UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA
DIVISIÓN DE INGENIERÍA ELÉCTRICA

"PROTECCIÓN DE LA PROPIEDAD INTELECTUAL EN IMÁGENES UTILIZANDO MARCAS DE AGUA DIGITALES"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A N:

LOGAN CANDIA RIVERA
RICARDO ENRIQUE IZQUIERDO VALENCIA

DIRECTOR: DR. VÍCTOR GARCÍA GARDUÑO.

CD. UNIVERSITARIA, MÉXICO, D.F. DICIEMBRE 2004

A mi padre
A mi madre
A mi hermana

Logan Candia Rivera

A mi madre, a quien debo todo lo que soy y por quien quiero seguir creciendo
A mi padre, por ser el hombre que algún día quisiera llegar a ser

A mis hermanas que cantaban en la sala y que
a veces se metían a mirar qué tanto hacía

A mi abuelo Juan. No tuve tiempo para darte esta satisfacción
A mi abuela Rubi, a mis primos y tíos

A mis compañeros y profesores de la facultad, por enseñarme que siempre hay algo que aprender
A mis amigos, que hicieron de mi paso por la facultad una experiencia maravillosa

A María. Cada vez que me llegué a sentir perdido, podía ver tu luz a la distancia. Gracias por darme una razón para querer seguir. Esto habría sido muy difícil sin ti.

Ricardo Enrique Izquierdo Valencia

Índice

PROTECCIÓN DE LA PROPIEDAD INTELECTUAL EN IMÁGENES UTILIZANDO MARCAS DE AGUA DIGITALES

Introducción

Watermarking

Los tiempos modernos han venido marcados por el boom de la internet y la información digital. La información en diferentes tipos de formatos se coloca en los servidores y puede ser vista, y modificada, por personas en todo el mundo. Esto ha creado una gran preocupación para los propietarios de dicha información; preocupación que va desde la autenticidad de la información hasta asuntos relacionados con los derechos de autor: la protección de la propiedad intelectual se ha visto más y más amenazada en estos últimos años, pues se ha visto cómo es posible realizar miles de copias de un documento, ya sea de audio, de video, de imagen, etc.; con una calidad muy cercana a la original y por supuesto, a un precio mucho menor.

Las técnicas de marcas de agua son utilizadas para la autenticación (tanto del distribuidor o propietario legal, como de que el original no ha sido falsificado) de la información, así como para el seguimiento de copias, ya que permiten la identificación del autor, propietario, distribuidor y/o consumidor autorizado de un documento digital. Esta técnica de protección requiere básicamente dos herramientas:

- Introducción de la firma o marca en la información a proteger.
- Extracción e identificación de la marca.

Las técnicas de inserción existentes se pueden clasificar en dos grupos, en función del tipo de elemento de la imagen al que la marca de agua afecta de manera directa:

- Técnicas en el dominio del espacio: la marca modifica directamente el valor de luminancia y/o crominancia de los pixeles.
- Técnicas en el dominio de la frecuencia: la marca modifica directamente el valor de los coeficientes espectrales de la imagen. La mayor parte de las técnicas desarrolladas en este dominio están inspiradas en métodos de codificación y compresión.

Las técnicas de decodificación de la marca de agua también se pueden clasificar en dos grupos, según sea que necesiten o no la información original para extraer la marca. Los sistemas de marcas de agua en que se utiliza la información original para la detección de la marca se denominan sistemas privados, en caso contrario reciben el nombre de sistemas públicos.

Una vez que la marca de agua ha sido introducida en un documento digital, es susceptible de un amplio abanico de ataques que la distorsionarán, así como al documento en el que está inserta. Según la causa y objetivo que los origina, éstos se pueden agrupar en ataques no intencionados e intencionados.

Los ataques no intencionados son aquellos a los que la marca de agua está sometida de manera casi inevitable. Ejemplos claros son:

- El propio proceso de recuantificación del documento marcado antes de ser expedido.

- El ruido introducido por el canal de transmisión por el que se envía dicho documento marcado [1].

Entre las propiedades deseables de un sistema de marcas de agua se encuentran la robustez, la resistencia a las manipulaciones, imperceptibilidad, el costo computacional y la baja probabilidad de error.

Robustez

Los archivos digitales de imágenes, audio y video, están expuestos a muchos tipos de distorsiones o modificaciones: las pérdidas por compresión, los cambios producidos por el mejoramiento de imágenes, la amplificación de las señales de audio, etc. Una marca de agua se considera robusta si perdura después de esas operaciones, y en el caso de las marcas en imágenes y en video, también deben persistir después de las transformaciones geométricas (recortado, rotación, escalado). Esto quiere decir que la marca ha de estar presente en los archivos y que debe ser detectada después de las distorsiones. De acuerdo a los trabajos de Cox, para consolidar su robustez, los sistemas de marcas de agua deben insertar la marca en las regiones perceptualmente significativas de los archivos multimedia [2].

Existen diversas opiniones en cuanto a la definición de robustez se refiere. Nuestro punto de vista coincide con las formulaciones en los trabajos de Jiri Fridrich entre otros, por tanto aclaramos que la valoración de esta propiedad de los sistemas de marcas de agua, no incluye los ataques basados en el conocimiento de los algoritmos de incrustado y detección de la marca, la robustez significa resistencia a ciegas frente a aquellas modificaciones producidas por las operaciones comunes a las que estarán expuestos los archivos multimedia.

La robustez no debe exigirse incondicionalmente, ya que un sistema de marcas de agua puede necesitar ser robusto respecto a determinados procesos y frágil respecto a otros. Si un sistema de marcas de agua requiere que ciertas modificaciones de los archivos dañen la marca se le denomina sistema de marcas de agua frágiles y es muy importante en determinadas aplicaciones como veremos más adelante.

Resistencia a manipulaciones

La resistencia a manipulaciones de un sistema de marcas de agua es un aspecto que puede relacionarse con la seguridad del mismo; se refiere a su resistencia frente a los ataques hostiles basados en el total conocimiento de los algoritmos de incrustado y detección y de los archivos marcados, excepto de la clave utilizada. Se incluyen aquí los ataques a los protocolos y los ataques basados en la estimación del sistema. Según la aplicación de que se trate, unos ataques serán más importantes que otros. En general, un ataque efectivo deberá eliminar la marca de agua sin cambiar la calidad perceptual del archivo en cuestión; sin embargo, existen varias aplicaciones en las que la resistencia a determinadas manipulaciones es un aspecto indeseable. A continuación se revisan algunos de los ataques básicos a los sistemas de marcas de agua [3]

Ataques activos. Se trata de eliminar la marca o de hacerla indetectable, algo crítico para muchas aplicaciones incluyendo la identificación de propietarios, pruebas de propiedad, etc. En otro tipo de aplicaciones, como las de autenticación (cuyo objetivo es comprobar la integridad del archivo) el ataque activo consistiría en la restauración de una marca dañada por una manipulación no autorizada. Entre los ataques activos pueden catalogarse los siguientes:

- Ataques por promediación estadística que, en general, intentan estimar la marca y extraerla del archivo marcado.

- Ataques al detector de la marca. El intruso intentará por diversos medios deducir el comportamiento del detector, encontrando las posibles regiones donde se inserta la marca y tratando de eliminarla. Existen diversas versiones de este tipo de ataque.
- Ataques al dispositivo insertador de la marca, en un sistema donde el usuario tiene acceso a éste (aplicaciones de control de copias donde se requiere de un sistema de administración de la generación de las copias). Este puede hacerse, por ejemplo, en los sistemas de DVD donde el usuario tiene acceso a un disco original con una marca que permite una sola copia. Una vez que ésta se realiza, el dispositivo insertador incrustará una nueva marca que prohibirá efectuar más copias. El ataque necesita el acceso al archivo original (con la primera marca) y a su versión marcada (no más copias), con lo cuál se puede estudiar el comportamiento del insertador y crear un archivo diferencia restando el archivo original del marcado, de manera que podría predistorsionarse el archivo original con la finalidad de deshacer la adición de la nueva marca que haga el insertador.
- Ataque por Confabulación: Es otro tipo de ataque activo en el cual se utilizan varias copias de los datos marcados con diferentes marcas (dentro de un mismo sistema) para construir una copia que no contenga la marca. La resistencia a estos ataques es crítica en las aplicaciones de marcas de agua transaccionales explicadas posteriormente ya que estos sistemas incrustan marcas diferentes sobre un mismo original del producto, cuando este va dirigido a varios destinatarios.
- Falsificación: Se trata de insertar una marca falsa que pueda ser reconocida como válida. Es un ataque crítico cuando utilizamos las marcas de agua para autenticar.

Aunque se han mencionado los principales tipos de ataque, evidentemente, existen un sin número de ataques a las marcas de agua que no hemos mencionado. Una herramienta de ataque a los sistemas de marcas de agua que ha servido como estándar de referencia para determinar su robustez, es el Stirmark [4]. Sin embargo, para obtener la evaluación completa de un sistema de marcas de agua, robustez y seguridad, deben aplicarse también otros estándares de comparación que tomen en cuenta el resto de ataques como, por ejemplo, los basados en la estimación.

La carencia de un estándar de referencia que evalúe y certifique oficialmente a los sistemas de marcas de agua, es uno de los problemas más importantes a resolver por la comunidad de científica dedicada al tema, recientemente los investigadores involucrados en el proyecto Europeo Certimark (CERTIfication for waterMARK) han dado a conocer una aplicación orientada a la evaluación de los software de marcas de agua denominada Checkmark, que incluye nuevas funcionalidades que no se habían tenido en cuenta en el Stirmark [5].

Ataques pasivos

En este caso, el intruso no pretende eliminar la marca, su intención es detectarla solamente. En la mayoría de las aplicaciones este ataque no tiene mayor trascendencia, en todo caso, serviría como fuerza disuasoria frente a la intención de sustraer el material informático. Sin embargo, es un ataque válido para la aplicación de las comunicaciones encubiertas, ya que su razón de existir se basa en ocultar que la comunicación se está efectuando.

Imperceptibilidad e Indetectabilidad

La imperceptibilidad y la indetectabilidad de las marcas de agua son dos conceptos que tienden a confundirse frecuentemente, aunque son muy distintos y no están relacionados entre sí: la imperceptibilidad o transparencia de la marca tiene como base el comportamiento del sistema perceptual humano. Una marca de agua es imperceptible (transparente), si la degradación que causa en los

archivos donde se ha insertado es muy difícil de apreciar. Este concepto se contrapone al de la robustez, si tenemos en cuenta que un sistema robusto debe insertar la marca en las regiones perceptualmente significativas del archivo. En algunas aplicaciones se puede aceptar una pequeña degradación de los datos, a cambio de lograr mayor robustez o menor costo del sistema. La indetectabilidad en cambio, está relacionada con el modelo estadístico del archivo antes y después de ser marcado. Se dice que la marca es indetectable si después de haberla insertado, el archivo marcado conserva las mismas propiedades estadísticas que su original. Lo que quiere decir que una persona no autorizada no podrá detectar la presencia de la marca utilizando métodos estadísticos. Esta propiedad es muy deseable en el caso de las comunicaciones encubiertas en las que el principal objetivo es ocultar la presencia del mensaje incrustado en el archivo.

Viabilidad del sistema

Toda tecnología que pretende ser comercializada, debe tener en cuenta varios aspectos, entre ellos: el coste computacional, el coste económico y la escalabilidad del sistema. En muchos sistemas, tales como los de audio y video, la marca debe ser insertada y/o detectada en tiempo real, lo que requiere una gran capacidad computacional de los equipos.

En algunas aplicaciones el número de equipos que insertan la marca de agua difiere de la cantidad de detectores, lo que marcará la diferencia de precio entre unos y otros de acuerdo a la aplicación concreta.

Los requerimientos computacionales exigen a los sistemas de marcas de agua simplicidad, pero ésta puede significar la reducción de la resistencia a las manipulaciones. Sin embargo, hay que tener en cuenta que la capacidad de procesamiento se dobla anualmente, de manera que un algoritmo que hoy no nos parezca razonable, podrá rápidamente convertirse en algo factible; es muy deseable diseñar sistemas de marcas de agua que sean escalables con cada generación de computadoras

Baja probabilidad de error

En la mayoría de los sistemas de marcas de agua es muy importante distinguir entre los archivos que contienen una marca y los que no.

La probabilidad de error al detectar una marca debe ser muy pequeña. Se denomina probabilidad de falso negativo a la probabilidad de que, habiendo estado presente una marca en determinado archivo, el detector asuma que no hay tal marca. Por otro lado, la probabilidad de falso positivo es la probabilidad de que no estando la marca presente en un archivo, el detector asuma que la marca está presente.

En algunas aplicaciones interesará minimizar la probabilidad de falsos positivos. Por ejemplo, en el caso de las restricciones de copias en los DVD, si un equipo de éstos detecta la existencia de una marca falsa no leerá la pista y le dará muy mala reputación a las firmas que lo comercializan.

Este requisito debe ser debidamente probado a priori, con independencia de la aplicación, si se quiere que el sistema pueda ser utilizado en disputas legales.

Cuando se habla de razón de falso positivo, se refiere a la relación entre el número de detecciones que pueden resultar falsos positivos y el número total de detecciones realizadas en un sistema dado. El consenso general sobre los detectores de marcas de agua para video DVD, ha determinado que la razón de falso positivo debe ser de 1 por cada 1012 imágenes.

Marcas de agua múltiples

Los discos de video DVD pueden contener una marca que indique la posibilidad de realizar una sola copia del mismo; una vez que esta copia se ha realizado es necesario alterar la marca para prohibir copias posteriores. En general, es recomendable insertar una segunda marca de manera que ambas sean igualmente detectables y no interfieran entre sí.

El hecho de que puedan coexistir múltiples marcas de agua facilita también el seguimiento de un archivo multimedia desde su punto de confección hasta sus distribuidores y compradores, pudiendo cada uno de ellos insertar su propia marca. En este escenario, el hecho de que las marcas insertadas no interfieran entre sí, significa que cada usuario autorizado podrá detectar su marca.

Un parámetro de especial interés a tener en cuenta aquí es la tasa de bits de la marca de agua (data payload), es decir, la cantidad de información que ésta contiene y se expresa en número de bits. Este parámetro brindará información acerca de cuántas marcas de agua distintas podrán insertarse en un archivo dado. En general, el diseño de sistemas de marcas de agua que permitan la inserción de múltiples marcas debe ser extremadamente cuidadoso para evitar los riesgos de un ataque por confabulación.

Como se ha visto anteriormente, la mayoría de los sistemas de marcas de agua, involucran un compromiso entre la robustez deseada, la tasa de bits de la marca y la imperceptibilidad. Es evidente que estos requerimientos no pueden optimizarse al mismo tiempo, y que el tipo de compromiso entre ellos dependerá rigurosamente de la aplicación.

Objetivos de la presente Tesis de Licenciatura

En la presente Tesis, intentaremos emplear las técnicas de marcado de agua digital, junto con las características del sistema visual humano y la codificación de imágenes, según descrita por la norma JPEG2000, para insertar en imágenes de mapas de bits marcas de agua digitales que permitan proteger la propiedad intelectual de el o los dueños legales de las imágenes.

Trataremos en todo momento que dicha marca de agua digital presente las características de robustez, imperceptibilidad, indetectabilidad y bajo costo computacional, todo esto con un esquema de marcas de agua públicas, es decir, en el detector no se necesitará la presencia de la imagen original para poder determinar si la marca de agua se encuentra o no en la imagen atacada o modificada por ruido u operaciones comunes de procesamiento digital de imágenes: serán necesarias solamente la llave de inserción, la imagen atacada o modificada y la imagen marcada.

El segundo de los objetivos finales será el desarrollo e implementación de un sistema de Software que ejemplifique los conceptos abordados por esta tesis, mostrando en la práctica los procesos de inserción, detección y algunos de los ataques más comunes a los que puede someterse una imagen de mapa de bits con una marca de agua digital.

Referencias

[1] Publicado en el Boletín del Criptonomicón #64. José M. Martínez es doctor ingeniero de telecomunicación, del Grupo de Tratamiento de Imágenes del Dpto. de Señales, Sistemas y Radiocomunicaciones de la E.T.S. Ing. Telecomunicación de la Universidad Politécnica de Madrid.

[2] I. J. Cox, J. Kilian, T. Leighton, y T. Shamoon, "A secure, robust watermark for multimedia".

Ed. R. Anderson, No. 1174, pp. 185–206, Mayo/Junio 1996.

[3] F. Hartung, J.K. Su, B. Girod "Spread Spectrum Watermarking: Malicious Attacks and Counter-Attacks" Proc. of SPIE Vol. 3657: Security and Watermarking of Multimedia Contents, San Jose, CA, USA, Jan. 1999.

[4] http://www.petitcolas.net/fabien/watermarking/evaluation/index.html

[5] http://www.certimark.org

[6] http://www.iec.csic.es/criptonomicon/articulos/expertos64.html, artículo publicado por José M. Martínez

[7] http://www.iec.csic.es/criptonomicon/articulos/expertos88.html, artículo publicado por Amalia Beatriz Orúe López

1. MARCAS DE AGUA

1. a Técnicas en el dominio del espacio

1.a.1 Operando en el dominio del espacio

El marcado en el dominio del espacio, consiste en un una operación para modificar directamente el valor de luminancia y/o crominancia de los pixeles de una imagen. Es importante recalcar que tanto en la literatura encontrada como en la práctica, las marcas de agua insertadas en el dominio del espacio son de fácil implementación desde el punto de vista computacional, ya que son algoritmos más rápidos y más eficientes, aunque son muy frágiles a una larga lista de ataques externos. Por otra parte, las otras técnicas de inserción de marcas de agua poseen un mayor grado de robustez, pero tienen una mayor complejidad computacional. Como otra mención acerca de las cualidades de esta técnica, las técnicas en el dominio de la transformada tienen en general una naturaleza de bloques y esto les otorga un desinterés sobre la degradación de la imagen original. En las técnicas en el dominio del espacio, en cambio, la degradación de la imagen original con la inserción de la marca de agua puede ser controlada localmente, dejando nuestra región de interés sin modificaciones observables.

El marcado en el dominio del espacio posee varias características deseables para un marcado eficiente:

En el dominio del espacio es posible distribuir la marca de forma estadística con un enfoque en el conocimiento de las características del sistema visual humano, con la finalidad de que a la percepción de un atacante o un usuario final de la imagen, sea insospechable que la imagen que posee tenga una marca de agua, y por lo consiguiente, que trate de leerla o modificarla. Como ejemplo cabe mencionar, que si en una imagen existe una redundancia de algún valor con significativa frecuencia, la marca podría quedar comprendida en ese conjunto de valores.

Para la definición de la marca, podemos decir que se utilizaran cadenas de caracteres, las cuales se pretenden insertar en la imagen digital.

Método de marcado de una imagen digital robusto e invisible en el dominio del espacio por medio de modelos estadísticos de inserción

Este método es una propuesta de Santi Prasad Maity y Malay Kumar Kundu, desarrollado en la India, y basado en un modelo estadístico para la inserción y extracción de un símbolo de marca de agua de dos niveles, obteniendo una inserción robusta e invisible, utilizando el conocimiento de la imagen sobre su promedio de luminancia y/o crominancia e insertando, el principio de la selección del trozo de imagen requerido para la inserción de la marca de agua, esta basado en la varianza de el trozo y la inserción de la marca de agua explota el promedio de la brillantez de los bloques.

El proceso de recuperación de la marca de agua no requiere ni la imagen original, ni la imagen marcada, solamente se requiere de la imagen secreta.

Principios del marcado en una imagen digital en el dominio del espacio

El proceso genérico de recuperación de la marca de agua necesita la imagen marcada, la llave secreta o la llave pública y dependiendo del método, la imagen original y/o la marca de agua insertada en la imagen, dependiendo de la combinación de las entradas y salidas de estos elementos se les llamó

sistema de recuperación de marcas de agua: privado, semi-privado o público, que a continuación se describe.

Todos los métodos de inserción de marcas de agua en imágenes digitales, poseen el mismo esquema de trabajo, un sistema de incrustación deberá tener como entradas: imagen original (I), el símbolo de la marca de agua (W) y una llave secreta o pública (K). La salida de el proceso de incrustación de la marca de agua siempre es el mismo, la imagen marcada (I').

El proceso genérico de recuperación de la marca de agua, necesita como entradas: la imagen marcada, la llave secreta o una llave publica y dependiendo del método, la imagen original o el símbolo de la marca de agua, y como salida se obtendrá la marca de agua recuperada (W) con cierto grado de confidencialidad sobre el estado de la imagen, resultando si esta tuvo algún tipo de ataque o modificación.

Marca de Agua Privada (también llamada marca de agua visible "non blind")

La cual requiere al menos la imagen original y/o la marca de agua y la llave, para recuperar la información oculta.

Marca de Agua Pública (también llamada marca de agua invisible "blind" u "oblivious")

Este sistema requiere necesariamente la imagen original y la llave secreta, pero solo en la detección de la información oculta pero no es requerida la marca de agua insertada en la imagen (I" X k → W).

Marca de Agua Semi-Privada (también llamada marca de agua semi-invisible "semi-blind")

Se podría ver estos sistemas como una subclase de los sistemas invisibles, y la detección u obtención de la información sólo es posible con la presencia de la marca de agua insertada y la ayuda de la llave secreta, pero sin la necesidad de la imagen original(I" X k X w) →(0,1)

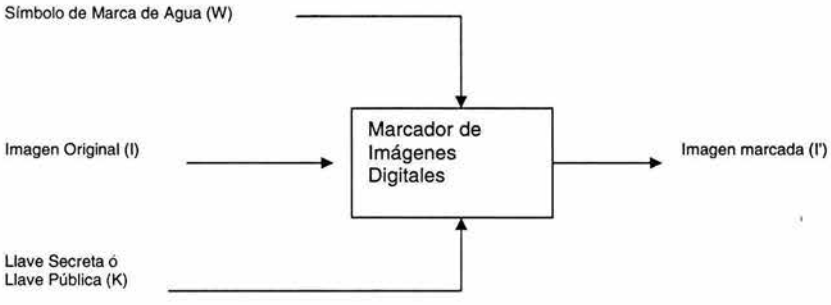


Fig 1.a.1-1 Sistema genérico de incrustación de marca de agua.

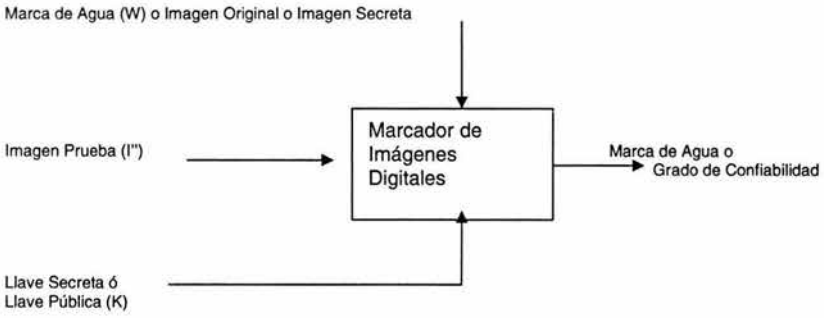


Fig 1.a.1-2 Sistema genérico de recuperación de la marca de agua.

Características para la Inserción y Extracción de la marca de agua

La imagen original (I) es una imagen de escala de grises de tamaño N X N en donde $N=2^p$. Y una marca de agua digital (W) es un símbolo de dos niveles de tamaño M X M en donde $M=2^n$. Definiendo las relaciones que deben de existir entre los valores p y n, p>>n y la relación de (p/n) debería ser del orden de 4. En este escrito propone, trabajar con una imagen binaria de tamaño (16 X 16) como la marca de agua y una imagen de tamaño (256 X 256) como la imagen original la cual se desea marcar.

Inserción de la marca de agua en el dominio del espacio

El algoritmo en el dominio del espacio es usado como una marca de derechos de copia, con un logo invisible, tomando a la imagen original como una región homogénea y tomando partido de la luminancia y/o crominancia de los pixeles.

Paso 1

La imagen original es particionada en bloques cuadrados de tamaño (8 X 8) pixeles que no se traslapen uno con el otro. Un bloque es denotado por su localización de si píxel inicial (x,y). SI la imagen original es de tamaño (N X N), entonces el número total de bloques para la inserción de la marca de agua será dado por (N/8 X N/8). Como siguiente punto, todos los bloques, son ordenados en orden ascendente basándose en su valor de varianza obtenidos. La varianza (σ^2) de cada bloque de tamaño (m X n) se obtiene con la siguiente expresión:

$$\sigma^2 = \frac{1}{mn}\sum_{x=0}^{m-1}\sum^{n-1}\left[f(x,y)-\mu\right]^2$$

En donde el valor promedio de cada bloque es obtenido por:

$$\mu = \frac{1}{mn}\sum_{x=0}^{m-1}\sum_{y=0}^{n-1} f(x,y)$$

Los bloques que tienen valores de varianza pequeños, pueden ser llamados bloques homogéneos y, por supuesto, los valores de varianzas más pequeñas, depende de las características de la imagen que será marcada. Si el Símbolo de Marca de Agua es una imagen binaria de (N X N), solo un número de bloques

N2 será suficiente para insertar un pixel de la marca de agua, en cada uno de los bloques homogéneos de la imagen original.

Un mapa de dos niveles de tamaño (N/8 X N/8) esta construido basándose en la localización de los bloques homogéneos de la imagen original asignándole a cada bloque homogéneo de la imagen original el valor "1" y a todos los demás bloques de la imagen el valor "0". Este mapa de dos niveles de tamaño después se modificará a una imagen multi-nivel, la cual llamaremos imagen secreta (s) y será usada para la extracción de los pixeles de la marca. Esta conversión de una imagen multi-nivel a un mapa de dos niveles se describe en el paso 3.

Paso 2

En el esquema propuesto, un pixel de la marca de agua es insertado en cada bloque homogéneo. Después de la inserción, la marca de agua binaria es espacialmente dispersa usando un sistema caótico llamado "automorfismo del toro" (torus automorphism). Básicamente, el automorfismo del toro es un tipo de permutación de imagen independiente, que genera un número pseudo-aleatorio de longitud variable. Este número es generado usando Registro de Desplazamiento Lineal con Realimentación LFSR por sus siglas en ingles Linear Feedback Shift Register. El número pseudo-aleatorio utilizado en este caso es de una longitud de 256 y la distribución de la marca de agua obtenida, esta denotada por L.

Paso 3

Para las imágenes de dos niveles formadas en el paso anterior, los bloques deseados de la imagen original, son seleccionados y organizados estadísticamente por los valores de sus promedios, utilizándolos para la inserción de la marca de agua. Para cada uno de los bloques semejantes esos valores promedio y sus partes integrantes serán denotados por A y $\overline{A} = \lfloor A \rfloor$ respectivamente. Ahora, un pixel perteneciente a L1, remplazará a un bit en particular, preferentemente el bit menos significativo, un bit plano en representación de A por cada bloque homogéneo. La selección de un bit en particular de un plano de bit puede ser determinado basado en las características (regiones de altas frecuencias o bajas frecuencias) del bloque.

La selección del bit plano está fuertemente gobernada por las características generales de la imagen original a los lados del lugar apropiado del bloque candidato. Puede ser como un promedio de la escala de grises del bloque. Por cada bloque de valor de varianza baja (bloque homogéneo), el bit plano mayor puede ser escogido con tal condición que el valor medio de grises en el bloque sea menor que T1 que sea mayor que T2, donde T1 y T2 son valores especificados, con anterioridad, y T1 debe estar cercano al '0' (el valor mínimo) y T2 debe estar cercano a '255' (el valor máximo). De tal forma que la cercanía entre T1 y T2 es relativa, y fuertemente dependiente de la imagen original. La selección de los valores T1 y T2 no debe de permitir que la calidad de la imagen original sea degradada o afectada por la inserción del logo o marca de agua.

La imagen secreta multi-nivel es construida por la inserción de los valores de posición de los bits seleccionando diferentes bloques homogéneos localizados en la posición '1' de la imagen secreta. Esta información posicional es un valor en la escala de grises de la imagen secreta y nos ayudará a extraer el pixel de la marca de agua de la apropiada posición de ese valor en la escala de grises del bloque.

La inserción de la marca de agua mantiene todos los valores de cada pixel de un bloque homogéneo sin cambio, incrementando o decrementando un valor determinado un valor constante (basado en la apropiada selección del mapa de bits).

Paso 4

La elección en el plano de bits más significativo de menor orden puede resultar una marca de agua muy robusta, pero a costo de una gran distorsión visual de la imagen original. La manipulación de bits se hace para minimizar esta aberración y contar los efectos de alisamiento de la imagen. El proceso

efectivamente cambia esos valores medios de los bloques que están siendo usados en la inserción de la marca de agua. La implementación de esto está basada en un estimado de la posibilidad de un probable cambio en el valor medio por medio de un filtro de mediana. Una máscara de gran tamaño, podría ser de (7 X 7) y es usada para ajustar a conveniencia los valores de gris de todos los pixeles del bloque. El uso de mascaras reduce la distorsión visual en un cincuenta por ciento de las veces.

Extracción de la marca de agua en el dominio del espacio.

La extracción propuesta en este trabajo requiere la imagen secreta (s) y la llave que fueron usadas para la dispersión de la imagen marca de agua (W). La imagen marcada bajo inspección con o sin ataques externos es particionada en bloques cuadrados de tamaño (8 X 8) pixeles que no se traslapen uno con el otro. Ahora para la imagen secreta, la posición del bloque homogéneo son seleccionados y tomando el valor de gris respectivo de la imagen secreta, indicando la correspondiente posición del bit, en los valores de la media de gris, donde el píxel de la marca de agua fue insertado. Desde aquí, la función de la imagen secreta el otorgarnos el valor promedio de los bloques de la imagen marcada o la imagen marcada distorsionada, para poder calcular las diferencias con la imagen original y así, el pixel de la marca de agua es extraído.

La dispersión espacial de la imagen de la marca de agua

La misma llave pseudo-aleatoria (k) y la imagen original es obtenida, como se explica por el proceso de extracción completo.

Una estimación cuantitativa de la cualidad de la imagen marcada extraída W'(x,y) con referencia en la imagen original W(x,y)puede ser expresada y normalizada como una correlación cruzada NCC en donde:

$$NCC = \frac{\sum_x \sum_y W(x,y)w'(x,y)}{\sum_x \sum_y [W(x,y)]^2}$$

Resultados

En la figura 3 se muestra la imagen original usada y en la figura 4 se muestra la imagen marcada usando como logo oculto el símbolo M mostrado en la figura 11. Peak Signal to Noise Ratio (PSNR) El PSNR de la imagen marcada con respecto a la original es de aproximadamente 42.40 dB y se puede observar que las degradaciones son difícilmente percibidas por el ojo humano. La robustez puede ser observada con la explicación de los siguientes ataques.

Filtro de media

La figura 12 muestra la marca de agua extraída de la imagen marcada después del filtro de media usando una mascara de (5 X 5) con un NCC=0.80. El valor PSNR de la imagen marcada es de 23.80 dB y se muestra en la figura 5.

Filtro Gaussiano

La imagen marcada muestra un valor PSNR de 24.15 dB después de tratar la imagen con dos filtros gaussianos de varianza igual a 1, con una ventana de tamaño (9 X 9), mostrado en la figura 6. La figura 13 muestra la imagen marcada con un valor NCC=0.88.

Filtro de Mediana

La imagen marcada tiene un valor de PSNR igual a 25.22 dB obtenido después de tratarla cinco veces con un filtro de mediana de máscara de tamaño (3 X 3) el cual esta mostrado en la figura 7. La figura 14 muestra la imagen de la marca de agua extraída con un valor de NCC igual a 0.94.

Reescalamiento de imagen

La imagen marcada fue reescalada a la mitad de sus dimensiones originales y después fue reescalada a sus dimensiones originales. En la figura 8 se muestra la imagen modificada con un valor de PSNR igual a 24.85 dB con muchos detalles perdidos. La marca de agua extraída obtuvo un valor de NCC de 0.87 y esta mostrado en la figura 15.

Compresión JPEG

La figura 16 muestra la marca de agua extraída de la imagen marcada con un valor de NCC de 0.958 y de la imagen marcada se obtuvo un valor de PSNR de 18.73 con un nivel de compresión de 45.0 y se muestra en la figura 9. Se observo que incrementando el nivel de compresión se decrementa el valor de NCC de la marca de agua extraída de la imagen marcada y por consiguiente decrementa la calidad del marcado.

Manipulación del bit menos significativo (LSB)

Se tomaron aleatoriamente por todos los pixeles, dos bits menos significativos de la imagen marcada modificándolos y la imagen modificada, arrojó un valor de PSNR de 40.94 dB mostrado en la figura 10. La marca de agua extraída de la imagen marcada tuvo un valor de NCC de 0.88 y esta mostrada en la figura 17.

Ataque de cosecha en la imagen (Image Cropping Operation)

La robustez del método propuesto reta a diferentes tipos de operaciones de siembra que pueden ser adecuados para efectuar el ataque en una imagen marcada en particular.

Conclusiones

La técnica propuesta describe un esquema de inserción de marca de agua robusto e invisible en el dominio del espacio, además de ser computacionalmente eficiente. La inserción de la marca de agua es significativa y reconociblemente igual que una secuencia de números reales con una distribución normal generada por una secuencia pseudo-aleatoria. La técnica propuesta ha sido probada por los ataques más usados. Además ha sido propuesta en la comunidad watermarking y los resultados obtenidos han sido de gran satisfacción. Por otro lado, este trabajo se deberá tomar como iniciador de investigación para explotar algunas otras características de las imágenes como son tamaño, color, ubicación y contorno, y tomar en cuenta los planos de la imagen. También deberá de ser cubierta la robustez a ataques de rotación y manipulaciones geométricas.


Figura 3


Figura 4


Figura 5


Figura 6


Figura 7


Figura 8


Figura 9

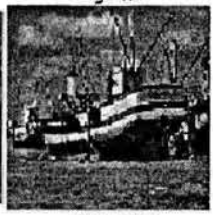


Figura 10


Figura 11


Figura 12


Figura 13


Figura 14


Figura 15


Figura 16


Figura 17


Figura 18


Figura 19


Figura 20


Figura 21


Figura 22

Figura 3: Imagen Original barco pesquero, Figura 4: Imagen Marcada, Figura 5: Imagen Marcada después de un filtro de media usando una máscara de (5 X 5), Figura 6: Imagen Marcada después de atacarla con dos filtros Gaussianos de varianza 1, Figura 7: Imagen Marcada después de tratarla con cinco filtros de mediana usando máscaras de (3 X 3), Figura 8: Imagen Marcada después de reescalamiento, Figura 9: Imagen Marcada después de compresión JPEG (C.R = 45.00), Figura 10: Imagen Marcada después de dos manipulaciones de LSBs , Figura 11: Marca de agua digital, Figura 12: Marca de agua extraída de la figura 5, Figura 13: Marca de agua extraída de la figura 6, Figura 14: Marca de agua extraída de la figura 7, Figura 15: Marca de agua extraída de la figura 8, Figura 16: Marca de agua extraída de la figura 9, Figura 17: Marca de agua extraída de la figura 10, Figura 18: Imagen Original Oso, Figura 19: Imagen Original Nueva York, Figura 20: Imagen Original Lena, Figura 21: Imagen Original Opera, Figura 22: Imagen Original Pastillas.

1. b Técnicas en el dominio de la frecuencia

1.b.1 Operando en el dominio de la transformada

El marcado en el dominio de la transformada aplica alguna transformación reversible a la imagen soporte antes de insertar la marca. Entonces, los coeficientes transformados son modificados para insertar la marca y por último se aplica la transformación inversa para obtener la imagen marcada. Las

transformaciones que comúnmente se usan para esto son la Transformada Coseno Discreta (DCT), la Transformada Discreta de Fourier (DFT), la Transformada Wavelet (WT) y la Transformada Fractal. Hay también algunas otras transformaciones más exóticas, como la transformada Wavelet compleja, la transformada Fresnell, la transformada Fourier-Mellin, etc.

El marcado en el dominio de la transformada posee varias características deseables para un marcado eficiente: al aplicarse la transformación inversa sobre los coeficientes transformados modificados, la marca se expande de manera irregular por toda la imagen, lo que dificulta la labor del atacante que trata de leerla o modificarla. Ejemplos de lo anterior son los métodos de marcado basados en la DCT global. Por supuesto, la transformación Wavelet o la DCT basada en bloques afectan sólo regiones de la imagen: No obstante, lo anterior permite seleccionar sólo ciertas bandas de la imagen soporte para insertar la marca, tomando siempre como base las características conocidas del sistema visual humano.

Usualmente, los algoritmos que operan en el dominio de la transformada, añaden la marca, o su señal de espectro extendido, a un pequeño subconjunto de coeficientes transformados de frecuencias medias o bajas. Para las técnicas de espectro extendido (Spread Spectrum), una señal de banda angosta que representa el mensaje a insertar, es modulada por una señal portadora de banda ancha, lo cual aumenta la banda de la señal original, de ahí el término "Espectro Extendido" [1].

1.b.2 Espectro extendido

Las siguientes características del espectro extendido son particularmente útiles para el watermarking:

Anti-Jamming. Esta propiedad resulta del hecho de que el atacante no cuenta con la información privilegiada con que cuentan en transmisor y el receptor autorizado de la imagen. En consecuencia, el atacante debe analizar todo el espectro de la señal de banda ancha. Aunado a esto, no tiene mucho poder, lo que significa que sólo puede analizar cada frecuencia con poca energía. Aplicada al watermarking, esta propiedad significa que para arrancar la marca, un atacante debe distorsionar imagen marcada de una manera tan severa que la imagen resultante tras el ataque pierde todo valor comercial, o por lo menos disminuye su calidad hasta un extremo no aceptable.

Baja probabilidad de intercepción. Esta probabilidad es consecuencia de la difusión: una potencia grande es distribuida sobre todo el espectro de frecuencia, de este modo sólo una pequeña cantidad de energía se añade a cada frecuencia. Muy a menudo, este incremento es menor que los niveles normales de ruido que podrían agregarse a la señal durante la transmisión, de modo que el atacante bien podría no darse cuenta de que se está transmitiendo una señal de espectro extendido.

Pseudo-ruido. Por seguridad, frecuentemente la portadora es una señal de pseudo-ruido, lo cual significa que posee características estadísticamente parecidas a las de una señal verdaderamente aleatoria, pero que puede ser regenerada exactamente con la ayuda de cierta información privilegiada. Por ejemplo, la portadora podría ser la salida de un generador de números aleatorios inicializado por cierta semilla, conocida sólo por el dueño de la imagen. Esta propiedad es útil para el marcado porque hace difícil para el atacante estimar la marca de la imagen marcada. Además, con la selección de la señal pseudoaleatoria correcta, si bien el atacante podría estimar pequeños fragmentos de la marca, es casi imposible determinarla completamente.

Las componentes de las frecuencias medias y bajas de una imagen representan la mayor parte de la información perceptualmente importante, por lo tanto la compresión y algunas otras operaciones comunes de PDI afectan significativamente estas porciones de la imagen. Agregar la marca a los coeficientes significativos del dominio de la transformación generalmente incrementa la robustez. Los métodos de marcado en el dominio del espacio tienen que modelar indirectamente las componentes de baja frecuencia de la imagen soporte, lo cual puede ser difícil de lograr.

En el dominio de la DCT, la energía se concentra alrededor de la esquina superior izquierda. La DWT de múltiple resolución tiene las componentes de baja frecuencia de la imagen en la subbanda de aproximación, también localizada en la esquina superior izquierda, mientras que las componentes de altas frecuencias se representan en las bandas de detalle de diferentes resoluciones.

Las principales desventajas de las técnicas en el dominio de la transformada son el costo computacional, y en el caso de las transformaciones globales, el problema de adaptar la fuerza de la marca a la actividad local de la imagen, dificultando la explotación de las características del HVS, como los efectos de enmascaramiento, aunque esta última desventaja puede resolverse usando la transformada Wavelet, la cual provee información tanto espacial como de frecuencia para la imagen soporte.

1.b.3 Método de marcado en el dominio DCT por medio de modelos gaussianos generalizados

Este método es una propuesta de J.R. Hernández, F. Pérez González y M. Amado, desarrollado en el departamento de las tecnologías de las comunicaciones de la universidad de Vigo, en España, y se basa en un modelo gaussiano generalizado que incluye como caso especial el detector basado en la correlación cruzada.

Definiciones

Sea x(n) una secuencia bidimensional que representa la luminancia de la imagen original, con n=(n1,n2). Sea X(k) el resultado de aplicar una transformación DCT a x(n), tomando como base bloques de 8x8 pixeles. Una marca de agua W(k) con cierta información se agrega a la imagen original en el dominio DCT obteniendo la versión marcada

$$Y(k)=X(k)+W(k)$$

La marca de agua W(k) se genera en el dominio DCT empleando un esquema bidimensional de modulación de amplitud de multipulsos, o dicho de otro modo, W(k) puede expresarse como la suma de N pulsos ortogonales $P_i(k)$, o bien:

$$W(k)=\sum_{i=1}^{N} b_i p_i(k) \qquad (1)$$

En donde los coeficientes bi=(b1,b2,...,bn), se usan para codificar el mensaje oculto. Los pulsos de modulación Pi(k) se generan como función de una llave secreta, k, conocida sólo por el dueño del copyright. Se expresan como:

$$p_i(k)=\begin{cases}\alpha(k)s(k); k \in S_i \\ 0; k \notin S_i\end{cases} \qquad (2)$$

En donde s(k) es una secuencia pseudoaleatoria que depende de una llave tal que s(k) ε {-1,1}, para cualquier valor de k, y el conjunto de índices T=S_i es también dependiente de una llave y determina la forma espacial de los pulsos. La secuencia $\alpha(k)$ es llamada máscara perceptual e indica la máxima magnitud de alteración permisible que el coeficiente X(k) puede sufrir sin alcanzar distorsiones visibles. Los conjuntos S_i se asumen mutuamente excluyentes, lo cual significa que la intersección de un conjunto S_i con un conjunto S_j es el conjunto vacío, siempre que se cumpla la condición de que i y j sean diferentes. Estos conjuntos se asumen también distribuidos por toda la imagen de manera pseudoaleatoria, a modo de proporcionar seguridad y robustez contra el cropping.

Dada la imagen marcada Y(k) y la llave k, el proceso de extracción obtiene un estimado del mensaje secreto b. Se asume que no se han efectuado ataques para desincronizar la marca.

Estructura del detector

Usualmente, la estructura del detector se basa en la correlación cruzada entre la imagen marcada Y(k) y la secuencia pseudoaleatoria s(k). Este esquema funciona si el ruido originado al marcar la imagen sigue una distribución gaussiana, lo cual para la mayoría de las imágenes no se cumple. Algunos autores proponen la función de densidad de probabilidad gaussiana generalizada como una alternativa para mejorar los modelos estadísticos. A continuación se presenta dicha función:

$$f_x(x) = A e^{-|\beta x|^c} \tag{3}$$

Trabajos previos muestran que los coeficientes DCT a bajas frecuencias son razonablemente bien modelados por la distribución gaussiana generalizada con c=1/2, mientras que a altas frecuencias son mejor aproximados por la distribución gaussiana (c=2) y a veces por la laplaciana (c=1).

