PROYECTO DE SISTEMAS INFORMÁTICOS

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

CURSO 2006/2007

CAPTURA Y PROCESADO REMOTO DE IMÁGENES Y VIDEO

PROFESOR DIRECTOR: Dr. Segundo Esteban San Román

Dr. Luis Piñuel Moreno

AUTORES: Alberto Fernández-Corroto de Pazos

Jorge de Iscar González

Carlos Spitzer López

Índice

1. Introducción y objetivos del proyecto 4
2. Hardware empleado 5
    a. DSP de propósito general TMSS320C6713 DSK 5
        i. Acerca de Spectrum Digital Inc. y Texas Instruments 5
        ii. Referencia técnica del C6713 DSK 6
        iii. Diagrama de bloques conceptual 7
        iv. Ruta de datos 7
    b. Sistema DSKEye 9
        i. Acerca de BiTec Ltd. 9
        ii. Referencia técnica 9
        iii. Diagrama de bloques conceptual 12
    c. Circuito emisor de luz 13
        i. Estructura y componentes 13
        ii. Finalidad 13
    d. Receptor inalámbrico multibanda DWL-G122 14
        i. Acerca de D-Link 14
        ii. Referencia técnica del DWL-G122 14
        iii. Configuración genérica de red y enrutamiento de paquetes 15
3. Software empleado 19
    a. CCStudio 3.1.0 19
        i. Instalación y configuración del entorno 19
        ii. Conceptos y funciones principales 20
        iii. Otras referencias 25
    b. Borland C++ Builder 6.0 26
        i. Instalación y configuración del entorno 26
        ii. Conceptos y funciones principales 27
        iii. Otras referencias 29
    c. Visual Studio 2005 30
        i. Instalación y configuración del entorno 30
        ii. Conceptos y funciones principales 30
        iii. Otras referencias 34
    d. Otros programas 35
4. Desarrollo del proyecto 36
    a. Codificación de imágenes estáticas bajo el estándar JPEG 2000 36
        i. ¿De qué partimos? 36
        ii. ¿Cuál es el objetivo? 43
        iii. Estructura del subproyecto y diagramas de bloques 43
        iv. Ejecución de las herramientas desarrolladas 48
        v. Presentación de resultados 49
        vi. Rendimiento y puntos críticos 53
        vii. Cuellos de botella y posibles soluciones 57
        viii. Mejoras o ideas principales a desarrollar en un futuro 58
        ix. Conclusiones 58
    b. Codificación de imágenes en movimiento bajo el estándar H.263 60
        i. ¿De qué partimos? 60
        ii. ¿Cuál es el objetivo? 66

iii. Estructura del subproyecto y diagramas de bloques 68
1. Codificador INTRA 69
2. Codificador INTER 88
3. Diagramas de bloques y estructura funcional del proyecto 91
iv. Ejecución de las herramientas desarrolladas 94
v. Presentación de resultados, rendimiento y puntos críticos 96
vi. Cuellos de botella y posibles soluciones 99
vii. Mejoras o ideas principales a desarrollar en un futuro 101
viii. Conclusiones 102
c. Detección de bordes y LEDs 104
i. ¿De qué partimos? 104
ii. ¿Cuál es el objetivo? 104
iii. Estructura del subproyecto y diagramas de bloques 104
iv. Compilación y generación de código objeto 114
v. Ejecución de las herramientas desarrolladas 115
vi. Presentación de resultados, rendimiento y puntos críticos 116
vii. Cuellos de botella y posibles soluciones 121
viii. Mejoras o ideas principales a desarrollar en un futuro 122
ix. Conclusiones 122
5. Bibliografía 123
6. Anexos 125
a. Consideraciones legales y transferencia de derechos 125

Introducción y objetivos del proyecto

Como proyecto de Sistemas Informáticos para el curso académico 2006/2007 inicialmente se plantearon diferentes objetivos que abarcaban un amplio abanico de tareas a realizar conjuntamente entre los tres componentes del grupo. Como elementos base se disponía de una cámara fotosensible y un DSP de propósito general. Una vez determinadas las posibilidades que dichos elementos Hardware nos ofrecían, se fijaron los siguientes objetivos:

1. Ser capaces de tomar, procesar, codificar y representar imágenes estáticas utilizando los componentes Hardware especificados anteriormente. Para ello se implementará sobre el DSP cualquier estándar ISO a efecto.
2. Ser capaces de capturar, codificar y mostrar video en tiempo real utilizando los componentes Hardware especificados anteriormente. Para ello se implementará sobre el DSP cualquier estándar fijado por la UIT (Unión Internacional de Telecomunicaciones).
3. Ser capaces de reconocer, localizar, procesar y tratar imágenes estáticas o fragmentos de las mismas utilizando los componentes Hardware especificados anteriormente. Para ello se implementarían sobre el DSP diferentes algoritmos prácticos utilizados en teoría de visión por computador y robótica avanzada.

Por tanto, era necesario repartir trabajo entre los tres componentes del grupo y encargar cada subproyecto a un determinado alumno. En concreto, Alberto Fernández-Corroto se encargó de gestionar toda la parte de toma de imágenes estáticas, Carlos Spitzer de la parte de video en tiempo real y Jorge de Íscar de todo lo relacionado con visión por computador y procesamiento de atributos de imágenes estáticas.

Hardware empleado

DSP de propósito general TMSS320C6713 DSK

Acerca de Spectrum Digital Inc. y Texas Intruments

Spectrum Digital Inc. es una empresa dedicada al desarrollo de herramientas de diseño para procesadores digitales de señal y microcontroladores de alto rendimiento. Entre sus productos se encuentran emuladores, depuradores, módulos para evaluación y hardware para aplicaciones de propósito específico. Sus productos están encaminados a facilitar la programación en los DSP a través de un PC, y proporciona soluciones para los DSP's de Texas Instruments, OMAP, DaVinci y microcontroladores en general.

Texas Instruments es una empresa dedicada a la fabricación de chips semiconductores para el procesamiento digital de señal. Es una de las empresas punteras en procesamiento en tiempo real y en innovación en cuanto a diseño y arquitectura de sus productos. En cuanto a DSP's, Texas Instruments produce varias familias distintas de chips, cada una orientada a cierto tipo de tareas:

1. DaVinci™ Digital Media Processors. Diseñados para el procesamiento de imagen y vídeo digital.
    - TMS320DM644x DSPs
    - TMS320DM643x DSPs
    - TMS320DM64x DSPs
2. High Performance TMS320C6000™ DSPs. Soluciones de alto rendimiento con aritmética en punto fijo. Adecuados para proceso de imágenes y audio digital y para trabajar con infraestructuras de banda ancha.
    - TMS320C645x DSPs
    - TMS320C6414T/15T/16T DSPs
3. C6000™ Performance Value DSPs. Soluciones de rendimiento medio con aritmética de punto fijo. Optimizados para trabajar con audio digital.
    - TMS320C642x DSPs
    - TMS320C6410/12/13/18 DSPs
    - TMS320C62x DSPs
4. C6000™ Floating-point DSPs. Familia de DSP con aritmética de punto flotante. Sus características los hacen perfectos para procesamiento de audio que requiera alta fidelidad y rendimiento en tiempo real.
    - TMS320C672x DSPs
    - TMS320C67x DSPs
5. C5000 DSPs. Se trata de la familia de DSPs de bajo consumo, pensados para las aplicaciones móviles o en las que la dispación de energís sea un factor importante.

El DSP utilizado en la práctica pertenece a los DSPs de punto flotante, se trata concretamente del C6713, montado en el módulo de evaluación de Spectrum Digital llamado C6713 DSK (DSP Starter Kit).

Referencia técnica del C6713 DSK

El C6713DSK es una plataforma autónoma de bajo coste para el desarrollo de aplicaciones en DSP de la familia C67xx. La plataforma cuenta con un completo set de dispositivos que abarcan una amplia variedad de entornos de aplicación. Las características clave son:

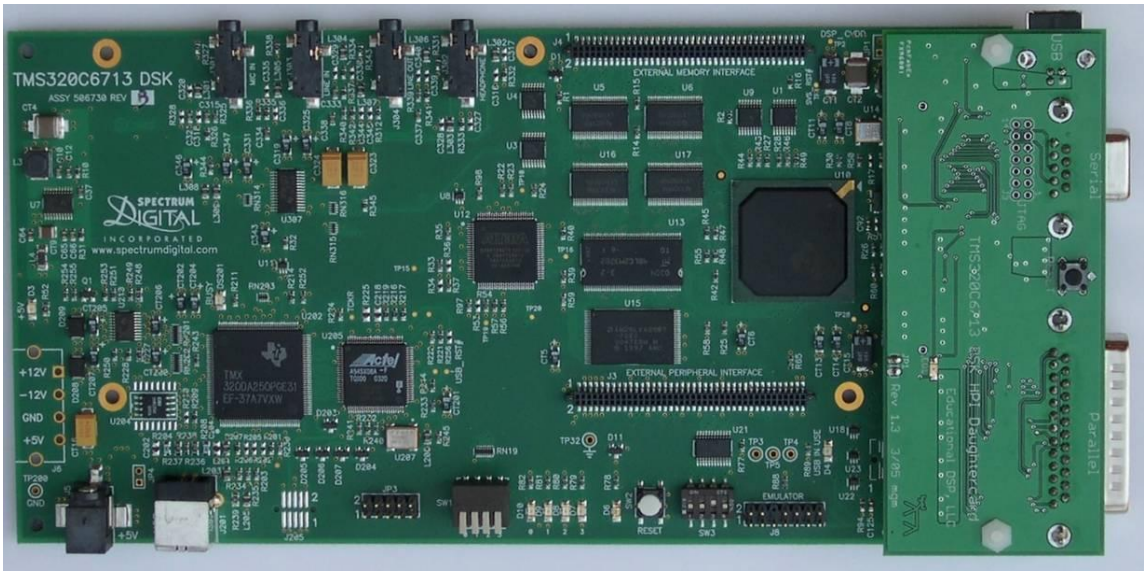


Figura 1: Sistema C6713DSK

- El DSP T320C6713 de Texas Instruments con un reloj de 225MHz
- Un códec de audio estéreo AIC23.
- 16MBytes de SDRAM
- 512 Kbytes de memoria Flash (de los cuales solo están disponibles 256 por defecto).
- 4 diodos LED e interruptores configurables por el usuario.
- Configuración Software de la placa en un chip CPLD.
- Opciones de arranque configurables.
- Conectores estándar para placas de periféricos
- Emulación de cable JTAG mediante USB.
- Alimentación de 5 voltios para toda la placa.

El DSP se comunica con los periféricos de la placa a través de un EMIF (External Memory Interface) de 32 bits. La SDRAM, la memoria Flash y el CPLD están conectados a dicho bus. Además, los conectores para las placas de expansión de terceros también reciben las señales del EMIF.

Las señales de audio analógico se reciben a través de un códec estéreo AIC23 y cuatro conectores *jack* de 3.5 mm.

El CPLD contiene la lógica que comunica los componentes de la placa entre sí. Dispone de una interfaz de usuario basada en registros que permite configurar la placa leyendo y escribiendo dichos registros.

La comunicación con el entorno de programación (Code Composer Studio) se realiza mediante un emulador de JTAG integrado con un interfaz de host USB. También se puede utilizar un emulador externo mediante el conector JTAG de la placa.

Diagrama de bloques conceptual

La familia de DSPs C67xx dispone de un gran espacio de memoria direccionable a nivel de byte. Todas las direcciones tienen 32 bits de anchura y tanto el código como los datos pueden situarse en cualquier sitio del espacio de direcciones unificado.

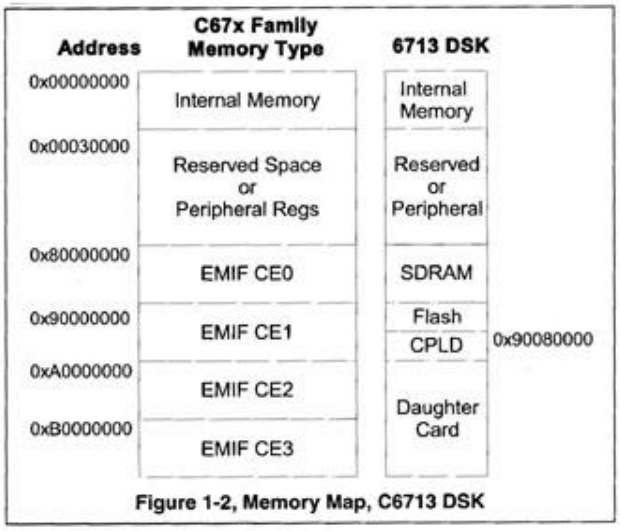


Figure 1-2, Memory Map, C6713 DSK

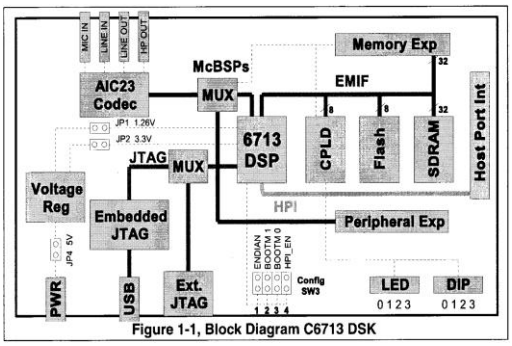


Figure 1-1, Block Diagram C6713 DSK

Figura 2: Mapa de memoria y diagrama de bloques del sistema C6713DSK

Ruta de datos del C6713

La CPU del TMS320C6713 está basada en el diseño de CPU C67x de Texas Instruments. La CPU procesa instrucciones de tipo VLIW (Very Long Instruction Word) de 256 bits para enviar hasta 8 instrucciones de 32 bits a las 8 unidades funcionales en cada ciclo de reloj. La arquitectura VLIW presenta controles mediante los cuales las unidades funcionales no tienen por qué recibir instrucciones si no están listas para ejecutarlas.

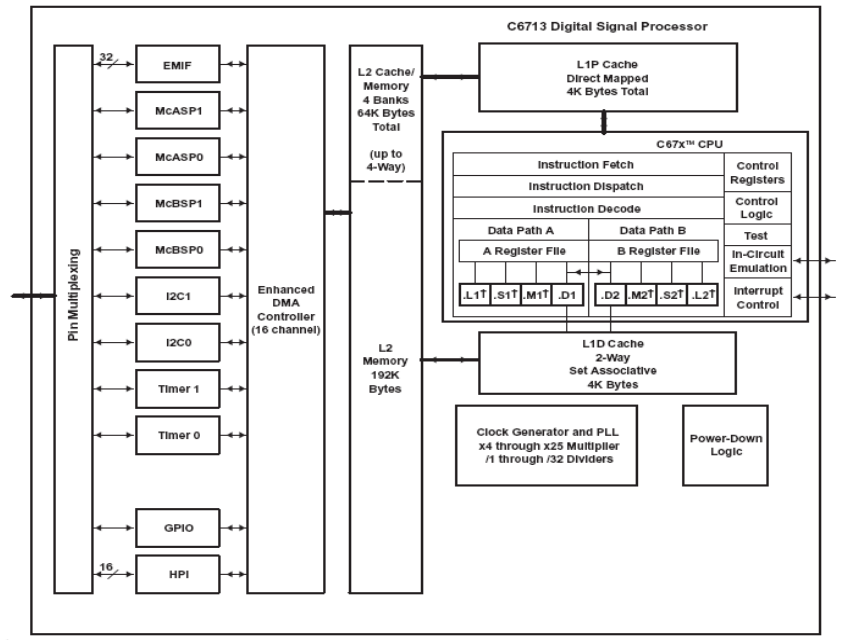
Las unidades funcionales están divididas en dos conjuntos, cada uno de los cuales dispone de cuatro unidades y un banco de registros. El primer conjunto contiene las unidades .L1, .S1, .M1 y .D1; el otro conjunto contiene las unidades .D2, .M2, .S2 y .L2. Cada banco de registros posee 16 registros de 32 bits, lo que supone un total de 32 registros de propósito general. Estos conjuntos componen la parte A y la parte B de la CPU. Los registros de la parte B se comunican con la parte A mediante un único bus y viceversa, lo que significa que una parte de la CPU puede acceder a un registro de la otra parte una vez en cada ciclo de reloj.

La arquitectura es de carga almacenamiento, y las unidades encargadas del tráfico de datos entre memoria y registros son .D1 y .D2. Salvo estas dos, el resto de unidades pueden trabajar con datos en punto fijo y datos en punto flotante. La CPU soporta múltiples modos de direccionamiento directo e indirecto, y todas las instrucciones son condicionales. Las unidades funcionales .M están dedicadas a la multiplicación, mientras que .S y .L se utilizan para otros cálculos aritméticos, lógicos y de saltos, generando resultados disponibles en cada ciclo de reloj.

Presenta:

1. Una caché de nivel 2 de 64 Kbytes, que puede funcionar como una caché de hasta cuatro vías.
2. Una caché de nivel 1 con asociación directa para instrucciones.
3. Una caché de nivel 2 de datos con 192 Kbytes de capacidad junto con una de nivel 1 de 4Kbytes y 2 vías.

functional block and CPU (DSP core) diagram



† In addition to fixed-point instructions, these functional units execute floating-point instructions.

Figura 3: Diagrama de interfaces de E/S con la CPU

Sistema DSKEye

Acerca de BiTec Ltd.

BITEC fue fundada en el año 2002. Ofrece una amplia oferta de productos, tanto hardware como software, dedicados al desarrollo de sistemas de procesamiento de alto rendimiento. Las áreas de mercado que se benefician de estos productos son las de investigación, tanto a nivel industrial como a nivel académico. Además, ofrece una gran gama de productos que permiten una integración rápida y efectiva de tecnologías basadas en FPGA y DPS en plataformas de procesamiento de alto rendimiento.

Referencia técnica

La DSKeye es una tarjeta de expansión para los kits del desarrollo de Texas Instruments C6x y C5x. Dirigiendo la DSKeye está una FPGA Cyclone II. Este dispositivo conecta la cámara fotográfica al bus externo del DSP permitiendo una transferencia de imágenes eficiente hacia la memoria principal, con poca sobrecarga para el DSP. También es utilizado para implementar interfaces definidas por el usuario a través de los conectores Santa Cruz de ALTERA. Un número cada vez más grande de placas Santa Cruz están disponibles en el mercado para trabajar sobre VGA, USB…etc.

Figura 4: Sistema DSKEye

Las características más importantes de DSKEye son:

- Sensor CMOS de 5.2Megapixel
- Última versión de la FPGA ALTERA Cyclone II
- Puerto embebido para interfaz TCP/IP
- Interfaz Santa Cruz de expansión

Examinando más profundamente cada una de las características anteriores:

Cámara:

• Sensor CMOS de 5.2Megapixel

• Tamaño de la imagen:

QSXGA: 2592x1944

SXGA: 1280x960

VGA: 640x480; 1280x480

HF: 320x200; 1280x200

- Exposición electrónica:

  QSXGA: Hasta 1998:1

  SXGA: Hasta 978:1

  VGA: Hasta 488:1

  HF: Hasta 208:1

- Formato de salida: flujo digital de 10 bits en RGB.
- Tamaño de la lente: 1/1.8".
- Ángulo del rayo principal de la lente de 15 grados.
- Tasa máxima de refresco de imágenes:

  QSXGA: 8 fps

  SXGA: 30 fps

  VGA: 60 fps

  HF: 140 fps

- Sensibilidad: 1.2 V/Lux/seg.
- Radio de S/N: 42dB.
- Rango dinámico: 60 dB.
- Tamaño del pixel: 2.775 um x 2.775 um.
- Corriente de oscuridad: 10mV/sec @ 60 deg.
- Ruido Electrónico Con Patrón Fijo: 0.05% de Vpeak-to-peak.
- Área de la imagen: 14.22mm x 14.22mm.
- Salida ADC de 10 bits.
- Salida Bayer pattern.

Lente:

- Longitud focal: 9.4mm.
- F No: 3.0.

- Distorsión: <3.0%.
- Enfoque: “M hasta infinito.
- FOV: 49 grados diagonal.
- Filtro de IR: 650nm.
- Se aceptarán lentes con características compatibles.
- Disponible disparador mecánico.

Interfaz TCP/IP:

- Puerto TCP/IP embebido con versión de Wireless opcional.
- Protocolos soportados: TCP, UDP, IP, ICMP, ARP/RARP y MAC.
- Protocolos DHCP, SMTP…etc. pueden ser soportados vía software.
- WiFi disponible.

Interfaz Santa Cruz:

- Señales I/O para 40 usuarios, configurables y en buffer conectadas a la FPGA. Todos sensibles a 5V.
- 3 pins PLL con reloj conectados.

Compatibilidad:

- Windows, CCS 2.1 y superior.
- Quartus II Web Edition.

Requisitos del sistema:

- CCS 2.1 y superior.
- Quartus II Web Edition 5.1 o superior.

Las características principales de Wireless LAN son las siguientes:

- Chip set: GSV / Intersil Prism III.
- Velocidad de transferencia: 11, 5.5, 2, 1 Mbps (CCK, DQPSK, DBPSK).
- Asignación de canal para España: 2 canales centrados entre 2457Mhz y 2462Mhz.

• Cobertura:

- Espacios abiertos: 150m a 11Mbps, 200m a 2Mbps.

- Espacios cerrados: 30m a 11Mbps, 50m a 2Mbps.

Diagrama de bloques conceptual

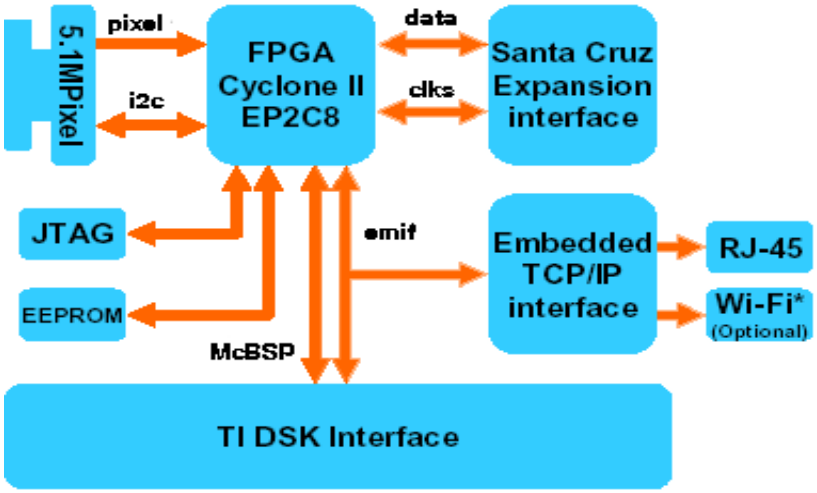


Figura 5: Diagrama estructural

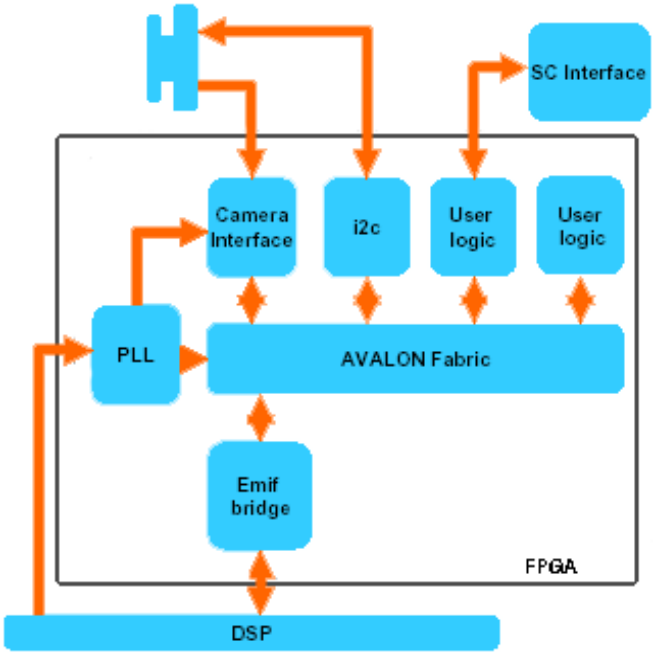


Figura 6: Diagrama de E/S por bloques

Circuito emisor de luz

Para determinar la posición de los barcos, hemos utilizado un circuito formado por unas baterías y un par de diodos LED. Con esos dos diodos seremos capaces de identificar tanto la posición como la dirección del barco en cada momento.

Estructura y componentes

El primer elemento de este circuito es el diodo LED. Un diodo LED (Light Emitting Diode) es un dispositivo semiconductor que emite luz monocromática cuando se polariza en directa y es atravesado por la corriente eléctrica.

El color depende del material semiconductor que se haya empleado en la construcción del diodo y puede variar desde el ultravioleta, pasando por el espectro visible, hasta el infrarrojo. En este caso disponemos de un diodo LED de color rojo y otro diodo LED de color verde.

El otro componente que forma parte de este circuito es la alimentación del mismo. Para las pruebas iniciales utilizamos cuatro pilas de 1.5V (6V) en paralelo alimentando al circuito. Una vez que el circuito está integrado a los barcos, la alimentación del circuito viene dada por la propia energía que suministra el barco. Esta energía es suficiente para alimentar correctamente el circuito, cosa que no sucede cuando las pilas empiezan a descargarse. Cuando esto ocurre se nota que la intensidad de los LEDs baja considerablemente.

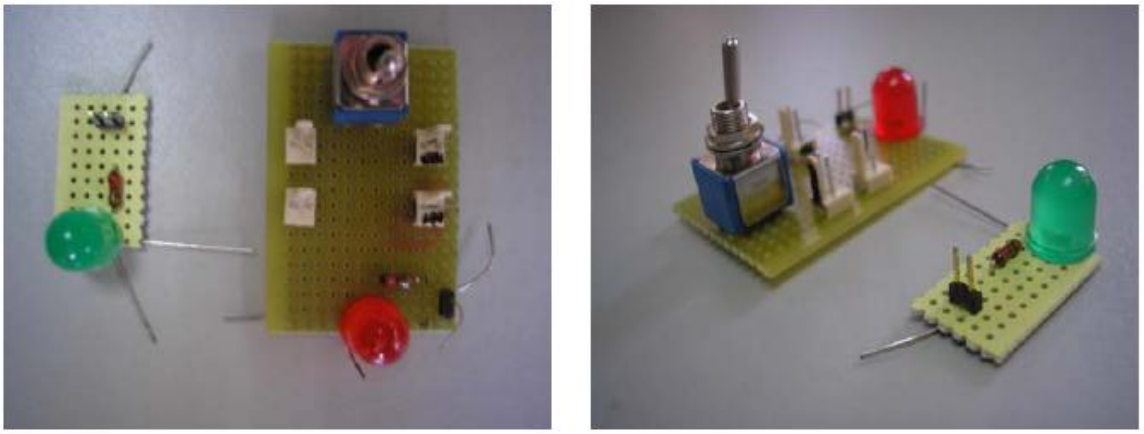


Figura 7: Circuito de LEDs 1

Finalidad

El uso que vamos a dar a este circuito dentro del proyecto es sencillo. La idea es colocar los LEDs tanto en la proa como en la popa del barco que queremos localizar. Una vez que el barco se encuentra en marcha, los LEDs lucirán, ya que están conectados a la batería interna del mismo. En esta situación, cada vez que tomemos una imagen del barco, deberían distinguirse claramente los dos LEDs en la imagen. A continuación pasaríamos a realizar el análisis mediante el DSP de la imagen tomada. En esta imagen debemos ser capaces de reconocer los dos LEDs con algún algoritmo eficiente basado tanto en la intensidad del LED (ya que emite luz y debería sobresalir del resto de la imagen que sólo reflejaría luz) como en su propio color. Cuando hayamos reconocido la posición de los LEDs, estaremos en situación de saber tanto la posición como el rumbo del propio barco, que es el objetivo final.

Receptor inalámbrico multibanda DWL-G122

Acerca de D-Link

D-Link es una empresa cuya actividad económica se centra en el desarrollo tecnológico de dispositivos hardware orientados al sector de las telecomunicaciones y la informática. Entre sus productos más destacados se encuentran:

- Tarjetas de red
- Access Points
- Routers
- Pasarelas
- Firewalls
- Switches

Tiene su sede en Taiwán y como dato anecdótico se podría mencionar el hecho de haber sido la primera compañía de redes aceptada en la lista del mercado de valores de Taiwán, concretamente en la primavera del año 1994.

Desde un tiempo a esta parte se puede considerar que éste último hecho facilitó enormemente su crecimiento y la construcción en España de una de sus sedes, en torno al año 2000.

Actualmente, es propietaria y tiene los derechos de explotación del protocolo ISO de OSI 802.11 super-G, estándar que aumenta la velocidad de transmisión teórica de un canal inalámbrico tipo G de 54 a 108 Mbps.

En concreto, para nuestro proyecto hemos utilizado como principal interfaz de comunicación con el DSP su receptor de banda inalámbrica DWL-G122.

Referencia técnica del DWL-G122

La tarjeta DWL-G122 es un adaptador inalámbrico de muy bajo peso. Su interfaz de conexión USB 2.0 es ideal para poder conectarlo a cualquier equipo, ya sea portátil o sobremesa, factor importante que nos ha ayudado a la hora de conectar el DSP y nuestros portátiles con el fin de depurar y solucionar cada uno de los problemas que nos hemos ido encontrando a medida que el proyecto iba creciendo.

Aunque es compatible con el estándar 802.11g, no hemos podido beneficiarnos del mayor ancho de banda de este estándar (54 Mbps) ya que el emisor inalámbrico instalado en el DSP únicamente puede trabajar bajo 802.11b y por tanto alcanzando como máximo 11 Mbps de tasa de transferencia teórica.

Figura 8: Receptor inalámbrico USB

Como tampoco hemos decidido encriptar el tráfico de datos entre el DSP y el equipo receptor, no hemos necesitado hacer uso de sus controladores de encriptación/desencriptación WEP y WPA. Sus características técnicas más a fondo son las siguientes:

1. Alto rendimiento, hasta 5 veces por encima del 802.11b (característica no aprovechada)
2. Soporta WEP a 64/128-bits
3. Soporta WPA/PSK, 802.1x
4. Ancho de Banda de 54Mbps, en 2.4GHz
5. Soporte USB 1.1 y USB 2.0

Configuración genérica de red y enrutamiento de paquetes

La arquitectura de red que hemos utilizado puede resumirse en el siguiente esquema:

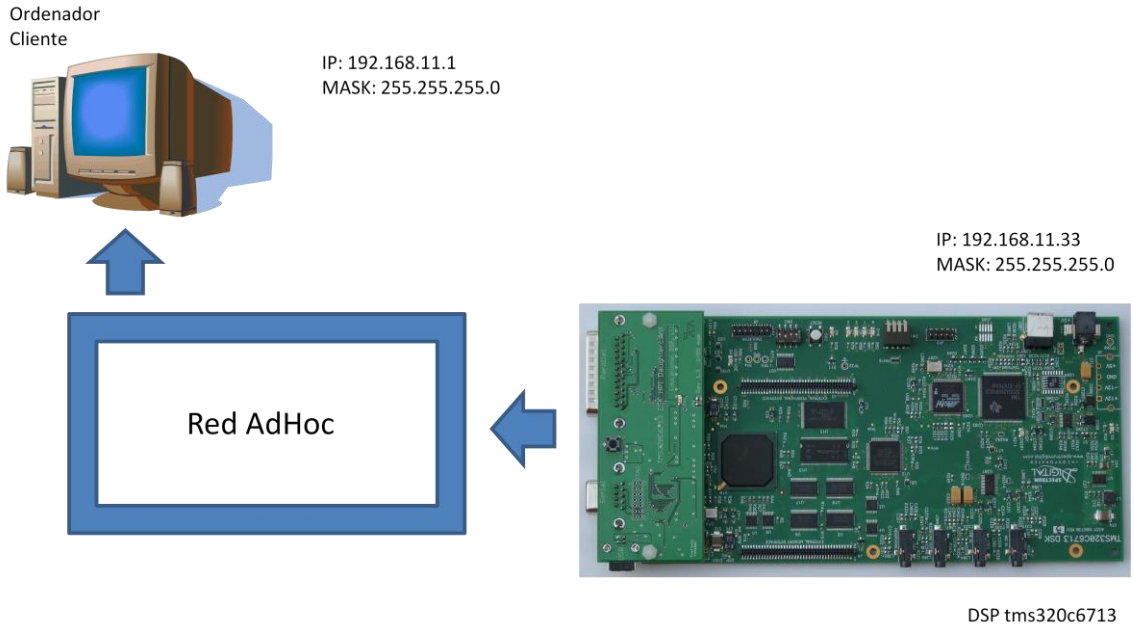
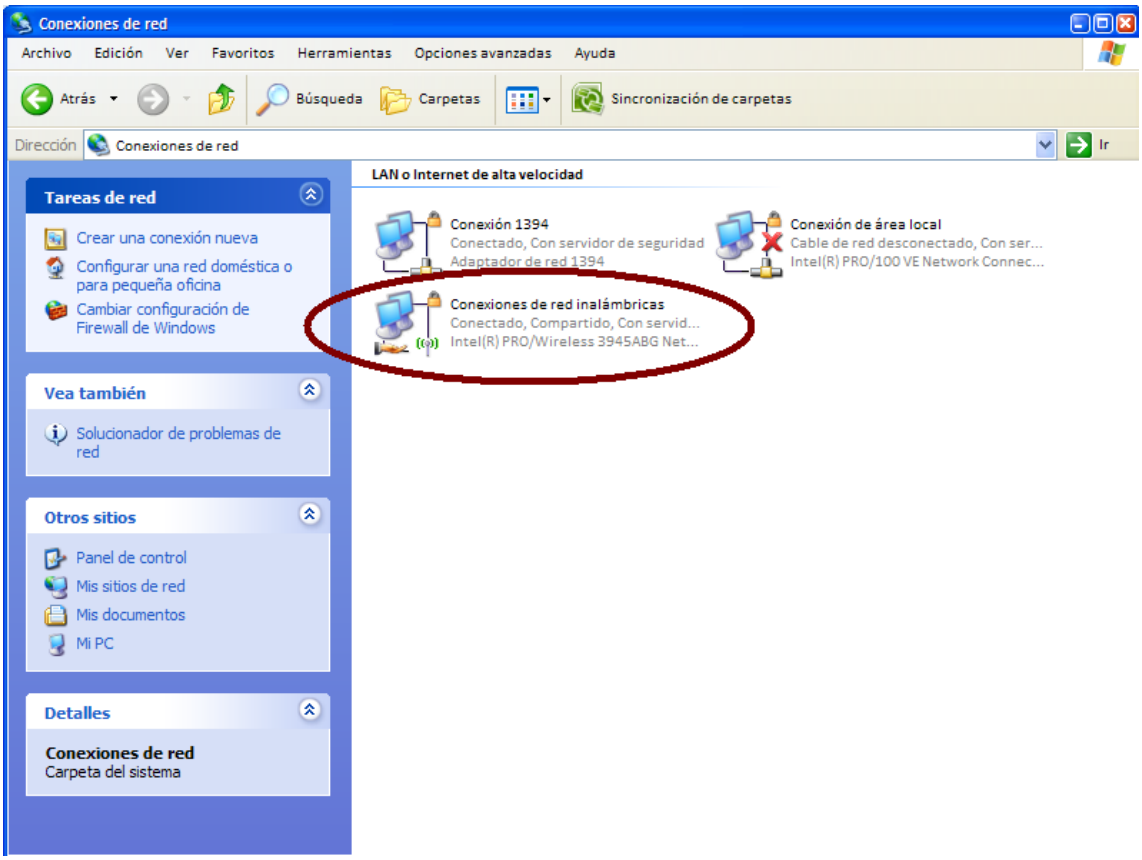


Figura 9: Diag. conceptual de conexión

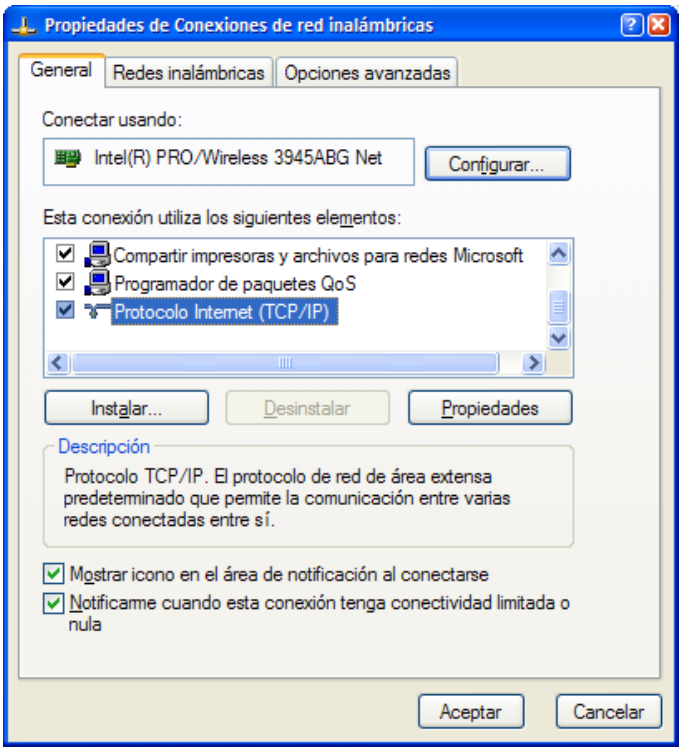
Como puede verse, en vez de crear una red propia para todo el conjunto se crea un vínculo equipo a equipo, para facilitar así la configuración de red.

No obstante, la posibilidad de poder crear una red inalámbrica local con un número limitado de hosts es factible si se consigue dotar al DSP de un emisor de datos más potente que de mayor ancho de banda, para poder, por ejemplo utilizarlo como servidor multidifusión de cualquier tipo de datos.

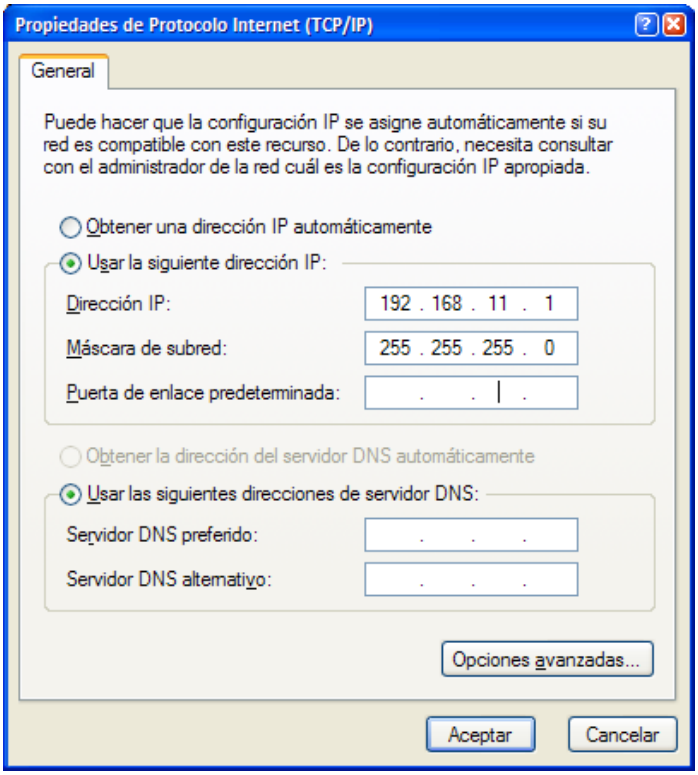
Para poder configurar la ruta de datos, primero será necesario instalar y comprobar el correcto funcionamiento del emisor/receptor inalámbrico de D-Link, tras lo cual aparecerá una nueva interfaz de red en la carpeta de “Conexiones de red” de nuestro sistema:



Una vez comprobado esto, deberán modificarse las propiedades de red de dicha interfaz. Para ello hacemos doble click el icono de la interfaz y seleccionamos en el menú pop-up que aparece la opción “Propiedades”. Tras esto aparecerá la siguiente ventana:



Seleccionamos “Protocolo Internet TCP/IP y aparecerá la siguiente ventana:



Rellenamos los campos con la configuración de red del cliente y pulsamos el botón “Aceptar”. Ahora ya tenemos nuestro cliente configurado para poder recibir datos del DSP.

Pero todavía no está todo listo, ya que falta configurar el DSP para que pueda comunicarse con nuestro cliente. La configuración IP del DSP se asigna de forma estática en tiempo de ejecución, llamando a un conjunto de funciones de la librería de Sockets que viene precompilada con el entorno de desarrollo CCStudio. Cada una de estas rutinas asigna un parámetro concreto de la configuración de red de la máquina, que no es más que una escritura en unas determinadas direcciones de memoria que el DSP utiliza para fijar los parámetros de red.

Dichas función son las siguientes:

setMACAddr(unsigned char*);

Utilizada para asignar una determinada dirección física de dispositivo, más conocida como dirección MAC. El formato de entrada debe ser un array de unsigned char con 6 posiciones, una para cada dígito hexadecimal.

setIP(unsigned char*);

Utilizada para asignar una determinada dirección IP de dispositivo. El formato de entrada debe ser un array de unsigned char con 4 posiciones, una para cada dígito entero de 0 a 255.

setgateway(unsigned char*);

Utilizada para asignar la puerta de enlace del dispositivo. El formato de entrada debe ser un array de unsigned char con 4 posiciones, una para cada dígito entero de 0 a 255.

setsubmask(unsigned char*);

Utilizada para asignar una determinada máscara de red. El formato de entrada debe ser un array de unsigned char con 4 posiciones, una para cada dígito entero de 0 a 255.

sysint(unsigned char, unsigned char);

Esta función informa al sistema de que protocolo se va a utilizar: TCP, UDP, etc…

Para mayor simplicidad, se creó una función llamada que contiene cada una de las llamadas necesarias para establecer dicha configuración:

```
void InitNetConfig(void)
{
  unsigned char  ip[6];

  ip[0] = 0x00; ip[1] = 0x08; ip[2] = 0xDC; ip[3] = 0x00; ip[4] = 0x00; ip[5] = 0x00;
  setMACAddr(ip); // MAC = 00 08 DC 00 00 00

  ip[0] = 192; ip[1] = 168; ip[2] = 11; ip[3] = 33;
  setIP(ip); // IP = 192.168.11.33

  ip[3] = 1;
  setgateway(ip); // GW = 192.168.11.1

  ip[0] = 255; ip[1] = 255; ip[2] = 255; ip[3] = 0;
  setsubmask(ip); // MASK = 255.255.255.0 (tipo C)

  // Inicializamos el sistema TCP
  sysinit(0x55,0x55);

}
```

Esta función deberá ejecutarse en el método principal del proyecto que posteriormente será volcado a la memoria del DSP. Si se omite, el DSP no tendrá asignada una configuración de red y por tanto no podrá intercambiar datos por la interfaz inalámbrica.

Ya por último, la recepción de datos deberá ser gestionada por un programa externo, por ejemplo el decodificador h263 o el programa “DSK Watcher” diseñados en este proyecto. Hay que darse cuenta que el DSP ejecutará el código volcado a través del entorno de programación CCStudio y por tanto los datos deberán de ser recogidos en el exterior y posteriormente ser interpretados de forma correcta conforme a un protocolo determinado prefijado de antemano por el diseñador.

Software empleado

CodeComposer Studio 3.1.0

CodeCompser Sudio (CCStudio a partir de ahora) es el entorno integrado de programación que ofrece Texas Instruments para trabajar con sus procesadores digitales de señal de las familias C5000 y C6000.

Instalación y configuración del entorno

El DSK6713 viene acompañado de un CD-ROM que contiene el IDE. Al ejecutar este CD-ROM se nos permite instalar tanto el IDE como los controladores para el chip o la placa que deseemos, así como emuladores y simuladores de dichos sistemas. Debemos instalar en primer lugar el entorno propiamente dicho, después la herramienta FlashBurn, utilidad que permite escribir en la memoria no volátil de la placa de desarrollo y por último los controladores y el emulador de nuestro DSP, en este caso el C6713. Los tres procesos se llevan a cabo de forma autómatica y sólo tenemos que ejecutarlos en el orden que se nos indica. Tras la instalación aparecen cuatro nuevos iconos en el escritorio:

Figura 10: Iconos de escritorio

Para completar la configuración hacemos clic en “Setup CCStudio v3.1” y añadimos el DSP con el que vamos a trabajar a la lista de dispositivos válidos:

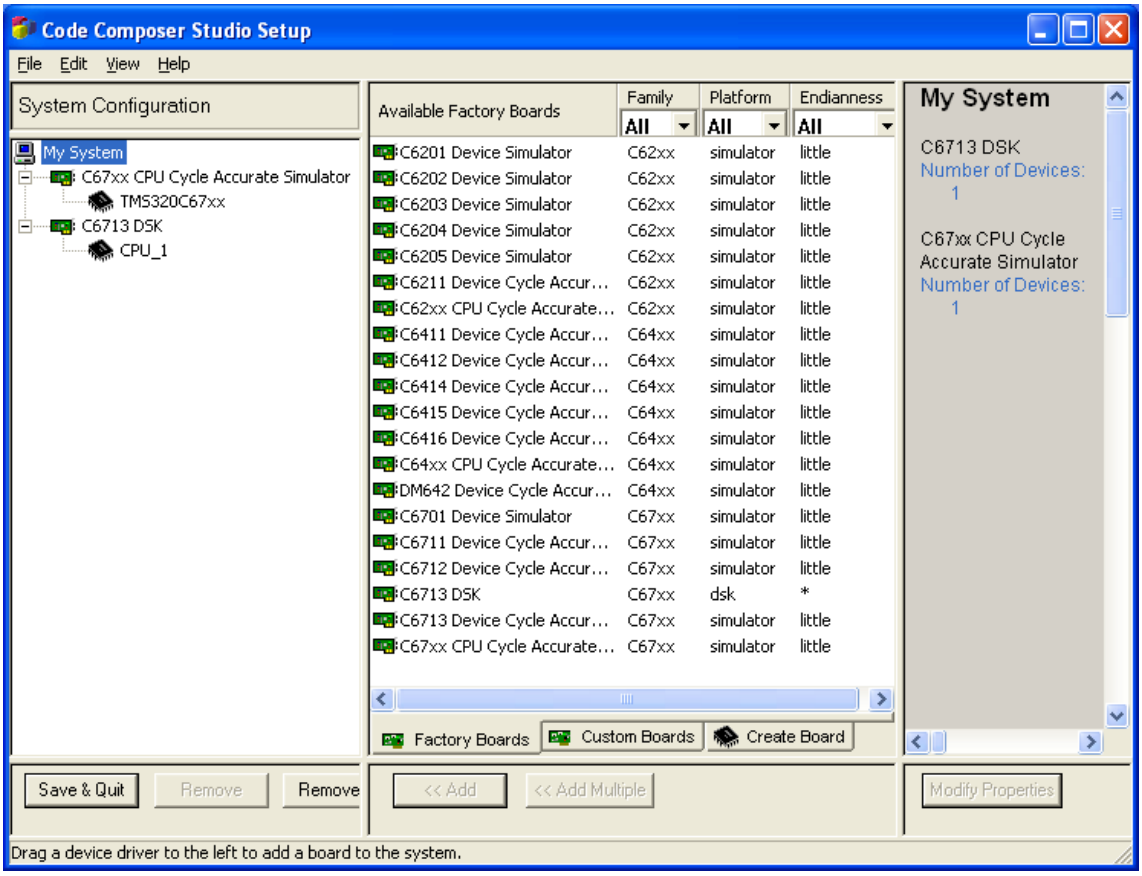


Figura 11: Inclusión del procesador

Tras esto, ya es posible ejecutar el programa para trabajar con el DSP o bien un simulador del mismo (Simulando sólo la CPU o la CPU y la memoria externa). La herramienta de diagnóstico incluida sirve para comprobar la conexión a la placa mediante USB y el funcionamiento de sus distintos componentes.

