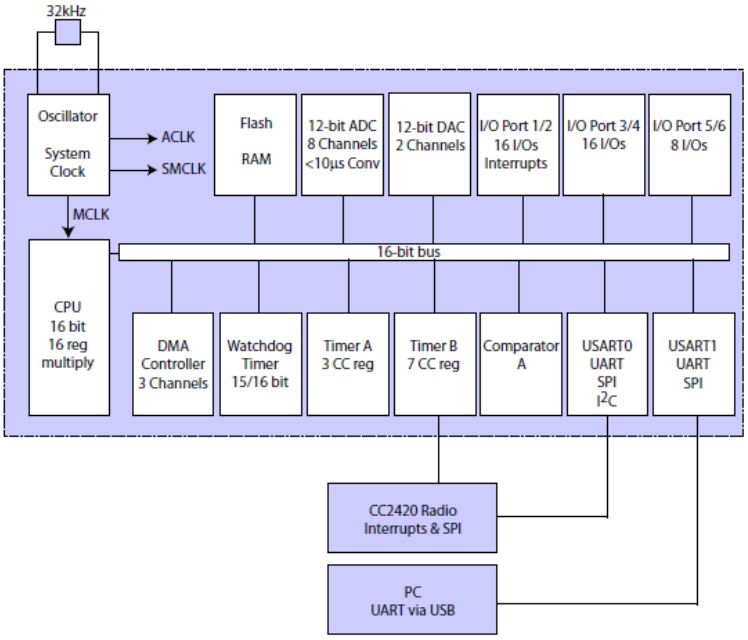


Figura 3. 4: Diagrama de bloques del microprocesador del Telosb.

La figura 3.4 anterior representa el microprocesador que lleva interno del Telosb, en el cual se puede observar un oscilador, los diferentes *timers*, el *watch-dog*, los bloques correspondientes a las UART, una memoria flash para la memoria RAM, los diversos bloques de entradas/salidas, un comparador, el controlador DMA, y un bus de 16 bits.

3.2.4 Radio

Para el envío de datos entre motes y el envío de información, Telosb cuenta con el módulo de radio Chipcon CC2420 para comunicaciones inalámbricas. El protocolo utilizado es el IEEE 802.15.4 con un bajo consumo de energía, que proporciona una comunicación inalámbrica bastante fiable. Para el modo *sleep*, generalmente el microcontrolador apaga la radio, disminuyendo el consumo y provocando una mayor duración en la vida de los motes.

El CC2420 proporciona una RSSI (*Received Signal Strength Indicator*) que se puede leer en cualquier momento. Además, en la recepción de cada paquete, el CC2420, calcula el error y produce una indicación de la calidad del enlace, para cada uno de los valores recibidos.

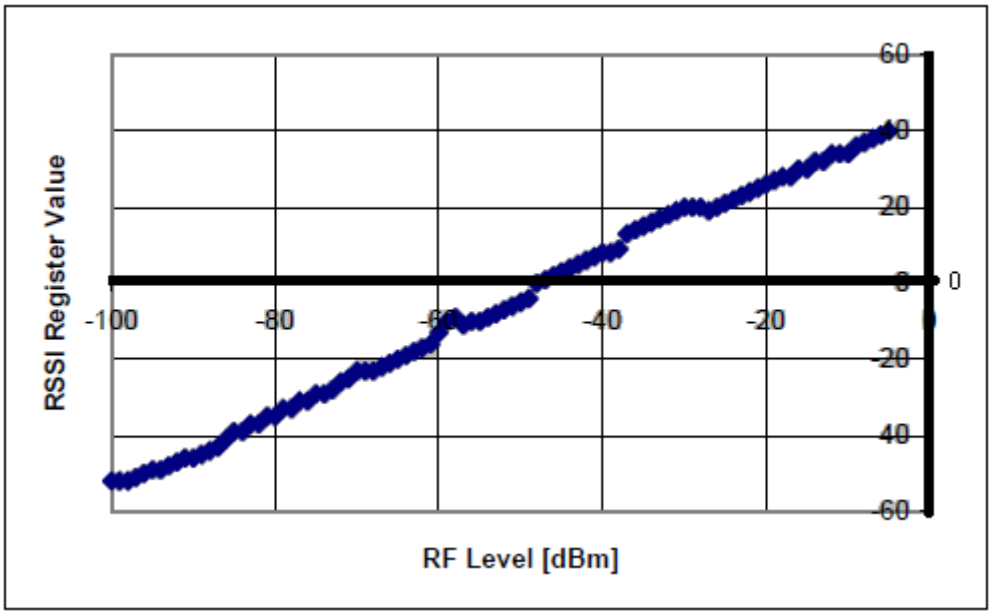


Figura 3. 5: Gráfica relación RSSI – RF.

- Condiciones típicas de trabajo:

	MÍNIMO	NOMINAL	MÁXIMO	UNIDAD
Alimentación durante operación de radio	2,1		3,6	V
Temperatura ambiente	-40		85	ºC
Rango de frecuencias de	2400		2483,5	MHz

RF				
Velocidad transmisión de bit	250		250	Kbps
Energía nominal de salida	-3	0		dBm
Rango programable energía de salida		40		dBm
Sensibilidad del receptor	-90	-94		dBm
Consumo corriente: Radio envia a 0 dBm		17,4		mA
Consumo corriente: Radio recibiendo		19,7		mA
Consumo corriente: Radio y oscilador on		365		µA
Consumo corriente: Oscilador off		20		µA
Consumo corriente: Sleep y Vreg off			1	µA
Corriente del regulador de tensión	13	20	29	µA
Tiempo puesta en marcha radio		580	860	µs

Tabla 3. 3: Características de la radio del Telosb.

Por último, cabe destacar que el rango de la radio está entre los 20 a los 30 metros en interiores, y hasta los 150 metros en el exterior. Es posible aumentar más estas distancias conectando una antena SMA adicional.

3.2.5 Sensores internos del Telosb

El Telosb incluye varios sensores:

- SHT11
  El sensor SHT11, que en algunos casos es el SHT15, es el que se encarga de conocer la temperatura y la humedad existentes en el lugar de localización del mote. Están preparados para producir una salida digital, además de utilizar una EEPROM para almacenamiento de datos y calibración de los coeficientes. La diferencia entre ambos está en la precisión.

PARÁMETRO	**MÍNIMO**	**TÍPICO**	**MÁXIMO**	**UNIDAD**
HUMEDAD				
Resolución	0,5	0,03	0,03	%RH
	8	12	12	Bit
Repetibilidad		±0,1		%RH
Rango	0		100	%RH
TEMPERATURA				
Resolución	0,04	0,01	0,01	ºC
	0,07	0,02	0,02	ºF
	12	14	14	Bit
Repetibilidad		±0,1		ºC
		±0,2		ºF
Rango	-40		123,8	ºC

	-40		254,9	ºF

Tabla 3. 4: Especificaciones de los sensores de humedad y temperatura.

En las siguientes gráficas muestra la precisión de los sensores SHT11 y SHT15 ante la temperatura y la humedad relativa:

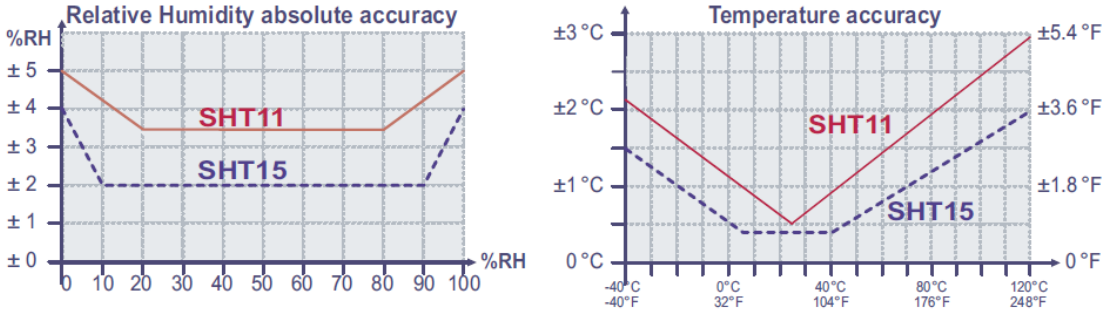


Figura 3.6: Precisión de SHT11 y SHT15.

- Sensores de luz

  El mote incluye dos sensores de luz, el S1087 para mediciones de radiación solar total y el S1087-01 para medir la radiación fotosintéticamente activa, que incluye todo el espectro visible incluido la luz infrarroja.
  Estos fotodiodos son de la marca Hamamatsu, aunque cualquier sensor del mismo estilo puede ser sustituido o utilizado en su lugar.
  En la siguiente gráfica se muestra la relación entre la foto sensibilidad y la longitud de onda para una temperatura ambiente de 25ºC.

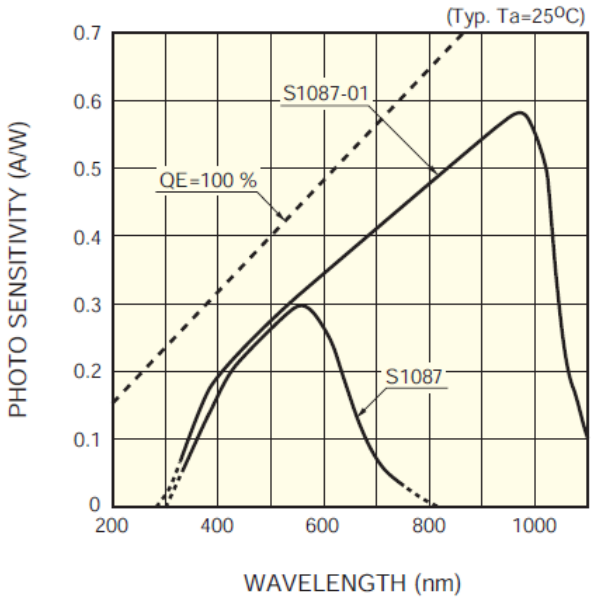


Figura 3. 7: Gráfica foto sensibilidad - longitud de onda.

- Conectores para expansión.

  El Telosb posee dos conectores y un par de *jumpers*, que deben de ser configurados para que sea posible conectar periféricos, tales como un display LCD u otros tipos de sensores analógicos. Los conectores se encuentran

accesibles en la posición U28 y en la U2, de 6 y 10 pines respectivamente, como se puede observar en las siguientes imágenes:

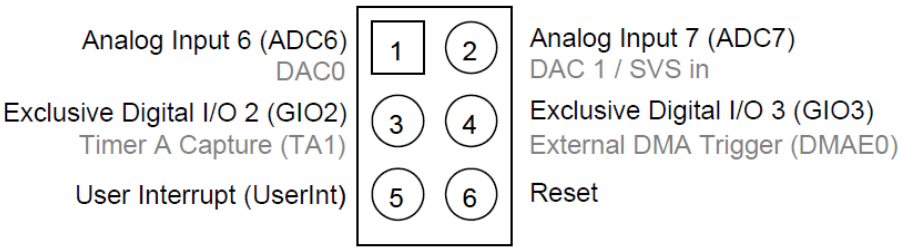


Figura 3.8: Conector de 6 pines (U28)

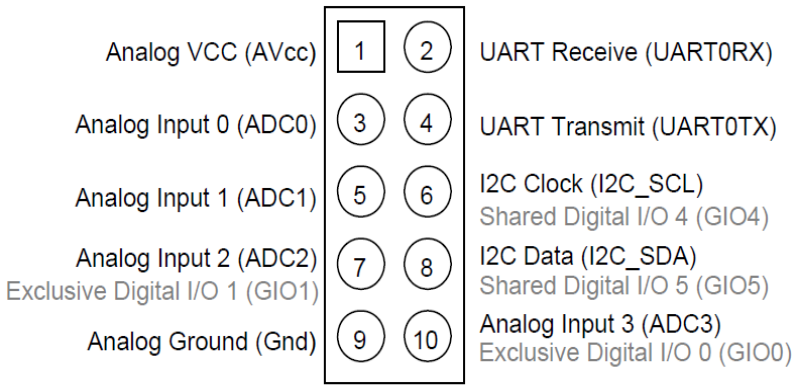


Figura 3. 9: Conector de 10 pines (U2)

3.3 CMUcam3 (Cámara)

3.3.1 Introducción

En la figura 3.10 se puede ver la CMUcam3 [17]:

a) **b)**

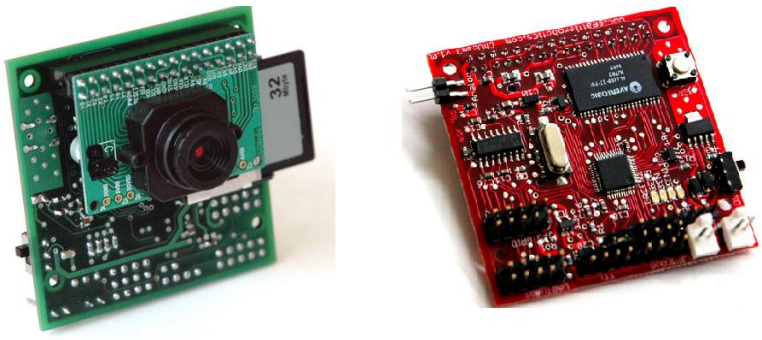


Figura 3.10: Imágenes de la CMUcam3: a) parte frontal b) parte posterior

Para la realización de este proyecto se ha utilizado una cámara con un sistema de código abierto y realización de fotografías en formato digital llamada CMUcam3. Se trata de un sensor de visión de bajo coste que es totalmente programable, por lo que se puede utilizar en múltiples aplicaciones incluyendo reconocimiento de formas,

detección de colores, seguimiento de objetos en movimiento, etc.; todo ello dependiendo del software empleado. Este sensor de visión ha sido desarrollado por la universidad Carnegie Mellon y se ha convertido en todo un estándar que cuenta con gran cantidad de software disponible de muchas fuentes diferentes. Cuenta con procesador NXP LPC2106 y un sensor CMOS de la firma Omnivision, formando un sensor de visión basado en el ARM7TDMI. La CMUcam 3 tiene entre otras características: Sensor RGB en color con una resolución de 352 x 288, conector para tarjeta SD o MMS con soporte para FAT16, conexión para 4 servos, alta velocidad de 26 imágenes por segundo, compresión por software en modo JPEG, modo de emulación de la CMUcam2.

3.3.2 Características

A continuación se muestran las características de la CMUcam3:

- Resolución CIF (352x288) sensor de color RGB.
- Entorno de desarrollo de código abierto compatible con Windows y Linux.
- Slot de Flash para MMC con soporte FAT16.
- Cuatro puertos para conectar servos.
- Capacidad de carga de imágenes de hasta 26 imágenes por segundo.
- Lenguaje LUA, es un lenguaje de programación imperativo, estructurado y bastante ligero que fue diseñado como un lenguaje interpretado con una semántica entendible y rápido para prototipos.
- Software para compresión de las imágenes en formato JPEG.
- Librería para manipulación básica de la imagen:
  - Recorte de imagen arbitraria.
  - Reducción de la resolución de la imagen.
  - Configuración de las propiedades de la cámara para la obtención de las imágenes.
  - Umbral y funciones de convolución.
  - RGB, YCrCb y HSV Color Space.
- Emulación de CMUcam2.
  - Manchas de color definidas por el usuario.
  - Estructura de diferenciación.
  - La media y la varianza de los datos de recogida.
  - Las imágenes en formato RAW se transmite en forma serie.
  - Generación de histograma.
- B/W salida analógica de video (PAL o NTSC).
- FIFO buffer de imagen para el procesamiento de imágenes múltiples de alta resolución.
- Conector compatible con redes de sensores (Tmote Sky, FireFly, 802.15.4).

3.3.3 Aplicaciones

La CMUcam3 es una plataforma de hardware, junto con un entorno de desarrollo de código abierto. Está dirigido a usuarios que ya están familiarizados con el procesamiento de imágenes básico y que se sienten cómodos con la programación de microcontroladores. Para los usuarios que quieren el procesamiento básico de imágenes accesibles a través de una interfaz en serie simple y no desean poner en práctica sus

propios algoritmos, es posible instalar una utilidad de flash y firmware CMUcam2 de emulación. Esto permite que la CMUcam3 pueda ser emulada imitando la interfaz proporcionada por la CMUcam2. La CMUcam3 se puede utilizar para una variedad de aplicaciones, tales como en robots, vigilancia, en redes de sensores, en actividades educativas, en juegos interactivos, en reconocer y rastrear objetos, para la programación y control de Servos, para series de registros de datos mediante una Flash MMC, etc.

3.3.4 Diagramas de bloques

A continuación se muestra un diagrama de bloques de alto nivel de los componentes principales que se encuentran en el CMUcam3:

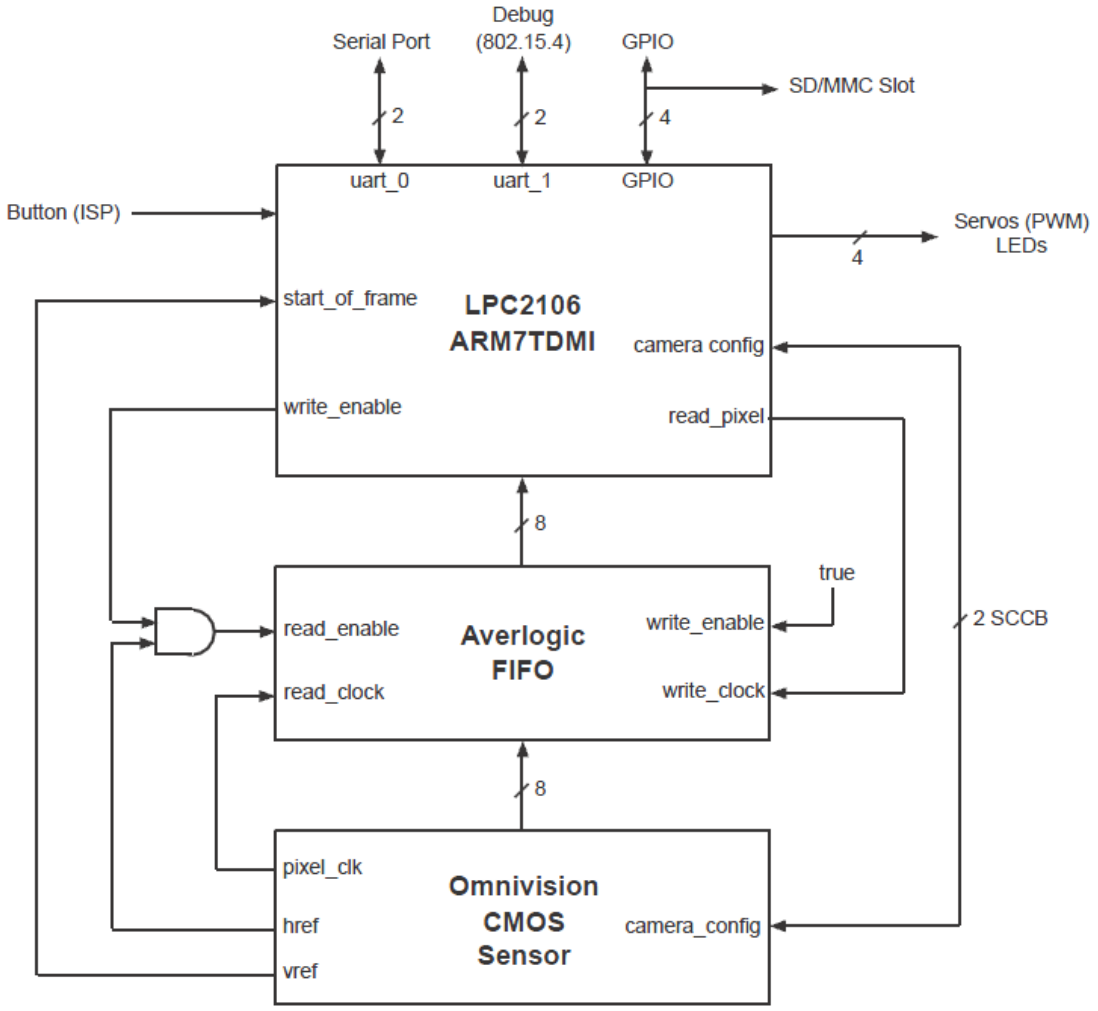


Figura 3.11: Diagrama de bloques CMUcam3

3.3.5 Conexiones hardware

A continuación, se explicará las distintas conexiones que dispone la CMUcam3, así como las posibilidades de las que dispone para añadir más periféricos o dispositivos para una determinada aplicación.

Se profundizará más adelante, en el cápitulo 4, las conexiones llevadas a cabo para realizar el proyecto tratado, así como los circuitos acondicionados que se han diseñado y desarrollado para su correcta conexión con el Telosb.

Todo esto para realizar las capturas de imágenes, que serán enviadas por radio, para posteriormente ser procesadas en la aplicación java ejecutada en un PC.