Los parámetros A y β de la ecuación 3 se expresan por:

$$\beta = \frac{1}{\sigma}\left(\frac{\Gamma\left(\frac{3}{c}\right)}{\Gamma\left(\frac{1}{c}\right)}\right)^{\frac{1}{2}}$$

$$A = \frac{\beta c}{2\Gamma\left(\frac{1}{c}\right)}$$

En donde σ es la desviación estándar. De este modo la fdp está completamente especificada por c y σ.

Sea la secuencia $C_{i,j}(k_1,k_2)=X(8k_1+i,8k_2+j)$; I,j={0,1,...,7}, la resultante de tomar el (i,j)-ésimo coeficiente DCT de cada bloque. Se modelará cada una de esas 64 secuencias como la salida de un proceso aleatorio bidimensional cuya distribución marginal sigue la ecuación 3, con parámetros c(i,j) y σ(i,j). Sean las secuencias:

$$c(k)=c(k_1 mod 8, k_2 mod 8)$$
$$\sigma(k)=\sigma(k_1 mod 8, k_2 mod 8)$$

Entonces, estas secuencias indican los parámetros c y σ asociados a nuestra X(k).

Si se asume que pueden codificarse M diferentes mensajes usando el vector $b=(b_1,b_2,...,b_N)$, y si b_L, L ε {1,2,...,N} indica la palabra de código asociada a cada uno de esos mensajes; así mismo, si $W_L(k)$, L ε {1,2,...,M} es la marca obtenida a partir de $b_L=(b_{L,1},b_{L,2},...,b_{L,N})$ usando la ecuación 1, entonces, asumiendo el modelo gaussiano generalizado para X(k), puede demostrarse que el detector óptimo es el que escoge el índice L ε {1,2,...,M}, verificando:

$$\sum_k \frac{|Y(k)-W_m(k)|^{c(k)} - |Y(k)-W_L(k)|^{c(k)}}{\sigma(k)^{c(k)}} > 0$$
$$\forall m \neq 1$$

Asumiendo que $b_{L,i} \varepsilon \{-1,1\}$ para cualquier valor de L comprendido en el intervalo {1,2,...,M} y cualquier valor de i en el intervalo {1,2,...,N} , este problema de optimización es el equivalente a hallar la palabra de código que maximiza la expresión

$$\sum_{i=1}^{N} b_{L,i} r_i$$

en donde los coeficientes r_i son estadísticamente suficientes para el problema de detección y se definen como:

$$r_i = \sum_{k \in S_i} \frac{|Y(k) + \alpha(k)s(k)|^{c(k)} - |Y(k) - \alpha(k)s(k)|}{\sigma(k)^{c(k)}}$$

cuando se usa una constelación binaria antipodal para codificar M=2^N posibles mensajes, la estructura del detector ML es equivalente a un decisor bit a bit y sus salidas son:

$$b_i = sgn(r_i),\ i \varepsilon (1,2,...,N)$$

1.b.4 Marca de múltiple resolución basada en la transformada Wavelet

Método propuesto por A.H. Paquet y R.K. Ward en su libro "Wavelet based digital watermarking" [3]. Consiste en añadir códigos pseudoaleatorios a los coeficientes grandes en las bandas de frecuencia altas y medias de la transformada Wavelet de la imagen. Es importante saber que se usa la transformación Wavelet pues sus características son buenas para consideraciones de enmascaramiento, ya que está bien localizada tanto en tiempo como en frecuencia. Aunado a esto, la transformación Wavelet es congruente con el modelo "multicanal" del sistema visual humano, de modo que es posible poner un límite a la alteración de los coeficientes Wavelet a modo de mantenerse debajo de, por ejemplo, la JND 3 (Just Noticeable Difference 3), en la cual el ojo humano empieza a notar las modificaciones hechas a la imagen. Además, la transformación Wavelet es parte de los nuevos estándares de compresión, como el JPEG2000, por lo cual es lógico suponer que las técnicas basadas en la transformación Wavelet proveerán un camino más fácil y optimizado para incluir protección del copyright dentro del mismo código de compresión.

El segundo aspecto importante de esta técnica es la manera en que se introduce la marca en la imagen soporte. Para que una marca de agua sea robusta, debe introducirse en las componentes perceptualmente significativas de la imagen, lo que trae sin embargo el riesgo de alterarla. Esta técnica trata de cumplir con el requerimiento para la robustez hasta cierto punto, mientras que también trata de mantenerla invisible. Esto se logra introduciendo la marca sólo en las frecuencias medias y altas, manteniendo casi sin alteración las frecuencias bajas de resolución 3, a las cuales el ojo humano es más sensible.

El modelo implementado usa una marca de agua del tipo "non-oblivious" o privada, lo cual significa que para realizar la detección de la imagen marcada, es necesario tener acceso a la imagen original. Esto no es tan descabellado si se tiene en cuenta que este esquema de marcado tiene como objetivo la protección del copyright: es lógico suponer que el dueño del copyright tiene acceso a la imagen original.

El modelo de multiresolución y jerárquico ahorra carga computacional en el momento de la detección y permite una buena transparencia pues usa las características de nuestro sistema visual. Se sabe que el HVS es más sensible a las alteraciones pequeñas en las bandas inferiores de frecuencia, por lo tanto, manteniendo intactas las bajas frecuencias se asegura que la imagen marcada será lo más parecida posible a la imagen original. Es importante notar que desde el punto de vista del procesamiento de señales, los requerimientos de robustez y transparencia están en conflicto mutuamente. Este método permite negociar ambos requerimientos.

Antes de iniciar con la descripción del método, analicemos sus objetivos:

- Implementar un esquema de marcas de agua para imágenes digitales basado en la transformada Wavelet.
- Examinar la robustez de la anterior implementación frente a diferentes tipos de ataques, como las operaciones comunes de procesamiento digital de imágenes.

Codificador

El primer paso a realizar en el codificador es descomponer la imagen en diferentes bandas de frecuencia usando tres resoluciones de wavelets Haar. En la figura 1.b.4-1se muestran bancos de filtros que representan la idea de la óctuple banda de las wavelets de Haar, lo cual nos da una estructura piramidal de localización de frecuencias como la que se aprecia en la figura 1.b.4-2.

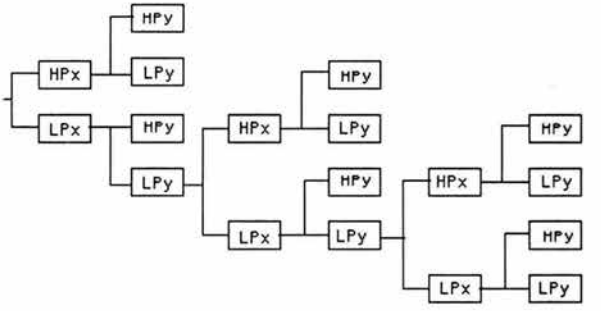


Fig 1.b.4-1 Tres niveles de descomposición Haar

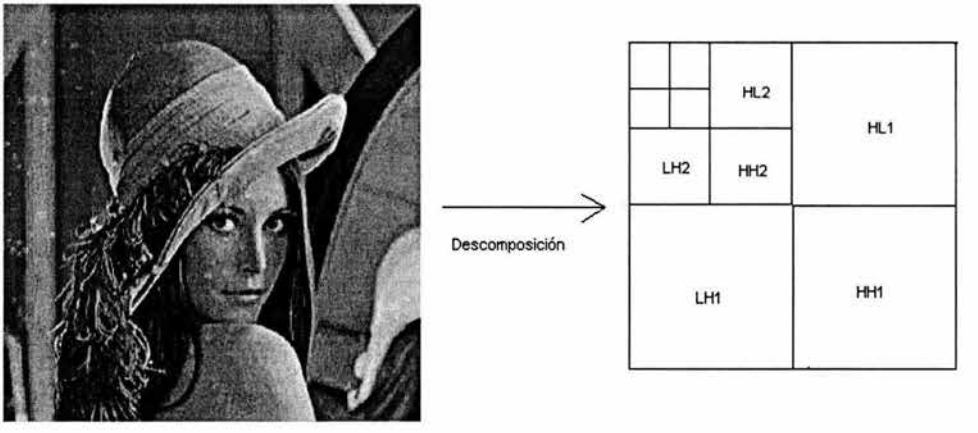


Fig 1.b.4-2 Figura descompuesta resultante

Antes de continuar, debe notarse que tras cada filtrado, se realiza una decimación y que las Wavelets Haar se eligieron por simplicidad.

La siguiente operación es agregar a los coeficientes de las bandas de frecuencia media y alta, una secuencia pseudoaleatoria N, que es de hecho una distribución gaussiana con media de cero y varianza de uno (esto es, a todas las bandas excepto la más baja, representada por la esquina superior izquierda de la figura 1.b.4-2. Se usa la distribución normal pues ha mostrado robustez en contra de los ataques por confabulación.

Para asignar un peso a la marca de agua de acuerdo a la magnitud de los coeficientes wavelet, se usa una de dos relaciones entre los coeficientes originales *y* los coeficientes ÿ, que contienen la marca de agua:

$$\ddot{y}(m,n) = y(m,n) + \ddot{u}\,(y(m,n))^{\wedge}2 * N(m,n)$$

$$\ddot{y}(m,n) = y(m,n) + \ddot{u}\ddot{u}(y(m,n)| * N(m,n)$$

A pesar de que las anteriores fórmulas son matemáticamente diferentes, ambas tienen el mismo objetivo, que es agregar peso a la marca agregada a los coeficientes wavelet de mayor valor. El parámetro ü se usa para controlar el nivel de la marca, y de hecho proporciona una buena manera para escoger entre una buena transparencia, una buena robustez o un equilibrio entre ambas. Finalmente se computa la transformada wavelet inversa de dos dimensiones para formar la imagen marcada ÿ.

Decodificador

Al otro extremo del canal de comunicación, se usa un decodificador para extraer la marca. Al recibir la información supuestamente marcada, el algoritmo primero aísla la firma incluida en la imagen comparando los coeficientes DWT de la imagen con los de la imagen original, sin marcar, entonces se toma la llave identificada para contrastarla con la llave que se halló mediante el cálculo de la correlación cruzada en el primer nivel de resolución (los coeficientes de las más altas frecuencias). Se considera que se ha hallado una marca de agua si hay un pico en la correlación cruzada, correspondiendo a una identificación positiva. Si no hay un pico central, el decodificador agrega el segundo nivel de resolución (el cuadrado inferior izquierdo en la figura 1.4.b-2) a la búsqueda de picos. De nuevo, si se halla un pico, se ha hallado una marca, de lo contrario se procede a analizar la tercera resolución y así sucesivamente hasta alcanzar el noveno nivel de resolución.

La principal ventaja de esta técnica es que mientras permite una buena detección, aún para una imagen corrupta, mantiene el nivel de falsos positivos al mínimo: la firma hallada tiene que atravesar el paso de detección de identificación positiva para considerarse detectada. El paso de detección intenta asegurar la máxima exactitud en la detección del dueño de la llave de identificación y, como se mencionó, disminuir el número de falsos positivos.

1.b.5 Algoritmos aditivos

En los algoritmos aditivos, la información a insertar es una secuencia *w* de números, con una longitud N, la cual es embebida en el conjunto apropiado de coeficientes de la image soporte, *f*. La fórmula básica comúnmente usada para embeber dicha información es:

$$f'(m,n) = f(m,n)*(1+\alpha w)$$

en donde α es un factor de peso y f' es el coeficiente de la imagen soporte resultante, que contiene la información de la marca de agua. J. Cox [1] ha propuesto fórmulas alternas para la inserción de la información, tales como:

$$f'(m,n) = f(m,n) + \alpha * w$$

o, usando el logaritmo de los coeficientes originales:

$$f'(m,n) = f(m,n) * \exp(\alpha * w)$$

Una propiedad importante de la fórmula anterior es que la función inversa de la inserción:

$$w^\circ = (f^\circ(m,n) - f(m,n))/\alpha * f(m,n)$$

puede ser usada para computar w° dada f°, teniendo los coeficientes soporte originales como referencia. Por f° se denota la imagen recibida, posiblemente alterada, que podría contener la marca w. El siguiente paso es comparar la secuencia marcada w° con la secuencia w original usando la correlación normalizada de secuencias como una medida de similitud.

$$\delta = (w^\circ * w) / |w^\circ| * |w|$$

La similitud δ varía en el intervalo [-1,1]. Un valor muy por encima de cero y cercano a uno indica que la secuencia extraída w° coincide con la secuencia original w, lo que nos lleva a concluir que la imagen ha sido marcada con la secuencia w. Puede establecerse un umbral de detección, T para tomar la decisión $\delta > T$. El umbral T puede determinarse experimentalmente observando la correlación de secuencias aleatorias o analíticamente.

Por supuesto, la elección del umbral de decisión tendrá un efecto significativo en la detección tanto de falsos positivos como de falsos negativos. Se ha dedicado un gran esfuerzo a hallar métodos confiables para computar umbrales de correlación y sistemas eficientes de detección de marcas de agua.

El factor de peso α no tiene que ser constante para toda la imagen, en lugar de esto, es posible variarlo a lo largo de ésta para capturar sus características locales y de esta manera poder aplicar más o menos energía en la marca según se requiera, logrando con esto una mayor robustez. Por ejemplo, pueden modelarse y explotarse ciertas propiedades del HVS como el enmascaramiento.

Antes de insertar la marca, la imagen soporte F es usualmente expuesta a una transformación bidimensional como la DCT, la DWT o la DFT, para obtener su representación en frecuencia, f. Después de las modificaciones realizadas por el marcado, se aplica la transformación inversa para obtener nuevamente la representación espacial de la imagen, la cual contendrá la marca de agua insertada.

1.b.6 Extracción de la información

Una de las características de los algoritmos aditivos, tal como los propuso Cox, es que la imagen original debe estar disponible como una referencia para la extracción de la marca de agua. Por lo tanto, usando la correlación, lo único que puede lograrse es *detectar* la información, es imposible (o por lo menos muy tardado) recuperar la información en sí misma, esto debido a que la información embebida es generalmente la semilla de un generador de números pseudoaleatorios. Para recuperar esta semilla sería necesario correlacionar cada uno de los posibles valores de ésta, con la secuencia marcada extraída. Se identifican así dos problemas en este tipo de algoritmos: La disponibilidad de la imagen original y la recuperación de la información.

Para solucionar el problema de la disponibilidad de la imagen original, logrando así un esquema de marcas de agua *oblivious*, algunos autores han propuesto métodos que correlacionan la secuencia

marcada w directamente con todos los coeficientes de la imagen recibida, f° (correlación mutua), usando el teorema del límite central:

$$d = (\Sigma_N f(m,n)^\circ * w) / N$$

y después comparando el valor de la correlación, δ, con algún umbral de decisión, T.

La principal desventaja de este método es que la propia señal debe ser tratada como ruido, lo que hace la detección más difícil: para poder detectar la presencia de la marca de agua, más coeficientes deben estar correlacionados, lo cual decrementa la capacidad y robustez de la marca.

Para solucionar el problema de la recuperación de la información, existen también varios algoritmos. El problema con el uso de la correlación y el umbral de detección es que este esquema sólo puede decir si la marca existe o no en la imagen recibida. Si lo que nos interesa es la información contenida en la marca, se tendrían que probar todas las posibles secuencias, lo cual hasta el momento no es computacionalmente factible dada la longitud de la semilla de los generadores de números pseudoaleatorios usados comúnmente, de por lo menos 32 bits.

Una solución práctica es codificar los bits de la información en una secuencia aproximadamente gaussiana de números reales, la cual se discutirá más a detalle en las páginas siguientes.

1.b.7 Algoritmo de Cox

Propuesto por J. Cox, Joe Kilian, Tom Leighton y Talal G. Shamoon [1]. Trabaja en el dominio de la DCT y fue desarrollado en el NEC Research Institute.

La marca de agua es una secuencia gaussiana de números reales pseudoaleatorios, con una longitud de 1000. Esta secuencia se inserta en los 1000 coeficientes más grandes de la DCT, siguiendo el mismo patrón de zig-zag que usa el estándar JPEG. Hay que mencionar que la imagen se somete a una DCT global, lo cual es computacionalmente costoso.

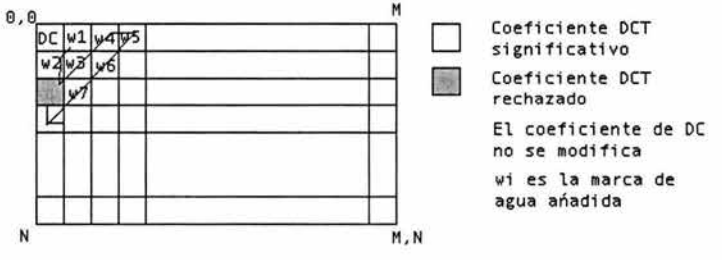


Fig 1.b.7-1 Algoritmo de marcado de Cox.

La inserción de la marca se realiza siguiendo la fórmula expuesta en el apartado anterior, y del mismo modo, la extracción se realiza usando la fórmula de inserción inversa.

1.b.8 Algoritmo de Barni de secuencia Gaussiana

Propuesto por Mauro Barni, et al. Universidad de Siena, Departamento de Ingeniería de la Información. La marca de agua consiste en una secuencia binaria pseudoaleatoria, w ε {1,-1} con una longitud determinada por las dimensiones de la imagen, M y N.

Los autores proponen el uso de filtros Daubechies-6 para realizar una descomposición Wavelet de 4 niveles. Sólo los coeficientes de la subbanda de detalle en el primer nivel de resolución son modificados para insertar la marca de agua, los coeficientes más grandes se usan para calcular la escala para el enmascaramiento visual.

Los coeficientes seleccionados se modifican aplicando la regla descrita anteriormente para la inserción aditiva, aunque se toma en cuenta la sensitividad local al ruido:

$$f'(m,n) = f(m,n) + \alpha * \beta(m,n) * w$$

La función de peso $\beta(m,n)$ toma en consideración la orientación de la subbanda, (LH,HL,HH), el brillo local basado en el coeficiente correspondiente en la imagen de aproximación (la subbanda LL) y la actividad local de textura en el entorno. El último término combina la actividad local en las subbandas de detalle en el nivel más grande y la varianza local de la banda pasa bajas LL, ambas computadas en un vecindario de 2x2 correspondiente a la localidad del coeficiente f(m,n).

$$\beta(m,n) = \Theta(l,o) * \Lambda(l,m,n) * \Xi(l,m,n)$$

En donde el primer término representa la subbanda y la sensitividad del nivel de resolución, el segundo término representa el brillo local y el último término pesa la varianza local de la actividad de la textura.

La marca se detecta correlacionando la secuencia marcada w directamente con los coeficientes de la imagen transformada seleccionados, permitiendo la detección a ciegas.

Este esquema usa un modelo explícito del HVS derivado del problema de la cuantización de los coeficientes. Cada valor binario de la marca se multiplica con un parámetro de peso obtenido del modelo de sensitividad al ruido antes de insertarlo a los coeficientes. De este modo el coeficiente se altera a un grado apenas debajo del umbral JND.

1.b.9 Macado por cuantización dentro del esquema de codificación JPEG2000

Algoritmo propuesto por Peter Meerwald en el departamento de computación científica de la Universidad de Salzbury. Salzbury, Austria. Es otro algoritmo que trabaja en el dominio de la transformada Wavelet, aunque a diferencia de otros algoritmos que trabajan en ese mismo dominio, el autor se ha enfocado a la integración de su algoritmo en el estándar JPEG2000, por medio de una implementación de dicho estándar, llamada JJ2000.

En este método, el mensaje de la marca de agua es una secuencia de N valores $m_n \varepsilon \{0,1\}$. Se usan dos vectores pseudoaleatorios di y la siguiente regla para agregar un bit de la información de la marca de agua a un vector x de la imagen soporte, con una longitud L:

$$s(x_i,m) = Q\Delta(x_i + d_i(m)) - d_i(m) \qquad (1)$$

en donde $Q\Delta(°)$ representa una cuantización escalar con un tamaño de escalón Δ. Para la detección de la marca de agua, se calcula la distancia mínima entre el dato recibido, x* y su punto de reconstrucción, ya sea m = 0 ó m = 1:

$$m^* = \arg_m \min ||x^* - s(x^*,m)||$$

Esquema de codificación JPEG2000

Está basado en el esquema originalmente propuesto por Taubman, conocido como EBCOT (Embedded Block Coding with Optimized Truncation). La más grande diferencia con esquemas de compresión basados en la transformada wavelet como el SPHIT o el EZW es que el EBCOT, al igual que el JPEG2000 [6],[7], trabajan sobre bloques independientes que no se sobreponen, que se codifican en varias capas para crear un flujo de bits escalable.

Los objetivos de diseño son la versatilidad y la flexibilidad, los cuales se alcanzan por medio del procesamiento y codificación independiente de cada uno de los bloques. El default del JPEG2000 es realizar una descomposición wavelet con 7 filtros biortogonales-9 y luego segmentar la imagen transformada en bloques de no más de 4096 coeficientes.

Inserción de la marca

Se realiza después después de la cuantización y el escalamiento de la región de interés (ROI) y antes de la codificación de la entropía (estos aspectos se explicarán más detalladamente en el capítulo 3, en donde se discute el estándar JPEG2000). En este punto, cada bloque transporta coeficientes enteros signados normalizados: el MSB contiene la información de signo mientras que el resto de los bits dan la magnitud absoluta del coeficiente. Se debe distinguir entre los bloques que pertenecen a la imagen de aproximación (subbanda LL) o a las subbandas de detalle (LHj, HLj, HHj en donde j=1). Las subbandas de resolución más finas no pueden usarse para codificar información confiablemente.

Para el primer caso se aplica una ventana sobre todo el bloque. En cada posición de la ventana, se codifica un bit de la información de la marca usando la ecuación 1. El tamaño de la ventana determina la taza de codificación. Dada una escala en niveles de gris de 512x512, el número de bits de la información de la marca de agua que puede ser insertada en la imagen de aproximación es de

$$\frac{512\times512}{2^{2j}}\cdot\frac{1}{w}$$

Los valores típicos de w oscilan desde 2 hasta 8.

Para los bloques de las subbandas de detalle, se debe usar una ventana más grande puesto que la energía es mucho menor. Se necesita cuantificar por lo menos 256 coeficientes para codificar un bit de la marca de agua. Por esto, si el tamaño del bloque lo permite, se divide en subbloques para incrementar la capacidad de inserción. Los coeficientes de mayor magnitud representan información de textura y contornos, a los cuales se sabe que el HVS no es muy sensible, por lo tanto se debe explotar esta característica para incrementar la fuerza de la marca. Para mantener la implementación lo más simple posible, se usa una función de escalamiento no lineal dada por:

$$f(x) = sign(x)\cdot\left|x^{\beta}\right|,\ \beta>1$$

que se aplica a todos los bloques.

El parámetro β se escoge entre 5 y 6.5. Después de la cuantización, se aplica la función inversa, f^-1 para obtener el bloque marcado.

1. c Introducción de la firma o marca en la información a proteger

1.c.1 Inserción de la información

Las formas más antiguas de ocultar la información son consideradas como técnicas muy "crudas" de criptografía privada para las cuales la llave era el conocimiento del método empleado, lo cual se conoce como seguridad a través de la oscuridad. En los libros de esteganografía se describen varios métodos empleados en la antigüedad, por ejemplo, a los mensajeros griegos se les tatuaba el mensaje en sus cabezas rapadas, el cual se ocultaba al crecerles nuevamente el cabello. También se empleaban tablas enceradas, a las cuales se les raspaba la cera y sobre la madera se tallaba el mensaje. Después las tablas se volvían a encerar, ocultando el mensaje de este modo. Con el transcurso del tiempo, estas técnicas criptográficas se fueron mejorando, con lo cual se logró aumentar tanto la velocidad como la seguridad con la que se transmitía el mensaje oculto, hasta llegar a los sistemas criptográficos modernos, en los cuales la comunicación puede asumirse como segura, por lo menos durante el ciclo de vida útil de la misma: se ha proyectado que los algoritmos más poderosos, que usan llaves de varios kilobits no podrían ser quebrantados a través de la fuerza bruta, aún si todo el poder computacional del mundo se usara conjuntamente para el ataque durante los próximos 20 años. Por supuesto, no puede descartarse que se descubran vulnerabilidades o que haya algún avance importante en el poder de procesamiento, pero para la mayor parte de los usuarios, las técnicas criptográficas que se poseen son suficientes para garantizar la seguridad.

Elección de la marca

La primera pregunta a responder para cualquier sistema esteganográfico o de marcas de agua es la forma que debe tomar el mensaje que va a insertarse. La solución más trivial es introducir cadenas de texto, por ejemplo en una imagen, permitiendo que en ésta se almacene información como autor, título, fecha, etc. La desventaja es que el texto en ASCII, en el cual cada letra se representa por un cierto patrón de bits, no es una buena manera de insertar información pues no es robusto contra ataques comunes: dada su naturaleza, un error de un bit debido a un ataque podría alterar la naturaleza de la información. También sería posible para una tarea sencilla, como la compresión JPEG, convertir la información de copyright en una secuencia ininteligible de caracteres aleatorios.

Entonces, por qué no insertar la información de una forma que sea altamente redundante, como una imagen? De esta manera pueden aprovecharse las características del HVS, además de que las imágenes resultan propicias para aplicaciones de marcas de agua. Por ejemplo, considerando la siguiente figura

Fig 1.c.1-1 Marca de agua original vs marca de agua con un 25% de ruido gaussiano agregado

Se puede observar que a pesar de que la marca de agua se ha sometido a un ruido considerable, aún es altamente identificable.

Modificación del bit menos significativo

Una manera muy sencilla de insertar una marca de agua consiste en usar los bits menos significativos de la imagen soporte. Dada la enorme capacidad de canal que implica usar toda la imagen para este método, un objeto pequeño puede insertarse muchas veces, y aún si la mayor parte de éstas se pierden, una sola marca que sobreviva se considera un éxito.

Por supuesto, este esquema presenta numerosas desventajas: a pesar de que es resistente a ataques como el cropping, la marca puede ser eliminada por medio de la adición de ruido o de compresión con pérdidas. Incluso, un ataque efectivo sería llevar todos los LSB de la señal a 1, efectivamente borrando la marca y con un impacto muy pequeño en la imagen. Por lo tanto, una vez que el algoritmo es descubierto, la marca puede ser modificada o borrada por terceras personas.

Una mejora a este esquema es usar un generador de números pseudoaleatorios para escoger los pixeles en los que se va a realizar la sustitución del LSB, basado en una cierta semilla. In embargo, el algoritmo seguiría siendo vulnerable a ataques como reemplazar en LSB con una constante, aún en los pixeles que no contuvieran información de la marca de agua.

La modificación del LSB es una técnica simple y relativamente poderosa para la esteganografía, aunque carece totalmente de la robustez requerida para las marcas de agua.

Técnicas basadas en la correlación

Otra técnica para insertar marcas de agua es explotar las propiedades de correlación de los patrones de ruido gaussiano pseudoaleatorio al ser aplicados a una imagen. Un patrón de ruido pseudoaleatorio, denominado en este caso W(m,n), se agrega la imagen soporte X(m,n), de acuerdo a la ecuación:

$$X_w(m,n) = X(m,n) + k * W(x,y)$$

En donde k representa un factor de ganancia y X_w es la imagen marcada. Al incrementar k se incrementa la robustez de la marca a expensas de la calidad de la imagen marcada.

Para recuperar la marca., el mismo generador de ruido pseudoaleatorio se inicializa con la misma semilla y se computa la correlación entre el patrón de ruido y la imagen posiblemente marcada. Si esta correlación supera un cierto umbral T, se ha detectado una marca. Este método puede usarse para insertar varias marcas, dividendo la imagen en bloques e insertando una marca en cada uno de ellos. Además, existen maneras de mejorar este algoritmo, por ejemplo, la noción de un umbral que da por resultado un "1" o un "0" puede eliminarse usando dos patrones pseudoaleatorios, uno que corresponde al cero lógico, y otro para el uno lógico. Entonces para cada patrón se realiza el procedimiento antes mencionado y se selecciona el patrón con la correlación más grande. Esto incrementa la probabilidad de una detección correcta, aún si la imagen se ha sometido a un ataque.

Se puede aumentar aún más la capacidad de este método si se pre-filtra la señal antes de aplicar la marca. Si podemos reducir la correlación de la imagen soporte y la secuencia de ruido pseudoaleatorio, podremos incrementar la inmunidad de ésta al ruido adicional que pudiera agregarse durante la transmisión. Si se aplica el filtro de realce de contornos que se muestra a continuación, la robustez de la marca aumenta sin pérdida de capacidad y una reducción muy pequeña de la calidad de la imagen.

$$F_{edge} = \frac{1}{2}\begin{vmatrix} -1 & -1 & -1 \\ -1 & 10 & -1 \\ -1 & -1 & -1 \end{vmatrix}$$

En vez de determinar los valores de la marca de agua a partir de bloques en el dominio del espacio, podemos emplear técnicas de espectro ampliado para distribuir los bits de manera aleatoria en toda la imagen, aumentando la capacidad del marcado y su resistencia al cropping. La marca primero se formatea como una cadena en vez de como una imagen 2D. Para cada valor de la marca, una secuencia de pseudo ruido se genera por medio de una semilla distinta. Las semillas pueden almacenarse o incluso, ellas mismas pueden a su vez ser generadas por medio de secuencias pseudoaleatorias. La suma de todas esas secuencias representa la marca de agua, la cual luego se escala y se agrega a la imagen soporte.

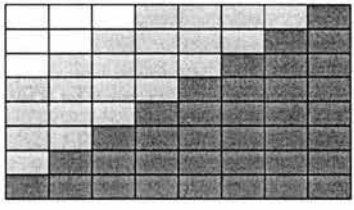
Para detectar la marca, cada semilla se usa para generar su secuencia, la cual es luego correlacionada con toda la imagen. Si la correlación es alta , ese bit en la marca se evalúa a "1", de lo contrario, se evalúa a "0". El proceso se repite para todos los valores de la marca de agua. A pesar de que esto incrementa la robustez, también incrementa el costo computacional.

Técnicas en el dominio de la frecuencia

Una desventaja de las técnicas en el dominio del espacio es que no permiten explotar las características del sistema visual humano, y en consecuencia no se adaptan bien a algunas de las operaciones comunes del procesamiento digital de imágenes que sí lo hacen, como la compresión JPEG, etc., además de que dificultan la implementación de técnicas de marcado adaptativo: es preferible esconder la marca de agua en las regiones ruidosas y en los contornos de una imagen que en regiones con cambios más sutiles.

Tomando en consideración estos aspectos, trabajar en algún tipo de dominio de frecuencia se vuelve muy atractivo. El dominio más popular es el de la DCT, que permite descomponer la imagen en diferentes bandas de frecuencia. Teniendo esta representación en frecuencia, se escogen las bandas altas y medias para insertar la marca, aprovechando las características del HVS.

Una de las técnicas en este dominio usa la comparación de coeficientes DCT de las bandas medias para codificar un bit dentro de un bloque DCT. Para comenzar, se definen las bandas medias de la DCT como se muestra en la tabla:



Componentes DCT de frecuencias bajas

Componentes DCT de frecuencias medias

Componentes DCT de frecuencias altas

Se escoge la banda media de frecuencias para proporcionar resistencia adicional a técnicas de compresión con pérdidas y evitar modificaciones significativas a la imagen soporte.

Entonces, se escogen dos localidades, $X_1(m_1,n_1)$ y $X_2(m_2,n_2)$ dentro de esta región de frecuencias para ser comparadas. Es importante aclarar que estas localidades no se escogen arbitrariamente: para lograr una robustez extra a la compresión, la selección de los coeficientes se basa en la tabla de cuantización recomendada por JPEG, la cual se muestra a continuación:

16	11	10	26	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Si se escogen dos localidades con los mismos valores de cuantización, podemos tener la certeza de que el escalamiento de un coeficiente escalará al otro coeficiente de la misma manera, preservando su tamaño relativo.

Técnicas en el dominio wavelet

Otro de los dominios en los que se pueden insertar marcas de agua es el dominio wavelet. La DWT, que se discutirá con más detalle en las secciones siguientes, separa una imagen en una imagen de aproximación de menor resolución (LL), así como en componentes de detalle horizontales (HL), verticales (LH) y diagonales (HH), por medio de un proceso que puede repetirse para calcular la descomposición wavelet de múltiple escala.

Una de las ventajas de la transformada wavelet es que es capaz de modelar las características del HVS aún mejor que la DCT o la FFT, lo cual nos permite usar marcas de agua de más energía en las regiones en que se sabe que el HVS es menos sensible, como las bandas de detalle de alta resolución. Insertar las marcas de agua en estas regiones nos permite incrementar la robustez de nuestra marca con un impacto muy pequeño en la calidad de la imagen soporte.

Una de las técnicas más inmediatas es similar a la usada para la DCT, de acuerdo a la siguiente ecuación:

$$Xw_{u,v} = \begin{cases} W_{u,v} + \alpha |W_{u,v}| w_{u,v} \text{ ---> } u, v \varepsilon HL, LH \\ W_{u,v} \text{ ---------> } otro\_caso \end{cases}$$

En donde $W_{u,v}$ representa el coeficiente de la imagen transformada, $w_{u,v}$ un bit de la marca de agua a insertar y α un factor de escala. Para detectar la marca de agua se genera la misma secuencia pseudoaleatoria que en la generación (CDMA) y se determina su correlación con las dos bandas de detalle transformadas. Si la correlación excede algún cierto umbral T, la marca es detectada.

1.d Conceptos básicos de la transformada wavelet

1.d.1 Análisis de resolución múltiple y la transformada wavelet continua

A pesar de que el problema de la resolución de tiempo y frecuencia es el resultado de un fenómeno físico (el principio de incertidumbre de Heissenberg) que existe sin importar la transformación usada, es posible analizar cualquier señal usando un enfoque conocido como Análisis de Resolución Múltiple (MRA por sus siglas en inglés). Como su nombre lo indica, el MRA analiza la señal a diferentes frecuencias con diferentes resoluciones. No todas las componentes espectrales se resuelven de la misma manera, como en el caso de transformaciones como la Short Time Fourier Transform.

El MRA está diseñado para dar una buena resolución temporal y una mala resolución de frecuencia para las frecuencias altas, y una mala resolución temporal y buena resolución de frecuencia para las frecuencias bajas. Si la señal analizada posee componentes de alta frecuencia en intervalos temporales cortos y componentes de baja frecuencia en intervalos temporales grandes, este enfoque es adecuado para su análisis. Afortunadamente, la mayoría de las señales encontradas en aplicaciones prácticas cumplen con las anteriores condiciones

La transformada wavelet continua fue desarrollada como una alternativa a la Short Time Fourier Transform para resolver el problema de resolución. El análisis wavelet se realiza de manera similar al análisis STFT en el sentido de que la señal es multiplicada por una función (la *wavelet*), similar a la función de ventana en la STFT, y la transformación es computada separadamente para diferentes segmentos de la señal en el dominio del tiempo. Sin embargo, hay dos diferencias fundamentales entre la CWT y la STFT:

1. La transformada de Fourier de la función de ventana no se calcula, por lo tanto para señales sinusoidales se verán espigas aisladas. No se computan las frecuencias negativas.

2. El ancho de la ventana varía al computarse la transformada para cada componente espectral, lo cual es probablemente la característica más significativa de la transformada wavelet.

La transformada wavelet continua se define como se muestra a continuación:

$$CWT_x^{\Psi} = \Psi_x^{\Psi}(\tau, s) = \frac{1}{\sqrt{|s|}} \int x(t) \psi^* \left(\frac{t-\tau}{s} \right) dt$$

Como puede observarse en la ecuación anterior, la señal transformada es función de dos variables, **tau** y **s**, que se conocen como los parámetros de *translación* y *escala*, respectivamente. Psi(t) es la función de transformación, llamada *wavelet madre*. Se le da el término wavelet madre debido a dos importantes propiedades del análisis wavelet:

El término wavelet significa "onda pequeña". Esta pequeñez se refiere al hecho de que esta función de ventana es de una longitud finita. El término "onda" se refiere a que la función es oscilatoria. Por último, el término "adre" implica que las funciones con diferente región de soporte que se usan para realizar la transformación, son todas derivadas de esta función. La wavelet madre es el prototipo para generar las demás funciones de ventana.

El término "translación" se usa en el mismo sentido que en la STFT: se relaciona con la posición de la ventana al ser esta última desplazada por toda la señal, y se refiere a la información temporal en el dominio de la transformada. Sin embargo, no existe un parámetro de frecuencia; en vez de eso, se tiene un parámetro de escala, definido como 1/f.

La escala

Una manera fácil para comprender el parámetro de escala empleado en el análisis wavelet, es pensar en la escala en los mapas geográficos: una escala grande corresponde a una perspectiva general, con poco detalle, del mapa que se observa. Por el contrario, una escala pequeña se centra en un área más pequeña, la cual es observada a gran detalle. Similarmente, para el análisis wavelet las escalas grandes (frecuencias bajas) corresponden a la información global de la señal, la cual usualmente se encuentra en toda la señal, mientras que las escalas pequeñas (frecuencias altas) corresponden a la información detallada de un patrón escondido en la señal, que generalmente tiene una duración corta.

El escalamiento, visto como una operación matemática, dilata o comprime una señal. Las escalas más grandes corresponden a señales dilatadas o ensanchadas y las escalas más pequeñas corresponden a señales comprimidas. En términos matemáticos, si f(t) es una función, f(st) corresponde a una versión contraída o comprimida de f(t) siempre que s>1 y a una versión expandida o dilatada de f(t) si s<1.

Sin embargo, en la definición de la transformada wavelet se ve que el término de escala se usa en el denominador, por lo cual se verifica lo contrario que en el párrafo anterior: s>1 dilata la señal y s<1 comprime la señal. Esta interpretación de la escala es muy importante para el entendimiento de las consecuencias físicas y matemáticas de la transformada wavelet.

Cálculo de la CWT

Sea x(t) la señal a ser analizada. La wavelet madre se escoge para servir como un prototipo para todas las ventanas en el proceso. Todas las ventanas usadas son versiones dilatadas o comprimidas y desplazadas de la wavelet madre. Hay algunas funciones usadas para este propósito, entre las cuales figuran la wavelet de Morlet y el Sombrero Mexicano.

Una vez que se ha escogido la wavelet madre, el cálculo comienza con s = 1 y se calcula la transformada wavelet continua para todos los valores de s, mayores y menores que 1. Sin embargo, dependiendo de la señal, usualmente no es necesaria una transformación completa. Para propósitos prácticos, las señales están limitadas en banda, por lo cual se calcula la transformación para un intervalo de escalas limitado.

Por conveniencia, el proceso se inicia en s = 1 y continúa para valores más grandes de s, o sea, el análisis comenzará en las frecuencias altas y continuará hacia las frecuencias bajas. El primer valor de s corresponde a la wavelet más comprimida, y al incrementarse, la wavelet se dilatará.

La wavelet se coloca al inicio de la señal, en el punto que corresponde a t=0. la función wavelet en la escala 1 es multiplicada por la señal y luego es integrada para todos los tiempos. El resultado de la integración es entonces multiplicado por la constante 1/sqrt(s). Esta multiplicación es para la normalización de la energía, de modo que la señal transformada tenga la misma energía en todas las escalas. El resultado final es el valor de la transformación: el valor de la transformada wavelet continua en el tiempo 0 y escala 1. en otras palabras, es el valor que corresponde al punto con tau=0 y s=1 en el plano tiempo-escala.

La wavelet en la escala 1 es entonces recorrida hacia la derecha **tau** unidades, hasta llegar a la posición t=tau y se repite el proceso para obtener el valor de la transformada en el punto tau=1, s=1 en el plano tiempo-escala. El proceso se repite hasta que la wavelet llega al final de la señal. En este momento se ha completado una línea de puntos en el plano tiempo-escala con s=1.

Entonces s es incrementada por un valor pequeño. Esta es una transformación continua, lo que significa que tanto tau como s se deben incrementar de manera continua. Sin embargo, si se ha de calcular por medio de una computadora o algún otro dispositivo digital, ambos parámetros deben incrementarse por

un tamaño de escalón *suficientemente pequeño*. Esto corresponde a muestrear el plano tiempo-escala. Este procedimiento se repite para todos los valores de s. Cada cálculo para un valor dado de s llena un renglón en el plano tiempo-escala. Cuando el cálculo se ha realizado para todos los valores deseados de s, se ha obtenido la DWT para la señal.

Resolución de tiempo y de frecuencia

La ilustración en la siguiente figura es comúnmente usada para explicar cómo deben interpretarse las resoluciones espacial y de frecuencia. Cada caja en la figura corresponde a un valor de la transformada wavelet en el plano tiempo-frecuencia. Las cajas tienen una cierta área, que es diferente de cero, lo que implica que el valor de un punto en particular del plano no puede ser conocido. Todos los puntos del plano que caen en una caja son representados por un valor de la WT.

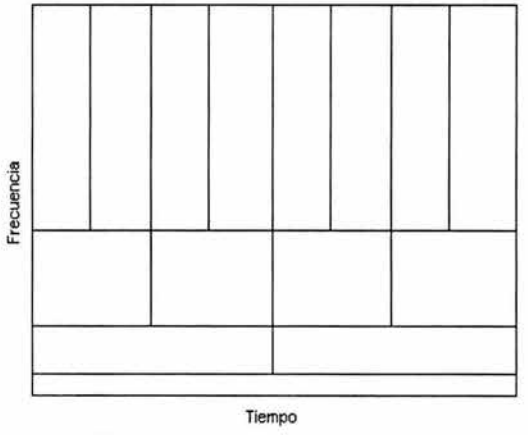


Fig 1.d.1-1 Multiresolución de la transformada wavelet

Analizando la figura con más detenimiento, lo primero que observamos es que a pesar de que la altura y el ancho de las cajas cambian, el área es constante. Cada caja representa una porción del mismo tamaño del plano tiempo-frecuencia, pero dando diferentes proporciones al tiempo y la frecuencia. A bajas frecuencias la altura de las cajas es menor, lo que significa que se tiene una mejor resolución en la frecuencia al existir menos ambigüedad en el valor de la frecuencia real. Sin embargo, la anchura es mayor, lo que significa que la ambigüedad en el tiempo es grande: no es posible conocer el valor exacto del tiempo. Para las frecuencias altas, se observa lo contrario.

Sin importar las dimensiones de las cajas, el área está determinada por la desigualdad de Heissenberg y es la misma para cada función de ventana dada, ya sea STFT o wavelet madre. A pesar de que diferentes ventanas o wavelets madre resultan en diferentes áreas, el área menor que puede alcanzarse está determinada por ¼*π. Esto es, no podemos reducir las áreas de las cajas arbitrariamente buscando funciones de ventana adecuadas debido al principio de incertidumbre de Heissenberg. Por otra parte, una wavelet madre dada, las dimensiones de las cajas pueden cambiarse manteniendo el área igual. Esto es exactamente lo que hace la transformada wavelet.

