Universidad de la República
Facultad de Ingeniería

PLAGAVISIÓN

Desarrollo de un prototipo de nodo WSN con capacidad visual para la detección temprana de plagas.

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Mauricio González, Javier Schandy, Nicolás Wainstein

en cumplimiento parcial de los requerimientos
para la obtención del título de
Ingeniero Electricista.

Tutor

Dr. Ing. Leonardo Barboni ..... Universidad de la República

Co-Tutor

Mag. Ing. Álvaro Gómez ..... Universidad de la República

Tribunal

Dr. Ing. Leonardo Barboni ..... Universidad de la República
Mag. Ing. Álvaro Gómez ..... Universidad de la República
Mag. Ing. Julio Pérez ..... Universidad de la República
Dr. Ing. Conrado Rossi ..... Universidad de la República
Ing. Matías Tassano ..... Universidad de la República

Montevideo
viernes 6 junio, 2014

PLAGAVISIÓN, Mauricio González, Javier Schandy, Nicolás Wainstein.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.2).
Contiene un total de 179 páginas.
Compilada el viernes 6 junio, 2014.
http://iie.fing.edu.uy/

Algunas polillas fueron lastimadas durante el transcurso de este proyecto

Agradecimientos

Dedicamos esta página a agradecer a todas las personas que colaboraron con el desarrollo de este proyecto, particularmente:

A nuestras familias y amigos, por su apoyo incondicional no solo durante el transcurso de este proyecto sino durante toda la carrera, por su paciencia y cariño en todo momento.

A Leonardo Barboni y Álvaro Gomez, por su labor como tutores del proyecto.

A Leonardo Steinfeld, por la total disposición a despejar nuestras mas diversas dudas.

A Ignacio de León, Matías Di Martino, Gonzalo Gutierrez, Gonzalo Sotta, Matías Tassano, Pablo Pérez, Conrado Rossi y Francisco Veirano por la enorme disposición al momento de ayudar.

Al Taller de Electrónica Libre, Sergio Beheregaray y Roberto Rodríguez por facilitarnos las herramientas y el conocimiento para construir el prototipo.

Al IIE por permitirnos tomar la sala 206 durante los últimos meses de trabajo.

Resumen

En el presente documento se describe el trabajo realizado en el proyecto PLAGAVISIÓN, donde se desarrolló un prototipo funcional de un sistema de detección temprana de plagas en cultivos frutícolas mediante tecnologías de redes de sensores inalámbricos con capacidad visual. El mismo se enmarca dentro del convenio FPTA entre el Grupo de Microelectrónica del IIE-FING-UdelaR, el INIA y JUMECAL para el desarrollo de soluciones de WSN para el uso en el sector agropecuario.

La necesidad surge por parte de los productores para mejorar el control de poblaciones de polillas *Cydia Pomonela* y *Cydia Molesta* en los cultivos. Tradicionalmente estos controles se realizaron mediante el conteo manual y local de los especímenes atrapados en trampas que en su piso contienen un adhesivo y feromonas de la especie. Éstas se encuentran ubicadas de manera distribuida en el terreno, colgadas de las ramas de los árboles. En este contexto, el diferencial que ofrece esta solución es obtener en tiempo real imágenes distribuidas de las trampas, para así generar una base de datos mediante una red inalámbrica integrada por nodos de costo accesible con autonomía energética de varios meses. La información obtenida permite tomar decisiones en torno al uso de pesticidas y control mediante confusión sexual, repercutiendo en una mejor calidad de las cosechas.

El prototipo diseñado e implementado se compone por un nodo TelosB compatible CM3000 (integrado por un MCU MSP430 y una radio CC2420), una cámara Linksprite LS-Y201, LEDs para la iluminación, un circuito interfaz con switches para controlar la alimentación de la cámara y la iluminación, y dos baterías AA. El sistema propuesto es capaz de capturar, almacenar, procesar y transmitir las imágenes hacia un nodo *sink* dos veces al día con una autonomía energética esperada superior a los 28 meses. Se diseñó además un gabinete en acrílico para proteger al sistema de las condiciones ambientales.

Los nodos utilizan Contiki OS, un RTOS que realiza el control de los recursos hardware e integra distintos *stacks* de red para WSN. Sobre este sistema operativo se escribieron los distintos módulos de la aplicación entre los que se encuentran el *driver* de la cámara, el procesamiento de las imágenes y las transmisiones por radio.

En el proyecto se exploró en técnicas de procesamiento de imágenes en hardware reducido y se alcanzó a implementar, por fuera del alcance estipulado inicialmente, un algoritmo de decodificación parcial de imágenes JPEG y de detección de polillas mediante un clasificador lineal Fisher. El procesamiento permite reducir la cantidad de datos a ser transmitidos, al enviar únicamente el conteo de polillas o tomar decisiones locales sobre cuándo enviar las imágenes (e.g. no enviar hasta alcanzado cierto umbral o enviar solamente cuando existan cambios incrementales significativos en el conteo) repercutiendo en una mayor autonomía energética y en una simplificación de la red.

El principal desafío en torno al estudio, diseño e implementación de la red consistió en la necesidad de encontrar protocolos orientados a conexión y confiables en WSN que tuvieran el menor

consumo energético posible. La necesidad de una calidad de servicio orientado a conexión y confiable surge por tratarse de imágenes JPEG donde no son admisibles pérdidas, duplicados o desorden de los paquetes.

Se realizaron ensayos de caracterización del consumo de corriente y se estimó la carga consumida en cada ciclo de ejecución. Se logró obtener niveles de bajo consumo de corriente, lo que permite una autonomía energética superior a la planteada en los objetivos del proyecto.

Glosario

ACK: Del inglés (*Acknowledgment*), reconocimiento de datos en comunicaciones digitales.

ADC: Del inglés (*Analog-to-Digital Converter*), conversor analógico-digital.

API: Del inglés (*Application Programming Interface*), conjunto de funciones para programar en determinado lenguaje o sistema operativo.

AUC: Del inglés (*Area Under Curve*), área bajo la curva.

CCA: Del inglés (*Clear Channel Assesment*), medida del canal de comunicación para sensar actividad en el mismo.

CCD: Del inglés (*Charge-Coupled Device*), dispositivo de carga acoplada utilizado en sensores de imágenes.

CFS: Del inglés (*Contiki File System*), nombre del sistema de archivos de Contiki OS.

CMOS: Del inglés (*Complementary Metal Oxide Semiconductor*), es una de las familias lógicas empleadas en la fabricación de circuitos integrados.

CSMA-CA: Del inglés (*Carrier Sense Multiple Access with Collision Avoidance*), protocolo de control de acceso al medio con detección de portadora evitando colisiones.

DCO: Del inglés (*Digitaly Controlled Oscillator*), oscilador controlado digitalmente.

DCT: Del inglés (*Discrete Cosine Transform*), transformada discreta del coseno.

DSSS: Del inglés (*Direct Sequence Spread Spectrum*), espectro ensanchado por secuencia directa.

EEPROM: Del inglés (*Electrically Erasable Programmable Read-Only Memory*), ROM programable y borrable eléctricamente.

FIFO: Del inglés (*First In First Out*), es un método de manipulación y escritura de *buffers*.

FLD: Del inglés (*Fischer Linear Discriminator*), discriminante lineal Fischer.

FTDI: Del inglés (*Future Technology Devices International*), empresa que fabrica chips conversores de RS232 o TTL a USB.

FTP: Del inglés (*File Transmission Protocol*), protocolo de transmisión de archivos en redes de datos.

GPIO: Del inglés (*General Purpose Input Output*), pines de entrada/salida de propósito general.

HAL: Del inglés (*Hardware Abstraction Layer*), capa de abstracción de hardware.

IC: Del inglés (*Integrated Circuit*), circuito integrado.

ISR: Del inglés (*Interrupt Service Routine*), rutina de atención a la interrupción en MCU.

I^2C: Protocolo de comunicación serial en dos hilos.

JTAG: Del inglés (*Joint Test Action Group*), es el nombre comúnmente utilizado para la norma IEEE 1149.1 utilizada para testear PCB e IC.

LPM: Del inglés (*Low Power Mode*), modo de bajo consumo en un MCU de la familia MSP430.

MAC: Del ingés (*Medium Access Control*), control de acceso al medio.

MCU: Del ingés (*MicroController Unit*), refiere a un microcontrolador.

O-QPSK: Del inglés (Offset Quadrature Phase Shift Keying), modulación por desplazamiento de fase con *offset*.

PA-LNA: Del inglés (*Power Amplifier - Low-Noise Amplifier*), amplificador de potencia de salida para TX, y amplificador en RX con bajo nivel de ruido.

PCB: Del inglés (*Printed Circuit Board*), circuito impreso.

PIO: Del inglés (*Parallel Input/Output*), interfaz de entrada/salida paralelo.

RAM: Del inglés (*Random Access Memory*), memoria de acceso aleatorio.

RDC: Del inglés (*Radio Duty Cycling*), ciclo de trabajo de una radio.

RGB: Del inglés (*Red, Green and Blue*), por rojo, verde y azul en el modelo de color en sistemas de formación de imágenes basados en la mezcla de estos tres colores de luz primarios.

RISC: Del inglés (*Reduced Instruction Set Computing*), arquitectura de microcontroladores donde las instrucciones tienen largo fijo y sólo las instrucciones de carga y almacenamiento acceden a la memoria de datos.

ROC: Del inglés (*Receiver Operating Characteristic*), característica operativa del receptor.

ROM: Del inglés (*Read-Only Memory*), memoria que sólo puede ser leída.

RPL: Del inglés (*Routing Protocol for Lossy Networks*), protocolo de ruteo para redes susceptibles a pérdidas de paquetes.

RSSI: Del inglés (*Received Signal Strength Indicator*), indicador que se utiliza para medir el nivel de potencia de las señales recibidas en las redes inalámbricas.

RTOS: Del inglés (*Real-time Operating System*), Sistema Operativo de Tiempo Real.

RX: Receptor o recepción.

SMA: Del inglés (*SubMiniature version A*), es un tipo de conector coaxial utilizado en radiofrecuencia.

SMD: Del inglés (*Surface Mounted Device*), dispositivos de montaje superficial.

SoC: Del inglés (*System-on-a-Chip*), sistema completo embebido en un chip.

SOIC: Del inglés (*Small Outline Integrated Circuit*), dispositivos de encapsulado pequeño.

SPI: Del inglés (Serial Peripheral Interface), protocolo de comunicación serial en 4 hilos.

TCP:	Del inglés (*Transmission Control Protocol*), protocolo de control de transmisiones.
TX:	Transmisor o transmisión.
UART:	Del inglés (*Universal Asynchronous Receiver/Transmitter*), receptor-transmisor asíncrono universal.
UDP:	Del inglés (*User Datagram Protocol*), protocolo de capa de transporte basado en datagramas.
USART:	Del inglés (*Universal Synchronous/Asynchronous Receiver/Transmitter*), receptor-transmisor síncrono y asíncrono universal.
USB:	Del inglés (*Universal Serial Interface*), interfaz serial universal.
WSN:	Del inglés (*Wireless Sensor Networks*), redes de sensores inalámbricos.
YCbCr:	Es una familia de espacio de colores utilizada en sistemas de formación de imágenes.

Tabla de contenidos

Capítulo 1

Introducción

Resumen

Este capítulo contiene la introducción al contenido y la descripción del proyecto. Inicia con la definición del problema y la solución propuesta, presentando antecedentes en tecnologías similares realizadas por el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería, Universidad de la República, Uruguay (de aquí en más IIE-FING-UdelaR). Se resumen algunos trabajos relacionados en el área de WSN dotadas de capacidad visual, se describen los objetivos generales, alcance, supuestos y restricciones para la realización del proyecto. Se detalla un diagrama de bloques del sistema y se resume el contenido de los capítulos que componen la presente documentación. Por último se mencionan trabajos académicos presentados en conferencias internacionales arbitradas.

Contenido

1.1. Definición del problema

Actualmente el monitoreo del nivel de población de insectos que infectan y afectan los frutales se realiza mediante el uso de trampas construidas con pisos adhesivos donde se colocan feromonas que atraen a los machos de la especie. Este método comenzó a tener un uso generalizado entre los productores porque proporciona la información necesaria para aplicar estrategias de control de estas plagas. Este sistema de monitoreo ya se encuentra en proceso de ser regulado y controlado en forma centralizada por intermedio del MGAP [1] (Sistema PRONOS [1]).

La eficacia del sistema de detección de plagas actual depende de la precisión que tenga el operario que revise las trampas para su identificación y conteo. Así, una falla del operario conllevará a que no se detecten las plagas en forma temprana. Esto puede causar daños en los cultivos, pérdidas en la producción de frutas por infección y obligar al uso de insecticidas.

Por lo tanto, este sistema resulta: *i)* poco práctico ya que requiere entrenamiento del personal responsable de realizar el recorrido periódico por las trampas (varias veces por semana) para contar el número de insectos plagas que han sido capturados, comunicando y centralizando todos los datos obtenidos (dicho personal también se encarga del mantenimiento de las trampas) y *ii)* costoso en términos del combustible usado para recorrer todos los cultivos de distintos productores en donde se han instalado trampas. También es costoso en cuanto a las horas hombre invertidas en el personal que recorre los cultivos.

1.2. Descripción del proyecto

En el marco descripto, el proyecto pretende explorar el uso de tecnologías de redes de sensores inalámbricos y algunas técnicas de tratamiento de imágenes en plataformas con recursos hardware limitados (como son los nodos) y con reducida disponibilidad de energía de las baterías de alimentación, para la detección temprana de plagas, en particular poblaciones de *Cydia Pomonela*[2] [2] y *Cydia Molesta*[3] [3] en cultivos frutícolas. Estos tipos de tecnologías son de vital importancia para los productores agrícolas ya que permiten optimizar la recolección de datos para poder coordinar estrategias de combate de plagas.

En este contexto, el proyecto consiste en el diseño y la implementación de un prototipo de nodo de WSN embebido en una trampa tipo Delta[4] con feromonas ISOMATE ® [4], que sea capaz de adquirir, procesar y transmitir imágenes dos veces al día hacia un nodo *sink*[5] conectado a un PC. La adquisición se realiza a partir de una cámara digital conectada al nodo, donde se almacena y procesa la imagen. El procesamiento permite reducir el tamaño de la información transmitida. La comunicación entre los nodos se realiza utilizando los protocolos disponibles en el RTOS. La figura 1.1 es una representación del despliegue del sistema en el terreno.

[1] Ministerio de Ganadería, Agricultura y Pesca del Uruguay

[2] Variedades de polillas que depositan huevos en frutos creando infección de larvas, comúnmente llamada Carpocapsa.

[3] Variedad de polillas que atacan principalmente durazneros y membrillos, comúnmente llamada Grafolita.

[4] Trampa de plástico ondulado de polipropileno con fondo engomado y feromonas, utilizada para monitorear especies de plagas.

[5] Cuando hablamos de *sink* nos referimos a un nodo igual a los otros pero con el supuesto de que tiene alimentación externa, por lo tanto no tiene restricciones de energía

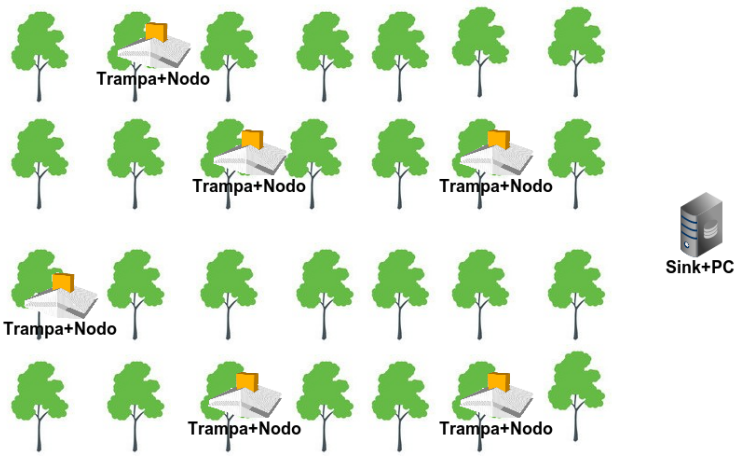


Figura 1.1: Representación del sistema en el terreno

El sistema ha de ser alimentado por baterías mediante las cuales se alcance una autonomía energética mayor a seis meses por lo que la arquitectura de los componentes debe tener un bajo consumo de corriente y modos *sleep* [6] cuando no están en uso.

1.3. Antecedentes

Las WSN están formadas por una gran cantidad de sistemas microelectrónicos embebidos individuales de reducido tamaño denominados nodos. Los nodos son plataformas de hardware reducido, es decir que priorizan el bajo consumo energético por sobre la capacidad de procesamiento. En particular, los nodos contienen un MCU, memoria, circuitos electrónicos auxiliares, sensores y una radio para comunicarse con otros nodos. Pueden adquirir señales de varios sensores, transmitir esa información a distancias de decenas a cientos de metros y recibir información de configuración de la medida de los sensores u otros (e.g. configuración de frecuencia de muestreo para los sensores o configuración de la red). Los nodos de WSN se conectan en forma inalámbrica con una topología tipo *mesh*[7] o árbol[8], permitiendo así obtener medidas distribuidas en grandes áreas.

La potencialidad del uso de estas redes en aplicaciones agrícolas fue demostrada a través de un proyecto conjunto entre el Grupo de Microelectrónica del IIE-FING-UdelaR y el INIA [9], denominado *FPTA 280 - SIMPA: Sensores Inalámbricos para Manejo Informado de Producciones Agraria (Diciembre/2011)* [5], donde se probaron y evaluaron dos redes de sensores inalámbricos aplicadas a la producción citrícola en la Quinta 1 de Milagro S.A. situada en Chapicuy en el departamento de Paysandú, Uruguay.

[6] Llamamos modo *sleep* a un estado de los componentes en el que se tiene un consumo de corriente menor al que se tiene en modo activo, o en trabajo normal.
[7] La topología *mesh* es una topología de red en la que cada nodo está conectado a todos los demás, permitiendo llevar los mensajes de un nodo a otro por distintos caminos.
[8] La topología árbol es una topología de red en la que se parte de un nodo central (*sink* en este caso) y se va ramificando la conexión hacia nodos hijos.
[9] Instituto Nacional de Investigación Agropecuaria, Uruguay

1.4. Trabajos relacionados

En la primer etapa del proyecto se buscaron artículos científicos referidos a la utilización de tecnologías de WSN con capacidad visual en detección de plagas y agricultura de precisión.

En la Università degli Studi di Milano se implementó un sistema de detección de insectos en plantaciones de Vincas [6] mediante un sistema compuesto por un nodo TelosB [7] el cual posee una radio CC2420 [8] basada en la norma IEEE 802.15.4 [9] y una cámara C328R del fabricante Comedia capaz de realizar compresión JPEG por hardware. El sistema utiliza TinyOS como RTOS y la topología de red utilizada es del tipo lineal (funciona de manera similar a una rama en una topología del tipo árbol). Las imágenes son procesadas en una PC (a la cual está conectada el nodo *sink*). Primero se le aplica un filtro a la imagen para eliminar ruido, luego se binariza la imagen con el método Otsu [10], se identifica los píxeles conexos que están por debajo del umbral del fondo y por último se calcula los píxeles que ocupa cada conjunto descartándose aquellos que no se encuentren en el rango de 2 a 30 píxeles.

En [11], Lloret y otros implementan un sistema WSN para monitoreo de viñedos realizando procesamiento de imágenes. A diferencia del anterior utilizan routers Netgear WNDR3700 de tecnología IEEE 802.11 (Wi-Fi), a los cuales se los reprograma con el firmware OpenWRT [10]. La cámara es una Hercules Webcam Classic conectada mediante USB al router y es capaz de rotar 360^o gracias a un motor de pasos controlado por un microcontrolador PIC que recibe órdenes desde el router mediante comunicación serial RS232.

Debido a las características de la cámara y la capacidad de procesamiento del router, el sistema es capaz de procesar las imágenes en el espacio RGB detectando hojas en mal estado a partir del color y el tamaño de las mismas.

En lo que respecta a producciones nacionales en WSN, específicamente proyectos realizados en la Facultad de Ingeniería, Universidad de la República se encuentran el FPTA-280 SIMPA, DIAMETRONCO [12], RSIS [13], entre otros donde se exploran diferentes aplicaciones de WSN.

1.5. Objetivo general

El objetivo general del proyecto consiste en la implementación y evaluación de un sistema para procesamiento de información visual en nodos de redes de sensores inalámbricos. Se pretende que éste realice la adquisición, un reducido procesamiento de imágenes y su transmisión a un PC remoto para registrar los insectos capturados.

Dicho sistema estará dentro de trampas tipo Delta con feromonas ISOMATE® destinadas a capturar poblaciones de *Cydia Pomonela* y *Cydia Molesta* y estará compuesto por:

i) Un nodo de WSN,

ii) cámara(s) para obtener imágenes,

iii) sistema de iluminación,

[10] Distribución de Linux para routers y puntos de acceso

iv) hardware adicional para el manejo de la alimentación de la cámara y la iluminación,

v) software para gestionar la actividad del nodo (comprende los procedimientos para la adquisición y procesamiento de imágenes, actividad general del nodo, transmisión por la radio del nodo).

1.6. Alcance

1. Se estudiarán ventajas y desventajas que ofrecen las distintas tecnologías para implementar el sistema propuesto, se evaluará cuáles son las más adecuadas y se implementará un prototipo. Se seleccionará una cámara, un nodo (junto a su RTOS) y el sistema de iluminación más apropiado para la correcta obtención de las imágenes.

2. Se estudiarán y se identificarán posibles técnicas de procesamiento de imágenes capaces de ser implementadas por un nodo (algoritmos de manipulación de las mismas, análisis de los ángulos de captura, foco, e iluminación entre otros).

3. Se implementará únicamente una comunicación punto a punto desde un nodo ubicado en una trampa hacia el nodo *sink* conectado a un PC.

4. Se elaborará un estudio mediante simulaciones para comprender el comportamiento y detectar posibles problemas del protocolo MAC y de las capas superiores que se seleccionarán para la transmisión imágenes.

1.7. Criterios de éxito

Relacionado con los **Alcances - ítem 1**:

1. Construir un prototipo funcional del sistema propuesto, testeado y caracterizado en condiciones controladas. El mismo constará de:

   - Cámara.
   - Nodo.
   - Encapsulado.
   - Dispositivo de iluminación de ser necesario.
   - Trampa Delta.

   El mismo deberá ser capaz de tomar una imagen, hacer un procesamiento a determinar de la misma usando alguno de los algoritmos estudiados y transmitirla por la radio hacia una computadora. También deberá ser capaz de recibir una imagen desde la computadora y almacenarla por un tiempo determinado.

Relacionado con los **Alcances - ítem 2**:

2. Implementar un algoritmo de procesamiento de la imagen en el nodo y sacar la misma por la radio.

Relacionado con los **Alcances - ítem 3**:

3. Lograr transmitir la imagen del interior de la trampa con polillas sin errores de un nodo al otro conectado en un PC (visualizar la imagen en el PC correctamente.)

Relacionado con los **Alcances - ítem 4**:

4. Evaluar y determinar si la subcapa MAC y capas superiores soportan el tráfico de imágenes. Identificar si existen problemas.

Otros criterios de éxito:

5. Cumplimiento de los requerimientos de los especialistas (entómologos) relativo a la calidad de la imagen, la cual debe permitir identificar y contar los insectos plagas sin lugar a duda.
6. Tiempo de vida de las baterías del sistema superior a 6 meses.

1.8. Actores

Los actores involucrados en el proyecto son:

- Equipo del proyecto.
- Cooperativa JUMECAL [11]
- Tutores del proyecto.
- Grupo de microelectrónica del IIE-FING-UdelaR.

1.9. Supuestos

Se trabajará bajo los siguientes supuestos:

- Existe un acuerdo de cooperación entre el JUMECAL y el IIE-FING-UdelaR, el cual se sostendrá al menos un año a partir de la fecha. De este acuerdo surgen algunos de los recursos materiales para el proyecto.

[11] JUventud MElilla Cooperativa Agraria R. Ltda. `http://www.jumecal.com.uy/`

- JUMECAL proporcionará ayuda de técnicos e información para el proyecto.
- Las trampas proporcionadas son un medio efectivo para capturar especímenes y entre imágenes consecutivas la trampa no colmará su capacidad.
- Las condiciones climáticas, sobre todo temperatura y humedad ambiente, son las típicas de los cultivos en territorio nacional
- Hay disponibilidad de nodos en el Grupo de Microelectrónica del IIE-FING-UdelaR.
- El sistema operativo (Contiki OS [14] o TinyOS) está disponible para los nodos y funciona correctamente.
- Existen recursos económicos suficientes para comprar los materiales (cámaras, componentes electrónicos, construcción de PCBs)

1.10. Restricciones

- El sistema a desarrollar, que incluye el nodo con la cámara incorporada, debe ser alimentado por baterías económicas y accesibles en el mercado. Por lo tanto el diseño electrónico, la selección de componentes y las metodologías usadas deben resultar en un diseño que requiera reducido consumo de corriente. Esto es uno de los requerimientos más restrictivos a considerar durante el proyecto. Es deseable obtener una vida útil de las baterías superior a seis meses.
- El costo del sistema total no debe superar los 300 USD, por lo que se debe minimizar el costo asociado a la cámara y a su interfaz.
- Los protocolos más usados en las redes de sensores inalámbricos no fueron diseñados para transportar grandes cantidades de información, por lo tanto se deberá realizar una evaluación mínima del desempeño de los protocolos al implementar estas funciones, buscando alternativas en caso de aparecer problemas de excesivo *throughput*[12].
- No se realizará identificación automática de las plagas, pero la imagen deberá ser manipulada en algún sentido y criterio para optimizar la transmisión, evitando perder la información que permita identificar los insectos plagas por un humano al reconstruir la imagen en una computadora.
- El plazo de entrega de la documentación o del proyecto finaliza el 30/4/2014.
- Cada integrante del grupo de proyecto cuenta con una disponibilidad de 14 horas semanales, por semana con actividad (i.e. equivalente a 35 créditos al año tomando en cuenta las semanas de no actividad por pausas para parciales, exámenes y vacaciones.)

1.11. Diagrama de bloques del sistema

El sistema debe ser capaz de tomar dos imágenes diarias de una trampa para polillas *Cydia Pomonella* y *Cydia Molesta* y transmitirlas por la red hacia un nodo *sink* conectado a un PC. Para

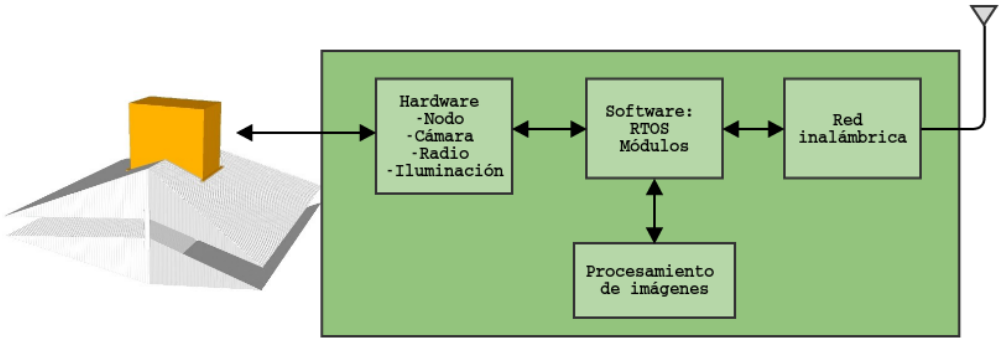


Figura 1.2: Diagrama de bloques del sistema

lograr dicho cometido se propone el sistema que se observa en la figura 1.2. La implementación de cada bloque y la interacción entre ellos supone un desafío técnico a nivel de hardware y software.

El bloque *Hardware* se compone de todos los dispositivos electrónicos necesarios para la implementación del sistema y el cumplimiento de sus funcionalidades. Estos son: un nodo, una cámara digital, sistema de iluminación, circuito interfaz y alimentación. Es además fundamental que el gabinete donde se colocará el prototipo sea estanco y a su vez simple de manipular para permitir realizar tareas de mantenimiento (e.g. cambio de baterías, limpieza, etc.). Detalles de la implementación de este bloque se pueden encontrar en el capítulo 2.

El bloque *Software* consiste en los módulos implementados sobre el RTOS que controlan los recursos hardware y realizan el procesamiento necesario para capturar, almacenar, procesar y transmitir las imágenes. Este bloque se encarga además de llevar al sistema a estados de bajo consumo de corriente y gestionar la red. Los detalles de diseño e implementación se encuentran en el capítulo 3.

El bloque *Procesamiento de Imágenes* corresponde particularmente al estudio e implementación de algoritmos de procesamiento de imágenes en WSN. Dada la importancia de esta funcionalidad del sistema, se trató como un bloque en sí mismo y se separó de la implementación de *Software*. Mediante el procesamiento de las imágenes capturadas se logran comprimir los datos, facilitando la detección y disminuyendo el caudal de datos que se transportarán por la red, aumentando así la autonomía del sistema. Más detalles de este bloque se encuentran en el capítulo 4.

El bloque *Red* comprende el análisis, diseño e implementación del *stack* de comunicaciones mediante el cual se logra realizar una red de sensores inalámbricos. Una exigencia en las WSN es alcanzar un bajo consumo para garantizar autonomía de varios meses a los nodos, por esto se buscarán protocolos de capa de enlace y capa de red que se adecúen a estas exigencias. La descripción de la misma se encuentra en el capítulo 5.

1.12. Trabajos publicados

Durante la realización del proyecto de fin de carrera se publicó un artículo académico [15] que será expuesto oralmente en la conferencia internacional arbitrada IEEE International Instrumentation

[12]Se define como *throughput* al cociente entre la carga útil enviada y el tiempo de transmisión

and Measurement Technology Conference [13] a realizarse el día 15 de mayo de 2014 en Montevideo, Uruguay. El artículo resume gran parte de los puntos estudiados en la presente documentación.

1.13. Descripción de los capítulos

La presente documentación está organizada de la siguiente manera:

Capítulo 1 - Introducción

El primer capítulo corresponde a una introducción al contenido del proyecto y la descripción formal del mismo. Los objetivos y los requerimientos funcionales fueron definidos en conjunto con los tutores del proyecto atendiendo las necesidades planteadas por JUMECAL. Se realiza un resumen de algunos trabajos en relación al área de investigación comprendido por el proyecto y se describe en un diagrama los bloques que integran el sistema. Estos bloques son el foco de investigación y desarrollo del proyecto, y son el desafío técnico del mismo.

Capítulo 2 - Selección de hardware

El segundo capítulo describe los componentes hardware que integran el sistema: plataforma con microcontrolador, radio, cámara, LEDs y otra electrónica agregada. Se describe el proceso de evaluación y selección de los mismos en relación de los requerimientos funcionales del proyecto. Por otra parte se describe el diseño del gabinete y la modificación al techo de la trampa, necesaria para que el piso de la trampa quede comprendido en el área de visión del lente de la cámara.

Capítulo 3 - Diseño e implementación del Software

El tercer capítulo consta de una descripción del diseño y la implementación software del sistema. En las primeras secciones se realiza una introducción a los RTOS y una descripción de los RTOS considerados, TinyOS y Contiki OS, haciendo hincapié en el último. En las restantes secciones se detallan los módulos que componen la aplicación implementada sobre Contiki OS y el proceso de ejecución de pruebas y depuración del código.

Capítulo 4 - Procesamiento de imágenes

El cuarto capítulo describe los algoritmos utilizados para procesar en el nodo las imágenes JPEG obtenidas desde las cámara. Se realiza una breve introducción a la compresión JPEG y al algoritmo de clasificación lineal Fisher (FLD), en el cual se basa el diseño y la implementación de nuestra aplicación. Por último se describe la implementación del algoritmo y los resultados obtenidos a partir de simulaciones en Matlab y pruebas en el nodo.

Capítulo 5 - Análisis y diseño de la red

En el quinto capítulo se realiza un análisis de algunos protocolos de comunicación existentes para WSN, principalmente aquellos de capa de enlace y capa de red implementados en Contiki OS que se ajustan a las necesidades funcionales del sistema. Se justifica la elección de un protocolo de ciclo de trabajo de la radio (RDC) y del protocolo de red dentro de la librería Rime del sistema operativo mediante simulaciones en Cooja y pruebas de laboratorio.

[13] URL: http://imtc.ieee-ims.org/

Capítulo 6 - Caracterización y ensayos

En el sexto capítulo se describen los procedimientos, resultados y análisis de las medidas de perfiles de corriente, medidas de alcance de las radios y pruebas de iluminación. Se realizan además las estimaciones de autonomía del sistema para las baterías consideradas y distintas configuraciones del sistema.

Capítulo 7 - Conclusiones y trabajo futuro

En el séptimo y último capítulo se exponen conclusiones y aprendizajes del trabajo realizado. En la sección de trabajo futuro se mencionan aspectos pendientes o mejoras del sistema para seguir desarrollando.

Capítulo 2

Selección de Hardware

Resumen

En este capítulo se encuentran descritos los componentes hardware estudiados y seleccionados para la realización de este proyecto. El sistema se compone de una plataforma con microcontrolador, radio (en algunos casos incluida en la plataforma), cámara, circuito interfaz, sistema de iluminación y sistema de alimentación. El lector podrá encontrar una sección dedicada a cada uno de estos componentes.

El sistema elegido finalmente consta de un nodo TelosB compatible CM3000 [16] (contiene un MCU MSP430 [17] y una radio CC2420), la cámara Linksprite LS-Y201 [18], dos LEDs amarillos para el sistema de iluminación, dos switches FPF2004 [19] para controlar la alimentación de la cámara y de la iluminación. La alimentación del sistema se realiza a través de dos baterías AA.

Contenido

2.1. Introducción

El diseño del hardware a utilizar en el proyecto consistió en una de las principales tareas desarrolladas en el transcurso del mismo. Como se observa en el diagrama de bloques hardware del sistema en la figura 2.1, el sistema está compuesto por: un nodo (compuesto por una plataforma con MCU y radio), una cámara, el sistema de iluminación, un circuito interfaz y la alimentación.

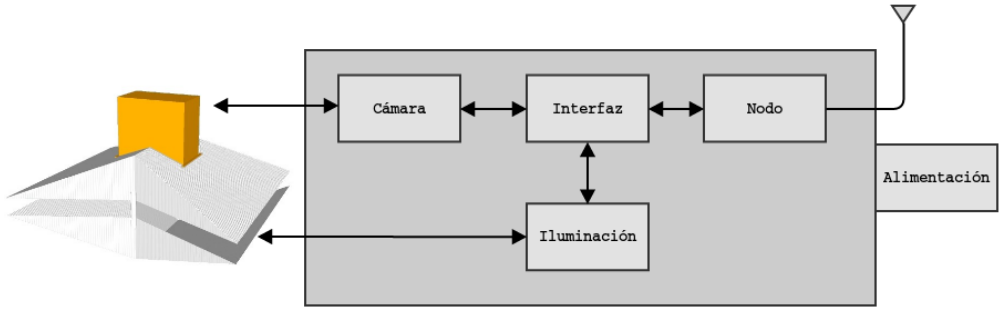


Figura 2.1: Diagrama de bloques hardware del sistema

El *Nodo* está constituido por un MCU, una radio y opcionalmente otros componentes como ser FTDI para programar el nodo, sensores diversos, portapilas, entre otros. El nodo conforma el centro del sistema y sus tareas son comunicarse con la cámara para adquirir las imágenes a través de alguna de sus interfaces seriales o paralelas, manejar la iluminación, procesar las imágenes y enviar las mismas o los datos obtenidos a través de la red. La elección de la plataforma se encuentra en la sección 2.2.

La *Cámara* corresponde al dispositivo encargado de capturar las imágenes. Está compuesto por un sensor de imagen, un lente, interfaz de comunicación y opcionalmente hardware capaz de realizar la compresión JPEG de la imagen y/o almacenar la misma en una memoria. En el mercado existe una oferta importante de estos dispositivos, los cuales se diferencian en cuanto al tipo de sensor utilizado (CCD o CMOS), resolución, lente, rango dinámico, interfaz de comunicación, consumo de corriente y costo, entre otros. En la sección 2.3 del presente capítulo se analizan distintas cámaras y se selecciona una de ellas.

El bloque *Iluminación* corresponde al sistema capaz de mantener un nivel adecuado de luminancia al momento de sacar una foto para una correcta adquisición por parte del sistema. Está compuesto por LEDs y switches que son controlados por el nodo a través de sus pines GPIO y se encuentran encastrados al techo de la trampa. En la sección 2.5.1 se detalla el tipo y la cantidad de LEDs elegidos. La descripción del circuito que incluye los switches que manejan los LEDs y su conexión al nodo se encuentra en la sección 2.6.

El bloque *Interfaz* consta de un circuito que posee electrónica agregada para prender y apagar periféricos, específicamente la cámara y el sistema de iluminación. Además este circuito funciona como puente entre la cámara y el nodo a partir de diversos pines. Detalles de este circuito se encuentran en la sección 2.6.

Las especificaciones funcionales para el sistema fueron las siguientes:

- Autonomía del sistema en el entorno de los seis meses.
- Diez nodos en 10 hectáreas.
- Capacidad de tomar imágenes a una resolución tal que permita la correcta identificación de las polillas por parte de los operarios.
- Capacidad de procesamiento y almacenamiento para un posible procesamiento de imágenes.
- Transmitir dos imágenes diarias.

A lo largo de este documento se encontrarán varias referencias al consumo del sistema total o de un componente en particular, en todos los casos -salvo que se especifique lo contrario- se trata del consumo de corriente, ya que para una fuente con tensión constante el producto de corriente por tiempo corresponde a una medida indirecta de consumo de energía.

Dadas las exigencias energéticas del proyecto es importante que cada uno de los componentes tenga el menor consumo posible. No obstante, la transmisión de grandes cantidades de datos a distancias considerables requiere mucha energía (ver sección 6.3), por lo que es importante tener un correcto balance entre el bajo consumo y la performance del sistema. Por otra parte, es deseable que todos los componentes implementen modos de bajo consumo que sean accesibles cuando no están en uso.

Las WSN están orientadas a transmitir unos pocos paquetes, correspondiente algunas centenas de bytes de datos tomados de sensores (e.g. temperatura, humedad, etc.). En aplicaciones que implican datos visuales el tamaño de los datos a propagarse por la red se encuentran en el entorno de las decenas o centenas de kilobytes, por lo que la autonomía del sistema se encuentra sujeta fuertemente al consumo de la radio. Nuevamente vale destacar que se debe buscar un balance

apropiado entre el bajo consumo y la performance, ya que cuanto mayor sea el *throughput* en transmisión mayor será el tiempo en el que el sistema se encuentre en modo de bajo consumo.

El alcance de las transmisiones depende de la potencia de transmisión de la radio del transmisor y de la sensibilidad en recepción de la radio del receptor. La potencia de las mismas se mide en dBm que equivale a $20log\left(\frac{P_{out}}{P_{ref}}\right)$ siendo $P_{ref} = 1\ mW$. Las distancias típicas en estas aplicaciones son de unos pocos metros hasta algunos cientos de metros [7].

Dado que el sistema debe enviar solamente dos imágenes por día, es menester que el tiempo en el que no se está tomando imágenes o transmitiendo las mismas el consumo sea el menor posible ya que también es determinante en la autonomía del sistema como se corroboró en el capítulo 6.

Una dificultad encontrada al momento de elegir la composición del sistema residió en la fuerte interdependencia que existe entre los distintos componentes. En primera instancia, no todas las plataformas de MCUs y radios están portadas a todos los RTOS. La portabilidad de las plataformas implica que existen librerías dentro del RTOS para que este último sea capaz de manejar al MCU, radio y demás periféricos incluidos en la plataforma. En segundo lugar, no todas las cámaras pueden ser controladas por cualquier MCU sino que como se expondrá más adelante en este capítulo, algunas cámaras necesitan de interfaces paralelo y procesadores potentes en velocidad de procesamiento.

El proceso de selección de hardware comenzó analizando las distintas plataformas con MCU, así como los módulos de cámaras y radios disponibles en el mercado. El análisis de los mismos se puede ver en las secciones 2.2, 2.3 y 2.4. En forma paralela a este proceso se investigaron los distintos RTOS considerados, concluyéndose utilizar Contiki OS. Una vez que se decidió el RTOS, la elección de la plataforma con MCU quedó ligada a la compatibilidad con el mismo. No obstante, al momento de elegir la composición del sistema, varias de las plataformas analizadas estaban en proceso de ser portadas a Contiki OS, por lo que se mantuvo su análisis por si en un futuro se decidiera cambiar de plataforma.

Luego de decidirse la plataforma con MCU, la elección de la cámara y la radio quedaron prácticamente determinadas por la compatibilidad con la plataforma. Los resultados del sistema total utilizado se pueden ver en la sección 2.5.

2.2. Plataformas con microcontroladores

2.2.1. Introducción

Las plataformas con microcontroladores embebidos consisten en placas de expansión que permiten programar a los MCU a través de FTDI o JTAG y a su vez acceder a todos los pines del mismo. En muchos casos poseen además sensores de distinta índole, sistema de alimentación, comunicación USB, RS232 o Ethernet, LEDs, pantallas LED y/o radios en el propio circuito integrado. Cuando las plataformas poseen una radio embebida o conectada se refiere a ellas como nodos de WSN, puesto que permiten ser utilizados para armar una red y son los componentes centrales en cualquier aplicación de WSN. [20]

Los MCU utilizados son generalmente restrictivos en lo que refiere a su capacidad de procesamiento y almacenamiento. Su estructura básica incluye:

- Un CPU de 4 a 64 bit, con frecuencias de reloj desde pocos MHz hasta algunas centenas de MHz.
- Unos pocos kB de memoria RAM para almacenamiento dinámico.
- Algunos pocos MB de memoria ROM, EEPROM o flash para almacenamiento de datos o código.
- Interfaces de comunicación serial y/o paralelo.
- Pines E/S de propósito general (GPIO).
- Conversor/es ADC para efectuar diversas medidas.
- Oscilador/es de cristal para generar señales de reloj.

En las secciones que siguen a continuación se realizará una breve descripción de las plataformas evaluadas, comparando las mismas y justificando la elección de la plataforma utilizada en el proyecto. Los criterios utilizados para evaluar las distintas plataformas fueron:

- Capacidad de procesamiento.
- Memoria ROM, RAM y flash.
- Interfaces de comunicación.
- Compatibilidad con Contiki OS o TinyOS.
- Radio embebida.
- Soporte y precio.
- Consumo.

2.2.2. TelosB

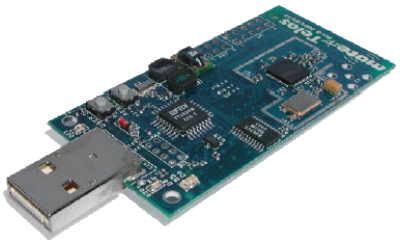


Figura 2.2: Plataforma TelosB

Figura 2.3: Plataforma CM3000 *TelosB compatible*

La plataforma TelosB que se observa en la figura 2.2 es una plataforma abierta creada en la Universidad de Berkeley, EE.UU. Está compuesto por un MCU MSP430F1611 [21], una radio CC2420 (ambos fabricados por Texas Instruments) y una memoria flash externa M25P80 [22] de ST Microelectronics. Existen varias versiones de esta plataforma que forman una serie de placas TelosB compatibles las cuales se diferencian principalmente en que algunas incluyen conector USB (ver figura 2.2), otras RS232 y otras necesitan de una placa de expansión para programarse.

Otras diferencias sustanciales entre los modelos son: las antenas (entre las cuales se encuentran las F invertidas u omnidireccionales de conexión SMA), los sensores embebidos (e.g. temperatura, humedad, luz) y los pines accesibles desde la placa. Estos nodos presentan la ventaja de estar completamente portados a los RTOS considerados: Contiki OS y TinyOS.

Una de las plataformas TelosB compatibles disponibles es la CM3000 que se observa en la figura 2.3. Esta última tiene una antena externa omnidireccional de conexión SMA de 5 dBi de ganancia y una impedancia de 50 Ω. Además no posee sensores de temperatura, humedad ni luz y se conecta al PC mediante una interfaz externa denominada USB1000 [23] que contiene el FTDI. Posee también otra interfaz externa llamada EX1000 [24] que incorpora un chip MAX232 [25] y acceso a pines del ADC del MCU. Una desventaja del CM3000 es que los esquemáticos están protegidos por licencias, pero por el otro lado la placa es más modular ya que se pueden agregar placas extensoras según la necesidad. El precio de las distintas plataformas TelosB compatibles se encuentra en el rango de 90 a 125 USD [26].

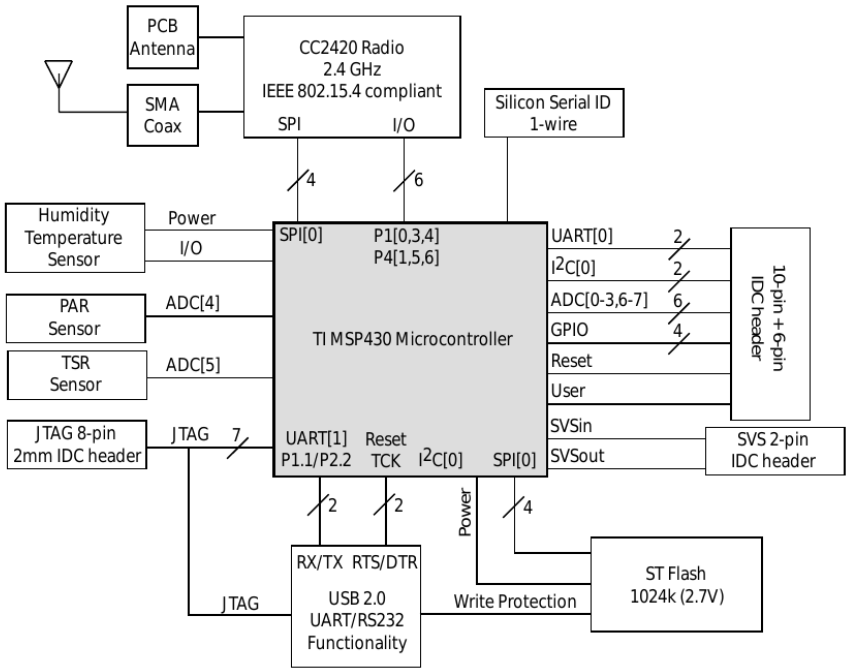


Figura 2.4: Diagrama de bloque del TelosB

En el diagrama de bloques de la plataforma TelosB de la figura 2.4 se puede observar el conexionado de la plataforma. La distribución de puertos del MCU de interés para el proyecto es la siguiente:

- *Memoria flash externa*: SPI0 (a través de la USART0) con puertos `P3[0..3]` y puertos `P4[4,7]` como *chip select* y *hold* respectivamente.
- *Radio CC2420*: SPI0 y puertos `P1[0,3,4]`, `P4[1,2,5,6]`.
- *Conector USB*: UART1 (a través de la USART1) puertos `P1.1, P2.2 P3[6,7]`.
- *Conector JTAG*: Para programar y realizar depuración *on-target*.
- *GPIOs*: E/S de propósito general en puertos `P2[0,1,3,6]`.

Resulta importante destacar el hecho de que en estos nodos el bus de la USART0 se encuentra compartido entre la radio y la memoria flash externa. Dado que la utilización de estos periféricos es prioritaria, no está recomendado utilizar dicha interfaz o de lo contrario se debe implementar un árbitro por software que administre estos recursos.

Microcontrolador MSP430F1611

El MSP430F1611 incorpora un CPU RISC de 16 bit, periféricos y un sistema de reloj utilizando arquitectura del tipo von-Neumann [1]. Las principales características de interés son:

- Sistema de reloj capaz de generar señales entre 32 kHz y 8 MHz.
- 10 kB de memoria RAM y 48 kB de ROM.
- Comunicación serial por dos USART . La USART0 puede funcionar como UART, SPI o I^2C mientras que la USART1 solo funciona como UART.
- Varios GPIO.
- Cuatro modos de bajo consumo (llamados LPM[0,2,3,4]) donde se alcanzan consumos de corriente de menos de 1 μA.
- Consumo de corriente en modo activo operando a 1 MHz de 500 μA.

Radio CC2420

El chip CC2420 es una radio de 2,4 GHz orientada a conexiones inalámbricas que cumplen el estándar IEEE 802.15.4. La radio está orientada a paquetes y admite una velocidad de transmisión de hasta 250 $kbps$ utilizando DSSS. Tiene una potencia de transmisión configurable entre -25 y 0 dBm, donde el consumo de corriente declarado por el fabricante es entre $8,5$ y $17,4$ mA respectivamente. En cuanto al consumo de corriente en recepción, se declara que es de $19,7$ mA. Para comunicarse con la radio, el MCU utiliza SPI y algunos GPIO.

Como se mencionó anteriormente, en la familia TelosB compatible hay dos tipos de antena: externas de conexión SMA y F invertida en el PCB. Para las antenas externas se tiene un conector hembra SMA, mientras que la F invertida es un monopolo hecho con una pista que se dobla para ser paralela al plano de tierra.

Memoria flash M25P80

El chip M25P80 es una memoria flash de 1 MB de capacidad con interfaz de comunicación SPI. Los 1024 kB están divididos en 16 segmentos de 64 kB cada uno. En las plataformas TelosB compatibles, la radio CC2420 y esta memoria comparten el bus de SPI0 por lo que es necesario un protocolo de arbitraje para manejar los dispositivos correctamente.

[1] Utiliza el mismo espacio de direcciones y el mismo bus para instrucciones y datos.

El fabricante declara un consumo de corriente de 4 y 20 mA de lectura y escritura respectivamente, mientras que la corriente consumida en modo *Deep Power Down* es de 8 μA nominales. La memoria provee además un modo de bajo consumo con el cual el consumo de corriente desciende a 1 μA nominal.

