Universidad Zaragoza
1542

# Proyecto Fin de Carrera

## Extensión de funcionalidad en una red visual de procesamiento distribuido: inclusión de BRISK e implementación de un protocolo de retransmisiones

Autor/es

Alberto Aurensanz Clemente

Director/es y/o ponente

György Dán, Laboratory of Communication Networks, KTH, Stockholm
Emiliano Bernués del Río, Ing. Electrónica y de Comunicaciones, EINA, Zaragoza

Escuela de Ingeniería y Arquitectura, Zaragoza
2014

## RESUMEN

Este proyecto parte de un sistema que realiza la extracción de características de imagen de forma distribuida en una red de nodos. Su objetivo es extender la funcionalidad de este sistema. Llevar a cabo la extracción de características de forma distribuida es un proceso exigente que requiere el desarrollo de algoritmos que manejen la asignación y delegación de tareas computacionales a cada nodo, de protocolos encargados de asegurar una transmisión correcta de datos, etc. El sistema que lo lleva a cabo está escrito en C++ y usa los ordenadores de tamaño de tarjeta de crédito BeagleBone Black como nodos y las unidades XStick USB como dispositivos de comunicación inalámbrica. La comunicación entre nodos se lleva a cabo a través de tipos definidos con la sintaxis ASN.1 [1]. La primera parte del proyecto consiste en introducir un nuevo algoritmo de extracción de características de imagen (BRISK [2]) que funcione junto con el que el sistema ya incluía (SURF [3]). El usuario debe ser capaz de poder elegir entre ellos de manera indistinta en el momento de lanzar el programa. La segunda parte del proyecto consiste en modificar los aspectos de transmisión del sistema, incluyendo las clases necesarias para que en la transmisión no se produzcan pérdidas de datos. El protocolo de transmisión usado es una versión del método ARQ Selective Repeat. La evaluación de las prestaciones del sistema se realiza detallando el tiempo necesario para completar cada paso del proceso de extracción de características de imagen, presentando así una comparación entre ambos algoritmos presentes, y por otro lado, calculando el porcentaje de paquetes perdidos y el *throughput* o bits por segundo alcanzables en la transmisión una vez que se ha implementado el método de control de errores.

# Tabla de contenidos

# 1. Introducción

Este proyecto se ha realizado en el departamento Laboratory of Communications Networks, en la universidad Kungliga Tekniska Höskolan de Estocolmo, Suecia, bajo el tutelaje del profesor György Dán. Parte de un sistema ya creado, que es el resultado del trabajo de un Master's Thesis Project llevado a cabo por Andreas Brykt [4]. Este sistema realiza la extracción de un tipo de características de imagen (SURF [3]) de manera distribuida en una red de varios nodos. Antes de detallar las modificaciones que introduje en el sistema y su motivación, voy a explicar brevemente la idea de extracción de características de imagen y el sistema previo que me encontré.

El concepto más general de visión por computador mezcla procesamiento de imagen, extracción de características de imagen y aprendizaje automático, entre otras áreas de estudio, para proporcionar a sistemas informáticos la capacidad de manejar datos, actuar sobre ellos correspondientemente e interaccionar así con el mundo físico. En este proyecto me centro en la extracción de características de imagen.

La extracción de características se utiliza hoy en día para la búsqueda y clasificación de información presente en imágenes, vídeos, etc. Esta extracción puede verse como una forma especial de reducción dimensional, ya que el objetivo es detectar y guardar la información relevante contenida en la imagen, pero no la imagen en sí. Esta información se compara posteriormente con la existente en una base de datos para poder así reconocer un objeto dentro de la imagen. De este modo, la extracción de características de imagen se puede emplear en multitud de aplicaciones: reconocimiento de objetos, reconocimiento facial, seguimiento de un objeto en una secuencia de vídeo, etc.

Existen muchos tipos de características posibles en una imagen. En este proyecto, las características toman en primer lugar la forma de puntos de interés, que tienen que ser detectados sobre la imagen original. Los puntos de interés se pueden definir como aquellas áreas de la imagen que el algoritmo de extracción considera que contienen la información más relevante y significativa. Tras ser detectados, los puntos de interés se convierten en descriptores, que son vectores de tamaño variable que contienen esta información. La comparación de un descriptor con otro presente en una base de datos es el paso final del proceso de la extracción, paso que no se trabaja en este proyecto.

El sistema previo en el que se basa este proyecto realiza esta extracción de forma distribuida en varios nodos. La figura 1 muestra la topología de los nodos que intervienen en el proceso. El nodo conectado a la cámara la controla y transmite la imagen que esta toma para que se realice la extracción de características en los nodos de procesamiento. Para ello, este nodo divide la imagen en tantas partes como nodos de procesamiento haya, y las asigna. El nodo sink se encarga de la comunicación con

el servidor, en nuestro caso, un ordenador portátil.

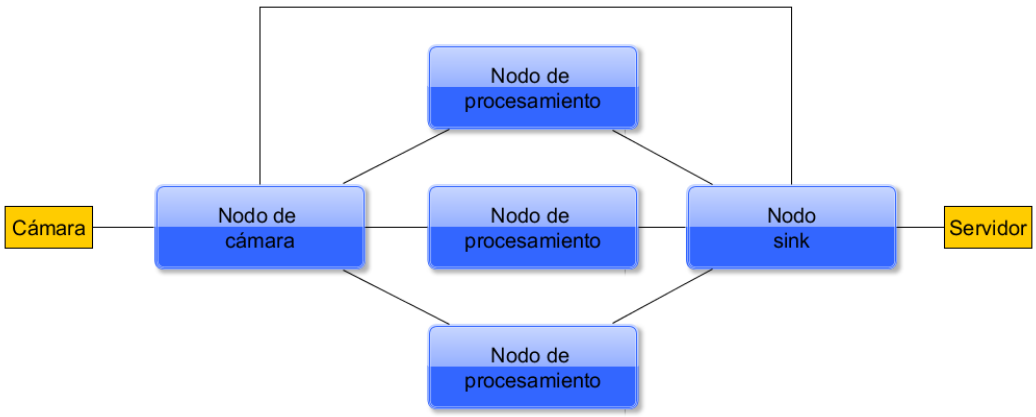


Figura 1: Topología de los nodos del sistema

Por lo tanto, el sistema previo a la realización de este proyecto tiene dos áreas de trabajo diferenciadas. La primera es la relacionada con la extracción de características de imagen. La segunda es la relacionada con los mecanismos de transmisión inalámbrica entre nodos para poder realizar la extracción de manera distribuida. El trabajo desarrollado en este proyecto intenta mejorar y optimizar ambas áreas, mostradas en la figura 2.

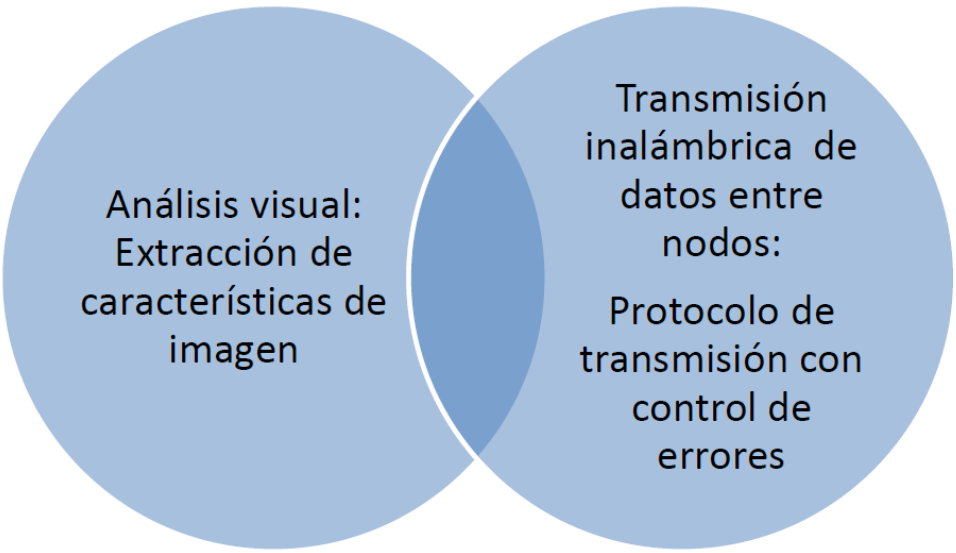


Figura 2: Áreas de trabajo del proyecto. La zona común a ambas es donde el proyecto reside.

En primer lugar, para la parte de la extracción, se quiere introducir un nuevo algoritmo que calcule unas características de imagen diferentes, BRISK [2]. Las razones por las que se eligió BRISK serán detalladas más adelante en el informe. La idea es que el usuario pueda elegir entre las dos propuestas de extracción en el momento de lanzar el programa. Esta capacidad se empleará para comparar las prestaciones entre ambos algoritmos y extraer conclusiones sobre su modo de funcionamiento.

El segundo objetivo del proyecto es la optimización de la parte de transmisión. Se quiere introducir un protocolo que minimice las pérdidas de datos que se producían en el sistema previo cuando se quería usar una velocidad de transmisión relativamente alta. Para ello, se va a diseñar e implementar un protocolo de transmisión basado en la retransmisión de paquetes perdidos, *acknowledgements* o reconocimientos y *timeouts* o tiempos de fin de espera. Además de reducir las pérdidas, este protocolo también traerá un incremento en la velocidad de transmisión.

## 1.1 Metodología

La aproximación a este sistema requirió un estudio previo de la literatura disponible en un cierto número de campos, principalmente un estudio sobre la extracción de características de imagen y sobre los distintos métodos de control de errores ARQ (Automatic Repeat reQuest), utilizados para conseguir una transmisión de datos sin pérdidas sobre un servicio no fiable, además de otros conceptos más específicos que se utilizarían en este sistema, como la parte hardware o la sintaxis que se emplearía para transmitir los datos.

Más tarde, realicé un profundo estudio del sistema descrito en [4], que incluyó conocer el propósito, la estructura y la jerarquía de las clases definidas en el código C++ (utilizando herramientas de análisis como doxygen) y conceptos de programación como multi-threading o construcción de sistemas en Linux, sobre los que no voy a entrar en este informe.

A continuación, y tras instalar Linux como sistema operativo en el ordenador portátil que actuaría como servidor y que controlaría los nodos, introduje los primeros cambios en el sistema, que estuvieron orientados a la integración del nuevo algoritmo de extracción de características, BRISK. Una vez que el sistema funcionaba con los dos algoritmos, procedí a calcular sus prestaciones temporales. Después de eso, se discutieron diferentes aproximaciones para dotar al sistema de fiabilidad y se creó el núcleo de la solución, seguido de un rediseño de determinadas partes y depuración de fallos. Tras ello, se llegó a un acuerdo en la definición de los tipos usados en la transmisión de datos para asegurar la continuidad en el desarrollo del sistema en otros proyectos. Finalmente, se llevó a cabo la evaluación de los aspectos de transmisión del sistema, midiendo el tiempo necesario para transmitir las imágenes sobre las que más tarde se realizaría la extracción de características.

## 1.2 Estructura del informe

Este informe está estructurado de la siguiente manera. Cada apartado se divide en aquellos aspectos relacionados con la extracción de características y en aquellos relacionados con la transmisión de datos.

El apartado 2 presenta los conocimientos previos necesarios para entender plenamente los objetivos y el trabajo realizado en el proyecto. Esta parte incluye una aproximación más profunda a la extracción de características de imagen; una breve mención a los distintos métodos de control de errores ARQ y un estudio del estándar IEEE STD 802.15.4 para la transmisión de datos y de la tecnología inalámbrica ZigBee. Además, también se incluyen los diferentes componentes software y hardware empleados en el sistema.

En el apartado 3 se explica el trabajo central llevado a cabo en el proyecto. La primera parte corresponde a la integración del nuevo algoritmo de extracción BRISK, y la segunda, al diseño e implementación del nuevo protocolo de transmisión de datos.

El apartado 4 incluye los experimentos que evalúan el funcionamiento del sistema. Una vez más, estos experimentos se dividen en aquellos relacionados con la parte de extracción de características y aquellos relacionados con los aspectos de transmisión.

En el apartado 5 se encuentran las conclusiones. Aquí se incluye una valoración del trabajo realizado e indicaciones de cómo se podría seguir desarrollando el sistema en el futuro.

Por último, cierran el informe los anexos y la bibliografía. El anexo A incluye un recorrido histórico por los distintos algoritmos de extracción de características de imagen desde su inicio alrededor de 1980 hasta las últimas novedades. El apéndice B detalla los parámetros de funcionamiento de los XSticks utilizados en los distintos experimentos. Por último, el apéndice C presenta la definición de los distintos tipos ASN.1 que se utilizan en la transmisión. Los anexos se muestran antes que la bibliografía debido a que incluyen referencias mencionadas en esta.

# 2. Conocimientos previos

## 2.1 Extracción de características de imagen

Como ya se ha apuntado en la introducción, la extracción de características de imagen es un proceso que intenta extraer la información relevante de una imagen y guardarla de manera que sea más fácilmente tratable. En este apartado se presentan los diferentes pasos de los que está compuesto el proceso.

### 2.1.1 Pasos

Generalmente, el proceso de extracción de características se puede dividir en tres pasos.

En el primero, se deben detectar los puntos de interés contenidos en la imagen para su procesamiento subsecuente. Estos puntos corresponden a lugares distintivos en la imagen, como esquinas o cruces en T. Los algoritmos que llevan a cabo esta tarea se llaman detectores, siendo su propiedad más importante la repetibilidad, que se define como la propiedad de obtener resultados similares si el experimento se repite bajo las mismas condiciones en un intervalo relativamente corto de tiempo.

Una vez que se han encontrado los puntos de interés, se deben extraer los descriptores relacionados. Los descriptores de imagen representan el área próxima de los puntos de interés y deben contener la información más importante y reconocible de esa región, pero manteniendo al mismo tiempo un nivel de abstracción suficiente. Esto es tanto como decir que deben ser distintivos, para diferenciar unos objetos de otros, pero también abstractos, para poseer robustez frente al ruido y a todas las posibles deformaciones de la imagen.

Por último, los vectores que contienen los descriptores deben ser comparados con otros descriptores contenidos en bases de datos. El proceso de correspondencia de imágenes se basa en el cálculo de la distancia entre vectores de descriptores (por ejemplo, distancia de Mahalanobis, euclídea o de Hamming). La dimensión de los vectores de descriptores juega un papel importante en la rapidez con la que se completa este paso. Un vector de pocas dimensiones conlleva un reconocimiento de los puntos de interés más rápido. Sin embargo, estos vectores tienden a ser menos distintivos.

### 2.1.2 Trabajos previos

En el anexo A se hace un recorrido histórico por los diferentes algoritmos que se han desarrollado en décadas recientes en relación a la extracción de características de imagen. También se incluye una descripción pormenorizada de los algoritmos SURF, que es el que estaba implementado en el sistema previo a la realización de este proyecto, y BRIEF [5], que es un precursor de BRISK, el que será integrado en el nuevo sistema.

## 2.2 Protocolos ARQ para el control de errores

Se pueden emplear diversidad de técnicas para conseguir una transmisión de datos sin errores. Los protocolos ARQ (Automatic Repeat reQuest) son métodos de control de errores que usan *acknowledgements* (mensajes enviados por el receptor para informar al transmisor de que ha recibido correctamente un paquete o *frame*) y *timeouts* (límites de tiempo que la aplicación espera para recibir un determinado mensaje, por ejemplo, un *acknowledgement*).