En la figura 3.12 se muestra las conexiones Hardware que dispone y ofrece este dispositivo, la CMUcam3:

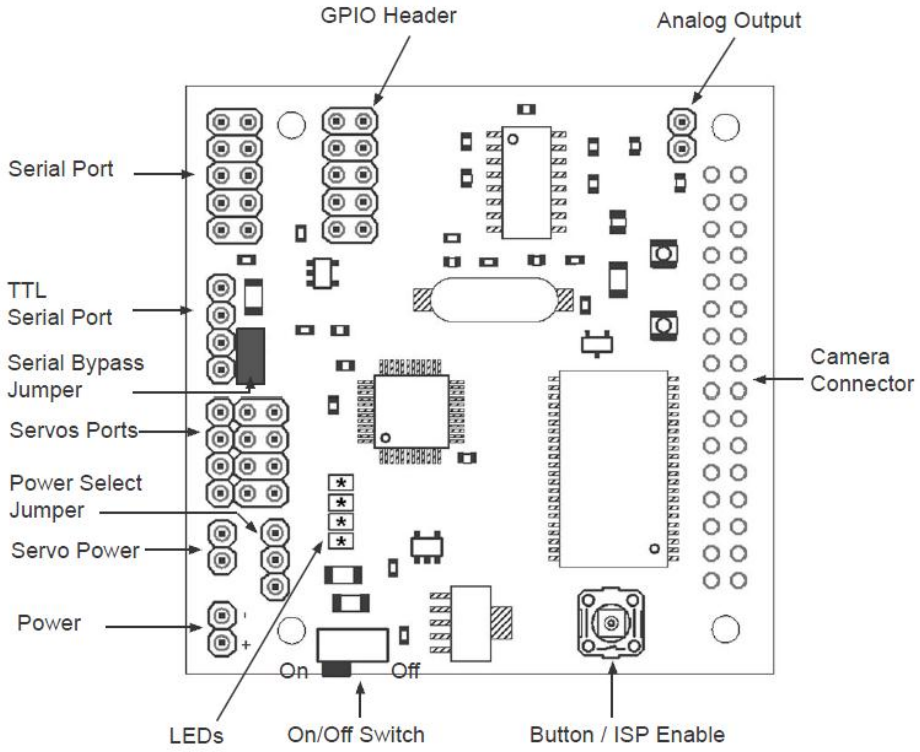


Figura 3.12: Esquema de las conexiones Hardware

3.3.5.1 Alimentación (Power)

La alimentación de la placa pasa a través de un regulador de 5 voltios. Es ideal para suministrar a la placa entre 6 y 15 voltios de corriente continua que es capaz de suministrar un mínimo de 150 miliamperios.

Los servos pueden ser alimentados por una fuente interna, o por el conector de servo de alimentación externa. Para ejecutarlos mediante la fuente de alimentación externa, es necesario retirar el puente de servo de alimentación interna, como se muestra a continuación.

Se debe de tener en cuenta que alimentar los servos con la energía interna puede llevar a que los servos requieran de más corriente que la permitida por la fuente, esto puede llevar a que tanto los servos y el procesador pueden dejar de funcionar correctamente.

3.3.5.2 Puerto Serie (Serial Port)

La CMUcam3 tiene un nivel estándar de puerto serie para permitir la comunicación con un ordenador, así como un puerto serie TTL para comunicarse con un microcontrolador.

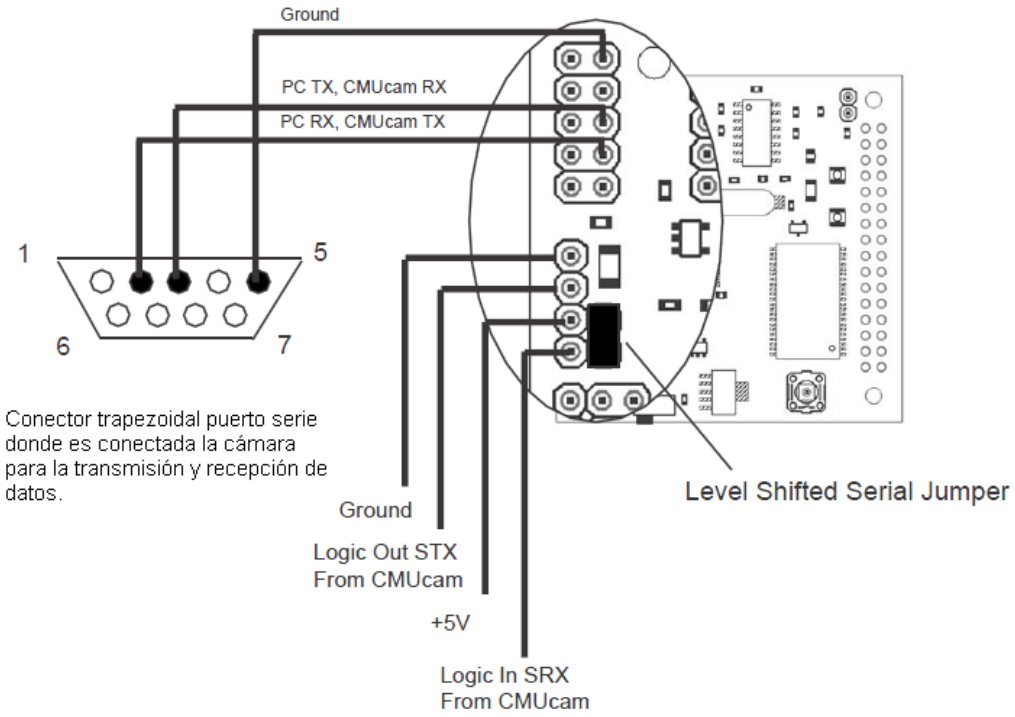


Figura 3.13: Ampliación de las conexiones del puerto serie

El nivel de puerto serie sólo utiliza 3 de los 10 pines. Se encuentra en una configuración de pines 2x5 que es compatible con un cable serie de 9 pines de los habituales en la comunicación serie hembra de 10 pines, al que es posible conectar un cable para establecer también comunicaciones. Si esto no funciona, es necesario voltear el sentido del cable plano que conecta a la tarjeta CMUcam2. Es necesario que el puente de serie este en su lugar cuando se utiliza este modo. El conector TTL se puede utilizar para conectarse con un microcontrolador sin que sea necesario el uso de un chip de adaptación de niveles. Los pines de salida TTL tienen un voltaje entre 0 y 3.3V, pero son tolerantes a 5 voltios en la entrada. Es necesario retirar el puente de serie cuando se utiliza este modo.

3.3.5.3 BUS de la CMUcam3

La interfaz de bus está conectada con el chip de la cámara CMOS. La cámara CMOS de la placa se monta en paralelo a la parte de procesamiento de la tarjeta y se conecta a partir del pin 1.

La cabecera de la cámara debe ser soldada en la parte posterior de la placa principal CMUcam3.

El CMUcam3 actualmente trabaja con los módulos CMOS de la cámara, OV6620 y OV7630.

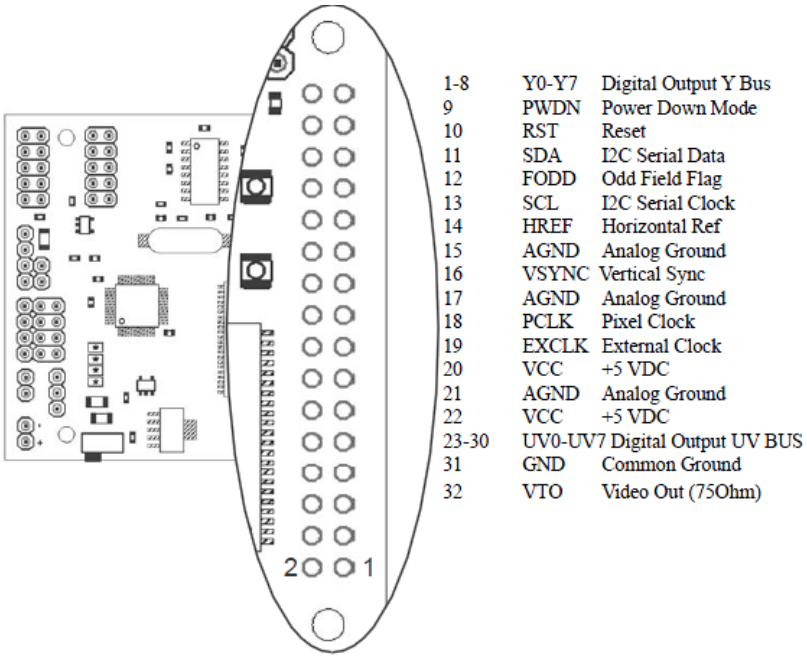


Figura 3.14: Ampliación de los módulos OV6620 y OV7630

3.3.5.4 Puerto de conexión de servos

La CMUcam3 tiene la capacidad para controlar hasta 4 servos. Esto puede ser útil si no se desea utilizar un controlador de servo por separado. El puerto de servo también se puede utilizar como una salida digital de propósito general.

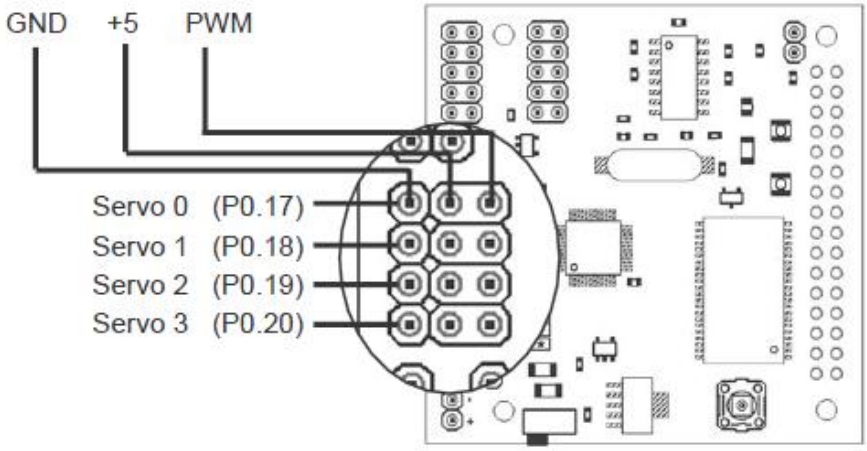


Figura 3.15: Ampliación del puerto para conectar los servos

3.3.5.5 Puerto de expansión GPIO

El conector I/O de propósito general permite el acceso a la UART secundaria, a varios pines de control de alimentación y a los pines de comunicación mediante el bus SPI.

***Power Enable*:** Cuando está baja la alimentación, el principal regulador de CMUcam3 se desactiva haciendo que el consumo de la placa no sea superior a 0.01uA. Todos los

dispositivos se desconectarán y se perderá toda la información del estado en activo. Cuando la línea se libera o la alimentación sea alta, la placa se reiniciará.

AUX Power: Este pin se puede configurar para cualquier fuente externa de la placa, o para la alimentación de una tarjeta de expansión. Por defecto, el pin está conectado a la fuente de alimentación interna a 3.3volt. Mediante la eliminación de la resistencia R11 y la adición de una resistencia de puente en lugar de R6, el pin se conecta a la corriente principal antes que el regulador de 5 voltios.

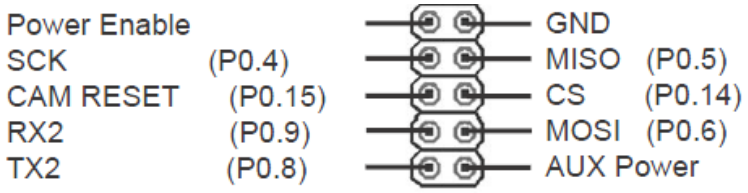


Figura 3.16: Ampliación del puerto de expansión GPIO.

***CAM RESET*:** Este pin se puede utilizar como E / S externos si el estado de la cámara no es necesario. Normalmente, este pin es el encargado de reiniciar el módulo de la cámara y no se debe utilizar.

***TX2*:** Es el pin que transmite en UART2. Si no se cambia el nivel no se puede conectar directamente a un PC o dispositivo no TTL externo. Este pin también se puede utilizar como GPIO.

***RX2*:** Es el pin que recibe señales en UART2 y, al igual que sucede con el pin TX2, si no se cambia el nivel no se puede conectar directamente a un PC o dispositivo no TTL externo. Este pin también se puede utilizar como GPIO.

CS: En el reinicio, si este pin se mantiene bajo, el LPC2106 entrará en el modo de arranque. El reinicio puede ser inducido desde el exterior por un pulso de entrada por pin AUX Power. Normalmente este pin será controlado por el slot de la tarjeta de memoria (MMC). Este pin también se puede utilizar como GPIO o como el chip SPI cuando la tarjeta MMC no está insertada.

MOSI: Normalmente este pin es controlado por el controlador de la MMC. Este pin también se puede utilizar como GPIO o como una salida SPI cuando no haya ninguna tarjeta MMC insertada.

MISO: Como el pin MOSI, es controlado por el controlador de MMC. Este pin también se puede utilizar como GPIO o como entrada SPI cuando no hay una tarjeta insertada.

SCK: También este pin es controlado por el controlador de MMC. Este pin también se puede utilizar como GPIO o como el pin de reloj SPI cuando no hay una tarjeta MMC insertada.

3.3.5.6 Puerto analógico de salida

Utilizando el módulo de la cámara OV6620, se puede obtener una señal de vídeo PAL desde el puerto analógico de la CMUcam3. Esta se puede sincronizar con cualquier monitor PAL, pero no va a funcionar con un monitor estándar NTSC. El módulo de la cámara OV7620 genera una señal de video de salida basada en el estándar NTCS en blanco y negro. Para utilizar esta salida, es necesario mantener la cámara a su máxima velocidad de reproducción (por defecto) y ponerla en el modo de YCrCb con el fin de ver la imagen en un monitor.

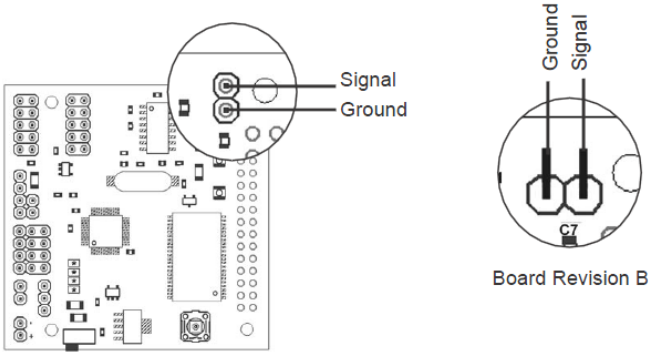


Figura 3.17: Ampliación del puerto de salida analógico.

3.3.5.7 LEDs

LED 0 - Este pin se comparte con el pin MOSI y debe ser utilizado solamente cuando CS se encuentra inhabilitado mientras haya una tarjeta MMC insertada en el slot correspondiente.

LED 1 - Este pin se comparte con el servo de 2 pines. Al utilizar Servo 2, el LED parpadea.

LED 2 - Este pin se comparte con el servo de 3 pines. Al utilizar Servo 3, el LED parpadea.

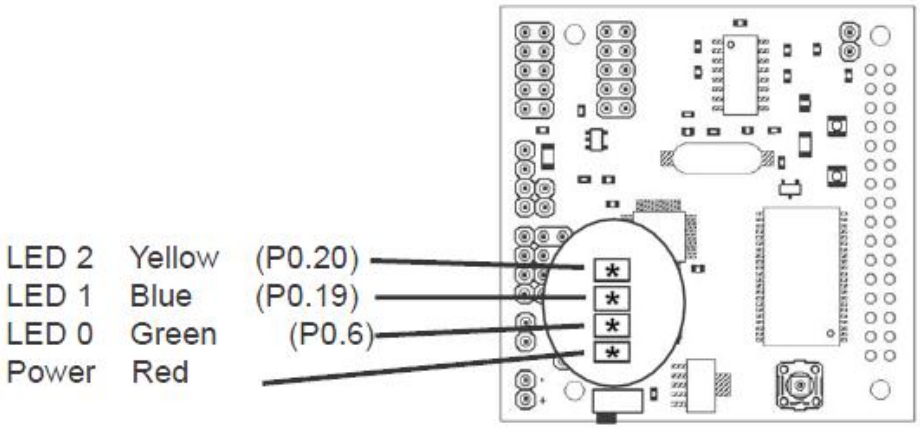


Figura 3.18: Ampliación de la ubicación de los LED

3.3.5.8 Botón ISP

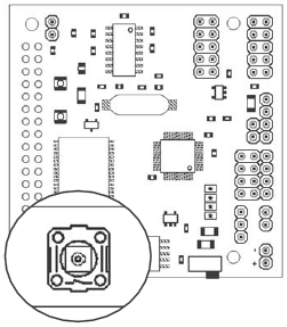


Figura 3. 19: Ampliación del botón ISP en la placa

Cuando se mantiene presionado durante el encendido el botón ISP, se habilita la carga del *bootloader* al LPC2106. Después de que el procesador se ha puesto en marcha, el botón se puede leer como GPIO normal. Internamente está puesto a nivel alto, y se pone a cero cuando se deja de pulsar. Cuando se utiliza la tarjeta MMC, CS es activa a nivel bajo y, por lo tanto, el botón no puede afectar negativamente a la transferencia de datos.

3.3.6 Características Hardware

A continuación, se muestran las características Hardware de la CMUcam3 utilizada en este proyecto:

Power State	Active Current	Idle Current	Voltage
All Active	130mA	25mA	5V
External Regulator Disabled	n/a	0.01uA	n/a
CPU (@60MHZ)	30mA	10uA	1.8V
CPU Peripherals	30	10uA	3.3V
CMOS camera	25mA	10uA	5V
MAX232	8mA	n/a	3.3V
FIFO	52mA	14mA	3.3V
MMC card	4mA	4mA	3.3V
Misc	10mA	10mA	3.3V

Tabla 3.5: Información sobre el consumo de energía para varios componentes.

COMPONENTE		DESCRIPCIÓN
CPU		
	RAM	64KB
	ROM	128KB (8KB del bootloader)
	Frequencia	(14-60)MHZ
FIFO		
	Capacidad	1 MB
	Ratio máximo	50 FPS
CMOS camera		
	Máxima resolución	352x288

	Profundidad de color	8bits por segundo
Misc		
	Ratio máximo por puerto serie	115,200 bits por segundo
	Resolución completa de la carga de la imagen y tasa de toque de píxeles	26 FPS
	Frequencia del servo	50 HZ

Tabla 3.6: Características CMUcam3.

Capítulo 4

COMPONENTES HARDWARE

4.1 Introducción

Hasta ahora se ha presentado los distintos dispositivos que componen el proyecto, así como las especificaciones, el funcionamiento, y el papel que representan. No obstante para poder conectar estos distintos dispositivos entre sí se han tenido que diseñar y realizar una serie de circuitos acondicionadores. Es decir, la CMUcam3, o la cerradura eléctrica, para poder conectarlos a los motes es necesaria una serie de circuitos auxiliares para la sincronización y un correcto funcionamiento entre ellos. De otra forma se produciría un funcionamiento indeseado o no el funcionamiento de los dispositivos, incluso la destrucción de los mismos.

En este capítulo se describe los circuitos de acondicionamiento que se han tenido que llevar a cabo, sin otra finalidad, que la de conseguir la estabilidad en el funcionamiento del prototipo de videoportero.

4.2 Acondicionamiento de la cámara CMUcam3

Para el acondicionamiento del CMUcam3 con el mote era necesario realizar una adaptación de niveles de tensión, para hacer posible una comunicación serie asíncrona [19] entre ellos.

Este tipo de conexión es muy típica entre dispositivos, como un ordenador con un microcontrolador, en este caso un Telosb. En ella los bits de datos se transmiten "en serie" (uno detrás de otro) y cada dispositivo tiene su propio reloj. Estableciendo previamente que ambos dispositivos transmitirán datos a la misma velocidad.

A continuación se muestran los fundamentos de estas comunicaciones, los pines empleados y esquemas del circuito para conectar la CMUcam3 con el Telosb, además de mostrar los cables que se han empleado.

4.2.1 Comunicaciones Serie Asíncronas

Los datos serie se encuentran encapsulados en tramas de la forma siguiente:

Figura 4.1: Trama de comunicación asíncrona

Primero se envía un **bit de *start***, a continuación los **bits de datos** (primero el bit de mayor peso) y finalmente los **bits de *STOP***.

Tanto el número de bits de datos y el de bits de Stop es uno de los parámetros configurables, así como el criterio de paridad par o impar para la detección de errores. Normalmente, las comunicaciones serie tienen los siguientes parámetros: **1 bit de *Start*,** 8 bits **de Datos,** 1 bit de ***Stop*** **y** sin **paridad**.

En esta figura 4.2 se puede ver un ejemplo de la transmisión del dato binario 10011010. La línea en reposo está a nivel alto:

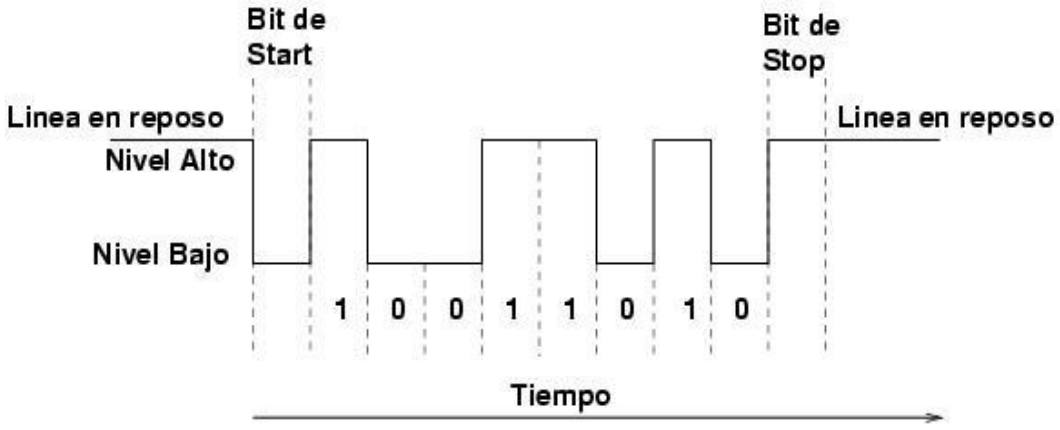


Figura 4.2: Ejemplo de transmisión de datos binarios

4.2.2 Conexión de un microcontrolador al puerto serie del PC

Para conectar la CMUcam3 al microcontrolador interno del TelosB por el puerto serie, en este caso el MSP430, se utilizan las señales *Tx*, *Rx* y *GND*. La CMUcam3 utiliza la norma RS232, por lo que los niveles de tensión de los pines están comprendidos entre +15 y -15 voltios. Los microcontroladores normalmente trabajan

con niveles TTL (0-5v). Es necesario por tanto intercalar un circuito que **adapte los niveles**:

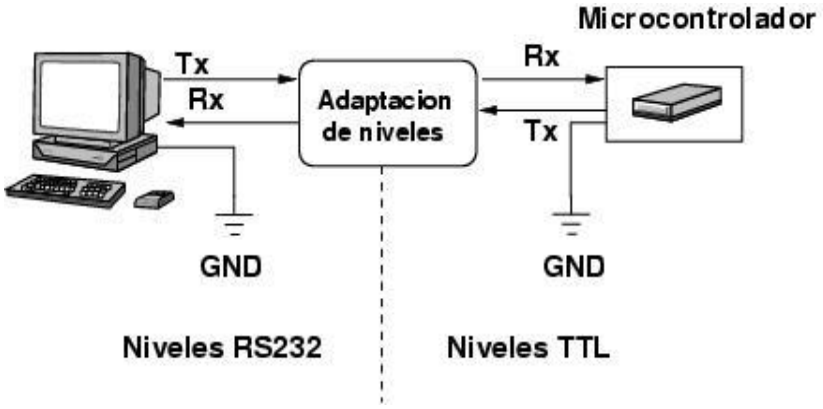


Figura 4.3: Ejemplo de adaptación de niveles entre PC – Microcontrolador

Uno de estos circuitos, que se utiliza mucho, es el MAX232, y es el que se ha utilizado para la adaptación de niveles entre el mote y la CMUcam3 en este proyecto.