2.2.3. STM32W-RFCKIT

La STM32W-RFCKIT [27] es una plataforma fabricada por ST Microelectronics integrada por un SoC STM32W108 [28] el cual posee un CPU ARM Cortex-M3 [29], una radio de 2,4 GHz compatible con la norma IEEE 802.15.4 y memoria flash con capacidad de 128, 192 o 256 kB (dependiendo de la versión). La plataforma incluye botones de usuario, LEDs, sensor de temperatura y un conector miniUSB con el cual se puede programar el nodo mediante un PC tal como se observa en la figura 2.5. La antena en este nodo es también un monopolo hecho como una pista en el PCB. Por otra parte la plataforma está portada solamente a Contiki OS. El precio de la misma es de 38 USD. [30].

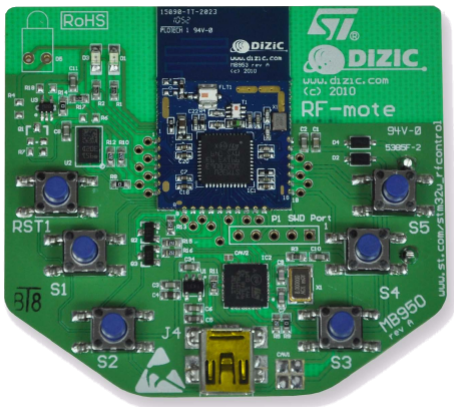


Figura 2.5: Plataforma STM32W-RFCKIT

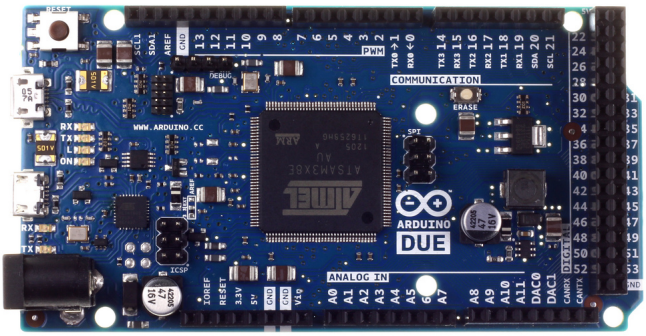


Figura 2.6: Plataforma Arduino Due

SoC STM32W108

Como se mencionó anteriormente la familia de SoC STM32W108 está integrada por un CPU ARM Cortex-M3, una radio de 2,4 GHz compatible con la norma IEEE 802.15.4 y memoria flash con capacidad de 128, 192 o 256 kB. El Cortex-M3 es un procesador de 32 bit con arquitectura Harvard modificada [2] que opera a 6, 12 o 24 MHz. La operación normal es a 12 MHz ya que el consumo aumenta conforme se incrementa la frecuencia y la operación a 6 MHz sólo es posible mientras no se esté utilizando la radio puesto que esta necesita una fuente de reloj de 12 MHz.

Otras características del SoC que resultan de interés para el proyecto son:

- 8, 12 o 16 kB de RAM.
- 24 GPIOs.

[2] Presenta un único espacio de direcciones pero mantiene buses independientes para datos e instrucciones.

- Dos USARTs (llamadas SC1 y SC2). La SC1 funciona como UART, SPI o I^2C mientras que la SC2 funciona como SPI o I^2C.
- Cuatro modos de bajo consumo con los que se alcanzan consumos de corriente de hasta 1 μA.
- El consumo de corriente en modo activo es de 7 mA a 12 MHz.
- Potencia de la radio en transmisión configurable entre -55 y 8 dBm con un consumo de corriente de 26 mA. El consumo en recepción es de 22 mA.

2.2.4. Arduino Due

Aruino Due es una plataforma del proyecto Arduino [31] que contiene un MCU AT91SAM [32] del fabricante ATMEL basado en un microprocesador ARM Cortex-M3 de 32 bit. Está diseñada de forma que todos los pines sean accesibles para el usuario e incorpora un conector micro USB, LEDs y botones de usuario como se observa en la figura 2.6. Por un lado esta plataforma tiene la ventaja de contar con un gran soporte, una comunidad activa y la posibilidad de utilizar *shields* de Arduino para cámaras o módulos de radio. Por el otro lado tiene las desventajas de no tener incorporado un sistema de comunicación inalámbrico, no estar portado a Contiki OS y el consumo de corriente típico del MCU en modo activo es de 80 mA a 84 MHz. El precio de esta plataforma es de 54 USD [33].

Microcontrolador AT91SAM

Este MCU incorpora un CPU ARM Cortex-M3 de 32 bit con un reloj de 84 MHz. Además posee 96 kB de memoria RAM y 512 kB flash. Otras características destacables son entre otros:

- Hasta cinco interfaces UART, seis SPI y dos I^2C.
- Hasta 54 GPIO, 12 pines de entrada analógica y 2 de salida para ADC.
- Interfaz USB embebida.
- Interfaz Ethernet 10/100.
- Interfaz HSMCI para conectar tarjetas de memoria SD y MMC.
- Interfaz EBI para memorias flash externas.
- Interfaces para manejar periféricos de E/S paralelo (PIO).
- Cuatro modos de bajo consumo donde se alcanza un consumo de corriente mínimo de 2,5 μA.

2.2.5. STM32F4 Discovery

Desarrollada por ST Microelectronics, la STM32F4 Discovery [34] es una plataforma basada en un MCU STM32F407 [35] de la misma empresa, integrado por un CPU ARM Cortex-M4 de 32 bit que opera a una frecuencia de hasta 168 MHz y posee una unidad de punto flotante (FPU).

A su vez el MCU incorpora 192 kB de memoria RAM y 1 MB de memoria flash. La plataforma incorpora además un conector microUSB, LEDs y botones de usuario como se observa en la figura 2.7. Su precio es de 15 USD [36].

Las desventajas de esta plataforma, al igual que la Arduino Due es que no tiene una radio incorporada, el consumo de corriente típico en modo activo es de 40 mA a 168 MHz y que no esta portado en Contiki OS ni TinyOS.

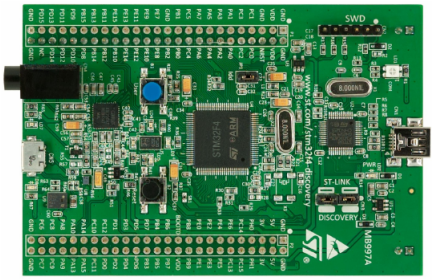


Figura 2.7: Plataforma STM32F4 Discovery

Micocontrolador STM32F407

El MCU STM32F407 contiene un CPU ARM Cortex-M4 de 32 bit, también con arquitectura Harvard modificada, capaz de operar a una frecuencia máxima de 168 MHz. Como se mencionó anteriormente posee una FPU, lo que le permite realizar operaciones en punto flotante a mayor velocidad que los CPU que emulan esta función. A su vez este MCU incorpora 192 kB de memoria RAM + 4 kB de memoria RAM para respaldos y 1 MB de memoria flash. Dentro de las restantes características destacables se encuentran:

- Controlador para memorias estáticas externas.
- Reloj dedicado para generar el *tick* del sistema en un RTOS.
- Hasta tres interfaces I^2C, seis UARTs y tres SPI.
- Interfaz para manejar tarjetas de memoria SD/MMC.
- Interfaz Ethernet 10/100 con una unidad para el cálculo de Códigos de Reduncancia Cíclica.
- Dos interfaces USB.
- Una interfaz para cámaras digitales, tanto módulos como sensores CMOS a través de una interfaz de 8 a 14 bits paralelos que opera a 54 MB/s.
- Modos de bajo consumo que alcanzan un consumo mínimo de corriente de aproximadamente 1 μA.

2.2.6. Resumen de características

En la tabla 2.1 se resumen las características de las plataformas analizadas en el presente capítulo. Cabe destacar que para los modos *sleep* se consideran aquellos en los que un reloj de tiempo real permanece encendido.

Características	STM32W	Arduino Due	STM32F4 Discovery	TelosB
Microcontrolador	STM32W108xx	AT91SAM3X8E	STM32F407	MSP430F1611
Frecuencia de procesador	24 MHz	84 MHz	168 MHz	8 MHz
Memoria flash	256 kB	512 kB	1 MB	1 MB
Memoria RAM	8 kB	96 kB	192 kB	10 kB
Compatible con Contiki	SI	NO	MCU en proceso de ser portado	SI
Radio	Incorporada	No	No	Incorporada
Interfaces de Comunicación	UART, I^2C y SPI	USB, UART, SPI, I^2C, Ethernet, HSMCI y PIO	USB, UART, SPI, I^2C, Ethernet, Cámara, HSMCI	UART, SPI e I^2C
Consumo μC modo *sleep* aprox.	1 μA	2,5μA	1 μA	2 μA
Precio (USD)	40	54	15	96 a 125

Tabla 2.1: Tabla de plataformas de desarrollo evaluadas

2.2.7. Criterios de análisis de las plataformas

El cuadro 2.1 muestra un resumen de las especificaciones de las plataformas consideradas para el proyecto. Para comparar y elegir entre las distintas plataformas nos basamos en los criterios mencionados anteriormente: capacidad de procesamiento, capacidad de almacenamiento en memoria RAM y flash, interfaces de comunicación, compatibilidad con Contiki OS o TinyOS, radio embebida, soporte, precio y consumo. A cada item de la plataforma se le asignó un puntaje del 1 al 3.

Capacidad de procesamiento

Esta característica refiere a la potencia de procesamiento de la plataforma. Los indicadores considerados fueron la velocidad del procesador del MCU y el tamaño de palabra. La cantidad de instrucciones por segundo no están caracterizadas para todos los MCU por lo que no fueron utilizadas. Los criterios fueron:

1 - Procesador de 16 bit, frecuencia de reloj menor a 24 MHz

2 - Procesador de 32 bit, frecuencia de reloj menor o igual a 24 MHz

3 - Procesador de 32 bit, frecuencia de reloj mayor a 24 MHz

Memoria RAM y flash

Esta característica refiere a la cantidad de memoria RAM y flash disponibles para datos y código. Mayor cantidad de memoria RAM disponible posibilita ejecutar algoritmos más complejos (e.g. operaciones con matrices) sin necesidad de utilizar memoria flash como espacio de intercambio de datos (o *swapping*). En cuanto a la memoria flash es necesario disponer de la mayor cantidad posible para poder almacenar varias imágenes. Los criterios utilizados fueron:

1 - Memoria RAM menor a 10 kB y memoria flash menor a 512 kB.

2 - Memoria RAM entre 10 y 96 kB, memoria flash mayor o igual a 512 kB.

3 - Memoria RAM mayor a 96 kB y memoria flash mayor o igual a 512 kB.

Interfaces de comunicación

En esta característica se analiza la cantidad y variedad de interfaces disponibles en las plataformas ya sean seriales, paralelas, específicas para redes o cámaras entre otras. Estas son necesarias para poder conectar una cámara digital, una radio (de ser necesario) y poder comunicarse con un PC. Tener una interfaz de comunicación de 8 o 10 bits paralelo posibilita el manejo de una mayor variedad de cámaras. La puntuación otorgada en este caso fue:

1 - Al menos una interfaz serial ya sea SPI, I^2C o UART disponible para el usuario.

2 - Más de una interfaz serial y al menos una interfaz paralelo.

3 - Más de una interfaz serial y paralelo, interfaz exclusiva para cámara digital, etc.

Compatibilidad con Contiki OS y TinyOS

Esta característica refiere al nivel de portabilidad de las distintas plataformas en alguno de los RTOS considerados, es decir si existen librerías que permiten manejar las plataformas completas o los MCU unicamente. Por lo tanto, el criterio considerado fue:

1 - Ni plataforma ni MCU portados.

2 - MCU portado o plataforma portada en uno de los dos RTOS.

3 - Plataforma completamente portada en ambos.

Radio embebida

Esta característica evalúa el hecho de que la plataforma incluya una radio compatible con la norma IEEE 802.15.4 siendo innecesario comprar y configurar una radio externa a la plataforma. Para poder distinguir entre aquellas plataformas que tienen radio, se comparó la potencia de transmisión, ya que cuanto mayor es la potencia, mayor es el alcance. El puntaje asignado corresponde al siguiente criterio:

1 - La plataforma no incluye radio.

2 - La plataforma incluye una radio con potencia de transmisión menor o igulal 0 dBm.

3 - La plataforma incluye una radio con potencia de transmisión mayor a 0 dBm.

Soporte y experiencia

En esta característica se analiza el soporte de la plataforma. Por soporte se entiende el servicio de soporte técnico que provee el vendedor y en este caso se incluye también la existencia de una comunidad activa que utilice la plataforma. Por experiencia previa se considera la experiencia de los integrantes del proyecto así como de los docentes el Grupo de Microelectrónica del IIE-FING-UdelaR. Dada la continua innovación es probable que no se tenga experiencia en la mayoría de las plataformas consideradas por lo que se le dio más peso relativo al soporte frente a la experiencia. Los criterios utilizados fueron:

1 - No existe soporte ni experiencia en el uso de la plataforma.

2 - Existe soporte pero poca o ninguna experiencia en el uso de la plataforma.

3 - Existe soporte y experiencia en el uso de la plataforma.

Precio

Esta característica refiere al costo de las distintas plataformas. Se debe poner en consideración que para las plataformas sin radio embebida se deberá comprar un módulo de radio para el cual se estima un precio de 30 USD. El puntaje asignado corresponde al siguiente criterio:

1 - Precio plataforma + radio mayor a 90 USD.

2 - Precio plataforma + radio entre 50 y 90 USD.

3 - Precio plataforma + radio menor a 50 USD.

Consumo

Esta característica evalúa el consumo de corriente del MCU. Dado que el consumo de corriente en los modos de bajo consumo son muy similares entre sí (del orden del μA) se consideraron los consumos de corriente en modo activo de los mismos.

1 - Consumo de corriente en modo activo mayor a 50 mA.

2 - Consumo de corriente en modo activo entre a 10 y 50 mA.

3 - Consumo de corriente en modo activo menor a 10 mA.

2.2.8. Análisis de las plataformas

En las figuras 2.8, 2.9, 2.10 y 2.11 se pueden observar los puntajes de cada plataforma para los distintos criterios considerados. Se observa que las plataformas Arduino Due y STM32F4 Discovery poseen mejores desempeños en lo que refiere a capacidad de procesamiento, memoria e interfaces.

Por lo tanto resultan más versátiles para ejecutar algoritmos complejos y al poseer mayor capacidad de almacenamiento podrían eventualmente procesar imágenes en formato matricial. Dado que ambas tienen interfaces paralelo, podrían comunicarse con la gran mayoría de cámaras digitales del mercado. Como se ha mencionado anteriormente poseen la gran desventaja de no ser compatibles con los RTOS considerados para este proyecto, no incluir una radio embebida y tener un consumo en modo activo superior a las restantes plataformas.

Otras ventajas de estas plataformas son su bajo costo y estar respaldadas por fabricantes reconocidos cuyo soporte está garantizado. Sin embargo, en el caso de la STM32F4 Discovery no existe experiencia en su utilización lo cual podría resultar en una dificultad extra a la hora de comenzar a programar.

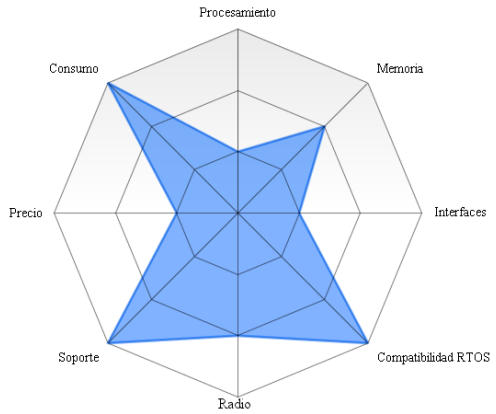


Figura 2.8: Características plataforma TelosB

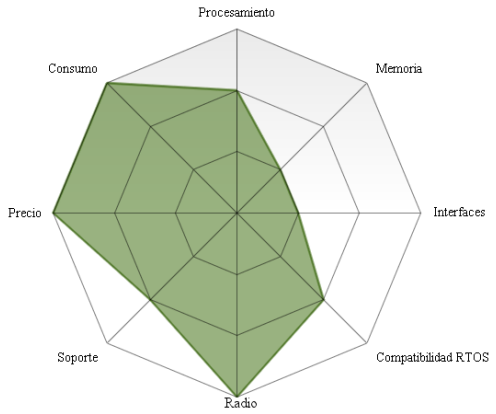


Figura 2.9: Características plataforma STM32W

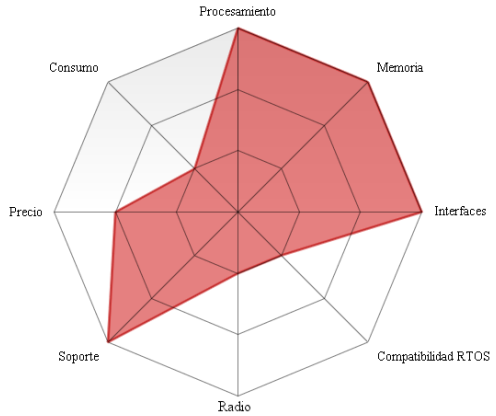


Figura 2.10: Características plataforma Arduino Due

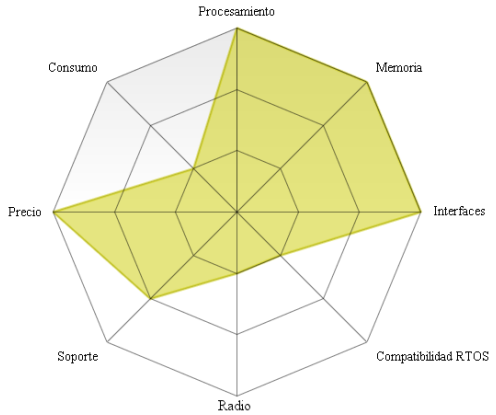


Figura 2.11: Características plataforma STM32F4

Las plataformas TelosB compatibles y STM32W tienen mejores desempeños en cuanto a la compatibilidad con los RTOS, consumo y poseer una radio embebida. Las plataformas TelosB compatibles se encuentran portadas tanto a Contiki OS como a TinyOS mientras que la STM32W solamente a Contiki OS. En el caso de las TelosB compatibles, se tiene el mayor nivel de soporte ya que se ha trabajado con la plataforma en varios proyectos de fin de carrera [12] [37], proyectos del Grupo de Microelectrónica del IIE-FING-UdelaR [5] y por nuestra parte en el proyecto final del curso de la asignatura Sistemas Embebidos para Tiempo Real de la misma institución [38].

En lo que respecta a las radios de las plataformas, el SoC STM32W108 posee una potencia en transmisión mayor a la de las TelosB compatibles a costa de un consumo de corriente levemente mayor. No obstante, para algunas plataformas TelosB compatibles existe la posibilidad de utilizar un circuito PA-LNA, con el cual se puede lograr una potencia de transmisión de 23 dBm.

Las desventajas de estos nodos es la poca memoria RAM y en el caso de la plataforma STM32W también se suma la poca capacidad de almacenamiento en memoria flash. Si bien la memoria RAM de estos nodos es admisible para la aplicación, la memoria flash disponible en la plataforma STM32W se consideró insuficiente para las cámaras analizadas en una primera instancia.

Otra desventaja de estas últimas es el hecho de que solo incluyen interfaces de comunicación seriales por lo que solo podrán conectarse cámaras digitales que tengan esa interfaz.

2.3. Cámaras

2.3.1. Introducción

Se estudiaron distintas cámaras disponibles en el mercado y se evaluaron distintas características deseadas para elegir una de ellas. La elección estuvo muy vinculada también a la selección del MCU, ya que muchas características de las cámaras dependen de las interfaces de comunicación disponibles, la capacidad de procesamiento, memoria y voltaje de alimentación del MCU. Las cámaras se componen de un sensor de imágenes CMOS o CCD, un lente y en algunos casos electrónica agregada para facilitar el almacenamiento y la comunicación de la misma.

Algunas de las características evaluadas fueron:

- Costo.
- Formato de la imagen de salida.
- Resolución de la imagen de salida.
- Interfaz de comunicación con el MCU.
- Consumo de corriente.
- Dificultad.

En cuanto al costo, se buscó que fuera lo menor posible para abaratar el costo final de todo el sistema. El formato y resolución de salida dependen fuertemente de la memoria estática y dinámica disponibles en el MCU elegido, pero para la aplicación particular buscamos una resolución mínima de 640×480 píxeles ya que consideramos que esta resolución era la mínima con la que se podían distinguir sin ambigüedad las polillas. Si bien esto se hizo en base a una suposición, luego se corroboró su validez.

Otro aspecto buscado fue que al menos soportara JPEG comprimido como formato de salida para facilitar la transmisión de las imágenes, ya que estas tienen un tamaño considerablemente menor

al de las imágenes en formato crudo[3] (entre uno y dos órdenes de magnitud). La compatibilidad con los MCU aumenta cuando la interfaz de comunicación es serial, ya que permite utilizarla con la mayoría de los MCU del mercado. Cuando la conexión es vía 8 *bits* paralelos se requiere un MCU que incluya esa interfaz o un reloj suficientemente rápido para manejarlos y la utilización de hardware adicional. La dificultad está vinculada a la disponibilidad de hojas de datos completas, disponibilidad de *drivers* ya escritos para el manejo de la cámara y necesidad o no de hardware adicional. El consumo de corriente se buscó que fuera lo menor posible.

2.3.2. Cámara Micron MT9D111

La cámara Micron MT9D111 [39] que se observa en la figura 2.12, se estudió por ser compatible con el proyecto de código abierto ArduCAM [40], el cual ofrece *shield* que hace de interfaz de comunicación entre la cámara y la plataforma Arduino. Se trata de un SoC con un sensor de imágenes CMOS con una resolución de hasta 2 MP (1600×1200) y con hardware capaz de realizar compresión JPEG. También tiene salida en formato crudo RGB y YCbCr.

El consumo de corriente promedio declarado por el fabricante es de 50 mA, con un voltaje de alimentación de 3 V. El consumo de corriente del ArduCAM *shield* no está declarado, pero no esta diseñado para aplicaciones de bajo consumo. El costo de la cámara es de 22 USD [41] y el costo del ArduCAM es de 30 USD [42], por lo que el costo total es de 52 USD.

Figura 2.12: Cámara Micron MT9D111

Esta cámara tiene como ventajas que tiene una muy buena resolución y ofrece variedad de formatos de salida. Además, al utilizarse con el ArduCAM *shield*, se facilita mucho su manejo, ya que hay bibliotecas disponibles para la plataforma Arduino. Su principal desventaja es que necesita un hardware adicional para controlarla, o un MCU con una interfaz PIO de 8 bits.

2.3.3. Cámara OV7670 AL422

La cámara OV7670 AL422 [43] que se observa en la figura 2.14, tiene un sensor CMOS OV7670 que soporta resoluciones de hasta 640×480 píxeles pero solamente en formato crudo (RGB o YCbCr). Al igual que la MT9D111, tiene una interfaz de comunicación PIO de 8 bits, pero posee una memoria FIFO AL422B [44] de 380 kB y un oscilador de cristal de 24 MHz que permite adquirir imágenes utilizando MCUs con velocidad de procesamiento reducida. Funciona a una tensión de alimentación de $3V$ y el fabricante declara un consumo de corriente de 20 mA. El costo de la cámara es de 22 USD [45].

[3]Se dice que una imagen está representada en formato crudo cuando su archivo contiene los valores numéricos de cada uno de los píxeles para los distintos canales, sin ninguna forma de compresión

Esta cámara tiene como ventaja que tiene un consumo de corriente y un costo económico significativamente menor al de las otras cámaras evaluadas. Como principal desventaja tiene que no hace compresión JPEG, por lo que la plataforma utilizada tiene que ser capaz que alojar la imagen en formato crudo (900 kB para 8 bits por canal) para posteriormente procesarla y/o comprimirla.

2.3.4. Cámara uCAM

La cámara uCam [46] que se observa en la figura 2.13, tiene un sensor CMOS OV7640 que soporta resoluciones de hasta 640×480 pixeles, tanto en formato crudo RGB como JPEG comprimido. Tiene una memoria EEPROM que permite ser controlada por un MCU vía UART (TTL o RS232) en base a comandos pre-programados. Alcanza una velocidad de 115200 *bps* y se puede alimentar a 3 V como a 5 V. Tiene un consumo promedio de corriente declarado por el fabricante de 60 mA. El costo de la cámara en el momento que se evaluó el hardware era de 60 USD [47] pero actualmente no está más a la venta.

Figura 2.13: Cámara uCAM

Figura 2.14: Cámara OV7670 AL422

Tiene como ventaja que se puede manejar con cualquier MCU con interfaz UART sin hardware adicional, ofrece variedad de formatos de salida, y tiene un consumo de corriente dentro de los parámetros buscados para nuestra aplicación. También está bien documentada, lo cual no es un detalle menor, sobre todo en el mercado de las cámaras donde la documentación suele ser muy pobre. Una desventaja es que si bien se puede programar con comandos pre-programados en la EEPROM, hay que escribir los *drivers* para manejarla. Finalmente esta cámara no fue tenida en cuenta porque los fabricantes la descontinuaron durante la realización del proyecto.

2.3.5. Cámara Linksprite LS-Y201

La cámara Linksprite LS-Y201 [18] (figuras 2.15 y 2.16) incluye un sensor CMOS OV7640 que soporta resoluciones de hasta 640×480 pixeles y un chip que realiza compresión JPEG por hardware. El formato de salida es únicamente JPEG. Al igual que la uCam estudiada en la sección 2.3.4, tiene una memoria EEPROM que le permite ser controlada por un MCU vía UART con comandos pre-programados, a velocidades de hasta 115200 *bps*. Se puede alimentar entre 3 V y 5 V y el fabricante declara un consumo de corriente promedio de 80 mA con los LEDs infrarrojos apagados y de 220 mA

con los LEDs infrarrojos prendidos. El rango de temperaturas de funcionamiento declaradas por el fabricante es de $-20°$C a $60°$C y el costo de la cámara es de 47 USD [48].

Esta cámara también tiene la particularidad de que posee iluminación propia. Tiene un sensor de luz, y cuando detecta que la iluminación no es suficiente, activa seis LEDs infrarrojos que tiene ubicados en dos columnas, una de cada lado del lente.

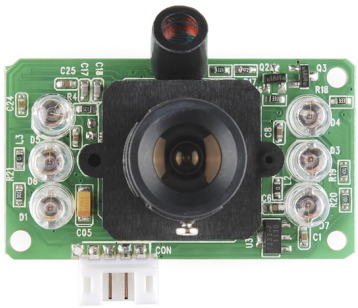


Figura 2.15: Frente de la cámara Linksprite LS-Y201

Figura 2.16: Parte posterior de la cámara Linksprite LS-Y201

La cámara tiene tres resoluciones posibles que son 160×120, 320×240 y 640×480 y para la compresión JPEG se le puede fijar el nivel de compresión con un factor configurable entre `0x00` (mínima compresión) y `0xFF` (máxima compresión). Variando los parámetros de configuración de resolución y factor de compresión se pueden obtener imágenes de tamaño muy variado, desde alrededor de 3 kB a 80 kB, lo cual puede ser muy útil dependiendo de la aplicación. El ángulo de visión del lente es de $60°$.

Como principales ventajas tiene que permite una comunicación via UART que permite ser manejada por una gran cantidad de MCUs, soporta el formato JPEG comprimido como salida, posee iluminación propia y tiene un consumo de corriente dentro de los parámetros buscados.

Tiene como desventaja que no soporta formato crudo de salida, por lo tanto se dificulta mucho el procesamiento posterior. También tiene un costo elevado respecto a las otras cámaras evaluadas en esta sección. Otra desventaja es que de utilizarse, se deben escribir los *drivers* para su manejo.

2.3.6. Criterios de análisis de las cámaras

Para comparar las cámaras se evaluaron las características mencionadas en la sección 2.3.1 y se les asignó un puntaje del 1 al 3, dependiendo de varios factores.

Costo

Para el costo se asignó el siguiente puntaje:

1. Costo mayor a 50 USD.

2. Costo entre 25 y 50USD.
3. Costo menor a 25 USD

Formato de salida

En cuanto al formato de salida se asignó el siguiente puntaje:

1. Sólo formato crudo.
2. Sólo formato JPEG comprimido.
3. Tanto JPEG comprimido como formato crudo.

Resolución

La resolución se calificó mediante el siguiente puntaje:

1. Soporta unicamente resoluciones menores a 640×480 pixeles.
2. Soporta resoluciones hasta 640×480 pixeles.
3. Soporta resoluciones mayores a 640×480 pixeles

Interfaz de comunicación

A la interfaz de comunicación con el MCU se le asignó el siguiente puntaje:

1. Interfaz PIO de 8 bits sin hardware adicional disponible.
2. Interfaz PIO de 8 *bits* pero con FIFO disponible.
3. Interfaz serial UART.

Consumo de corriente

El consumo de corriente se puntuó con las siguientes consideraciones:

1. Consumo mayor a 60 mA.
2. Consumo entre 30 mA y 60 mA.
3. Consumo menor a 30 mA.

Dificultad

Finalmente se consideró la dificultad de implementación de los *drivers* para el manejo de la cámara y se consideró la posibilidad de que ya existieran implementaciones del mismo. Se asignaron los siguientes puntajes:

1. No hay *drivers* disponibles para su manejo.
2. No hay *drivers* disponibles pero hojas de datos completas.
3. *Drivers* disponibles para el manejo.

2.3.7. Análisis de las cámaras

La comparación de las cámaras se muestra en las figuras 2.17, 2.18, 2.19 y 2.20.

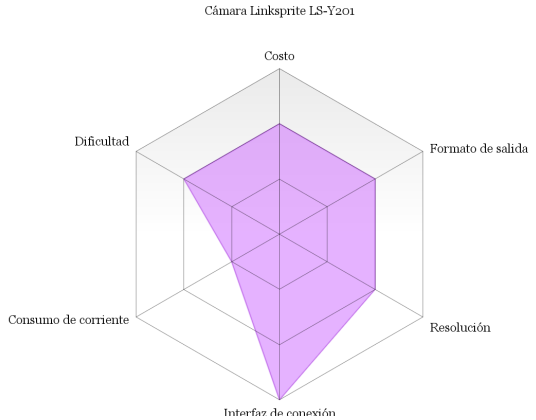


Figura 2.17: Características de la cámara Linksprite LS-Y201

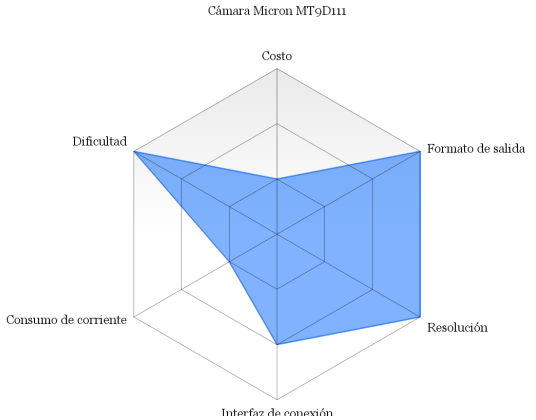


Figura 2.18: Características de la cámara Micron MT9D111

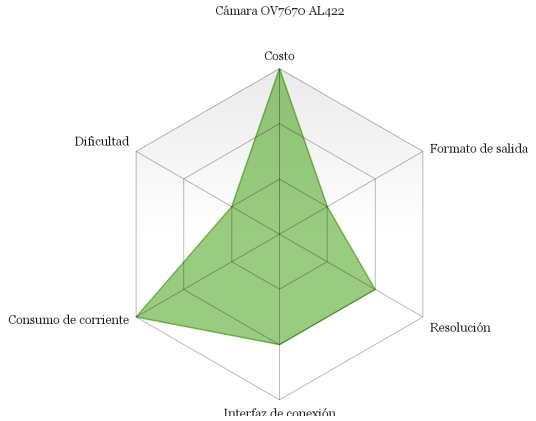


Figura 2.19: Características de la cámara OV7670 AL422

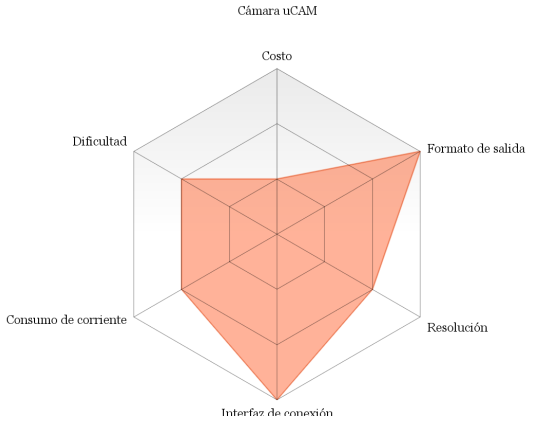


Figura 2.20: Características de la cámara uCAM

La elección final de la cámara a utilizar estuvo muy ligada a la elección de la plataforma con MCU, principalmente las restricciones surgen en cuanto a la interfaz de comunicación disponible y la capacidad de almacenamiento de la misma.

La cámara Micron MT9D111 puede ser utilizada únicamente con una plataforma Arduino Due (utilizando el *shield* ArduCAM) o el STM32F4 Discovery ya que necesita una interfaz PIO. La cámara OV7670 AL422 puede utilizarse en aquellas plataformas que tengan suficiente memoria como para almacenar la imagen en formato crudo y posean además una interfaz PIO de 8 bits como son el Arduino Due y la STM32F4 Discovery.

Las cámaras uCAM y la Linksprite LS-Y201 son las únicas que pueden ser utilizadas con todas las plataformas analizadas, en particular las TelosB compatibles y STM32W-RFCKIT que sólo tienen interfaz de comunicación UART.

2.4. Radios

2.4.1. Introducción

La transmisión de datos debe cubrir un área de 10 hectáreas con 10 nodos, transmitiendo imágenes de un tamaño aproximado de 18 kB, con el menor consumo energético posible. Las radios más utilizadas en las WSN son las compatibles con el estándar IEEE 802.15.4, ya que especifican la capa física y de acceso al medio del modelo de capas , con el objetivo de obtener conectividad inalámbrica de baja complejidad, bajo costo y ultra bajo consumo en dispositivos baratos, portables y móviles. El mismo está orientado hacia transmisiones de bajo *throughput* y baja latencia.

En el capítulo 5 se detallan modificaciones que se hicieron a los protocolos de ciclo de trabajo de la radio para intentar optimizar las transmisiones y aumentar el *throughput*.

Otra opción evaluada fue utilizar modems GPRS. Los mismos tienen la gran ventaja que permiten independizarse de todo el manejo de la red ya que utilizan la infraestructura de alguna compañía de telefonía celular. Como desventaja se tiene un elevado consumo respecto de las radios compatibles con el estándar IEEE 802.15.4 y un costo superior, ya que no sólo se tiene la inversión inicial en los modems, sino que también se debe pagar una cuota mensual a la compañía celular que provea el servicio de datos.

2.4.2. Modem SIM900

El modem SIM900 [49] transmite a una velocidad de hasta 80 *kbps* y puede ser programado mediante comandos AT[4] via UART. Durante la transmisión tiene un consumo pico de 2 A. Su precio en el mercado es de 15 USD [50].

2.4.3. Radios XBee

Las radios de la familia XBee [51] (figura 2.22) cumplen el estándar IEEE 802.15.4 y están disponibles en distintos modelos, los cuales varían principalmente por su antena y la potencia de

[4]El conjunto de comandos AT es un lenguaje desarrollado por la compañía Hayes Communications que prácticamente se convirtió en estándar abierto de comandos para configurar y parametrizar módems

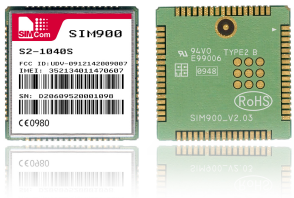


Figura 2.21: Modem GPRS SIM900

Figura 2.22: Radio XBee Pro

salida.

Las radios XBee transmiten en la frecuencia de 2,4 GHz y dependiendo del modelo pueden llegar a una potencia de salida de 60 mW (18 dBm), lo cual equivale a un alcance máximo teórico de 1600 m. Se alimenta de una tensión de 3 V y tiene un consumo promedio en transmisión (a 60 mW) de 215 mA. Se programa mediante comandos seriales por UART y permite cargar configuraciones en *runtime*. Su precio en el mercado es de 60 USD [52].

2.4.4. Radio CC2420 y radio del Soc STM32W108

Estas radios se encuentran integradas en las plataformas TelosB compatibles y STM32W108 respectivamente tal como se describió en las secciones 2.2.2 y 2.2.3.

2.4.5. Análisis de las radios

En los modems SIM900 la carga consumida no sólo es mucho mayor a la que consumen las radios basadas en el estándar IEEE 802.15.4, sino que además el tener picos tan grandes puede dañar las baterías, especialmente si se usan pilas AA. Por otra parte las XBee son una muy buena alternativa en el caso de utilizar un nodo que no tenga radio integrada, pero con la dificultad de que se deben implementar los drivers para su manejo.

La radio CC2420 incluida en las plataformas TelosB compatibles y la radio del SoC STM32W108 tienen la ventaja de que su manejo ya está integrado en los RTOS a los que están portadas las plataformas respectivas. Tiene un consumo considerablemente menor al de las otras radios evaluadas y no encarecen el costo del sistema ya que es parte integral de la plataforma.

Otra ventaja a destacar sobre estas radios es que al estar integradas en las plataformas, el conexionado ya está resuelto y no ocupa un espacio adicional.

2.5. Selección del sistema

Según lo mencionado en las secciones 2.2.8, 2.3.7 y 2.4.5 los sistemas candidato son:

1. Arduino Due + Radio + ArduCAM + Cámara Micron MT9D111 u OV7670 AL422.
2. STM32F4 Discovery + Radio + Cámara (cualquiera).
3. TelosB + Cámara Linksprite LS-Y201.
4. STM32W + Cámara Linksprite LS-Y201.

Los candidatos número 1 y 2 presentan un problema en cuanto al consumo puesto que cada uno de los componentes tiene un consumo considerable respecto a las otras opciones. Además, tal como se expone en el capítulo 3, para esta aplicación resulta por demás útil contar con un RTOS que tenga implementado un sistema de archivos, *drivers* de interfaces de comunicación y un *stack* de red que contenga protocolos orientado a conexión y confiable.

Este último argumento reduce las opciones, hasta que se porten las otras plataformas a los RTOS considerados, a los candidatos 3 y 4. Entre ellos se optó por el sistema 3 ya que tiene un nodo con cual se ha trabajado anteriormente, hay stock disponible en el IIE-FING-UdelaR y principalmente por tener una mayor capacidad de almacenamiento.

La iluminación de sistema se hace a través de dos LEDs amarillos como se explica en la sección 2.5.1 y se alimenta con dos pilas AA como se explica en la sección 2.5.2.

2.5.1. Selección de iluminación

Como las trampas están a la intemperie, las condiciones lumínicas pueden variar considerablemente según la estación y el momento del día en que se tomen. Puede suceder tanto que la imagen no tenga la luz necesaria como para visualizar correctamente el contenido, o por el contrario, que salga saturada por exceso de luz.

Los entomólogos sugirieron tomar las imágenes el amanecer y al atardecer, momentos de mayor actividad de los insectos plaga. Esto hace que se tengan que tomar las imágenes en momentos del día con poca luz y como consecuencia haya que agregar un sistema de iluminación artificial. Si bien esto aumenta el consumo, como se detalla en el capitulo 6, el aumento no influye sustancialmente sobre la autonomía energética del sistema y nos permite tener todas las imágenes en condiciones similares de iluminación. Esto facilita el procesamiento posterior, ya sea el conteo manual por un operario observando la imagen, o un conteo automatizado en el nodo.

La cámara Linksprite LS-Y201 incluye seis LEDs infrarrojos distribuidos en dos columnas como se muestra en la figura 2.15, por lo que en primera instancia se trató de utilizarlos.

Para esto se tomó una imagen en un ambiente oscurecido y se utilizó la iluminación propia de la cámara. El resultado se muestra en la figura 2.23.

Se puede observar que la sustancia adhesiva que se encuentra en el piso de la trampa utilizada para atrapar a las polillas, refleja la luz si se ilumina directamente sobre el piso. Se ven claramente dos focos de reflejo, uno por cada fila de LEDs.

Para tratar de solucionar este problema se probaron distintos métodos para intentar difundir el haz de luz y que no se vieran estos focos de reflejo. Se probó iluminar a través de distintos materiales a modo de filtros, pero ninguno resultó en una mejora sustancial respecto de la situación anterior.

Figura 2.23: Foto tomada en un ambiente oscurecido usando como iluminación los LEDs infrarrojos de la cámara.

La solución planteada consistió en anular los LEDs infrarrojos, sustituyendo la resistencia variable por luz de la cámara por una de valor 180 Ω que asegura que los LEDs permanecen apagados. A su vez se probó iluminar desde el piso de la trampa, con distintos tipos de LEDs y en distintas configuraciones, apuntando hacia el techo de la trampa para que el mismo reflejara la luz hacia el piso, dando una iluminación más uniforme. Se ensayó con cuatro LEDs (amarillos y blancos) dispuestos uno en cada esquina y luego con dos LEDs dispuestos en esquinas cruzadas. Los ensayos realizados se pueden ver en la sección 6.4, donde se concluyó utilizar dos LEDs amarillos.

2.5.2. Selección de alimentación

Para alimentar el sistema, se buscó una solución que tuviera suficiente carga como para cumplir con las restricciones de autonomía energética del sistema, y que a su vez fueran de fácil acceso en el mercado local, para que los operarios que mantienen las trampas puedan comprar los reemplazos y cambiarlas sin necesidad de soporte técnico.

Se decidió entonces utilizar dos baterías AA de $1,5\ V$. La carga que pueden almacenar varía de acuerdo a la composición química de las mismas. Se sugiere utilizar las de ion de litio ya que su carga varía en el rango de 2700 mAh a 3400 mAh. De todas formas, se puede alimentar el sistema con cualquier par de baterías AA, siempre y cuando tengan un voltaje de $1,5\ V$.

La carga de la batería en mAh es un dato que proporciona el fabricante y para tener como referencia, en la figura 2.24 se muestra el perfil de descarga de una batería de ion de litio de 2800 mAh tomada de la hoja de datos de una batería marca Energizer [53].

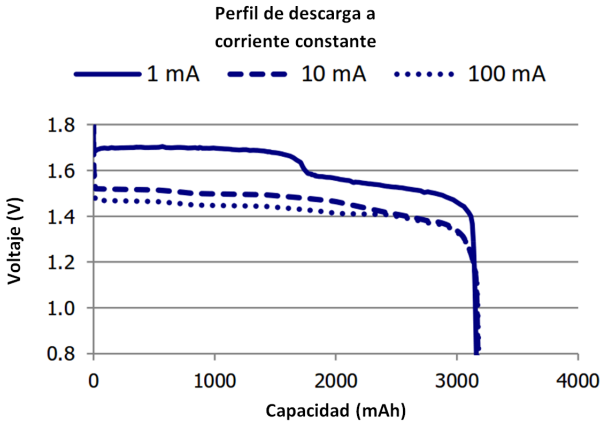


Figura 2.24: Perfil de descarga a corriente constante de una batería AA

Como se puede ver en la figura 2.24, a corrientes de descarga menores o iguales a 100 mA como es el caso, se puede obtener una carga de 2800 mAh a un voltaje mayor o igual a $1,5\ V$. Otro dato que se puede observar en la hoja de datos de la batería es que la capacidad de la misma no varía con la temperatura (en un rango de $-40°$C a $60°$C).

2.6. Circuito interfaz

Si bien la cámara puede funcionar conectada directamente a la alimentación, los requerimientos de bajo consumo del proyecto no son compatibles con su consumo en funcionamiento. Como se mostró en 2.3.5, la cámara Linksprite LS-Y201 presenta la particularidad de tener un consumo de alrededor de 80 mA durante todo su funcionamiento, independientemente de si se está adquiriendo una imagen o no. Si dicho consumo no se elimina cuando el sistema está en modo *sleep*, las baterías utilizadas se agotarían en cuestión de algunas decenas de horas.

Para solventar este problema se propone la utilización del *switch* FPF2004, un dispositivo de manejo de carga que básicamente consiste en un transistor MOSFET con su *gate* conectado a un circuito de control que recibe una señal externa para conmutar entre sus estados. El circuito de control también incluye una realimentación de corriente y temperatura para evitar sobrecargas que puedan causar daños.

Este integrado resulta especialmente conveniente para nuestra aplicación ya que tiene las siguientes características:

- Resistencia de encendido en el orden de 1 Ω, lo que produce una caída de tensión pequeña.
- Corrientes de fuga típicas de 10 nA.
- Consumo del *switch* en estado de desconexión de 1 μA, despreciable frente al consumo del sistema en modo *sleep* (del orden de decenas de μA).
- Corriente de salida máxima de 100 mA, lo cual no genera una restricción en nuestro sistema ya que el consumo máximo es del orden de 80 mA como se puede ver en el capítulo 6.

Alimentando la cámara a través de este dispositivo se puede disminuir considerablemente el consumo del sistema en modo *sleep*. La figura 2.25 muestra el esquemático de la interfaz completa, que permite manejar la alimentación de la cámara y el sistema de iluminación.

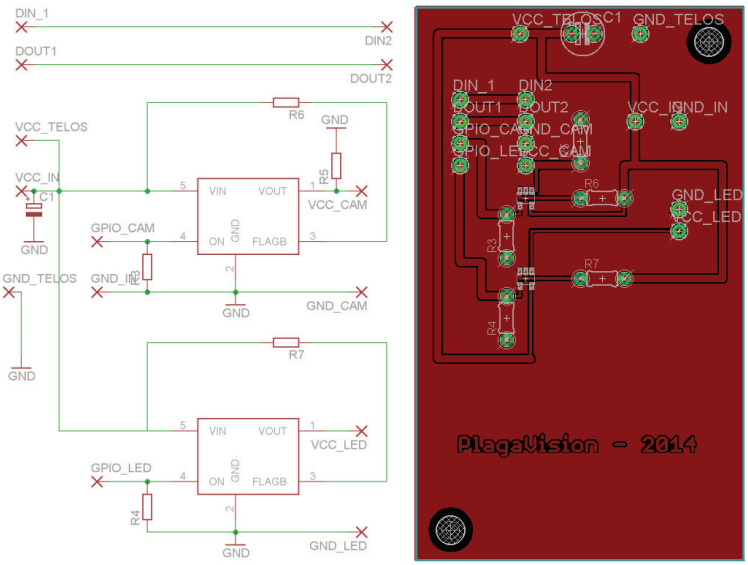


Figura 2.25: Esquemático de la placa que contiene la electrónica a utilizar.

En la figura 2.25 pueden apreciarse dos circuitos de administración de energía, uno para la cámara y otro para la iluminación, controlados por pines diferentes del MCU.

La placa se diseñó de forma tal que coincide perfectamente con el diseño del modelo TelosB compatible CM3000, permitiendo montar uno encima del otro. Se optó por hacer todas las conexiones cableadas desde la placa para hacer más cómodo y ordenado el armado del sistema, por lo que algunos pines se conectan directamente entre sí (e.g los pines TX y RX de la UART).

El esquemático mostrado en la figura 2.25 se observan resistencias de *pull-down* en las entradas de control de ambos switches. Éstas se colocan para llevar dicha señal a cero si los puertos GPIO correspondientes del MCU se llevan a estado de alta impedancia.

Pese a que las resistencias de *pull-down* se utilizan normalmente en señales de control, en nuestro caso colocamos una en la pista de alimentación de la cámara. Esto es imprescindible dado que los capacitores de la misma mantienen el nivel de la alimentación por encima del voltaje de tierra, lo que hace que no se apague en forma correcta e impide volver a inicializarla correctamente.

Otro inconveniente al momento de apagar la cámara surge del hecho de que si no se apaga la UART en forma manual colocando los puertos RX y TX como salidas GPIO a nivel 0, la cámara se mantiene cargada en un voltaje intermedio por dichos puertos. Esto se debe a los diodos de protección de la interfaz de comunicación digital.

2.7. Modificaciones sobre la trampa y gabinete

La cámara presentada en 2.3.5 se fabrica con lentes con ángulos de visión de 60° o de 120°. Debido a problemas de stock solamente nos fue posible adquirir el modelo que abarca 60°, por lo

que fue necesario realizar adaptaciones en el diseño de la trampa. La figura 2.26 muestra el diseño original con las superficies abarcadas por cada uno de los dos modelos.

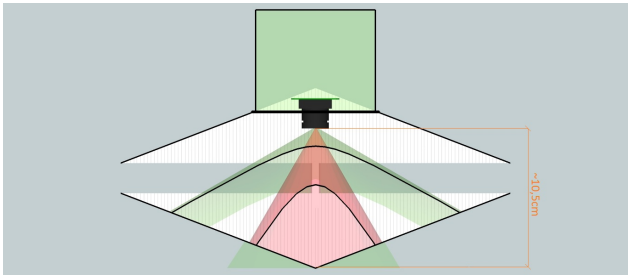


Figura 2.26: Imagen del diseño original de la trampa, con el cono de visión de 120° en rojo y de 60° en verde

Como se puede apreciar en 2.26, la distancia entre el techo de la trampa y el piso es insuficiente para tomar una imagen completa de la superficie adhesiva por lo que se modificaron las proporciones del techo, según la imagen 2.27.

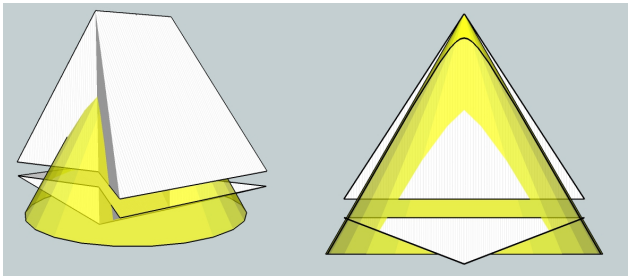


Figura 2.27: Diseño de la nueva trampa, con el cono de visión de 120°

En el nuevo diseño el piso de la trampa permanece incambiado, mientras que la parte superior se diseñó de forma que se mentiene el corte horizontal del modelo anterior pero se incrementa suficientemente la altura. La figura 2.27 muestra que la cámara con el lente de 60° puede cubrir toda la superficie. La parte superior de la trampa rediseñada incrementa en 15 *cm* su altura.

