ESTUDIO COMPARATIVO
DE
MARCA DE AGUA DIGITAL
EN
JPEG Y JPEG2000

Realizado por:

Francisco Cerezo Leal
Luis Felipe Macías Ramos
Sergio Salcedo Cámara

GRUPO 37

ÍNDICE

1. Resumen del trabajo 3

2. Preliminares 4

    2.1 Compresión de imágenes 5
    2.2 JPEG 6
    2.2 JPEG2000 14
    2.3 Marcas de agua 25

3. Algoritmo implementado 28

    3.1 Inserción de la marca 28
    3.2 Recuperación de la marca 36

4. Consideraciones sobre la aplicación desarrollada 39

5. Conclusiones 40

6. Bibliografía 41

7. Cuadernos de trabajo 43

1. Resumen del trabajo

Nuestro trabajo consiste en desarrollar un sistema de procesamiento de marcas de agua para imágenes digitales en JPEG y JPEG2000 que nos permita estudiar comparativamente la resistencia de las marcas de agua a los ataques realizados por uno y otro estándar de compresión.

Con esta idea, hemos desarrollado una aplicación que permite insertar y recuperar marcas sobre imágenes en esos dos formatos de compresión.

El algoritmo de procesamiento de marcas ha sido desarrollado por nosotros, basándonos en un documento realizado por diversos investigadores y que puede consultarse en la bibliografía. Para el manejo de imágenes en JPEG y JPEG2000 nos hemos servido de librerías de terceros accesibles de manera gratuita desde Internet, que se encuentran referenciadas también en la bibliografía.

Junto con la aplicación se adjunta este documento, entre otros, el cual sigue el siguiente planteamiento:

1. Resumen del trabajo
Se indica un resumen del trabajo realizado y de la documentación generada como resultado.

2. Preliminares
Se exponen detalladamente los conceptos de formatos de compresión JPEG y JPEG200 así como de las marcas de agua.

3. Algoritmo implementado
Se explica detalladamente el algoritmo de procesamiento de marcas de agua implementado, incluyéndose fragmentos de su código.

4. Consideraciones sobre la aplicación desarrollada
Se realizan algunas indicaciones sobre la manera de operar de la aplicación desarrollada y de cómo puede usarse para realizar un estudio comparativo como el que nos ocupa.

5. Conclusiones
Se exponen las conclusiones extraídas tras la realización del trabajo.

6. Bibliografía
Se indica la bibliografía usada para desarrollar el trabajo.

7. Cuadernos de trabajo
Se indica aquí la evolución del trabajo desarrollado por los componentes del grupo.

Anexo
Se adjunta además un manual de usuario de la aplicación desarrollada.

2. PRELIMINARES

2.1 Compresión de Imágenes

Para representar una imagen con una cantidad de colores similar a la que nuestro ojo puede reconocer se necesitan 24 bits de información por cada píxel de la imagen, es decir 8bits/píxel x 3colores/píxel. No es raro encontrar scanner con una resolución de 600 x 600 puntos por pulgada cuadrada. Cuando esta pulgada cuadrada de información se encuentra digitalizada, cada pequeño punto será representado por un píxel, en conclusión, para una pulgada cuadrada con un equipo con resolución de 600 x 600 puntos por pulgada, estamos hablando de 600 x 600 píxeles = 360.000 píxeles por pulgada cuadrada. Ahora bien un documento tamaño carta tiene 8.5 pulgadas de ancho y 11 pulgadas de alto, es cuestión de multiplicar y obtenemos 8.5 x 11 x 600 x 600 = 33.660.000 píxeles.

Con ello obtenemos la cantidad de píxeles que se necesitan para representar una imagen de gran calidad. Si queremos imágenes de color verdadero, es decir, usando 24 bits ó 3 bytes por píxel, obtenemos un tamaño para la imagen de 98,613 Kbytes. Esta cantidad era algo impresionante para los equipos de hace algunos años, y aunque hoy día hay ordenadores mucho más potentes tampoco es poco.

Este tamaño, se aprecia más cuando transmitimos imágenes a través de Internet u otra red, sobre todo, cuanto mayor es el tamaño de una imagen, mayor coste obtenemos en la transmisión de la misma. Por tanto el poder tener una misma imagen con un menor tamaño es muy deseable. Esta reducción de tamaño es la compresión. A la hora de comprimir imágenes hay tres métodos fundamentales:

Por Patrones (Mapping)

La idea consiste en minimizar la cantidad de datos mediante el reconocimiento de patrones que encuentren la repetición o la redundancia de información en los valores de los datos. Es decir, una imagen completamente aleatoria no puede comprimirse de esta forma. Distinguimos dentro de la técnica de compresión por patrones los siguientes esquemas fundamentales:

1. **Encontrar patrones unidimensionales**, es decir procesar los datos como si fueran una lista, o sea, uno detrás de otro. Se trata de reducir el número de datos quitando los que son redundantes. Este esquema es utilizado por la codificación Run-Lenght y la codificación LZW.
2. **Encontrar patrones bidimensionales**, es decir, en este caso procesar los datos como si se tratasen de matrices. Consiste en reducir la importancia de los datos captando la esencia de los mismos. Este esquema es utilizado por la transformada discreta del coseno (DCT) y los fractales.
3. **Encontrar patrones tridimensionales**, es decir procesar los datos como si fueran una lista de matrices. Consiste en reducir la magnitud de los valores de los datos. Este esquema es utilizado por la motion compensation y la predicción con una reducción de la resolución.

2.2 El formato JPEG

Para satisfacer esa necesidad de compresión necesaria a la hora de navegar por la Web por ejemplo, el Joint Photographic Expert Group (JPEG) creó el formato de compresión de su mismo nombre (JPEG) siguiendo las siguientes directrices generales para la compresión y codificación de imágenes estáticas de tono continúo basadas en matrices:

- *Uso corriente*. Se trataba de desarrollar un estándar que se adaptase a las necesidades de las artes en cuanto a las tasas de compresión y a la calidad de la imagen.
- *Controlable por el usuario*. El codificador de compresión debía contener parámetros que permitiesen a las aplicaciones elegir intercambios entre la calidad de la imagen y la razón de compresión.
- *Propósito general*. Debía ser aplicable a prácticamente todas las imágenes digitales de tono continuo. No debían aparecer restricciones por el tamaño de la imagen, las bandas de color o el rango del píxel.
- *Tratable*. Debía poseer una complejidad computacional tratable para hacer una implementación software factible.
- *Flexible*. Debía facilitar un juego de codificadores que permitiesen tres órdenes de datos diferentes y un modo de compresión sin pérdida.

Todos estos objetivos se plasmaron en el estándar de compresión JPEG, que por su efectividad ha pasado a ser uno de los formatos de imágenes más utilizados en Internet y en el almacenamiento de grandes cantidades de imágenes indexadas como por ejemplo en la redacción de un periódico. Una definición más clara y concisa de JPEG es que se trata de un conjunto de algoritmos de compresión que se aplican uno detrás de otro hasta conseguir reducir o comprimir la imagen.

El algoritmo de compresión fundamental de JPEG está basado en la codificación por transformación usando la transformada discreta del coseno (DCT) para bloques 8×8 píxeles pues se comprobó que se trataba de la mejor relación calidad/compresión.

Este formato no sólo tiene virtudes, la pérdida de calidad a la hora de comprimir imágenes implica que el formato JPEG tenga una serie de limitaciones. JPEG fue especialmente creado para comprimir imágenes en color o en escala de grises, pero no en blanco y negro. Igualmente, funciona mejor cuando la imagen presenta una coloración continuada y suave ya que las imágenes que presentan cambios bruscos de color no se comprimen demasiado bien.

Una de las primeras preguntas que nos podemos hacer es ¿Por qué se puede comprimir una imagen? Esto es posible por la forma en la que el ojo humano percibe las imágenes. Una imagen antes de ser comprimida está en un formato llamado RGB, es decir, un formato en el que el color de cada píxel viene dado por la suma de tres colores, una componente roja, una verde y otra azul. Por lo tanto, su codificación viene dada por 3 bytes, uno por cada color. Este sistema es el que se usa por ejemplo en el formato BMP (BitMaP format).

Para comprimir la imagen en formato JPEG, antes de aplicar el esquema de compresión, se transforman las tres matrices (R, G, B) en otras tres. Dos matrices denominadas de crominancia y otra denominada de luminancia. La de luminancia contiene la

información de escala de grises y las otras dos la información de color. Cada una de las matrices es comprimida por separado ya que por motivos visuales se puede perder más información de color que de escala de grises, debido a que el ojo humano utiliza la escala de grises para reconocer bordes.

Ventajas e inconvenientes de JPEG

El estándar JPEG tiene una fuerte aceptación en el mundillo informático. Esto se debe a varios factores clave en JPEG:

1. Bajo consumo de memoria, que permite implementaciones hardware de bajo coste, por ejemplo en cámaras de fotografía digitales.
2. Baja complejidad del algoritmo, lo que de nuevo, abarata el diseño de los chips decodificadores de JPEG.
3. Alcanza una buena tasa de compresión de imágenes naturales, todo usando el modelo visual.
4. Está muy extendido para el intercambio de imágenes, en la Web, etc.

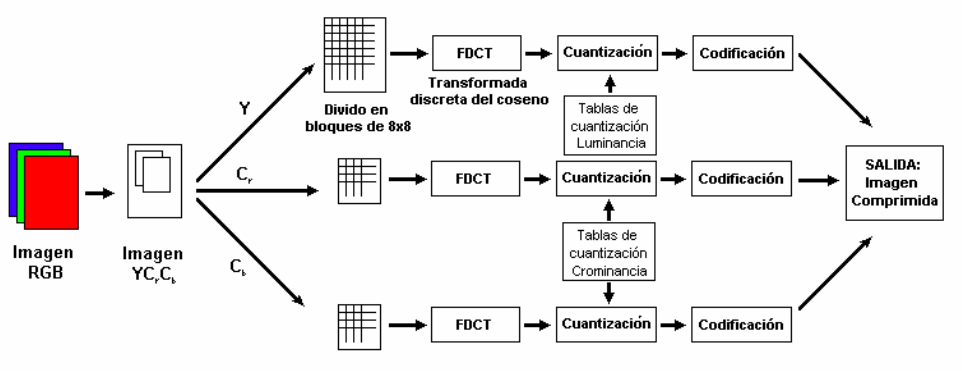
Sin embargo, es un estándar con una larga historia tras de sí, y con el tiempo, la mejora de las tecnologías y la aparición de nuevas aplicaciones con requerimientos más exigentes hace necesario la aparición de un nuevo estándar que supere las limitaciones de JPEG, un subconjunto de las mismas son:

1. JPEG solo permite una única resolución y calidad.
2. En compresión con perdida la tasa de compresión es baja.
3. En imágenes fuertemente comprimidas aparecen los famosos artefactos con forma de cuadrados.
4. Es un formato poco resistente a errores, por ejemplo, en transmisión de un JPEG a través de una red inalámbrica propensa a errores.
5. No ha sido pensado para la compresión de imágenes sintéticas, solamente para imágenes naturales.
6. No admite tener zonas de la imagen codificadas con mayor nivel de detalle que otras. -No ofrece buena calidad en compresión de imágenes de dos niveles (B/N).