1.d.2 La transformada wavelet discreta

A pesar de que la discretización de la transformada wavelet continua permite que ésta sea calculada por las computadoras, no es en realidad una transformada discreta: de hecho, representa simplemente un muestreo de la CWT y la información que provee es altamente redundante en cuanto a la reconstrucción de la señal se refiere. Esta redundancia implica un consumo bastante significativo de recursos computacionales. Es por esto que se desarrolló la transformada wavelet discreta, o DWT, la cual provee suficiente información tanto para el análisis como para la síntesis de la señal original con una disminución significativa su tiempo de computación. Al mismo tiempo, comparada con la CWT, la DWT es considerablemente más fácil de implementar.

Historia

La DWT tiene sus orígenes en 1976, cuando Croiser, Esteban y Galand descubrieron una técnica para descomponer señales discretas. En ese mismo año, Crochiere, Weber y Flanagan realizaron un trabajo similar codificando señales de voz. Nombraron su trabajo "Codificación de subbanda". En 1983 Burt definió una técnica muy similar a la codificación de subbanda, llamada "Codificación piramidal", conocida también como Análisis de resolución múltiple. Más tarde, en 1989, Vetterli y Le Gall hicieron algunas contribuciones al esquema de codificación de subbanda, removiendo la redundancia existente en el esquema piramidal. A continuación se explica la codificación de subbanda.

La codificación de subbanda y el análisis de resolución múltiple

La idea principal a tener en mente es la misma que en el caso de la CWT: una representación en tiempo-escala de una señal digital. Esta escala es obtenida usando técnicas de filtrado digital. Recordemos que la CWT es la correlación entre una wavelet a diferentes escalas y la señal, en la que se usa la escala como una medida de similitud. La transformada continua se calculaba cambiando la escala de la ventana de análisis, desplazándola en el tiempo, multiplicándola por la señal y finalmente, integrándola para todos los tiempos. En el caso discreto se usan filtros con diferentes frecuencias de corte para analizar la señal en diferentes escalas. La señal se pasa por una serie de filtros pasa altas para analizar las frecuencias altas y luego por una serie de filtros pasa bajas, para analizar las frecuencias bajas.

La resolución de la señal, que es una medida de la cantidad de información detallada contenida en la señal, se modifica mediante el filtrado. La escala se modifica mediante el submuestreo o el sobremuestreo.

El procedimiento inicia al pasar la señal discreta por un filtro pasa bajas de media banda, con respuesta a impulso h(n). Filtrar una señal corresponde a la operación matemática de convolucionar su espectro de frecuencia con la respuesta al impulso del filtro. La convolución en el tiempo discreto se define como sigue:

$$x(n) * h(n) = \sum x(k)h(n-k)$$

Con k variando desde infinito negativo hasta infinito.

Un filtro pasa bajas de media banda elimina todas las frecuencias que están por encima de la mitad de la más alta frecuencia presente en la señal. Por ejemplo, si la máxima frecuencia presente en la señal fuera de 1000 Hz, entonces este tipo de filtro eliminaría todas aquellas componentes de frecuencia superiores a los 500 Hz.

La unidad con la que se mide la frecuencia es de particular importancia en este punto del análisis: para las señales discretas, la frecuencia se expresa en rad/s. Por convención, la frecuencia de muestreo de la señal es de 2π rad/s. Entonces, si se considera que se ha cumplido con el teorema del muestreo de Nyquist, la máxima componente de frecuencia que existe en la señal es de π rad/s . Entonces, el Hz no es la medida más apropiada para las señales discretas. Sin embargo, al ser el Hz una unidad muy común se empleará para clarificar algunos conceptos, aunque siempre hay que tener en mente que la unidad de medición de frecuencias para las señales discretas es el rad/s.

Después de haber pasado la señal por el filtro pasa bajas de media banda, la mitad de las muestras pueden ser eliminadas, de acuerdo con la regla de Nyquist, dado que la máxima frecuencia de la señal es ahora $\pi/2$ rad/s, en lugar de π rad/s. Con simplemente desechar cada tercera muestra de la señal se efectuará un submuestreo de soporte 2 y la señal tendrá sólo la mitad de los puntos. La escala de la señal se ha duplicado. Nótese que el filtrado pasa bajas remueve la información de alta frecuencia pero deja la escala sin cambios, sólo el proceso de submuestreo cambia la escala. La resolución en cambio, se relaciona con la cantidad de información contenida en la señal y se ve afectada por el filtrado. El filtrado pasa bajas de media banda remueve la mitad de las frecuencias, lo cual puede interpretarse como la pérdida de la mitad de la información. Por lo tanto, la resolución se divide a la mitad después de cada filtrado. Sin embargo, hay que hacer notar que el submuestreo no afecta la resolución pues después de aplicar el filtrado, la mitad de las muestras de la señal son redundantes de cualquier manera. Estas muestras pueden descartarse sin ninguna pérdida de información. En resumen, el filtrado pasa bajas disminuye la resolución dejando la escala intacta, mientras que el submustreo de soporte 2 duplica la escala al eliminar la mitad de las muestras, las cuales son redundantes. Esto puede ser expresado matemáticamente por medio de la fórmula:

$$y(n) = \sum h(k)x(2n-k)$$

nuevamente, k varía discretamente desde menos infinito hasta infinito.

Ahora veremos como es que se calcula en la práctica la DWT: la DWT analiza la señal en diferentes bandas de frecuencia con diferentes resoluciones al descomponer la señal en una aproximación amplia y una aproximación detallada (información detallada). Se emplean dos conjuntos de funciones, llamadas funciones wavelet y funciones de escala, que se asocian con las frecuencias bajas y altas respectivamente. La descomposición de la señal en diferentes bandas de frecuencia se logra simplemente por diversas operaciones sucesivas de filtrado pasa bajas y pasa altas. La señal original, x(n) se pasa inicialmente por un filtro pasa altas de media banda g(n) y un filtro pasa bajas h(n). Después del filtrado, la mitad de las muestras puede eliminarse de acuerdo con la regla de Nyquist, pues la máxima frecuencia presente en la señal se ha reducido a la mitad. Esto constituye un nivel de descomposición que matemáticamente puede expresarse de la siguiente forma:

$$y_{high}(k) = \sum x(n)g(2k-n)$$

$$y_{low}(k) = \sum x(n)h(2k-n)$$

En donde $y_{high}(k)$ y $y_{low}(k)$ son las salidas de los filtros pasa altas y pasa bajas, respectivamente, tras submuestrear por 2.

La descomposición disminuye a la mitad la resolución temporal de la señal, puesto que ahora la mitad de las muestras caracterizan a la señal completa. Sin embargo, la resolución en frecuencia se eleva al doble puesto que la banda de la señal ahora ocupa sólo la mitad de la banda que ocupaba anteriormente, reduciendo con esto la incertidumbre también a la mitad. El procedimiento anterior, conocido como codificación de subbanda, puede repetirse para alcanzar una mayor descomposición. En cada nivel, el filtrado y el submuestreo servirán para eliminar la mitad de las muestras (disminuyendo la resolución temporal) y la mitad de las frecuencias (aumentando al doble la resolución de frecuencia). A

continuación se ilustra este procedimiento, en donde x(n) es la señal original a ser descompuesta y h(n) y g(n) son filtros pasa bajas y pasa altas respectivamente. El ancho de banda de la señal en cada nivel se denota como "f"

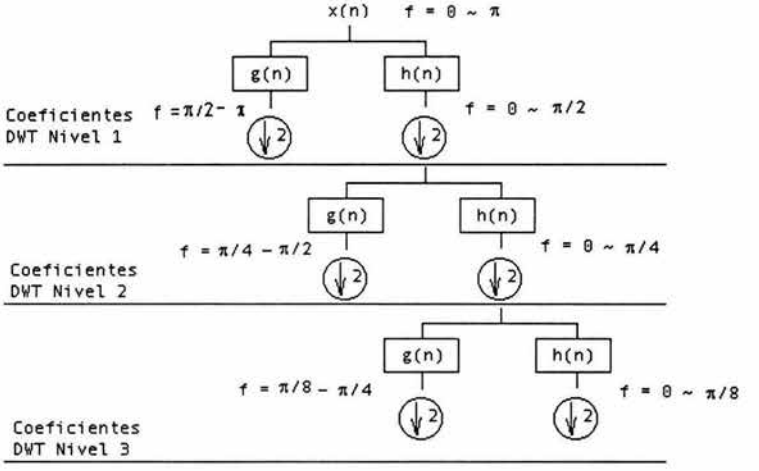


Fig 1.d.2-1 Algoritmo de codificación de subbandad

La figura 1.d.2-1 muestra un ejemplo del algoritmo de codificación de subbanda. Supongamos que la señal original x(n) tiene 512 puntos, con una banda de frecuencia de 0 a π rad/s. Tras el primer nivel de descomposición, la señal se pasa por los filtros pasa altas y pasabajas, tras lo cual se efectúa un submuestreo con soporte 2. A la salida del filtro pasa altas hay 256 puntos (la mitad de la resolución espacial) con componentes de frecuencia que van desde $\pi/2$ hasta π rad/s (el doble de la resolución de frecuencia). Estas 256 muestras corresponden al primer nivel de coeficientes DWT. La salida del filtro pasa bajas también tiene 256 muestras, con frecuencias que van de 0 a $\pi/2$ rad/s. Esta señal se pasa por los mismos filtros pasa bajas y pasa altas para lograr una mayor descomposición. La salida del segundo filtro pasa bajas, seguida del submuestreo, tiene 128 muestras con componentes de frecuencia que van desde $\pi/4$ hasta $\pi/2$ rad/s. La salida del segundo filtro pasa altas constituye el segundo nivel de coeficientes DWT. Esta señal tiene la mitad de la resolución temporal pero el doble de la resolución de frecuencia de la señal obtenida a la salida del primer set de filtros. En otras palabras, la resolución temporal se ha reducida 4 veces respecto a la señal original mientras que la resolución en frecuencia ha aumentado 4 veces. De nuevo se filtra la salida del filtro pasa bajas para lograr una descomposición aún mayor, proceso que continúa hasta que sólo quedan dos muestras. La DWT de la señal original se obtiene concatenando todos los coeficientes, comenzando desde el último nivel de descomposición. Entonces, la señal transformada tendrá el mismo número de muestras que la señal original.

Nótese como debido al submuestreo con soporte 2, la longitud de la señal debe ser una potencia de 2, o por lo menos, múltiplo de una potencia de 2 para que este esquema pueda funcionar. La longitud de la señal determina el número de niveles de descomposición que es posible lograr. Por ejemplo, para una señal de 1024 muestras, es posible lograr 10 niveles de descomposición wavelet.

Las frecuencias predominantes en la señal original aparecen como amplitudes grandes en la región de la señal DWT que incluye esas frecuencias particulares. La diferencia respecto a la transformada de

Fourier es que la localización temporal de esas frecuencias no se perderá; aunque depende del nivel en el que aparece: si la información principal de la señal aparece en las frecuencias altas, como ocurre comúnmente, su localización temporal será más precisa, al ser caracterizada por un mayor número de muestras. En caso contrario, su localización temporal será imprecisa. En general, y al igual que en el caso continuo, este procedimiento ofrece una buena resolución temporal a altas frecuencias y una resolución temporal pobre en bajas frecuencias. El análisis de la mayoría de las señales que se observan en el mundo real resulta beneficiado por estas características de la DWT.

Las bandas de frecuencia que no son predominantes en la señal original tendrán amplitudes DWT pequeñas y por lo tanto esa parte de la señal DWT puede ser desechada sin que esto suponga una gran pérdida de información, permitiendo así la compresión de la señal.

Una propiedad importante de la DWT es la relación entre la respuesta a impulso de los filtros pasa bajas y pasa altas empleados, los cuales no son independientes entre sí, dada por:

$$g(L-1-n) = h(n)(-1)^n$$

donde como ya se ha visto, g(n) es el filtro pasa altas, h(n) es el filtro pasa bajas y L es la longitud del filtro (dada en número de puntos). Ambos filtros son versiones impares alternadas uno del otro. La conversión de pasa altas a pasa bajas se da por el término (-1)^n. Estos filtros se conocen como filtros QMF (Quadrature Mirror Filters). Las dos operaciones de filtrado y submuestreo pueden expresarse como:

$$y_{high}(k) = \sum_n x(n)g(-n+2k)$$

$$y_{low}(k) = \sum_n x(n)h(-n+2k)$$

La reconstrucción en este caso se simplifica puesto que los filtros de media banda forman bases ortonormales. El procedimiento descrito anteriormente se sigue en orden inverso para realizar la reconstrucción: las señales obtenidas en cada nivel de descomposición son sobremuestreadas por 2, pasadas por los filtros de síntesis g'(n) y h'(n) y después sumadas. El punto interesante es que los filtros de análisis (los filtros empleados para la descomposición) y síntesis son idénticos. Entonces, la fórmula de reconstrucción es, para cada nivel:

$$x(n) = \sum_k (y_{high}(k)g(-n+2k)) + (y_{low}(k)h(-n+2k))$$

Sin embargo, si los filtros no son filtros de media banda ideales, la reconstrucción no será perfecta. Aunque no es posible implementar filtros ideales, bajo ciertas condiciones es posible hallar filtros que provean una reconstrucción perfecta. Los más famosos son los desarrollados por Ingrid Daubechies, conocidos como *wavelets Daubechies*.

Referencias

[1] I. J. Cox, J. Kilian, T. Leighton, T. Shamoon, "Secure spread spectrum watermarking for multimedia" IEEE Trans. Image Proc., vol. 6, no. 12, pp. 1673{1687, Dic. 1997.

[2] P. G. Flikkema, "Spread-spectrum techniques for wireless communications" IEEE Signal Proc. Magazine, vol. 14, no. 3, pp. 26{36, May 1997.

[3] A H Paquet , R K Ward , "Wavelet Based Digital Watermarking".

[4] I J Cox , M L Miller and J A Bloom, "Watermarking Applications and their Properties".

[5] H Y Lo , S Topiwala and J Wang, "Wavelet Based Steganography and Watermarking"

[6] J.M. Shapiro, "Embedded image coding using zerotrees of Wavelet coefficients". IEEE Transactions on signal processing 41, pp. 3445-3462, Dic.1993.

[7] A. Said, W.A. Pearlman, "A new, fast and efficient image Codec based on set partitioning in hierarchical trees". IEEE transactions on circuits and systems for video technology, vol 6, pp. 243-250, Jun 1996

[8] M. D. Swanson, M. Kobayashi, A. H. Tew_k, "Multi media data-embedding and watermarking techniques". Proc. IEEE, vol. 86, no. 6, pp. 1064{1087, Jun. 1998.

[9] J. K. Su, F. Hartung, B. Girod, "Digital watermarking of text, image, and video documents". Computers & Graphics, vol. 22, no. 6, pp. 687{695, Dec. 1998.

[10] F. Hartung, J. K. Su, B. Girod, "Spread spectrum watermarking: Malicious attacks and counterattacks". Proc. SPIE Security and Watermarking of Multimedia Contents, San Jose, CA, USA, Jan. 1999, vol. 3657, pp. 147{158.

2. VISIÓN HUMANA

2.a Anatomía básica

[1] A pesar de su pequeño tamaño (aproximadamente de 2 cm y medio de ancho y profundidad y 2.3 cm de alto), el ojo es un órgano muy complejo.

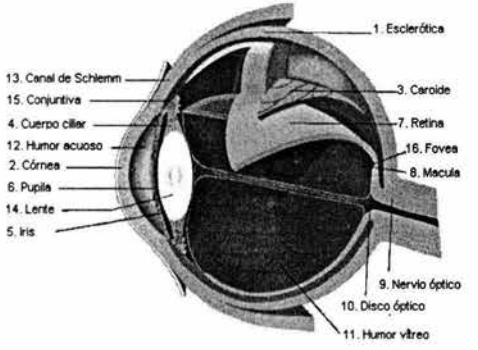


Fig 2.a-1 Anatomía del ojo humano.

La parte externa del ojo, llamada esclerótica, mantiene la forma del ojo. La parte frontal de ésta es la córnea. La luz debe atravesar la córnea antes de entrar en el ojo. Unidos a la esclerótica se encuentran los músculos extraoculares, cuya función es dar movilidad al ojo.

La caroide es la segunda capa del ojo y contiene los vasos sanguíneos que alimentan a las estructuras de éste. Su parte frontal se compone de dos estructuras:

- El cuerpo ciliar. Área muscular unida al cristalino cuya función es contraerlo o relajarlo a modo de controlar su tamaño, esto para enfocar.
- El iris. Parte coloreada del ojo cuya función es actuar como un diafragma controlando la cantidad de luz que entra a la pupila.

El iris tiene dos músculos, uno encargado de hacer a la pupila más grande, y uno de hacerla más pequeña. La pupila puede cambiar su tamaño de 2 a 8 mm, lo cual significa un cambio de hasta 30 veces en la intensidad de la luz que por ella entra.

La parte más interna es la retina, la parte del ojo sensible a la luz, la cual contiene células llamadas conos y bastones. En la parte trasera del ojo, al centro de la retina, se encuentra la mácula, en cuyo centro se localiza la fovea, región que contiene sólo conos y es responsable de la visión de los detalles finos.

Las fibras del nervio retinal se unen en la parte trasera del ojo, formando el nervio óptico, el cual conduce los impulsos eléctricos al cerebro. El punto en donde el nervio óptico y los vasos sanguíneos salen de la retina se conoce como disco óptico y constituye un punto ciego pues en él no existen conos ni bastones. Esto no es fácilmente detectable puesto que cada ojo cubre el punto ciego del otro.

El cristalino (lente) es una estructura transparente biconvexa de unos 10 mm de diámetro, formado por capas concéntricas de células fibrosas. Debido al hecho de que está unido al cuerpo ciliar, es capaz de cambiar de forma. Se usa para enfocar. Contiene alrededor de un 60 ó 70% de agua, 6% de grasa y más proteínas que cualquier otro tejido del ojo. Está también coloreado por un pigmento ligeramente amarillo que aumenta con la edad.

El cristalino absorbe aproximadamente el 8% de del espectro de la luz visible, con una absorción mayor a anchos de banda menores, aunque hay proteínas encargadas de absorber también las luces infrarroja y ultravioleta, las cuales en cantidades excesivas pueden dañar al ojo.

2.b Percepción de la luz

Cuando la luz entra en el ojo, lo hace atravesando la córnea, el humor acuoso, el cristalino y el humor vítreo hasta finalmente llegar a la retina. La retina contiene dos tipos de células: bastones (encargados de la visión a intensidades de luz bajas) y conos (que se encargan de la percepción de los detalles y la visión del color). Cuando la luz entra en contacto con estas células, ocurre una serie de reacciones químicas de modo que el químico que se forma debido a éstas (Rodopsina activada), crea impulsos eléctricos en el nervio óptico. A continuación se muestra un ejemplo de un bastón y un cono.

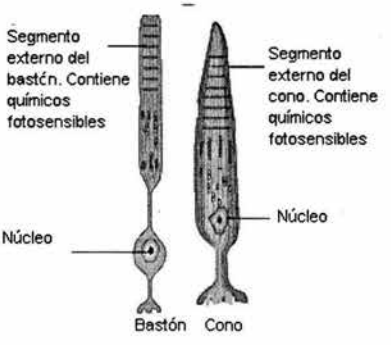


Fig 2.b-1 Diagrama de un bastón y un cono, las células fotosensibles de la retina.

El segmento externo de un cono o un bastón contiene los químicos fotosensibles. Para los bastones, éste químico se llama Rodopsina mientras que para los conos son los pigmentos de color. La retina contiene aproximadamente 100 millones de bastones y 7 millones de conos y su interior es negro, como el de una cámara, a manera de disminuir reflexiones internas. En el centro de la retina, la fovea es el área responsable de la visión detallada.

Los seres humanos podemos percibir los detalles finos por medio de los conos principalmente porque cada uno de ellos está conectado a su propia terminal nerviosa. Los músculos que controlan el movimiento del ojo lo hacen rotar hasta que la región de interés queda sobre la fovea. La visión de los conos se denomina visión *fotópica* o "de luz brillante". Por otro lado, los bastones en general no se encuentran limitados a la fovea, sino que se distribuyen sobre un área mayor dentro de la retina. Esta mayor área de distribución y el hecho de que varios conos se conectan a una misma terminal nerviosa, reducen la cantidad de detalle discernible por estos receptores, los cuales sirven para dar una perspectiva general del campo de visión. No se relacionan con la visión a color pero en cambio, son

sensibles a niveles bajos de iluminación. Por ejemplo, objetos que aparecen brillantemente coloreados a la luz del sol, al ser vistos a la luz de la luna aparecen como formas casi sin color, esto debido a que a estas intensidades de luz tan bajas, sólo los bastones son estimulados. Este fenómeno se conoce como visión *escotópica* o de "luz débil".

A continuación se presenta una gráfica que muestra la densidad de conos y bastones en una sección del ojo derecho, pasando por la región en que emerge el nervio óptico. La ausencia de receptores en esta área resulta en el llamado punto ciego. Excepto por esta región, la distribución de los receptores es radialmente simétrica alrededor de la fovea.

Nótese que la concentración de conos es mayor en el centro de la retina (o sea, en el área central de la mácula, la fovea), y que los bastones por el contrario, aumentan su densidad al alejarse de la fovea hasta un máximo de aproximadamente 20°, tras lo cual vuelven a decrementarse en la extrema periferia de la retina.

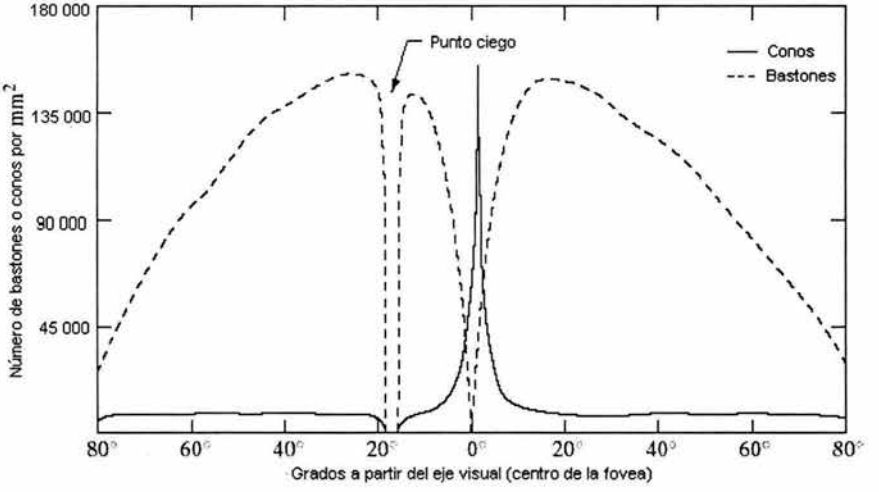


Fig 2.b-2 Distribución angular de conos y bastones dentro del ojo humano

Volviendo a las reacciones químicas que tienen lugar en el interior del ojo, cuando la luz entra en él, ésta entra en contacto con la Rodopsina (la cual es una mezcla de una proteína llamada escotopsina y un derivado de la vitamina A llamado 11-cis-retinal). La Rodopsina se descompone cuando entra en contacto con la luz, esto porque la luz causa un cambio físico en el 11-cis-retinal que lo transforma en *all-trans retinal.* Esta primera reacción tarda unas pocas trillonésimas de segundo. El 11-cis-retinal es una molécula angulada mientras que el all-trans-retinal no lo es. Esto hace al químico inestable. Entonces la Rodopsina se rompe en varios compuestos intermedios pero eventualmente se forma la Metarodopsina II (Rodopsina activada). Este químico causa impulsos eléctricos que se transmiten al cerebro, donde se interpretan como luz. A continuación, se muestra un diagrama de la reacción química descrita.

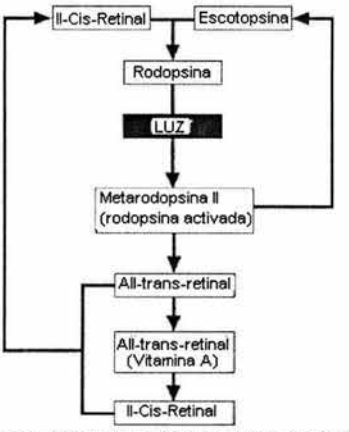


Fig 2.b-3 Diagrama del proceso por el cual la luz interactúa como los fotorreceptores de las células sensibles a la luz

La Rodopsina activada produce impulsos eléctricos de la siguiente manera:

1. La membrana celular de los bastones tiene una carga eléctrica. Cuando la luz activa la Rodopsina, esta carga se incrementa debido a un proceso de reducción química. Lo anterior produce una corriente eléctrica a lo largo de la célula. Mientras más luz se detecta, más Rodopsina es activada y la corriente producida es mayor.
2. Este impulso eléctrico eventualmente alcanza un ganglio y después el nervio óptico.
3. Los nervios alcanzan el punto en que las fibras nerviosas de la mitad interna de cada retina cruzan al lado opuesto del cerebro, mientras que los de la mitad externa permanecen en el mismo lado.
4. Estas fibras eventualmente alcanzan el lóbulo occipital del cerebro. Aquí es donde la visión es interpretada y también se llama "Corteza visual primaria". Algunas de las fibras visuales van a otras regiones del cerebro, en donde se controlan los movimientos de los ojos, la respuesta de las pupilas y el iris, etc.

Eventualmente, la Rodopsina necesita reformarse para que el proceso pueda volver a ocurrir. El All-trans-retinal es convertido en 11-cis-retinal, el cual vuelve a recombinarse con la Escotopsina para formar la Rodopsina. Entonces el proceso vuelve a empezar al ser expuestos nuevamente a la luz.

2.c Visión a color

[1] Los químicos responsables de la visión a color son llamados pigmentos de color y son muy similares a los químicos en los bastones. La porción de retinal en el químico es la misma pero la escotopsina es reemplazada por la Fotopsina. Existen 3 clases de pigmentos de color:

- Pigmento sensible al rojo.
- Pigmento sensible al verde.
- Pigmento sensible al azul.

Cada cono tiene uno de estos pigmentos, de modo que es sensible a ese color. El ojo humano puede percibir casi cualquier graduación de color cuando se mezclan el rojo, verde y azul.

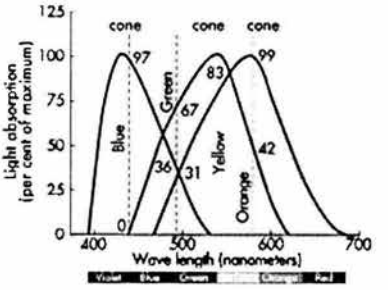


Fig 2.c-1 Absorción de luz para los tres tipos de cono.

En la figura 2.c.1 se muestran las longitudes de onda de los tres tipos de conos. La absorción pico del pigmento sensible al azul es de 445 nanómetros, la del pigmento sensible al verde es de 535 nanómetros y la del pigmento sensible al rojo es de 570 nanómetros.

2.d Características de la visión humana

[2][3][4] El sistema visual humano percibe los colores como una combinación lineal de intensidades de tres colores básicos. Con base en esto, las TV's a color, monitores de PC y otros dispositivos son capaces de formar casi cualquier color controlando las intensidades relativas de fuentes de luz rojas, verdes y azules, de lo cual se deduce que las imágenes a color que existen en forma electrónica se representan por 3 intensidades (R,G,B) en la posición de cada píxel (picture element).

Los valores numéricos de estas intensidades usualmente se eligen de manera tal que incrementos iguales en sus valores resulten aproximadamente en incrementos iguales en brillo. En la práctica esto significa que dichos valores numéricos son aproximadamente proporcionales al logaritmo de la verdadera intensidad de la luz (la energía de la onda), esta es la ley de Webber.

El ojo es mucho más sensible a los cambios en la intensidad "general" (luminancia), que a los cambios en el color. Usualmente, la mayor parte de la información en una escena o imagen está contenida en su luminancia y no en el color (crominancia); es por eso que las imágenes monocromas o en blanco y negro fueron aceptables para la fotografía y la televisión durante muchos años, antes de que apareciera una tecnología para la representación a color que además fuera lo suficientemente barata como para las modestas ventajas que la misma implica. La luminancia (L) de in pixel se puede obtener de sus componentes RGB mediante la siguiente fórmula:

$$L = 0.3R + 0.6G + 0.1B$$

Estos coeficientes son sólo aproximaciones definidas por JPEG. Los valores 0.3, 0.59 y 0.11 son también usados.

Las representaciones en RGB de la imágenes se definen normalmente de modo que si R=G=B, el pixel será siempre de algún tono de gris, y además L = R = G = B. Por esto, los tres coeficientes de la anterior ecuación deben sumar la unidad.

Cuando L define la luminancia del pixel, la crominancia se define por U y V de modo que:

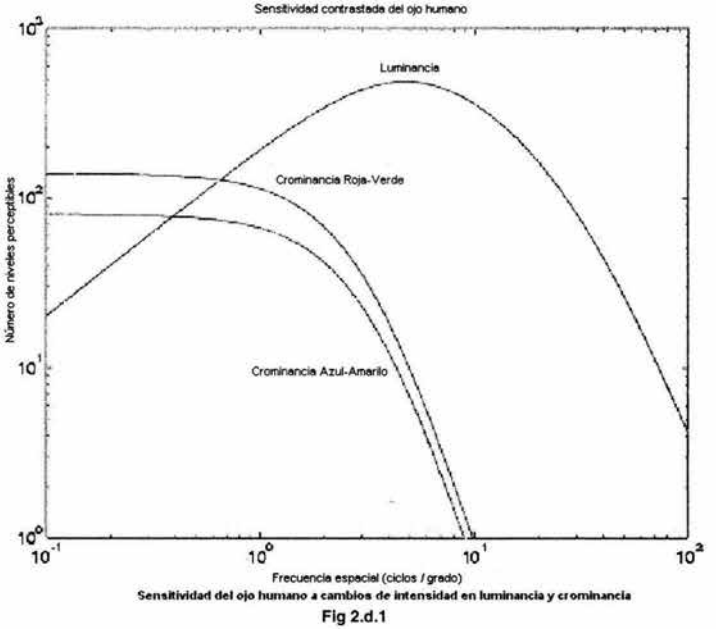
$$U = 0.5 (B - L)$$

$$V = 0.625 (R - L)$$

Nótese como para los pixeles grises U = V = 0

2.d.1 Sensitividad visual

En la siguiente figura se muestra la sensitividad del ojo humano a las componentes de luminancia (L) y crominancia (U,V) de las imágenes. La escala horizontal es la frecuencia espacial y representa la frecuencia de un patrón alternado de líneas paralelas con una intensidad variando sinusoidalmente. La escala vertical es la sensitividad al contraste para la visión humana, que es la razón entre el rango máximo de intensidades visibles y la mínima variación en la intensidad pico a pico a la frecuencia especificada.



Sensitividad del ojo humano a cambios de intensidad en luminancia y crominancia

Fig 2.d.1

[5] Se observa en la figura que:

- La máxima sensitividad a L ocurre para frecuencias espaciales alrededor de 5 ciclos / grado, lo que corresponde a patrones con un semiperíodo de 1.8 mm a una distancia de 1 metro.
- El ojo tiene una respuesta muy pequeña arriba de 100 ciclos / grado, lo que corresponde a patrones con un semiperíodo de 0.1 mm a 1 m. En un display de PC estándar, de 250 mm de ancho, esto requeriría 2500 pixels por línea. Debido a esto, el estándar actual de SVGA, que es de 1024X768 pixeles está aún lejos del ideal.
- La sensitividad a la luminancia cae a bajas frecuencias espaciales, mostrando que no somos buenos para estimar valores absolutos de luminancia si éstos no cambian con el tiempo. La sensitividad a la luminancia que fluctúa con el tiempo no cae a bajas frecuencias temporales.

- La máxima sensitividad a la crominancia es mucho más baja que la máxima sensitividad la luminancia, siendo la sensitividad azul-amarillo (U) la mitad de la sensitividad verde-rojo (V) y alrededor de la sexta parte de la máxima sensitividad a la luminancia.
- Las sensitividades a la crominancia caen para frecuencias espaciales por encima de 1 ciclo / grado, requiriendo un ancho de banda espacial mucho menor que la luminancia.

Esta teoría, llamada Tricromática, también dice que la luz estimula los tres tipos de conos de manera distinta y que es la razón de estos estímulos y no sus valores absolutos, lo que crea la sensación de color en el cerebro. Cada sensor del ojo, bastones y conos, manda un mensaje al cerebro, mensaje que puede ser interpretado como un pixel, y el cerebro combina estos mensajes en una imagen continua. De este modo los ojos son similares a cámaras digitales.

2.d.2) Discriminación y adaptación al brillo

Debido a que las imágenes digitales se despliegan como un conjunto discreto de intensidades, la habilidad del ojo para discriminar entre diferentes niveles de intensidad es una consideración importante al presentar resultados de procesamiento de imágenes. El rango de niveles de intensidad de la luz al que puede adaptarse el sistema visual humano es enorme, del orden de 10E10 y la evidencia experimental indica que, como ya se ha mencionado, el brillo subjetivo (la intensidad que percibe el sistema visual humano) es una función logarítmica de la intensidad luminosa que incide en el ojo.

En la figura se muestra una gráfica del brillo subjetivo contra la intensidad luminosa. La línea sólida representa el rango de intensidades al que se adapta el sistema visual [6].

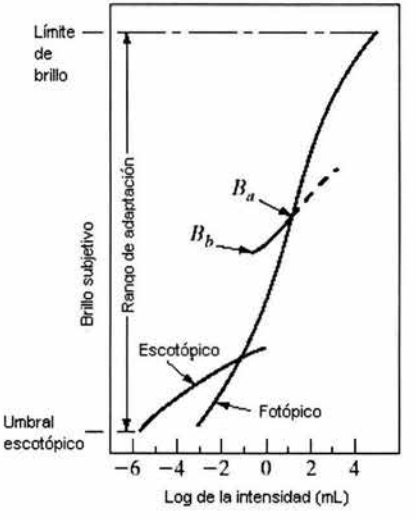


Fig 2.d-2 Gráfica del brillo subjetivo contra el logaritmo de la intensidad luminosa.

Para la sola visión fotópica, el rango es de aproximadamente 10E6. la transición de la visión escotópica a fotópica es gradual sobre el rango aproximado de 0.001 a 0.1 mili Lambert (de -3 a -1 mL en la escala logarítmica), como lo muestran las dobles ramas en la curva de adaptación.

El punto esencial al interpretar el rango dinámico mostrado es que el sistema visual no puede operar sobre este rango simultáneamente, en lugar de eso, logra esta gran variación por cambios en su sensitividad general, un fenómeno conocido como adaptación al brillo. El rango total de los distintos niveles de intensidad que puede discriminar simultáneamente es pequeño comparado con el rango total de adaptación. Para un conjunto de condiciones dado, la sensitividad actual del sistema visual se llama *nivel de adaptación al brillo*, lo que correspondería por ejemplo, al brillo Ba en la figura. La pequeña curva de intersección representa el rango de brillo subjetivo que el ojo puede percibir al adaptarse a este nivel. Este rango es muy restringido, teniendo un nivel Bb, en el cual, y debajo del cual, todos los estímulos son percibidos como "negros indistinguibles".

La porción punteada superior de la curva no está de hecho restringida, pero si se extiende a la distancia pierde sus significados pues intensidades mucho más altas sencillamente aumentarían el nivel de adaptación más allá de Ba. La habilidad del ojo para discriminar entre cambios en la intensidad de la luz en cualquier nivel de adaptación es también de gran interés. Un experimento clásico usado con el fin de determinar la capacidad del sistema visual humano para la discriminación del brillo consiste en hacer que un sujeto mire hacia un área plana iluminada uniformemente y suficientemente grande como para ocupar todo el campo de visión. Esta área es típicamente un difusor, como un vidrio opaco, que se ilumina por detrás por una luz cuya intensidad luminosa I puede variarse. A este campo se agrega un incremento en la iluminación, ΔI, en forma de un "flash" de corta duración que aparece como un círculo en el centro del campo uniformemente iluminado. Si ΔI no es suficientemente brillante, el sujeto dice "no", indicando que no hubo un cambio perceptible. Al hacerse ΔL más intenso, el sujeto podría dar un "sí", indicando un cambio percibido. Finalmente, cuando ΔI es suficientemente intenso, el sujeto dará un "sí" todo el tiempo. La cantidad (ΔIc / I), donde ΔIc es el incremento de iluminación discriminable el 50% de las veces con la iluminación de fondo I, se denomina **razón de Weber**. Un valor pequeño de (ΔIc / I) significa que un pequeño porcentaje de cambio en la intensidad es discriminable. Esto representa una "buena discriminación al brillo". Inversamente, un valor grande significa que se necesita un gran porcentaje de cambio en la intensidad, lo cual representa una "discriminación al brillo pobre".

Una gráfica de (ΔIc / I), como una función del log de I tiene la forma que se muestra en la figura 2.d-3. Esta curva muestra que la discriminación de color es pobre (radio de Weber grande) a bajos niveles de iluminación y se incrementa significativamente cuando la iluminación de fondo aumenta. Las dos ramas en la curva reflejan el hecho de que a bajos niveles de iluminación, la visión se lleva a cabo principalmente por los bastones, mientras que a altos niveles de iluminación, se lleva a cabo por los conos (y se muestra mejor discriminación).

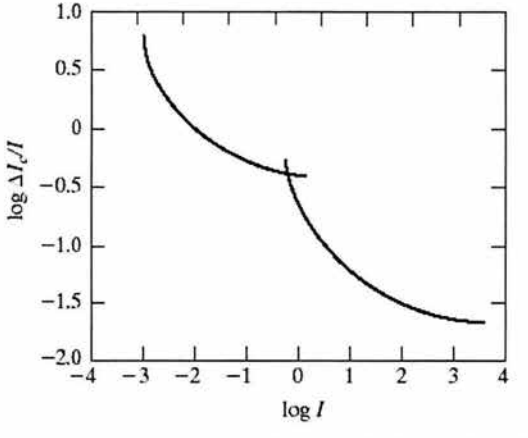


Fig 2.d-3 A altos niveles de iluminación se tiene una mejor discriminación del brillo.

Si la iluminación de fondo se mantiene constante y a la intensidad de la otra fuente, en vez de ser intermitente, se le permite variar incrementalmente desde no ser percibida nunca hasta ser percibida siempre, el observador típico puede discernir un total de una o dos docenas de cambios de intensidad. Este resultado se relaciona con el número de diferentes intensidades que una persona puede ver en cualquier punto de una imagen monocroma.

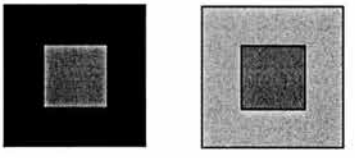
Este resultado no significa que esa imagen puede ser representada por tan pequeño número de valores de intensidades pues, al desplazarse el ojo por la imagen, el fondo promedio cambia, permitiendo que un conjunto *diferente* de cambios incrementales sea detectado en cada nuevo nivel de adaptación. La consecuencia real es que el ojo es capaz de un rango mucho más ancho de discriminación de intensidades.

2.e El sistema visual humano y la compresión de imágenes digitales

Hasta el momento se han revisado algunas de las características del sistema visual humano. En los siguientes puntos se pretende ligar algunas de estas y otras características a los procesos de compresión de imágenes digitales y marcas de agua, pues son parte la parte fundamental en que varios de los algoritmos se basan.

2.e.1 Contraste y umbrales de contraste

El brillo aparente de un punto cualquiera en el campo visual no sólo depende de la luminancia absoluta de ese punto sino también de las variaciones locales de las luminancias en su entorno. El contraste es la medida de estas variaciones relativas de luminancia y existen dos definiciones de contraste que se han usado para medir variaciones en patrones simples. El *contraste de Weber* [8] se usa para medir el contraste local de un solo objetivo de luminancia uniforme que se observa sobre un fondo también uniforme y se define como $C=\Delta L/L$, donde ΔL es la diferencia entre la luminancia del objetivo y la luminancia del fondo, L [7].



A) B)

Fig 2.e-1 Demostración de que el brillo aparente no depende sólo de la luminancia absoluta. El cuadrado interior parece más oscuro en B) que en A) a pesar de que ambos tienen el mismo valor absoluto de luminancia.

La segunda definición de contraste es el *contraste de Michelson*, usado para medir el contraste de un patrón periódico, por ejemplo, un patrón sinusoidal, se define como: $C = (L_{max}-L_{min}) / (L_{max}+L_{min})$, en donde L_{max} y L_{min} son respectivamente los valores máximo y mínimo de luminancia presentes. La siguiente figura ilustra el contraste de Michelson.

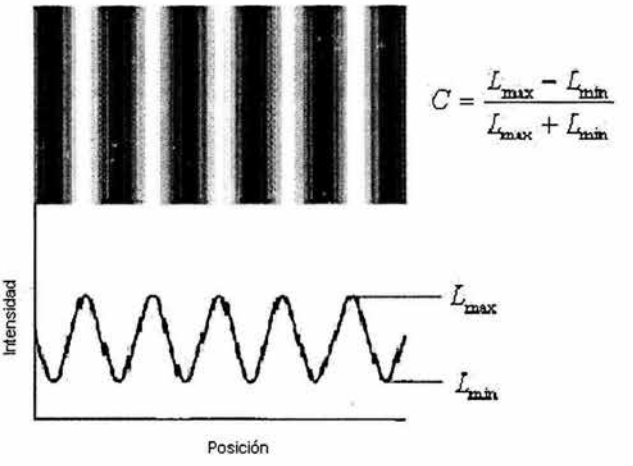


Fig 2.e-2 El contraste de Michelson se refiere a los cambios en la intensidad de un patrón periódico

El umbral de contraste se define como el mínimo nivel en que el contraste del objetivo visual es visible y se determina por medio de experimentos visuales. Por ejemplo, los experimentos visuales para el contraste de Weber se realizan de la siguiente manera: inicialmente la luminancia del objetivo es igual a la luminancia del fondo y las figuras 2.e-1.a) y 2.e-1.b) se presentan aleatoriamente a los sujetos. Entonces éstos, a una distancia específica del objetivo visual son interrogados acerca de cuál de las dos regiones (la interior al cuadrado o la exterior al mismo) es más brillante. Cuando la luminancia de las dos regiones es igual, el sujeto dará una respuesta correcta el 50% de las veces, entonces la luminancia del objetivo se incrementa hasta que el sujeto da una respuesta correcta el 75% de las veces. Este nivel de ΔL se define como la JND (*Just Noticeable Difference*) o diferencia apenas perceptible para esa luminancia de fondo. La razón de la JND a la luminancia de fondo es el umbral de contraste de Weber.

Para el contraste de Michelson, los umbrales de contraste se determinan como sigue:

Los sujetos se colocan a una determinada distancia de los objetivos visuales y son interrogados acerca de si pueden diferenciar el patrón de la figura 2.e-3.b) del patrón sin contraste de la figura 2.e-3.a). En caso de que no puedan, la amplitud del patrón se incrementa, como en 2.e-3.c) hasta que el sujeto logre distinguir el contraste el 50% de las veces. La razón de esta amplitud a la suma de los valores máximo y mínimo de luminancias presentes es el umbral de contraste para esa frecuencia espacial. Este umbral se mide *para cada frecuencia espacial.* La *sensitividad al contraste* para una frecuencia espacial es el inverso del umbral de contraste a esa misma frecuencia.