4.2.3 El conector DB9 de la CMUcam3

La CMUcam3 tiene un conector DB9 macho, de 9 pines, para su conexión por el puerto serie. Los conectores hembra que se enchufan tienen una colocación de pines diferente al de los conectores macho, de manera que se conectan el pin 1 del macho con el pin 1 del hembra, el pin2 con el 2, etc.

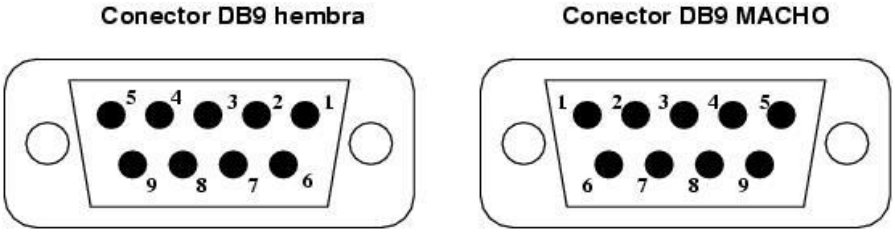


Figura 4.4: Conector DB9 hembra y Macho

La información asociada a cada uno de los pines es la siguiente:

Número de pin	**Señal**
1	**DCD** (*Data Carrier Detect*)
2	**RX**
3	**TX**
4	**DTR** (*Data Terminal Ready*)
5	**GND**
6	**DSR** (*Data Sheet Ready*)
7	**RTS** (*Request To Send*)
8	**CTS** (*Clear To Send*)

9	**RI** (*Ring Indicator*)

Tabla 4.1: Pines del DB9

4.2.4 El chip MAX 232

Como ya se comentó anteriormente, el chip utilizado para la adaptación de los niveles entre el mote y la CMUcam3 es el MAX 232.

Este chip permite adaptar los niveles RS232 y TTL, permitiendo conectar un PC, o en este caso, la CMUcam3 con un microcontrolador. Sólo es necesario este chip y 4 condensadores electrolíticos de 22 microfaradios. El esquema está representado en la siguiente figura 4.5:

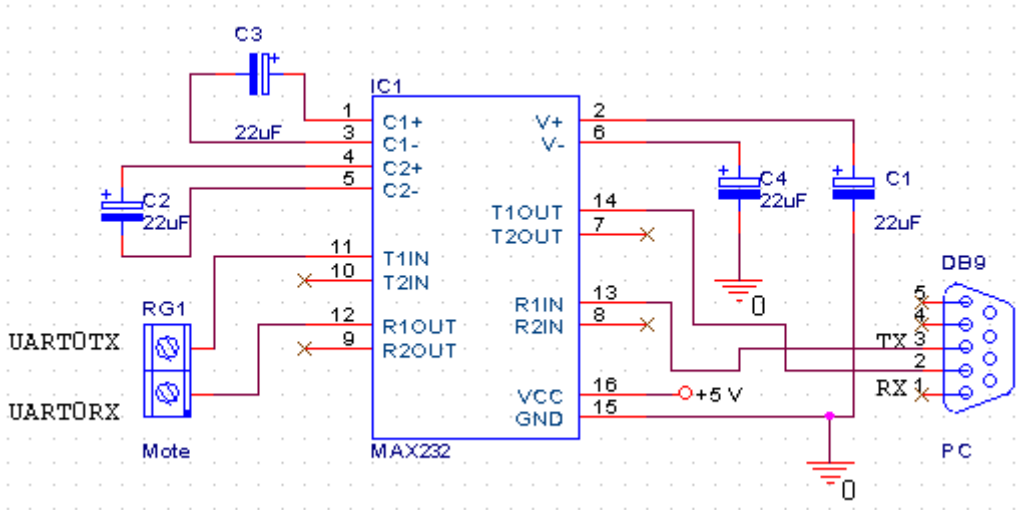


Figura 4.5: Conexión chip MAX 232 con el PC y el Mote

Los pines UART0RX y UART0TX del mote se encuentran en la ranura de expansión (figura 4.6) del mismo, PIN2 y PIN4 respectivamente. Con los cuales se hará las pertinentes conexiones con el chip MAX232.

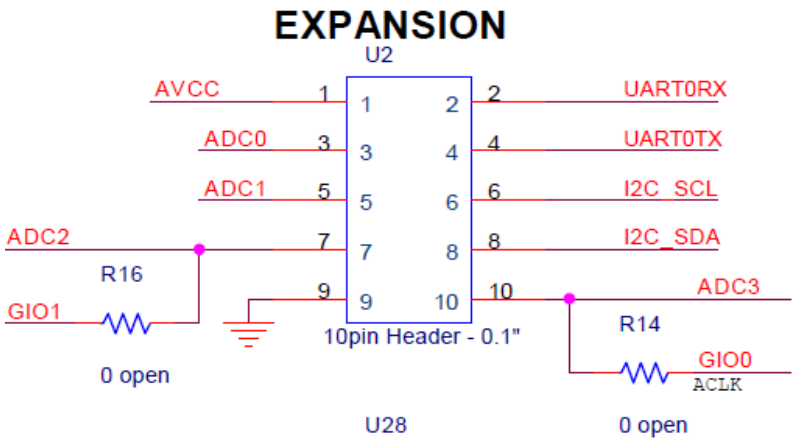


Figura 4.6: Bloque de expansión del Telosb

Cabe destacar, que la alimentación y la masa del MAX 232 será compartida con las del Mote-Portero. Para alimentar el MAX 232 se conectaría al pin 1 del bloque de expansión del Telosb, y la masa se conectaría con el pin 9 del mismo.

4.3 Acondicionamiento de la cerradura eléctrica

Cabe destacar que la cerradura eléctrica será utilizada para la apertura del portal del supuesto bloque de viviendas para permitir la entrada de las visitas, y que está conectada al mote principal.

Para llevar a cabo el acondicionamiento de la cerradura eléctrica se ha diseñado dos circuitos, donde uno de ellos es el encargado de la alimentación, tanto de la cerradura eléctrica, como de la alimentación del circuito auxiliar, cuya función será la activación de un relé para alimentar la cerradura eléctrica. En la figura 4.7 se puede ver el circuito final para el acondicionamiento de la cerradura eléctrica con el mote:

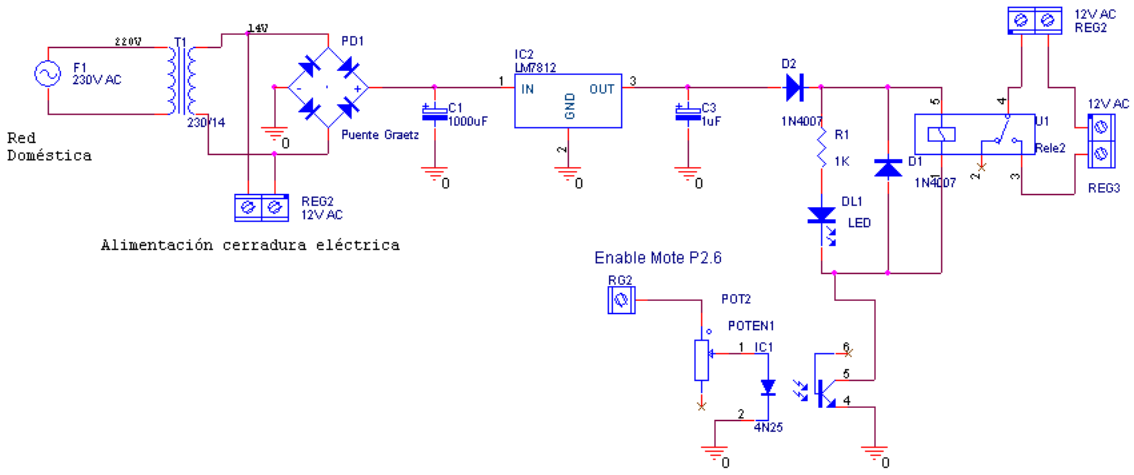


Figura 4.7: Esquemático de una Fuente de Tensión Regulada

A continuación se describirá cada circuito por separado:

4.3.1 Fuente de tensión regulada Lineal

En la figura 4.8 se puede observar el esquemático de una fuente de tensión regulada con una salida de 12 voltios, cuya finalidad es la alimentación del circuito que activa el relé así como la alimentación de la cerradura eléctrica.

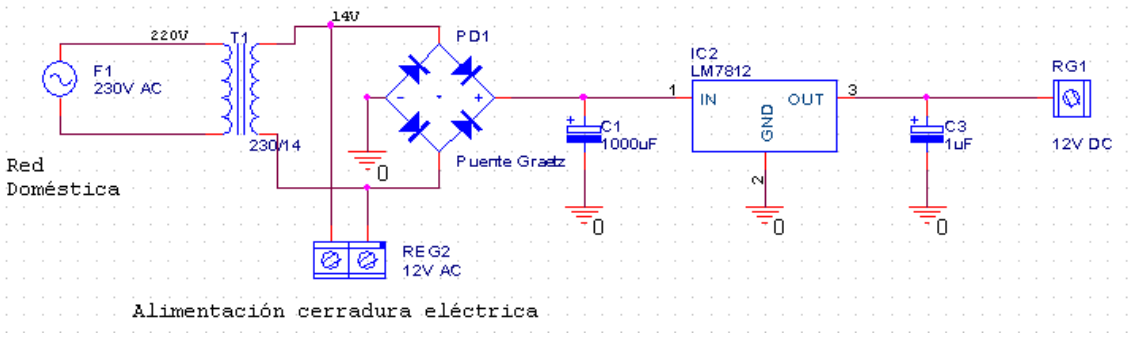


Figura 4.8: Esquemático de una Fuente de Tensión Regulada

Para la realización de este circuito se ha utilizado los siguientes componentes:

- Un trasformador con una relación de 230V/14V entre el voltaje de entrada y el de salida.
- Dos condensadores electrolíticos de 1000µF y de 1µF.
- Un circuito regulador L7812CV.
- Un puente de diodos.

La finalidad por la que se ha desarrollado este circuito es conseguir una fuente de alimentación con una salida estabilizada a 12 Voltios, basada en el circuito integrado L7812.

Para el montaje de este circuito se ha necesitado un transformador, con un secundario de una tensión de al menos 12 voltios de tensión alterna, también diodos para rectificar esta señal alterna, el circuito integrado mencionado y los condensadores que se encargarán de filtrar la señal de salida.

A continuación se detalla la funcionalidad de cada unos de los componentes de este circuito:

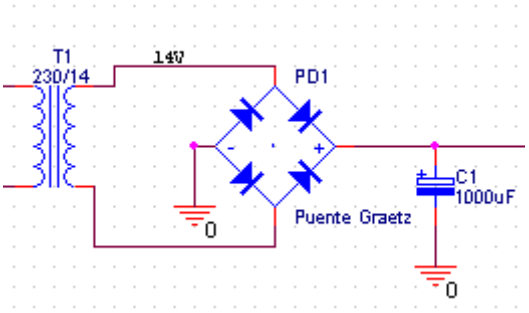


Figura 4.9: Esquema de un Puente de Graetz

El rectificador en puente de Graetz tiene la función de hacer que la corriente alterna no cambie su sentido de circulación, forzando la circulación de corriente en un solo sentido.

Como la corriente procedente del transformador es estabilizada en el puente de Graetz, y todavía no es una señal de corriente continua, capaz de alimentar circuitos que necesiten una señal continua estable, será necesario el filtrado de la señal.

Para el filtrado de la señal, se utilizarán filtros con la capacidad de almacenar energía eléctrica de los componentes reactivos (bobinas y condensadores). Los condensadores almacenan energía debido a su carga rápida a través de la pequeña resistencia directa de los diodos y la pierden cuando se descargan muy lentamente a través de la resistencia de salida, consiguiendo como resultado mantener una tensión prácticamente constante en extremos de ésta.

Una vez que es filtrada la señal, es el momento de insertar el circuito integrado regulador de tensión, el cual proporcionará una corriente continua estabilizada a 12 Voltios, que es la tensión de salida elegida para este montaje.

Existen diferentes tipos de circuitos reguladores de tensión, dependiendo de la señal estabilizada que proporcionan a la salida así, el 7812, que es el que se va a utilizar en este montaje, estabiliza la tensión a 12 Voltios, el 7805 a 5 Voltios, el 7824 a 24 Voltios, etc.

La corriente máxima de carga que soportan estos integrados es de 1 A, desconectándose el circuito en caso que la demanda de corriente sea mayor a 1 A.

Este circuito integrado se compone de los siguientes bloques, tal como se puede ver en la figura 4.4:

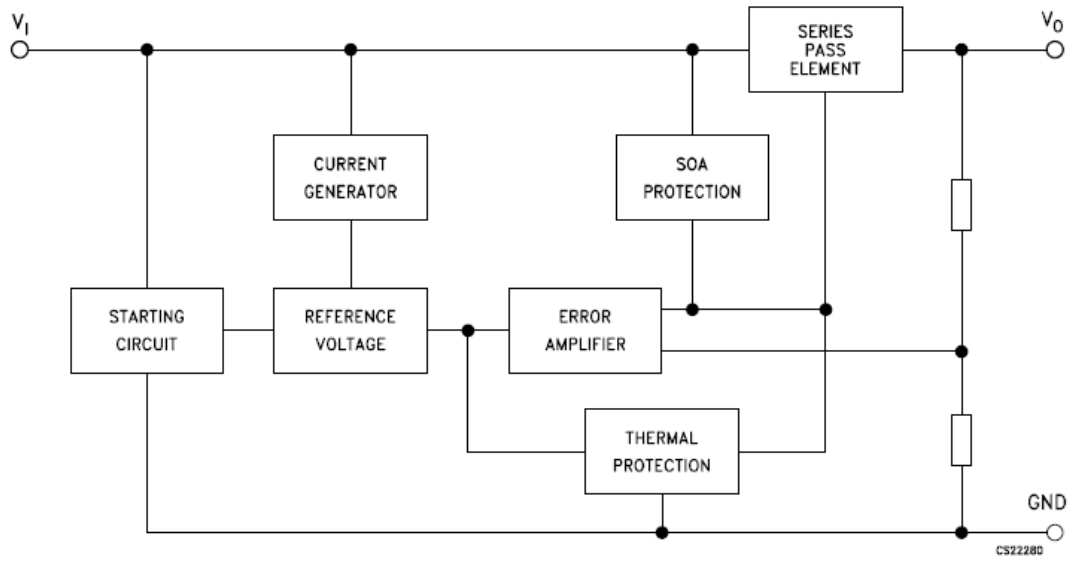


Figure 4.10: Bloque de diagramas del circuito L78xx

A continuación se explica, de forma detallada, la funcionalidad de cada uno de los bloques presentes en la figura 4.10

- **Circuito de arranque (***Starting Circuit)***.** Este circuito de protección inhibe la salida del regulador cuando la tensión Vi no supera en cierta cantidad (típicamente 2V) a la tensión nominal de salida.
- **Generador de corriente (***Current Generator***).** Proporciona una corriente constante al elemento de referencia, independientemente de la entrada y la salida.
- **Elemento de referencia (***Reference Voltage***).** Mantiene una tensión constante entre sus extremos y la envía al amplificador de error.
- **Amplificador de error (***Error Amplifier***).** Compara la tensión del elemento de referencia con una porción de la tensión de salida, obtenida del divisor de tensión formado por R1-R2. El resultado es amplificado y enviado al elemento de control.
- **Elemento de control (***Series Pass Element***).** Recibe la señal proveniente del amplificador de error y varía su caída de tensión interna en función de dicha señal. Es el elemento que soporta la diferencia de tensión entre Vi y Vo nominal.
- **Protección térmica (***Thermal Protection***).** Protección contra cortocircuitos. Si la intensidad de salida supera cierto nivel al elemento de control, se interrumpe la corriente de salida.

- **Protección de sobrecarga (***Soa Protection***):** Protege al elemento de control cuando el regulador se desconecta, permitiendo el paso de la corriente inversa que de otra forma dañaría a dicho elemento.

4.3.1.1 Regulador de Tensión Lineal L7812CV

A continuación se muestra la figura 4.2 de una tabla con las características eléctricas del L7812:

Símbolo	Parámetro	Condiciones del Test	Min.	Typ.	Max.	Unidad
V_0	*Output voltage*	$T_j = 25ºC$	11.5	12	12.5	V
V_0	*Output voltage*	$I_0 = 5mA\ to\ 1\ A,\ P_0 \leq 15W$ $V_1 = 15.5\ to\ 27\ V$	11.4	12	12.6	V
ΔV_0	*Line regulation*	$V_1 = 14.5\ to\ 30\ V, T_j = 25ºC$			120	mV
		$V_1 = 16\ V\ to\ 30\ V,\ T_j = 25ºC$			60	mV
ΔV_0	*Load regulation*	$I_0 = 5mA\ to\ 1.5A, T_j = 25ºC$			100	mV
		$I_0 = 250\ to\ 750mA, T_j = 25ºC$			60	mV
I_d	*Quiescent current*	$T_j = 25ºC$			6	mA
ΔI_d	*Quiescent current change*	$I_0 = 5mA\ to\ 1\ A$			0.5	mA
		$V_1 = 15\ to\ 30\ V$			0.8	mA
$\Delta V_0/\Delta T$	*Output voltage drift*	$I_0 = 5mA$		1.5		mV/ºC
eN	*Output noise voltage*	$B = 10\ Hz\ to\ 100\ KHz,$ $T_j = 25ºC$			40	µV/V_0
SVR	*Supply voltage rejection*	$V_1 = 15\ to\ 25\ V, f = 120Hz$	61			dB
V_d	*Dropout voltage*	$I_0 = 1\ A, T_j = 25ºC$		2	2.5	V
R_0	*Output resistance*	$f = 1KHz$		18		mΩ
I_{SC}	*Short circuit current*	$V_1 = 35\ V, T_j = 25ºC$		0.75	1.2	A
I_{SCP}	*Short circuit speak current*	$T_j = 25ºC$	1.3	2.2	3.3	A

Tabla 4.2: Características eléctricas del L7812

4.3.2 Circuito para la activación de la cerradura

En la figura 4.11 se puede observar el esquemático del circuito de acondicionamiento entre el mote y la cerradura eléctrica.

Los elementos que lo componen son los siguientes:

- Dos diodos 1N4007.
- Tres resistencias, una 10kΩ, otra de 1kΩ y otra de 330Ω.
- Un optoacoplador NPN 4n25.
- Un diodo LED.
- Un relé.

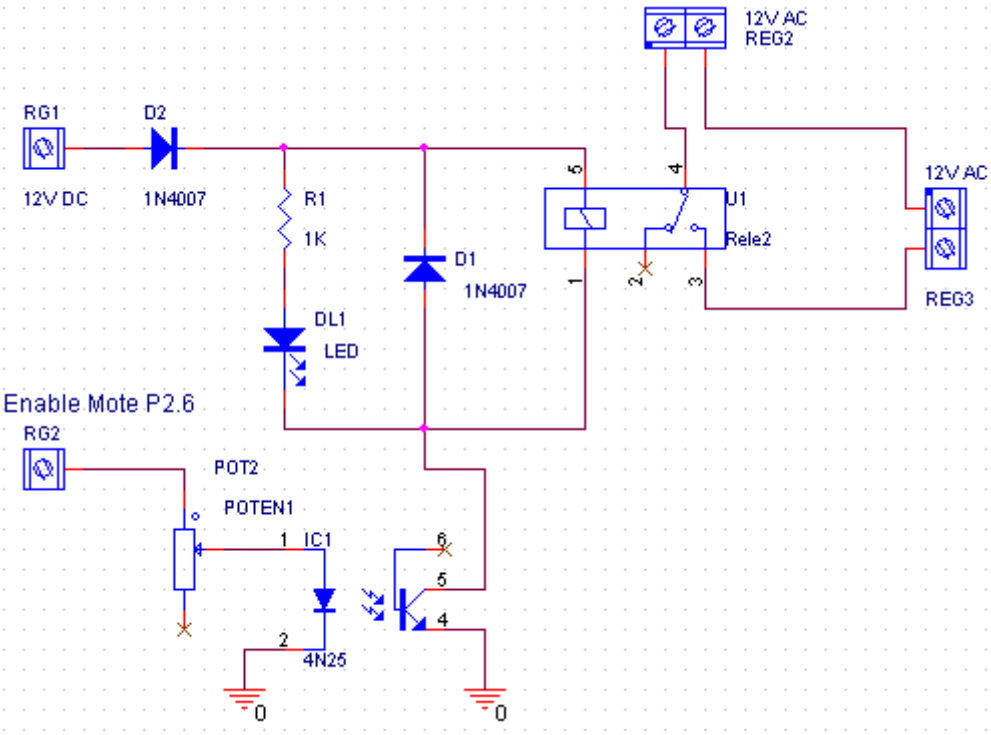


Figura 4.11: Circuito que activa la cerradura eléctrica

4.3.2.1 Relé

El relé [20] es un dispositivo electromecánico. Su funcionamiento es similar al de un interruptor controlado por un circuito eléctrico, en el que, por medio de una bobina y un electroimán se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independiente. Este dispositivo fue inventado por Joseph Henry en 1835.