Debido a estos factores y a los avances en la investigación en compresión de imágenes, se decidió desarrollar un nuevo estándar para compresión de imágenes que se adapte a las nuevas necesidades en los campos en los que son necesarios.

Esquema de compresión JPEG

Básicamente, el esquema de compresión sigue el siguiente esquema:



En resumen, dividimos la imagen original RGB en tres matrices, dos matrices de crominancia y una matriz de luminancia. A continuación le aplicamos a cada una de las matrices la transformada discreta del coseno (DCT), después cuantizamos y por último codificamos obteniendo de esta forma la imagen comprimida. Para poder volver a ver la imagen original seguiremos el proceso inverso.

División de la imagen en 8 x 8

La compresión JPEG se puede aplicar a cualquier tamaño en el bloque de datos, pero los test nos han revelado que con bloques de tamaño 8 x 8 píxeles se consiguen buenos índices de compresión manteniendo la calidad en las resoluciones que permiten los monitores actuales.

Además, cuando estos algoritmos fueron implementados comercialmente, el bloque de 8 x 8 se podía acomodar fácilmente en un chip LSI, mientras que el bloque de 16 x 16 no. Por tanto, para aplicar la transformada discreta del coseno (DCT) tendremos que dividir la imagen en bloques de 8 x 8 píxeles.

Pero existe un problema, ¿qué ocurre si el número de píxeles de la imagen, en la fila o en la columna no es un múltiplo de ocho? Existen dos soluciones posibles que podemos adoptar:

1. Repetir los bits adyacentes hasta tener un múltiplo de ocho alargando la imagen y por tanto, añadiendo información ficticia.
2. Recortar los últimos bits hasta obtener un múltiplo de ocho de manera que nos queda una imagen ligeramente inferior por los bordes.

Transformada discreta del coseno

La transformada discreta del coseno (DCT) es un método de compresión por transformación (mapping), es decir, lo que se intenta es minimizar la cantidad de datos mediante patrones que explotan el orden existente en los datos. Se basa principalmente en el reconocimiento de patrones de dos dimensiones.

Las transformadas matemáticas transforman los datos de un sistema de medida en otro de forma, que algunas veces los datos representados en el nuevo sistema facilitan la compresión de los datos. De esta forma, algunas transformaciones han sido inventadas exclusivamente para este propósito. Por ejemplo, la transformada discreta de Fourier (DFT), la transformada discreta del coseno (DCT), la transformada de Hadammard-Haar (HHT), la transformada de Karhunen-Loeve (KLT) o la transformada de Slant-Haar (SHT)

De entre todas estas transformadas se eligió la transformada discreta del coseno (DCT) por tener, en primer lugar, un algoritmo más rápido para implementarlo y en segundo lugar porque después de numerosas pruebas se ha demostrado que con esta transformada se consiguen imágenes de mayor calidad.

En teoría a la hora de aplicar la transformada discreta del coseno (DCT) no hay pérdidas de información, pero en la práctica si las hay, debido a que no puede calcularse exactamente el valor del coseno y a cálculos repetidos que usan números limitados en precisión que introducen errores de redondeo. De todas formas, estos errores son tan pequeños que es muy difícil apreciarlos. La fórmula de la DCT es de la forma:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{i=0}^{7}\sum_{j=0}^{7} f(i,j)\cos\left(\frac{(2i+1)u\pi}{16}\right)\cos\left(\frac{(2j+1)v\pi}{16}\right)$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & x = 0 \\ 1 & otrocaso \end{cases}$$

En esta fórmula, (u, v) son las coordenadas de la matriz transformada del coseno (DCT) y f (i, j) son los valores de la matriz original. El resto son constantes.

La inversa de la transformada discreta del coseno (IDCT) se utiliza para volver a tener los valores originales de la matriz original. Es de la forma:

$$f(i,j) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)\cos\left(\frac{(2i+1)u\pi}{16}\right)\cos\left(\frac{(2j+1)v\pi}{16}\right)$$

Cuantización

La precisión se reduce en el proceso de cuantización dividiendo cada valor en el bloque de 8 x 8 por un divisor único que se almacena en una tabla de dimensión 8 x 8 también denominada tabla de cuantización. Esta tabla se aplica a todos los bloques de la matriz obtenidos tras aplicar la transformada discreta del coseno (DCT). Matemáticamente el paso de cuantizar es una simple operación de división, pero se puede realizar de forma muy eficiente mediante hardware con operaciones de desplazamiento si los factores de cuantización se restringen a potencias de dos.

El proceso de decodificación recupera las magnitudes de los términos multiplicando por los mismos valores de cuantización aunque es imposible recuperar los términos con su precisión original. El proceso de cuantización se puede resumir en dos formulas una utilizada en la compresión y otra en la descompresión.

Codificación

$$\text{Término cuantizado} = \text{Redondeo}\left(\frac{\text{Término Original}}{\text{Factor de cuantización}}\right)$$

Decodificación

Término restituido = Término cuantizado * Factor de cuantización

Se puede observar claramente que la descompresión no puede nunca restituir exactamente el valor cuantizado debido al redondeo que se realiza durante la cuantización en la compresión. Normalmente los valores de la tabla de cuantización son potencias de dos para facilitar su implementación en hardware.

Desde el punto de vista cuantitativo se puede decir que para una compresión de buena calidad los errores no superan el 2%, errores que no son apreciados por el ojo humano. De esta forma podemos reducir la precisión de una imagen, provocando una disminución en su tamaño sin que el ojo humano pueda detectar que se ha producido un cambio en la imagen.

Debemos indicar que en la mayoría de los casos los estándares JPEG no definen una tabla de cuantización por defecto sino que dejan que cada una de las aplicaciones defina su propia tabla según sus necesidades, además de indicar que una imagen comprimida solo podrá ser recuperada si el elemento o aplicación decodificadora posee las tablas de cuantización que se utilizaron en la compresión, es decir, si compresión y descompresión tienen las mismas tablas.

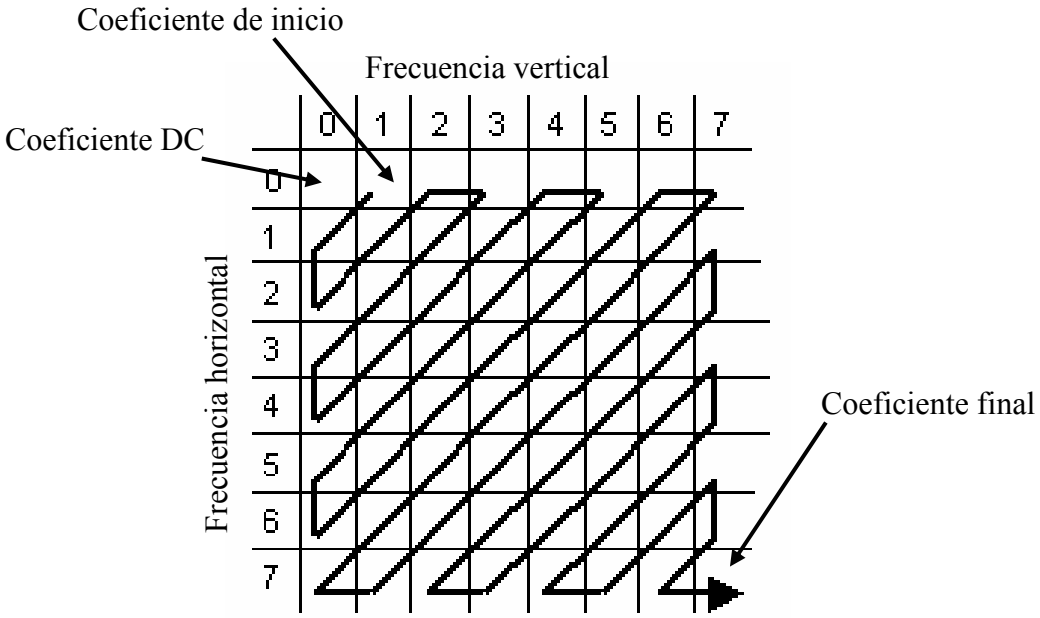
Por todo esto, en algunos casos por temas de compatibilidad, las tablas pueden ser incluidas en las cabeceras de los ficheros comprimidos JPEG, de manera que el estándar permite que puedan almacenarse hasta cuatro tablas de cuantización, de forma que puede utilizarse cada una de estas sobre diferentes componentes de los datos almacenados de la imagen como por ejemplo una para la crominancia y otra para la luminancia.

Codificación

La codificación es una combinación de dos transformaciones y una asignación de códigos. Mediante estos tres pasos se nos permitirá reducir el tamaño de los datos, aumentando así su portabilidad. En primer lugar, los bloques de datos de 8 x 8 píxeles son transformados de tablas bidimensionales a listas secuénciales de 64 valores. En segundo lugar, estas listas se codifican con código run-length o código de secuencias o código de cadenas, para minimizar cualquier redundancia. Y en último lugar, a esta lista de secuencias se le aplica el algoritmo de Huffman para asignar a bloques con alta entropía (alta frecuencia de aparición) códigos de bits de menor longitud.

Transformación de bloque a cadena

La transformación de un bloque de datos a una lista secuencial se hace usando un esquema en zig-zag como el que se muestra en la figura:



La razón de porqué usar un esquema en zig-zag está en la importancia y la magnitud de los valores de frecuencia del coseno en el bloque de datos. Los valores de frecuencia de orden bajo, situados en la esquina superior izquierda, tienden a ser más grandes en magnitud que los de alta frecuencia, esquina inferior derecha, debido a la cuantización. El orden en zig-zag produce una secuencia al final de la cual predominan los ceros.

Código de cadenas

La secuencia obtenida es ahora codificada con run-length. Sin embargo, el código de cadenas utilizado es diferente al ya conocido. Lo que se intenta es codificar de manera eficiente las secuencias de ceros repetidos. La repetición de los valores distintos de cero no se tiene en cuenta al codificar ya que las secuencias serán muy cortas. Este esquema también minimiza el número de bits necesarios para codificar cada valor de la secuencia usando enteros de longitud variable. Cada término de la secuencia codificada consiste en tres valores:

- *Secuencia (4 bits)*: número de términos cero que lo preceden.
- *Tamaño (4bits)*: número de bits usados para representar el dato.
- *DATA-VALUE (variable)*: valor actual del término.