En la figura se observa una gráfica de la sensitividad al contraste como función de la frecuencia espacial. Muestra características pasa banda. El sistema visual humano es más sensible a las frecuencias espaciales medias, decreciendo notablemente después de una cierta frecuencia de corte tanto para frecuencias altas como para frecuencias bajas.

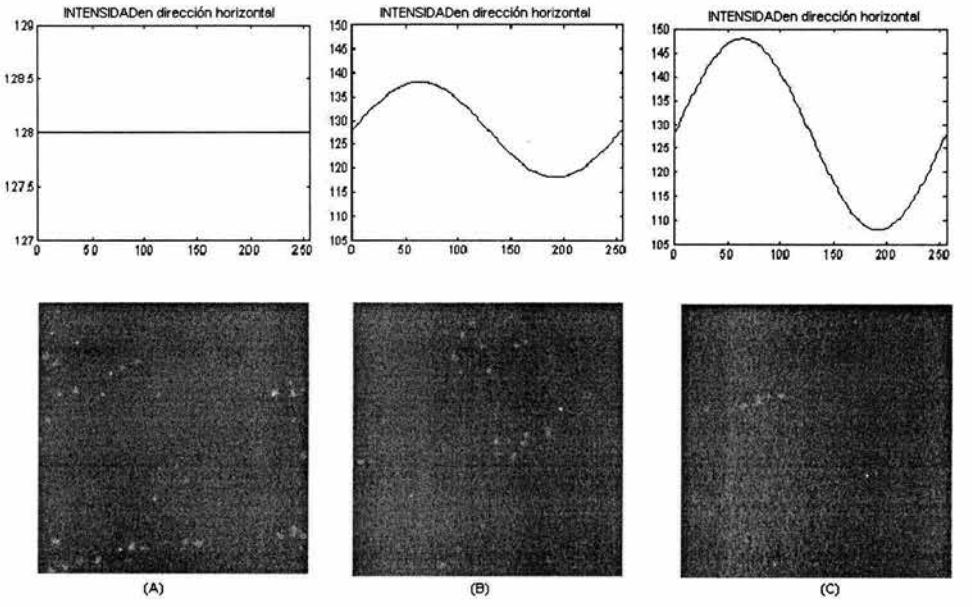


Fig 2.e-3 Configuración del experimento para medir el umbral de contraste para cada frecuencia espacial.

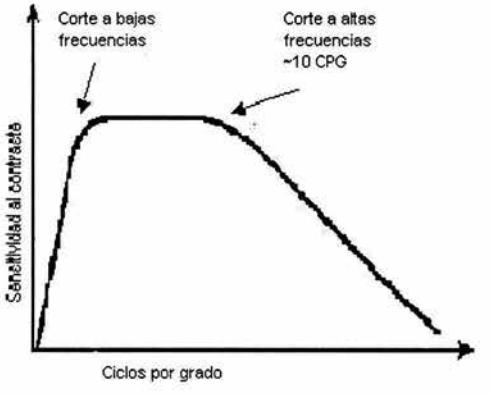


Fig 2.e-4 Sensitividad al contraste como una función de la frecuencia espacial

Adaptación al brillo

[9][10] El umbral de contraste para una frecuencia depende de la luminancia media del patrón sinusoidal, por ejemplo, en la figura 2.e-3, el umbral se mide para una intensidad media de 128. Si ésta fuera distinta, el umbral medido también sería distinto. El umbral de contraste se incrementa al incrementarse la luminancia media, fenómeno que se conoce como adaptación al brillo, como se mencionaba en la sección 2.d.2. En la figura 2.e-5, el cambio en el umbral se muestra como una función de la luminancia

media, L. al decrecer la luminancia media, el umbral también decrece. Nótese que los umbrales que se ilustran son medidos para determinar el nivel máximo de cuantización para una frecuencia espacial de la transformación DCT que producirá una distorsión imperceptible en la imagen resultante para el caso de una codificación basada en un bloque DCT de 8x8. en otras palabras, estos umbrales corresponden a la amplitud del patrón sinusoidal ilustrado en la figura 2.e-3. No corresponde al umbral de contraste, que es la razón de la amplitud del patrón sinusoidal a la media del mismo.

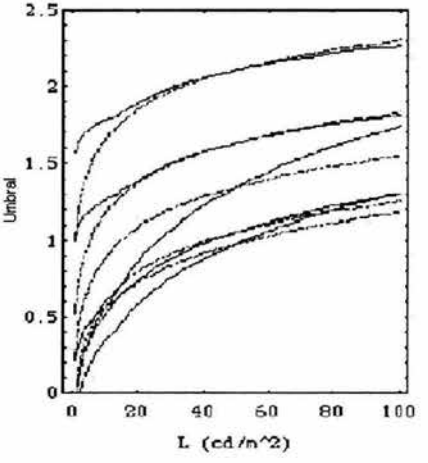


Fig 2.e-5 El cambio en el umbral de detección como una función de la luminancia media. De arriba abajo, las curvas para frecuencias espaciales DCT de {7,7}, {0,7}, {0,0}, {0,3} y {0,1}

Los umbrales que se muestran en 2.e-5 se formulan por la siguiente ecuación:

$$t_{ijk} = t_{ij}.(c_{ook} / \overline{c_{oo}})^{a_T}$$

En donde t_{ij} es el umbral para el (i,j)-ésimo coeficiente de la transformada DCT de 8x8 medida cuando la luminancia media corresponde a un nivel de gris de 128, c_{ook} es el coeficiente DC para el k-ésimo bloque de 8x8 de la imagen, a_T es el parámetro que controla la fuerza de la máscara, con un valor sugerido de 0.649, y c_{oo} testado es el coeficiente DC que corresponde a la luminancia media, que es igual a 1024 para una imagen de 8 bits. Entonces, se tiene que para un bloque de 8x8 con un valor medio de 128, $t_{ijk}=t_{ij}$.

Enmascaramiento de contraste

El enmascaramiento se refiere al efecto que tiene un estímulo sobre la detectabilidad de otro estímulo. Por ejemplo, un ruido fuerte puede hacer imperceptible un sonido más débil, como una conversación. En el caso de las imágenes, se refiere a la disminución en la visibilidad de una componente de la imagen debida a la presencia de otra imagen. Los experimentos referentes al enmascaramiento del contraste se realizan de la siguiente forma: se pide a los sujetos discriminar la superposición de las señales a y b (figura 2.e-6.c) de la señal b presentada independientemente (figura 2.e-6.b). La señal b es llamada *máscara*, mientras que la señal a es sencillamente, la *señal*. El contraste se varía para hallar el umbral de visibilidad. La configuración de los experimentos se muestra en las figuras 2.e-6 y 2.e-7. Mientras

para la figura 2.e-6 no hay una gran diferencia observable, ésta se hace más evidente en la figura 2.e-7, en la cual se ha incrementado la amplitud de la *señal.*
Para el caso de una codificación DCT basada en bloques de 8x8, hay 64 frecuencias DCT y cada frecuencia es enmascarada por ella misma y por otras 63 frecuencias, aunque hay algunos autores que consideran despreciables los efectos de enmascaramiento de otras componentes de frecuencia DCT, lo cual significaría que cada frecuencia es enmascarada sólo por ella misma. La formulación es la siguiente:

$$m_{ijk} = \max(t_{ijk}, |c_{ijk}|^{w_{ij}} \cdot t_{ijk}^{1-w_{ij}})$$

En donde m_{ijk} es el umbral enmascarado de la señal, c_{ijk} es el (i,j)-ésimo coeficiente DCT del k-ésimo bloque de la imagen, t_{ijk} es el umbral después de la adaptación a la luz y w_{ij} es un exponente con valores comprendidos entre cero y uno. La función es graficada en la figura 2.e.8 para el valor empírico típico de $w_{ij} = 0.7$ y $t_{ij} = 2$. El incremento en el contraste de la máscara trae consigo el incremento en el contraste de detección de la señal. En realidad, la función que muestra el cambio del umbral de contraste de la señal debería ser tetradimensional. El umbral de contraste es la variable dependiente y las variables independientes son el contraste y la frecuencia espacial de la máscara, y la frecuencia espacial de la señal.

Los datos en la figura 2.e-8 pueden interpretarse de la siguiente manera: asumiendo que el valor del (i,j)-ésimo coeficiente del k-ésimo bloque de la imagen es 10000, c_{ijk} es 10000 (pues la gráfica es logarítmica), el umbral de contraste que corresponde a $c_{ijk} = 10000$ es aproximadamente igual a 1000, por ejemplo, $m_{ijk} = 1000$. entonces el sistema visual humano no puede percibir la diferencia entre las dos imágenes, en donde una es la imagen original, con $c_{ijk} = 10000$, y la otra es la imagen modificada, con $c_{ijk} = 10000 \pm 1000$.

En resumen, la adaptación a la luz y el fenómeno de enmascaramiento del contraste se estudian para determinar los máximos niveles de cuantización dependientes de la imagen que producen distorsión imperceptible. En el proceso, la imagen es primero dividida en bloques de 8x8 y se calcula la transformada DCT de cada bloque. Los coeficientes DCT son ilustrados como c_{ijk}, donde i y j son frecuencias DCT y k es el número de bloque. El umbral de visibilidad para cada frecuencia espacial DCT (i,j), t_{ij}, se determina por medio de experimentos visuales (figura 2.e-3). entonces, los efectos de la luminancia media del patrón sinusoidal en el umbral de visibilidad se toman en cuenta. En el siguiente paso se inserta el efecto del enmascaramiento del contraste. Los umbrales resultantes dan los máximos niveles de cuantización que producirán distorsiones imperceptibles a la imagen y pueden usarse en aplicaciones como codificación de imágenes e inserción de marcas de agua de fuerza máxima que sean al mismo tiempo invisibles.

Nótese que se han medido los umbrales para frecuencias espaciales DCT pues la compresión de imágenes basada en DCT es ampliamente usada. Otro método de compresión es la compresión basada en wavelet, la imagen se descompone en subbandas cuya frecuencia espacial y orientación varían. Una cuantización uniforme de los coeficientes en cada subbanda tendría resultados visibles. Para eliminar las distorsiones visibles de la imagen comprimida, los umbrales de visibilidad que determinarán el máximo nivel de cuantización para cada subbanda se determinan por experimentos visuales. Estos umbrales son también usados para el marcado de imágenes. La marca de agua es insertada en coeficientes que son más grandes que esos umbrales. La fuerza de la marca de agua obviamente no debe exceder el umbral de visibilidad de la subbanda.

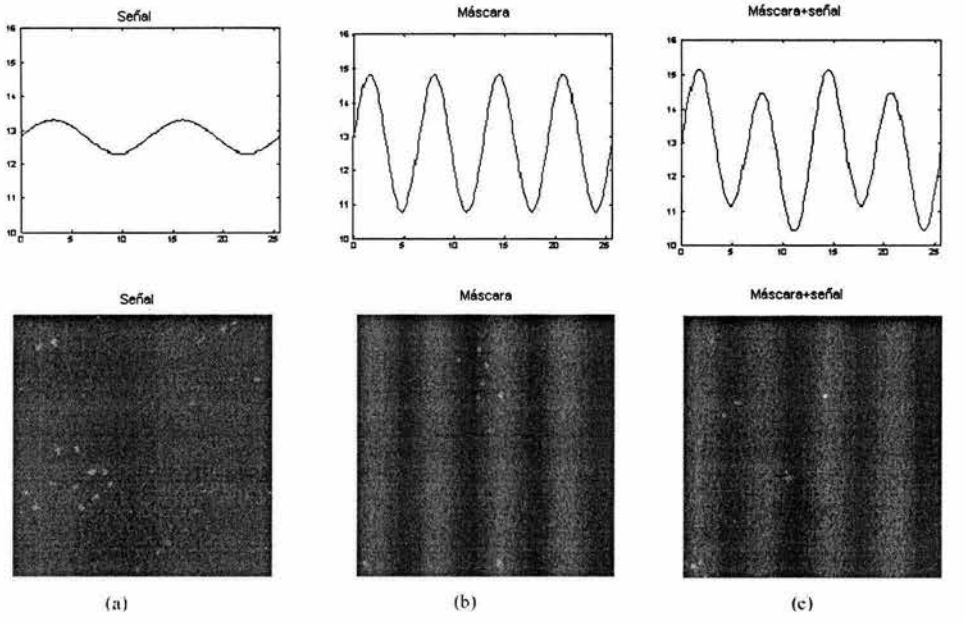


Fig 2.e-6 Esquema del experimento realizado para medir el enmascaramiento de contraste. La figura a es la señal. El objetivo es medir el umbral de contraste de la señal en presencia de la máscara b. Se pregunta al sujeto si es capaz de distinguir la máscara de la señal+máscara (señales b y c).

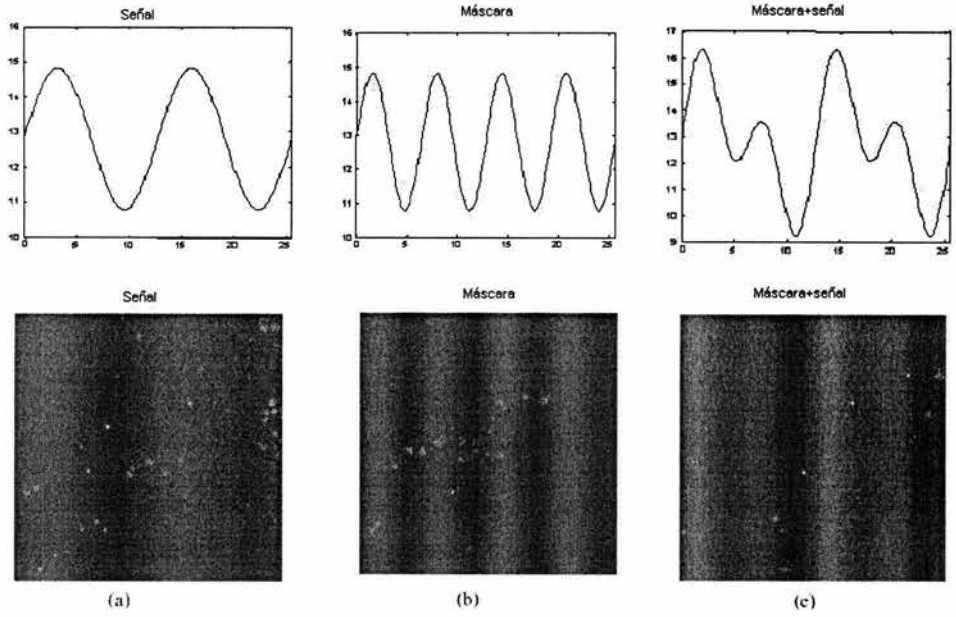


Fig 2.e-7 Al aumentar la amplitud de la señal, la diferencia entre las señales b y c se vuelve más evidente.

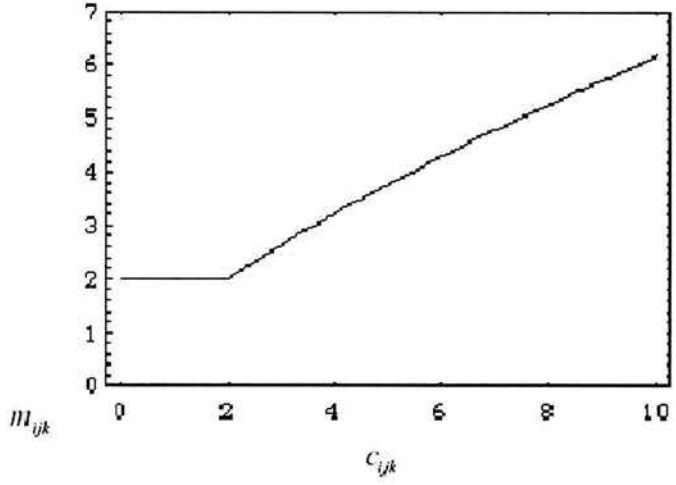


Fig 2.e-8 El cambio en el umbral de contraste es función del contraste de la máscara. Cijk es el contraste de la máscara y Mijk es el contraste de la señal. La gráfica es en escala logarítmica.

Enmascaramiento espacial y temporal

El fenómeno del enmascaramiento espacial y temporal se puede estudiar proponiendo un modelo espacio-temporal del umbral de la visión humana. Las predicciones arrojadas por el modelo se comparan con los datos experimentales de los fenómenos de enmascaramiento espacial y temporal del sistema visual humano. Tras corroborar que el modelo refleja las propiedades del SVH adecuadamente, se investiga la máxima tasa de bits (bit rate) para la codificación de la imagen tomando en consideración las características del SVH.

Enmascaramiento espacial se refiere al enmascaramiento de contornos de luminancia espacial. La configuración y resultados de los experimentos visuales conducidos para analizar el enmascaramiento espacial se dan en la figura 2.e-9. La varianza de la barra angosta de ruido blanco es incrementada hasta que el ruido es visible para los sujetos. La varianza en la que el ruido se vuelve visible es llamada *umbral de visibilidad*. El umbral de visibilidad se grafica como una función de la distancia entre la barra de ruido y los contornos espaciales y crece especialmente en el lado oscuro del contorno y un poco en el lado brillante del mismo. En términos de codificación de imágenes, este fenómeno nos hace pensar que las regiones de la imagen que se encuentran cercanas a un contorno espacial pueden cuantizarse a niveles más grandes para decrementar la cantidad de bits necesarios para dicha cuantización. La idea correspondiente en el dominio de las marcas de agua, es insertar una marca más fuerte en las regiones cercanas a un contorno espacial, a modo de incrementar la robustez de la marca.

El enmascaramiento temporal se refiere al enmascaramiento de discontinuidades de luminancia temporales. En los experimentos correspondientes, un flash (ruido) con una duración de 40 milisegundos se superpone a un campo espacialmente uniforme. Entonces, la luminancia del campo uniforme se cambia repentinamente de obscuro a brillante y de brillante a oscuro. Los umbrales de visibilidad se hacen mayores para transiciones tanto de obscuro a brillante como de brillante a obscuro de unos 100 milisegundos. Los resultados de estos experimentos y las predicciones del modelo propuesto se ilustran en la figura 2.e-10

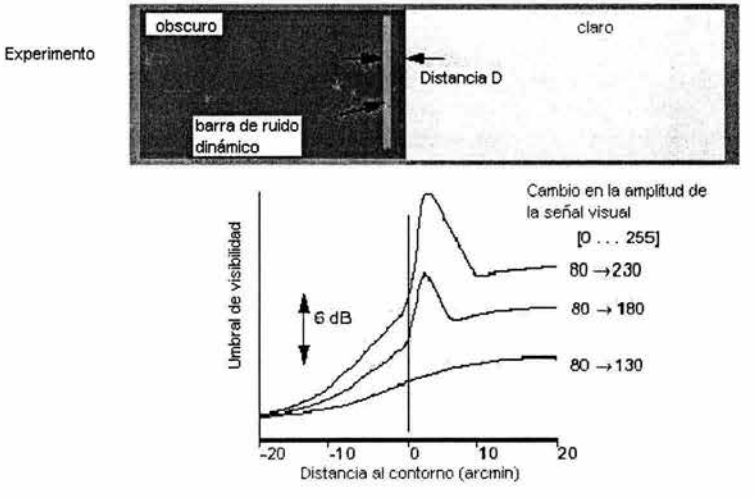


Fig 2.e-9 Umbrales de visibilidad para una barra angosta de ruido blanco en las cercanías de un contorno espacial.

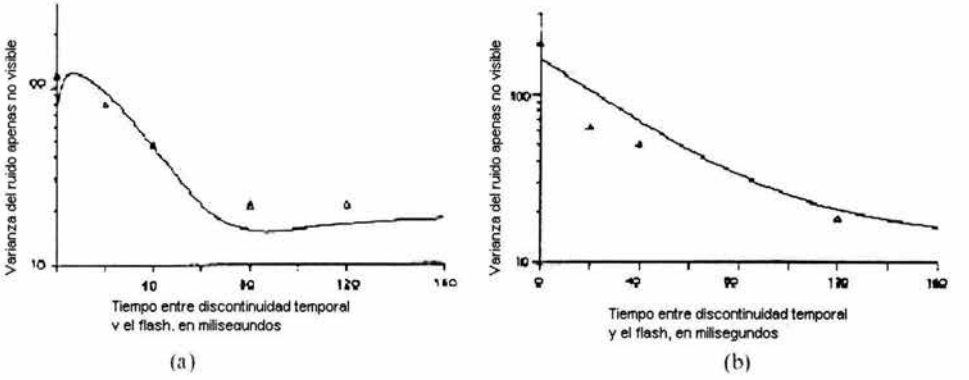


Fig 2.e-10 Umbrales de visibilidad para un flash de 40 ms de ruido blanco dinámico tras un salto en el brillo temporal que en a es de I=50 a I=80 y en b es de I=180 a I=50. Las Δ son umbrales de visibilidad medidos por experimentos visuales y las líneas sólidas son las predicciones del modelo.

Foveación

Como ya se ha discutido previamente, existen en la retina dos tipos de células sensibles a la luz: los bastones, encargados de la visión en blanco y negro, no detallada y a bajos niveles de intensidad luminosa; y los conos, encargados de la visión a color, con detalle y a altos niveles de intensidad luminosa. También se vio cómo la distribución de ambos tipos de receptores varía según su excentricidad, o distancia angular al centro de la fovea, siendo en este punto donde se encuentra la mayor concentración de conos. Esto tiene efectos muy interesantes en la visión: las imágenes que percibimos con mayor detalle y color se encuentran confinadas a una pequeña área del campo visual. La región de la imagen que se proyecta a la fovea es percibida claramente mientras que el resto del campo visual se percibe un poco borroso. A continuación se muestran dos imágenes de Lena, la versión original y la versión *foveada*. Si un observador mira exclusivamente al centro de la imagen de Lena, ambas versiones, la original y la foveada parecerán perceptualmente iguales.

(a) (b)

Fig 2.e-11 Imagen original de Lena (a) y la versión *foveada* (b).

Sensitividad temporal

La sensitividad temporal se refiere a la sensitividad del SVH respecto a fluctuaciones temporales en un patrón espacial. Estas fluctuaciones pueden ser lentas, como el crecimiento de una planta, o muy rápidas, como las fluctuaciones del nivel de intensidad de una lámpara.

De una manera más formal, la sensitividad temporal se refiere a la influencia de la dimensión temporal de la luz (estímulo para la visión) sobre la percepción del SVH. No sólo depende de la configuración temporal del estímulo visual sino también de la configuración espacial del objetivo, su tamaño, su iluminación de fondo y la iluminación en su entorno.

Referencias

[1] http://science.howstuffworks.com/eye1.htm. Por Carl Bianco, M.D., Artículo publicado en Howstuffworks en donde se estudia la visión humana desde un punto de vista anatómico y fisiológico.

[2] Floyd, R., y L. Steinberg (1975) "An Adaptive Algorithm for Spatial Gray Scale", Para la *Society for Information Display* Simposio de 1975, p. 36.

[3] Jarvis, J. F., C. N. Judice, and W. H. Ninke (1976) "A Survey of Techniques for the Image Display of Continuous Tone Pictures on Bilevel Displays" *Computer Graphics and Image Processing* **5**(1):13–40.

[4] Jarvis, J. F. and C. S. Roberts (1976) "A New Technique for Displaying Continuous Tone Images on a Bilevel Display" *IEEE Transactions on Communications* **24**(8):891–898, August.

[5] http://cnx.rice.edu/content/m11084/latest/, Artículo Human Vision, por Nick Kingsbury.

[6] Bernd Girod: EE368b Image and Video Compression Human Visual Perception no. 28

[7] Stefan Winkler, Pierre Vandergheynst, " Computing Isotropic Local Contrast from Oriented Pyramid Decompositons", in Proc. ICIP, Vol. 4, PP. 420-424, Kyoto, Japan, 1999.

[8] Jae S. Lim, *Two-dimensional Signal and Image Processing*, PP. 429-430. Prentice Hall, 1990.

[9] T. N. Cornsweet, *Visual Perception,* PP. 152-154. New York: Academic, 1970.

[10] A.B. Watson, "DCT quantization matrices visually optimized for individual images", Proc. of SPIE on Human Vision, Visual Processing, and Digital Display IV, 1993.

[11] G. E. Legge and J. M. Foley, "*Contrast Masking in Human Vision*", J. Opt. Soc. Am.,70(12), PP. 1458-1471, 1980.

[12] A. Watson, G. Yang, J. Solomon, J. Villasenor, "Visibility of wavelet quantization noise", IEEE Transactions on Image Processing, vol. 6, no.8, pp. 1164-1175, August, 1997

3. JPEG2000 (JOINT PHOTOGRAPHIC EXPERTS GROUP 2000)

Debido a la expansión continua de aplicaciones de multimedia e Internet, las necesidades y requerimientos de las tecnologías usadas crecieron y evolucionaron, por lo que en Marzo de 1997 un nuevo llamado para contribuciones fue emitido para el desarrollo de un nuevo estándar de compresión de imágenes fijas. En Noviembre de 1997, el comité de JPEG 2000 decidió sobre una estructura que usaba la codificación de sub-banda wavelet para un nuevo estándar. La decisión fue basada en los resultados de las pruebas de 24 propuestas mostradas por varias compañías y universidades. Estos resultados mostraron que la codificación wavelet es capaz de mejorar la calidad de una imagen objetivamente y subjetivamente mejor que la Transformada Coseno Discreta (TCD).

Para facilitar el desarrollo del estándar, un modelo de verificación fue establecido, éste es básicamente la especificación de un sistema de codificación compuesto de una colección de utilerías. Este proyecto tuvo la intención de crear un nuevo sistema de codificación de imágenes para diferentes tipos de imágenes fijas (bi-nivel, escalas de grises, color, multi-componentes), con diferentes características (imágenes naturales, científicas, médicas, sensoriales remotas, de prueba, gráficas, etc.) permitiendo diferentes modelos de imagen (cliente-servidor, transmisiones en tiempo real, archivos de colecciones de imágenes, buffer limitado y recursos de ancho de banda, etc.) preferentemente dentro de un sistema unificado. Este sistema de codificación proveería operaciones a bajos índices de bits con rendimientos superiores en índices de distorsión y calidad de imagen subjetiva que otros estándares ya existentes, sin sacrificar el rendimiento de otros puntos en el espectro del índice de distorsión, incorporando al mismo tiempo muchas características importantes.

El proceso de estandarización, el cual es coordenado por la JTC1/SC29/WG1 de la ISO/IEC produjo el diseño final del estándar internacional [1,2] (FDIS por sus siglas en inglés) y el estándar internacional (IS por sus siglas en inglés) fue registrado para Diciembre del 2000. Solo los cambios editoriales son esperados a este estado y por lo tanto, no habrá más cambios en la Parte I del estándar.

El estándar JPEG 2000 [3] tiene todas las características de los viejos estándares y muchos más en un codestream y formato de archivo coherente. Específicamente éste incluye:

- Codificación o decodificación sin pérdidas y con pérdidas (la decodificación sin pérdidas requiere de codificación sin perdidas)
- Progresión para ambas fidelidades por resolución y píxel seleccionables en el tiempo de decodificación
- Codificación y decodificación a tasas fijas y tamaños fijos
- Decodificación de regiones de interés
- Soporte de imágenes de color, escala de grises, gráficos, textos, bi-niveles
- Análisis gramatical del codestream sin decodificar
- Acceso aleatorio y procesamiento (operaciones de rotación, traslación, filtrado, etc.) del codestream
- Diferentes estilos de codificación para los múltiples componentes
- Descripción de color
- Rastreo de imagen (protección de la imagen)
- Inclusión de meta-datos (descripción basada en contenido)
- Arquitectura abierta

- Robustez en los errores de bits
- Decodificación fuera de orden

3.a Estructura del estándar

El estándar JPEG2000 esta comprendido en siete partes, como se lista en la tabla 2.1.

Parte	Título
1	Sistema de codificación principal
2	Extensiones
3	Movimiento JPEG-2000
4	Conformidad
5	Software de referencia
6	Formato de imagen compuesto
7	Reporte técnico

Tabla 3.a-1. Partes del estándar JPEG 2000.

La parte I [4] es el algoritmo básico (núcleo) y es libre de regalías. Éste especifica la mínima funcionalidad que un codec debe proveer para ser compatible con el estándar. La parte II consiste de tecnologías opcionales no requeridas para todas las implementaciones. Evidentemente las imágenes codificadas con la tecnología de la parte II no podrán ser decodificadas por un decodificador de la parte I. La parte II incluye cuantización de codificación de malla (TCQ por sus siglas en inglés), wavelet definidas por el usuario, esquemas mejorados de capacidad de recuperación de error, escalamiento basado en métodos de codificación ROI, codificación mezclada de longitud fija y variable, etc. La parte III añade movimiento a JPEG 2000 y esta basado en la parte I. Las ventajas adicionales de esta parte son video de alta calidad, capacidad de recuperación de error en el video en situaciones de canal ruidoso tales como wireless e Internet. El formato de archivo de movimiento de JPEG 2000 (MJ2) será interoperable con el formato de archivo JPEG 2000 (JP2) y el formato de archivo de MPEG-4 MP-4. La parte IV del estándar define las pruebas de conformidad y la parte V describe los dos sistemas de referencia como software de alta calidad (libres). Actualmente dos implementaciones de software de referencia existen: el software JJ2000 (desarrollado por Canon Research France, EPFL y Ericsson), que es una implementación en Java del JPEG2000. El software JasPer es una implementación en C por Image Power y la Universidad de British Columbia. La parte VI define el formato de imagen compuesto, mientras la parte VII concluye la serie de JPEG2000 con un reporte técnico con la directriz de una función de soporte mínimo de la parte I.

3.b Arquitectura básica del estándar

El motor de compresión de JPEG2000 (codificador y decodificador) es ilustrado en el diagrama de bloques de la figura 2.1. En el codificador, la transformada discreta es primero aplicada sobre los datos de la imagen fuente. Los coeficientes transformados son entonces cuantizados y codificados

entrópicamente antes de formar la salida del codestream (bitstream). El decodificador es el proceso inverso del codificador.

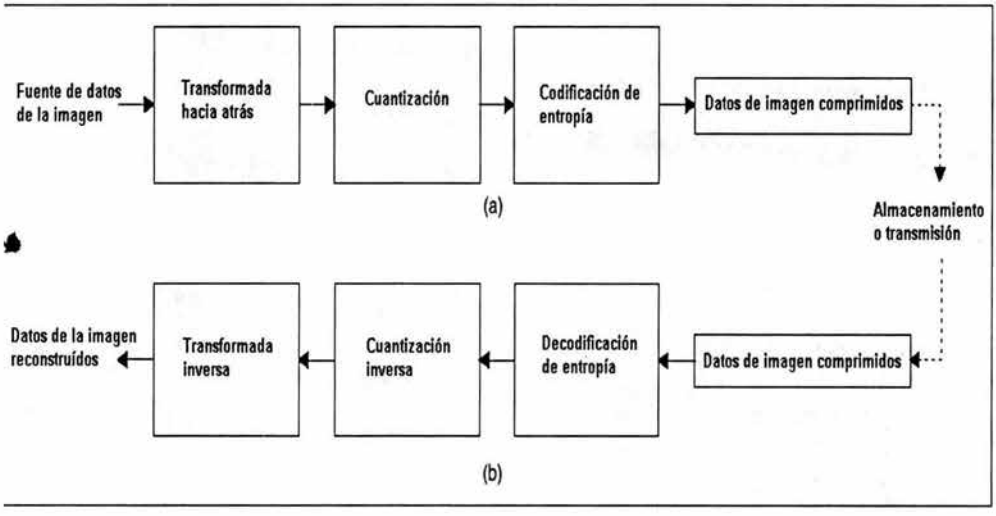


Figura 3.b-1 Diagrama de bloques general de JPEG2000

Aunque este diagrama de bloques general se parece a un codificador JPEG convencional, hay diferencias radicales en todos los procesos de cada bloque del diagrama. Una rápida visión general del sistema completo es la siguiente:

- La imagen fuente es descompuesta en componentes.
- Los componentes de la imagen son (opcionalmente) descompuestos en tiles rectangulares. La componente-tile es la unidad básica de la imagen original o la imagen reconstruida.
- Una transformada wavelet es aplicada sobre cada tile. El tile es descompuesto en diferentes niveles de resolución.
- Los niveles de descomposición se componen de subbandas de coeficientes que describen las características de frecuencias de las áreas locales de los componentes de los tiles.
- Las subbandas de coeficientes son cuantizadas y colocadas en arreglos rectangulares de "codeblocks".
- Los planos de bits de coeficientes en un codeblock (por ejemplo, los bits de igual importancia a través de los coeficientes en un codeblock) son codificados entrópicamente.
- La codificación puede ser hecha de tal forma que ciertas regiones de interés puedan ser codificadas a calidad mayor que el resto.
- Los marcadores son añadidos al bitstream para permitir la capacidad de recuperación de error (error resilience).
- El codestream tiene un encabezado principal al inicio que describe la imagen original y varios estilos de descomposición y codificación que son usados para localizar, extraer, decodificar y reconstruir la imagen con las resoluciones, fidelidad, regiones de interés u otras características deseadas.

- El formato de archivo opcional describe el significado de la imagen y sus componentes en el contexto de la aplicación.

Habría que hacer notar que el motor de codificación básico de JPEG2000 esta basado en el algoritmo de codificación de bloque incrustado con un truncamiento optimizado de bitstreams incrustados (EBCOT por sus siglas es inglés) [5,6].

3.b.1 Modelo de imagen fuente.

Antes de analizar internamente el codec, es importante entender el modelo de imagen que el codec empleará. Desde el punto de vista de un codec, una imagen esta compuesta de una o más componentes (hasta un límite de 2^{14}) como se muestra en la figura 2.2a. Como se ilustra en la figura 2.2b cada componente consiste de arreglos rectangulares de muestras. Los valores de las muestras de cada componente son de valor entero, y pueden ser ambas con signo o sin signo con una precisión de 1 a 128 bits/muestra. El signo y la precisión de los datos muestra son especificados sobre una base por componente.

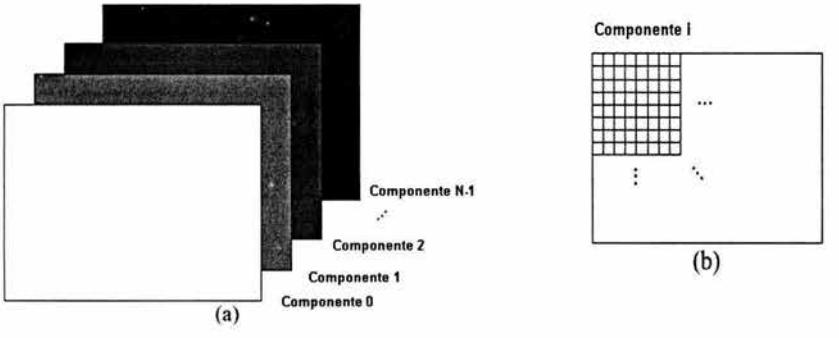


Figura 3.b.1-1 Modelo de imagen de fuente a) imagen con N componentes b) componente individual.

Todos los componentes son asociados con la misma dimensión espacial en la imagen fuente, pero representan diferentes espectros o información auxiliar. Por ejemplo, una imagen de color RGB tiene tres componentes, donde una componente representa cada uno de los planos de color rojo, verde y azul. En el caso de una imagen en escala de grises, hay solo una componente, que corresponde al plano de luminancia. Las diversas componentes de una imagen no necesitan ser muestreadas con la misma resolución, por lo cual las componentes por sí mismas pueden tener diferentes tamaños.

3.b.2 Cuadricula de referencia

Dada una imagen el codec describe la geometría de las diversas componentes en términos de una cuadricula rectangular llamada cuadricula de referencia. La cuadricula de referencia tiene la forma general mostrada en la figura 2.3. La cuadricula es de tamaño XsizxYsiz con el origen localizado en su esquina izquierda superior. La región de la esquina izquierda superior (X0sizxYosiz) a la esquina derecha inferior (Xsiz-1xYsiz-1) es llamada el área de imagen, y corresponde a los datos de la pintura a

representar. El ancho y el largo de la cuadricula de referencia no puede exceder 2^{23}-1 unidades, imponiendo un límite superior en el tamaño de una imagen que puede ser tratado por el codec.

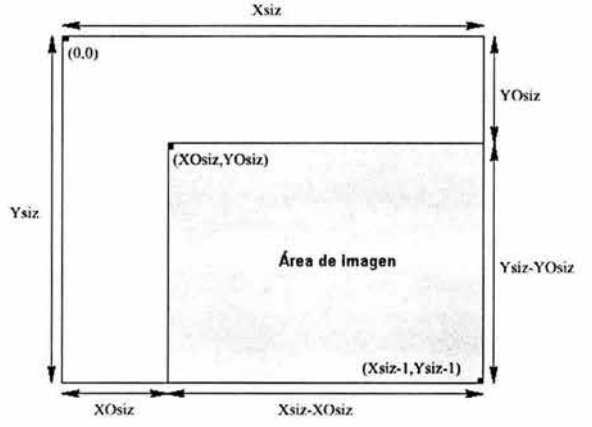


Figura 3.b.2-1 Cuadricula de referencia.

Todos los componentes son representados sobre el área de imagen de la cuadricula de referencia. Puesto que los componentes no necesitan ser muestreados a la resolución completa de la cuadricula de referencia, información adicional es requerida para establecer su correspondencia. Para cada componente se indica el período de muestreo horizontal y vertical en unidades de la cuadricula de referencia, denotadas como XRsiz y YRsiz, respectivamente. Estos dos parámetros únicamente especifican una cuadricula de muestreo (rectangular) constando de todos los puntos cuyas posiciones horizontales y verticales son múltiplos enteros de XRsiz y YRsiz, respectivamente. Todos los puntos que caigan dentro del área de imagen, constituyen muestras de la componente en cuestión. Así, en términos de la del sistema de coordenadas, una componente tendrá el tamaño $\left(\left\lceil\frac{Xsiz}{XRsiz}\right\rceil-\left\lceil\frac{X0siz}{XRsiz}\right\rceil\right)\times\left(\left\lceil\frac{Ysiz}{YRsiz}\right\rceil-\left\lceil\frac{Y0siz}{YRsiz}\right\rceil\right)$ y su muestra izquierda superior corresponde al punto $\left(\left\lceil\frac{X0siz}{XRsiz}\right\rceil,\left\lceil\frac{Y0siz}{YRsiz}\right\rceil\right)$.

Del diagrama, el tamaño del área de imagen es (Xsiz-X0siz)×(Ysiz-Y0siz). Para una imagen dada, muchas combinaciones de los parámetros Xsiz, Ysiz, X0siz y Y0siz pueden ser escogidos para obtener un área de imagen con el mismo tamaño. Los parámetros X0siz y Y0siz no son fijos a cero mientras que los parámetros Xsiz y Ysiz son conjunto para el tamaño de la imagen. Por lo que hay sutiles implicaciones para cambiar los parámetros X0siz y Y0siz (guardando el tamaño del área de imagen constante). Tales cambios afectan las características del codec en diversas formas importantes. Estas características permiten un número de operaciones básicas para ser llevadas a cabo eficientemente en las imágenes codificadas tales como cortado, flipping horizontal/vertical, y rotación por un entero múltiple de 90 grados.

3.b.3 Tiling

El término tiling se refiere a la partición de la imagen fuente original dentro de bloques rectangulares no sobrepuestos (tiles) sobre la cuadricula de referencia, los cuales son comprimidos independientemente como si fueran imágenes enteramente distintas. Todas las operaciones incluyendo la mezcla de componentes, la transformada wavelet, la cuantización y codificación de entropía son llevadas a cabo independientemente de los tiles imagen como se muestra en la figura 3.b.3-1. La componente tile es la unidad básica de la imagen original o reconstruida. El tiling reduce los requerimientos de memoria y puesto que son también reconstruidos independientemente, pueden ser usados para decodificar partes especificas de la imagen en lugar de toda la imagen. Todos los tiles tienen exactamente las misma dimensiones, excepto quizás los localizados en la frontera de la imagen. Tamaños arbitrarios de tiles son permitidos hasta incluir la imagen entera o a ella misma, es decir un tile puede ser la imagen entera.