Conceptos y funciones principales

El programa nos ofrece un editor para C/C++ y código ensamblador, y las herramientas para generar código máquina a partir de dichos lenguajes, un compilador, un ensamblador y un enlazador, todas ellas integradas en el entorno. Además, el compilador permite realizar optimizaciones automáticas sobre el código generado para obtener más rendimiento o menor gasto de memoria. También se dispone de un depurador integrado en el entorno que permite ejecución paso a paso tanto de líneas de código C/C++ como de líneas de ensamblador, así como visualizar el código fuente y el código máquina que se asocia a cada línea del programa de forma mixta. Por supuesto, es posible observar y cambiar los valores de las variables en tiempo de depuración, y existen herramientas de temporización y monitorización también incluidas en el entorno. La configuración de la placa en cuanto a regiones de memoria y administración de buses, etc. también cuenta con un asistente visual que facilita enormemente su ajuste a los valores deseados.

Para iniciar un proyecto en CCStudio simplemente seleccionamos en el menú “Project/New…” , y se nos presenta la siguiente pantalla:

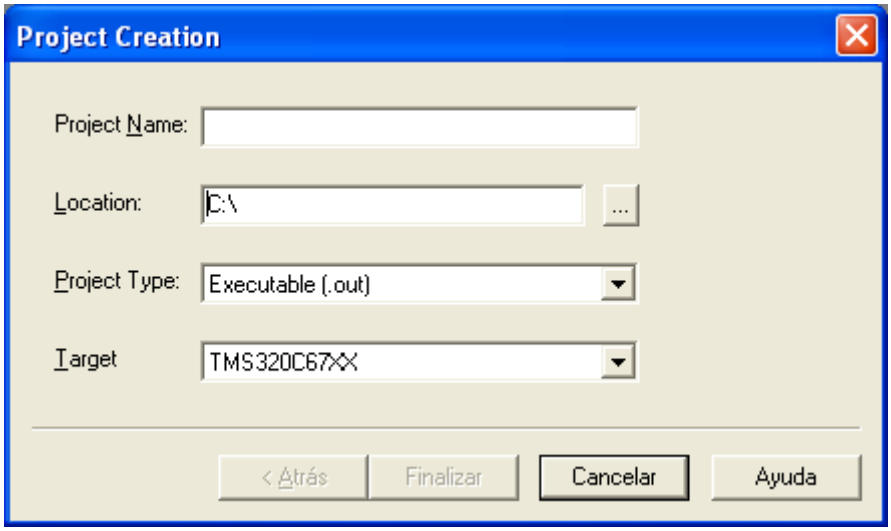


Figura 12: Creación de un nuevo proyecto

Elegimos el nombre del proyecto, la carpeta en la que lo vamos a guardar, si es una librería (.lib) o un ejecutable (.out), y el dispositivo que vamos a usar. Una vez hecho esto, el proyecto se encontrará activo en el panel de la izquierda del CCStudio:

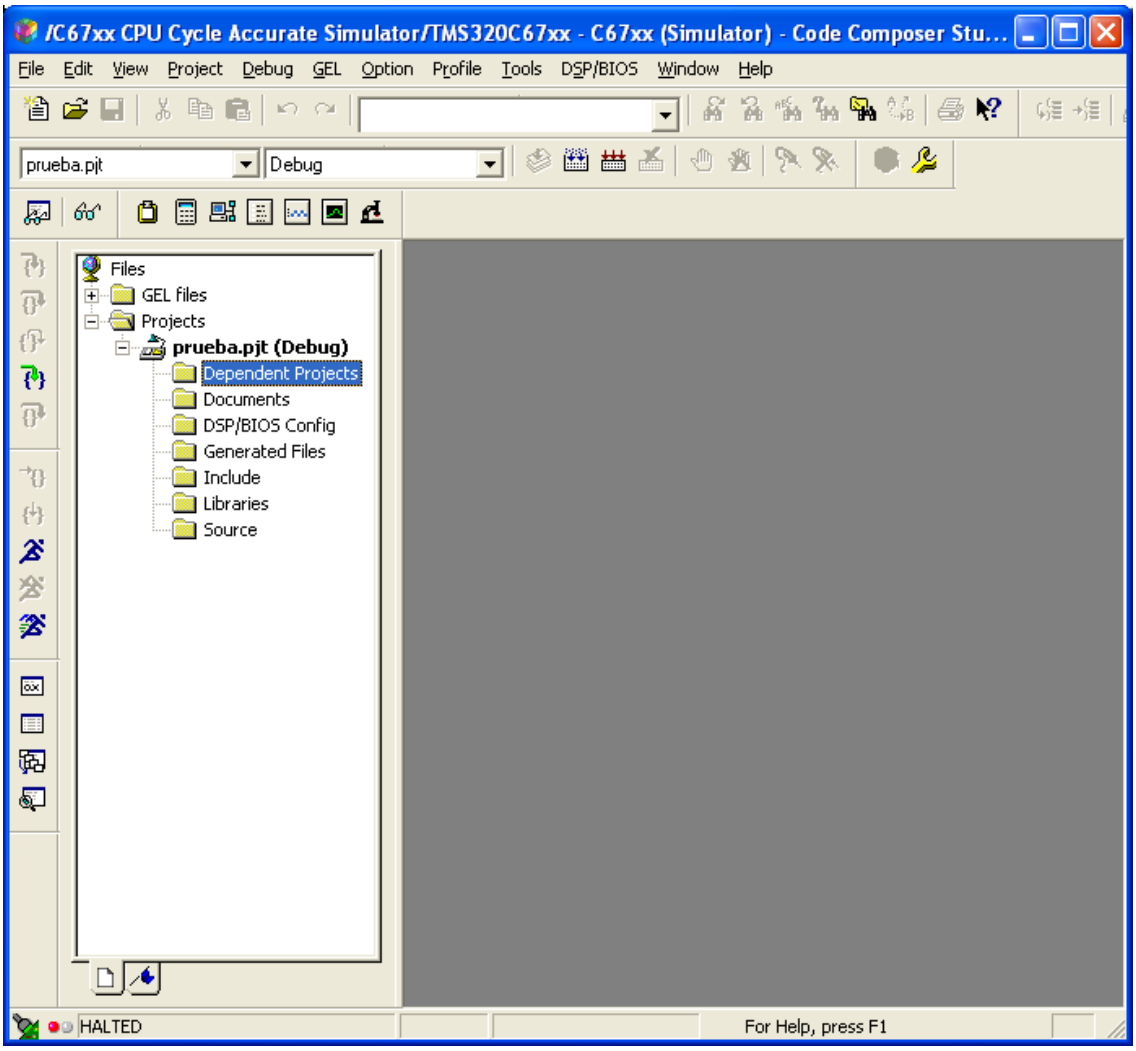


Figura 13: Localización del proyecto

A continuación podemos añadir o crear ficheros de código fuente (con extensión .c o .cpp para C/C++ o .asm para código ensamblador), ficheros de cabecera (.h), librerías usadas por el proyecto u otros proyectos requeridos para el actual. Si queremos generar un ejecutable, un paso importante será añadir una configuración. Para ello hacemos clic en "File->New->DSP/BIOS Configuration…" y seleccionamos la configuración base de nuestro dispositivo.

Se nos presenta una pantalla con todas las características configurables del DSP en la que podemos añadir y modificar según nuestras necesidades. Una vez hecho esto, guardamos la configuración en un archivo .cdb y lo agregamos al proyecto:

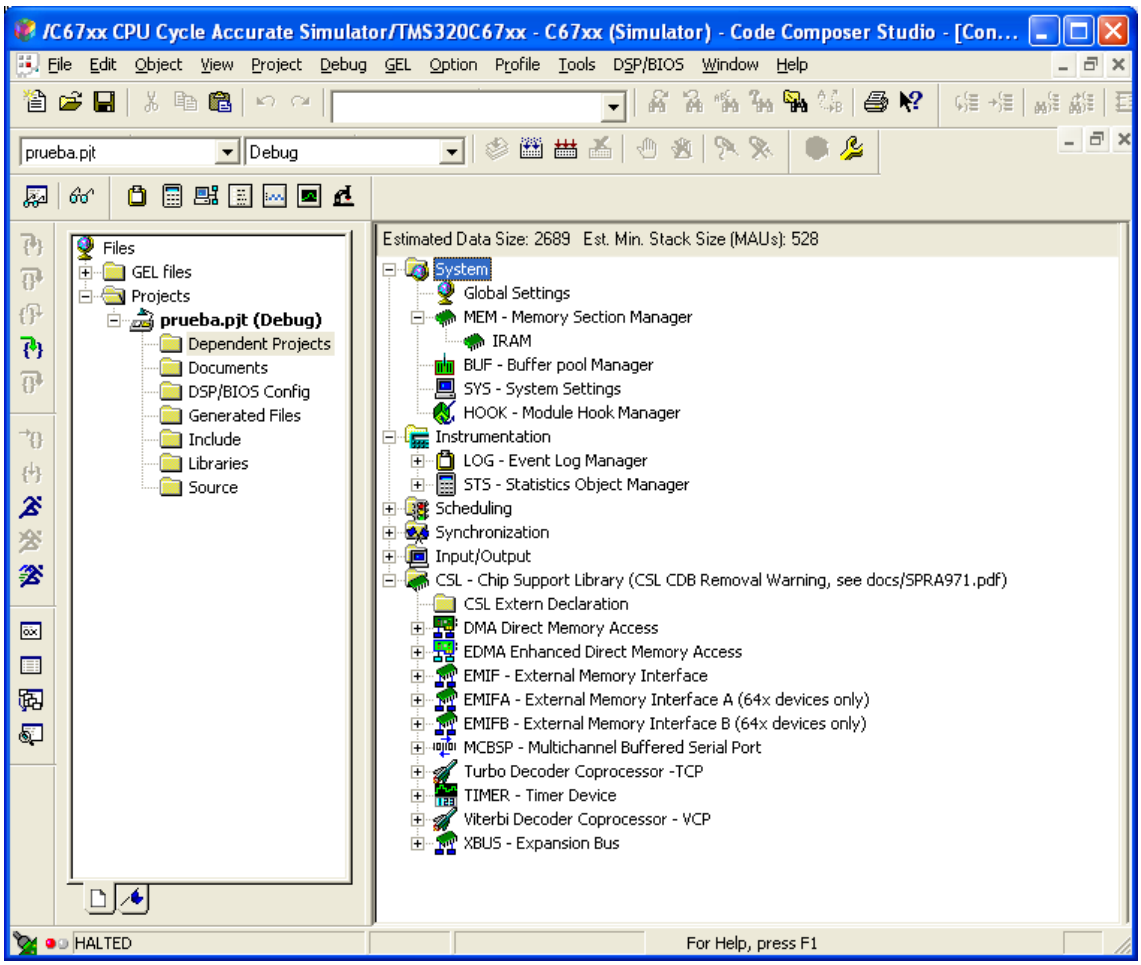


Figura 14: Propiedades del proyecto

Al agregar el fichero de configuración, en la carpeta “Generated Files” del proyecto se crean de forma automática los ficheros de configuración e inicialización de dispositivo necesarios para el proyecto.

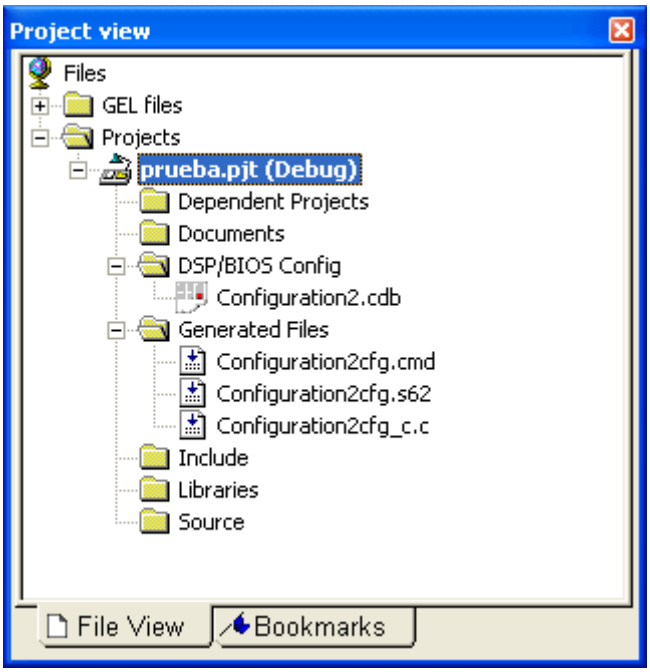


Figura 15: Ficheros base del proyecto

Una vez hecho esto, el proyecto ya está preparado para añadir los archivos de código fuente y las librerías necesarias y compilarlo sin problemas. Una vez compilado, el ejecutable resultante, un archivo con extensión .out, se puede cargar en el DSP mediante el menú "File->Load Program…" y ejecutarlo, o depurarlo, pudiendo añadir "Breakpoints" o "ProbePoints" para la entrada/salida de datos desde por ejemplo archivos que se encuentren almacenados en el PC.

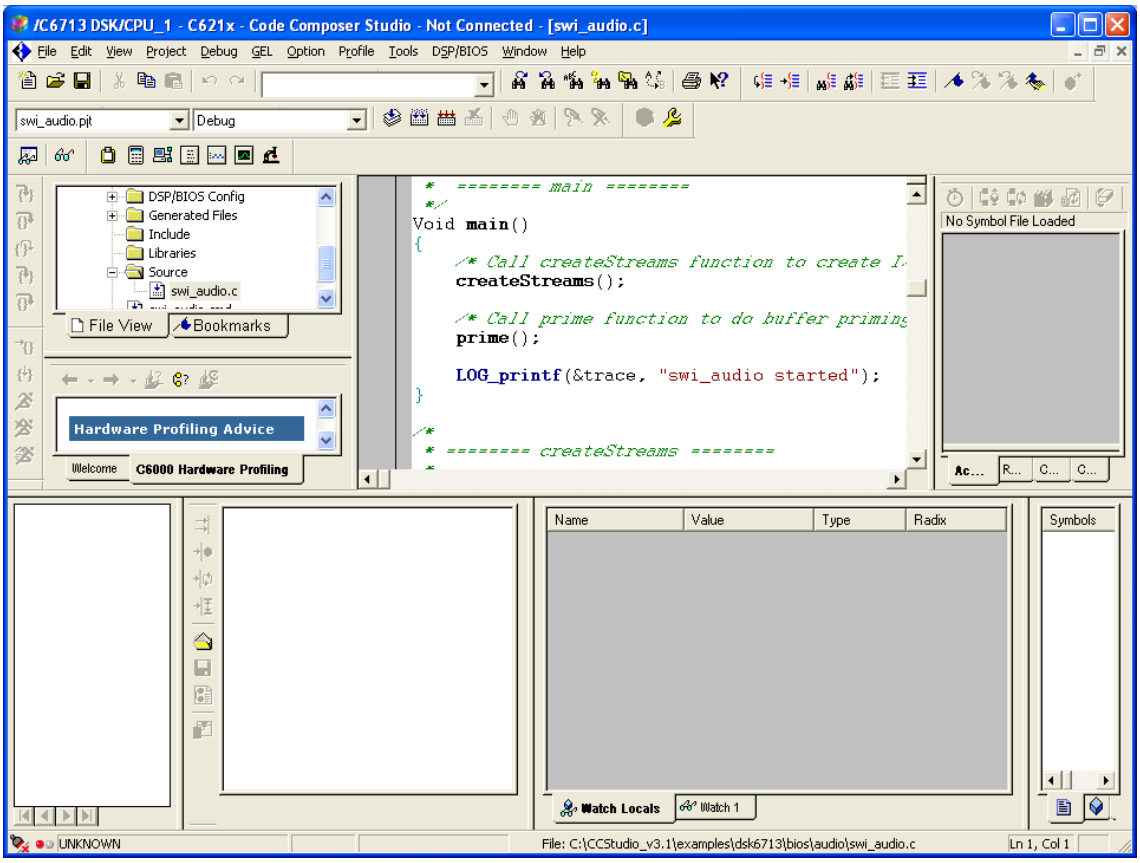


Figura 16: Entorno y herramientas

El entorno pone a disposición del usuario un completo tutorial que resulta muy útil para familiarizarse con todas sus opciones y posibilidades, así como diversos archivos de ayuda que reducen el tiempo de aprendizaje necesario para empezar a trabajar con él de forma eficiente.

Otras Referencias

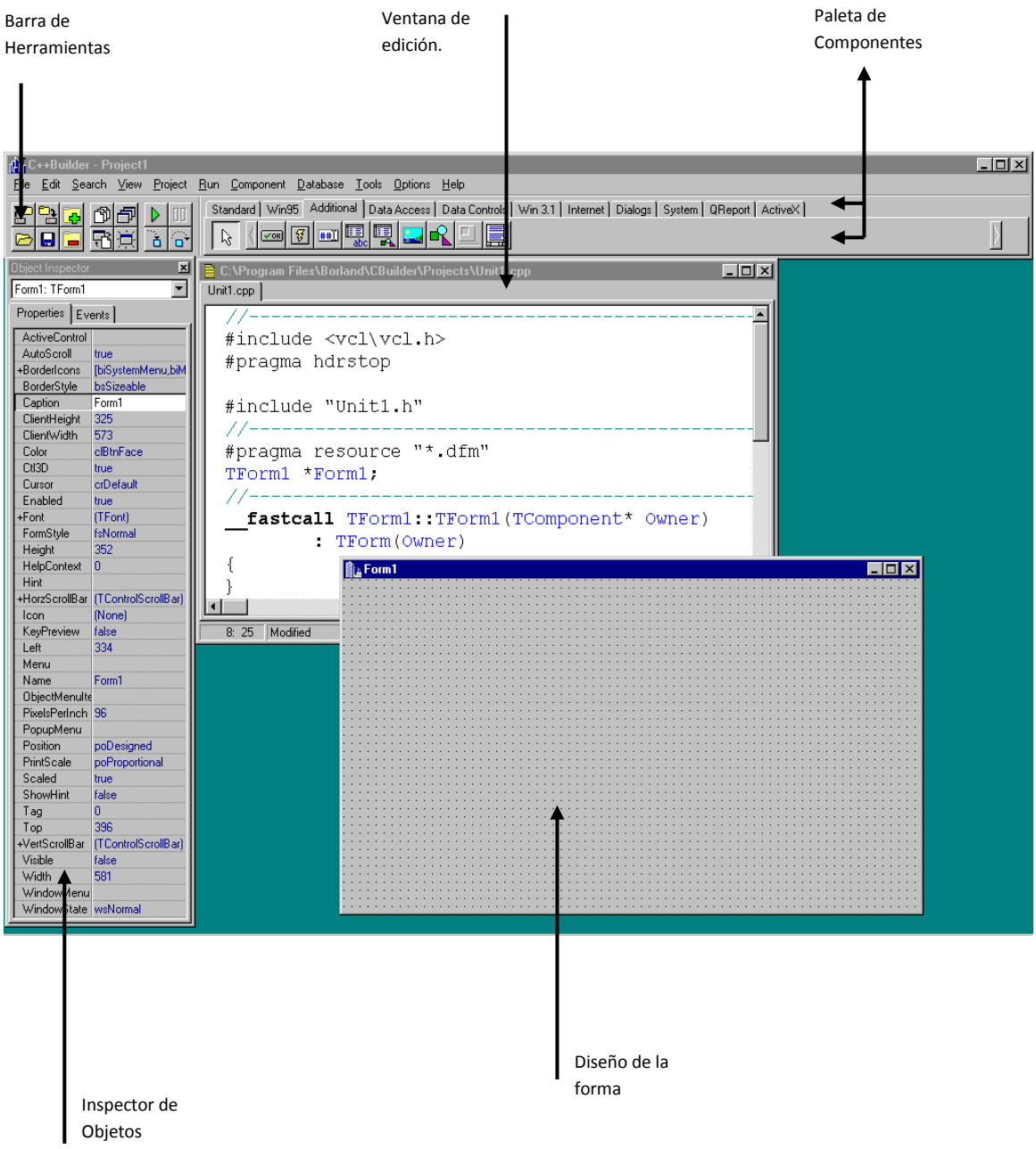
Como fuente de información alternativa, aparte de la extensa ayuda incluida en programa, podemos consultar en la web de Texas Instruments, donde encontraremos gran cantidad de documentación sobre el mismo:

http://focus.ti.com/docs/toolsw/folders/print/ccstudio.html

Borland C++ Builder 6

Instalación y configuración del entorno

Borland ofrece tres versiones diferentes de C++ Builder y cada versión será de acuerdo a las necesidades de programación que tengamos. Nosotros hemos utilizado la versión C++ BUILDER ESTÁNDAR. Esta versión proporciona todas las herramientas necesarias para el desarrollo de aplicaciones que necesiten incluso interactuar con bases de datos. Esta versión ocupa aproximadamente 75 MB de espacio en el disco duro.



Para instalar C++ Builder en el PC, simplemente se coloca el CD ROM Borland C++ Builder en el lector de discos compactos. No se necesitas teclear nada, automáticamente el CD comienza la ejecución, comenzando por las preferencias de instalación del programa (Completa, Compacta, Personalizada) y la propia información del usuario para registrar el producto. Si no se tiene suficiente espacio en disco duro, se puede instalar la versión mínima y habrá que tener el CD en el lector de disco compacto cada vez que se quiera modificar una aplicación.

Cuando se inicia el programa C++ Builder presenta un grupo de ventanas dispersas alrededor de su ventana principal.

En la figura se presenta los principales elementos de C++ Builder (Ambiente de Desarrollo Integrado, por sus siglas en Ingles IDE). Cada parte en el ambiente de desarrollo trabaja conjuntamente, diseños visuales y editor de código donde la edición es similar a otros editores, solo que con el ambiente de desarrollo integrado, se puede observar realmente lo que esta construyendo al momento de crearlo.

Idealmente se debería trabajar con una resolución de 800 x 600 o tal vez mayor en su monitor, ya que estas resoluciones dan una sensación de amplitud.

Conceptos y funciones principales

Esta herramienta nos ha sido de gran utilidad a la hora de afrontar el proyecto. Hemos implementado una interfaz capaz de comunicarse mediante sockets con el DSP. La función principal de dicha interfaz es recibir la imagen a través de WiFi y mostrarla para ver el buen funcionamiento de los algoritmos que utilicemos durante el procesamiento de las imágenes.

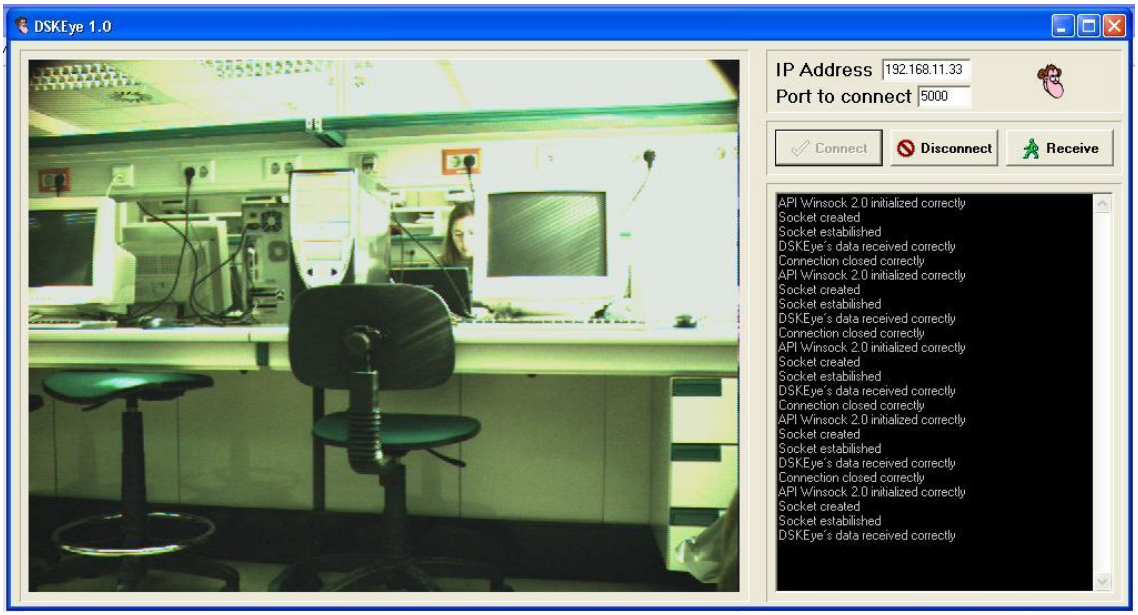


Figura 17: Entorno DSK Watcher

En la interfaz se distinguen varios elementos claramente diferenciados. Primero nos encontramos con la zona de gestión de la conexión. Desde este lugar podremos cambiar tanto la dirección IP como el puerto de escucha de la conexión. A través de los botones de “Conectar”, “Desconectar” y “Recibir” podremos decidir en cada momento el estado de nuestra conexión.

La segunda zona a destacar es la consola de información. En ella podremos ver las acciones que se han realizado en el proceso de visionado de la imagen. Esta zona nos ha sido de gran ayuda en el proceso de depuración de los programas que implementábamos ya que por el tipo de programas utilizados, los entornos de depuración de los que disponíamos y el gran tamaño de los datos que manejábamos, era más fácil utilizar este sistema para ayudarnos a ver los cambios en los estados del programa.

Por último nos encontramos con la parte de visionado de la imagen, donde se muestra la imagen captada por la cámara. Según avanzaba el proyecto, en esta zona no sólo se mostraba la imagen capturada, sino que también se podían ver imágenes intermedias en el recorrido del procesamiento de la imagen. Para elegir que imágenes queríamos mostrar en cada momento fue necesario añadir al entorno gráfico botones y elementos de selección antes de mostrar la imagen requerida.

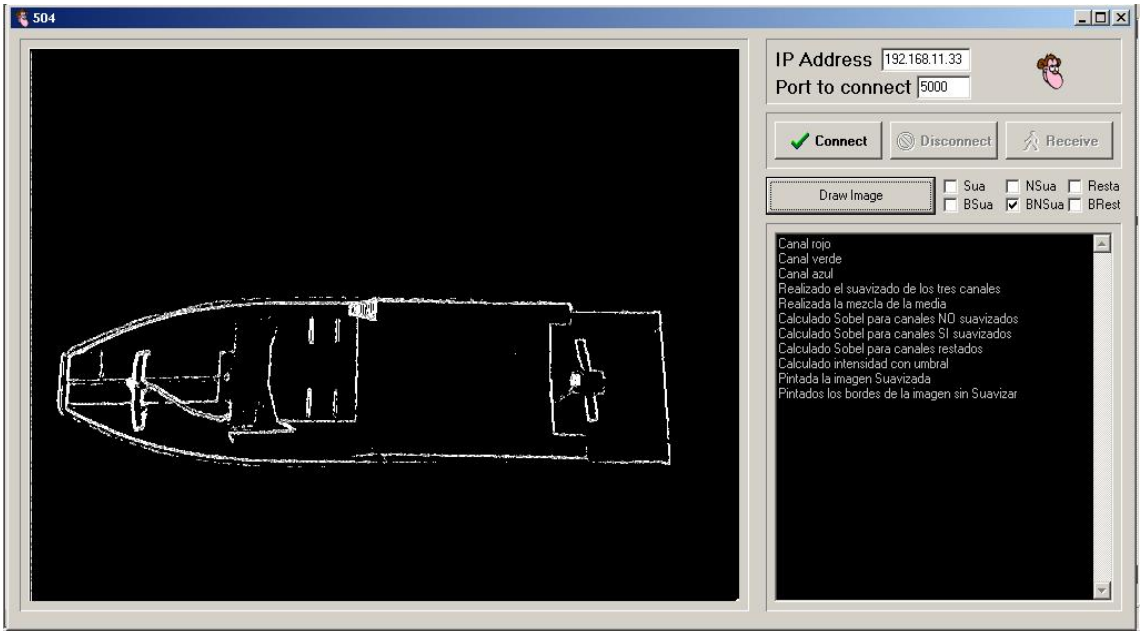


Figura 18: Bordes en el DSK Watcher

Aquí podemos ver un claro ejemplo en el que la imagen mostrada es aquella en la que hemos detectado los bordes de una imagen previamente captada. Mediante el menú hemos seleccionado dicha imagen y el programa solamente se encarga de mostrar la imagen pedida.

El Borland C++ Builder ha tenido especial importancia en el desarrollo de la parte de procesamiento de la imagen, dentro de las tres ramas del proyecto. Debido a la dificultad de tener que transportar el DSP fuera de la zona de trabajo asignada dentro de la facultad se optó por crear un programa paralelo que al recibir la imagen desde la cámara, guardara los datos de los tres canales recogidos en archivos de texto, para posteriormente trabajar con esa imagen en horas y lugares en los que era imposible acudir al lugar donde se encontraba el DSP. Con esos archivos, con los canales almacenados, se implementaron los diferentes algoritmos de tratamiento y reconocimiento que forman parte del proyecto. Después de la implementación se llevaban a cabo las diferentes pruebas y optimización del código. Todo esto era un paso previo a la implantación de dichos programas en el DSP. Debido a la incompatibilidad del código utilizado para programar el DSP y para implementar programas en el entorno Borland C++ Builder, aunque fue tenido en cuenta desde un primer momento, no fue fácil la tarea de migrar el código de un sistema a otro.

Otras referencias

Dentro del propio entorno de desarrollo podemos encontrar una gran información sobre el mismo entorno. Además posee una ayuda de gran utilidad a la que es necesario acudir en muchos momentos a lo largo del desarrollo del proyecto.

Si se necesita una información adicional sobre este entorno se puede acudir a la página oficial de Borland sobre este producto: http://www.codegear.com/products/cppbuilder

Además existe una gran cantidad de webs con ayuda y librerías que facilitan la programación y consiguen mejores resultados a la hora de trabajar con este IDE.

Microsoft Visual Studio 2005

Instalación y configuración del entorno

El entorno de desarrollo de Visual Studio 2005 de Microsoft ha sido utilizado principalmente para programar el decodificador de video h263. Los principales motivos por los que se eligió dicho entorno son los siguientes:

1. Al ser estudiantes, se nos permite usar dicho software tanto de forma particular como en los laboratorios de la facultad, por lo que su adquisición es gratuita bajo licencia exclusivamente docente.
2. La interfaz de programación es intuitiva y clara, facilitando enormemente el desarrollo de aplicaciones Win32 basadas en C o C++.
3. El sistema de ayuda y consulta interactiva es muy avanzado, con una gran base de datos de consulta acerca de las principales funciones del sistema y API de Windows.

Para poder instalar el entorno de programación es necesario tener los ficheros de instalación del Visual Studio 2005. Su configuración es automática y directa, por lo que no es necesario enlazar nuevas variables de entorno o modificar las ya existentes.

Los requisitos mínimos para poder trabajar de forma adecuada con el entorno son los siguientes:

- Ordenador a 600 MHz
- Microsoft® Windows® 2000 Service Pack 4,
  Microsoft® Windows® XP Service Pack 2
  Microsoft® Windows® XP Professional x64
  Microsoft® Windows Server™ 2003 Service Pack 1
  Microsoft® Windows Server™ 2003, x64
  Microsoft® Windows Server™ 2003 R2
  Microsoft® Windows Server™ 2003 R2, x64
  Microsoft® Windows Vista™
- 192 MB de RAM
- 2 GB de espacio en disco
- Unidad DVD-ROM
- Monitor de al menos 256 colores con una resolución mínima de 1024x768 píxels
- Teclado y ratón

Conceptos y funciones principales

Visual Studio 2005 permite programar en diferentes lenguajes de programación. Dado nuestro conocimiento previo en lenguajes C y C++, decidimos desde un principio optar por dicho lenguaje como plataforma básica para desarrollar el decodificador de video. Además, los ficheros fuente de código abierto del tmndec (bajo licencia GNU) estaban escritos en ANSI C y dado que nuestro decodificador se

basa en él prácticamente en su totalidad queda suficientemente justificada la decisión tomada acerca del lenguaje de programación que se usará con Visual Studio.

La herramienta de desarrollo de software de Microsoft tiene un entorno particular optimizado para programar en C/C++, llamado Visual Studio C++. Ésta rama de la suite de programación permite crear diferentes aplicaciones según qué objetivo se persiga, es decir, sobre que API va a correr nuestra aplicación.

En nuestro caso, el decodificador es un programa de consola Win32 por lo que no hará uso de las librerías gráficas propias de Windows ni de ningún Framework base o MFC alguna.

Para poder empezar a programar el decodificador debemos entrar en el entorno de desarrollo y crear un nuevo proyecto C++ de tipo "Consola Win32". Los pasos desde el comienzo son los siguientes:

1. Arrancamos el entorno de desarrollo Visual Studio 2005
2. Sobre la barra de menú pulsamos "Archivo->Nuevo->Proyecto"
3. Se nos abrirá una ventana como la siguiente en la que deberemos seleccionar el tipo de proyecto (en nuestro caso "Aplicación de consola Win32"):

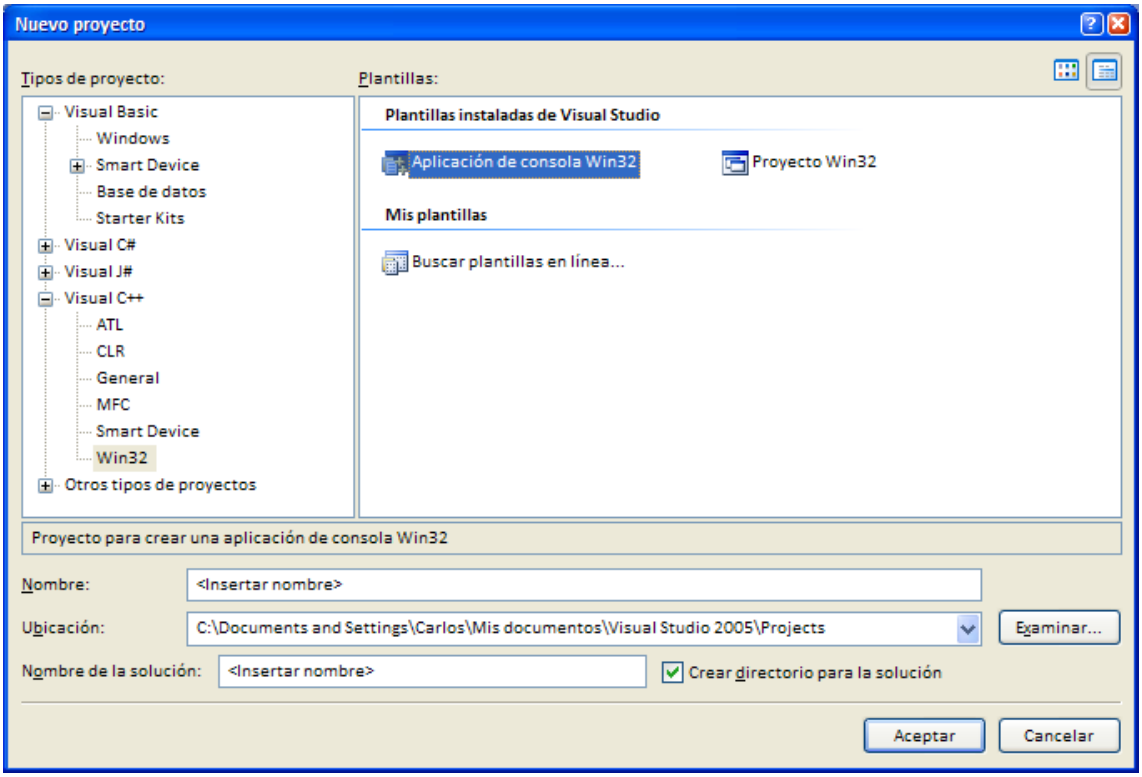


Figura 19: Selección de tipo de proyecto

4. Damos un nombre concreto al proyecto y pulsamos el botón "Aceptar".

Una vez tengamos listo nuestro proyecto será necesario empezar a crear los ficheros ".c" y las cabeceras ".h". Este proceso se detallará más a fondo en la sección de "Procesamiento y tratamiento de imagen en movimiento bajo el estándar ITU h.263".

Si observamos con atención, son muy pocas opciones las que necesitamos hacer uso. En principio, las principales son:

1. Crear un nuevo archivo: Para ello pulsamos "Archivo->Nuevo->Archivo" y seleccionamos su tipo (en nuestro caso ".c" o ".h").
2. Compilar la solución: Para ello hacemos click derecho sobre el proyecto en el explorador de proyecto (situado en la parte derecha superior de la pantalla) y seleccionamos la opción "generar solución".

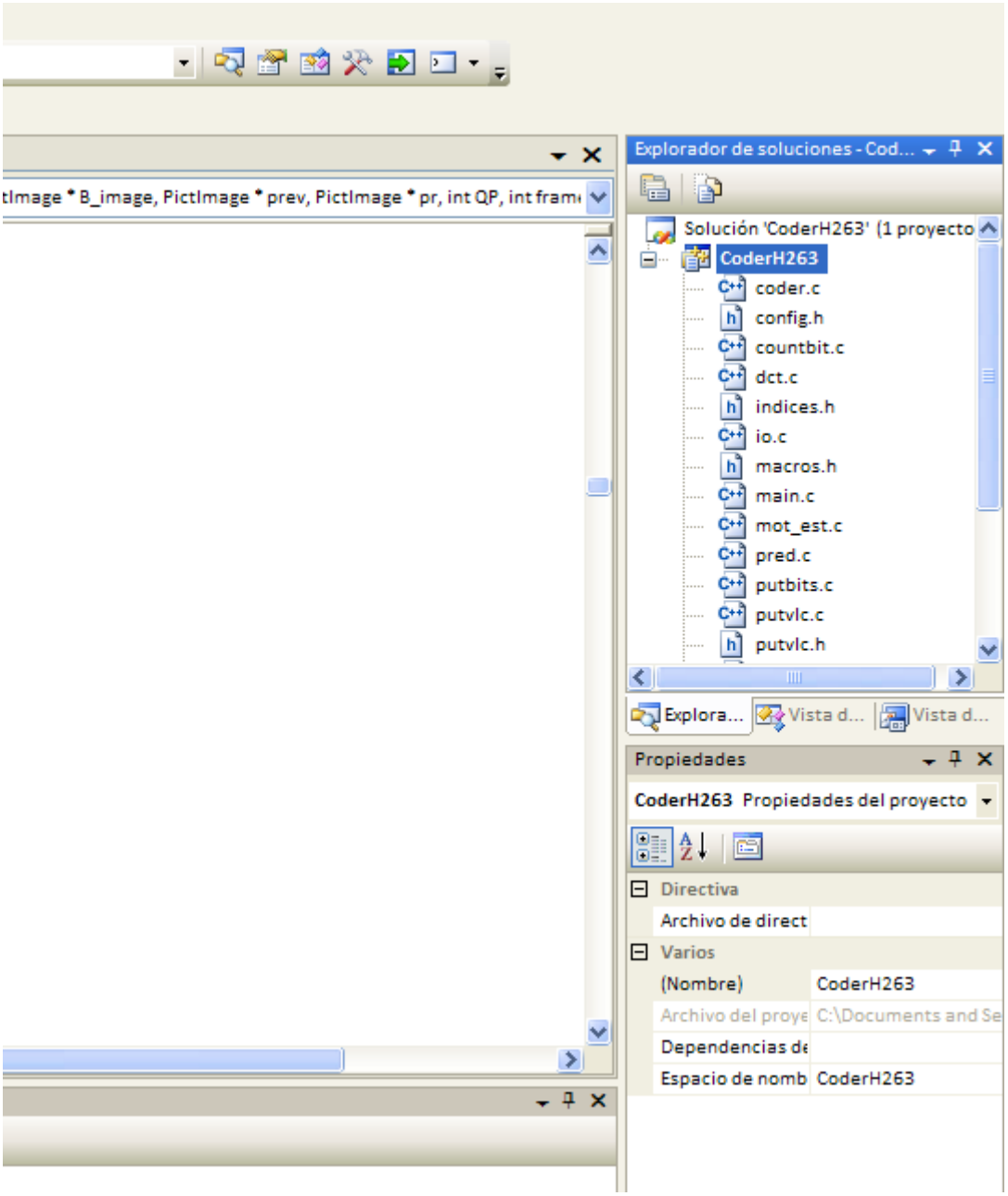


Figura 20: Proyecto y ficheros fuente

3. Consultar la ayuda en línea MSDN: Para acceder al sistema de ayuda interactivo es necesario pulsar “Ayuda->Buscar” y seleccionar el modo de búsqueda que queramos (online o ayuda local bajo base MSDN).

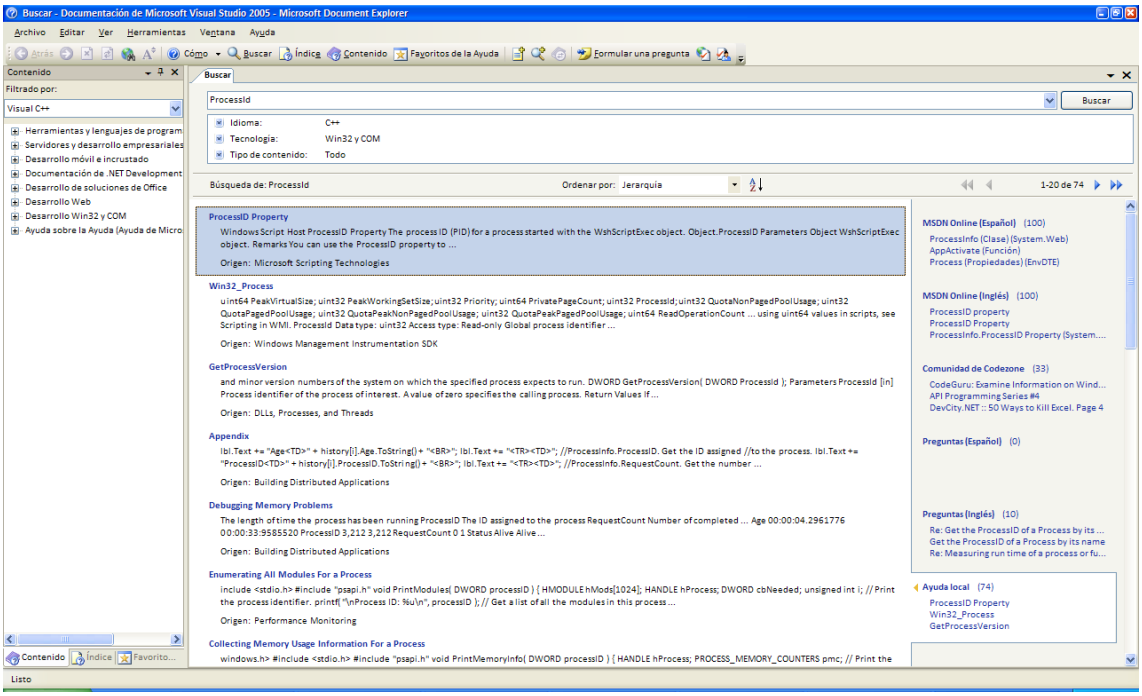


Figura 21: Ayuda del Visual Studio 2005

4. Depurar la solución: Para ello deberemos pulsar en el botón “Debug” (flecha de color verde orientada hacia la derecha y situada en la parte superior central de la pantalla) e ir trazando con ayuda de puntos de ruptura previamente establecidos en determinadas secciones de código.

De todas formas, existe abundante documentación acerca de este entorno de programación. Para más información se pueden consultar los links de la sección siguiente.

Otras referencias

A continuación se detallan algunas de las más importantes páginas de referencia acerca de este entorno de programación multiplataforma desarrollado por Microsoft:

Información general:

http://www.microsoft.com/spanish/msdn/vs2005/default.mspx

http://www.microsoft.com/spanish/msdn/vs2005/editions/pro/default.mspx

http://msdn2.microsoft.com/es-es/vstudio/default(en-us).aspx

http://www.microsoft.com/spanish/msdn/centro_recursos/vs2005/default.mspx

Soporte C/C++ para Visual Studio 2005:

http://www.microsoft.com/spanish/msdn/vstudio/express/VC/default.mspx

Otros programas

En la parte de reconocimiento de LEDs y del barco hemos hecho uso de otros programas, para ver diferentes técnicas y distintos resultados de efectos y algoritmos que hemos ido implementando.

Los dos programas que hemos utilizado han sido el entorno MATLAB 7.0.1 y Adobe Photoshop CS2. Nos han resultado de gran utilidad para probar los resultados en las imágenes captadas, antes de implementarlas en el DSP. Esto nos ha hecho ahorrar bastante tiempo, ya que ha evitado que perdiéramos tiempo implementando algoritmos que no hubieran sido de ninguna utilidad para conseguir los objetivos marcados en el proyecto.

Desarrollo del proyecto

Como anteriormente habíamos citado, este proyecto de Sistemas Informáticos está subdividido en 3 subproyectos, que serán detallados todos ellos manteniendo una misma estructura con la finalidad de ayudar al lector a comprender cada una de las motivaciones que han permitido llevarlos a cabo.

Codificación de imágenes estáticas bajo el estándar JPEG 2000

¿De qué partimos?

Inicialmente contamos con el C6713 DSK y la placa de expansión DSKeye, que proporciona un sistema de captura de imágenes y una interfaz wifi bajo protocolo TCP/IP junto a la capacidad de proceso del DSP.

El sistema DSKeye viene acompañado por un software con las librerías para usar la cámara y el módulo wifi y dos proyectos de ejemplo. Uno de ellos, que es el utilizado en este apartado del proyecto, realiza una captura y la envía al navegador de un PC conectado mediante wifi al DSP en respuesta a una petición de éste. Para ello no realiza ningún tipo de modificación de los datos de la cámara, sólo envía una cabecera BMP y luego los datos de los píxeles línea a línea. Es de este proyecto de ejemplo del que partimos para completar los objetivos.

El estándar JPEG-2000 surge como una mejora o actualización del estándar JPEG, ambos del Joint Photographic Expert Group. Se basa en la transformación de la imagen mediante *wavelets*, en lugar de utilizar la transformada directa del coseno como en JPEG. En teoría, las imágenes codificadas mediante JPEG-2000 presentan menos irregularidades debidas a la compresión que las codificadas en JPEG, aunque uno de sus problemas es que también pierden algo de nitidez.

En la práctica, el formato JPEG-2000 no ha podido de momento sustituir al JPEG, y por norma general los navegadores no lo soportan. Aunque hay extensiones para ellos y aplicaciones que permiten trabajar con el nuevo formato no ha tenido el impacto que en su día tuvo JPEG ni cuenta con la variedad de soluciones que existen para su predecesor. En este proyecto utilizaremos la implementación del estándar que realizar la librería JASPER.