Como el relé es capaz de controlar un circuito de salida de mayor potencia que el de entrada, se puede considerar como un amplificador eléctrico. De este modo ya se emplearon en telegrafía, haciendo la función de repetidores que generaban una nueva señal con corriente procedente de pilas locales a partir de la señal débil recibida por la línea. De ahí que se denomine como un dispositivo "revelador", del cuál procede su nombre "relé".

Actualmente, existen una gran variedad de relés comerciales, dependiendo del número de contactos, de la intensidad admisible por los mismos, tipo de corriente de accionamiento, tiempo de activación y desactivación, etc. Así que cuando son usados para controlar grandes potencias se les denomina como contactores en lugar de relés.

- Entre los **Relés electromecánicos** destacan los siguientes**:**

- **Relés de tipo armadura**: son los más antiguos y siguen siendo los más utilizados en multitud de aplicaciones. Al ser excitado, Un electroimán provoca

la basculación de una armadura, cerrando o abriendo los contactos dependiendo de si es NA o NC.

- **Relés de núcleo móvil**: están formados por un émbolo en lugar de una armadura. Como tiene mayor fuerza de atracción, se utiliza un solenoide para cerrar sus contactos. Es muy útil cuando hay que controlar altas corrientes.

- **Relé tipo *reed* o de lengüeta**: están formados por una ampolla de vidrio, con una serie de contactos en su interior, montados sobre delgadas láminas de metal. Estos contactos conmutan por la excitación de una bobina, que se encuentra alrededor de la mencionada ampolla.

- **Relés polarizados o biestables**: están compuestos de una pequeña armadura, solidaria a un imán permanente. El extremo inferior gira dentro de los polos de un electroimán, mientras que el otro lleva una cabeza de contacto. Cuando es excitado el electroimán, la armadura se mueve provocando el cierre de los contactos. Si se polariza al revés, el giro será en sentido contrario, abriendo los contactos ó cerrando otro circuito.

  - **Relé de estado sólido**

  Este dispositivo es usado para aplicaciones donde se presenta un uso continuo de los contactos del relé que en comparación con un relé convencional generaría un serio desgaste mecánico, además de poder conmutar altas corrientes y a más velocidad, a diferencia entre el rele electromecánico.

  - **Relé de corriente alterna**

  Es aquel que es excitada la bobina del con corriente alterna, donde el flujo magnético en el circuito magnético, también es alterno, produciendo una fuerza pulsante, con frecuencia doble, sobre los contactos.

  - **Relé de láminas**

  Está formado por un electroimán excitado con la corriente alterna de entrada que atrae varias varillas, que accionan los contactos, sintonizadas para resonar a sendas frecuencias de interés.

  En conclusión, la gran ventaja de los relés electromagnéticos es la completa separación eléctrica entre la corriente de accionamiento, la que circula por la bobina del electroimán, y los circuitos controlados por los contactos, lo que hace que se puedan manejar altos voltajes o elevadas potencias con pequeñas tensiones de control. También ofrecen la posibilidad de control de un dispositivo a distancia mediante el uso de pequeñas señales de control.

  En este caso, se activa la cerradura eléctrica alimentándola con una tensión alterna de 12V, y que es aislada de la tensión de alimentación del relé, que es de 12V en continua, y así conseguir la compatibilidad con la tensión en continúa del resto de los dispositivos utilizados.

4.3.2.2 Optoacoplador

El optoacoplador [21], nombrado también optoaislador o aislador óptico acoplado, es un dispositivo de emisión y recepción cuyo funcionamiento es parecido al de un interruptor, excitado mediante la luz emitida por un diodo LED que satura un componente optoelectrónico, normalmente en forma de fototransistor o fototriac. De esta forma se combinan en un solo dispositivo semiconductor, un fotoemisor y un fotorreceptor cuya conexión entre ambos es óptica. Es muy utilizado también para aislar eléctricamente a dispositivos muy sensibles, en este caso es utilizado para el aislamiento del TelosB ante posibles sobretensiones ante el circuito de alta potencia utilizado para la activación de la cerradura eléctrica.

La figura 4.12 muestra un optoacoplador formado por un LED y un fototransistor. Al cerrarse el interruptor S1 se establece una corriente que circula por la fuente V1 y la resistencia en serie con en el LED emisor. Si esta corriente proporciona un nivel de luz adecuado, al incidir sobre el fototransistor lo saturará, generando una corriente en R2. De este modo la tensión de salida será igual a cero con S1 cerrado y a V2 con S1 abierto.

Si varía el voltaje de entrada la cantidad de luz también lo hará, lo que significa que la tensión de salida cambia de acuerdo con la tensión de entrada. De esta forma el dispositivo puede acoplar una señal de entrada con el circuito de salida, aunque hay que tener en cuenta que las curvas tensión/luz del LED no son lineales, por lo que la señal puede distorsionarse. Se venden optoacopladores especiales con este fin, diseñados de con un rango de trabajo en el que la señal de salida sea casi idéntica a la de entrada.

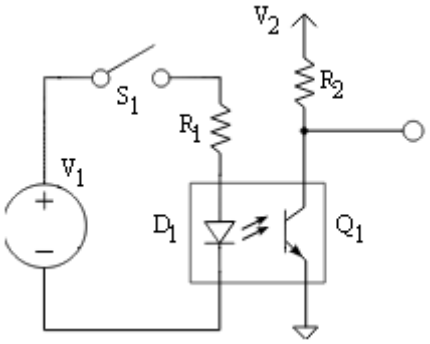


Figura 4.12: Ejemplo acondicionamiento optoacoplador

Como se ha comentado anteriormente, la ventaja principal, y por lo que es muy utilizado, es su capacidad aislamiento eléctrico entre los circuitos de entrada y salida. Así el único contacto entre ambos circuitos es un haz de luz. Esto es equivalente a una resistencia de aislamiento entre los dos circuitos del orden de miles de MΩ. Estos aislamientos son útiles en aplicaciones de alta tensión en las que los potenciales de los dos circuitos pueden diferir en varios miles de voltios.

En la aplicación realizada en este proyecto, la diferencia de voltaje de ambos circuitos es de unos pocos voltios, el circuito de entrada, el que está conectado al Telosb, es del orden de 3V, y el circuito de salida ,el que alimenta el réle que activa la cerradura eléctrica, es del orden de los 12V.

Existen varios tipos de Optoacopladores, entre los que cabe destacar el fototransistor, el fototriac y el fototriac de paso por cero. En este último mencionado, su

etapa de salida es un triac de cruce por cero, que tiene un circuito interno que conmuta al triac sólo en los cruce por cero de la fuente.

El modelo de optoacoplador utilizado como aislador acoplador entre los circuitos de baja y alta potencia de este proyecto es un 4n25. Elegido por ser ideal para el asilamiento entre circuitos con diferencial de voltaje no muy grande.

En conclusión, el uso del optoacoplador, ha sido fundamental para el aislamiento del mote y del circuito de alta potencia, y de esta forma evitando cualquier posible sobretensión o desperfecto de los dispositivos. De esta manera, queda totalmente aislada la fuente de tensión regulada con respecto al Telosb o el circuito de acondicionamiento de la CMUcam3, ambos circuitos de baja potencia. Así, cuando es activado el pin P2.6 del mote, circulará una intensidad suficiente que excitará el led interno del optoacoplador, activando este y, permitiendo el paso de la corriente por el fototransistor, y por consiguiente activando el relé, que estará alimentado con una tensión de 12 V en continúa, suministrada por la fuente regulada de tensión. Una vez que es activado el relé, éste permitirá el paso de la corriente procedente del puente de diodos Graetz, con una tensión de 12V en alterna, y que finalmente activará la cerradura eléctrica. De esta forma se consigue la apertura del portal del bloque de viviendas de un piso, o la entrada principal de una casa.

4.4 Acondicionamiento de los pulsadores

Como es lógico, cada vivienda dispondrá de su propio pulsador que irá conectado al Telosb, el cual está situado en el portal del bloque de viviendas, y que es el mismo que conecta con la cámara para la adquisición de las imágenes. En este proyecto se ha configurado dos pulsadores para dos viviendas, un pulsador por cada vivienda. Para esto, ha sido necesario realizar un pequeño circuito de acondicionamiento para el correcto funcionamiento de los pulsadores a la entrada del Telosb, situado en el portal, como se puede observar en la figura 4.13:

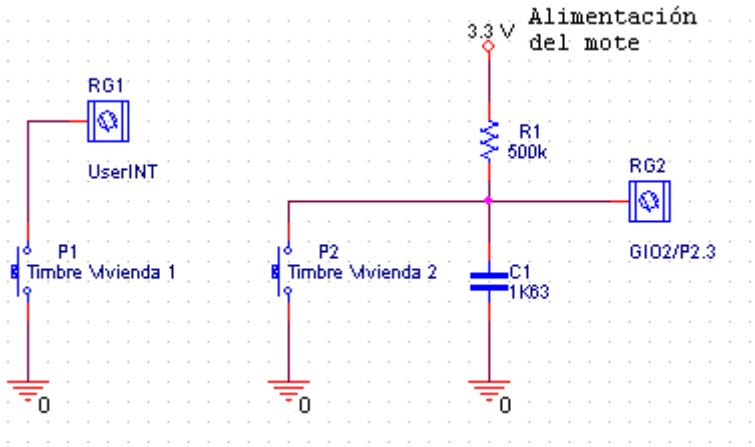


Figura 4.13: Esquemáticos de los pulsadores

Estas conexiones se ha realizad en la ranura de cabecera de 6 pines del mote, concretamente en el pin GIO2, que fue explicada anteriormente en el capítulo 3, y cuyo esquemático se representa en la figura 4.14.

El botón User es un pulsador interno del mote, pero hemos colocado otro en paralelo a este, con la finalidad de tener los dos pulsadores de las viviendas fuera del mote, para así hacer más fácil su manejo. El esquemático del botón User interno se muestra en la siguiente figura 4.15.

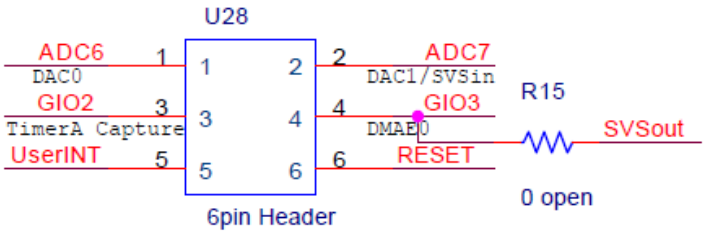


Figura 4.14: Esquemático de la cabecera de 6 pines interno del mote

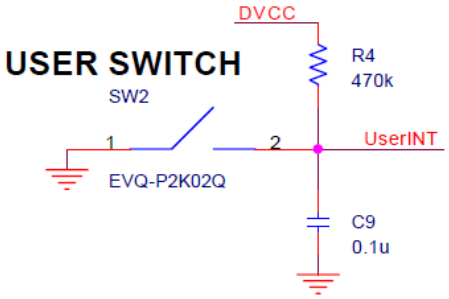


Figura 4.15: Esquemático del switch User interno del mote

4.5 Conclusiones

Con estos circuitos se consigue una adecuada sincronización entre el mote y la CMUcam3, así como un adecuado uso de los demás dispositivos, como la cerradura eléctrica, o el uso y posterior añadido de más pulsadores, (colocados en el portal del bloque), para el aviso a los propietarios de cada vivienda. Añadiendo la programación de los motes así como la aplicación java ejecutada en el ordenador se consigue el funcionamiento deseado. La programación realizada para el correcto funcionamiento de los circuitos explicados anteriormente se presenta en el siguiente capítulo 5.

Capítulo 5

DESCRIPCIÓN DEL SOFTWARE

5.1 Introducción

En este capítulo se presenta la programación que ha sido llevada a cabo para la realización de este proyecto, así como una explicación del código utilizado y su funcionalidad dentro de los programas.

En total se ha desarrollado dos programas y se ha utilizado un programa más, que venía implementado con TinyOS, llamado BaseStation, el cual se explicará más adelante, y que será ejecutado en el Mote-Vivienda de cada una de las viviendas, y que son totalmente distintos, tanto en el lenguaje de programación utilizado, como en el hardware que ha sido implementado. Sin embargo, estas aplicaciones tienen en común el objetivo por el que han sido programadas, y cuyas tareas están muy relacionadas entre sí, tanto que, dependen de unas de las otras para cumplir dicho objetivo funcional.

Su desarrollo ha sido necesario para el correcto funcionamiento de los dispositivos, así como sus circuitos acondicionados comentados en el capítulo 4, que han sido necesarios realizar para la conexión entre los distintos dispositivos empleados.

En primer lugar, se explicará la programación llevada a cabo para su implementación en el mote principal, que no es otro que el que estaría ubicado a la entrada del hipotético bloque de viviendas, y al que se hallaría conectado a la CMUcam3, así como a la cerradura electrónica para la apertura de la puerta de dicha entrada. Se expondrá partes de su código para su análisis y una explicación detallada del mismo en cuanto a su funcionalidad.

En segundo lugar, se explicará el programa ejecutado en el mote usado de base, BaseStation, y el que está conectado al PC de cada vivienda. Este actúa de puente entre la comunicación inalámbrica entre el mote principal y la comunicación serie emulada por vía USB con el PC.

Y por último, se expondrá la aplicación visual que sería ejecutada en el PC de cada vivienda, cuya función es la de mostrar al usuario la imagen captada por la CMUcam3, y que es transferida al ordenador por el mote principal a través de la radio, así como la apertura de la puerta de la entrada, por parte del mismo usuario.

A continuación se presenta los códigos empleados, y una descripción del funcionamiento de los mismos, así como el papel que juega dentro del conjunto que forma este proyecto.

5.2 Software del Mote-Portero

El lenguaje empleado en el software del Mote-Portero, o mote principal, es el NesC, el cual es ejecutado en el sistema operativo embebido empleado en este proyecto, y que ha sido presentado en el capítulo 3.

A continuación, en la figura 5.1 se muestra un diagrama con la vista estática de componentes nesC del software desarrollado.

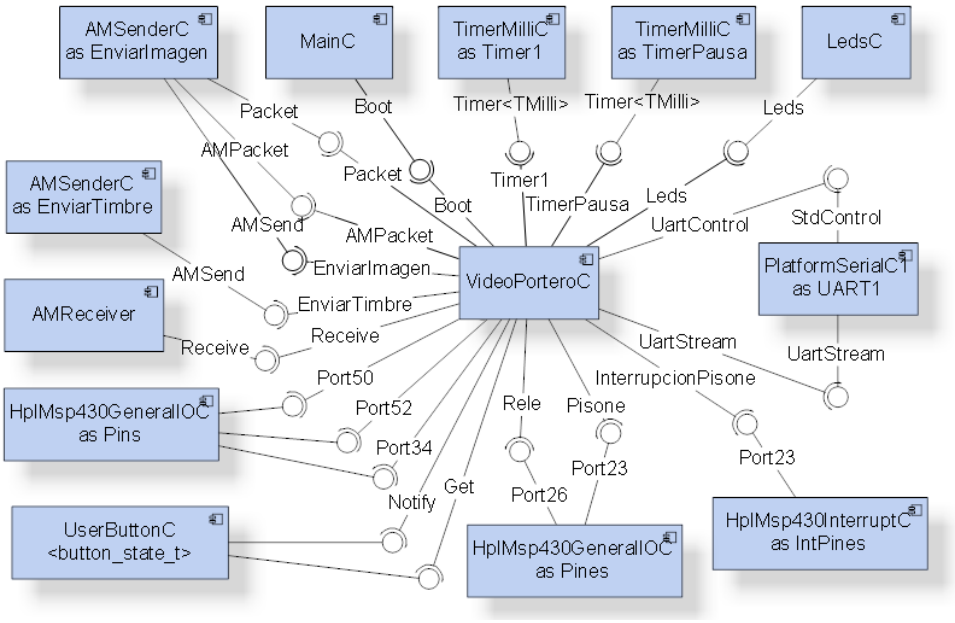


Figura 5.1: Vista estática de componentes nesC empleados.

En ella se muestra el componente asociado al módulo VideoPorteroC desarrollado y los componentes instanciados del sistema operativo para conseguir esta funcionalidad. Los componentes se comunican a través de las interfaces. Así, por ejemplo, el componente AMReceiver proporciona la interfaz Receive y el componente VideoPorteroC la utiliza.

En el siguiente listado 5.1 se presenta todos los #define y las bibliotecas que se incluyen en el archivo *VideoPorteroC.nc*, las librerías "*Timer.h*", "*EC250.h*", "*UserButton.h*" contienen los principales comandos para el correcto funcionamiento de la aplicación. La librería "*Timer.h*" incluye el código necesario para la ejecución de los

temporizadores utilizados, mientras que, la librería "*UserButton.h*" contiene el código prescindible para la habilitación del botón User interno del mote.

```
1  #include <Timer.h>
2  #include "EC250.h"
3  #include <UserButton.h>
4  #define TOS_DATA_LENGTH 109 /* Default TOS data length */
```

Listado 5.1: Librerías incluidas en el Software del Mote-Portero.

En el listado 5.2 se presenta las interfaces que han sido utilizadas en el módulo *VideoPorteroC* y que han sido necesarias, con anterioridad, realizar su hilado en el archivo *VideoPorteroAppC.nc*, cuyo código se muestra más adelante en el listado 5.3:

```
1  module  VideoPorteroC
2  {
3    uses interface Timer<TMilli> as Timer1;
4    uses interface Timer<TMilli> as TimerPausa;
5    uses interface Leds;
6    uses interface Boot;
7    uses interface Packet;
8    uses interface AMPacket;
9    uses interface AMSend as Send;         // Para enviar el comando y la respuesta recibida del sensor
10   uses interface AMSend as SendRing;  // Para enviar el comando y la respuesta recibida del sensor
11   uses interface Receive;
12   uses interface SplitControl as AMControl;
13   uses interface HplMsp430GeneralIO as P50;    //Pin a controlar de la fuente
14   uses interface HplMsp430GeneralIO as P52;    //Control del buffer.
15   uses interface HplMsp430GeneralIO as P36;
16   uses interface StdControl as UartControl;
17   uses interface UartStream;
18   uses interface Get<button_state_t>;
19   uses interface Notify<button_state_t>;
20   uses interface HplMsp430GeneralIO as Pisone;
21   uses interface HplMsp430Interrupt as InterrupcionPisone;  //Habilita un segundo Pulsador
22   uses interface HplMsp430GeneralIO as Rele;    // P2.6, salida del micro, activar rele
23 }
```

Listado 5.2: Interfaces del modulo VideoPorteroC.

En este código del Listado 5.2 se puede ver las interfaces de los dos temporizadores utilizados *Timer1*, y *TimerPausa*, que se deben implementar los controladores para los eventos *Timer1.fired()* y *TimerPausa.fired()*, los cuales se explicará más adelante. El interfaz *Leds* es para el uso de los Leds y el interfaz Boot para el inicio del sistema, también denominada interfaz de arranque.

Los interfaces *Packet* y *Receive*, son interfaces de comunicación básica. El interfaz Packet proporciona los métodos de acceso básico para el tipo de datos abstractos *message-t.* Esta interfaz proporciona comandos para borrar el contenido de un mensaje, obtener su longitud de carga útil, y obtener un puntero de su área de carga útil. El interfaz *Receive* proporciona la interfaz de mensajes básicos de acogida. Esta interfaz proporciona un evento para la recepción de mensajes. También ofrece, para mayor comodidad, los comandos para obtener la longitud de un mensaje de carga útil y obtener su puntero a la zona de carga de un mensaje.

Los interfaces *AMPacket* y *AMSend* son interfaces de mensajes activos. Como es muy común tener múltiples servicios con la misma radio para comunicarse, TinyOS proporciona el mensaje activo (AM) de la capa de acceso múltiple a la radio. El término "Tipo de AM" se refiere al campo que se utiliza para la multiplexación. Los tipos de AM son similares en función al campo de tipo de trama Ethernet, al campo de protocolos IP, y al puerto UDP, en el que todos ellos se utilizan para el acceso múltiple a un servicio de comunicación. Un paquete AM también incluye un campo de destino, que almacena una dirección AM para hacer frente al flujo de paquetes.

La interfaz *AMPacket* es similar a la interfaz *Packet*, proporciona los descriptores para el acceso básico AM para el tipo de dato abstracto *message_t*. Esta interfaz proporciona comandos para obtener la dirección AM de un nodo, el destino de un paquete AM, y el tipo de un paquete AM. Los comandos también están disponibles para establecer el destino de un paquete AM, el tipo y la comprobación de si el destino es un nodo local. La interfaz *AMSend*, ofrece el mensaje fundamental que activa el envío de la interfaz, y toma como destino la dirección AM en su comando de envío.

La dirección AM de un nodo se puede establecer durante la instalación, utilizando los comandos "*make telosb install,n",* donde "n" es la dirección que se establece al nodo. En los tres motes utilizados en este proyecto, se ha establecido al Mote-Portero la dirección 0, al resto de motes base que estarían ubicados en las viviendas se establecería las direcciones 1, 2, 3, 4, etc. Estas direcciones son necesarias para evitar, por ejemplo, que al pulsar el timbre para llamar a la vivienda 1 llegue el aviso también al resto de viviendas, o que cuando en una vivienda se quiera realizar captura de imágenes, estas no se envíen a las demás viviendas que no lo han solicitado.