### 2.2.1 Stop-And-Wait

Stop-And-Wait es el protocolo ARQ más simple. El emisor transmite un solo *frame* en cada ocasión. Un *frame* se define como una unidad de transmisión que incluye información de sincronización (por ejemplo, una secuencia de bits) que permite al receptor detectar el comienzo y el final del paquete. Cuando el emisor transmite un *frame*, espera hasta que recibe una señal de reconocimiento (ACK). El ACK puede ser positivo, si el *frame* se recibe correctamente, o negativo, en caso contrario (en algunos casos un *acknowledgement* negativo consiste en no enviar ningún ACK). Si el ACK no llega al emisor dentro de un cierto límite temporal (*timeout*), el emisor transmite el mismo *frame* de nuevo.

Se puede ver al método Stop-And-Wait como un caso especial del protocolo general de ventana deslizante, con la peculiaridad de tener un solo slot en las ventanas de transmisión y recepción.

### 2.2.2 Go-Back-N

Go-Back-N se diferencia de Stop-And-Wait en que el emisor transmite el número de *frames* especificado por el tamaño de la ventana de transmisión sin necesidad de recibir un ACK para cada uno de ellos. Se trata, de igual manera, de un caso especial del protocolo general de ventana deslizante con la ventana de transmisión igual a N y la ventana de recepción igual a 1.

Go-Back-N es más eficiente que Stop-And-Wait, ya que el emisor continúa transmitiendo paquetes en vez de esperar reconocimientos para cada uno de ellos. Sin embargo, esto puede llevar a enviar el mismo *frame* en múltiples ocasiones. Este problema puede ser evitado si se considera el uso de un método basado en Selective Repeat.

### 2.2.3 Selective Repeat

En este método, el emisor transmite el número de *frames* especificado por el tamaño de la ventana incluso tras un *frame* perdido, lo que significa que el receptor continuará aceptando *frames* enviados después de un error. Cuando se llena la ventana del receptor, este responde con un ACK que contiene el número de secuencia del primer *frame* perdido. El emisor enviará ese *frame* una vez que ha enviado todos los demás contenidos en su ventana. Por eso, se puede ver este método como un caso del protocolo de ventana deslizante con ambas ventanas de transmisión y recepción de tamaño mayor que 1.

Sin embargo, estas ventanas no pueden tener cualquier tamaño. Estos deben ser iguales y también iguales a la mitad del número de secuencia máximo (si se definen los números de secuencia desde 0 hasta n-1) para evitar confusiones en la interpretación si, por ejemplo, todos los ACKs se perdieran.

## 2.3 Software

### 2.3.1 OpenCV

Open Source Computer Vision Library (OpenCV) es una librería de funciones diseñada con el objetivo de eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real. Está escrita en C/C++ y aprovecha el procesamiento multi-core. Concretamente, el detector y el descriptor SURF están implementados en esta librería. Por esta razón se usa en el proyecto.

La versión adecuada para cada sistema operativo puede ser descargada de `http://opencv.org/`. OpenCV soporta Windows, Linux, Mac OS, iOS y Android. Todas las clases y funciones de OpenCV están contenidas en el namespace cv, por lo tanto, pueden ser accedidas usando el especificador **cv::** o la directiva **using namespace cv**. Se puede encontrar más información online en `http://docs.opencv.org/`.

### 2.3.2 ASN.1

"Abstract Syntax Notation number One es un estándar que define un formalismo para la especificación de tipos abstractos de datos" [1]. Las reglas definidas en este estándar incluyen estructuras que describen datos transmitidos por protocolos de telecomunicaciones. Estas estructuras son usadas para representar, codificar, transmitir y decodificar estos datos independientemente de cualquier implementación física o de lenguaje.

Los tipos usados para la transmisión en este proyecto se definen siguiendo este estándar, y se pueden consultar en el anexo C de este informe.

ASN.1 se asocia con varias reglas estandarizadas de codificación, ya que define la sintaxis abstracta pero no la manera en la que los datos son codificados. Las reglas originales de codificación son las llamadas Basic Encoding Rules (BER), que serán explicadas a continuación.

#### *2.3.2.1 Basic encoding rules (BER)*

Hay dos tipos diferentes de datos que pueden ser transmitidos:

- *Tipos primitivos*
  Estos tipos incluyen tipos predefinidos, como INTEGER o BOOLEAN.
- *Tipos construidos*
  Los miembros de tipos construidos son tipos primitivos u otros tipos construidos definidos en otro lugar. Cuando la longitud es enunciada explícitamente, el tipo se llama construido de longitud definida. Cuando el objeto tiene un miembro cuya longitud no es expresada forma parte de un tipo construido de longitud indefinida.

La estructura de codificación para esos dos diferentes tipos consiste en los siguientes componentes:

- *Octetos de identificación*
  Los bits 8 y 7 describen la clase del objeto. Los posibles valores son Universal, Application, Context-specific o Private. El bit 6 diferencia si el objeto es primitivo o construido. El resto de bits constituyen el tag, que identifica el tipo del contenido.
- *Octetos de longitud*
  Los octetos de longitud se presentan en dos formas, la definida y la indefinida. La forma definida se usa para tipos primitivos o construidos cuando toda la información está disponible inmediatamente y la forma indefinida se usa para los tipos construidos cuya información no lo está. Para la forma definida, el

bit 8 diferencia si se usa la forma corta o la larga. Para la forma corta, los bits del 7 al 1 codifican el número de octetos para los octetos de contenido. En la forma larga, se usan uno o más octetos subsecuentes. La forma indefinida comienza con un octeto predefinido y siempre debe acabar con un octeto de fin de contenido.

- *Octetos de contenido*
  Los datos están codificados en estos octetos.

- *Octetos de fin de contenido*
  Como se ha dicho antes, el octeto de fin de contenido señala el fin de los octetos de contenido para los tipos indefinidos.

#### *2.3.2.2 Compilador ASN1.c*

Para obtener archivos .c y .h que contienen las estructuras definidas usando el lenguaje ASN.1, se usa este compilador: `http://lionet.info/asn1c/`.

### 2.3.3 IEEE STD 802.15.4

Hasta la introducción del IEEE STD 802.15.4 existía una ausencia de estandarización global en el entorno de las redes de sensores. El IEEE 802 Working Group, en colaboración con otros grupos, como ZigBee Alliance, combinó esfuerzos para satisfacer la necesidad de conexiones inalámbricas de bajo coste y de baja potencia para entornos residenciales e industriales, permitiendo al mundo electrónico interactuar con el entorno físico. El estándar 802.15.4 fue introducido y ratificado en 2003. Su objetivo era proporcionar un estándar con muy baja complejidad, coste y potencia para conexiones inalámbricas de baja tasa de datos entre dispositivos baratos fijos, portátiles y móviles. Este protocolo apuntó a "conferir a dispositivos simples una fiable y robusta tecnología inalámbrica que pudiera funcionar durante años con baterías regulares y primarias" [6].

El estándar 802.15.4 define las capas física (PHY) y de control de acceso al medio (MAC), dejando las capas superiores de la estructura OSI/ISO sin especificar. Está dirigido, como ya se ha dicho antes, a comunicaciones *low-duty-cycle* (10 - 115.2 kb/s), donde al dispositivo se le permite permanecer la mayor parte del tiempo en un estado de consumo ultra-bajo. Comunicaciones *high duty-cycle*, como voz o vídeo de baja calidad, no se pueden beneficiar tanto en el consumo de potencia.

La especificación 802.15.4 proporciona guía en posibles tipos de redes, sugiriendo principalmente dos topologías: la estrella y la punto a punto. Queda a estándares superiores como ZigBee proporcionar medios para el manejo de la red, la seguridad y el enrutamiento.

### 2.3.4 ZigBee Alliance

El estándar 802.15.4 no especifica más allá de una comunicación punto a punto una topología de red, esquemas de enrutamiento o de crecimiento de la red. Por esa razón, varios métodos para emplear el estándar 802.15.4 en redes más grandes y organizadas han sido propuestos, siendo ZigBee uno de ellos. "ZigBee explota el protocolo inalámbrico de corto de alcance 802.15.4 para añadir conexiones de red flexibles, herramientas de seguridad poderosas, perfiles de aplicaciones bien definidos y un completo programa de interoperabilidad, cumplimiento y certificación" [6]. En otras palabras, ZigBee se construye sobre IEEE STD 802.15.4, define las especificaciones de la capa de red y proporciona un marco de trabajo para la programación de aplicaciones. Es lo suficientemente flexible para satisfacer las necesidades de muchas aplicaciones comerciales y, como en nuestro caso, puede también actuar como un banco de pruebas para la investigación de nuevos algoritmos. Las características de la capa de red de ZigBee incluyen unirse/dejar una red, seguridad y enrutamiento. La división entre lo cubierto por 802.15.4 y ZigBee se puede observar en la figura 3.

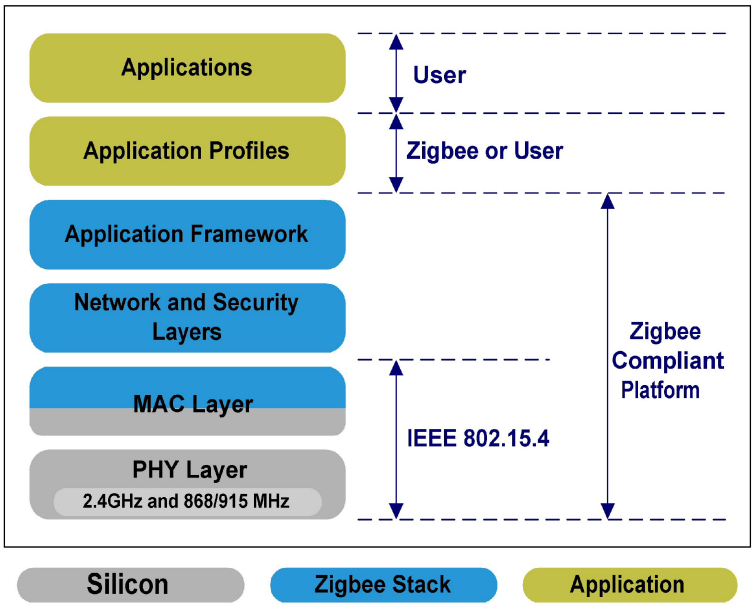


Figura 3: 802.15.4 y ZigBee en el modelo ISO/OSI

## 2.4 Hardware

### 2.4.1 BeagleBone Black

El BeagleBone Black es un dispositivo que pertenece a la familia BeagleBoard. Es un tarjeta-ordenador de baja potencia y código abierto producido por Texas Instruments en asociación con Digi-Key y Newark element14. Es compatible con Linux, Android y Ubuntu con tan solo un cable USB. El BeagleBone Black es uno de los miembros más nuevos de esta familia, siendo de bajo coste y centrado en futuras expansiones.

En el sistema en el que este proyecto se ha desarrollado, los nodos son dispositivos BeagleBone Black. Son conectados mediante un cable Ethernet al ordenador portátil que trabajará como servidor $S$ y las comunicaciones entre los distintos nodos se llevarán a cabo conectando un dispositivo XStick a cada nodo a través de su puerto USB.

La tarjeta mide 86.36mm x 53.34mm y cuesta aproximadamente 45 dólares US.

Algunos aspectos técnicos de los BeagleBones se declaran en la siguiente lista:

- Procesador: AM335x 1GHz ARM Cortex-A8
  - 512MB DDR3 RAM
  - 2GB 8-bit eMMC on-board flash storage
  - 3D graphics accelerator
  - NEON floating-point accelerator
  - 2x PRU 32-bit microcontrollers
- Conectividad
  - USB client for power and communications
  - USB host
  - Ethernet
  - HDMI
  - 2x 46 pin headers

La figura 4 muestra un BeagleBone Black. Se puede obtener más información en: `http://beagleboard.org`.

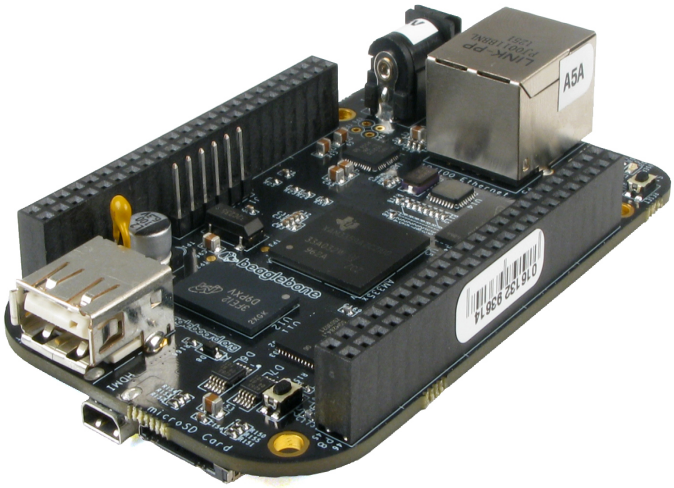

Figura 4: BeagleBone Black

### 2.4.2 XStick

El dispositivo usado en el sistema para llevar a cabo la transmisión inalámbrica es el XStick de Digi International. La versión empleada es el XStick 802.15.4, que es compatible con dispositivos XBee y 802.15.4. Los XSticks se conectan a través de su puerto USB a los BeagleBones descritos en el apartado anterior.

Algunos aspectos técnicos de los XStick:

- Indoor/Urban Range: 15m.
- Transmit Power Output (Nominal 25Âž C): 1 mW (0 dBm).
- Serial Interface Data Rate (Software Selectable): 1200 bps - 115.2 Kbps.
- Supply Voltage: 5V from USB port of PC.
- Frequency Band: 2.4 GHz ISM.
- Dimensions (L x W x H): (5.83 cm x 2.03 cm x 1.08 cm).

La tasa de datos del interfaz en serie puede ser modificada, además de otros aspectos. El XStick aparece como un puerto serie en /dev/ttyUSBx, siendo x un número que empieza la cuenta en 0. El XStick puede ser configurado usando un software llamado X-CTU, disponible en la página web de Digi. Además de la tasa

de datos, usando X-CTU se es capaz de modificar el número de reintentos de transmisión, opciones de paridad u opciones relativas a las direcciones.

Las configuraciones usadas en los experimentos de este proyecto se puede ver en el anexo B, al final del informe.

La figura 5 muestra un XStick. Se puede encontrar más información en: `http://www.digi.com`.


Figura 5: XStick de Digi

# 3. Diseño e implementación del sistema

Como se ha anunciado previamente, el trabajo desarrollado durante este proyecto fin de carrera se divide en dos áreas diferentes de trabajo. La primera se centra en el análisis visual de una imagen, llevando a cabo la extracción de características, y la segunda se centra en aquellos aspectos relacionados con la transmisión de datos inalámbrica entre los nodos del sistema. Los apartados 3 y 4 tienen ambos dos partes claramente diferenciadas, cada una relacionada con estas dos áreas de trabajo, que se presentan en el orden mencionado.

## 3.1 Integración del algoritmo de extracción BRISK

El objetivo para esta primera parte del proyecto era incluir en el sistema un nuevo algoritmo de extracción, que funcionara conjuntamente con el ya implementado, que era SURF. Se decidió que el nuevo algoritmo introducido fuera BRISK, por razones que se comentarán más adelante cuando se hayan realizado los experimentos que ilustren las prestaciones de ambos algoritmos. BRISK es una evolución del algoritmo BRIEF [5], explicado en el anexo A.4.