A su vez también se diseñó un gabinete para alojar la electrónica del sistema con el objetivo de aislar lo mejor posible su interior de la humedad, el polvo y las inclemencias climáticas. La figura 2.28 muestra distintos planos del mismo donde puede apreciarse que la tapa presenta un borde para protegerlo de la lluvia. Las baterías se colocan en un compartimiento diferente al resto de los componentes, de forma de que al cambiarlas no se comprometa la hermeticidad del sistema. La base del contenedor es transparente para permitir que la cámara tome imágenes sin tener contacto con el exterior.

El contenedor consta de tres orificios, uno para la salida de la antena y otros dos para el cableado de los LEDs. La antena se monta directamente sobre una de las paredes, quedando sujeta a un cable SMA que la comunica con el TelosB, mientras que los LEDs se colocan en dos espacios especialmente diseñados para sostenerlos, tal como se aprecia en la sección central de la figura 2.29.

La figura 2.29 muestra el prototipo completo. Éste se construyó en acrílico cortado a medida en un *plotter* láser, ya que éste material permite la construcción de modelos en forma rápida y económica. Los diseños tridimensionales se realizaron utilizando el software Google SketchUp, mientras que los planos para el *plotter* se dibujaron en Corel Draw X6. El costo total de la construcción del gabinete fue de 23 USD + IVA. La figura 2.30 muestra fotografías del gabinete.

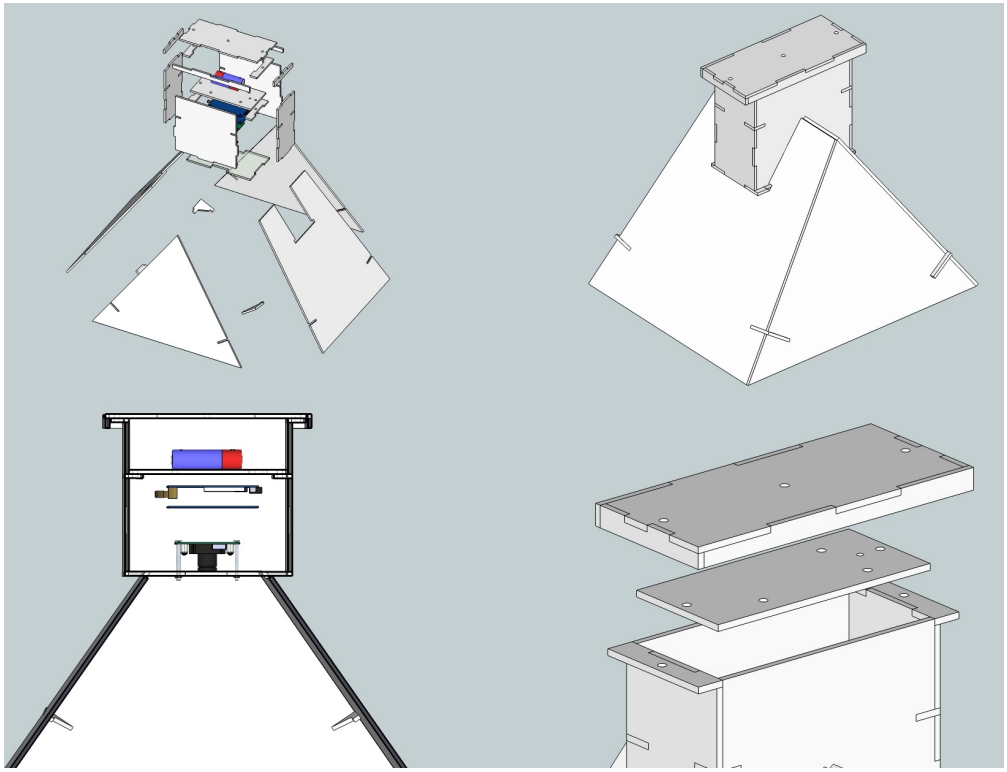


Figura 2.28: Diseño de la nueva trampa y detalles del gabinete

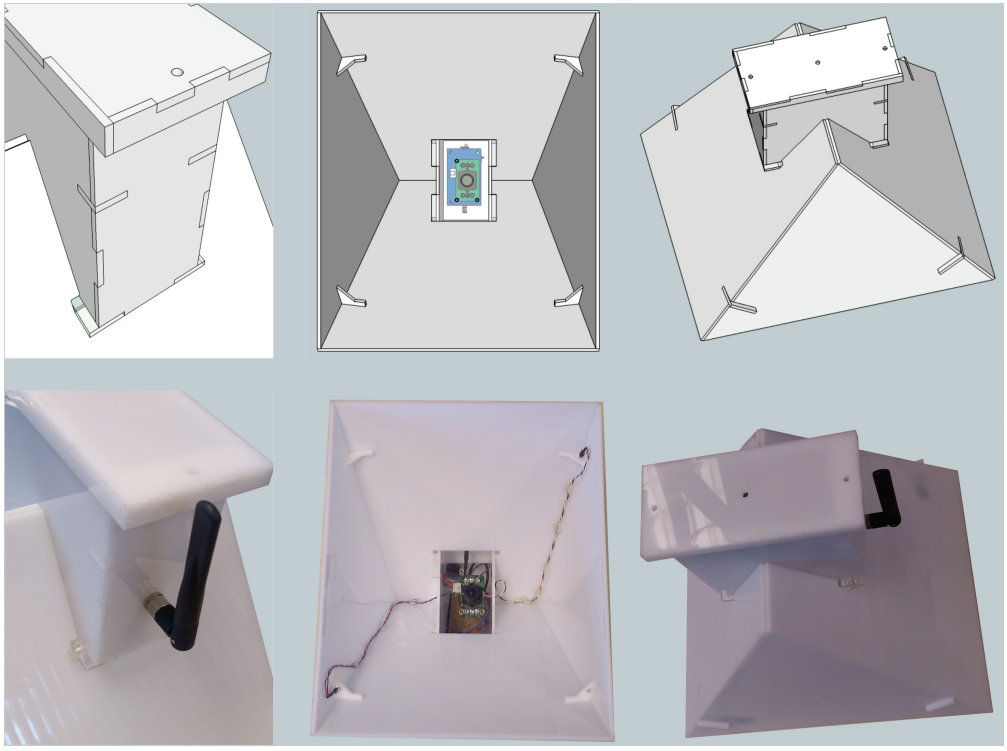


Figura 2.29: Diseño del prototipo construido comparado con el diseño

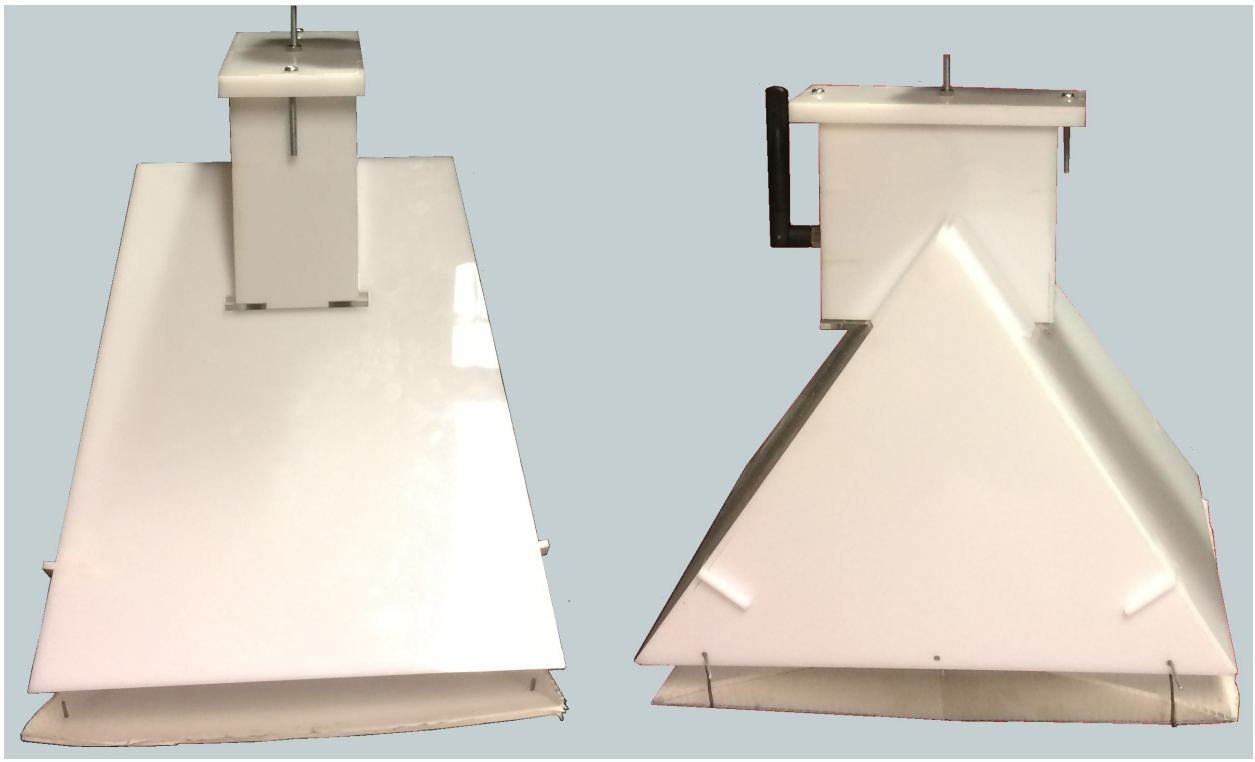


Figura 2.30: Imagen del gabinete

2.8. Costo

El costo del hardware del sistema total es de 180 USD, y se detalla en la tabla 2.2. Éste precio corresponde al desarrollo de un prototipo por lo que es de esperar que para una eventual puesta en producción se pueda disminuir significativamente.

Componente	Detalle	Precio (USD)
Nodo	CM3000	90
Cámara	Linksprite LS-Y201	47
Switches	FPF2004	6
LEDs	Amarillo de 5 mm	2
Gabiente	Fabricado en acrílico	27
Materiales	PCB, resistencias, capacitores, extensor SMA. etc	10

Tabla 2.2: Tabla de costos del hardware.

Capítulo 3

Diseño e implementación del software

Resumen

Este capítulo consta de cuatro grandes secciones, la primera da un marco teórico sobre el uso de los RTOS así como el análisis, la evaluación y la comparación de Contiki OS con TinyOS justificándose la elección de Contiki OS. La segunda sección describe el proceso de diseño del software para el manejo de los nodos que integran la WSN y la tercera describe la implementación del mismo. Por último, la cuarta sección describe las pruebas y los métodos de depuración del código.

Contenido

3.1. RTOS

En los primeros meses del proyecto se analizó qué RTOS utilizar. Las opciones manejadas fueron Contiki OS y TinyOS por ser los dos sistemas operativos con los que se tiene experiencia dentro del Grupo de Microelectrónica del IIE-FING-UdelaR y son conocidos por utilizarse en sistemas con bajos recursos energéticos. En esta sección se realizará una breve introducción a los RTOS y se comparará entre TinyOS y Contiki OS dando los argumentos para la elección del último.

Los RTOS son sistemas operativos creados para servir en aplicaciones de tiempo real. Como cualquier otro sistema operativo, proveen una abstracción del nivel de hardware (HAL), llenando el espacio entre las aplicaciones y el hardware existente a partir de una serie de APIs [54].

La parte central de un RTOS es su kernel, el cual maneja el acceso al hardware (MCU, memorias y periféricos) por parte de las aplicaciones en forma segura. El kernel contiene al *scheduler*, organizador que establece la ejecución de las distintas tareas o procesos estableciendo orden y prioridades. Además el kernel maneja las interrupciones, el control de concurrencia, de periféricos y en algunos casos soporte de red.

Por otra parte los RTOS pueden clasificarse por ser monolíticos o modulares. Los primeros se caracterizan por tener un kernel de gran tamaño y complejo, donde en un mismo espacio de memoria y claramente separado del espacio de usuario, se concentran todas las funcionalidades del sistema. En cambio los modulares poseen un kernel mucho menor, conteniendo únicamente servicios básicos. Otros servicios como gestión de memoria, sistema de archivos, etc. se delegan a servicios en el espacio de usuario. A su vez los programas son compilados individualmente y cargados por el sistema.

Otro aspecto de clasificación entre los RTOS es por si el sistema es estático o dinámico. Los sistemas estáticos deben asignar todos los recursos en al momento de diseño y compilación, mientras que los dinámicos pueden asignar o quitar recursos en *run-time*.

Dado que los RTOS se utilizan en sistemas con hardware limitado, deben ocupar poco espacio en memoria RAM y ROM[1], lo que se conoce como bajo *footprint*. Además es deseable que el cambio de contexto entre tareas, así como la latencia de interrupción (tiempo entre que se genera la interrupción y ésta es atendida), sea lo más rápido posible.

Por su modelo de ejecución, pueden clasificarse entre *Event-driven* o *Multithreading*. Los RTOS *Event-driven* cambian de tarea cuando un evento de mayor prioridad necesita ser atendido. Los kernels capaces de quitar el control del MCU a la tarea actual para ejecutar una de mayor prioridad se denominan *preemptives* y tienen como ventaja un uso eficiente de memoria así como un bajo *overhead* (tiempo extra computacional) en el cambio de contexto.

En el caso del *Multithreading* se utilizan procesos en lugar de tareas, que son implementados como hilos de ejecución que se ejecutan de manera concurrente y el kernel intercambia el uso del procesador al darse un evento. Tiene como desventaja un gran consumo de memoria ya que cada hilo requiere su propio *stack* de espacio en memoria, lo que repercute a su vez en que el *dispatcher* (despachante de procesos) tenga un *overhead* mayor al cambiar de proceso. Una de las ventajas más significativas es que simplifica la implementación de máquinas de estado.

Las tareas o procesos pueden tener tres estados posibles:

[1]Por ejemplo TinyOS utiliza 3450 bytes de ROM y 226 bytes de RAM

- *Running*: El proceso o tarea se está ejecutando.
- *Blocked*: Proceso o tarea está bloqueada, no puede ser ejecutada.
- *Ready*: Proceso o tarea está lista, esperando para ser ejecutada.

3.1.1. TinyOS

TinyOS es un RTOS de código abierto desarrollado en la Universidad de Berkeley basado en el modelo *Event-driven*. El sistema provee un kernel que consume pocos recursos de memoria ya que utiliza un único *stack* para todas las tareas. Dado que está pensado para sistemas de bajos recursos energéticos, cuando el MCU está sin uso se lo mantiene en un nivel de bajo consumo donde se apagan distintos periféricos y osciladores. TinyOS está escrito en nesC y se trata de una arquitectura basada en componentes, donde cada uno consiste en comandos, manejo de eventos, tareas y módulos [55].

Por otra parte, TinyOS es un sistema estático y monolítico que resulta menos flexible a la hora de programar o reprogramar a través de la red. En lo que respecta al sistema de archivos, Matchbox [56] y Blackbook [57] proveen un primer nivel de abstracción para lectura y escritura de archivos.

El stack de red en TinyOS incluye una implementación de IPv6 llamada BLIP [58], una versión propia de RPL llamada TinyRPL [59] y Active Message [60]. Con estos se pueden implementar aplicaciones de recolección y diseminación de datos para WSN.

3.1.2. Contiki OS

Contiki OS es un RTOS de código abierto creado en SICS[2] con un kernel que implementa un modelo *Event-driven*, con funcionalidades de *Multithreading*. Éste es básicamente dirigido por eventos y utiliza un único *stack* para los procesos pero a su vez es capaz de utilizar *Multithreading* del tipo *preemptive*. Está escrito completamente en C y portado a varias plataformas, particularmente a las plataformas TelosB compatibles que fueron utilizadas en este proyecto. Dentro de las ventajas destacables se encuentra el hecho que posee un pequeño *footprint* (aproximadamente $3,8\ kB$), una comunidad creciente de desarrolladores y una actualización periódica.

Dentro de las funcionalidades que ofrece Contiki OS, útiles para el desarrollo del proyecto, se encuentran: manejo de sistemas de bajos recursos energéticos (e.g. alimentados con baterías AA), *stack* completo de red IPv4 e IPv6 con los protocolos de capa de transporte UDP, TCP y FTP implementados, utilización de ContikiMAC [61] y otros como protocolo de RDC, el sistema de archivos CFS-Coffee [62] y el *stack* de primitivas de red Rime [63] con el que se pueden armar protocolos de red adaptados a las necesidades de la aplicación.

[2]Swedish Institute of Computer Science, SICS, es una organización de investigación sin fines de lucro. Sus líneas de investigación se centran en computación, sistemas embebidos e interacción hombre-máquina.

3.1.3. Comparación y elección entre TinyOS y Contiki OS

La tabla 3.1 compara las distintas características de los RTOS considerados.

Características	TinyOS	Contiki OS
Lenguaje	nesC	C
Event-driven	√	√
Mulithreading o "time-sharing"	Parcial	√ Librería multithread
Tam. kernel (bytes)	400	810
Soporte de red	Active Message y BLIP	uIP y Rime
File system	Blackbook, Matchbox	CFS-Coffee
Reprogramación "on the fly"	No	√
Entorno de desarrollo	Eclipse GCC	Eclipse GCC

Tabla 3.1: Tabla comparativa de los RTOS considerados

Algunas ventajas de Contiki OS frente a TinyOS son que está escrito completamente en C, lo que implica no tener que aprender un nuevo lenguaje, la versatilidad del *stack* de red y que permite la reprogramación de manera inalámbrica (que por más que no se utilizó en este proyecto podría ser de gran utilidad con el sistema en funcionamiento).

La investigación sobre estos RTOS nos permite además afirmar que Contiki OS está portado en una mayor cantidad de plataformas que TinyOS. A su vez, la gran actividad que presenta la colectividad de desarrolladores de Contiki OS repercute en que constantemente se estén agregando nuevas y más recientes plataformas al repositorio.

Por otro lado TinyOS presenta la ventaja de ser un RTOS más antiguo, por lo que es de esperar que sea más estable y exista mayor documentación asociada. A su vez tiene un *footprint* menor al de Contiki OS.

Si bien ambos RTOS presentan características similares, por lo antes expuesto y la experiencia que se tiene dentro del Grupo de Microelectrónica del IIE-FING-UdelaR se eligió utilizar Contiki OS. La elección de éste fue parte fundamental en el inicio del proyecto. Esta elección repercutió fuertemente en la elección de la plataforma hardware ya que como se mencionó en el capítulo 2, se buscaron nodos portados a este sistema operativo.

3.1.4. Generalidades de Contiki OS

En las secciones siguientes se explican brevemente algunos de los conceptos básicos de Contiki OS como ser procesos, *protothreads* [14], eventos, CFS y Rime. Esto servirá para entender conceptos manejados en lo que respecta al diseño y la implementación de las aplicaciones software de este proyecto.

Procesos

Este RTOS busca simular la multitarea ejecutando en paralelo hilos de procesamiento. Estos están implementados a partir de un bloque de control y un hilo a partir de un *protothread*. Los *protothreads* suponen una nueva abstracción de programación escrita en C que no utiliza *stack*, provee control de flujo secuencial y funciones de espera condicional bloqueante `wait()` propias del *multithreading*.

Los *protothreads* corren de manera cooperativa, es decir que cada uno corre de manera secuencial sin interrumpir al otro (no son *preemptive*) por lo que el cambio de contexto entre tareas sólo puede ocurrir cuando una tarea se bloquea (salvo que la interrupción sea por hardware o se trate de una interrupción del Rtimer). Por otra parte, al no tener *stack* se deben utilizar variables globales para preservar variables en los cambios de contexto.

El usuario crea los procesos y estos se almacenan en una lista encadenada, pudiendo ser ejecutados desde el inicio o por una función u otro proceso. Al igual que en la mayoría de los RTOS los procesos pueden tener los tres estados antes mencionados de *running, ready* y *blocked*. Estos son alternados a partir de eventos por el *dispatcher* del kernel.

En las aplicaciones del proyecto los procesos atienden las principales funciones del sistema, manejo de la cámara, manejo de la red y control global del sistema. El último se encarga de entregar los eventos a los demás procesos para así controlar la secuencia de ejecución de los hilos.

Eventos

Los eventos son aquellos que desencadenan la ejecución de un proceso, ya que cada evento tiene asociado un proceso a despertar. Puesto que pueden ocurrir varios eventos en períodos cortos de tiempo, se almacenan en un *buffer* circular y luego son procesados en orden cronológico.

Los eventos pueden ser originados por: hardware a partir de interrupciones, software a través de alguna aplicación del sistema (Etimers, *stacks* de red, etc.) o por el usuario. La función de *Mailbox* de un RTOS se implementa a través de un *post* síncrono o asíncrono. Estos agregan a un *buffer* circular un evento asociado a un único proceso o a todos, con un puntero a los datos a entregar. Cabe destacar que el *post* síncrono interrumpirá el proceso en ejecución para atender el evento en caso de que los requerimientos de tiempo sean críticos.

Otra manera de ejecutar un proceso es realizando un *poll*, el cual modificará la bandera `needspoll`, que le indica al *scheduler* si un proceso debe ser atendido. Antes de procesar la cola de eventos el *scheduler* verifica esta bandera.

Timers

Contiki OS provee una serie de librerías *timer*, los cuales pueden ser usadas por las aplicaciones o por el sistema operativo mismo. Con ellas se pueden generar *timers* que encolen un evento (Etimer), relojes de tiempo real para el *scheduler* o la ejecución de tareas críticas (Rtimer), reloj como temporizador (Stimer) y reloj lanzador de *callbacks* (Ctimer). Todos ellos se basan en el módulo *Clock* donde está definido `CLOCK_SECOND` que corresponde a la cantidad de *ticks* por segundo para

cada plataforma. [3]

Los Etimers tienen un mecanismo para generar eventos a partir de un *timer*. Este realizará un *post* al proceso que esté esperando que expire dicho *timer*.

Contiki File System

Contiki OS ofrece una serie de sistemas de archivos para distintos tipos de dispositivos de almacenamiento como ser memorias EEPROM, RAM y flash en sistemas con pocos recursos de procesamiento a partir del Contiki File System (CFS). Dentro de este sistema se encuentra la funcionalidad Coffee, especialmente diseñada para nodos con memoria flash externa y que se utilizará para guardar temporalmente las imágenes.

El Coffee File System es una mejora del CFS que agrega funciones para: reservar espacio para un archivo, formatear el área de memoria asignado a Coffee y configurar un registro de entradas. Cabe destacar que para poder utilizar correctamente Coffee es necesario formatear el espacio de memoria asignado antes de realizar cualquier escritura en el sistema de archivos. Además el sistema reserva los primeros 64 kB de la memoria flash externa para guardar la configuración del nodo y los segundos 64 kB para guardar el código en una eventual programación por la red en *runtime*.

Los parámetros de CFS-Coffee como el tamaño de una página de memoria, de un sector de memoria y el tamaño total, así como los *drivers* para los dispositivos E/S que manejan la memoria están definidos para cada plataforma en el archivo `cfs-coffee-arch.h`. En el caso de la memoria flash ST M25P80 se tienen 16 segmentos, cada uno de 64 kB. Cabe mencionar que existe un tamaño máximo predefinido para los archivos definido en `platform-conf.h`, el cual corresponde a un segmento de la memoria.

Stack de red

El *stack* de red será analizado en profundidad en el capítulo 5 pero a modo de resumen éste ofrece dos grupos de protocolos de red: Rime y uIP. El primero está compuesto por primitivas de red de diversa índole como ser *unicast, multicast* y *broadcast* en cuanto a métodos de comunicación, *single-hop* o *multi-hop*, así como *route discovery* y *neighbour discovery* en cuanto a ruteo dinámico entre otros. A partir de ellos se puede implementar un *stack* de protocolos acorde a las necesidades de la aplicación.

A su vez, algunas de estas primitivas son utilizadas a nivel de capa de red por uIP para la implementación de *Internet of Things*. Con uIP se pueden utilizar protocolos de capa de transporte como TCP y UDP rediseñados para sistemas de bajos recursos hardware.

[3] En el caso del MSP430F1611 128 *ticks* son un segundo

3.2. Diseño de la aplicación

Para diseñar la aplicación software se dividió la misma en cinco módulos (*Cámara, Radio, JPEG, Sleep* y *Plagavisión*), cada uno según a la funcionalidad deseada.

El módulo *Cámara* gestiona el manejo de la cámara Linksprite LS-Y201 y almacena la imagen capturada en la memoria flash externa. Ésta es utilizada por el módulo *JPEG*, que realiza una descompresión parcial de la imagen JPEG mediante la cual se obtienen dos productos. Como se explica en la sección 3.2.4, el primero es una imagen de tamaño 4800 bytes con información reducida y el segundo es el recuento de polillas.

El módulo *Radio* es el responsable de enviar tanto las imágenes JPEG, como la imagen reducida y el conteo de polillas al nodo *sink* según la configuración definida por el usuario. Por otra parte, el módulo *sleep* implementa las funciones que son capaces de llevar al sistema a niveles de bajo consumo gestionando el hardware involucrado. En último lugar, el módulo *plagavision* se encarga del manejo general del sistema. Enciende el mismo dos veces al día y ejecuta las distintas funciones siguiendo el diagrama de aplicación de software que se observa en la figura 3.1.

Por otra parte, el nodo *sink* conectado al PC corre el módulo *receptor* por el cual recibe los datos y los imprime vía UART al PC donde son almacenados.

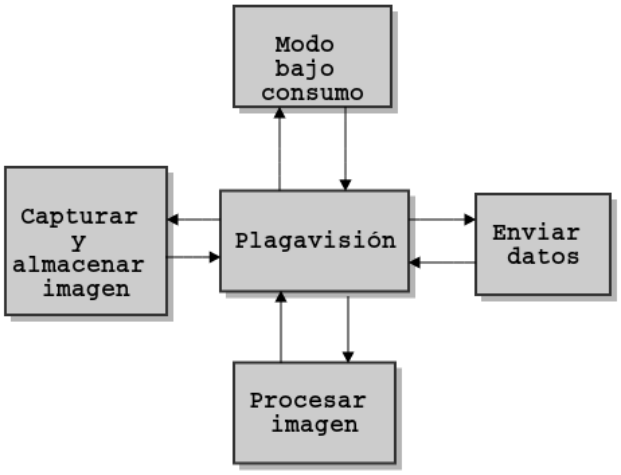


Figura 3.1: Diagrama de aplicación de software

3.2.1. Comunicación serial con la cámara

Como se mencionó en la sección 2.2.2, el MSP430F1611 posee dos USART para comunicación serial. En los nodos TelosB compatibles el bus de la USART0 se encuentra compartido entre la radio CC2420 y la memoria flash externa configurado como SPI, mientras que en la USART1 se encuentra conectado el FTDI (en el caso del CM3000 se conecta a través del extensor USB1000) por el cual se puede establecer una comunicación serial con un PC por el USB. Debido a que los creadores de Contiki OS consideraron que el uso de la USART0 (como bus SPI) era crítico ya que la radio se utiliza constantemente, sólo implementaron el *driver* para el manejo de la UART1 en estos nodos.

En el modelo TelosB original y en el TelosB compatible CM5000 [64] no existen pines accesibles

a la USART1 (salvo a través del FTDI por ende el USB) por lo que en primera instancia se consideró que resultaría imposible realizar la depuración por el puerto USB ya que la cámara y la PC podrían solicitar simultáneamente las líneas de la UART generando conflicto entre ellas.

Durante la primera etapa del proyecto se consideró realizar una implementación del *driver* de la UART0 estableciendo un arbitraje entre el servicio SPI y UART. Consultando los foros de desarrolladores de Contiki OS se encontró que los autores del RTOS desalentaban dicha implementación por lo que rápidamente se diluyó esta idea. El argumento expuesto era que un árbitro podría generar conflicto con algunos servicios del sistema como ser ContikiMAC que tiene requerimientos de tiempo críticos por lo que el uso de la USART0 debe ser exclusivo y con prioridad a la radio.

Al no recomendarse la utilización de la UART0, se decidió utilizar la UART1 con aquellos modelos TelosB compatibles que no tienen el FTDI integrado como es el modelo CM3000. El problema que surgió entonces fue encontrar los pines correspondientes a la UART1 ya que los esquemáticos de estos modelos no estaban disponibles[4]. En todos los modelos TelosB compatibles el MCU se encuentra en la parte inferior de la placa justo encima del portapilas por lo que no era posible medir continuidad directamente desde el MSP430F1611. No obstante el CM3000 tiene un conector donde se conecta una tarjeta de expansión (la EX1000) que incluye dos puertos RS232 que claramente están conectados a las interfaces UART. Midiendo continuidad entre los pines del controlador MAX232 de la placa EX1000 y los correspondientes a dicho conector se encontraron los pines buscados. En la figura 3.2 se muestra la conexión de la cámara a los *pads* del conector.

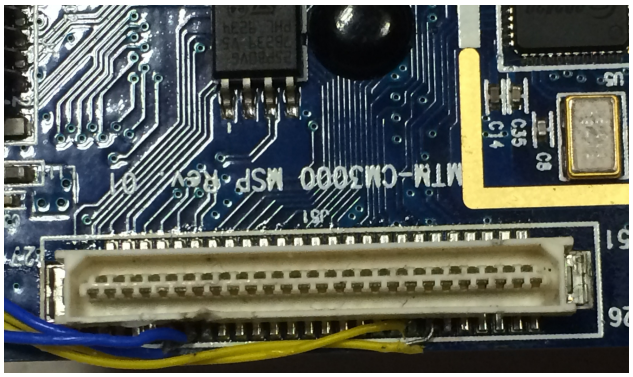


Figura 3.2: Conexión de las líneas RX y TX (cables amarillos) de la UART CM3000

Otras opciones manejadas en torno a este problema fueron implementar una UART mediante *bit-banging* con puertos GPIO o utilizar un conversor UART a I^2C y el *driver* I^2C de *bit-banging* ya implementado en Contiki OS. El problema de estas soluciones se encuentra en la restricción de tiempo que imponen al sistema ya que la aplicación debe estar verificando y cambiando periódicamente el estado de los pines. A su vez se estimó que el tiempo necesario para implementarlas sería mayor a conectar la cámara directamente a la UART1.

Luego los esfuerzos se concentraron en investigar sobre cómo depurar dispositivos seriales ya que no se podría usar el USB luego de conectada la cámara a la UART1. Se halló la posibilidad de realizar una conexión virtual entre la cámara y el nodo utilizando la PC, previo a conectar la cámara directamente al nodo. Para ello se utilizó la interfaz serial/USB de un Arduino UNO y

[4]Al momento de escribir esta documentación se encontraron las hojas de datos correspondientes al CM3000 y al CM5000 donde se pueden ver los pines correspondientes a los conectores

un software libre llamado *Sersniff* [65] disponible para Linux. Detalles del *setup* de depuración se pueden encontrar en la sección 3.4.

En resumen se manejó la configuración Cámara ⇔ Arduino ⇔ PC ⇔ CM5000 en la etapa de desarrollo y la configuración Cámara ⇔ CM3000 para las pruebas con la radio y la versión definitiva del prototipo.

El *driver* incluído en Contiki OS para la UART1 contiene funciones para inicialización, escritura y lectura del dispositivo E/S. Para el caso de la recepción es necesario implementar una función *callback* para manejar los datos recibidos. En nuestro caso se implementó un *buffer* circular que realiza un *poll* al proceso en ejecución para permitir que este recurso pueda ser compartido.

3.2.2. *Driver* de la cámara

La cámara utiliza comunicación serial de niveles TTL entre $+3\ V$ y $0\ V$. Todos los comandos comienzan con los valores hexadecimales `0x56 0x00` y cualquier otra secuencia inicial no es reconocida por la cámara por lo que se puede imprimir en consola sin problemas para depurar. Cada comando correctamente enviado es reconocido por la cámara enviando un ACK los cuales comienzan con los valores `0x76 0x00`.

La cámara se maneja a través de distintos comandos que incluyen:

- seleccionar la resolución de la imagen entre 640×480, 320×240 y 160×120 pixeles,
- seleccionar el factor de compresión de la imagen entre `0x00` y `0xFF`,
- seleccionar el *baud rate* entre $9600, 19200, 38400, 57600$ y 115200 *bps*,
- tomar foto,
- detener la captura de imágenes,
- leer el tamaño de la imagen,
- descargar la imagen,
- reiniciar.

Inicialización de la cámara

Una secuencia de inicialización es enviada por la cámara al encenderse o al reiniciarse y corresponde al siguiente mensaje:

Ctrl infr exist
User-defined sensor
625
Init end

El nodo entonces debe esperar que lleguen los datos correspondientes a `Init end` que corresponden a los valores hexadecimales `0x36 0x32 0x35 0x0D 0x0A 0x49 0x6E 0x69 0x74 0x20 0x65 0x6E 0x64 0x0D 0x0A`. Luego se deben esperar dos segundos para empezar a enviar comandos. El diagrama de flujo de inicialización se puede observar en la figura 3.3.

La etapa actual de la aplicación no implementa ninguna acción de contingencia si no se recibe la secuencia esperada por tratarse de la primera etapa de un prototipo, lo cual queda como trabajo futuro.

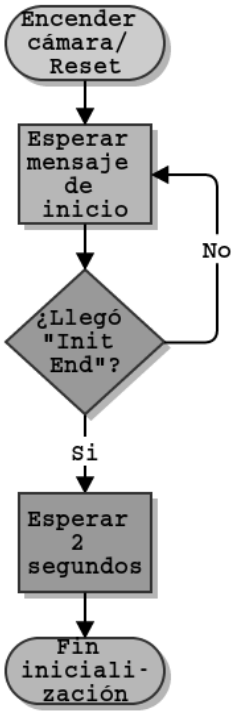


Figura 3.3: Diagrama de inicialización

Envío de comandos

Luego de inicializada la cámara se pueden comenzar a enviar los comandos antes mencionados. En nuestro caso, antes de tomar una imagen se cambia la resolución a 640×480 píxeles, luego de lo cual se debe ejecutar un reset y realizar la secuencia de inicialización. También se cambia el factor de compresión de la imagen, así como el *baud rate* que por defecto es 38400 *bps* a 115200 *bps*.

En cada caso se envía el comando y se espera por el ACK correspondiente. Se comparan el ACK recibido con el esperado, dando un mensaje de error en caso de no ser iguales. Por los mismos motivos que para la inicialización de la cámara, no se tomaron medidas de contingencia en caso de no recibir el ACK esperado.

La secuencia para tomar una imagen se observa en la figura 3.4. Se comienza enviando el comando para tomar una foto. Luego se debe enviar el comando para leer el tamaño del JPEG y de los últimos dos bytes del ACK recibido desde la cámara, se obtiene el tamaño en bytes de la imagen (el valor corresponde a un entero de 16 bits). Dado que la cámara posee una memoria incluida en el SoC, la foto puede ser retirada a demanda por el MCU en bloques cuyo tamaño ha de ser múltiplo entero

de 8. Los bloques son enviados con un ACK inicial y final entre los datos.

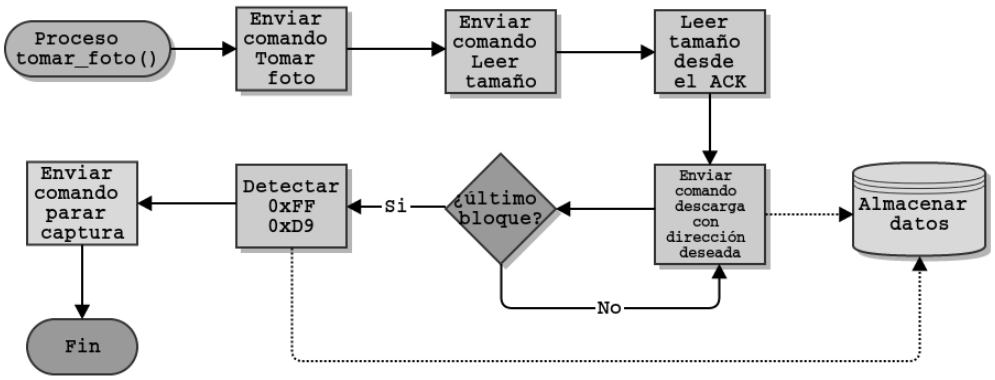


Figura 3.4: Diagrama para la captura de imágenes

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Valor	56	00	32	0C	00	0A	00	00	MH	ML	00	00	KH	KL	XH	XL
Descripción	Incio								Dirección de memoria				Tamaño del bloque		Intervalo de tiempo	

Tabla 3.2: Comando de descarga de la imagen

En el comando de descarga que se observa en la tabla 3.2 se debe especificar la dirección de memoria que se desea acceder en los bytes MH y ML, y el tamaño del bloque solicitado en los bytes KH y KL. Se debe indicar a su vez un intervalo de tiempo entre los ACK iniciales y finales y los datos, dados por los bytes XH XL. De esta manera se itera tantas veces como el cociente de la división entera entre el tamaño del JPEG y el tamaño del bloque más una vez, esto es $\lfloor \frac{Tamano}{Bloque} \rfloor + 1$ veces.

En cada iteración se debe incrementar la dirección de memoria en una vez el valor del tamaño de bloque y almacenar los datos recibidos en memoria flash mediante CFS-Coffee, salvo en la última iteración donde se debe encontrar el marcador de fin de un archivo JPEG correspondiente a `0xFF 0xD9` y almacenar los datos hasta ese marcador [5]. Por último se envía el comando para parar de tomar fotos culminando el proceso.

En caso de que se quiera continuar con la captura sin apagar la cámara (ya sea para pruebas o futuros cambios en la aplicación), no es necesario que se ejecute nuevamente la secuencia de inicialización de la cámara pudiéndose iniciar solamente el proceso encargado de tomar imágenes.

En lo que refiere a la iluminación, los dos LEDs colocados en el techo de la trampa son encendidos por el sistema un segundo antes de que se envíe el comando de tomar foto y se apaga inmediatamente después de que se recibió el ACK de dicho comando. La razón por la cual se enciende los LEDs un segundo antes es porque la cámara tiene incorporado un sistema de balance automático de blancos y experimentalmente se comprobó que necesita aproximadamente ese tiempo para que la imagen no salga saturada.

[5]El último bloque se rellena con datos cualesquiera desde `0xFF 0xD9` hasta el fin del mismo

3.2.3. Almacenamiento en CFS-Coffee

Debido a la restricción en memoria RAM, es necesario almacenar la imagen en la memoria flash externa. Esto es posible utilizando el anteriormente mencionado CFS-Coffee, mediante el cual se logran almacenar las imágenes en archivos diferenciados por nombre.

Las imágenes JPEG tomadas por la cámara, con el factor de compresión por defecto, tienen un tamaño aproximado de 18 kB, por lo que cumplen con la restricción de tamaño máximo de 64 kB mencionada en 3.1.4. Aún con el factor de compresión al mínimo estas ocupan menos de 50 kB. Por otra parte, las imágenes con información reducida obtenidas a partir de la descompresión parcial del JPEG tienen un tamaño fijo de 4800 bytes por lo que deberían entrar al menos 39 archivos [6].

De todas maneras no se pretende que el nodo mantenga una base de datos de las imágenes sino que las transmita en caso de ser necesario, entonces para evitar llenar la memoria, el sistema borra dicha imagen luego de enviarla al *sink*. Lo que sí es necesario es mantener las imágenes recibidas por nodos vecinos en caso de que el nodo actúe como un salto en la red, debiendo guardar las mismas hasta que puedan ser transmitidas al *sink*.

3.2.4. Procesamiento de imágenes

Los detalles de este módulo se encuentran en el capítulo 4, pero a modo de resumen esta etapa ejecuta un algoritmo de procesamiento sobre la imagen obtenida por la cámara, es decir sobre los datos comprimidos en JPEG, con el fin de disminuir el tamaño de datos a enviar por la red sin que exista una reducción significativa en la cantidad de información de los datos.

Una de las restricciones más fuertes presentes en los nodos de WSN es la falta de memoria, tanto dinámica como estática. Una imagen cruda de resolución 640×480 píxeles con profundidad de color de 8 bits por canal ocupa 900 kB en memoria, por lo que en el caso de los TelosB compatibles la memoria externa no podría almacenar ninguna imagen. Es por esta razón que se descartó el procesamiento de imágenes en el espacio matricial.

Estudiando el estándar JPEG se encontró la posibilidad de implementar un decodificador parcial que permita obtener una nueva imagen en formato crudo a partir del valor medio del canal de luminancia (canal Y) de cada bloque de 8×8 pixeles. Esta nueva imagen ocupa 4800 bytes correspondientes a $\frac{640}{8} \times \frac{480}{8}$, lo que reduce en un 70 % el tamaño de los datos.

Esta decodificación también permite, utilizando un algoritmo de detección de patrones FLD, realizar una clasificación y ejecutar un recuento de polillas.

En consecuencia el sistema implementa dos funciones complementarias que logran disminuir el número de transmisiones de la red y con ello el consumo energético del nodo. La primera proporciona datos visuales de la trampa con información reducida (y de menor tamaño) y la segunda suministra un valor cuantificado de la cantidad de insectos. Esto permite distintas configuraciones en donde se puede enviar dependiendo de la cantidad de información deseada: la imagen JPEG, la imagen con información reducida, el conteo de polillas únicamente o las distintas imágenes cuando hay un cambio significativo en el recuento.

[6] Sin los dos primeros segmentos quedan $1024 - 128 = 896$ kB, por lo que entran $896/(18 + 4,8) = 39$ fotos

3.2.5. Transmisión por radio

El propósito de este módulo es manejar la comunicación entre los nodos, transmitiendo las imágenes y realizando la administración de la red. La comunicación entre nodos elegida es del tipo *unicast* y *multihop* con una topología del tipo árbol donde todos los nodos envían sus datos al *sink* (véase capítulo 5). Los protocolos utilizados corresponden a las primitivas existentes en Contiki OS en el *stack* de comunicaciones Rime.

Puesto que se desea transmitir imágenes sin errores por la red, es necesario contar con un protocolo orientado a conexión y confiable, que garantice que los datos lleguen sin errores ni duplicados y en el orden correcto. Esto es posible en protocolos que implementan reconocimiento de paquetes, control de secuencia e intercambio de datos al establecer y cerrar la conexión.

Otro requerimiento que determina el funcionamiento del sistema es el hecho de que se desea tomar, en principio, dos imágenes diarias: en el amanecer y en el atardecer. Esto implica que durante gran parte del día no se transmitirán datos por la red siendo innecesario mantenerla activa en ese lapso, pudiéndose apagar las radios de los nodos.

El alcance de este proyecto no incluye la configuración definitiva de la red, pero se realizaron pruebas implementando la misma mediante ruteo estático, donde cada nodo tiene programada la dirección del próximo salto para los paquetes, el cual los acerca al nodo *sink*.

Con esta configuración, entre los nodos más alejados del *sink* y éste último se encuentran los nodos *router* o con funcionalidad completa, los cuales deben ser capaces de almacenar de manera provisoria las imágenes recibidas por los nodos vecinos y guardarlas en memoria flash hasta que puedan enviársela al *sink*. A su vez estos nodos deben ser capaces de distinguir entre las imágenes tomadas por ellos y las recibidas desde otro nodo, indicándole al *sink* el origen de cada una.

El *sink* se encuentra conectado a una PC por lo que no tiene restricciones de energía. Éste sólo implementa la recepción de los datos por parte de los nodos vecinos y por comunicación serial envía los datos al PC donde se almacenan los mismos.

En resumen, para el diseño de la red se tendrían tres tipos de dispositivos:

- **Nodos con funcionalidad reducida:** Toman imágenes pero no actúan como routers.
- **Nodos con funcionalidad completa:** Toman imágenes y actúan como routers.
- **Nodo *sink*:** No toma imágenes, está conectada a la PC donde se encuentra la base de datos.

El software de los nodos con funcionalidad completa fue implementado y probado, en particular el almacenamiento correcto de las imágenes recibidas en CFS-Coffee y su transmisión al nodo *sink*. El análisis de aquí en más se hará únicamente para los nodos de funcionalidad reducida ya que éstos son los que se encuentran dentro del alcance del proyecto aunque se puede extender fácilmente para los nodos de funcionalidad completa. El código de la implementación se puede consultar en el CD adjunto.

3.2.6. Modo sleep

Para cumplir con las especificaciones de consumo se buscó que el nodo y los periféricos permanecieran en un estado de bajo consumo durante la mayor parte del tiempo posible. Los periféricos incluidos en el nodo tienen modos de bajo consumo configurables. Contiki OS implementa algunas funciones para este cometido, pero aún insuficientes para los requerimientos energéticos de este proyecto.

El sistema operativo fue pensado para trabajar en una red, donde todos los nodos mantienen un RDC que enciende la radio por intervalos de tiempo reducidos para transmitir señales de datos y de control, por lo que no apagan la radio completamente. Este no es el caso en esta aplicación, donde tiene sentido apagar completamente la radio incluyendo el oscilador y el regulador de voltaje, ya que solamente se enviarán datos dos veces al día. En lo que respecta a la memoria flash, esta aplicación ejecuta los comandos que llevan a la misma a un estado de bajo consumo luego de que se haya terminado de utilizar el CFS.

El MSP430F1611 tiene cuatro modos de bajo consumo que son LPM0, LPM2, LPM3 y LPM4, donde LPM0 es el de mayor y LPM4 el de menor consumo. LPM4 no mantiene ningún oscilador encendido por lo que no se puede salir de este modo sin una interrupción externa. En LPM3 se mantiene encendido únicamente el oscilador de cristal de $32kHz$ (que genera los *ticks* del sistema operativo) y cualquier actividad del MCU se ejecuta a partir de la interrupción de éste o algún agente externo. Por lo que la aplicación activa el LPM3 del MCU luego de efectuarse todas las tareas del sistema antes mencionadas y apagar los perifericos.

Por último, el hardware externo es manipulado por los GPIO del MCU a través de switches que encienden la cámara y los LEDs a demanda. En el caso de la cámara, estará encendida mientras se toma y se descarga la foto al nodo mientras que los LEDs solamente se encienden en el instante en que se captura la imagen.

3.2.7. Nodo *sink*

El nodo *sink* tiene la funcionalidad de recibir los datos desde los nodos con funcionalidad completa y reducida e imprimir los mismos por el puerto serial hacia el PC al que está conectado, que actúa como base de datos.

Mediante los campos de dirección Rime contenidos en las tramas intercambias, el nodo *sink* obtiene el identificador de quién le envió los datos. Por medio de etiquetas y datos de tamaño de imágenes intercambiados al inicio, conoce el origen de los datos así como si se trata de una imagen JPEG, una imagen decodificada o del recuento de polillas.

En una futura aplicación del sistema, como se muestra en la sección 5.5, el nodo *sink* podría enviar datos de sincronismo y control de la red hacia los nodos aguas abajo como ser el horario de encendido, cambios en el ruteo, entre otros.

3.3. Implementación

El software se implementó a partir de una serie de procesos de Contiki OS y funciones complementarias que manejan los distintos módulos del sistema utilizando servicios proporcionados por el RTOS como ser el *stack* de comunicación Rime, Etimers y CFS-Coffee. El hilo principal es el encargado de ejecutar secuencialmente los demás módulos: inicialización de la cámara, capturar y procesar la imagen, enviar por radio y manejo de modo *sleep*.

Los procesos y funciones están implementados en los distintos módulos que se detallarán en las secciones siguientes. También se detallan los servicios de Contiki OS utilizados y su interacción con los mismos.

En el anexo B se encuentra una descripción de la compilación, carga y puesta en marcha de las aplicaciones.

3.3.1. Módulos

Para cumplir con los requerimientos funcionales del sistema se implementaron los módulos mencionados en la sección 3.2, que para los nodos con funcionalidad completa y reducida corresponden a:

- `camara.c`: Contiene el *driver* de la cámara, incluyendo el proceso que inicializa la cámara y el que captura imágenes.
- `jpeg.c`: Realiza el procesamiento de las imágenes JPEG tomadas por la cámara.
- `radio.c`: Contiene el proceso que maneja la transmisión de datos por la red.
- `sleep.c`: Implementa las funciones que llevan al sistema a un nivel de bajo consumo.
- `plagavision.c`: Ejecuta el proceso general que manjea al nodo.

En la figura 3.5 se puede observar un diagrama de interacción entre los distintos módulos de la aplicación, los servicios de Contiki OS, *drivers*, servicios de comunicación y periféricos. A lo largo de esta sección se detalla la implementación de cada uno y los obstáculos que se tuvieron en el transcurso de la programación de los mismos. Los códigos se encuentran en el CD adjunto y están comentados para facilitar la comprensión del lector.