La interfaz *SplitControl* definida como *AMControl* se utiliza para la manipulación de la radio. La inicialización de la radio debe comenzar cuando se inicia el sistema, por lo que se llama al AMControl.start dentro del evento *Boot.booted*. La radio no puede ser utilizada hasta que haya completado la puesta en marcha, para ello se mueve la convocatoria a *AMControl.startDone*. En el listado 5.3 se muestra los controladores de eventos de *AMControl.startDone* y *AMControl.stopDone*:

```
1   event void AMControl.startDone(error_t err)
2    {
3      // Si se ha iniciado la radio se inician los temporizadores. En caso contrario se intenta de nuevo.
4      if (err != SUCCESS)
5      {
6        call AMControl.start();
7      }
8                FuncionEnviarRing(1);
9    }
10
11
12   event void AMControl.stopDone(error_t err) {}
```

Listado 5.3: Eventos de *AMControl.startDone y AMControl.stopDone.*

La función *FuncionEnviarRing(1)* es llamada para el envío del mensaje timbre al pulsar uno de los pulsadores conectados al Mote-Portero y que más adelante se explicará con más detalle.

La interfaz *StdControl* es utilizada para todos aquellos componentes que requieren la inicialización o pueden ser a pagados, por lo que deben proporcionar esta interfaz. *Start()* y *Stop()* significa encendido y apagado respectivamente. En este caso es

llamada como *UartControl*, y permite la habilitación y deshabilitación de la *UART*, que es donde está conectado el mote con la CMUcam3.

La interfaz *UartStream* es llamada para la transmisión y recepción de datos a través de la comunicación serie establecida entre la CMUcam3 y el Mote-Portero por medio de la Uart. Esta interfaz es utilizada para el envío de comandos a la CMUcam3, así como para obtener sus respuestas, es decir, si se desear obtener una imagen de la CMUcam3 es necesario enviarle un comando, establecido dentro de su sistema. A continuación se muestra, en el listado 5.4, una parte del código para el envío de un comando cualquiera, en este caso con el que se obtiene la imagen de la CMUcam3.

```
1   comando[0] = 0x53;  // S
2   comando[1] = 0x4A;  // J
3   comando[2] = 0x0A;  // Espacio
4   comando[3] = 0x0D; // Salto de Línea
5    longitudComando = 4;
6   modo = 2;
7                   while(FAIL == call UartControl.start());
8                   call UartStream.enableReceiveInterrupt();
9   datosRecibidos = 0;
10  ant3 = ant2 = ant1 = ant = 0;
11  for(iteracion = 0;iteracion < 8000;iteracion++) recibidos[iteracion] = 0;
12  call UartStream.send(comando, longitudComando);
```

Listado 5.4: Envió de un comando por UART a la CMUcam3.

El comando enviado se recoge en 4 punteros en hexadecimal. El comando para hacer una imagen es el SJ seguido de un espacio y un salto de línea como se puede ver en las líneas del 1 al 4. La variable longitudComando indica el número caracteres a enviar. La línea 6 indica la instrucción siguiente que debe seguir. El *while* que se extiende de la línea 7 a la 8 deshabilita la interfaz UarStream en el caso que se produzca un fallo al iniciar la Uart. En la línea 9, 10 y 11 se ponen los datos recibidos anteriormente, y que puedan ocasionar algún error para el envío del nuevo comando, a cero. Y por último, en la línea 12 se llama a la interfaz UartStream para que envíe el comando, así como su longitud recogido en la línea 5.

Una vez enviado el comando para solicitar una imagen, se implementa el siguiente código (modo ==2) mostrado en el listado 5.5, para recibirla.

```
1   if ((ant2 == 0x41) && (ant1 == 0x43) && (ant == 0x4B) && (byte == 0x0D) && (modo == 2))
2     {
3       datosRecibidos = 0;
4      modo = 3;
5     }
6   else if ((ant3 == 0x45) && (ant2 == 0x4E) && (ant1 == 0x44) && (ant == 0x0D) && (byte == 0x3A) && (modo ==
7   3))
8     {
9       call UartStream.disableReceiveInterrupt();
10      call UartControl.stop();
11      posicion = 0;
12
13              if ((datosRecibidos - posicion) > 109) bytes = 109;
14              else bytes = datosRecibidos - posicion;
15
16              if (posicion < datosRecibidos)
17              {
18                      FuncionEnviarDatos(posicion, bytes);
19                      posicion = posicion + bytes;
20              }
```

```
21    }
```

Listado 5.5: Código para procesar la respuesta recibida por parte de la CMUcam3.

Como se puede observar, se establece una condición que depende del tipo de respuesta que haya recibido por parte de la cámara, recogida en un array, y que, en el listado 5.4, habíamos asignado a cero para el envío del comando. Si el comando enviado es correcto, la cámara da una respuesta de confirmación "ACK" que es recogida por el array, y borra esta respuesta para no multiplexarla con los datos de la imagen que recibe a continuación en el modo = 3. Además, en esta condición de la línea 6, se debe cumplir que la imagen recibida contenga los siguientes caracteres para tomarla como correcta, "END :" y que es comparado con el array de la condición. Una vez recibido los datos de la imagen, se interrumpe la recepción de la Uart, como se ve en la llamada de la línea 9, se desactiva la Uart, como se muestra en la línea 10, y la posición se vuelve a cero para futuras recepciones (línea 11). La condición, de la línea 13 a la 20, empaqueta los datos recibidos en paquetes de 109 bytes cada uno, hasta recoger todos los datos recibidos por la Uart, donde posteriormente son recogidos en una función denominada *FuncionEnviarDatos*, para su posterior envío por radio a la interfaz visual de la vivienda en la que ha sido solicitada dicha imagen, y en la cual será procesados estos datos para la visualización de dicha imagen.

Cabe destacar, que el código mostrado en el listado 5.4 se presenta también dentro de un condicional como el del código del listado 5.5. En la práctica es necesario además, que antes de enviar el comando de captar una imagen a la CMUcam3 es necesario antes enviarle un salto de línea para que se sincronice con el Mote-Portero. Lógicamente este comando también iría dentro de un condicional y asignándole un modo como los del listado 5.4 y 5.5. En el listado 5.6 siguiente se muestra el tipo de comando necesario para la sincronización con la CMUcam3:

```
1   if ((ant3 == 0x20) && (ant2 == 0x63) && (ant1 == 0x36) && (ant == 0x0D) && (byte == 0x3A) && (modo == 1))
2   //Esperando la respuesta 1 del reset
3           {
4                   comando[0] = 0x0D;
5                   longitudComando = 1;
6                   modo = 2;
7                   datosRecibidos = 0;
8                   ant3 = ant2 = ant1 = ant = 0;
9                   call UartStream.send(comando, longitudComando);
10          }
11
```

Listado 5.6: Condicional para el envío de un salto de línea (esperando un reset de la CMUcam3).

El evento en el cual todas estas condiciones presentadas en los listados 5.4, 5.5 y 5.6 se muestra a continuación en el listado 5.7:

```
1   async event void UartStream.receivedByte(uint8_t byte)
2    {
3      uint16_t bytes;   // variable bytes necesaria para realizar paquetes  de  109 bytes
4      recibidos[datosRecibidos] = byte;   // Empaquetamiento de los datos recibidos
5      datosRecibidos = datosRecibidos + 1;  // Incremento de los datos recibidos
6
7   if ((ant3 == 0x20) && (ant2 == 0x63) && (ant1 == 0x36) && (ant == 0x0D) && (byte == 0x3A) && (modo == 1))
8   //Esperando la respuesta 1 del reset
9           {
10                  // Se envía un salto de línea  (Código del listado  5.6)
11                  //  En este caso la CMUcam3 responde además con un ACK muestra su versión
12          }
13  else if ((ant3 == 0x41) && (ant2 == 0x43) && (ant1 == 0x4B) && (ant == 0x0D) && (byte == 0x3A) && (modo
```

```
14  == 2)) //Esperando la respuesta del CR
15              {
16                          // Se envía un salto de línea  (Código del listado  5.6) solo ante respuesta ACK
17              }
18  else if ((ant2 == 0x41) && (ant1 == 0x43) && (ant == 0x4B) && (byte == 0x0D) && (modo == 3))    //Eliminar
19  ACK al recibir la imagen
20     {
21                          // Se elimina  toda respuesta  de confirmación de la CMUcam3 (listado 5.5 modo=2)
22     }
23  else if ((ant3 == 0x45) && (ant2 == 0x4E) && (ant1 == 0x44) && (ant == 0x0D) && (byte == 0x3A) && (modo ==
24  4))
25     {
26                         // Se almacena los datos de la imagen para su posterior envío por radio (listado 5.5
27                          // modo =4)
28     }
29     ant3 = ant2;
30     ant2 = ant1;
31     ant1 = ant;
32     ant = byte;
33   }
```

Listado 5.7: Código del evento UartStream.

Entre las líneas 29 a la 32, del extracto presentado en el listado 5.7, se actualizan las variables auxiliares necesarias para detectar las secuencias.

En el listado 5.8 se presenta el código correspondiente de la *FuncionEnviarDatos*:

```
1   void FuncionEnviarDatos(uint16_t elemento, uint8_t bytes)
2    {
3     uint8_t i;
4
5     if (!EnviarDatosBusy)
6     {
7      MensajeSalida* btrpkt = (MensajeSalida*)(call Packet.getPayload(&PaqueteSalida, sizeof(MensajeSalida)));
8
9      btrpkt->longitud = bytes;
10     for (i = 0;i < bytes;i++) btrpkt->cmdRsp[i] = recibidos[i+elemento];
11
12     //Se envía a todos los nodos que tengan el mismo tipo AM con que se envía.
13     if (call Send.send(casa, &PaqueteSalida, sizeof(MensajeSalida)) == SUCCESS)
14     {
15      EnviarDatosBusy = TRUE;
16       //call Leds.set(6);
17     }
18    }
19   }
```

Listado 5.8: Código de la Función *FuncionEnviarDatos.*

Como se puede observar, son declaradas tres variables, una de 16 bytes denominada "*elemento"* y otras dos de 8 bytes, denominadas "*bytes"* y "*i"*. Entre las líneas 5 a la 17 se declara una condición donde se carga, mediante un *Payload*, en el MensajeSalida los paquetes recibidos por la Uart, así como la cantidad de los mismos en dos variables, "*cmdRsp*" y "*longitud*" respectivamente. Como se puede ver, esta condición se cumple mientras este negado la función *EnviarDatosBusy.*

Una vez procesados todos los paquetes y contabilizado el número de los mismos, es enviado a la dirección del nodo base indicado mediante la variable "casa" (línea 13), que recoge la dirección del usuario que ha solicitado la captura de imagen. Mediante la condición de la línea 13 y una vez cumplida con éxito, se vuelve a poner la variable

EnviarDatosBusy a verdadero, para evitar un nuevo procesado de paquetes durante el envio. Cabe destacar que, en el evento *Boot.booted(,) EnviarDatosBusy* está declarado a *FALSE*.

El evento *Send* es declarado en la interfaz *sendDone*, como se muestra en el listado 5.9:

```
1  event void Send.sendDone(message_t* msg, error_t error)
2   {
3     uint16_t bytes;
4     if (&PaqueteSalida == msg)
5     {
6       EnviarDatosBusy = FALSE;
7               if ((datosRecibidos - posicion) > 109) bytes = 109;
8               else bytes = datosRecibidos - posicion;
9
10              if (posicion < datosRecibidos)
11              {
12                      FuncionEnviarDatos(posicion, bytes);
13        posicion = posicion + bytes;
14              }
15              else       //Ya se ha enviado todo
16              {
17              }
18    }
19  }
```

Listado 5.9: Declaración del *Send* en la interfaz *senDone*.

Esté código es muy parecido al explicado en los listados 5.4 y 5.6, lo único que cambia es que ha sido introducido dentro de un condicional que comprueba que no se haya producido ningún error, desde el envío del comando a la CMUcam3, hasta su respuesta con la imagen para su posterior reenvió a la interfaz gráfica ejecutada en el PC de la vivienda solicitante.

Es necesario realizar un evento del *UartStream* para indicar la finalización de la recepción de datos. En listado 5.10 se muestra dicho evento:

```
1  async event void UartStream.receiveDone(uint8_t* buf, uint16_t len, error_t error) {}
```

Listado 5.10: Evento que indica la finalización del uso de la Uart.

Las interfaces *Get* y *Notify* proporcionan los métodos de acceso básico para el tipo de datos abstractos *button_state_t*, es decir, proporciona los comandos para el uso del botón *User* interno del mote. En el listado 5.11 se muestra el evento asociado de dicha interfaz:

```
1  event void Notify.notify( button_state_t state )
2   {
3               if ( state == BUTTON_PRESSED )
4               {
5     call Leds.led0Toggle();
6                FuncionEnviarRing(1);
7    }
8               else if ( state == BUTTON_RELEASED )
9               {
10              }
11  }
```

Listado 5.11: Evento asociado a la interfaz Notify.

Como se puede observar, en este evento se ha utilizado una condición, en la cual establece que, al mantener pulsado el botón User llame a la *FuncionEnviarRing(1),* que enviaría el mensaje timbre a la vivienda 1, además de notificarlo mediante el encendido del Led 0 del Mote-Portero. Cuando se deja de presionar no hace nada, deja de enviar el mensaje timbre.

La interfaz *HplMsp430GeneralIO* proporciona la habilitación de los pines de entrada y salida E/S del Mote-Portero. En este proyecto ha sido necesaria la habilitación de dos pines E/S, uno para la conexión de un segundo pulsador, el cual cumple la función de timbre para una segunda vivienda, ya que solo disponemos del botón User interno del Mote-Portero, y el otro pin E/S para la apertura de la cerradura eléctrica de la puerta de entrada del hipotético bloque de viviendas.

Cabe destacar, que en este proyecto solo se ha configurado dos botones para dos viviendas, uno el botón User y otro el pulsador auxiliar, comentado anteriormente. En el supuesto caso que hubiera más viviendas, se podría añadir más botones realizando las modificaciones pertinentes.

La interfaz *HplMsp430GeneralIO,* denominada como Pisone, junto con la interfaz *HplMsp430GeneralInterrupt,* denominada como *InterrupcionPisone* habilita este segundo pulsador. A continuación, en el listado 5.12 se muestra el código del evento asociado a este pulsador:

```
1  async event void InterrupcionPisone.fired( )
2   {
3            call InterrupcionPisone.clear();
4            call Leds.led1Toggle();
5            FuncionEnviarRing(2);
6   }
```

Listado 5.12: Evento de interrupción del pulsado auxiliar.

Al ejecutarse este evento, se introduce un 0 para la activación de este pin, ya que se trata de un pulsador a masa, como se puede observar en la línea 3. En la línea 4 y 5, al pulsar este segundo pulsador se enciende el Led1 y ejecuta la función FuncionEnviarRing(2), el cual envía un mensaje timbre a la segunda vivienda correspondiente a este nuevo pulsador. La función FuncionEnviarRing(1) y la función FuncionEnviarRing(2) solo se diferencian por la dirección del nodo a la que es enviado el mensaje timbre, más concretamente, la variable casa posee diferente valor según la dirección del nodo al que va dirigido el mensaje timbre. A continuación, en el listado 5.13 se muestra el código empleado en dicha función:

```
1  void FuncionEnviarRing(uint8_t numpiso)
2   {
3    if (!EnviarRingBusy)
4    {
5        MensajeTimbre* btrpkt = (MensajeTimbre*)(call Packet.getPayload(&PaqueteRing,
6  sizeof(MensajeTimbre)));
7        btrpkt->numpulsador = numpiso;
8        casa = btrpkt->numpulsador;
9        //Se envía a todos los nodos que tengan el mismo tipo AM con que se envía.
10       if (call SendRing.send(numpiso, &PaqueteRing, sizeof(MensajeTimbre)) == SUCCESS)
11           {
12               EnviarRingBusy = TRUE;
13           }
14   }
15  }
```

Listado 5.13: Función FuncionEnviarRing para el envío del mensaje timbre por radio.

Se presenta otro condicional, que mientras esté negada la variable booleana *EnviarRingBusy*, se cargue el mensaje de timbre en un payload, denominado MensajeTimbre, y enviado por radio al nodo correspondiente al pulsador timbre presionado, cuya dirección es recogida en la variable "*casa"* y asignado a la variable "*numpiso".* Una vez que se haya enviado el mensaje timbre, y siempre que haya resultado un éxito, se vuelve a poner el booleano EnviarRingBusy a TRUE (línea 12). La variable "numpulsador" contiene la dirección del nodo destino, y que es declarada en el archivo de cabecera EC250.h, ya que es una variable compartida con la aplicación Java ejecutada en el PC. Más adelante, se mostrará y explicará el código del archivo de cabecera *EC250.h.*

En el listado 5.14 se muestra la inicialización del pulsador auxiliar:

```
1  event void Boot.booted()
2   {
3            call Pisone.makeInput();  // Habilita el Puerto 2.3 del microcontrolador como entrada
4            call InterrupcionPisone.enable();
5            call InterrupcionPisone.edge( FALSE );
6   }
```

Listado 5.14: Inicialización del segundo pulsador auxiliar.

Igual que sucedía con el Mensaje de Salida, esta función ha de ir acompañada de un evento tipo *sendDone*, denominada como *SendRing*, y cuyo código se muestra a continuación en el listado 5.15:

```
1  event void SendRing.sendDone(message_t* timbre, error_t error)
2   {
3     if (&PaqueteRing == timbre)
4     {
5       EnviarRingBusy = FALSE;
6     }
7   }
```

Listado 5.15: Envento de la función SendRing.

Una vez que se ha enviado el PaqueteRing que contiene el mensaje timbre, la variable booleana EnviarRingBusy vuelve a ponerse a FALSE, para que esté el programa listo para la próxima vez que se vuelva a pulsar el timbre, y volver a repetir la operación.

La interfaz *HplMsp430GeneralIO* declarada como Rele proporciona la habilitación del pin del puerto 2.6 del microcontrolador interno del Mote-Portero como salida, y de esta manera activar la cerradura eléctrica del portal del bloque de las viviendas. Para llevar esto a cabo, es necesario su inicialización en la interfaz de arranque:

```
1  event void Boot.booted()
2   {
3            call Rele.makeOutput();  // makeOutput para establecer este pin como salida
4   }
```

Listado 5.16: Inicialización del Pin P2.6 del telosb.

La llamada para la activación de la cerradura eléctrica sería un "*call Rele.set();*" y para desactivarlo se llamaría a "*call Rele.clr()*". Mediante un temporizador "Timer", se puede programar la activación de la cerradura eléctrica durante el tiempo necesario para que el visitante le dé tiempo entrar al bloque de viviendas. Más adelante se

mostrará donde realizan estas llamadas dentro del programa y como se ha configurado dicho Timer.

Para completar y entender, como el usuario de una determinada vivienda, a través de la interfaz gráfica ejecutada en el PC, que se haya conectado a uno de los Motes-Base, envía el comando para capturar la imagen a la CMUcam3, se muestra el siguiente código, recogido en el listado 5.17 y, el cual, se explicará a continuación:

```
1  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len)
2   {
3     MensajeEntrada* btrpkt;
4     if (len == sizeof(MensajeEntrada))
5     {
6               btrpkt = (MensajeEntrada*)payload;
7               call Leds.set(1);
8               casa = btrpkt->piso;
9               icomando = btrpkt->comando;
10              //Comando 1...cambiar el modo vídeo
11              //comando 2...abrir la puerta
12              if(icomando == 1)
13              {
14
15                                      if ( modovideo)
16                                      {
17                                                              //inicializar la cámara
18                                                              //1º resetear la cámara
19                                                              modovideo = FALSE;
20                                                              comando[0] = 0x52;
21                                                              comando[1] = 0x53;
22                                                              comando[2] = 0x0A;
23                                                              comando[3] = 0x0D;
24                                                              tempo = 1;
25                                                              longitudComando = 4;
26                                                              modo = 1;
27                                                              while(FAIL == call UartControl.start());
28                                                              call UartStream.enableReceiveInterrupt();
29                                                              call Leds.set(3);
30                                                              datosRecibidos = 0;
31                                                              ant3 = ant2 = ant1 = ant = 0;
32                                                              call UartStream.send(comando, longitudComando);
33                                      }
34                                      else if (!modovideo)
35                                      {
36                                                      modovideo = TRUE;
37                                                      call TimerPausa.stop();
38                                      }
39
40              }
41              else if(icomando == 2)
42              {
43                              call Rele.set();
44                              call Leds.set(5);
45                              call Timer1.startPeriodic(4000);
46                              call TimerPausa.stop();
47              }
48    }
49    return msg;
50  }
```

Listado 5.17: Sentencias nesC ejecutadas en respuesta a la recepción de un mensaje.