### 3.1.1 Características de BRISK

Aunque BRIEF cumplía satisfactoriamente sus objetivos de diseño, a saber: descriptores rápidos de computar y correspondencias a través de vectores binarios que contuvieran resultados de comparaciones de intensidad de pixels, este método presentaba algunos inconvenientes, siendo la sensibilidad a rotaciones de imagen y de escala el más importante de todos ellos, lo que impedía su aplicación de manera generalizada.

Binary Robust Invariant Scalable Keypoints (BRISK) es un método para la detección de puntos de interés, cálculo de descriptores y de correspondencias de manera rápida, pero que a la vez sean "invariantes a rotación y escala hasta cierto punto, logrando criterios de actuación comparables a los disponibles actualmente en el estado del arte al mismo tiempo que se reducen drásticamente los costes computacionales" [2].

La detección de puntos de interés de BRISK se basa en el trabajo de Mair *et al.* [7]. En ese artículo, se define AGAST como una mejora de FAST, considerado una base muy eficiente para la extracción de características. La mejora de AGAST consiste en una aceleración en la actuación de FAST.

Por otra parte, BRISK, para ser capaz de proporcionar al método invarianza

a escala, busca los máximos no solo en el plano de imagen, sino también en el espacio de escalas. Esto se hace representando la imagen original como una familia de un parámetro de imágenes suavizadas. El parámetro corresponde a la escala de las imágenes. Por tanto, lo que se obtiene en este caso es una familia consistente en la imagen original reescalada a distintos tamaños. La forma piramidal de esta familia hace que se conozca como pirámide del espacio de escalas. La figura 6 es una reconstrucción de la pirámide usada en posteriores experimentos.

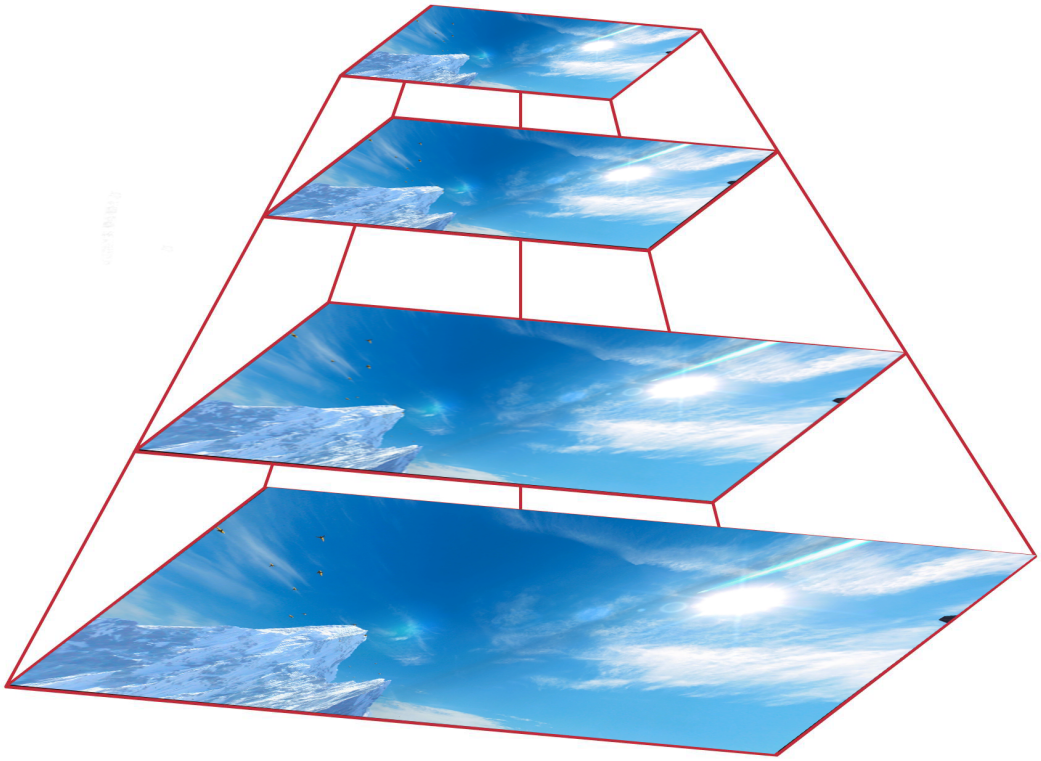

Figura 6: Pirámide del espacio de escalas con 4 tamaños de imagen diferente. La imagen que se muestra como ejemplo se usará más adelante.

BRISK divide la pirámide en capas consistentes en $n$ octavas $c_i$ y $n$ intra-octavas $d_i$, para $i = 0, 1, ..., n\text{-}1$ y normalmente $n = 4$. Las octavas en este marco de trabajo son calculadas progresivamente re-muestreando a la mitad la imagen original $c_0$. Es decir, cada octava se calcula reduciendo a la mitad la imagen anterior. En cuanto a las intra-octavas, cada una de ellas, $d_i$, está localizada entre las capas $c_i$ y $c_{i+1}$. La primera de ellas $d_0$ se obtiene reduciendo el tamaño de la imagen original $c_0$ por un factor de 1.5. El resto de ellas están formadas también por sucesivos re-muestreos a la mitad, como ocurría con las octavas.

BRISK usa una máscara de 9-16 para la detección de puntos de interés. Esta máscara requiere que al menos 9 pixels consecutivos en el área de 16 pixels sean lo suficientemente o más brillantes o más oscuros que el pixel central. Este detector se aplica primero a cada octava e intra-octava y para cada región se obtiene el resultado *s* o *score*. A continuación, este resultado de las potenciales regiones de interés se compara con los resultados de las regiones vecinas en la misma capa y más tarde con los resultados de las capas superior e inferior.

Figura 7: Imagen usada para el cálculo de las prestaciones temporales de los algoritmos SURF y BRISK y para ilustrar la teoría sobre la fase de detección.

Como hemos dicho, esto se hace para proporcionar al método invarianza a escala. El resultado es que los puntos de interés se detectan en varias escalas, obteniéndose puntos de diferentes tamaños, cada uno con un resultado *s* o *score* asociado. Estos resultados se ordenan posteriormente. La figura 7 es una imagen que se va a usar en varios experimentos durante todo el proyecto y que ahora sirve para presentar un ejemplo visual de lo dicho hasta ahora.

Sobre esta imagen se van a detectar los puntos de interés usando detectores con diferentes parámetros. En esencia, se ajustan los detectores para que detecten 50 y 1000 puntos de interés. Además, para cada caso se usa un detector con 2 octavas y otro con 4 octavas. Las figuras 8, 9, 10 y 11 muestran las diferencias en los resultados que el uso de estos diferentes detectores produce. Los círculos verdes son los puntos de interés detectados con el algoritmo BRISK. También se incluyen los puntos de interés detectados con SURF (círculos rojos).

Figura 8: *sky_large* con 50 puntos de interés detectados en 2 octavas

Las figuras 8 y 9 muestran los 50 puntos de interés que han obtenido el resultado más alto en la fase de detección, para los detectores de 2 y 4 octavas. SURF y BRISK producen resultados similares, aunque los círculos SURF muestran información adicional respecto a la orientación de los puntos (los de BRISK también contienen esta información pese a que no se muestre). La diferencia en usar 2 o 4 octavas para la detección reside en que para 4 octavas aparecen mayores círculos en la imagen final, que se traducirán en descriptores representando un área más grande.

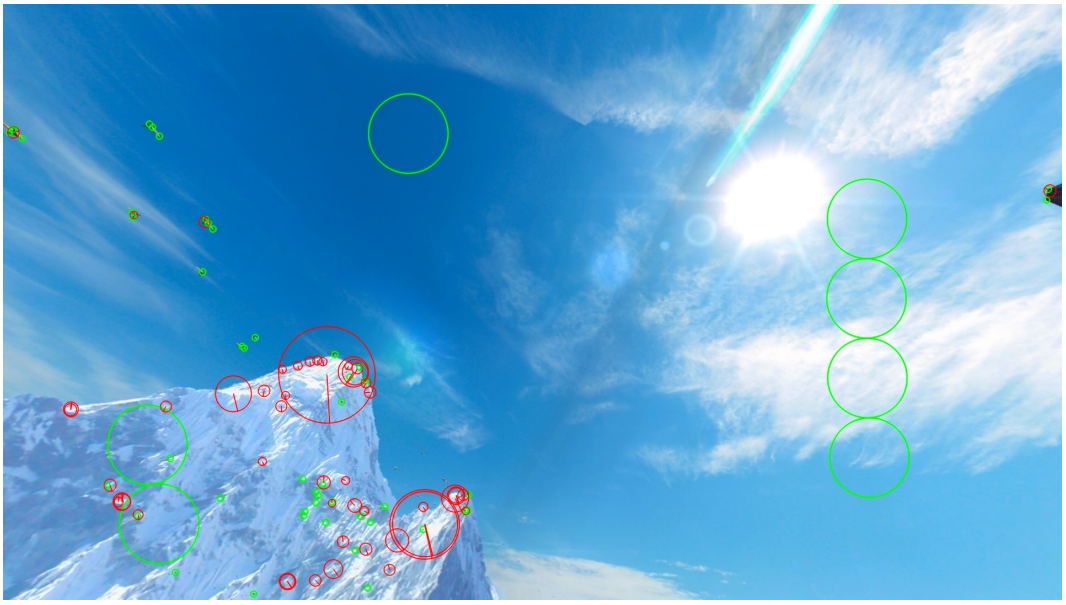

Figura 9: *sky_large* con 50 puntos de interés detectados en 4 octavas

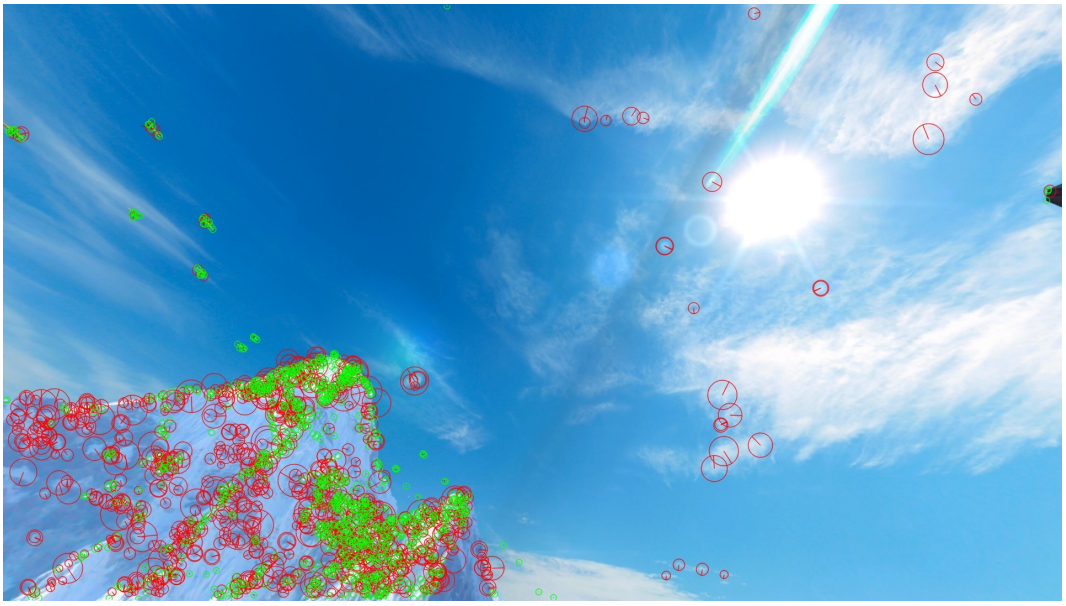

Figura 10: *sky_large* con 1000 puntos de interés detectados en 2 octavas

Las figuras 10 y 11 muestran la misma imagen, pero con 1000 puntos de interés detectados. Para los detectores de 2 octavas, los puntos de interés tienden a concentrarse en áreas de la imagen donde la información está guardada en regiones más pequeñas. En las imágenes, podemos ver cómo cubren la montaña casi por completo, mientras que el cielo, con sus transiciones de color y textura más suaves, está

prácticamente vacío. Sin embargo, esto no ocurre para los detectores con 4 octavas, donde los resultados cambian ligeramente. Áreas más grandes son detectadas como puntos de interés, lo que proporciona al algoritmo mayor invarianza a escala.

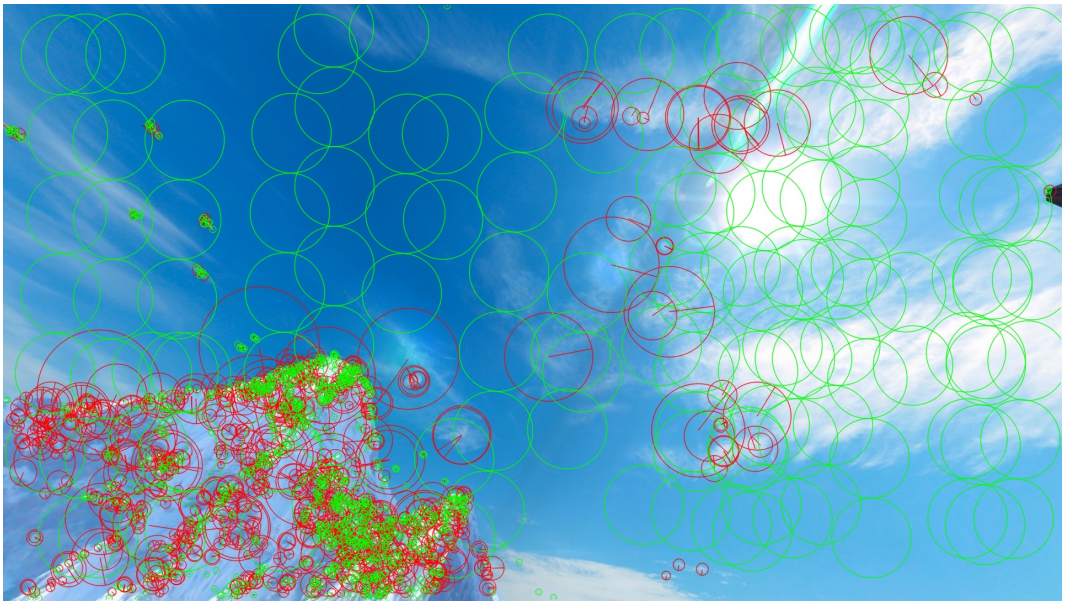

Figura 11: *sky_large* con 1000 puntos de interés detectados en 4 octavas

Con un conjunto de puntos de interés ya definido, el descriptor BRISK se forma como un vector binario, resultado de concatenar simples tests de comparación de brillantez. A diferencia de BRIEF, BRISK detecta las direcciones características de cada punto de interés para crear descriptores normalizados en orientación y así lograr invarianza rotacional. Además, "las comparaciones de brillantez se seleccionan cuidadosamente para centrarse en maximizar la descriptividad" [2].

Para establecer correspondencias entre dos descriptores BRISK, se calcula la distancia Hamming, como se hacía en BRIEF. De este modo, el número de bits diferentes en los dos descriptores es una medida de su disparidad. Esta operación se puede reducir a una XOR bit a bit seguida de un conteo de bits, lo que puede ser llevado a cabo muy eficientemente en los procesadores actuales.