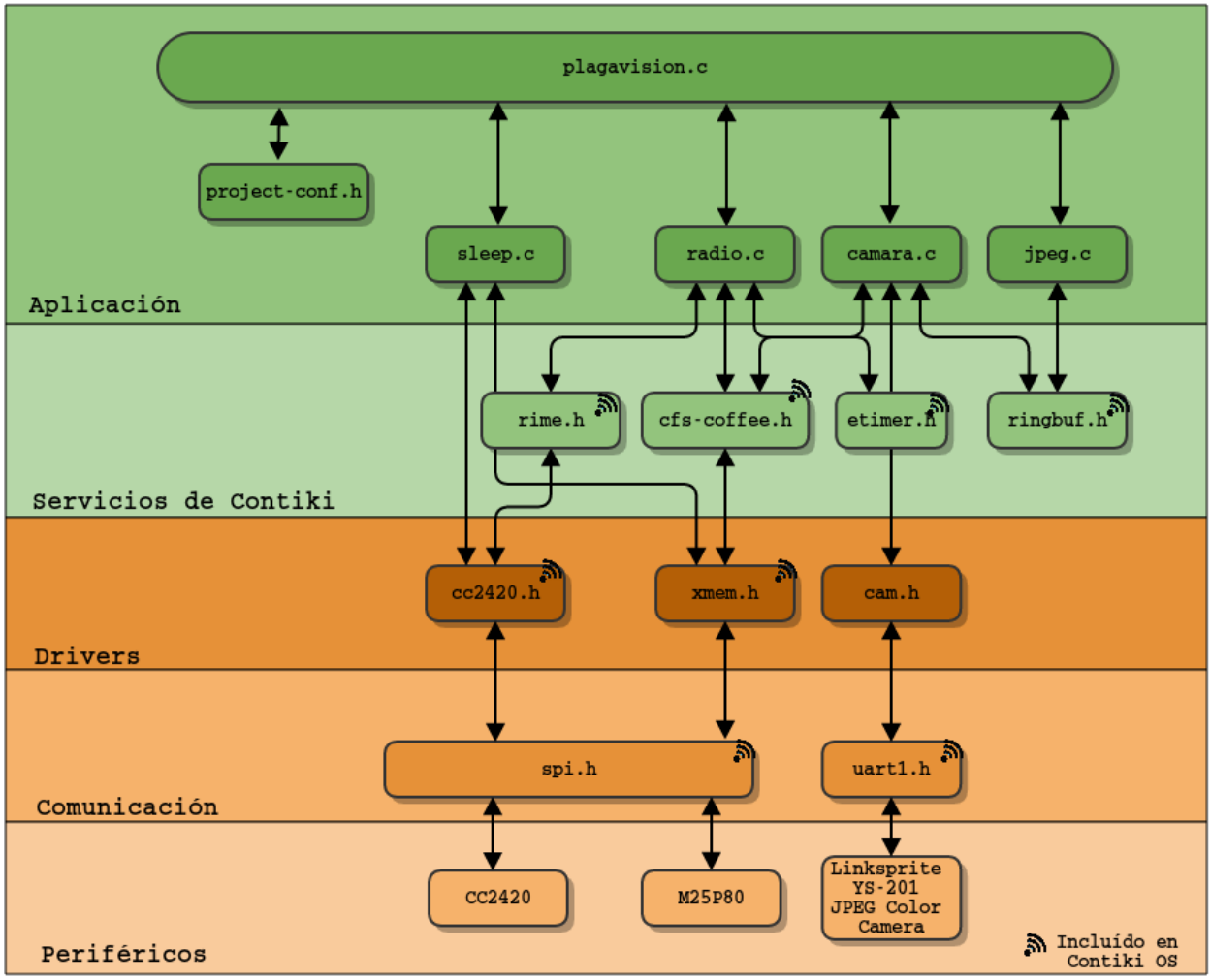


Figura 3.5: Diagrama de interacción entre módulos

Módulo Cámara

Este módulo implementa el *driver* de la cámara y tiene definidos todos los comandos detallados en la hoja de datos de la misma. Consta de tres procesos: `comando_cam`, `camara` y `foto`. El proceso `comando_cam` se utiliza para enviar los comandos simples a la cámara, es decir aquellos en los que la misma responde con un ACK sin datos extra (todos menos el comando de reseteo, el de descarga y el de lectura de tamaño). El proceso `comando_cam` es ejecutado a partir de funciones que le realizan un *post* luego de cambiar la estructura `scomando` que guarda: el comando que se desea enviar, el ACK esperado y el tamaño de los mismos.

Los tres procesos mencionados se comunican con la cámara a través del *driver* de la UART1. Para enviar comandos, se escribe por la UART1 y se espera la respuesta cediendo el CPU. Si el mismo no llega, se puede configurar un Etimer que luego de cierto tiempo tome una acción de contingencia. En lo que refiere a la recepción de datos, se debió configurar el *driver* de la UART1 para habilitar las interrupciones en recepción. Además fue necesario implementar una función de *callback* que es llamada por el *driver* cada vez que se ejecuta una ISR de recepción y realiza un *post* al proceso que solicitó la UART.

Los datos recibidos son almacenados en un *buffer* circular implementado por el sistema operativo y su tamaño debe ser potencia de 2 y como máximo 128 bytes (detalle que no se tuvo en cuenta en un principio). Además se debe tener en cuenta que cuando el *buffer* se llena, no sobreescribe el primer dato sino que se descarta la operación.

El tamaño del bloque de descarga del JPEG se estableció en 112 bytes, cumpliendo el requerimiento de ser múltiplo de 8 como se menciona en la sección 3.2.2. El tamaño del *buffer* circular se fijó en 128 bytes para poder almacenar los 112 bytes del bloque de descarga y los ACKs iniciales y finales.

El proceso `camara` se encarga de la inicialización de la cámara. En orden de ejecución: selecciona la resolución de la imagen y realiza la secuencia de reinicio necesaria, cambia el factor de compresión, cambia el *baudrate* de la comunicación a 115200 bps y finalmente llama al proceso `foto`.

La captura de imágenes es ejecutada en el hilo del proceso `foto`. Éste envía los comandos de capturar imagen, leer el tamaño de la misma, descargar la imagen y detener captura. Previo a enviar el comando de tomar foto, el proceso enciende los LEDs y los apaga luego de recibir el ACK del comando.

Durante la descarga de la imagen se debe incrementar la dirección de memoria a solicitar (cambiando los bytes MH y ML), guardar los datos en CFS-Coffee y a su vez controlar que llegue el fin de la imagen (`0xFF 0xD9`).

Las dificultades en la implementación de este módulo surgieron principalmente por la pobre documentación de la cámara, en la cual existen secuencias que no están debidamente explicadas. En el caso de la inicialización no se expresa la cantidad y el valor hexadecimal de los caracteres del *string* salvo el fin, mientras que en el caso de la captura de imágenes el diagrama de flujo es incorrecto. Estas fueron superadas leyendo foros y utilizando algunos códigos para la cámara implementados en otros lenguajes. En cuanto a Contiki OS las dificultades fueron menores, la mayoría por falta de una documentación estructurada y por falta de experiencia.

Módulo JPEG

La implementación de este módulo se encuentra detallada en la sección 4.3.2. La función central es `decodificar()` que ejecuta la decodificación del último archivo JPEG almacenado en CFS-Coffee. Ésta aplica un decodificador parcial JPEG con el que se puede obtener la imagen con información reducida y el recuento de polillas a través de un algoritmo de detección de patrones y otro de *labeling*.

Módulo Radio

Este módulo implementa las funciones y el proceso encargados de transmitir la información por la red. Como se detalla en el capítulo 5, se utilizó el *stack* de protocolos Rime dentro de las cuales se eligió Runicast. En la figura 3.6 se observa un diagrama del funcionamiento de este módulo.

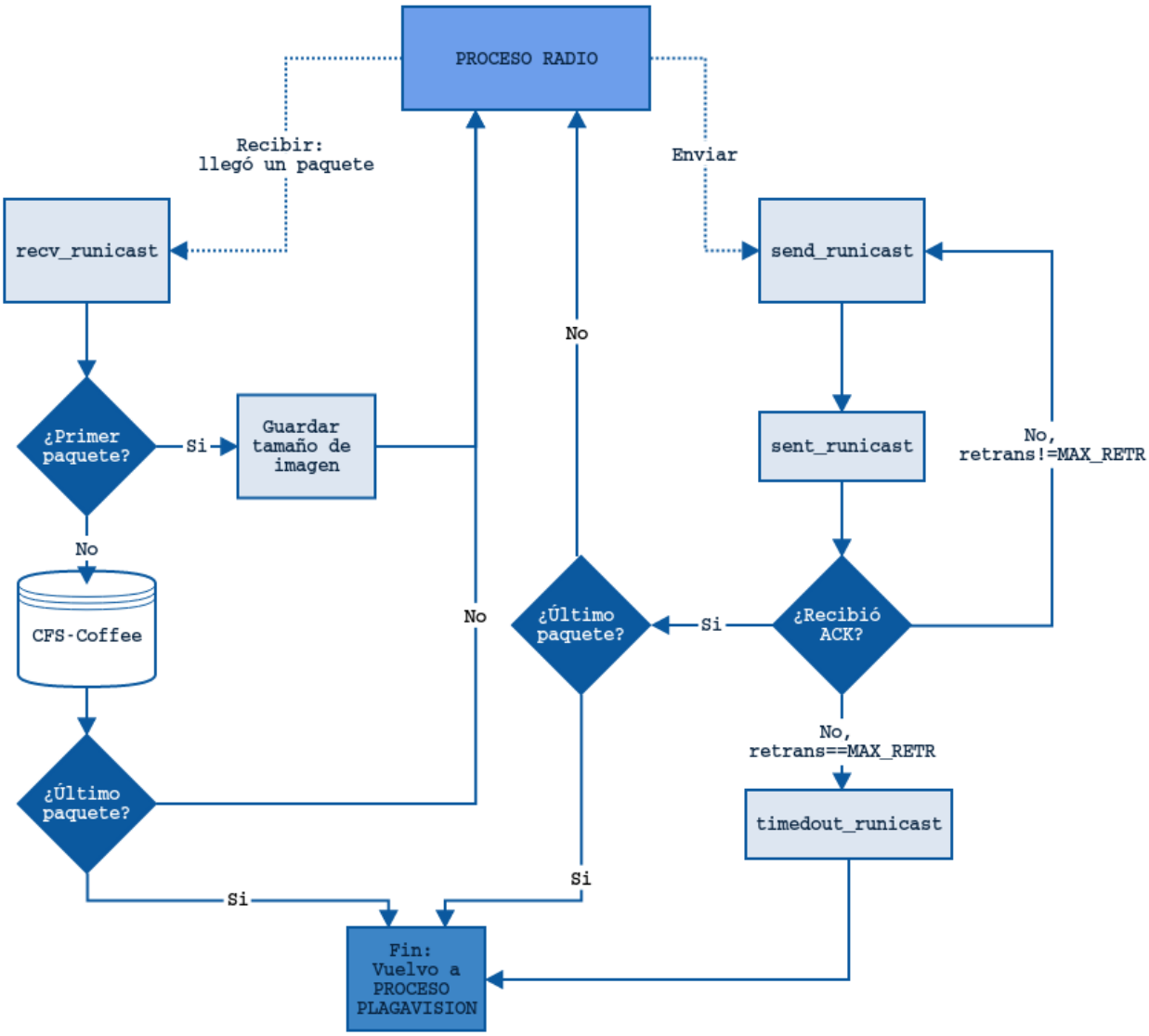


Figura 3.6: Diagrama del módulo de red

Para configurar Runicast se deben definir la cantidad máxima de retransmisiones, la carga útil del paquete y la dirección del receptor. Además es necesario implementar tres funciones de *callback* que Contiki OS provee como prototipo, `recv_runicast`, `sent_runicast` y `timedout_runicast`. La primera es llamada cada vez que se recibe correctamente un paquete por la radio, siendo necesario definir qué se debe realizar con el mismo. En este caso el nodo detecta el paquete inicial donde se encuentra el tamaño del archivo y luego almacena en CFS los datos hasta que se alcanza el tamaño especificado al inicio de la transmisión. `sent_runicast` es llamada cada vez que se envía correctamente un paquete por la radio. Se utilizó una implementación de ejemplo escrita por los creadores del RTOS que consiste en imprimir un error en consola vía UART para poder depurar. Por último, `timedout_runicast` es llamada cada vez que se alcanza el máximo de retransmisiones y también se utilizó el código de ejemplo para la etapa de pruebas cuyo funcionamiento es análogo al anterior para imprimir un error.

El proceso principal `radio` abre el canal de comunicación para utilizar Runicast y luego envía secuencialmente los datos, cargando el paquete con los datos leídos desde CFS-Coffee. En el primer

paquete se envía el tamaño de la imagen a transmitir y sucesivamente se envían paquetes de tamaño fijo hasta completar la imagen (salvo el último ya que la variabilidad del tamaño hace que no sea necesariamente múltiplo de la carga útil).

Si se desea transmitir más imágenes, como es el caso de los nodos con funcionalidad completa, se comienza la secuencia anterior nuevamente. De la misma manera, si se desean transmitir otros datos se envía un paquete inicial con una etiqueta y el tamaño de los datos a enviar. La etiqueta funciona como una bandera de inicio y como identificación para que el receptor realice un tratamiento diferente de los datos.

Módulo *Sleep*

El módulo *Sleep* consta de seis funciones que administran los periféricos del sistema para garantizar un bajo consumo de energía, tres para encender y tres para apagar (o llevar a modo *sleep*) respectivamente la radio, la memoria flash y la cámara. En esta sección se detallarán algunas fracciones de código de las funciones implementadas puesto que llevar a cabo este módulo supuso una de las dificultades más duras en el transcurso del proyecto. En la comunidad de Contiki OS no hay información suficiente acerca de cómo bajar el consumo de corriente de los nodos más allá de lo implementado en el RTOS, por lo que se tuvieron que implementar funciones específicas para lograr este cometido. Éstas corresponden a:

- `dormir_radio()`
- `despertar_radio()`
- `dormir_flash()`
- `despertar_flash()`
- `apagar_camara()`
- `prender_camara()`

La función `dormir_radio` realiza las siguientes tareas:

1. Apaga la radio con `NETSTACK_MAC.off(0)`
2. Apaga el oscilador de la radio: `CC2420_STROBE(CC2420_SXOSCOFF)`
3. Apaga el regulador de voltaje de la radio: `SET_VREG_INACTIVE()`

Para encender nuevamente la radio CC2420 es necesario ejecutar toda la rutina de inicialización, esto se realiza mediante la función `cc2420_init()` contenida en el *driver* implementado por Contiki OS. Luego se debe reasignar la dirección de Rime y reestablecer el identificador de red del nodo (PAN ID). Por último se reinician los módulos del NETSTACK (véase el capítulo 5). En resumen la secuencia es:

1. Inicializar la radio: `cc2420_init()`.

2. Reasignar dirección Rime con `rimeaddr_node_addr` y PAN ID mediante `cc2420_set_pan_addr`.

3. Reiniciar RDC: `NETSTACK_RDC.init()`.

4. Reiniciar MAC: `NETSTACK_MAC.init()`.

5. Reiniciar red: `NETSTACK_NETWORK.init()`.

6. Encender MAC y RDC: `NETSTACK_MAC.on()`

En lo que respecta a la memoria flash externa, ésta posee un modo *sleep* (llamado *Deep Power Down*) con el cual el consumo es como máximo 10 μA según declara el fabricante. Para que la memoria entre en dicho modo de trabajo simplemente se envía el comando SPI `SPI_FLASH_INS_DP` con código `0xB9`. Para despertar la memoria flash se envía el comando SPI de reinicio `SPI_FLASH_INS_RES` cuyo código hexadecimal es `0xAB`.

Para llevar el MCU a LPM3 se utiliza la función `msp430_remove_lpm_req()` con la que se cambia el estado de una bandera que le indica al RTOS si debe mantenerse encendido el oscilador interno controlado digitalmente (DCO). Cuando no hay requerimientos de DCO se ejecuta el macro `_BIS_SR(GIE | SC1 | SC2 | CPUOFF)` que introduce el LPM3.

Por último, las funciones para encender y apagar la cámara solamente manejan pin GPIO 2.1 del MSP430 para activar y desactivar el switch FPF2004. La función `prender_camara` se ejecuta antes de comenzar con el proceso `camara` y `apagar_camara` justo después de terminado el proceso *foto*.

Módulo Plagavisión

Este módulo implementa el proceso central para el manejo del nodo y ejecuta secuencialmente los procesos y funciones que cumplen con las especificaciones funcionales del sistema. El diagrama de la figura 3.7 muestra cómo corre el hilo principal llamado `nodo`. Comienza inicializando distintos parámetros y módulos (incluido el Etimer para encender al nodo cuando corresponda), ejecuta las funciones para apagar la radio, llevar la memoria flash a modo *Deep Power Down* y llevar el MCU a LPM3. Vencido el Etimer se enciende la cámara y la memoria flash y se lanza el hilo del proceso *camara*. Luego se ejecuta el proceso *foto* y a su término se apaga la cámara. A continuación se enciende la radio y se realiza un *post* al proceso *radio* para enviar los datos. Luego se apaga la radio y se procesan las imágenes mediante la función `decodificar`. Al finalizar el procesamiento, se prende nuevamente la radio y se envía el conteo y/o la imagen con información reducida. Por último se borran las imágenes enviadas y se vuelve al comienzo.

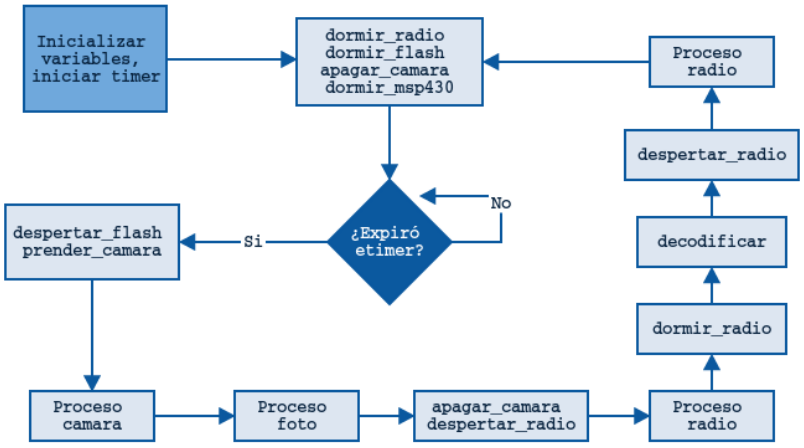


Figura 3.7: Diagrama de ejecución del proceso nodo en el módulo Plagavisión

Tamaño del código compilado

El código compilado de la aplicación completa incluyendo el RTOS ocupa $7,45\ kB$ de memoria RAM y $35,67\ kB$ de memoria ROM. Se puede observar que la aplicación ocupa el $74,5\,\%$ de la memoria RAM disponible y el $74,3\,\%$ de la memoria ROM disponible por lo que se están usando gran parte de los recursos de memoria del nodo. Existen métodos para optimizar la memoria consumida por la aplicación y reducir el *firmware* del RTOS que no fueron exploradas por resultar innecesarias.

3.4. Tests y depuración

Uno de los grandes obstáculos sorteados durante el proyecto fue encontrar un monitor serial adecuado para poder depurar el *driver* de la cámara Linksprite. No existía experiencia en utilizar este tipo de aplicaciones ya que en general se depura *on-target* mediante JTAG o con algún entorno de desarrollo mediante el USB del nodo.

Nuevamente se debe destacar el hecho de que algunos pines de la UART1 están compartidos con el FTDI en la versión TelosB o la TelosB compatible CM5000. Esto implica que al conectar la UART de la cámara a esta interfaz se podría producir un cortocircuito entre la línea de RX de la cámara y el FTDI, si ambos dispositivos utilizaran la línea simultáneamente. Ese conexionado se ejemplifica en la figura 3.8. Las interfaces UART poseen protecciones para evitar estos cortocircuitos pero aún así la comunicación será fallida. Por esta razón no existe una configuración posible que permita monitorear la comunicación serial con la cámara ya conectada al nodo sin involucrar hardware adicional.

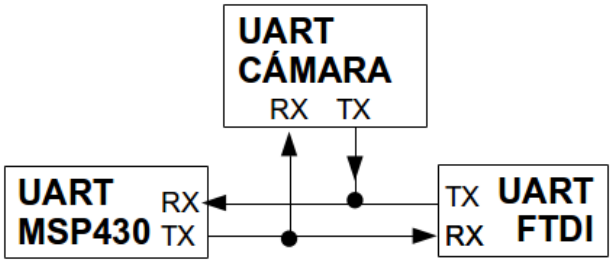


Figura 3.8: Ilustración de conexión UART que generaría conflicto entre el FTDI y la cámara

Las opciones manejadas fueron:

i) Utilizar dos UARTs adicionales, una que recibiera lo enviado por el nodo a través del FTDI y la otra para recibir los datos de la cámara por ejemplo utilizando un Arduino.

ii) Implementar una UART mediante *bit-banging* con puertos GPIO para comunicarse con la cámara.

iii) Utilizar un monitor serial (*serial sniffer*) en una PC que estableciera un túnel entre ambos dispositivos.

La opción elegida fue la *iii)* ya que se encontró una aplicación conveniente en Linux para ello y a través del conversor serial a USB de un Arduino se pudo conectar la cámara a la PC. En la figura 3.9 se ilustra el setup de trabajo donde se tiene un nodo TelosB y el Arduino conectado a la PC corriendo la aplicación *Sersniff*. El Arduino tiene cortocircuitado Reset con GND lo que permite que el microcontrolador permanezca apagado.

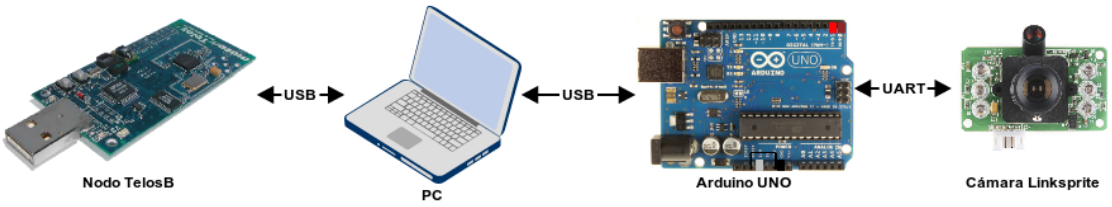


Figura 3.9: Setup para realizar la depuración de la comunicación Nodo-Cámara

Los módulos y distintas funciones del software fueron depurados en el transcurso de la realización de los mismos. Las principales herramientas de depuración y prueba fueron *Serial-dump*, *Sersniff* y *GTKTerm*, mientras que para la simulación de red se utilizó el software Cooja.

Serial-Dump es una terminal serial provista por Contiki OS con la que es posible establecer una comunicación serial con los nodos que corren este RTOS a través del puerto USB de una PC. Para lanzar dicha aplicación basta con abrir una terminal de Linux y en el directorio donde se encuentra el proyecto en el cual se está trabajando escribir `make login`. Con esta aplicación se realizaron las primeras pruebas en Contiki OS, compilando ejemplos y posteriormente para probar la comunicación UART.

Sersniff [65] es una aplicación de código abierto escrita en C para Linux. Permite establecer un túnel entre dos puertos seriales conectados a un PC de manera que exista una conexión virtual entre ambos. La PC actúa de pasamanos y es posible observar en la terminal de Linux los datos intercambiados en formatos hexadecimal o ASCII. El comando para ejecutar el programa tiene básicamente la siguiente sintaxis:

```
sersniff [-i DEVICE ] [-o DEVICE] [-b BAUD] [-s] [-x] [-f] [-z]
```

La configuración es la que sigue:

- DEVICE debe ser sustituido por los puertos entre los que se quiere establecer la comunicación.
- BAUD corresponde a la velocidad de la conexión en *bps*.
- -s se coloca si se quiere que no se haga un pasamanos entre los dispositivos y solo se muestre lo que estos envían.
- -x se utiliza si se quiere que se desplieguen los datos en hexadecimal.
- -z es utilizado para que no se corte la conexión al recibir un caracter de *End Of File* (EOF).
- -f permite cambiar el formato de salida de los hexadecimales.

El *Sersniff* se utilizó para realizar la depuración de los módulos relacionados con la cámara, siendo necesario para saber si los comandos enviados y recibidos eran los esperados. También permitió depurar otros módulos al poderse imprimir en consola desde el nodo, ya sea para evaluar el funcionamiento del CFS-Coffee y la radio, así como para observar el estado de distintas variables.

GTKTerm es una terminal serial para Linux que permite establecer una comunicación con un dispositivo serial desde la PC. Solamente se debe configurar el puerto, *baud rate* y pocos parámetros más en el entorno gráfico para establecer la comunicación. Los caracteres se despliegan en ASCII o hexadecimal según la configuración. Este software se utilizó para verificar que la recepción de datos funcionaba correctamente.

Las pruebas realizadas a los distintos módulos y los indicadores de éxito corresponden a:

- **Comunicación UART:** Se realizó un pequeño módulo que recibe datos y replica los mismos por la UART1. Se probó para distintos tamaños de *buffer* circular y ráfagas.
- **Cámara:** Utilizando *Sersniff* se implementó un proceso que envía distintos comandos a la cámara y compara la respuesta obtenida con la esperada. Este módulo se fue ampliando resultando en el proceso *camara* que conforma el controlador de la cámara Linksprite LS-Y201.
- **CFS-Coffee:** Se utilizó una aplicación de ejemplo incluida en el repositorio de Contiki OS llamada `example-coffee` con la cual se probó la escritura y lectura desde el sistema de archivos homónimo.
- **Integración Cámara-Coffee:** Utilizando el monitor serial se probó que lo descargado desde la cámara y lo guardado en flash se correspondiera, conformando una imagen válida. Para esto último se tuvo especial cuidado al leer los caracteres como números hexadecimales y no ASCII de manera que el visor de imágenes interpretara correctamente el archivo. En nuestro caso se guardó la salida en consola del *GTKTerm*.

- **Runicast radio:** Se realizaron pruebas del funcionamiento del protocolo *Runicast* modificando un ejemplo de Contiki OS llamado `example-runicast` para que los paquetes se armen con lecturas desde CFS-Coffee. Para verificar el correcto funcionamiento se conectó el receptor a un PC, el cual imprime vía UART en una consola serial.
- **Receptor:** Este módulo de prueba se implementó sobre el anterior, pero sólo configurando la función de *callback* de recepción de manera que imprima por UART todo lo recibido. Luego se le agregó el control de secuencia y otras funciones resultando en la aplicación final del receptor. Para evaluar el correcto funcionamiento de este módulo se comparó el archivo de salida del receptor con el guardado en el CFS del transmisor.
- **Modo *Sleep*:** Se escribió un módulo que implementa las funciones que llevan al nodo a estados de bajo consumo. El mismo consiste en llevar la radio, la memoria flash y el MCU a estados de bajo consumo y enciende todo al vencerse un Etimer. Se realizaron medidas tal como se menciona en la sección 6.1.1. Estas funciones fueron integradas y depuradas en el módulo *sleep* de la aplicación final.
- **Procesamiento de imágenes:** Para evaluar el algoritmo de procesamiento de imágenes se utilizó también *Sersniff* para permitir la comunicación con la cámara e imprimir en consola la salida de las distintas funciones del módulo. Estas pruebas se realizaron de manera integral con la aplicación restante, fundamentalmente lo pertinente a la captura de las imágenes y su almacenamiento en memoria flash.

3.4.1. Interfaz de Matlab

Para desplegar las fotos tomadas por la cámara y la detección automática se creó una interfaz serial en Matlab, cuya única función es desplegar la información enviada por el nodo de manera amigable. Es importante destacar que no se hace ningún procesamiento en este software, es simplemente una interfaz para mostrar los resultados.

El programa escrito lee el puerto serial al cual está conectado el nodo y lee la información que llega por el mismo buscando caracteres especiales de inicio o fin. Si detecta el caracter de inicio de una imagen JPEG (`0xFF 0xD8`), empieza a guardar todo lo que le llega a continuación en una matriz hasta recibir el caracter de fin (`0xFF 0xD9`). Luego de recibido, muestra la imagen en pantalla.

Si detecta el caracter de inicio del vector de coordenadas x o y (`0xFF 0x0B` y `0xFF 0x0C` respectivamente), guarda las coordenadas en una matriz hasta recibir el caracter de fin (`0xFF 0xC8`). Una vez que recibió ambos vectores de coordenadas, superpone cruces en la imagen original JPEG en las coordenadas indicadas por el vector.

De esta manera se pueden sacar y desplegar muchas imágenes consecutivas y así testear con rigurosidad tanto el software del nodo como el algoritmo de procesamiento.

Capítulo 4

Procesamiento de imágenes

Resumen

El presente capítulo demuestra la pertinencia de aplicar técnicas de procesamiento de imágenes en WSN, en nodos con hardware reducido. Contiene la descripción del decodificador JPEG implementado en el sistema, así como la descripción de las funciones de detección y conteo de polillas.

Para facilitar la familiarización del lector con el estándar de compresión JPEG, se incluye una pequeña introducción a los principios sobre los que se basa su funcionamiento y una descripción de la estructura de los archivos.

Se trabajó en colaboración con un grupo de estudiantes del curso *Tratamiento de imágenes en computadora* del IIE-FING-UdelaR, quienes investigaron acerca de algoritmos de procesamiento en el espacio frecuencial y sugirieron aplicar un algoritmo de clasificación lineal Fisher que actúe directamente en el espacio frecuencial.

Se detalla la implementación de un algoritmo de *labeling* adaptado a las capacidades del hardware utilizado y de un método de entrenamiento propuesto en [66] con un banco de imágenes especialmente recolectado. Finalmente se muestran los resultados del conjunto, demostrando la aplicabilidad de la técnica a las WSN, en particular logrando hacer un conteo certero de la cantidad de polillas en la trampa.

Contenido

4.1. Introducción

4.1.1. Motivación del tratamiento de imágenes en WSN

Tradicionalmente las WSN han orientado su propósito al relevamiento remoto de magnitudes físicas representables por algunas decenas de bytes con la mayor autonomía posible [67], por lo que se prioriza la economía energética sobre la capacidad de procesamiento y la disponibilidad de memoria. Esta característica supone un obstáculo mayor a la hora de implementar una aplicación con una gran demanda de recursos, como son las relacionadas con el análisis y transmisión de imágenes. Por otro lado la evolución de la tecnología en los sensores de imagen CMOS permite obtener dispositivos de bajo consumo y de costo accesible, lo que hace factible la creación de una WSN con capacidad de obtención de imágenes. Algunas de las soluciones propuestas se orientan a la adecuación del hardware a la aplicación [68], mientras que en otros casos se utiliza el nodo de forma que el MCU actúe como un intermediario entre la cámara y la radio [69], sin intervenir la información de ninguna manera.

Como complejidad añadida, los archivos de imagen suelen tener tamaños de decenas a cientos de kilobytes por lo que su transmisión de un nodo a otro demanda de cientos a miles de paquetes, lo que a su vez supone una gran cantidad de energía invertida en dicha transmisión. En la aplicación abordada en este proyecto cabe la posibilidad de que la mayoría de las imágenes tomadas no representen capturas de plagas, por lo que no tendría sentido invertir energía en enviarlas. Esta situación despierta el interés en tener la capacidad de procesar y analizar el archivo de imagen en el nodo para así poder tomar una decisión preliminar sobre la información que contiene.

En los trabajos relacionados estudiados no se encontraron otros sistemas en los que se implementaran técnicas de procesamiento de imágenes en WSN con las restricciones de hardware de este proyecto. En [6] y [11] se implementan técnicas de procesamiento de imágenes, pero en sistemas con la capacidad de realizar una descompresión JPEG completa. En [38] se hizo una primera aproximación a las técnicas de procesamiento de imágenes en el espacio matricial sobre una plataforma TelosB compatible sin lograr resultados prometedores, ya que el tiempo de procesamiento se hacía por

demás extenso. En función de esto se buscó implementar técnicas de procesamiento de la imagen JPEG comprimida sobre el espacio frecuencial, haciendo solamente una descompresión parcial.

La investigación en el área del procesamiento de imágenes comprimidas fue popular en los años 90, pero el incremento exponencial en la capacidad de procesamiento y memoria de las computadoras hizo que este tipo de algoritmos perdiera vigencia. Los requerimientos funcionales del sistema planteado en este proyecto hacen que vuelva a ser interesante desarrollar este tipo de técnicas.

4.1.2. Introducción a la implementación

La solución abordada en esta sección se diseñó tomando en cuenta las limitaciones del hardware mencionadas en el capítulo 2, particularmente las referentes a la memoria RAM y flash [1]. La cámara empleada en el proyecto envía las imágenes comprimidas en formato JPEG, por lo que adicionalmente fue necesario implementar un decodificador que permitiera acceder a la información contenida en el archivo.

Los requerimientos de bajo consumo del proyecto despiertan interés en desarrollar un algoritmo que sea capaz de interpretar una imagen para determinar cuántos insectos ha capturado la trampa, basándose en la comparación con determinadas características propias de las imágenes de las polillas. Esta capacidad permite que en lugar de enviar una foto completa (o en el mejor de los casos una imagen con información reducida) se pueda enviar un conteo de polillas que requiera de unos pocos bytes. Sin embargo el procedimiento sólo es válido si la técnica es lo suficientemente robusta, por lo que se debe caracterizar y comprobar su funcionamiento.

Los decodificadores que se encuentran disponibles no son compatibles con nuestras necesidades ya que requieren una cantidad de recursos mayor a la disponible [70] [71]. El decodificador implementado permite recorrer el archivo JPEG y acceder a los coeficientes DCT de cada bloque, permitiendo tomar distintas medidas a partir de esta información. La primera consiste en tomar solamente los coeficientes DC de cada bloque y generar una imagen reducida de 80×60 píxeles en escala de grises, con una profundidad de color de 8 bits que se corresponde con un submuestreo de 64 a 1 en la imagen original. Esta nueva imagen representa los objetos capturados en la trampa con muy pocos píxeles por lo que se hacen difícilmente reconocibles pero presenta la ventaja de poseer un tamaño fijo (4800 bytes) considerablemente menor al del archivo comprimido completo. Otra posibilidad es clasificar los bloques usando características calculadas a partir de los componentes DCT con un clasificador lineal Fisher. Por otra parte deja la puerta abierta a realizar otros tipos de algoritmos sobre el dominio comprimido.

4.1.3. Funcionalidades de la aplicación

Las funciones básicas de la aplicación son:

- Decodificar parcialmente el archivo JPEG, generar la imagen en formato crudo con información reducida y almacenarla en CFS-Coffee.

[1] Esto se debe a que una descompresión JPEG requiere la utilización simultánea de múltiples arreglos de gran tamaño, que no son manejables en una plataforma como el TelosB, limitada a 10kB de RAM

- Analizar cada uno de los bloques del archivo JPEG para determinar la probabilidad de que éste contenga un espécimen de *Cydia Pomonela* o *Cydia Molesta*.
- Agrupar los bloques cuyo análisis haya arrojado resultados positivos por regiones conexas y almacenar las coordenadas del centroide de cada grupo, de forma que cada una de ellas corresponda a una polilla.

4.2. Metodología de trabajo

La implementación de un decodificador JPEG es sumamente trabajosa debido a lo compleja que se hace la depuración de errores. La decodificación debe contar con varios mecanismos perfectamente sincronizados para funcionar, y es difícil acceder a resultados intermedios. Como complejidad añadida, la ejecución en un MSP430 requiere la utilización constante de CFS-Coffee y la implementación de un mecanismo de almacenamiento provisorio para el manejo de la imagen, ya que la memoria RAM no es suficiente para alojar el archivo.

Para atacar estos problemas de forma ordenada se decidió implementar una versión preliminar del decodificador en lenguaje C compilado en Matlab mediante la utilización de archivos MEX [72]. Una vez solucionado este problema se procedió a portar el programa a Contiki OS, implementando funciones de administración de un *buffer* que mantiene provisoriamente en RAM secciones de los archivos de CFS-Coffee.

4.3. Decodificador JPEG

4.3.1. Composición de un archivo JPEG

JPEG es un estándar de compresión de imágenes que permite disminuir el tamaño de una imagen hasta uno o dos órdenes de magnitud respecto a su equivalente en formato crudo, aunque introduce pérdidas de información. Para lograr factores tan altos de compresión se basa en las siguientes particularidades:

- El ojo humano es más sensible a las variaciones de iluminación que a las de color, lo que permite submuestrear los canales correspondientes sin perder información relevante.
- Si se hace una transformación de la imagen que pase del dominio espacial al frecuencial, puede observarse que la eliminación o atenuación de la información de alta frecuencia no es distinguible a simple vista. Esto se debe a que el ojo humano actúa como un filtro pasabajos (e.g. si se observa un patrón de franjas negras y blancas a la distancia se observará un color gris uniforme).
- Es altamente probable que dos píxeles cercanos sean muy similares. Hay una fuerte correlación entre bloques adyacentes.

Antes de analizar el decodificador JPEG resulta interesante estudiar el proceso de codificación, dado que permite explicar la razón de cada una de las operaciones ejecutadas [73] [74].

El diagrama de la figura 4.1 muestra los pasos para realizar una compresión JPEG.

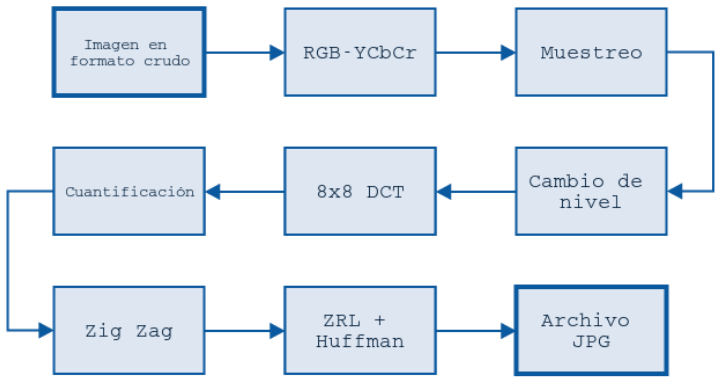


Figura 4.1: Diagrama de flujo de la compresión JPEG

- **Conversión RGB - YCbCr**:

  Un color debe ser definido por al menos tres parámetros. Generalmente se trabaja con bases ortonormales, donde las coordenadas sobre cada versor definen dicho color. Dos de las bases más utilizadas son (R,G,B), donde los versores son el canal rojo, verde y azul. Otra base utilizada regularmente es (Y,Cb,Cr), donde el versor Y se denomina luminancia y se define como la energía lumínica irradiada por una superficie, ponderada de acuerdo a una función característica del ojo humano que balancea los colores rojo, verde y azul. Los versores Cb,Cr definen un plano perpendicular a Y denominado croma, donde el versor Cb es cercano al color azul y Cr al rojo. JPEG trabaja sobre la base (Y,Cb,Cr) para explotar las características de la visión humana.

  La primera acción que se lleva a cabo para crear un archivo comprimido consiste en llevar la imagen al espacio de color YCbCr para distinguir la información de iluminación de la de color. Sencillamente se debe multiplicar cada tríada RGB por una matriz de cambio de base.

- **Muestreo**: Una vez realizado el cambio de base en la imagen se submuestrean los canales de color a una relación determinada, en el caso de nuestra cámara 2:1:1 (esto es, dos píxeles adyacentes horizontalmente del canal Y se corresponden con uno del canal Cb y uno del Cr). Debido a la diferente sensibilidad del ojo humano a la luminancia y a la crominancia este submuestreo permite eliminar gran parte de la información sin alterar la percepción visual de la imagen.

- **Cambio de nivel**: Se traslada la imagen a una representación de 8 bits con signo desde una de 8 bits sin signo.

- 8×8 **DCT**: Una vez que se tienen los canales submuestreados se agrupan los píxeles de cada canal en grupos de 8×8, denominados bloques, y se aplica la DCT sobre cada uno de ellos. Dicha transformación tiene la capacidad de trasladar los bloques del dominio espacial al frecuencial, lo que nos permite depreciar la información de alta frecuencia tal como se mencionó en la sección 4.3.1. En la figura 4.2 se observa una idea intuitiva del funcionamiento de la operación.

  La expresión formal de la DCT no es relevante para nuestra aplicación y puede consultarse en [73]. Si bien se podría usar otro tipo de transformación (como por ejemplo la de Fourier), se utiliza la DCT debido a su mayor tolerancia al truncamiento de coeficientes, esto significa

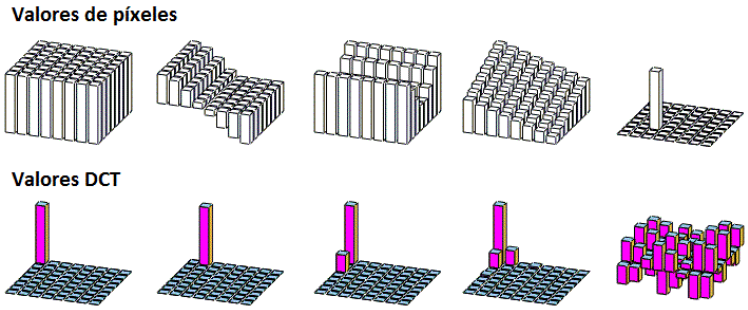


Figura 4.2: Funcionamiento de la DCT. Tomada de `www.hdtvprimer.com`

que la eliminación de los coeficientes de alta frecuencia distorsiona la reconstrucción en una forma menos apreciable.

- **Cuantificación y ordenamiento en Zig Zag**: Una vez que se tiene cada bloque transformado por la DCT se reordena de acuerdo a la figura 4.3

  Luego de calcular la DCT de cada uno de los cuadros se deprecia la parte de la información menos apreciable por el ojo humano. Para esto se divide elemento a elemento por una matriz que posee valores cada vez mayores al aumentar la frecuencia, denominada matriz de cuantificación. Al decodificar la información debemos multiplicar por la misma matriz de cuantificación, por lo que si la representación de los coeficientes no estuvieran cuantificados (recordemos que cada coeficiente tiene una precisión de 8 bits) la operación no tendría efecto alguno. Como los coeficientes tienen valores discretos, lo que estamos logrando en realidad es disminuir la apreciación de los coeficientes de alta frecuencia, que es uno de los objetivos mencionados. Como ejemplo podemos suponer que uno de los valores de un supuesto cuadro es 66 y el valor correspondiente de la matriz de cuantificación es 8. La división entera de 66 entre 8 da 8, y al multiplicar nuevamente por 8 obtenemos 64. Lo que se hizo fue representar el valor 66 con saltos discretos de 8 unidades.

  Luego de cuantificar los coeficientes de la DCT se reordena el cuadro de acuerdo a la figura 4.3.

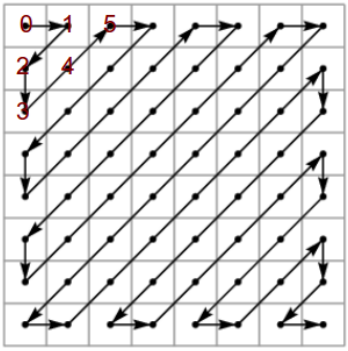


Figura 4.3: Esquema de reordenamiento de un bloque JPG

El resultado de este proceso es un vector de 64 elementos para cada uno de los cuadros. Su primer elemento (coeficiente de DC) representa el valor promedio del bloque de 8×8 y los 63 elementos restantes (coeficientes de AC) representan la información de frecuencia.

- **Codificación ZRL y Huffman**: Esta es la operación más compleja de la codificación. Se parte de la base de que el vector conformado por el bloque transformado, reordenado en zigzag

y cuantizado tiene secuencias largas de ceros, principalmente en los elementos correspondientes a las frecuencias mayores.

Se utilizan técnicas diferentes para comprimir los coeficientes de DC y de AC. Los valores de DC se codifican en diferencias[2] mediante el método de Huffman. Este método de compresión aprovecha la correlación entre cuadros adyacentes.

Para los coeficientes de AC, en lugar de crear un diccionario de codificación y codificar cada byte en forma directa se opta por utilizar una técnica denominada Zero Run Length Coding (ZRL), que consiste en codificar cada elemento distinto de cero como la cantidad de ceros que lo preceden seguido del elemento. Por ejemplo, la cadena [1 3 0 0 0 0 0 12 0 0 5 8] se codifica como (0,1)(0,3)(5,12)(2,5)(0,8). Como se verá a continuación, la cantidad de ceros precedentes debe quedar almacenada en cuatro bits por lo que no es posible codificar cadenas de más de quince ceros. En el caso que se tengan más, se codifica (15,0) y se sigue con la diferencia.

El estándar determina que en lugar de almacenar cada uno de los valores distintos de cero se debe almacenar el mínimo número de bits en el cual se puede representar dicho valor, seguido de un complemento según se observa en la tabla 4.1.

Valor	Min. bum. de bits	Complemento
0	0	-
-1,1	1	`0,1`
-3,-2,2,3	2	`00,01,10,11`
-7,-6,-5,-4,4,5,6,7	3	`000,001,010,011,100,101,110,111`
...	...	...
-32767,..,-16384,16384,..,32767	15	...

Tabla 4.1: Tabla de valores a codificar

Sea n el mínimo numero de bits necesario, c el complemento de la tabla y b_n el bit más significativo de dicho complemento, el número x a codificar se reconstruye como:

$$x = \frac{1 - b_n}{2}(-2^n + 1) + c \tag{4.1}$$

Aprovechando que el mínimo número de bits necesario puede almacenarse en un *nibble*[3] (ya que como se observa en la tabla 4.1, es un entero entre 0 y 15), se utiliza un único byte para almacenar este número y la cantidad de ceros que preceden al valor que estamos representando. La siguiente expresión se corresponde con el ejemplo anterior, donde el byte mencionado se representa entre paréntesis, seguido del complemento antes mencionado

[0,1]`1` [0,2]`11` [5,4]`1100` [2,3]`101` [0,4]`1000`

El último paso consiste en codificar mediante el método de Huffman cada uno de los bytes representados entre paréntesis rectos, que contienen el número de ceros y la mínima cantidad de bits necesarios, codificando hasta terminar el cuadro.

Suponiendo que se utiliza una relación de submuestreo 2:1:1, los bloques se extraen del archivo JPEG en el siguiente orden:

[2]En lugar de almacenar el valor de DC del bloque actual se almacena la diferencia de DC con el bloque anterior.

[3]Un nibble está compuesto por cuatro bits, dos nibbles componen un byte

DC_{Y1}	AC_{Y1}	DC_{Y2}	AC_{Y2}	DC_{Cb}	AC_{Cb}	DC_{Cr}	AC_{Cr}	...

Una vez comprendido el proceso de compresión JPEG podemos pasar a analizar la composición de un archivo ya comprimido. Los archivos JPEG tienen secciones delimitadas por marcadores de la estructura `0xFF 0xXY`, donde el segundo byte varía de acuerdo al marcador. Las secciones más relevantes dentro del archivo son:

- Inicio de imagen - `0xFF 0xD8`. Marca el comienzo del archivo.
- Definición de tablas de cuantificación - `0xFF 0xDB`. Determina el comienzo de las tablas de cuantificación correspondientes a los canales Y y CbCr.
- Base de la DCT - `0xFF 0xC0`. Esta sección contiene la siguiente información:
  - Altura de la imagen en píxeles.
  - Ancho de la imagen en píxeles.
  - Orientación.
  - Relación entre cada uno de los componentes (canales de color) con las tablas de cuantificación, relación de submuestreo.
- Definición de la tabla de Huffman - `0xFF 0xC4`. Determina cuatro tablas para la codificación de Huffman, ya que los códigos cambian según se codifiquen elementos de luminancia o de color, de DC o de AC. Para cada tabla se envían 16 bytes que definen la cantidad de elementos del diccionario, donde el primer byte corresponde a códigos de largo 1, el segundo a códigos de largo 2 y así sucesivamente. A continuación se colocan las palabras del diccionario de codificación concatenadas, siendo responsabilidad del decodificador ordenarlas según su largo. El origen de cada diccionario depende de la implementación de la cámara, pudiendo ser determinístico o dependiente de la imagen.
- Inicio de escaneo - `0xFF 0xDA`. Precede a la información codificada de la forma explicada anteriormente.
- Fin de imagen - `0xFF 0xD9`. Marca el final de la imagen.

4.3.2. Características del decodificador implementado

El proceso de decodificar un archivo JPEG resulta muy exigente para un sistema de recursos hardware limitados como el que se utiliza en este proyecto. Además, en el caso de ser posible una descompresión completa, ésta no tiene utilidad ninguna ya que ni siquiera es posible almacenarla en la memoria flash del nodo. El decodificador implementado tiene la capacidad de crear una imagen en escala de grises a partir de los coeficientes de DC del canal Y (o lo que es equivalente, los primeros coeficientes de la DCT). Como la imagen original es de 640×480 y cada bloque es de 8×8 la imagen resultante tiene un tamaño de 80×60 píxeles. Adicionalmente se realiza una clasificación lineal de cada uno de los bloques del canal Y a partir de cinco parámetros que se detallan en la sección 4.4.1, almacenando las coordenadas de los bloques que el clasificador indique que son parte del insecto.

Este par de funcionalidades dan la posibilidad al sistema de enviar una imagen JPEG completa, una imagen en escala de grises con información reducida, las coordenadas de los objetos identificados como polillas, un recuento de los mismos u alguna combinación de ellas.

El diagrama de flujo de la figura 4.4 muestra la estructura general del decodificador implementado.