La motivación de esta librería es proporcionar soporte en código abierto a aplicaciones que realicen conversiones entre JPEG-2000 y otros formatos, por lo que su estructura es muy modular, y pueden añadirse nuevos formatos a la misma sólo indicando las funciones de codificación y decodificación asociadas a cada uno y añadiendo el código fuente de los *códecs*.

En cuanto al *códec* JPEG-2000 de JASPER, se trata de una implementación altamente portable y muy modular realizada en C/C++, que permite su configuración para un gran número de plataformas con características muy diferentes. Es necesario señalar que está librería no está pensada para la arquitectura del DSP ni orientada a ofrecer el mejor rendimiento en ninguna plataforma, y que dada la escasa adopción del JPEG-2000 no existen soluciones de código abierto para mejorar el rendimiento en las partes críticas del algoritmo. La estructura del *códec* es la siguiente:

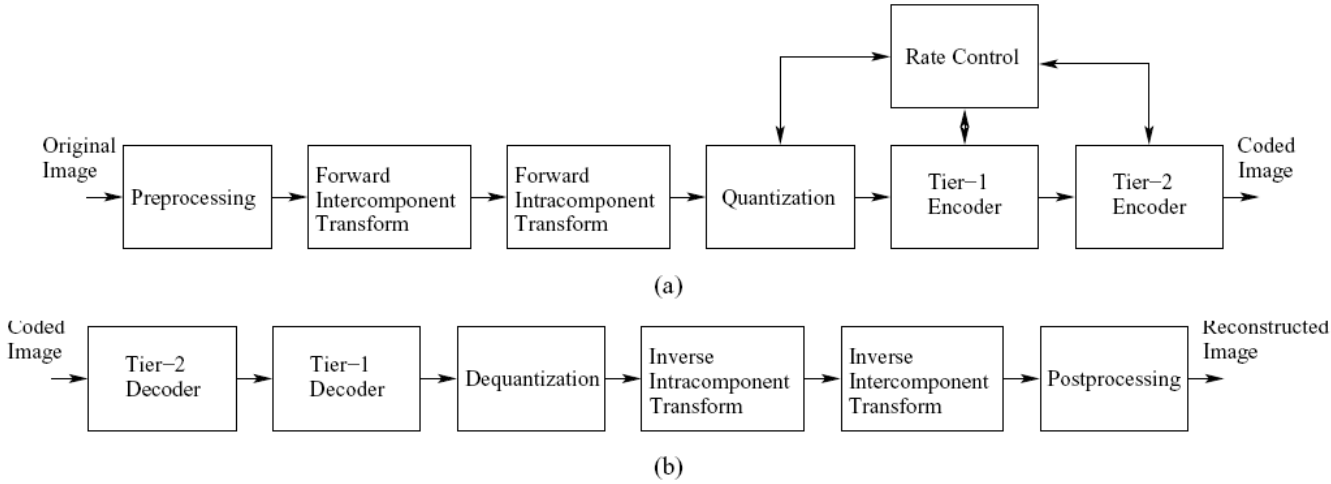
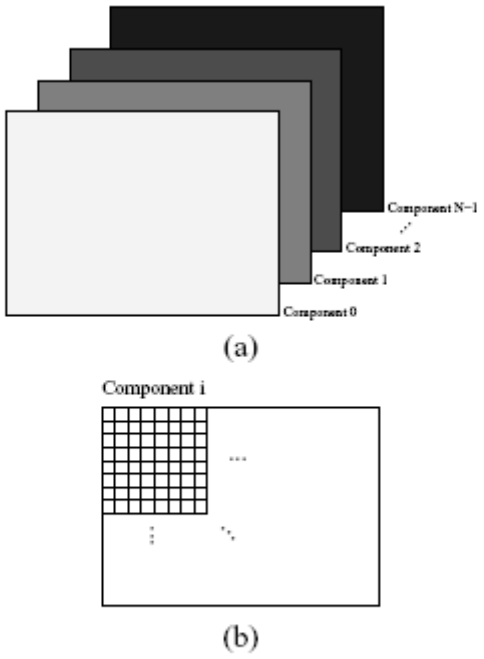


Fig. 5. Codec structure. The structure of the (a) encoder and (b) decoder.

La imagen original debe corresponder al modelo de una lista de planos, cada uno correspondiente a un componente de la imagen, es decir, una imagen en color RGB consta de tres componentes, cada uno de los cuales contiene la información de un color (rojo, verde y azul) para cada píxel de la imagen. Así, una imagen con resolución VGA (640x480) consiste en tres listas de 640x480 valores cada uno de los cuales representa la intensidad del color correspondiente a la lista en ese punto concreto de la imagen. Además a cada imagen se le asocia una matriz de referencia, que indica el origen de la imagen y el sentido de lectura de los píxeles.



Source image model. (a) An image with N components. (b) Individual component.

Preprocesado/Postprocesado de la imagen. Se espera que el rango de valores de los píxeles esté centrado alrededor del cero, por lo que si tenemos valores sin signo para cada píxel y P bits por píxel, el preprocesamiento consiste en restar a cada valor 2^{P-1} para centrar los valores en el rango $[-2^{P-1}$,

2^{P-1}-1]. El postprocesado consiste precisamente en deshacer esta operación sumando la misma cantidad para recuperar los simples sin signo originales. Si se utilizan muestras con signo, este paso no es necesario.

Intercomponent Trasform. Esta operación tiene como objetivo reducir la correlación entre los componentes de la imagen, y se realiza sobre los tres a la vez. En JPEG-2000 sólo se contemplan dos tipos de transformación entre componentes, ICT (Irreversible Color Transform) y RCT (Reversible Color Transform). Las dos consisten en transformar el espacio de color de RGB a YCbCr, por lo que se espera que la imagen tenga al menos tres componentes, siendo el 0 el componente rojo, el 1 verde, y el 2 correspondiente al azul. ICT es un procedimiento no reversible y que opera con valores reales, mientras que RCT es reversible y utiliza una aproximación entera a la ICT. Debido a la naturaleza de estas transformadas, deben operar con datos muestreados con la misma precisión(es decir, el mismo tamaño). Como consecuencia de esto, tanto la ICT cono la RCT requieren para su empleo que la imagen a codificar tenga al menos tres componentes, y que los tres primeros componentes estén muestreados con la misma resolución. La ICT sólo debería usarse en caso de que se desee una codificación con pérdida, mientras la RCT puede usarse tanto para compresión con pérdida como sin ella. La decisión de utilizar una transformada intercomponente depende del codificador, y no es imprescindible aplicarla para llevar a cabo la codificación. Después de esta transformación los datos de cada componente se tratan siempre por separado. La ICT es la transformada clásica de RGB a YCrCb, dada por la siguiente relación:

$$\begin{bmatrix} V_o(x,y) \\ V1(x,y) \\ V2(x,y) \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} U_0(x,y) \\ U_1(x,y) \\ U_2(x,y) \end{bmatrix}$$

Donde U_0, U_1, U_2 son los componentes de entrada que corresponden a rojo, verde y azul, y V_0, V_1, V_2 son los componentes resultantes, que corresponden a los canales Y, Cr y Cb respectivamente. La transformada inversa se realiza mediante esta operación:

$$\begin{bmatrix} U_o(x,y) \\ U_1(x,y) \\ U_2(x,y) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34413 & -071414 \\ 1 & -1.772 & 0 \end{bmatrix} \begin{bmatrix} V_0(x,y) \\ V_1(x,y) \\ V_2(x,y) \end{bmatrix}$$

La transformada entera reversible no es más que una aproximación con valores enteros a la ICT, dada por las siguientes ecuaciones:

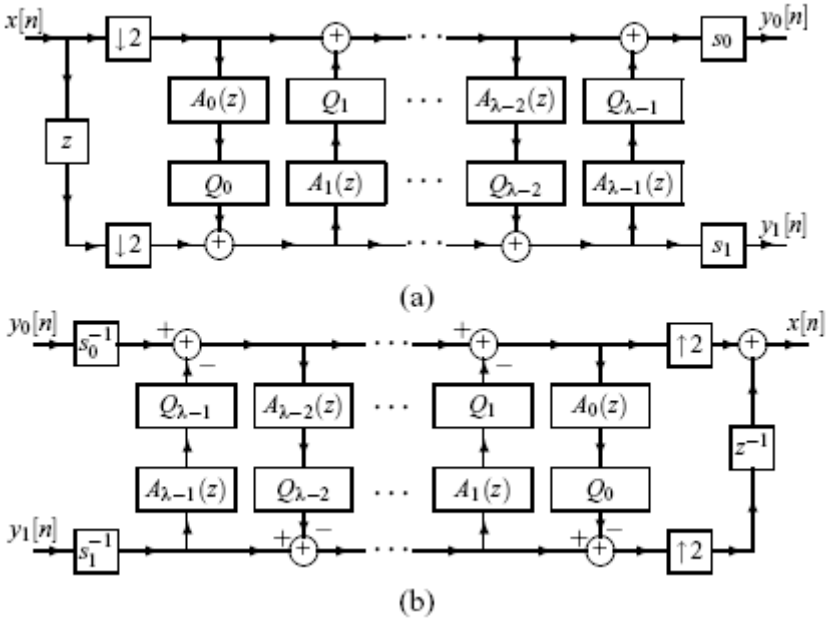
$$V_0(x,y) = \left\lfloor \tfrac{1}{4}\left(U_0(x,y) + 2U_1(x,y) + U_2(x,y)\right) \right\rfloor$$
$$V_1(x,y) = U_2(x,y) - U_1(x,y)$$
$$V_2(x,y) = U_0(x,y) - U_1(x,y)$$

La inversa de la RCT se obtiene del siguiente sistema:

$$U_1(x,y) = V_0(x,y) - \left\lfloor \tfrac{1}{4}\left(V_1(x,y) + V_2(x,y)\right)\right\rfloor$$
$$U_0(x,y) = V_2(x,y) + U_1(x,y)$$
$$U_2(x,y) = V_1(x,y) + U_1(x,y)$$

Cualquiera que sea la transformada multicomponente que se aplique en esta etapa durante la codificación, su inversa se aplica al realizar la decodificación. Aunque la transformada sea reversible, la inversa sólo puede ser aproximada dependiendo de la precisión aritmética finita con la que se trabaje.

Intracomponent Transform. Se trata de la transformación de cada componente por separado. Aquí es dónde entra en juego la transformada *wavelet*. Es un cambio introducido con respecto a JPEG, en el que se usaba la transformada del coseno, sin embargo, la transformada *wavelet* permite obtener imágenes con una granularidad más fina que la transformada directa del coseno, en imágenes con el mismo ratio de compresión. Aplicando esta transformación, cada componente se divide en varias bandas de frecuencia. Debido a las propiedades estadísticas de estas subbandas, la transformada permite que la codificación sea más eficiente. También puede aplicarse de forma reversible de enteros a enteros, o de forma irreversible con valores reales. La transformada inversa que se realiza en el decodificación consiste simplemente en deshacer este proceso, aunque debido a la precisión de la representación no se puede garantizar que no se pierda información al aplicar la transformada inversa.
El bloque básico de estas transformadas se conoce como 1-D 2-Channel perfect-reconstruction(PR) uniformly-maximally-decimated(UMD) filter bank(FB), cuyo diagrama de bloques general es el siguiente:

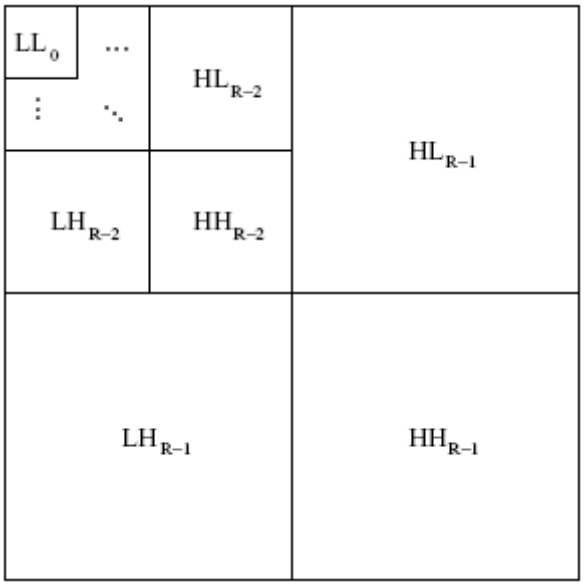


La parte a) corresponde a la trasformada directa para codificación (análisis), y la parte b) se asocia a la transformada inversa (síntesis). En la librería JASPER se centran es esta implementación de la transformada, porque permite realizar tanto la transformación reversible de enteros a enteros como la no reversible mediante números reales. En el diagrama, tomando 0<=i<λ, $\{A_i\}$ representa funciones de filtrado, $\{Q_i\}$ funciones de cuantización y $\{s_i\}$ ganancias escalares. Para construir la transformada reversible se utilizan $\{Q_i\}$ que generen sólo valores enteros y las ganancias escalares también se eligen como enteros, mientras que para la transformada irreversible se toma simplemente la función identidad para $\{Q_i\}$ y valores reales para las ganancias. Puesto que una imagen es una señal de dos dimensiones necesitamos un 2-D UMDFB, éste se obtiene aplicando el 1-D UMDFB en ambas direcciones vertical y horizontal. La transformada *wavelet* se calcula aplicando recursivamente el 2-D UMDFB a la señal de baja frecuencia de la subbanda obtenida en cada nivel de descomposición.

Una transformada de nivel R-1 se asocia con R niveles de resolución, numerados desde 0 hasta R-1, con 0 y R-1 correspondientes a la peor y la mejor resolución, respectivamente. Supongamos que se va a emplear una transformada de nivel R-1, por ejemplo. Para calcular la transformada directa, se aplica el 2-D UMDFB a los datos del componente de manera iterativa, produciendo un cierto número de subbandas. Cada aplicación del banco de filtrado produce 4 subbandas:

1. Paso-baja horizontal y vertical(LL)
2. Paso-baja horizontal y Paso-alta vertical (LH)
3. Paso alta horizontal y Paso-baja vertical (HL)
4. Paso-alta horizontal y vertical (HH)

Cada subbanda se define por su orientación (LL, LH, HL y HH) y por su nivel de resolución. EL fragmento de imagen de entrada se considera la subbanda LL_{R-1}. En cada nivel de resolución, la banda LL es divida en subbandas de nuevo. Este proceso se repite hasta obtener LL0 y produce la siguiente estructura de subbandas en la imagen:



Tras describir el cálculo de la transformada en general, explicaremos las dos transformadas concretas que se utilizan en el códec básico. La transformada reversible de enteros a enteros es la llamada transformada 5/3, y su 1-D UMDFB tiene los siguientes parámetros:

$$\lambda = 2, \quad A_0(z) = -\tfrac{1}{2}(z+1), \quad A_1(z) = \tfrac{1}{4}(1+z^{-1}),$$
$$Q_0(x) = -\lfloor -x \rfloor, \quad Q_1(x) = \lfloor x + \tfrac{1}{2} \rfloor, \quad s_0 = s_1 = 1.$$

La transformada no reversible de reales a reales se conoce como la transformada 9/7, y los parámetros del filtro 1-D UMDFB que utiliza son los siguientes:

$$\lambda = 4, \quad A_0(z) = \alpha_0(z+1), \quad A_1(z) = \alpha_1(1+z^{-1}), \quad ($$
$$A_2(z) = \alpha_2(z+1), \quad A_3(z) = \alpha_3(1+z^{-1}),$$
$$Q_i(x) = x \text{ for } i = 0,1,2,3,$$
$$\alpha_0 \approx -1.586134, \quad \alpha_1 \approx -0.052980, \quad \alpha_2 \approx 0.882911,$$
$$\alpha_3 \approx 0.443506, \quad s_0 \approx 1/1.230174, \quad s_1 = -1/s_0.$$

Cuantizacíon/Decuantización. Después de la transformación inter e intracomponente, los coeficientes son cuantizados. La cuantización permite conseguir una mayor compresión, ajustando el nivel de precisión de los coeficientes al mínimo requerido por el nivel de calidad deseado. Aquí es donde se descarta la mayor parte de la información al aplicar compresión con pérdida. Los coeficientes cuantizados se obtienen aplicando la siguiente operación a los coeficientes resultantes de la transformada:

$$V(x,y) = \lfloor |U(x,y)| / \Delta \rfloor \operatorname{sgn} U(x,y)$$

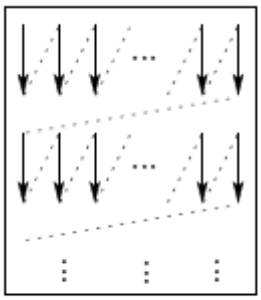
Donde Δ representa el tamaño del paso de cuantización.

Asociado a los modos de operación del códec existen dos pasos de cuantización diferentes. En caso del modo reversible de enteros a enteros, el paso de cuantización se considera la unidad, con lo que este paso se omite y los coeficientes cuantizados son iguales que los producidos por la transformada intracomponente. En el caso del modo real, los coeficientes se eligen en función del ratio de compresión requerido. El proceso inverso en el decodificador, o decuantización, no suele ser reversible, y consiste en la operación siguiente:

Tier-1 Coding. Éste es el primer paso de codificación que se va a aplicar sobre los coeficientes cuantizados. Cada componente se divide en celdas rectangulares llamadas *codeblocks*. Cada *codeblock*

$$U(x,y) = (V(x,y) + r \operatorname{sgn} V(x,y))\,\Delta$$

se codifica de forma independiente mediante un bit-plane coder. Durante la codificación *bit-plane*, no se explotan interdependencias entre subbandas, ya que de esa forma se reducen los errores y la pérdida de información, y tienen lugar tres pasos de codificación en cada *codeblock*. Los pasos son los siguientes: significación, refinamiento y limpieza. Los tres pasos recorren cada codeblock en el orden siguiente:



Sample scan order within a code block.

En el primer paso de codificación se obtiene información de significancia y signo para los píxeles que aún no se consideran como significantes pero que se predice que pasarán a serlo durante el proceso actual. El algoritmo en pseudocódigo es el siguiente:

Algorithm 1 Significance pass algorithm.

```
1: for each sample in code block do
2: if sample previously insignifi cant and predicted to become signifi cant dur-
   ing current bit plane then
3: code signifi cance of sample /* 1 binary symbol */
4: if sample signifi cant then
5: code sign of sample /* 1 binary symbol */
6: endif
7: endif
8: endfor
```

El segundo paso afecta a los bits que siguen al bit de mayor significancia para cada muestra. Si un bit se consideraba significante en un bit-plane anterior, el siguiente bit más significante se transporta usando un único símbolo binario. El algoritmo es el siguiente:

Algorithm 2 Refinement pass algorithm.

```
1: for each sample in code block do
2: if sample found signifi cant in previous bit plane then
3: code next most signifi cant bit in sample /* 1 binary symbol */
4: endif
5: endfor
```

El tercer y último paso es el limpiado. En él se tratan los simples que se consideran insignificantes y que no se espera que cambien de estado para el bit-plane actual. El pseudocódigo en este paso es el siguiente:

Algorithm 3 Cleanup pass algorithm.

```
1: for each vertical scan in code block do
2: if four samples in vertical scan and all previously insignifi cant and unvisited
   and none have signifi cant 8-connected neighbor then
3: code number of leading insignifi cant samples via aggregation
4: skip over any samples indicated as insignifi cant by aggregation
5: endif
6: while more samples to process in vertical scan do
7: if sample previously insignifi cant and unvisited then
8: code signifi cance of sample if not already implied by run /* 1 binary symbol
   */
9: if sample signifi cant then
10: code sign of sample /* 1 binary symbol */
11: endif
12: endif
13: endwhile
14: endfor
```

Este proceso genera una lista de bits que contiene la información codificada.

Tier-2 Coding. En este paso de codificación se toma como entrada la lista de bits de Tier-1 y se agrupan en unidades de codificación llamadas *packets*. Cada *packet* consiste en una cabecera y un cuerpo del paquete. La cabecera indica que tipo de codificación se ha aplicado sobre los datos, mientras el cuerpo contiene los datos propiamente dichos. La cabecera se codifica en función del componente, nivel de resolución, capa de la imagen, y posición en la imagen. Esto supone que puede haber paquetes

vaciós, ya que se requieren que exista uno para cada combinación de dichos parámetros, aunque no incluya nueva información sobre los datos. Sobre el cuerpo no se realiza codificación adicional. El algoritmo para codificar las cabeceras en pseudocódigo es el siguiente:

Algorithm 4 Packet header coding algorithm.

```
1:  if packet not empty then
2:  code non-empty packet indicator /* 1 binary symbol */
3:  for each subband in resolution level do
4:  for each code block in subband precinct do
5:  code inclusion information /* 1 binary symbol or tag tree */
6:  if no new coding passes included then
7:  skip to next code block
8:  endif
9:  if first inclusion of code block then
10: code number of leading insignificant bit planes /* tag tree */
11: endif
12: code number of new coding passes
13: code length increment indicator
14: code length of coding pass data
15: endfor
16: endfor
17: else
18: code empty packet indicator /* 1 binary symbol */
19: endif
20: pad to byte boundary
```

Los tres últimos pasos (la cuantización, y los dos de codificación) pueden someterse a control para obtener un ratio de compresión determinado. El valor del ratio puede ajustarse mediante dos mecanismos: la elección de los niveles de cuantización, o bien la elección de los subconjuntos de pasos de codificación que se desean incluir. El primer mecanismo sólo se usa en modo no reversible de reales a reales, ya que en el modo de entero a entero el nivel de cuantización siempre es igual a la unidad. El segundo permite al códec descartar algunos pasos de codificación para reducir el tamaño de los datos codificados.

¿Cuál es el objetivo?

El objetivo del proyecto es realizar la codificación de la imagen bajo el estándar JPEG-2000 mediante el DSP antes de mandarla por wifi. Los resultados deben compararse con lo que supone mandar un BMP directamente, sin compresión. Se espera que dado el menor tamaño de la imagen en JPEG-2000 el tiempo de envío se reduzca.

Estructura del subproyecto

Para realizar los objetivos del proyecto y poder comprobarlos, es necesario dividir el proyecto en dos apartados: la implementación del algoritmo en el DSP, y la creación de un programa de PC para solicitar los datos e interpretarlos. Además, los dos proyectos pasarán conjuntamente a través de varias etapas, que parten del envío de los datos sin procesamiento hasta el envío de un *JPEG-2000 code-stream*.

Programa decodificador: Este programa realizado en C++ permite comunicarse con el programa que se ejecuta en el DSP, a través de la interfaz TCP/IP que proporciona el módulo wifi. El programa tiene las siguientes funcionalidades:

- Abrir un socket de comunicación con el DSP en la IP y el puerto que haya sido configurado y realizar la conexión.
- Solicitar los datos al DSP y recibirlos adecuadamente.
- Decodificar los datos del DSP.
- Mostrar la imagen recibida.
- Guardar la imagen en disco.

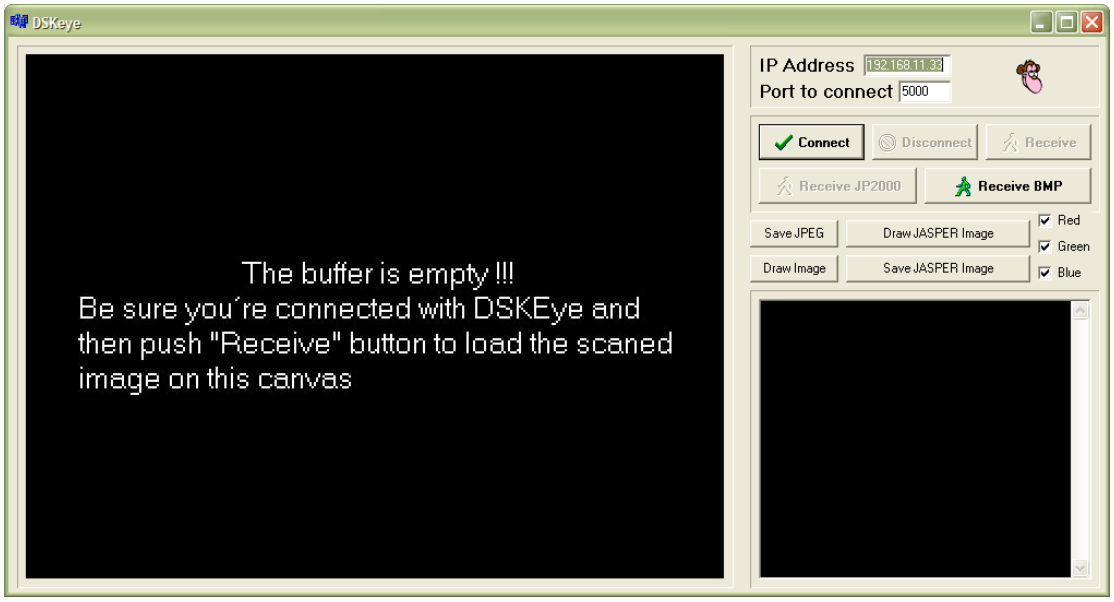


Figura 27: GUI del programa receptor

Programa de captura: Este es el programa que está ejecutando el DSP. Su funcionamiento básico consiste en una serie de acciones cuyo orden es el siguiente:

- Inicialización de las librerías para el control de la cámara y la interfaz de red.
- Espera de la conexión desde el decodificador.
- Recibe la petición del decodificador y la interpreta adecuadamente.
- Captura la imagen.
- Preprocesado de la imagen.
- Procesado y codificación de la imagen.
- Envío de la imagen.
- Cierren de conexión e inicio de un nuevo ciclo de espera.

Inicialmente, el DSP capturaba la imagen en formato RAW, es decir, se hacía una captura entrelazada de píxeles, por lo que para obtener el valor del píxel correspondiente se requería tomar el sample de esa posición, que contenía una componente del píxel y los dos samples adyacentes que contenían las componentes que faltaban. Además, para conseguir un nivel de color y contraste adecuado, era necesario realzar los componentes rojos y azules de la imagen.

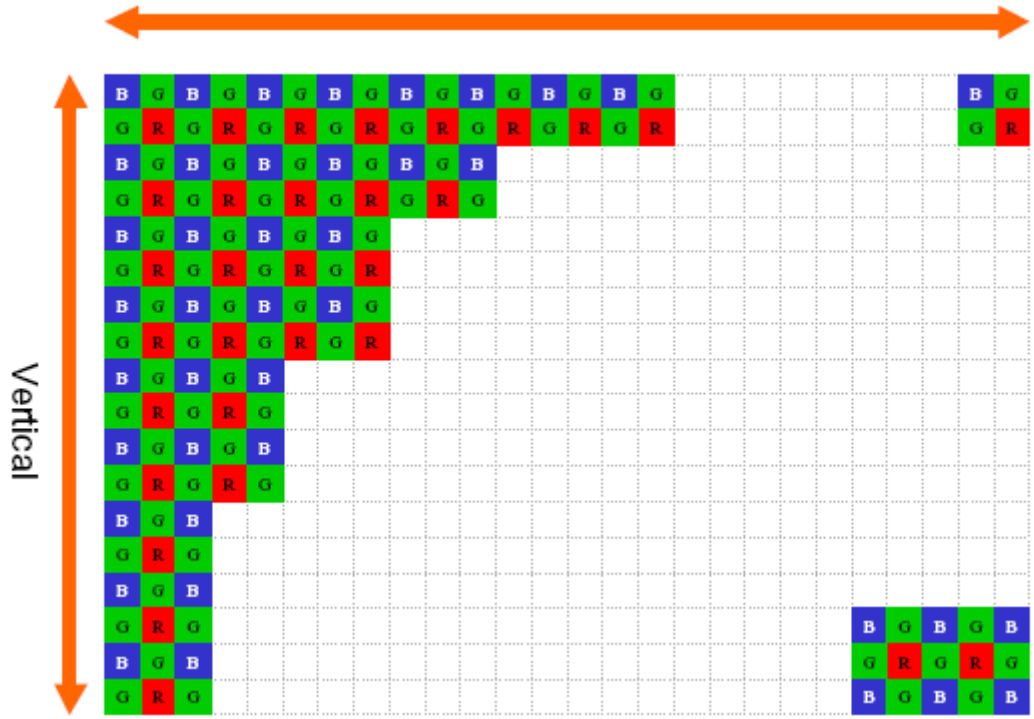


Figura 28: Organización en píxels

La matriz de píxeles que resulta de este proceso tiene tres veces el tamaño del frame de componentes entrelazados inicial. En la primera etapa de desarrollo, se enviaba una cabecera BMP y esta matriz al decodificador en el PC, que las recibía e interpretaba con funciones *ad-hoc*. Esta fase permitió ajustar los valores de corrección de color que se aplican en el preprocesado.

La siguiente etapa consistió en configurar e introducir la librería JASPER en el programa decodificador del PC receptor, para comprobar su funcionamiento. Para ello, se editó el archivo *jas_config.h* que permite especificar los recursos disponibles en la arquitectura destino de la implementación, además de solucionar errores o fallos de adaptación que impedían la compilación correcta de la librería inicialmente. En el DSP también se realizaron cambios, ya que la cabecera BMP se unió a los datos de los píxeles para enviarlo como si se tratara de un archivo BMP, la funcionalidad que se buscaba era que el DSP enviase un BMP y fuera el PC receptor el que lo codificase en JPEG2000 haciendo uso de la librería. Una vez en el decodificador, los datos se decodificaban mediante las funciones de la librería y se mostraban en pantalla. Además se introdujo la posibilidad de guardar la imagen en formato JPEG-2000, realizando la codificación en el PC receptor.

Una vez comprobado que la librería funcionaba correctamente en el PC y que su rendimiento era bueno, las siguientes etapas consistían en trasladar la funcionalidad de la librería al DSP, para ello, hubo que configurar los recursos del sistema de nuevo mediante el archivo *jas_config.h* limitando el uso de archivos y librerías del sistema a las disponibles para el DSP, hasta conseguir que realizara la codificación en JPEG-2000 de los datos. Para ello, se introdujo un paso intermedio de decodificación del BMP al formato de imagen proporcionado por la librería para aplicar la conversión sobre este último. Entre otros problemas debidos a que la librería no está optimizada para la arquitectura del C6713, en esta etapa se producía un desbordamiento de pila durante el análisis de las subbandas de frecuencia, recordemos que es en esta etapa donde se ponen de manifiesto las propiedades estadísticas de la imagen que permiten su compresión óptima y su codificación de forma eficiente. Para poder efectuar la transformación intracomponente al menos una vez sobre cada canal RGB de la imagen, la rutina

recursiva que efectuaba el cálculo hubo de ser modificada por una versión iterativa de la misma. También se adaptó el programa receptor introduciendo la posibilidad de solicitar tanto la codificación en BMP como en JPEG-2000 para poder comparar el rendimiento de ambos procesos más fácilmente.

A continuación se detallan sendos diagramas de flujo que describen exactamente cada uno de los procesos llevados a cabo en este apartado:

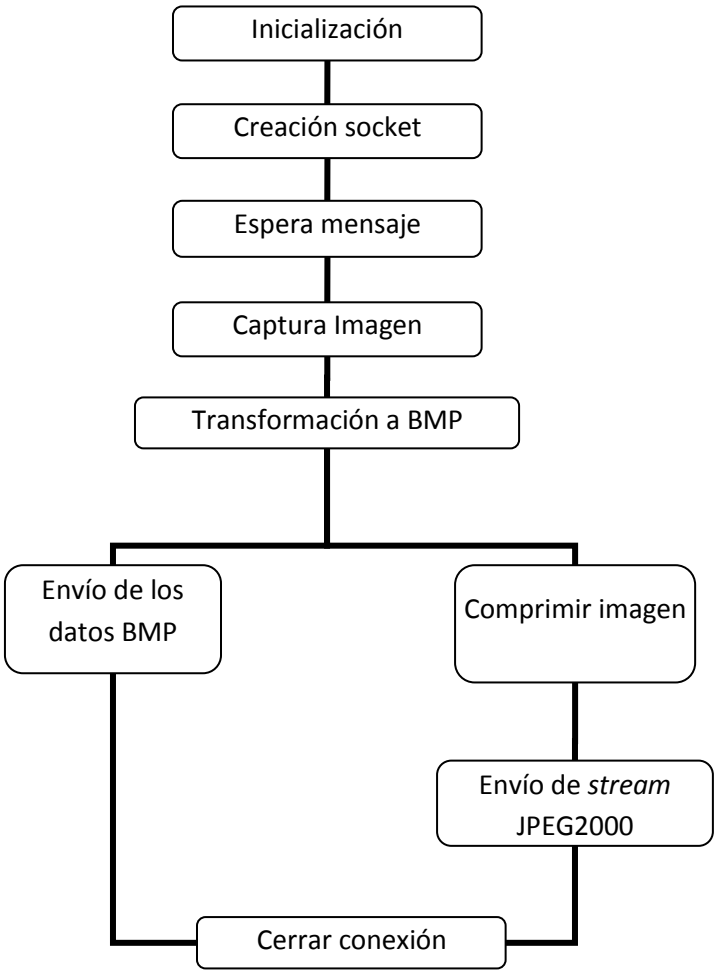


Figura 29: Diagrama de flujo principal

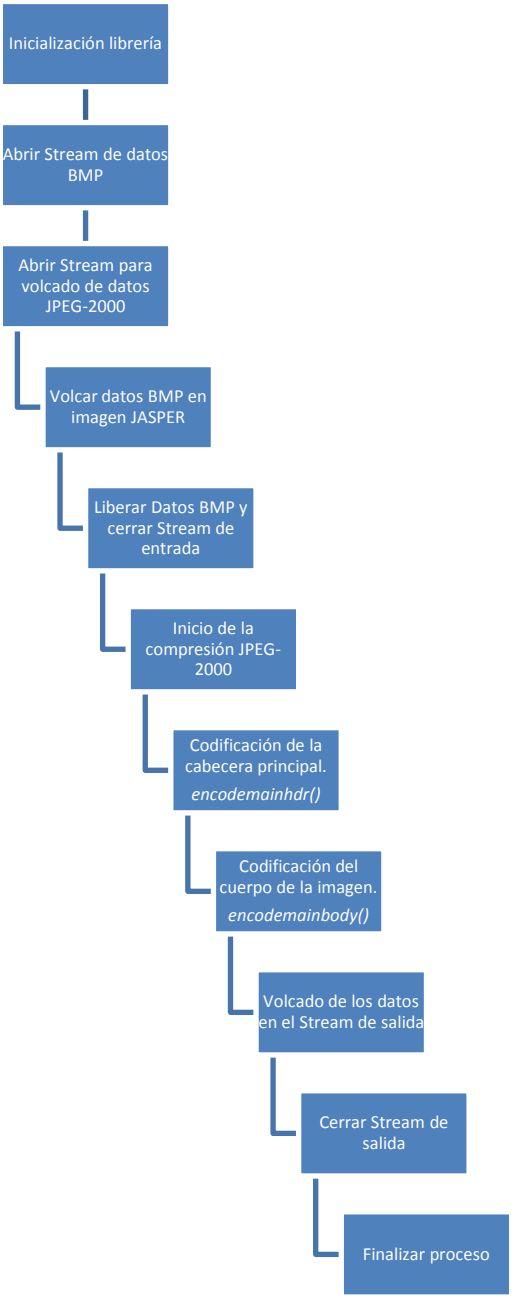


Figura 30: Estructura del compresor

Y ya por último, comentar brevemente las principales funciones y ficheros fuente de utilidad que han servido para desarrollar este subproyecto:

Main.c

Es el fichero principal de la aplicación. En él se encuentran definidas, entre otras, las funciones:

- toBMP() .- Transforma el frame capturado por la cámara a un mapa de bits interpolando cada componente a partir de sus vecinos.
- sendData() .- Envía una cantidad de datos especificada al socket abierto durante la inicialización en el programa principal.

Jas_image.c

Es el fichero de la librería que gestiona el modelo de imagen. Entre sus funciones se encuentra la creación de imágenes y componentes de una imagen, así como su destrucción y la interfaz de codificación/decodificación a los formatos soportados.

Jas_stream.c

Contiene las funciones que la librería emplea para la gestión de la entrada/salida con memoria o ficheros. Entre sus funciones están:

- jas_stream_memopen() .- Prepara una variable en memoria para que funcione como búfer de datos. Permite abrir streams de tamaño fijo o variable.
- Jas_stream_flush() .- Fuerza el volcado de los datos al búfer correspondiente, en caso de que todavía no se haya llevado a cabo.
- Jas_stream_close() .- Cierra el stream, liberando el espacio que ocupa, salvo los datos del búfer, que permanecen alojados en memoria, en el objeto subyacente.

Jpc_enc.c

Contiene el algoritmo de codificación en JPEG-2000 *code stream*. Las funciones implicadas en este proceso son, entre otras:

- Encodemainhdr() .- Codifica la cabecera del JPEG2000 y la escribe en el stream de salida.
- Encodemainbody() .- Realiza el grueso de la codificación, y gestiona las llamadas a otras funciones.
- Rateallocate() .- Se utiliza para ajustar el ratio de compresión de los datos al valor que recibe el códec como parámetro.

Jpc_tfsb.c

Contiene la función tfsb_analize(), que es la encargada de realizar la transformación intracomponente y el análisis en subbandas de los datos.

Jpc_t1enc.c

Agrupa las funciones necesarias para realizar la codificación Tier-1, siendo la principal de ellas enccodeblks(), que realiza la codificación de una serie de *code-blocks*.

Jpc_t2enc.c

Contiene la función enccodepackets(), que se corresponde con la etapa de codificación Tier-2.

DSK6713_camera.lib

Se trata de la librería que ofrece la placa de expansión con las funciones necesarias para ajustar los parámetros de la cámara.

DSK6713_socket.lib

Esta librería contiene las funciones necesarias para inicializar y gestionar el módulo de comunicación inalámbrico de la placa de expansión.

Ejecución de las herramientas desarrolladas

Una vez cargado y ejecutado el programa en el DSP, se lanza el decodificador desde el PC y se establece la conexión. Una vez hecho esto, tenemos la posibilidad de solicitar un *stream* BMP o un

stream JPEG-2000 al DSP. He aquí unas capturas de pantalla del resultado de cada uno de estos procesos.

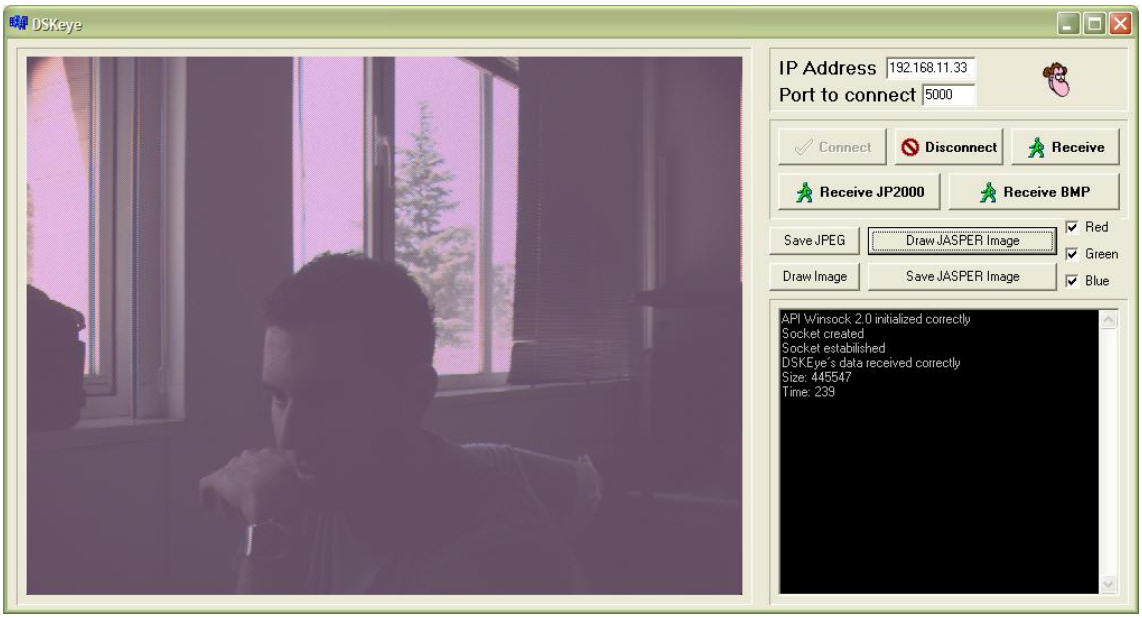


Figura 31: Recepción de una imagen JPEG

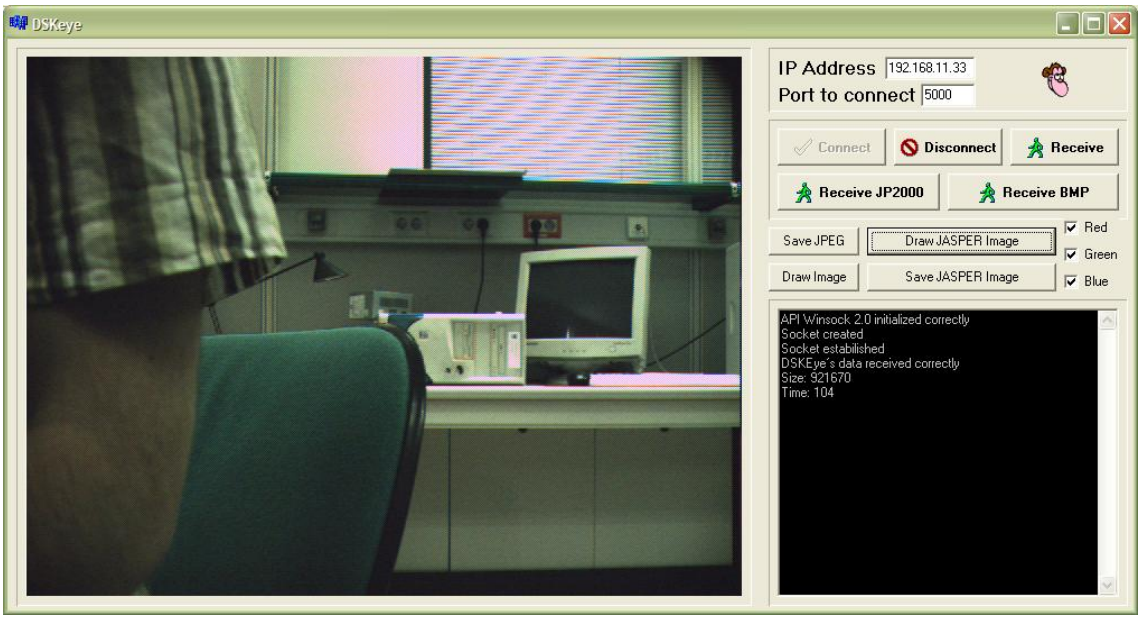


Figura 32: Recepción de una imagen BMP

Presentación de resultados

La calidad de las imágenes es muy buena, y con un ratio de compresión de entre el 70 y el 80% apenas se distingue la imagen de un BMP completo.