### 3.1.2 Clases y funciones necesarias para la integración de BRISK

Dado que la implementación del algoritmo BRISK en la librería de OpenCV no era enteramente correcta en el momento de realizar este proyecto, se usó la implementación original. Esta versión puede ser encontrada en `http://www.asl.ethz.`

`ch/people/lestefan/personal/BRISK/brisk.zip`. Concretamente, la implementación original de BRISK se empleó para crear las instancias de clases como el detector y el extractor. Para la definición de los puntos de interés y de los descriptores se usó OpenCV como librería auxiliar.

La inclusión de estas clases de la librería de BRISK no fue sencilla, ya que se debían adaptar a la manera en la que se realizaba la extracción en el sistema previo. Los puntos de interés y descriptores, por ejemplo, debían tener el mismo formato para ser capaces más adelante de comparar fielmente la actuación de ambos algoritmos. La definición de los detectores y extractores debía estar en consonancia con los parámetros que se pasaban a cada función, como el umbral o *threshold* o el número de octavas. La manera en la que se codificaban los detectores y extractores que serían enviados a cada nodo para realizar allí su función variaba en función de un algoritmo u otro. Estos son algunos de los pormenores del trabajo de programación para este apartado.

Por otro lado, se quería que el usuario fuera capaz de elegir entre SURF o BRISK en el momento de lanzar el programa. Para ello, se creó la siguiente estructura de clases: la clase **DfeSettings**, además de la definición de los parámetros generales del sistema, contendría un atributo de la clase **SurfSettings** y otro de la clase **BriskSettings**, además de otro atributo que marcaría cuál de ellas se usaría en la ejecución del programa. Las clases **SurfSettings** y **BriskSettings** contienen los atributos necesarios para llevar a cabo la extracción.

El atributo diferenciador del algoritmo en uso se iría comprobando en cada función relacionada con la detección o extracción de características. No fue necesario crear ninguna clase adicional de detección o extracción ya que, como se ha dicho, se modificaron las de la versión previa para que contuvieran el código necesario para el funcionamiento de ambos algoritmos. Si se quisiera añadir al sistema un tercer (o cuarto) algoritmo de extracción, se podrían aprovechar los puntos en los que la ejecución se decanta por SURF o BRISK, añadiendo el código necesario para el funcionamiento de ese nuevo algoritmo (habría que añadir simplemente un nuevo *case* a todos los *switches* creados).

## 3.2 Protocolo de transmisión con control de errores

Esta sección describe el diseño y la implementación del protocolo de transmisión que dota al sistema de fiabilidad. La inclusión de este protocolo de retransmisiones se debe al hecho de que la tecnología ZigBee tiene un control de errores para las transmisiones unicast (punto a punto), pero no para las transmisiones broadcast (punto a resto de puntos) o multicast (punto a un conjunto de puntos). Aun así, también se producían errores en la transmisión de paquetes unicast en el anterior

sistema con una velocidad de transmisión de media a elevada, así que la inclusión de este protocolo era si cabe más necesaria.

### 3.2.1 Modo de funcionamiento algorítmico

En esta sección, se presenta una descripción de alto nivel del modo de operación del protocolo de transmisión. Los conceptos clave, ya mencionados en la explicación de los métodos Automatic Repeat Request en el apartado 2.2, son la transmisión de paquetes perdidos, los *acknowledgements* y los *timeouts*.

El esquema de transmisión del diseño anterior se comportaba de la siguiente manera. Dos nodos estaban involucrados en la transmisión, el sender y el receiver. Cuando el sender quería transmitir un mensaje al receiver, codificaba los datos y luego formaba paquetes que no excedían los 100 bytes de longitud, para ser enviados usando ZigBee. El grupo de paquetes que pertenecía a un mismo mensaje se llamaba *SendItem*, de acuerdo a la terminología empleada. Cuando el receiver recibía un paquete, esperaba hasta que todos los paquetes del correspondiente *SendItem* hubieran llegado; en ese momento, los decodificaba y tomaba el correspondiente camino de ejecución.

Para el nuevo diseño, el objetivo era recibir un *acknowledgement* por cada *SendItem*. Cada vez que el sender terminara de transmitir algo, esperaría una confirmación del receiver. Si todos los paquetes hubieran sido recibidos correctamente, se transmitiría un *acknowledgement* positivo de vuelta al sender. La figura 12 ilustra este comportamiento.

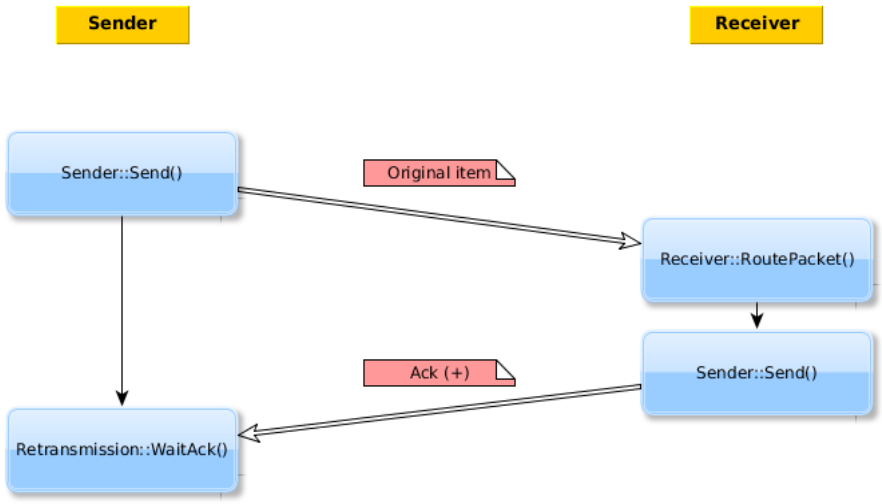


Figura 12: Esquema de retransmisión, caso 1. El sender envía el *SendItem* y espera un *acknowled-*

*gement*. El *SendItem* es transmitido correctamente, entonces el receiver crea un *acknowledgement* positivo y lo envía como respuesta al sender. Las funciones nombradas en las figuras 12, 13 y 14 serán explicadas en el apartado 3.2.2.2

Este es el caso óptimo en el que no hay pérdida de paquetes. Esta situación no siempre ocurre. Cuando al receiver le llega un *SendItem* incompleto debido a la pérdida de paquetes, este crea un *acknowledgement* negativo, en el que los paquetes perdidos durante la transmisión son especificados. Entonces, el sender lee este *acknowledgement* y retransmite esos paquetes. Este caso se muestra en la figura 13.

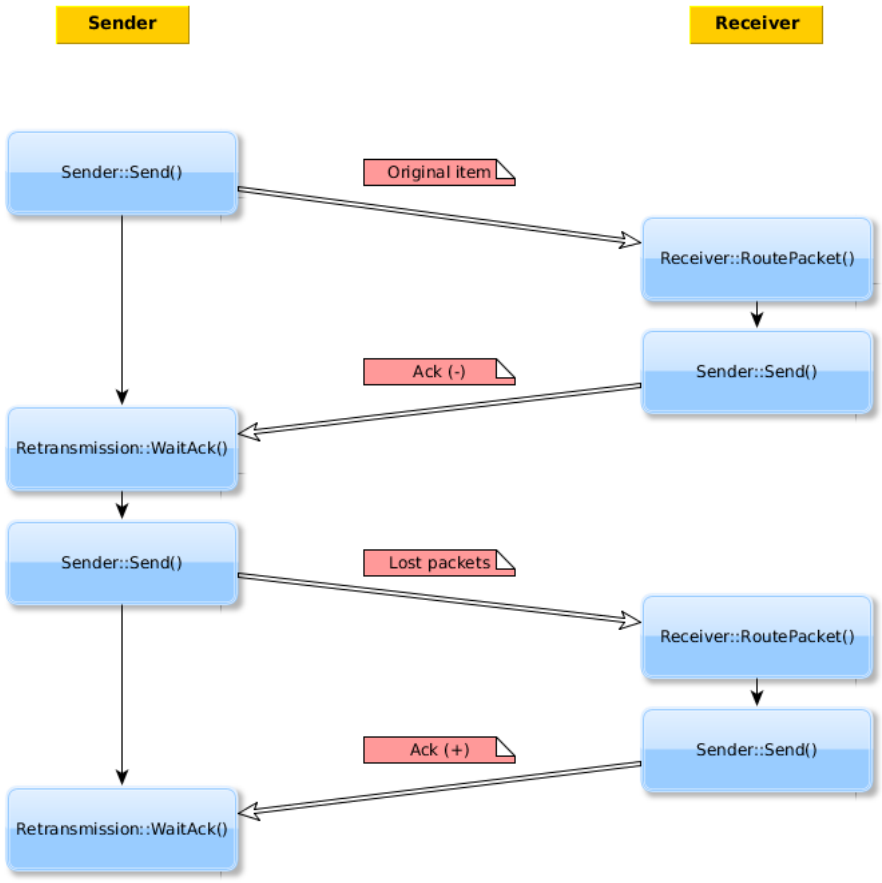


Figura 13: Esquema de retransmisión, caso 2. El sender envía el *SendItem* original y espera un *acknowledgement*. El receiver aprende que no todos los paquetes han sido recibidos, entonces crea un *acknowledgement* negativo y lo envía de vuelta al sender. El sender recoge la información que especifica qué paquetes del mensaje original necesitan ser enviados otra vez y los transmite de nuevo. Este proceso se repite hasta que el sender recibe un *acknowledgement* positivo o se llega al límite de intentos de retransmisión.

Por supuesto, es posible que sea el acknowledgment lo que se pierda. En ese caso, el sender es capaz de pedir al receiver que lo retransmita de nuevo. Esta acción se denomina *ping*. En ese caso, el receiver debe responder al sender con dos *acknowledgements*: uno para el paquete recién recibido (el *ping*) y otro para el *acknowledgement* que se había perdido previamente. Este modo de operación se puede observar en la figura 14.

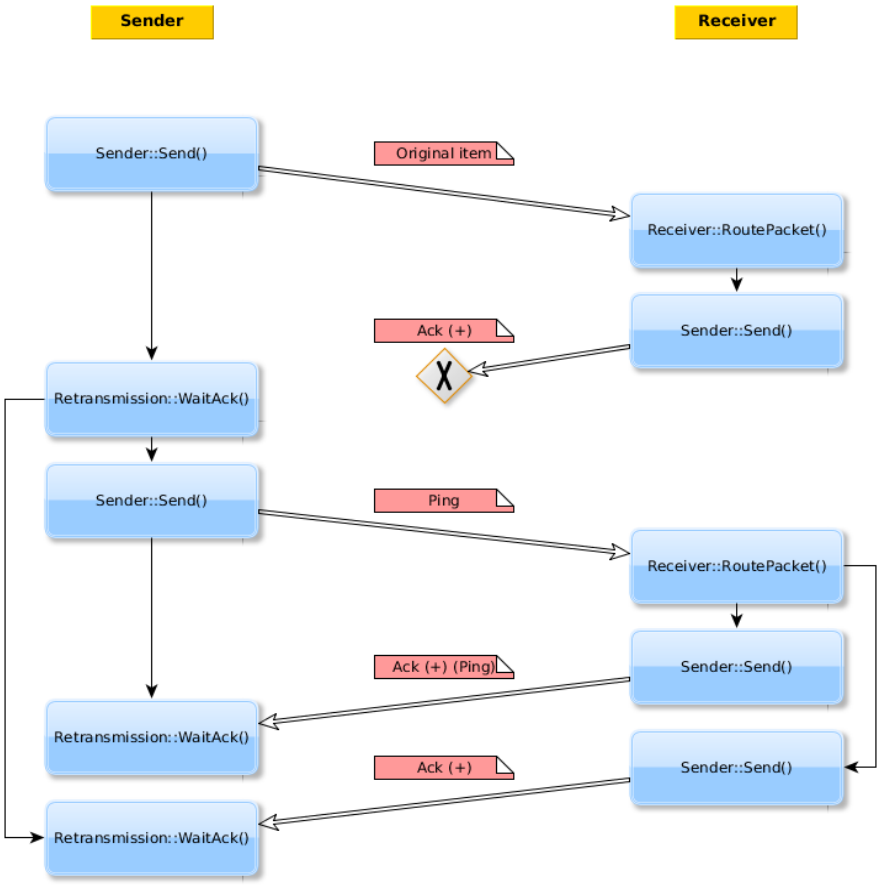


Figura 14: Esquema de retransmisión, caso 3. El sender envía el *SendItem* original y espera un *acknowledgement*. El receiver aprende que no todos los paquetes han sido recibidos, entonces crea un *acknowledgement* positivo y lo envía de vuelta al sender. Este *acknowledgement* se pierde en el proceso. El sender espera el *acknowledgement* hasta que se alcanza un límite de tiempo y al no haber recibido todavía nada, envía un *ping* para pedir una retransmisión del *acknowledgement*. El receiver lo transmite de nuevo y esta vez se transmite correctamente y la transmisión se completa satisfactoriamente.

Teniendo estos tres casos en cuenta, el protocolo implementado sería una variación del método Selective Repeat, ya que solo se retransmiten aquellos paquetes perdidos una vez que se llega al límite de la ventana deslizante (en este caso, el número de paquetes de cada *SendItem*). Selective Repeat es el método más eficiente de los tres descritos en el apartado 2.2.

Los *pings* son necesarios para aquellas ocasiones en las que los *acknowledgements* se pierden. Sin ellos, el sistema entraría en una fase de espera de reconocimientos y nunca completaría la operación si un *acknowledgement* se perdiera. Sin embargo, usar *pings* no es la única alternativa. Podríamos haber diseñado un sistema que enviara *acknowledgements* periódicamente, no solo cuando llegara al receiver el último paquete de un *SendItem*. La ventaja de usar un *ping* es su simplicidad y una reducción en el flujo de bits transmitidos. Por otro lado, esta opción requiere el cálculo de los apropiados *timeouts*. Por ejemplo, el sistema no funcionaría adecuadamente si los *timeouts* fueran valores muy pequeños, o el *throughput* podría caer drásticamente si los valores de los *timeouts* fueran muy grandes.

Es interesante notar que el sender siempre espera un *acknowledgement* después de enviar un *SendItem*, excepto en el caso de que lo que se esté enviando sea un propio *acknowledgement*. De otro modo, la ejecución entraría en un bucle infinito de esperas y confirmaciones de mensajes. También tenemos que tener en cuenta que se pueden producir toda clase de combinaciones de los casos 1, 2 y 3 en nuestra transmisión. Por ejemplo: un *acknowledgement* negativo que se pierda.

Para que este sistema funcione adecuadamente, se deben calcular valores apropiados para el conjunto de *timeouts*. Este conjunto incluye los siguientes miembros:

- Estimación del tiempo de llegada del último paquete. Cuando se recibe correctamente el último paquete de un *SendItem*, el receiver sabe que la transmisión ha terminado y entonces procede a enviar el *acknowledgement* correspondiente. Pero si este paquete se pierde, el receiver debe haber estimado previamente su tiempo de llegada y enviar el *acknowledgement* si se alcanza este límite de tiempo. Cómo estimar este tiempo de llegada se explicará en el apartado 3.2.1.1.
- Número de intentos de retransmisión. El sender intenta reenviar los paquetes perdidos en un cierto número de ocasiones. Si se alcanza este número y todavía hay paquetes transmitidos incorrectamente, quizá haya un problema de funcionamiento en algún otro lugar del sistema. Este límite existe para ser capaz de salir de la ejecución en ese caso.
- Límite de tiempo para recibir un *acknowledgement*. El sender debe saber cuándo dejar de esperar un *acknowledgement* y enviar un *ping* pidiendo uno nuevo.