El DATA-VALUE sólo se codifica con el número de bits necesario para representarlo. La siguiente tabla muestra los rangos de valores que pueden ser representados por los distintos valores del campo tamaño.

Tamaño	Rango de Valores Posible
1	-1, 1
2	-3, -2, 2, 3
3	-7..-4, 4..7
4	-15..-8, 8..15
5	-31..-16, 16..31
6	-63..-32, 32..63
7	-127..-64, 64..127
8	-255..-128, 128..255
9	-511..-256, 256..511
10	-1023..-512, 512..1023

Los DATA-VALUEs se almacenan usando los bits menos significativos del valor del dato actual como enteros binarios en complemento a dos. Los bits más significativos se recuperan en el proceso de decodificación del valor del campo Tamaño. Si el valor del dato es negativo, se resta uno antes de codificar. Hay que darse cuenta que todos los valores negativos tienen los bits más significativos a 0 y que los positivos los tienen a 1.

Valor a ser codificado	Conversión si negativo	Representación del complemento a dos en 8 bits	Bits usados para DATA-VALUE
-7	(-7-1)=-8	11111000	000
-6	(-6-1)=-7	11111001	001
-5	(-5-1)=-6	11111010	010
-4	(-4-1)=-5	11111011	011
4		00000100	100
5		00000101	101
6		00000110	110
7		00000111	111

Finalmente, esta secuencia de valores se codifica con el algoritmo de entropía de Huffman. Los DATA-VALUE tienden a ser aleatorios, pero ciertos pares de Secuencia y Tamaño se repiten con más frecuencia que otros. Esta alta frecuencia de aparición de ciertos pares puede ser explotada mediante el código de Huffman.

Algoritmo de codificación de Huffman

Este algoritmo se basa en la entropía de los símbolos. Sigue los siguientes pasos:

1. Ordenar las probabilidades de los símbolos, p_i, decrecientemente y considerarlas como nodos hojas de un árbol.
2. Mientras haya más de un nodo:
   - Unir los dos nodos con menor probabilidad en un nuevo nodo con probabilidad igual a la suma de las probabilidades de cada nodo.
   - Asignar, arbitrariamente, un uno (1) a una de las ramas que llegan al nodo y un cero (0) a la otra.
3. Leer la secuencia de unos y ceros desde el nodo raíz al nodo hoja donde se encuentra el símbolo.

Este algoritmo da un *libro de códigos Huffman* para un conjunto de probabilidades dado. El tamaño del libro de códigos es *L* y el código más largo tiene como mucho *L* bits. De esta forma los símbolos que más se repiten tienen un código de menor longitud, siempre a costa de alargar la longitud de los que se repiten poco. Por ejemplo si pensamos en el siguiente conjunto de oraciones:

Mama la sala de la casa es esa.

Esta es la sala de casa mama.

Se puede observar que símbolos como la "a" pueden ser codificados con un menor número de bits y símbolos como la "i" deben ser codificados con un mayor número de bits. El código de Huffman se construye con las frecuencias de aparición de cada símbolo como se muestra en la siguiente tabla:

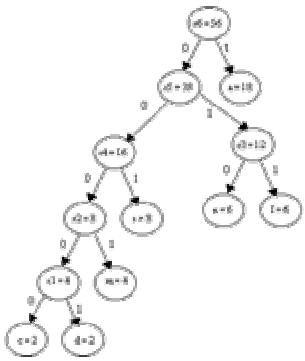
Símbolo	Frecuencia	Código (bits)
m	4	0001
a	18	1
l	6	011
s	8	001
d	2	00001
e	6	010
c	2	00000

Por lo tanto la frase anterior se puede codificar de la siguiente manera:

0001 | 1 | 0001 | 1 | 011 | 1 | 001 | 1 | 011 | 1 | 00001 | 010 | 011 | 1 | 00000 | 1 | 001 | 1 | 010 | 001 | 010 | 001 | 1

Mama la sala de la casa es esa

Lo cual tomaría 184 bits representarlo en código ASCII con un byte por símbolo. El ejemplo muestra que esta frase puede ser representada con tan sólo 8 bytes (64 bits).



Árbol de Huffman construido. El recorrido del nodo hacia las hojas determina el código en bits asociado a cada símbolo en las hojas.

2.3 JPEG 2000

Introducción

Tras la supremacía de JPEG como estándar de compresión de imágenes durante varios años, aparece un nuevo competidor al mismo. Debido al incremento en el uso de las tecnologías multimedia, y a los grandes avances técnicos en informática de los últimos años, la compresión de imágenes requiere mayor potencia así como nueva funcionalidad. Es por ello que se desarrolla JPEG 2000. No solo se ha pretendido que este estándar ofrezca una mejor calidad subjetiva que JPEG y una mayor tasa de compresión, sino que además ofrezca una rica gama de nuevas características que consigan el mismo éxito para el nuevo estándar que el que tuvo su predecesor. Además JPEG 2000 ha sido pensado con multitud de campos de aplicación en mente, de todo ello y de cómo funciona el nuevo esquema de codificación hablaremos en las siguientes páginas.

Presentando a JPEG 2000

El proyecto JPEG 2000 fue motivado durante el proceso de estandarización de JPEGLS (JPEG Lossless) un nuevo algoritmo de compresión de imágenes sin perdida, en dicho proceso, Ricoh envió un algoritmo llamado CREW, que finalmente no fue escogido para la implementación de JPEG-LS, pero se vio que ofrecía un conjunto interesante de características merecedor de un nuevo esfuerzo de estandarización. Esto fue durante 1996. El grupo de trabajo encargado de desarrollar el estándar es el ISO/IEC JTC1/SC29/WG1. El estándar JPEG 2000 es el ISO/IEC 15444.

Finalmente durante un largo proceso de varios años, concretamente hasta diciembre del 2000, se eligió finalmente como algoritmo base para JPEG2000 uno basado en EBCOT (Embedded Block Coding with Optimization Truncation) desarrollado por David Taubman que trabajaba para HP.

Hubo bastante excitación en la ISO con el nuevo estándar, y todo el mundo quería participar y poner su granito de arena en el estándar. El estándar solo especifica el decodificador y la sintaxis del bitstream y quedó dividido en varias partes:

Parte I. Codec básico (core)

Es la implementación básica mínima de JPEG 2000, es necesario tener un codec con un conjunto de opciones básicas para la implementación del mismo en dispositivos electrónicos como PDAs, impresoras, cámaras, etc. La parte I del estándar, por tanto, define un decodificador básico que minimiza el número de opciones y maximiza el intercambio, es decir, poder visualizar / tratar las imágenes en todos los dispositivos. Otro objetivo es solo incorporar a esta parte del estándar, técnicas que se encuentre libre de pago de royalties y licencias.

Parte II. Extensiones al codec

Las imágenes codificadas con la tecnología de la Parte II no podrán ser decodificadas por la de la Parte I, pero la de la Parte II si debe poder decodificar la de la Parte I. Incorpora mejoras respecto al codec base, además se usan algunas técnicas que si

pueden requerir el pago de royalties o licencias. Como ejemplo de técnicas mejoradas tenemos el uso de cuantización Trellis, o la posibilidad de usar wavelets personalizadas para la codificación.

Parte III. Motion JPEG 2000

Puesto que Motion JPEG ha sido un método muy usado en la edición de video de alta calidad (en estudios de producción) sin la existencia de un estándar ISO, ha sido atractivo definir esta parte del estándar.

Parte IV. Tests de conformidad al estándar

Parte V. Software de referencia.

Con esto se ha pretendido fomentar el desarrollo de software libre para la implementación del estándar.

Parte VI. Formato de fichero de imagen

Para evitar los errores que se produjeron en este aspecto en JPEG se ha definido de forma precisa el formato de imagen que debe seguir JPEG 2000.

Campos de aplicación de JPEG 2000

JPEG 2000 tiene múltiples campos de aplicación como son por ejemplo:

- Internet.
- Telefonía móvil de nueva generación.
- Aplicaciones móviles (cámaras, PDAs, sistemas embebidos,...).
- Impresión.
- Escaneado de imágenes en alta resolución.
- Fotografía digital.
- Aplicaciones médicas.
- Comercio electrónico.
- Librerías digitales.

Se aprecia que un requerimiento serio es una alta calidad de compresión tanto en bitrates altos (impresión, escaneado, etc) con en bitrates bajos (Internet, telefonía, etc).

Características fundamentales de JPEG 2000

A la hora de desarrollar JPEG 2000, se plantearon una serie de objetivos que debían ser alcanzados por el estándar. De hecho JPEG 2000 es un estándar realmente potente y complejo, mucho más que su predecesor. Podemos ver algunas de las características fundamentales de JPEG 2000 en la siguiente lista:

1. Alta tasa de compresión de imágenes.
2. Posibilidad de admitir transformación de color con pérdida y sin pérdida.
3. Codificación con y sin pérdida integrada en un único algoritmo.
4. Permitir compresión progresiva por resolución y calidad.

5. Región de interés estática o dinámica.
6. Fuerte resistencia frente a errores.
7. Acceso y procesamiento aleatorio de la imagen (espacialmente), concretamente
   permite rotación, traslación, escalado, aplicación de filtros y extracción de rasgos de forma eficiente. -Compresión de gran calidad en bit-rates altos y en bit-rates bajos de entre 0.0625 bpp y 0.25 bpp.
8. Permite codificar eficientemente imágenes de tipos distintos: escala de grises, bi-level (B/N), color, de múltiples componentes que incluso pueden estar almacenados cada uno con un distinto número de bits.
9. Incorpora métodos de manipulación de imagen sin introducir perdidas en la imagen, por ejemplo para la rotación, o edición de una parte de la imagen sin producir perdidas en el resto de la misma.
10. Incorporar una arquitectura abierta libre de pago de royalties, al menos en el codec básico.
11. Mecanismos de protección de los derechos de propiedad intelectual.

Codec básico (ISO–15444–1)

A pesar de que el estándar sólo define normativamente el decodificador, pienso que es más sencillo comprender como puede funcionar el codificador, para comprender mejor el funcionamiento de las técnicas empleadas en JPEG 2000.