Imagen correspondiente a un ratio del 60% de compresión:

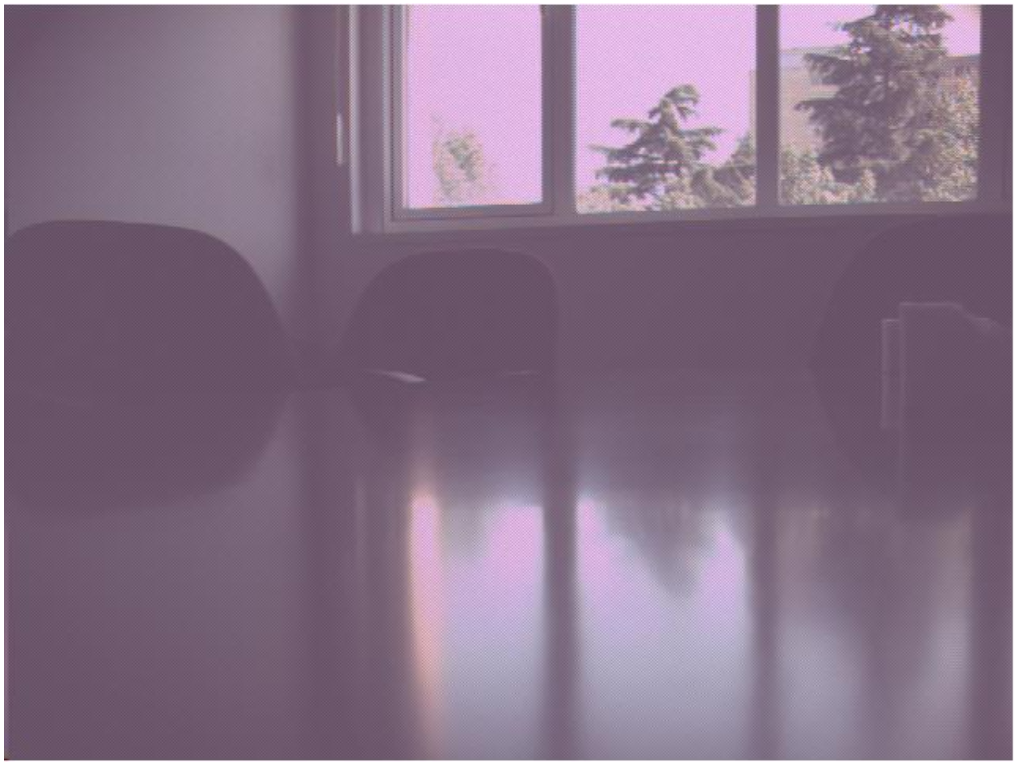


Imagen correspondiente a un 70% de compresión:

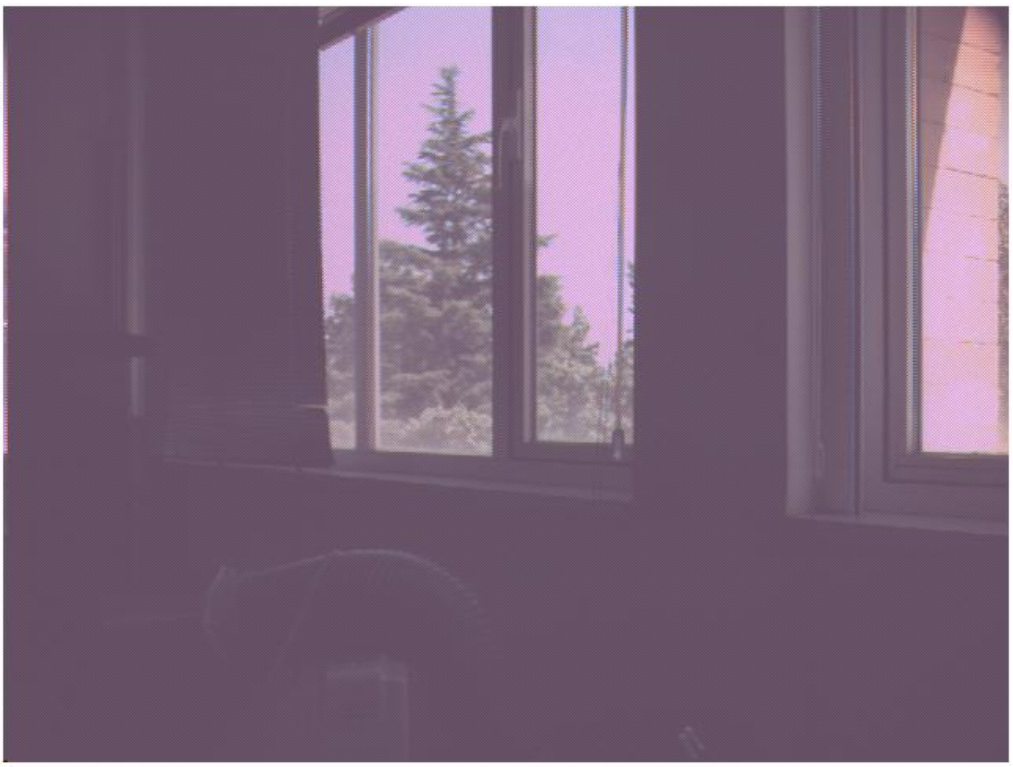
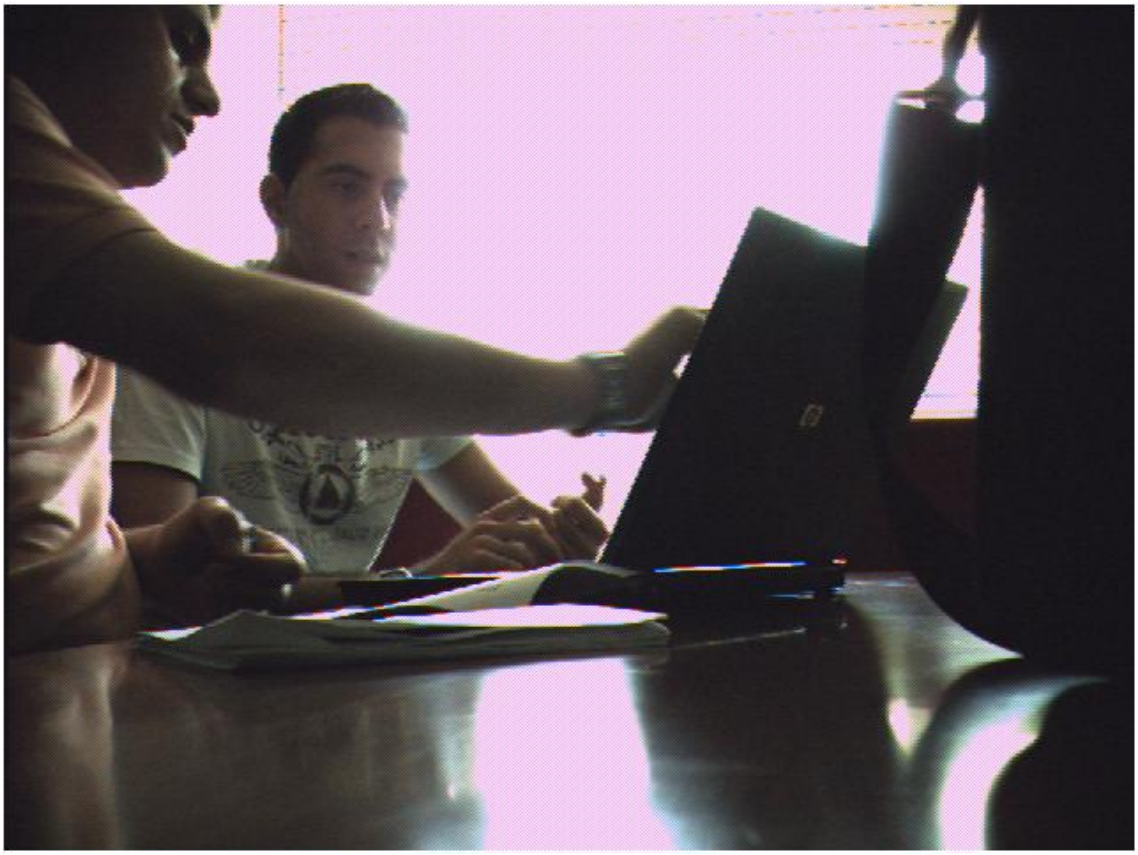
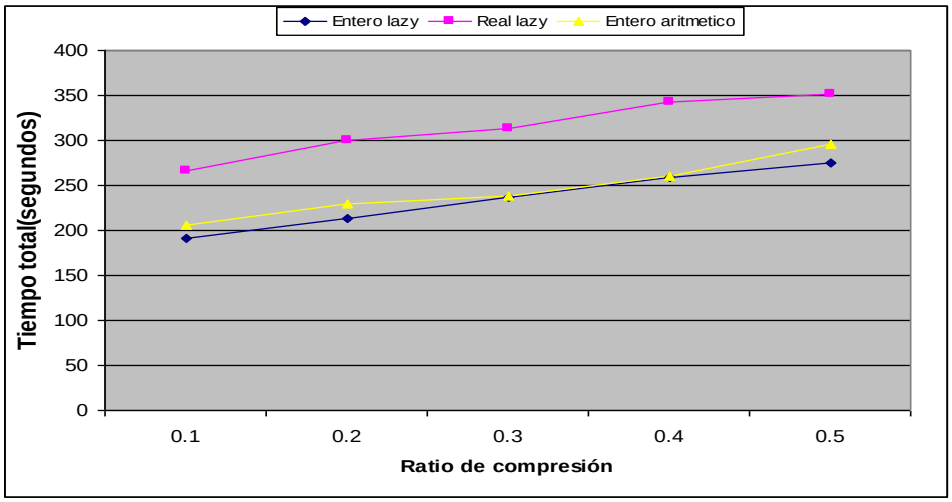


Imagen correspondiente a un 80% de compresión:



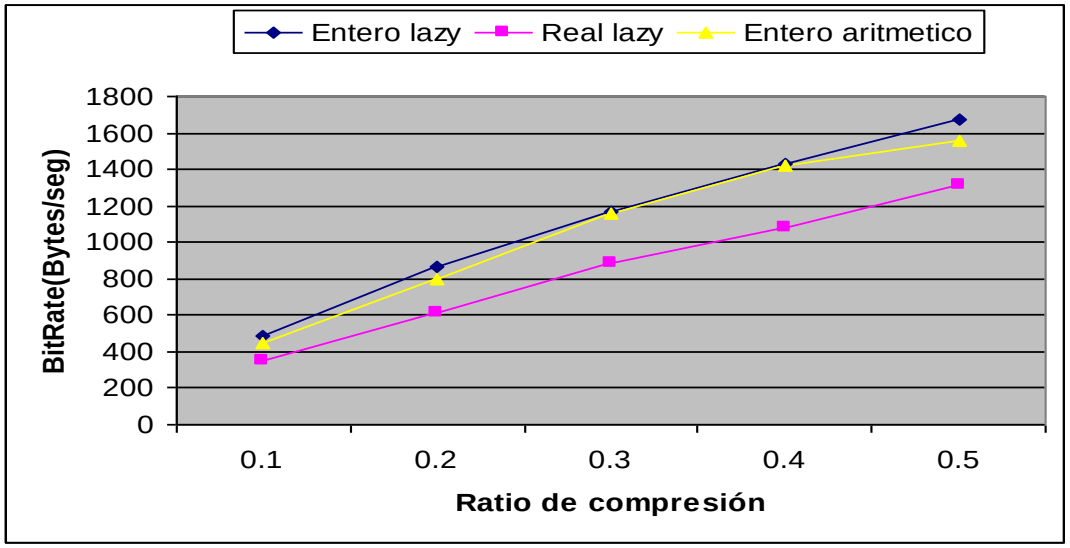
El tiempo de proceso y envío completo de un BMP con cabecera es de unos 13 segundos, salvo que ocurra algún problema con la conexión inalámbrica. El tiempo para un stream JPEG2000 varía dependiendo del ratio de compresión y el modo de codificación (entero o real), pero es significativamente más lento. Se tarda entre 15 y 20 veces más en crear y enviar un JPEG2000 que un BMP.

El tiempo total para la codificación JPEG en función del ratio de compresión evoluciona de acuerdo con la siguiente gráfica:



Como puede observarse, la combinación de opciones más rápida es la que combina el modo reversible de enteros a enteros con la codificación *lazy* que permite descartar más tramas que la codificación aritmética en las dos últimas etapas del códec (Tier-1 y Tier-2). El modo reversible de entero a real ofrece un peor rendimiento, siendo la diferencia entre ejecutar uno u otro que en el modo entero se salta la etapa de cuantificación, lo que explicaría el ahorro de tiempo de proceso. Utilizar el modo entero junto a la codificación aritmética proporciona un rendimiento similar en cuanto a tiempo total, pero con un ratio similar de compresión la calidad con éste último es superior. En realidad con codificación *lazy* las imágenes presentan una calidad bastante buena a partir de un ratio de compresión de 0'3, mientras que con codificación aritmética ésta se alcanza a partir de un ratio de 0'2.

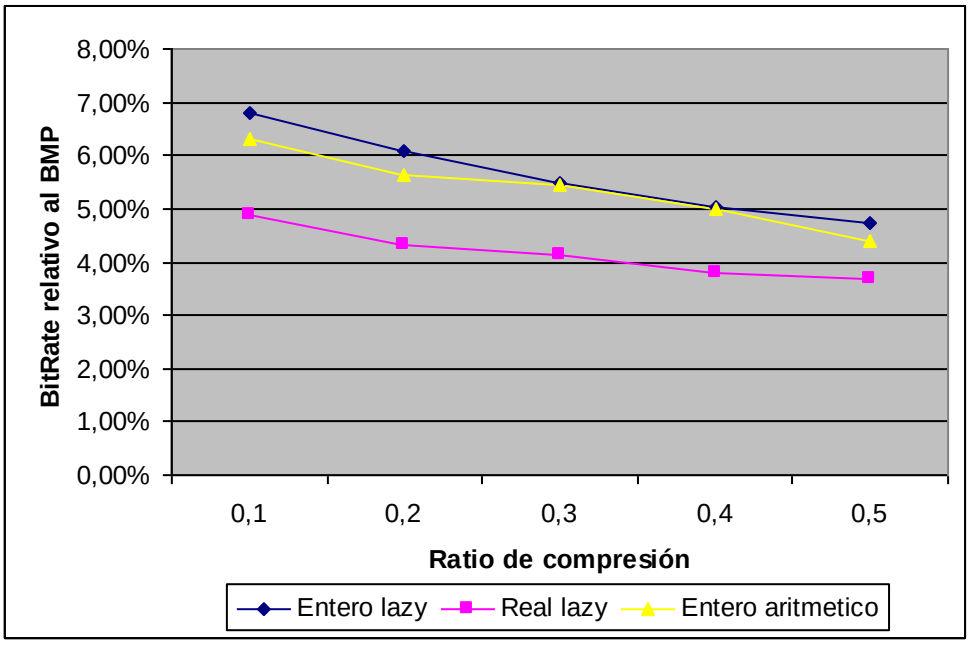
Si estudiamos la relación entre el tamaño de los datos que se envían y el tiempo que se tarda en generarlos se obtiene la siguiente gráfica:



En ella se aprecia que el BitRate, es decir, los bytes de datos que se envían por segundo, aumenta con el tamaño de los datos que se envían, de forma aproximadamente lineal. El BitRate en bytes por segundo para una imagen BMP de 921670 bytes enviada en 13 segundos es de aproximadamente 70897 Bytes/s. Si tomamos los datos del BitRate de JPEG2000 y los relacionamos con el BitRate base para un BMP mediante la fórmula:

$$\frac{TiempoBMP}{Tiempo \times Ratio} \times \frac{Tamaño}{TamañoBMP}$$

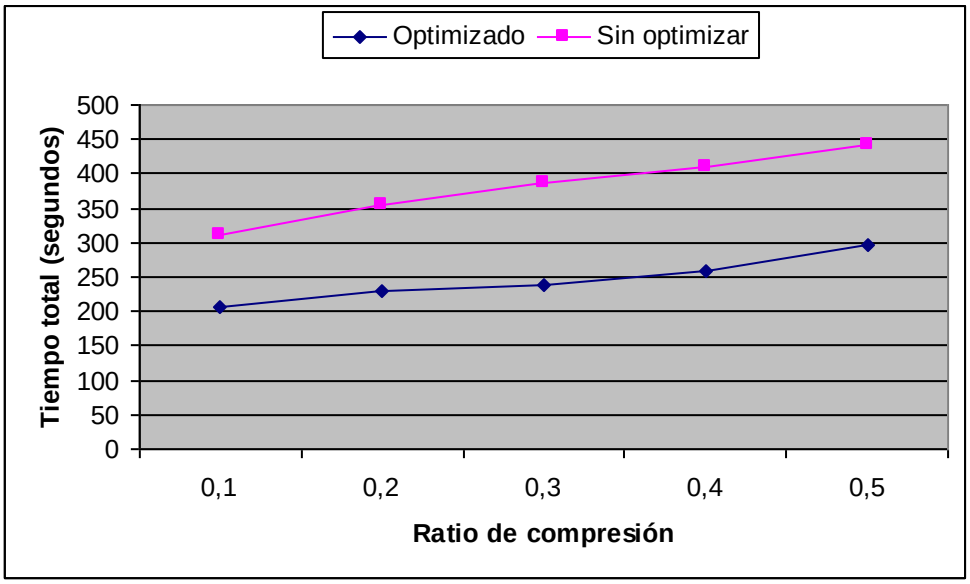
Obtenemos la relación entre el rendimiento del BMP frente al JPEG2000 para cada ratio de compresión, es decir, obtenemos el BitRate relativo entre los dos formatos.



En la gráfica se aprecia que cuánta mayor es la compresión que se aplica más próximos al BitRate del BMP nos encontramos. Desgraciadamente, con una compresión del 90% (un ratio de 0,1) apenas se alcanza un 6,8% del BitRate del BMP y la calidad de la imagen enviada se resiente bastante con una compresión tan elevada.

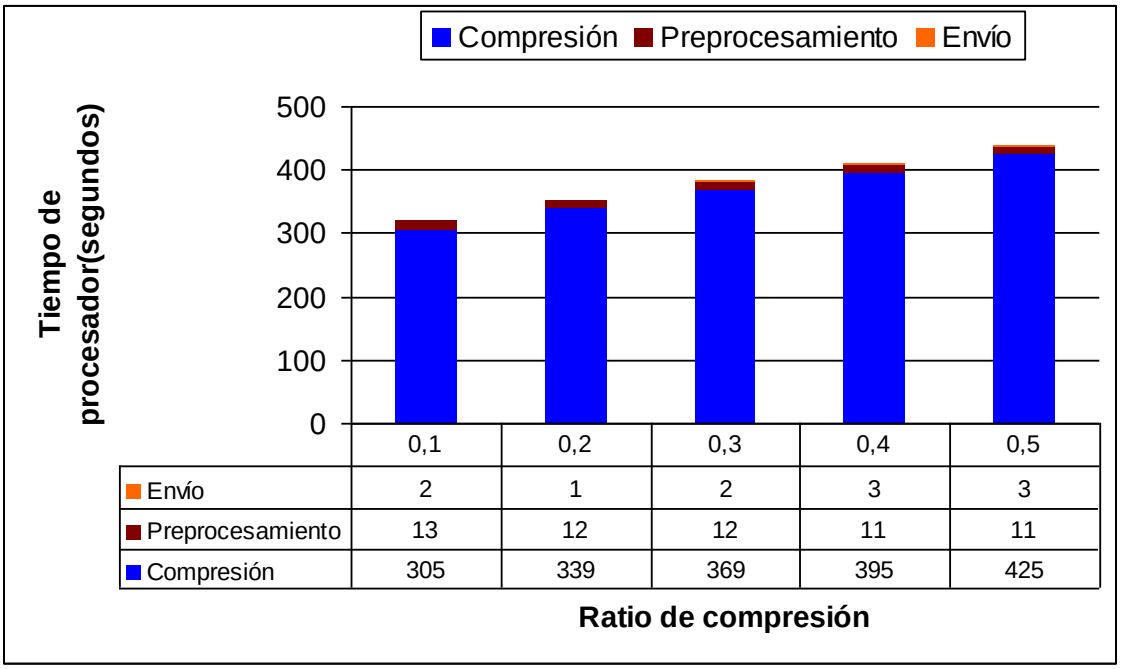
Rendimiento y puntos críticos

Para analizar la distribución de tiempo de procesador en las diferentes etapas de procesado de la imagen, dividimos el proceso en tres partes: preprocesamiento, que representa las operaciones anteriores a la primera llamada a la librería de codificación; compresión, que representa el proceso entero de creación del stream JPEG2000, y envío, que es el tiempo que se tarda en enviar la totalidad de los datos. Para medir el tiempo consumido por cada etapa, se hace uso de las funciones que permiten acceder al reloj del DSP, concretamente la función time(), y se calculan diferencias entre los puntos de medición situados antes y después de cada etapa. Además, se elimina toda reordenación de instrucciones u optimización que pueda realizar el compilador, ya que eso podría falsear los resultados, y se activa el modo de depuración del entorno de programación para poder obtener los datos necesarios. Esto hace que el proceso tenga lugar de forma más lenta. La comparación entre la versión optimizada por el compilador y la versión de depuración puede verse para distintos ratios en la siguiente gráfica:



La mejora introducida por la optimización automatizada está en torno al 35%de ahorro de tiempo.

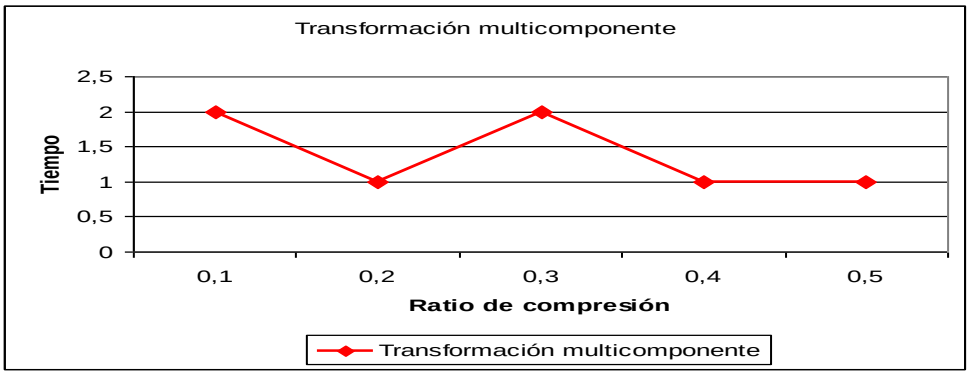
Para las distintas etapas mencionadas, la distribución de tiempos dependiendo del ratio de compresión aparece en la siguiente gráfica:



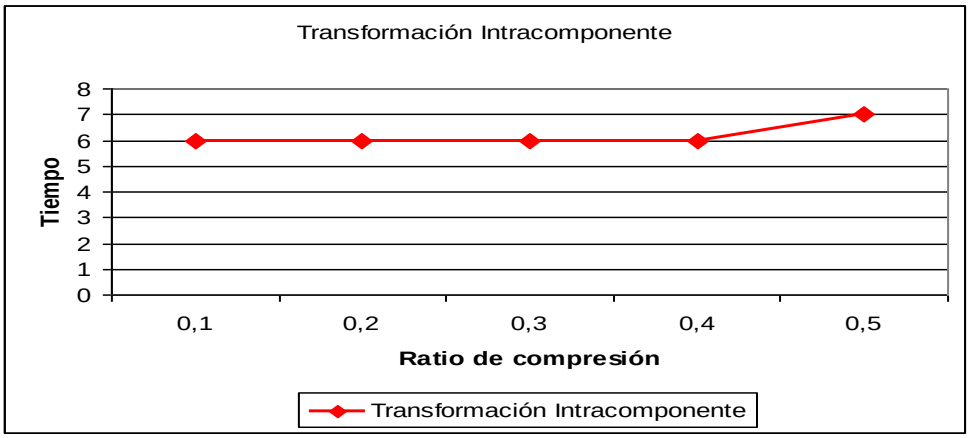
Como se puede ver, el grueso de los cálculos corresponde a la compresión, representando ésta más del 95% del tiempo de procesador empleado. El tiempo de envío es despreciable, y el tiempo de preprocesamiento es ligeramente superior pero tampoco es comparable al tiempo de compresión.

Puesto que el punto crítico parece ser el uso de la librería JASPER sobre la arquitectura DSP, centraremos el análisis en esta etapa, subdividiéndola de forma aproximada en cada una de las etapas de creación de un stream JPEG2000. Las etapas consideradas son: transformada multicomponente, transformada intracomponente, primera etapa de codificación, segunda etapa de codificación y

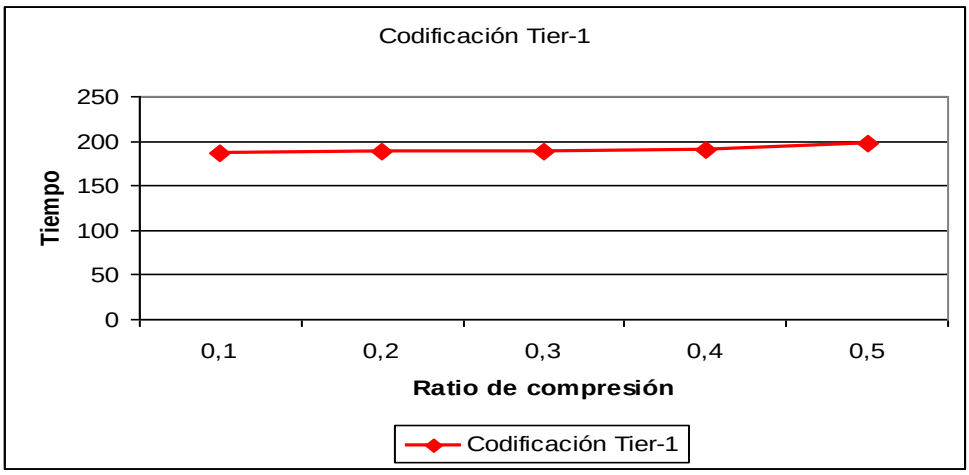
empaquetado de los datos. Aunque en la función de codificación de la librería la división en estás etapas no es explícita, se sitúan los temporizadores de forma aproximada, asegurando que las instrucciones que puedan aparecer entre dos etapas representen un mínimo porcentaje de tiempo que no afecte a los resultados. Los resultados de estas mediciones se recogen en las gráficas a continuación. La primera de ellas corresponde a la evolución del tiempo de proceso de la transformada multicomponente (la conversión del espacio de color RGB a YCrCb). Puede verse que ésta se realiza de forma eficiente, tardando entre 1 y 2 segundos sea cual sea el ratio de compresión, ya que no depende de éste.



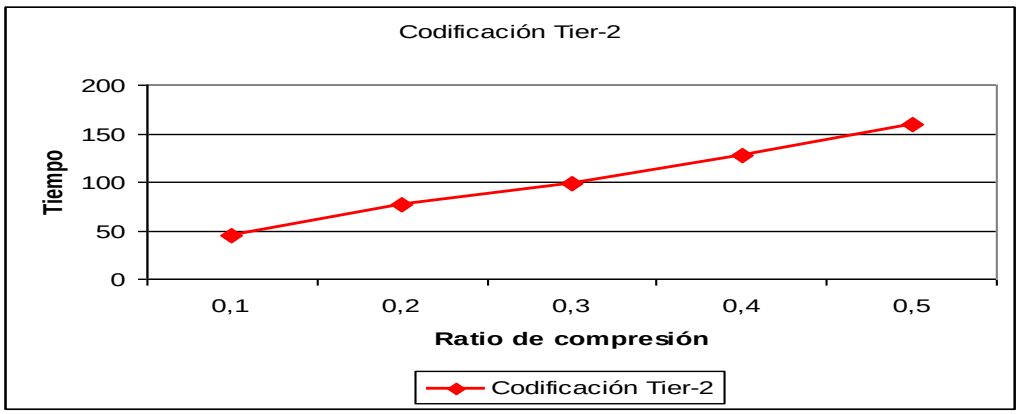
La transformación intracomponente tiene un comportamiento similar, el ratio de compresión no afecta a la ejecución de esta etapa de la conversión, y el tiempo medio de ejecución sin optimizar está en torno a los 6 segundos para 2 niveles de resolución.



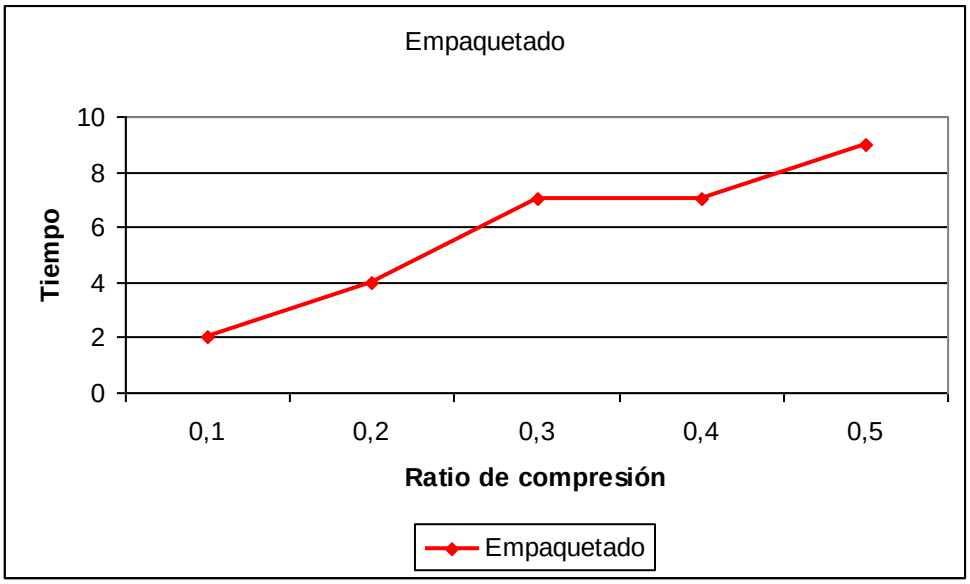
La primera etapa de codificación Tier-1 también se mantiene constante, ya que el mecanismo de control del ratio no actúa hasta que se inicia el codificador Tier-2.



Sin embargo, se aprecia que el tiempo consumido está en torno a 190 segundos sin optimizar, siendo ésta la función que consume la mayora parte del tiempo de proceso. La segunda etapa de codificación sí que se ve afectada por el ratio de compresión, siendo ésta la que afecta a la disminución de rendimiento conforme dicho ratio. Su evolución es la siguiente:

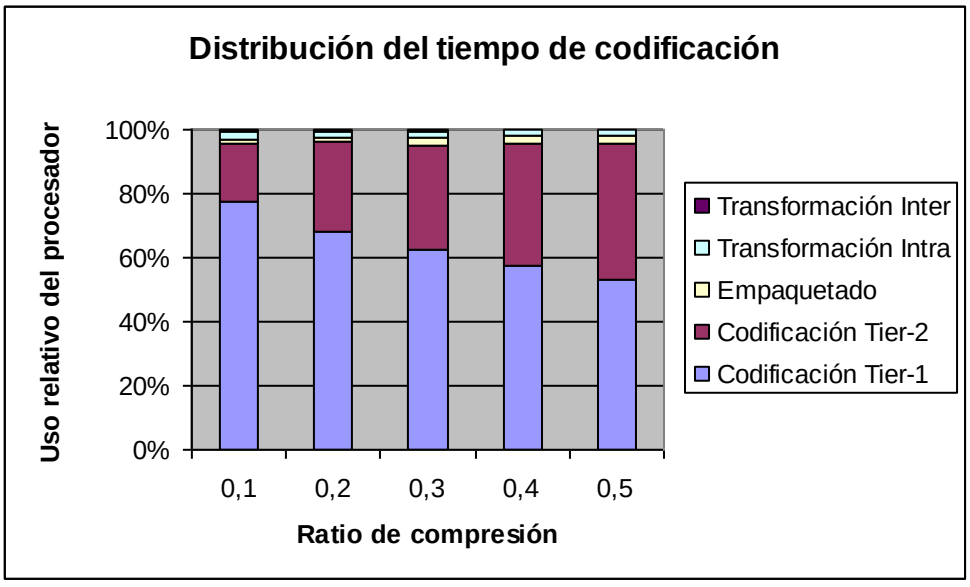


Aumenta de forma lineal con el ratio, al igual que lo hace el tiempo total, por lo que se deduce que ésta es la etapa responsable de que a mayor ratio de compresión sea necesario mayor tiempo de proceso. Por último, el tiempo de empaquetado también aumenta con el ratio, obviamente, a mayor tamaño de datos mayor número de paquetes, pero el tiempo consumido no es comparable a las etapas de codificación.



Cuellos de botella y posibles soluciones

La distribución del tiempo para distintos ratios entre las etapas antes mencionadas se muestra en la siguiente gráfica, considerando los tiempos de forma relativa:



Como puede verse, es la codificación Tier-1 la que supone el mayor gasto de tiempo, y, en segundo lugar, la codificación Tier-2.

El cuello de botella del envío del JPEG-2000 es claramente la codificación, debido a que exige recorrer varias veces cada fragmento de la imagen y la los numerosos bucles y bifurcaciones que aparecen. Dado que la arquitectura DSP no está orientada a optimizar el rendimiento en instrucciones de control si no a aplicar los mismos cálculos sobre grandes cantidades de datos en tiempo real, todo el flujo condicional de instrucciones durante las dos etapas de codificación y el hecho de que los bucles no se encuentran optimizados para aprovechar paralelismo en los fetch-packet ni para reducir las instrucciones de control necesarias en su ejecución, provocan que se degrade el rendimiento hasta

niveles que no son aceptables sobre todo en comparación con el envío de un BMP que supone mucha menor carga de proceso.

La solución a este problema pasa por la optimización de esas partes de la librería, introduciendo desenrollado de bucles y eliminación de bifurcaciones para conseguir un flujo de programa lo más uniforme posible, adaptado a la planificación agresiva de instrucciones que realiza la arquitectura VLIW del C6713. Además pueden estudiarse otras optimizaciones en la etapa de preprocesado, que permitirían saltarse algunos de los pasos que es necesario ejecutar actualmente. Esto mejoraría el rendimiento general, aunque las partes críticas son las etapas de codificación Tier-1 y Tier-2, como ya se ha señalado.

En cuanto al envío de BMP, la posible optimización recae sobre el envío mediante wifi. Si se adaptan los parámetros de envío a las características específicas de la red, es posible una cierta mejora del rendimiento, aunque no demasiado.

Mejoras o ideas principales a desarrollar en un futuro

Si se desea trabajar en JPEG-2000, queda pendiente la optimización del algoritmo utilizado y su adaptación a la arquitectura del C6713. En caso de explorar otras posibilidades, una posible mejora es añadir soporte para otros formatos que exijan un a carga de trabajo menor al DSP como la compresión RLE del mapa de bits, por ejemplo, o elegir formatos que cuenten con librerías optimizadas para este tipo de arquitecturas, como el formato JPEG, antecesor del JPEG-2000, que precisamente por llevar más tiempo como estándar y estar muy extendido su uso en la red y otros sistemas, pueden encontrarse numerosas implementaciones orientadas a rendimiento y algunas librerías que proporcionan soporte para sistemas como el utilizado en este proyecto.

El rendimiento obtenido con esta primera aproximación al JPEG-2000 sobre una arquitectura DSP es bastante pobre. Se obtienen resultados mucho mejores cuanto menos procesamiento se aplica a los datos de partida, y además no se pierde información de la imagen. El rendimiento no se ve afectado por el preprocesamiento de la imagen para crear un BMP válido, ya que esto apenas significa un porcentaje muy pequeño respecto al tiempo total de la codificación JPEG-2000. El punto crítico para conseguir que sea mejor enviar una imagen JPEG-2000 frente a una BMP está en conseguir que la diferencia de tiempo entre mandar una u otra, estimando que el envío de JPEG-2000 es entre un 50 y un 60% más rápido, sea mayor que el tiempo de codificación en JPEG-2000. Mientras no se consiga eso, será más eficiente enviar todos los datos al PC receptor y procesarlos en éste.

El cuello de botella es el análisis en subbandas, ya que se hace de forma recursiva y eso no es eficiente para la arquitectura del DSP. LA estructura de pila de llamadas de que dispone es muy limitada para mantener toda la información necesaria.

Conclusiones

Los resultados obtenidos con la implementación de JPEG-2000 utilizada han sido peores de lo que se esperaba. A pesar del correcto funcionamiento de la codificación, su rendimiento la hace inaplicable frente al proceso de un BMP para procesar imágenes estáticas sobre el DSP.

Aunque el DSP proporciona un alto rendimiento y una gran potencia de cálculo, su aplicación es útil en problemas específicos, que están adaptados a la arquitectura interna de su procesador. Esto no significa que no sea posible tratar imágenes mediante el C6713, sino que el algoritmo implementado en

esta ocasión no ha resultado ser el más idóneo. El hecho de que el mismo código ejecutándose en un PC tenga un rendimiento casi perfecto y sin embargo en el DSP no ofrezca buenos resultados pone de manifiesto las diferencias entre ambas arquitecturas, y la necesidad de estudiar el problema desde el punto de vista del flujo de datos sobre el DSP, para adaptar la forma de abordarlo de manera que se aproveche toda la potencia de cálculo que ofrece la plataforma.

Sin embargo, teniendo en cuenta que la comunicación inalámbrica tiene un rendimiento limitado, creemos que la idea de reducir el tamaño de los datos para ahorrar tiempo de comunicación es una opción viable en el futuro, que puede proporcionar un rendimiento óptimo siempre que se empleen de forma exhaustiva los recursos del dispositivo, y se consiga convertir la codificación de la imagen en un problema de proceso de una señal que se adapte a la filosofía de trabajo del DSP C6713 y saque partido de su potencia de cálculo.

La solución pasa por renunciar a analizar parte de la redundancia de información de una imagen, favoreciendo que ésta se pueda transformar en tiempo real, sin necesidad de volver atrás sobre cálculos realizados, alterando el flujo de información que sería natural en un DSP.

Procesamiento y tratamiento de imágenes en movimiento bajo el estándar UIT H.263

¿De qué partimos?

Existen multitud de algoritmos y estándares de compresión de video que actualmente están implementados en una amplia gama de componentes Hardware. Entre ellos existen diferencias muy importantes que erróneamente pueden llevarnos a preferir un sistema u otro, pero que en realidad lo único que permiten discriminar son principalmente dos factores:

- Arquitecturas objetivo: Dependiendo de las capacidades de cada dispositivo y de la tecnología actual de desarrollo, será posible utilizar uno o varios algoritmos de compresión, más o menos avanzados, que únicamente serán rentables si proporcionan el resultado esperado bajo unos costes ajustados. Esto no ocurre actualmente en la realidad, aunque conforme pasa el tiempo la tecnología va avanzando y cada año salen a la luz nuevos dispositivos y arquitecturas especializadas que permiten aprovechar, con mayor o menor eficiencia, todos y cada uno de los beneficios en rendimiento que ofrecen dichos algoritmos. Éstos factores de rendimiento pueden agruparse principalmente en dos clases:
    - Calidad de imagen: Factor clave que depende sobre todo de los soportes electrónicos que muestran la imagen (pantallas TFT-LCD, plasma, etc) y el tratamiento numérico que los algoritmos hacen de ella. Por supuesto, para poder realizar tareas de compresión pueden ser necesarios o no procesadores potentes que aporten una gran capacidad de cálculo, todo dependerá del dispositivo que vaya a comercializarse y sobre todo de su diseño y arquitectura.
    - Continuidad entre tramas: Cuando se habla de procesamiento de imagen en movimiento en tiempo real directamente una persona lo relaciona erróneamente con video en tiempo real y diferencia entre una o varias muestras quedándose con la que mayor sensación de realismo ofrece. Según estudios médicos, el ojo humano no es capaz de discriminar más de 30 imágenes por segundo y ése es el objetivo de todos los algoritmos de compresión de video, llegar a alcanzar los 30 *frames* por segundo (de ahora en adelante fps). Lógicamente, todo esto depende directamente de la capacidad de cómputo y cálculo del dispositivo físico con el que se va a trabajar, siendo imposible en muchos casos llegar a dicha cantidad bien sea por limitaciones derivadas del propio hardware o por recortes de recursos en un determinado proyecto.

- Objetivo perseguido por los desarrolladores: Debido a la cantidad de áreas de negocio existentes y sobre todo a la cantidad de componentes hardware diseñados exclusivamente para estas tareas, existen multitud de objetivos "meta" que tienen que ser alcanzados y que determinan de forma directa qué resultados han de alcanzarse para poder determinar que un proyecto ha cumplido las expectativas. Por ejemplo, no se utilizarán algoritmos de compresión de video que obtienen muy buenas tasas de continuidad a costa de perder calidad de imagen si lo que se desea es obtener imágenes con una gran resolución para poder trabajar con sectores o partes de dichas imágenes (por ejemplo, las cámaras de video vigilancia de grandes centros comerciales o sucursales bancarias).

Debido precisamente a esto, es muy difícil decidir cuáles algoritmos son mejores que otros, pero si discernir entre cuáles aprovechan más eficientemente los recursos del sistema objetivo y consiguen sacar un rendimiento óptimo.

Para poder situarnos de forma más precisa, se detallan brevemente algunos de los más importantes algoritmos de compresión de imagen en movimiento, destacados cada uno en una determinada área funcional:

MPEG

En el año de 1990, la ISO, preocupada por la necesidad de almacenar y reproducir imágenes de video digitales y su sonido estereofónico correspondiente, creó un grupo de expertos que llamó MPEG (Moving Pictures Expert Group) procedentes de aquellas áreas implicadas en el problema (telecomunicaciones, informática, electrónica, radio difusión, etc).

El primer trabajo de este grupo se conoció como la norma ISO/IEC 11172, mucho más conocida como MPEG-1, en el año 1992. La idea inicial era la de permitir el almacenamiento y reproducción en soporte CD-ROM con un flujo de transmisión de datos del orden de 1,5 Mbps, transportando tanto imagen como sonido.

El estándar MPEG además de aprovechar la redundancia espacial intrínseca de una imagen fija utilizada en la codificación JPEG, aprovecha la redundancia temporal que aparece en la codificación de imágenes animadas, permitiendo encontrar similitudes entre las imágenes sucesivas de video.

Debido a que la calidad en la compresión de video en el estándar MPEG-1 era de baja calidad y no servía para otras aplicaciones, se creó la norma ISO/IEC 13818, mucho más conocida con el nombre de MPEG-2. Esta norma permite un flujo de transmisión hasta el orden de los 20 Mbps, transportando tanto imagen como sonido. Norma que se utilizaría en la televisión de alta definición.

La norma MPEG-4 fue originalmente desarrollada para video conferencia, distribución de video por internet y aplicaciones similares de bajo ancho de banda, aunque será el formato estrella para la trasmisión de video de alta definición ya que es el estándar escogido por la mayoría de fabricantes de discos laser de lente azul más conocidos como "Blue laser optical discs" o Blue-Ray Discs.

Motion JPEG

Motion JPEG *Joint Photographic Experts Group* es una extensión del estándar de UIT/ISO JPEG para imágenes sin movimiento. El Motion JPEG es un método de compresión simétrico y típicamente consigue niveles de compresión de 10:1 hasta 50:1. Como es una extensión del estándar JPEG para imágenes sin movimiento, el Motion JPEG sólo elimina redundancia dentro de una imagen (redundancia espacial) y no la redundancia inter-imagen (temporal).

El resultado es una compresión significativamente menor a la que efectuaría un método de compresión que eliminase los dos tipos de redundancia. No obstante, la falta de codificación inter-imagen puede ser positiva para algunas aplicaciones de vídeo. Por ejemplo, si se desea tener acceso a una imagen de vídeo aleatoria, Motion JPEG permite un acceso mucho más rápido y eficiente que el MPEG puesto que no es necesario esperar un ciclo completo de secuencia para decodificar un frame específico (las imágenes no dependen unas de otras).

H.261

Es el primer estándar preparado para la codificación y transmisión de secuencias de vídeo digital en redes de telecomunicación elaborado por la UIT (Unión Internacional de Telecomunicaciones)

en 1990 (versión modificada en 1993). Esta recomendación está orientada, principalmente, a videotelefonía e integrada en un conjunto de estándares descritos en la recomendación H.320 que permiten su uso en redes RDSI junto con la transmisión de sonidos y datos, enlaces multipunto, etc.

La recomendación H.261 está concebida para transmitir vídeo codificado a velocidades de 8 a 250 KBps. La compresión se basa en la utilización de la transformada DCT para disminuir la redundancia espacial de las imágenes y en el uso de varias de las técnicas descritas anteriormente. Permite obtener calidades próximas a las del formato de vídeo VHS utilizando velocidades de transmisión de 250 KBps, pero no presenta un comportamiento de calidad con todo tipo de imágenes.

Aspectos interesantes de este estándar son su perfecta adaptación a las redes RDSI y la existencia numerosos productos que lo implementan (videoteléfonos, sistemas de videoconferencia, kits para PC que permiten dotarlos de capacidad de videotelefonía o de acceso a bases de datos, etc.). Esto hace posible realizar bases de datos multimedia accesibles a través de la RDSI con productos comerciales ya disponibles.

Como siempre ocurre en cualquier proyecto, primero es necesario plantear unas bases teóricas que fundamenten todo el peso de algunas de las técnicas empleadas en el desarrollo de cualquier aplicación en una determinada arquitectura objetivo. En nuestro caso, vamos a introducir brevemente determinados conceptos básicos en referencia al proceso de codificación de video y las técnicas relacionadas con él.

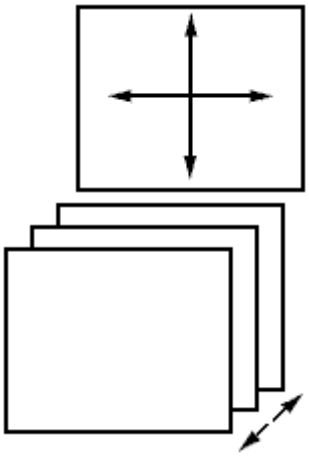
Compresión de video

La compresión de video surge de la necesidad de transmitir imágenes a través de un canal que contenga un ancho de banda aceptable. A continuación se examinarán cuales son los métodos más utilizados que permiten obtener este resultado, y las diferentes normas que se utilizan hoy día.

Estos métodos de compresión, recurren a los procedimientos generales de compresión de datos, aprovechando además la redundancia espacial de una imagen (áreas uniformes), la correlación entre puntos cercanos y la menor sensibilidad del ojo a los detalles finos de las imágenes fijas (JPEG) y, para imágenes animadas (MPEG), se saca provecho también de la redundancia temporal entre imágenes sucesivas.

La figura muestra que cuando las imágenes individuales son comprimidas sin referencia a las demás, el eje del tiempo no entra en el proceso de compresión, esto por lo tanto se denomina codificación intra (intra=dentro) o codificación espacial. A medida que la codificación espacial trata cada imagen independientemente, esta puede emplear ciertas técnicas de compresión desarrolladas para las imágenes fijas. El estándar de compresión ISO (International Standards Organization) JPEG, está en esta categoría.

Se pueden obtener grandes factores de compresión teniendo en cuenta la redundancia entre imágenes sucesivas. Esto involucra al eje del tiempo, por lo que este proceso se denomina codificación inter (inter=entre) o codificación temporal (la siguiente figura muestra esto).



Codificación intra o espacial, explora la redundancia dentro de la imagen

Codificación inter o temporal, explora la redundancia entre imágenes

La codificación temporal permite altos factores de compresión, pero con la desventaja de que una imagen individual existe en términos de la diferencia entre imágenes previas. Si una imagen previa es quitada en la edición, entonces los datos de diferencia pueden ser insuficientes para recrear la siguiente imagen.

Codificación intra o espacial

Un análisis de las imágenes de televisión revela que existe un alto contenido de frecuencias espaciales debido al detalle en algunas áreas de la imagen, generando una cantidad pequeña de energía en tales frecuencias. A menudo las imágenes contienen considerables áreas en donde existen pixeles con un mismo valor espacial. Además, como el promedio de brillo de la imagen se caracteriza por componentes de frecuencia de valor cero, siempre será una buena opción trabajar con histogramas y componentes en frecuencia de imágenes, ya que tendremos unas estructuras más manejables dado que la mayoría de transformaciones y algoritmos de codificación utilizan dicho desarrollo. Así que, si lo que deseamos es deshacernos de gran parte de información irrelevante presente en una imagen, simplemente deberemos omitir los componentes de alta frecuencia de la misma.

Una disminución en la codificación se puede obtener, tomando como ventaja que la amplitud de los componentes espaciales disminuye con la frecuencia. Si el espectro de frecuencia espacial es dividido en subbandas de frecuencia, las bandas de alta frecuencia se pueden describir en pocos bits, no solamente porque sus amplitudes son pequeñas sino porque puede ser tolerado más ruido.

Codificación inter o temporal

La codificación inter aprovecha la ventaja que existe cuando las imágenes sucesivas son similares. En lugar de enviar la información de cada imagen por separado, el codificador inter envía la diferencia existente entre la imagen previa y la actual en forma de codificación diferencial. La figura siguiente muestra este principio. El codificador necesita de una imagen referencia, la cual fue almacenada con anterioridad para luego ser comparada entre imágenes sucesivas y de forma similar se requiere de una imagen previamente almacenada para que el decodificador desarrolle las imágenes siguientes.

Los datos que se generan al hacer la diferencia entre dos imágenes, también se pueden tratar como una nueva imagen, la cual se debe someter al mismo tratamiento de transformadas utilizado en la compresión espacial.

Un sistema básico de codificación inter se muestra en la figura siguiente. Desafortunadamente existe la posibilidad de transmitir errores, si se utiliza una secuencia ilimitada de imágenes previstas. Por esto es mejor utilizar un número limitado de imágenes previstas para de este modo garantizar una menor acumulación de errores.