- Número máximo de *pings*. Este *timeout* es similar al de número de intentos de retransmisión, pero para los *pings*.

#### *3.2.1.1 Estimación del tiempo de llegada del último paquete*

Como ya ha sido explicado en apartados anteriores, un *acknowledgement* se envía como respuesta al sender cuando el último paquete de un mensaje llega al receiver. La clase **Packet** tiene dos atributos que se refieren al número de secuencia del paquete (*pkt_num_*) y al número total de paquetes para cada mensaje (*tot_pkts_*); por lo tanto, se puede realizar una comparación en el receiver cada vez que llega un paquete para ver si este es el último paquete del mensaje. Pero si este paquete se perdiera por cualquier causa durante la transmisión, el receiver seguiría teniendo que enviar el *acknowledgement*. Por esta razón, debe existir una estimación de cuándo el último paquete de cada mensaje debería llegar, y si ese tiempo se alcanza y sobrepasa, el receiver debe preparar el *acknowledgement* y enviarlo.

Para calcular esta estimación, se debe mantener un registro de los tiempos de llegada de los paquetes. Para cada llegada de un paquete, la estimación debe ser actualizada. La solución empleada se basa en el cálculo de dos tiempos diferentes: el *InterPacketTime* ($IPT$) y el *InterPacketDeviationTime* ($IPDT$). El primero da una medida de la diferencia de tiempo de llegada de dos paquetes consecutivos y el segundo da una medida de la variación de este tiempo. Estos dos valores se actualizan de acuerdo a las siguientes reglas:

$$\begin{cases} IPT = (1-\alpha) * IPT + \alpha * TimeSinceLastPacket, \\ IPDT = (1-\beta) * IPDT + \beta * |TimeSinceLastPacket - IPT|, \end{cases}$$

con $\alpha = 0.1$ y $\beta = 0.2$.

Hay, por lo tanto, dos valores que se actualizan con la llegada de cada paquete. Con estos dos valores la estimación para la llegada del último paquete, *LastPacketExpectedArrival* ($LPEA$), se calcula como sigue:

$$LPEA = (tot\_pkts\_ - pkt\_num\_) * (IPT + 4 * IPDT)$$

Tenemos que tener en cuenta que el tiempo de llegada del último paquete debe ser estimado para el *SendItem* original, pero también para aquellos paquetes que se pierden y que son reenviados por el sender. Debemos hacer cambios en la lógica de acuerdo a esto. Estos cambios se resumirían en saber cuántos paquetes van a ser

enviados en cada retransmisión y operar con ese valor.

#### *3.2.1.2 Fiabilidad Unicast vs. Broadcast*

El funcionamiento del sistema para paquetes unicast y broadcast difiere ligeramente. Para los elementos unicast, la lógica es exactamente la descrita anteriormente. Para mensajes broadcast, la operación del algoritmo es algo más compleja. El sender debe recibir un número de reconocimientos igual al número de conexiones activas con unidades xbee. Por ejemplo, considerando la topología mostrada previamente en la figura 1, que consiste en un nodo de cámara, tres nodos de procesamiento y un nodo sink, cuando el nodo de cámara envía una imagen por broadcast, tiene que esperar 4 *acknowledgements* y tratar cada uno de ellos separadamente. Esto significa que una vez que todos los *acknowledgements* son recibidos, el sender transmitiría por unicast los paquetes perdidos que son especificados en los *acknowledgements*, uno a uno.

Por supuesto, podría pasar que uno o varios de esos mensajes se perdieran. Entonces, un *ping* sería enviado a cada nodo que lo requiriera, del mismo modo que sucedería con una conexión unicast. Los *acknowledgements* recibidos serían guardados hasta que el último de ellos llegara. Si no llegaran todos los esperados, el mensaje original sería reenviado a cada nodo que no hubiera devuelto un *acknowledgement* y a los que sí que lo hubieran hecho se les retransmitiría los paquetes perdidos correspondientes, tratando cada conexión a partir de entonces como unicast. Afortunadamente, cada *SendItem* incluye un número de secuencia que se usa para evitar que debido a retransmisiones de mensajes enteros, como en este caso, se ejecute la misma acción dos veces.

Se podría mejorar el sistema con un simple cambio. Se podría calcular la intersección de los paquetes perdidos para todos los nodos y enviar este grupo de paquetes por broadcast en vez de unicast, minimizando así la cantidad de bytes transmitidos y maximizando la eficiencia.

### 3.2.2 Implementación del protocolo

Esta sección incluye las clases, los atributos y las funciones necesarias para implementar el modo de operación descrito en el apartado 3.2.1.

#### *3.2.2.1 Clase Retransmission*

Los nodos sender y receiver, involucrados en el protocolo de transmisión de la manera descrita en apartados anteriores, son implementados en dos clases llamadas **Sender** y **Receiver**, respectivamente.

El sistema usa programción por multi-threading. Multi-threading es la capacidad de un solo procesador para ejecutar eficientemente múltiples hilos de ejecución. Esto se hace asociando a cada clase especificada un hilo o cola, en la que se añadirán las tareas a ejecutar. Ambas clases **Sender** y **Receiver** heredan de una clase común, **ThreadedQue**, que implementa una cola. En el archivo *threaded_que.h*, donde la clase **ThreadedQue** es definida, el programa entra en un bucle que continuamente comprueba si algo se añade a la cola asociada a cada instancia de esta clase.

En la versión anterior del sistema, la función GetNextObjectFromQue(), que extrae los elementos de la cola, usaba la función pthread_cond_wait(). Esta función hacía que cada vez que hubiera algo listo para ser procesado por el sender o el receiver y fuera añadido a la cola, se llamara a la función DoStuffToObject() para tomar el camino de ejecución apropiado.

Con la definición de varios *timeouts* en el nuevo protocolo de transmisión, esta aproximación ya no es válida. Además de ser capaz de ejecutar DoStuffToObject() tan pronto como algo sea añadido a la cola, el sistema necesita llamar a una función que compruebe continuamente si los *timeouts* se han alcanzado o no. Esta función es TimeTracking() y la función usada en GetNextObjectFromQue() con este fin es pthread_cond_timedwait(), que espera a que algo se añada a la cola durante un límite definido de tiempo, tras el cual sale de su ejecución y se procede a realizar las comprobaciones temporales del sistema.

En la versión antigua del sistema, había una instancia de la clase **Sender** para cada unidad xbee. Sin embargo, había varias de la clase **Receiver** para cada unidad xbee: una para cada conexión (hay conexiones para el resto de unidades xbee y una adicional para transmisiones broadcast). Teniendo esto en cuenta, se creó la clase **Retransmission**. Solo una instancia de esta clase es inicializada para cada instancia de **Sender**. De este modo, un nodo solo puede enviar un *SendItem* cada vez, y hasta que no reciba un reconocimiento positivo, ningún otro elemento puede ser enviado.

La clase **Retransmission** toma como entrada una instancia de la clase **NodeManager**. Esto permite ganar conocimiento previo del número de conexiones activas que tiene cada unidad xbee, lo que es útil para la fiabilidad en transmisiones broadcast, como ya hemos visto.

#### *3.2.2.2 Camino de ejecución*

En este apartado se explica qué funciones se ejecutan cuando el sender transmite un *SendItem*. Para entender el proceso, se necesita conocer la definición de algunos tipos ASN.1. La lista completa de estos tipos se encuentra en el anexo C.

El *ReceptionReportMessage* actúa como el *acknowledgement*. Su miembro *droppedPackets* es una secuencia de enteros. Un *ReceptionReportMessage* con el campo *droppedPackets* vacío equivale a un *acknowledgement* positivo. Por otro lado, el miembro *mode* nos dice de qué manera los paquetes perdidos son especificados. Este campo es del tipo *ReceptionReportModes*.

El tipo *ReceptionReportModes* puede tomar dos valores: *specifyEach* y *specifyIntervals*. La elección de la primera opción, *specifyEach*, significa que los paquetes perdidos deben ser especificados en una lista simple que consista en los números de secuencia de esos paquetes. La segunda opción, *specifyIntervals*, considera grupos en vez de paquetes sueltos. Estos grupos son delimitados por dos enteros, el primero marca el número de secuencia del primer paquete perdido del grupo y el segundo el número de secuencia del último paquete perdido. La opción elegida es aquella que requiera la transmisión de un menor número de enteros. Dotar al sistema de esta capacidad de elegir entre dos opciones apunta a ganar eficiencia en el número de bytes enviados en la transmisión de un ACK. El uso de un método u otro de codificar los paquetes perdidos depende de la causa física por la que esos paquetes no se hayan transmitido correctamente.

Finalmente, tenemos el *ReceptionReportRequestMessage*. Este tipo actúa como el *ping* explicado en el apartado 3.2.1; pide al receiver la retransmisión de un *ReceptionReportMessage*. Su único campo especifica el número de secuencia del mensaje del que el sender está pidiendo el ACK. Cada *SendItem* tiene su propio número de secuencia. El receiver guarda un registro del último número de secuencia recibido para evitar ejecutar la misma acción por duplicado.

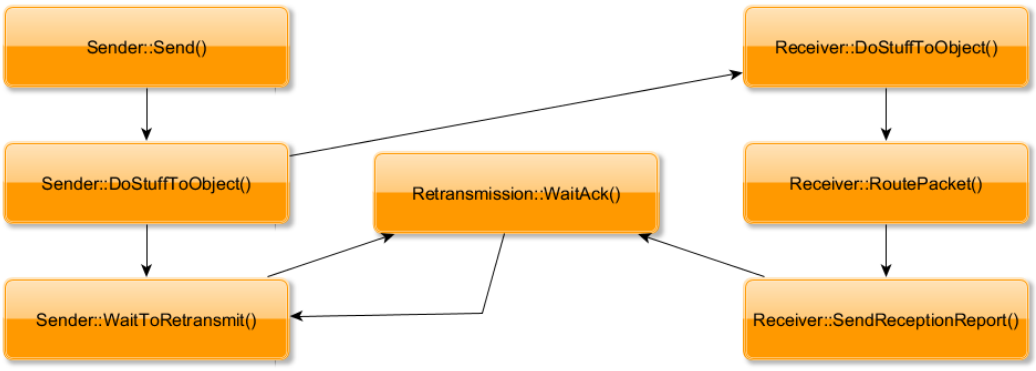


Figura 15: Principales clases que influyen en el proceso de transmisión de un mensaje completo.

Con lo anterior en mente, se puede explicar el camino de ejecución que sigue el programa. Cuando el sender quiere transmitir algo, llama a la función Sender::Send(), donde el *SendItem* se añade a la cola del sender. Cuando eso ocurre, se

llama a Sender::DoStuffToObject() y se transmiten los paquetes xbee usando la librería libxbee. Después de eso, el programa ejecuta la función Sender::WaitToRetransmit(), que llama a su vez a la función Retransmission::WaitAck(). En esta función, se espera la llegada de los *ReceptionReportMessage* necesarios. El receiver siempre devuelve al sender un *ReceptionReportMessage* (a no ser que lo que haya recibido haya sido otro *ReceptionReportMessage*) cuando el último paquete de un *SendItem* llega o cuando la estimación del tiempo de llegada del último paquete ha sido alcanzada. Para ello se emplea Receiver::SendReceptionReport(). Si el *acknowledgement* es negativo, se llama de nuevo a Sender::DoStuffToObject() para reenviar los paquetes perdidos. Si el *acknowledgement* es positivo, significa que no hay paquetes que transmitir de nuevo y el sender puede salir de la función DoStuffToObject() y que la tranmisión se ha completado correctamente. El camino descrito se puede observar en la figura 15.

# 4. Resultados experimentales

Para ayudar a ilustrar la actuación del sistema diseñado en este proyecto y explicado en el apartado 3, se llevó a cabo una serie de experimentos divididos en dos grupos que abarcan las áreas del proyecto ya sabidas: el primero se centra en los aspectos relacionados con la extracción de características de imagen y el segundo en los relacionados con la transmisión y fiabilidad del sistema.

## 4.1 Experimentos de extracción de características de imagen

En esta sección del informe, se presenta una comparación entre los tiempos necesarios para completar diferentes pasos del proceso de extracción de características, lo que nos permite obtener una idea en cuanto a las prestaciones temporales de los dos algoritmos soportados. Estos pasos corresponden a las fases de detección de puntos de interés y extracción de descriptores.

Se realiza la detección y extracción en un solo nodo o BeagleBone, de ahí que los tiempos dependan de las características mencionadas en el apartado 2.4.1. La imagen que se emplea para estos cálculos es la de la figura 7, mostrada en el apartado 3.1.1.

Para obtener información relevante y significativa de la actuación de estos algoritmos, la imagen original ha sido reescalada a 3 tamaños diferentes: 1920x1080, 1244x756 y 941x529 pixels, y se han calculado los tiempos para cada uno de estos tamaños. De aquí en adelante, estas imágenes se llamarán *sky_large*, *sky_medium* y *sky_small*, respectivamente. Para ganar robustez frente a deformaciones de escala y ver cómo afecta eso a los tiempos de cálculo, ambos métodos detectan puntos de interés en varias octavas; concretamente, los puntos de interés se detectan usando detectores de 2 y 4 octavas. En consecuencia, tratamos 6 casos diferentes: calculamos los puntos de interés y los descriptores para *sky_large*, *sky_medium* y *sky_small* con detectores de 2 y 4 octavas.

Para la primera parte, la detección de puntos de interés, la figura 16 muestra los tiempos de detección SURF para cada uno de esos 6 casos, mientras que los de BRISK se presentan en la figura 17. El eje de abscisas en cada método hace referencia al número de puntos de interés encontrados. Concretamente, se calcularon los umbrales o *thresholds* para obtener 50, 100, 200, 250, 300, 400, 500, 600, 700, 800, 900 y 1000 puntos de interés. Esto se hizo utilizando primero un umbral muy bajo para obtener más de 1000 puntos de interés, y más tarde, ordenando de mayor a menor los resultados *s* o *score* asociados a los primeros 1000 puntos de interés. El valor número 50 correspondería al *threshold* que se debía usar si se querían detectar 50 puntos de interés, lo mismo para el 100, 200, etc.

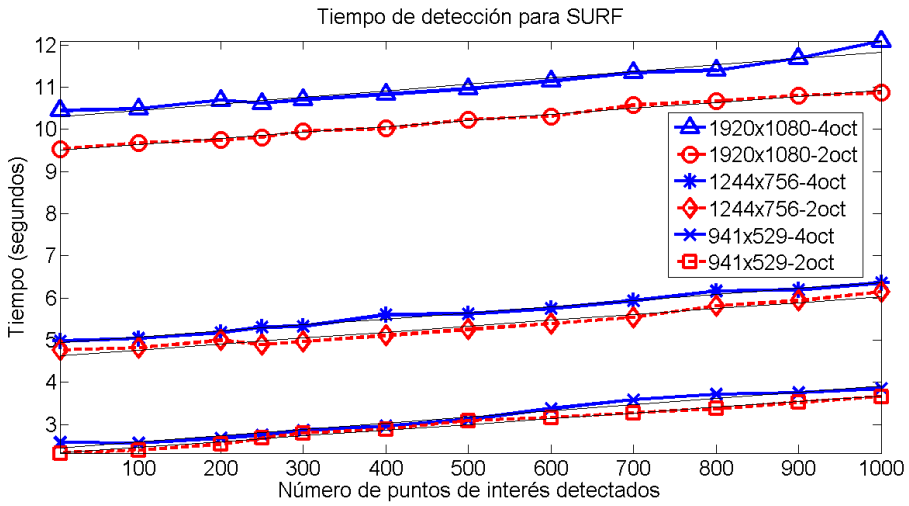


Figura 16: Tiempos de detección de puntos de interés para SURF. Las rectas finas representan regresiones lineales para cada una de las 6 variables.