Tampoco voy a entrar en detalle, porque los detalles de implementación son muchos, solo pretendo dar una explicación a grandes rasgos de cómo funciona el algoritmo básico de codificación. Inicialmente partimos de una imagen a codificar. Sobre dicha imagen debemos efectuar un preprocesamiento, que consiste en aplicar de los siguientes pasos los que sean necesarios:

Tiling

El estándar funciona sobre tiles, donde un tile es una partición rectangular de la imagen. Cada tile se trata de forma independiente del resto, con lo que vemos que es posible una implementación más eficiente del codec en paralelo. Si no queremos dividir la imagen en tiles, por ejemplo porque la imagen sea pequeña o por el motivo que sea, podemos tratar a la imagen como a un único tile. Los tiles permiten reducir el consumo de memoria, podemos comprimir un tile a la vez en cada paso. Todos los tiles deben tener las mismas dimensiones salvo, tal vez, los del borde derecho e inferior de la imagen. Presentan un inconveniente, cuanto menor es el tamaño del tile tenemos una menor tasa de compresión.

DC Level Shifting

En caso de que las muestras de la imagen sean sin signo, a cada muestra de cada tile, se le resta una cantidad, la profundidad del componente del tile. En el decodificador, se aplica el proceso inverso. Con este proceso, se consigue que los valores de las muestras en media se aproxime a 0 (DC).

Transformación de color

Ya hemos comentado que JPEG 2000 soporta imágenes tanto de uno como de múltiples componentes, cada componente puede además estar muestreado con distinto número de bits, por ejemplo, en imágenes con luminancia y crominancia la luminancia se suele muestrear con un mayor número de bits ya que el ojo percibe más la información de intensidad que la de color.

El estándar soporta dos transformaciones de color, una reversible: RCT (Reversible Color Transformation) y otra irreversible idéntica a la usada en JPEG: ICT (Irreversible Color Transformation) que es la transformación del espacio de color RGB al espacio de color YCbCr. En ambos casos se requiere que las imágenes tengan las 3 componentes RGB y que estas estén muestreadas con el mismo número de bits. La RCT se puede usar tanto para la transformación con pérdida como sin pérdida. Es una transformación que mapea de enteros a enteros sin pérdida, salvo un pequeño error debido a la precisión de la aritmética entera. Solo se puede usar con la Wavelet Reversible 5/3 que trabaja sobre enteros.

$$\begin{pmatrix} Y_r \\ U_r \\ Vr \end{pmatrix} = \begin{pmatrix} \left\lfloor \frac{R+2G+B}{4} \right\rfloor \\ R-G \\ B-G \end{pmatrix}$$

- *Ecuación 1 : transformación RCT*

$$\begin{pmatrix} G \\ R \\ B \end{pmatrix} = \begin{pmatrix} Y_r - \left\lfloor \frac{U_r+V_r}{4} \right\rfloor \\ U_r+G \\ V_r+G \end{pmatrix}$$

- *Ecuación 2 : transformación RCT Inversa*

La ICT es la transformación irreversible, más potente que la RCT pero que introduce pérdidas al aplicarse. Se usa exclusivamente con la transformada discreta Wavelet 9/7 irreversible que mapea de reales a reales.

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

- *Ecuación 3 : Transformación ICT*

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0 & 0 & 1.402 \\ 1.0 & -0.34413 & -0.71414 \\ 1.0 & 1.772 & 0 \end{pmatrix} \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix}$$

- *Ecuación 4 : Transformación ICT inversa*

Una vez terminado el preprocesamiento de la imagen se aplica el algoritmo de codificación, que comienza con la aplicación de una Wavelet discreta.

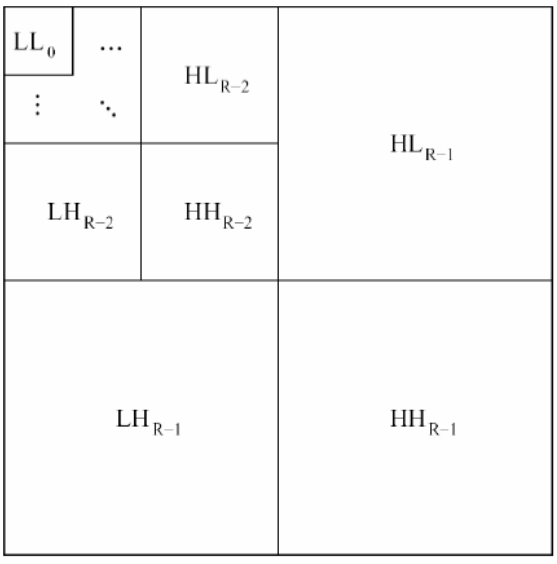
Wavelets

Las Wavelets son unas herramientas matemáticas que permiten la representación jerárquica de funciones. Frente a la transformada de Fourier tienen ciertas ventajas importantes como que funcionan bien tanto con ondas estacionarias como no estacionarias, el gran problema de la DFT es que no permite conocer en una posición determinada de la señal las distintas frecuencias que actúan sobre dicho punto, cosa que si es posible con una Wavelet. Esencialmente la representación Wavelet de una función es una representación "burda" de la misma, junto con unos coeficientes que influencian la función a distintos niveles de detalle. Se pretende representar una función con una colección de coeficientes, cada uno de los cuales provee una información limitada sobre la posición y la frecuencia de la función.

En JPEG 2000 se aplica una descomposición diádica a varios niveles, típicamente 6 niveles para imágenes lo suficientemente grandes. A través de la aplicación de la Transformada Wavelet Discreta (DWT) se dividen los tiles en múltiples bandas de frecuencias llamadas subbandas. Estas subbandas tienen una serie de propiedades estadísticas que permiten que los datos transformados se puedan codificar más eficientemente que los datos originales sin transformar. Una subbanda contendrá una serie de coeficientes que describirán las características de las frecuencias espaciales horizontales y verticales de la imagen original.

La descomposición diádica proporciona 4 subbandas:

- LL: bajas frecuencias horizontales y verticales.
- HL: altas frecuencias horizontales y bajas frecuencias verticales.
- LH: bajas frecuencias horizontales y altas verticales.
- HH: altas frecuencias horizontales y verticales.



- *Figura 1. Descomposición diádica de R niveles.*

Para aplicar una Wavelet 2D sobre una imagen primero se aplica sobre una dimensión y luego sobre la otra. En cada descomposición de la Wavelet se separan las altas frecuencias de las bajas frecuencias. En el codec básico (15444-1) se admite el uso de dos Wavelets, una reversible (5/3) y otra irreversible (9/7). En la segunda parte del estándar (15444-2) se admiten Wavelets personalizadas por el usuario o la aplicación. Las Wavelets se pueden aplicar tanto por el método de convolución como por un método llamado de elevación (lifting) que consiste en aplicar una secuencia simple de operaciones de filtrado, este es el método más común en las implementaciones.

Al terminar el proceso tenemos una serie de coeficientes agrupados en subbandas y niveles de resolución. Estos serán la entrada al algoritmo de cuantización.

Cuantización

La cuantización puede introducir o no perdidas, depende del ajuste de un parámetro. Se usa cuantización escalar con zona muerta. Sencillamente consiste en aplicar la siguiente ecuación.

$$q_b(u,v) = sign(a_b(u,v)) \left\lfloor \frac{|a_b(u,v)|}{\Delta_b} \right\rfloor$$

- *Ecuación 5 : Cuantización escalar donde $a_b(u,v)$ es el coeficiente de la transformada en la banda b y $q_b(u,v)$ es el coeficiente cuantizado.*

El parámetro Δb es el llamado escalón de cuantización y se puede ajustar para intentar conseguir un bitrate y una calidad determinada, hay varias posibilidades que se pueden combinar en el algoritmo para ajustar el bitrate del stream de salida, está es la primera opción. Si dicho parámetro vale 1 y los coeficientes se han obtenido por aplicación de la DWT 5/3, entonces no se produce perdida de precisión en el coeficiente, con lo que tenemos que es adecuada para la compresión sin pérdidas. El estándar no define como calcular dicho parámetro, pero hace una recomendación de como hacerlo.

$$\Delta_b = 2^{R_b - \varepsilon_b}\left(1 + \frac{\mu_b}{2^{11}}\right)$$

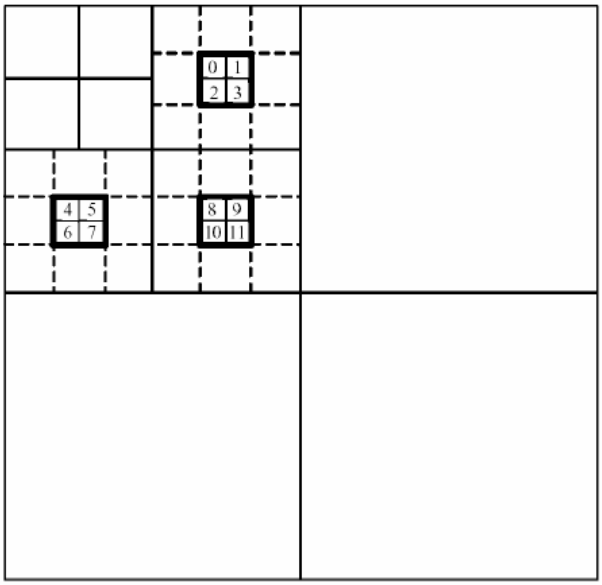
- *Ecuación 6 : Calculo del escalón de cuantización. Rb es el rango dinámico de la subbanda b, ε_b es el exponente y μ_b es la mantisa.*

El rango dinámico va a depender del número de bits usados para representar el componente del tile de la imagen original. Los coeficientes cuantizados van a ser la entrada para el siguiente bloque del algoritmo, la codificación de entropía.

Codificación de entropía.

Tras la cuantización hemos obtenido un conjunto de coeficientes pertenecientes a cada subbanda de cada nivel de resolución. En la cuantización dividimos cada subbanda en bloques rectangulares, tres bloques rectangulares (uno para cada subbanda a cada nivel de resolución) forman lo que se llama un precinct o packet partition location. Cada precinct a su vez está compuesto por varios bloques rectangulares más pequeños llamados code-blocks.

Cada code-block va a ser codificado independientemente. El uso de los precincts va a permitir, una vez comprimida la imagen, un acceso a diferentes zonas de la imagen más fino que con los tiles. También permiten, además, una reducción del consumo de memoria y facilitan el streaming.



- *Figura 2. 12 code-blocks de un precinct a nivel de resolución 2 de una descomposición diádica de tres niveles de una DWT.*

El codificador de entropía comprende dos partes, el TIER-1 Coding y el TIER-2 Coding.

TIER-1 Coding

En esta etapa, se toman como entrada cada uno de los code-blocks, independientemente unos de otros, y se aplica una codificación aritmética, binaria, dependiente del contexto de bitplanes (concretamente una variante del MQ Coder similar a la usada en el estándar JBIG-2).