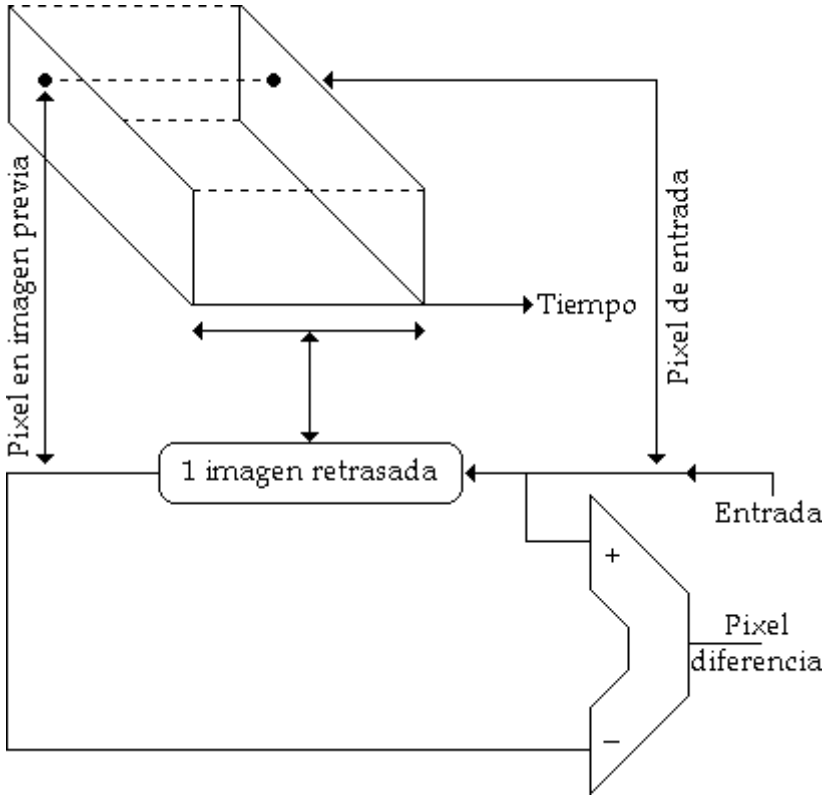
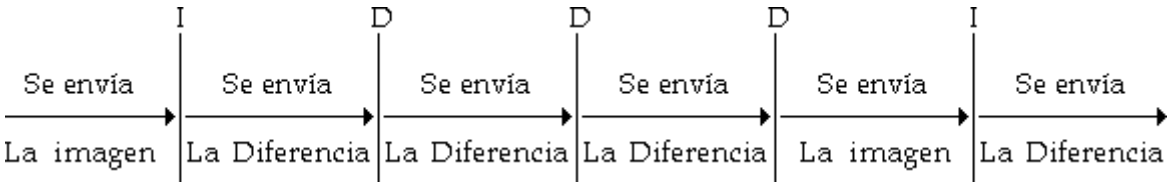


Figura 44: Sistema de codificación inter

La próxima figura muestra el recorrido de una imagen original, llamada imagen I o intra, la cual es enviada entre imágenes que han sido creadas usando una diferencia entre imágenes, llamada imágenes P o previstas (en el diagrama siguiente aparecen tramas D que simbolizan “diferencias” con respecto a la imagen de referencia I).

La imagen I requiere grandes cantidades de información, mientras que las imágenes P requieren una cantidad menor. Esto ocasiona que el flujo de transmisión de datos sea variable hasta cuando llegan a la memoria intermedia, la cual genera a su salida una transmisión de datos de forma constante. También se puede observar que el predictor necesita almacenar datos de menor proporción puesto que su factor de compresión no cambia de una imagen a otra.



I = Imagen codificada intra.
D = Imagen codificada diferencialmente.

Una secuencia de imágenes constituida por una imagen I y las siguientes imágenes P hasta el comienzo de otra imagen I, se denomina grupo de imágenes GOP (Group Of Pictures). Para factores de compresión altos se utiliza un número grande de imágenes P, haciendo que las GOPs aumenten de tamaño considerablemente; sin embargo un GOP grande evita recuperar eficazmente una transmisión que ha llegado con errores.

La estructura sintáctica que forma un GOP puede resumirse en la siguiente figura:

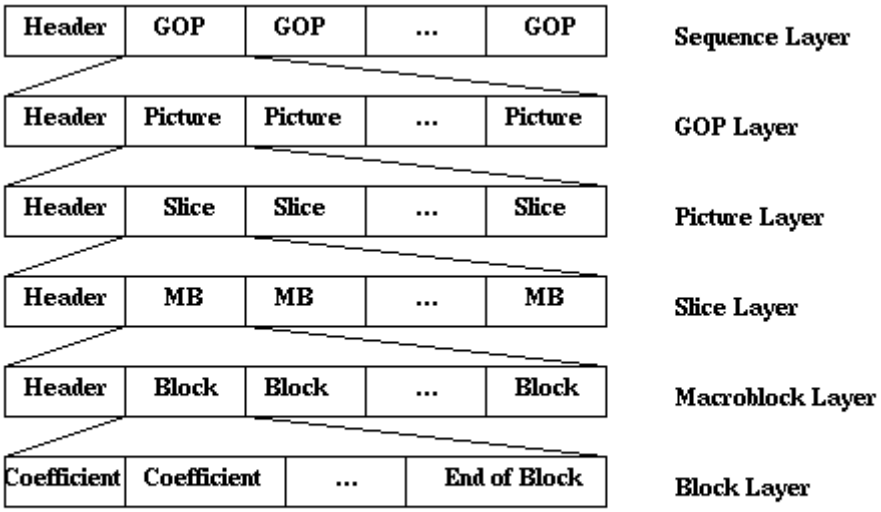


Figure 2: Components of an MPEG GOP

Figura 45: Estructura de un GOP

Como podemos ver, un *GOP* engloba varias *imágenes* (pictures) y éstas a su vez están formadas por *rodajas* (slices). Cada *rodaja* equivale a una pseudocapa agrupada en *macrobloques* de tamaño constante (MB), formados por *bloques* (Block) que poseen los *coeficientes* que llevan la información de cada zona de una rodaja. Cada *subcapa* tiene una *cabecera* que sirve para localizar de forma absoluta cada *subsegmento* de la imagen con respecto a la imagen general.

Codificación bidireccional

Cuando un objeto se mueve, este oculta lo que hay detrás de él, pero esto va cambiando a medida que se va moviendo, permitiendo observar el fondo. El revelado del fondo exige nuevos datos a ser transmitidos, ya que el área del fondo había sido ocultada anteriormente y la información no pudo ser obtenida desde una imagen previa.

Un problema similar ocurre si se hace una toma panorámica con una cámara de video ya que aparecen nuevas áreas ante el observador y nada se sabe acerca de ellas. El proceso de codificación bidireccional ayuda a minimizar este problema ya que deja información para ser tomada de imágenes anteriores y posteriores a la imagen observada. Si el fondo ya ha sido revelado, y este será presentado en una imagen posterior, la información puede ser movida hacia atrás en el tiempo, creando parte de la imagen con anticipación.

La figura muestra en qué se basa la codificación bidireccional. En el centro del diagrama un objeto se mueve revelando su fondo, pero éste no se conoce hasta la siguiente imagen. Entonces se toman los datos de las imágenes anteriores y posteriores, o incluso se utiliza el promedio de los datos, descubriendo de esta forma el fondo. En la figura se muestra una codificación bidireccional. Primero se toma una imagen I y, con la ayuda de una imagen P se pueden obtener imágenes B, las cuales son llamadas también imágenes bidireccionales.

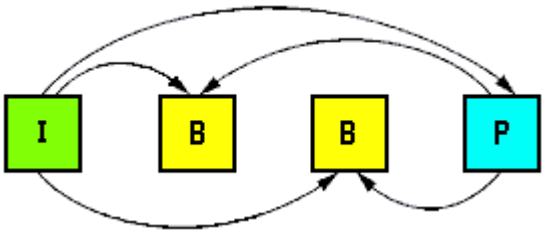


Figura 46: Codificación bidireccional

Las **imágenes I** se codifican como si fuesen imágenes fijas utilizando la norma JPEG, por tanto, para decodificar una imagen de este tipo no hacen falta otras imágenes de la secuencia, sino sólo ella misma. No se considera la redundancia temporal (compresión *intraframe*). Se consigue una moderada compresión explotando únicamente la redundancia espacial. Una imagen I siempre es un punto de acceso en el flujo de bits de vídeo. Son las imágenes más grandes.

Las **imágenes P**: Están codificadas como predicción de de la imagen I ó P anterior usando un mecanismo de compensación de movimiento. Para decodificar una imagen de este tipo se necesita, además de ella misma, la I ó P anterior. El proceso de codificación aquí explota tanto la redundancia espacial como la temporal.

Las **imágenes B**: Se codifican utilizando la I ó P anterior y la I ó P siguiente como referencia de compensación y estimación de movimiento. Para decodificarlas hacen falta, además de ellas mismas, la I ó P anterior y la I ó P siguiente. Estas imágenes consiguen los niveles de compresión más elevados y por tanto son las más pequeñas.

¿Cuál es el objetivo?

En un principio, nos planteamos conseguir una codificación de video que permitiese obtener una relación calidad/rendimiento óptima, que por un lado facilitase el envío y recepción de video en tiempo real y por el otro pudiese analizar imágenes en secuencia para poder localizar la posición y orientación en un espacio limitado de una serie de barcos controlados automáticamente. Ante estos requisitos se decidió optar por un algoritmo de compresión de video basado en el estándar UIT H.261 pero implementando algunas de las mejoras que finalmente se verían introducidas en el estándar de UIT que posteriormente difundió con el apelativo de H.263.

Como dispositivos base disponíamos únicamente del DSP de Texas Instruments TMSS320C6713, que junto con la cámara DSKEye y el módulo emisor Wifi especialmente diseñados para este tipo de dispositivos constituían nuestra única herramienta de trabajo. Sobre él se ejecuta absolutamente todo el proceso de codificación y envío de imagen, aspectos que a la postre nos plantearían serios problemas que afectarían al rendimiento global de todo el conjunto. Como puntos en contra es inevitable tener en cuenta la nula experiencia de trabajo que teníamos de partida con arquitecturas hardware avanzadas

basadas en DSP´s, por lo que durante más o menos 3 meses, desde Octubre a Diciembre, tuvimos que ir aprendiendo a sacar partido a este dispositivo con ayuda de tutoriales de muy diversa índole, desde creación de librerías de trabajo hasta aplicaciones de manipulación de sonido, que nos ayudaron a familiarizarnos con este tipo de arquitecturas.

Como es sabido, el DSP de Texas Instruments viene acompañado de un completo entorno de desarrollo denominado Code Composer Studio en su versión 3.1. Este programa permite construir aplicaciones específicas para la familia de procesadores C6000, en concreto en nuestro caso el C6713. Además, como todo proceso de codificación necesita un decodificador, éste último ha sido construido utilizando la herramienta de desarrollo de aplicaciones multilenguaje Microsoft Visual Studio 2005 en su edición para centros de enseñanza y universidades.

Pero como todo, sin herramientas matemáticas un algoritmo no es un algoritmo y por ello hemos necesitado utilizar algunas transformaciones y secuencias de filtrado que hoy en día se utilizan en infinidad de campos y para multitud de áreas de negocio. Y es que en realidad una imagen puede verse de forma directa como una codificación de caracteres numéricos en un sistema binario, es decir, vectores de ceros y unos. Su posterior interpretación y manipulación entran dentro de las particularidades propias de cada algoritmo. Todas y cada una de las técnicas descritas a continuación han sido utilizadas para este proyecto.

Transformada Directa del Coseno: También conocida como DCT. Es un tipo de compresión basada en transformaciones matemáticas que básicamente prepara una serie de estructuras de datos para que posteriormente puedan ser eliminadas las altas frecuencias del histograma de una imagen. De ésta forma se pierden datos de una imagen pero se consiguen transmitir un menor número de información que puede descartarse, liberándose de esta forma el ancho de banda teórico del canal.

Codificación Huffman: Es un tipo de codificación sin pérdida de información que se basa en la asignación de códigos de longitud variable a los valores de brillo de una imagen, de tal forma que los valores más comunes presentes en una misma imagen tendrán asignados códigos más cortos, de tal forma que se disminuya también el tráfico de información que circula por el canal. Se dice que es un algoritmo sin pérdida porque a todos y cada uno de los valores de brillo de una imagen tienen asignado un código Huffman.

Codificación basada en vectores de cuantificación: En ella se descartan aquellos valores de una imagen que no sobrepasen cierto umbral máximo fijado por una matriz de cuantificación. Dicha matriz tiene distribuidos una serie de coeficientes en zigzag que formarán una máscara que recorrerá todos y cada uno de los frames. La diferencia entre los coeficientes de la matriz y los valores para cada píxel de la imagen determinará si se elimina dicha información o no. Cuantos más píxels queden fuera, mayor será el factor de compresión, simplemente es necesario cambiar la matriz de cuantificación.

Compensación de movimiento: Esta técnica tiene como objetivo principal eliminar la redundancia temporal entre las imágenes que componen una secuencia con el fin de aumentar la compresión. Para eliminar dicha redundancia, la idea inicial es transmitir la diferencia entre un píxel en una posición de un fotograma (imagen) y el píxel situado en la misma posición pero en el fotograma siguiente. Esto sirve cuando las imágenes son estáticas. Pero lo normal es tener imágenes dinámicas y por tanto no podemos implementar lo anterior tal cual, sino que previamente habrá que estimar el movimiento que ha sufrido un píxel de un objeto de un fotograma al siguiente. Habrá que calcular el vector de movimiento asociado a cada píxel de la imagen. Al decodificador se transmite la diferencia y los vectores de movimiento calculados. Si los vectores están bien calculados la diferencia entre una imagen y la siguiente compensada debe ser muy pequeña, ya que la escena no cambia bruscamente en un corto intervalo de tiempo. En cambio, si la escena posee cambios muy bruscos en intervalos de

tiempo muy pequeños, la utilización de esta técnica no aporta grandes beneficios y sí un elevado coste de cómputo predictivo.

Representación del color: Según la primera ley de Grassmann toda sensación de color puede obtenerse por suma de tres fuentes de color denominadas primarias. Estas fuentes de color son el rojo, el verde y el azul, que componen el conocido sistema de representación RGB. Uno de los conceptos importantes introducidos que permitió el avance en los sistemas de vídeo fue separar la luminancia de la crominancia. Inicialmente el sistema NTSC, desarrollado sobre todo en Norteamérica definió la transmisión de las señales en un formato de luminancia y crominanacia (antes se utilizaba un formato que utilizaba las tres componentes de color). El nuevo espacio de color se denominó YIQ, donde las letras representan la luminancia, la componente en fase de la crominancia y la componente en cuadratura de la croma, respectivamente. Posteriormente el PAL y el SECAM, europeos, optaron por un espacio de colores idéntico pero con una rotación de 33 grados. Es el denominado espacio YUV. El equivalente digital de YUV es el YCbCr, donde Cb es la componente de crominancia que corresponde con la componente U y la Cr es análoga a la V. El formato YCbCr, concentra la mayor parte de la información de la imagen en la luminancia y menos en la crominancia. El resultado es que los elementos de YCbCr están menos correlacionados y pueden ser codificados por separado. Otra ventaja que se consigue es la reducción de la velocidad de transmisión de las componentes de crominancia.

Estructura del subproyecto y diagramas de bloques

Debido principalmente al entorno de desarrollo que acompaña al DSP y a la arquitectura preparada para cómputo en punto flotante de su procesador, es posible implementar cualquier tipo de programa secuencial apoyado en los anteriores algoritmos matemáticos y teóricos. De ésta forma y salvaguardando las especificaciones técnicas del sistema de memoria del DSP, es relativamente sencillo implementar un algoritmo de codificación de video basado en el estándar UIT H.263.

Es cierto que el código que se genere debe cumplir determinadas características muy particulares y que en muchos sectores del programa ha sido necesario reconvertir el codificador estándar creado para PC, debido a que utilizaba excesivas estructuras de control y saltos condicionales que ralentizaban la velocidad de procesamiento del DSP, pero esto es sólo uno de los principales problemas que cualquier desarrollador de Software para dispositivos empotrados debe salvaguardar en cualquier ocasión. No obstante, inicialmente se plantearon dos metas importantes que debían alcanzarse:

Construir un codificador de tratamiento de imagen UIT-H263: Construir un codificador H.263 que facilitase la manipulación y el trabajo con imágenes en tiempo real, por ejemplo, para poder detectar la posición y orientación de una flota de barcos distribuida aleatoriamente sobre un canal. Dicho programa debería de tratar imágenes en movimiento con una alta resolución, para así poder asegurar una precisión más o menos acertada. Para ello, fue necesario prescindir de la codificación temporal únicamente, de tal forma que se construyese un codificador intra, con la consecuente pérdida de rendimiento en lo que tasa de frames por segundo se refiere. Además, factores como el ancho de banda teórico del dispositivo de transmisión inalámbrico instalado en el DSP que implementa el estándar ISO 802.11b y que como máximo alcanza 1,38 MBps, fueron y son cuellos de botella insalvables que no permitieron obtener mejores resultados para este modelo de codificador.

Construir un codificador de video en tiempo real basado en el estándar UIT-H263: En este caso lo que se pretendía era crear un codificador específico para conseguir una tasa de frames por segundo elevada, intentando alcanzar los 30 fps. Para ello, se utilizarían todas y cada una de las herramientas y mejoras desarrolladas desde el H261 al H263, como la codificación temporal o intra y la codificación bidireccional

con tramas PB. No obstante, dicho desarrollo será incremental y el resultado final vendrá determinado por factor tiempo, es decir, se irán aplicando nuevas técnicas conforme se vaya experimentando que las anteriores funcionan y aportan beneficios significativos. A priori, esta opción aprovecha mejor la redundancia temporal de las imágenes y genera una menor tasa de transmisión de información a costa de pérdida de calidad y precisión de imagen. No obstante la pérdida de paquetes entre el DSP y el dispositivo receptor pueden provocar errores en la decodificación final del video e incluso perder fragmentos más o menos grandes si dicha pérdida de paquetes se produce cuando se envían frames de referencia.

A continuación, se detallarán en profundidad las técnicas empleadas para construir los dos tipos de codificadores, uno sin compresión temporal (codificador INTRA) y otro con compresión temporal (codificador INTER).

Codificador INTRA

Este codificador debe emplearse siempre que se desee extraer información presente en imágenes, como detección de bordes, formas o estructuras. Este apartado se detalla más adelante y todas y cada una de las aplicaciones que en él se desarrollen pueden ser utilizadas junto a este codificador en tiempo real.

El porqué se tiene que trabajar con un codificador intra siempre que se desee realizar tratamiento de imagen es bastante sencillo. Al trabajar siempre con tramas completas, el análisis de un frame puede iniciarse desde el primer instante en que el codificador empieza a trabajar y el DSP en transmitir. Esto es debido a que por cada envío de imagen se recibe toda la información comprimida relevante de la escena.

A continuación se describe un diagrama de bloques conceptual con los pasos a seguir para codificar de cada uno de los frames, así como su porcentaje en tiempo de procesamiento medido en las pruebas de rendimiento efectuadas por los desarrolladores del estándar pertenecientes a la UIT (ver referencia oficial del estándar para más información):

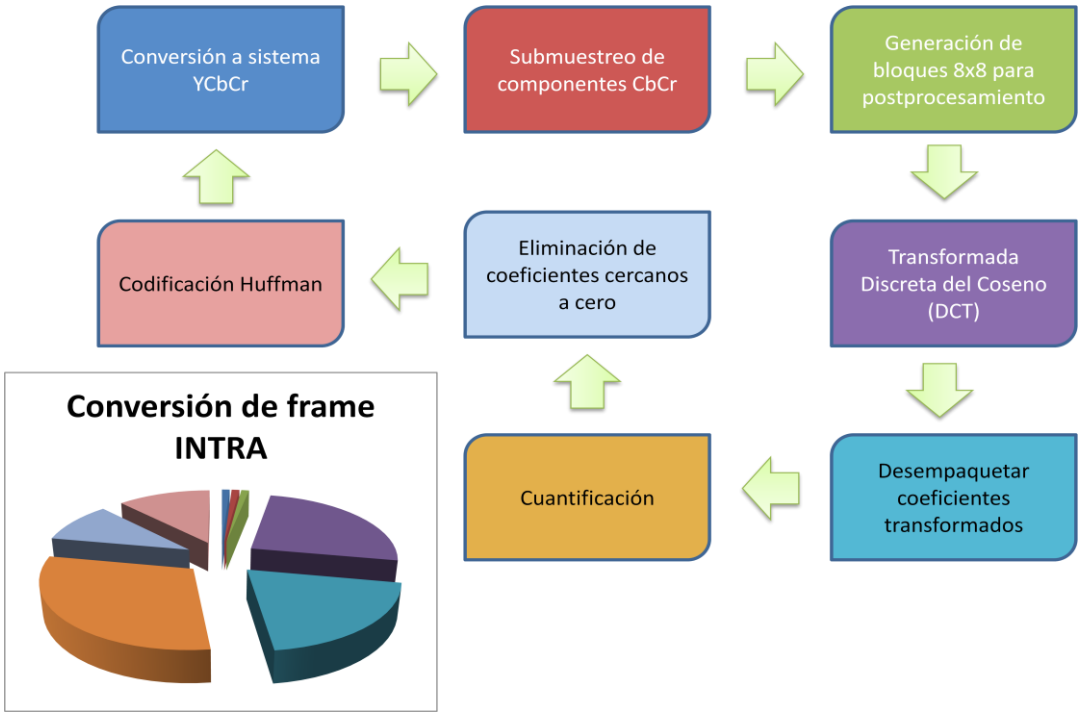


Figura 47: Estructura y tiempos

Este proceso se representa de forma circular porque es necesario aplicarlo a cada una de las imágenes tomadas por la lente instalada sobre el DSP. Traducido a algoritmo de programación podría verse como un bucle infinito en el que se realizan secuencialmente cada una de las acciones detalladas anteriormente.

Tras esto, es necesario describir en profundidad estas rutinas con la finalidad de aclarar conceptualmente cada una de las operaciones que se realizan sobre la imagen.

Conversión al sistema YCbCr y submuestreo de los componentes CbCr

Este es el primer paso para la codificación de una imagen en JPEG, en el que inicialmente es necesario convertir el color de la imagen a un nuevo espacio de color. Según las especificaciones del estándar, únicamente se permite realizar conversiones a los siguientes formatos:

- Escala de grises.
- Espacio de color YIQ.
- Espacio de color YCbCr.
- Matriz de color CMYK.

La mayoría de codificadores utilizan el espacio YCbCr y precisamente por este motivo se decidió optar por él como sistema de referencia de información del color. El espacio YCbCr está formado por tres componentes, al igual que el RGB:

- Y: Es la componente de luminancia y también es el valor usado por los monitores monocromos para representar el Color RGB. Además, en las ecuaciones puede observarse que el mayor factor multiplicativo de componentes se encuentra en G, ya que el ojo humano es más sensible a la componente verde.
- Cb: Componente de crominancia. Indica cómo es de azul la imagen origen.
- Cr: Componente de crominancia. Indica cómo es de roja la imagen origen.

Figura 48: Imágen RGB a YUV

Pero como la cámara DSKEye posee una distribución particular que no ofrece una configuración RGB nativa, es necesario primero realizar una transformación previa de RAW a RGB y después de RGB a YUV. Éste proceso se puede englobar en uno sólo y formar una función característica que realiza dicha transformación (la cual se encuentra en el código fuente del proyecto). Dado que en el anterior subproyecto se detalló el paso de RAW a RGB, en este caso y para no repetir, nos centraremos en el

paso de RGB a YUV. Para poder pasar del sistema RGB al YCbCr debemos aplicar las siguientes ecuaciones matemáticas:

$$Y = 0.3R + 0.6G + 0.1B \qquad Cb = \frac{U}{2} + 0.5$$
$$U = B - Y$$
$$V = R - Y \qquad Cr = \frac{V}{1.6} + 0.5$$

(Ecuación 1)

El siguiente paso es submuestrear los canales Cb y Cr por 2 en ambas direcciones ya que el ojo humano es relativamente sensible al contenido en altas frecuencias de los canales de crominancia. En la siguiente gráfica puede observarse cómo responde el ojo humano a las variaciones de luminancia y crominancia:

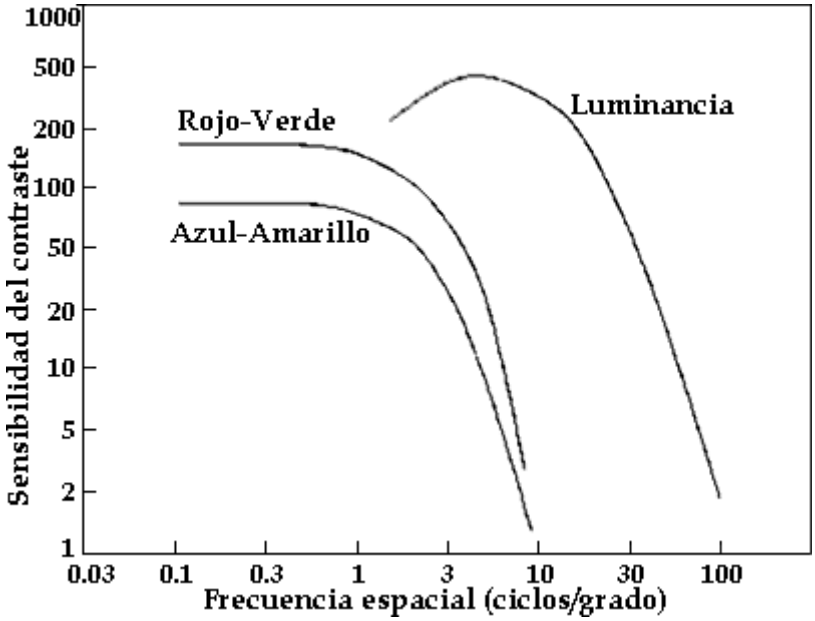


Figura 49: Respuesta del ojo humano

Como puede verse, las componentes Cb y Cr tienen mucho menos contraste y por tanto menos información que la luminancia. Debido a este fenómeno, es muy recomendable submuestrear los canales de crominancia, para evitar cargar con información innecesaria. Así pues, los canales finales tendrán la siguiente proporción:

Y1	Y2	Y3	Y4	Cr1	Cr2	Cb1	Cb2
Y5	Y6	Y7	Y8	Cr3	Cr4	Cb3	Cb4
Y9	Y10	Y11	Y12				
Y13	Y14	Y15	Y16				

Figura 50: Submuestreado de la imagen

El siguiente paso es ordenar los píxeles de la imagen de forma desentrelazada (3 pasadas, una por canal) o entrelazada (una exploración sencilla). Precisamente este proceso permite la decodificación futura de la imagen, es decir, pasar de la representación luminancia-crominancia a RGB para visualizarla con una mínima cantidad de memoria intermedia. Para los datos entrelazados, los bloques DCT son ordenados de acuerdo a los parámetros especificados en la trama.

Ya por último el estándar especifica un cambio en el nivel de color para poder operar en un rango de 2^7 términos, es decir, será necesario restar a cada píxel el valor 128, para tener números positivos y negativos.

Generación de bloques 8x8 para postprocesamiento

Para poder aplicar en un futuro la DCT es necesario trocear la imagen en bloques de 8x8. Hay que tener en cuenta que todas las imágenes no van a tener dimensiones que sean divisibles entre 8, es decir que la dimensión X e Y módulo 8 de la imagen sea 0.

Para solventar esto, tendremos que realizar dos operaciones: una para la dimensión X(anchura de imagen) y otra para la dimensión Y (altura) de la imagen. Para hacer que la anchura de la imagen sea divisible entre 8, debemos completar las columnas restantes con las columnas más a la derecha de la imagen original. Para el caso de que la anchura de la imagen sea divisible entre 8, debemos hacer la misma operación de antes pero esta vez, completaremos usando la fila más abajo de la imagen original. Para mayor claridad puede verse el siguiente ejemplo:

Ejemplo: La imagen original en la forma *Y*, *Cr*, *Cb* se divide en bloques de 8x8 pixeles siendo, para una imagen en formato CCIR 601 de 720 x 576, un total de 6480 bloques de luminancia *Y* y 3240 bloques para cada una de las componentes *Cr* y *Cb*. Cada uno de estos bloques forma una matriz de 64 números de 0 a 255 (en imágenes de 8 bits) para la luminancia, y de –128 a +127 para las componentes *Cr* y *Cb :*

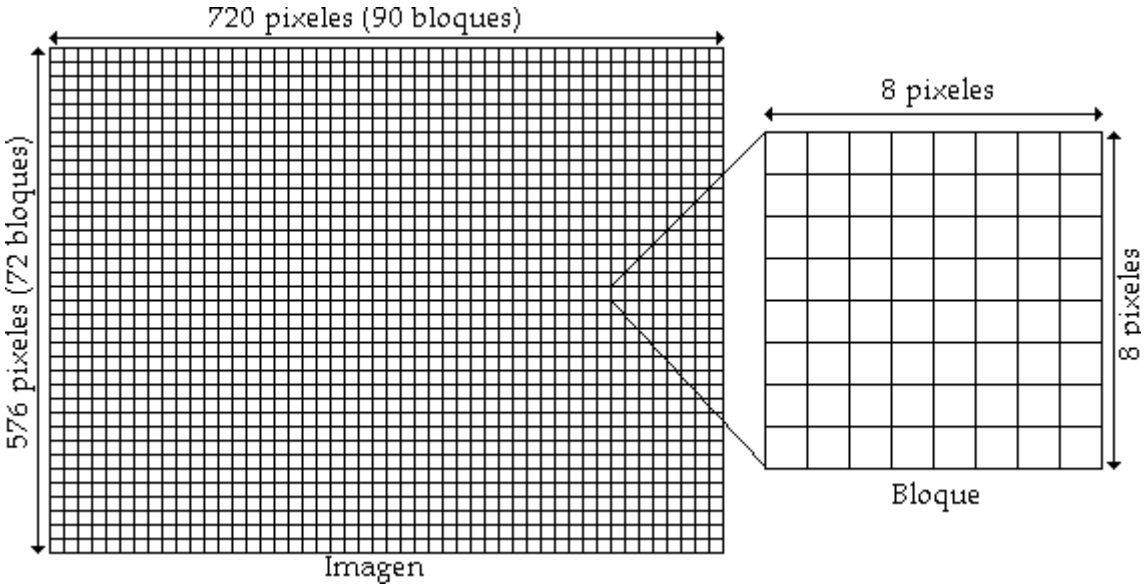


Figura 51: Distribución en bloques

Cada una de estas tres matrices (Y, Cr, Cb) es necesario dividirla en submatrices de 8x8 valores, que se conocen como Unidad de Datos (Data Unit). A esta Unidad de Datos se la conoce también con el nombre de "segmento". El estándar H.263 define también el concepto de Unidades Mínimas Codificables (Minimum Coded Units, ó MCU), como una submatriz compuesta por la mínima cantidad de Unidades de Datos de cada uno de los 3 componentes (Y, Cr, Cb), de tal forma que se tenga la misma cantidad de píxeles en cada componente original, antes del submuestreo. Por ejemplo, si se utilizase el factor de submuestreo habitual (2 para la escala horizontal y 2 para la vertical en las matrices Cr y Cb), cada MCU estaría compuesta por 4 Unidades de Datos de luminancia y dos Unidades de Datos de crominancia (una Cr y una Cb).

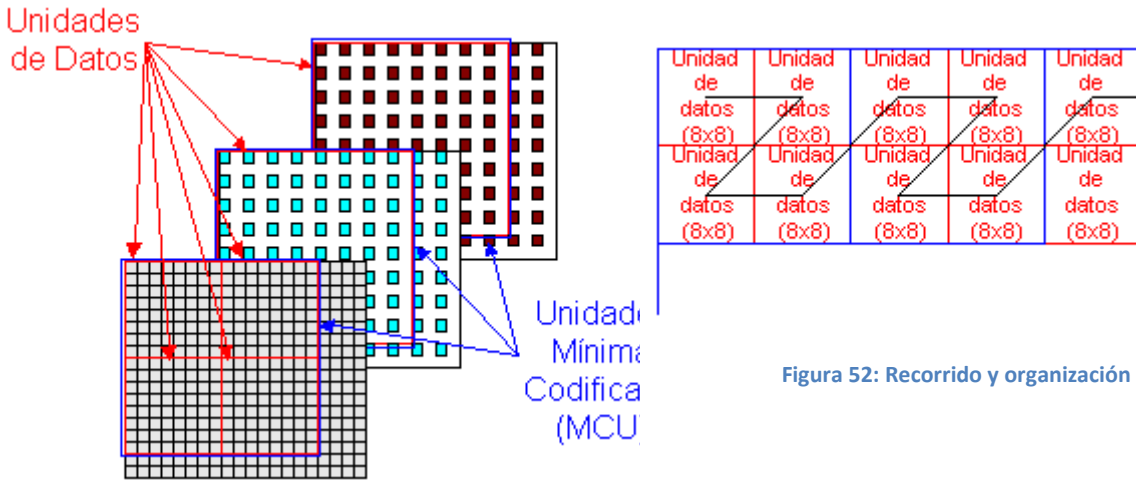


Figura 52: Recorrido y organización

Toda imagen INTRA debe componerse de un número entero de MCU, de tal forma que en caso de que las dimensiones de la imagen no permitan efectuar esta división en forma entera, la imagen deberá ser completada. Esto se hace repitiendo el último valor.

El archivo se codificará cada MCU, de izquierda a derecha y de arriba para abajo, y en cada MCU será codificada cada Unidad de Datos en forma independiente.

En el encabezado del archivo se especifica si las MCU de cada componente se encuentran intercaladas, o si se almacena cada componente por separado. Estas dos posibilidades se muestran a continuación:

MCUY0, MCUY1, ..., MCUYN, MCUCr0, MCUCr1, ..., MCUCrN, MCUCb0, MCUCb1, ..., MCUCbN

MCUY0, MCUCr0, MCUCb0, MCUY1, MCUCr1, MCUCb1, ..., MCUYN, MCUCrN, MCUCbN

Un frame INTRA almacena en primer lugar los atributos de la imagen, tales como el tamaño en pixels y centímetros/pulgadas, tipo de imagen y las tablas que serán necesarias durante el proceso de compresión. A continuación se almacena la imagen, codificando cada una de las Unidades de Datos, es decir, de cada MCU.

Transformada Discreta del Coseno

La **Transformada DCT** es una transformada semejante a la transformada rápida **2D de Fourier**, en la que se parte de un conjunto de puntos en un dominio espacial que posteriormente se transforman en una representación equivalente en el dominio de frecuencias.

La DCT está bastante relacionada con la DFT, con la diferencia de que la DCT es una transformada real, debido a que los vectores base se componen exclusivamente de funciones coseno muestreadas. Además la DCT minimiza algunos de los problemas que surgen con la aplicación de la DFT a series de datos, como veremos más adelante.

La transformada del coseno es la más ampliamente utilizada en compresión de imágenes. Esta transformada cuenta con una buena propiedad de compactación de energía, que produce coeficientes descorrelacionados donde los vectores base de la DCT dependen sólo del orden de la transformada seleccionado y no de las propiedades estadísticas de los datos de entrada.

La descorrelación de coeficientes es muy importante para cualquier proceso de compresión, ya que el posterior tratamiento de cada coeficiente se puede realizar de forma independiente, sin pérdida de eficiencia de compresión. Otro aspecto importante de la DCT es la capacidad de cuantificar los coeficientes utilizando valores de cuantificación que se eligen de forma visual.

Otro motivo por el que el estándar H.263 especifica la utilización de la transformada del Coseno en lugar de la de Fourier radica en que la primera puede codificar mejor funciones lineales con menos componentes. Como puede verse en el siguiente ejemplo, si se codifica una función lineal utilizando ambas transformadas y luego las componentes correspondientes a frecuencias más altas son descartadas, la función original puede ser reconstruida mejor utilizando los componentes de la DCT.

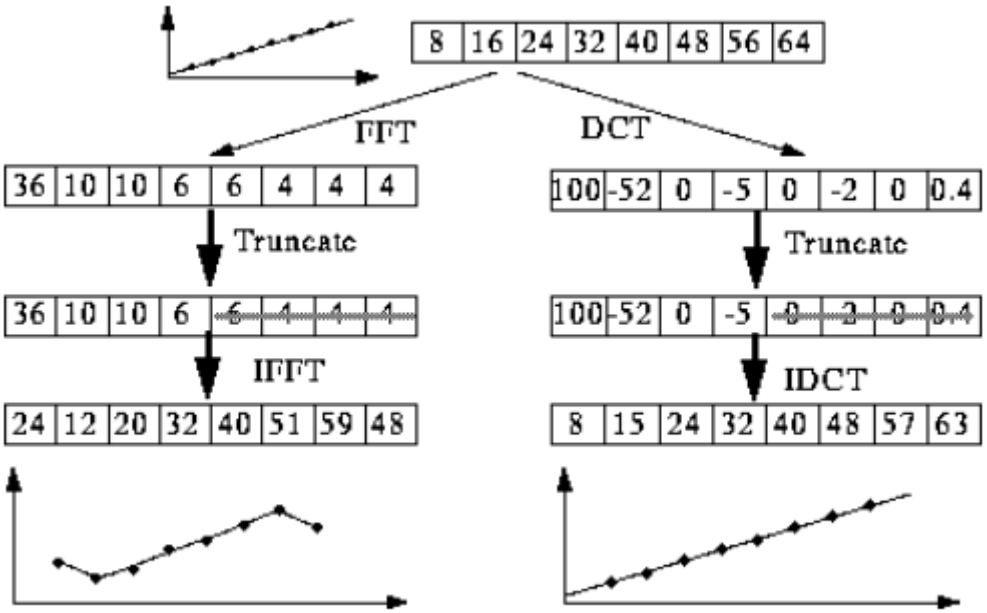


Figura 84 Compresión JPEG - FFT vs DCT

Figura 53: FFT versus DCT

El propósito de la **Transformada DCT** es el de procesamiento de las muestras originales y para ello se trabaja con las frecuencias espaciales presentes en la imagen original. Estas frecuencias espaciales son muy relativas al nivel de detalle presente en una imagen. El espacio de las altas frecuencias corresponde a los niveles de detalle más altos, mientras que las bajas frecuencias corresponden a los niveles más bajos de detalle. La **DCT** es aplicada sobre cada matriz de 8x8 y nos devuelve una matrixz de 8x8 con los coeficientes de la frecuencia. Dichos coeficientes multiplican a funciones coseno para reconstruir la imagen original. Estas funciones pueden verse a continuación:

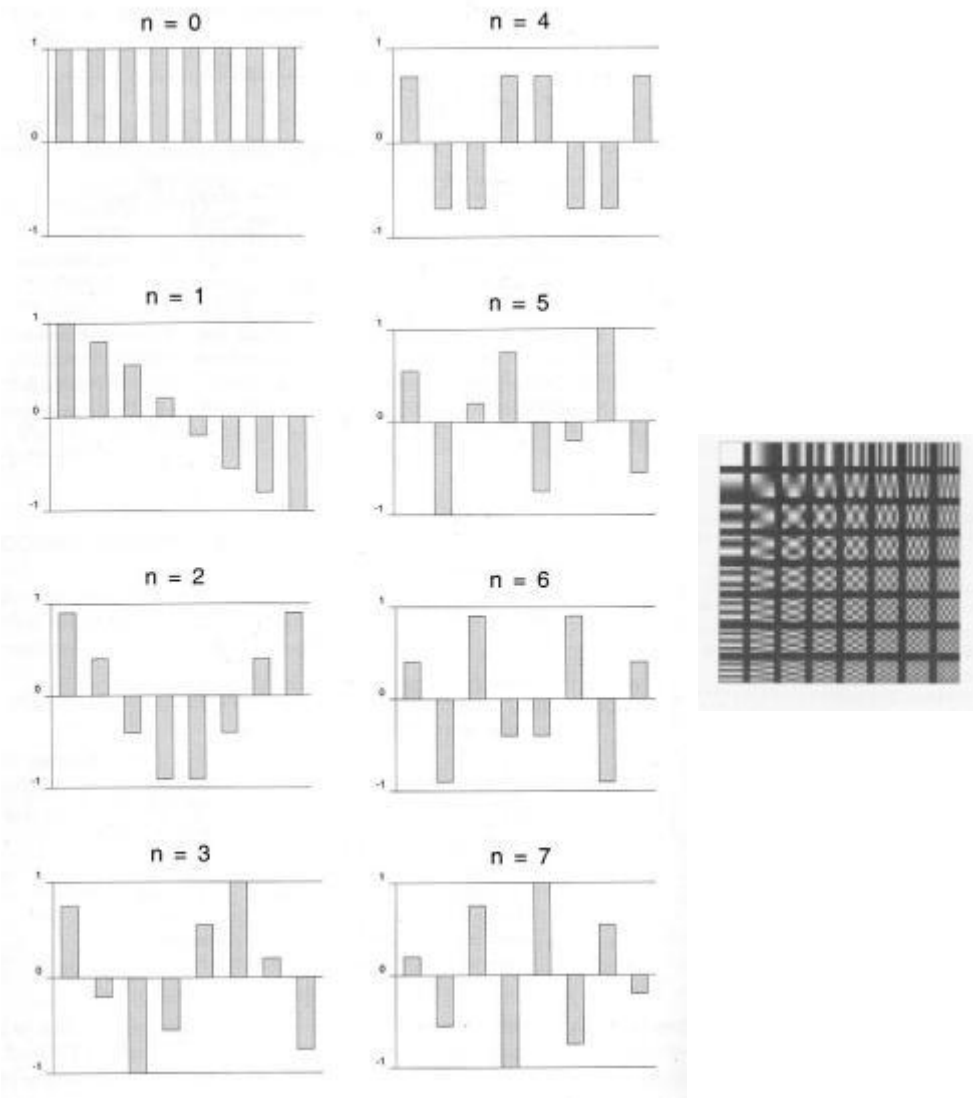


Figura 54: Tipos de funciones coseno

La primera componente corresponde a la señal continua de la imagen, es decir, el promedio de todos los valores de la imagen y se denomina componente de continua o DC. Los restantes valores se conocen como componentes AC, y las frecuencias aumentan a medida que avanzamos en dirección vertical u horizontal. El resultado de la DCT son valores reales. En teoría este paso debería ser completamente reversible. Sin embargo, ya que en una computadora no existen números reales, sino números de punto flotante y se cometen errores de redondeo, este paso no es completamente reversible, sino que hay una pérdida de datos.

Ya que la DCT opera con valores reales, puede resultar sumamente costosa en tiempo de procesamiento. Para acelerar este proceso, se suele utilizar una representación de números de punto fijo utilizando números enteros, produciéndose así algunos errores de redondeo, pero lográndose una velocidad de compresión muy superior.

La ecuación de la **Transformada Discreta del Coseno** es la siguiente:

$$DCT_{uv} = \frac{1}{\sqrt{2N}} C_u C_v \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p_{xy} \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

$$C_u, C_v = \begin{cases} \frac{1}{\sqrt{2}} \; para & u, v = 0 \\ 1 \quad otro & caso \end{cases}$$

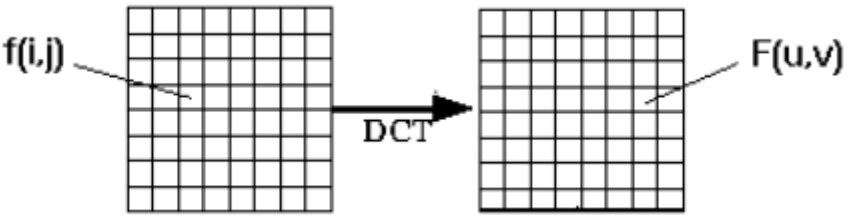


Figura 55: Paso de dominios

Donde **N** es el tamaño del bloque cuadrado, en este caso como los bloques de la imagen son de 8x8 pixels, **N** será igual a 8 y **'u'** y **'v'** son cada uno de los elementos del bloque que van desde **0,1,2,...N,** en este caso, como los bloques tienen un tamaño de 8, **'u'** y **'v'** irán desde 0,1,2,...,7 y serán cada una de las componentes de una matriz de 8x8, donde la coordenada (0,0) será la esquina superior izquierda.

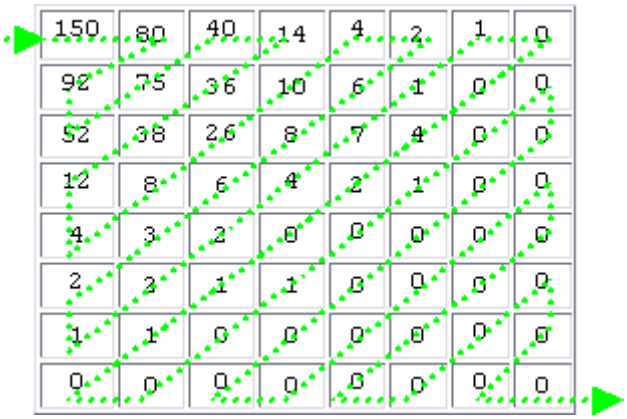
Luego aplicando esta Transformacion a cada elemento que compone la matriz de 8x8, obtenemos otra matriz de 8x8 elementos que son los coeficientes de las frecuencias. Un ejemplo de la aplicación de la transformada puede ser este:

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

En la tabla observamos una matriz de 8x8 elementos que corresponden a un bloque de 8x8 píxeles de un bloque de la imagen original. Se puede observar como gran parte de la triangular inferior está llena de ceros, esto hará que más tarde se pueda comprimir más la imagen.

Desempaquetar coeficientes transformados y eliminación de los coeficientes cercanos a cero

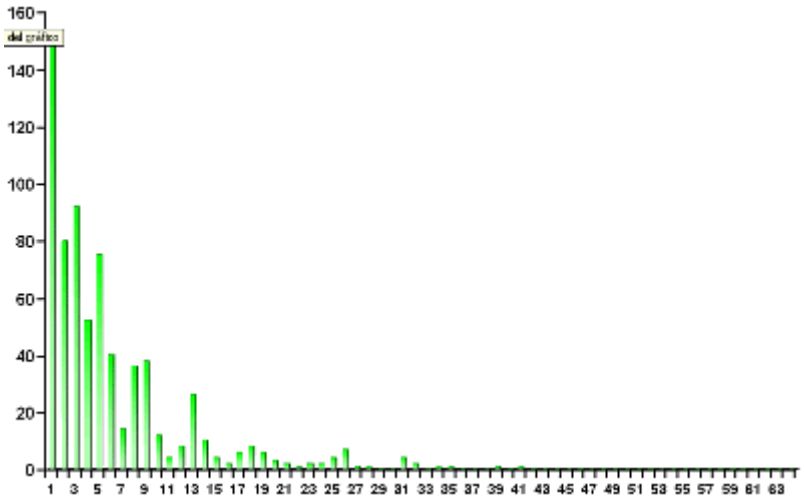
Una vez obtenida esta matriz (bloque de datos de la DCT correspondiente a un bloque de la imagen original), se le somete a una reordenación en zig-zag, es decir, debemos recorrer la matriz del siguiente modo, que gráficamente se podrá observar como es (lo veremos sobre el ejemplo anterior):



Lo que hacemos es recorrer la matriz comenzando en 150, después 80,92,52,75,40,...,0.

La razón por la que se hace el recorrido en zig-zag de los coeficientes del bloque obtenido de la aplicación de la DCT, es debido al orden creciente del espacio de frecuencias, ya que los primeros valores que nos encontramos al recorrer la matriz pertenecen a los valores de espacio de frecuencias más bajo (es decir a los niveles de detalle más bajos en la imagen original) y los valores finales del recorrido en zig-zag de la matriz corresponden a las altas frecuencias (es decir, los detalles más altos).

Luego al hacer esto nos queda un vector de 64 elementos que sería : (150, 80, 92, 52, 75, 40, 14, 36, 38, 12, 4, 8, 26, 10, 4, 2,...,0,). Este vector representado en una gráfica en 2D quedaría de la siguiente forma:



En el eje X tenemos todos los 64 valores de vector, y en el eje Y tenemos las amplitudes del espacio de frecuencias. El primer valor del vector (correspondiente al que tiene índice 0) es el que tiene

la más baja frecuencia en la imagen y le llamaremos término **DC**; en cambio todos los demás términos del vector serán llamados términos **AC.**

Cuantificación

En esta fase de la codificación INTRA es donde se va a producir en la imagen original las pérdidas de calidad típicas de JPEG y que serán aprovechadas para permitir grandes ratios de compresión en las imágenes.