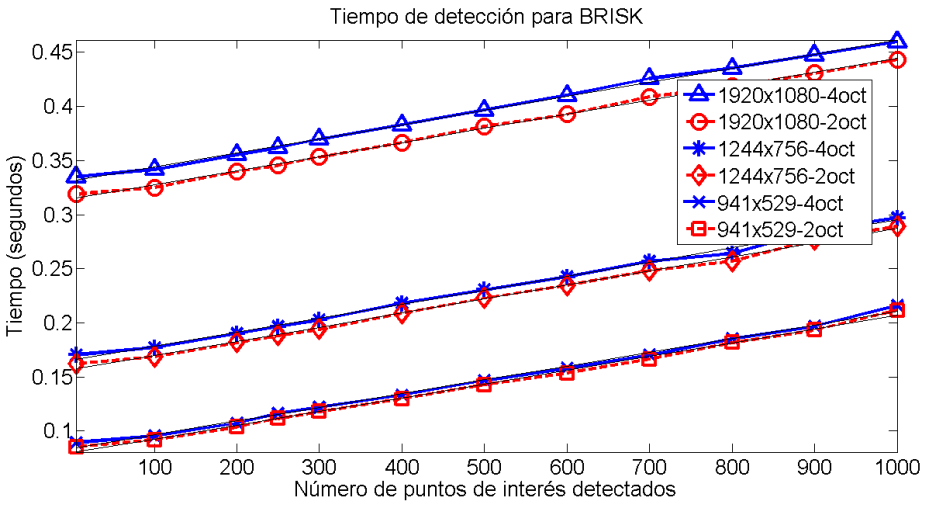


Figura 17: Tiempos de detección de puntos de interés para BRISK. Las rectas finas representan regresiones lineales para cada una de las 6 variables.

Estas dos figuras tienen estructuras similares. El tiempo necesario para obtener puntos de interés incrementa con el número de puntos (siguiendo una regresión lineal), pero también con el tamaño de la imagen y con el número de octavas. Estos resultados son consistentes con la teoría de ambos algoritmos, dado que una imagen más grande significa un mayor número posible de localizaciones de puntos de interés que necesitan ser comprobadas aplicando los filtros necesarios, y un número mayor de octavas significa una imagen siendo estudiada en un mayor número de escalas y, consecuentemente, un coste computacional mayor.

Sin embargo, la noción más importante que debemos extraer de estas dos figuras es que BRISK es mucho más rápido que SURF.

Las siguientes dos figuras, 18 y 19, muestran los tiempos de extracción de los descriptores. Estos descriptores se calculan de manera diferente en cada algoritmo, de ahí que los resultados no sean similares, lo que sí ocurría en la fase de detección. Lo que se puede decir sobre ellos es que las comparaciones de brillantez bit a bit de BRISK son mucho más rápidas que las respuestas wavelet de SURF.

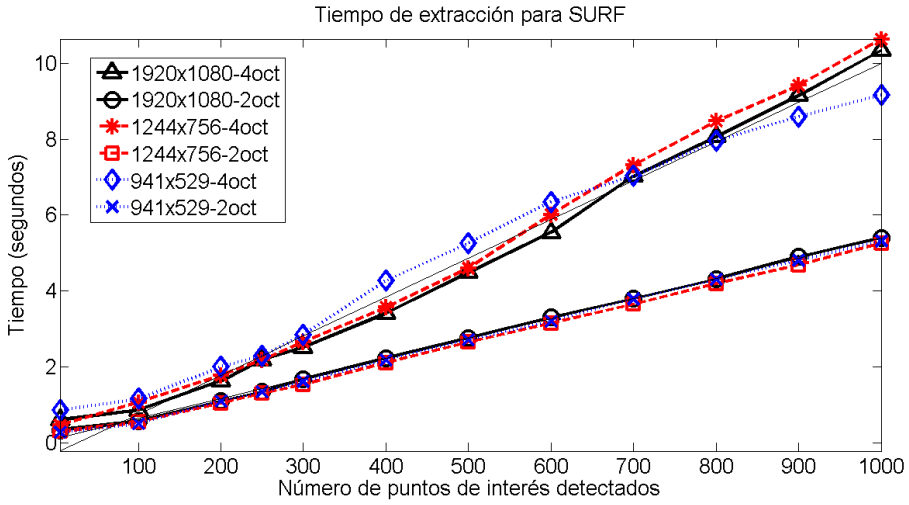


Figura 18: tiempos de extracción de descriptores para SURF. Las rectas finas representan regresiones lineales para *sky_large* con detectores de 2 y 4 octavas.

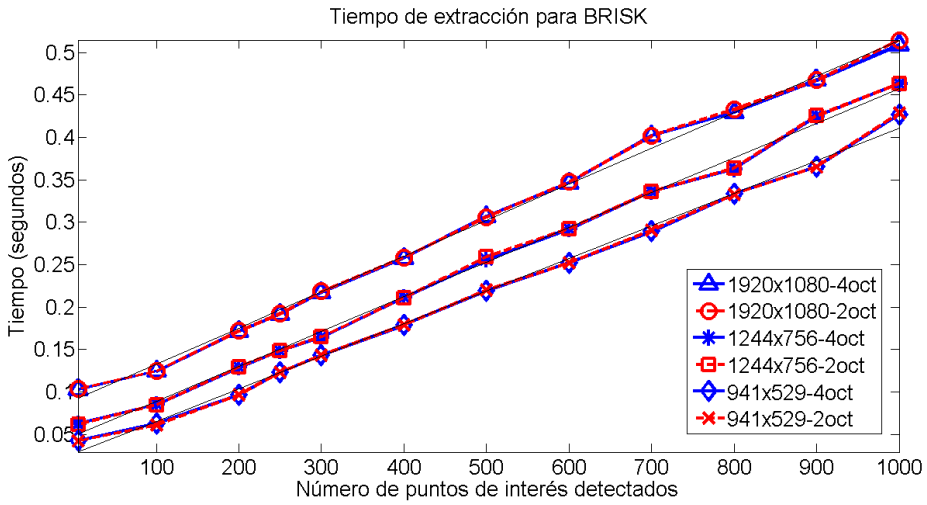


Figura 19: tiempos de extracción de descriptores para BRISK. Las rectas finas representan regresiones lineales para *sky_large*, *sky_medium* y *sky_small* con detectores de 4 octavas.

Las figuras 16, 17, 18 y 19 muestran que los tiempos de detección y extracción son lineales en el número de puntos de interés. Además, estos tiempos son también lineales en el tamaño de la imagen en pixels (asumiendo el mismo número de puntos de interés). Las cuatro figuras siguientes, de la 20 a la 23, confirman esta idea.

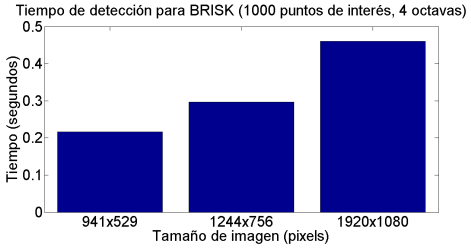


Figura 20: tiempos de detección de BRISK, para 1000 puntos de interés en 4 octavas.

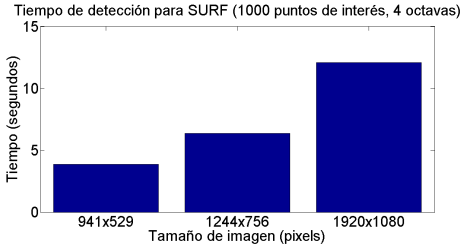


Figura 21: tiempos de detección de SURF, para 1000 puntos de interés en 4 octavas.

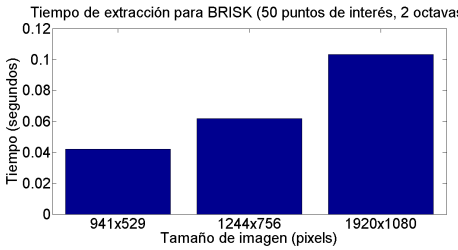


Figura 22: tiempos de extracción de BRISK, para 50 puntos de interés en 2 octavas.

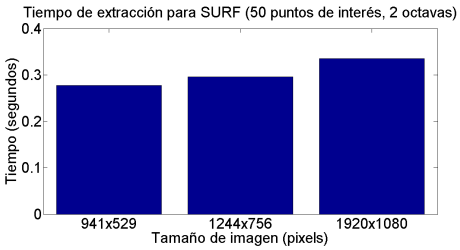


Figura 23: tiempos de extracción de SURF, para 50 puntos de interés en 2 octavas.

## 4.2 Experimentos de transmisión con control de errores

Antes de la implementación del protocolo de retransmisiones, los XSticks estaban limitados a un *baud rate* o tasa de baudios seleccionable de 9600 debido a la pérdida de paquetes que se producía si se usaba una tasa mayor. Esto llevaba a un *throughput* efectivo de aproximadamente 4.2 kbits/s. Con el diseño explicado en el apartado 3.2, el sistema trabaja ahora con un protocolo de transmisión con control de errores. Esto permite incrementar la *Serial Interface Data Rate* seleccionable de los XSticks.

Los diferentes valores que este valor, modificable por software, puede tomar son 1200, 2400, 4800, 9600, 19200, 38400, 57600 y 115200. Se experimentó con todos ellos enviando *sky_large*, *sky_medium* y *sky_small* para cada uno y se midió el tiempo que costaba transmitirlas, incluyendo las retransmisiones de paquetes perdidos y recepciones de *acknowledgements*. Se transmitió cada imagen 10 veces y se calculó la media de los tres resultados. Llevamos a cabo estos experimentos transmitiendo con los XSticks conectados a un ordenador portátil primero y conectados a los BeagleBones después, en la forma descrita en la figura 1. La figura 24 muestra los resultados.

Para ilustrar mejor el proceso, se detalla a continuación la transmisión de *sky_small* en el ordenador portátil con 115200 como *baud rate*. *Sky_small*, después de la codificación y empaquetado, ocupa 1478 paquetes. Los primeros 127 paquetes ocupan 99 bytes (la cabecera necesita un byte menos para el número de secuencia del paquete), el último paquete ocupa solo 58 bytes y el resto 100 bytes. El número total de bytes transmitidos es 127*99 + (1478-128)*100 + 58 = 147631 bytes, ó 1181048 bits. El tiempo necesario para completar la transmisión es aproximadamente 99.0 segundos. Esto resulta en un *throughput* de aproximadamente 11930 bps, con un margen de error de ± 6 bps. Este proceso se repite 10 veces para cada imagen y la media es obtenida.

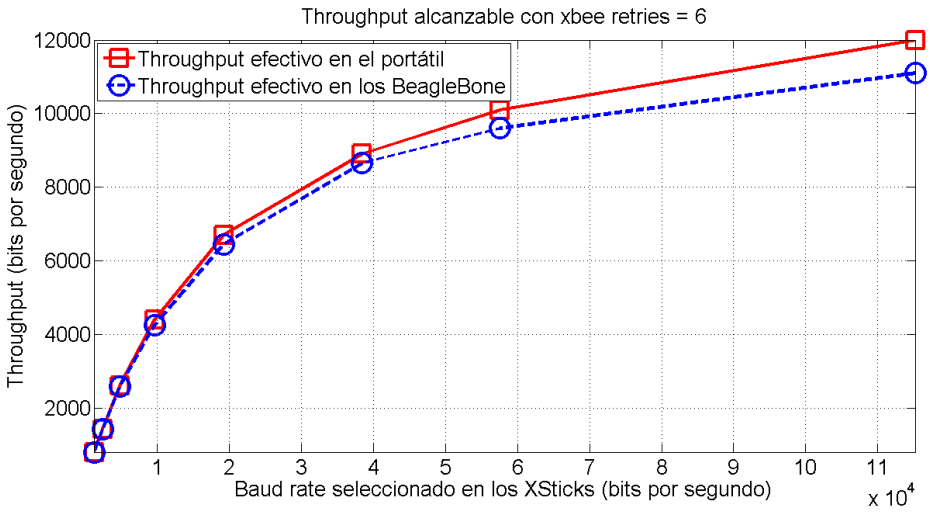


Figura 24: *Throughput* alcanzable con el parámetro *xbee retries* igual a 6.

El máximo *throughput* conseguido en los BeagleBones es aproximadamente 11.1 kbps. Este valor es más pequeño que el conseguido realizando el experimento con el ordenador portátil debido a que en la interfaz entre los BeagleBones y los XSticks se producen más pérdidas de paquetes que entre el ordenador y los XSticks. La razón por la que esto pasa es desconocida para el autor de este proyecto, aunque se pueda deducir que tiene relación con las especificaciones técnicas diferentes de un ordenador portátil y de un BeagleBone.

A medida que incrementamos el *baud rate*, el *throughput* efectivo se distancia del máximo teórico. Esto ocurre porque el porcentaje de paquetes que se pierden crece. La configuración de los XSticks (con *xbee retries* igual a 6) y el nuevo protocolo de transmisión se aseguran de que todos los paquetes lleguen correctamente a su destino. Pero este mismo intento de proporcionar fiabilidad explica la pendiente decreciente de ambas curvas.

Para dar un medida más precisa de la actuación del protocolo de transmisión, se cambió el parámetro de *xbee retries* de la configuración de los XSticks a 0. De este modo, se tienen más paquetes perdidos que con el anterior valor (6). Este cambio también resultó en un incremento del *throughput* final, llegando hasta un máximo de 32280 bps. La figura 25 muestra cómo varía el nuevo *throughput* en los BeagleBones y da una medida de la ”bondad” del protocolo de transmisión en una manera que se detalla a continuación.

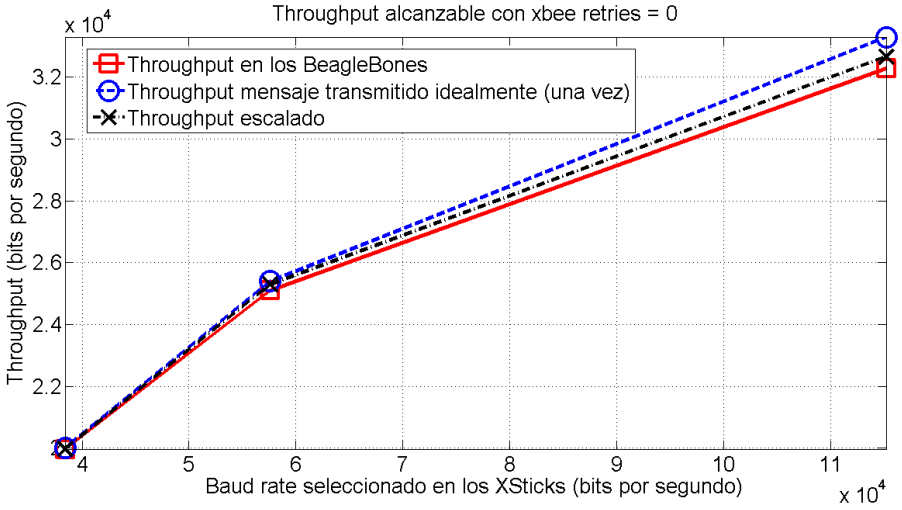


Figura 25: *Throughput* alcanzable con el parámetro *xbee retries* igual a 0.