Recordando la idea de bitplane, un plano de bits o bitplane se obtiene tomando un mismo bit para todas las muestras, por ejemplo, para una imagen con componente de escala de grises muestreada a 8 bits tendremos 8 bitplanes, comenzando por el bitplane

mas significativo que contendrá el MSB (Most Significative Bit) de cada muestra y terminando por el bitplane que contendrá el LSB de todas las muestras. El uso de bitplanes se justifica porque en muchos casos los bits menos significativos de las muestras contienen más información que los más significativos, en algunos casos los bitplanes de los bits más significativos están totalmente compuestos por ceros, mientras que los bitplanes inferiores contienen mucha más información. Además los bitplanes que sean 0 son codificables con muy poca información, generando, por tanto, compresiones muy eficientes de la información.

La codificación aritmética empleada en JPEG 2000 ha sido elegida con más acierto que la variante empleada en JPEG, ya que está última resultó estar patentada, aunque en JPEG también se permitía el uso de la codificación Huffman. Hay que recordar que uno de los objetivos del codec base de JPEG2000 es implementar un algoritmo libre de patentes y pago de royalties.

A grandes rasgos, la variante de la codificación aritmética empleada usa la subdivisión recursiva de intervalo por probabilidades de la codificación de Elias. Tiene la ventaja de que las decisiones binarias más probables pueden a menudo ser codificadas con un coste de mucho menos de 1 bit por decisión binaria.

Por otro lado, la variante de MQ Coder ha sido simplificada para su uso en JPEG 2000, con objeto de reducir la complejidad del algoritmo y mejorar la resistencia a errores. Para aplicar el proceso, se comienza desde el MSB y se busca el primer bitplane con al menos 1 bit. A continuación, se codifican los code-blocks con tres pasadas:

- Pasada de propagación de significancia.
- Pasada de refinamiento de magnitud.
- Pasada de limpieza (clean-up).

La entrada al codificador es un code-block y la salida es una serie de pasadas, cada una a su vez conteniendo una serie de símbolos binarios. Cada una de estas pasadas recoge información contextual sobre los datos del bit-plane, que luego es codificada por el codificador aritmético.

Solo voy a dar unas pinceladas de la función de cada pasada, ya que el algoritmo es medianamente complejo:

1. Pasada de propagación de significancia:

    - Se intenta determinar y predecir que muestras son significantes y su signo si lo tienen, y se codifican con un símbolo binario:

        - Si el plano del MSB se procesa, todas las muestras se predicen insignificantes, en caso contrario una muestra se predice significante si cualquiera de sus 8 vecinos se ha predicho significante.

2. Pasada de refinamiento:

    - Se predicen como significantes las muestras que siguen a las predichas en la primera pasada.

3. Pasada de limpieza (clean-up):

    - Se codifican las muestras no codificadas en las pasadas anteriores.

La idea general es que con cada pasada vamos refinando la información obtenida en las pasadas anteriores, de esta forma, es trivial ver que no almacenando el resultado de todas las capas podemos introducir más perdidas en el proceso de compresión, y a la vez reducir el tamaño de la imagen original. Esta técnica también se emplea para ajustar el caudal de bits resultante de la imagen original tras la compresión. También debe quedar claro que en el caso de querer una compresión sin perdidas debemos codificar todas las capas para poder reconstruir la imagen original.

TIER-2 Coding

Tras el TIER-1 coding tenemos una serie de símbolos binarios pertenecientes a varias pasadas aplicadas sobre cada code-block de la imagen. Antes de proseguir es necesario introducir dos nuevos términos, el término packet y el término layer.

La información obtenida del codificador TIER-1 se empaqueta en unidades de datos llamadas packets. Un packet está compuesto por varios sub-bitplanes de cada code-block en un nivel de resolución en un precinct.

Un layer es una colección de packets. Un packet se puede interpretar como un incremento de calidad para un nivel de resolución en una localización espacial (precinct) mientras que, un layer se puede interpretar como un incremento de calidad para la imagen entera a máxima resolución. Cada resultado de una pasada de codificación se asigna a un layer o se descarta.

En los primeros layers se almacena la información más importante y en los últimos los que aportan los detalles más finos. Para codificación sin perdidas hemos de almacenar todos los layers, mientras que para codificación con perdidas no.

Durante la decodificación la imagen reconstruida aumenta su calidad de forma progresiva conforme vamos procesando nuevos layers. Mediante estos mecanismos podemos implementar dos aspectos esenciales de JPEG 2000, la decodificación progresiva por resolución (gracias principalmente a las Wavelets que están íntimamente con el concepto de multirresolución, y también gracias a la progresión empleada en el codestream de salida) y la decodificación progresiva por calidad. En JPEG 2000 hay cinco progresiones válidas distintas, aunque solo voy a comentar dos de ellas, ya que las otras tres son un poco esotéricas.

Una progresión es una ordenación de packets en el stream final y las fundamentales son:

- Layer-resolución-componente-precinct que permite una recuperación progresiva por calidad.
- Resolución-Layer-componente-precinct que permite una recuperación progresiva por resolución.

Una progresión se puede aplicar mediante la aplicación de cuatro bucles anidados, donde el término de la izquierda indica el bucle más externo y el de la derecha el más interno. Otra posibilidad sería ordenarlos de forma que la información de color venga al final con lo que primero se obtendría la imagen en escala de grises y finalmente la información de color.

Esto es útil tanto para la transmisión por internet como para la edición de la imagen con una resolución determinada o con un nivel de calidad dado. Esto también se puede hacer básicamente al uso de una Wavelet.

Características adicionales

JPEG 2000 además incorpora algunas características que lo hacen más interesante. Destacamos por ejemplo las siguientes:

1. Posibilidad de definir una Región de Interés (ROI) en las imágenes. Puede ser interesante poder comprimir una cierta parte de la imagen de forma irregular a más calidad que el resto de la misma, por ello en JPEG 2000 se permite hacer esto de forma integrada con el algoritmo de codificación / decodificación.

2. Puesto que la codificación aritmética es propensa a errores de transmisión o de canal, es decir, que si alteramos uno o varios bits durante la transmisión, copia o almacenamiento de una imagen podemos producir un deterioro muy serio a la calidad de la imagen. El simple fallo de un bit puede provocar un fallo de sincronización en el decodificador de entropía y deteriorar la imagen. En JPEG2000 un aspecto crucial ha sido el de la resistencia a errores, para ello se usan diversas técnicas como por ejemplo :

    a. Usar bloques de información de pequeño tamaño (code-blocks), así un error en un code-block no afectará a los demás.
    b. También usar paquetes de tamaño reducido, eliminando campos innecesarios de las cabeceras.
    c. Añadir marcas de resincronización a los paquetes para que si se pierde la sincronización esta se pueda recuperar lo antes posible. El inconveniente está es que a mayor cantidad de información designada a la recuperación de errores se produce un aumento del tamaño de la imagen.
    d. Añadir símbolos de segmentación.

3. IPR (Intellectual Property Rights).

    a. El formato de fichero de JPEG2000 (J2P) permite añadir una serie de meta datos (en formato XML principalmente) para la protección de la propiedad intelectual de la imagen, añadiendo información del autor por ejemplo.

b. Se permite obtener los meta datos de la imagen previamente a la decodificación de la misma. Junto a los meta datos puede ir información necesaria para la decodificación correcta de la imagen.

Por último algunos comentarios adicionales, el algoritmo permite aplicar la Wavelet teniendo en memoria solamente una línea de la imagen, o del tile, con lo que se reduce el consumo de memoria. También permite dos cosas muy importantes, como la posibilidad de aplicar operaciones simples como rotación de la imagen sin volver a comprimir la imagen y por tanto perder información, y permite la edición y modificación o acceso aleatorio a una parte de la misma sin tener que comprimir el resto de la imagen, solo la parte modificada si es necesario.

2.4 Marcas de agua

Concepto de marca de agua

Una marca de agua no es más que un *mecanismo de protección* que se emplea para que *el propietario de ciertos datos pueda demostrar que le pertenecen y al mismo tiempo comprobar que no han sufrido alteración* al transmitirse por un canal no seguro.

La marca consiste en *información de algún tipo* (nombre del propietario, firma digital, etc.) *que se inserta en los datos a proteger y que puede luego ser recuperada para comprobar su integridad*.

El nombre de este mecanismo *(water-marking* en inglés) proviene de las tradicionales técnicas empleadas por algunas entidades para certificar la autenticidad y el origen de documentos de papel a través de variaciones en la opacidad de los mismos. Se trata de algo que no es nuevo, como atestiguan los siguientes ejemplos:

- El logotipo o "mosca" que las cadenas de TV impresionan es sus transmisiones.
- Las imágenes ocultas que se ven al mirar un billete al trasluz.
- Las imágenes ocultas en los juegos visuales en tres dimensiones que se descubren tras visualizar detenidamente otra imagen

En el campo de la *informática* se opera con *marcas de agua digitales* que, al contrario que las marcas impresas, que suelen ser visibles, se diseñadas para ser completamente invisibles. Además los bits representan las marcas se dispersan por toda la información a proteger para que la marca no pueda eliminarse sin degradar irremediablemente los datos protegidos.

Así, se preservarán los derechos de autor de forma perramente e inalterable, de manera que si se usa de forma ilegal la información se podrá rastrear e identificar al violador de estos derechos.

Consideraciones a tener en cuenta sobre las marcas de agua digitales

Las marcas de agua digitales existen en múltiples formas y pueden clasificarse atendiendo a diversos criterios.

Naturaleza de la información protegida

Los datos a proteger pueden corresponder a una imagen, a un fichero de audio, a una base de datos, a un correo electrónico, etc., y en general a una información de cualquier tipo. En función de la naturaleza de dicha información se emplearán distintos procedimientos de marcado.

Procedimiento de generación de la marca

Nos referimos aquí a la manera en la que se determina la marca a insertar en una cierta información. Esto puede hacerse aleatoriamente, a partir de fragmentos de los datos a proteger (p. ej. replicando los bits menos significativos), insertando una información de

identificación (p. ej. firma digital), siguiendo una clave de generación y recuperación, etc.

Procedimiento de recuperación de la marca

Nos referimos al método empleado para recuperar la marca insertada en una cierta información.

Existen procedimientos que requieren de la información original no marcada, mientras que otros no (p. ej. en el procesamiento de imágenes se usan métodos de estimación mediante gradientes para inducir cómo era la imagen original no marcada a partir de la imagen marcada).

Otros procedimientos pueden requerir de la existencia de una clave de extracción, para aumentar la seguridad del sistema de protección.

Robustez y resistencia a ataques

Se define la robustez de una marca de agua a la capacidad que tiene para resistir las alteraciones o ataques (activos para eliminarla o pasivos para detectarla, intencionados o no) sobre la información en la que va incluida. La naturaleza de los ataques puede ser muy variada, desde ataques premeditados para inutilizar la marca hasta involuntarios como pérdidas por compresión.