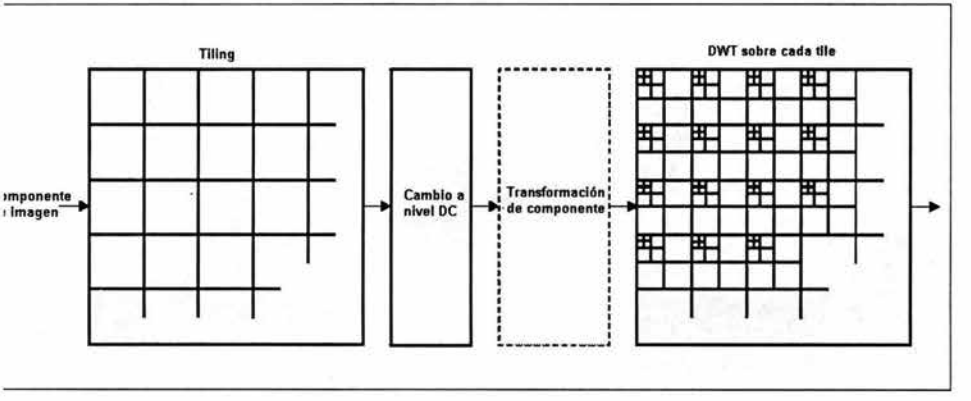
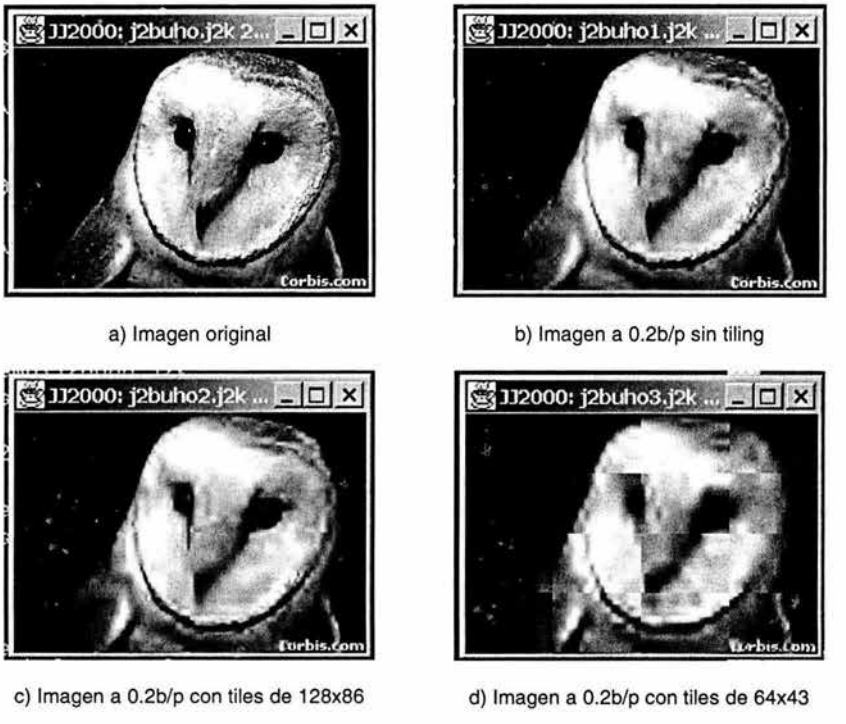


Figura 3.b.3-1 Tiling, cambio a nivel DC, transformada de color (opcional) y DWT de cada componente de la imagen

Los componentes con diferentes factores de submuestreo son tileados con respecto a una cuadricula de alta resolución, la cual asegura la consistencia espacial sobre los componentes tile resultantes. Como es de esperar el tiling afecta la calidad de la imagen objetivamente y subjetivamente. Tiles más pequeños crean más artefactos tiling comparados con tiles más grandes (los valores de PSNR son el promedio de todos los componentes). En otras palabras, los tiles más grandes funcionan visualmente mejor que los tiles más pequeños. La degradación es más severa en el caso de índices de bits bajos que en el caso de índices de bits altos. Esto se puede observar en la figura 3.b.3-2.



a) Imagen original

b) Imagen a 0.2b/p sin tiling

c) Imagen a 0.2b/p con tiles de 128x86

d) Imagen a 0.2b/p con tiles de 64x43

Figura 3.b.3-2 Imagen original y con tiling

3.b.4 Cambio a nivel DC

El codec espera que sus datos de muestra de entrada tengan un rango dinámico nominal que este simétricamente distribuido alrededor del cero. El estado de preprocesamiento del codificador simplemente asegura que esta esperanza sea conocida. Supóngase que una componente particular tiene P bits/muestra. Las muestras pueden ser con signo o sin signo, conducidas a un rango dinámico nominal de $[-2^{P-1}, 2^{P-1}-1]$ y $[0, 2^{P-1}]$, respectivamente. Si los valores de las muestras son sin signo, el rango dinámico nominal claramente es no simétrico alrededor del cero. Por lo que prioritario a la operación de la transformada wavelet discreta (DWT por sus siglas en inglés) sobre cada tile imagen, todas las muestras de la componente tile imagen son cambiadas al nivel DC por la sustracción de la misma cantidad del sesgo 2^{P-1}, donde P es la precisión de componente. El cambio de nivel DC solamente es llevado a cabo sobre las muestras sin signo de las componentes. El cambio de nivel no afecta las variancias. Esto actualmente convierte una representación sin signo a una representación de complemento a dos, o viceversa. Si la transformada de color es usada, el cambio a nivel DC es llevada a cabo antes de la operación de la transformada de componente hacia delante como se muestra en la figura 2.4 y 2.6. En el lado del decodificador el cambio de nivel DC inverso es llevado a cabo sobre las muestras reconstruidas por la adición a ellas del sesgo de 2^{P-1} después de la operación de la transformada de componente inversa.

3.b.5 Transformación de componentes

JPEG 2000 soporta imágenes de múltiples componentes. Por lo que en el codificador, el estado de preprocesamiento es seguido por un estado de transformación de componentes, que opera sobre todos los componentes juntos y sirve para reducir la correlación entre los componentes permitiendo una codificación más eficiente.

El estándar soporta dos diferentes transformaciones de componentes, una transformación de componente irreversible (ICT por sus siglas en inglés) que puede ser usada para codificación con pérdidas y una transformación de componentes reversibles (RCT por sus siglas en inglés) que puede ser utilizada para codificación con pérdidas y sin pérdidas. El diagrama de bloques del codificador de multicomponentes de JPEG2000 es descrito en la figura 3.b.5-1.

Las transformadas son definidas para operar sobre las tres primeras componentes de una imagen, con la suposición que los componentes 0, 1 y 2 corresponden a los planos de color rojo, verde y azul. Debido a la naturaleza de estas transformadas, las componentes en las cuales las transformadas operan deben ser muestreadas a la misma resolución. Como consecuencia de los hechos de arriba, la ICT y RCT solamente pueden ser empleados cuando la imagen codificada tiene por lo menos tres componentes, y las primeras tres componentes son muestreadas a la misma resolución. Después del estado de transformación de componentes en el codificador, los datos de cada componente son tratados independientemente.

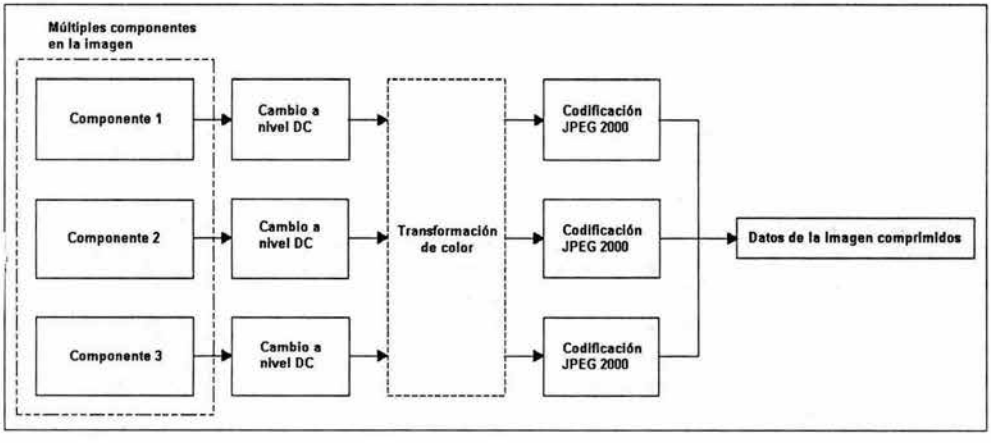


Figura 3.b.5-1 Codificador de componente múltiple de JPEG 2000. La transformación de color es opcional. Si es empleada esta puede ser reversible o irreversible

La ICT es nada más que la transformada de espacio color RGB a YCrCb clásica. Esta transformada solamente puede ser usada con la transformada wavelet irreversible 9/7 La transformada hacia delante es definida como:

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

La transformada inversa es mostrada por:

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

La RCT es simplemente una aproximación entero a entero reversible, ésta además solo puede ser usada con la transformada wavelet reversible 5/3. La transformada hacia delante es dada por:

$$\begin{pmatrix} Y \\ V \\ U \end{pmatrix} = \begin{pmatrix} \left\lfloor \frac{R+2G+B}{4} \right\rfloor \\ R-G \\ B-G \end{pmatrix}$$

La transformada inversa es dada por:

$$\begin{pmatrix} G \\ R \\ B \end{pmatrix} = \begin{pmatrix} Y-\left\lfloor \frac{U+V}{4} \right\rfloor \\ V+G \\ U+G \end{pmatrix}$$

Si la transformada de multicomponentes fue aplicada durante la codificación, su inversa es aplicada en la decodificación. A menos que la transformada sea irreversible, sin embargo, la inversión solo puede ser aproximada debido a los efectos de la aritmética de precisión finita.

3.b.6 Transformada Wavelet

El estándar JPEG2000 emplea una trasformada wavelet discreta (DWT por sus siglas en inglés) para operar sobre los componentes individuales. A través de esta transformada, una componente tile es dividida en numerosas bandas de frecuencias (subbandas). Estas subbandas consisten de coeficientes que describen la frecuencia espacial horizontal y vertical características de la componente tile original. Debido a las propiedades estadísticas de estas señales de subbandas, los datos transformados pueden ser codificados más eficientemente que los datos originales sin transformar.

Ambas transformadas DWT reversible e irreversible pueden ser empleadas en el estándar y este último soporta dos modos de filtrado: el basado en convolución y el basado en levantamiento. En ambos modos para facilitar el filtrado en las fronteras de la señal se emplea una extensión simétrica periódica [7]. El bloque de construcción básica usando levantamiento para cada transformada es la descomposición de subbanda de una dimensión (1-D) por 2 canales (muestras paso altas y paso bajas) del banco de filtros máximamente decimados uniformemente (UMDFB por sus siglas en inglés) cuya forma general se muestra en la figura 2.7. El lado de análisis del UMDFB descrito en 6a es asociado con la transformada hacia delante y el descrito en la figura 3.b.6-1 es asociado con la transformada inversa. En el diagrama $\{A_i(z)\}_{i=0}^{\lambda-1}$, $\{Q_i(z)\}_{i=0}^{\lambda-1}$ y $\{s_i(z)\}_{i=0}^{\lambda-1}$ denotan la función de transferencia del filtro, el operador de cuantización y la ganancia (escalar), respectivamente. Como una imagen es una señal de 2-D, claramente renecesitaría un UMDFB de 2-D, pero en la aplicación de un UMDFB de 1-D de manera

separada, se obtiene eficientemente un UMDFB de 2-D. Entonces la transformada wavelet es calculada por la aplicación recursiva de un UMDFB de 2-D para las señales de subbanda paso bajas obtenidas de cada nivel de descomposición. Una muestra paso bajas representa un muestreo bajo, una versión de resolución baja del conjunto original; mientras una muestra paso altas representa un versión residual de muestreo bajo del conjunto original, necesitada para la reconstrucción perfecta del conjunto original del conjunto paso bajas.

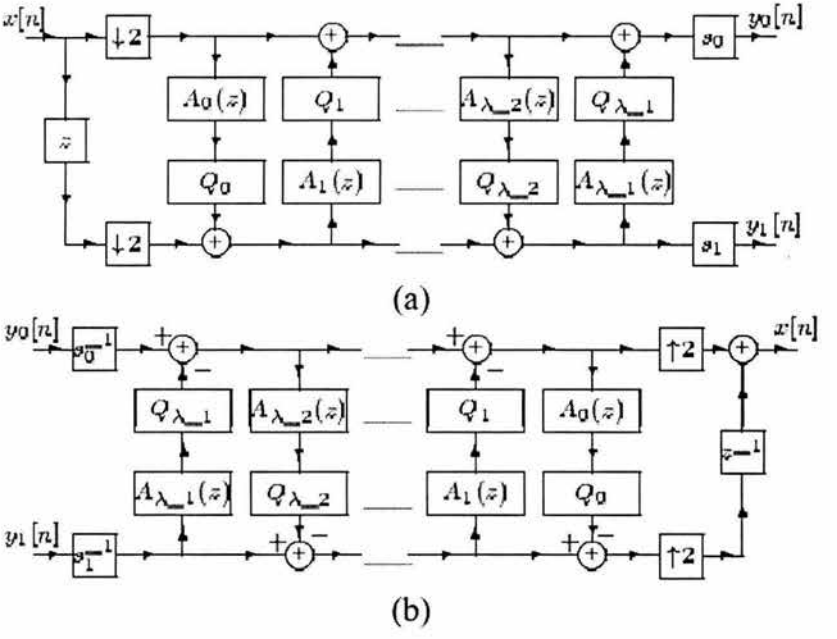


Figura 3.b.6-1 Realización de UMDFB de 2 canales de 1-D. a) lado de análisis b) lado de síntesis

Supóngase que una transformada wavelet de nivel (*R-1*) es aplicada a los datos de una componente tile, los cuales pasan de una manera iterativa a través de los filtros, produciendo señales de subbandas. Cada aplicación del lado de análisis de los filtros produce cuatro subbandas: una paso bajas horizontal y vertical (LL), una paso bajas horizontal y paso altas vertical (LH), una paso altas horizontal y paso bajas vertical (HL) y una paso altas horizontal y vertical (HH). Una descomposición wavelet de nivel (*R-1*) es asociada con *R* niveles de resolución, numerados del *0* hasta *R-1*, con *0* y *R-1*, correspondiendo a las resoluciones más gruesas y más finas, respectivamente. Cada subbanda de la descomposición es identificada por su orientación (por ejemplo LL, LH, HL, HH) y su correspondiente nivel de resolución (por ejemplo 0,1,...,R-1). La señal de componente tile de entrada es considerada para ser una banda LL_{R-1}. A cada nivel de resolución (excepto la más baja) de la banda LL es otra vez descompuesta. Por ejemplo, la banda LL_{R-1} es decompuesta para producir las bandas LL_{R-2}, LH_{R-2}, HL_{R-2} y HH_{R-2}. Entonces para el próximo nivel, la banda LL_{R-2} es decompuesta y así por el estilo. Este proceso se repite hasta obtener la banda LL_0 y resulta en una estructura de subbandas como la que se ilustra en la figura 3.2.6-2.

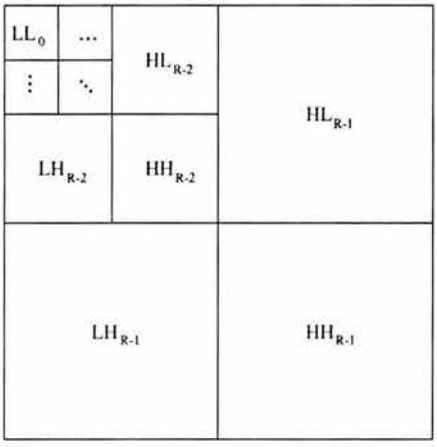


Figura 3.b.6-2 Estructura de subbandas

Teniendo la estructura de transformación general se pueden describir las dos transformadas soportadas por el estándar, la transformada 5/3 y 9/7. La transformada 5/3 es reversible, entero a entero y no lineal. Esta transformada fue propuesta en [8], y es simplemente una aproximación a una transformada wavelet lineal propuesta en [9] por Le Gall. Como la transformada 5/3 es reversible puede ser empleada para ambas codificaciones con pérdidas y sin pérdidas.

La transformada 9/7 es no reversible y real a real. Sin embargo es también lineal en aritmética exacta. Esta transformada propuesta en [10] por Daubechies, es empleada en el estándar de compresión de huellas digitales del FBI. La transformada 9/7, carente de la propiedad reversible, puede ser solamente usada para codificación con pérdidas. El número de niveles de resolución es un parámetro de cada transformada. Un valor típico para este parámetro es seis (asumiendo una imagen lo suficientemente grande).

La transformada inversa wavelet se realiza en el decodificador, pero debido a los efectos de la aritmética de precisión finita, el proceso inverso no es garantizado para ser exacto si la transformada reversible fue empleada.

3.b.7 Cuantización

Después de la transformación, todos los coeficientes son cuantizados. La cuantización permite mayor compresión en la representación de los coeficientes de la transformada con solo la precisión mínima requerida para obtener el nivel deseado de calidad en la imagen. En otras palabras, la cuantización facilita una compresión mayor por el descartamiento de información visualmente insignificante. La cuantización es una de las dos fuentes primarias de pérdidas de información en la ruta de la codificación.

Los coeficientes transformados son cuantizados usando una cuantización escalar con una zona muerta cerca del origen en la parte I del estándar, y una cuantización vectorial de codificación de malla en la parte II del estándar. Un diferente cuantizador es empleado para los coeficientes de cada subbanda, y cada cuantizador tiene solo un parámetro, su tamaño de paso. Matemáticamente en el estándar el proceso de cuantización es definido por:

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

donde Δ_b es el tamaño del paso de cuantización, $a_b(u,v)$ son los coeficientes de transformación de la subbanda b y $q_b(u,v)$ es la salida del cuantizador de la subbanda.

En el caso de una compresión con pérdidas el tamaño del paso del cuantizador es elegido en conjunción con el control de índice. En el caso sin pérdidas, el tamaño del paso del cuantizador es forzado a ser uno. El tamaño del paso de cuantización es especificado relativo al rango dinámico de subbanda *b*.

En el decodificador, la decuantización intenta deshacer los efectos de la cuantización. Si los coeficientes de la transformada son enteros y el tamaño del paso de cuantización son todos iguales a uno (como en el caso de codificación sin pérdidas), habrá pérdida de información, ya que el proceso de inversión es solamente aproximado. Los valores de los coeficientes cuantizados son obtenidos de los índices del cuantizador. Matemáticamente, el proceso de decuantización es definido como:

$$a_b(u,v) = \left(q_b(u,v) + r\, sign(q_b(u,v))\right) \cdot \Delta_b$$

donde r es un parámetro de sesgo. Aunque el valor de r normativamente no es especificado en el estándar, probablemente muchos decodificadores usaran el valor de un medio.

3.b.8 Codificación de entropía

La codificación de entropía es lograda por medio de un sistema de codificación aritmética que comprime símbolos binarios relativos a un modelo de probabilidad adoptiva asociado con cada 18 contextos de codificación diferente. El algoritmo de codificación MQ es usado para llevar a cabo esta tarea y administrar la adaptación de los modelos de probabilidad condicional. Este paso se describe más adelante.

3.b.9 Recintos - Capas - Paquetes

Después de la cuantización, cada subbanda es dividida dentro de bloques rectangulares, es decir rectángulos no sobrepuestos. El cuantizador indica para cada subbanda la repartición en bloques de código (codeblocks). Los codeblocks son rectangulares en forma, y su tamaño nominal es un parámetro libre del proceso de codificación, conforme a ciertas limitaciones, más notablemente: la anchura nominal y la altura del codeblock debe ser un número entero con potencia de dos, y el producto de la anchura nominal y altura no puede exceder de 4096.

Supóngase que el tamaño nominal del codeblock es tentativamente elegido para ser 2^{xcb} x 2^{ycb}. Los codeblocks se agrupan en lo qué se llaman recintos. Puesto que los codeblocks no pueden cruzar los límites del recinto, una reducción en el tamaño nominal del codeblock puede ser requerida si el tamaño del recinto es suficientemente pequeño. Supóngase que el tamaño nominal del codeblock después de cualquier ajuste es $2^{xcb'}$ x $2^{ycb'}$ donde xcb' ≤ xcb y ycb' ≤ ycb. La subbanda es repartida en bloques de código (codeblocks) sobreponiendo la subbanda con una cuadrícula rectangular que tiene espaciamientos horizontales y verticales de $2^{xcb'}$ x $2^{ycb'}$ respectivamente, según lo mostrado en figura 3.2.9-1. El origen de esta cuadrícula se ancla en (0,0) en el sistema de coordenadas de la subbanda. Una opción típica para el tamaño nominal del codeblock es 64x64 (es decir, xcb=6 y ycb=6) y no menos de 32x32.

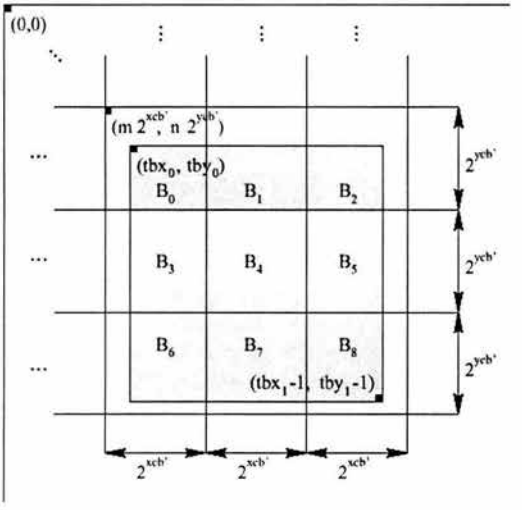


Figura 3.b.9-1 Repartición de las subbandas en codeblocks y recintos

Después de que una subbanda ha sido repartida en codeblocks, cada uno de los codeblocks se codifica independientemente. La codificación se realiza usando un codificador de plano de bit (bit-plane) descrito más adelante. Para cada codeblock, se produce un código encajado (embedded), compuesto de numerosos pasos de codificación.

En el lado del decodificador, los pasos de codificación del bit-plane para varios codeblocks, son decodificados, y los datos que resultan son reunidos en subbandas. De esta manera, se obtienen los índices reconstruidos del cuantizador para cada subbanda. En el caso de una codificación con pérdidas, los índices reconstruidos del cuantizador pueden ser solamente aproximaciones de los índices del cuantizador originalmente disponibles en el codificador. Esto es atribuible al hecho de que el codestream puede incluir solamente un subconjunto de pasos de codificación generados en la codificación. En el caso sin pérdidas, los índices reconstruidos del cuantizador deben ser iguales a los índices originales del codificador, puesto que todos los pasos de codificación deben ser incluidos en la codificación sin pérdidas.

3.b.10 Codificación del bit-plane

Después de que todos las subbandas se han repartido en los bloques del código (codeblocks), cada uno de los codeblocks que resultan se codifican independientemente usando un codificador bit-plane. Aunque la técnica de codificación del bit-plane empleada es similar a las usadas en los codecs de embedded zerotree wavelet (EZW) y en set partitioning in hierarchical trees (SPIHT), hay dos diferencias notables: no se explota ninguna dependencia de interbanda, y hay tres pasos de codificación por bit-plane en vez de dos. La primera diferencia sigue el hecho de que cada codeblock está contenido totalmente dentro de un sola subbanda, y los codeblocks son codificados independientemente uno del otro. No explotando las dependencias de interbanda, mejorando la recuperación de error que puede ser alcanzada. La segunda diferencia es discutiblemente menos fundamental. Usar tres pasos por bit-plane en vez de dos reduce la cantidad de datos asociados a cada paso de codificación, facilitando un control más fino sobre el índice de cambio.

Los tres pasos de codificación por bit-plane en orden son: significación, refinamiento y limpieza. Los tres tipos de pasos de codificación exploran las muestras de un codeblocks en el mismo orden fijo mostrado en la figura 3.2.10-1. El codeblocks se divide en franjas horizontales, cada una tiene una altura nominal de cuatro muestras. Si la altura del codeblock no es un múltiplo de cuatro, la altura de la franja inferior será menor que el valor nominal. Según lo mostrado en el diagrama, las franjas se exploran de arriba hacia abajo. Dentro de una franja, las columnas se exploran de izquierda a derecha. Dentro de una columna, las muestras se exploran de arriba hacia abajo. Es decir, empezando de la esquina izquierda superior, los primeros cuatro bits de la primera columna se exploran. A continuación los primeros cuatro bits de la segunda columna, hasta que el ancho del codeblock se cubre. Entonces los segundos cuatro bits de la primera columna se exploran y así por el estilo. Una exploración vertical similar continúa para cualquier fila sobrante en los codeblocks más bajos de la subbanda [11]. Esta altura de 4 en la franja se ha seleccionado cuidadosamente para facilitar implementaciones eficientes de hardware y software.

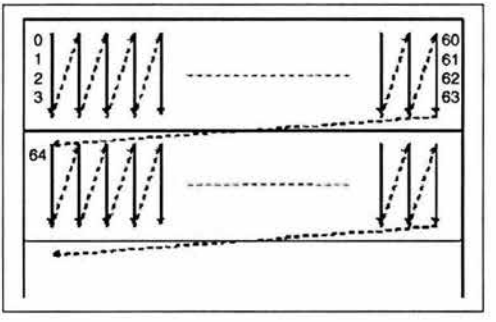


Figura 3.b.10-1 Orden de exploración de las muestras en un codeblock

Cada paso de codificación genera una secuencia de símbolos. Algunos o todos estos símbolos pueden ser codificados entrópicamente. Para los propósitos de la codificación de entropía, un codificador aritmético adaptativo basado en contexto es usado, más específicamente, el codificador MQ del estándar JBIG2 [12]. Para cada paso, todos los símbolos son codificados aritméticamente, o no codificados (es decir, los símbolos binarios se emiten como bits sin procesar). Los pasos de limpieza emplean siempre la codificación aritmética. En el caso de los pasos de significación y refinamiento, existen dos posibilidades, dependiendo de si el modo llamado lazy está permitido. Si el modo lazy esta habilitado, sólo los pasos de significación y refinamiento para los cuatro bit-planes más significativos utilizan la codificación aritmética, mientras que los pasos restantes son sin codificar. En otro caso, todos los pasos de significación y refinamiento son codificados aritméticamente. El modo lazy permite complejidad computacional de la codificación del bit-plane para ser significativamente reducido, disminuyendo el número de los símbolos que deben ser codificados aritméticamente. Esto viene, por supuesto, en el costo de codificación eficiente.

Si los múltiples pasos de codificación consecutivos son codificados aritméticamente, pueden formar un solo codeword o cada paso puede formar un codeword separado. Estés casos son determinados por el modo de terminación en efecto. Dos modos de terminación son soportados: la terminación por paso y la terminación por segmento. En el primer caso, solamente el paso pasado de un segmento es terminado. En el segundo caso, se terminan todos los pasos de codificación. Terminar todos los pasos de codificación facilita el mejoramiento de la recuperación de error.

Puesto que se emplea la codificación aritmética basada en contexto, un medio para la selección del contexto es necesario. La selección del contexto es realizada examinando la información de estado de los 4 vecinos conectados o los 8 vecinos conectados de una muestra según lo mostrado en figura 3.2.10-2.

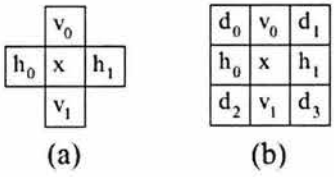


Figura 3.2.10-2 Plantilla para la selección de contexto. a) 4 vecinos conectados b) 8 vecinos conectados

Paso de significación

El primer paso de codificación para cada bit-plane es el paso de la significación. Este paso se utiliza para transportar la trascendencia y la información de signo de las muestras que aun no se han encontrado significativas y se predicen para llegar a ser significativas durante el proceso del actual bit-plane. Las muestras en el codeblocks se exploran en la orden demostrada previamente. Si una muestra todavía no se ha encontrado significativa, y se predice para llegar a ser significativa, la trascendencia de la muestra se codificada con un solo símbolo binario. Si la muestra también pasa a ser significativa, su signo es codificado usando un solo símbolo binario.

Si se está procesando el bit-plane más significativo, todas las muestras se pronostican para continuar siendo insignificantes. Si no, una muestra es pronosticada significativa si cualquier vecino de los 8 conectados ha sido encontrado significativo. Como consecuencia de esta política de predicción, los pasos de significación y refinamiento para el bit-plane más significativo son siempre vacíos (y no necesite ser codificado explícitamente).

Los símbolos generados durante el paso de significación pueden o no pueden ser codificados aritméticamente. Si se emplea la codificación aritmética, el símbolo binario que transporta la información de trascendencia se codifica usando uno de nueve contextos. El contexto particular usado se selecciona basándose en la trascendencia de los 8 vecinos conectados de la muestra y en la orientación de la subbanda con la cual la muestra es asociada (por ejemplo, LL, LH, HL, HH). En el caso que la codificación aritmética es utilizada, el signo de una muestra es codificado como la diferencia entre la signo actual y predicho. Si no, el signo es codificado directamente. La predicción de signo se realiza usando la trascendencia y la información de signo para los cuatro vecinos conectados.

Paso de refinamiento

El segundo paso de codificación para cada bit-plane es el paso de refinamiento. Este paso señala los bits subsecuentes después del bit más significativo de cada muestra. Las muestras del codeblock son exploradas usando el orden mostrado anteriormente. Si una muestra es encontrada significativa en un bit-plane anterior, el bit-plane más significativo siguiente de esa muestra se transmite usando un solo símbolo binario.

Como en el paso de significación, los símbolos del paso de refinamiento pueden ser o no codificados aritméticamente. Si se emplea la codificación aritmética, cada símbolo del refinamiento es codificado usando uno de tres contextos. El contexto particular empleado se selecciona basándose en si se está refinando la segunda posición del MSB y la trascendencia de los 8 vecinos conectados.

Paso de limpieza

El tercer paso de codificación para cada bit-plane es el paso de la limpieza. Este paso se utiliza para transportar la trascendencia e información de signo para esas muestras que todavía no se han encontrado significativas y se pronostican para continuar como insignificantes durante el procesamiento del bit-plane actual.

Conceptualmente, el paso de limpieza no es muy diferente al paso de significación. La diferencia clave es que el paso de limpieza transporta la información sobre las muestras que se predicen para llegar a ser significativas, más bien las que se pronostiquen para seguir siendo insignificantes. Algorítmicamente, sin embargo, hay una diferencia importante entre el paso de limpieza y el de significación. En el caso del paso de limpieza, las muestras se procesan a veces en grupos, más bien que individualmente como en el paso de significación.

Recordando el patrón de exploración de las muestras en un codeblock, el paso de limpieza es mejor explicado como mantenimiento de la exploración vertical. El paso de limpieza procesa simplemente cada una de las exploraciones verticales en orden, cada exploración vertical es procesada como sigue. En el caso en que la exploración vertical contiene cuatro muestras (es decir, exploración completa), la información trascendental es necesaria para todas las muestras, y todas las muestras se pronostican para seguir siendo insignificantes, un modo especial, llamado modo de agregación, es presentado. En este modo, el número de muestras insignificantes al frente en la exploración vertical se codifican. Entonces, las muestras cuya información trascendental es transportada por agregación son saltadas, y el procesamiento continúa con las muestras restantes de la exploración vertical exactamente como se hace en el paso de significación.

Cuando el modo de agregación es introducido, las cuatro muestras de la exploración vertical se examinan. Si las cuatro muestras son insignificantes, un símbolo de agregación todo insignificante es codificado, y el proceso de la exploración vertical es completo. Si no, un símbolo de agregación algo significativo es codificado, y dos símbolos binarios entonces son utilizados para codificar el número de muestras insignificantes al frente en la exploración vertical.

Los símbolos generados durante el paso de limpieza siempre se codifican aritméticamente. En el modo de agregación, se codifica el símbolo de agregación usando un solo contexto, y dos símbolos run length se codifica usando un solo contexto con una distribución de probabilidad uniforme fija. Cuando el modo de agregación no se emplea, la codificación de trascendencia y de signo funciona como en el caso del paso de significación.

3.b.11 Paquetes y capas

La información del paso de codificación se empaqueta en unidades de datos llamadas paquetes, en un proceso designado paquetización. Los paquetes resultantes entonces son la salida del flujo de código (codestream) final.

El proceso del paquetización impone una organización particular sobre los datos del paso de codificación en la salida del codestream. Esta organización facilita muchas de las características deseadas del codec incluyendo el índice de escalabilidad y la recuperación progresiva por fidelidad o resolución. Un paquete no es nada más una colección de datos del paso de codificación. Como se muestra en la figura 3.b.11-1, cada paquete incluye dos partes: una cabecera y un cuerpo. La cabecera indica qué pasos de codificación se incluyen en el paquete, mientras que el cuerpo contiene los datos por sí mismos del paso de codificación. Aunque, en el diagrama, la cabecera es seguida inmediatamente por el cuerpo, ésto es solamente conceptual. En la codestream, la cabecera y el cuerpo pueden aparecer juntos o por separado, dependiendo de las opciones de codificación en efecto.

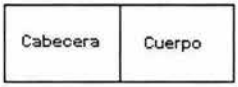


Figura 3.b.11-1 Estructura del paquete

El índice de escalabilidad se alcanza con capas. Los datos codificados de cada tile se organiza en L capas, numeradas a partir de la 0 a L-1, donde L≥1. Cada paso de codificación es asignado a uno de las L capas o se desecha. Los pasos de codificación que contienen los datos más importantes se incluyen en las capas más bajas, mientras que los pasos de la codificación asociados a detalles más finos se incluyen en capas más altas. En el caso de compresión con pérdidas, algunos pasos de codificación pueden ser desechados (es decir, no incluidos en alguna capa) en los cuales el control de índice de caso debe decidir que pasos incluye en el codestream final. En el caso sin pérdidas, todos los pasos de codificación deben ser incluidos. Si se emplean las capas múltiples (es decir, L > 1), el índice de control debe decidir en qué capa debe incluir cada paso de codificación.

Hay que recordar que cada paso de codificación se asocia a una componente, a un nivel de la resolución, a un subbanda, y a un codeblock. En la codificación, un paquete se genera para cada componente, nivel de la resolución, capa, y recinto cuadrutuple (figura 3.b.11-2). Un paquete puede ser vacío. Los paquetes vacíos son a veces necesarios puesto que un paquete se debe generar para cada combinación de componente-nivel-capa-recinto incluso si el paquete resultante no transporta ninguna nueva información.

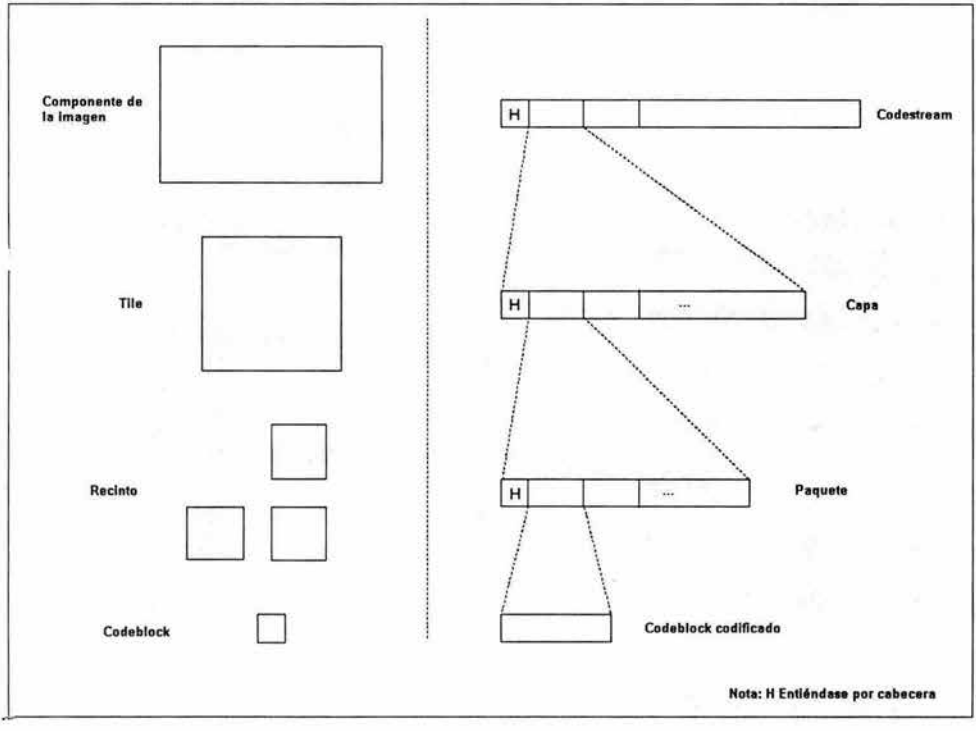


Figura 3.b.11-2 Correspondencia conceptual entre el espacio y las representaciones del bitstream

Un recinto es esencialmente la agrupación de codeblocks dentro de una subbanda. Puesto que los pasos de codificación de los datos de diversos recintos se codifican en paquetes separados, usar recintos más pequeños reduce la cantidad de datos contenidos en cada paquete. Si un paquete contiene menos datos, un probable bit de error resulta en menos pérdida de información. Así, usando tamaños pequeños de recintos se mejora la recuperación de error, mientras la eficiencia de codificación es degradada debido al incremento de número de paquetes.

Más de un ordenamiento de paquetes en el codestream es soportado. Tales ordenamientos se llaman progresiones. Hay cinco progresiones incorporadas definidas: ordenación de capa-resolución-componente-posición, ordenación resolución-capa-componente-posición, ordenación resolución-posición-componente-capa, ordenación posición-componente-resolución-capa, y ordenación componente-posición-resolución-capa. El orden de clase para los paquetes es dado por el nombre de ordenación, donde la posición se refiere al número del recinto, y las claves de clasificación se enumeran de la más significativa al menos significativo. Por ejemplo, en el caso de primera ordenación dada arriba, los paquetes son ordenados primero por capas, en segundo lugar por resolución, tercero por componente, y por último por el recinto. Esto corresponde a una recuperación progresiva por el panorama de fidelidad. El segundo ordenamiento arriba es asociado con la recuperación progresiva por resolución. Los tres ordenamientos restantes son algo más esotéricos. Es también posible especificar progresiones adicionales definidas por el usuario a expensas del incremento de codificación.

3.b.12 Codificación de la cabecera del paquete

La cabecera del paquete correspondiente a una componente, nivel de resolución, capa, y a un recinto en particular, se codifica como sigue. Primero, un solo símbolo binario se codifica para indicar si algunos datos del paso de codificación se incluyen en el paquete (es decir, si el paquete es no vacío). Si el paquete es vacío, no se requiere la transformación posterior y el algoritmo termina. Si no, se procede a examinar cada subbanda en el nivel de resolución en un orden fijo. Para cada subbanda, se inspeccionan los codeblock que pertenecen al recinto de interés en el orden de exploración de trama según lo mostrado en figura 3.b.12-1. Para procesar un solo codeblock, se comienza determinando si algunos nuevos datos del paso de codificación se van a incluir. Si todavía no se ha incluido ningún dato del paso de codificación para este codeblock, la información de inclusión se transporta por medio de un procedimiento de codificación basado en árbol cuadrienal (quadtree). Si no, un bit no procesado se emite para indicar la presencia o la ausencia de nuevos datos del paso de codificación para el codeblock. Si no hay nuevos pasos de codificación incluidos, se procede al proceso del siguiente codeblock en el recinto. Asumiendo que nuevos datos del paso de codificación están para ser incluidos, se continúa con el procesamiento del actual codeblock. Si esta es la primera vez que los datos del paso de codificación se han incluido para codeblock, se codifica el número de bit-planes insignificantes principales para el codeblock usando un algoritmo de codificación basado en árbol cuadrienal. Entonces, el número de nuevos pasos de codificación, y la longitud de los datos asociados a estos pasos se codifica.

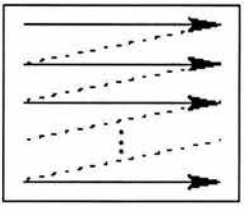


Figura 3.b.12-1 Orden de exploración del codeblock en un recinto

3.b.13 Codificación del cuerpo del paquete.

El algoritmo usado para codificar el cuerpo del paquete es relativamente simple. Los codeblocks se examinan en el mismo orden que en el caso de la cabecera del paquete. Si algunos nuevos pasos fueron especificados en la cabecera correspondiente del paquete, los datos para estos pasos de codificación se concatenan al cuerpo del paquete.

3.c Control de índice

En el codificador, el control de índice se alcanza a través de dos mecanismos distintos: la elección del tamaño del paso del cuantizador, y la selección del subconjunto de pasos de codificación incluidos en el codestream. El estándar no especifica cómo estos mecanismos deben ser empleados, y es posible utilizar un mecanismo exclusivamente o ambos juntos.

Cuando se emplea el primer mecanismo, los tamaños de paso del cuantizador se ajustan para el control de índice. Como se aumentan los tamaños de paso, el índice disminuye, con el costo de mayor distorsión. Aunque este mecanismo de control de índice es conceptual simple, tiene una desventaja potencial. Cada vez que se cambian los tamaños de paso del cuantizador, los índices del cuantizador cambian, y la codificación se debe realizar otra vez. Puesto que la codificación requiere una cantidad considerable de cómputo, esta aproximación para el índice de control puede no ser práctica

Cuando se utiliza el segundo mecanismo, el codificador puede elegir, desechar pasos de codificación para el índice de control. El codificador sabe la contribución que cada paso de codificación hace para el índice, y también puede calcular la reducción de la distorsión asociada a cada paso de codificación. Usando esta información, el codificador puede entonces incluir los pasos de codificación en orden de la reducción de distorsión que disminuye por índice de unidad hasta que se haya agotado el presupuesto del bit. Esta aproximación es muy flexible en métricas diferentes de distorsión y pueden ser fácilmente acomodadas. Para un tratamiento más detallado del control de índice [3] y [5].

3.d Codificación de una región de interés

El codec permite que diversas regiones de una imagen sean codificadas con diferente fidelidad. Esta característica se conoce como codificación de la región de interés (ROI). La funcionalidad del ROI es importante donde ciertas partes de la imagen son de mayor importancia que otras. En tal caso, estas regiones necesitan ser codificadas en calidad más de alta que el fondo. Durante la transmisión de la imagen, estas regiones necesitan ser transmitidas primero o en una prioridad más alta, como por ejemplo en el caso de la transmisión progresiva.

El esquema de codificación del ROI en la parte I del estándar se basa en el método llamado MAXSHIFT de Christopoulos et al. [13]-[15]. El método de MAXSHIFT es una extensión del método de codificación basado en escalamiento del ROI general [16].

Cuando una imagen se sintetiza de sus coeficientes de transformada, cada coeficiente contribuye solamente a una región específica en la reconstrucción. Así, una forma para codificar una ROI con mayor fidelidad que el resto de la imagen sería identificar los coeficientes que contribuyen a la ROI, y después codificar algunos o todos de estos coeficientes con mayor precisión que los otros. Esto es, de hecho, la premisa básica detrás de la técnica de la codificación de la ROI empleada en el codec JPEG-2000.

Cuando una imagen es codificada con una ROI, algunos de los coeficientes de la transformada son identificados más importantes que otros. Los coeficientes de mayor importancia se refieren como coeficientes de la ROI, mientras que los coeficientes restantes se conocen como coeficientes del fondo. Note que hay una correspondencia uno a uno entre los coeficientes de la transformada y los índices del cuantizador, por lo cual se definen índices del cuantizador para la ROI y los coeficientes del fondo, como índices del cuantizador de la ROI y del fondo, respectivamente.

La funcionalidad de la codificación de la ROI afecta el proceso de codificación. En el codificador, antes de que los índices del cuantizador para las diversas subbandas sean codificadas por el bit-plane, los índices del cuantizador de la ROI son escalados hacia arriba por una potencia de dos. Este escalamiento se realiza de tal manera que se asegura de que todos los bits de los índices del cuantizador de la ROI se hallen en planos de bits más significativos que los bits potencialmente distintos a cero de los índices del cuantizador del fondo. Por consiguiente, toda la información sobre los índices del cuantizador de la ROI serán señalizados antes de la información de los índices de la ROI de fondo. De esta manera, la ROI se puede reconstruir a una fidelidad más alta que el fondo.

Antes de que los índices del cuantizador sean codificados en el bit-plane, el codificador examina los índices del cuantizador del fondo para todas las subbandas, buscando el índice con la magnitud más grande. Suponga que este índice tiene su bit más significativo en la posición del bit N-1. Todos los índices de la ROI entonces son cambiados por N bits a la izquierda, y la codificación bit-plane procede como en el caso de una no ROI. El valor N del cambio de la ROI se incluye en el codestream.

Puesto que los bits-planes con la información perteneciente a la ROI son completamente separados de ésos que pertenecen al fondo, el número de bits-planes para la ROI y para el fondo puede ser elegido independientemente. Esto da la posibilidad de elegir diversos bits rates para la ROI y para el fondo. Para hacer esto, es suficiente con descartar los bits–planes menos significativos de la ROI y del fondo.

Durante la decodificación, cualquier índice del cuantizador con bits distintos a cero que se hallen en el bit plane N o arriba, se puede deducir que pertenece al sistema de la ROI. Después de que los índices reconstruidos del cuantizador se obtienen del proceso de decodificación del bit-plane, todos los índices en el conjunto de la ROI son reducidos por un cambio a la derecha de N bits. Esto deshace el efecto de escalamiento en el lado del codificador.