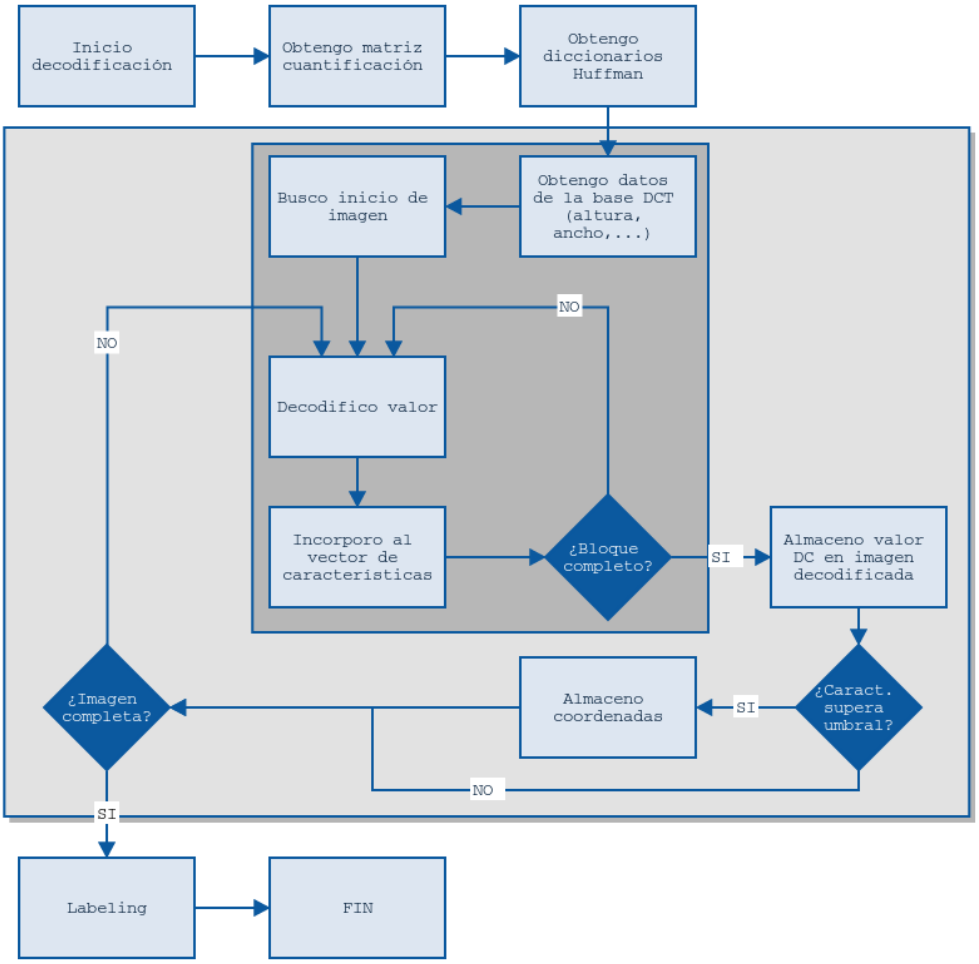


Figura 4.4: Diagrama de flujo del decodificador

La primera acción a realizar es recuperar las tablas de cuantificación, los diccionarios y los datos alojados en la base DCT (como relación de submuestreo, altura, ancho y profundidad de color). Una vez obtenida esta información se comienza la decodificación propiamente dicha de la imagen. El rectángulo oscuro en la figura 4.4 delimita el proceso iterativo que decodifica cada bloque, mientras que el claro delimita el encargado de decodificar la imagen entera. La decodificación de un valor se realiza de acuerdo a lo explicado en 4.3.1, obteniendo los bits mediante una función especialmente implementada para tal fin, que gestiona en forma automática un *buffer* que almacena 64 bytes del archivo JPEG en forma provisoria. Esto se hace para minimizar el acceso a flash mediante CFS-Coffee ya que éste es energéticamente costoso. Con los coeficientes decodificados se construye un vector de características que representará al bloque en la clasificación lineal.

Una vez terminada la decodificación del cuadro, se almacena el componente de DC en un archivo de CFS-Coffee que conforma la imagen con información reducida en formato crudo de 4800 bytes. Para esto también se se usa una función interfaz que maneja un *buffer* para minimizar los accesos a la flash.

El paso siguiente consiste en aplicar un clasificador lineal (tal como se explica en la sección

4.4.1), y en caso de que el mismo resulte positivo, se almacenan las coordenadas del bloque. Este proceso se repite hasta terminar la imagen, para luego aplicar un algoritmo de labeling sobre las coordenadas almacenadas, tal como se explica en 4.5.

Por una cuestión de complejidad del diagrama, en la figura 4.4 se omite mencionar que no se analizan las características de las componentes de color de los cuadros.

El programa se diseñó para que utilizara la menor cantidad de memoria RAM posible, siendo los mayores consumidores:

- Tabla de cuantificación del canal Y (64 bytes).
- Diccionarios de Huffman para DC Y, DC CbCr, AC Y y AC CbCr (16×4 almacenando cuantos códigos de cada largo hay en cada diccionario, 12×2 bytes para códigos de DC y 162×2 bytes para códigos de AC lo que totaliza 412 bytes).
- *Buffer* de lectura del archivo JPEG (64 bytes).
- *Buffer* de escritura del archivo comprimido (64 bytes).
- Vector de almacenamiento y comparación de características (2×5 bytes).
- Vectores de almacenaje de coordenadas (2×200 bytes).

El total de memoria RAM consumida es entonces del orden de 1 kB.

4.4. Detección de patrones

Una de las funcionalidades deseables del código consiste en la capacidad de distinguir entre bloques con imágenes del fondo y bloques con imágenes de polillas. Se propone para este fin la utilización del clasificador lineal Fisher (FLD) [75], ya que se trata de una técnica de clasificación lineal ampliamente conocida, aplicable en el dominio de la frecuencia y que requiere poco más de una decena de operaciones para clasificar cada bloque.

4.4.1. Principio de funcionamiento del FLD

Dado un grupo de objetos, puede seleccionarse un conjunto de características que los identifiquen. En nuestro caso, el vector de características representará a cada bloque de luminancia y estará contenido en $\mathbb{R}^5$, con cada coordenada correspondiendo a nivel de DC, energía total y energía ponderada contenida en los elementos 1+2, 3+4+5 y 6+7+8+9 de cada bloque, numerados según la imagen 4.3.

Sea $Y = (y_0, y_1, ..., y_{63})$ el vector que contiene los coeficientes de la DCT de un bloque Y ordenado en zig zag, la expresión de su vector de características es la siguiente:

$$\left(y_0, \sum_{i=1}^{63} {y_i}^2, \frac{\sum_{i=1}^{2} {y_i}^2}{\sum_{j=1}^{63} {y_j}^2}, \frac{\sum_{i=3}^{5} {y_i}^2}{\sum_{j=1}^{63} {y_j}^2}, \frac{\sum_{i=6}^{9} {y_i}^2}{\sum_{j=1}^{63} {y_j}^2} \right) \tag{4.2}$$

Supongamos que los objetos están separados en dos clases D_1 y D_2 (en nuestro caso, píxeles que conforman una polilla y píxeles que representan el fondo de la trampa), cuyas representaciones en el espacio de características ocupan distintas regiones, tal como se muestra en la figura 4.5.

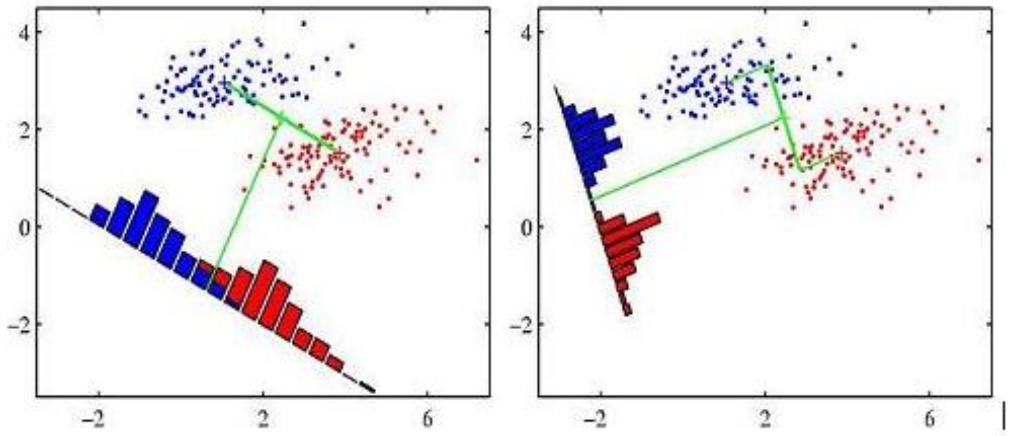


Figura 4.5: Ejemplo de un espacio de características en $\mathbb{R}^2$, de dos posibles vectores sobre los cuales proyectar. Tomada de `/www.projectrhea.org`

Resulta evidente que por más de que los puntos de cada clase estén notoriamente separados, la proyección sobre una recta cualquiera producirá una mezcla confusa de muestras. El objetivo del clasificador de Fisher consiste en encontrar la recta óptima sobre la cual proyectar las características.

Sea $x = (x_1, x_2, ..., x_n)$ una muestra del espacio de características, y w un vector en el mismo espacio, con $||w|| = 1$. La proyección de x en la dirección de w equivale a calcular:

$$y = w \cdot x \tag{4.3}$$

Un bloque pertenecerá a una clase en tanto su proyección se encuentre más cercana al conjunto de proyecciones de la misma. Siendo D una clase en el espacio de características, la media de las representaciones de los bloques pertenecientes a la clase se calcula como:

$$m_i = \frac{1}{n_i} \sum_{x \in D} x \tag{4.4}$$

La expresión para la diferencia entre las proyecciones de la media de cada clase es:

$$|\tilde{m_1} - \tilde{m_2}| = |w^t m_1 - w^t m_2| = |w^t (m_1 - m_2)| \tag{4.5}$$

Puede observarse que es posible aumentar dicha diferencia tanto como se quiera, escalando el vector w. Sin embargo el aumento en el módulo del vector de dirección también aumenta la desviación estándar de las medidas, por lo que la expresión anterior no resulta suficiente para realizar la diferenciación entre las proyecciones de las muestras.

La dispersión de las muestras proyectadas para una determinada familia se define según la siguiente expresión:

$$\tilde{s_i}^2 = \sum_{y \in Y_i} (y - \tilde{m_i})^2 \tag{4.6}$$

Observemos que $\tilde{\sigma}^2 = \frac{\tilde{s_1}^2 + \tilde{s_2}^2}{n}$ es un estimador de la varianza intraclase de los datos. Luego, la dirección w elegida es la que maximiza la función 4.7.

$$J(w) = \frac{|\tilde{m_1} - \tilde{m_2}|^2}{\tilde{s_1}^2 + \tilde{s_2}^2} = \frac{|\tilde{m_1} - \tilde{m_2}|^2}{\tilde{\sigma_1}^2.n_1 + \tilde{\sigma_2}^2.n_2} \tag{4.7}$$

Analizando la expresión 4.7 se observa que el numerador está compuesto por el cuadrado de la diferencia entre las proyecciones de las medias de ambas clases, mientras que el denominador se compone por la suma de los cuadrados de las varianzas de cada una de las clases, ponderadas de acuerdo al número de elementos que las componen. Al maximizar $J(w)$ estamos determinando que las proyecciones de las medias de cada clase estén lo más separadas posible, manteniendo las proyecciones de las muestras de cada clase tan juntas como sea posible. Desarrollando la ecuación 4.7 [75], se puede ver que el w que maximiza $J(w)$ es:

$$w = \Sigma^{-1}(\mu_1 - \mu_2) \tag{4.8}$$

Donde Σ es la matriz de covarianza y μ_i son las medias de las muestras que se obtienen experimentalmente. Una vez obtenido el vector de proyección se debe determinar el umbral contra el cual se comparará la proyección para determinar si un bloque pertenece a una clase u otra. Este umbral se determinará de forma experimental a partir de los datos obtenidos en el algoritmo de entrenamiento, tal como se exhibe en la sección 4.6.

4.4.2. Implementación del algoritmo

La técnica de detección se implementa de forma tal que corre en paralelo con la creación de la imagen con información reducida. El vector de características se determina a medida que

se decodifica el bloque. Una vez que se termina la decodificación del mismo, se proyecta según la dirección indicada por el algoritmo expuesto en la sección 4.4.1, para luego compararlo con un umbral. En el caso de que la medida de la proyección supere el umbral se almacenan la coordenadas (x, y) del bloque en dos vectores especialmente dispuestos para ello[4]. Como los vectores de coordenadas tienen un largo predefinido, la cantidad de bloques almacenables queda limitada de antemano.

4.5. Agrupamiento de bloques

4.5.1. Introducción

Una vez que el sistema determina las coordenadas de los bloques en los cuales es más probable que se encuentre contenida una polilla resulta interesante agruparlos de forma de que cada par (x, y) se corresponda con una polilla. Esta funcionalidad permite que el sistema obtenga un valor cuantificado de la cantidad de insectos en la trampa, pudiendo eventualmente tomar decisiones en función de este valor (e.g. enviar las imagen de la trampa sólo cuando haya un cambio significativo en el conteo de polillas).

Para lograr unir los bloques que corresponden a un mismo ejemplar se utiliza una técnica conocida como *labeling*. Dada una matriz donde cada elemento tiene dos estados posibles (en nuestro caso, 0 si el bloque esta vacío y 1 si contiene un objeto) la técnica de *labeling* asigna un grupo a cada bloque de la imagen, donde cada grupo corresponde a un objeto aislado.

El algoritmo clásico [76] recorre la matriz según el diagrama de flujo de la figura 4.6. Funciona recorriendo dos veces la imagen completa elemento a elemento. En la primera pasada realiza las siguientes tareas:

- En caso de que el elemento no pertenezca al fondo, verifica si tiene vecinos etiquetados.
- Si tiene vecinos etiquetados, le asigna la menor etiqueta de sus vecinos y en caso contrario le asigna la menor etiqueta disponible.
- Si tiene vecinos con etiquetas distintas, las almacena como equivalentes.

Luego de la primera iteración realiza una segunda cambiando las etiquetas equivalentes por la menor de ellas.

Esta implementación requiere un vector del mismo largo que el que contiene las coordenadas de los bloques, ya que cada par (x, y) debe estar acompañado de una etiqueta. Además se requiere un vector para almacenar las etiquetas equivalentes, lo que totaliza un uso de memoria RAM considerable.

Para evitar este consumo de memoria RAM se propone una implementación alternativa de la técnica, basada en las siguientes características de la aplicación:

[4]Hay algunas salvedades respecto al almacenamiento, como se verá en la sección 4.5.2.

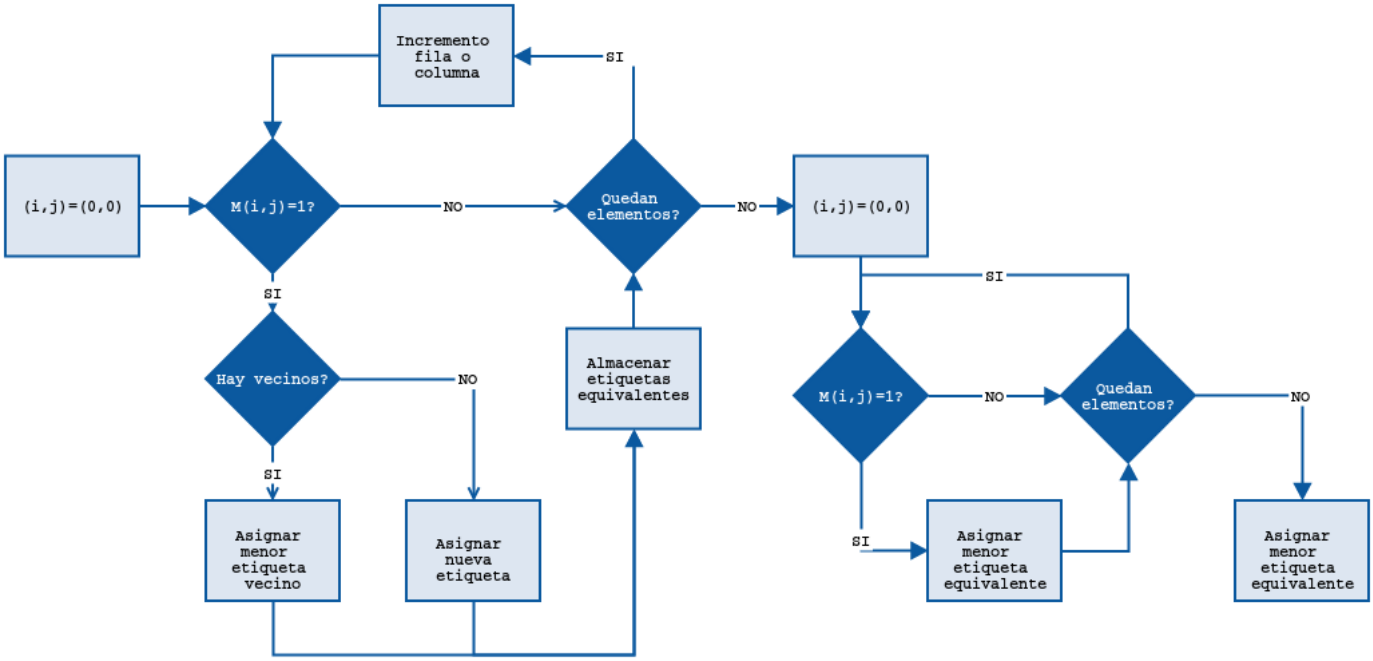


Figura 4.6: Diagrama de flujo del algoritmo de *labeling* clásico

- Los objetos fotografiados presentan cierta simetría respecto a las columnas, además de no presentar contornos demasiado irregulares. En nuestro caso esto se cumple porque los ejemplares capturados quedan representados por unos pocos bloques.
- Es más importante la precisión que la exactitud de las medidas, es decir que no importa el número exacto de polillas detectadas mientras no cambie considerablemente la situación de la trampa.
- Las coordenadas almacenadas se encuentran ordenadas en orden creciente de filas y dentro de cada fila en orden creciente de columnas

4.5.2. Algoritmo implementado

La modificación propuesta se basa en minimizar la cantidad de bloques almacenados en la etapa previa al *labeling*. Generalmente al aparecer un bloque cuya proyección del vector de características indica que contiene un fragmento de polilla, los bloques que lo siguen también arrojan un resultado positivo. En lugar de almacenar la serie de positivos se busca guardar únicamente el promedio entre el primero y el último, tal como se muestra en la figura 4.7. Esta preclasificación permite crear vectores de coordenadas mucho menores que los originales.

Para clasificar estos bloques se crea un vector de etiquetas inicializado en cero del mismo largo que los vectores de muestras y se recorren los vectores de coordenadas chequeando si la etiqueta correspondiente es cero. En caso de ser cero se asigna la etiqueta libre más baja y se recorre buscando un bloque en la siguiente fila que cumpla $x_0 - 1 < x < x_0 + 1$, donde x_0 es la columna del bloque original. Si se encuentra un bloque con dichas características, se le asigna la etiqueta del primer bloque y a continuación se procede de la misma forma con el nuevo bloque hallado. La figura 4.8 ilustra el procedimiento.

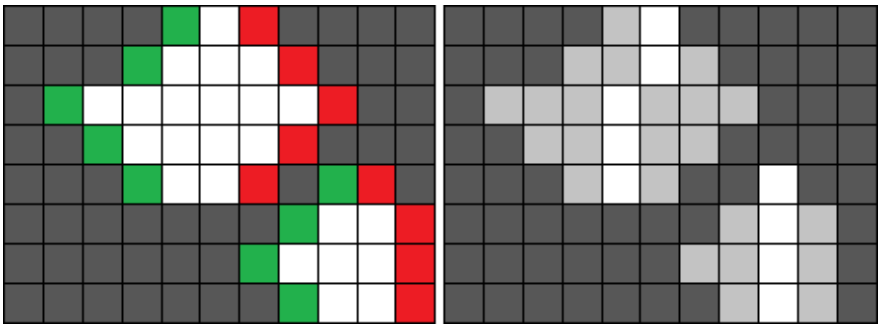


Figura 4.7: Explicación gráfica del método de almacenamiento de coordenadas implementado

Una vez que se tienen las regiones diferenciadas se promedian los bloques de cada región y se almacena el resultado como indicador de posición del objeto. Los resultados experimentales validan la técnica, como se verá en la sección 4.7.

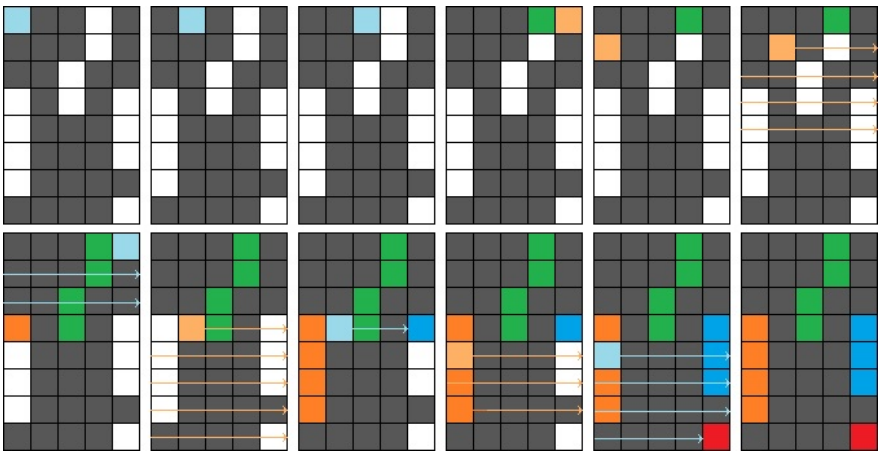


Figura 4.8: Explicación gráfica del método de *labeling*. El iterador celeste busca el comienzo de una región nueva, mientras que el rosado busca la continuidad una vez que se detecta una zona nueva

4.6. Entrenamiento del algoritmo

4.6.1. Introducción

En la práctica las medias de cada clase y sus varianzas no son conocidas, por lo que a priori no es posible determinar una dirección en el espacio de características sobre la cual proyectar los elementos que conforman una imagen. Para estimar esos valores se utiliza lo que se conoce como un set de entrenamiento, esto es un conjunto de imágenes en las cuales las instancias se han clasificado en forma manual (la pertenencia a una u otra clase es determinada) y en definitiva determinan las características de la clase. A partir de este set de entrenamiento se determina el vector sobre el cual se proyectan las características mediante las ecuaciones mostradas en la sección 4.4.1.

Una vez determinado el vector de proyección la pertenencia a una u otra clase depende únicamente del umbral y se pueden dividir las decisiones en cuatro categorías. Dada una categoría denominada *positiva*, y otra llamada *negativa* definimos como:

- Verdaderos Positivos (TP) a aquellos casos en los que la predicción define una instancia positiva como positiva.
- Falsos Negativos (FN) a aquellos casos en los que la predicción define una instancia positiva como negativa.
- Verdaderos Negativos (TN) a aquellos casos en los que la predicción define una instancia negativa como negativa.
- Falsos Positivos (FP) a aquellos casos en los que la predicción define una instancia negativa como positiva.

Con estas definiciones se genera la matriz de confusión [77], tal como se muestra en la figura 4.9.

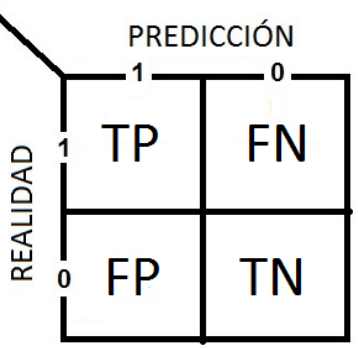


Figura 4.9: Matriz de confusión

Para definir la calidad de un algoritmo de detección se definen los siguientes valores:

- Precisión: $p = \dfrac{TP}{TP + FN}$
- Especificidad: $e = \dfrac{TN}{FP + TN}$

En la práctica se observa un *trade-off* entre la precisión y la especificidad. Este fenómeno se representa en un gráfico denominado curva ROC, que grafica p en función de $1 - e$ al variar el umbral de decisión.

Curva ROC

En la figura 4.10 pueden apreciarse cuatro curvas ROC correspondientes a cuatro algoritmos de decisión diferentes.

La curva A corresponde a un algoritmo de detección perfecto, en el cual es posible obtener una precisión igual a 1 (es decir, no hay falsos negativos), con una especificidad igual a 1 (no hay falsos positivos), es decir, ubicarnos en el punto (0,1) del gráfico. Una variación en el umbral permite bajar la especificidad manteniendo la precisión, pese a que este cambio nos aleje del caso ideal. La curva

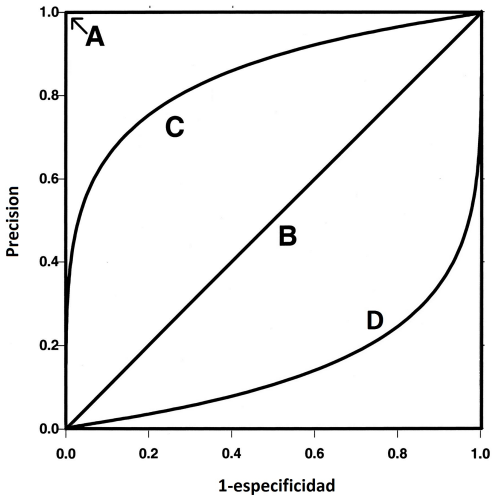


Figura 4.10: Curvas ROC para distintos algoritmos de detección.

B corresponde a un proceso de detección aleatorio, en el cual la decisión no tiene ninguna relación con la información aportada. Puede demostarse que si la pertenencia a la clase se decide con una probabilidad aleatoria de $0,5$ el algoritmo se sitúa en el punto $(0,5\ ,\ 0,5)$ del gráfico ROC. Si en su lugar se decide asignar la clase positiva aleatoriamente un $90\,\%$ de las veces, aumentará la precisión pero bajará la especificidad. Mejora la cantidad de verdaderos positivos pero empeora la relación de falsas alarmas.

La curva C de la figura 4.10 corresponde a un algoritmo que no es ideal pero es mejor que el caso aleatorio. Esta curva ejemplifica el intercambio que existe entre especificidad y precisión mencionado anteriormente. La decisión sobre cuál es el punto de decisión más conveniente depende de la aplicación, hay casos en los que es preferible una gran precisión contra una baja especificidad y viceversa, pero en general buscamos situarnos lo más cerca posible del punto óptimo (0,1).

Usualmente se considera que el área bajo la curva ROC (AUC) es un indicador de qué tan bueno es el algoritmo, ya que cuanto más cercana a 1 nos encontramos más cerca del caso ideal. En [78] se considera que un AUC mayor a $0,75$ corresponde a un algoritmo bueno. Otros puntos notables de este gráfico son:

- (0,0), corresponde a una precisión igual a cero (no hay verdaderos positivos) y una especificidad igual a uno (no hay falsos positivos). Es el caso en el que nunca se da una predicción positiva, independientemente de la información recibida.

- (1,1), que se corresponde a una precisión de uno (sin falsos negativos) con una especificidad de cero (sin verdaderos negativos). Es el caso en el que nunca se clasifica una instancia como negativa.

Por último la curva D corresponde a un algoritmo peor que la aleatoriedad. Sin embargo, un algoritmo de este caso puede convertirse en uno mejor simplemente invirtiendo la decisión, es decir asignando un negativo si la decisión es positiva y viceversa.

4.6.2. Método de entrenamiento

La figura 4.11 muestra los pasos necesarios para el entrenamiento del algoritmo.

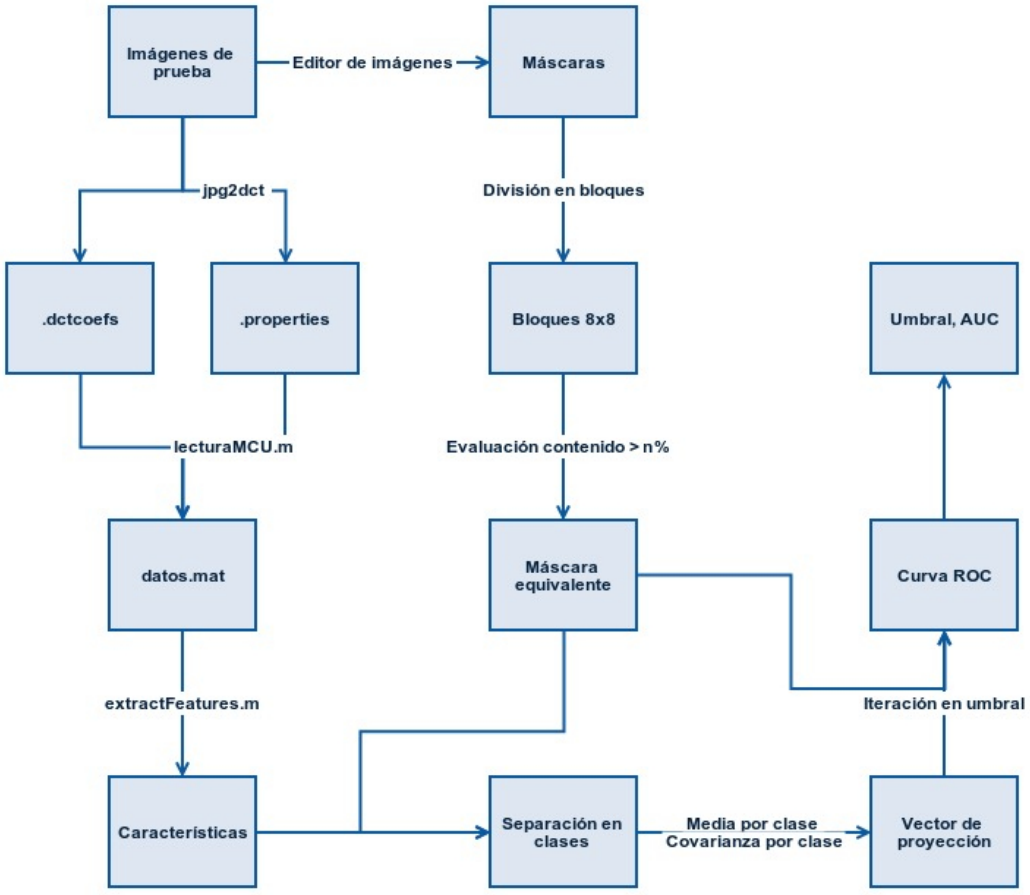


Figura 4.11: Diagrama de bloques del script de entrenamiento

El proceso comienza con un banco de imágenes de prueba, que serán las encargadas de definir las características de cada clase. Como se mencionó en la sección 4.6, es necesario contar con la clasificación manual de estas imágenes. Esta clasificación se expresa mediante una imagen complementaria (denominada máscara) que presenta color blanco para la clase *polilla* y color negro para la clase dual *fondo*, tal como se aprecia en las figuras 4.12, 4.13, 4.14 y 4.15.

Las imágenes del banco de prueba son procesadas con un ejecutable basado en el decodificador picoJPEG [70] que arroja dos archivos como salida:

- `*.properties` contiene las características del archivo JPEG decodificado (canales de color, submuestreo, tablas de cuantificación, etc.).
- `*.dctcoefs` contiene los coeficientes de la DCT de cada uno de los bloques.

A su vez estos dos archivos ingresan a un *script* de Matlab denominado `lecturaMCU.m`, que separa los canales de color en vectores distintos y expresa la información de un modo más conveniente, almacenándola en el archivo `datos.mat`. A continuación `datos.mat` es procesado por `extractfeatures.m`, un *script* dedicado a calcular las características que identifican un bloque.

Por otro lado las imágenes máscara se subdividen en bloques de 8×8 píxeles y se evalúa el contenido de cada bloque. Si el bloque de la máscara contiene más de determinado porcentaje de bloques blancos (*polillas*) se considera que el bloque debe clasificarse como positivo. Con esta información se genera una máscara equivalente, que en lugar de clasificar píxeles clasifica bloques. A partir de esto, la información contenida en `datos.mat` es clasificada en las clases *polilla* y *fondo*, para luego calcular las medias y las covarianzas por clase y con ellas obtener el umbral óptimo de proyección.

Una vez que contamos con el vector óptimo de proyección se clasifican las imágenes del banco de prueba con el umbral como parámetro y se traza la curva ROC. Observando la curva se elige un punto de funcionamiento y el programa devuelve el umbral y el AUC del algoritmo.

4.7. Validación de resultados

Los resultados se validaron usando dos colecciones de imágenes tomadas en condiciones de laboratorio. Uno de los sets (al que denominaremos de entrenamiento) se utiliza para obtener el vector de proyección óptimo, mientras que el otro (denominado de validación) se utiliza para seleccionar el umbral óptimo y evaluar la eficacia del método.

Debido a la escasa disponibilidad de ejemplares de polillas *Cydia Pomonela* y *Cydia Molesta* se hizo necesario utilizar algunas imágenes con objetos que simulen las mismas. En particular se utilizaron secciones de cable negro de 3 *mm* de grosor para simular la presencia de polillas y se incluyeron secciones de alambre de cobre de un grosor un poco menor para simular la presencia de objetos extraños. El set de entrenamiento se compone de:

- Dieciocho imágenes de objetos que simulan polillas. Quince de ellas se tomaron en condiciones de iluminación normales (sin luz exterior, con la iluminación LED encendida) y tres en condiciones anormales (con iluminación exterior).
- Cuatro imágenes que muestran ejemplares reales.

El set de validación se compone de:

- Cuatro imágenes de objetos que simulan polillas. Tres de ellas se tomaron en condiciones de iluminación normales y una en condiciones anormales.
- Dos imágenes que muestran ejemplares reales.

Si bien el número de imágenes puede parecer insuficiente, es necesario recordar que el algoritmo trabaja sobre cada bloque del archivo JPEG, por lo que en la práctica tenemos 105.600 elementos en el set de entrenamiento y 28.800 en el de validación. Las figuras 4.12, 4.13, 4.14 y 4.15 muestran el tipo de fotografías utilizadas y sus respectivas máscaras.

Se decidió realizar un entrenamiento con las fotos con objetos que simulan polillas y otro con las imágenes con los insectos reales, validando cada uno de los resultados con la parte correspondiente del algoritmo de validación. En el anexo A se muestran todas las imágenes de los sets de entrenamiento y validación, así como los resultados de la misma.

Figura 4.12: Imagen con captura de objetos que simulan polillas

Figura 4.13: Máscara correspondiente a 4.12

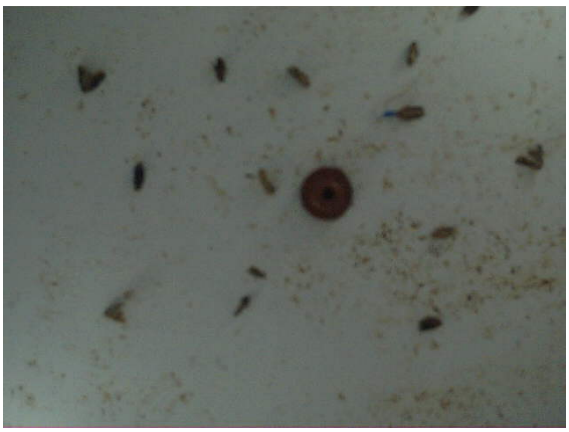


Figura 4.14: Imagen con captura de polillas

Figura 4.15: Máscara correspondiente a 4.14

Las curvas ROC para cada uno de los experimentos se muestran en las figuras 4.16 y 4.17. El AUC obtenida se encuentra entre $0,85$ y $0,90$ en ambos casos, por lo que podemos concluir que el algoritmo se desempeña muy bien. En las figuras 4.18 y 4.19 se muestran dos imágenes con cruces blancas sobre las polillas detectadas. Es importante usar un punto de funcionamiento que garantice pocos falsos positivos aunque sea a costa de un número mayor de falsos negativos, ya que cada polilla está compuesta por varios bloques. Dado que en las imágenes utilizadas cada polilla ocupa varios bloques y suponiendo que el clasificador detecta al menos un bloque de cada ejemplar, el sistema es más robusto frente a falsos negativos que a falsos positivos. Un falso negativo puede ser compensado por el algoritmo de *labeling* y lograr que el conteo sea correcto, sin embargo es altamente probable que un falso positivo genere una nueva polilla y altere el conteo. Cabe desatacar que los resultados obtenidos se encuentran muy relacionados con la calidad de las máscaras utilizadas, por lo que una mejora en las mismas repercutirá positivamente en la calidad del entrenamiento.

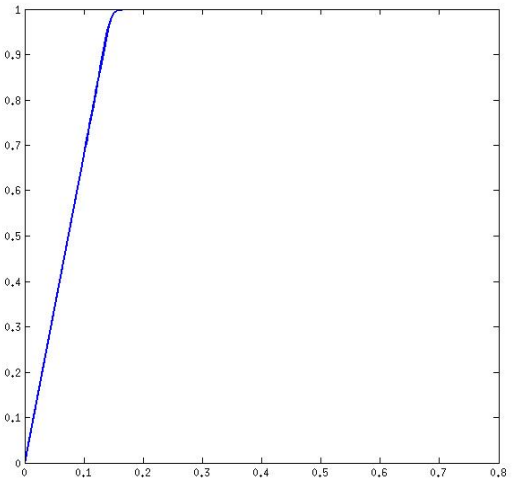


Figura 4.16: Curva ROC correspondiente a las imágenes con objetos que simulan polillas. AUC=0,90

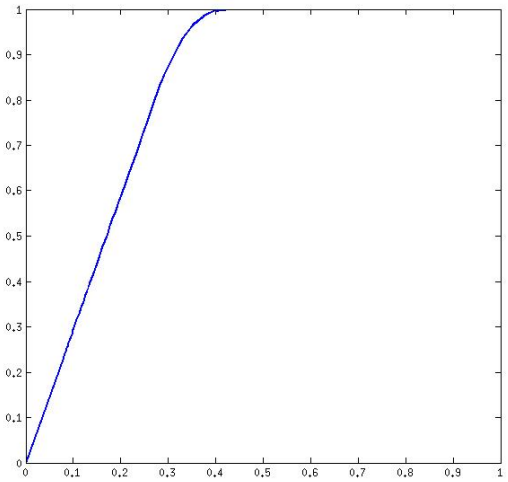


Figura 4.17: Curva ROC correspondiente a las imágenes de polillas. AUC=0,86

En las figuras 4.18 y 4.19 puede observarse cómo el algoritmo es capaz de discriminar las polillas con éxito.

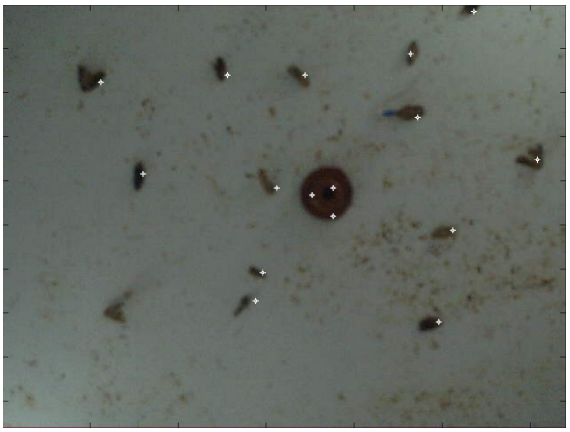


Figura 4.18: Imagen del banco de validación con los bloques detectados sobreimpresos

Figura 4.19: Imagen del banco de validación con los bloques detectados sobreimpresos

4.7.1. Conclusiones parciales

Los resultados experimentales muestran la conveniencia de aplicar una técnica de discriminación como FLD en sistemas distribuidos con recursos de hardware limitados, ya que les brinda la posibilidad de tomar decisiones autónomas a partir de la información contenida en las imágenes. La aplicación de una técnica de procesamiento de imágenes en WSN permite:

- Ahorrar energía, ya que no resulta necesario enviar la imagen sino solamente el recuento de polillas y a lo sumo sus coordenadas, que entran en un par de paquetes de radio. El envío de

imágenes JPEG representa el 7,3 % (ver sección 6.3) del consumo de energía de las baterías. En el caso que haya una rama de la red con 7 saltos, el nodo más cercano al *sink* deberá transmitir su imagen y recibir y transmitir las 6 imágenes de los nodos hijos. Esto implica que la energía consumida por la actividad de la radio se incremente considerablemente, acortando sustancialmente su autonomía energética. Al utilizar el algoritmo de detección, se emplea el 1,2 % de la energía total, pero las transmisiones pasan a representar el 0,1 %, por lo que la autonomía de los nodos más cercanos al *sink* no se ve tan afectada.

- Utilizar protocolos de red convencionales en WSN: Como se desarrolla en el capítulo 5, para la transmisión de imágenes existen problemas inherentes para las redes basadas en el estándar IEEE 802.15.4. Sin embargo la detección distribuida permite enviar el conteo en pocos paquetes, siendo innecesaria la transmisión de la imagen. Esto hace que se puedan utilizar distintas aplicaciones ya implementadas en Contiki OS para recolección de pequeñas cantidades de datos, sin la necesidad de hacer grandes modificaciones.

Sobre las cuatro imágenes con objetos que simulan polillas con las que se ensayó el algoritmo, tres dieron un conteo exactamente igual al esperado y una obtuvo una diferencia de un ejemplar. En las dos imágenes con polillas reales con las que se ensayó, se contaron correctamente 13 de las 14 polillas y se produjeron falsos positivos debido a la pastilla de feromonas que se encuentra en el centro de la trampa. Entrenando el algoritmo con una mayor cantidad de imágenes de polillas reales, se pueden obtener mejores resultados. En caso que se detecte un error constante debido a la pastilla de feromonas, el mismo se puede restar al total de la cuenta o modificar el algoritmo de *labeling* para que trate de manera diferencial a aquellos grupos que superen un número determinado de bloques positivos.

El enfoque de este proyecto busca dar el puntapié inicial al análisis de imágenes embebido en nodos de WSN. Si bien solamente se utilizó el clasificador lineal Fisher, con una herramienta como el decodificador JPEG implementado (capaz de funcionar en menos de 1kB de RAM), es posible experimentar con otros algoritmos de detección en el espacio frecuencial más sofisticados. Con estos se podrían obtener mejores resultados en situaciones donde la diferencia entre las clases fuera más difusa. Estos algoritmos comparten la característica de ser entrenados en un PC, donde se realiza el cómputo más pesado.

Capítulo 5

Análisis y diseño de la red

Resumen

En el presente capítulo se realiza un análisis de los distintos protocolos de red implementados en Contiki OS que se ajustan a las necesidades funcionales del sistema. Se analiza la capa física, entramado y protocolo de acceso al medio CSMA/CA definidos en el estándar IEEE 802.15.4. Se justifica la elección de nullRDC como protocolo de ciclo de trabajo de la radio (RDC), y el uso de las primitivas definidas en librería Rime del sistema operativo como protocolo de capa de red, mediante simulaciones en Cooja y medidas de laboratorio.

Contenido

5.1. Introducción

Los modelos de interconexión de sistemas de comunicación (como son las WSN) están basados en capas, que corresponden a distintos niveles de abstracción y las cuales simplifican el diseño e implementación de las mismas. Cada capa realiza un conjunto de funciones bien definidas que ofrece como servicio a las capas superiores.

Las funciones de las capas son implementadas por las entidades de la capa, es decir el hardware y el software que realiza la función. La especificación formal de un servicio (cómo se usa, qué información es necesario proveer para que pueda actuar, dónde se accede al servicio, etc.) está dada por las primitivas de servicio [1]. A su vez, son los protocolos los que definen las reglas mediante las cuales dos entidades de la misma capa interactúan entre sí y definen los requisitos de las primitivas de la capa.

Para la correcta elección del protocolo que mejor se adapte a una aplicación es necesario evaluar la topología de la red, esto es el tipo de nodos (de funcionalidad completa, de funcionalidad limitada o *sink*) que conformarán la red y cómo se interconectan entre ellos, la calidad de servicio que se necesita para la aplicación, su eficiencia energética, robustez y escalabilidad. En el contexto de este proyecto se define la eficiencia energética como la energía necesaria para realizar una trasmisión exitosa.

En este capítulo nos concentramos en explorar protocolos a nivel de capa de enlace y capa de red, donde pretendemos utilizar aquellos que sean orientados a conexión y confiables. Por orientado a conexión se entiende que las entidades intercambian información referente al inicio y fin de la transmisión y por confiable que el protocolo garantiza que los datos serán pasados a la aplicación sin pérdidas, errores ni duplicados, y en el orden correcto. La necesidad de utilizar un protocolo confiable y orientado a conexión está dada por el hecho que se desean trasmitir imágenes desde distintos nodos hacia el *sink* y la pérdida de datos no es admisible.

Los protocolos de red implementados en Contiki OS están pensados para la transmisión de pequeñas cantidades de datos (algunos bytes) de sensores como ser de temperatura o humedad. Esto hace que la transmisión de imágenes con un tamaño aproximado de 18 kB sea todo un desafío a la hora de planificar la red. Es por eso que se hizo un estudio extensivo de todos los protocolos disponibles para tratar de encontrar el óptimo.

Se planteó una idea fuera de lo convencional en WSN que consistió en modificar el RDC, ya que se cuestionó el beneficio de utilizar un protocolo de ciclo de trabajo pequeño cuando se va a transmitir una gran cantidad de paquetes (180 aproximadamente en nuestro caso). En los protocolos que mantienen la radio la mayor parte del tiempo apagada, en la etapa inactiva del RDC, el MCU apaga el *chip* de la radio pero mantiene el oscilador de cristal y el regulador de voltaje de la misma encendidos. Esto hace que de todas formas exista una base de consumo de corriente significativa incluso cuando no se está escuchando el canal.

En función de esto, se propuso hacer la transmisión de archivos con el mayor *throughput* posible para luego apagar completamente, no solo el chip de la radio, sino también su oscilador de cristal y su regulador de voltaje.

Dado que existen varios protocolos para el manejo del ciclo de trabajo en Contiki OS, entre

[1] Son las funciones más elementales de cada capa.

ellos X-MAC [79], ContikiMAC y nullRDC, se analizaron las ventajas y desventajas de cada uno, así como el más apropiado para la aplicación.

5.2. Protocolos

Para la transmisión de datos por la radio CC2420 se usó la *stack* de protocolos que se muestra en la figura 5.1, correspondiente al *stack* de comunicaciones implementado en Contiki OS (`NETSTACK`)

Figura 5.1: *Stack* de protocolos

Las capas física y de acceso al medio cumplen el estándar 802.15.4 como se detalla en las secciones 5.2.1 y 5.2.2 respectivamente. Para el manejo del ciclo de trabajo de la radio hay varios protocolos implementados en Contiki OS que se describen en la sección 5.2.3. El entramado se hace siguiendo el estándar 802.15.4 como se detalla en la sección 5.2.4. Finalmente, para el manejo de red se utilizan las primitivas de Rime que se explican en detalle en la sección 5.2.5.

5.2.1. Capa física

La capa física definida en el estándar IEEE 802.15.4 utiliza la banda que comprende las frecuencias entre $2,4000$ y $2,4835$ GHz, permite velocidades de transmisión de hasta 250 $kbps$ utilizando DSSS y utiliza modulación O-QPSK.

DSSS es una técnica de modulación utilizada para la transmisión de señales digitales, en donde la señal transmitida ocupa un ancho de banda mayor al que ocupa la información de la señal que modula la portadora. Los beneficios de usar esta técnica son: mayor inmunidad a interferencias (aumenta redundancia), optimización del uso de los canales, y mayor seguridad.

Esta técnica se logra multiplicando los datos a transmitir por una señal pseudo-aleatoria [2], la

[2]Es una secuencia que no muestra ningún patrón o regularidad aparente desde un punto de vista estadístico, a pesar de haber sido generadas por un algoritmo completamente determinista, en el que las mismas condiciones iniciales producen siempre el mismo resultado.

cual es una secuencia de 1 y -1 a una frecuencia mucho mayor que la señal original. Los valores de dicha secuencia pseudo-aleatoria se llaman chips, y cada chip tiene una duración mucho menor que un bit de información. Cada bit de información está modulado por una secuencia de chips con una tasa mucho mayor a la de bits.

La información se puede recuperar en el receptor multiplicando por la misma secuencia pseudo-aleatoria(la misma debe ser conocida por ambas partes), y se logra reconstruir exactamente la señal original.

En la banda de 2.4 GHz se utiliza una modulación O-QPSK, 16-aria, cuasi-ortogonal. La modulación 16-aria cuasi-ortogonal es una técnica de modulación de datos que utiliza una secuencia pseudo-aleatoria de 32-chip para representar cuatro bits y simultáneamente ampliar el espectro. La modulación se logra haciendo una rotación cíclica y conjugando algunos chips de la misma secuencia como se muestra en la figura 5.2.

Input bits (b_0 b_1 b_2 b_3)	Chip values (c_0 c_1 ... c_{31})
0000	1101 1001 1100 0011 0101 0010 0010 1110
1000	1110 1101 1001 1100 0011 0101 0010 0010
0100	0010 1110 1101 1001 1100 0011 0101 0010
1100	0010 0010 1110 1101 1001 1100 0011 0101

Figura 5.2: Tabla con algunos ejemplos de conversión de bits a chips

La secuencia pseudo-aleatoria comienza en distintos lugares, dependiendo de los datos a modular, transmitiendo 4 bits en cada período de símbolo. Los símbolos 0-7 son permutaciones cíclicas de a cuatro chips de la secuencia, y los símbolos 8-15 son las mismas permutaciones pero usando la secuencia conjugada, es decir, invirtiendo los chips impares.

El estándar especifica una tasa de símbolos de 62.500 símbolos por segundo, con cuatro bits por símbolo, por lo que se obtienen tasas de transmisión de 250 $kbps$.

En una modulación QPSK, la fase de la señal puede cambiar 180 grados de un símbolo a otro. Las señales luego de moduladas típicamente pasan por un pasabajos, generalmente un filtro apareado. Esto hace que grandes cambios de fase resulten en grandes fluctuaciones de la amplitud.

Para evitar este problema, en el canal Q se agrega un retardo de medio chip, para crear el offset de la modulación. Esto hace que no haya saltos de dos bits, es decir que la constelación no pase por cero como se muestra en la figura 5.3.

La transmisión es de 32 chips complejos en un tiempo de símbolo de 16 μs, entonces la tasa de chips es dos millones de chips por segundo (un millón en cada canal).

Por lo tanto el procesamiento consiste en ensamblar 4 bits en un símbolo, luego convertir ese símbolo a una secuencia de 32 chips rotados e invertidos cíclicamente, para luego modularlos en los canales I y Q como se muestra en la figura 5.4.

La potencia de transmisión de la radio debe ser mayor a -3dBm, y el límite superior difiere en cada país, según las regulaciones. En Uruguay la potencia máxima de transmisión es de 4 W [80].

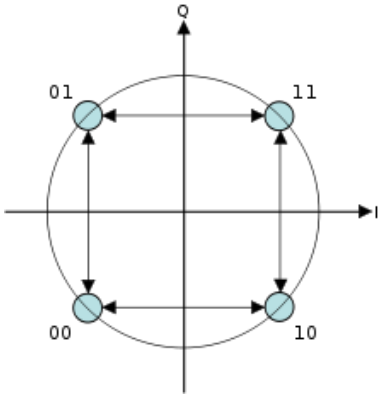


Figura 5.3: Efecto en la constelación de usar O-QPSK

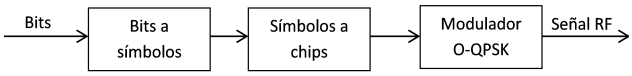


Figura 5.4: Diagrama de bloques de comunicación

5.2.2. Capa de acceso al medio

Para la capa de acceso al medio, el estándar IEEE 802.15.4 establece el uso del protocolo CSMA/CA no persistente.