Para que una marca sea robusta debe introducirse en partes significativas de la información a proteger.

En general, según la utilidad que se le quiera dar a la marca, puede desearse que sea resistente a ciertos ataques y frágil a otros (p. ej. para usar la marca como medidor de la envergadura de una alteración de la información).

Imperceptibilidad e Indetectabilidad

Una marca de agua es imperceptible (transparente), si la degradación que causa en los archivos donde se ha insertado es muy difícil de apreciar por los seres humanos.

En algunas aplicaciones se puede aceptar una pequeña degradación de los datos, a cambio de lograr mayor robustez, ya que por insertarse la marca en partes significativas de la información para aumentar su robustez, puede hacerse patente una degradación.

Así, por ejemplo, puede producirse unas ciertas alteraciones en el color de una imagen en la que se inserta una marca robusta. El usuario deberá decidir si esto es importante o no.

Se dice que la marca es indetectable si después de haberla insertado, el archivo marcado conserva las mismas propiedades estadísticas que su original. Esto asegura que una persona no autorizada no podrá detectar la presencia de la marca utilizando métodos estadísticos.

Tolerancia a error

La tolerancia a error mide la capacidad del sistema de detección de marca para no emitir falsos juicios. Esto es, para no asegurar que no se detecta marca, cuando ésta existe (falso negativo) o para asegurar que existe cuando no es así (falso positivo).

Aplicaciones de las marcas de agua

Las marcas de agua, como mecanismo de protección que son, tienen gran número de aplicaciones. Exponemos aquí brevemente algunas de ellas a modo ilustrativo.

Comunicaciones secretas

Una marca incrustada archivos multimedia se utiliza por dos o más personas para comunicarse secretamente sin levantar la sospecha de terceros. Es la aplicación clásica de la esteganografía (ocultar una información dentro de otra).

Firmas digitales

Consiste en insertar una marca de agua con información del propietario de la información a efectos de identificación del propietario y como prueba de propiedad.

Autenticación de la información

Consiste en el uso de marcas digitales frágiles insertadas en información importante que permita detectar cualquier alteración en la misma.

Control de copias

Contienen información del propietario acerca de las reglas de uso y copiado de los archivos en los que se insertan. Las marcas de agua usadas en el control de copias restringen la utilización de los archivos de acuerdo a las regla de uso y copiado que porten.

Sistemas de procesamiento de marcas de agua

Un sistema de procesamiento de marcas de agua tiene dos partes fundamentales

La primera se encarga de generar la marca y manipular la información a proteger para insertar en ella dicha marca. Esto suele usarse empleando operaciones matemáticas.

La segunda se encarga de tomar la información (posiblemente manipulada) e intentar recuperar a partir de ella una marca reconocible para confirmar la integridad de los datos y / o la pertenencia a su autor.

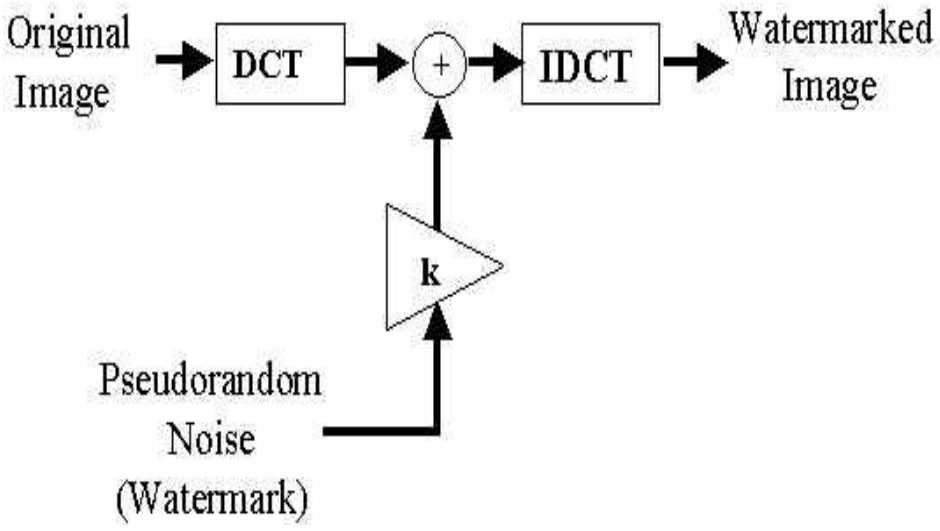
Por recuperar entiéndase recrear la marca, no eliminarla.

3. Algoritmo implementado

A continuación recogemos el algoritmo que hemos implementado para desarrollar nuestro sistema de procesamiento de marcas de agua.

3.1 Inserción de la marca

El proceso de inserción es el siguiente:



Esquema de marcado

Dada una imagen a marcar aplicamos la DCT (véase JPEG), se genera una marca pseudoaleatoria y se inserta según un cierto factor de energía k (véase más adelante). Al conjunto resultante se le aplica la IDCT y se obtiene una imagen marcada.

Veamos el proceso en detalle.

Paso 1: División de la imagen en bloque de 8x8

Supongamos que partimos de una imagen expresada mediante sus componentes de luminancia (Y) y crominancia (Cb, Cr).

El primer paso consiste en dividir la imagen almacenada en la componente Y en bloques de 8x8 píxeles (en caso de que sus dimensiones no sean múltiplos de 8 rellenamos con ceros).

Hay que señalar que este proceso lo hacemos tanto para JPEG como JPEG2000, ya que nos vimos obligados a usar la DCT en uno y otro caso debido a lo poco específico que resultaba el documento que seguíamos en lo referente al uso de tecnología *wavelet* para inserción de marcas de agua.

En la codificación, a efectos prácticos nos limitamos a recorrer según fuera conveniente distintas secciones de la matriz que representaba la imagen (no se realizó división real). Para ello se usaron fragmentos de código como éste:

```
for (y=0; y < nMaxY; y=y+8){
          for (x=0; x < nMaxX; x = x+8){
               TRATAMIENTO();
```

En el caso de que las dimensiones de la imagen no sean múltiplos de 8, se realiza un preprocesado en el que se rellena con 0s la matriz.

Para el resto del algoritmo, cada vez va realizando un cierto tratamiento sobre cada bloque o bien sobre información extraída de él.

Paso 2: Calcular la DCT al bloque de 8x8

Este paso consiste en tomar la matriz que representa al bloque de 8x8 y calcular su TDC usando la *expresión matricial de dicha transformación*, la cual se recoge a continuación:

$$I'(y, x) = C(y, x) * I(y, x) * CT(y, x)$$

Se considerará I(y,x) como la matriz del bloque e I'(y,x) como la matriz resultante de la transformación. C(y,x) es la matriz de la TDC para n = 8 y CT(y,x) su traspuesta. C(y,x) tiene la siguiente forma:

C(y,x) = {
{0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536},
{0.4904, 0.4157, 0.2778, 0.0975, -0.0975, -0.2778, -0.4157, -0.4904},
{0.4619, 0.1913, -0.1913, -0.4619, -0.4619, -0.1913, 0.1913, 0.4619},
{0.4157, -0.0975, -0.4904, -0.2778, 0.2778, 0.4904, 0.0975, -0.4157},
{0.3536, -0.3536, -0.3536, 0.3536, 0.3536, -0.3536, -0.3536, 0.3536},
{0.2778, -0.4904, 0.0975, 0.4157, -0.4157, -0.0975, 0.4904, -0.2778},
{0.1913, -0.4619, 0.4619, -0.1913, -0.1913, 0.4619, -0.4619, 0.1913},
{0.0975, -0.2778, 0.4157, -0.4904, 0.4904, -0.4157, 0.2778, -0.0975}
}

Para implementar la TDC de cada bloque, se efectuaron llamadas a la siguiente función:

```
void TDC8(int fInicio, int cInicio,
                  double ** Componente)

{

double Maux[8][8];
double Caux[8][8];

int i,j,k;
double aux;

       double C[8][8] = {
              {0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536},
              {0.4904, 0.4157, 0.2778, 0.0975, -0.0975, -0.2778, -0.4157, -0.4904},
```

```
                {0.4619, 0.1913, -0.1913, -0.4619, -0.4619, -0.1913, 0.1913, 0.4619},
                {0.4157, -0.0975, -0.4904, -0.2778, 0.2778, 0.4904, 0.0975, -0.4157},
                {0.3536, -0.3536, -0.3536, 0.3536, 0.3536, -0.3536, -0.3536, 0.3536},
                {0.2778, -0.4904, 0.0975, 0.4157, -0.4157, -0.0975, 0.4904, -0.2778},
                {0.1913, -0.4619, 0.4619, -0.1913, -0.1913, 0.4619, -0.4619, 0.1913},
                {0.0975, -0.2778, 0.4157, -0.4904, 0.4904, -0.4157, 0.2778, -0.0975}
                                                };

        double CT[8][8] = {
                {0.3536,0.4904,0.4619,0.4157,0.3536,0.2778,0.1913,0.0975},
                {0.3536,0.4157,0.1913,-0.0975,-0.3536,-0.4904,-0.4619,-0.2778},
                {0.3536,0.2778,-0.1913,-0.4904,-0.3536,0.0975,0.4619,0.4157},
                {0.3536,0.0975,-0.4619,-0.2778,0.3536,0.4157,-0.1913,-0.4904},
                {0.3536,-0.0975,-0.4619,0.2778,0.3536,-0.4157,-0.1913,0.4904},
                {0.3536,-0.2778,-0.1913,0.4904,-0.3536,-0.0975,0.4619,-0.4157},
                {0.3536,-0.4157,0.1913,0.0975,-0.3536,0.4904,-0.4619,0.2778},
                {0.3536,-0.4904,0.4619,-0.4157,0.3536,-0.2778,0.1913,-0.0975}
                                                };

/* PRIMERA MULTIPLICACION */

for (i=0; i<8;i++)
{
        for (j=0; j<8;j++)
        {
                aux = 0;
                for(k = 0; k < 8; k++)
                {
                aux = aux + C[i][k]*Componente[fInicio + k][cInicio + j];
                }

                Maux[i][j] = aux;
        }

}

/* SEGUNDA MULTIPLICACION */

for (i=0; i<8;i++)
{
        for (j=0; j<8;j++)
        {
                aux = 0;
                for(k = 0; k < 8; k++)
                {
                        aux = aux + Maux[i][k]*CT[k][j];
                }

                Caux[i][j] = aux;
        }

}

/* Actualizacion de la matriz */

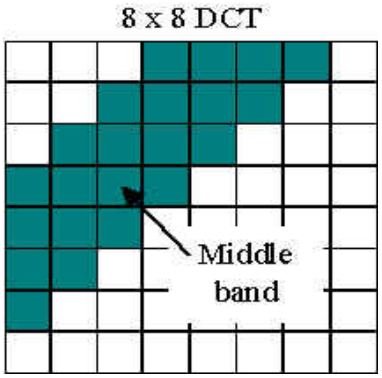
for (i=0; i<8;i++)
{
        for (j=0; j<8;j++)
        {
                Componente[fInicio + i][cInicio + j] = Caux[i][j];
        }
}

}
```

Componente es la matriz de la imagen, quedando delimitado el bloque a tratar por los valores de fInicio y cInicio (coordenadas de la fila y la columna de la matriz donde empieza el bloque).