El sistema de la ROI se puede elegir para corresponder a los coeficientes de transformación que afectan una región particular en una imagen o un subconjunto de ésos que afectan la región. Esta técnica de codificación de la ROI tiene un número de características deseables. Primero, el ROI puede tener cualquier forma arbitraria y ser disjunta. En segundo lugar, no hay necesidad de señalar explícitamente el sistema de la ROI, puesto que puede ser deducida por el decodificador por el valor del cambio de la ROI y de la magnitud de los índices del cuantizador.

a) Imagen a 0.2b/p

b) Imagen a 0.5 b/p

c) Imagen a 1b/p

d) Imagen a 2 b/p

Figura 3.d-1 Codificación de una región de interés

3.e Code Stream

Para especificar la representación codificada de una imagen, dos diferentes niveles de sintaxis son empleados por el codec. El nivel más bajo de sintaxis se asocia con el flujo de código (codestream). El codestream es esencialmente una secuencia de registros etiquetados y sus datos de acompañamiento.

El bloque de construcción básica del codestream es el segmento del marcador. Según lo demostrado en figura 2.16, un segmento del marcador abarca tres campos: el tipo, la longitud, y los campos de los parámetros. El campo (o marcador) de tipo identifica la clase particular de segmento del marcador. El campo de longitud especifica el número de bytes en el segmento del marcador. El campo de parámetros proporciona información adicional específica del tipo de marcador. No todos los tipos de segmentos del marcador tienen campos de longitud y de parámetros. La presencia (o ausencia) de estos campos es determinada por el tipo de segmento del marcador. Cada tipo de segmento de marcador señala su propia clase particular de información.

Tipo	Longitud (si requiere)	Parámetros (si requiere)
16 bits	16 bits	Long. variable

Figura 3.e-1 Estructura del codestream

El codestream consiste en una cabecera principal, seguido por la cabecera y el cuerpo del tile, seguidos por un acoplado principal. Una lista de algunos de los segmentos del marcador más importantes es dada en la tabla 2.2. Los parámetros especificados en segmentos del marcador en la cabecera sirven como defectos para el codestream entero. Estos ajustes de defecto, sin embargo, se pueden eliminar para un tile particular especificando nuevos valores en un segmento del marcador en la cabecera del tile.

Tipo	Descripción
Inicio del codestream (SOC)	Señaliza el inicio del codestream. Primer segmento de marcador en el codestream.

Fin del codestream (EOC)	Señaliza el fin del codestream. Último segmento de marcador en el codestream.
Inicio del tile (SOT)	Indica el inicio de la cabecera de un tile. Primer segmento de marcador en la cabecera del tile.
Inicio de datos (SOD)	Señala el final de la cabecera del tile. Último T en la cabecera del tile. El cuerpo del tile sigue inmediatamente después de este segmento de marcador.
Tamaño de imagen y tile (SIZ)	Transmite las características básicas de la imagen y los parámetros de tiling. Segundo segmento del marcador en el codestream.
Estilo de codificación por defecto (COD)	Especifica los parámetros de codificación
Componente de estilo de codificación (COC)	Especifica un conjunto de parámetros de codificación para una sola componente.
Cuantización por defecto (QCD)	Especifica los parámetros de cuantización.
Componente de cuantización (QCC)	Especifica los parámetros de cuantización para una sola componente.
Región de interés	Especifica los parámetros de la codificación de la región de interés.

Tabla 3.e-1 Tipos de segmentos de marcador

Todos los segmentos del marcador, cabeceras del paquete, y cuerpos del paquete son múltiplo de 8 bits de longitud. Por consiguiente, se alinean todos los marcadores en bytes, y el codestream por sí mismo es siempre un número entero de bytes.

3.f Formato de archivo

Un codestream proporciona solamente la información más básica requerida para decodificar una imagen. Mientras que en algunas aplicaciones simples esta información es suficiente, en otras aplicaciones datos adicionales se requiere. Para permitir que datos adicionales sean especificados, un nivel adicional de sintaxis es empleado por el codec. Este nivel de sintaxis se refiere como formato de archivo. El formato de archivo se utiliza para transportar datos codificados e información auxiliar de la imagen. Aunque este formato de archivo es opcional, éste indudablemente será utilizado extensivamente por muchas aplicaciones, particularmente en aplicaciones de software basadas en computadoras.

El bloque de construcción básico del formato de archivo es referido como una caja. Según lo mostrado en figura 3.f-1, la caja abarca cuatro campos. El campo LBox especifica la longitud de la caja en bytes. El campo TBox indica el tipo de caja (es decir, la naturaleza de la información contenida en la caja). El campo XLBox es un indicador de longitud extendida que proporciona un mecanismo para especificar la longitud de una caja cuyo tamaño sea demasiado grande para ser codificado solamente en el campo de longitud. El campo DBox contiene datos específicos de un tipo de caja particular.

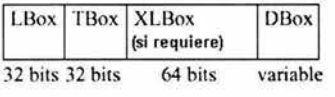


Figura 3.f-1 Estructura de la caja

Un archivo es una secuencia de cajas. Puesto que ciertos tipos de cajas se definen para contener otras, hay una estructura jerárquica natural en un archivo. La estructura general de archivo se demuestra en figura 3.f-2. La caja de firma siempre es la primera, proporcionando una indicación de que el bitstream está ajustado a formato correctamente. La caja de perfil está siempre en segundo lugar, e indicando la versión del formato de archivo con el cual el bitstream se conforma. La caja de la cabecera contiene simplemente un número de otras cajas. La caja de cabecera de la imagen especifica varias características básicas de la imagen (incluyendo el tamaño de la imagen, el número de componentes, etc.). La caja de especificación de color identifica el espacio de color de los datos de la imagen (para los propósitos de exhibición) e indica cual mapa de componentes para el tipo de información espectral (es decir, la correspondencia entre los componentes y los planos de color/opacidad). Cada archivo debe contener por lo menos una caja contigua del codestream. Cada caja contigua de codestream contiene un codestream como datos. En esta manera, los datos codificados de la imagen se encajan en un archivo. Además de los tipos de cajas discutidos, hay también tipos de caja para especificar la resolución de captura y exhibición de una imagen, la información de la gama de colores, la información de propiedad intelectual, y datos específicos de aplicación/vendedor.

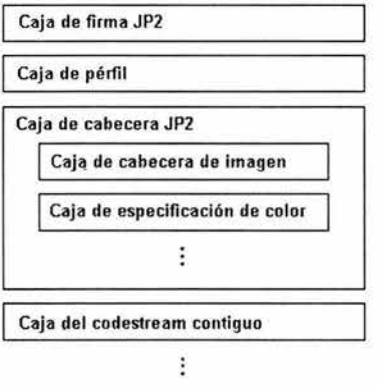


Figura 3.f-2 Estructura general del formato de archivo

Aunque alguna información almacenada en el nivel de formato de archivo es redundante, esta redundancia permite la manipulación trivial de archivos sin ningún conocimiento de la sintaxis del codestream. La extensión de nombre de archivo jp2 es utilizada para identificar los archivos que contienen datos en el formato de archivo JP2 (JPEG2000).

3.g Escalabilidad

La codificación escalable de imágenes fijas significa la capacidad de alcanzar codificaciones de más de una calidad y/o resolución simultáneamente. La codificación escalable de imágenes implica la generar de una representación codificada (bitstream) en una manera que facilite la derivación de imágenes de más de una calidad y/o de resolución por decodificación escalable. La escalabilidad del bitstream es una propiedad del bitstream que permite decodificar subconjuntos apropiados del bitstream para generar imágenes completas de calidad y/o resolución conmensuradas con la proporción del bitstream decodificado. Los decodificadores de diversas complejidades (de funcionamiento bajo a alto) pueden coexistir para un bitstream escalable. Mientras que los decodificadores de bajo funcionamiento pueden decodificar solamente porciones pequeñas del bitstream generando una calidad básica, los

decodificadores del alto rendimiento pueden decodificar mucho más y producir perceptiblemente calidad más alta.

Los tipos más importantes de escalabilidad son la escalabilidad SNR y la escalabilidad espacial o de resolución. El sistema de compresión JPEG2000 soporta escalabilidad. Una ventaja clave de la compresión escalable es que el bit rate deseado o la resolución de reconstrucción no necesita ser conocido en la compresión. Una ventaja adicional de la escalabilidad es su capacidad de proporcionar resistencia a los errores de la transmisión, pues los datos más importantes de la capa más baja se pueden enviar sobre el canal con un mejor funcionamiento de error, mientras que los datos menos críticos de la capa de extensión se pueden enviar sobre el canal con un pobre funcionamiento de error. Ambos tipos de escalabilidad son muy importantes para aplicaciones de acceso a Internet y bases de datos y el escalamiento del ancho de banda para una distribución robusta.

Escalabilidad SNR

La escalabilidad SNR es propuesta para sistemas con la característica común primaria de que un mínimo de dos capas de calidad de imagen son necesarias. La escalabilidad SNR implica generar por lo menos dos capas de la imagen de la misma resolución espacial, pero a diversas calidades, de una sola fuente de imagen. La capa más baja es codificada por sí misma para proporcionar una calidad básica de imagen y las capas de extensión se codifican para realzar la capa más baja. Una capa de extensión, cuando es agregada de nuevo a la capa más baja, genera una reproducción de más alta calidad de la imagen de entrada. La figura 3.g-1 ilustra un ejemplo de escalabilidad SNR. La primera imagen es comprimida sin pérdidas y descomprimida a 0,125 b/p, 0,25 b/p, y 0,5 b/p.

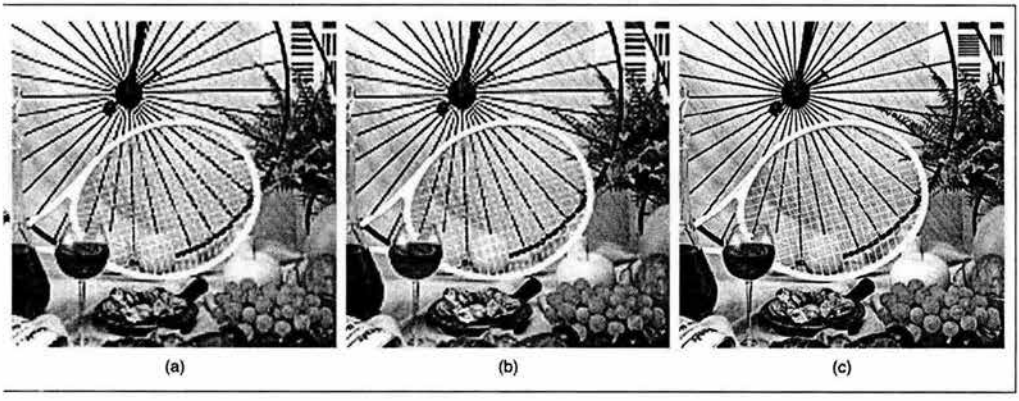


Fig 3.g-1 Ejemplo de escalabilidad SNR. Imagen comprimida a) 0.125b/p, b) 0.25b/p, c) 0.5b/p

Escalabilidad espacial

La escalabilidad espacial es propuesta para sistemas con la característica común primaria de que un mínimo de dos capas de resolución espacial son necesarias. La escalabilidad espacial implica generar por lo menos dos capas espaciales de resolución de una sola fuente tales que la capa más baja es codificada por sí misma para proporcionar una resolución espacial básica y la capa del extensión emplea la capa más baja interpolada espacialmente y lleva la resolución espacial completa de la fuente de la imagen de entrada. La figura 3.7-2 muestra un ejemplo de tres niveles de decodificación progresiva por

resolución de una imagen. La escalabilidad espacial es útil para el acceso rápido a bases de datos así como la distribución a diversas resoluciones a terminales con diversas capacidades en términos de exhibición y ancho de banda. JPEG2000 soporta también una combinación de escalabilidad espacial y SNR.

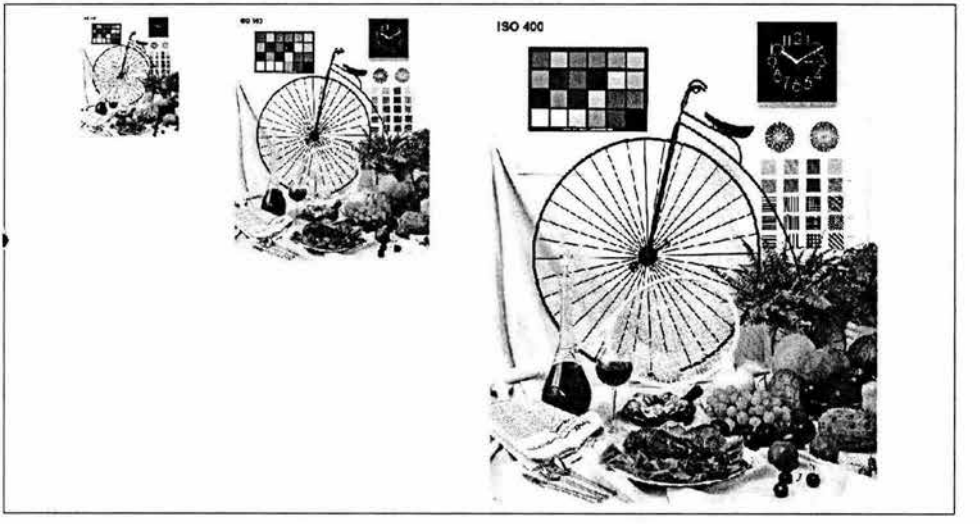


Figura 3.g-2 Ejemplo de decodificación progresiva por resolución

El bitstream contiene los marcadores que identifican el tipo de progresión del bitstream. Los datos almacenados en paquetes son idénticos sin importar el tipo de escalabilidad usado. Por lo tanto es trivial cambiar el tipo de la progresión o extraer cualquier dato requerido del bitstream. Para cambiar la progresión de SNR a progresión por resolución, un programa de análisis puede leer los marcadores, cambiar el tipo de progresión en los marcadores, y después escribir los marcadores nuevos en el nuevo orden. De este modo, la transcodificación rápida del bitstream se puede alcanzar por un servidor o gateway, sin requerir el uso de la decodificación y re-codificación de la imagen.

3.h Recuperación de error (error resilience)

La recuperación de error es una de las características más deseadas en aplicaciones sobre móviles e Internet. JPEG2000 utiliza un codificador de longitud variable (codificador aritmético) para comprimir los coeficientes wavelet cuantizados. Como es sabido, la codificación de longitud variable es propensa a los errores de canal y transmisión. Un bit erróneo resulta en pérdida de sincronización en el decodificador de entropía y la imagen reconstruida puede ser seriamente dañada. Para mejorar el funcionamiento de transmisión de imágenes comprimidas sobre canales propensos a error, se incluyen en el estándar la recuperación de error (error resilience) y utilerías. Las utilerías de recuperación de error tratan con los errores de canal usando las siguientes aproximaciones: partición y resincronización de datos, detección y ocultación de error, y transmisión con calidad de servicio (QoS por sus siglas en inglés) basada en prioridad. La recuperación de error se alcanza en el nivel de codificación de entropía y en el nivel de paquete. La tabla 3.h-1 resume varias formas de lograr esto.

Tipo de utilería	Nombre
Nivel de codificación de entropía	- Codeblocks - terminación del codificador aritmético para cada paso - reinicio de contextos después de cada paso de codificación. - selección del puente de codificación aritmética - símbolos de segmentación
Nivel de paquete	- formato corto de paquetes - paquetes con marcador de sincronización

Tabla 3.h-1 Utilerías para la recuperación de error

La codificación de entropía de los coeficientes cuantizados se realiza dentro de los codeblocks. Ya que la codificación y decodificación de los codeblocks son procesos independientes, los bits erróneos del bitstream de un codeblock serán restrictos dentro de ese codeblock. Para aumentar recuperación de error, la terminación del codificador aritmético se permite después de cada paso de codificación y los contextos se pueden reajustar después de cada paso de codificación. Esto permite que el decodificador aritmético continúe el proceso de decodificación incluso si ha ocurrido un error.

El modo de codificación lazy es también útil para la recuperación de error. Este se relaciona con puente de codificación aritmética opcional en el cual los bits son alimentados como bits sin procesar en el bitstream sin la codificación aritmética. Esto previene los tipos de propagación de error para los cuales la codificación de longitud variable es susceptible.

En el nivel de paquete, un paquete con un marcador de resincronización permite la partición y resincronización espacial. Esto se coloca delante de cada paquete en un tile con un número de serie que comienza en cero y se incrementa con cada paquete.

3.i Características visuales

El sistema visual humano desempeña un papel importante en la calidad percibida de la representación de imágenes comprimidas. Por lo que se utilizan modelos de sensibilidad de variación del sistema visual para frecuencias espaciales, como medida en la función de sensibilidad de contraste (CSF por sus siglas en inglés). Puesto que el peso del CSF es determinado por la frecuencia visual de los coeficientes de la transformada, habrá un peso del CSF por subbanda en la transformada wavelet. El diseño de los pesos CSF es una edición del codificador y depende de la condición de visión específica bajo la cual la imagen decodificada debe ser vista.

El estándar del JPEG2000 soporta dos tipos de peso (weighting) de frecuencia visual. El peso visual fijo (FVW por sus siglas en inglés) y la codificación progresiva visual o peso progresivo visual (VPW por sus siglas en inglés). En el FVW, solamente un conjunto de pesos CSF se eligen y se aplican de acuerdo con las condiciones de visión. En el VPW, diversos conjuntos de pesos CSF se utilizan en varias etapas de la codificación encajada (embedded). Esto es porque durante un estado de transmisión progresiva, la imagen es vista a varias distancias. Por ejemplo, en bitrates bajos, la imagen es vista a distancias relativamente grandes, y mientras más y más bits son recibidos se mejora la calidad y la distancia de visión. FVW se puede considerar como caso especial de VPW.

Referencias

[]

[]

[]

[]

[]

[]

[]

4. FIRMA DIGITAL EN IMÁGENES CON WATERMARKING

4.a Implementación de una marca de agua digital por medio de las aplicaciones Watermarking.exe y mrkrdll.dll

Watermarking.exe y mrkrdll.dll son dos aplicaciones, o mejor aún, una aplicación y una biblioteca enlazada dinámicamente cuyo objetivo es demostrar de manera práctica la implementación de un sistema de marcas de agua invisibles en imágenes digitales, con dos objetivos principalmente: que los conceptos expuestos en este trabajo sean "aterrizados" de manera que sea más fácil comprenderlos por el usuario promedio de cualquier tipo de herramienta que genere información digital, en este caso aquellas que cómo escáneres o cámaras digitales, produzcan imágenes; y en segundo lugar, servir como una introducción al desarrollo de aplicaciones más complejas que usen la tecnología de las marcas de agua para la protección de imágenes digitales, campo que aún es muy joven incluso a nivel internacional, por lo cual existen aún muchas mejoras y actualizaciones por hacer. La última palabra en cuanto al desarrollo de esquemas más y más eficientes de marcas de agua aún no está dicha.

4.a.1 Watermarking

Watermarking.exe es la aplicación final en la cual se implementa el sistema de marcas de agua digitales. Por simplicidad está construido en VB 6.0 y funciona sobre la API de Windows, por lo cual por el momento no es una aplicación portable, excepto para las plataformas MS Windows 98, ME, 2000 y XP. Es importante hacer notar que ninguna funcionalidad es implementada por Watermarking.exe, esta aplicación es sencillamente un "cascarón", la GUI o *Graphical User Interface* que hace transparente al usuario el proceso del marcado. Toda la funcionalidad es implementada por la biblioteca mrkrdll.dll como se explica en el apartado posterior.

Watermarking se compone de una ventana principal, en la cual se incluye la barra de menús, una ventana de información acerca del archivo BMP y por último, de una o más ventanas en las cuales se despliegan la imágenes original y marcada.

La barra de menús se divide en tres submenús: Archivo, Atacar y Detectar, los cuales a su vez tienen las siguientes funcionalidades:

Archivo

- Abrir. Abrir un archivo de mapa de bits, extensión BMP
- Marcar. Insertar una marca de agua digital en la imagen de mapa de bits
- Salir. Salir de la aplicación

Atacar

- Geométrico. Realiza un ataque geométrico a la imagen
- Modificación de pixeles. Modifica directamente el valor del pixel dado por el usuario
- Filtrado. Ataca a la imagen por medio de un el filtrado pasa altas, pasa bajas y de mediana

Detectar

Realiza una correlación entre las imágenes marcada y atacada para determinar si la marca de agua está presente en la segunda.

A continuación, un *snapshot* de la ventana principal de la aplicación:

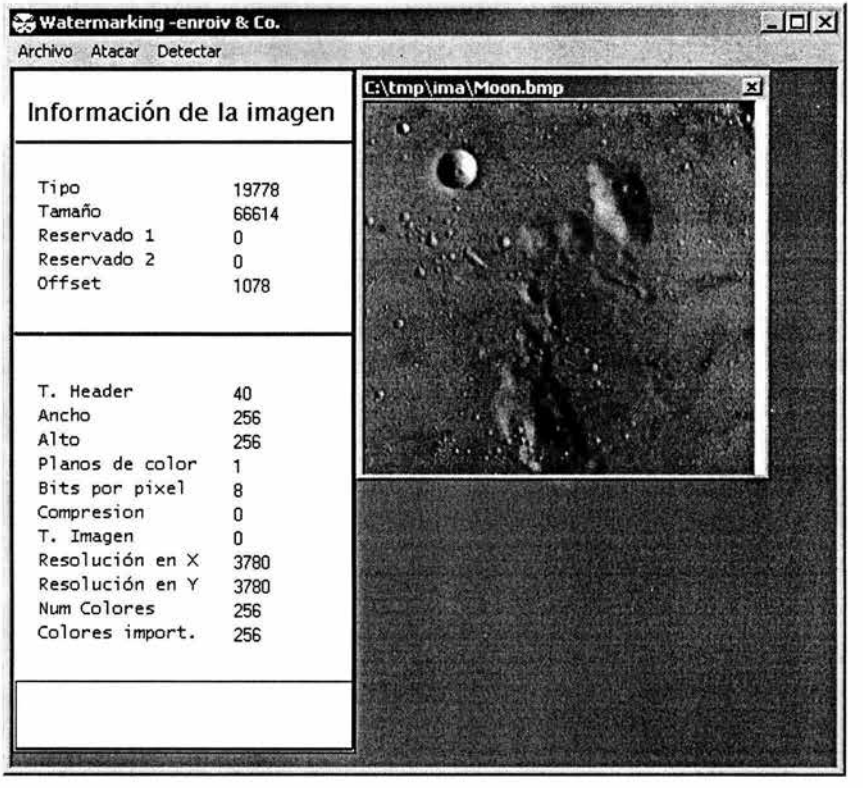


Fig 4.a.1-1 Vista de la ventana principal de Watermarking.exe.

Como se observa claramente, en la parte superior de la ventana principal se halla la barra de menús de usuario, en la parte izquierda se encuentra la ventana de información, en la cual se proporcionan las características de la imagen, definidas en la cabecera BMP, entre las cuales está tipo de archivo, que siempre es 19778 para imágenes de mapa de bits, el tamaño en bytes del archivo, el tamaño de su cabecera, que siempre es de 40, la compresión y el bitdepth entre otras.

Finalmente, a la derecha de la ventana de información se observa la ventana de imagen, en donde se muestra la imagen de trabajo, esto es, la imagen a la cual se insertará la marca de agua o la cual se probará para detectar la existencia de tal marca.

Intencionalmente hemos tratado de mantener la aplicación al nivel más simple posible, por lo cual no posee menús obscuros o sintaxis rebuscadas: lo único que el usuario debe proveer a la aplicación es la llave de inserción y el tamaño del tile que se usará para el marcado de la imagen, todo de un modo gráfico y sin que sea necesario el conocimiento de ningún tipo de conceptos del procesamiento digital de imágenes, transformadas DCT o wavelets ni ninguna terminología matemática de tipo alguno.

El proceso de inserción de la marca es también muy simple y se detalla a continuación:

- El usuario inicia la aplicación Watermarking.exe
- En el menú archivo selecciona la opción Abrir, con lo que aparece un cuadro de diálogo que le ayuda a escoger el archivo con el que se va a trabajar
- Después de haber escogido el archivo, éste se carga en la ventana de imagen
- Nuevamente en el menú Archivo, se selecciona la opción Marcar
- Aparece una ventana en la que se le pide al usuario el ancho del tile de inserción, el ancho del tile de inserción y la llave de inserción que desea usar
- El usuario selecciónalos parámetros adecuados y presiona el botón Aceptar

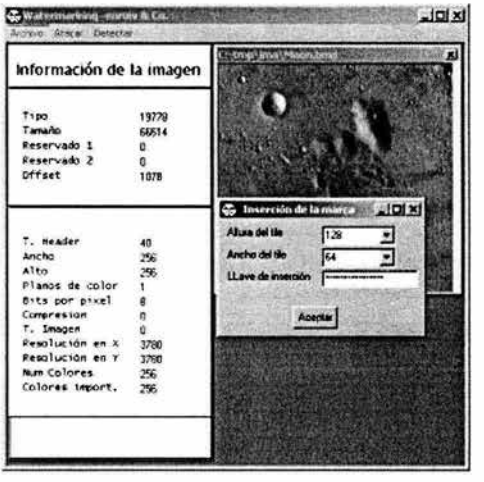


Fig 4.a.1-2 Proceso de inserción de la marca de agua.

Tras esto y si la imagen ha sido marcada exitosamente, aparece un mensaje en el que se indica al usuario la dirección en que se ha guardado la imagen con la marca de agua, la cual es:

.\marcadas\Wnombre-de-la-imagen.bmp

en donde .\ se refiere al directorio actual y nombre-de-la-imagen es el nombre de la imagen original. A éste se la a agregado el prefijo "W", que indica que a dicha imagen se le ha insertado un marca de agua o *Watermark.*

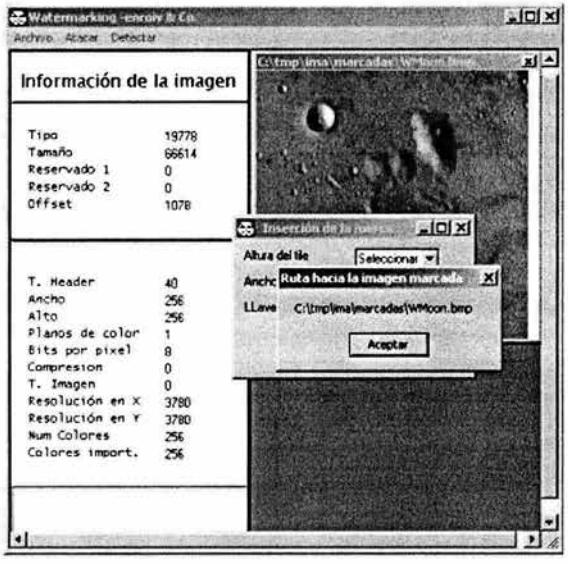


Fig 4.a.1-3 Proceso de inserción de la marca de agua (continuación).

Para realizar un ataque a una imagen digital el proceso es muy parecido y se describe a continuación:

- El usuario inicia la aplicación Watermarking.exe
- En el menú archivo selecciona la opción Abrir, con lo que aparece un cuadro de diálogo que le ayuda a escoger el archivo con el que se va a trabajar
- Después de haber escogido el archivo, éste se carga en la ventana de imagen
- En el menú Atacar, se selecciona la opción que indique el tipo de ataque que se desea realizar, el cual puede ser filtrado de mediana, pasa altas, pasa bajas, ataque geométrico o modificación de pixeles
- En caso de haber escogido un ataque por filtrado, aparece una ventana en la que se le pide al usuario el soporte del filtro a emplear, esto es, el tamaño de la ventana de convolución que habrá de recorrer la imagen

Tras esto y si la imagen ha sido atacada exitosamente, la imagen atacada se guarda en la siguiente dirección:

.\marcadas\atacadas\Wnombre-de-la-imagen.bmp, o bien

.\marcadas\Wnombre-de-la-imagen.bmp

dependiendo si la imagen atacada ha sido la imagen marcada o la imagen original respectivamente.

en donde .\ se refiere al directorio actual y nombre-de-la-imagen es el nombre de la imagen original. A éste se le ha agregado el prefijo "A", que indica que a dicha imagen se le ha realizado algún tipo de ataque.

Es importante mencionar que idealmente la imagen original debe permanecer intacta, por lo cual los ataques realizados a ésta en el módulo de ataques son sólo con fines experimentales y se realizan a una copia de ésta.

A continuación se muestra gráficamente el módulo de ataques, al cual se accede por medio del menú Atacar:

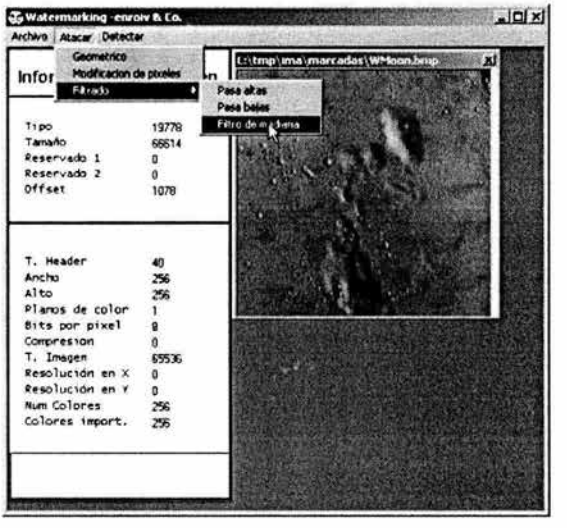


Fig 4.a.1-4 Módulo de ataques de Watermarking.exe.

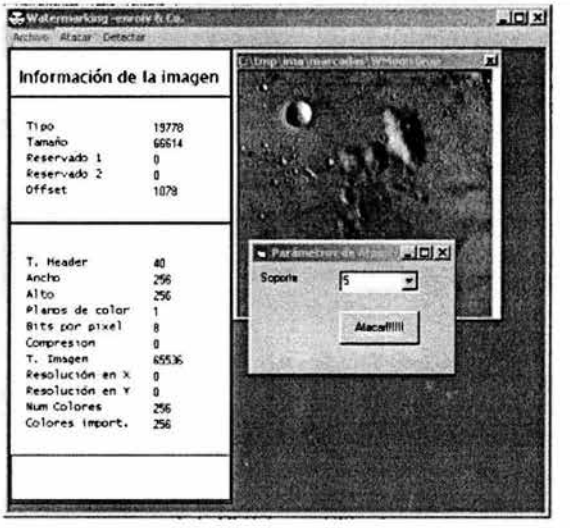


Fig 4.a.1-5 Módulo de ataques de Watermarking.exe (Continuación).

Para realizar la detección de una marca de agua sólo basta con presionar el submenú Detectar dentro del menú principal y después seleccionar dos archivos, el archivo marcado originalmente y algún otro archivo, o sea, el archivo atacado en el que se espera encontrar la marca, todo esto por medio de los cuadros de diálogo que se proveen en el sistema.

Debido a su simplicidad, se deja al usuario el uso del módulo de detección y no se mostrará en este apartado.

4.a.2 mrkrdll.dll

La biblioteca de enlace dinámico mrkrdll.dll es en realidad la encargada de realizar todo el proceso del marcado y se comunica con Watermarking.exe para crear la aplicación final.

Al igual que Watermarking.exe, está construida sobre la API de Windows, lo cual hace que por el momento no sea portable entre diferentes plataformas, a excepción de aquellas que soporten la API para MS Windows de 32 bits.

Al ser mrkrdll la encargada de realizar el procesamiento digital de la imagen, tiene un grado de complejidad mayor que el de la aplicación Watermarking.exe, y es por eso que se decidimos escribirla en C++, debido a que este lenguaje es uno de los más potentes y nos permite tener un control de la imagen a un nivel muy bajo, que es precisamente lo que necesitamos: ser capaces de modificar los valores de los pixeles directamente, así como realizar operaciones de I/O como la lectura y escritura de archivos de imagen.

Mrkrdll.dll se compone de seis diferentes módulos, cada uno encargado de proveer una funcionalidad diferente, los cuales al unirse dan como resultado la biblioteca de funciones de procesamiento digital de imágenes de la cual Watermarking.exe hace uso, siendo al mismo tiempo una interfaz entre el almacenamiento físico y lógico y el procesamiento de la imagen y el usuario final. Cinco de estos módulos son implementados por medio de objetos de usuario definidos con la ayuda de C++ y el último es el repositorio de funciones definidas para accesar a la funcionalidad de estos objetos. A continuación se ilustra cada uno de estos módulos.

BmpIm

BmpIm es el objeto principal de mrkrdll.dll y representa una abstracción de una imagen digital de mapa de bits, esto es, una imagen en formato BMP.

Antes de ahondar en el objeto BmpIm, repasemos el concepto de imagen digital: una imagen digital es sencillamente un conjunto de datos organizado de manera tal que representa el muestreo, cuantización y codificación de los valores de luminancia de una imagen continua para su almacenamiento y/o transmisión por medios digitales. En las plataformas Windows se necesita además información adicional para la representación de una imagen digital, la cual se guarda en una estructura llamada Cabecera BMP.

Cabecera BMP

Header 14 Bytes
Header de información 40 Bytes
Paleta (Opcional) ¿? Bytes
Imagen ¿? Bytes

Header:

- **Tipo de archivo (2 B)**. Siempre tiene el valor de 0x424D, el cual indica que es un archivo BMP.
- **Tamaño (4B)**. Indica el tamaño en Bytes del archivo
- **Bytes reservados 1 y 2 (2B c/u)**. No se utilizan
- **Offset(4B)**. Offset en Bytes al inicio de la imagen

Header de información:

- **Tamaño(4B)**. Tamaño en Bytes de la cabecera de información, este valor siempre es de 40 Bytes
- **Ancho, Alto(4B c/u)**. Dimensiones de la imagen, en pixeles
- **Planos(2B)**. Número de planos de color
- **Bits(2B)**. Número de bits por pixel, bitdepth
- **Compresión(4B)**. Tipo de compresión:

  0 – Sin compresión
  1 – Corrida de 8 bits
  2 – Corrida de 4 bits
  3 – Bitmap RGB con máscara

- **Tamaño de la imagen(4B)**. Tamaño en bytes de la imagen
- **Resolución en X, Resolución en Y (4B c/u)**. Número de pixeles por metro
- **Numero de colores(4B)**.
- **Colores importantes(4B)**.

Para los BMP de 24 bits no hay paleta y la imagen viene a continuación del header de información en orden BGR (azul, verde, rojo).

Para los BMP de color indexado, sigue una tabla de (infoheader.numcolores) colores. Los primeros 3 Bytes corresponden a componentes BGR y el último no es utilizado. Para escalas de grises de 8 bits, este índice de color generalmente será una rampa de niveles de gris. Entonces, Header.Offset = 14 + 40 + (4*infoheader.numcolores)

Con esta información, estamos listos para empezar a analizar la estructura de la clase BmpIm, la cual se muestra a continuación:

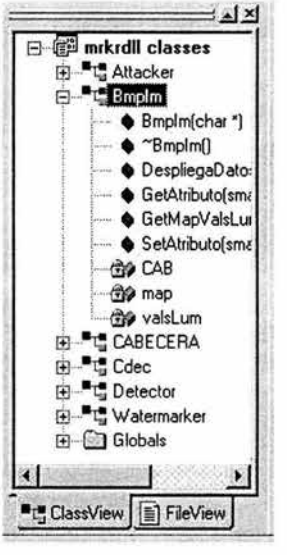


Fig. 4.a.2-1 La clase BmpIm

Entre los elementos que se observan están los métodos del objeto BmpIm, los cuales son BmpIm(char *), ~BmpIm(), DespliegaDatos(), GetAtributo(small), GetMapValsLum(small) y SetAtributo(small, unsigned long). Los dos primeros métodos no importan en este estudio pues sencillamente definen al constructor y destructor de la clase a nivel compilador. Los métodos que nos interesan son los últimos 4, pues son éstos los que nos permitirán accesar a los elementos de la imagen, tales como la información de su cabecera, con el método DespliegaDatos(), el cual sirve para desplegar las características de la imagen en la ventana de información de Watermarking.exe.

El método GetAtributo es uno de los más importantes puesto que nos permite conocer el valor de cierto atributo de la imagen BMP definido de antemano, como si bitdepth, su altura o su ancho en pixeles. Este es uno de los métodos del que más se echa mano durante todo el procesamiento digital de la imagen.

El método GetMapValsLum sencillamente regresa un apuntador hacia los valores de luminancia o el colormnap de la imagen BMP, según su argumento sea 0 ó 1 respectivamente. A pesar de no usarse en muchas ocasiones dentro del procesamiento, es un método muy importante puesto que es el encargado de recuperar la imagen y el colormap.

Por último, el método SetAtributo(small, unsigned long) sirve para modificar el valor de algún atributo de la imagen digital. No se empleó en absoluto durante el procesamiento pero pudiera ser útil cuando la aplicación crezca y adquiera nueva funcionalidad; es por eso que decidimos incluirlo.

Finalmente, el objeto BmpIm tiene 3 elementos, un colormap (map), un arreglo de los valores de luminancia de la imagen digital (ValsLum) y una cabecera BMP (CAB) tal como la que ya se ha definido anteriormente

Cdec

La clase Cdec es la abstracción de un codificador y es quien realiza gran parte de las operaciones de procesamiento digital de imágenes sobre la imagen de trabajo. Está íntimamanete ligada a la clase BmpIm y de hecho, es necesaria la existencia de un objeto de la clase BmpIM para poder instanciar un objeto Cdec, y a cada uno de éstos se le asignará un Cdec distinto. Esto es lógico pues para poder realizar operaciones de PDI es necesaria una imagen de trabajo y además es conveniente que cada imagen de trabajo a ser codificada tenga parámetros de codificación distintos. Incluso distintas instancias de la misma imagen BmpIm pueden codificarse usando objetos Cdec distintos.

La clase Cdec modela algunos de los parámetros de una imagen BMP como sus dimensiones físicas (altura y ancho en pixeles), su matriz o matrices de luminancias según se trate de una imagen en nivel de gris o RGB respectivamente, así como el número de componentes de color, que en esta fase preliminar de desarrollo pueden ser de una o 3 componentes. Actualmente no son soportadas imágenes con más de 3 componentes de color pero esto podría modificarse al continuar el desarrollo de la aplicación según convenga a los objetivos de la misma y/o a las necesidades del usuario promedio.

Además de modelar características de las imágenes BMP, la clase Cdec modela un par de parámetros específicos para el procesamiento digital de imágenes y la inserción de las marcas de agua digitales en conformidad con el estándar JPEG 2000: el *tiling* y el cambio a nivel de DC. El tiling es modelado con ayuda de un par de parámetros que nos ayudan a definir las dimensiones físicas de los tiles a emplear mientras que l cambio a nivel de DC es modelado por medio de un tercer parámetro que nos indica el nivel de DC a emplear para que el codec reciba una señal aproximadamente gaussiana con media de cero, según indica el estándar JPEG 2000.

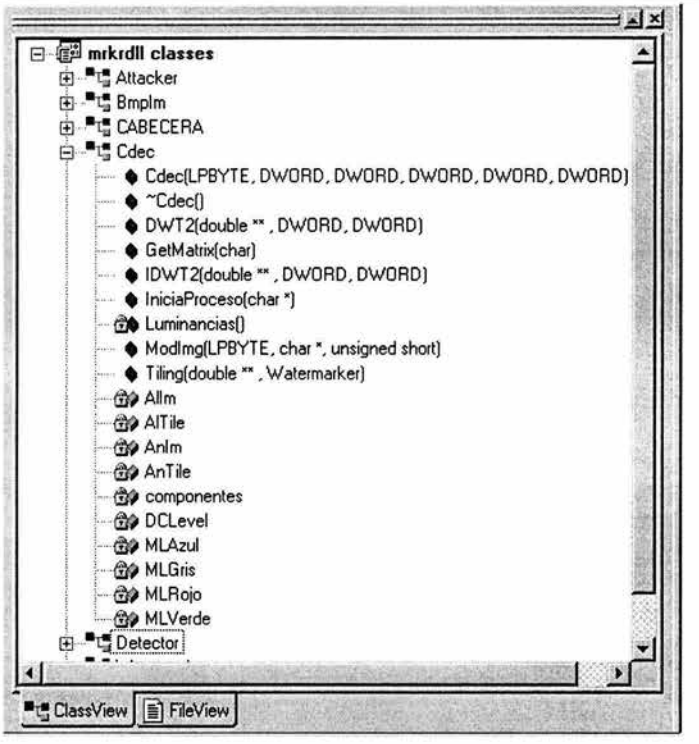


Fig. 4.a.2-2 La clase Cdec

En la figura anterior se pueden observar las propiedades de la clase Cdec empleadas para modelar las matrices de luminancias de la imagen BMP (MLGris, MLRojo, MLVerde y MLAzul), el cambio a nivel e DC (DCLevel), el número de componentes de color, las dimensiones del *tile* (AlTile, AnTile) y las dimensiones de la imagen (AlIm, AnIm). A continuación se describen los métodos de la clase Cdec, que son quienes implementan gran parte de la funcionalidad de esta aplicación y son llamados desde Watermarking.exe para operaciones de procesamiento digital de imágenes y E/S de archivos sobre la plataforma MS Windows.

- **Cdec.** Es el constructor de la clase Cdec, el cual toma como parámetros un arreglo lineal de los valores de luminancia de la imagen BMP, sus dimensiones físicas, su número de componentes de color y finalmente, las dimensiones físicas del tile a emplear para realizar el procesamiento, definidas por el usuario. Instancia un codificador con las características mencionadas anteriormente, además de calcular el nivel de DC para la imagen.

- **~Cdec.** No es de importancia para este estudio pues es sencillamente el destructor de la clase, a nivel del compilador.

- **DWT2.** Esta función se encarga de efectuar la transformada wavelet directa, discreta y bidimensional de la imagen o el tile que se le proporcione. Toma como parámetros dicha imagen o tile y sus dimensiones físicas, definidas por el usuario y regresa las imágenes de aproximación, y de detalle horizontal, vertical y diagonal de la transformada wavelet. Usa el algoritmo Daubechies-4, definido por la matemática belga Ingrid Daubechies.

- **GetMatrix.** La función GetMatrix sencillamente retorna la matriz de luminancias especificada. Recibe como argumento una letra que representa la matriz de luminancias con la cual nos interesa trabajar ('r' para la matriz de luminancias en rojo, 'v' para la matriz de luminancias en verde, 'a' para la matriz de luminancias en azul y 'g' para la matriz de luminancias en nivel de gris).

- **IDWT2.** Realiza la transformada wavelet inversa, discreta y bidimensional de la imagen o tile que se le proporcione. Al igual que la función DWT2, toma como parámetros un arreglo que representa la transformada wavelet de una imagen o tile y sus dimensiones físicas.

- **IniciaProceso.** Esta función se emplea para indicar al codec que inicie las operaciones de PDI, por medio de la función Tiling. Además de esto, instancia un objeto de la clase Watermarker, que se encarga de insertar la marca de agua en la imagen.

- **Luminancias.** Se emplea para regresar a la imagen antitransformada wavelet a su nivel de DC inicial.

- **ModImg.** Es la función encargada de escribir en disco duro la imagen procesada digitalmente, ya sea para insertar una marca de agua o una imagen que ha sido atacada. Recibe como parámetros un colormap (para imágenes en nivel de gris), la ruta de la imagen original y un parámetro que indica si la imagen a escribir ha sido marcada o atacada.