Una vez que tenemos el vector de elementos de la DCT ordenado mediante el método de Zig-Zag, debemos aplicar la cuantificación a cada uno de los 64 elementos que forman el vector que tenemos. La cuantificación por tanto consistirá en dividir cada valor de este vector por un número correspondiente a una tabla de cuantificación prediseñada que permitirá redondear cada elemento de la DCT al entero más cercano.

La Matriz de cuantificación es una matriz de 8x8 píxeles formada por elementos denominados cuantos, esta matriz posee un elemento por cada coeficiente de la DCT y es siempre simétrica.

Esta matriz tendrá la característica de que los elementos más arriba y a la izquierda tendrán un menor valor, mientras que aquellos elementos que estén más abajo a la derecha tendrán un valor más alto. El Cuantificador divide los elementos de la DCT por el correspondiente valor del cuanto de la matriz de cuantificación. La matriz de cuantificación podría ser cualquier matriz que cumpla los requisitos anteriores. Un ejemplo de tabla de cuantificación podría ser la siguiente:

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

El motivo de la singular disposición de los valores de los elementos en la tabla de cuantificación es debido a que cuando dividimos los valores correspondientes de altas frecuencias del Vector de la DCT con los valores más grandes de la tabla de cuantificación el resultado obtenido es como si se hiciera un redondeo. Teniendo alguna de estas tablas de cuantificación lo que se hace es recorrer esta matriz en forma de Zig-Zag, como hicimos anteriormente y obtener de nuevo otro vector, este vector contendrá los valores de la tabla de cuantificación ordenados con el metodo Zig-Zag, y de este modo, ya podremos aplicar la cuantificación de la siguiente forma:

```
for (i=0; i<64 ; i++)
VectorDCT[i]=(int) (VectoDCT[i] / VectorQuant[i]);
```

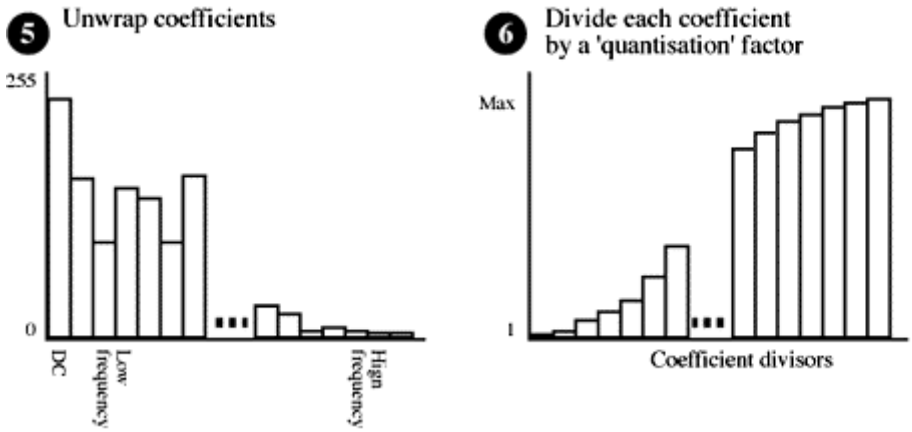


Figura 56: Ordenación de coeficientes

Donde *VectorDCT* es el Vector de datos obtenidos de la DCT y *VectorQuant* es vector que resulta del recorrido en Zig-Zag de la tabla de Cuantizacion.

Una vez realizada esta transformación obtenemos un nuevo Vector, que será el vector Cuantificado. En el ejemplo que estamos siguiendo el vector sería el siguiente: 150, 80, 92, 26, 75, 20, 4, 18 ,19, 3, 1, 2, 13, 3, 1, 0, 1, 2, 2, 0, 0, 0, 0, 0, 1, 1, 0,....0.

Una vez que ya tenemos por cada bloque de 8x8 de la imagen su vector cuantificado correspondiente, podremos empezar a realizar la codificación binaria. Lo que realmente hace la fase de cuantificación por tanto es redondear el elemento de la DCT al entero más cercano, de este modo los coeficientes de las altas frecuencias (elementos de la DCT próximos a cero) se convierten prácticamente en un valor 0, y por tanto fácilmente codificable en la siguiente fase de codificación y además permitirá una más eficiente compresión.

El proceso de cuantificación de la DCT es una pieza fundamental de la codificación INTRA. De este modo quitamos las altas frecuencias de la imagen original (los detalles de la imagen) ya que el ojo humano es menos sensible a las altas frecuencias que a los espacios de frecuencias bajos, así podemos eliminar las altas frecuencias de la imagen, sin tener una pérdida significativa de la calidad.

Es en esta fase, una vez que ya se ha Cuantificado el vector de la DCT, cuando se nos permite variar la calidad de la imagen, y por tanto el tamaño del fichero (es la opción que nos permiten los programas de retoque fotográfico en el que se puede variar la *calidad-tamaño_fichero* de la imagen). Por tanto, retocando y variando los coeficientes de cuantificación se podrán fijar de antemano cuantos bytes van a transmitirse por cada frame (muy útil a la hora de realizar pruebas de rendimiento).

Codificación Huffman

Exceptuando el primer coeficiente de la matriz de valores Cuantificados, todos se codifican utilizando el método conocido como **DPCM (Differential Pulse Code Modulation)**. Este método codifica

cada coeficiente DC como la diferencia con el anterior. Para el primer componente DC se supone que el anterior fue 0.

DC Coef Difference	Size	Typical Huffman codes for Size	Additional Bits (in binary)
0	0	00	-
-1,1	1	010	0,1
-3,-2,2,3	2	011	00,01,10,11
-7,…,-4,4,…,7	3	100	000,…,011,100,…111
-15,…-8,8,…,15	4	101	0000,…,0111,1000,…,1111
⋮	⋮	⋮	⋮
-1023,…-512,512,…,1023	10	1111 1110	00 0000 0000,…,11 1111 1111
-2047,…-1024,1024,…2047	11	1 1111 1110	000 0000 0000,…,111 1111 1111

Los restantes 63 componentes, son conocidos como componentes AC. Luego de aplicarse la cuantificación, es de esperar que varios de estos componentes tengan valor 0, por lo que una imagen INTRA codifica antes de cada componente AC la cantidad de componentes con valor 0 que hay antes de este componente, obteniéndose así tuplas de dos valores. Se dispone también de un código para indicar "todos los restantes componentes son 0", que es el par (0,0).

Dadas las características del método de codificación de entropía utilizado, no es posible codificar un valor de RLE superior a 15, por lo cual, en caso de presentarse esta situación, el par (RLE, AC) deberá ser descompuesto en varios pares donde RLE tenga un valor 15 o inferior, y codificar el valor AC como 0. Por ejemplo, si se tuviese: (20, 5) este par debería ser codificado como (15,0) (4,5).

Teniendo el Vector Cuantificado que se ha obtenido de la fase anterior que contiene 64 elementos, quitando el primer elemento nos quedan 63 (lo que hacemos es saltarnos el primer coeficiente de vector, que es el que posee la más baja frecuencia que será codificado con un bit diferente).

Ahora, comenzado en el elemento de valor 80 (posición 1 del vector Cuantificado), tenemos que hacer parejas del siguiente modo (x, y), de donde x correspondería al número de ceros consecutivos anteriores al valor actual, e y es el valor del elemento; de este modo con el ejemplo anterior tendremos lo siguiente:

(0,80), (0,92), (0,26), (0,75), (0,20), (0,4), (0,18), (0,19), (0,3), (0,1), (0,2), (0,13), (0,3), (0,1), (1,1), (0,2), (0,2), (5,1), (0,1), EOB.

EOB, es una marca que indica que todas las demás parejas son ceros, es decir, que indica que tenemos el final del vector repleto de ceros. Por tanto es un indicador de fin de bloque (End Of Block) y es equivalente a (0,0). Posteriormente, esta misma operación se realizará para todos vectores Cuantificados de los bloques que forman la imagen.

Hay que tener en cuenta una cosa muy importante y es que para la codificación se impone una restricción en la que el número de ceros anteriores a un valor no puede sobrepasar el valor de 15, (0xF) ya que tiene que ser codificado únicamente con 4 bits (16 elementos). Para ver esto no nos vale el ejemplo anterior y por ello proponemos un nuevo ejemplo:

Si tenemos el siguiere código [(0,80), (32,2), (5,26), (6,8), (0,0)], teniendo en cuenta lo que hemos dicho anteriormente esto quedaría codificado de la siguiente forma:

[(0,80), (15,0), (15,0), (0,2), (5,26), (6,8), (0,0)].

Luego una vez realizado esto ya únicamente nos queda codificar esto mediante la Codificación Huffman teniendo en cuenta la siguiente tabla especificada por el estándar H.263 , en la que un determinado valor posee una categoría determinada y una determinada representación en binario:

Table 1: Coefficient coding categories

Range	DC difference category	AC category
0	0	0
-1, 1	1	1
-3, -2, 2, 3	2	2
-7,..., -4, 4,..., 7	3	3
-15,..., -8, 8,..., 15	4	4
-31,..., -16, 16,..., 31	5	5
-63,..., -32, 32,..., 63	6	6
-127,..., 64, 64,..., 127	7	7
-255,..., -128, 128,..., 255	8	8
-511,..., -256, 256,..., 511	9	9
-1023,..., -512, 512,..., 1023	10	10
-2047,..., -1024, 1024,..., 2047	11	11
-4095,..., -2048, 2048,..., 4095,	N/A	12
-8191,..., -4096, 4096,..., 8191	N/A	13
-16383,..., -8192, 8192,..., 16383	N/A	14
-32767,..., -16384, 16384,..., 32767	N/A	15

Figura 57: Categorías de coeficientes

Table 2: Huffman codes for DC categories

Category	Code
0	010
1	011
2	100
3	00
4	101
5	110
6	1110
7	11110
8	111110
9	1111110
10	11111110
11	111111110

Figura 58: Códigos para DC

Table 3: AC Huffman table

Run	Category	Code	Run	Category	Code	Run	Category
0	0	1010	6	0		12	0
0	1	00	6	1	1111011	12	1
0	2	01	6	2	11111111000	12	2
0	3	100	6	3	1111111110100111	12	3
0	4	1011	6	4	1111111110101000	12	4
0	5	11010	6	5	1111111110101001	12	5
0	6	111000	6	6	1111111110101010	12	6
0	7	1111000	6	7	1111111110101011	12	7
0	8	1111110110	6	8	1111111110101100	12	8
0	9	1111111110000010	6	9	1111111110101101	12	9
0	10	1111111110000011	6	10	1111111110101110	12	10
1	0		7	0		13	0
1	1	1100	7	1	11111001	13	1
1	2	111001	7	2	11111111001	13	2
1	3	1111001	7	3	1111111110101111	13	3
1	4	111110110	7	4	1111111110110000	13	4
1	5	11111110110	7	5	1111111110110001	13	5
1	6	1111111110000100	7	6	1111111110110010	13	6
1	7	1111111110000101	7	7	1111111110110011	13	7
1	8	1111111110000110	7	8	1111111110110100	13	8
1	9	1111111110000111	7	9	1111111110110101	13	9
1	10	1111111110001000	7	10	1111111110110110	13	10
2	0		8	0		14	0
2	1	11011	8	1	11111010	14	1
2	2	11111000	8	2	111111111000000	14	2
2	3	1111110111	8	3	1111111110110111	14	3
2	4	1111111110001001	8	4	1111111110111000	14	4
2	5	1111111110001010	8	5	1111111110111001	14	5
2	6	1111111110001011	8	6	1111111110111010	14	6
2	7	1111111110001100	8	7	1111111110111011	14	7
2	8	1111111110001101	8	8	1111111110111100	14	8
2	9	1111111110001110	8	9	1111111110111101	14	9
2	10	1111111110001111	8	10	1111111110111110	14	10
3	0		9	0		15	0
3	1	111010	9	1	111111000	15	1
3	2	111110111	9	2	1111111110111111	15	2
3	3	11111110111	9	3	1111111111000000	15	3
3	4	1111111110010000	9	4	1111111111000001	15	4
3	5	1111111110010001	9	5	1111111111000010	15	5
3	6	1111111110010010	9	6	1111111111000011	15	6
3	7	1111111110010011	9	7	1111111111000100	15	7
3	8	1111111110010100	9	8	1111111111000101	15	8
3	9	1111111110010101	9	9	1111111111000110	15	9
3	10	1111111110010110	9	10	1111111111000111	15	10

4	0		10	0	
4	1	111011	10	1	111111001
4	2	1111111000	10	2	1111111111001000
4	3	1111111110010111	10	3	1111111111001001
4	4	1111111110011000	10	4	1111111111001010
4	5	1111111110011001	10	5	1111111111001011
4	6	1111111110011010	10	6	1111111111001100
4	7	1111111110011011	10	7	1111111111001101
4	8	1111111110011100	10	8	1111111111001110
4	9	1111111110011101	10	9	1111111111001111
4	10	1111111110011110	10	10	1111111111010000
5	0		11	0	
5	1	1111010	11	1	111111010
5	2	1111111001	11	2	1111111111010001
5	3	1111111110011111	11	3	1111111111010010
5	4	1111111110100000	11	4	1111111111010011
5	5	1111111110100001	11	5	1111111111010100
5	6	1111111110100010	11	6	1111111111010101
5	7	1111111110100011	11	7	1111111111010110
5	8	1111111110100100	11	8	1111111111010111
5	9	1111111110100101	11	9	1111111111011000
5	10	1111111110100110	11	10	1111111111011001

Figura 59: Tabla de valores para AC

A la hora de codificar haciendo uso de esta tabla, debemos tener en cuenta que lo que se codifica en los pares de elementos resultantes del RLC es únicamente el elemento de la derecha del par, excepto en el caso en el que tengamos (0,0) [EOB] o (15,0). Para el siguiente ejemplo:

[(0,57), (4,23), (0,8), (2,1) , (0,0)].

Tendremos que la codificación será la siguiente:

Pareja	Run	Cat.	Valor	Valor Bin.
(0,57)	0	6	57	111001
(4,23)	4	5	23	10111
(0,8)	0	4	8	1000
(2,1)	2	1	1	1

Luego una vez que ya tenemos esto, es ahora cuando aplicamos la Codificación de Huffman:

Codificamos la pareja (0,57) como 111000.
Codificamos la pareja (2,1) como 11011.

Ahora debemos poner el código Huffman generado seguido de la codificación obtenida de la tabla del siguiente modo (para el ejemplo anterior):

Primera Pareja	Pareja n-sima	Ultima Pareja	Fin de Bloque

Codficacion Huffman	Tabla de Codificacion	Demas elementos	Codificacion Huffman	Tabla de Codificacion	EOB(Codigo Huffman asignado).
111000	111001	.....	11011	1	1010

Luego en el fichero para generar la imagen INTRA, debemos incluir la secuencia: 111000111001.....1101111010 que será la codificación de un bloque de la imagen con todas las transformaciones realizadas, pero antes de incluirlo en el fichero, necesitamos codificar el coeficiente DC de la cada Bloque.

Para codificar el Coeficiente DC que anteriormente se eliminó de los cálculos, sabemos que este coeficiente es el correspondiente a las bajas frecuencias en la imagen y es matemáticamente igual a la suma de las muestras 8x8 de la imagen dividida entre 8 (normalmente toma valores muy elevados).

El estándar nos dice que existe una gran conexión entre los coeficientes DC de los bloques consecutivos, por lo que se decidió codificar la imagen INTRA como la diferencia entre los coeficientes DC de bloques consecutivos de 8x8.

DiferenciaDC = DCi-DC(i-1)

Por tanto el DC del bloque actual será:

DCi = DC(i-1) + DiferenciaDC

Así para la decodificación se comenzará con el valor de DC igual a 0, y a los siguientes le aplicaremos la diferencia anterior. Por tanto, a la hora de codificar este el valor DC, tenemos que tener en cuenta los valores de la diferencia *DiferenciaDC* que se han calculado antes, de modo que si por ejemplo la *DiferenciaDC* es igual a -255, esto se codificará atendiendo a la tabla de codificación 1 , por tanto para codificar -255, tendriamos:

(DiferenciaDC = catergoriaCodificación, representación en bits)

Y nos quedaría la siguiente pareja:

(8,00000000) usando la tabla 1.

Suponiendo que 8 tenga el código Huffman 111110, nos quedaría que la codificación de esta pareja sería:

(111110, 00000000),

Luego teniendo en cuenta la codificación que se hizo antes de los demás elementos del vector, el flujo de bits que debemos escribir a fichero será (primero siempre se incluirá el coeficiente DC y luego los AC´s):

111111 00000000 111000 111001.....11011 1 1010

Y con esto terminamos la fase de codificación de un frame INTRA.

Ejemplo de una estructura de datos INTRA:

ff d8 ff e0 00 10 4a 46 49 46 00 01 01 00 00 01 00 01 00 00 **ff fe** 00 46

76 77 78 79 7a 83 84 85 86 b5 b6 b7 b8 b9 ba c2 03 00 04 ff db 00 43

00 08 06 06 07 06 05 08 07 07 07 09 09 08 0a 0c 14 0d 0c 0b 0b 0c 19 12

13 0f 14 1d 1a 1f 1e 1d 1a 1c 1c 20 24 2e 27 20 22 2c 23 1c 1c 28 37 29

2c 30 31 34 34 34 1f 27 39 3d 38 32 3c 2e 33 34 32 **ff c0** 00 0b 08 01 00

01 00 01 01 11 00 **ff c4** 00 1f 00 00 01 05 01 01 01 01 01 01 00 00 00 00

00 00 00 00 01 02 03 04 05 06 07 08 09 0a 0b **ff c4** 00 b5 10 00 02 01 03

03 02 04 03 05 05 04 04 00 00 01 7d 01 02 03 00 04 11 05 12 21 31 41 06

13 51 61 07 22 71 14 32 81 91 a1 08 23 42 b1 c1 15 52 d1 f0 24 33 62 72

82 09 0a 16 17 18 19 1a 25 26 27 28 29 2a 34 35 36 37 38 39 3a 43 44 45

46 47 48 49 4a 53 54 55 56 57 58 59 5a 63 64 65 66 67 68 69 6a 73 74 75

76 77 78 79 7a 83 84 85 86 87 88 89 8a 92 93 94 95 96 97 98 99 9a a2 a3

a4 a5 a6 a7 a8 a9 aa b2 b3 b4 b5 b6 b7 b8 b9 ba c2 c3 c4 c5 c6 c7 c8 c9

ca d2 d3 d4 d5 d6 d7 d8 d9 da e1 e2 e3 e4 e5 e6 e7 e8 e9 ea f1 f2 f3 f4

f5 f6 f7 f8 f9 fa **ff da** 00 08 01 01 00 00 3f 00 **xx xx xx xx xx xx xx xx**

xx xx xx xx xx xx xx xx xx xx xx xx ff d9

Marcas	**Significado**
FF D8	Inicio
FF FE	Comentarios (pueden omitirse)
FF E0	Marca de aplicación (no se usa)
FF DB	Tabla de quantización
FF C0	Inicio de imagen
FF C4	Tabla Huffman

FF DA	Inicio de los datos
XX XX	Datos codificados
FF D9	Fin de la imagen

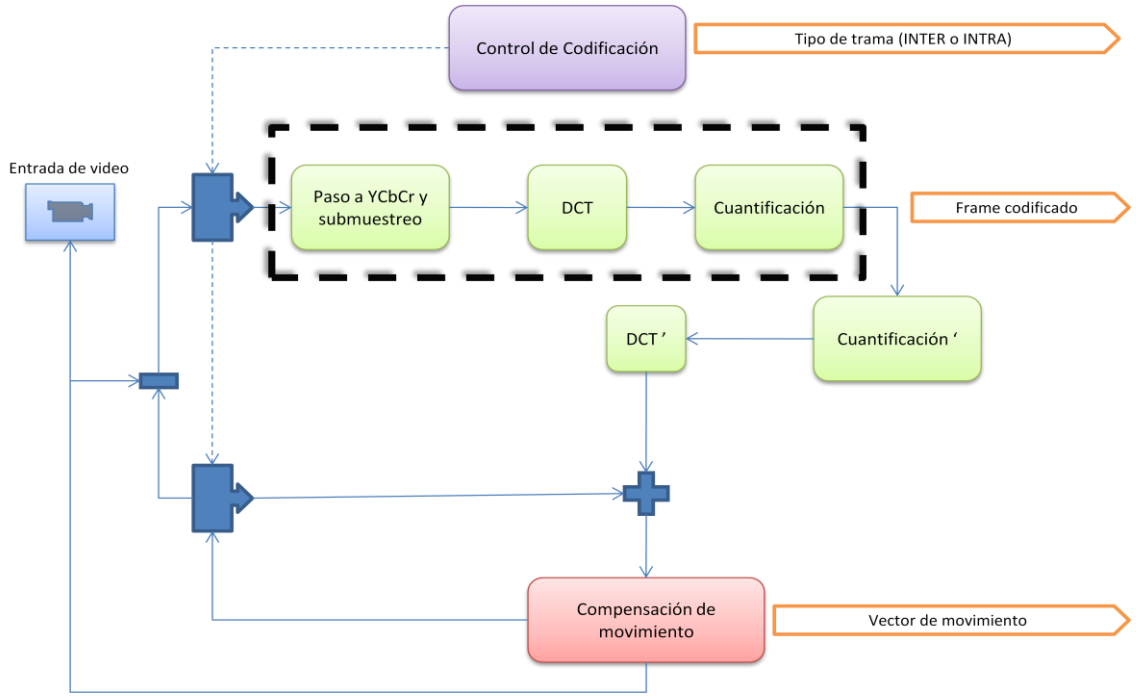
Codificador INTER

Para el desarrollo del codificador INTER se ha partido en exclusiva de las librerías H.263 estándar que pueden conseguirse en cualquier sitio web especializado dado que es código abierto bajo licencia GNU.

En concreto, para nuestro proyecto hemos utilizado la versión del tmnenc (TMN Encoder) desarrollado por Karl Olav Lillevold y Robert Danielsen, profesores adjuntos del instituto Telenor Research & Development de Kjeller, Noruega.

En esencia, la estructura y composición del codificador no ha sufrido ningún cambio significativo, simplemente adaptaciones aisladas de código para permitir un mejor funcionamiento bajo la arquitectura nativa del DSP.

La estructura vista como un diagrama ilustrativo del algoritmo, básicamente consta de los siguientes bloques:



En él se ve como la zona remarcada es la codificación INTRA vista en el apartado anterior para el otro codificador. Esta fase es idéntica, salvo aplicación de anexos especificados por UIT para H.263.

En la parte superior se encuentra el proceso de control de codificación, que abstractamente se puede identificar sin pérdida de la generalidad con las reglas de control de flujo y tratamiento de la

imagen especificadas por UIT para el H.263. En ellas se detallan cuales son las acciones a tomar dependiendo del tipo de frame que toque codificar, es decir, si se realiza codificación INTRA o por el contrario se utiliza el modo de predicción con diferencia de imagen. Se pueden cambiar varios parámetros para controlar la velocidad de generación de datos de video codificados.

Por otro lado, el proceso de compensación de movimiento está formado por un conjunto de reglas que permiten aprovechar la redundancia temporal de la secuencia de video capturada. Como resultado se genera un vector de movimiento, cuyas componentes horizontal y vertical tienen valores de un entero o mitad de entero. En el modo de predicción por defecto, estos valores están restringidos a la gama [-16, 15.5] que también es válida para los componentes de vectores en movimiento hacia delante y hacia atrás en las imágenes B (en el caso de que se desee codificar usando entramado PB). Un valor positivo del componente horizontal o vertical significa que la predicción se forma a partir de los píxeles de la imagen referenciada que están espacialmente a la derecha o por debajo de los píxeles que se predicen.

Ya por último, el decodificador recibe los flags enviados por el codificador donde se indica cuál es la fuente de datos y como debe decodificarlos, es decir, si es una trama INTER o una trama INTRA y en éste último caso de que trama P se deriva.

Como condiciones de partida, el estándar H.263 de UIT fija de antemano una serie de pautas que deben cumplirse al pie de la letra al igual que otras que simplemente sirven para adaptar mejor el algoritmo al dispositivo objetivo. A continuación se describen brevemente estas pautas, indicando en cada caso si el estándar las considera opcionales o no:

Formatos de entrada

El formato digital de entrada es CIF, con sus submúltiplos QCIF y sub QCIF. Como posibilidades se han añadido, además, los múltiplos 4CIF (4 veces CIF) y 16CIF (16 veces), aunque su ámbito de aplicación excede con respecto al vídeo de baja velocidad. En la práctica, el formato considerado más usual es QCIF. En la siguiente tabla se relacionan cada uno de los formatos con su 'bitrate' teórico calculado sin compresión:

Formato	**Pixels de luminancia**	**Líneas de luminancia**	**Pixels de Crominancia**	**Líneas de Crominancia**	**Bitrate sin compresión (Mbit/s)**			
					10 frames/s		**30 frames/s**	
					Gris	**Color**	**Gris**	**Color**
SQCIF	128	96	64	48	1.0	1.5	3.0	4.4
QCIF	176	144	88	72	2.0	3.0	6.1	9.1
CIF	352	288	176	144	8.1	12.2	24.3	36.5
4CIF	704	576	352	288	32.4	48.7	97.3	146.0

16CIF	1408	1152	704	576	129.8	194.6	389.3	583.9

Estimación de movimiento

La estimación de movimiento en el codificador base H.263 es similar a H.261. Se realiza, al igual que en este último, la búsqueda en el último cuadro transmitido del mejor predictor para bloques de imagen de 16x16 puntos. A fin de mejorar el resultado de la estimación, se ha incorporado la predicción a medio pixel de MPEG, en la cual los puntos intermedios se calculan por simple interpolación bilineal entre los puntos reales. El rango máximo de vectores es, al igual que en H.261, de aproximadamente ±15 puntos enteros.

DCT y cuantificación

Las DCT y la cuantificación se efectúan de la misma manera que en H.261. La DCT es del tamaño estándar 8x8. La componente continua en bloques INTRA (sin predicción) utiliza un paso de cuantificación fijo de 8 y los restantes coeficientes INTRA e INTER (con predicción) usan un cuantificador uniforme con zona muerta alrededor del cero. Los coeficientes se ordenan en zigzag antes de codificar, para disponer valores similares juntos.

Codificación

Para los vectores de movimiento, los tipos de macrobloque y los coeficientes DCT, se utilizan códigos de longitud variable tipo Huffman, al igual que en H.261. Para los vectores de movimiento se emplea predicción diferencial, de forma que lo que se transmite es el residuo de la resta entre el vector y un predictor formado a partir de los vectores anteriores. El predictor es más elaborado que en H.261 o MPEG (calcula la mediana entre tres vectores adyacentes).

Vectores ilimitados (opcional)

En este modo está permitido que los vectores de movimiento apunten a zonas que están, en parte, fuera de la imagen. Para esos puntos de referencia fuera de la imagen, se usa el punto más próximo del borde de la imagen.

Este modo es útil cuando existen objetos entrando en la imagen. En este caso, se podrán usar vectores prohibidos en el modo base para aproximar, al menos, la parte del objeto que ya había entrado en el cuadro de referencia.

Codificador aritmético (opcional)

En este modo se sustituyen los códigos Huffman por un codificador aritmético basado en un modelado de datos igualmente especificado en el estándar. Los códigos aritméticos posibilitan reducir el espacio ocupado por la trama binaria respecto al usado por una codificación Huffman, pero a costa de una mayor complejidad.

Modo de predicción avanzado (opcional)

Existen dos tipos de predicción permitidos en este modo: el especificado en el codificador base (vectores que cubren un área de 16x16 puntos) y otro en el cual el macrobloque se descompone en

cuatro bloques de 8x8, se busca y transmite un vector para cada bloque de 8x8. Cada macrobloque de la imagen puede utilizar uno cualquiera de los dos métodos, de forma que en áreas estáticas de la imagen o con objetos grandes podrán usarse vectores de 16x16, en zonas de la imagen donde los objetos sean más pequeños y tengan distinto movimiento se podrá realizar una mejor aproximación usando vectores que abarquen áreas de 8x8.

Además, en este modo se realiza compensación de movimiento promediada (sólo para los bloques con predicción 8x8), de forma que para cada punto se toman 3 predictores a partir de otros tantos vectores: el correspondiente al bloque donde está el punto y otros dos de bloques adyacentes. El objetivo es que la diferencia de predicciones entre bloques adyacentes no provoque residuos de predicción demasiado distintos que incrementen la visibilidad de los bloques que forman la imagen (efecto bloque).

Cuadros PB (opcional)

El objetivo de la opción es doblar la frecuencia de cuadro introduciendo muy poca carga adicional al canal. Por tanto, a igual velocidad binaria, un codificador H.263, con cuadros PB, presentará un refresco de imagen doble al que se presentaría si no se empleasen, aunque empeorará un poco la calidad individual de cada imagen.

Diagramas de bloques y estructura funcional del proyecto

Para facilitar la compresión directa del código generado se ha construido un diagrama de bloques que discrimina cada uno de los ficheros fuente de cada una de las aplicaciones desarrolladas. Es importante resaltar que en esta sección no se documentan cada una de las funciones programadas en cada fichero fuente, sino que se da una ligera orientación al lector para que pueda establecer una relación directa entre la secuencia de codificación teórica explicada en apartados anteriores y el código generado.

Así mismo no se ha considerado oportuno incluir el código fuente a efecto que permite la codificación de imágenes en movimiento, dado que dichos ficheros se encuentran disponibles en el CD del proyecto, por lo que en cualquier momento podrán ser consultados siempre y cuando lo consideren oportuno los profesores responsables de este proyecto.

A continuación se presentan dos diagramas conceptuales, uno relacionado con la aplicación que codifica video de forma espacial y otro relacionado con la codificación de video espacio-temporal.

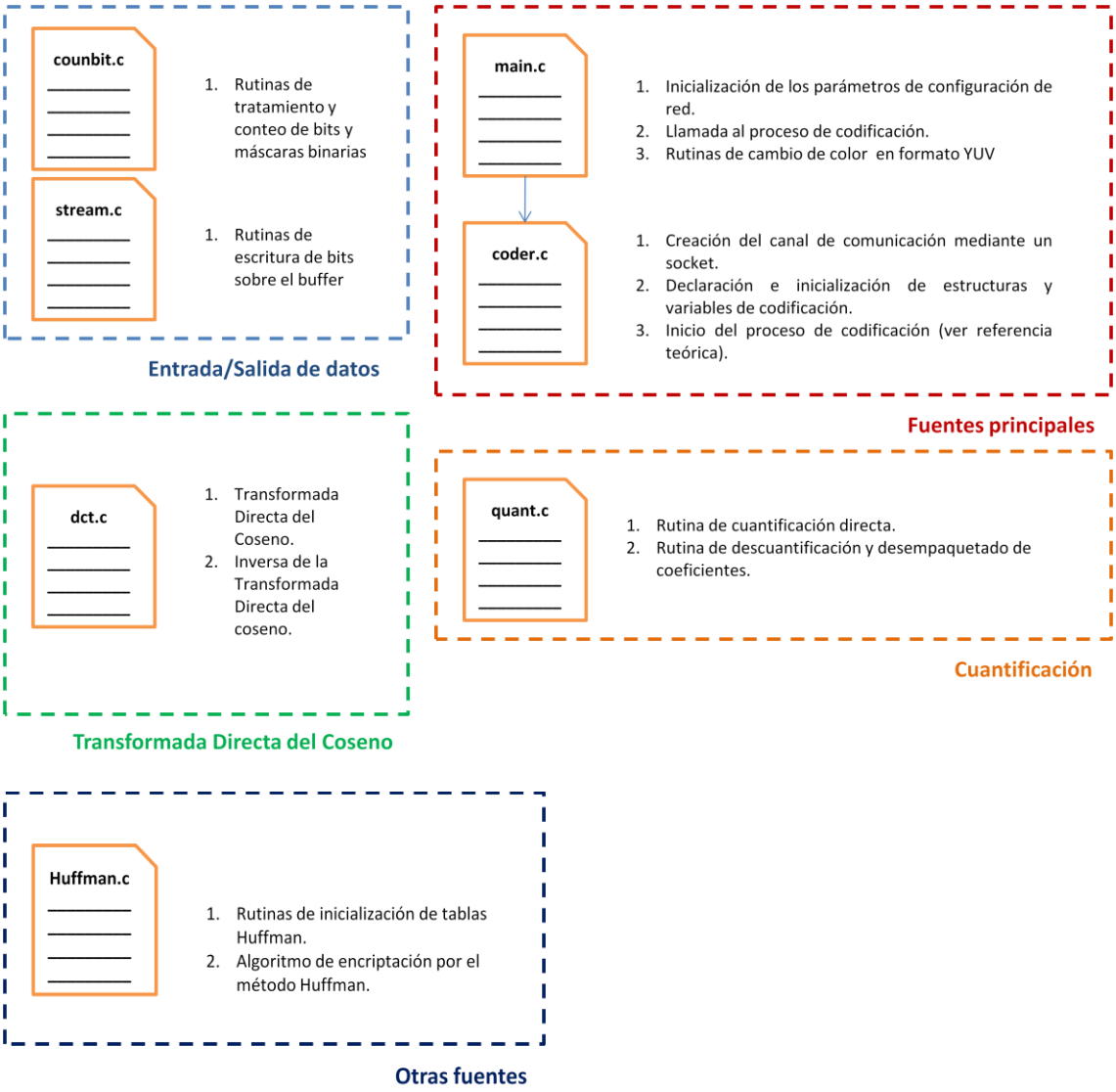


Figura 64: Codificador INTRA


io.c
1. Rutinas de entrada/salida de datos.
putbits.c
1. Rutinas de escritura de bits sobre el buffer
Entrada/Salida de datos
Main.c
1. Inicialización de los parámetros de configuración de red.
2. Llamada al proceso de codificación.
TSK_h263.c
1. Creación del canal de comunicación mediante un socket.
2. Declaración e inicialización de estructuras y variables de codificación.
3. Inicio del proceso de codificación (ver referencia teórica).
coder.c
1. Rutinas de codificación de cada tipo de frame (INTRA, INTER, PP, PB...)
2. Rutinas de interpolación y reconocimiento de patrones de video.
Fuentes principales
dct.c
1. Transformada Directa del Coseno.
2. Inversa de la Transformada Directa del coseno.
Transformada Directa del Coseno
counbit.c
1. Rutinas de tratamiento y conteo de bits y máscaras binarias
counbit.c
1. Rutinas de manejo de vectores de compensación.
Conteo y operaciones sobre bits
mot_est.c
1. Rutinas de cálculo de estimación de movimiento sobre macrobloques.
2. Rutinas SAD.
pred.c
1. Rutinas de predicción y reconocimiento de secciones de imagen.
2. Predicción de tramas P.
3. Predicción de tramas B.
4. Predicción de tramas PB y PP.
Predicción y compensación emovimiento
quant.c
1. Rutina de cuantificación directa.
2. Rutina de descuantificación y desempaquetado de coeficientes.
ratectrl.c
1. Rutinas relacionadas con el control de retardo en tiempo real.
2. Rutinas para la inicialización de los procesos de cuantificación.
Cuantificación

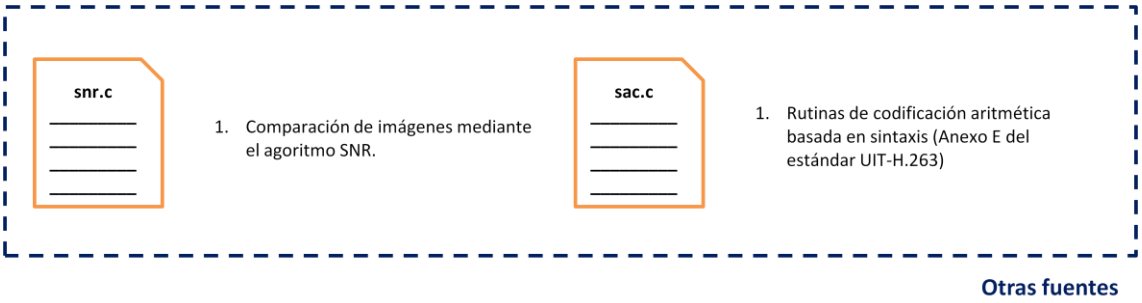


Figura 65: Codificador INTER

Para configurar cada uno de los parámetros de ejecución y control (ejecución de anexos opcionales y modos de control) será necesario modificar el fichero de configuración *sim.h* y el *config.h*, básicos para cualquier modificación de la secuencia normal de acciones de codificación.

Ejecución de herramientas desarrolladas

Debido a la naturaleza del proyecto, ha sido necesario crear por un lado el codificador de video y por el otro el decodificador. La ejecución del codificador es inmediata tras concluir con éxito el paso descrito en el anterior punto, pero la ejecución del decodificador es algo más compleja, pero no mucho más.

Una vez el codificador está activado y ejecutándose en el DSP, éste tiene abierto un puerto de escucha por la red inalámbrica Ad-Hoc previamente configurada y se encuentra en un bucle de espera activa hasta que un proceso le interrumpa y le pida nuevos datos.

Dicho proceso será el decodificador, que gracias a que se han seguido pauta a pauta cada una de las normas establecidas en el estándar H.263, es posible utilizar cualquier decodificador basado en este formato, como por ejemplo el utilizado en nuestro caso para el proceso de testeo y evolución del proyecto, basado en el decodificador diseñado por Karl Olav Lillevold y Robert Danielsen.

Éste decodificador bien puede ser descargado como programa ejecutable desde cualquier link especificado en los apéndices de esta memoria o utilizar código fuente abierto para crear nuestro propio decodificador.

En nuestro caso, no hemos necesitado retocar ninguna sección del decodificador, pero para futuras aplicaciones y desarrollos posteriores esta opción es bastante potente puesto que permitiría entre otras cosas postprocesado de imagen utilizando una arquitectura basada en PC.

Esta última opción será muy interesante a la hora de, por ejemplo, localizar ciertos sectores en las imágenes que van llegando que obedezcan un patrón preestablecido (reconocimiento de patrones). Para más información puede consultarse en la sección de esta memoria dedicada a tal efecto.

En definitiva, como decodificador lo único que se necesita es un fichero ejecutable que implemente un decodificador H.263 cualquiera e invocarlo con la siguiente configuración:

tmndec32.exe -o6 192.168.11.33 out.yuv

En dicha invocación se especifica la dirección IP asignada estáticamente en el DSP y el nombre del fichero donde se grabará la captura. Si se desea, es posible utilizar el decodificador con el que hemos estado trabajando todos estos meses. Para encontrarlo, simplemente hay que buscar en la ruta siguiente:

C:\CCStudio_v3.1\MyProjects\h263\host_app\tmndec32.exe

Tras su ejecución, ambos programas (codificador y decodificador) deberán atarse mediante protocolo TCP/IP y empezar a transmitir y reproducir video respectivamente. De forma automática, aparecerá en la pantalla una ventana en la que podrá verse la recepción en tiempo real de la secuencia de imágenes, así como un fichero temporal donde se irá almacenando el video para su posterior visualización (si se requiere). A modo ilustrativo se adjunta una captura donde se ve la captura de video en tiempo real:

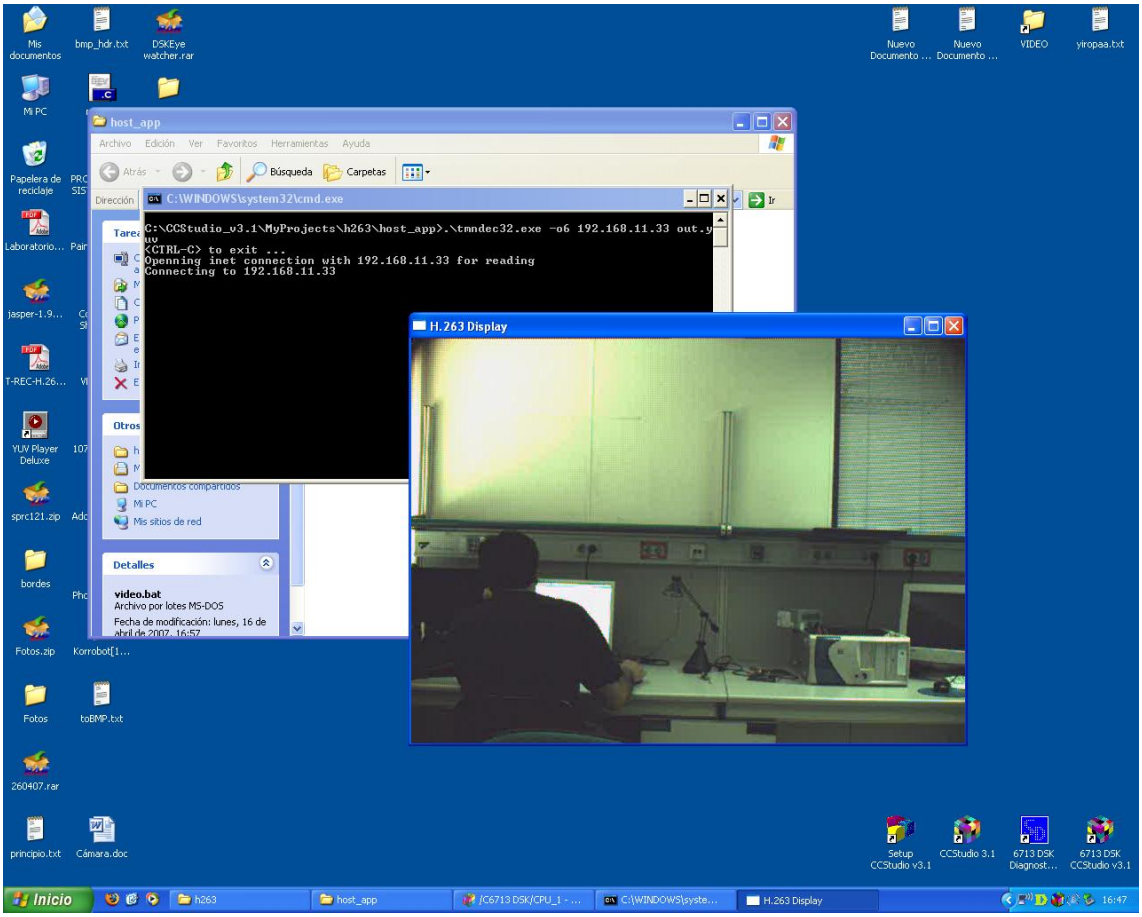


Figura 60: Visualización de video

Para acabar la reproducción simplemente deberemos cerrar la ventana de imagen y la ventana de consola que está situada por detrás.

Presentación de resultados, rendimiento y puntos críticos

Uno de los principales problemas a los que nos hemos enfrentado ha sido la limitación dispuesta por la cámara DSKEye en lo referente a resoluciones de captura. La cámara está preparada únicamente para trabajar de forma optimizada con reproducción de video a resoluciones de 640 x 480 y 320 x 200 píxeles.

Precisamente por este motivo no nos ha sido posible fijar ni probar los tamaños reales que el estándar H.263 fija de antemano (para consultarlos, ver tabla de formatos en la sección del codificador INTER). A día de hoy, únicamente se ha podido probar la aplicación del codificador INTRA a resoluciones de 640 x 480 y a 320 x 200, que corresponden con los formatos ópticos de la DSKEye VGA y HF.

Con la resolución menor se puede observar una correcta visualización temporal de la secuencia de video en tiempo real, alcanzándose de forma completa velocidades y tasas de refresco de entre 26 y 30 fps.

Con la resolución VGA no ocurre lo mismo, limitándose el framerate a 15 fps en el caso mejor. A este último modo de ejecución le afecta muchísimo la pérdida de paquetes durante la comunicación, debido a que prácticamente el 80% de las interrupciones se producen mientras se envía información de imagen (componentes codificadas que forman parte de la propia imagen, excluyendo por tanto cabeceras e información de sincronía temporal).

Además, el ancho de banda teórico del canal (1,38 MBps) se ve reducido de forma considerable conforme aumenta la distancia entre el DSP y el receptor inalámbrico D-Link. Este factor unido a la pérdida de paquetes hacen que este tipo de reproducción no sea la adecuada para mantener una visualización en tiempo real, sino para buscar y extraer datos de la propia imagen (por ejemplo, para poder averiguar la posición y orientación de un grupo de barcos situados en un canal). De hecho, como la resolución a la que se trabaja es muy buena y el grano de punto es bastante fino, la nitidez con la que se reciben las imágenes es óptima para futuros análisis que requieran precisión.

¿Por qué se producen pérdidas de paquetes?

Principalmente por falta de sincronía y por el ruido externo al que se someten las microondas que transportan la información a través del aire. En este proceso intervienen mucho las interferencias sufridas entre varios canales de emisión provenientes de otras fuentes de datos.

¿En qué se ve reflejado a la hora de utilizar la aplicación?

Si se produce pérdida de paquetes el video deja de tener continuidad y se producen parones o congelaciones de imagen debido sobre todo a que como la reproducción de video es en tiempo real y el decodificador va recogiendo tramas conforme van llegando, si alguna de éstas se pierde la reproducción se ve muy afectada, perdiéndose incluso la sincronía debido a la activación del time-out en las funciones correspondientes a la librería Winsock de Windows (que es la que permite que el decodificador pueda establecer una comunicación vía TCP/IP con el servidor de imágenes).

Tras un número considerable de pruebas de rendimiento, podemos ofrecer una gráfica en la que se muestra la relación entre distintos fps alcanzados en ejecuciones sucesivas tanto a resoluciones bajas como altas. Por cada resolución se probaron distintas distancias de referencia todas ellas acotadas entre [0,5 – 5] metros y por cada distancia tres pruebas de test, cuyos resultados han sido promediados.

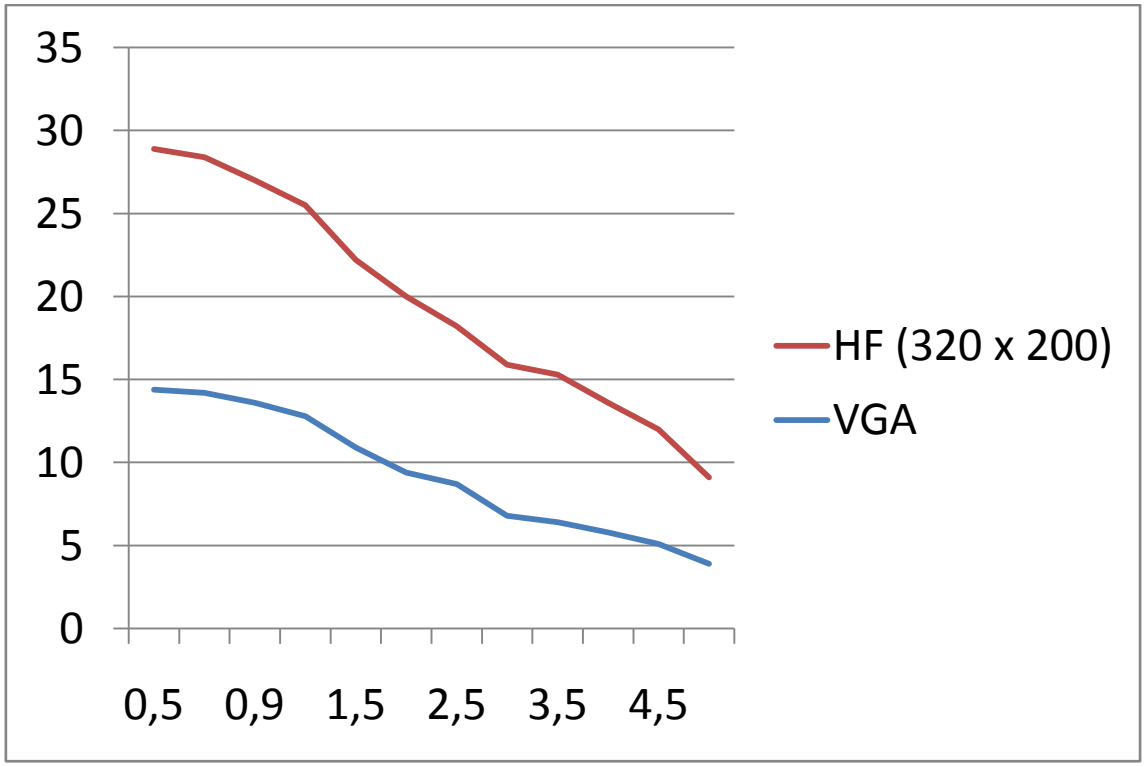


Figura 61: Distancia x fps

En ella pueden comprobarse los efectos que produce la falta de sincronía causada por la pérdida de paquetes.