Este código, presentado en el listado 5.17, está recogido en el evento de la interfaz Receive, es decir, la interfaz que proporciona los recursos necesarios para la recepción de datos a través de la comunicación inalámbrica del mote. Esto permite la comunicación y, lo más importante, permite al usuario interactuar a través de la interfaz gráfica ejecutada en el PC con el mote. Mediante el *payload* contenido en el mensaje tipo *MensajeEntrada*, (declarado en la línea 3), se obtienen las instrucciones enviadas desde la vivienda solicitante. En este caso, estas instrucciones contienen dos posibles comandos, (denominados como *icomando* para evitar confusiones con los comandos enviados por la Uart a la CMUcam3), y que son implementados dentro de un condicional, ante la carga del *MensajeEntrada*. Una vez determinada la dirección del nodo de la vivienda solicitante, línea 8, se procede a determinar el tipo de comando recibido a través de una nueva condición, contenido entre la línea 12 y la línea 47, y se procede a su ejecución, es decir, si "*icomando* = 1" se abre una nueva condición la captura de una imagen, o este caso para activar el modo video (varias imágenes seguidas), y si "icomando = 2", se procede a la activación del pin de salida del puerto 2.6, el cual activa el rele para alimentar la cerradura eléctrica, a la vez que llama a Timer1, el cual se establece que dure durante cuatro segundo, y cuyo evento asociado después de transcurridos los 4 segundos desactivará el relé. En el listado 5.18 se muestra el evento asociado al Timer1:

```
1  event void Timer1.fired()
2   {
3            call Rele.clr();
4            call Leds.set(2);
5   }
```

Listado 5.18: Evento asociado al Timer1.

En la línea 8 del listado 5.17, "*icomando"* toma el valor del comando recibido por la radio, el cual se debe indicar dentro del archivo de cabecera EC250.h, al ser una variable compartida con la aplicación visual ejecutada en el PC.

Como se ha podido observar en el listado 5.17, el comando enviado por el Mote-Portero a la CMUcam3, comprendido entre las líneas 25 a la 28, es para realizar un reset a la CMUcam3, para posteriormente volver a realizar la resincronización con ella y volver a capturar nuevas imágenes.

Para conseguir el modo video se ha configurado un nuevo Timer, el TimerPausa, el cual, transcurrido un segundo realiza una nueva imagen, pasa otro segundo y otra nueva imagen, así consecutivamente hasta volver a establecer la variable booleana "*modovideo"* a FALSE.

A continuación se muestra, en el listado 5.19, el código del archivo de cabecera EC250.h:

```
1  ifndef EC250_H
2  #define EC250_H
3  #define SIZE 109 // Tamaño de los paquetes que contienen los datos de la imagen captada por la CMUcam3
4
5  enum
6  {
7   AM_MENSAJEENTRADA = 10,          // Tipo AM asociado al componente que recibe mensajes del programa
8                                    // Java con el comando
9   AM_MENSAJESALIDA = 20,  // Tipo AM asociado al componente que envía mensajes con la respuesta del
10                          // comando
11  AM_MENSAJETIMBRE = 30    // Tipo de AM asociado al componente que envía mensajes para que suene el
```

```
12                                  // timbre
13  };
14
15
16  // Mensajes procedentes del programa Java
17  typedef nx_struct MensajeEntrada
18  {
19    nx_uint8_t piso;    //1 byte para identificar el comando a enviar
20    nx_uint8_t comando; //Identificar el comando y procesar adecuadamente
21  } MensajeEntrada;
22
23
24  // Payload del mensaje que enviará al PC con el comando enviado más la respuesta recibida de la CMUcam3
25  //typedef nx_struct MensajeSalida
26  {
27    nx_uint8_t longitud;                  // Para definir el tamaño de la respuesta asociada a cada comando
28    nx_uint8_t cmdRsp [109];              // Comando enviado más la respuesta
29
30  } MensajeSalida;
31
32  typedef nx_struct MensajeTimbre
33  {
34    nx_uint16_t numpulsador;              // Para definir el tamaño de la respuesta asociada a cada comando
35  } MensajeTimbre;
36
37  #endif
```

Listado 5.19: Archivo de cabecera EC250.h.

En el listado anterior se ha definido las constantes necesarias y los campos asociados a cada mensaje que se envía inalámbricamente por la red.

Como ya se comentó en el capítulo 2, para la ejecución adecuada de las interfaces, es necesaria su implementación dentro del archivo *VideoPorteroApp.C*, cuyo código se muestra en el listado 5.20:

```
1   #include <Timer.h>
2   #include "EC250.h"
3   #include <UserButton.h>
4
5   configuration BlinkToRadioAppC
6   {
7   }
8   implementation
9   {
10    components MainC, BlinkToRadioC as App, LedsC;
11    components new TimerMilliC() as Timer1;
12    components new TimerMilliC() as TimerPausa;
13    components ActiveMessageC;
14    components new AMSenderC(AM_MENSAJESALIDA) as EnviarImagen;
15    components new AMSenderC(AM_MENSAJETIMBRE) as EnviarTimbre;
16    components new AMReceiverC(AM_MENSAJEENTRADA);
17    components HplMsp430GeneralIOC as Pins;
18    components PlatformSerialC1 as UART1;
19    components UserButtonC;
20    components HplMsp430GeneralIOC as Pines, HplMsp430InterruptC as IntPines;
21
22    App.Boot -> MainC;
23    App.Timer1 -> Timer1;
24    App.TimerPausa -> TimerPausa;
25    App.Leds -> LedsC;
26    App.Packet -> EnviarImagen;
27    App.AMPacket -> EnviarImagen;
```

```
28    App.Send -> EnviarImagen.AMSend;
29    App.SendRing -> EnviarTimbre.AMSend;
30    App.AMControl -> ActiveMessageC;
31    App.Receive -> AMReceiverC;
32    App.P50 -> Pins.Port50;
33    App.P52 -> Pins.Port52;
34    App.P36 -> Pins.Port34;
35    App.UartControl -> UART1;
36    App.UartStream -> UART1;
37    App.Get -> UserButtonC;
38    App.Notify -> UserButtonC;
39    App.Pisone -> Pines.Port23;
40    App.InterrupcionPisone -> IntPines.Port23;
41    App.Rele -> Pines.Port26;
42  }
```

Listado 5.20: Hilado de las interfaces.

5.3 Software del Mote-Vivienda

El Mote-Vivienda de cada uno de las viviendas, se le instala la aplicación que viene incluida en el sistema operativo TinyOS, denominada *BaseStation*.

La función de este mote es establecer un puente entre la interfaz física y la comunicación inalámbrica con el Mote-Portero. En este caso la interfaz física es de tipo USB, concretamente el modo de emulación de puerto serie o RS-232.

Se destaca 2 etapas en la programación de esta aplicación:

- **Primera etapa.**

Se lleva a cabo la recepción de los datos provenientes de la aplicación Java ejecutada en el PC al que está conectado el Mote-Vivienda, y la retransmisión de los mismos a través de la radio hacia el Mote-Portero. En el listado 5.21 se muestra el código del evento correspondiente de esta etapa:

```
1   event message_t *UartReceive.receive[am_id_t id](message_t *msg,
2                                                    void *payload,
3                                                    uint8_t len) {
4     message_t *ret = msg;
5     bool reflectToken = FALSE;
6
7     atomic
8       if (!radioFull)
9         {
10          reflectToken = TRUE;
11          ret = radioQueue[radioIn];
12          radioQueue[radioIn] = msg;
13          if (++radioIn >= RADIO_QUEUE_LEN)
14            radioIn = 0;
15          if (radioIn == radioOut)
16            radioFull = TRUE;
17
18          if (!radioBusy)
19            {
20              post radioSendTask();
21              radioBusy = TRUE;
22            }
23        }
24      else
```

```
25              dropBlink();
26
27    if (reflectToken) {
28      //call UartTokenReceive.ReflectToken(Token);
29    }
30    return ret;
31  }
```

Listado 5.21: Recepción de los datos por el puerto serie.

Como se puede observar, una vez recogidos los datos provenientes de la aplicación Java, ejecutada en el PC, por el puerto serie, se ejecuta el evento *radioSendTask*, cuyo código se presenta en el listado 5.22, que realiza la retransmisión de los datos recibidos al Mote-Portero a través de la radio (línea 23).

```
1   task void radioSendTask() {
2     uint8_t len;
3     am_id_t id;
4     am_addr_t addr,source;
5     message_t* msg;
6
7     atomic
8       if (radioIn == radioOut && !radioFull)
9             {
10              radioBusy = FALSE;
11              return;
12            }
13
14    msg = radioQueue[radioOut];
15    len = call UartPacket.payloadLength(msg);
16    addr = call UartAMPacket.destination(msg);
17    source = call UartAMPacket.source(msg);
18    id = call UartAMPacket.type(msg);
19
20    call RadioPacket.clear(msg);
21    call RadioAMPacket.setSource(msg, source);
22
23    if (call RadioSend.send[id](addr, msg, len) == SUCCESS)
24      call Leds.led0Toggle();
25    else
26      {
27            failBlink();
28            post radioSendTask();
29      }
30  }
```

Listado 5.22: Retransmisión de los datos por radio a los demás motes.

- **Segunda etapa**.

En esta etapa se lleva a cabo la recepción de los datos que llegan por radio desde el Mote-Portero, para posteriormente retransmitirlos al PC mediante el puerto serie. En el listado 5.23 se muestra la parte del programa que realiza la recepción de los datos provenientes del Mote-Portero a través de la radio.

```
1   event message_t *RadioReceive.receive[am_id_t id](message_t *msg,
2                                                     void *payload,
3                                                     uint8_t len) {
4     return receive(msg, payload, len);
5   }
6
7   message_t* receive(message_t *msg, void *payload, uint8_t len) {
```

```
8     message_t *ret = msg;
9
10    atomic {
11     if (!uartFull)
12            {
13              ret = uartQueue[uartIn];
14              uartQueue[uartIn] = msg;
15
16              uartIn = (uartIn + 1) % UART_QUEUE_LEN;
17
18              if (uartIn == uartOut)
19                uartFull = TRUE;
20
21              if (!uartBusy)
22                {
23                  post uartSendTask();
24                  uartBusy = TRUE;
25                }
26            }
27     else
28            dropBlink();
29    }
30    return ret;
31  }
```

Listado 5.23: Recepción de los datos por radio.

Una vez recibidos todos los datos por la radio, se realiza la retransmisión de los mismos por el puerto serie, (cuyo proceso está dirigido por el evento *uartSendTask*), hacía la aplicación Java ejecutada en el PC. La parte del programa correspondiente a este proceso se muestra en el listado 5.24 siguiente:

```
1   task void uartSendTask() {
2     uint8_t len;
3     am_id_t id;
4     am_addr_t addr, src;
5     message_t* msg;
6     atomic
7       if (uartIn == uartOut && !uartFull)
8             {
9               uartBusy = FALSE;
10              return;
11            }
12
13    msg = uartQueue[uartOut];
14    tmpLen = len = call RadioPacket.payloadLength(msg);
15    id = call RadioAMPacket.type(msg);
16    addr = call RadioAMPacket.destination(msg);
17    src = call RadioAMPacket.source(msg);
18    call UartPacket.clear(msg);
19    call UartAMPacket.setSource(msg, src);
20
21    if (call UartSend.send[id](addr, uartQueue[uartOut], len) == SUCCESS)
22      call Leds.led1Toggle();
23    else
24      {
25            failBlink();
26            post uartSendTask();
27      }
28  }
```

Listado 5.24: Retransmisión de los datos al PC a través del puerto serie.

Igual que en el programa de la aplicación ejecutada en el Mote-Portero, para el enviado de los datos tanto por el puerto serie como por la radio, se debe implementar un evento para indicar la finalización de cada proceso de comunicación, del tipo *sendDone*. En el listado 5.25 se muestra el código de señalización de finalización de ambas comunicaciones, tanto del puerto serie, (Uart), como de la radio.

```
1   // Señal de finalización de la comunicación por el puerto serie (Uart)
2   event void UartSend.sendDone[am_id_t id](message_t* msg, error_t error) {
3     if (error != SUCCESS)
4       failBlink();
5     else
6       atomic
7           if (msg == uartQueue[uartOut])
8           {
9             if (++uartOut >= UART_QUEUE_LEN)
10              uartOut = 0;
11            if (uartFull)
12              uartFull = FALSE;
13          }
14    post uartSendTask();
15  }
16
17  // Señal de finalización de la comunicación inalámbrica
18  event void RadioSend.sendDone[am_id_t id](message_t* msg, error_t error) {
19    if (error != SUCCESS)
20      failBlink();
21    else
22      atomic
23          if (msg == radioQueue[radioOut])
24          {
25            if (++radioOut >= RADIO_QUEUE_LEN)
26              radioOut = 0;
27            if (radioFull)
28              radioFull = FALSE;
29          }
30    post radioSendTask();
31  }
```

Listado 5.25: Señal de finalización de la comunicación serie y de la comunicación por radio.

Cabe destacar, que tanto en el archivo *makefile* del Mote-Portero como el del Mote-Vivienda, se establece el tamaño de los paquetes a 110 bytes, para una óptima transmisión de los datos. Para ello se añade el siguiente comando, (línea 1), en el archivo *makefile* de ambas aplicaciones, mostrado en el listado 5.26 siguiente:

```
1   CFLAGS += -DTOSH_DATA_LENGTH=110
2   #CFLAGS += -DCC2420_NO_ADDRESS_RECOGNITION  // Línea excluida
3   #así se comenta una línea
```

Listado 5.26: Comandos presentes en el archivo *makefile* de la aplicación del Mote-Vivienda.

El comando de la línea 2 es un comando excluido del *makefile* del Mote-Vivienda, y el cual venía ya predeterminado en el archivo, para poder asignar direcciones a los motes utilizados.

5.4 Interfaz gráfica Java ejecutada en el PC

La interfaz gráfica o GUI, ejecutada en el PC de la vivienda, presenta un papel muy importante para la interactuación y el manejo de los dispositivos empleados, utilizando un conjunto de imágenes y objetos gráficos para representar la información y

acciones disponibles en dicha interfaz. Este control sobre los dispositivos se realiza de forma directa por parte del usuario, lo que permite una gran flexibilidad y manejo en tiempo real.

Como ya se ha comentado, el lenguaje utilizado para la programación de esta aplicación se ha realizado en Java. A continuación, en la figura 5.2, se muestra una imagen de la aplicación final:

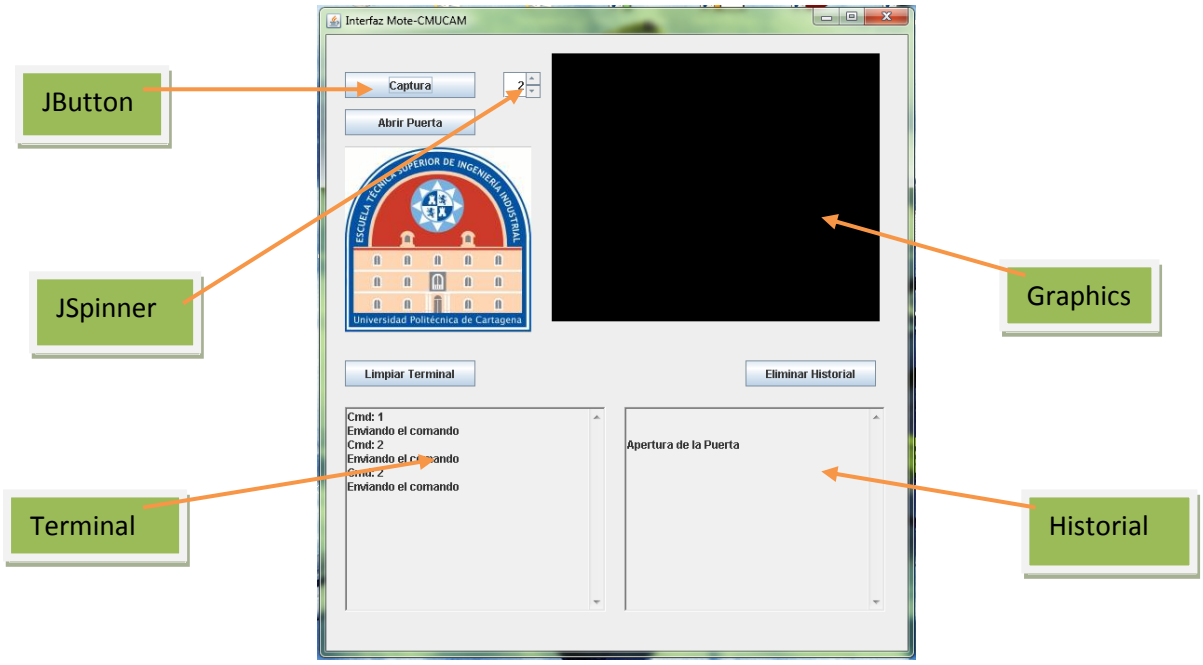


Figura 5.2: Captura de la interfaz gráfica Java del Video Portero

A continuación se explica la función de cada uno estos objetos gráficos utilizados:

- **JButton.** Se han definido 4 botones, el Botón de "Captura" envía un comando al Mote-Portero para que le devuelva una imagen. Además, ha sido configurado para que cuando sea clickeado ponga en marcha el modo video presente en el código del Mote-Portero, recibiendo una imagen cada segundo. Cuando es clickeado de nuevo, deja de recibir imágenes, ya que es desactivado el Timer TimerPausa implementado en el Mote-Portero. El botón "Abrir Puerta" envía el comando para la activación del relé que a su vez activa la cerradura eléctrica para abrir la puerta del hipotético bloque de viviendas. El resto de botones, "Limpiar Terminal" y "Eliminar Historial", borra todas las alertas generadas en el Terminal y el Historial, respectivamente.
- **Graphics.** Este objeto muestra al usuario las imágenes captadas por la CMUcam3.
- **JSpinner.** En este recuadro numérico se establece la dirección del Mote-Vivienda al que se haya conectado por el puerto serie.
- **JTextArea.** El "Terminal" y el "Historial" han sido realizados con este objeto gráfico. El primero muestra cuando es enviado un comando al Mote-Portero, indicando la dirección del Mote-Vivienda. El segundo objeto, muestra la fecha y la hora a la que es pulsado el timbre correspondiente a la

dirección del Mote-Vivienda, o lo que es lo mismo, la dirección de la vivienda. Además muestra cuando es realizada la apertura de la puerta.

- **JLabel.** Para el etiquetado de textos, en este caso se ha utilizado para insertar la imagen del escudo de la Universidad Politécnica de Cartagena.
- **JPanel.** Es el panel en el que se han montado todos los objetos gráficos anteriormente mencionados.

Para entender como se ha realizado la programación de esta interfaz gráfica, así como la elaboración de todos sus componentes o la comunicación con el mote, a continuación se explicará detalladamente su código, mostrando solo los fragmentos más importantes.

En el listado 5.28, se muestra las librerías utilizadas para la realización de esta aplicación, y que se encuentran en la cabecera de su código. Estas librerías son necesarias para el manejo y utilización de los recursos de Java, y permitir la ejecución de las funciones, así como permitir el uso los objetos gráficos comentados anteriormente.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.io.IOException;
5  import java.io.*;
6  import net.tinyos.message.*;
7  import net.tinyos.packet.*;
8  import net.tinyos.util.*;
9  import java.io.BufferedOutputStream;
10 import java.io.FileOutputStream;
11 import java.awt.image.BufferedImage;
12 import javax.imageio.ImageIO;
13 import javax.swing.JTextArea;
14 import java.awt.Toolkit;
15 import java.util.Calendar;
16 import java.util.GregorianCalendar;
17 import javax.swing.JSpinner;
18 import javax.swing.event.ChangeEvent;
19 import javax.swing.event.ChangeListener;
20 import javax.swing.SpinnerNumberModel;
```

Listado 5.28: Librerías empleadas en la interfaz gráfica Java.

Para la implementación de los objetos gráficos, por ejemplo, de un JButton o de un JTextArea ha sido necesaria la realización del siguiente código, que se presenta en el listado 5.29:

```
1  // Objetos declarados en el constructor.
2  public TestSerial(MoteIF moteIF) {
3
4     BorderLayout thisLayout = new BorderLayout();
5     getContentPane().setLayout(thisLayout);
6     this.setTitle("Interfaz Mote-CMUCAM");  //El nombre que aparecerá como título de la interfaz
7     this.setSize(640, 700);      //Para definir el tamaño de la interfaz
8     this.setVisible(true);  //Mantener la aplicación visible
9
10    panel = new JPanel(new BorderLayout(640,700));   //Para definir el tamaño del panel
11    BorderLayout jPanelSuperiorLayout = new BorderLayout();
12    panel.setLayout(jPanelSuperiorLayout);
13    getContentPane().add(panel, BorderLayout.CENTER);
14    panel.setBackground(new java.awt.Color(200, 200, 200));
15    panel.setPreferredSize(new java.awt.Dimension(640, 700));
```

```
16    panel.setLayout(null);
17
18    botonenviarcomando = new JButton("Captura");       //Para nombrar el botón
19    botonenviarcomando.setActionCommand("SendButton");  //Para asociarlo a un evento
20    botonenviarcomando.addActionListener(this);              //Realiza la lectura de su estado
21    panel.add(botonenviarcomando);                           //Lo agrega al panel
22    botonenviarcomando.setBounds(20, 40, 140, 28);  //Define su tamaño así como su localización en el panel
23    historial = new TextArea(5,40);
24    historial.setEditable(false);       //Para que no pueda ser editado durante su ejecución
25    panel.add(historial);
26    historial.setBounds(320, 400, 280, 220);    //Define su tamaño así como su localización en el panel
27    historial.setFont(new java.awt.Font("Arial", 1, 12));   //Tamaño y tipo de letra del texto
28  } // Se cierra contructor
```

Listado 5.29: Declaración de un JButton y de un JTextArea en el constructor

Como se ha podido observar, para declarar un objeto gráfico, es necesario introducir su tamaño, su localización dentro de la interfaz gráfica, al objeto panel en el que se haya contenido, el tipo de letra y el tamaño de la misma, el evento al que es asociado para luego ser invocado para la ejecución de la tarea a la que esté asociado, etc.

En el listado 5.30, se presenta las acciones vinculadas a cada uno de los objetos gráficos definidos en este proyecto:

```
1   public void actionPerformed(ActionEvent e)
2    {
3           if("OpenButton".equals(e.getActionCommand()))
4           {
5                   MensajeEntrada payload = new MensajeEntrada();
6                   cmd1 = 2;   //Variable asociada al comando icomando=2
7                   historial.append("\n\nApertura de la Puerta");
8                   System.out.println("\nPuerta Abierta\n");
9
10                  try
11                  {
12                          payload.set_comando(cmd1);
13                          moteIF.send(0x0000, payload);
14                  }
15          catch (IOException exception)
16                  {
17                          terminal.append("Exception thrown when sending packets. \n");
18                          terminal.append(exception + "\n");
19                   }
20          }
21
22          else if("BorrarHistorial".equals(e.getActionCommand()))
23          {
24                          historial.setText("");
25                          System.out.println("\nHistorial Borrado\n");
26          }
27
28          else if("BorrarTerminal".equals(e.getActionCommand()))
29          {
30                          terminal.setText("");
31                          System.out.println("\nTerminal Limpiado\n");
32          }
33
34          else if ("SendButton".equals(e.getActionCommand()))
35          {
36                      cmd1 = 1;   //Variable asociada al comando icomando=2
37                      cmd = piso;   //Variable asociada a la dirección del Mote-Vivienda
```

```
38                      MensajeEntrada payload = new MensajeEntrada();
39                      terminal.append("Cmd: " + cmd + "\n" + "Enviando el comando \n");
40
41                      try
42                       {
43                              payload.set_piso(cmd);
44                              payload.set_comando(cmd1);
45                              moteIF.send(0x0000, payload);
46                       }
47                      catch (IOException exception)
48                       {
49                              terminal.append("Exception thrown when sending packets. \n");
50                              terminal.append(exception + "\n");
51                       }
52              }
53   } //Fin
```

Listado 5.30: Acciones vinculadas a los objetos gráficos declarados.

Se presenta una serie de condiciones asociadas a cada uno de los objetos gráficos. En la condición establecida entre la línea 4 y la 20, realiza la acción de la apertura de la puerta. Para ello, el mensaje es cargado en un *payload* asociado al subprograma *MensajeEntrada.java* generado por la herramienta MIG. Este mensaje contiene el valor de la variable *"icomando*" declarada en el Mote-Portero, además de mostrar en el historial el mensaje de apertura de puerta, así como el mensaje de puerta abierta en el terminal *Cygwin*, donde es ejecutada esta aplicación gráfica. En las líneas 12 y 13 asocia el valor de la variable "cmd1" a la variable "comando", compartida con la aplicación Mote-Portero y declarada en el archivo cabecera *EC250.h.* Además establece la dirección del Mote-Portero, que es la dirección 0.

Las condiciones comprendidas entre las líneas 23 a la 26, y las líneas 29 a la 32, realizan el borrado tanto del JTextArea del terminal como del JTextArea del historial.

La condición comprendida entre las líneas 35 a la 52, envía el mensaje para la solicitud de captura de imágenes por parte de la CMUcam3. En este caso, la variable *cmd1* es igual a 1, para indicarle al Mote-Portero la acción a realizar, es decir, envía los comandos correspondientes para la captura de una imagen a la CMUcam3. Como es lógico carga este valor en el payload. Cabe destacar que entra en juego la variable "*piso*", que le indica al Mote-Portero la dirección del Mote-Vivienda que le ha solicitado la imagen, de esta forma se evita que se envíe a todas las viviendas.

En el listado 5.31, se presenta el código empleado para la recepción de la imagen y de los mensajes timbres que provienen del Mote-Portero.