- **Tiling.** Tiling es una de las funciones más complejas de la clase Cdec, pues es la encargada de dividir a la imagen original en tiles de tamaño definido por el usuario, de formatear estos tiles para evitar que se solapen dentro de la imagen y de efectuar el cambio a nivel de DC de la imagen original. Además de lo anterior, es quien realiza las llamadas a las funciones DWT2 e IDWT2, funcionando como un "director de orquesta", encargada de controlar el flujo del procesamiento de la imagen. Recibe como parámetro una imagen en nivel de gris o una de las componentes de color de una imagen RGB y un objeto de la clase Watermarker, el cual se encargará de insertar la marca de agua en los tiles generados por esta función.

Watermarker

La clase Watermarker es la encargada de insertar la marca de agua digital en la imagen BMP. Es instanciada en la función IniciaProceso() de la clase Cdec, creando así un módulo capaz de insertar una marca de agua invisible en los coeficientes de la transformada wavelet de una imagen, ya sea en nivel de gris o en RGB.

Esta clase es realmente muy sencilla puesto que cuenta solamente con dos funciones y con dos propiedades. Ambas propiedades son en realidad dos representaciones distintas de la llave de inserción de la marca de agua, la cual se ha modelado con dos propiedades distintas exclusivamente para facilitar la interoperabilidad entre los compiladores de Microsoft Visual Basic y Microsoft Visual C++, por lo cual puede pensarse en ambas como una sola propiedad de la clase Watermarker: la llave de inserción de la marca de agua.

Dicha marca de agua se insertará en los coeficientes wavelet, como ya se ha mencionado, de la forma que se detalla a continuación.

Dentro de la funciónIniciaProceso() de la clase Cdec, se instancia un objeto de la clase Watermarker usando como argumento la llave de inserción especificada desde la aplicación Watermarking.exe; esta llave se convierte en una representación numérica de ella misma y se formatea como un arreglo de valores que representan el código ASCII de los valores de los caracteres que la conforman. En este momento estamos listos para insertar una marca de agua en un tile de la imagen original.

Posteriormente, en IniciaProceso() se llama a la función Tiling() de la misma clase Cdec, la cual indica al objeto de la clase Watermarker que inserte una marca de agua en los coeficientes wavelet del tile que ésta misma ha obtenido. Esta es la secuencia de inserción de la marca de agua:

1. Dentro de su función miembro IniciaProceso(), Cdec instancia un objeto de la clase Watermarker y llama a la función Cdec::Tiling().

2. Tiling() obtiene un tile a partir de la imagen original y realiza su transformación wavelet directa por medio de la función Cdec::DWT2(), la cual realiza una transformación de 1, 2 o tres niveles dependiendo del tamaño del tile.

3. Tiling() indica a la función Watermarker::Watermark() que inserte una marca en el tile transformado. Watermark() es capaz de determinar los niveles de transformación a los que se ha sometido el tile y dependiendo de estos inserta la marca de agua.

4. Tiling() indica a la función Cdec::IDWT2() que efectúa la transformada inversa del tile marcado.

Finalmente, se muestra la definición de los métodos de la clase Watermarker()

- **Watermarker.** Es el constructor de la clase Watermarker, el cual toma como parámetro una cadena que representa la llave de inserción de la marca de agua. Posteriormente convierte esta cadena a una representación entera de la misma, la cual se usará como semilla de un generador de números aleatorios que se multiplicará por los coeficientes de la transformada wavelet de la imagen y por un factor que modela el umbral JND para obtener la marca de agua final que se insertará empleando un algoritmo aditivo.

- **Watermark.** Se encarga de insertar la marca de agua en los coeficientes de la imagen transformada. Determina el nivel de transformación wavelet y después de esto inserta la marca en los coeficientes que satisfacen cierta condición de umbral.

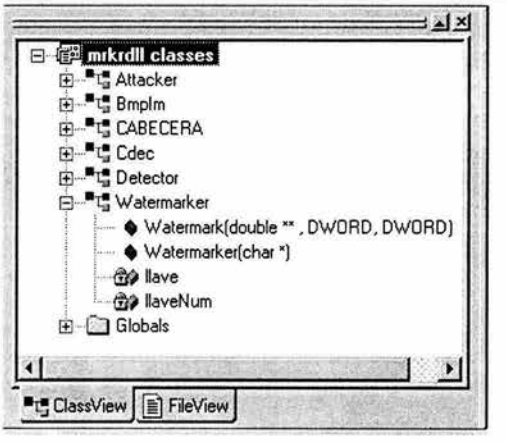


Fig. 4.a.2-3 La clase Watermarker

Detector

La clase Detector se encarga de detectar la presencia de una marca de agua en una imagen digital. Esto lo realiza calculando la correlación que existe entre los coeficientes wavelet importantes (es decir, los de mayor valor) de la imagen marcada original y de una imagen atacada que pudiera o no haber sido originada por medio de procesamientos o ruido aplicados a la primera.

El módulo de detección funciona de manera independiente del flujo de marcado de la imagen, por lo cual, a pesar de hacer uso tanto de las funciones de la clase BmpIm como de las funciones de la clase Cdec, lo hace de manera distinta que en el proceso de marcado.

Hay que hacer notar que de momento la capacidad de procesamiento del detector es muy pequeña, por lo cual solamente funciona para imágenes de las mismas dimensiones físicas y componentes de colores, esto es: no puede detectarse la correspondencia entre una imagen RGB y una en nivel de gris, aún cuando ambas representaran la misma escena en la realidad. Tampoco puede detectarse la correspondencia entre una imagen, ya sea en nivel de gris o RGB de dimensiones NxN y una segunda imagen, de dimensiones $\alpha N \times \alpha N$ obtenida a partir de la primera, en donde α es un número cualquiera que pertenece tanto al conjunto de los números naturales como al de los racionales positivos.

Esto parecería indicar que el algoritmo planteado es vulnerable a los ataques llamados "cropping", los cuales consisten en literalmente, cortar un pedazo de la imagen en el cual no se halle presente la marca de agua y después hacer pasar dicho pedazo como la imagen original, no marcada. Esto no es así, puesto que el algoritmo de inserción es en efecto resistente a este tipo de ataques al insertar la marca de agua en todas aquellas regiones de la imagen con alta actividad, que son las que nos brindan las texturas: es claro que una región con poca actividad tendrá también poca información importante y en consecuencia generará poco interés por parte de un atacante que intente averiguar si existe en ella una marca de agua.

No obstante, conforme el desarrollo de esta aplicación continúe se encontrarán esquemas de detección más eficientes que permitan observar este tipo de ataques y corregir los puntos mencionados anteriormente.

En la figura 4.a.2-4 se muestra la estructura de la clase Detector:

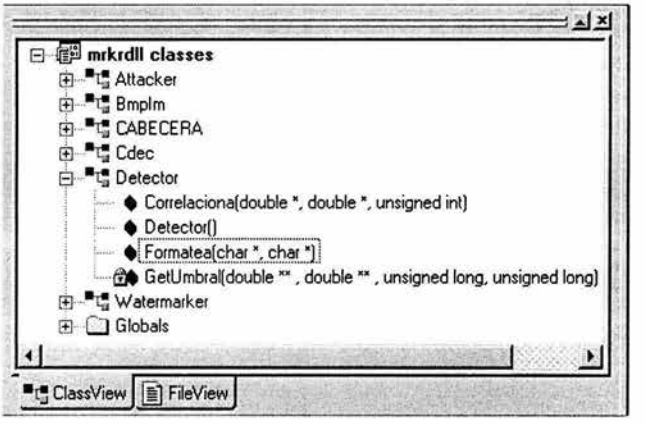


Fig. 4.a.2-4 La clase Detector

Como puede observarse, la clase Detector no tiene propiedades sino que es sencillamente un conjunto de métodos empleados para la detección de una marca de agua digital. Podría pensarse como una interfaz o un repositorio de funciones, sin embargo decidimos construir una clase por sencillez.

- **Detector.** Es el constructor de la clase Detector. No toma parámetros ni realiza ningún tipo de inicialización, sencillamente se usa para indicar al compilador que va a usarse un objeto capaz de accesar a todos los métodos de esta clase.

- **Formatea.** Realiza la mayor parte del proceso de detección al coordinar los módulos de correlación y de umbral. Instancia también un par de objetos BmpIm y un par de objetos Cdec, un par por imagen. La función de estos objetos es: para cada imagen se instancia un objeto BmpIm para obtener sus valores de luminancia y un codificador. Finalmente, cada codificador se usa para obtener la transformada wavelet de la imagen. Después de esto, se llama a la función GetUmbral() para obtener los valores de la transformada wavelet que son mayores a cierto umbral (que son los que se emplearán para la detección al ser éstos en los que se ha insertado la marca) y finalmente a Correlaciona(), que es la que realmente se encarga de realizar la detección.

- **Correlaciona.** La función correlaciona es quien realmente se encarga de realizar la comparación entre las dos imágenes de estudio. En realidad no compara las imágenes sino dos arreglos lineales especialmente formateados por medio de las funciones Formatea() y GetUmbral(), de manera que contengan solamente los coeficientes importantes de la transformada wavelet. Esto se hace así para ahorrar procesamiento pues al aumentar el tamaño físico de las imágenes, éstas se vuelven mucho más grandes a nivel de almacenamiento, lo cual es especialmente cierto para las imágenes RGB, que se componen de 3 matrices de N x N elementos. Una vez obtenidos estos arreglos lineales, se calcula la correlación cruzada entre ellos y se regresa un número real en el intervalo (-1,1) que indica qué tanto se parecen entre sí las imágenes. Un valor de 1 corresponde a una correspondencia perfecta mientras que al acercarnos al cero, la correspondencia va degenerando hasta llegar al punto en que las imágenes son totalmente

diferentes (correspondencia de cero). Los valores negativos de correspondencia no son considerados.

- **GetUmbral.** Esta función tiene como objetivo establecer el umbral que los coeficientes de la transformada wavelet deben superar para ser tomados en cuenta en el proceso de detección de la marca de agua. Toma como parámetros las matrices de luminancia de las dos imágenes a comparar y sus dimensiones físicas. Devuelve un arreglo de dos elementos de los cuales el primero es el umbral para la primera imagen y el segundo es el umbral para la segunda imagen. Estos valores son evaluados por la función Formatea() para obtener un par de arreglos lineales que representan los coeficientes wavelet importantes para las imágenes, los cuales serán luego comparados por medio de la función Correlaciona().

Attacker

La clase Attacker no es realmente parte del proceso de marcado de la imagen sino que se ha incluido para simular algunos de los ataques más usuales a los que podría ser sometida la imagen, ya sea intencional o no intencionalmente. Entre estos ataques están los ataques geométricos, la modificación directa de los pixeles de la imagen marcada y tres tipos de filtrado: filtrado pasa altas o realce de contornos, filtrado pasa bajas y filtrado de mediana.

El ataque a la imagen se realizará sobre sus pixeles y no sobre sus coeficientes wavelet, los valores de la DCT, transformaciones Affine o cualquier otro tipo de proceso avanzado, puesto que los ataques modelados, los cuales además son los más usuales, son aquellos efectuados por medio de las operaciones comunes del procesamiento digital de imágenes. Los ataques denominados "quirúrgicos", que son todos aquellos procesamientos efectuados sobre la imagen con el total conocimiento del algoritmo de inserción de la marca de agua, son un tema que queda pendiente para el posterior desarrollo de la aplicación y deberán ser abordados con una profundidad suficiente para lograr implementar un esquema eficiente de marcas de agua.

En los siguientes párrafos se estudiará la estructura y los métodos de la clase Attacker

La clase attacker posee únicamente tres propiedades, que son los kernels para los filtros pasa altas, pasa bajas y de mediana, que se convolucionarán con la imagen a manera de obtener una nueva imagen: la imagen atacada. A este respecto, cabe aclarar que la clase Attacker también hace uso tanto de la clase BmpIm como de la clase Cdec para realizar operaciones de E/S de archivos únicamente, esto debido a la incompatibilidad de algunos tipos de datos entre las plataformas Microsoft Visual Basic 6.0 y Microsoft Visual C++. Fuera de estos detalles, la clase Attacker es autosuficiente y podría trabajar por sí misma bajo ciertas condiciones en las cuales no se necesitara almacenar la imagen atacada, sino sencillamente estudiarla y desecharla. En la figura 5.b.5-1 se muestra el esquema de la clase Attacker.

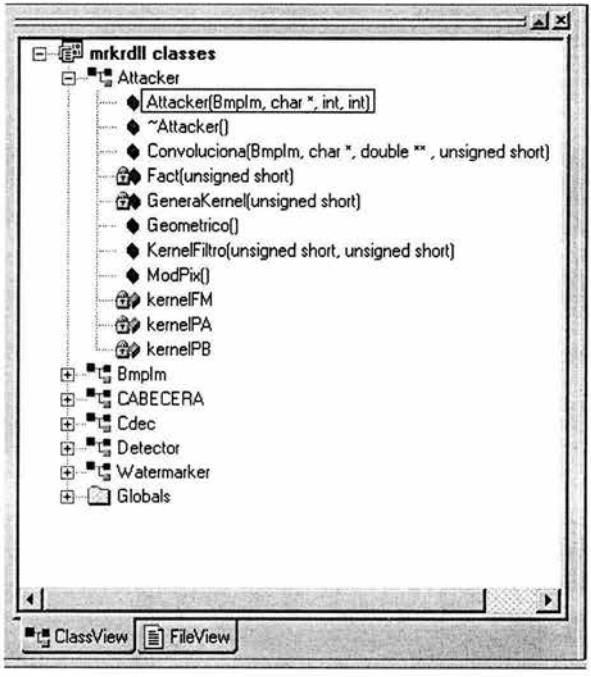


Fig. 4.a.2-5 La clase Attacker

Métodos de la clase Attacker:

- **Attacker.** Es el constructor del módulo de ataques y se encarga de realizar llamadas a las funciones miembro KernelFiltro() y Convoluciona(). La primera de éstas comienza con el proceso de inicialización de los kernels para los filtros a emplear en el ataque mientras que la segunda es la que efectivamente realiza los ataques al convolucionar la imagen con el filtro deseado. Attacker() toma como parámetros un objeto Bmplm que representa la imagen a atacar, la ruta de dicha imagen, el soporte del filtro que se desea emplear y finalmente el tipo de filtro que se usará para el ataque. Los últimos dos parámetros son definidos por el usuario desde la aplicación Watermarking.exe. Actualmente el soporte del filtro puede ser de 3, 5 ó 7, puesto que para la implementación de filtros con soportes mayores se requiere un poder computacional mayor al que poseemos actualmente, aunque hay que hacer notar que éste no es ni con mucho el *state of the art* del poder computacional que existe al momento.

- **~Attacker.** Es el destructor de la clase. De nuevo es un método no relevante en este estudio y se utiliza sólo a nivel del compilador .

- **Convoluciona.** La función convoluciona cumple con dos tareas, la primera y la más importante de ellas es la de efectuar el ataque a la imagen digital por medio de la convolución de ésta con un filtro previamente obtenido, del tipo definido por el usuario y con las dimensiones que éste mismo ha determinado. La segunda tarea de la función convoluciona es instanciar un objeto de la clase Cdec que se empleará tanto para obtener la matriz de luminancias a convolucionar con el filtro como para escribir la imagen atacada en un nuevo archivo, usando para esto la función

Cdec::ModImg con los parámetros adecuados. Convoluciona() recibe como parámetros un objeto de la clase BmpIm, que representa la imagen a atacar y se usa además para instanciar un objeto Cdec que nos permita accesar a sus valores de luminancia y realizar operaciones de E/S; la ruta hasta dicha imagen, el tipo de filtro a emplear para la convolución, y finalmente el soporte de dicho filtro. Como valor de retorno nos da una cadena que indica la ruta en la que se almacena la imagen atacada, según lo que se ha mencionado en el módulo de ataques de la aplicación Watermarking.exe.

- **Fact.** La función miembro privada Fact() se encarga de calcular el factorial de un número y es empleada en Attacker::GeneraKernel() para generar un kernel pasa bajas unidimensional el cual será convolucionado consigo mismo para obtener un kernel bidimensional. Recibe como parámetro un número entero positivo o cero y regresa su factorial.

- **GeneraKernel.** La función privada GeneraKernel() es la encargada de generar el kernel unidimensional de los filtros pasa bajas y pasa altas, aunque es de hacer notar que en sí misma, GeneraKernel() solamente es capaz de generar filtros pasa bajas. Esta función es llamada por la función KernelFiltro() para obtener un filtro unidimensional a partir del cual se inicializan las propiedades kernelPA y kernelPB de la clase Attacker. Recibe como parámetro el soporte del filtro a generar y entrega como resultado un arreglo unidimensional de *n* elementos de los valores obtenidos por la función Fact(), en donde *n* representa el soporte del filtro. Este arreglo está formateado especialmente, según la regla del binomio de Newton, para que al ser convolucionado consigo mismo en el dominio de Fourier, se pueda obtener un kernel pasa bajas de dos dimensiones.

- **Geométrico.** Esta función se encarga de realizar los ataques geométricos a la imagen, esto es, de realizar rotaciones y/o translaciones a la misma. El modelo a emplear es el modelo lineal simplificado para transformaciones no deformables, puesto que creemos que es suficiente para ejemplificar este tipo de ataques, además de que para la mayoría de los casos, una transformación deformable de una imagen digital la volvería inutilizable. Ejemplo de esto son las imágenes médicas, en las que la deformación de la imagen podría ser incluso fatal para un paciente. Los parámetros de entrada son el desplazamiento en x, el desplazamiento en y, el desplazamiento en z (opcional, aunque en la práctica será casi siempre de cero), la rotación en z (también opcional) y el centro de rotación, para el caso en que la rotación en z sea distinta de 0 o de n x 360 grados, en donde *n* es un número entero.

- **KernelFiltro.** KernelFiltro() es la función encargada de inicializar las propiedades kernelPA y kernelPB de la clase Attacker, aunque delega gran parte de esta labor a las funciones miembro privadas GeneraKernel() y Fact(), las cuales trabajando en conjunto entregan a KernelFiltro() un arreglo unidimensional que representa un filtro pasa bajas de soporte n. Si el filtro deseado en un filtro pasa bajas, KernelFiltro() realiza la convolución de dicho filtro consigo mismo a manera de obtener el filtro de dos dimensiones deseado; en caso contrario, antes de realizar la convolución, KernelFiltro() convierte el kernel unidimensional pasa bajas en un kernel unidimensional pasa altas, el cual nuevamente es convolucionado consigo mismo para obtener un kernel pasa altas de dos dimensiones. Recibe como parámetros el soporte o máscara del filtro a emplear así como su tipo (pasa altas, pasa bajas o filtro de mediana).

  Para el caso del filtro de mediana, no es necesario hacer uso de las funciones miembro privadas de la clase, en lugar de esto GeneraKernel() obtiene un filtro de tipo *boxfilter* con el soporte adecuado.

  Esta función no tiene ningún valor de retorno, en lugar de esto, modifica directamente las propiedades del objeto de la clase Attacker empleado para el ataque.

- **ModPix.** La función ModPix() representa un ataque extremadamente sencillo que se ha incluido en la aplicación solamente con fines ilustrativos, puesto que en principio de cuentas no debería representar una gran amenaza para un esquema de marcas de agua digitales. Consiste en

modificar un pixel en una posición definida por el usuario, con un valor también definido por el usuario, mayor o igual que cero y menor que 255, de acuerdo con los valores de luminancia aceptados para una imagen BMP. Recibe como parámetros las coordenadas del pixel que se desea modificar y el nuevo valor que se le asignará. No tiene ningún valor de retorno ni requiere de ningún tipo de kernel.

A pesar de lo anterior, se ha incluido para que el usuario pueda ver de manera directa la cantidad de distorsión que debe aplicarse a una imagen para poder deshacer la marca de agua.

mrkrdll

Mrkrdll es la rutina principal de la biblioteca de enlace dinámico mrkrdll.dll, la cual es un repositorio de llamadas a funciones que implementan los objetos definidos anteriormente y hacen uso de sus métodos.

Es la encargada del ligado con la aplicación Watermarking.exe, sirviendo como una puerta de entrada y salida de información al direccionar las llamadas a las funciones y los argumentos entre módulos definidos en Visual Basic y en Visual C++. También define la visibilidad de los métodos de la biblioteca y restringe el tipo de datos que pueden importarse o exportarse entre ambas plataformas.

La última de sus funciones es la de establecer la separación entre los módulos de marcado de la imagen, detección de la marca de agua y ataque de la imagen marcada, ruteando cada uno de éstos a la función pertinente de la biblioteca.

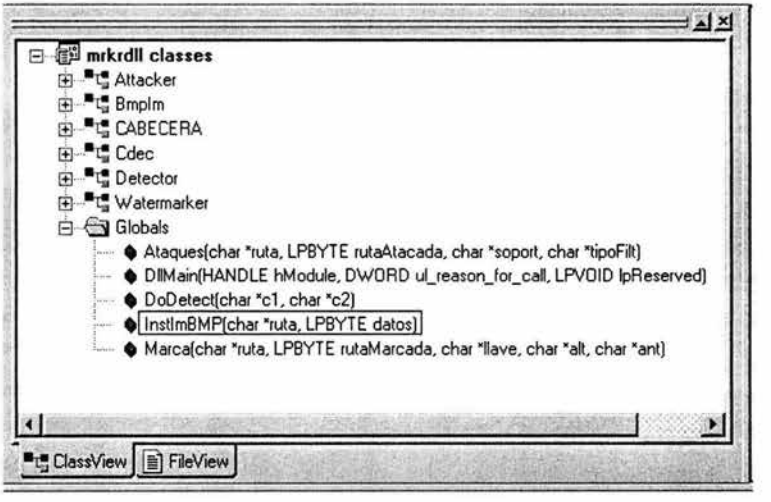


Fig. 4.a.2-6 Las funciones de la rutina principal de la biblioteca

A continuación se describen las funciones de la rutina principal de la biblioteca mrkrdll.dll:

- **Ataques.** Evidentemente, es la puerta de entrada al módulo de ataques, tanto para la biblioteca de funciones como para la aplicación Watermarking.exe, siendo implementada por ambas. Recibe desde la aplicación la ruta de la imagen a atacar, un arreglo de bytes que al terminar el proceso representará la ruta de la imagen atacada, una cadena que representa el soporte de los filtros de la clase Attacker y por último, otra cadena que representa el tipo del filtro a emplear.

  Se encarga también de instanciar los objetos Attacker y BmpIm que se usarán para realizar el ataque.

  Regresa un código de error de cero si la función ha terminado exitosamente y un número diferente de cero si no ha sido así.

- **DllMain.** No tiene ningún valor para este proceso, aunque define el punto de entrada a la rutina principal a nivel del ligado entre la aplicación y su biblioteca de funciones.

- **DoDetect.** Se encarga del módulo de detección de la marca de agua y al igual que en el caso del módulo de ataques, es implementada tanto a nivel de la aplicación como a nivel de la biblioteca de funciones. A pesar de esto es muy simple y su única funcionalidad es la de instanciar un objeto de la clase Detector (cuya instanciación es vacía) y después, de llamar al método Formatea() de dicho objeto, usando como parámetros dos cadenas obtenidas desde Watermarking.exe, que representan las rutas de las imágenes a comparar. Su valor de retorno es el coeficiente de correlación cruzada entre las imágenes.

- **InstImBMP.** Función implementada en la aplicación y la biblioteca. Se encarga de alimentar la ventana de información de imagen en Watermarking.exe con los valores de la cabecera de un objeto BmpIm que ella misma implementa. Toma como parámetros de entrada la ruta de la imagen a abrir y un arreglo de bytes que representa la información de la imagen. Ambos parámetros son alimentados desde la aplicación Watermarkingexe.

  Si la función ha terminado exitosamente, regresa un cero, en caso contrario regresa un 1.

- **Marca.** Es la más compleja de las funciones de la biblioteca y se encarga de administrar al módulo de ataques, para lo cual instancia antes un objeto de la clase BmpIm y posteriormente un objeto Cdec que será el encargado de proveer la funcionalidad necesaria para marcar la imagen y para escribir la imagen marcada en disco duro.

  Toma como parámetros la ruta de la imagen original, un arreglo de bytes que representará la ruta de la imagen marcada al terminar el proceso, y tres cadenas, que representan la llave de inserción y la altura y el ancho del tile respectivamente.

  Regresa un cero si termina exitosamente y un valor diferente de cero si no ha sido así.

De esta manera hemos visto a detalle la implementación de ambas aplicaciones: Watermarking.exe y su biblioteca de funciones mrkrdll.dll, para lograr tener una idea clara del método empleado por éstas para la inserción de marcas de agua en imágenes digitales. Aún es necesaria mucha investigación respecto a este tema, la cual estará sustentada en teoremas matemáticos y probabilísticos que permitan desarrollar esquemas más eficientes computacionalmente hablando y por supuesto, más robustos, de manera que logran insertar una marca de agua muy próxima a la invulnerabilidad, pero sin sacrificar la calidad visual de la imagen.

Hay que notar que esta implementación no es esteganográfica, lo cual significa que a pesar de insertar una marca de agua, ésta no contiene ninguna información más que su propia presencia, por lo cual también es necesaria la investigación de métodos que combinen las marcas de agua digitales con las

técnicas de esteganografía y criptografía de manera que la marca insertada sea capaz de proporcionar información más allá de ella misma, aunque esto será objetivo de posteriores desarrollos que, basados en el presente trabajo servirán para la implementación de sistemas más complejos.

Finalmente, a pesar de que este trabajo no es una tesis de desarrollo de software, sino la implantación de un esquema de marcas de agua digitales que tuvo como resultado final la creación de un sistema, creemos importante incluir el código fuente de las aplicaciones, con el fin de que el lector sea capaz de comprender los detalles más mínimos del sistema que proponemos.

Dicho código fuente puede encontrarse en el anexo 1 de este trabajo.

5. CONCLUSIONES

Al inicio del presente trabajo se contaban entre los objetivos la implementación de un sistema de marcas de agua digitales que cumpliera con los requisitos de indetectabilidad, imperceptibilidad y bajo costo computacional, que además utilizara algunas de las características del sistema visual humano para incrementar su robustez.

Además, se deseaba desarrollar un programa de aplicación que sirviera para ejemplificar los conceptos expuestos en el presente trabajo. Hemos obtenido un programa de aplicación y un esquema de marcado digital que nos permite considerar que los objetivos planteados al inicio de esta tesis se han cumplido cabalmente, de la manera que se describe a continuación:

Respecto a los requisitos de la imperceptibilidad y el uso de las características del sistema visual humano, el algoritmo que empleamos para el mecanismo de inserción de la marca de agua digital se basa en la transformada wavelet discreta en dos dimensiones, usando para ésta el algoritmo Daubechies-4, descrito por la matemática de origen belga Ingrid Daubechies, y una de las características de las transformadas llamada *separabilidad*, lo cual sencillamente nos indica que para obtener la transformada bidimensional de una imagen sencillamente basta con aplicar un par de transformadas unidimensionales en cada una de sus dos dimensiones. Con lo anterior logramos separar las componentes de frecuencias altas de las imágenes de prueba y distribuirlas en cuatro tipos de regiones, según lo indica la teoría referente a la transformada wavelet discreta. Estas regiones se denominan *coeficientes*, y se clasifican en: coeficiente de aproximación, coeficiente de contraste horizontal, coeficiente de contraste vertical y coeficiente de contraste diagonal. Cada una de estas regiones presenta características distintas que la hacen candidata a ser usada para insertar en ella una marca de agua digital de manera que dicha marca sea robusta e imperceptible para el ojo humano.

El conocimiento obtenido acerca del HVS (sistema visual humano, por sus siglas en inglés), nos permitió decidir el número de niveles de transformación wavelet a emplear para insertar la marca de manera tal que una persona promedio en situaciones de iluminación promedio y sentada a una distancia dada del monitor, sea incapaz de ver la marca de agua que se ha insertado en la imagen, nominalmente, tres niveles de transformación deberían de ser necesarios para una persona que mira una imagen en un monitor VGA o SVGA de unas 15", con una resolución de 1024x768 a una distancia de entre 40 cm y 1 metro.

El requisito de indetectabilidad de la marca de agua lo cumplimos insertando la marca de agua en las regiones de altas frecuencias de la imagen. Esto puede verse desde dos enfoques cuasi equivalentes: si se ve en el dominio del espacio, se inserta la marca de agua en las regiones de la imagen que presentan una actividad de por sí grande, es decir en las regiones con textura, insertando en la imagen intensidades inferiores al JND para el ojo humano, usando para esto un generador de números pseudoaleatorios inicializado a una semilla dada por la llave de inserción, de modo que no sea posible detectar ni siquiera matemáticamente el grado de modificación al que se ha sometido el coeficiente de luminancia. Si se ve desde el punto de vista de la frecuencia, es en los coeficientes wavelet mayores que cierto valor de umbral T, en los cuales se ha insertado la marca de agua, de modo que si la imagen es analizada en el dominio de la frecuencia, sólo podrán detectarse picos aleatorios de frecuencias altas que no tienen ninguna interpretación práctica y no pueden usarse para discernir el valor de la marca de agua.

En cuanto al costo computacional, hemos reducido la cantidad de operaciones necesarias para insertar la marca de agua separando la transformación wavelet bidimensional en un par de transformaciones unidimensionales a lo largo de los renglones y después de las columnas de las imágenes de prueba, lo cual ahorra varios milisegundos por transformación unidimensional.

Finalmente, respecto a la aplicación obtenida, Watermarking.exe y mrkrdll cumplen cabalmente con el objetivo planteado para las mismas. Cabe mencionar que ni la aplicación Watermarking.exe ni la biblioteca de funciones mrkrdll.dll están pensadas como un release comercial, por lo menos por el momento, por lo cual aún necesitan mucho desarrollo sobre todo en aspectos como la estabilidad, la optimización de código y sobre todo, la portabilidad.

Sin embargo, ambas demuestran el proceso de inserción de una marca de agua, la separación de la imagen de mapa de bits en Tiles a los cuales se insertará la marca de agua por separado, un detector incipiente de la marca de agua y algunos de los ataques y operaciones más comunes a los que puede verse sometida una imagen marcada.

Para todo nuestro trabajo de investigación, nos hemos apoyado en el sistema de desarrollo propietario Microsoft Visual Studio 6.0, por la confianza y facilidad con la que este se opera, aunque nuestra limitante clave es que nuestra aplicación está confinada a trabajar en un ambiente Windows 98, Me, 2000 o XP. En un principio se construyó un sistema que operaba en modo texto, cuya entrada y salida se manejaba a partir de la consola. Sin embargo esto nos pareció insuficiente en la práctica, pues deseábamos que el usuario pudiera tener interacción con un entorno gráfico a manera de hacer la aplicación un poco más amigable: recordemos que después de todo el objetivo principal que perseguimos es el de ejemplificar al usuario promedio el uso de las marcas de agua digitales, por lo cual es de esperar que el usuario promedio se sienta más a gusto trabajando en un entorno puramente gráfico. Por esta razón, decidimos utilizar el poder de cómputo que nos otorga C++, su velocidad y su orientación a objetos, y complementarlo con la facilidad para construir interfaces gráficas de Visual Basic. Para que estos dos lenguajes de programación se comunicaran de una forma sencilla y confiable, tuvimos que crear una Biblioteca de vínculos dinámicos (DLL por las siglas en inglés de Dynamic Link Library) programada enteramente en C++ mediante llamadas a la API de Windows (razón por la cual la aplicación no es portable entre plataformas), y relacionar las variables de entrada en Visual Basic con las variables de C++ por medio de las equivalencias entre los tipos de datos en la plataforma Windows y casts explícitos declarados en las cabeceras de la dll y en el repositorio principal de funciones de la misma para que se efectuara dicha conexión entre los lenguajes ya mencionados. Con esta implementación corroboramos que el poder de la programación orientada a objetos y la versatilidad del uso de la API de Windows, son dos grandes herramientas para el desarrollo de software.

Para desarrollos futuros quedan los temas de la implementación de detectores más potentes, efectivos y sobre todo, de un menor costo computacional, y el uso de técnicas esteganográficas para insertar en la imagen marcas de agua que contengan cualquier tipo de información que el usuario quiera o necesite agregar a sus imágenes. También para desarrollos futuros quedan los temas como la inserción y extracción de marcas de agua en imágenes de cualquier tipo, así como en cualquier otro tipo de soporte digital, como audio y video. Otra de las tareas para las cuales dedicaremos tiempo y esfuerzo, será para alcanzar la portabilidad de nuestro sistema a sistemas basados en la licencia GPL y poder distribuir nuestro sistema bajo la misma filosofía.

Estas son las conclusiones de nuestra tesis de licenciatura, pero tenemos la firme determinación de continuar con nuestros estudios de postgrado e incrementar nuestra investigación con técnicas y herramientas matemáticas de mayor complejidad, otorgándole a nuestro sistema un mayor rango de confiabilidad, seguridad, y robustez, y con esto ser parte de la comunidad que busca mejoras en la inserción de marcas de agua digitales en diversos documentos.

Anexo 1

Código fuente de las aplicaciones

Componentes de mrkrdll.dll
Clase Attacker
La clase Attacker se implementa por medio de dos archivos: Attacker.h y Attacker.cpp, los cuales definen la clase Attacker y la implementación de sus métodos respectivamente.

Attacker.h
```
class Attacker{
private:
	unsigned short Fact(unsigned short);
	double **kernelPA;
	double **kernelPB;
	double **kernelFM;
	unsigned short *GeneraKernel(unsigned short);
public:
	Attacker(BmpIm,char*,int,int);
	~Attacker();
	char *Convoluciona(BmpIm,char *,double**,unsigned short);
	void Geometrico();
	void KernelFiltro(unsigned short,unsigned short);
	void ModPix();
};
```

Attacker.cpp
```
#include <windows.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "stdafx.h"
#include "Attacker.h"


/*****************************************************************************
 * Constructor de la clase Attacker                                          *
 *****************************************************************************/

Attacker::Attacker(BmpIm attackedIm,
                   char *ruta,
                   int soporte,
                   int tipoFiltro){

	unsigned short tipoFilt = (unsigned short) tipoFiltro;


	KernelFiltro((unsigned short)soporte,tipoFilt);

	switch(tipoFilt){
	case 0:
		ruta = Convoluciona(attackedIm,ruta,kernelPB,(unsigned short)soporte);
		break;
	case 1:
		ruta = Convoluciona(attackedIm,ruta,kernelPA,(unsigned short)soporte);
		break;
	case 2:
		ruta = Convoluciona(attackedIm,ruta,kernelFM,(unsigned short)soporte);
		break;
```

```
	default:
		SetLastError(EPT_S_INVALID_ENTRY);
		break;
	}

}


/*****************************************************************************
 * Segundo constructor de la clase Attacker                                  *
 *****************************************************************************/

Attacker::Attacker(BmpIm attackedIm,
                   char *ruta,
                   int despX,
                   int despY,
                   int despZ,
                   int teta,
                   int XG,
                   int YG){

	ruta = Geometrico(attackedIm,ruta,despX,despY,despZ,teta,XG,YG);

}

/*****************************************************************************
 * Destructor de la clase Attacker                                           *
 *****************************************************************************/
Attacker::~Attacker(){
}


/*****************************************************************************
 * unsigned short Fact(unsigned short)                                       *
 * Calcula el factorial del argumento proporcionado. Se emplea para obtener  *
 * los coeficientes del kernel de 1 dimensión para la transformada           *
 *****************************************************************************/
unsigned short Attacker::Fact(unsigned short num){

	unsigned short numFact=1;

	do{
		if(num>0){
			numFact *= num;
			num--;
		}
		else
			return 1;
	}
	while(num>1);

	return numFact;
}

/*****************************************************************************
 * void Convoluciona(BmpIm,double**)                                         *
 * Realiza la convolución de las componentes de luminancia con los filtros   *
 *****************************************************************************/
char* Attacker::Convoluciona(BmpIm attackedIm,
                             char *ruta,
                             double **filtro,
                             unsigned short soporte){
```

```
	unsigned long bitdepth = attackedIm.GetAtributo(6);
	double **matrizLuminancias;
	unsigned short offsetFiltro = (soporte - 1)/2;
	unsigned short coordInic = ((soporte + 1)/2) - 1;
	unsigned short coordFinal = (unsigned short)attackedIm.GetAtributo(4) -
((soporte + 1)/2);
	unsigned short i,j;
	short k=-offsetFiltro,l=-offsetFiltro;
	char *atacada;

	Cdec codec = Cdec(attackedIm.GetMapValsLum(0),
		attackedIm.GetAtributo(3),
		attackedIm.GetAtributo(4),
		bitdepth,
		attackedIm.GetAtributo(3),
		attackedIm.GetAtributo(4));

	switch(bitdepth){

	case 8:

		matrizLuminancias = codec.GetMatrix('g');

		for(i=coordInic;i<coordFinal;i++){
			for(j=coordInic;j<coordFinal;j++){

				k=-offsetFiltro;
				while(k<=offsetFiltro){
					l=-offsetFiltro;
					while(l<=offsetFiltro){
						matrizLuminancias[i+k][j+l] =
matrizLuminancias[i+k][j+l] * filtro[k+offsetFiltro][l+offsetFiltro];
						l++;
					}
					k++;
				}

			}
		}

		break;

	case 24:

		matrizLuminancias = codec.GetMatrix('r');
		for(i=coordInic;i<coordFinal;i++){
			for(j=coordInic;j<coordFinal;j++){
				k=-offsetFiltro;
				while(k<=offsetFiltro){
					l=-offsetFiltro;
					while(l<=offsetFiltro){
						matrizLuminancias[i+k][j+l] =
matrizLuminancias[i+k][j+l] * filtro[k+offsetFiltro][l+offsetFiltro];
						l++;
					}
					k++;
				}
			}
		}

		matrizLuminancias = codec.GetMatrix('v');
		for(i=coordInic;i<coordFinal;i++){
			for(j=coordInic;j<coordFinal;j++){
```

```
                        k=-offsetFiltro;
                        while(k<=offsetFiltro){
                              l=-offsetFiltro;
                              while(l<=offsetFiltro){
                                    matrizLuminancias[i+k][j+l] =
matrizLuminancias[i+k][j+l] * filtro[k+offsetFiltro][l+offsetFiltro];
                                    l++;
                              }
                              k++;
                        }
                  }
            }

            matrizLuminancias = codec.GetMatrix('a');
            for(i=coordInic;i<coordFinal;i++){
                  for(j=coordInic;j<coordFinal;j++){
                        k=-offsetFiltro;
                        while(k<=offsetFiltro){
                              l=-offsetFiltro;
                              while(l<=offsetFiltro){
                                    matrizLuminancias[i+k][j+l] =
matrizLuminancias[i+k][j+l] * filtro[k+offsetFiltro][l+offsetFiltro];
                                    l++;
                              }
                              k++;
                        }
                  }
            }

            break;

      default:
            break;
      }


      if (bitdepth == 8 )
            atacada = codec.ModImg(attackedIm.GetMapValsLum(1),ruta,1);
      else if (bitdepth == 24){
            LPBYTE lpbyte = new BYTE;
            atacada = codec.ModImg(lpbyte,ruta,1);
      }

      return atacada;

}


/*****************************************************************************
 * Realiza la transformación Affine de la imagen. Usa el modelo lineal simpli- *
 * ficado.                                                                    *
 *****************************************************************************/
char *Attacker::Geometrico(BmpIm attackedIm,
                           char *ruta,
                           int tx,
                           int ty,
                           int k,
                           int teta,
                           int XG,
                           int YG){


      ///// Dimensiones y características de la imagen
```

```
	unsigned long bitdepth = attackedIm.GetAtributo(6);
	unsigned long alto = attackedIm.GetAtributo(3);
	unsigned long ancho = attackedIm.GetAtributo(4);
	////////////////////////////////////////////////

	///// Variables de trabajo
	double **matrizLuminancias;
	double **matrizTemporal;
	unsigned short i,j,xf,yf;		// Contadores de desplazamiento original y
Affine
	double a,b;
	const double c = pow(teta,2)+pow((1-k),2);
	////////////////////////////////////////////////

	///// Ruta hacia la imagen atacada
	char *atacada;

	Cdec codec = Cdec(attackedIm.GetMapValsLum(0),
		alto,
		ancho,
		bitdepth,
		alto,
		ancho);

	matrizTemporal = new double *[alto];
	for(i=0;i<alto;i++)
		matrizTemporal[i] = new double [ancho];


	switch(bitdepth){

	case 8:

		matrizLuminancias = codec.GetMatrix('g');

		for(i=0;i<alto;i++){
			for(j=0;j<ancho;j++){

				// Antes de comenzar, se inicializan las variables a y b que
				// se utilizarán para calcular los desplazameintos Affine.

				a = i + tx - k*XG + teta*YG;
				b = j + ty - teta*XG + k*YG;

				// Y a continuación se calculan los desplazamientos
				xf = (unsigned short)ceil((a*(1-k)-b*teta)/c);
				yf = (unsigned short)ceil((a*teta+b*(1-k))/c);

				// Y se informa la matriz temporal de luminancias
				matrizTemporal[xf][yf] = matrizLuminancias[i][j];


			}
		}

		for(i=0;i<alto;i++){
			for(j=0;j<ancho;j++){
				matrizLuminancias[i][j]= matrizTemporal[i][j];
			}
		}


		break;
```

```
case 24:

      matrizLuminancias = codec.GetMatrix('r');
      for(i=0;i<alto;i++){
            for(j=0;j<ancho;j++){

                  // Antes de comenzar, se inicializan las variables a y b que
                  // se utilizarán para calcular los desplazamientos Affine.

                  a = i + tx - k*XG + teta*YG;
                  b = j + ty - teta*XG + k*YG;

                  // Y a continuación se calculan los desplazamientos
                  xf = (unsigned short)ceil((a*(1-k)-b*teta)/c);
                  yf = (unsigned short)ceil((a*teta+b*(1-k))/c);

                  // Y se informa la matriz temporal de luminancias
                  matrizTemporal[xf][yf] = matrizLuminancias[i][j];


            }
      }

      for(i=0;i<alto;i++){
            for(j=0;j<ancho;j++){
                  matrizLuminancias[i][j]= matrizTemporal[i][j];
            }
      }


      matrizLuminancias = codec.GetMatrix('v');
      for(i=0;i<alto;i++){
            for(j=0;j<ancho;j++){

                  // Antes de comenzar, se inicializan las variables a y b que
                  // se utilizarán para calcular los desplazamientos Affine.

                  a = i + tx - k*XG + teta*YG;
                  b = j + ty - teta*XG + k*YG;

                  // Y a continuación se calculan los desplazamientos
                  xf = (unsigned short)ceil((a*(1-k)-b*teta)/c);
                  yf = (unsigned short)ceil((a*teta+b*(1-k))/c);

                  // Y se informa la matriz temporal de luminancias
                  matrizTemporal[xf][yf] = matrizLuminancias[i][j];


            }
      }

      for(i=0;i<alto;i++){
            for(j=0;j<ancho;j++){
                  matrizLuminancias[i][j]= matrizTemporal[i][j];
            }
      }


      matrizLuminancias = codec.GetMatrix('a');
      for(i=0;i<alto;i++){
            for(j=0;j<ancho;j++){
```

```
                        // Antes de comenzar, se inicializan las variables a y b que
                        // se utilizarán para calcular los desplazamientos Affine.

                        a = i + tx - k*XG + teta*YG;
                        b = j + ty - teta*XG + k*YG;

                        // Y a continuación se calculan los desplazamientos
                        xf = (unsigned short)ceil((a*(1-k)-b*teta)/c);
                        yf = (unsigned short)ceil((a*teta+b*(1-k))/c);

                        // Y se informa la matriz temporal de luminancias
                        matrizTemporal[xf][yf] = matrizLuminancias[i][j];


                }
        }

        for(i=0;i<alto;i++){
                for(j=0;j<ancho;j++){
                        matrizLuminancias[i][j]= matrizTemporal[i][j];
                }
        }


        break;

    default:
        break;
    }


    if (bitdepth == 8 )
        atacada = codec.ModImg(attackedIm.GetMapValsLum(1),ruta,1);
    else if (bitdepth == 24){
        LPBYTE lpbyte = new BYTE;
        atacada = codec.ModImg(lpbyte,ruta,1);
    }

    delete [] matrizTemporal;

    return atacada;

}


/*****************************************************************************
 * unsigned short* GeneraKernel(unsigned short)                              *
 * Obtiene el kernel de una dimensión para los filtros pasa altas y pasa bajas *
 *****************************************************************************/
unsigned short* Attacker::GeneraKernel(unsigned short soporte){

    unsigned short *kernel1D;
    unsigned short sopmenosuno = soporte - 1;
    int i=0;

    kernel1D = new unsigned short[soporte];

    while(i<=sopmenosuno){
        kernel1D[i] = Fact(sopmenosuno)/(Fact(sopmenosuno - i)*Fact(i));
        i++;
    }
    /*for(int i=0; i==sopmenosuno; i++){
        kernel1D[i] = Fact(sopmenosuno)/(Fact(sopmenosuno - i)*Fact(i));
```

```
	}*/

	return kernel1D;

}


/*****************************************************************************
 * void KernelFiltro(unsgined short,char)                                    *
 * Obtiene el kernel 2D para los filtros pasa altas y pasa bajas             *
 *****************************************************************************/
void Attacker::KernelFiltro(unsigned short soporte,
                                        unsigned short tipoFiltro){

	unsigned short i,j,*kernel1D;
	//double **kernel2D;
	double coefNormalizacion=0;

	kernel1D = GeneraKernel(soporte);
	//kernel2D = new double *[soporte];

	for(i=0;i<soporte;i++){
		//kernel2D[i] = new double [soporte];
		coefNormalizacion += kernel1D[i];
	}

	coefNormalizacion = pow(coefNormalizacion,-2);

	switch (tipoFiltro){

	case 0:

		kernelPB = new double*[soporte];

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

			kernelPB[i] = new double[soporte];

			for(j=0;j<soporte;j++){
				//kernel2D[i][j] =
kernel1D[i]*kernel1D[j]*coefNormalizacion;
				kernelPB[i][j] = kernel1D[i]*kernel1D[j]*coefNormalizacion;
			}

		}


		break;

	case 1:

		kernelPA = new double*[soporte];

		for(i=0;i<soporte;i++){
			if(i>(soporte+1)/2)
				kernel1D[i] *= -1;
			kernelPA[i] = new double[soporte];
		}

		for(i=soporte-1;i==0;i--){
			for(j=soporte-1;j==0;j--)
				kernelPA[i][j] = kernel1D[i]*kernel1D[j];
		}
```

```
            //kernelPA = kernel2D;
            break;

      case 2:

            kernelFM = new double*[soporte];
            coefNormalizacion = 0;

            for(i=0;i<soporte;i++){
                  kernel1D[i] = 1;
                  coefNormalizacion +=1;
                  kernelFM[i] = new double[soporte];
            }

            coefNormalizacion = pow(coefNormalizacion,-2);

            for(i=0;i<soporte;i++){
                  for(j=0;j<soporte;j++)
                        kernelFM[i][j] = kernel1D[i]*kernel1D[j]*coefNormalizacion;
            }

            //kernelFM = kernel2D;
            break;
      }

      //delete [] kernel2D;
}

/*****************************************************************************
 *****************************************************************************/
void Attacker::ModPix(){
}
```

Clase BmpIm y estructura CABECERA

La clase BmpIm y la estructura CABECERA se implementan por medio de los archivos BmpIm.h y BmpIm.cpp, en los cuales se definen la clase BmpIm y la estructura CABECERA, y la implementación de los métodos de la clase BmpIm respectivamente.