En la siguiente gráfica se observa el ratio de compresión conseguido utilizando diferentes matrices de cuantificación, así como el rendimiento en fps obtenido tanto a resoluciones bajas como altas.

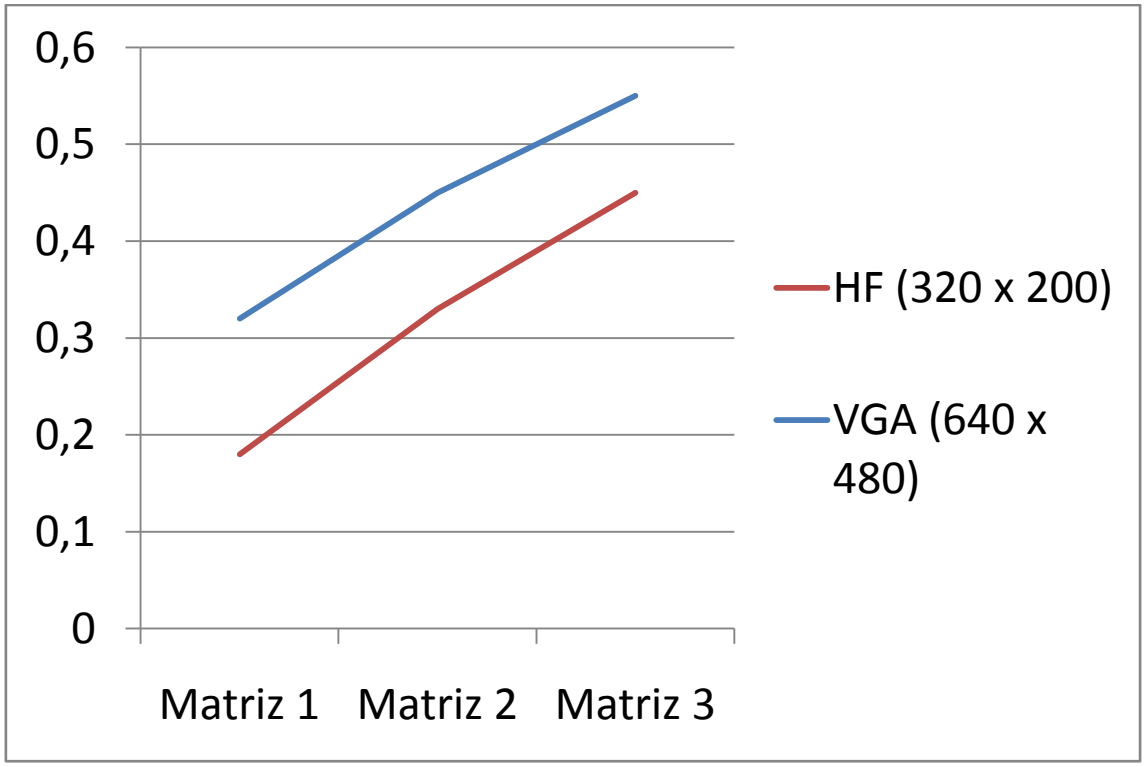


Figura 62: Quantización x Ratio

Las matrices de cuantificación utilizadas en el estudio han sido las siguientes:

$$Matriz\ 1 = \begin{bmatrix} 128 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \end{bmatrix}$$

$$Matriz\ 2 = \begin{bmatrix} 128 & 124 & 112 & 104 & 96 & 83 & 69 & 64 \\ 124 & 112 & 104 & 96 & 83 & 69 & 64 & 64 \\ 112 & 104 & 96 & 83 & 69 & 64 & 64 & 64 \\ 104 & 96 & 83 & 69 & 64 & 64 & 64 & 64 \\ 96 & 83 & 69 & 64 & 64 & 64 & 64 & 64 \\ 83 & 69 & 64 & 64 & 64 & 64 & 64 & 64 \\ 69 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \end{bmatrix}$$

$$Matriz\ 3 = \begin{bmatrix} 128 & 124 & 120 & 116 & 112 & 108 & 104 & 100 \\ 100 & 120 & 116 & 112 & 108 & 104 & 100 & 96 \\ 120 & 116 & 112 & 108 & 104 & 100 & 96 & 92 \\ 116 & 112 & 108 & 104 & 100 & 96 & 92 & 88 \\ 112 & 108 & 104 & 100 & 96 & 92 & 88 & 82 \\ 108 & 104 & 100 & 96 & 92 & 88 & 82 & 76 \\ 104 & 100 & 96 & 92 & 88 & 82 & 76 & 70 \\ 100 & 96 & 92 & 88 & 82 & 76 & 70 & 64 \end{bmatrix}$$

Con respecto a la codificación INTER, debido a la falta de tiempo para poder depurar el codificador, se producen errores durante el envío de coeficientes de la imagen y por tanto no se muestra con normalidad el video, lo que implica no poder mostrar ningún tipo de tabla ni resultado.

Por último, simplemente reseñar que para poder calcular los fps se han utilizado las herramientas de debug diseñadas por Karl Olav Lillevold y Robert Danielsen e implementadas en el decodificador base que hemos utilizado y para calcular el ratio de compresión se necesitó habilitar el modo traza continua de ejecución en el decodificador de referencia.

Los casos en los que la reproducción se paró automáticamente por falta de sincronía no han sido contabilizados para su estudio, con la finalidad de mostrar fielmente resultados numéricos lo más exactos posible.

Cuellos de botella y posibles soluciones

Por lo que hemos podido comprobar hasta la fecha, el principal cuello de botella al que nos enfrentamos no es el tiempo de procesamiento sino el ancho de banda alcanzado por la red inalámbrica.

Existen diversos factores que lo limitan muy por debajo del ancho de banda ideal de 11 Mbps (1,38 MBps), como son el ruido y el radio de recepción. Si hubiésemos podido contar con un emisor capaz de trabajar según el estándar ISO 802.11g que puede alcanzar velocidades de hasta 54 Mbps (6,75 MBps) se podrían haber conseguido unos resultados mucho más logrados para resoluciones más altas, como VGA.

Sabemos que el problema principal se encuentra en el canal porque estudiando las gráficas anteriores en las que se relacionan ratios de compresión con resoluciones de imagen por un lado y velocidades de muestreo a distintas distancias para determinadas resoluciones por otro, se pueden sacar conclusiones bastante importantes.

Trabajando a cualquier resolución, pero especialmente en VGA, los tiempos de respuesta medidos en fps tienen una relación inversamente proporcional al tamaño en píxeles de la imagen, es decir, conforme aumentamos el tamaño de la misma en teoría el proceso de codificación debería verse afectado, tardando más tiempo.

En la práctica, el proceso interno de codificación tarda menos en relación al tiempo de envío de coeficientes codificados por Huffman, es decir, que se pierde mucho más tiempo en el envío de los coeficientes Huffman que en el procesado de tramas INTRA. Para poder calcular las tasas de rendimiento ideales se utiliza un cálculo bastante sencillo que puede resumirse en la siguiente expresión:

$$fps_{ideales} = \frac{vel_{canal}}{ancho_{imagen} * alto_{imagen} * bits_{pixel}} =$$

$$= \begin{matrix} \nearrow \\ \searrow \end{matrix} \begin{matrix} \frac{11 * 1024 * 1024}{640 * 480 * 8} = \frac{11.534.336}{2.457.600} = 4{,}69\ (VGA) \\ \frac{11 * 1024 * 1024}{320 * 200 * 8} = \frac{11.534.336}{512.000} = 22{,}53\ (HF) \end{matrix}$$

Una vez calculados los fps ideales, se comparan con los fps reales y se puede comprobar que las diferencias entre ambos son bastante palpables. De hecho, es necesario tener en cuenta el factor de compresión que, dado que el proceso de codificación es lineal, simplemente hay que dividir los fps ideales entre el ratio de compresión para averiguar que tasa en fps se obtiene con distintos niveles de cuantificación (utilización de diferentes matrices de cuantificación).

Resumiendo, cuantos más datos se envíen mayor retardo sufrirá el proceso de decodificación al tener que esperar la recepción de coeficientes. Por tanto, el cuello de botella principal es el ancho de banda efectivo del canal inalámbrico. En la siguiente gráfica, se muestra la relación existente entre los tamaños de imagen (VGA y HF), la tasa en fps alcanzada y el ratio de compresión obtenido:

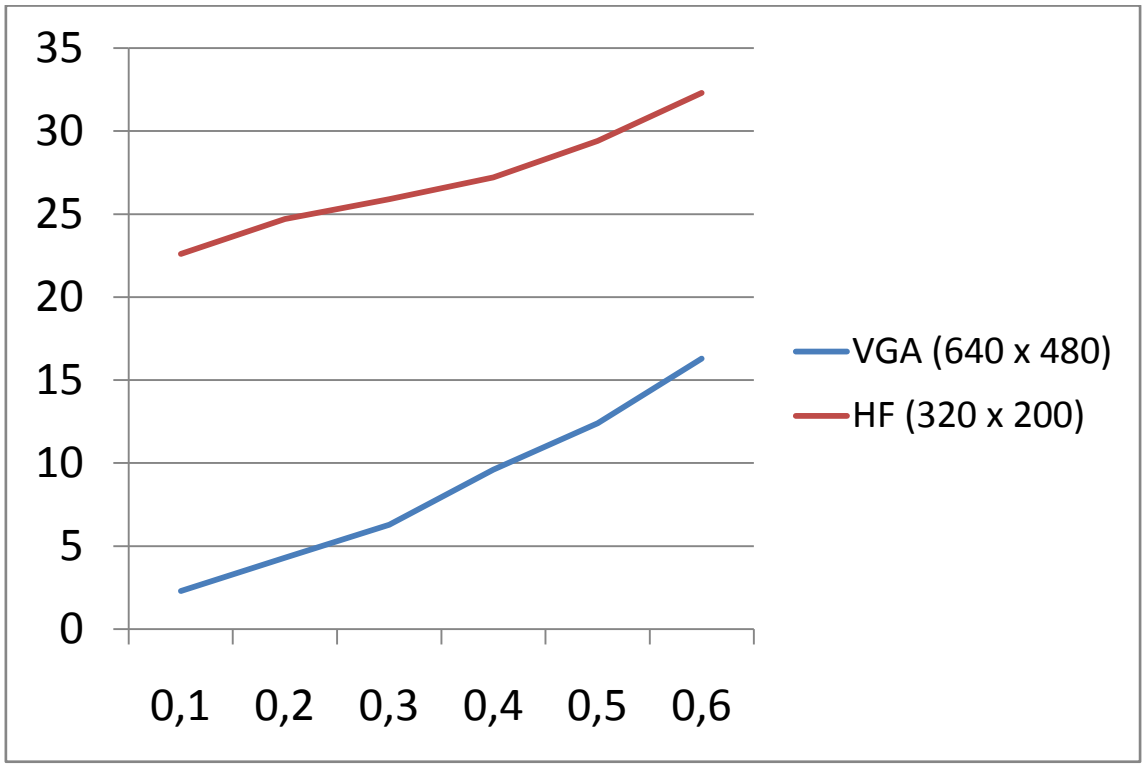


Figura 63: Ratio x fps obtenidos

Mejoras o ideas principales a desarrollar en un futuro

En primer lugar, sería interesante poder reprogramar la FPGA para que permitiese trabajar con tamaños ajustados al estándar H.263, como son el SQCIF, QCIF, CIF, 4CIF y 16CIF detallados con anterioridad. Este proceso puede ser difícil y llevar mucho tiempo de estudio, pero es necesario si de verdad se quiere trabajar cumpliendo cada una de las normas de este estándar.

Otra de las tareas podría enfocarse a solucionar los problemas de comunicación que posee el codificador INTER, que aunque funcione, no envía los datos correctamente porque se producen errores de desbordamiento y anidamiento de memoria que deberían ser estudiados con más detenimiento ayudándose con las herramientas de las que el CCStudio dispone, principalmente su modo Debug.

Una vez solucionados estos problemas, sería interesante poder trabajar con el codificador INTRA añadiéndole todas y cada una de las herramientas y funciones de manipulación de imágenes estáticas que se han desarrollado en la tercera parte de este proyecto, con el fin de poder obtener información en tiempo real acerca de bordes y posición de barcos en el canal.

Otras posibles vías de desarrollo es actualizar los algoritmos e incorporar los últimos avances desarrollados por diferentes grupos de investigación:

Evolución del H.263

Actualmente el grupo de estudio 15 de UIT trabaja en el sistema que le sucederá. Existen dos líneas diferentes de estudio, a corto y largo plazo, denominadas H.263+ y H.263L.

H.263+

El objetivo de H.263+ es actualizar a corto plazo el estándar H.263, con el fin de aumentar la calidad de sus aplicaciones y permitir otras nuevas. Las mejoras previstas son:

1. Aumento de la eficiencia de compresión.
2. Reducción del retardo.
3. Mayor protección frente a errores del canal.

El modelo actual para H.263+ mantiene el codificador base H.263 y sus cuatro modos opcionales e incorpora otros tantos, que se indican a continuación:

1. Modo INTRA avanzado (Advanced Intra Coding Mode): En él se emplea predicción dentro de la propia imagen para los macrobloques INTRA.
2. Filtro de eliminación de bloques (Deblocking Filter Mode): Utiliza un filtro paso bajo alrededor de las fronteras entre bloques de 8x8, con el fin de reducir su visibilidad. El filtro está sUITado dentro del lazo de predicción.
3. Bandas estructuradas (Slice Structured Mode): SustUITye el concepto de grupo de bloques (GOB) por el de banda o slice, definida como en MPEG (una banda de imagen de longUITd arbitraria), pero permite el reordenamiento de slices para su transmisión. El objetivo es mejorar la resistencia frente a errores.
4. H.263+ hace posible, además, tamaños de imagen arbitrarios e incorpora una funcionalidad extendida que posibilita la congelación de la imagen o de parte de ella, con un escalado opcional de la parte congelada.

H.263L

H.263L es un grupo de trabajo a más largo plazo. Sus objetivos son muy sencillos:

1. Avanzar aún más en la calidad subjetiva a bajas velocidades.
2. Incrementar la robustez frente a errores en canales ruidosos.
3. Avanzar en los requisitos de retardo y complejidad.

El grupo está centrado en servicios conversacionales en tiempo real. Este último detalle es la principal distinción del H.263L respecto al MPEG-4, cuyo campo de actuación es más amplio. Por lo demás, el trabajo de ambos grupos está bastante solapado, por lo que se espera una profunda colaboración entre ellos.

Debido al objetivo específico de comunicación conversacional en tiempo real, los requisitos impuestos a H.263L mantienen una escala de prioridades:

1. El tipo de vídeo más importante es el de escenas de busto parlante o de grupos de gente.
2. La calidad de la imagen debe favorecer tareas como:
    a. El reconocimiento de caras y expresiones.
    b. La comunicación por lenguaje de signos.
3. El retardo debe ser bajo (150/250 ms).
4. Deben existir medios de detección, ocultamiento y recuperación de errores.

Conclusiones

Tras la finalización del proyecto, las conclusiones que se pueden extraer directamente del desarrollo de las aplicaciones de procesamiento y tratamiento de imágenes en movimiento son bastante directas, englobando por un lado el rendimiento del DSP y por el otro las dificultades técnicas a las que nos hemos visto enfrentados.

Por un lado, podemos afirmar que la capacidad de procesamiento del DSP es bastante avanzada en lo que a cálculo en punto flotante se refiere, pero quizá una arquitectura especializada en cálculo de enteros hubiera venido mejor para este apartado del proyecto, ya que el proceso de codificación de video bajo el formato H.263 se basa casi en exclusiva en operaciones de manipulación y cálculo de enteros.

¿Qué beneficios se obtendrían si se utilizase una arquitectura de cálculo basada en enteros?

1. Mejor tratamiento de operaciones sobre vectores o matrices: En el proceso de codificación se hace referencia a multitud de tablas y matrices de datos enteros, como las matrices de cuantificación, las tablas Huffman, las tablas VLC, etc…
2. Mejor comportamiento ante códigos con estructura no lineal: Sobre todo en el proceso de compensación de movimiento y predicción de tramas se efectúan gran cantidad de saltos condicionales y bucles de control, estructuras para las que el DSP no está del todo optimizado.

De todas formas, hay que tener en cuenta que para el procesamiento de video se han desarrollado dos herramientas, una para codificación espacial y otra para codificación espacio-temporal de imágenes en movimiento. Sobre la primera se puede asegurar su correcto funcionamiento y el bajo coste temporal que supone el proceso de codificación, dado que la secuencia de acciones es bastante lineal. En cambio, para la codificación espacio-temporal se han producido diversos problemas relacionados con el tamaño del código y su volcado en memoria de instrucciones (en el proceso de

compilación se genera un código excesivamente grande para que entre en memoria RAM de instrucciones, por lo que se tuvo que tomar la decisión de que se volcase en la RAM dinámica).

En definitiva, se podrían resumir brevemente las principales ventajas y desventajas de cada uno de las aplicaciones desarrolladas:

1. Codificación espacial: Indicada en exclusiva para manipulación de tramas, como por ejemplo, detección de bordes, reconocimiento de patrones y colores, visión artificial y demás tareas relacionadas. El porqué es bastante sencillo: se dispone en todo momento de la totalidad del frame decodificado, por lo que es posible aplicarle cualquier tipo de algoritmo de entre los desarrollados en la fase de detección y reconocimiento de bordes y LED´s. Por tanto, queda abierta una puerta muy importante que debería ser desarrollada para potenciar e incrementar la funcionalidad que el DSP puede ofrecernos.
2. Codificación espacio-temporal: Debido al corto tiempo del que hemos dispuesto para desarrollar esta parte del proyecto, no se ha podido optimizar de forma adecuada varias de las fases de codificación, en concreto las relativas a la generación de vectores de movimiento y tramas PB. Por tanto, una de las primeras tareas que deberán llevarse a cabo será la de optimización y testeo de esta aplicación, con la única motivación de intentar aprovechar el DSP como procesador de video de alto rendimiento. Esta última aplicación es importante dado que pueden llegar a obtenerse resultados más que interesantes en este campo, teniendo en cuenta que aunque sea posible efectuar cálculos y operaciones sobre los frames recibidos por el DSP siempre se obtendrá menor rendimiento dado que será necesario esperar un cierto tiempo hasta poder recoger suficiente información sobre la imagen para luego ser procesada, con el consiguiente retardo en la generación de resultados.

Detección de bordes y LEDs

¿De qué partimos?

Para solucionar esta parte del proyecto contamos solamente con el conocimiento teórico de algunos algoritmos que nos podrían ser útiles para solucionar los problemas que se nos plantean. Buscando en la extensa bibliografía que existe sobre tratamiento digital de la imagen, se pueden encontrar infinidad de opciones para manejar las imágenes que captamos. Lo importante será estudiar cuales de las opciones dan un mejor resultado con un rendimiento medianamente aceptable.

Además el conocimiento que tenemos de la cámara y de su manejo es inexistente. Necesitamos estudiar concienzudamente la documentación que acompañaba al hardware. A pesar de hacerlo repetidamente, no fuimos capaces hasta bien avanzado el proyecto de controlar los niveles de ganancia, tiempo de exposición etc…No controlar este elemento importante del proyecto ha sido muy importante para las pruebas realizadas a lo largo del mismo, ya que las imágenes tomadas tenían una saturación y un color irreal, que no eran del todo útiles para el buen desarrollo de las pruebas.

¿Cuál es el objetivo?

El objetivo final de esta parte del proyecto consiste en obtener la posición y orientación de un objeto en un plano 2D. Dicho objeto será un barco utilizado en otros proyectos de investigación. En él acoplaremos un par de LEDs que nos serán de utilidad en uno de los subproyectos del reconocimiento del objeto.

El primero de los subproyectos es implementar un algoritmo capaz de identificar la posición de cada LED en una imagen tomada. Vamos a comprobar hasta que distancia somos capaces de alejar los LEDs para identificarlos. También tenemos que asegurarnos de que las condiciones de luz requeridas para que al tomar la imagen, son posibles controlarlas en el laboratorio para que los LEDs sean reconocidos por el algoritmo implementado. Las condiciones que vamos a intentar simular en el laboratorio son las detectadas en el CEHIPAR (Canal de Experiencias Hidrodinámicas de El Pardo), es decir, tomar imágenes en lugares con poca luz y situar el barco sobre una superficie oscura al realizar la captura de la imagen.

El segundo subproyecto consiste en detectar tanto la posición como la dirección de los barcos, en una imagen tomada, a partir de invariantes. Esta segunda parte es independiente de la primera, es decir, el algoritmo que implementemos para cumplir con este apartado no contará con la ayuda de los LEDs, que ya habrán sido identificados anteriormente dentro de la imagen. Para esta parte no sólo es importante ver el tamaño del barco en la imagen, que depende de la distancia a la que se encuentre de la cámara, sino que también tenemos que tener en cuenta el fondo sobre el cual va a estar el barco en cada momento. Las condiciones que vamos a buscar, para las pruebas realizadas en el laboratorio, serán lo más cercanas posibles a la toma de una imagen en exteriores.

Estructura del proyecto

Como hemos comentado anteriormente, esta parte del proyecto consta de dos partes claramente diferenciadas. Decidimos dividir el tiempo para realizar cada parte en momentos diferentes para evitar que se solapen. Hemos comenzando afrontando la localización de los LEDs dentro de la imagen. Una vez localizados somos capaces de dar una posición y una dirección para el barco. En

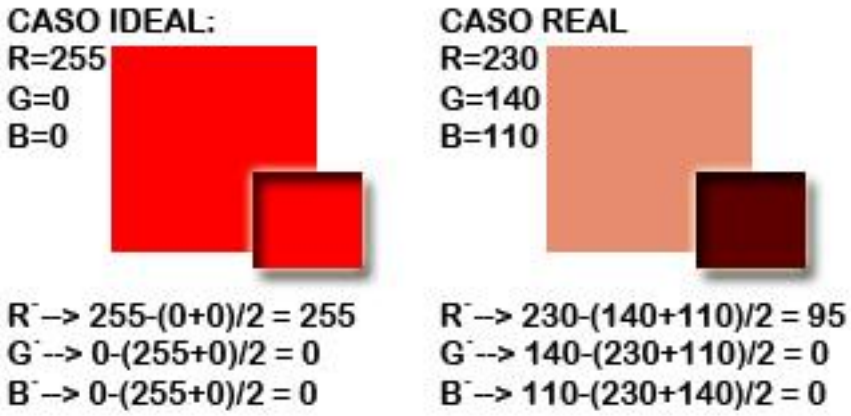
segundo lugar vamos a estudiar y a implementar algún algoritmo que nos permita localizar el barco sin tener en cuenta los LEDs. Una vez que localicemos el barco dentro de la imagen tendremos que diseñar una manera que nos permita mostrar la posición y la dirección que sigue este en cada imagen que se toma.

Después de haber sido capaces de solucionar la localización del barco, por los dos métodos comentados, hemos hecho un estudio de la eficiencia de los algoritmos obtenidos. Con esos resultados, y tras hacer un estudio de ellos, hemos intentado unir ambas opciones de localización para utilizar el método más eficiente, siempre que sea posible. Con esta idea, el resultado es intentar utilizar los LEDs para la localización, siempre que sean visibles en la imagen tomada, ya que necesita menos tiempo. Si los LEDs no están encendidos o estamos capturando la imagen en un espacio abierto, en el que no se distinguiesen los LEDs, pasaríamos a la localización mediante el otro método implementado.

Ahora vamos a mostrar las diferentes técnicas que hemos estudiado, analizado e implementado para solucionar cada uno de los problemas que se nos han ido planteando.

LOCALIZACIÓN DE LOS LEDS

La primera parte que decidimos afrontar fue la de la detección de los LEDs. Teniendo en cuenta que los LEDs no reflejan la luz, sino que la emiten, creímos que el algoritmo más conveniente para la detección de estos era restar a cada canal la media de los otros dos. Esta idea estaba basada en que los píxeles donde no estuviera el LEDs se convertirían en colores muy oscuros, mientras que en las zonas con LEDs, el canal rojo y verde tendrían valores muy superiores al resto. Idealmente el LED rojo debería tener unos valores de 255 para el canal rojo y de 0 tanto para el canal azul como para el verde. Los valores ideales para el LED verde serían similares, sólo cambiando el valor de 255 para el canal verde y 0 para el rojo y azul. Con esta idea, el algoritmo estaría claro, ya que tendríamos una imagen totalmente oscura con una zona claramente diferenciada en verde y otra en rojo muy intensas.

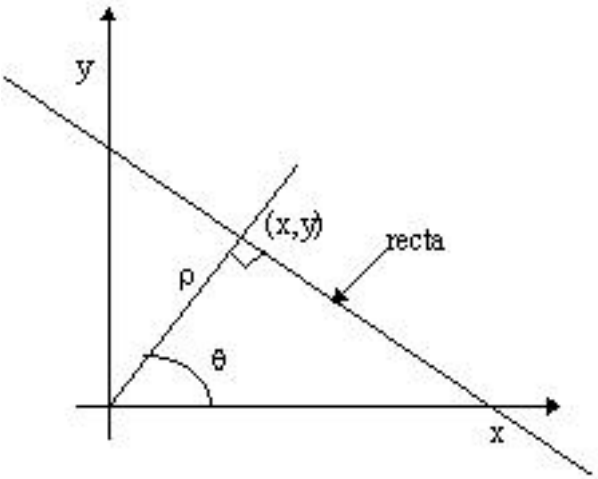


Como se observa en la imagen la idea de tener el caso ideal, al final fue muy lejana a la realidad. Los LEDs no emitían ni de una manera pura ni con una intensidad suficiente. La cámara captaba imágenes muy claras que igualaba los niveles de todos los canales demasiado, con lo cual, en la imagen final no resaltaban tanto los píxeles colocados en las posiciones de los LEDs, o directamente no resaltaba sobre el resto. Esto hizo que tras unas pruebas descartáramos este método de detección de los LEDs.

Buscando en la bibliografía dedicada a procesamiento de imagen encontramos métodos para calcular los bordes de las imágenes captadas. Hay una gran cantidad de métodos para el cálculo de

bordes. Necesitamos una implementación que no necesite muchos recursos ni un gran número de cálculos. En las siguientes líneas explicaremos los métodos implementados para hacer pruebas con ellos.

La primera opción que barajamos fue utilizar la transformada de Hough para reconocer los dos EDs, que se manifestarán, muy probablemente, en forma de circunferencia en la imagen.



La transformada Hough se utiliza para el enlace de puntos de borde y la extracción de rectas. Implica la transformación de coordenadas Cartesianas a coordenadas polares, tal y como se muestra en la figura.

Si tenemos un punto (xp,yp), podrán pasar infinitas rectas por ese punto; en el plano transformado, podemos ver esto como una función sinusoidal: $\rho=x_p \cos\theta_o + y_p \operatorname{sen}\theta_o$

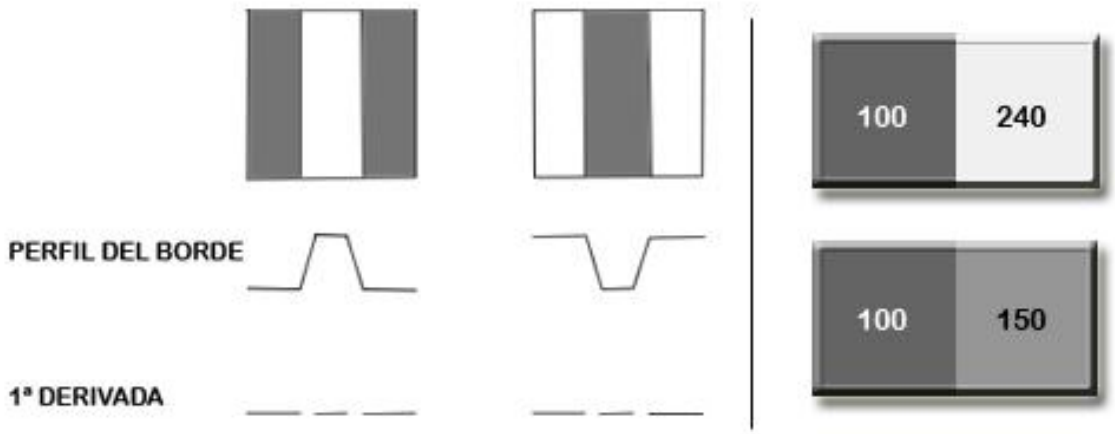
Si representamos esta función para todos los puntos de una recta en el primer cuadrante, y desechamos los valores de ρ negativos, estas funciones se cortarán todas en un punto (ρ_o , θ_o)que nos definirá la ecuación de la recta $\rho_o=x \cos\theta_o + y \operatorname{sen}\theta_o$

Para nuestro caso, lo único que deberíamos buscar es la ecuación de una circunferencia de un determinado radio, que dependería de la distancia a la que tomáramos la imagen.

Es una implementación compleja que requiere de cálculos excesivos para el objetivo de este proyecto. Como los cálculos no pueden realizarse con una precisión absoluta y son muy complejos decidimos aparcar este método que nos ayudaría a reconocer los LEDs.

Sumado al problema anterior, podemos añadir que, dependiendo de la distancia a la que se encuentren los LEDS de la cámara, la manifestación de estos elementos emisores de luz en la imagen no siempre va a ser una circunferencia perfecta, con lo que la utilización de este algoritmo estaría desaconsejada. A parte de todas las razones anteriores, la utilización de esta transformada está sujeta a derechos por parte de la familia del inventor. Esto nos complicaría el uso de esta técnica de cálculo de bordes ya que debemos garantizar su posibilidad de uso para proyectos posteriores.

Así pues, decidimos pasar a la detección de bordes mediante métodos diferenciales. En general, los bordes de objetos en una imagen los podemos distinguir por los cambios más o menos bruscos de valor entre dos o más píxeles adyacentes. Podemos hacer una clasificación de los bordes en horizontales, verticales y oblicuos según la diferencia de valor se produzcan entre píxeles, anteriores y posteriores, conectados de manera horizontal, vertical o una combinación de los dos (esta clasificación del tipo de borde no va a tener relevancia en este proyecto, sin embargo puede ser interesante su estudio para la resolución de otros problemas tales como reconocimiento de matrículas). La diferencia entre los valores de los píxeles nos indica lo acentuado del borde, de forma que a mayores diferencias tenemos bordes más marcados y a menores tenemos unos bordes suavizados, tal y como podemos observar en la parte derecha de la imagen.



Los filtros diferenciales se basan en la derivación. Como el promediado de los píxeles de una región tiende a difuminar o suavizar los detalles y bordes de la imagen, y esta operación es análoga a la integración, es de esperar que la diferenciación tenga el efecto contrario, el de aumentar la nitidez de la imagen, resaltando los bordes. La mayoría de las técnicas basadas en diferenciación se apoyan en la utilización de máscaras de 3x3. El uso de máscaras de mayor tamaño nos ayudaría a contrarrestar el efecto que tenga el ruido sobre la propia imagen. Sin embargo una matriz más pequeña tendrá un comportamiento más eficiente.

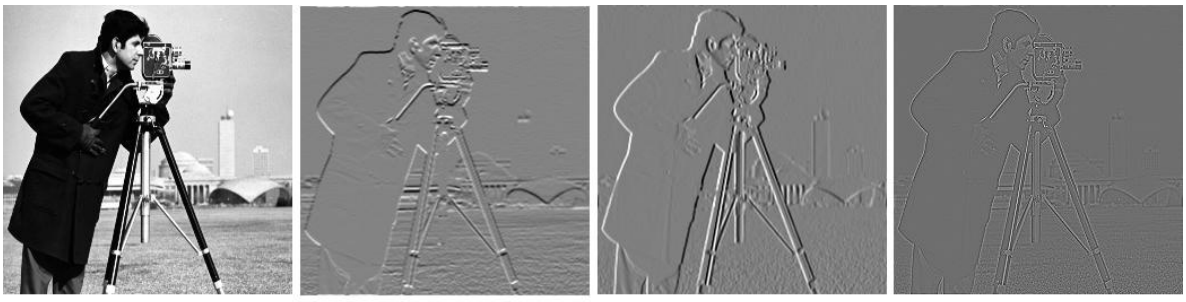
Por otro lado, las máscaras normalmente tienen tamaños impares, de forma que los operadores se encuentran centrados sobre los puntos en donde se calculan los gradientes. Los operadores de gradiente encuentran bordes horizontales y verticales. Estos operadores trabajan mediante convolución. Lo único que necesitamos es ir convolucionando la máscara a lo largo de todos los píxeles de la imagen. Al hacer esto, iremos consiguiendo en cada píxel el valor asociado al nivel de cambio de intensidad en ese píxel. Cuanto mayor sea el cambio de intensidad, tendremos un borde más marcado. Para conseguir los bordes de la imagen lo único que tenemos que hacer es decidir un valor umbral que nos dirá si un borde de los detectados es lo suficientemente marcado e intenso para los fines que necesitamos.

Los operadores de Prewitt, Sobel, Roberts y Frei-Chen son operadores dobles o de dos etapas. La detección de bordes se realiza en dos pasos, en el primero se aplica una máscara para buscar bordes horizontales, y en el segundo paso buscamos los verticales, el resultado final es la suma de ambas

etapas.

	Roberts	*Prewitt*	*Sobel*	*Frei-Chen*
Detector Horizontal	0 0 -1 0 1 0 0 0 0	1 0 -1 1 0 -1 1 0 -1	1 0 -1 2 0 -2 1 0 -1	1 0 -1 √2 0 -√2 1 0 -1
Detector Vertical	-1 0 0 0 1 0 0 0 0	-1 -1 -1 0 0 0 1 1 1	-1 -2 -1 0 0 0 1 2 1	-1 -√2 -1 0 0 0 1 √2 1

Al utilizar los operadores diferenciales del gradiente obtenemos una respuesta grande a través de un área donde un borde está presente. Para obtener una localización más correcta de los bordes es necesario usar detectores de bordes de la derivada de segundo orden. Además con los operadores vistos anteriormente las curvas que obtenemos no son cerradas, mientras que el uso de la derivada segunda para la detección de bordes nos va a devolver una curva cerrada en cada uno de los bordes que localice.



ORIGINAL SOBEL HORIZONTAL SOBEL VERTICAL 2ª DERIVADA

El operador estrella de la derivada de segundo orden, para la detección de bordes, es el operador laplaciano. Suele utilizarse en ocasiones en las que las variaciones de intensidad no son suficientes para el uso de un operador de primera derivada. Este modelo también puede ser implementado mediante convolución de máscaras. Aquí podemos ver las máscaras que podemos utilizar para calcular los bordes de la imagen mediante la utilización del operador laplaciano.

1	-2	1
-2	4	-2
1	-2	1

Laplaciano 1

-1	-1	-1
-1	8	-1
-1	-1	-1

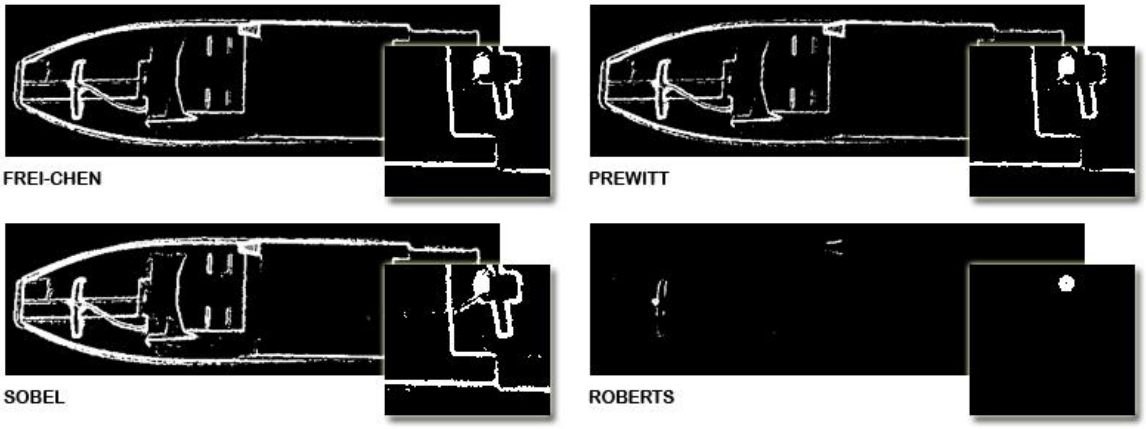
Laplaciano 2

0	-1	0
-1	4	-1
0	-1	0

Laplaciano 3

Aun pareciendo de mayor utilidad y sabiendo que la implementación mediante máscaras puede ser parecida a la anterior, decidimos no utilizar este método para el cálculo de los bordes ya que tiene tres importantes inconvenientes. El operador laplaciano es más sensible al ruido que los operadores basados en la primera derivada, genera bordes dobles y además no proporciona una información extra sobre la dirección de los ejes detectados.

Con todos estos argumentos en nuestra mano, empezamos a realizar diferentes pruebas con los diferentes operadores, de la primera derivada, vistos en esta sección.



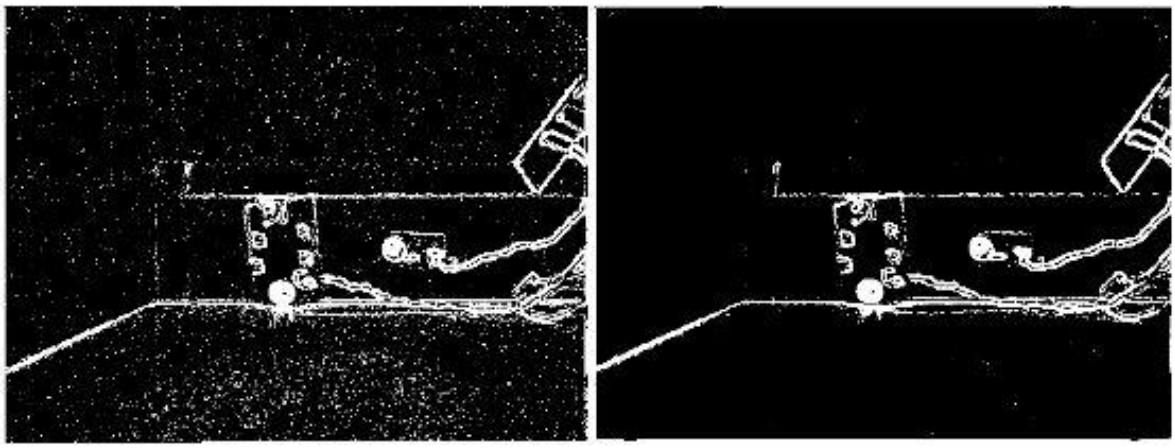
El que mejores resultados nos ofreció fue el operador de Sobel, ya que a pesar de ser más sensible al ruido, necesita una menor intensidad para reconocer un borde. Teniendo en cuenta que los LEDs no emiten con la intensidad esperada, este es el operadores el que hemos confiado. Este será el operador utilizado hasta el final del proyecto para detectar los bordes en las imágenes.

Para usar el operador de Sobel debemos pasar la imagen captada, mediante la media de los tres canales en cada píxel, de formato RGB a una imagen de intensidad. En esta imagen de intensidad es por la que tenemos que hacer la convolución de la máscara característica de este operador. Por último, si el valor obtenido tras el paso de la máscara es superior a un umbral definido, tomaremos un valor de 255 y si esto no sucede mantendremos el valor 0 en ese píxel. Con esto conseguimos la imagen binaria final.

Los resultados obtenidos en las imágenes captadas fueron imágenes binarias en las que había gran cantidad de ruido. Estudiando el problema nos dimos cuenta que, debido al método de captación de la imagen y a su posterior paso a RGB, la imagen obtenida contenía una gran cantidad de píxeles que eran tomados como ruido de la imagen. Este problema hizo plantearnos la posibilidad de realizar un

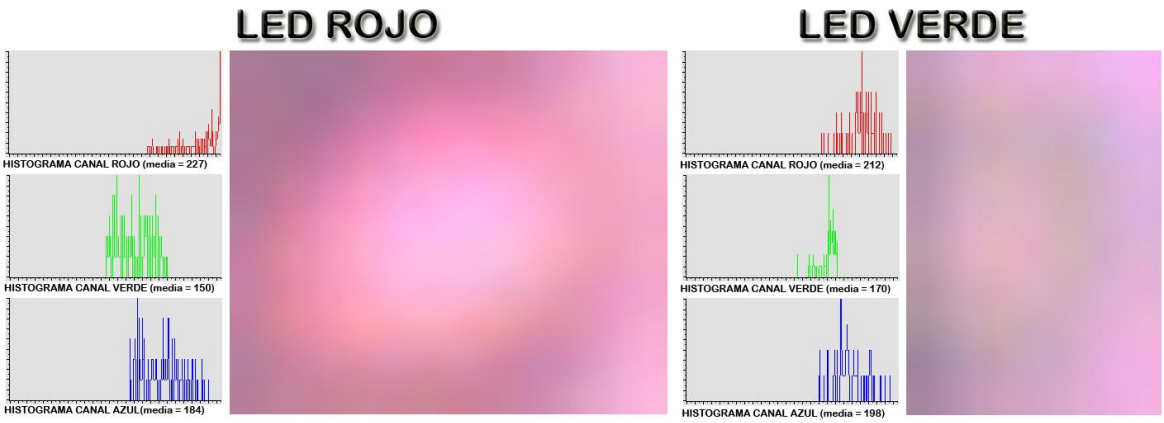
suavizado previo de la imagen, antes de su uso, ya que el cambio de operador para calcular los bordes de la imagen no solucionaba el problema.

Los dos tipos principales de suavizado para estos casos y que no requieran una gran carga computacional son el suavizado a través de la media y el suavizado a través de la mediana. Ambos métodos son parecidos. Consisten en que, para cada píxel, vamos a calcular la media, o mediana, de los píxeles adyacentes (en todas direcciones) y del mismo píxel. A pesar de que el cálculo de la media deteriora y suaviza los bordes de la imagen, la implementación de su algoritmo sobrecarga menos al sistema. Teniendo en cuenta que los bordes de los LEDs con el resto de la imagen deben ser bastante claros decidimos implementar este algoritmo.

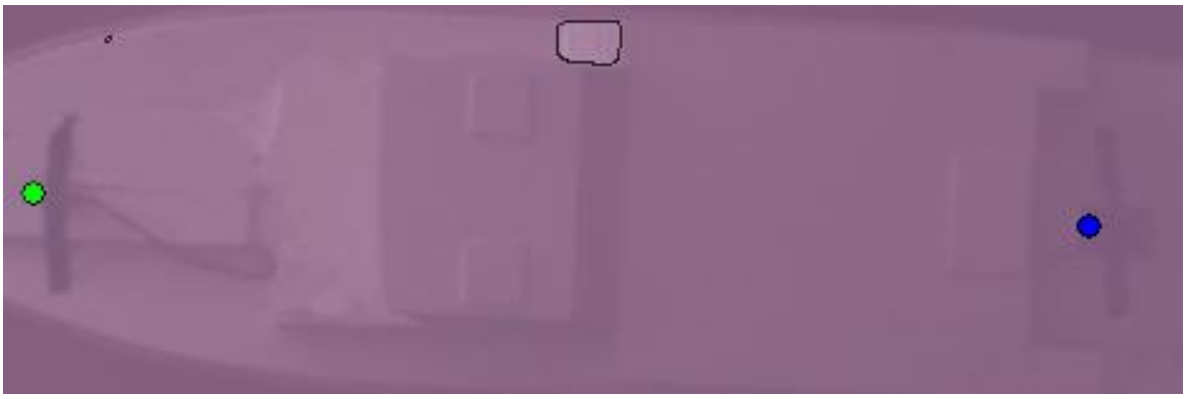


En la imagen mostrada anteriormente, que muestra los LEDs utilizados para el proyecto, observamos que para un mismo umbral para la detección de los bordes, mediante el operador de Sobel, la imagen que ha sido previamente suavizada nos ofrece una visión más limpia de los elementos de la imagen capturada. También podemos ver cómo el suavizado a través de la media no supone un gran problema a la hora de detectar los LEDs en la imagen, ya que a pesar de suavizarlos sigue permitiendo su captación

Apoyándonos en los bordes que tendremos de una imagen, ya estamos en disposición de ultimar el algoritmo que nos permita detectar completamente los LEDs. Lo único que nos falta por hacer es buscar puntos dentro de la imagen que cumplan ciertas características con respecto a los bordes y que, además, tengan unos niveles de color en cada uno de los tres canales que nos lleven a asegurar que en ese lugar existe un LED. En las siguientes imágenes se muestra un ejemplo de una imagen tomada de los LEDs emitiendo.



Para implementar este algoritmo hay que tener en cuenta la distancia a la que se encuentra el LED de la cámara, la luz que emite el propio LED y las condiciones de luz del ambiente en el que estamos tomando la imagen. Una vez planteado el algoritmo, el refinamiento va acompañado de métodos experimentales con los que ajustaremos los parámetros tanto de nivel del color que muestran los LEDs en la imagen, como de las características que debe cumplir con respecto a los bordes calculados.



En esta imagen podemos observar como en una imagen captada, el algoritmo es capaz de reconocer la posición tanto del LED rojo (círculo azul) como del LED verde (círculo verde).

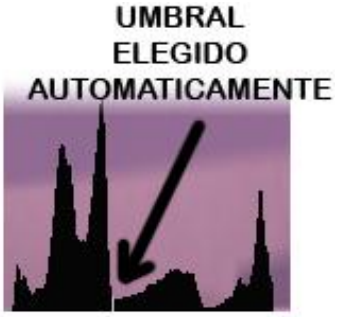
LOCALIZACIÓN DEL BARCO EN MEDIOS ABIERTOS

Tras tener solucionado el reconocimiento de los leds nos encaminamos a reconocer el barco dentro de una imagen tomada en medios abiertos. Los únicos parámetros que nos interesan conocer del barco son su posición y su dirección. Pensamos bastantes métodos para conseguir esas dos variantes de la manera más fácil. Al final nos decidimos por el uso del centro del barco como posición del mismo y por el cálculo del eje mayor y menor para controlar su dirección.