A continuación, se da una medida de la cantidad de paquetes perdidos para diferentes *baud rate* (y *xbee retries* = 0). Solo se enuncian las más significativas. Para 38400, el 0.1 % de los paquetes se pierden y necesitan reenviarse, para 57600, el 0.6 % y para 115200 el 2 %. En el último caso, esto significa 32 paquetes perdidos al enviar *sky_small*. Este porcentaje se denomina *frame loss rate*.

En la figura 25 tenemos tres curvas diferentes. El *throughput* real alcanzable en los BeagleBones corresponde a la curva roja. Para un *baud rate* de 115200 en los XSticks, tenemos un *throughput* de 32280 bps. La curva azul se calcula utilizando los tiempos que tardarían las imágenes en ser transmitidas idealmente, es decir, todos los paquetes se reciben correctamente a la primera, y no se tiene en cuenta el tiempo que el sender espera el *acknowledgement*. Por último, la curva negra es el *throughput* escalado usando el *frame loss rate*. Los tiempos utilizados para calcular la curva azul se escalan por $1/(1-p)$, donde $p$ es la probabilidad de pérdida durante la transmisión o *frame loss rate*.

La relación entre las tres curvas es la siguiente. La curva roja corresponde al *throughput* real y, por tanto, ha sido calculada usando los tiempos reales que el sistema tardaba en completar la transmisión de cada imagen, incluidas retransmisiones. La curva azul corresponde al *throughput* que se consiguiría si el *frame loss rate* fuera cero, es decir, no hay necesidad de retransmisiones de paquetes, y obviando a su vez el tiempo necesario para la llegada del *acknowledgement*. Esto se hace así porque si esos tiempos se escalan por $1/(1-p)$, siendo $p$ el *frame loss rate* del sistema, los valores resultantes corresponden a los tiempos que tardarían todos los paquetes en transmitirse correctamente, incluidas retransmisiones, y obviando una vez más los *acknowledgements*. Por lo tanto, la diferencia entre las curvas azul y negra ilustra la influencia que tienen las pérdidas de datos en el sistema y en su *throughput* fi-

nal. La diferencia entre entre la curva negra (*throughput* escalado) y la curva roja (*throughput* real) indica la disminución del *throughput* debido a la implementación del protocolo. En otras palabras, nos muestra cuánto tiempo espera el sistema a los distintos *acknowledgements*, *timeouts*, etc., que permiten una transmisión libre de errores.

# 5. Conclusiones

En este proyecto se ha llevado a cabo la integración de un nuevo algoritmo de extracción de características de imagen (BRISK) en un sistema que realiza este proceso de forma distribuida en varios nodos. Esto ha permitido llevar a cabo una comparación del tiempo que tarda el algoritmo que ya estaba implementado en el sistema (SURF) y BRISK en realizar esta extracción de características que facilitan el reconocimiento de objetos dentro de la imagen. Dado que SURF y BRISK producen resultados similares en cuanto a precisión y tasa de reconocimiento (esto no ha sido estudiado en este proyecto) y que BRISK es mucho más rápido que SURF en las fases de detección y de extracción, se puede concluir que BRISK es una alternativa preferible si se quiere desarrollar cualquier aplicación de reconocimiento de objetos.

Por otro lado, se ha diseñado e implementado un protocolo de transmisión que garantiza una transmisión de datos libre de errores que se utiliza para la comunicación entre los distintos nodos del sistema. Antes de empezar el trabajo de este proyecto, el sistema sufría pérdidas de datos de transmisión si se utilizaba un *baud rate* superior a 9600 en los XSticks, de manera que el máximo *throughput* alcanzable sin errores era de 4.2 kbits por segundo. Con las modificaciones introducidas, y seleccionando los parámetros adecuados de actuación del protocolo (*timeouts* y número de reintentos), no se producen pérdidas de datos en la transmisión y el *throughput* que se consigue ha aumentado hasta aproximadamente 33 kbits por segundo.

Por tanto, si se junta la ganancia en tiempo que se produce con el nuevo algoritmo de extracción con la que se produce con el nuevo protocolo de transmisión, el programa completa su ejecución en una mínima parte del tiempo que antes necesitaba. Además, el protocolo de transmisión desarrollado en este proyecto puede ser adaptado fácilmente para funcionar con otras tecnologías de transmisión inalámbrica, como puede ser Wi-Fi. De hecho, este es el objetivo de otro Master's Thesis Project en la KTH de Estocolmo. Para facilitar la continuidad del sistema en este nuevo proyecto, se llegó a un acuerdo en la forma de funcionamiento del protocolo (por ejemplo, se definió un número de secuencia máximo, Nmax, para los *SendItems*, y cada vez que se recibía uno correctamente, había que marcar los anteriores Nmax/2 como recibidos, para evitar futuras ambigüedades) y en la definición de los tipos ASN.1 empleados.

La realización satisfactoria de estas dos mejoras permite al sistema considerar la posibilidad de trabajar con aplicaciones en tiempo real. Sin embargo, antes de ser capaz de hacerlo, se debería introducir una mejora adicional. El sistema debería ser capaz de conseguir un reparto igualitario en la carga computacional de cada nodo. Para ello, y como se explica en [8] y [9], un algoritmo de predicción para el *threshold* o umbral de detección es esencial para controlar el tiempo que se tarda en completar el análisis de una imagen, tiempo que es función de la cantidad de

información relevante que la imagen posee y de su distribución espacial. En [8] y en [9] se presentan algunas alternativas y se aplican en un entorno de aplicación en tiempo real, concretamente, en dos vídeos de vigilancia.

# Anexos

## A: Recorrido histórico por los algoritmos de extracción

### *A.1 Detectores de puntos de interés*

Uno de los primeros algoritmos de detección fue el detector de esquinas de Moravec [10], propuesto en 1980. Moravec considera una ventana local en la imagen y los cambios en su respuesta cuando la ventana se mueve una pequeña cantidad en varias direcciones. Si el movimiento de la ventana en cualquier dirección resulta en una gran cambio, entonces el área de la imagen cubierta por la ventana es una esquina. El operador de Moravec tiene algunas limitaciones: su respuesta es anisotrópica porque solamente se considera un conjunto discreto de movimientos cada 45 grados; también es ruidosa, debido al uso de una ventana binaria y rectangular y, por último, el operador responde demasiado fácilmente a los bordes que no son esquinas.

El detector de esquinas de Moravec es la base de uno de los detectores más ampliamente utilizados posteriormente: el detector de esquinas de Harris [11], introducido en 1988. Este detector resuelve los problemas previos de los que adolecía el detector de Moravec: su respuesta es isotrópica debido al uso de una expansión analítica sobre el origen de los cambios de la ventana y al uso de una ventana circular suavizada (Gaussiana). Además, el detector de Harris también es capaz de diferenciar bordes, con lo que se convierte en un detector de esquinas y bordes combinado.

La principal desventaja del detector de Harris es que no es invariante respecto a cambios en escala. Este problema sería solucionado cuando Lindeberg [12] introdujera en 1998 el concepto de selección automática de escala. El objetivo de este mecanismo es la estimación del tamaño aproximado de las estructuras de la imagen que son detectadas. Mientras que los detectores previos eran detectores de esquinas, el de Lindeberg es un detector de regiones o de gotas, traducido literalmente (*blob detector* en inglés). Hasta entonces, este tipo de detectores operaba solamente en una escala predeterminada favoreciendo las estructuras de ese tamaño. Lindeberg trabajó con el determinante de la matriz hessiana y el Laplaciano (la traza de esa matriz) para detectar estructuras con forma de gota para todos los tamaños.

En esta misma línea de trabajo, Lowe [13] propuso aproximar el operador laplaciano de Gauss (LoG en inglés) con un filtro de diferencia Gaussiana (DoG), como Crowley & Parker [14] and Lindeberg [15] habían propuesto previamente para otros propósitos. Lowe selecciona los lugares clave en la imagen que corresponden a los máximos y mínimos de la función de diferencia Gaussiana aplicada a cada escala. Esto se calcula eficientemente construyendo una pirámide de la imagen con un re-muestreo entre cada nivel. Los máximos y los mínimos son determinados comparando cada pixel en la pirámide a sus vecinos, primero a los 8 de su mismo nivel

de la pirámide y luego al pixel más cercano del siguiente nivel. La mayoría de los pixels son eliminados tras pocas comparaciones y es esto lo que trae eficiencia a este método.

[16] y [17] son dos comparaciones publicadas de los detectores existentes. Argumentan que los detectores basados en la matriz hessiana son ligera pero sistemáticamente mejores que sus competidores basados en el detector de Harris. Adicionalmente, enuncian que el uso del determinante de la matriz hessiana es preferible a usar su traza, que detecta estructuras mucho más alargadas. Por último, también muestran que aproximaciones como DoG pueden traer velocidad gracias a un pequeño coste en términos de precisión.

### *A.2 Descriptores asociados a puntos de interés*

Existe una gran variedad de descriptores en la literatura, como derivadas Guassianas [18], momentos (media ponderada) invariantes [19], características complejas [20], [21], filtros dirigibles [22] y características locales basadas en fase [23].

Sin embargo, uno de los descriptores más usados es SIFT, introducido por Lowe [24]. Este descriptor representa la distribución de características a una escala menor dentro de la vecindad del punto de interés. Específicamente, SIFT calcula un histograma de gradientes localmente orientados alrededor de los puntos de interés y guarda los valores en un vector de 128 dimensiones (8 valores de orientación por cada 4x4 valores de localización).

Los descriptores SIFT han demostrado ser invariantes a rotaciones de imagen y escala y robustos en un rango substancial de distorsión afín, adición de ruido y cambios en iluminación. Un gran número de puntos de interés son extraídos, lo que lleva a robustez extrayendo objetos pequeños entre otros no deseados. Los puntos de interés son detectados a lo largo de un gran rango de escalas, de manera que hay disponibles características pequeñas y locales en la imagen para objetos pequeños y parcialmente ocultos, mientras que los puntos de interés de mayor tamaño también funcionan bien para imágenes sujetas a ruido y borrosidad.

El descriptor SIFT, además de ser uno de los más usados, es la base de una gran variedad de refinamientos y mejoras en nuevos descriptores.

### *A.3 Detector y descriptor SURF*

Speeded-Up Robust Features (SURF) [3] es un detector y descriptor invariante a escala y rotación que supera a los esquemas anteriormente propuestos en cuanto a repetibilidad, distintividad y robustez, con un tiempo de computación y compa-

ración con la base de datos menor.

El detector SURF usa una aproximación muy básica de una matriz hessiana. Esto conlleva el uso de imágenes integrales [25], lo que reduce el tiempo de cálculo drásticamente. Las imágenes integrales son parte de la idea más general de cajas o *boxlets*, propuesta por Simard *et al.* [26].

El uso de imágenes integrales lleva a una rápida computación de filtros de convolución de tipo caja. La entrada de una imagen integral $I_\Sigma(\mathbf{x})$ en un punto $\mathbf{x} = (x, y)^T$ representa la suma de todos los pixels en la imagen de entrada $I$ dentro de una región rectangular formada por el origen y $\mathbf{x}$. Una vez que la imagen integral ha sido calculada, únicamente se requieren tres sumas para calcular la suma total de las intensidades sobre cualquier área rectangular, y así el tiempo de cálculo es independiente de su tamaño, lo que es importante cuando se usan filtros de gran tamaño.

La matriz hessiana, $\mathcal{H}(\mathbf{x}, \sigma)$, en un punto $\mathbf{x}$ a escala $\sigma$, puede ser definida como:

$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix},$$

donde $L_{xx}(\mathbf{x}, \sigma)$ es la convolución de la derivada de segundo orden de la función Gaussiana con la imagen $I$ en $\mathbf{x}$, y de igual manera para $L_{xy}(\mathbf{x}, \sigma)$ y $L_{yy}(\mathbf{x}, \sigma)$.

Las funciones Gaussianas son aquí discretizadas y truncadas, lo que conlleva una pérdida de repetibilidad para rotaciones de imagen alrededor de los múltiplos impares de $\pi/4$. Esta debilidad está presente en todos los detectores basados en matrices hessianas pero no sobrepasa en importancia la ventaja de las convoluciones rápidas traída por la discretización y el truncamiento. Debido al éxito de Lowe con su aproximación LoG y al hecho de que los filtros reales son no ideales en cualquier caso, SURF usa otra aproximación; la matriz hessiana es construida con filtros de caja. Así, SURF aproxima las derivadas de segundo orden de la función Gaussiana a un coste computacional muy bajo usando imágenes integrales. Son llamadas $D_{xx}$, $D_{yy}$ y $D_{xy}$, y son usadas para calcular el determinante aproximado de la matriz hessiana:

$$\det(\mathcal{H}_{approx}) = D_{xx}D_{yy} - (\omega D_{xy})^2,$$

donde $\omega$ es un peso relativo a la respuesta del filtro necesario para la conservación de energía entre el núcleo Gaussiano y sus aproximaciones.

Este valor representa la respuesta de una imagen en un punto $\mathbf{x}$. Estas respuestas son guardadas en un mapa de respuestas para diferentes escalas, donde los máximos

locales serán detectados, usando una variante rápida de la supresión de no máximos (*non-maximum suppression*), introducida por Neubeck y Van Gool [27]. Los máximos son entonces interpolados en escala y posición de acuerdo al método propuesto por Brown *et al.* [28].

En vez de construir una pirámide de imágenes para obtener invarianza de escala, SURF reescala el tamaño del filtro. El espacio de escala es pues dividido en octavas. Una ocatava representa una serie de mapas de respuestas de filtros obtenidas convolucionando la misma imagen de entrada con un filtro de tamaño creciente. Una octava abarca un factor 2 de escala y es subdividida en un número constante de escalas.

El siguiente objetivo del algoritmo SURF es el cálculo de los descriptores. SURF usa un aproximación parecida a SIFT [24], describiendo la distribución de intensidad dentro de la proximidad del punto de interés. SURF encuentra una orientación reproducible para los puntos de interés para alcanzar invarianza de rotación. Esto es conseguido calculando las respuestas de onda Haar (*Haar wavelet responses*) en las direcciones x e y dentro de un círculo de proximidad, usando otra vez imágenes integrales para filtrar rápidamente y usando solo 64 dimensiones. Las respuestas son entonces ponderadas con una función Gaussiana centrada en el punto de interés, y más tarde representadas como puntos en un espacio con la respuestas horizontal a lo largo del eje de abscisas y la respuesta vertical a lo largo del de ordenadas. Usando una ventana deslizante de tamaño $\pi/3$, todas las respuestas dentro de la ventana son sumadas y producen un vector de orientación local. De esta manera, el vector más largo de todas las ventanas usadas define la orientación del punto de interés.