El protocolo CSMA tiene como función arbitrar el acceso al medio en canales compartidos y basa su funcionamiento en la capacidad de los nodos de detectar cuándo se está transmitiendo y actuar acorde a ello, de modo de bajar la probabilidad de colisiones.

Aún cuando se sense el medio antes de intentar transmitir, es posible que ocurran colisiones ya que un nodo puede ver el canal libre más o menos simultáneamente con otro y ambos pueden decidir comenzar a transmitir en función de los retardos inherentes al canal. Cuando ocurren colisiones, las señales eléctricas correspondientes a ambas tramas se suman en el canal, corrompiéndose los datos enviados por ambos nodos.

En la modalidad no persistente, cada vez que un nodo desea transmitir una trama, se escucha el canal (detección de portadora). Si el canal está libre, comienza la transmisión, y si está ocupado, en lugar de escuchar el canal hasta que el nodo que estaba haciendo uso del canal lo libere y comenzar a transmitir de inmediato (modo persistente), espera un tiempo aleatorio y vuelve a sensar el canal, siguiendo el mismo comportamiento luego. De esta manera se obtiene una mejor utilización del canal que en el caso persistente a costa de mayores retardos.

La modalidad CA agrega que cada nodo anuncia su intención de transmitir antes de hacerlo para evitar colisiones entre los paquetes de datos. De esta forma, el resto de los nodos de la red sabrán cuando hay colisiones y en lugar de transmitir la trama en cuanto el medio está libre, esperan un tiempo aleatorio adicional y si tras ese intervalo el medio sigue libre, proceden a la transmisión reduciendo la probabilidad de colisiones en el canal.

5.2.3. Ciclo de trabajo de la radio

Conceptos básicos

En Contiki OS se separa el manejo del ciclo de trabajo de la radio de la capa de acceso al medio en una nueva capa llamada RDC. El RDC es el cociente entre el tiempo que la radio se encuentra activa y el período total de la misma. Se detallan a continuación conceptos básicos que servirán para entender los distintos protocolos de RDC.

Escucha de bajo consumo

La escucha de bajo consumo tiene su base en el encendido periódico de la radio para muestrear el canal de forma de evitar colisiones. Utilizando el RSSI de la radio, que da una estimación de la potencia presente en la señal recibida, se logra discernir si existe actividad de radio en el canal y en caso positivo se mantiene encendida la radio para la recepción o transmisión de datos por un cierto período predefinido.

Prueba de bajo consumo

Al igual que la escucha de bajo consumo, se enciende y apaga la radio periódicamente. Difiere en el hecho de que el receptor envía un paquete de prueba para que el transmisor sea notificado de que el receptor está listo para recibir.

Enganche de fase

El mecanismo de enganche de fase (*phase-lock*) busca lograr un sincronismo entre el RDC del transmisor y el del receptor. Para esto se implementa en el receptor la transmisión de ACKs, lo que le permite al transmisor llevar un registro del tiempo en el que el receptor recibió la trama enviada. Asumiendo que los intervalos de *wake-up*[3] del receptor son conocidos, el transmisor se puede enganchar con la fase del receptor, permitiéndole transmitir sólo cuándo el receptor esté despierto.

Protocolos de capa RDC

Contiki OS tiene implementados cuatro protocolos de RDC que son nullRDC, ContikiMAC, X-MAC y Contiki LPP. Estos fueron estudiados salvo el último por tener problemas de ensamblado de las tramas en la versión 2.6 de Contiki OS. En esta sección se detallarán los fundamentos básicos de cada uno de ellos.

[3]Encendido de la radio

nullRDC

nullRDC, como lo indica su nombre, no implementa manejo del ciclo de trabajo de la radio. Es un protocolo que mantiene la radio encendida para transmitir o recibir durante toda la transmisión. Tiene como desventaja el alto consumo de corriente pero a su vez es el que obtiene mayor throughput y menores retardos.

ContikiMAC

ContikiMAC es el protocolo de RDC por defecto en Contiki OS. Propone una solución basada en ideas ya utilizadas por otros mecanismos e introduce algunas mejoras. De B-MAC [81], X-MAC y BoX-MAC [82] toma la idea de *wake-ups* periódicos (escucha de bajo consumo), de WiseMAC [83] toma el concepto del enganche de fase y de BoX-MAC el mecanismo de enviar múltiples copias de los datos como *strobe*[4] de encendido para el receptor.

El mecanismo de *wake-up* de la radio se implementa mediante el procedimiento llamado CCA, el cual utiliza el RSSI de la radio para detectar actividad en el canal. En cada *wake-up* de la radio, se realizan dos CCA de manera que si hay actividad en el canal, alguno de los dos la detecte.

En la radio CC2420 la duración de cada CCA es de $t_r = 192\ \mu s$. El tiempo t_c entre dos CCA está configurado para ser mayor que el intervalo t_i entre la transmisión de dos paquetes para que alguno de los dos CCA lo detecte. Esto se observa en la ecuación 5.1 y se aclara en el diagrama de la figura 5.5.

A su vez los tiempos t_r que demora cada CCA sumados, mas el tiempo entre ellos debe ser menor a t_s, tiempo mínimo de un paquete, para que no pueda entrar un paquete entre ambos CCA. Esto se muestra en la ecuación 5.3 y en el diagrama de la figura 5.6.

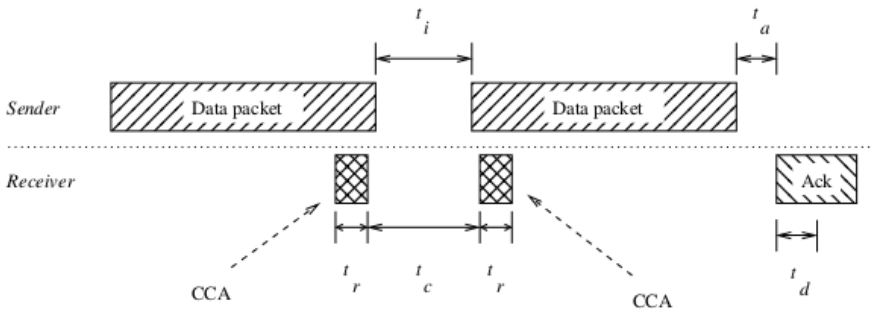


Figura 5.5: Diagrama que ejemplifica las ecuaciones 5.1 y 5.2

Finalmente el tiempo t_i debe ser mayor que el tiempo t_a entre la recepción de un paquete y el envío del ACK y el tiempo t_d en que el transmisor detecta el ACK como se muestra en la ecuación 5.3. Esto se hace para que los CCA puedan detectar el ACK que envía el receptor. El diagrama de la figura 5.5 ejemplifica esta situación.

$$t_i < t_c \tag{5.1}$$

[4]Paquete de señalización utilizado para sincronizar una comunicación

$$t_a + t_d < t_i \tag{5.2}$$

$$2t_r + t_c < t_s \tag{5.3}$$

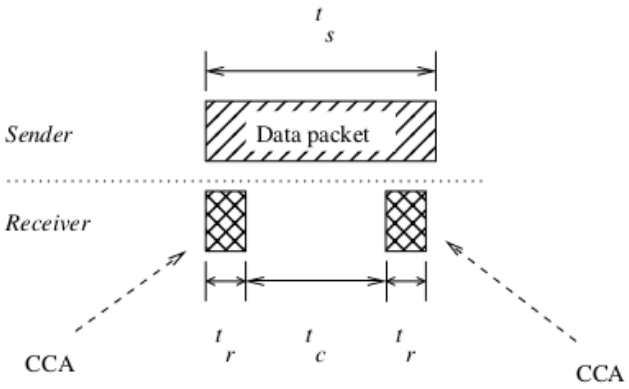


Figura 5.6: Diagrama que ejemplifica la ecuación 5.3.

Por especificaciones del estándar IEEE 802.15.4, se establece $t_a = 192\mu s$ y $t_d = 160\mu s$. A partir de ésto, los desarrolladores de Contiki OS fijaron el resto de los tiempos de forma de cumplir las especificaciones mencionadas en las ecuaciones 5.1, 5.2 y 5.3.

- $t_i = 0,4\ ms$
- $t_c = 0,5\ ms$
- $t_s = 0,884\ ms$

El proceso de detección que se ejemplifica en la figura 5.7 consiste en realizar un CCA. Si se detecta actividad en el canal, se mantiene la radio encendida por una ventana de recepción de tamaño $2t_l + t_i$ siendo t_l el tiempo de un paquete de tamaño máximo. De esta manera se evita el problema que la detección se haya realizado en el medio de un paquete. Si no se detectó actividad, se realiza un segundo CCA un tiempo t_c después. En caso de detectarse actividad, se hace lo mismo que la primera vez, y si no se detecta ninguna transmisión, se apaga el *chip* de la radio. En un período configurable (por defecto $T = 125ms$) se vuelve al principio.

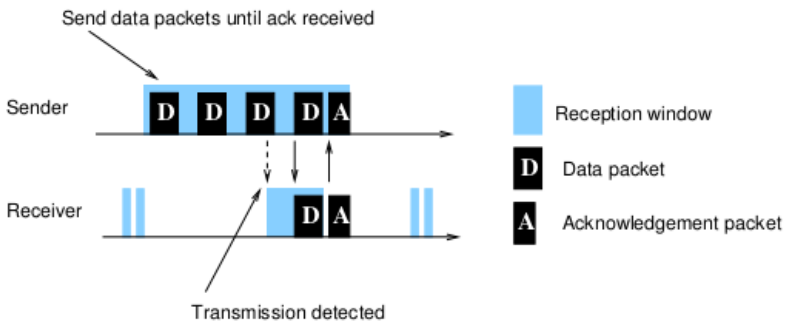


Figura 5.7: Transmisión unicast

Para la transmisión, en caso de que el transmisor no conozca la fase del receptor, se envía un tren de paquetes de datos para forzar que en algún momento el receptor escuche. Cuando se recibe un paquete, el receptor envía un ACK. Cuando el transmisor lo recibe, deja de enviar el paquete. En el caso de que se tenga una optimización de enganche de fase, se logran menos transmisiones ya que el transmisor comenzará de manera casi sincronizada sus envíos.

El procedimiento de *broadcast* difiere del caso de *unicast* en que no se tienen ACK de capa de enlace en las transmisiones por lo que, quien transmite datos por *broadcast*, envía la cantidad de paquetes que entren en una ventana de recepción.

X-MAC

X-MAC fue el primer protocolo de escucha de bajo consumo de corriente para WSN y fue desarrollado para radios que transmiten los datos en paquetes, como la CC2420 utilizada en este proyecto. Es el primer protocolo que incorpora *strobes* de escucha. Los *strobes* de escucha son paquetes que en el caso de X-MAC contienen la dirección del receptor, lo que permite que los vecinos a los cuales no está destinado el paquete mantengan sus radios en modo *sleep* cuando escuchan un *strobe* que no es para ellos. X-MAC también agrega un *strobe* de reconocimiento, que es mandado por el receptor en respuesta del *strobe* de escucha que tiene su dirección. Cuando el transmisor escucha dicho *strobe*, inmediatamente manda el paquete de datos.

Contiki OS utiliza una versión del protocolo X-MAC llamada CX-MAC que se basa en el protocolo descrito en el párrafo anterior, pero añadiendo soporte para tráfico *broadcast*, sincronización y retransmisiones a nivel de capa MAC. Su funcionamiento en recepción es el siguiente: enciende la radio y escucha si hay algún paquete de *strobe*. Si no recibe ninguno, apaga la radio por el resto del período del ciclo de trabajo. Si por el contrario esucha un *strobe*, manda un *strobe* de reconocimiento al transmisor. Cuando el transmisor lo recibe, inmediatamente le manda el paquete.

Si se desea enviar un paquete *broadcast*, se transmite directamente el paquete muchas veces sin enviar paquetes de *strobe*. Esto tiene como ventaja que los receptores no tienen que despertar la radio por un tiempo mayor al tiempo de chequeo del canal.

Para la sincronización, cada vez que un nodo le envía un paquete a un nodo vecino, registra el tiempo en el cual el receptor se despertó, ya que este último envía un *strobe* de reconocimiento al recibir el *strobe* de escucha. En función de éste tiempo puede calcular cuándo se va a volver a despertar el receptor, lo que le permite sincronizarse.

Cada vez que un nodo quiere enviar un paquete a un vecino con el cual está sincronizado, no manda los *strobes* de escucha hasta que el receptor se despierte. Si por algún error se equivoca y no envía el *strobe* cuando la radio del receptor está prendida, sigue enviando los *strobes* durante todo el período.

Esto garantiza la transmisión por más que la sincronización sea errónea, pero si la sincronización es correcta, la eficiencia del protocolo aumenta considerablemente.

Las retransmisiones a nivel de capa MAC son un mecanismo mediante el cual las capas superiores pueden establecer una bandera indicando que un paquete tiene que recibir reconocimiento. Cuando se hace ésto, X-MAC no apaga la radio después de transmitir el paquete sino que mantiene la radio prendida esperando el paquete de reconocimiento, el cual se puede enviar sin usar ningún *strobe*.

5.2.4. Entramado

El estándar IEEE 802.15.4 define cuatro tipos de tramas que son de *Beacon*, de *Datos*, de *Reconocimiento de paquetes (ACK)* y de *Comandos MAC*. Las tramas de *Beacon* son utilizadas como balizas para señalizar o establecer la configuración de red, las de *Datos* se utilizan para transmitir la información relevante o carga útil, la de *Reconocimiento de paquetes* se utiliza para confirmar las recepciones exitosas y por último las de *Comandos MAC* se utilizan para el control entre entidades pares de la capa de acceso al medio.

Todas están estructuradas de forma similar, diferenciándose en su propósito y en su carga útil. Están compuestas por un encabezado de sincronización (SHR por sus siglas en inglés), un encabezado de capa física (PHR) y una unidad de datos de capa física (PSDU). La PSDU a su vez está compuesta por un encabezado y pie MAC (MHR y MFR respectivamente) y por la carga útil de la capa MAC (MSDU). La estructura explicada anteriormente se puede ver en la figura 5.8.

Figura 5.8: Estructura de las tramas

5.2.5. Capa de red

Para la transmisión de grandes cantidades de datos sin pérdidas, errores ni duplicados y en el orden correcto, es necesario un protocolo confiable y orientado a conexión. En el *stack* TCP/IP esto es provisto en capa de transporte por el protocolo TCP donde se negocia el establecimiento y fin de la conexión y mediante distintos mecanismos se garantiza que los segmentos se pasarán a la capa de aplicación sin errores ni duplicados y en el orden correcto.

En el caso de las WSN se necesitan protocolos que ocupen menos espacio en memoria y requieran menor procesamiento, por lo que puede ser útil tener un protocolo confiable, que garantice la llegada de todos los paquetes y en el orden correcto. Para poder tener un protocolo de ventana deslizante como el que utiliza TCP/IP se necesitaría tener un *buffer* de tamaño considerable en RAM dedicado a la transmisión y energía suficiente para realizar el procesamiento y las retransmisiones. En ese caso, una gran cantidad de aplicaciones quedan restringidas a utilizar un protocolo que envíe de a un paquete a la espera del reconocimiento por parte del receptor (protocolos *stop and wait*).

Rime proporciona una serie de primitivas para que el desarrollador forme su propio *stack* de protocolos de red. En la figura 5.9 se puede observar cómo está estructurado el *stack* de protocolos a partir de las primitivas Rime en nuestra aplicación. Cada primitiva utiliza la primitiva inferior para

lograr su cometido. Como se ha mencionado, en nuestro caso precisamos una comunicación *unicast* (punto a punto), donde se envíen los datos de manera confiable. Esto es posible en Rime gracias a la primitiva `Runicast`, la cual se basa en una red *unicast*.

Figura 5.9: *Stack* de primitivas de Rime

`Runicast` envía cada paquete reiteradamente mediante la primitiva `Stunicast` (*stubborn unicast*), la cual envía varias veces el mismo paquete hasta que un protocolo de otra capa cancele la transmisión. `Runicast` proporciona un mecanismo para cortar las retransmisiones de `Stunicast` mediante un *callback* al recibir un paquete ACK o al alcanzarse el máximo de retransmisiones definido por el usuario y agrega un número de secuencia para evitar duplicados. El control de duplicados no está implementado en el propio protocolo por lo que fue incluido por nuestra parte, obteniéndose un protocolo de red confiable que garantiza que se envíen todos los datos de manera ordenada, sin errores y sin duplicados. Los duplicados podrían ocurrir, si no hubiera control de secuencia, en el caso en que el receptor envíe el ACK pero el transmisor no lo reciba y cuando este último retransmita el mismo paquete el receptor lo tome como un paquete nuevo y distinto al anterior.

`Stunicast` logra las transmisiones mediante `Unicast`, la cual utiliza la estructura de un *broadcast* con transmisor identificado (*identified sender broadcast*). Esta última utiliza la primitiva más básica que corresponde a un *broadcast* anónimo llamada `ABC`, transmitiendo un paquete sin encabezados de capa de red.

El estándar 802.15.4 define el tamaño máximo de paquete en 128 bytes. Teniendo en cuenta los encabezados de la subcapa MAC y de Rime, la carga útil que se puede transmitir es menor. Como no se dispone de datos exactos del tamaño de estos encabezados, se eligió en base a pruebas un tamaño de carga útil de 100 bytes.

Implementación de la red

Basándonos en la primitiva `Runicast`, implementamos una aplicación propia. En transmisión, primero se envía un paquete de 4 bytes con el tamaño total de la imagen con una bandera de comienzo de 2 bytes para reconocer el paquete. Luego se lee la imagen de la memoria flash en fragmentos de 100 bytes y se envía al receptor usando `Runicast`. Cuando se envían otros datos

que no son de la imagen, se agrega un marcador de fin para indicar que finalizaron los datos (en la imagen no es necesario ya que el archivo JPEG culmina siempre con el marcador `0xFF 0xD9`). Si bien este marcador se podría obviar, se implementó pensando en simplificar el despliegue de los datos en Matlab, como se detalla en la sección 3.4.1.

En recepción se implementa un control de secuencia utilizando los números de secuencia que agrega `Runicast` para evitar aceptar paquetes duplicados. Cuando se identifica el paquete de tamaño de imagen, se guarda el mismo en una variable y luego se incrementa un contador para controlar el fin de la imagen con cada paquete de datos recibido. Los datos se imprimen por la UART hacia un PC en el caso del nodo *sink* y se almacenan en la memoria flash en el caso de los nodos con funcionalidad completa. En las figuras 5.10 y 5.11 se puede ver un esquema de la aplicación mencionada.

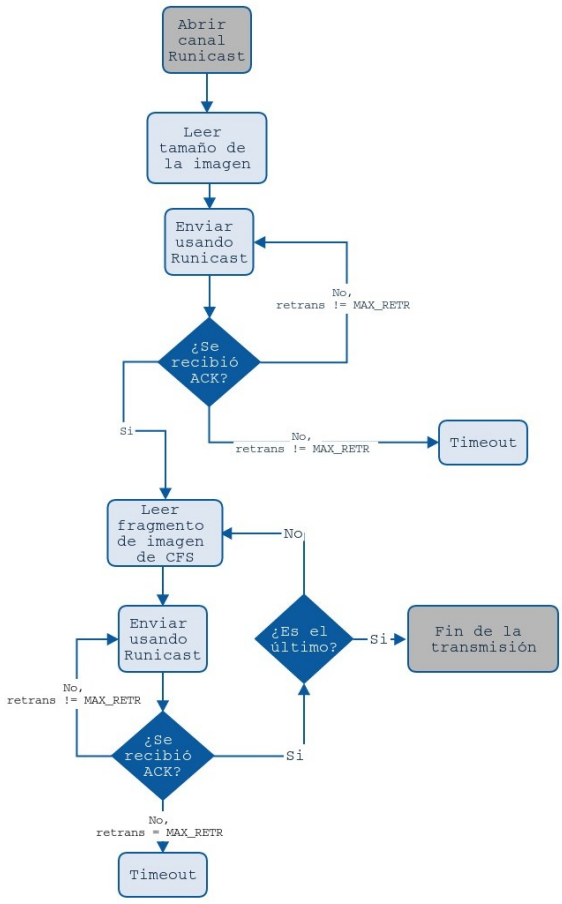


Figura 5.10: Diagrama de bloques de transmisión de radio

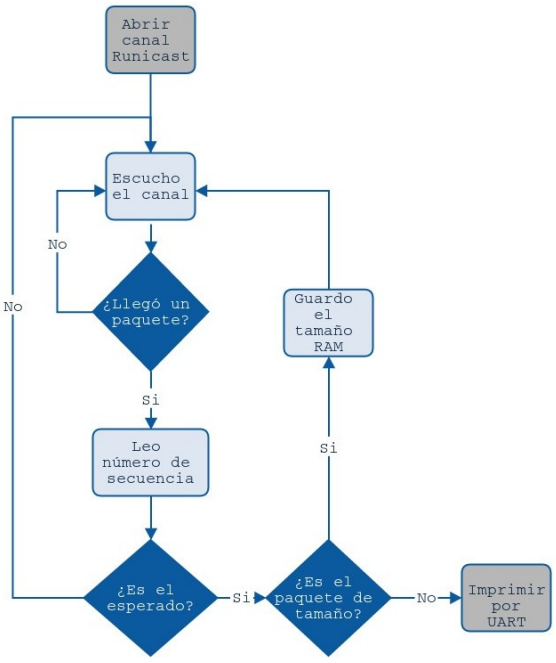


Figura 5.11: Diagrama de bloques de recepción de radio para el nodo sink

Las imágenes transmitidas tienen un peso aproximado de 18 kB, por lo que para transmitir una imagen se transmiten aproximadamente 180 paquetes.

5.3. Simulaciones en Cooja

Para evaluar el funcionamiento de los protocolos mencionados en la sección 5.2.5, se utilizó la herramienta Cooja [84], el simulador de red de Contiki OS. Los nodos se pueden emular a nivel de hardware, lo cual permite hacer una inspección precisa del funcionamiento del sistema. Permite simular varios modelos de nodos WSN y en particular los TelosB compatibles.

Primero se realizaron simulaciones punto a punto utilizando `Runicast` en donde se tenían dos nodos, uno transmitiendo y otro recibiendo, con un porcentaje de éxito de 100 %, es decir sin pérdida de paquetes.

5.3.1. Simulación nullRDC

Con nullRDC se completó la transmisión en $3,91\ s$ con una carga útil de 18 kB, por lo tanto el *throughput* de la transmisión fue de $36,8\ kbps$. El hecho de que la radio esté encendida en todo ese período hace que en todo momento se estén enviando los paquetes y los ACK.

En la figura 5.12 se puede observar como es un fragmento de la línea de tiempos de la transmisión y recepción de paquetes. La línea gris oscura continua representa que la radio está encendida permanentemente, los rectángulos azules corresponden a los paquetes transmitidos y los verdes implican la recepción de los mismos por parte del nodo receptor. Los rectángulos más pequeños representan los ACK. La transmisión de los paquetes de datos dura $3,85\ ms$.

Figura 5.12: Fragmento de la línea de tiempos de la transmisión y recepción de paquetes.

nullRDC con tres nodos

Se probó también intentar que dos nodos manden a un nodo *sink*. Debido a que el aire es un medio compartido existirán colisiones que demorarán las transmisiones. Puesto que la capa MAC utiliza CSMA/CA, al darse una colisión, ambos nodos iniciarán un *timer* que luego de expirar hará que se retransmitan los paquetes. Claramente, este proceso disminuye la eficiencia de utilizar nullRDC ya que se tiene un tiempo importante la radio encendida sin transmitir. Los tiempos de transmisión son ahora:

- Transmisor 1: $36,3\ s$ *Throughput*:$4\ kbps$
- Transmisor 2: $37,1\ s$ *Throughput*:$3,9\ kbps$

Esto reduce *throughput* en aproximadamente un 90 % respecto del caso de dos nodos transmitiendo sin interferencia, lo que hace necesaria cierta coordinación para evitar las colisiones. Una forma sería que cada nodo tuviera una ranura de tiempo para transmitir y luego se apague, dando lugar a un canal vacío para la transmisión del segundo. Serían entonces dos transmisiones independientes como la analizada en la sección 5.3.1.

La figura 5.13 muestra la colisión de dos paquetes con un rectángulo rojo y el hecho de que luego se deja de transmitir por un tiempo pseudo aleatorio.

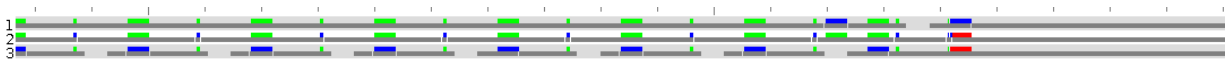


Figura 5.13: Colisiones en NullRDC

5.3.2. Simulación ContikiMAC

En una segunda instancia se simuló la transmisión de datos utilizando el RDC por defecto de Contiki OS, ContikiMAC con la misma topología de red mencionada en la sección 5.3. La duración de la transmisión es de 330 s, por lo que el *throughput* es de $0,44$ $kbps$. En la figura 5.14 se observa la línea de tiempos, allí se puede apreciar el mecanismo de ContikiMAC donde se realizan dos CCA cada 125 ms (se ven como dos comillas) para verificar si hay transmisiones en el canal. En caso de que se haya sincronizado la fase, entonces las transmisiones son más rápidas, pero cuando se desincroniza entonces pueden haber lapsos de tres o más segundos sin envíos. Por otra parte, todo el mecanismo debe realizarse tanto para enviar un paquete con datos como para enviar el ACK, lo que también contribuye a disminuir el *throughput*.

Figura 5.14: Línea de tiempos de ContikiMAC con envío de CCA

En algunos casos se envían demasiados paquetes antes que el transmisor reciba el ACK. Este hecho está manifiesto en la figura 5.15, donde se enviaron mas de 40 paquetes ACK de tamaño 20 bytes. Si bien en este caso se logra que el transmisor reciba el ACK, esto no ocurre en todos los casos y se necesita de otra transmisión similar para que se reciba correctamente el reconocimiento de paquete.

Figura 5.15: Envío sucesivo de ACK en ContikiMAC

5.3.3. Simulación X-MAC

En última instancia se simuló la transmisión de datos utilizando el protocolo de RDC X-MAC nuevamente con la topología de red mostrada en la sección 5.3. El tiempo de transmisión resultó en $45,2$ s lo que implica un *throughput* de $3,2$ $kbps$.

En la figura 5.16 se observa el funcionamiento del protocolo X-MAC, donde el transmisor envía un paquete de preámbulo con la dirección del nodo receptor y espera a recibir un ACK temprano, momento en el cual envía el paquete con los datos. De esta manera busca evitar escuchas innecesarias por parte de los demás nodos.

Figura 5.16: Diagrama de tiempo de X-MAC

En el ejemplo de la figura 5.16, para transmitir el ACK el receptor está 110 ms enviando el preámbulo para que el transmisor reciba el paquete de ACK.

Como se mencionó en la sección 5.2.3, el paquete de preámbulo funciona como un *strobe* de *wake-up* para despertar al receptor. En la figura 5.17 se tiene una visión más precisa de ese mecanismo.

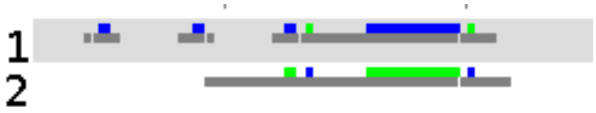


Figura 5.17: Preámbulo de X-MAC

5.4. Resultados experimentales

En esta sección se muestran las medidas experimentales de los perfiles de corriente del nodo transmitiendo con los distintos protocolos de RDC. Además de verificar el comportamiento teórico analizado en la sección 5.2.3 y que se observó en las simulaciones, se analiza el consumo de corriente de cada protocolo de RDC.

5.4.1. Perfiles de corriente

nullRDC

En la transmisión de una imagen completa (aproximadamente 18 kB o 180 paquetes), se observa un consumo de corriente elevado durante toda la transmisión. La figura 5.18 muestra un zoom durante la transmisión, en el que se ve un primer intervalo más largo y con un consumo mayor donde la radio está escuchando el canal, esperando a recibir un ACK, y un segundo intervalo más corto y con un consumo menor en el que la radio está transmitiendo. Esto se repite en intervalos regulares, sin esperar ningún tiempo entre escucha y transmisión hasta que se termine de mandar toda la carga útil.

ContikiMAC

Como se mencionó anteriormente, ContikiMAC utiliza un mecanismo por el cual el MCU está en LPM y el chip de la radio se encuentra apagado la mayor parte del tiempo. Realiza una medición del canal mediante dos CCA y luego envía copias del paquete de datos a modo de *wake-up*. Al recibir un ACK culmina el envío de paquetes.

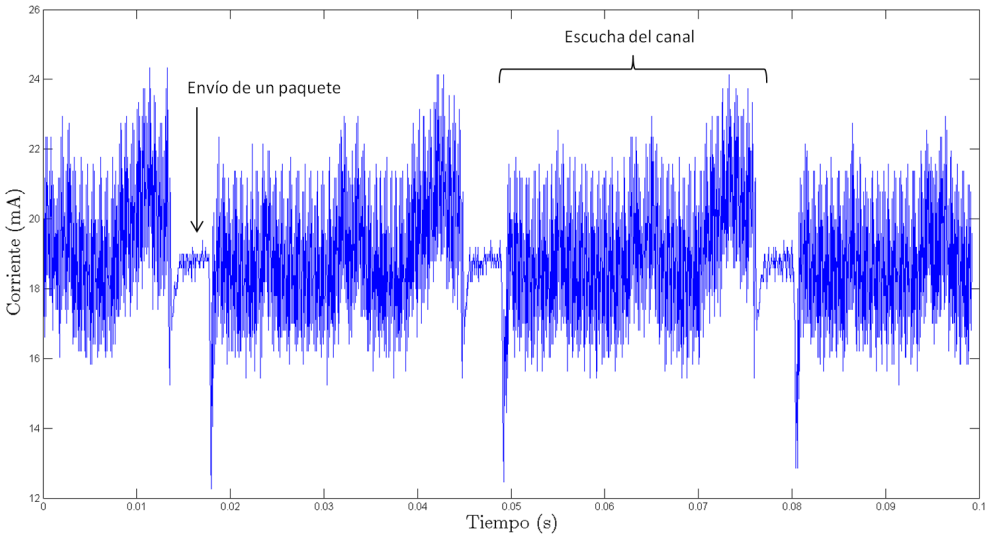


Figura 5.18: Zoom en transmisión para ver el envío de un paquete

En la figura 5.19 se ve la transmisión satisfactoria de un paquete. Primero el nodo está en modo sleep, por lo tanto se ve un consumo de corriente muy bajo y los ticks del sistema. Luego se ve como el nodo enciende la memoria flash para leer la información que va a mandar y luego escucha el canal previo a la transmisión. Una vez que determina que el canal está libre, transmite el paquete y escucha para ver si recibe un reconocimiento del receptor. Como no recibe nada, vuelve a enviar el paquete y repite el proceso hasta escuchar el ACK. Una vez que finalizó la transmisión correctamente, apaga la radio y vuelve al modo sleep.

Se pueden ver también dos CCA de los que hace el sistema cada 125 ms para ver si hay actividad en el canal.

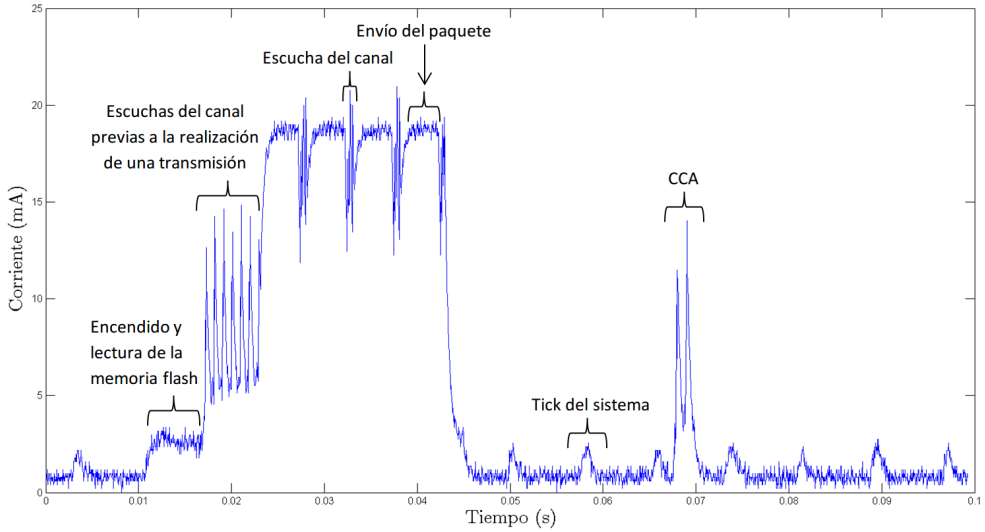


Figura 5.19: Zoom del perfil de corriente con ContikiMAC

X-MAC

Utilizando X-MAC se observa un consumo de corriente significativamente menor que utilizando nullRDC, pero la transmisión toma un tiempo considerablemente mayor. La figura 5.20 muestra un zoom durante la transmisión, en el que se puede observar el funcionamiento de X-MAC. Primero se ve como la radio transmite periódicamente el *strobe* de escucha y escucha el canal, esperando recibir un reconocimiento del receptor. Cuando lo recibe, inmediatamente comienza a transmitir. Cuando termina de transmitir se queda con la radio prendida, esperando escuchar el ACK del receptor. Una vez que recibe el ACK, primero apaga la radio y un tiempo después pone el MCU en LPM. Mientras el MCU esta en LPM tiene un consumo muy bajo, y se ven cada 8 ms los ticks del sistema.

Cabe destacar que no siempre quedan sincronizados transmisor y receptor, por lo que el transmisor vuelve mandar los *strobes* de escucha y escuchar el canal.

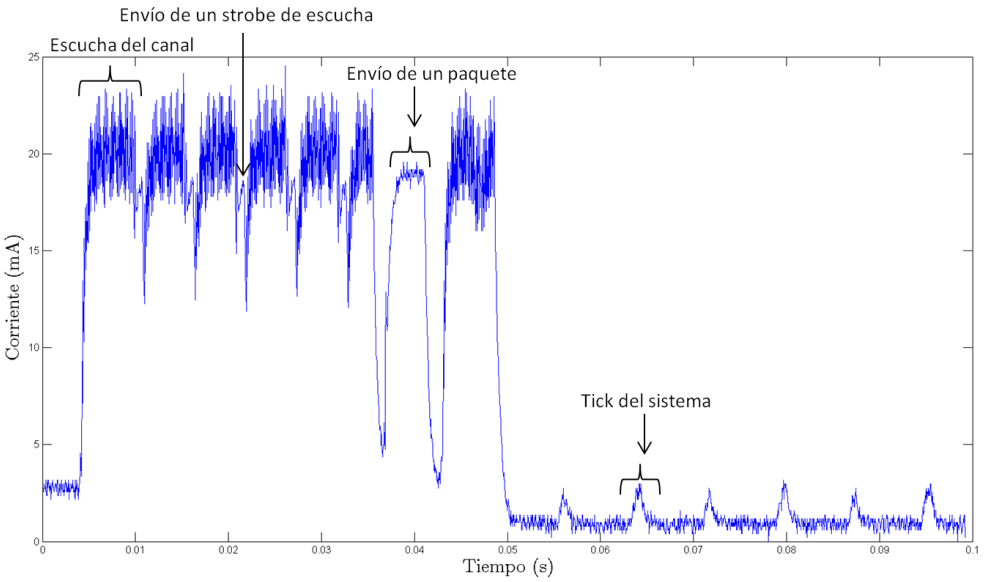


Figura 5.20: Perfil de corriente utilizando X-MAC

5.4.2. Análisis de performance

Como programa de prueba para medir el consumo se implementó una aplicación que envía una imagen para luego configurar el nodo en modo *sleep*. Para cada protocolo de ciclo de trabajo de la radio se hicieron cinco medidas a una distancia entre nodos de dos metros en transmisión y recepción y se promediaron.

Las medidas que se presentan en la tabla 5.1 corresponden a

- Consumo promedio: Corriente demandada media (en mA) durante la transmisión y recepción.
- Tiempo de transmisión y recepción de una imagen.

- Carga consumida: Carga consumida (en mAh) durante la transmisión y recepción de una imagen.

Protocolo	I_{TX} (mA)	t_{TX} (s)	Q_{TX} (mAh)	I_{RX} (mA)	t_{RX} (s)	Q_{RX} (mAh)
nullRDC	18,70	5,43	0,0282	21,20	5,63	0,0332
ContikiMAC	1,88	89,76	0,0470	2,08	92,36	0,0534
xMAC	4,85	47,60	0,0641	9,55	46,2	0,1226

Tabla 5.1: Tabla de resultados

Para poder comparar correctamente la carga consumida por los distintos protocolos de RDC, deberíamos agregar el consumo correspondiente al modo *sleep* del nodo para la diferencia de tiempos con el protocolo que demora más en enviar el paquete. Como se detalla en el capítulo 6, el consumo promedio del nodo en modo *sleep* es de $96,40\ \mu A$, por lo tanto en $90\ s$ significaría una carga de $2,41\ \mu Ah$, lo cual no altera el resultado de la comparación. Podemos afirmar entonces que nullRDC presenta los menores costos energéticos por imagen enviada.

5.5. Implementación futura de la red

En este capítulo se estudiaron distintos protocolos para la transmisión de imágenes de un nodo a otro. Esto sería suficiente en una comunicación punto a punto, donde el transmisor enviara únicamente a un receptor sin interferencia. Mediante tablas de ruteo estáticas y transmisiones en ranuras de tiempo asignadas a cada nodo, se puede extender este análisis a redes *multihop*, como se muestra en la figura 5.21.

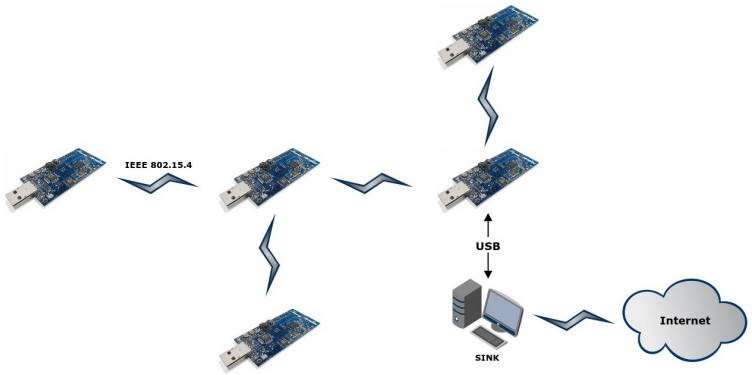


Figura 5.21: Topología de la red

En una red con varios nodos transmitiendo a un solo nodo receptor, ya sea directamente o ruteando a través de un nodo vecino, es fundamental la sincronización de los mismos. Este aspecto no se estudió en profundidad por quedar fuera del alcance del proyecto, pero de la investigación realizada se concluyó que una buena solución para este problema es usar una combinación de los RDC como se detalla a continuación.

Para transmitir imágenes entre dos nodos (punto a punto), es decir cientos de paquetes sin pérdidas, errores ni duplicados, y en el orden correcto, se concluyó que nullRDC era el protocolo

de RDC con el que que se consumía menos energía por transmisión. A su vez se verificó que el sistema, transmitiendo con los protocolos detallados en este capítulo, es capaz de transmitir imágenes cumpliendo holgadamente los requerimientos energéticos del proyecto, como se muestra en la sección 6.3.

Sin embargo, el armado de la red (incluyendo tablas de ruteo y sincronización) se asemeja más a una aplicación clásica de WSN y puede ser beneficioso usar un protocolo de RDC como ContikiMAC o X-MAC. Si bien el análisis del armado de una red de varios nodos quedó fuera del alcance del proyecto, quedó documentado en este capítulo el análisis teórico del funcionamiento de los distintos protocolos de RDC y queda planteado hacer un estudio exhaustivo de cuáles son los protocolos de capas superiores convenientes para esta última aplicación.

Una posible solución al problema de la sincronización podría ser activar el sistema con ContikiMAC un tiempo previo al enviar la imagen para ejecutar un ruteo dinámico y establecer la ranura de tiempo de la transmisión de cada nodo y luego pasar a nullRDC para enviar los datos.

En el caso de que solo se transmita el recuento de polillas, el problema se transforma en un problema clásico de WSN donde se intercambian unos pocos bytes de datos, siendo posible utilizar distintas aplicaciones ya implementadas en Contiki OS para recolección de datos en redes *mesh* o árbol y mantener un nivel de consumo dentro de los márgenes considerados. Para esta aplicación también queda pendiente hacer un análisis exhaustivo de cuál es el protocolo de RDC más conveniente.

Capítulo 6

Caracterización y ensayos

Resumen

El presente capítulo contiene una descripción de los distintos ensayos para caracterizar el sistema. Se detallan los procedimientos, resultados y análisis que permiten describir el consumo de corriente, el alcance de las transmisiones, performance del sistema de iluminación y del algoritmo de procesamiento de imágenes. Estos resultados fueron, en algunos casos, cruciales para tomar decisiones sobre detalles de implementación y construcción del prototipo.

En cuanto al consumo de corriente, se constató que el sistema cumple ampliamente con los requerimientos funcionales al obtenerse una independencia energética de aproximadamente 28 meses tomando y transmitiendo dos fotos por día a partir de medidas de perfiles de corriente. El alcance máximo en línea vista registrado de las transmisiones es de 144 m aunque se aprecia una significativa disminución en la performance de la red.

Se encontró que mediante la utilización de dos LEDs en lugar de cuatro se obtenía además de un consumo menor, una mayor separación entre el fondo y las polillas.

Contenido

6.1. Ensayos de perfiles de corriente

Mediante estos ensayos se pretende caracterizar el consumo de corriente de los distintos componentes que integran el sistema: el nodo CM3000, la cámara Linksprite LS-Y201, el circuito interfaz y el sistema de iluminación. En lo respectivo al nodo se caracterizan separadamente los consumos del MCU MSP430F1611, la radio CC2420 y la memoria flash externa M25P80 con el fin de evaluar los modos de bajo consumo del segundo y el tercero respectivamente.

La carga de las baterías AA se expresa en mAh. Como se trabaja a un voltaje aproximadamente constante, esta carga es una medida indirecta de la energía. Para poder comparar con este valor, los consumos medidos en todos los casos corresponden a consumos de corriente eléctrica por tiempos determinados.

A medida que se desarrolló el módulo `sleep` en Contiki OS, se comenzaron a realizar estas medidas para corroborar los consumos de corriente esperados. Se encontró que la implementación de los modos de bajo consumo del RTOS no eran suficientes para los requerimientos del sistema, por lo que se tuvo que investigar acerca de cómo reducir aún más el consumo de corriente. Lo implementado en la sección 3.3.1 es el resultado de un trabajo iterativo entre la configuración de distintos registros del MCU y el consumo obtenido.

6.1.1. *Setup* experimental

Para relevar los perfiles de corriente en cada uno de los casos de estudio se utilizó el *setup* que se muestra en la figura 6.1. La medida de la carga consumida se realiza en forma indirecta ya que se registra la corriente I que circula por una resistencia en serie con el circuito que se desea relevar, que es claramente la misma I que circula por el circuito.

Por lo tanto, el procedimiento de medida es el siguiente: *i)* Para una resistencia R caracterizada previamente se mide la diferencia de potencial en sus bornes con un osciloscopio durante el período de tiempo deseado y se almacenan los datos. *ii)* Utilizando software de procesamiento (en nuestro caso Matlab) se integra la medida de la diferencia de potencial en el tiempo y se divide entre la resistencia ya que de acuerdo a la ley de Ohm $I = V/R$.

Una vez que se tiene la carga consumida, se puede expresar el resultado en Ampere hora (mAh) para comparar de manera directa con la carga total de las pilas. Para hacer esta conversión basta con dividir entre un factor de 3,6 ya que:

$$1\ mAh = 1 \times 10^{-3}\ A \times 3600\ s = 3{,}6\ C \tag{6.1}$$

Lógicamente la precisión de los resultados está ligada en forma directa a la exactitud con la que se mida la resistencia patrón y la exactitud del osciloscopio para la medición de voltaje y registro del tiempo. En nuestro caso se utilizó una resistencia de valor nominal 10 Ω. La misma se midió con un multímetro Fluke 82, registrándose un valor de $10,2\ \Omega$.

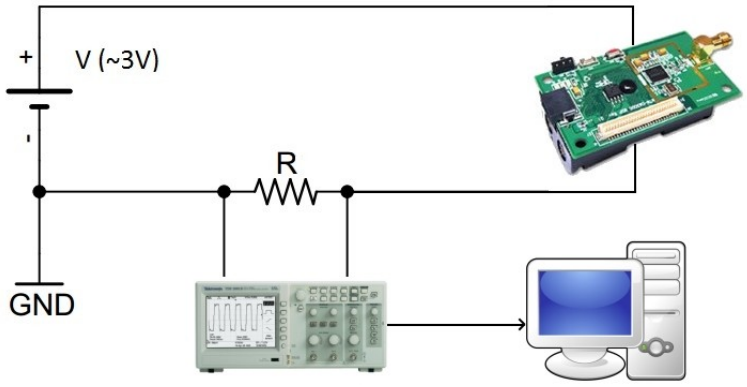


Figura 6.1: Circuito de prueba para tomar las medidas necesarias

6.1.2. Perfil de corriente del nodo CM3000

La motivación de caracterizar el perfil de corriente del nodo se basa en que alcanzar bajos niveles de consumo en modo *sleep* será determinante en la autonomía del sistema ya que el nodo se encuentra más del 99 % del día en este estado (véase sección 6.3). El fabricante declara los consumos de corriente observados en la tabla 6.1 para los modos de *sleep* de los distintos componentes [7] [8].

	Nominal	**Max**	**Unidades**
MCU idle	54,5	1200	μA
MCU idle, radio RX	18,8		mA
MCU on, radio RX	21,8	23	mA
MCU on, radio TX	19,5	21	mA
MCU LPM3	2,6	3	μA
Flash on (READ)	-	4	mA
Flash on (WRITE/ERASE)	-	20	mA
Flash idle	8	50	μA
Flash Deep Power Down Mode	1	10	μA
Radio idle, OSC on	426		μA
Radio Power Down Mode, OSC off, VREG on	20		μA
Radio Power Down Mode, OSC off, VREG off		1	μA

Tabla 6.1: Consumos declarados por fabricantes

En una primera instancia se probó el peso de los distintos componentes del nodo cambiando el estado de la radio, la flash externa y el MCU para verificar los valores declarados por los fabricantes. Para ello se escribió una aplicación de Contiki OS que apagara o llevara a modo *sleep* periódicamente (a intervalos de 5 s) la radio, el oscilador de cristal de la radio, el regulador de voltaje de la radio, la memoria flash externa y el MCU. Cabe destacar que la implementación de Contiki OS prevé que el nodo se lleve a estados de bajo consumo cuando no se realiza procesamiento, lo que se corrobora en cada *tick* del sistema (cada 8 ms). El diagrama de la figura 6.2 ilustra el módulo antes descrito con los valores de consumo nominales declarados en la tabla 6.1.

En la figura 6.3 se observa el perfil del nodo en todo el proceso descrito anteriormente. Por razones de escala en esta captura los cambios distinguibles solo se observan en los pasajes del estado

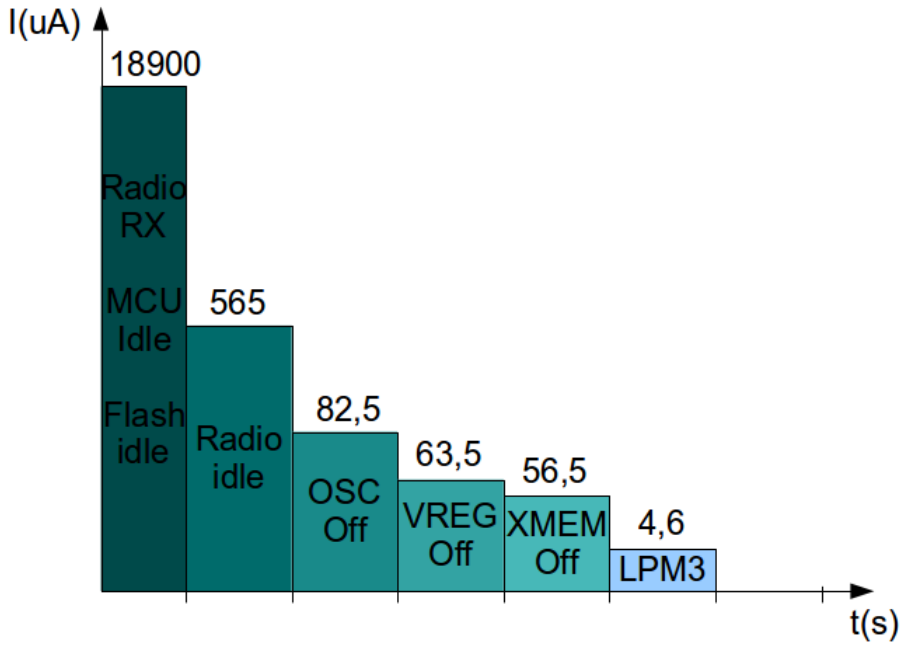


Figura 6.2: Diagrama ilustrativo del módulo de prueba, donde en cada leyenda se indica el cambio con respecto al estado anterior.