Paso 3: Extraer el zigzag del bloque de 8x8

Este paso consiste en seleccionar el conjunto de elementos del bloque sobre los que se insertará la marca. A este conjunto, que posee exactamente 22 elementos, se le llama *zigzag* porque pueden obtenerse mediante un recorrido de zigzag del bloque y puede observarse en la figura siguiente:



La elección de estos elementos y no otros es casual, *pues se corresponden con las componentes de media frecuencia del bloque de 8x8*.

El sistema visual humano es muy sensible a las componentes de baja frecuencia, ya que concentran la mayor parte de la información de las imágenes, por lo no son adecuadas para insertar una marca invisible, como es nuestro caso.

Por otro lado, los sistemas de compresión son agresivos con las componentes de alta frecuencia, precisamente porque no se suele almacenar información de interés en ella. Por lo tanto, las componentes de alta frecuencia no son aconsejables para insertar una marca razonablemente resistente a los ataques de compresión, como es nuestro caso.

De lo anterior, se deduce que un compromiso es usar las componentes de media frecuencia si se pretende una marca imperceptible y resistente.

Para obtener el zigzag, se realiza una llamada a la siguiente función:

```
void ZigzagDirecto8(int fInicio,int cInicio,double ** Y,double Z[22])
{


Z[0] = Y[fInicio + 3][cInicio + 0];
Z[1] = Y[fInicio + 2][cInicio + 1];
Z[2] = Y[fInicio + 1][cInicio + 2];
Z[3] = Y[fInicio + 0][cInicio + 3];

Z[4] = Y[fInicio + 0][cInicio + 4];
Z[5] = Y[fInicio + 1][cInicio + 3];
```

```
Z[6] = Y[fInicio + 2][cInicio + 2];
Z[7] = Y[fInicio + 3][cInicio + 1];
Z[8] = Y[fInicio + 4][cInicio + 0];

Z[9] =  Y[fInicio + 5][cInicio + 0];
Z[10] = Y[fInicio + 4][cInicio + 1];
Z[11] = Y[fInicio + 3][cInicio + 2];
Z[12] = Y[fInicio + 2][cInicio + 3];
Z[13] = Y[fInicio + 1][cInicio + 4];
Z[14] = Y[fInicio + 0][cInicio + 5];

Z[15] = Y[fInicio + 0][cInicio + 6];
Z[16] = Y[fInicio + 1][cInicio + 5];
Z[17] = Y[fInicio + 2][cInicio + 4];
Z[18] = Y[fInicio + 3][cInicio + 3];
Z[19] = Y[fInicio + 4][cInicio + 2];
Z[20] = Y[fInicio + 5][cInicio + 1];
Z[21] = Y[fInicio + 6][cInicio + 0];


}
```

Esta función almacena en Z los elementos del zigzag del bloque correspondiente de la matriz de la imagen Y.

Paso 4: Calcular la marca aleatoria para el zigzag

En este paso se generan aleatoriamente 22 números reales que se insertarán en los elementos del zigzag. Esos número se generan de manera que siguen una distribución normal de parámetros (0,1). Esto es así porque puede demostrarse que esa distribución confiere buenas propiedades a la marca.

El código usado para esto es el siguiente:

```
srand( (unsigned)time( NULL ) );

      for (y=0; y < tamMarca; y++)
            for (x=0; x < 11;x++)
            {

                  u1 = rand();
                  u2 = rand();

                  r = sqrt(-2*log(u1));
                  theta = 2 * pi * u2;

                  MarcaIn[y][21 - x]      = cos(theta);
                  MarcaIn[y][x]           = sin(theta);

            }
```

tamMarca almacena el número de bloques a procesar. MarcaIn es una matriz que almacena en cada fila los 22 elementos que corresponden a la marca a insertar en el bloque correspondiente. Como se ve, en la implementación, por motivos de eficiencia se genera la marca para la imagen completa de una sola vez.

Paso 5: Insertar la marca en el zigzag

En este paso se realiza la inserción de la marca m en cada píxel c del zigzag para obtener el píxel marcado c' correspondiente. Esto se realiza aplicando la siguiente fórmula:

$$c' = c + alfa * c * m$$

alfa es el factor de energía (k en el esquema visto anterior) de la marca, y nos permite ajustar tanto la robustez de la marca como su visibilidad, pues cuanto mayor sea alfa, mayores serán esas propiedades. Empíricamente se obtiene según el documento que hemos seguido que alfa = 0.1.

En codificación, este paso se ha resuelto mediante la llamada a la siguiente función:

```
void Marcar8(int cont, double ** Marca, double Z[22])
{


double alfa = 0.1;
int i;

      for(i = 0;i < 22; i++)
      {

            Z[i] = Z[i] + alfa*Marca[cont][i]*Z[i];

      }


}
```

Paso 6: Reintegración del zigzag marcado en el bloque de 8x8

Este paso consiste en la inversión del algoritmo de zigzag, para sustituir los elementos seleccionados en el bloque por los elementos marcados.

Esto se realiza mediante una llamada a la siguiente función:

```
void ZigzagInverso8(int fInicio,int cInicio,double ** Y,double Z[22])
{

Y[fInicio + 3][cInicio + 0] = Z[0];
Y[fInicio + 2][cInicio + 1] = Z[1];
Y[fInicio + 1][cInicio + 2] = Z[2];
Y[fInicio + 0][cInicio + 3] = Z[3];

Y[fInicio + 0][cInicio + 4] = Z[4];
Y[fInicio + 1][cInicio + 3] = Z[5];
Y[fInicio + 2][cInicio + 2] = Z[6];
Y[fInicio + 3][cInicio + 1] = Z[7];
Y[fInicio + 4][cInicio + 0] = Z[8];

Y[fInicio + 5][cInicio + 0] = Z[9];
Y[fInicio + 4][cInicio + 1] = Z[10];
Y[fInicio + 3][cInicio + 2] = Z[11];
```

```
Y[fInicio + 2][cInicio + 3] = Z[12];
Y[fInicio + 1][cInicio + 4] = Z[13];
Y[fInicio + 0][cInicio + 5] = Z[14];

Y[fInicio + 0][cInicio + 6] = Z[15];
Y[fInicio + 1][cInicio + 5] = Z[16];
Y[fInicio + 2][cInicio + 4] = Z[17];
Y[fInicio + 3][cInicio + 3] = Z[18];
Y[fInicio + 4][cInicio + 2] = Z[19];
Y[fInicio + 5][cInicio + 1] = Z[20];
Y[fInicio + 6][cInicio + 0] = Z[21];

}
```

Paso 7: Calcular la IDCT al bloque de 8x8

Este paso consiste en tomar la matriz que representa al bloque de 8x8 ya marcado y calcular su ITDC usando la *expresión matricial de dicha transformación*, la cual se recoge a continuación:

$$I'(y, x) = CT(y, x) * I(y, x) * C(y, x)$$

Se considerará I(y,x) como la matriz del bloque e I'(y,x) como la matriz resultante de la transformación. C(y,x) es la matriz de la TDC para n = 8 y CT(y,x) su traspuesta.

Para este paso, se han realizado llamadas a la siguiente función:

```
void ITDC8(int fInicio, int cInicio,
                  double ** Componente)

{

double Maux[8][8];
double Caux[8][8];

int i,j,k;
double aux;


       double C[8][8] = {
              {0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536, 0.3536},
              {0.4904, 0.4157, 0.2778, 0.0975, -0.0975, -0.2778, -0.4157, -0.4904},
              {0.4619, 0.1913, -0.1913, -0.4619, -0.4619, -0.1913, 0.1913, 0.4619},
              {0.4157, -0.0975, -0.4904, -0.2778, 0.2778, 0.4904, 0.0975, -0.4157},
              {0.3536, -0.3536, -0.3536, 0.3536, 0.3536, -0.3536, -0.3536, 0.3536},
              {0.2778, -0.4904, 0.0975, 0.4157, -0.4157, -0.0975, 0.4904, -0.2778},
              {0.1913, -0.4619, 0.4619, -0.1913, -0.1913, 0.4619, -0.4619, 0.1913},
              {0.0975, -0.2778, 0.4157, -0.4904, 0.4904, -0.4157, 0.2778, -0.0975}
                                          };

       double CT[8][8] = {
              {0.3536,0.4904,0.4619,0.4157,0.3536,0.2778,0.1913,0.0975},
              {0.3536,0.4157,0.1913,-0.0975,-0.3536,-0.4904,-0.4619,-0.2778},
              {0.3536,0.2778,-0.1913,-0.4904,-0.3536,0.0975,0.4619,0.4157},
              {0.3536,0.0975,-0.4619,-0.2778,0.3536,0.4157,-0.1913,-0.4904},
              {0.3536,-0.0975,-0.4619,0.2778,0.3536,-0.4157,-0.1913,0.4904},
              {0.3536,-0.2778,-0.1913,0.4904,-0.3536,-0.0975,0.4619,-0.4157},
              {0.3536,-0.4157,0.1913,0.0975,-0.3536,0.4904,-0.4619,0.2778},
              {0.3536,-0.4904,0.4619,-0.4157,0.3536,-0.2778,0.1913,-0.0975}
                                          };
```

```
/* PRIMERA MULTIPLICACION */

for (i=0; i<8;i++)
{
      for (j=0; j<8;j++)
      {
            aux = 0;
            for(k = 0; k < 8; k++)
            {
                  aux = aux + CT[i][k]*Componente[fInicio + k][cInicio
+ j];
            }

            Maux[i][j] = aux;
      }

}

/* SEGUNDA MULTIPLICACION */

for (i=0; i<8;i++)
{
      for (j=0; j<8;j++)
      {
            aux = 0;
            for(k = 0; k < 8; k++)
            {
                   aux = aux + Maux[i][k]*C[k][j];
            }

            Caux[i][j] = aux;
      }

}

/* Actualizacion de la matriz */

for (i=0; i<8;i++)
{
      for (j=0; j<8;j++)
      {

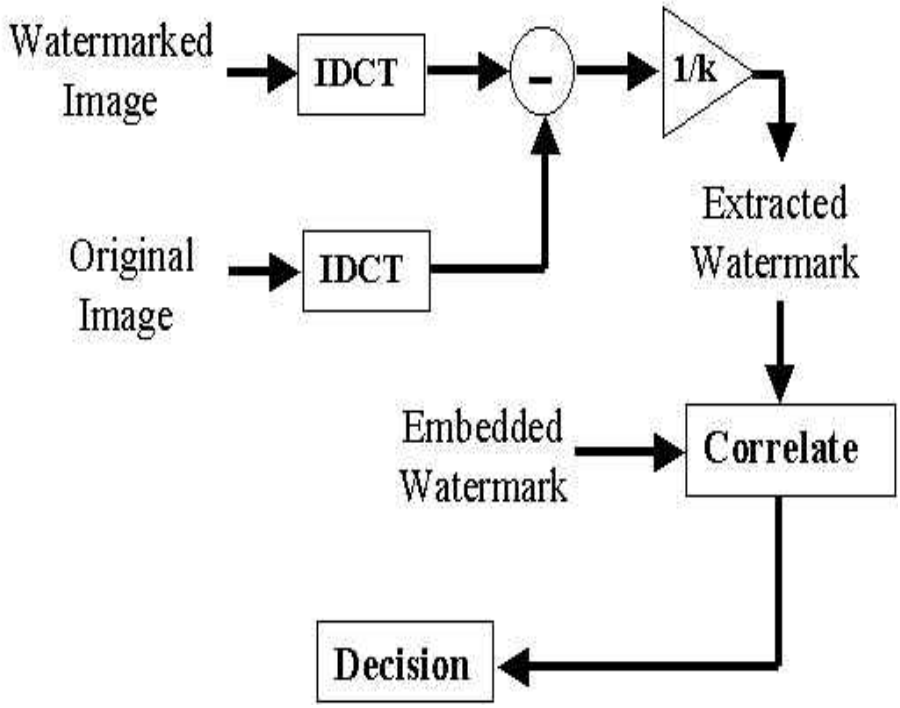
            Componente[fInicio + i][cInicio + j] = Caux[i][j];
      }
}


}
```