El siguiente paso es extraer los descriptores. Se construye una región cuadrada centrada en el punto de interés y orientada en la dirección previamente calculada. Esta región es dividida en sub-regiones cuadradas de 4x4 para preservar importante información espacial. Las respuestas de onda Haar horizontales y verticales en puntos separados regularmente cada 5x5 muestras son calculados y ponderados con una función Guassiana centrada en el punto de interés. Además, la suma de los valores absolutos de las respuestas se calcula también para conseguir información sobre la polaridad de los cambios de intensidad. Por lo tanto, cada sub-región tiene un vector descriptor de cuatro dimensiones $\mathbf{v} = (\Sigma d_x, \Sigma d_y, \Sigma|d_x|, \Sigma|d_y|)$, que representa la estructura de intensidad subyacente. Para cada sub-región 4x4, esto resulta en un vector descriptor de longitud 64, invariante a iluminación (offset), y la invarianza al contraste se alcanza convirtiendo el vector en un vector unitario.

La última tarea a realizar es el paso de correspondencia de los descriptores. Para una indexación rápida en esta fase se incluye el signo del Laplaciano. No hay que calcularlo ya que se ha guardado durante la fase de detección. Este signo distingue entre regiones brillantes sobre fondos oscuros de la situación inversa, lo que es usado para comparar características solo con este tipo de contraste, alcanzando una fase

de correspondencia más rápida.

Como conclusión, la actuación del detector SURF es ligeramente mejor que otros detectores, siendo sin embargo su velocidad su punto fuerte. Además, el descriptor SURF también sobrepasa al resto de descriptores en términos de precisión y velocidad, debido a su descripción de la naturaleza subyacente de los patrones de intensidad en la imagen y al uso de imágenes integrales y la estrategia de indexación basada en el Laplaciano.

### *A.4 Descriptor BRIEF*

BRIEF [5] nace de la necesidad de que los descriptores sean rápidos de computar, eficientes en memoria y exhiban la precisión adecuada para ser usados en dispositivos móviles con capacidad computacional limitada. BRIEF es un descriptor binario, basado en simples tests de diferencia de intensidad que produce resultados comparables en cuanto a precisión a SIFT y SURF siendo mucho más rápido. Existen varias técnicas propuestas en la literatura para acelerar el proceso de encontrar correspondencias y reducir consumo de memoria, como reducción dimensional (por ejemplo, Principal Component Analysis [29]), cuantización usando algunos bits de los descriptores existentes [30], [31], [32] e incluso binarización [33], [34]. Pero todas estas aproximaciones requieren primero la computación del descriptor entero mientras que BRIEF computa directamente los vectores binarios para trozos de imágenes, como se muestra en [35], [36].

BRIEF define el test $\tau$ en el área $\mathbf{p}$ de tamaño $S$ x $S$ como:

$$\tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if } I(\mathbf{p},\, \mathbf{x}) < I(\mathbf{p},\, \mathbf{y}) \\ 0 & \text{otherwise,} \end{cases}$$

donde $I(\mathbf{p},\, \mathbf{x})$ es la intensidad de pixel en una versión suavizada de $\mathbf{p}$ en $\mathbf{x} = (u, v)^T$. Elegir un conjunto de $n_d$ $(\mathbf{x},\, \mathbf{y})$-pares de localizaciones define de manera unívoca un conjunto de tests binarios. El descriptor BRIEF es el vector de bits $n_d$-dimensional que corresponde a la siguiente expresión en su equivalente decimal:

$$\sum_{1 \leq i \leq n_d} 2^{i-1} \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i)$$

Por último, para la fase de establecer correspondencias, se usan distancias Hamming, lo que confirma la validez de [37] and [38], que recomiendan abandonar la distancia euclídea para este propósito.

## B: Parámetros de los XSticks

Los parámetros de los XSticks se pueden modificar usando el programa X-CTU, que se pueden descargar de la página web de Digi: `http://www.digi.com/support/productdetail?pid=3352&osvid=57&type=utilities`. Solo está disponible para Windows. La configuración empleada en los experimentos se especifica en la tabla 1.

El *Serial Interface Data Rate* puede variar desde 1200 hasta 115200 bps. Este valor tiene que ser ajustado en los XSticks, pero también en el código fuente C++, concretamente en *dfe_typedefs.h*, con una orden #define BAUDRATE.

Se usaron diferentes configuraciones para los experimentos. Concretamente, dos parámetros fueron modificados. El primero es el *Serial Interface Data Rate*, que tomó todos los valores posibles: 1200, 2400, 4800, 9600, 19200, 38400, 57600 y 115200 bps. El segundo es *xbee retries*, que cambió su valor de 6 a 0, para obtener un mejor entendimiento de la actuación del protocolo de transmisión fiable. Cada combinación de diferentes valores para estos dos parámetros produce diferentes valores en el *throughput* y en el porcentaje de pérdida de paquetes, como se detalla en el apartado 4.2.

| Field | AT command | Value |
|---|---|---|
| Channel | CH | C |
| Pan id | ID | 3332 |
| Destination address high | DH | 0 |
| Destination address low | DL | 0 |
| 16 bit source address | MY | FFFF |
| Mac mode | MM | 0 |
| xbee retries | RR | 6-0 |
| Random delay slots | RN | 0 |
| Node discovery time | NT | 19 |
| Node discovery options | NO | 0 |
| Coordinator enabled | CE | 0 |
| Scan channels | SC | 1FFE |
| Scan duration | SD | 4 |
| End device association | A1 | 0 |
| Coordination association | A2 | 0 |
| AES encryption enabled | EE | 0 |
| Node identifier | NI | |
| Power level | PL | 4 |
| CCA Threshold | CA | 2C |
| Sleep mode | SM | 0 |
| Time before sleep | ST | 1388 |
| Cyclic sleep period | SP | 0 |
| Disassociated cyclic sleep period | DP | 3E8 |
| Sleep options | SO | 0 |
| Interface data rate | BD | 1200-2400-4800-9600-19200 38400-57600-115200 |
| Parity | NB | 0 |
| Packetization timeout | RO | 3 |
| API enabled | AP | 1 |
| Pin settings | D8 | 0 |
| Pin settings | D7 | 1 |
| Pin settings | D6 | 0 |
| Pin settings | D5 | 1 |
| Pin settings | D4 - D0 | 0 |
| Pull up resistor enbaled | PR | FF |
| I/O Enabled | IU | 1 |
| Samples before TX | IT | 1 |
| DIO change detect | IC | 0 |
| Sample rate | IR | 0 |
| PWM0 Configuration | P0 | 1 |
| PWM1 Configuration | P1 | 0 |
| PWM Output timeout | PT | FF |
| RSSI PWM Times | RP | 28 |
| I/O Input adresses | IA | FFFFFFFFFFFFFFFF |
| T0 - DO to T7 - D7 Output addresses | T0 | FF |
| Device type identifier | DD | 10000 |
| AT Command Mode Timeout | CT | 64 |
| Guard Time | GT | 3E8 |
| Command Sequence character | CC | 2B |

Tabla 1: Parámetros de los XStick

## C: Tipos ASN.1

Tipos ASN.1 usados para la estructura de los paquetes:

```
PktHeaderAsn ::= SEQUENCE {
    pktnum        INTEGER,
    totpkts       INTEGER,
    messageID     INTEGER,
    typeheaderasn BIT STRING
}

TypeHeaderAsn ::= CHOICE {
    imageasn          ImageAsn,
    imgtask           ImgTaskAsn,
    taskfinishedasn   TaskFinishedAsn,
    reportnewnodeasn  ReportNewNodeAsn,
    deletenodeasn     DeleteNodeAsn,
    takepicture       TakePicture,
    receptionreport   ReceptionReportMessage
    reportrequest     ReceptionReportRequestMessage
}
```

Tipos ASN.1 usados para los descriptores:

```
SurfParametersAsn ::= SEQUENCE {
    extended          BOOLEAN,
    hessianthreshold  REAL,
    noctavelayers     INTEGER,
    noctaves          INTEGER,
    upright           BOOLEAN
}

BriskParametersAsn ::= SEQUENCE {
    thresh            REAL,
    octaves           INTEGER,
    rotationInvariant BOOLEAN,
    scaleInvariant    BOOLEAN,
    patternScale      REAL
}

ParametersAsn ::= CHOICE {
    surfparameters  SurfParametersAsn,
    briskparameters BriskParametersAsn
}

AlgorithmParametersAsn ::= SEQUENCE {
    detectionparameters   ParametersAsn,
    extractionparameters  ParametersAsn
}
```

Tipos ASN.1 usados para el esquema de retransmisiones:

```
ReceptionReportMessage ::= SEQUENCE {
    mode            ReceptionReportModes,
    droppedPackets  SEQUENCE OF INTEGER
}

ReceptionReportModes ::= ENUMERATED {
    specifyEach,
    specifyIntervals
}

ReceptionReportRequestMessage ::= SEQUENCE {
    messageID  INTEGER
}
```

## Referencias


[1] Drisgill, P. "Introduction to ASN.1." *International Telecommunication Union (ITU).* 21/04/2013. 12/12/2013 <`http://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx`>

[2] Leutenegger, S., Chli, M., & Siegwart, R.Y. "BRISK: Binary Robust Invariant Scalable Keypoints." *Proc. International Conference on Computer Vision (ICCV) pp. 2548 - 2555. IEEE (2011).*

[3] Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. "Speeded-up robust features (SURF)." *Computer Vision and Image Understanding, vol. 110, no. 3, pp. 346 - 359, 2008.*

[4] A. Brykt, "A testbed for distributed detection of keypoints and extraction of descriptors for the Speeded-Up-Robust-Features (SURF) algorithm." *Master's Thesis Project, Kungliga Tekniska Högskolan, Stockholm, 2013.*

[5] Calonder, M., Lepetit, V., Strecha, C., & Fua, P. "BRIEF: Binary robust independent elementary features." *Proc. of ECCV, 2010.*

[6] Adams, J. "An Introduction to IEEE STD 802.15.4." *Aerospace Conf., IEEE, 2006.*

[7] Mair, E., Hager, G.D., Burschka, D., Suppa, M., & Hirzinger, G. "Adaptive and generic corner detection based on the accelerated segment test." *Proceedings of the European Conference on Computer Vision (ECCV), 2010.*

[8] Eriksson, E., Dán, G., & Fodor, V. "Prediction-based Load Control and Balancing for Feature Extraction in Visual Sensor Networks." *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), May 2014.*

[9] Eriksson, E., Dán, G., & Fodor, V. "Real-time Distributed Visual Feature Extraction from Video in Sensor Networks." *Proc. of IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS), May 2014.*

[10] Moravec, H. "Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover." *Tech Report CMU-RI-TR-3, Carnegie-Mellon University, Robotics Institute, September 1980.*

[11] Harris, C., & Stephens, M. "A combined corner and edge detector." *Proceedings of the Alvey Vision Conference, pages 147 - 151, 1988.*

[12] Lindeberg, T. "Feature detection with automatic scale selection." *IJCV, 30(2):70 - 116, 1998.*

[13] Lowe, D. "Object recognition from local scale-invariant features." *ICCV, 1999.*

[14] Crowley, J.L., & Parker, A.C. "A representation for shape based on peaks and ridges in the difference of low-pass transform." *IEEE Trans. On Pattern Analysis and Machine Intelligence, 6, 2 (1984), pp. 156 - 170.*

[15] Lindeberg, T. "Detecting salient blob-like image structures and their scales with a scale-space primal sketch: a method for focus-of-attention." *International Journal of Computer Vision, 11, 3 (1993), pp. 283 - 318.*

[16] Mikolajczyk, K., & Schmid, C. "Scale and affine invariant interest point detectors." *IJCV, 60(1):63 - 86, 2004.*

[17] Mikolajczyk, K., & Schmid, C. "A performance evaluation of local descriptors." *PAMI, 27(10):1615 - 1630, 2005.*

[18] Florack, L.M.J., ter Haar Romeny, B.M., Koenderink, J.J., & Viergever, M.A. "General intensity transformations and differential invariants." *JMIV, 4(2):171 - 187, 1994.*

[19] Mindru, F., Tuytelaars, T., Van Gool, L., & Moons, T. "Moment invariants for recognition under changing viewpoint and illumination." *CVIU, 94(1-3):3 - 27, 2004.*

[20] Baumberg, A. "Reliable feature matching across widely separated views." *CVPR, pages 774 - 781, 2000.*

[21] Schaffalitzky, F., & Zisserman, A. "Multi-view matching for unordered image sets, or How do I organize my holiday snaps?" *ECCV, volume 1, pages 141 - 431, 2002.*

[22] Freeman, W.T., & Adelson, E.H. "The design and use of steerable filters." *PAMI, 13(9):891 - 906, 1991.*

[23] Carneiro, G., & Jepson, A.D. "Multi-scale phase-based local features." *CVPR (1), pages 736 - 743, 2003.*

[24] Lowe, D. "Distinctive image features from scale-invariant keypoints, cascade filtering approach." *IJCV, 60(2):91 - 110, January 2004.*

[25] Viola, P.A., & Jones, M.J. "Rapid object detection using a boosted cascade of simple features." *CVPR (1), pages 511 - 518, 2001.*

[26] Simard, P., Bottou, L., Haffner, P., & LeCun, Y. "Boxlets: A fast convolution algorithm for signal processing and neural networks." *NIPS, 1998.*

[27] Neubeck, A., & Van Gool, L. "Efficient non-maximum suppression." *ICPR, 2006.*

[28] Brown, M., & Low, D. "Invariant features from interest point groups." *BMVC, 2002.*

[29] Mikolajczyk, K., & Schmid, C. "A Performance Evaluation of Local Descriptors." *IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 27, no. 10, pp. 1615 - 1630, Oct. 2004.*

[30] Tuytelaars, T., & Schmid, C. "Vector Quantizing Feature Space with a Regular Lattice." *Proc. Intel Conf. Computer Vision, 2007.*

[31] Winder, S., Hua, G., & Brown, M. "Picking the Best Daisy." *Proc. IEEE Conf. Computer Vision and Pattern Recognition, June 2009.*

[32] Calonder, M., Lepetit, V., Konolige, K., Bowman, J., Mihelich, P., & Fua, P. "Compact Signatures for High-Speed interest Point Description and Matching." *Proc. IEEE Intel Conf. Computer Vision, Sept. 2009.*

[33] Shakhnarovich, G. "Learning Task-Specific Similarity." *PhD dissertation, Massachusetts Inst. of Technology, 2005.*

[34] Strecha, C., Bronstein, A., Bronstein, M., & Fua, P. "LDAHash: Improved Matching with Smaller Descriptors." *IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 34, no. 1, pp. 66 - 78, Jan. 2012.*

[35] Ozuysal, M., Calonder, M., Lepetit, V., & Fua, P. "Fast Keypoint Recognition Using Random Ferns." *IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 32, no. 3, pp. 448-461, Mar. 2010.*

[36] Lepetit, V., & Fua, P. "Keypoint Recognition Using Randomized Trees." *IEEE Trans. Pattern Analysis an Machine Intelligence, vol. 28, no. 9, pp. 1465 - 1479, Sept. 2006.*

[37] Shakhnarovich, G., Viola, P., & Darrell, T. "Fast Pose Estimation with Parameter-Sensitive Hashing." *Proc. IEEE Intel Conf. Computer Vision, 2003.*

[38] Torralba, A., Fergus, R., & Weiss, Y. "Small Codes and Large Databases for Recognition." *Proc. IEEE Conf. Computer Vision and Pattern Recognition, June 2008.*