1 al 2 y del estado 2 al 3. En el estado 1 el MCU y la memoria flash se encuentran en modo *idle* y la radio en encendida en RX consumiendo en total 19 mA. En el estado 2 la radio pasa a modo *idle* cambiando el consumo total a 600 μA. En el estado 3 se apago el oscilador de la radio reduciendo el consumo total a 200 μA aproximadamente. Los datos relevados son consistentes por estar dentro del rango declarado por los fabricantes según la tabla anterior.

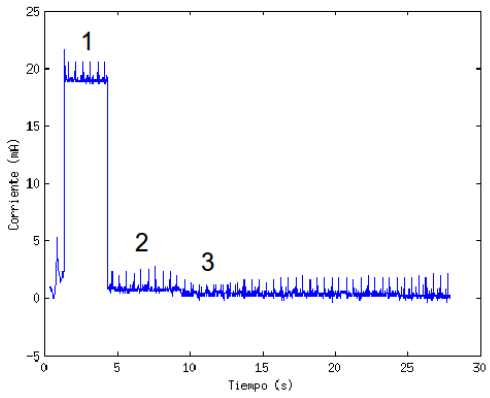


Figura 6.3: Perfil de corriente apagando progresivamente los periféricos

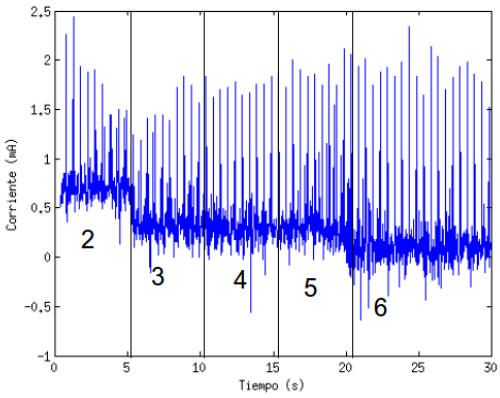


Figura 6.4: Zoom de estados 2 al 6

En la figura 6.4 se aprecia la misma secuencia a una escala menor, viendo en detalle los estados 2 al 6. En el estado 4 se apaga el regulador de voltaje de la radio, reduciéndose el consumo total a 180 μA. Este cambio no es apreciable en la figura por la escala. En el estado 5 se lleva la memoria flash a modo *Deep Power Down*, resultando un consumo total de 150 μA (este cambio tampoco es apreciable por la escala). Finalmente, en el estado 6 se lleva el MCU a LPM3, donde se apaga el oscilador DCO del mismo, reduciéndose el consumo a 40 μA.

Este último cambio de consumo es mayor a la diferencia entre el consumo nominal del MCU en modo *idle* y el consumo nominal en LPM3. Esto puede deberse a que en modo *idle* el MCU esté consumiendo una corriente mayor a la nominal. No obstante, estando el MCU en LPM3, el consumo del nodo debería ser de unos pocos μA, y no 40 μA.

Como se explicó en la sección 3.3.1 se utiliza la función `msp430_remove_lpm_req(0)` para introducir el MCU en LPM3. Esta cambia una variable llamada `msp430_dco_required` para distinguir si se requiere el uso del DCO por algún periférico y al no requerirse se ejecuta el macro `_BIS_SR(GIE | SC1 | SC2 | CPUOFF)` con la que se lleva a LPM3.

Para distinguir si efectivamente el sistema estaba entrando en LPM3 se manipuló el código del módulo tal que no ejecutara la función `msp430_remove_lpm_req(0)` y se midió nuevamente. En las figuras 6.5 y 6.6 se observan los modos LPM0 y LMP3 respectivamente, verificándose la diferencia de consumo base entre uno y otro.

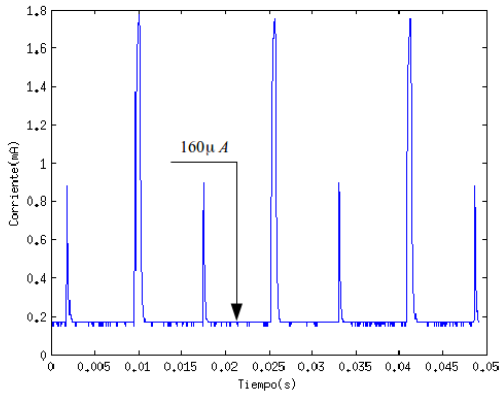


Figura 6.5: Perfil de corriente en LPM0

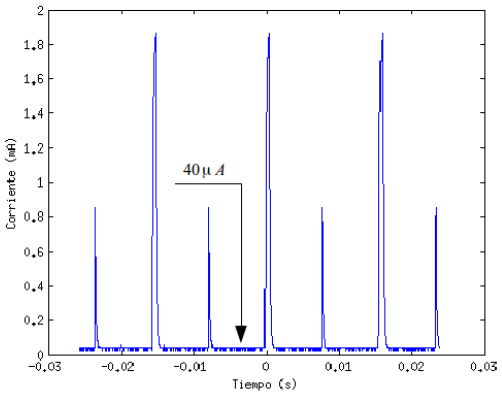


Figura 6.6: Perfil de corriente en LPM3

Esto aún no explica el por qué de los 40 μA, por lo que se atacó el problema buscando en dónde se ajusta el registro SR (*Status Register*) del MCU para cambiar los estados de bajo consumo. La función `main` de Contiki OS para la plataforma (ubicada en `contiki-sky-main.c`) es la que realiza las ejecuciones del sistema, en particular el pasaje a LPM. Para corroborar que se estuviera entrando en LPM3, se cambió el código para que en lugar de entrar a LPM3 se llevara el MCU a LPM4 al apagar el oscilador interno. Según declara el fabricante del MSP430, los consumos de LPM3 y LPM4 difieren en unos pocos μA.

Por más que este modo rompe la ejecución del Contiki OS (al no tener ticks del sistema, sólo se puede despertar por interrupciones externas), se verificó que el consumo se encuentra en el entorno de los 40 μA. Esto es un indicador concluyente de que se trata del consumo base de la plataforma CM3000 con la configuración que realiza Contiki OS sobre ellos.

Alternativas consideradas para bajar ese consumo fueron cambiar la configuración de los puertos I/O y colocarlos como salidas en nivel bajo para evitar posibles consumos pero tampoco se alcanzó un cambio.

De todas maneras, en experiencias previas en el uso de Contiki OS dentro del IIE-FING-UdelaR se alcanzaron valores mínimos de consumo base de 243 μA [12], por lo que el consumo alcanzado se consideró exitoso. A su vez el consumo alcanzado permite superar ampliamente los requerimientos

energéticos del proyecto.

6.1.3. Perfil de corriente de la cámara

La cámara es el componente de mayor consumo dentro del sistema. En la figura 6.7 se observa el perfil de corriente de la cámara Linksprite LS-Y201 tomado al sustituir la cámara por el nodo en el circuito de pruebas de la figura 6.1. Es apreciable que el perfil se mantiene en el tiempo independientemente de la actividad de la cámara, en cuyo caso el consumo de corriente promedio registrado es de 71,1 mA . De todas maneras el tiempo entre que se inicializa la cámara y se termina de descargar está en el entorno de los 12 segundos, por lo que no representa una gran consumo como se verá en la sección 6.3.

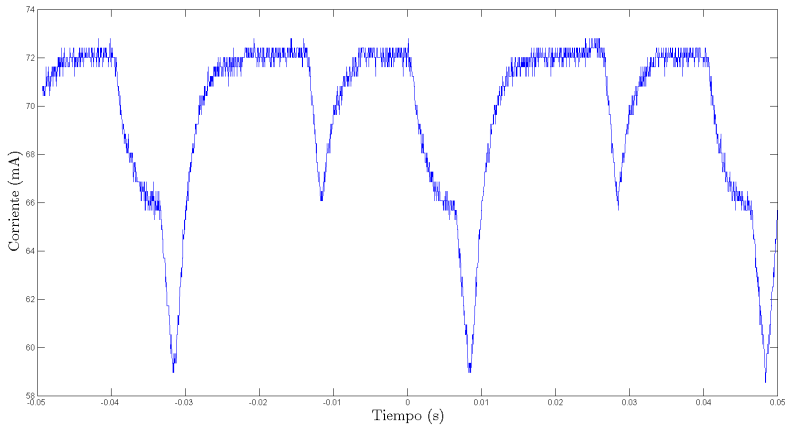


Figura 6.7: Perfil de consumo de la cámara

6.1.4. Perfil de corriente del sistema total

En esta sección de detalla el perfil de corriente del sistema completo en ejecución. Las medidas y análisis se realizaron únicamente para los nodos con funcionalidad reducida, es decir que no reciben y retransmiten imágenes ya que son los que están comprendidos en el alcance del proyecto. En la figura 6.8 se muestra un diagrama representativo de la ejecución en cuanto a tiempos y consumo de corriente medido experimentalmente. Cabe recordar que el flujo del sistema tiene los siguiente estados:

1. Inicialización del nodo. En este estado se inicializan variables y se formatea la memoria flash. El MCU y la memoria flash externa están prendidos mientras que la radio, con su oscilador y regulador de voltaje se encuentran apagados.

2. Inicialización de la cámara y captura y almacenamiento de foto. En este estado se inicializa la cámara, se encienden los LEDs, se toma una imagen, se apagan los LEDs y se transfiere la imagen a la memoria flash externa. Durante todo este tiempo hay un consumo de corriente predominante de la cámara. La diferencia con el estado anterior es que la cámara se encuentra encendida.

3. Envío por radio de la imagen. En este estado se transmite la imagen. Se tiene un consumo como el del estado en el ítem 1 pero con la radio, con su oscilador y regulador de voltaje encendidos.

4. Procesamiento de imagen. En este estado se procesa la imagen para hacer el conteo automático. Se tiene un consumo como el del ítem 1.

5. Envío por radio del conteo. En este estado se tiene un consumo como el del ítem 3.

6. Sistema en modo *sleep*. En este estado el MCU se encuentra en LPM3, la memoria flash externa se encuentra en modo *Deep Power Down*, la radio, con su oscilador y regulador de voltaje se encuentran apagados. Tanto la cámara como los LEDs y sensores externos también se encuentran apagados.

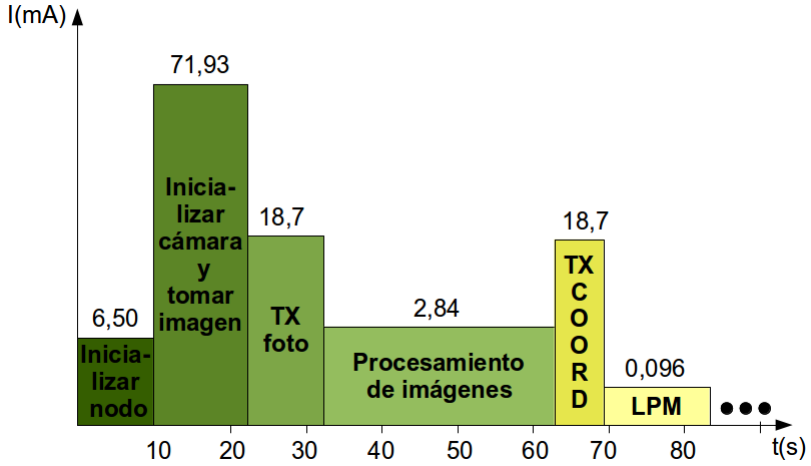


Figura 6.8: Diagrama de ejecución total y el peso ilustrativo de cada consumo

En la figura 6.9 se muestra una captura del perfil de corriente del sistema en la ejecución total. Los tiempos no se corresponden con los de la aplicación final en cuanto a los tiempos de transmisión de la imagen y sus coordenadas ya que se trata de condiciones de laboratorio. Además, entre los vectores de coordenadas se dejó un tiempo de un segundo para facilitar la depuración. Los tiempos de la aplicación final se pueden consultar en la sección 6.3.

Se observa que el consumo de corriente es el esperado. En el estado 1 el consumo es de 6,3 mA ya que se está formateando la memoria flash y se tiene el MCU en modo activo. En el estado 2 el consumo es de 79 mA puesto que se tiene la cámara encendida, el MCU en modo activo y se está escribiendo en flash. Cabe destacar que la caída en el consumo de corriente que se observa en esta etapa corresponde a la secuencia de reset que se ejecuta luego de cambiar la resolución de la imagen que se mencionada en la sección 3.2.2. En el estado 3 se tiene un consumo de 20,6 mA con la radio transmitiendo, la cámara apagada y el MCU activo leyendo los datos desde la flash. En el estado 4 la corriente consumida es de 4 mA ya que se está procesando (MCU activo) y leyendo y escribiendo la memoria flash. En el estado 5 el consumo es análogo al estado 3 ya que se transmite por radio y por último en 6 se está en LPM3 consumiendo 40 μA.

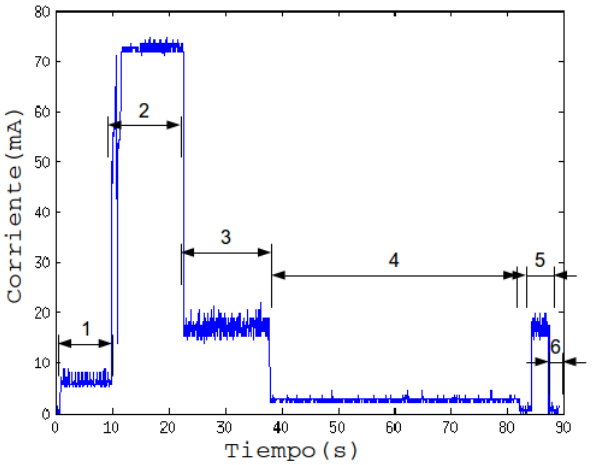


Figura 6.9: Perfil de corriente de la ejecución total de la aplicación

6.2. Alcance

Para medir el alance del nodo transmitiendo a una potencia de 0 dBm se hicieron medidas de campo. Se buscó un lugar con la menor interferencia posible en la frecuencia de 2,4 GHz, sin obstáculos que pudieran generar rebotes y en donde se pudieran hacer medidas con línea vista entre el nodo transmisor y receptor. Se transmitió una imagen tomada en el momento por el nodo transmisor y luego se transmitió un vector con los tiempos de transmisión. Los resultados se muestran en la tabla 6.2, donde el *throughput* se calculó tomando 18 kB como tamaño promedio de las imágenes.

Distancia (m)	**Tiempo (s)**	***Throughput* (kbps)**
10	7,59	19,0
20	9,73	14,8
50	11,6	13,1
80	33,2	4,34
100	39,5	3,65
120	80,1	1,80
141	142	1,01

Tabla 6.2: Alcance del nodo

El alcance máximo teórico en condiciones de línea vista declarado en la hoja de datos del TelosB es de 120 m. En la figura 6.10 se ve claramente que a partir de los 100 m el tiempo de transmisión crece con la distancia con una pendiente mucho mayor debido a la elevada pérdida de paquetes que se produce debido a la disminución de la potencia en recepción. Esto es coherente con el alcance teórico declarado por el fabricante.

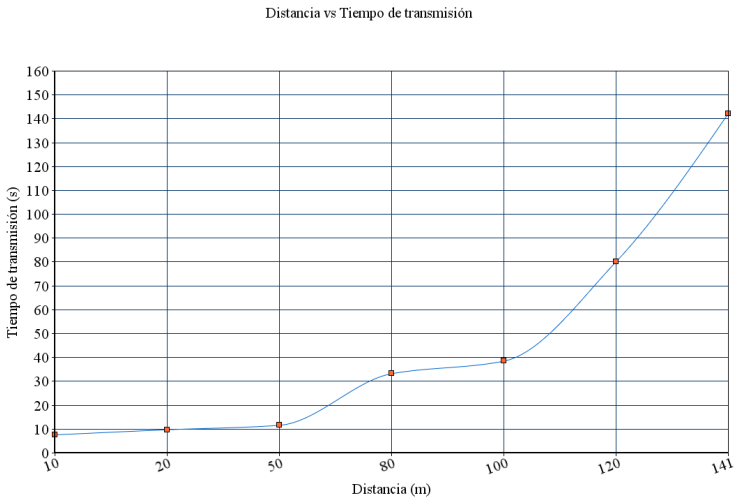


Figura 6.10: Tiempo de transmisión en función de la distancia

6.3. Estimación de la duración de las baterías

Con el objetivo de usar baterías que se consigan en el mercado local y sean fácilmente intercambiables, se usaron pilas AA y se tomó como referencia una capacidad de 2800 mAh. Para estimar la autonomía energética del sistema, se caracterizó el consumo en los estados que se describieron en la sección 6.1.4. En la tabla 6.3 se detalla la carga consumida en cada uno de los estados mencionados. Las transmisiones de la radio se hicieron con una potencia de transmisión de 0 dBm, a una distancia de 100 m entre el transmisor y el receptor.

Item	Estado	$I_{consumida}$	$t_{estado}(s)$	$Q_{estado}(mAh)$
1	Inicialización del nodo	$6,50\ mA$	$8,88$	$0,0160$
2	Inicializar cámara y sacar foto	$71,9\ mA$	$12,6$	$0,252$
3	Envío por radio de la imagen	$18,7\ mA$	$39,5$	$0,205$
4	Procesamiento de imagen	$2,84\ mA$	$44,1$	$0,0348$
5	Envío por radio del conteo	$18,7\ mA$	$0,826$	$0,00429$
6	Sistema en modo sleep	$96,4\ uA$	86300	$2,31$

Tabla 6.3: Carga consumida en cada etapa

Se observa que el tiempo de ejecución de la aplicación, cada vez que se sale de modo *sleep*, es de $t_{total} = 106\ s$ por lo que se desprende que el sistema se encuentra en modo de bajo consumo el 99 % del día.

6.3.1. Duración enviando una imagen diaria

En la hipótesis de tomar una imagen por día, la carga total consumida es como se muestra en la ecuación 6.2, donde cada Q_i corresponde a la carga en los estados mencionados en la tabla 6.3.

$$Q_{1\times imagen} = Q_1 + Q_2 + Q_4 + Q_4 + Q_5 + Q_6 = 2,82\ mAh \tag{6.2}$$

Por lo tanto la autonomía energética esperada del sistema, basada en dos baterías AA de 2800 mAh, da poco mas de 33 meses como se muestra en la ecuación 6.3.

$$\frac{2800\ mAh}{2,82\ mAh \times 30\ dias} = 33,1\ meses \tag{6.3}$$

El peso relativo de cada estado se puede ver en la gráfica de torta mostrada en la figura 6.11.

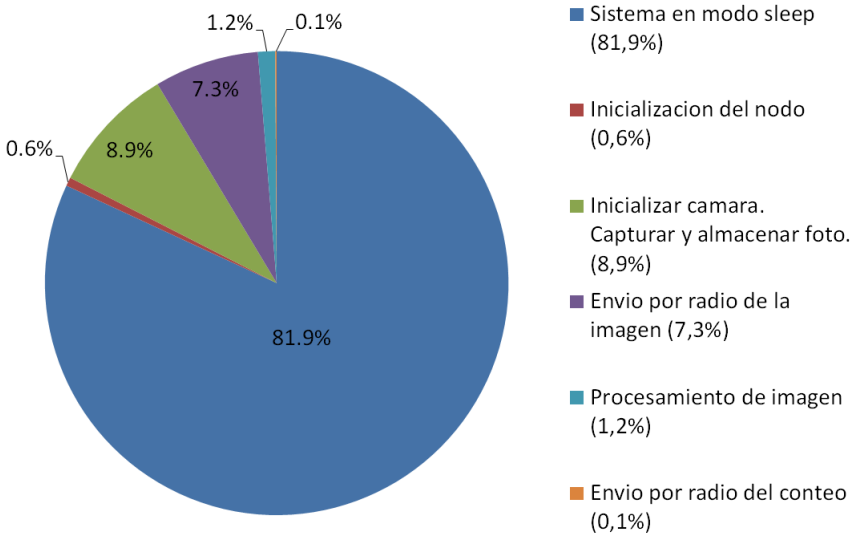


Figura 6.11: Peso de cada estado en la carga total

6.3.2. Duración enviando dos imágenes diarias

En la suposición de tomar dos imágenes por día, cambia ligeramente la carga Q_6 ya que el tiempo en modo *sleep* pasa a ser 100 segundos menor. Como se demostró en la sección 5.4.2, esta diferencia de carga es despreciable por lo que no será tenida en cuenta. Por lo tanto, la carga total consumida es como se muestra en la ecuación 6.4.

$$Q_{2\times imagen} = Q_6 + 2 \times (Q_1 + Q_2 + Q_3 + Q_4 + Q_5) = 3,33\ mAh \tag{6.4}$$

En esta situación la autonomía energética esperada del sistema, basado en dos baterías AA de 2800 mAh es de 28 meses como se muestra en la ecuación 6.5.

$$\frac{2800\ mAh}{3,33\ mAh \times 30\ dias} = 28,0\ meses \tag{6.5}$$

6.3.3. Duración enviando solamente el conteo dos veces al día

En la hipótesis de enviar únicamente el conteo de polillas dos veces al día, la carga total consumida es como se muestra en la ecuación 6.6.

$$Q_{2\times conteo} = Q_6 + 2 \times (Q_1 + Q_2 + Q_4 + Q_5) = 2,62\ mAh \tag{6.6}$$

En esta situación la autonomía energética esperada del sistema, basado en dos baterías AA de 2800 mAh es de un poco más de 35 meses como se muestra en la ecuación 6.7. Además del aumento en la autonomía energética del sistema, ésta alternativa supone la gran ventaja de facilitar en gran medida la planificación e implementación de la red. Esta diferencia se hace aún más significativa al trabajar en redes *mesh multi-hop*.

$$\frac{2800\ mAh}{2,62\ mAh \times 30\ dias} = 35,6\ meses \tag{6.7}$$

6.3.4. Duración en otras configuraciones

Otras alternativas plateadas a lo largo del proyecto fueron:

- Enviar la imagen sólo cuando el sistema detecta un aumento significativo (a determinar) en el recuento de polillas. Esto implicaría ahorrarse varias transmisiones de imágenes, lo que aumentaría la autonomía energética del sistema respecto al caso analizado en la sección 6.3.2. Tiene como ventaja que frente a un error en el conteo automático, la imagen puede ser supervisada por un operario.
- Enviar una alarma cuando el recuento supera un umbral establecido. Si bien esto no mejora significativamente la autonomía energética de sistema respecto al caso analizado en la sección 6.3.3, puede ser una buena aplicación futura que facilite el trabajo al cliente.

6.3.5. Análisis del resultado de la duración de las baterías

Considerando el peor caso en cuanto a consumo energético, que es en el que se envían dos imágenes diarias, el sistema consume en seis meses una carga de $3,33\ mAh \times 30 \times 6 = 600\ mAh \Rightarrow \frac{Q_{6meses}}{Qbaterias} = 21\,\%$. Por lo tanto queda un $79\,\%$ de la carga de las baterías para implementaciones futuras. Se considera que éste es un buen margen de guarda para trabajos futuros en cuanto a red y recolección de otras magnitudes físicas en la WSN.

El análisis anterior corresponde a una comunicación punto a punto entre dos nodos. Si tenemos por ejemplo, una topología tipo árbol en la que un nodo tiene N nodos hijos, tendrá que transmitir

N imágenes hacia el *sink* y recibir $N-1$ imágenes. Esto hace que la autonomía energética de dicho nodo disminuya sustancialmente respecto de los otros. Queda como trabajo a futuro caracterizar el consumo en las distintas topologías de red.

6.4. Iluminación

Como se mencionó en la sección 2.5.1, se seleccionó un sistema de iluminación integrado por dos LEDs de color amarillo. En esta sección se presentan los ensayos realizados para determinar el color y la cantidad de los mismos. Se tomaron imágenes con las distintas configuraciones mediante ensayos repetibles con la trampa colocada dentro de un recinto oscuro. Las imágenes capturadas se muestran en las figuras 6.12, 6.13, 6.14 y 6.15.

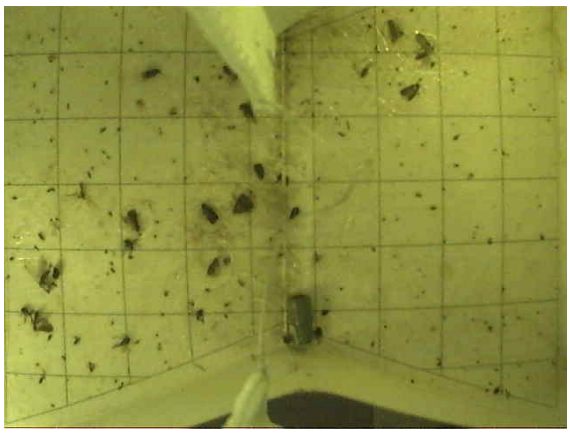


Figura 6.12: Iluminación con dos LEDs amarillos dispuestos en esquinas cruzadas

Figura 6.13: Iluminación con cuatro LEDs amarillos dispuestos uno en cada esquina

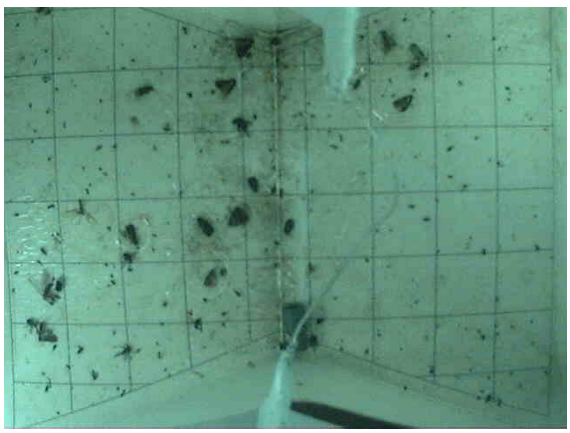


Figura 6.14: Iluminación con dos LEDs blancos dispuestos en esquinas cruzadas

Figura 6.15: Iluminación con cuatro LEDs blancos dispuestos uno en cada esquina

Como uno de los objetivos principales del proyecto es el bajo consumo de corriente, se decidió minimizar la cantidad de LEDs utilizados. Usando dos LEDs en lugar de cuatro, se baja el consumo

de corriente del sistema de iluminación a la mitad pero a su vez se pierde uniformidad en la iluminación.

Para elegir el color de los LEDs se determinó la entropía y se realizó el histograma de cada una de las imágenes. Los histogramas de la imagen en escala de grises nos muestran la frecuencia de píxeles con la que aparece cada cada nivel de gris, del 0 al 255.

La entropía es una medida estadística de la aleatoriedad que puede ser usada para caracterizar el promedio de bits por pixel necesarios para codificar una imagen sin distorsión. Se define como $E = -\sum(p \times \log_2(p))$, donde p es la frecuencia de cada punto del histograma.

Para el caso de dos LEDs blancos obtuvimos un valor de $E = 6,97$ bits/pixel mientras que para los LEDs amarillos obtuvimos un valor de $E = 7,14$ bits/pixel.

En las figuras 6.16 y 6.17 se pueden ver los histogramas de las imágenes tomadas al piso de la trampa iluminando con LEDs blancos y amarillos respectivamente.

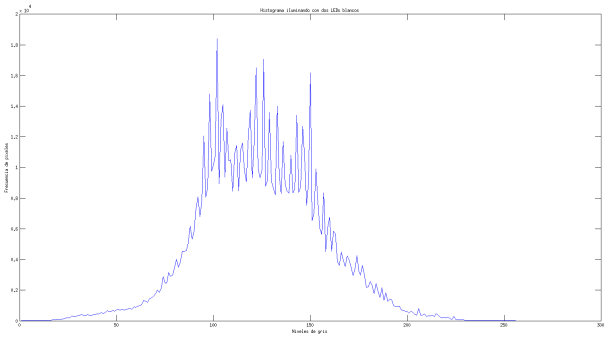


Figura 6.16: Histograma de la imagen iluminando con dos LEDs blancos

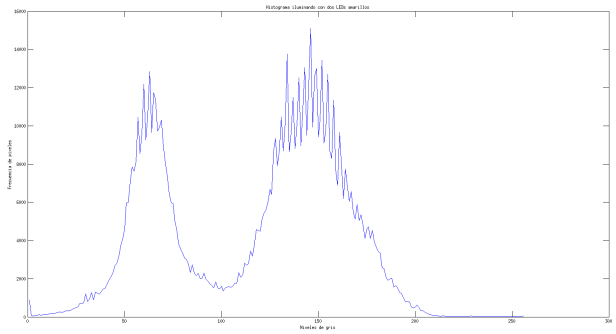


Figura 6.17: Histograma de la imagen iluminando con dos LEDs amarillos

Los histogramas muestran que la distribución de niveles de gris es más uniforme en el caso que iluminamos con LEDs blancos, mientras que con los LEDs amarillos queda más separado lo que es fondo de lo que son otros objetos, lo cual puede ser útil a la hora de detectar polillas de forma automática. Por otro lado la entropía muestra que la imagen tomada iluminando con LEDs amarillos tiene mayor información en promedio por pixel.

Estas herramientas dan información general sobre la imagen y no son una medida totalmente objetiva de cuál es la mejor iluminación. Creemos que la mejor forma de caracterizar la iluminación

hubiera sido tomando una cantidad considerable de imágenes con cada color de LED y aplicarle los algoritmos de procesamiento detallados en el capitulo 4, observando en qué caso se obtienen mejores resultados. Como no se disponían de imágenes suficientes para hacer estas pruebas, se resolvió iluminar con dos LEDs amarillos, en función de los parámetros analizados en esta sección.

Finalmente se ensayó el consumo de corriente del sistema de iluminación para determinar la carga diaria consumida. Cada LED aporta un consumo de corriente de 15 mA durante un tiempo de un segundo, lo que resulta en una carga de $8,33\ \mu Ah$ como se muestra en la ecuación 6.8. Esta carga no se tuvo en cuenta en los cálculos del consumo total de la sección 6.3 ya que es despreciable frente al consumo diario del sistema.

$$\frac{15\ mA \times 1\ s \times 2}{3600\ s/h} = 8,33\ \mu Ah \tag{6.8}$$

Capítulo 7

Conclusiones y trabajo futuro

Resumen

En este capítulo se exponen las conclusiones y reflexiones referentes al desarrollo del proyecto. Se realiza un análisis de las metas alcanzadas y se describen posibles ampliaciones o mejoras que se deberían realizar en un futuro.

Contenido

7.1. Conclusiones generales

Se desarrolló un prototipo funcional de un sistema de detección de plagas, específicamente *Cydia Pomonela* y *Cydia Molesta* en cultivos frutícolas, basado en tecnologías WSN y capaz de tener una

autonomía energética mayor a 28 meses para una comunicación punto a punto con baterías AA de 2800 mAh. Para alcanzar estos resultados se dividió el problema en cuatro distintos bloques de acuerdo a su funcionalidad: hardware, software, procesamiento de imágenes y red.

En la etapa de selección de hardware se evaluaron distintas plataformas de nodos de WSN, cámaras y otros componentes como switches y LEDs. El análisis de los distintos artículos sobre aplicaciones similares fue fundamental a la hora de establecer un diagrama de bloques hardware que cumpliera con los requisitos funcionales del proyecto. El hardware quedó compuesto por un nodo CM3000, la cámara Linksprite LS-Y201, dos LEDs de color amarillo, un circuito interfaz con dos switches FPF2004 y baterías AA. Además se diseñó un gabinete en acrílico para proteger al sistema de las inclemencias climáticas y objetos externos. Con el hardware seleccionado, se logró cumplir con todas las especificaciones funcionales del sistema, por lo que se concluye que la elección fue adecuada. El costo total del hardware del prototipo resultó en 180 USD. En la puesta en producción se espera que se puedan disminuir significativamente los costos al tener una economía de escala.

Luego de adquiridos los distintos componentes, se comenzaron a diseñar e implementar los distintos módulos de la aplicación sobre el RTOS Contiki OS. Se comenzó con el *driver* de la cámara, luego la transmisión y recepción por radio y por último el modo de bajo consumo. Se considera acertada la decisión de utilizar un RTOS ya que permite disminuir significativamente el tiempo de desarrollo de las aplicaciones software. En particular la elección de Contiki OS facilitó la implementación de los distintos módulos de la aplicación por tener resuelto los *drivers* de comunicación UART, el sistema de archivos CFS-Coffee, *stack* de red y manejo de modos de bajo consumo. Como contrapartida se destaca el hecho de que no existe bibliografía suficiente acerca de este RTOS, por lo que el funcionamiento de algunos módulos se debe aprender investigando el código fuente del mismo.

Se exploró el uso de algoritmos de procesamiento de imágenes en hardware reducido y se realizó un enfoque innovador, orientado a trabajar sobre la imagen JPEG comprimida en forma local en el nodo, en lugar de procesar de manera centralizada en un PC como es habitual en las WSN. Se logró implementar un decodificador parcial JPEG adaptado a las necesidades de la aplicación, capaz de armar una imagen de menor tamaño que la original formada únicamente por los componentes DC de cada bloque de la imagen y un clasificador lineal Fisher que utiliza la información frecuencial obtenida por el decodificador parcial para detectar y efectuar un recuento de polillas.

Mediante estas técnicas se logra reducir significativamente el consumo energético del sistema y facilita ampliamente el manejo de la red, ya que la transmisión de grandes cantidades de datos supone un gran desafío en WSN. El conteo automatizado de los insectos plagas estaba por fuera de alcance del proyecto y supone un diferencial respecto a los objetivos planteados.

En cuanto al manejo de la red, se estudió en profundidad el estándar IEEE 802.15.4 y se realizó un estudio de los protocolos de RDC y capa de red incluidos en Contiki OS. Se demostró mediante simulaciones y pruebas de laboratorio que el protocolo de RDC con menor consumo energético para la transmisión punto a punto de grandes cantidades de datos sin pérdidas, errores ni duplicados, y en el orden correcto es nullRDC. En capa de red se analizó el *stack* de Rime y se creó una aplicación que permite enviar los datos de forma confiable utilizando la primitiva `Runicast`. Quedó excluido de este análisis, por estar fuera del alcance del proyecto, el escalado de la red y con éste el armado de tablas de ruteo dinámicas, sincronización y otros elementos de control.

Por último se destaca el hecho de que se logró cumplir ampliamente el requerimiento de una autonomía energética, ya que en una comunicación punto a punto, en seis meses de funcionamiento

se consume aproximadamente el 20 % de la carga total de las baterías. Esto deja un buen margen de guarda para trabajos futuros en cuanto a red y recolección de otras magnitudes físicas en la WSN. Para alcanzar los niveles de bajo consumo obtenidos, se tuvieron que introducir nuevas funciones a la aplicación, que lograron reducir considerablemente el consumo base del nodo en modo *sleep* respecto a los valores obtenidos en proyectos anteriores dentro del IIE-FING-UdelaR.

Los resultados logrados no sólo son aplicables a la detección de plagas en cultivos frutícolas sino que abren la posibilidad de una gran cantidad de aplicaciones en las que se requiera la adquisición, procesamiento para detección de patrones y transmisión de imágenes en una WSN de bajo costo y bajo consumo.

7.2. Evaluación de los resultados respecto a los criterios de éxito

En la etapa primaria del proyecto se elaboró un plan de trabajo y se establecieron criterios de éxito. Se analizará el cumplimiento de los mismos.

1. **Construir un prototipo funcional del sistema propuesto, ensayado y caracterizado en condiciones controladas.**

   Se construyó un prototipo funcional que cumple con los requisitos planteados: capturar, procesar y transmitir imágenes sobre una WSN hacia un nodo conectado a un PC. Se realizaron ensayos de caracterización en condiciones de laboratorio del consumo de corriente de los distintos componentes, así como la estimación del consumo de los protocolos de red y ensayos de los distintos módulos del software. Por más que no estaba dentro de lo estipulado, se realizó un ensayo de alcance de las transmisiones y se corroboró que el mismo es de 100 m tal como indica la hoja de datos del TelosB.

   Para facilitar la interacción con el usuario se implementó un *script* de Matlab que almacena y despliega las imágenes recibidas desde el nodo conectado al PC. Se enviaron de manera consecutiva 100 imágenes para probar su funcionamiento, obteniéndose resultados exitosos.

   Restan realizarse en campo algunos ensayos como: efecto de la variación de temperatura, performance de la red ante inclemencias climáticas y interferencia de objetos externos, efecto de la variación de la alimentación, entre otros.

2. **Implementar un algoritmo de procesamiento de la imagen en el nodo y enviar la información obtenida por la radio.**

   El algoritmo de procesamiento implementado se realiza sobre la imagen JPEG comprimida. Se logró programar un decodificador parcial JPEG que almacena los componentes DC de la DCT del canal Y de los bloques (generando una imagen con información reducida), y mediante un algoritmo FLD de reconocimiento de patrones detecta los bloques correspondientes a polillas para luego agruparlos mediante un algoritmo de *labeling*.

   Luego la imagen con información reducida, así como las coordenadas de las polillas detectadas pueden ser enviadas hacia el nodo base. El *script* de Matlab integra toda la información desplegando la imagen JPEG con los puntos detectados marcados por cruces blancas. Sobre las imágenes con polillas reales ensayadas, se obtuvo un conteo muy cercano al esperado. Se cree que entrenando el algoritmo con un banco de imágenes de polillas reales de mayor tamaño se pueden mejorar estos resultados.

3. **Lograr transmitir la imagen del interior de la trampa con polillas sin errores de un nodo a otro conectado en un PC.**

   Como se mencionó en los puntos anteriores, se logró transmitir las imágenes desde un nodo dentro del gabinete colocado en la trampa hacia otro conectado a un PC de manera exitosa. En capa de red se utilizó `Runicast` presente en el stack de protocolos Rime incluido en Contiki OS al que se le agregó control de secuencia y detección de etiquetas para obtener un protocolo orientado a conexión y confiable. A nivel de capa RDC se utilizó el protocolo nullRDC.

4. **Evaluar y determinar si la capa MAC y capas superiores soportan el tráfico de imágenes. Identificar si existen problemas.**

   En la etapa de estudio y análisis de los protocolos de red presentes en el *stack* de Contiki OS se evaluaron particularmente los protocolos RDC donde se concluyó que la mejor alternativa para la transmisión punto a punto de una imagen (aproximadamente 180 paquetes) es utilizar *nullRDC*, ya que se alcanza un mayor *throughput* y un menor consumo energético.

   En lo que respecta a protocolos de red se optó por modificar `Runicast` para tener una calidad de servicio tal que se garantice que se envíen todos los datos sin errores ni duplicados y en el orden correcto.

   Las dificultades en esta área se encuentran en la escalabilidad de la red. Se realizó un análisis exhaustivo para una comunicación punto a punto que es extendible a una comunicación *multi-hop* con un ruteo estático, quedando pendiente la sincronización para garantizar ventanas de transmisión y que no hayan colisiones. Si se quiere hacer una red *mesh* quedan varios aspectos a resolver como implementar protocolos de ruteo dinámico, sincronización y intercambio de comandos de control.

5. **Otros criterios de éxito**

   Respecto a que el tiempo de vida de las baterías sea superior a seis meses, se demostró mediante ensayos que con el prototipo en su estado actual se alcanza en condiciones controladas una autonomía energética mayor a 28 meses con baterías AA de 2800 mAh. Cabe destacar que si bien el manejo de una red *mesh* con protocolos de ruteo dinámico implicará un consumo considerablemente mayor, existe un margen importante respecto al requerimiento inicial de seis meses.

   Resta realizar pruebas de campo para validar el prototipo en lo relativo a la posibilidad de distinguir y contar las polillas manualmente por parte de los entomólogos. En primera instancia con la resolución actual se considera que no habrá inconvenientes para el conteo de las plagas.

7.3. Dificultades y aprendizajes

En el transcurso del proyecto se debieron sortear dificultades de distinta índole, ya sea por falta de conocimiento sobre el problema o por la falta (o desconocimiento) de hardware o software capaces de proveer una solución. Se detallan a continuación las principales dificultades correspondientes a cada etapa.

7.3.1. Selección del hardware

Selección de componentes

En la etapa de selección de hardware la dificultad se centró en encontrar un sistema que interactuara correctamente. Principalmente se buscaron cámaras que pudieran ser controladas por nodos de WSN de bajo costo y bajo consumo.

Otra dificultad, fuertemente arraigada a la selección de los componentes, surgió en torno a la comunicación UART entre el nodo y la cámara. Los nodos utilizados poseen dos interfaces seriales, una reservada para la comunicación con la radio y la flash externa y la otra para la comunicación USB con el PC. Como la cámara se conectó a la segunda, no se pudo depurar directamente desde el PC siendo necesario implementar el *setup* de depuración expuesto.

La dificultad y el tiempo requerido de la selección de hardware fueron subestimadas, lo que se puede atribuir a la falta de experiencia por parte del equipo.

Circuito interfaz

Para armar y depurar el circuito interfaz se requirió un tiempo mayor al esperado. La primera versión del circuito no incluía las resistencias de *pull-down* para mantener los GPIO que controlan los switches a tierra cuando están apagados, así como la usada para descargar la capacidad de la cámara.

Por otra parte, soldar switches con montaje SMD y un empaquetado SOIC no resultó una tarea fácil y requirió una interiorización en las técnicas de soldadura para estos dispositivos.

7.3.2. Diseño e implementación del software

Contiki OS

Trabajar con Contiki OS supuso un aprendizaje mayor en este proyecto. Fue la primera experiencia en el uso de un RTOS por parte del equipo y la falta de manuales específicos dificultó en ciertas etapas la implementación de los distintos módulos. Asimismo se adquirieron conocimientos en el manejo del simulador Cooja para la simulación de la red utilizando los protocolos seleccionados.

Principalmente las mayores dificultades surgieron en torno al manejo de procesos, interacción con CFS-Coffee y los modos de bajo consumo.

Driver de la cámara y depuración

Las dificultades al implementar el *driver* de la cámara surgieron por la falta de una documentación clara y por desconocimiento de cómo depurar la comunicación serial. Los conocimientos

adquiridos para superar esta etapa fueron de gran utilidad para probar el resto del sistema, por lo que culminar el *driver* supuso un hito dentro del proyecto.

7.3.3. Procesamiento de imágenes

Decodificación JPEG

Una dificultad consistió en que las limitaciones del hardware empleado no permiten acceder al archivo completo en forma simultánea, lo que plantea una diferencia radical con los decodificadores preexistentes. Esto nos obligó a implementar una versión original sin basarnos en ningún código de dominio público.

Otra dificultad que surgió a la hora de programar el decodificador fue la necesidad de aprender a usar la herramienta MEX de Matlab para poder compilar código en C dentro del mismo.

Tratamiento de las imágenes

El procesamiento de las imágenes nos planteó el desafío de enfrentarnos a una disciplina en la que ninguno de los integrantes del equipo tiene experiencia formal. Se adquirió un importante grado de conocimiento sobre el análisis lineal y se logró una implementación que funciona en el nodo con resultados prometedores.

7.3.4. Análisis e implementación de la red

Medir la carga consumida en la transmisión de una imagen con cada protocolo de ciclo de trabajo de la radio disponible fue un desafío, ya que se necesitaba medir voltaje en una resistencia por tiempos prolongados (varios minutos) con una frecuencia de muestreo alta para no perder definición y poder analizarlos correctamente. Al no disponer de ningún equipo que permitiera adquirir tantos datos, se optó por dividir el análisis de consumo en dos partes. Primero se tomaron ventanas de tiempo cortas para obtener mayor definición y así poder analizar correctamente el funcionamiento de los protocolos de red. Luego se tomó una ventana de tiempo que cubriera toda la transmisión, perdiendo definición pero obteniendo una perspectiva global de la corriente consumida. Integrando la curva de corriente se pudo obtener una buena aproximación de la carga total consumida durante de la transmisión. Esto mismo aplica a la medida de los perfiles de corriente del sistema para determinar el consumo total.

Otra dificultad a la hora de analizar los protocolos fue que la documentación que provee la comunidad de Contiki OS acerca de sus protocolos de red es bastante pobre. Para analizarlos hubo que aprender a utilizar la herramienta de simulación Cooja y estudiar el código fuente del RTOS.

7.3.5. Modos de bajo consumo

Llevar el nodo al estado de más bajo consumo fue otro de los grandes desafíos que nos enfrentamos ya que su importancia es crucial para alargar la autonomía energética del sistema. El consumo en el modo *sleep* representa alrededor del 82 % del consumo total del sistema, por lo que es imprescindible minimizarlo.

En proyectos de fin de carrera anteriores el IIE-FING-UdelaR que trabajaron con la misma plataforma habían logrado obtener un consumo base en modo *sleep* de 243 μA, lo cual era inadmisible para los requerimientos funcionales de este proyecto. Para disminuir este consumo, se tuvo que estudiar profundamente los componentes del nodo CM3000 y tratar de apagar o llevar cada uno de ellos a un estado de mínimo consumo. Como en la comunidad de Contiki OS no había información acerca de este tema, se tuvo que programar a bajo nivel y escribir directamente en los registros del MCU para lograr el objetivo planteado. El resultado fue por demás positivo, ya que se logró reducir el consumo base a 40 μA, más de seis veces menor que el valor de la referencia que se tenía. El procedimiento de llevar el nodo CM3000 al modo de más bajo consumo será un legado que podrán utilizar proyectos futuros evitando pasar por las dificultades mencionadas.

7.4. Trabajo futuro

Durante el transcurso del proyecto surgieron muchos puntos e ideas a analizar que quedaron fuera del alcance del mismo por tratarse de un proyecto exploratorio de tecnologías en desarrollo. En esta sección se pretende analizar dichos conceptos y dejar planteado el posible trabajo a futuro.

7.4.1. Selección del hardware

Si bien el hardware del proyecto se eligió cuidadosamente para poder cumplir con todos los requerimientos funcionales, al estar trabajando con tecnologías en constante desarrollo, surgen nuevas plataformas constantemente que pueden mejorar el desempeño del sistema en cuanto a consumo, capacidad de procesamiento, alcance de la radio, etc.

Los resultados de bajo consumo del sistema permiten pensar que es posible utilizar energía solar mediante el uso de paneles fotovoltaicos para alimentar el sistema. El uso de energías renovables no solo puede aumentar la autonomía energética del sistema sino que también contribuye a disminuir la contaminación producida por las baterías.

Para el manejo de la alimentación de la cámara y los LEDs se utilizaron dos switches independientes. En una etapa posterior se podría simplificar el diseño y la implementación utilizando un solo switch con dos salidas.

El prototipo de gabinete desarrollado se realizó teniendo en cuenta los requerimientos de los productores pero no fue testeado intensivamente. Queda como trabajo a futuro el testeo del prototipo de gabinete en las condiciones de trabajo reales, así como la realización de ensayos de normas IP [85]. En función de estos estudios pueden surgir imprevistos que requieran alguna modificación en el diseño de la trampa.

7.4.2. Diseño e implementación del software

Al tratarse de un prototipo, aún resta una etapa de depuración de la aplicación de software. Una vez validado el algoritmo de procesamiento de imágenes se puede modificar el sistema para que envíe solamente el conteo automático de polillas. Esta funcionalidad permite aumentar significativamente la autonomía del nodo y simplificar la implementación de la red. Otra opción consiste en enviar imágenes únicamente en los casos en que el recuento de polillas supere cierto umbral. Eventualmente, luego de una larga validación de la técnica se podría pasar a prescindir totalmente del envío de paquetes diarios, enviando alarmas en el caso en que sea necesario.

Queda pendiente considerar todos los casos de contingencia frente a los errores que puedan surgir en los distintos módulos de la aplicación.

Otro aspecto a considerar es la base de datos de información. El nodo base debe ser capaz que almacenar todas las imágenes recibidas, así como guardar de forma ordenada y eficiente el conteo automático de cada trampa. Una opción es que el nodo *sink* se conecte a una base de datos remota por Internet y almacene en la nube toda la información.

Al estudiar el consumo de corriente en modo *sleep* del sistema se encontró que en promedio éste corresponde a más del doble del valor de la base de consumo debido a la presencia de los *ticks* del RTOS cada 8 ms. Teniendo esto en cuenta puede resultar interesante implementar un *Timer Sin Tick* (TST) con el objetivo de actualizar el *timer* sin la necesidad de despertar el MCU cada 8 ms.

7.4.3. Procesamiento de imágenes

El decodificador JPEG se desarrolló como una herramienta para poder acceder a la información codificada en JPEG, por lo que consideramos que no abre demasiadas posibilidades de por sí. Sin embargo su capacidad de generar una imagen comprimida quedó obsoleta al comprobarse la capacidad del detector de contar las polillas de forma fiable por lo que sería interesante eliminar esta función en virtud de la eficiencia de la ejecución. En el caso de utilizar una cámara diferente a la Linksprite LS-Y201 puede ser necesario revisar la implementación ya que es sumamente específica.

El algoritmo de detección de plagas se ideó sobre la premisa de que solamente se utilizaría la información de luminancia extraída del archivo JPEG, sin embargo puede resultar interesante agregar información proveniente de los canales Cb y Cr al discriminante en la forma de una nueva dimensión del espacio de características. Se sugiere encontrar una dirección de proyección óptima para distinguir las polillas del fondo en el espacio (Cb,Cr), y luego agregar una dimensión al espacio de características que consista en la proyección sobre esa recta de las componentes de continua de Cb y Cr de cada cuadro.

Quedan pendientes la validación del algoritmo con imágenes tomadas en el terreno y su correspondiente entrenamiento, pero las simulaciones arrojan resultados sumamente esperanzadores.

7.4.4. Análisis e implementación de la red

Los protocolos de red analizados se concentraron en cómo enviar punto a punto una imagen entre dos nodos de la forma más eficiente posible. Si bien este era un desafío importante al comienzo del proyecto queda trabajo para hacer con respecto al resto de la comunicación.

En primera instancia se deberían resolver los protocolos de red a utilizar, tanto para el ruteo como para la transmisión de datos. Con los avances realizados en la detección automática es de esperar que sea sencillo adaptar los protocolos `RPL` [86] y `Collect Tree` [87] incluidos en Contiki OS para esta aplicación.

Por otra parte está el problema de la sincronización de la red. En el proyecto se plantea enviar la imagen y luego permanecer en un estado de bajo consumo por 12 o 24 horas hasta el envío de una nueva imagen. En este caso, hay una etapa de sincronización a resolver cuando se despierten los nodos ya que pueden no despertarse todos al mismo tiempo exacto.