Si se realiza todo esto para cada bloque de 8x8 de la imagen resultante, obtendremos como resultado la imagen marcada.

3.2 Recuperación de la marca

El proceso de recuperación es el siguiente:



Esquema de recuperación

Dada una imagen marcada y la imagen original, realizamos un procesado en paralelo de ambas, de forma que aplicamos la IDCT (véase JPEG) a ambas y recuperamos la marca insertada (posiblemente distinta a la original) empleando el factor de energía k.

Comparando la marca insertada y la recuperada podemos determinar si la imagen ha sido alterada o no, según determinemos si se "ha detectado marca o no".

Veamos el proceso en detalle.

El algoritmo consiste en procesar en paralelo la imagen original y la imagen marcada. En cada paso trabajaremos con *un par de bloques de 8x8 píxeles*, uno perteneciente a la imagen original y el otro el correspondiente de la imagen marcada.

A cada bloque del par se le calcula la ITDC y posteriormente se le extrae su zigzag. Con esto obtenemos para cada par de bloques un par de zigzags.

Dicho par de zigzags es investigado para la recuperación de la marca. Esto se consigue *invirtiendo la fórmula de la inserción*, según se describe.

Si c es un píxel del zigzag de la imagen original, y c' es el píxel correspondiente de la imagen marcada, puede recuperarse la marca m insertada en c' mediante la operación:

$$m = (c' - c) / (alfa * c)$$

Si hacemos esto con todos los elementos de los zigzags, podemos obtener, para cada par de bloques, dos vectores de 22 elementos cada uno, W y W', que son respectivamente la marca insertada y la recuperada.

El código que se encarga de realizar esta operación es el siguiente:

```
void Desmarcar8(int cont,double ** MarcaOut, double ZOriginal[22],
                        double ZMarcada[22])

{


double alfa = 0.1;
int i;

      for(i = 0;i < 22; i++)
      {

            MarcaOut[cont][i] =
                  (ZMarcada[i] - ZOriginal[i]) / (ZOriginal[i] * alfa);


      }

}
```

ZMarcada almacena el zigzag de la imagen marcada (vector W'), ZOriginal el de la imagen original (vector W). En MarcaOut se almacena la marca recuperada.

Con estos vectores podemos evaluar la *fidelidad de la marca recuperada*, para determinar si se parece lo suficiente a la original como para considerar que el bloque que la contiene no ha sufrido ninguna alteración. Para ello usamos este razonamiento matemático:

Si tenemos que W * W' es el producto escalar de esos vectores y se verifica que

$$p(W,W') = \frac{W * W'}{\sqrt{W * W'}}$$

entonces diremos que se ha detectado una marca en el bloque si y sólo si p(W,W') >= 0.1.

Si hacemos esto con todos los bloques, podemos evaluar si se ha detectado la marca en el conjunto de la imagen.

Considerando el número de bloques en los que se ha detectado marca, el número de bloques en los que no se ha detectado marca y el número total de bloques analizados, podemos medir la resistencia de la marca como:

$$Z = \frac{\text{nº de bloques en los que se detecta marca}}{\text{nº total de bloques analizados}}$$

y concluir que se ha detectado marca en la imagen si y sólo si Z >= 0.8 .

Esto supone considerar como mínimo una resistencia de la marca del 80% y es un corte que hemos impuesto nosotros tras probar el algoritmo con diversas imágenes.

El código que se encarga de realizar esta última parte del algoritmo es el siguiente:

```
void MarcaDetectada8(int tamMarca,double ** MarcaIn,
                        double ** MarcaOut, int res[2])
{


int k;

int si_detectada = 0;

int no_detectada = 0;

double theta = 0.1;

double producto = 0.0;

int cont = 0;


      for (cont=0; cont<tamMarca;cont++)
      {
            producto = 0.0;

            for(k = 0; k < 22; k++)
            {
            producto = producto + MarcaIn[cont][k]*MarcaOut[cont][k];
            }

            producto = producto / sqrt(producto);

            if (producto >= theta)
                  si_detectada++;
            else
                  no_detectada++;
      }


res[0] = si_detectada;
res[1] = no_detectada;

return res;

}
```

Este código realiza la correlación expuesta almacenando en el vector res el nº de bloques en los que se detecta marca y el nº en los que no se detecta marca.

La decisión sobre la detección no se ha implementado, pues se deja en manos del usuario para lo cual se le comunica el porcentaje de resistencia de la marca.

4. Consideraciones sobre la aplicación desarrollada

Queremos en este apartado comentar la forma adecuada en la que se puede utilizar la aplicación desarrollada para realizar el estudio comparativo de marcas de agua digital en JPEG y JPEG2000. Aprovecharemos para comentar también algunos pormenores de su funcionamiento.

Una buena metodología es la siguiente:

Se parte de una imagen JPEG/JPEG2000

- Se convierte a BMP (descompresión: ataque número 1) (*)
- Se inserta una marca de agua
- Se convierte al formato original (compresión: ataque número 2) (*)
- Se convierte de nuevo a BMP (descompresión: ataque número 3) (*)
- Se recupera la marca
- Se compara con la marca insertada para determinar su integridad

(*) = Estos pasos no tiene que realizarlos el usuario ya que la propia aplicación se encarga de ello, pues internamente trabaja con imágenes en formato BMP. Por lo tanto, al estudiar la marca siempre se realizan estos tres ataques como mínimo. Este factor debe tenerse en cuenta al realizar el estudio.

Si se hace el proceso sobre una imagen tanto en JPEG como JPEG2000 se puede realizar la comparación.

Si además para recuperar la marca usamos como imagen marcada una que se haya comprimido y descomprimido varias veces, es posible estudiar la evolución de la resistencia a sucesivos ataques.

También puede resultar interesante como paso previo a la recuperación aplicar algún efecto digital a la imagen marcada, para estudiar los efectos de este tipo de ataques.

5. Conclusiones

Queremos en este apartado exponer brevemente las conclusiones que hemos extraído de la realización del presente trabajo.

En lo que respecta a conclusiones sobre el estudio comparativo que es la meta del trabajo, tenemos que decir que hemos observado una variación muy exigua en la resistencia de las marcas de agua al trabajar con JPEG y JPEG2000 (menor al 3%), lo cual puede resultar en principio chocante.

Tras meditarlo detenidamente, hemos llegado a la conclusión que ello se debe a los siguientes motivos:

- La virtual inexistencia de imágenes originales en JPEG2000 para las pruebas, debido a la novedad del formato, lo que nos ha obligado a trabajar con imágenes previamente comprimidas a JPEG y luego a JPEG2000. Esto puede alterar los resultados, si es el ataque de JPEG el que mayormente afecta a las marcas de agua.

- La no implementación del esquema de marcado mediante *wavelets* para JPEG2000 debido a la escasa documentación sobre el tema.

A pesar de lo anterior, tenemos que decir que, con todo, estamos satisfechos con el trabajo realizado, ya que nos ha permitido investigar en áreas poco exploradas por nosotros, como son el tratamiento de imágenes y el desarrollo de interfaces gráficas para aplicaciones.

Como último apunte, indicar que para el manejo de JPEG y JPEG2000 se han usado librerías de terceros (consultar bibliografía).

6. Bibliografía

Documento base del trabajo:

"A comparative study of digital watermarking in JPEG and JPEG 2000 environments"

M.A. Suhail (a), M.S. Obaidat (b), S.S. Ipson (b), B. Sadoun (c)
(a) Monmouth University, UK
(b) University of Bradford, UK
(c) Al-Balqa' Applied University (BAU), Jordan

Bibliografía adicional:

JPEG

Página web sobre JPEG
www.jpeg.org

Apuntes de la asignatura PID

JPEG2000

Página web sobre JPEG2000
www.jpeg2000.org

JPEG2000. Compresión Estándar de una Imagen.
Michael Adams

JPEG2000. The Next Generation Still Image Compression Standard
Ericson Research, University of Patras

An Overview of JPEG2000
Proc. Of IEEE Data Compression Conference

The JPEG2000 Still Image Coding System: An Overview.
IEEE Transactions On Consumer Electronics. Vol. 46, Nov 2000

Marcas de agua

Trabajos dirigidos de años anteriores sobre marcas de agua en la asignatura PID

Página web sobre marcas de agua
http://ise.stanford.edu/class/ee368a_proj01/dropbox/project06/#dct

Visual Basic (implementación)

Manual de usuario de Visual Basic

Librerías utilizadas

Para JPEG
Código que acompaña al libro Compressed Image File Formats, John Miano

Para JPEG2000
Aplicación Jasper, obtenida en www.watermarkingworld.com