```
1   public void messageReceived(int to, Message message)
2    {
3      int fin;
4
5      if (message.amType() == 30)
6      {
7
8       MensajeTimbre timbre = (MensajeTimbre)message;
9               fin = timbre.get_numpulsador();
10
11                                      for (int i = 0; i < 3; i++)
12                                     {
13
14                                      try
15                                      {
```

```
16
17                                          Toolkit.getDefaultToolkit().beep();
18                                          Thread.sleep(300);
19                                      }catch (InterruptedException ie){}
20                                  } //Fin del for
21
22                          System.out.println("Mensaje de Timbre recibido \n");
23      }
24      else if (message.amType() == 20)
25      {
26        MensajeSalida msg = (MensajeSalida)message;
27        fin = msg.get_longitud();
28
29        for (short i = 0;i < fin;i++)
30        {
31
32                                  vi[n_imagen] = (byte)msg.getElement_cmdRsp(i);
33            n_imagen++;
34                                  if ((n_imagen >= 9) && (vi[n_imagen-9] == 'J') && (vi[n_imagen-8] == 'P') &&
35    (vi[n_imagen-7] == 'G') && (vi[n_imagen-6] == '_') && (vi[n_imagen-5] == 'E') && (vi[n_imagen-4] == 'N') &&
36    (vi[n_imagen-3] == 'D') && (vi[n_imagen-2] == 13) && (vi[n_imagen-1] == ':'))
37                                  {
38                                          System.out.println("Imagen obtenida \n");
39                                          canalimagen = new ByteArrayInputStream(vi, 0, n_imagen-9);
40                                          System.out.println("Canal Imagen creado \n");
41                                          try
42                                          {
43                                                  imagen = ImageIO.read(canalimagen);
44                                                  ImageIO.write(imagen, formato, fichero);
45                                                  System.out.println("Imagen Guardada \n");
46                                          }
47                                          catch(IOException exception)
48                                          {
49                                                  terminal.append("Exception obteniendo la imagen. \n");
50                                          terminal.append(exception + "\n");
51                                          }
52                                          System.out.println("Imagen lista \n");
53                                          graficos.drawImage(imagen, 250, 50, 352, 288, this);
54                                          repaint();
55                                          n_imagen = 0;
56                                  }
57
58        } // Fin del for
59      } // Fin del condicional message.amType()==20
60    }
```

Listado 5.31: Evento asociado a la recepción de la imagen captada por la CMUcam3.

Se presenta dos partes en este listado 5.31 para la recepción de los datos provenientes del Mote-Vivienda.

La primera parte, corresponde a la parte de recepción de los mensajes timbres, cuya implementación se ha realizado dentro de un condicional, el cual comprueba el tipo de mensaje asociado, en este caso con el "*message.amType()==30",* variable enumerada en el archivo de cabecera *EC250.h.* Una vez recibido el mensaje timbre, y habiéndolo comprobado con el número del pulsador y con la dirección de la vivienda, a través de un *for*, comprendido entre las líneas 12 a la 20, se ejecuta 3 veces el sonido *beep*, llamado en la línea 17, *("Toolkit.getDefaultToolkit().beep();").*

Sino, el mensaje recibido tiene el "*amType()==20",* corresponde a la recepción de una imagen, que a través de un *for* procede a almacenar los datos de la misma. Una

vez que la recepción es finalizada, comprobando el carácter *"JPG_END :"* al final de los datos recibidos, líneas comprendidas entre la 34 a la 36, se procede a almacenar los datos de la imagen recibida en un *array*, línea 39. El *Try* comprendido entre las líneas 42 a la 46, le asigna a la imagen almacenada un fichero y dándole formato. A continuación es insertada en el objeto gráfico *Graphics*, línea 53, y mostrada por pantalla. Las líneas 54 y 55 proceden a la actualización del objeto gráfico ante la recepción de una nueva imagen, borrando el anterior asignando la variable n_imagen un 0.

En el anexo 1 se presenta el uso y utilización de este prototipo, así como, los pasos a seguir para la ejecución de sus programas y puesta de funcionamiento de los dispositivos empleados en este proyecto.

Capítulo 6

CONCLUSIONES Y TRABAJOS FUTUROS

6.1 Conclusiones

Para la realización de este proyecto, se ha abordado la realización de una red de sensores inalámbricos, mediante tres motes, tipo Telosb, con el objetivo de diseñar y desarrollar un prototipo de video portero.

Para ello, se llevó a cabo un estudio para la elección de los dispositivos apropiados para la realización de este proyecto presentes en el mercado. Para la adquisición de las imágenes se eligió la CMUcam3, puesto que era el dispositivo más apropiado para su conexión de forma estable con el Telosb y, lo más importante, era de código abierto. Además, este dispositivo ofrecía muchas posibilidades de conseguir una buena conexión con los microcontroladores presentes en el mercado, en este caso con el microcontrolador interno del Telosb, el MSP430.

Una vez conseguida la CMUcam3, había dos formas de conectarlo al mote, una de ellas explicada en la página oficial de la misma [22], en la cual se conectaba el mote directamente con la CMUcam3, aprovechando que esta tenía conexión TTL interna, y por lo tanto ofrecía una conexión directa. Sin embargo, para este tipo de conexión era necesaria la eliminación de unas resistencias internas de la CMUcam3, y debido a la posibilidad de dañarla se decidió desechar esta opción.

La opción, y la que se eligió para evitar cualquier modificación que afectara el funcionamiento de la CMUcam3, fue aprovechar su conexión RS-232, la cual permitía su conexión directa con los ordenadores para la programación de su microcontrolador interno. Para poder conectar el mote con la CMUcam3 mediante la interfaz RS-232 fue necesario realizar un circuito de acondicionamiento para la adaptación de los niveles de los distintos dispositivos, como ya se comentó en el capítulo 4.

En definitiva, se ha desarrollado un prototipo de video portero basado en redes inalámbricas de sensores. La arquitectura está basada, principalmente, por dos elementos hardware y una aplicación software ejecutada en un PC. El primer elemento hardware es el nodo sensor inalámbrico, basado en un mote Telosb, que se instalará en la puerta principal. Se ha desarrollado la interfaz gráfica para hacer posible el manejo y control de estos dispositivos para los habitantes de la vivienda. Además, ha sido necesario el desarrollo de un circuito de acondicionamiento para controlar la cerradura de la puerta principal, desde el mismo nodo al que está conectada la CMUcam3. Para conectar este último dispositivo, también ha sido necesario el desarrollo de un circuito adaptador de niveles.

6.2 Trabajos Futuros

Entre los trabajos futuros que se podría realizar para la mejora de este prototipo se proponen los siguientes:

- Este prototipo de video portero esta implementado con tan solo 3 motes, uno que estaría alojado en la puerta principal, y los otros dos, uno para cada una de las dos viviendas. Sin embargo, se puede aumentar el número de viviendas a las cuales dar servicio, mediante la conexión de más pulsadores, para su función como timbres. Para las modificaciones necesarias para aumentar el número de viviendas, tan solo bastaría con modificar el código del mote alojado en la puerta, para añadirle más pulsadores.

  Para completar, se ha supuesto que se va a llevar a cabo la instalación de este prototipo a un bloque de viviendas, de 5 plantas, con planta baja incluida, y con dos escaleras, además de dos portales al exterior. Cada escalera comunica con 4 viviendas, es decir, 8 viviendas por planta, lo que hace un total de 40 viviendas en todo el bloque. Además, se ha supuesto que el edificio tiene una altura de 16 metros, es decir que, entre cada planta, hay unos 3.2 metros. Según las especificaciones técnicas, un Telosb tiene un alcance de entre 20 a 30 metros en interiores. Dependiendo de si se le colocara o no una antena adicional SMA, sería adecuado colocar nodos intermedios, para que no se corte la comunicación entre los Motes-Base de cada una de las viviendas y, los Motes-Portero ubicados en cada portal. Al no tener datos para comprobar el alcance de la antena SMA, se añade dos motes más, programados para que funcionen como nodos intermedios, y ubicados de forma estratégica en la segunda planta, uno por escalera. De esta forma se mantienen comunicados los Motes-Base, de la tercera y cuarta planta, con los Motes-Portero ubicados en los portales de la planta baja. La ubicación de estos nodos intermedios en la segunda planta es debido a que estarían aproximadamente a 9 metros de los Motes-Portero, distancia más que suficiente para que lleguen las imágenes hasta las viviendas de la cuarta planta, y evitar todos los posibles obstáculos entre planta y planta (hormigón,

muebles, otros sistemas como la calefacción centralizada o la de la luz, así como tuberías de suministro o desagüe). En la figura 6.1 se presenta un croquis que representa el edificio con dada una de sus plantas, además de la ubicación de cada uno de los motes.

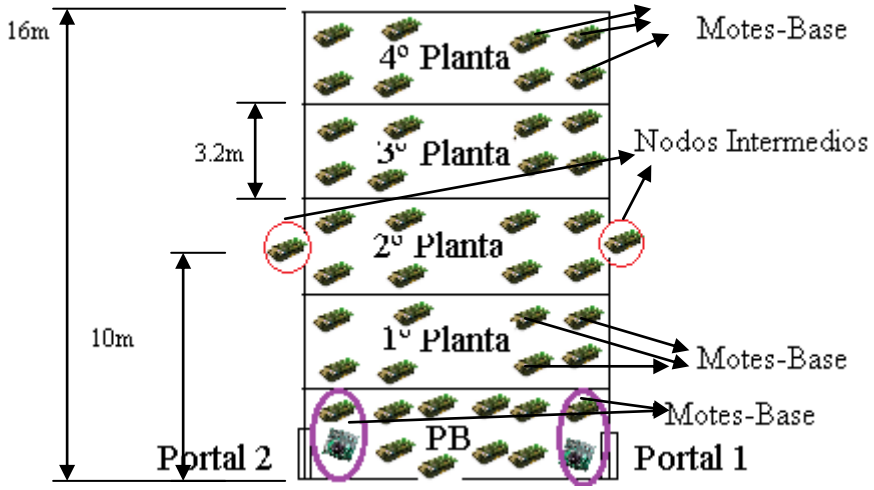


Figura 6.1: Representación del despliegue de los motes en el bloque de viviendas.

Para la asignación de la dirección de los motes y su programación se harán en dos agrupaciones, en relación a las viviendas de cada escalera. Es decir, los motes ubicados en las viviendas que den salida al portal número 1 serán independientes de los ubicados en las viviendas que den al portal número 2. Si será necesario asignar las direcciones de manera que se evite la repetición de las mismas, ya que supondría un mal funcionamiento del sistema, y por lo tanto no cumpliría su función. A continuación se muestra una tabla 6.1 con el coste aproximado de los motes que se emplearían, además de la dos CMUcam3 necesarias, una en cada portal.

DISPOSITIVOS	UNIDADES	PRECIO UNIDAD	PRECIO TOTAL
Telosb	44	80	3520
CMUcam3	2	131,48	262,96
Cerradura Electromecánica	2	26,5	53
		TOTAL	3835,96€

Tabla 6.1: Presupuesto de la aplicación práctica de este prototipo.

Faltaría por incluir el precio de los componentes utilizados en los circuitos de acondicionamiento.

- Otra mejora que se podría realizar es la programación de la CMUcam3 para realizar reconocimientos faciales a partir de una base de datos. De esta manera, no solo realizaría una imagen, sino que podría comunicar al usuario de la vivienda la identidad de la visita. Y no solo esto, se podría, por ejemplo, programar el sistema para que abriera la puerta automáticamente, y sin la intervención de los usuarios de la vivienda, a los propios propietarios

de la misma, a familiares o a personas de confianza, como el servicio de la limpieza, el cartero, etc.

- Una mejora destacada sería el desarrollo de una base datos, para registrar aquellas visitas que se realizan cuando los propietarios no se encuentran en la vivienda en ese momento. Para ello, se podría implementar un programa que guardara la imagen del visitante y la enviara por correo al propietario, indicándole la fecha y la hora de dicha visita, la identidad de la misma, hasta un mensaje de voz grabado por el visitante, indicando el motivo por el que el ha realizado dicha visita. También se podría subir las imágenes a un servidor alojado en una web de tipo PHP, que permita la consulta en directo a cualquier propietario, y en cualquier lugar, de las visitas que han sido realizadas en su domicilio durante su ausencia. Incluso, se podría crear otras interfaces gráficas para cualquier otro dispositivo, ya sea Tablet o Smartphone, para controlar la aplicación Java ejecutada en el PC de la vivienda y, de esta forma, interactuar a distancia con el visitante en el caso de que no estuviera el propietario en su domicilio.
- Cabe destacar que este proyecto no se ha incluido audio, pero sería posible mediante la implantación de un micrófono junto con el adecuado circuito de acondicionamiento para su conexión con el Mote-Portero, así como de unos altavoces para poder mantener conversaciones con el visitante. Lógicamente, sería también necesario la modificación del software, tanto del Mote-Portero como de la aplicación Java, para la manipulación de ficheros de audio. Una de las posibles desventajas de añadir audio, sería la relantelización del sistema, ya que tendría que manejar mayor cantidad de datos. Una de las posibles soluciones, ante esta limitación de manejo de datos, sería la de utilizar otro modelo de mote que tuviera mayor capacidad de memoria RAM, o incluso que proporcionara slots para conectar memorias flash externas. De esta forma sería posible incluso obtener las imágenes en alta resolución, ya que en este proyecto se han realizado en baja resolución por motivo de la limitación de la memoria RAM.

REFERENCIAS

[1]http://es.wikipedia.org/wiki/Red_de_sensores

[2]http://www.construmatica.com/construpedia/Dom%C3%B3tica

[3]http://es.wikipedia.org/wiki/Wi-Fi

[4]http://es.wikipedia.org/wiki/Bluetooth

[5]http://es.wikipedia.org/wiki/IEEE_802.15.4

[6]http://es.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_avoidance

[7]www.zigbee.org/

[8]http://lecs.cs.ucla.edu/~adparker/EE202A/hw2/index.html

[9]http://embedded.seattle.intel-research.net/wiki/index.php?title=Intel_Mote_2

[10]http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[11]http://www.capsil.org/capsilwiki/index.php/TELOSB/TMote_Sky

[12]http://focus.ti.com/mcu/docs/mcumspoverview.tsp?sectionId=95&tabId=140&familyId=342&DCMP=MCU_other&HQS=Other+IL+msp430

[13]http://www.sics.se/contiki/

[14]http://www.tinyos.net/

[15]http://nescc.sourceforge.net/

[16]http://ti.com/msp430

[17]http://cmucam.org/

[18]http://es.wikipedia.org/wiki/Lua

[19]http://www.iearobotics.com/proyectos/cuadernos/ct1/ct1.html

[20]http://es.wikipedia.org/wiki/Rel%C3%A9

[21]http://es.wikipedia.org/wiki/Optoacoplador

[22]http://cmucam.org/wiki/telos_tmote

BIBLIOGRAFÍA BÁSICA

- Instrumentación Electrónica - Miguel A. Pérez García - Ed. Thomson
- Circuitos Electrónicos. Análisis Simulación y Diseño. - Norbert R. Malik - Ed. Prentice Hall
- Procesamiento de Señales Digitales - Sanjit K. Mitra - Ed. McGraw Hill
- Building wireless sensor networks - Faludi Robert - Ed. O'Reilly

ANEXO I

USO Y FUNCIONAMIENTO

I.1 Introducción

Para comprender el funcionamiento y uso de este prototipo se ha realizado este anexo, el cual muestra, mediante imágenes, la ejecución de las diferentes aplicaciones realizadas y el funcionamiento de los circuitos de acondicionamiento que han sido necesarios diseñar.

I.2 Preparación del sistema

Para ejecutar la aplicación Java se realiza desde el terminal *Cygwin*, y desde el directorio donde se encuentra el archivo VideoPortero.java se introduce los siguientes comandos, mostrados en el listado I.1:

```
1  $export MOTECOM=serial@COM5:115200      # telosb baud rate
2  $java VideoPortero
```

Listado I.1: Comandos introducidos en *Cygwin* para ejecutar VideoPortero.java.

El comando de la línea 1 indica el puerto serie en el que se ha conectado el Mote-Vivienda en el PC, en este caso el puerto COM5, además de indicar su tasa de baudios para su correcta sincronización con el Telosb, en este caso a 115200 baudios.

El comando de la línea 2 ejecuta la aplicación java VideoPortero.java.

A continuación, en la figura I.1, se muestra una captura de la aplicación VideoPortero.java:

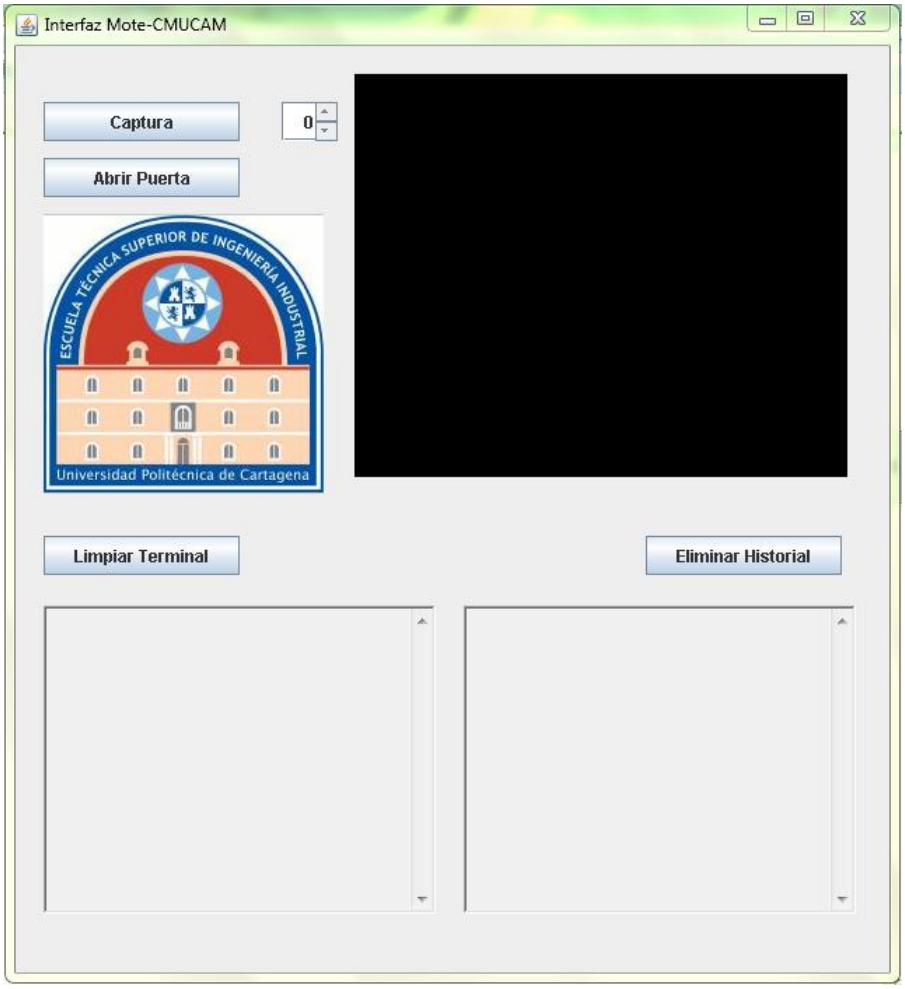


Figura I.1: Captura aplicación VideoPortero.java.

En la figura I.2 se muestra el terminal *Cygwin*, así como la respuesta de que sincronización con el Mote-Base ha sido realizada:

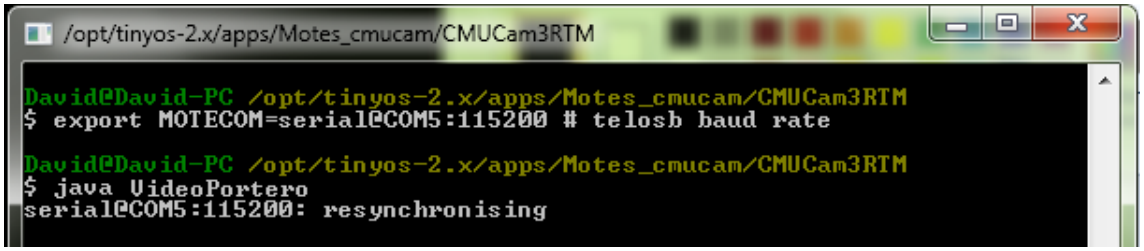


Figura I.2: Captura del terminal *Cygwin*.

De esta forma se ejecuta en las otras viviendas, no antes habiendo instalado en los Motes-Base sus direcciones correspondientes, tal como se indicó en el capítulo 5.

El siguiente paso sería indicar la dirección del nodo Mote-Base en la interfaz gráfica de la aplicación VideoPortero.java, exactamente en el objeto gráfico numérico de Java JSpinner.

I.3 Demostración del sistema

En la figura I.3 se presenta los dispositivos empleados para la implementación de este prototipo de videoportero basado en redes inalámbricas de sensores. Se puede ver la CMUcam3 a la derecha, el Mote-Portero el del centro, y los otros dos de la izquierda los dos Motes-Base de las dos viviendas. El circuito de acondicionamiento se puede ver en el centro de la imagen.

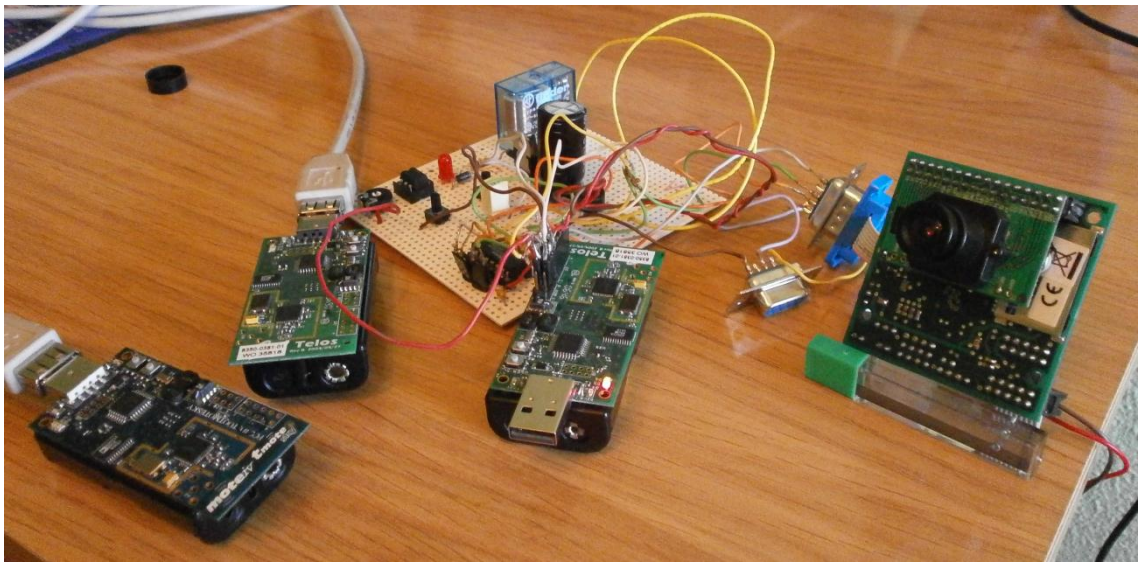


Figura I.3: Dispositivos y circuitos de acondicionamiento empleados.

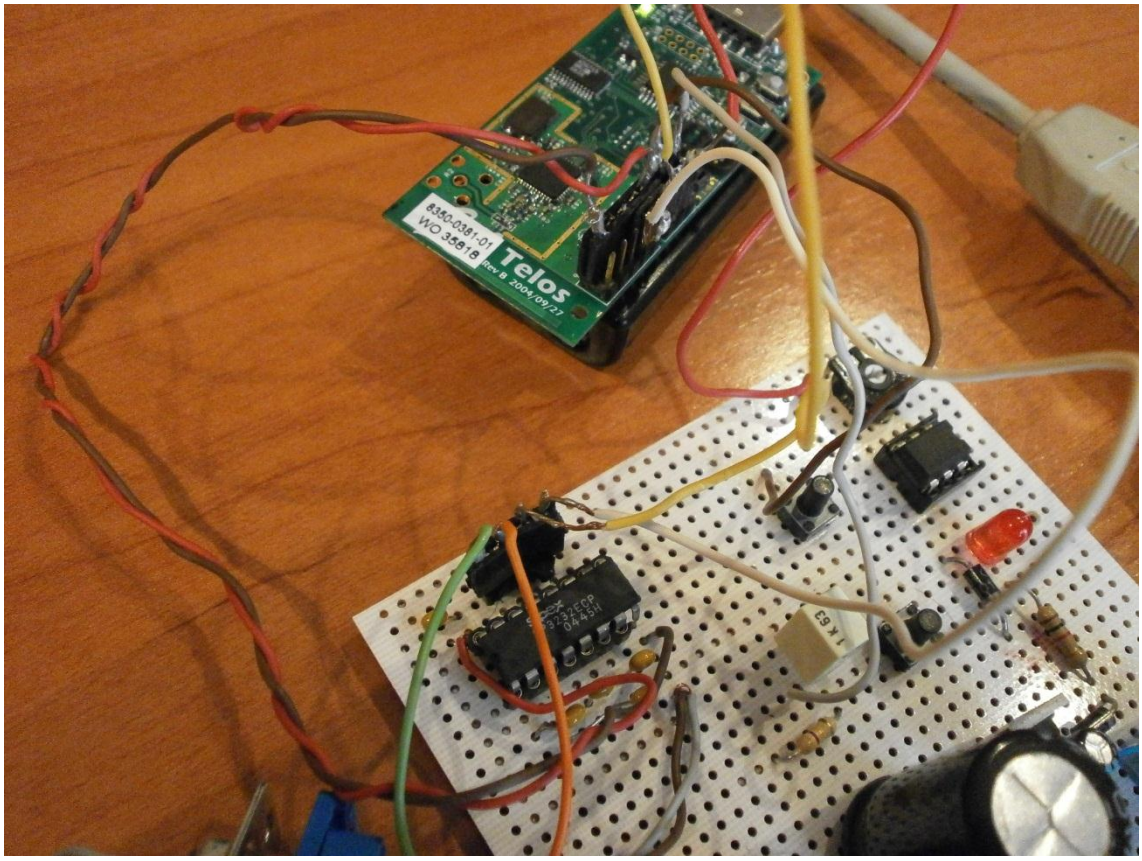


Figura I.4: Conexiones del circuito de acondicionamiento con el Mote-Portero.

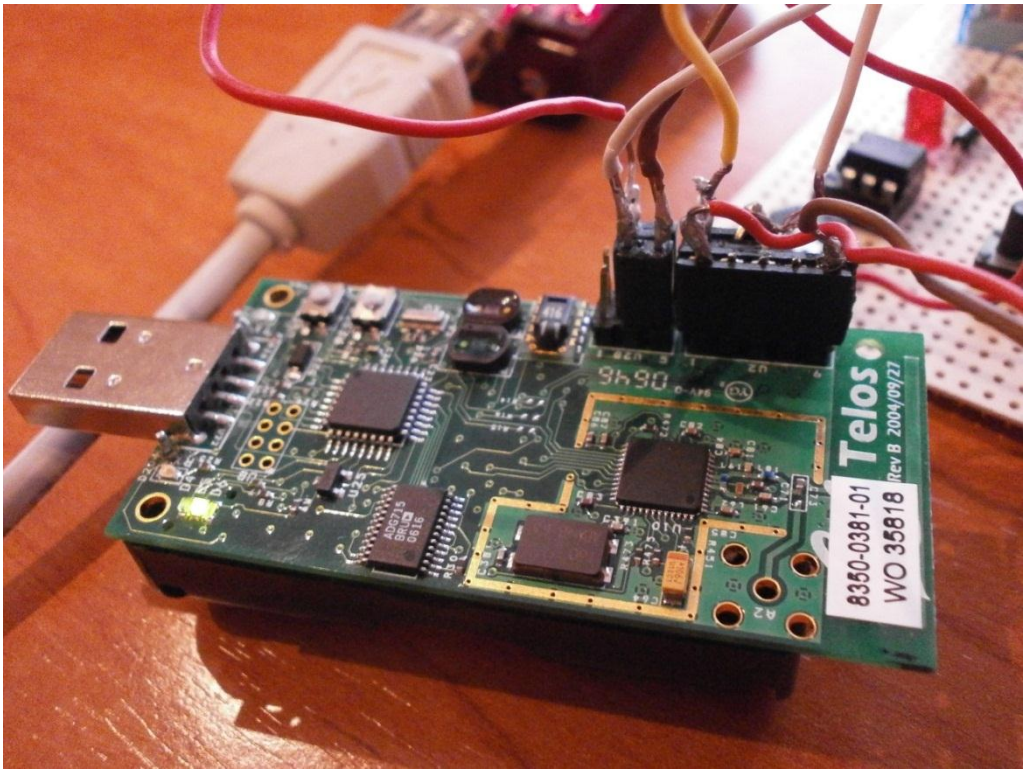


Figura I.5: Conexiones del circuito de acondicionamiento con el Mote-Portero.

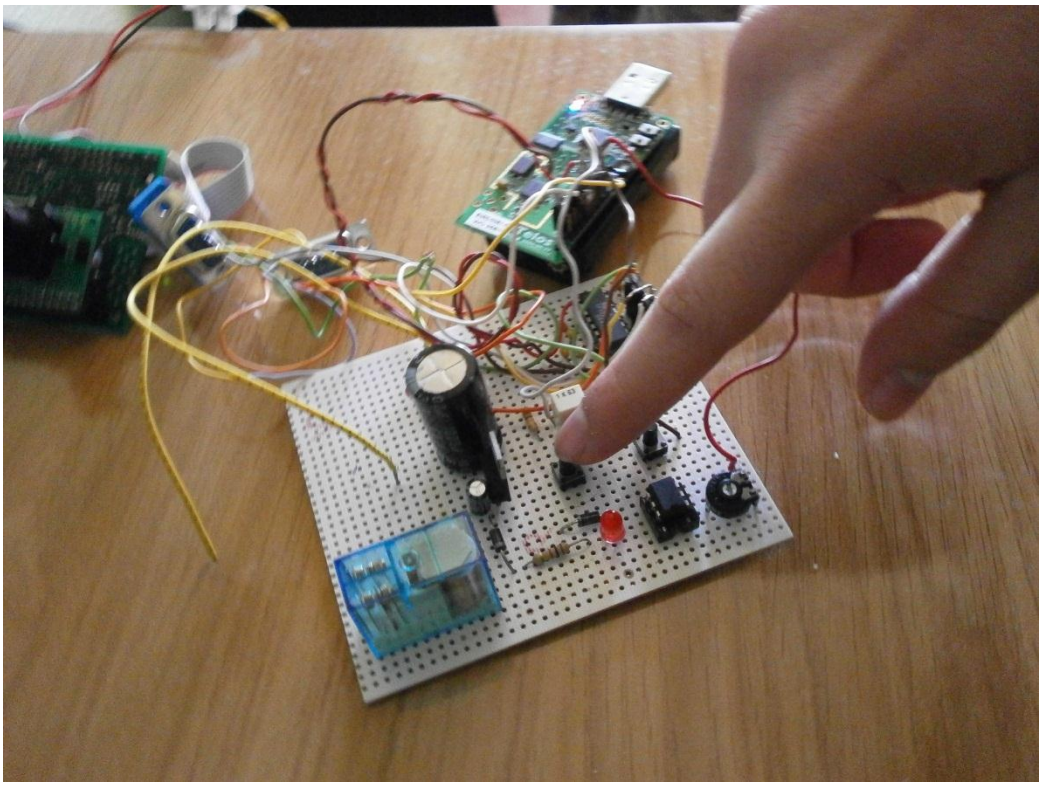


Figura I.6: Pulsación del botón timbre de la vivienda con dirección 1.

En la siguiente figura I.7 se observa la imagen enviada por el Mote-Portero al pulsar el botón timbre de una de las dos viviendas, en este caso la que corresponde con el Mote-Base dirección 1, configuradas para esta demostración:

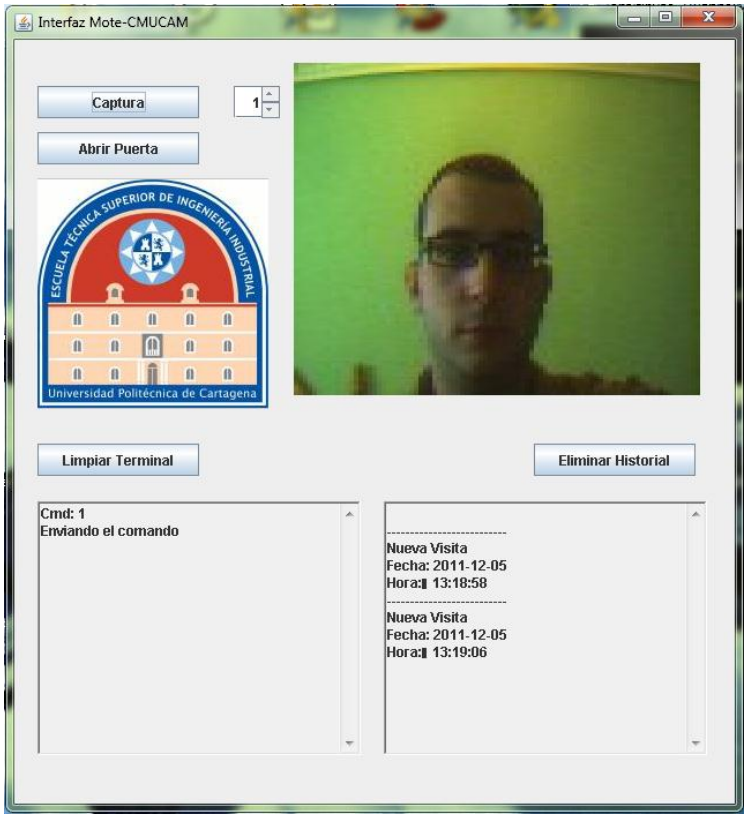


Figura I.7: Captura al pulsar botón timbre de una de las viviendas.

Una vez que el propietario ha identificado al visitante tiene dos opciones: Poner en marcha el modo video al pulsar el botón captura para obtener más imágenes ante una identificación dudosa del visitante, o directamente pulsar el botón abrir puerta que activaría el relé para la alimentación de la cerradura eléctrica, permitiendo de esta manera la apertura de la puerta al visitante.

Como es lógico, en la práctica se debe de situar la CMUcam3 a la altura suficiente para poder capturar la imagen del visitante. Además de situar los botones en un lugar más accesible, en una carcasa, donde dentro de la misma se situaría el circuito de acondicionamiento, y en la cual estaría protegido ante las condiciones ambientales.

Si se ha optado por realizar más capturas, al pulsar el botón captura, se activa el modo video, es decir, se recibirá una imagen del visitante por cada segundo transcurrido. De esta forma se puede realizar una mejor identificación. Para desactivarlo solo basta con abrir la puerta o volver a pulsar el botón de captura.

Ante la posibilidad de que no se encuentre nadie en la vivienda, en el historial queda marcado la hora y fecha que se ha pulsado el timbre del portero, además de que la imagen realizada, al pulsar el visitante el timbre, se guardará en una carpeta, previamente asignada, dentro del PC donde se ejecuta la interfaz. De esta forma, cuando vuelva el propietario, puede comprobar quién le ha visitado durante su ausencia.

Para finalizar, en al siguiente figura I.8, se presenta una captura donde se han ejecutado dos veces la aplicación Java, cada una de ellas conectadas a un Mote-Portero con dirección distinta, concretamente uno con dirección 2 y el otro con la dirección 1, donde se pueden comprobar en el objeto gráfico de Java JSpinner.

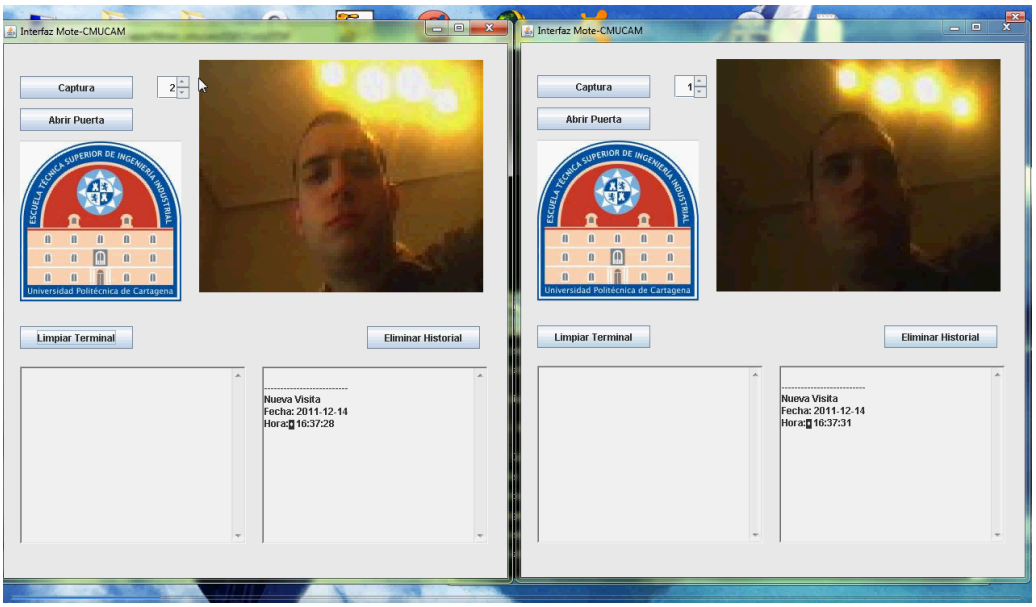


Figura I.8: Demostración final.

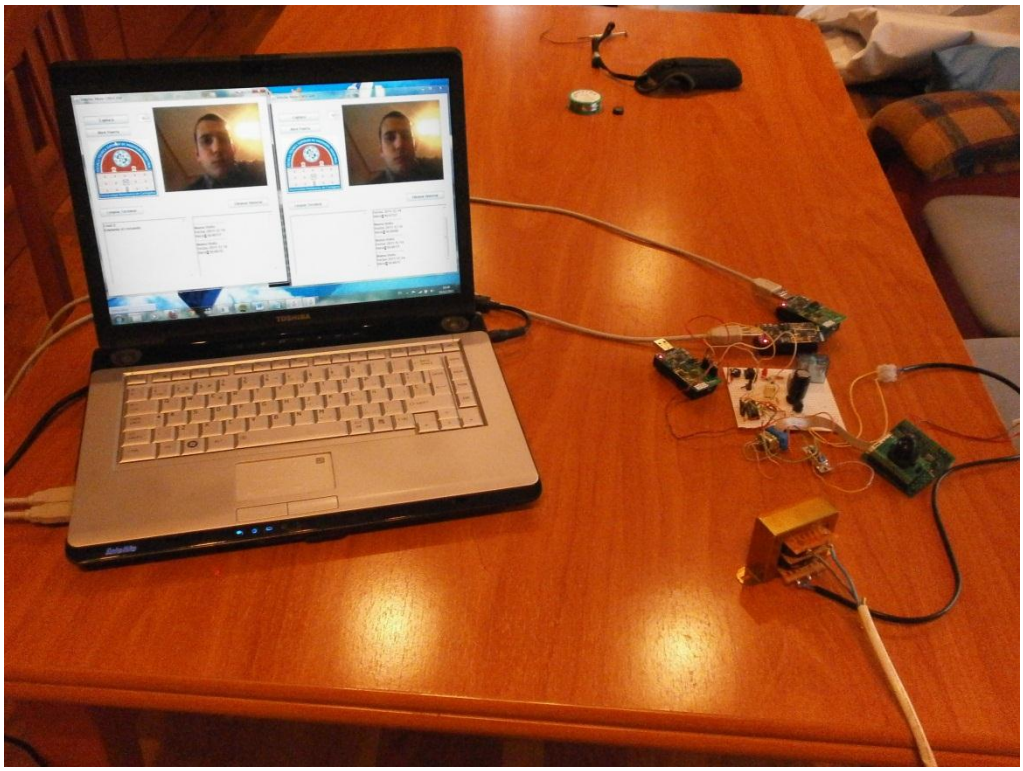


Figura I.9: Imagen global de los dispositivos constituyentes de este proyecto.