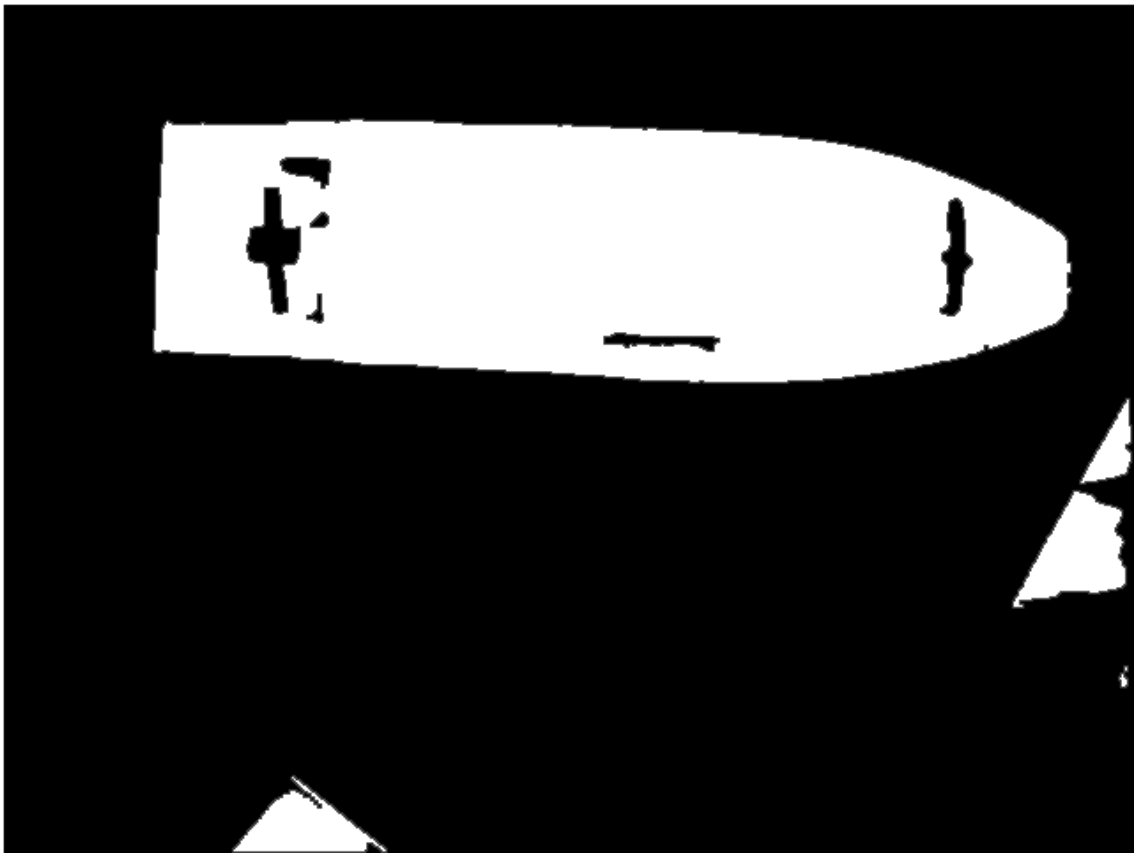
Las fotos tomadas del barco para su detección van a seguir siempre un mismo patrón. El barco, de tonos claros, siempre va a estar encima de una superficie algo más oscura como puede ser la del agua. Sabiendo esto pensamos directamente en la umbralización de la imagen para distinguir el barco del fondo. Conociendo el histograma de una imagen es muy sencillo ver el valor óptimo para realizar la selección dentro de la imagen. Con esta idea, lo primero que afrontamos fue implementar un algoritmo capaz de crear un histograma con la imagen de intensidad y decidir el umbral en el que nos apoyaremos para la umbralización de la imagen.

Para implementar la umbralización, simplemente cogemos los tres canales de la imagen y calculamos su media, para tener la imagen de intensidad. Una vez que disponemos de ella, simplemente tenemos que filtrar los valores por debajo del umbral elegido.

La tarea de elegir un umbral adecuado no es trivial, ya que será diferente en cada una de las imágenes que tomemos. Para solucionar este problema vamos a generar un histograma a partir de la imagen de intensidad calculada. Cuando disponemos de él, vamos a calcular el valor medio. A partir del valor obtenido vamos a buscar el valle más cercano, pero lo suficientemente profundo como para poder asegurar que el valor que tome el umbral en ese punto es óptimo para realizar la umbralización.



Cuando estamos en posesión del valor umbral adecuado para la imagen, tendremos que quedarnos con todo o nada. Por debajo del umbral elegiremos un valor de cero y el resto los llevaremos al blanco puro. Así formaremos una imagen binaria como la que mostramos a continuación.

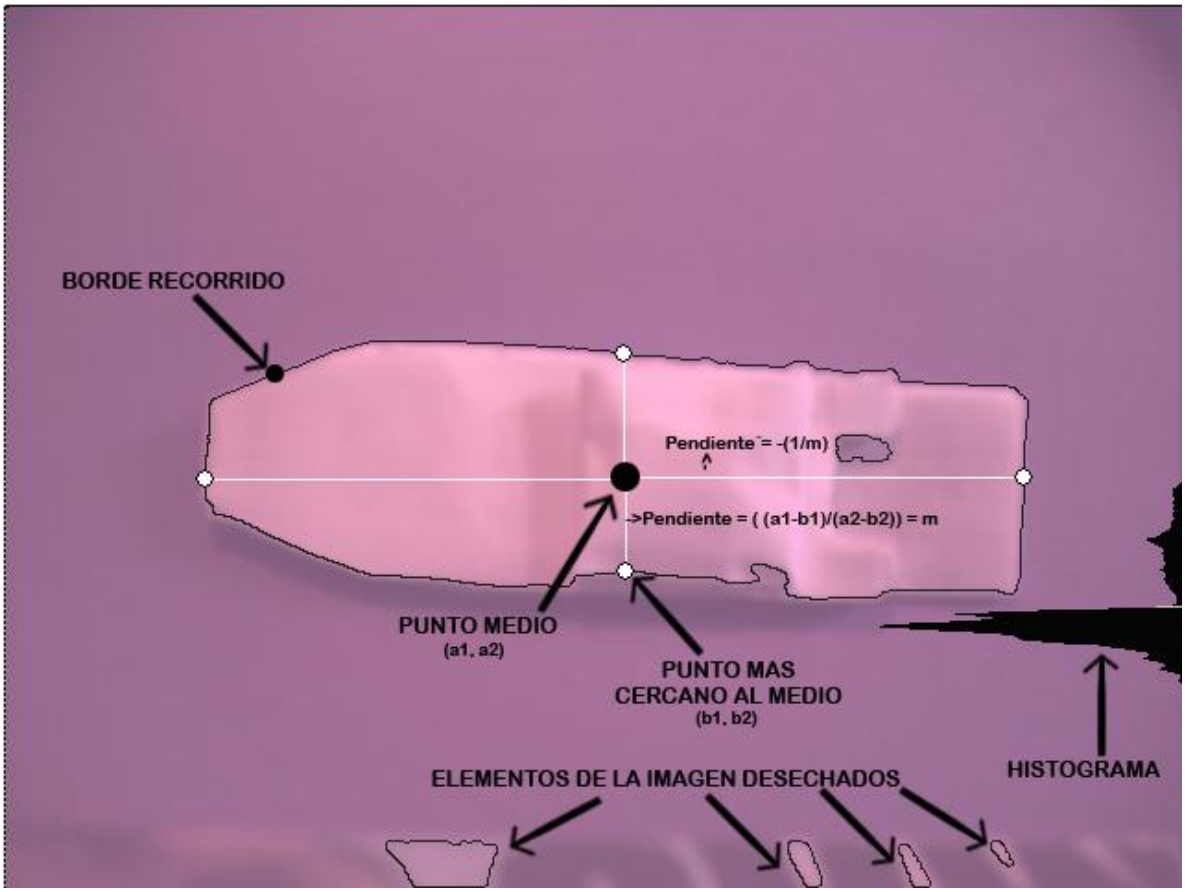


Será necesario tratar estas imágenes para evitar que los reflejos, causados por la luz, y cualquier otro elemento que haya superado la umbralización, interfieran en el reconocimiento del

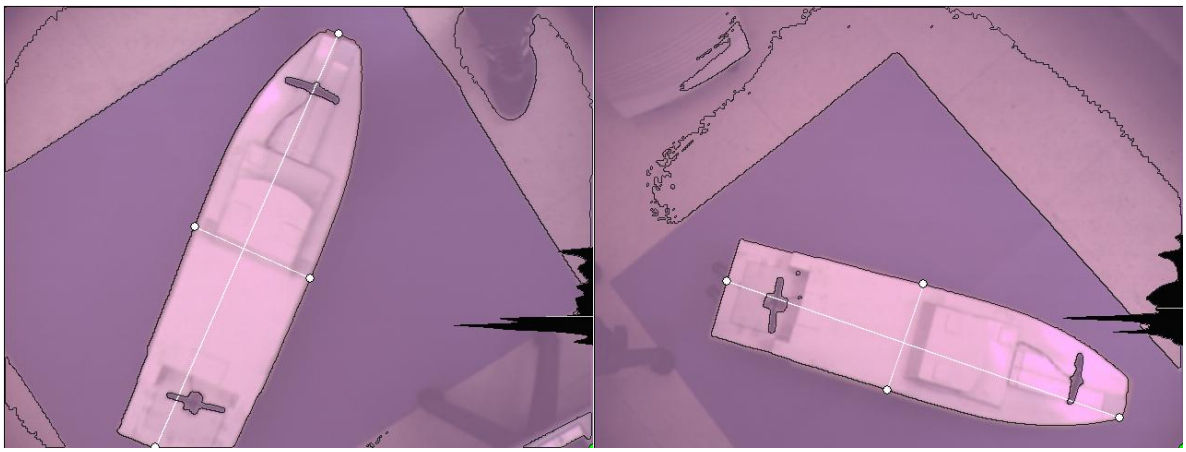
barco. El algoritmo que implementamos para tratar la imagen binaria, como preparación para calcular los ejes del barco, es hacer varias pasadas sobre la imagen eliminando puntos sueltos que aparezcan en ella o puntos que pueda unir la imagen de un reflejo con la imagen del propio barco. Una vez que trabajado con esta imagen, tendremos una o varias formas distinguibles en la misma imagen.

Si sólo destacara la imagen del barco no sería necesario hacer ninguna distinción. Sin embargo, el resto del algoritmo sólo se aplicará a los elementos que cumplan un invariante. El invariante que vamos a buscar es la excentricidad (relación entre el eje mayor y el eje menor) de cada uno de los elementos que destacan en la imagen binaria. El barco tiene una determinada relación de aspecto, que marca de manera única el valor de su excentricidad. Con esta idea, podemos decir, casi con toda seguridad, que todos los elementos cuyo valor de excentricidad sea el que buscamos será un barco. Lo único que tenemos que hacer es buscar la excentricidad de cada uno de los elementos que resaltan en la imagen y comprarlo con el valor buscado.

Con el barco ya localizado el algoritmo utilizado para calcular tanto el centro como los ejes del barco es bastante sencillo. Sólo es necesario recorrer el borde del elemento que tenemos localizado e ir calculando la media de dichos píxeles. Al final, el centro del barco será el valor resultante del cálculo de dicha media. Para los ejes buscamos el punto del borde más cercano al centro del barco que hemos hallado. Este punto siempre va a ser uno de los laterales del barco. Con esos dos puntos podemos calcular la pendiente de la recta que pasa por el centro y por el punto del borde más cercano a él. Ese será el eje menor. Para calcular el eje mayor simplemente tendremos que calcular la recta que pasa por el punto medio y cuya pendiente sea la inversa de la calculada anteriormente. En la imagen se muestra esto más claramente.

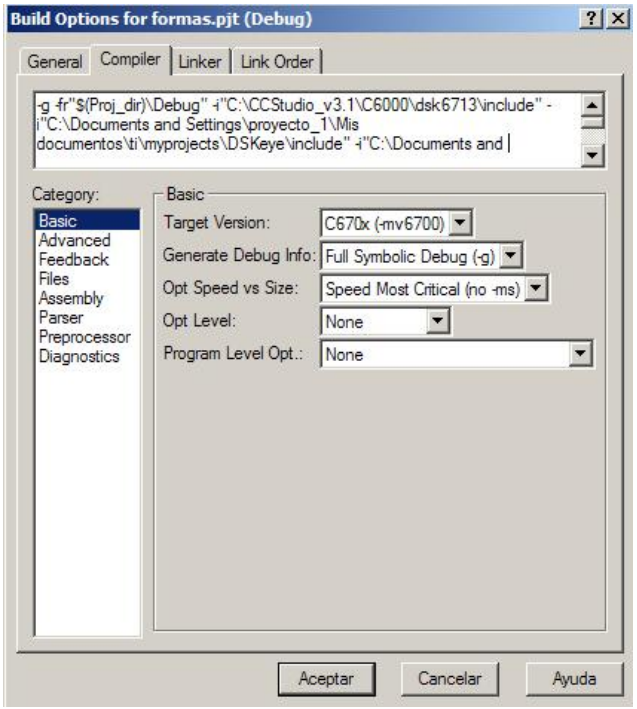


A continuación mostraremos el resultado de dos ejecuciones realizadas, mostrando el aspecto de la imagen que ha sido transmitida por el DSP en cada una de ellas y recibida por el ordenador, donde se analizarán los resultados obtenidos. En estas imágenes podemos observar el histograma estudiado, con el umbral tomado en él (muesca en blanco), los elementos reconocidos con ese umbral (bordeados en negro) y los ejes encima de aquellos elementos que cumplen el invariante.

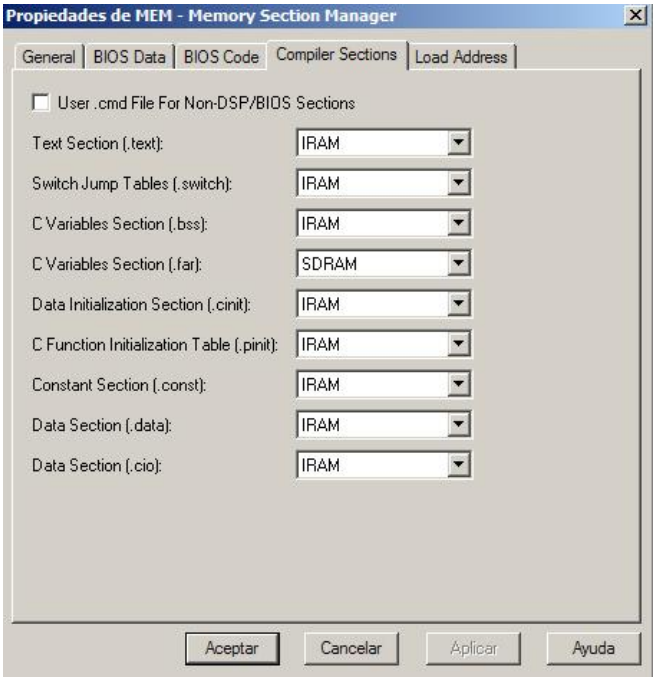


Compilación y generación de código objeto

La compilación de este proyecto se realiza de la manera usual en el CSStudio. Es una generación de código que no emite información de depuración alguna, y sin ninguna optimización. Cabe pensar que una optimización del código generado puede hacer que la eficiencia del programa aumente. Sin embargo nos ha dado muchos problemas cada vez que utilizamos un código mínimamente optimizado.

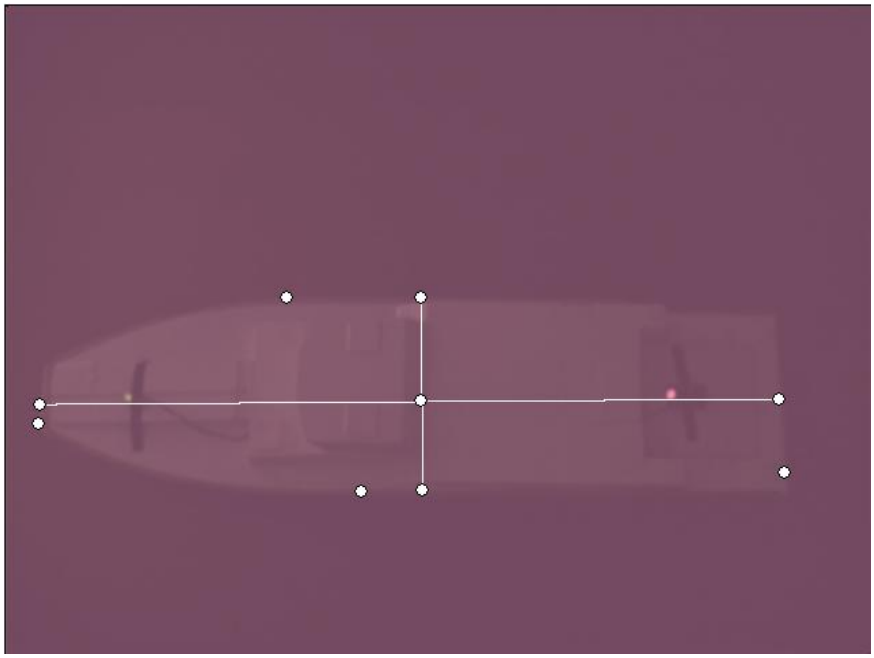


En cuanto al posicionamiento de los datos en la memoria, reseñaremos que todas las secciones, excepto las de variables están en la memoria IRAM del DSP. En los algoritmos de reconocimiento de leds y del barco necesitamos una gran cantidad de variables del tamaño de una imagen. Esta es la razón de que esta sección se encuentre situada en la memoria SDRAM



Ejecución de las herramientas desarrolladas

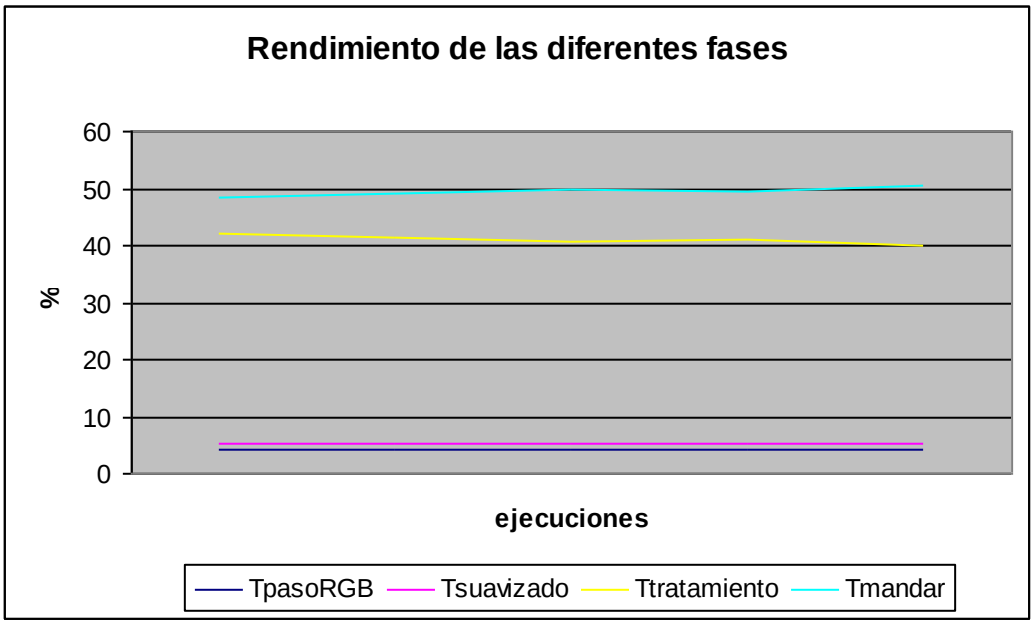
Esta parte del proyecto tiene una ejecución muy sencilla. Lo único que se necesita es conectar la aplicación que esté corriendo en el DSP con la aplicación implementada en el C++ Builder que estará corriendo en un PC. La conexión se realiza mediante WiFi. La interfaz da la orden de conexión y de captura de la imagen al DSP. Cuando éste recibe la orden, toma la imagen, la trata y devuelve los tres canales con los ejes del barco dibujados encima de él. Además una de las implementaciones realizadas no sólo muestra los ejes y el centro sino que hace saber al programa que está ejecutándose sobre el PC cual es el punto del barco situado más a la izquierda, más a la derecha, más arriba y más abajo dentro de la imagen y cueles son los puntos de corte del barco con los ejes.



Presentación de resultados, rendimiento y puntos críticos

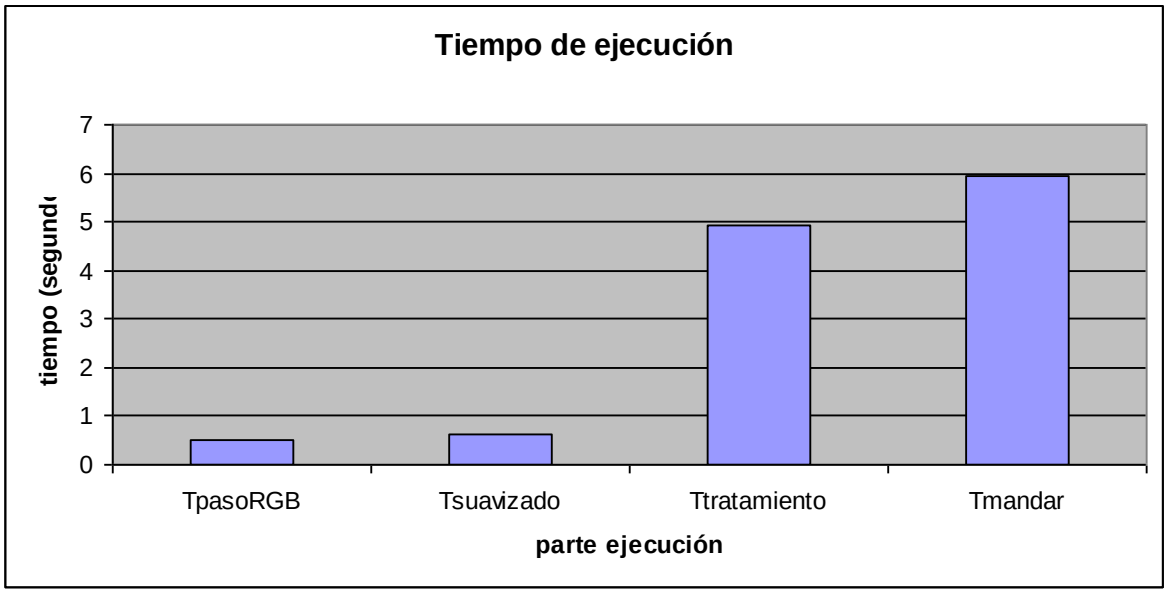
Al enfrentarnos a esta parte del proyecto, en todo momento intentamos generar un código que corriera de la manera más eficiente posible y que nos ofreciera los resultados en el mínimo tiempo posible. Tras la implementación y las pruebas nos centramos en el estudio de la eficiencia del proyecto.

Para comenzar con el análisis del tiempo que tarda en ejecutarse el algoritmo tomamos los tiempos en diferentes puntos del código para saber cuánto tiempo tardaba en ejecutar los cuatro bloques básicos del proyecto. Esos bloques son el de paso a RGB, el de suavizado de la imagen, el de tratamiento y el envío de la imagen.



En varias ejecuciones de prueba recogimos los resultados que se muestran en la gráfica anterior. Como podemos ver, el porcentaje del tiempo que consumen tanto el tratamiento como el

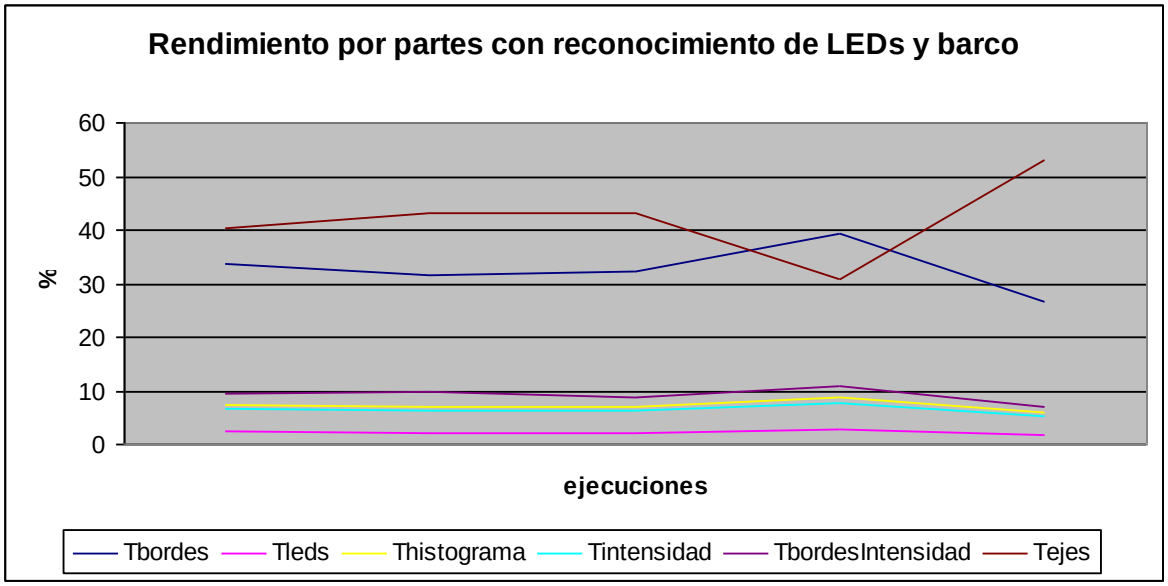
envío de la imagen es muy superior al consumido por el suavizado y el paso a RGB, llegando en algunos casos a rozar el 50% del tiempo utilizado.



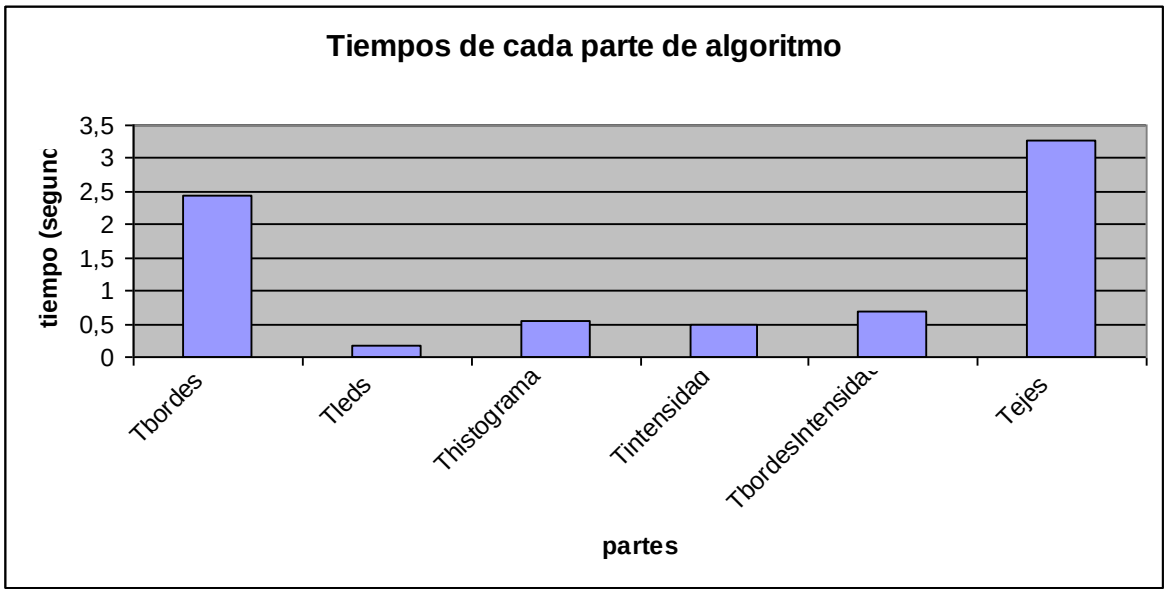
Como en los otros dos proyectos, el tiempo que tardamos en mandar una imagen es excesivo, ya que en este caso, es la parte que más tiempo consume. Mirando la gráfica, que muestra la media del tiempo consumido en las ejecuciones realizadas, podemos observar, como el tiempo de envío alcanza, prácticamente, los seis segundos. El tratamiento de la imagen, donde se calcula la dirección y la posición del barco, no llega a ocupar cinco segundos. Aún así parece un tiempo un tanto elevado. Para terminar, la suma del tiempo del paso de la imagen a RGB y del suavizado de la imagen capturada no llega al segundo.

Con estos datos y teniendo en cuenta que nos es imposible disminuir el tiempo de envío de la imagen, tendremos que centrarnos en optimizar el algoritmo de tratamiento de la imagen. Por esta razón empezamos un estudio de la eficiencia del propio algoritmo de tratamiento. Tomamos datos sobre el tiempo que tardaba en ejecutar diferentes pasos dentro del algoritmo de reconocimiento del barco.

Desglosamos el algoritmo en tiempo que tarda en calcular los bordes de la imagen, tiempo que tarda en encontrar los LEDs después de tener los bordes, tiempo que tarda en calcular y trabajar con el histograma de la imagen, tiempo que tarda en conseguir la imagen de intensidad, tiempo que tarda en calcular los bordes de esta última imagen y, por último, tiempo que tarda en calcular los ejes de la imagen, una vez encontrada la posición del barco.

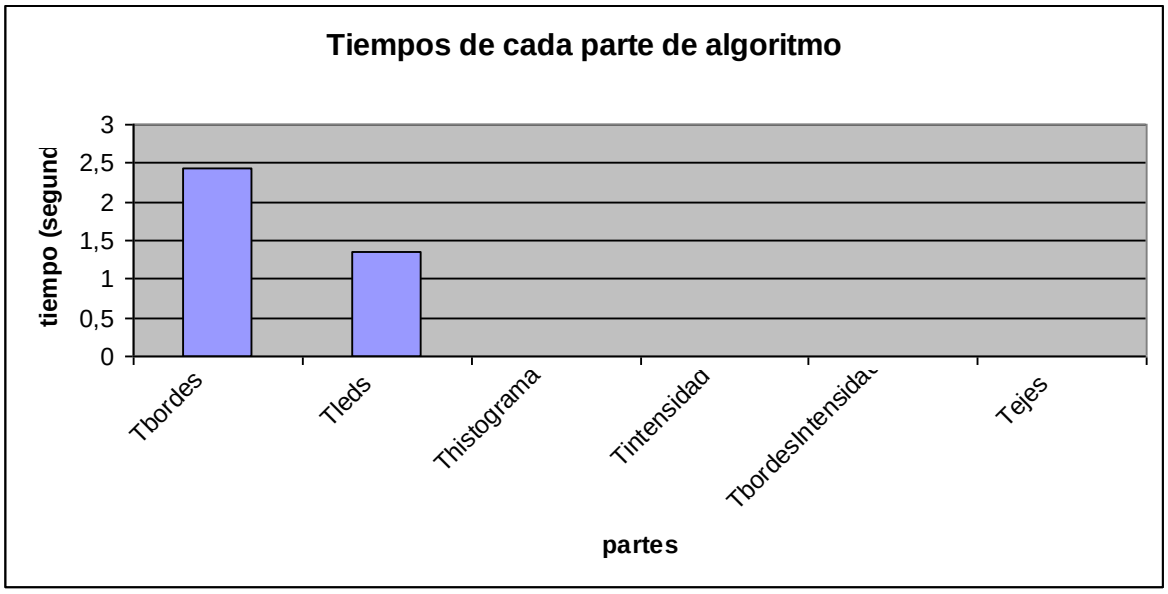


Con los datos recopilados de las diferentes ejecuciones del algoritmo hemos creado la gráfica anterior. Lo más significativo que vemos en ella es que si sumamos el tiempo que tarda en calcular los ejes y el tiempo que tarda en calcular los bordes, tendremos cerca del 80% del tiempo consumido. Las otras partes del algoritmo tiene más o menos el mismo peso. Revisando los tiempos medios reales de las ejecuciones realizadas nos damos cuenta de que tocando el algoritmo de cálculo de los bordes o el que nos da los ejes dentro de la imagen.



Tras mirar los algoritmos de estas dos partes tomamos la decisión de siempre ejecutar la parte de reconocimiento de LEDs y, tanto el cálculo del centro como de los ejes se hará en esta parte, siendo esto el final del algoritmo. La parte de reconocimiento del barco sin tener en cuenta los LEDs sólo se ejecutará si no se reconocen los LEDs. Esto evita, siempre que los LEDs del barco esten encendidos, la ejecución de la parte que más tiempo carga sobre la ejecución del algoritmo, el cálculo de los ejes y el centro del barco, dentro de la imagen.

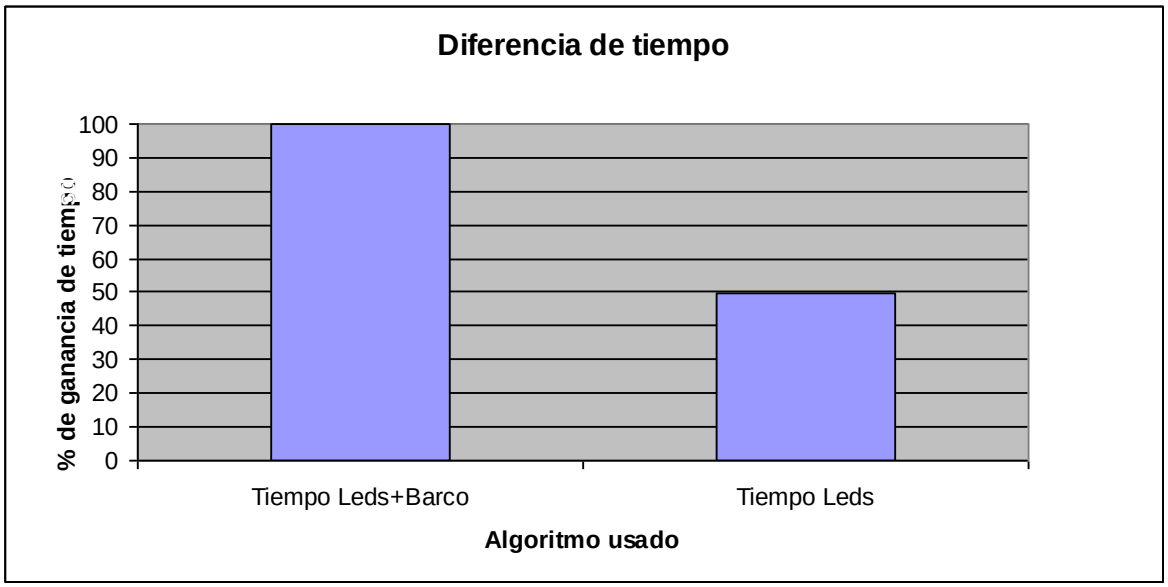
Con este cambio, conseguimos cambiar la gráfica anterior, consiguiendo la que mostramos a continuación:



El único cambio negativo provocado es el aumento del tiempo que tarda en ejecutar el reconocimiento de los LEDs. A pesar de no tocar el algoritmo, se observa un aumento de algo más de un segundo. Esto es debido al tiempo que tardamos en dibujar el eje sobre la imagen. Sin embargo, este punto no debería tomarse en cuenta, ya que, en implementaciones de uso real, no sería necesario

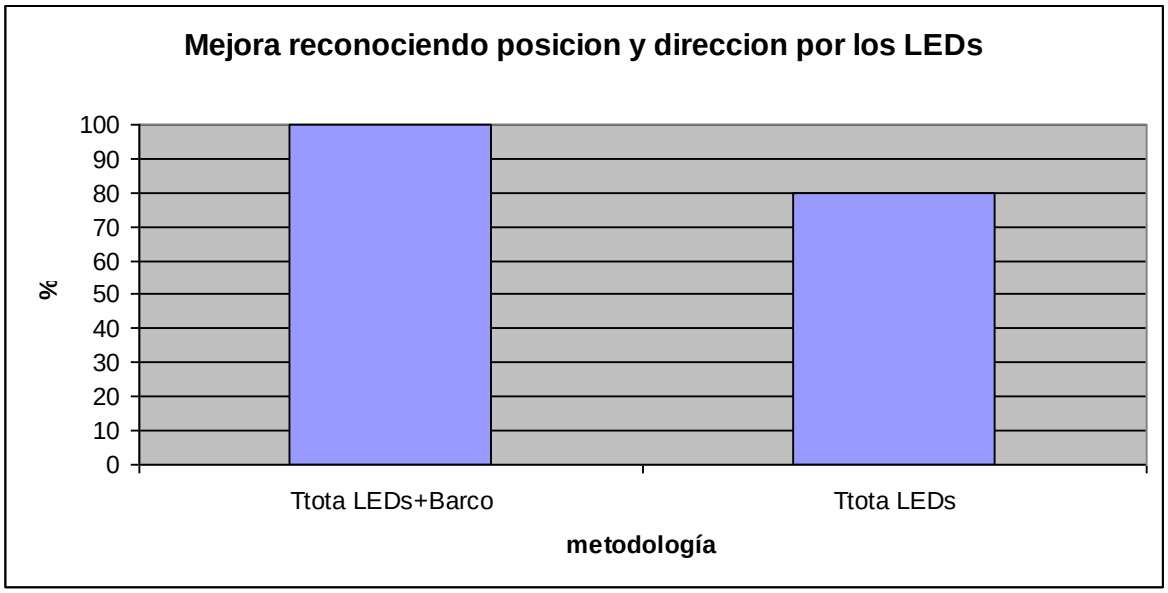
dibujar los ejes sobre la imagen, sino que bastaría con transmitir, vía WiFi, la posición del centro del barco y un parámetro que indicara la dirección.

A pesar de esto, en estas pruebas vamos a mantener estos resultados. Así, la ganancia en eficiencia del algoritmo puede verse en el siguiente gráfico bastante claramente.



Como observamos, el tiempo de ejecución es del 50% al que teníamos antes del cambio. Es decir, lo que antes del cambio tardaba una media de 7,6 segundos ahora tardará poco más de tres segundos y medio.

Si los datos conseguidos en este último estudio lo llevamos a los datos generales, obtenemos unos datos bastante interesantes.



Podemos ver que simplemente con el cambio que hemos hecho, el tiempo total se reduce hasta conseguir estar por debajo del 80% de lo que teníamos en un principio. Esto supone pasar de un tiempo de ejecución medio de doce segundos a tener ejecuciones completas de nueve segundos y medio.

Cuellos de botella y posibles soluciones

Creemos que tener que mandar la imagen mediante WiFi para poder contemplar los resultados y tomar decisiones acertadas es un error. Lo ideal sería aprovechar la parte del proyecto encargada de comprimir en JPGE y, simplemente, enviar al ordenador la posición y el sentido que lleva el barco en cada momento para que el controlador sea capaz de decidir. Así el tiempo se reduciría, como mínimo, a la mitad la mayor parte de las ejecuciones.

Tras suprimir la ejecución de la parte de localización del barco una vez se han detectado los LEDs, hemos reducido a la mitad el tiempo que tarda el algoritmo en ejecutarse. Ahora el mayor cuello de botella es el cálculo de los bordes de la imagen captada. Sería este punto en el que habría que centrarse para intentar conseguir un mejor rendimiento.

Resumiendo, si lográsemos enviar sólo un par de parámetros para indicar la posición y la dirección, el tiempo de envío sería prácticamente despreciable. Con esta idea, no seria necesario dibujar nada en ninguna parte del algoritmo, ni siquiera los ejes, con lo que tendríamos un tiempo de reconocimiento de los leds de menos de 0,2 segundos. Con esto, el tiempo del algoritmo, se reduciría prácticamente al tiempo de cálculo de los bordes de la imagen, con lo cual sería interesante reducir el tiempo de esta zona del código.

Mejoras o ideas principales a desarrollar en un futuro

Dentro de cada uno de los apartados de los que consta el proyecto puede que existan un par de mejoras que podrían llevarse a cabo. En el tema de los LEDs, hemos tenido muchísimos problemas con ellos. Para empezar, la intensidad con la que emiten la luz no es la esperada. El LED rojo, cuando la carga de las baterías era elevada, tiene un nivel de intensidad bastante aceptable. Sin embargo el LED verde en ningún momento emite tan intensamente como para hacer fácil su identificación dentro de una imagen captada. Otro problema grave es la longitud de onda con la que emiten. No emiten en una longitud de onda pura y eso hace que tengamos muchos problemas a la hora de afinar el algoritmo de detección de los LEDs. Una posible solución sería utilizar otro tipo de LEDs. Los usados para el proyecto suelen tener una holgura en la longitud de onda de unos 10nm, una intensidad de 11000mcd y un ángulo de visión optima de 20°. Podrían hacerse las pruebas con un LED del tipo "Seoul Z-Power LED P4 emitter" de LUMITRONIX. Son LEDs que emiten con una frecuencia pura (625nm para el rojo), que tienen un ángulo de visión de 125° y que pueden llegar a emitir con un flujo luminoso de hasta 240 lm. El precio de este tipo de LEDs es extremadamente superior al precio de los LEDs que estamos usando, pero para futuros proyectos sería recomendable hacer el gasto (8€) para facilitar el procesamiento de la imagen y para evitar hacer cálculos innecesarios causados por la poca eficiencia de los LEDs que estamos usando ahora.

Otro punto que nos ha dado problemas en el proyecto han sido las imágenes tomadas. El color que hemos conseguido captar ha sido siempre bastante irreal, oscuro y saturado. A pesar de terminar controlando algunos parámetros de la cámara como el de tiempo de exposición, entendemos que deberían existir otras serie de mejoras que hicieran las imágenes más naturales, ya que controlando el tiempo de exposición solo controlamos la cantidad de luz necesaria para conseguir captar la imagen del barco, por ejemplo, pero no logramos darle a la fotografía ni un color real ni saturado. El problema puede estar desde en la lente que usamos que no tenga más capacidad de captación hasta el algoritmo utilizado para pasar a RGB desde el flujo de datos obtenido por la cámara.

Otra posible mejora que tenemos que mencionar es la de la comunicación. El estándar utilizado para la conexión vía WiFi es un poco antiguo y da demasiados problemas, con multitud de desconexiones que pueden hacer fracasar el proyecto en el momento menos indicado. Una desconexión, no sólo supone tener que esperar a que la conexión se vuelva a establecer, sino que obliga a cargar el programa en memoria de nuevo. Cada desconexión de la red eléctrica del DSP supone una recarga del programa que se quiere ejecutar. Este es otro punto a investigar en proyectos sucesivos. Cargar el programa en memoria flash para evitar recargar el mismo programa multitud de veces o si no disponemos de red eléctrica.

Conclusiones

El DSP utilizado para este proyecto está optimizado para el trabajo, con señales digitales, consistente en cálculos con números en coma flotante, tales como señales de audio. Sin embargo, trabajando con imágenes no conseguimos utilizar toda la potencia que podría poner a nuestra disposición este dispositivo. A pesar de conseguir unos buenos resultados, el rendimiento conseguido podría verse aumentado con el uso de un DSP dedicado al tratamiento puro de imágenes.

Otro tema que cabe destacar es el de la mala calidad, en los colores, que hemos conseguido en las imágenes tomadas. A pesar de conseguir tomar las fotografías en unas condiciones de luz mucho más pobres que al comienzo del proyecto, los colores siguen saliendo muy igualados y es difícil conseguir un umbral óptimo y claro para el reconocimiento del barco, que es de color blanco, sobre fondos de colores oscuros.

Para concluir, volver a repetir la idea de aprovechar la parte de compresión de la imagen por medio de JPEG para no tener que enviar la imagen para apreciar los resultados obtenidos. Lo ideal sería mandar simplemente un parámetro que nos indicara la posición y el sentido que posee el barco en cada instante, ya que, mandar tanta cantidad de información, en las condiciones que nos ofrecen los elementos de los que disponemos para la comunicación, es un retraso en el rendimiento global de esta parte del proyecto.

Bibliografía

Documentación impresa de consulta y referencia

En este apartado se detallarán cada uno de los documentos técnicos de referencia que hemos necesitado consultar en algún momento durante el desarrollo de este proyecto. Por motivos de claridad, hemos discriminado, en la medida de lo posible, la naturaleza propia de la fuente, agrupándola en documentación técnica relacionada con el Hardware y Software utilizado en nuestro proyecto:

Hardware y fundamentos de algoritmia

- Manual de referencia sobre la cámara DSKEye: Se encontraba en formato digital en el CD de aplicaciones y ejemplos que acompañaba al dispositivo.
- Manual de referencia sobre el DSP: Nos fue entregado al mismo tiempo que la placa y desde el primer momento fue utilizado como herramienta indispensable para poder solucionar cualquier duda referente a la arquitectura del DSP.Visión por computador: imágenes digitales y aplicaciones / Gonzalo Pajares Martínsanz, Jesús Manuel de la Cruz García
- Tratamiento digital de imágenes / Rafael C. González, Richard E. Woods
- Apuntes de la asignatura de Robótica

Software

- Recomendación UIT-T para la comunicación de vídeo a baja velocidad binaria: Este documento de referencia ha sido básico para poder comprender y abstraer cada una de las fases que intervienen en la codificación de imágenes en movimiento por el algoritmo de codificación H.263.

Documentación digital

En este apartado se detallarán cada uno de los sitios interactivos a los que hemos tenido necesidad de acudir durante el desarrollo del proyecto con la finalidad de solucionar determinados problemas que han ido surgiendo a lo largo del tiempo. Hemos considerado oportuno clasificar dichas fuentes en tres grandes grupos:

Páginas y sitios web

Sobre JPEG:

- http://www.jpeg.org/
- http://www.maestrosdelweb.com/editorial/jpeg/
- http://www.jpeg.org/jpeg2000/
- http://www.jpeg2000info.com/
- http://www.ece.uvic.ca/~mdadams/jasper/

Sobre estándares:

- http://www.itu.int/net/home/index.aspx
- http://focus.ti.com/dsp/docs/dsphome.tsp?sectionId=46&DCMP=TIHomeTracking&HQS=Other+OT+home_p_dsp

- http://www.ti.com/

Sobre codificación de video:

- http://www.4i2i.com/h_263_software_video_codec.htm
- http://www.videoforums.co.uk/guide-encoding-47.htm

Repositorios digitales y bibliotecas de desarrollo

- Ayuda interactiva del CCStudio: Esta herramienta nos fue muy útil durante los meses de Septiembre a Diciembre, puesto que nos sirvió de guía completa para intentar dominar los aspectos más relevantes acerca de la funcionalidad y posibilidades que ofrecía el DSP y las herramientas software de las que disponíamos al utilizar dicho programa.
- Ayuda on-line MSDN: Microsoft posee una de las bibliotecas de consulta profesional más avanzada del momento y la herramienta de ayuda interactiva del Visual Studio nos facilitó enormemente el desarrollo de cada una de las aplicaciones cliente que hemos utilizado durante la elaboración de este proyecto.

Consideraciones legales y transferencia de derechos

Autorizamos a la universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Alberto Fernandez-Corroto de Pazos

Jorge de Iscar González

Carlos Spitzer López