Otro aspecto a resolver es la escalabilidad de la red. Si se quiere trabajar con una cantidad de nodos considerablemente mayor se debe cambiar el ruteo estático por dinámico. Si se hace un ruteo dinámico, también hay una etapa de armado de la red y tablas de ruteo a resolver, para la cual se deben seguir estudiando distintos protocolos de red.

7.4.5. Caracterización y ensayos

Queda como trabajo futuro explicar y reducir los 40 μA de consumo base del nodo cuando el mismo está en modo *sleep*. A su vez se deberán caracterizar los protocolos estudiados al escalar la red para las distintas topologías posibles. Se deberán corroborar los resultados frente a variaciones en la tensión de alimentación y temperatura. También quedan pendiente hacer los ensayos de campo correspondientes.

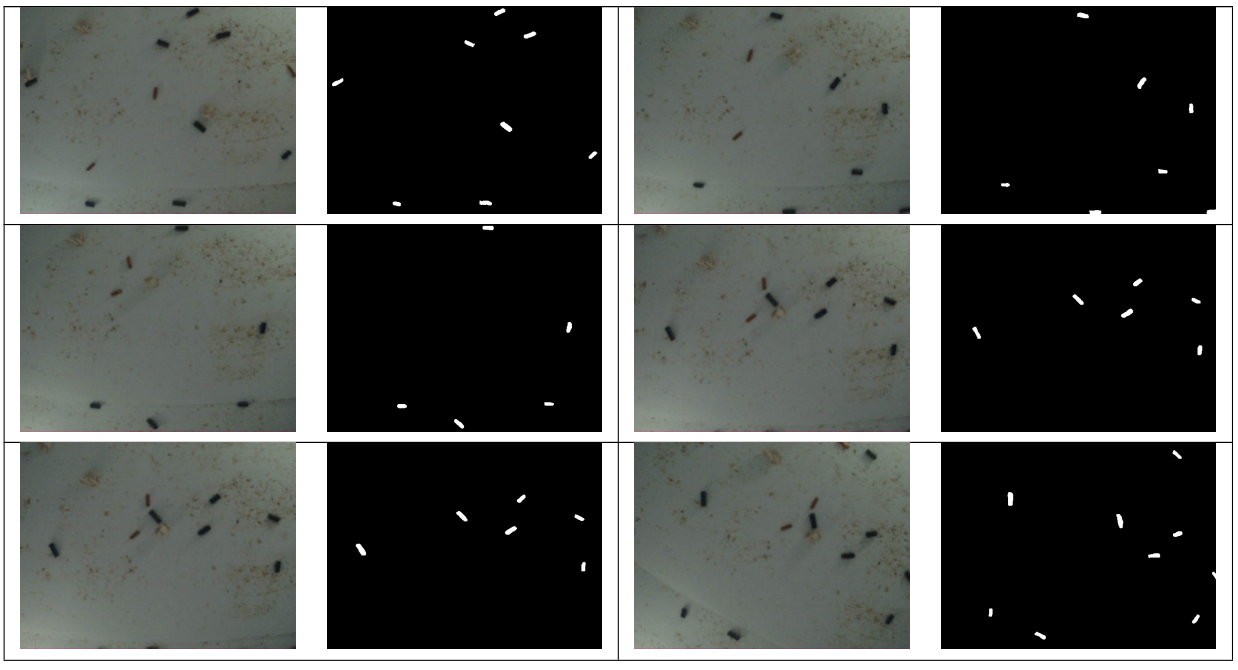
Apéndice A

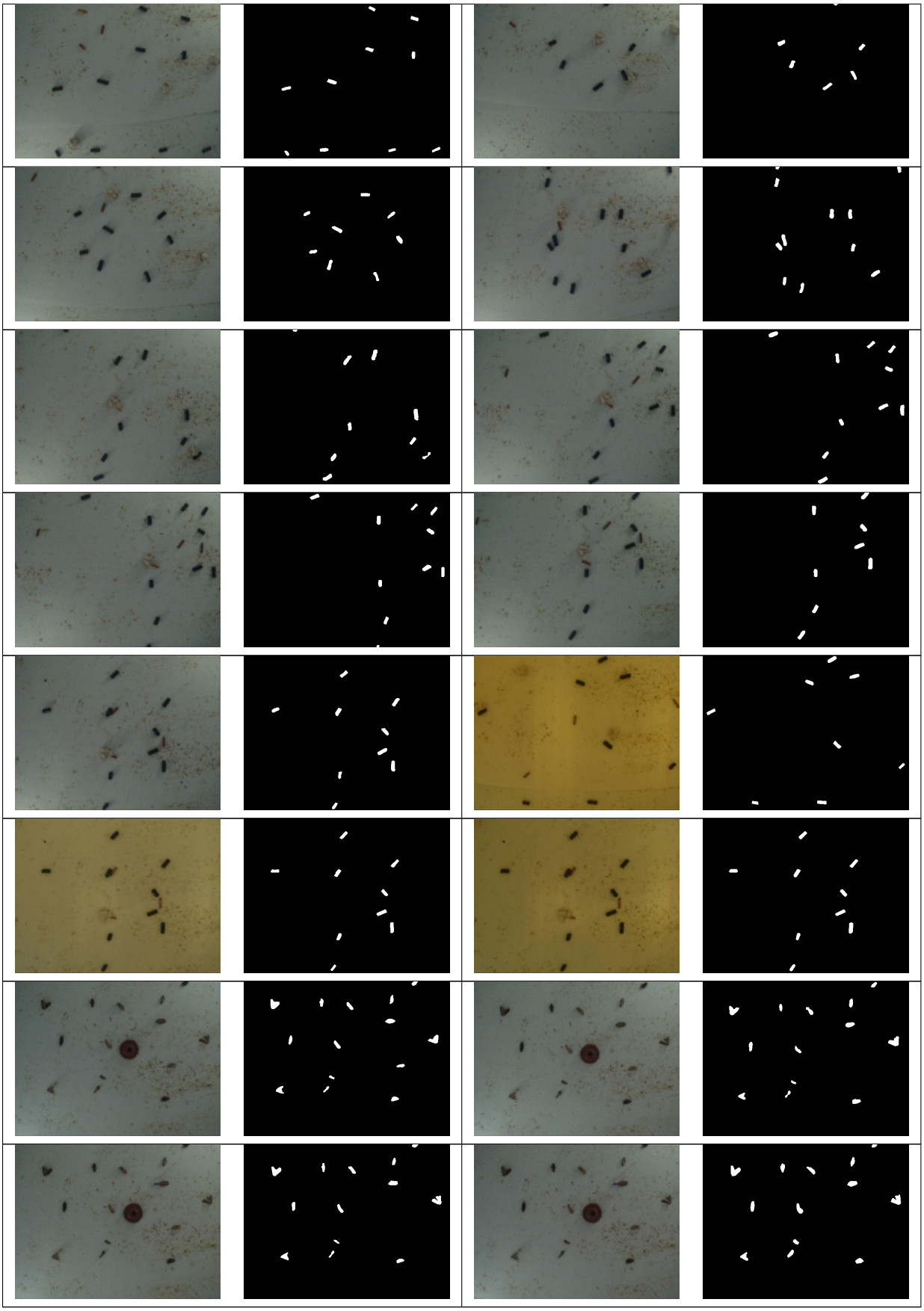
Set de entrenamiento del algoritmo

A.1. Introducción

En este anexo se presentan en detalle los sets de entrenamiento y validación del algoritmo FLD implementado. También se muestran las matrices de confusión para las imágenes de validación en el umbral considerado óptimo y luego se muestra una comparación entre las máscaras y lo determinado por el algoritmo.

A.2. Set de entrenamiento





A.3. Set de validación

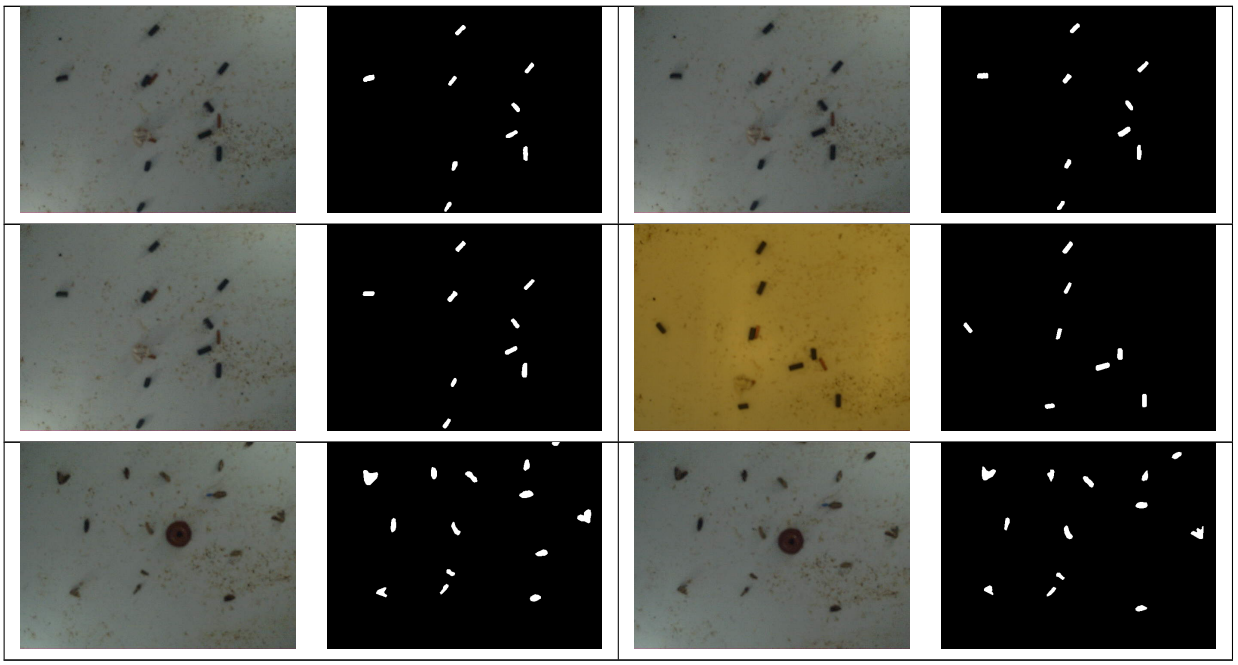


Tabla A.2: Imágenes de validación.

A.4. Matrices de confusión

Imagen	TP	FN	TN	FP
1	20	1	4753	26
2	30	0	4758	12
3	21	3	4764	12
4	23	2	4770	5
5	46	12	4669	73
6	39	7	4676	78
TOTAL	179	25	28390	206
	87,7 %	12,3 %	99,3 %	0,7 %

A.5. Resultados

La tabla A.3 muestra los reslutados de aplicar el algoritmo de detección sobre las imágenes de la tabla A.2. Cada imagen está compuesta por cuatro cuadrantes:

- El superior izquierdo muestra la imagen original con los puntos detectados superpuestos.
- El superior derecho muestra la máscara generada manualmente, que se considera como realidad.
- El inferior derecho muestra la máscara generada a partir de las coordenadas detectadas.
- El inferior izquierdo muestra ambas máscaras superpuestas, lo que en definitiva equivale a la matriz de confusión. Los verdaderos positivos aparecen en blanco, los verdaderos negativos en negro, los falsos positivos en rojo y los falsos negativos en verde.

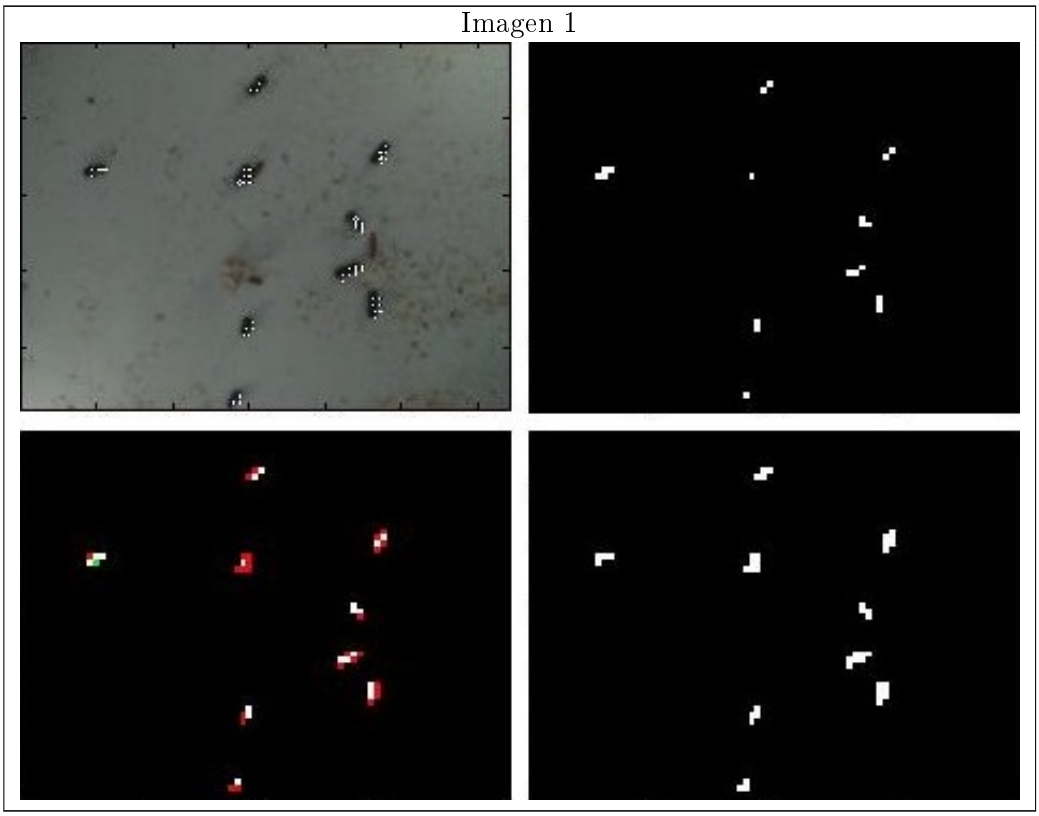


Imagen 2

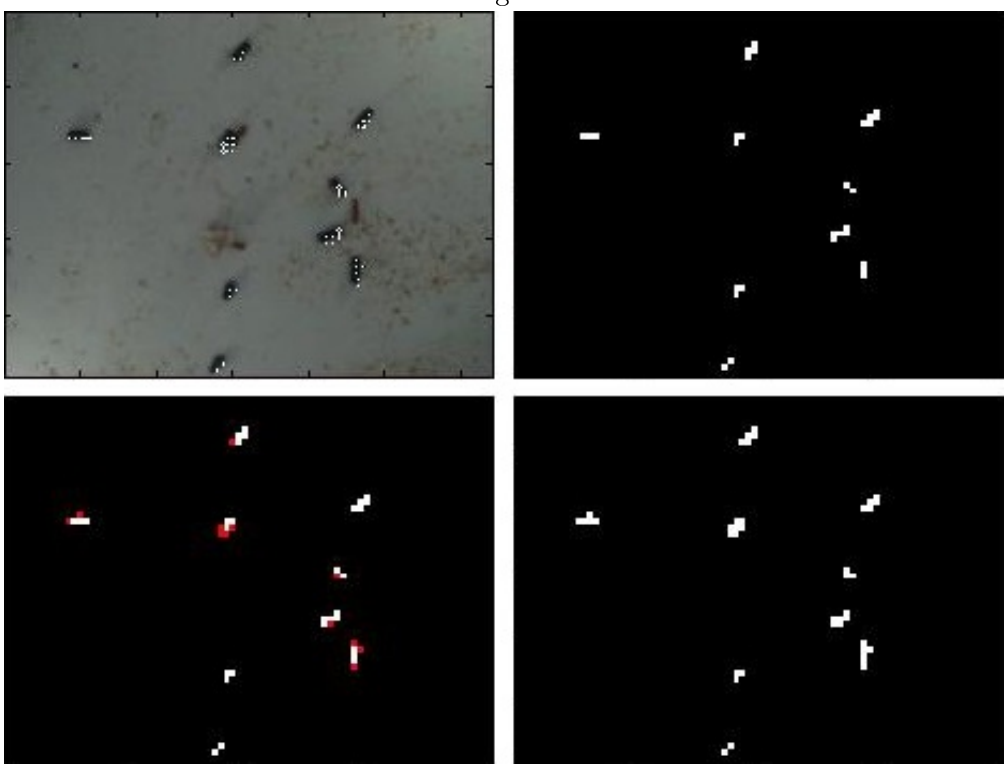


Imagen 3

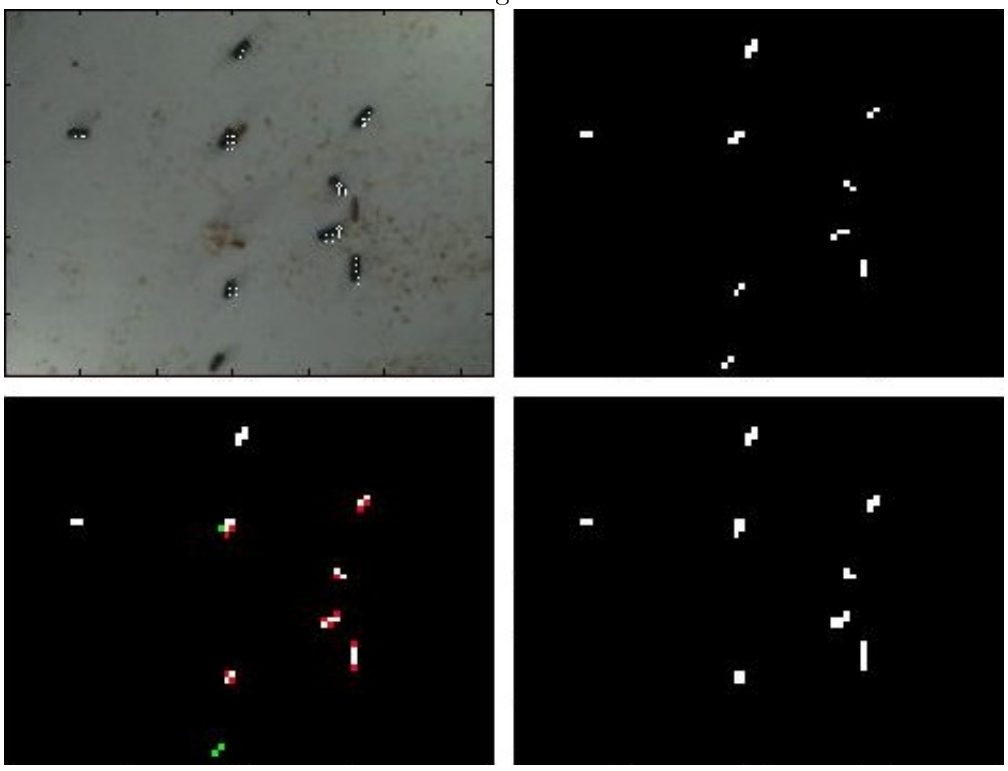


Imagen 4

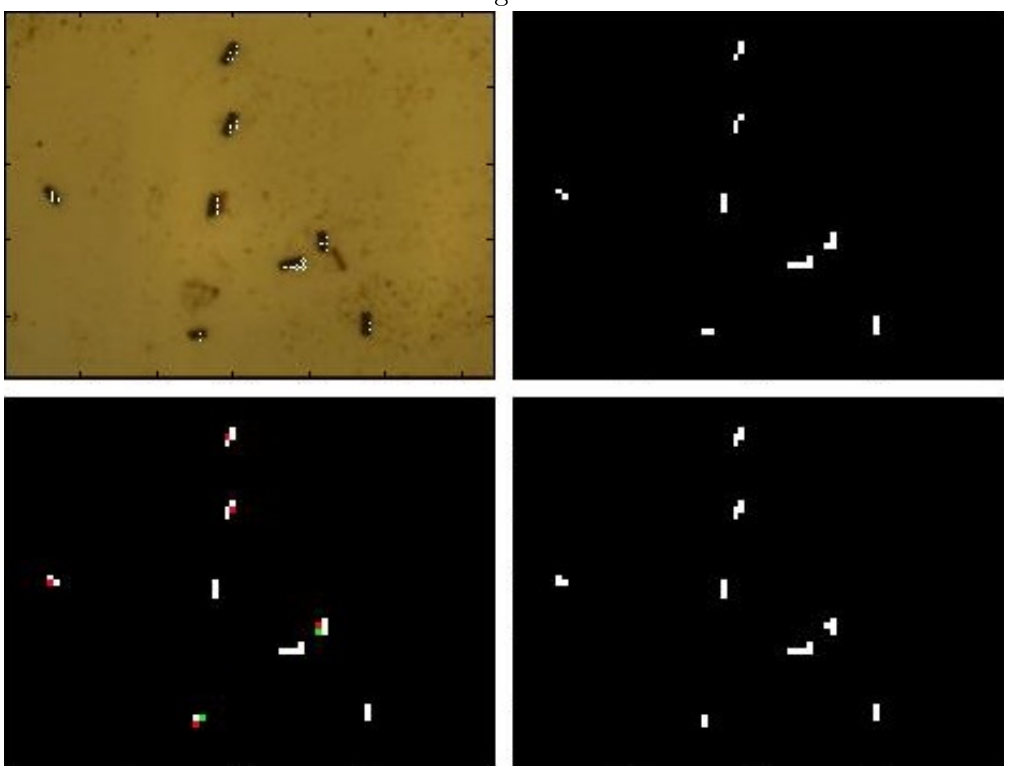
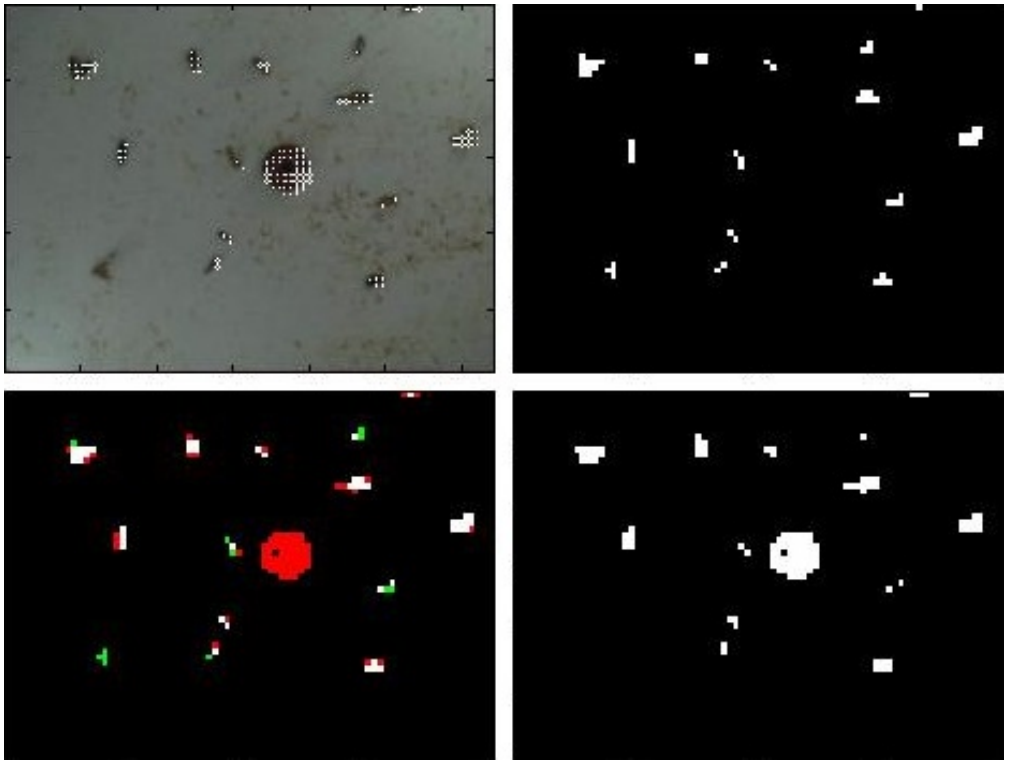


Imagen 5



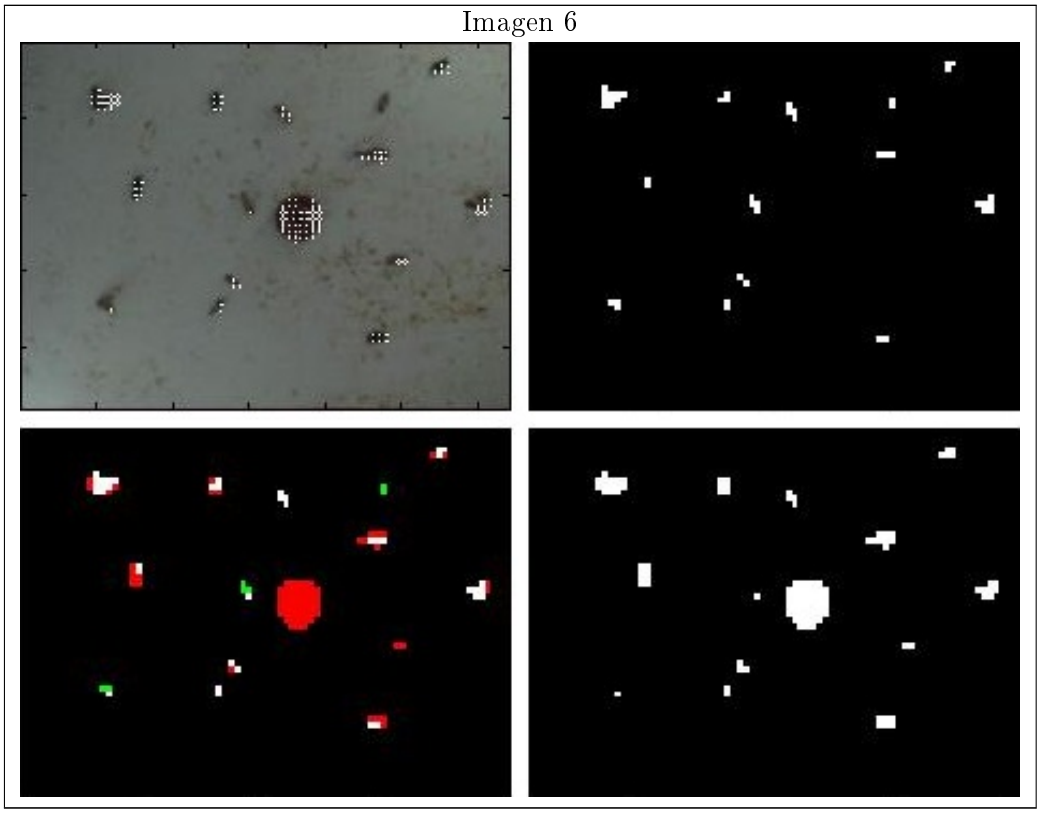


Tabla A.3: Resultados de aplicar el algoritmo de clasificación al set de validación

Apéndice B

Compilación y depuración

En este anexo se detallan los archivos necesarios para la compilación del código y la carga hacia la plataforma. El lector podrá encontrar cómo se debe configurar el archivo `Makefile` y capturas de pantalla del software *Sersniff* y *GTKTerm* para ejemplificar la depuración.

B.1. Compilación

Los archivos `Makefile` y `project-conf.h` especifican configuraciones y códigos del proyecto para que el compilador genere el binario correctamente. En el caso de Contiki OS se modulariza la configuración con el archivo `Makefile.include`, en el cual se incluyen todos los módulos que se quieren incluir del sistema operativo y otras directivas para el compilador. Este último archivo es incluido dentro del `Makefile`.

El archivo `project-conf.h` permite cambiar banderas y definiciones en la configuración de Contiki OS para un proyecto particular, evitándose la manipulación del código fuente. El `Makefile` contiene los siguientes campos:

```
 #El nombre del proyecto a compilar
CONTIKI_PROJECT= plagavision
#La direccion donde se encuentran los archivos de Contiki
CONTIKI = ../..
#Para agregar un archivo de configuracion del proyecto
CFLAGS += -DPROJECT_CONF_H=\ “project-conf.h\ ”
#Agregar archivos C para compilar
PROJECT_SOURCEFILES += ejemplo.c
#Indica que hay que compilar cuando se ejecuta el comando make
all: $(CONTIKI_PROJECT)
# Incluir el Makefile.include
include $(CONTIKI)/Makefile.include
```

Cabe destacar que para poder compilar la biblioteca `math.h` es necesario agregar `TARGET_LIBFILES+= -lm` en este archivo de configuración.

Finalmente para compilar el código se ejecuta el comando `make nombre_del_proyecto [TARGET]`. La etiqueta `TARGET` debe ser sustituía por el tipo de plataforma sobre la cual se quiere compilar (en nuestro caso plataformas `sky`).

Existe la opción de guardar el `TARGET` para no tener que declararlo constantemente ejecutando `make TARGET=plataforma savetarget`. Mediante este comando se crea un archivo llamado `Makefile.target` el cual es leído por el `Makefile` al ejecutarse un `make`.

Se destaca además que al modificarse un archivo de encabezados (un `archivo.h`) se debe ejecutar el comando `make clean` *`nombre_del_proyecto`* para que la compilación incluya las modificaciones.

B.2. Cargar la aplicación en la plataforma

Para cargar el código en la plataforma se debe ejecutar el comando `make` *`nombre_del_proyecto`*`.upload [MOTE]`. Este establece una comunicación serial con la plataforma y carga el archivo con con el código de máquina correspondiente.

La directiva `MOTE` indica qué plataforma utilizar en el caso que se tenga más de una conectada. Esta directiva cambia únicamente el puerto al que se desea acceder para cargar el programa. La asignación de puertos y plataformas se puede observar mediante el comando `make motelist`.

B.3. Depuración serial

Para depurar los códigos se utilizaron los programas *Sersniff* y *GTKTerm*, los cuales facilitaron la visualización de la comunicación serial entre los dispositivos. Para la etapa de depuración en la que se utilizó la configuración *Camara* ⇔ Arduino ⇔ PC ⇔ CM5000, la comunicación se estableció con el comando `./sersniff -i /dev/ttyUSB0 -o /dev/ttyACM0 -x -b115200 -z -f "%02hX s`. En la figura B.1 se observa una captura de pantalla de la comunicación. El puerto 1 corresponde al nodo, mientras que el puerto 2 a la cámara. Las salidas se muestran en notación hexadecimal.

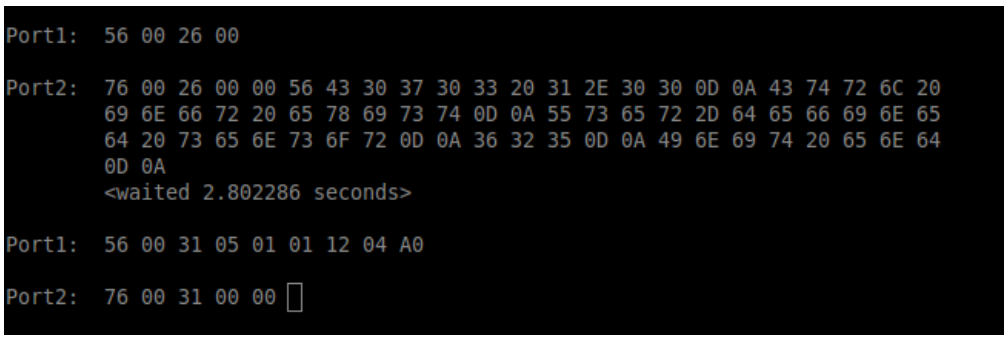


Figura B.1: Captura de pantalla de la salida del Sersniff

En la figura B.2 se observa una captura de pantalla de la salida del *GTKTerm* donde se pueden

ver los datos en ASCII y en hexadecimal. La interfaz gráfica permite que la configuración sea sencilla.

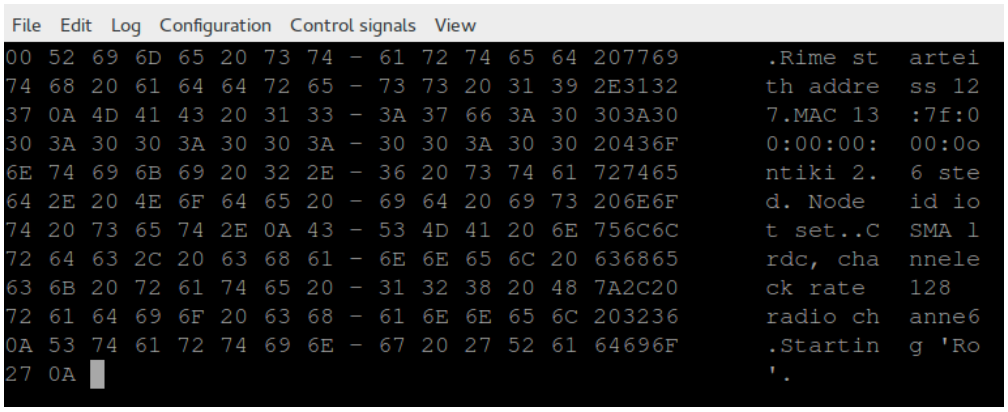


Figura B.2: Captura de pantalla de la salida del GTKTerm

Apéndice C

Contenido del CD

C.1. Estructura del CD

A continuación se encuentra el árbol de directorios correspondiente al CD que se adjunta a la presente documentación.

```
/
├── Códigos
├── Documentación
├── Medidas
└── Procesamiento de imágenes
```

C.2. Descripción de los archivos

En esta sección se describe el contenido de los distintos directorios y se describen los archivos incluidos en el CD.

La carpeta `Códigos` incluye los archivos fuente de la aplicación separados en los distintos módulos: `Nodo, Camara, JPEG, Radio y Sleep`. En `Nodo` se encuentran los archivos `plagavision.c`, `nodo_router.c` `Makefile` y `project-conf.h` que corresponden al módulo central, el módulo probado para realizar transmisiones *multihop*, el archivo configuración de la compilación y el de configuraciones particulares del proyecto respectivamente.

En `Camara` se encuentran los archivos `camara.c` y `camara.h` correspondientes al módulo que implementa el *driver* de la cámara Linksprite LS-Y201. La carpeta `JPEG` incluye los archivos `jpeg.c` y `jpeg.h` respectivos al algoritmo de procesamiento de las imágenes JPEG y clasificador lineal Fisher.

En la carpeta `Radio` se encuentran los archivos `radio.c` y `address.h`, que contienen el proceso que implementa el manejo de la red y las direcciones Rime de los nodos respectivamente. Por último la carpeta `Sleep` contiene el archivo `sleep.c` que contiene las funciones para el manejo de los modos de bajo consumo.

En `Documentación` se puede encontrar el archivo correspondiente a la presente documentación, llamado `tesis.pdf`.

Luego, en `Medidas` se encuentran los archivos de datos capturados con el osciloscopio en formato CSV. Todas las medidas corresponden a la caída de tensión en una resistencia de $10,2\ \Omega$ en serie con el sistema.

Por último en `Procesamiento de imágenes` se pueden encontrar los bancos de imágenes con los que se entrenó el algoritmo FLD, así como los códigos del entrenador y la implementación del decodificador parcial JPEG para Matlab.

C.3. Requerimientos del sistema

Para una correcta prueba de los códigos es necesario tener instalado el *toolchain* de Contiki OS[1]. Para la correcta visualización de los demás archivos otros requerimientos necesarios son:

- Ubuntu 13.04 o posterior.
- Lectora de CD-ROM.
- Visor de PDF.

[1] el cual se puede obtener en https://github.com/contiki-os/contiki/wiki/Setup-contiki-toolchain-in-ubuntu-13.04

Referencias

[1] MGAP. MGAP – Sistema PRONOS (PRONOSTICOS FITOSANITARIOS) - Integración de capturas prediales en el Pronóstico Fito-sanitario para el Gusano de la Pera y la Manzana y para el Gusano del Durazno y del Membrillo. [Online]. Disponible: http://www.mgap.gub.uy/dgssaa/DivOperaciones/Doper_Serfitos_PF_Pronos.html Citado en página 2.

[2] Facultad de Agronomía, UdelaR. Cydia Pomonella (Linneo). [Online]. Disponible: http://www.pv.fagro.edu.uy/fitopato/SSD/Insectos/Cydia/Carpocapsa.html Citado en página 2.

[3] I. B. Scatoni y S. Núñez, "Estrategias apropiadas para el control de Grafolita en montes de durazneros y membrilleros." Montevideo, Uruguay: Facultad de Agronomía, 2008. Citado en página 2.

[4] "Isomate®-C PLUS," Pacific Biocontrol Corporation, Vancouver, WA 98685 USA. [Online]. Disponible: http://www.pacificbiocontrol.com/Pacific_Biocontrol_Corporation/Brochures_&_Tech_Sheets_files/CPLUSSpanish.pdf Citado en página 2.

[5] P. Mazzara, L. Steinfeld, J. Villaverde, F. Silveira, G. Fierro, A. Otero, C. Saravia, N. Barlocco, P. Vergara, y D. Garín, "Despliegue y Depuración de Redes de Sensores Inalámbricos para Aplicaciones al Agro," *RPIC*, pp. 690–695, 2011. Citado en páginas 3 y 24.

[6] P. Tirelli, N. Borghese, F. Pedersini, G. Galassi, y R. Oberti, "Automatic monitoring of pest insects traps by Zigbee-based wireless networking of image sensors," *I2MTC-IEEE*, pp. 1–5, 2011. Citado en páginas 4 y 66.

[7] *Telos datasheet*, moteiv, 2004. Citado en páginas 4, 14 y 109.

[8] *CC2420 Datasheet*, Chipcon, Jun 2004. Citado en páginas 4 y 109.

[9] *IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE, 2013. Citado en página 4.

[10] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. Citado en página 4.

[11] J. Lloret, I. Bosch, S. Sendra, y A. Serrano, "A Wireless Sensor Network for Vineyard Monitoring That Uses Image Processing," *Sensors*, vol. 11, pp. 6165–6196, 2011. [Online]. Disponible: www.mdpi.com/journal/sensors Citado en páginas 4 y 66.

[12] F. Aguerre, M. J. González, y I. Rodríguez, "DIAMETRONCO, Desarrollo de un dendrómetro prototipo para su aplicación en Redes de Sensores Inalámbricos." Montevideo, Uruguay: IIE, FING, UdelaR, 2013. Citado en páginas 4, 24 y 111.

[13] P. M. Pérez, F. P. Arbiza, y L. P. Urgoiti, "Redes de Sensores Inalámbricos basado en el estándar 802.15.4." Montevideo, Uruguay: IIE, FING, UdelaR, 2009. Citado en página 4.

[14] T. C. Community. Contiki - The Open Source OS for the Internet of Things. [Online]. Disponible: http://www.contiki-os.org/ Citado en páginas 7 y 44.

[15] M.Gonzalez, L. Barboni, J.Schandy, N.Wainstein, A.Gomez, y C.Croce, "Wireless image-sensor network application for population monitoring of lepidopterous insects pest (moths) in fruit crops," en *IEEE International Instrumentation and Measurement Technology Conference*, May 2014. Citado en página 8.

[16] Advanticsys, *CM3000 Datasheet*, 2010. Citado en página 11.

[17] *MSP430 x1xx Family User's Guide*, Texas Instruments, 2006. Citado en página 11.

[18] *LinkSprite JPEG Color Camera Serial UART Interface*, Linksprite, 2012. Citado en páginas 11 y 27.

[19] *FPF2000-FPF2007 IntelliMAX Advanced Load Management Products*, Fairchild Semiconductor, 2008. Citado en página 11.

[20] W. Dargie y C. Poellabauer, *FUNDAMENTALS OF WIRELESS SENSOR NETWORKS: THEORY AND PRACTICE.* The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom: Wiley, 2010. Citado en página 14.

[21] *MSP430F15x, MSP430F16x, MSP430F161x datasheet*, Texas Instruments, 2011. Citado en página 15.

[22] *Micron M25P80 Serial Flash Embedded Memory Datasheet*, Micron, 2011. Citado en página 15.

[23] *USB1000 Datasheet*, Advanticsys, 2010. Citado en página 16.

[24] *EX1000 Datasheet*, Advanticsys, 2010. Citado en página 16.

[25] *MAX232,MAX232I DUAL EIA-232 DRIVERS/RECEIVERS*, Texas Instruments, 1989. Citado en página 16.

[26] Advanticsys. CM3000 Advanticsys. [Online]. Disponible: http://www.advanticsys.com/shop/mtmcm3000msp-p-6.html Citado en página 16.

[27] *UM1050 User manual, STM32W-RFCKIT RF control kit for STM32W108xx microcontrollers*, ST Microelectronics, 2012. Citado en página 18.

[28] *STM32W108xx Datasheet*, ST Microelectronics, 2013. Citado en página 18.

[29] *Cortex-M3 Technical Reference Manual*, ARM, 2006. Citado en página 18.

[30] S. Microelectronics. STM32W series. [Online]. Disponible: www.st.com/stm32w Citado en página 18.

[31] Arduino. Arduino Due. [Online]. Disponible: http://arduino.cc/en/Main/arduinoBoardDue Citado en página 19.

[32] *AT91SAM Datasheet*, ATMEL, 2012. Citado en página 19.

[33] Arduino. Arduino Due. [Online]. Disponible: http://store.arduino.cc/ Citado en página 19.

[34] *UM1472 User manual, STM32F4DISCOVERY*, ST Microelectronics, 2012. Citado en página 19.

[35] *STM32F405xx, STM32F407xx Datasheet*, ST Microelectronics, 2013. Citado en página 19.

[36] S. Microelectronics. STM32F4DISCOVERY. [Online]. Disponible: http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419 Citado en página 20.

[37] I. de Mula, G. Ferrari, y G. Firme, "ContikiWSN. Estudio, Análisis y Diseño de Redes de Sensores Inalámbricas con Contiki OS." Montevideo, Uruguay: IIE, FING, UdelaR, 2011. Citado en página 24.

[38] M. González, N. Lamath, y N. Wainstein, "Proyecto Final Curso Sistemas Embebidos para Tiempo Real." Montevideo, Uruguay: Sistemas Embebidos, IIE, Fing, UdelaR, 2013. Citado en páginas 24 y 66.

[39] *MT9D111 - 1/3.2-Inch 2-Megapixel SOC Digital Image Sensor Features*, Micron Technology Inc., 2004. Citado en página 26.

[40] Arducam. Arduino Based Camera. [Online]. Disponible: http://www.arducam.com/ Citado en página 26.

[41] Ebay. 2MP Camera Module MT9D111. [Online]. Disponible: www.ebay.com Citado en página 26.

[42] UCTRONICS. Arduino Camera Shield Arducam. [Online]. Disponible: http://www.uctronics.com/arduino-camera-shield-arducam-for-uno-mega2560-board-p-1445.html Citado en página 26.

[43] *OV7670/OV7171 CAMERA CHIP image sensor*, Omnivision, 2006. Citado en página 26.

[44] *AL422 Data Sheets*, AVERLOGIC, 1999. Citado en página 26.

[45] Ebay. OV7670+AL422 FIFO. [Online]. Disponible: www.ebay.com Citado en página 26.

[46] *uCAM Serial JPEG Camera Module Data Sheet*, 4DSYSTEMS, 2011. Citado en página 27.

[47] Mouser Electronics. 4D Systems uCAM TTL. [Online]. Disponible: www.mouser.com Citado en página 27.

[48] Linksprite. Infrared JPEG Color Camera Serial UART (TTL level). [Online]. Disponible: http://store.linksprite.com/infrared-jpeg-color-camera-serial-uart-ttl-level/ Citado en página 28.

[49] *SIM900 Hardware Design*, SIMCom, 2009. Citado en página 31.

[50] Ebay. SIMCOM SIM900 Quad-band GSM/GPRS Module. [Online]. Disponible: www.ebay.com Citado en página 31.

[51] *XBee Multipoint RF Modules*, Digi, 2011. Citado en página 31.

[52] Olimex. XBee Pro 60mW Wire Antenna. [Online]. Disponible: http://www.olimex.cl/ Citado en página 32.

[53] *ENERGIZER L91 Ultimate Lithium Product Datasheet*, Energizer, 2013. Citado en página 34.

[54] "Performance parameters of RTOSs; comparison of open source RTOSs and benchmarking techniques," *2013 International Conference on Advances in Technology and Engineering (ICATE)*, pp. 1–6, Ene. 2013. Citado en página 42.

[55] T. V. Chien, H. N. Chan, y T. N. Huu, "A Comparative Study on Operating System for Wireless Sensor Networks," en *2011 International Conference on Advanced Computer Science and Information System (ICACSIS)*, Dec. 2011, pp. 73–78. Citado en página 43.

[56] D. Gay, "Matchbox: A simple filing system for motes." Citado en página 43.

[57] D. Moss, "Blackbook File System Library." Citado en página 43.

[58] "BLIP 2.0," Standford University. [Online]. Disponible: http://tinyos.stanford.edu/tinyos-wiki/index.php/BLIP_2.0 Citado en página 43.

[59] "TinyRPL," Standford University. [Online]. Disponible: http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyRPL Citado en página 43.

[60] P. Buonadonna, J. Hill, y D. Culler, "Active Message Communication for Tiny Networked Sensors," Infocom, Tech. Rep., 2001. Citado en página 43.

[61] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," SICS, Tech. Rep., 2007. Citado en página 43.

[62] N. Tsiftes, A. Dunkels, Z. He, y T. Voigt, "Enabling Large-Scale Storage in Sensor Networks with the Coffee File System," SICS, Tech. Rep., 2009. Citado en página 43.

[63] A. Dunkels, "Rime — A Lightweight Layered Communication Stack for Sensor Networks," SICS, Tech. Rep., 2007. Citado en página 43.

[64] Advanticsys, *CM5000 Datasheet*, 2010. Citado en página 47.

[65] J. McDowell, "Sersniff 0.0.5," 2011. [Online]. Disponible: http://www.earth.li/projectpurple/progs/sersniff.html Citado en páginas 49 y 63.

[66] M. Bertrán y N. Martínez. Procesamiento de imágenes en el dominio comprimido. [Online]. Disponible: http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2013/jpeg/ClasificadorFinal.html Citado en página 65.

[67] J. A. Gutiérrez, E. H. Callaway, y R. L. Barrett, *Low Rate Wireless Personal Area Networks*. Standards Information Networks, IEEE Press, 2003. Citado en página 66.

[68] I. Downes, L. B. Rad, y H. Aghajan, "Development of a Mote for Wireless Image Sensor Networks." Stanford, CA: Stanford University. Citado en página 66.

[69] A.A.Shahidan, N.Fisal, A.H.A.H.Fikri, N.-S. N.Ismail, y F. Yunus, "Image Transfer in Wireless Sensor Networks." Malasia: Universiti Teknologi Malaysia. Citado en página 66.

[70] Extremely memory efficient embedded JPEG decompressor. [Online]. Disponible: https://code.google.com/p/picojpeg/ Citado en páginas 67 y 82.

[71] A compact JPEG decoder. [Online]. Disponible: http://keyj.emphy.de/nanojpeg/ Citado en página 67.

[72] Mathworks. Write C/C++ functions using MATLAB® API libraries. [Online]. Disponible: http://www.mathworks.com/help/matlab/write-cc-mex-files.html Citado en página 68.

[73] D. Salomon, *Data Compression.* Springer, 2007. Citado en páginas 68 y 69.

[74] C. Hass. JPEG Huffman Coding Tutorial. [Online]. Disponible: http://www.impulseadventure.com/photo/jpeg-huffman-coding.html Citado en página 68.

[75] R. Duda, P. Hart, y D. Stork, *Pattern Classification*, 2nd ed. Wiley, 2001. Citado en páginas 74 y 76.

[76] R. M. Haralick y L. G. Shapiro, *Computer and Robot Vision*, 1st ed. Addison Wesley, 1992, vol. 1. Citado en página 77.

[77] S. University. Signal Detection Theory Handout. [Online]. Disponible: http://www-psych.stanford.edu/~lera/psych115s/notes/signal/ Citado en página 80.

[78] T. Fawcett, "ROC Graphs: Notes and Practical Considerations for Researchers." MS 1143, 1501 Page Mill Road, Palo Alto, CA 94304: HP Laboratories, 2004. Citado en página 81.

[79] M. Buettner, G. V. Yee, E. Anderson, y R. Han, "X-MAC: a short preamble MAC protocol for duty- cycled wireless sensor networks," *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACMSenSys)*, 2006. Citado en página 89.

[80] *Resolución Nro. 35.2000 - Dirección nacional de Comunicaciones. Exp. 9903040/1/1802*, URSEC, 2000. Citado en página 90.

[81] Fakih, J. F. Diouris, y G. Andrieux, "BMAC: beamformed MAC protocol with channel tracker in MANET using smart antennas," en *Wireless Technology, 2006. The 9th European Conference on. IEEE*, May 2006. Citado en página 93.

[82] Moss, David, y P. Levis., "BoX-MACs: Exploiting physical and link layer boundaries in low-power networking." Computer Systems Laboratory Stanford University, Tech. Rep., 2008. Citado en página 93.

[83] El-Hoiydi, Amre, y J.-D. Decotignie, "WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks," en *ISCC 2004. Ninth International Symposium on. Vol. 1. IEEE*, Oct. 2004. Citado en página 93.

[84] An Introduction to Cooja. Contiki git. [Online]. Disponible: https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja Citado en página 99.

[85] "IEC 60529: Degrees of protection provided by enclosures (IP Code)," International Electrotechnical Commission, Tech. Rep. Citado en página 127.

[86] Internet Engineering Task Force (IETF). RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. [Online]. Disponible: http://tools.ietf.org/html/rfc6550 Citado en página 129.

[87] Tree-based hop-by-hop reliable data collection. Contiki git. [Online]. Disponible: http://contiki.sourceforge.net/docs/2.6/a01723.html Citado en página 129.

Índice de tablas

Índice de figuras

Esta es la última página.
Compilado el viernes 6 junio, 2014.
http://iie.fing.edu.uy/