BmpIm.h

```
#include<windows.h>

struct CABECERA{
      WORD Tipo;                    // 0x4d42
      DWORD Tam;                    // Tamaño del archivo.
      WORD Reservado1;
      WORD Reservado2;
      DWORD Offset;                 // Offset al inicio de la imagen.
      DWORD TamHdr;                 // Tamaño de la cabecera de información
      DWORD Alto;                   // Alto de la imagen.
      DWORD Ancho;                  // Ancho de la imagen.
      WORD Planos;                  // Planos de color.
      WORD Bits;                    // Numero de bits por pixel.
      DWORD Compresion;             // 0 si la imagen no esta comprimida.
      DWORD TamIm;                  // Tamaño en bytes de la imagen.
      DWORD ResX;                   // Resolucion X.
      DWORD ResY;                   // Resolucion Y.
      DWORD NumCols;                // Numero de colores.
      DWORD ImCols;                 // Colores importantes.
};

class BmpIm{
private:
```

```
        CABECERA CAB;
        LPBYTE valsLum;
        LPBYTE map;

public:
        BmpIm(char*);
        ~BmpIm();
        unsigned long GetAtributo(
                small);
        LPBYTE GetMapValsLum(small);
        void SetAtributo(
                small,
                unsigned long);
        char * DespliegaDatos();
};
```

BmpIm.cpp

```
#include "BmpIm.h"
#include "Cdec.h"
#include "stdafx.h"


/*************************************************************************
 * Consructor de la clase BmpIm                                          *
 *************************************************************************/
BmpIm::BmpIm(char *ruta){

HANDLE hFile;
DWORD valRetSFP;
unsigned long *lpNumOfBytesRead = new unsigned long;
LPWORD lpBuffer = new WORD[27];                 // Arreglo en el que se lee la
cabecera


hFile = CreateFile(TEXT(ruta),                  // archivo a abrir
GENERIC_READ,           // archivo abierto para lectura
FILE_SHARE_READ,        // lectura compartida
NULL,                   // seguridad default
OPEN_EXISTING,          // solo si el archivo existe
FILE_ATTRIBUTE_NORMAL, // archivo con atributo normal
NULL);



/****
* A continuación se  realizan tres lecturas al archivo: en la primera de ellas se leen
* 54 bytes correspondientes a la cabecera, la cual nos  indica el número de bytes que
* deberemos regresar  en la segunda lectura, que  es la que incluye el colormap. Por
ul
* timo, leeremos los valores de luminancia.
****/


valRetSFP =
SetFilePointer(hFile,0,NULL,FILE_BEGIN);        // LLeva el apuntador al inicio del
archivo



// Se realiza la lectura #1 (cabecera del archivo)

ReadFile(hFile,lpBuffer,54,lpNumOfBytesRead,NULL);
```

```
CAB.Tipo = lpBuffer[0];                          CAB.Planos = lpBuffer[13];
CAB.Tam = (lpBuffer[2]<<16)|lpBuffer[1];         CAB.Bits = lpBuffer[14];
CAB.Reservado1 = lpBuffer[3];                    CAB.Compresion =
(lpBuffer[16]<<16)|lpBuffer[15];
CAB.Reservado2 = lpBuffer[4];                    CAB.TamIm =
(lpBuffer[18]<<16)|lpBuffer[17];
CAB.Offset = (lpBuffer[6]<<16)|lpBuffer[5];      CAB.ResX =
(lpBuffer[20]<<16)|lpBuffer[19];
CAB.TamHdr = (lpBuffer[8]<<16)|lpBuffer[7];      CAB.ResY =
(lpBuffer[22]<<16)|lpBuffer[21];
CAB.Alto = (lpBuffer[10]<<16)|lpBuffer[9];       CAB.NumCols =
(lpBuffer[24]<<16)|lpBuffer[23];
CAB.Ancho = (lpBuffer[12]<<16)|lpBuffer[11];     CAB.ImCols =
(lpBuffer[26]<<16)|lpBuffer[25];


/****
 * Para la segunda lectura, traeremos en map  el colormap de  la imagen. El número de
 * bytes a leer será de CAB.Offset-54, esto es, el  offset al inicio de los valores de
 * luminancia menos 54 bytes de la cabecera, que ya han sido leídos.
 ****/

map = new BYTE[CAB.Offset - 54];

valRetSFP =
SetFilePointer(hFile,54,NULL,FILE_BEGIN);      // Desplaza el file pointer para leer el
colormap



// Realiza la lectura #2
ReadFile(hFile,map,CAB.Offset-54,lpNumOfBytesRead,NULL);


/****
* Para la tercera lectura, traeremos en valsLum los valores de luminancia de la
imagen.
* El número de bytes a leer será de CAB.Tam-CAB.Offset
****/

valsLum = new BYTE[CAB.Tam - CAB.Offset];

valRetSFP =
SetFilePointer(hFile,CAB.Offset,NULL,FILE_BEGIN);    // Desplaza el file pointer para
leer el resto del archivo


// Realiza la lectura #3
ReadFile(hFile,valsLum,CAB.Tam-CAB.Offset,lpNumOfBytesRead,NULL);


// Se libera la memoria usada por lpBuffer

delete [] lpBuffer;


// Y finalmente se cierra el archivo

CloseHandle(hFile);

}
```

```
BmpIm::~BmpIm(){
map = NULL;
valsLum = NULL;
}


/*****************************************************************************
 *****************************************************************************/
char* BmpIm::DespliegaDatos(){

char cadtemp[20];
char datos[255];

strcpy(datos,"");
strcat(datos,_itoa(CAB.Tipo,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.Tam,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.Reservado1,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.Reservado2,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.Offset,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.TamHdr,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.Ancho,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.Alto,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.Planos,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.Bits,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.Compresion,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.TamIm,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.ResX,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.ResY,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.NumCols,cadtemp,10));strcat(datos,"-");
strcat(datos,_itoa(CAB.ImCols,cadtemp,10));strcat(datos,"_");

return datos;

}



/****
* Para GetAtributo y SetAtrbuto:
*
* Código     Atributo            Código Atributo
* 1          CAB.Tam             6              CAB.Bits
* 2          CAB.Offset          7              CAB.Compresion
* 3          CAB.Alto            8              CAB.TamIm
* 4          CAB.Ancho           9              CAB.ResX
* 5          CAB.Planos          10             CAB.ResY
* 11         CAB.NumCols         23             CAB.ImCols
****/
unsigned long BmpIm::GetAtributo(small atrCde){
      switch(atrCde){
      case 1:
            return CAB.Tam;
            break;
      case 2:
            return CAB.Offset;
            break;
      case 3:
            return CAB.Alto;
            break;
      case 4:
            return CAB.Ancho;
```

```
            break;
      case 5:
            return CAB.Planos;
            break;
      case 6:
            return CAB.Bits;
            break;
      case 7:
            return CAB.Compresion;
            break;
      case 8:
            return CAB.TamIm;
            break;
      case 9:
            return CAB.ResX;
            break;
      case 10:
            return CAB.ResY;
            break;
      case 11:
            return CAB.NumCols;
            break;
      case 12:
            return CAB.ImCols;
            break;
      default:
            return 0;
      }
}

/*****************************************************************************
 *****************************************************************************/
LPBYTE BmpIm::GetMapValsLum(small cde){
if (cde == 0)
return valsLum;
else
return map;
}

/*****************************************************************************
 *****************************************************************************/
void BmpIm::SetAtributo(small atrCde,unsigned long atrVal){
      switch(atrCde){
      case 1:
            CAB.Tam = atrVal;
            break;
      case 2:
            CAB.Offset = atrVal;
            break;
      case 3:
            CAB.Ancho = atrVal;
            break;
      case 4:
            CAB.Alto = atrVal;
            break;
      case 5:
            CAB.Planos = (unsigned short) atrVal;
            break;
      case 6:
            CAB.Bits = (unsigned short) atrVal;
            break;
```

```
        case 7:
                CAB.Compresion = atrVal;
                break;
        case 8:
                CAB.TamIm = atrVal;
                break;
        case 9:
                CAB.ResX = atrVal;
                break;
        case 10:
                CAB.ResY = atrVal;
                break;
        case 11:
                CAB.NumCols = atrVal;
                break;
        case 12:
                CAB.ImCols = atrVal;
                break;
        default:
                break;
        }
}
```

Clase Cdec

La clase Cdec se implementa por medio de los archivos Cdec.h y Cdec.cpp, en los cuales se definen la clase Cdec y la implementación de sus métodos respectivamente.

Cdec.h

```
#include <windows.h>
#include "Watermarker.h"

class Cdec{
private:
        double DCLevel;
        DWORD AlIm;
        DWORD AnIm;
        DWORD componentes;
        DWORD AlTile;
        DWORD AnTile;
        double **MLGris;
        double **MLRojo;
        double **MLVerde;
        double **MLAzul;
        LPBYTE Luminancias();

public:
        Cdec(LPBYTE,DWORD,DWORD,DWORD,DWORD,DWORD);
        ~Cdec();
        void DWT2(double **,DWORD,DWORD);
        double** GetMatrix(char);
        void IDWT2(double **,DWORD,DWORD);
        void IniciaProceso(char*);
        char* ModImg(LPBYTE,char *,unsigned short);
        void Tiling(double **, Watermarker);
};
```

Cdec.cpp

```
#include "Cdec.h"
#include "BmpIm.h"
#include "stdafx.h"
```

```
/*****************************************************************************
 * Constructor de la clase Cdec                                              *
 *****************************************************************************/
Cdec::Cdec(LPBYTE valsLum,
                DWORD alto,
                DWORD ancho,
                DWORD bits,
                DWORD alt,
                DWORD ant)


{
      unsigned long i,j,k=0;

      AlIm = alto;
      AnIm = ancho;
      DCLevel = pow(2,7);                        // DCLevel se resta a los valores
de valsLum para obtener
                                                       // el cambio a nivel
de DC que el Codec JPEG2000 espera
                                                       // encontrar.

      MLGris = NULL;
      MLRojo = NULL;
      MLVerde = NULL;
      MLAzul = NULL;


      // Inicializa los tiles

      AlTile = alt;
      AnTile = ant;


      /****
       * A continuación los valores de luminancia contenidos en el arreglo valsLum se
       * convierten en una matriz de pixeles de alto x ancho para el caso de las imá-
       * genes en nivel de gris o en tres matrices de las mismas dimensiones que re-
       * presentan las componentes en rojo, verde y azul  para el  caso de  las  imá-
       * genes RGB, esto es, se obtienen las componentes. Por el momento no se sopor-
      * tan imágenes binarias ni con bitdepths mayores a 24
       ****/


      switch(bits){

      case 8:

            componentes = 1;

            MLGris = new double* [AlIm];

            for(i =0; i < AlIm; i++)
                  MLGris[i] = new double[AnIm];

            while(k < AlIm * AnIm){
                  for(i = 0; i < AlIm; i++){
                        for(j = 0; j < AnIm; j++){

                              MLGris[i][j] = valsLum[k] - DCLevel;

                              k++;
```

```
                        }
                }
        }

        break;

case 24:

        componentes = 3;

        MLRojo = new double* [AlIm];
        MLVerde = new double* [AlIm];
        MLAzul = new double* [AlIm];


        for(i =0; i < AlIm; i++){
                MLRojo[i] = new double[AnIm];
                MLVerde[i] = new double[AnIm];
                MLAzul[i] = new double[AnIm];
        }


        while(k < AlIm * AnIm * 3){
                for(i = 0; i < AlIm; i++){
                        for(j = 0; j < AnIm; j++){
                                MLAzul[i][j] = valsLum[k] - DCLevel;
                                MLVerde[i][j] = valsLum[k+1] - DCLevel;
                                MLRojo[i][j] = valsLum[k+2] - DCLevel;
                                k+=3;
                        }
                }
        }


        break;

default:
        SetLastError(EPT_S_INVALID_ENTRY);
        break;
}

}


/*****************************************************************************
 * Destructor de la clase Cdec                                               *
 *****************************************************************************/
Cdec::~Cdec(){

        MLGris = NULL;
        MLRojo = NULL;
        MLVerde = NULL;
        MLAzul = NULL;

}
```

```
/*****************************************************************************
 * void DWT2(double**,DWORD,DWORD)                                           *
 * Realiza la DWT2 de los tiles de la imagen. Recibe como parámetros un apunta *
 * dor al tile, su tamaño en renglones y su tamaño en columnas.  Modifica  los *
 * valores del tile directamente.                                            *
 *****************************************************************************/
void Cdec::DWT2(double **tile,
                DWORD sizeR,
                DWORD sizeC)
{

      const double sqrt_3 = sqrt( 3 );
      const double denom = 4 * sqrt( 2 );
      double h0, h1, h2, h3;                                  // coeficientes de escala
      double g0, g1, g2, g3;                                  // coeficientes wavelet
      unsigned long i,j,k,l,m=0,nivel=0;
      double **ImT;                                           // tile transformado

      h0 = (1 + sqrt_3)/denom;
      h1 = (3 + sqrt_3)/denom;
     h2 = (3 - sqrt_3)/denom;
     h3 = (1 - sqrt_3)/denom;

      g0 =  h3;
    g1 = -h2;
    g2 =  h1;
    g3 = -h0;


      ImT=new double* [sizeR];  // Arreglo temporal de coeficientes transformados

      for(i=0;i<sizeR;i++)
            ImT[i]=new double[sizeC];



      // Según el tamaño del tile se calcula el nivel de la DWT2

      if ((sizeR <= 8)||(sizeC <= 8))
            nivel = 1;
      else if(((sizeR >= 16)&&(sizeR <= 32))||((sizeC >= 16)&&(sizeC <= 32)))
            nivel = 2;
      else if((sizeR > 32)&&(sizeC > 32))
            nivel = 3;



      while(m < nivel) {

            unsigned long al=(unsigned long)(sizeR/pow(2,m));
            unsigned long an=(unsigned long)(sizeC/pow(2,m));

            unsigned long halfR = al>>1;
            unsigned long halfC = an>>1;

            double *tmp_r;                                    // Arreglo temporal para
los coeficientes renglon transformados
            double *tmp_c;                                    // Arreglo temporal para
los coeficientes columna transformados

            tmp_r = new double[an];  //Ancho
            tmp_c = new double[al];  //Alto
```

```
            // Se transforman los renglones
            for(i=0;i<al;i++){

                  if(m==0){
                        for(k=0,l=0;l<an-3;l+=2,k++){

      tmp_r[k]=tile[i][l]*h0+tile[i][l+1]*h1+tile[i][l+2]*h2+tile[i][l+3]*h3;

      tmp_r[k+halfC]=tile[i][l]*g0+tile[i][l+1]*g1+tile[i][l+2]*g2+tile[i][l+3]*g3;
                        }

                        tmp_r[k]=tile[i][an-2]*h0+tile[i][an-
1]*h1+tile[i][0]*h2+tile[i][1]*h3;
                        tmp_r[k+halfC]=tile[i][an-2]*g0+tile[i][an-
1]*g1+tile[i][0]*g2+tile[i][1]*g3;
                  }
                  else{
                        for(k=0,l=0;l<an-3;l+=2,k++){

      tmp_r[k]=ImT[i][l]*h0+ImT[i][l+1]*h1+ImT[i][l+2]*h2+ImT[i][l+3]*h3;

      tmp_r[k+halfC]=ImT[i][l]*g0+ImT[i][l+1]*g1+ImT[i][l+2]*g2+ImT[i][l+3]*g3;
                        }

                        tmp_r[k]=ImT[i][an-2]*h0+ImT[i][an-
1]*h1+ImT[i][0]*h2+ImT[i][1]*h3;
                        tmp_r[k+halfC]=ImT[i][an-2]*g0+ImT[i][an-
1]*g1+ImT[i][0]*g2+ImT[i][1]*g3;
                  }

                  for(k=0;k<an;k++)
                        ImT[i][k]=tmp_r[k];

            }

            // Se transforman las columnas
            for(j=0;j<an;j++){

                  for(k=0,l=0;l<al-3;l+=2,k++){

      tmp_c[k]=ImT[l][j]*h0+ImT[l+1][j]*h1+ImT[l+2][j]*h2+ImT[l+3][j]*h3;

      tmp_c[k+halfR]=ImT[l][j]*g0+ImT[l+1][j]*g1+ImT[l+2][j]*g2+ImT[l+3][j]*g3;
                  }

                  tmp_c[k]=ImT[al-2][j]*h0+ImT[al-
1][j]*h1+ImT[0][j]*h2+ImT[1][j]*h3;
                  tmp_c[k+halfR]=ImT[al-2][j]*g0+ImT[al-
1][j]*g1+ImT[0][j]*g2+ImT[1][j]*g3;

                  for(k=0;k<al;k++)
                        ImT[k][j]=tmp_c[k];

            }

            delete [] tmp_r;
            delete [] tmp_c;

            m++;
      }
```

```
	// Se informa el tile original con los coeficientes del tile temporal
transformado

	for(i=0;i<sizeR;i++){
		for(j=0;j<sizeC;j++)
			tile[i][j]=ImT[i][j];
	}

	delete [] ImT;                                          // Se borra el tile
temporal

}



double** Cdec::GetMatrix(char matriz){
	switch (matriz){
	case 'r':
		return MLRojo;
		break;
	case 'v':
		return MLVerde;
		break;
	case 'a':
		return MLAzul;
		break;
	case 'g':
		return MLGris;
		break;
	default:
		return MLGris;
		break;
	}
}



/*****************************************************************************
 * void IDWT2(double**,DWORD,DWORD)                                          *
 * Realiza la IDWT2 de los tiles de la imagen. Recibe como parámetros un apun- *
 * tador al tile y su tamaño en renglones y en columnas. Modifica los  valores *
 * del tile directamente.                                                    *
 *****************************************************************************/
void Cdec::IDWT2(double **tile,
		 DWORD sizeR,
		 DWORD sizeC)
{

	const double sqrt_3 = sqrt( 3 );
	const double denom = 4 * sqrt( 2 );
	double *tmp;
	// Arreglo auxiliar temporal

	unsigned short i,j,k,l,nivel;                                   //
Contadores varios de control

	const double Ih0 = (3 - sqrt_3)/denom;
    const double Ih1 = (3 + sqrt_3)/denom;
    const double Ih2 = (1 + sqrt_3)/denom;
    const double Ih3 = (1 - sqrt_3)/denom;

    const double Ig0 = (1 - sqrt_3)/denom;
```

```
    const double Ig1 = -(1 + sqrt_3)/denom;
    const double Ig2 = (3 + sqrt_3)/denom;
    const double Ig3 = -(3 - sqrt_3)/denom;


      // Según el tamaño del tile se calcula el nivel de la IDWT2

      if ((sizeR <= 8)||(sizeC <= 8))
            nivel = 1;
      else if(((sizeR >= 16)&&(sizeR <= 32))||((sizeC >= 16)&&(sizeC <= 32)))
            nivel = 2;
      else if((sizeR > 32)&&(sizeC > 32))
            nivel = 3;


      while(nivel>0) {

            unsigned long al=(unsigned long)(sizeR/pow(2,nivel-1));
            unsigned long an=(unsigned long)(sizeC/pow(2,nivel-1));

            unsigned long halfR = al>>1;
            unsigned long halfC = an>>1;

            // Se antitransforman los renglones

            tmp = new double[an];  //Ancho
            for(i=0;i<al;i++){

                  const int halfPls1 = halfC + 1;

                  tmp[0]=tile[i][halfC-1]*Ih0 + tile[i][an-1]*Ih1 + tile[i][0]*Ih2 +
tile[i][halfC]*Ih3;
                  tmp[1]=tile[i][halfC-1]*Ig0 + tile[i][an-1]*Ig1 + tile[i][0]*Ig2 +
tile[i][halfC]*Ig3;

                  for(k = 0,l = 2; k < halfC - 1; k++){
                        tmp[l++]=tile[i][k]*Ih0 + tile[i][k+halfC]*Ih1 +
tile[i][k+1]*Ih2 + tile[i][k+halfPls1]*Ih3;
                        tmp[l++]=tile[i][k]*Ig0 + tile[i][k+halfC]*Ig1 +
tile[i][k+1]*Ig2 + tile[i][k+halfPls1]*Ig3;
                  }


                  for(k=0;k<an;k++)
                        tile[i][k]=tmp[k];

            }
            delete [] tmp;

            // Se antitransforman las columnas
            tmp = new double[al];
            for(j=0;j<an;j++){

                  const int halfPls1 = halfR + 1;

                  tmp[0]=tile[halfR-1][j]*Ih0 + tile[al-1][j]*Ih1 + tile[0][j]*Ih2 +
tile[halfR][j]*Ih3;
                  tmp[1]=tile[halfR-1][j]*Ig0 + tile[al-1][j]*Ig1 + tile[0][j]*Ig2 +
tile[halfR][j]*Ig3;

                  for(k = 0, l = 2; k < halfR - 1; k++){
```

```
                        tmp[l++]=tile[k][j]*Ih0 + tile[k+halfR][j]*Ih1 +
tile[k+1][j]*Ih2 + tile[k+halfPls1][j]*Ih3;
                        tmp[l++]=tile[k][j]*Ig0 + tile[k+halfR][j]*Ig1 +
tile[k+1][j]*Ig2 + tile[k+halfPls1][j]*Ig3;
                    }


                for(k=0;k<al;k++)
                        tile[k][j]=tmp[k];

            }

            delete [] tmp;

            nivel--;
        }

}


/*****************************************************************************
 *****************************************************************************/
void Cdec::IniciaProceso(char *llav){

        // Instancia un marcador de la clase Watermarker
        Watermarker wm = Watermarker(llav);

        if (componentes==1)
                Tiling(MLGris,wm);
        else{
                Tiling(MLRojo,wm);
                Tiling(MLVerde,wm);
                Tiling(MLAzul,wm);
        }
}


/*****************************************************************************
 * void Luminancias(DWORD,DWORD,DWORD)                                       *
 * Calculan los valores de luminancia de la imagen marcada, para esto se deter *
 * mina el rango en el que se encuentran los coeficientes marcados de modo que *
 * el menor de és tos se considera como un "0" y el mayor de éstos un "255".   *
 *****************************************************************************/
LPBYTE Cdec::Luminancias(){


        unsigned short i,j;
        unsigned int k=0;


        LPBYTE lumn;                                        // Arreglo lineal de los
pixeles a escribir

        switch(componentes){
        case 1:

                lumn = new BYTE[AlIm*AnIm];

                while(k < AlIm * AnIm){
                        for(i = 0;i < AlIm;i++){
                                for(j = 0;j < AnIm;j++){

                                        lumn[k]=(unsigned char)(MLGris[i][j]+DCLevel);
```

```
                        k++;

                    }
                }
            }

            break;

        case 3:

            lumn = new BYTE[AlIm*AnIm*3];

            while(k < AlIm * AnIm * 3){
                for(i = 0;i < AlIm;i++){
                    for(j = 0;j < AnIm;j++){

                        lumn[k] = (unsigned char)(MLAzul[i][j] + DCLevel);
                        lumn[k+1] = (unsigned char)(MLVerde[i][j] + DCLevel);
                        lumn[k+2] = (unsigned char)(MLRojo[i][j] + DCLevel);
                        k+=3;

                    }
                }
            }

            break;
        default:
            SetLastError(EPT_S_INVALID_ENTRY);
            break;
        }

        return lumn;
}

/*****************************************************************************
 * void ModImg(double**,char*)                                               *
 * Escribe la imagen marcada en un nuevo archivo BMP                         *
 *****************************************************************************/
char* Cdec::ModImg(LPBYTE colormap,char *ruta,unsigned short param3){


        HANDLE hFile;
        unsigned long *lpNumOfBytesWritten = new unsigned long;


        // A continuación se da el formato a la nueva ruta del archivo

        char *rut;
        char marc[10];

        if(param3==0)
            strcpy(marc,"marcadas\\W");
        else
            strcpy(marc,"atacadas\\A");

        short despl= strlen(ruta);
        short cant = 0;

        rut = new char[despl + 10];
        strcpy(rut,"");
```

```
	while(despl > 0){

		if(ruta[despl - 1]=='\\'){

			strcpy(rut,ruta);
			strncpy(rut + despl,marc,10);
			cant = (strlen(ruta) - despl);
			strncpy(rut + despl + 10,ruta + despl,cant + 10);
			break;

		}

		despl--;
	}


	// Formatea la cabecera del nuevo archivo BMP

	LPWORD Cabecera;
	DWORD Tam, Of, TI;

	Cabecera=new WORD[27];

	int cont = 0;
	while (cont < 27){
		Cabecera[cont] = 0;
		cont++;
	}

	if(componentes == 1){
		Of = 54 + 1024;
		TI = AlIm * AnIm;
		Cabecera[23] = 256;
		Cabecera[25] = 256;
	}
	else{
		Of = 54;
		TI = AlIm * AnIm * 3;
		Cabecera[23] = 0;
		Cabecera[25] = 0;
	}

	Tam = Of + TI;

	Cabecera[0] = 0x4d42;                                   // Codigo para
formato BMP
	Cabecera[1] = Tam;                                      // Tamaño total del
archivo. Bytes bajos
	Cabecera[2] = (Tam &= 0xffff0000)>>16;          // Tamaño total del archivo. Bytes
altos
	Cabecera[3] = 0;
	Cabecera[4] = 0;
	Cabecera[5] = Of;                                       // Offset al inicio
de la imagen. Bytes bajos
	Cabecera[6] = (Of &= 0xffff0000)>>16;           // Offset al inicio de la imagen.
Bytes altos
	Cabecera[7] = 40;                                       // Tamaño del header
de información
	Cabecera[9] = AnIm;                                     // Ancho de la imagen
en pixeles
	Cabecera[11] = AlIm;                                    // Alto de la imagen
en pixeles
```

```
	Cabecera[13] = 1;                                          // Planos de color
	Cabecera[14] = componentes * 8;                        // Bits por pixel
(bitdepth)
	Cabecera[15] = 0;
	Cabecera[17] = TI;                                         // Tamaño de la
imagen en bytes. Bytes bajos
	Cabecera[18] = (TI &= 0xffff0000)>>16;        // Tamaño de la imagen en bytes.
Bytes altos



	// Usa la funcion miembro Luminancias() para obtener los pixeles a escribir
	LPBYTE lum = Luminancias();

	if (GetLastError()==EPT_S_INVALID_ENTRY){
		strcpy(rut,"Error en Cdec::Luminancias()");                //
Luminancias() acabó con errores
		return rut;
	}



	/****
	 * A continuación se inician las escrituras al archivo cuyo nombre está dado por
ruta
	 ****/

	// Abre el archivo

	hFile = CreateFile(TEXT(rut),                    // file to open
			GENERIC_WRITE,                   // archivo abierto para
escritura
			  FILE_SHARE_WRITE,    // escritura compartida
			    NULL,                        // seguridad default
			      CREATE_ALWAYS,           // si el archivo existe, lo
sobreescribe
				FILE_ATTRIBUTE_NORMAL, // archivo con atributo
normal
				NULL);


	if (hFile == INVALID_HANDLE_VALUE){              // No se pudo abrir el archivo
para escritura
		strcpy(rut,"No se puede abrir el archivo para escritura");
		return rut;
	}


	SetFilePointer(hFile,0,NULL,FILE_BEGIN);
	WriteFile(hFile,Cabecera,54,lpNumOfBytesWritten,NULL);

	Of = (Cabecera[6]<<16)|Cabecera[5];


	if(componentes == 1){

		SetFilePointer(hFile,54,NULL,FILE_BEGIN);
		WriteFile(hFile,colormap,Of - 54,lpNumOfBytesWritten,NULL);

		SetFilePointer(hFile,Of,NULL,FILE_BEGIN);
		WriteFile(hFile,lum,AlIm * AnIm,lpNumOfBytesWritten,NULL);

	}
```

```
	else{
		SetFilePointer(hFile,Of,NULL,FILE_BEGIN);
		WriteFile(hFile,lum,AlIm * AnIm * 3,lpNumOfBytesWritten,NULL);
	}

	CloseHandle(hFile);

	delete Cabecera;
	delete lpNumOfBytesWritten;

	return rut;

}


/*****************************************************************************
 * void Tiling(double**,DWORD,DWORD)                                         *
 * Separa la imagen en tiles y manda cada tile a transformarse en DWT2       *
 *****************************************************************************/
void Cdec::Tiling(double **componente,Watermarker wm)

{
	unsigned long modAlT,modAnT;		// Variables de ayuda para trabajar con
tiles en la
                                                        // frontera

	unsigned long difR,difC;

	unsigned long renglon, columna,r,c;
	unsigned int conteo=0;
	double **tile;


	if((AlTile<8)||(AlTile>=AlIm))
		AlTile=AlIm;
	else if(AlTile > (AlIm/2))
		AlTile = AlIm / 2;			// Forzamos a por lo menos 2 tiles
de alto

	if((AnTile<8)||(AnTile>=AnIm))
		AnTile=AnIm;
	else if(AnTile > (AnIm/2))
		AnTile = AnIm / 2;			// Forzamos a por lo menos 2 tiles
de ancho


	// Creamos el arreglo que contendrá los valores de luminancia del tile
	tile = new double* [AlTile];

	for(r=0;r<AlTile;r++)
		tile[r]=new double[AnTile];



	modAlT = AlIm % AlTile;
	modAnT = AnIm % AnTile;
	difR = AlIm - modAlT;
	difC = AnIm - modAnT;


	/****
	 * Conociendo el tamaño de los tiles, se realiza la DWT2 de cada tile
	 ****/
```

```
	for(renglon = 0; renglon < difR; renglon += AlTile){
		for(columna = 0; columna < difC; columna += AnTile){


				for(r = 0;r < AlTile;r++){
	// Se forma el tile con los coeficientes de ML
					for(c = 0;c < AnTile;c++){
						tile[r][c]=componente[renglon+r][columna+c];

					}
				}


				DWT2(tile,AlTile,AnTile);
	// Transformada wavelet
				conteo += wm.Watermark(tile,AlTile,AnTile);
	// Inserción de la marca
				IDWT2(tile,AlTile,AnTile);
	// Antitransformada wavelet


				for(r = 0;r < AlTile;r++){
	// Se forma la nueva ML con los coeficientes del tile marcado
					for(c = 0;c < AnTile;c++){
						componente[renglon+r][columna+c] = tile[r][c];

					}
				}


		}
	}

	delete [] tile;

	/****
	 * Hasta ahora se han manejado los tiles interiores a la imagen. A continuación
se manejarán
	 * todos los tiles pertenecientes a una de tres posibles fronteras:
	 * frontera de renglón, frontera de columna y frontera doble
	 ****/

	////////////////////////////// Frontera renglon
////////////////////////////////////

	if(modAlT!=0){

		tile = new double* [modAlT];

		for(r=0;r<modAlT;r++)
			tile[r]=new double[AnTile];

		columna = 0;
		while(columna < difC){


			for(r = 0;r < modAlT;r++){
		// Forma el tile con los coeficientes de ML
				for(c = 0;c < AnTile;c++){
					tile[r][c]=componente[difR+r][columna+c];

				}
```

```
                }

                DWT2(tile,AlTile,AnTile);
                conteo += wm.Watermark(tile,AlTile,AnTile);
                IDWT2(tile,AlTile,AnTile);


                for(r = 0;r < modAlT;r++){
        // Forma la nueva ML con los coeficientes del tile marcado
                        for(c = 0;c < AnTile;c++){
                                componente[difR+r][columna+c] = tile[r][c];

                        }
                }


                columna += AnTile;

        }

        delete [] tile;
}


        ////////////////////////////// Frontera columna
////////////////////////////////////

        if(modAnT!=0){

                tile = new double* [AlTile];

                for(r=0;r<AlTile;r++)
                        tile[r]=new double[modAnT];

                renglon = 0;
                while(renglon < difR){

                        for(r = 0;r < modAlT;r++){
                                for(c = 0;c < AnTile;c++){
                                        tile[r][c]=componente[renglon+r][difC+c];

                                }
                        }

                        DWT2(tile,AlTile,AnTile);
                        conteo += wm.Watermark(tile,AlTile,AnTile);
                        IDWT2(tile,AlTile,AnTile);


                        for(r = 0;r < modAlT;r++){
                                for(c = 0;c < AnTile;c++){
                                        componente[renglon+r][difC+c] = tile[r][c];

                                }
                        }


                        renglon += AlTile;

                }

                delete [] tile;
        }
```

```
        ////////////////////////////// Frontera doble
////////////////////////////////////

        if((modAlT!=0)&&(modAnT!=0)){

                tile = new double* [modAlT];

                for(r=0;r<modAlT;r++)
                        tile[r]=new double[modAnT];

                for(r = 0;r < modAlT;r++){
                        for(c = 0;c < modAnT;c++){
                                tile[r][c]=componente[difR+r][difC+c];
                        }
                }


                DWT2(tile,AlTile,AnTile);
                conteo += wm.Watermark(tile,AlTile,AnTile);
                IDWT2(tile,AlTile,AnTile);


                for(r = 0;r < modAlT;r++){
                        for(c = 0;c < modAnT;c++){
                                componente[difR+r][difC+c] = tile[r][c];
                        }
                }
                delete [] tile;
        }
}
```

Clase Detector

La clase Detector se implementa por medio de los archivos Detector.h y Detector.cpp, en los cuales se definen la clase Detector y la implementación de sus métodos respectivamente.

Detector.h

```
class Detector{
private:
public:
        Detector();
        double Detector::Correlaciona(double*,double*,unsigned int);
        double Detector::Formatea(char *,char *);
private:
        double *GetUmbral(double **,double**,unsigned long,unsigned long);
};
```

Detector.cpp

```
#include <iostream.h>
#include <math.h>
#include "Detector.h"
#include "BmpIm.h"
#include "Cdec.h"
#include "stdafx.h"


/*****************************************************************************
 * Constructor de la clase Detector                                          *
 *****************************************************************************/
Detector::Detector(){

}
```

```
/*****************************************************************************
 * double Detector::Correlaciona(double *,double *,unsigned int)             *
 *****************************************************************************/
double Detector::Correlaciona(double *lineal1,double *lineal2,unsigned int N){

      double SPxy,SSx,SSy;                                  // Variables de análisis
estadístico
      double sxy=0,sx=0,sy=0,sx2=0,sy2=0;
      double r=0;                                                  // Coeficiente de
correlacion cruzada
      unsigned int i;


      for(i = 0;i < N;i++){
            sxy += lineal1[i] * lineal2[i];
            sx  += lineal1[i];
            sy  += lineal2[i];
            sx2 += pow(lineal1[i],2);
            sy2 += pow(lineal2[i],2);
      }

      SPxy = sxy - (sx * sy / N);
      SSx  = sx2 - (pow(sx,2)/N);
      SSy  = sy2 - (pow(sy,2)/N);

      r = SPxy / sqrt(SSx * SSy);

      r = sqrt(1 - (((1 - pow(r,2)) * (N-1))/(N-2)));

      return r;

}


/*****************************************************************************

 *****************************************************************************/
double Detector::Formatea(char *a1,char *a2){

      double **im1,**im2,*arr,T,N;
      unsigned long alto,ancho,bits;
      unsigned char *X,*Y;                                  // Matrices de luminancias
      double r;
      unsigned int k=0,m=0,tamano;
      unsigned int i,j;



      // Se instancia un objeto BmpIm para cada archivo

      // Se llama a SetLastError() y GetLastError() para controlar errores
      // en la instanciación de los objetos BmpIm

      SetLastError(NO_ERROR);
      BmpIm Archivo1 = BmpIm(a1);
      if(GetLastError()!=NO_ERROR) return 6;

      SetLastError(NO_ERROR);
      BmpIm Archivo2 = BmpIm(a2);
      if(GetLastError()!=NO_ERROR) return 6;
```

```
// Y se obtienen sus luminancias

X = Archivo1.GetMapValsLum(0);
Y = Archivo2.GetMapValsLum(0);


// Sólo se hará la detección para archivos del mismo tamaño y componentes de
color
if ((Archivo1.GetAtributo(3) * Archivo1.GetAtributo(4) !=
      Archivo2.GetAtributo(3) * Archivo2.GetAtributo(4))||
      (Archivo1.GetAtributo(6)!=Archivo2.GetAtributo(6))){

      r = 5;
      return r;

}

alto = Archivo1.GetAtributo(3);
ancho = Archivo1.GetAtributo(4);
bits = Archivo1.GetAtributo(6);


// Se usan dos objetos Cdec para obtener las matrices de luminancias
// de los archivos a correlacionar
Cdec cdec1 = Cdec(X,alto,ancho,bits,alto,ancho);
Cdec cdec2 = Cdec(Y,alto,ancho,bits,alto,ancho);


if (bits==8){

      // Dos archivos bmp en nivel de gris
      double *lineal1,*lineal2;

      im1 = cdec1.GetMatrix('g');
      im2 = cdec2.GetMatrix('g');

      cdec1.DWT2(im1,alto,ancho);
      cdec1.DWT2(im2,alto,ancho);


      arr = GetUmbral(im1,im2,alto,ancho);
      T = arr[0];
      N = arr[1];
      tamano = (unsigned int)N;

      lineal1 = new double[tamano];
      lineal2 = new double[tamano];

      while(k<tamano){
            for(i=0;i<alto;i++){
                  for(j=0;j<ancho;j++){
                        if(im1[i][j]>T){
                              lineal1[k]=im1[i][j];
                              k++;
                        }
                  }
            }
      }
      while(m<tamano){
            for(i=0;i<alto;i++){
                  for(j=0;j<ancho;j++){
                        if(im2[i][j]>T){
```

```
                                        lineal2[m]=im2[i][j];
                                        m++;
                                }
                        }
                }
        }


        r = Correlaciona(lineal1,lineal2,tamano);


        //delete [] lineal1;
        //delete [] lineal2;
}
else{
        double temp;
        double *linealr1,*linealr2,*linealv1,*linealv2,*linealal,*lineala2;

        ////////////////////
        im1 = cdec1.GetMatrix('r');
        im2 = cdec2.GetMatrix('r');

        cdec1.DWT2(im1,alto,ancho);
        cdec2.DWT2(im2,alto,ancho);

        arr = GetUmbral(im1,im2,alto,ancho);

        T = arr[0];
        N = arr[1];
        tamano = (unsigned int)N;

        linealr1 = new double[tamano];
        linealr2 = new double[tamano];

        while(k<tamano){
                for(i=0;i<alto;i++){
                        for(j=0;j<ancho;j++){
                                if(im1[i][j]>T){
                                        linealr1[k]=im1[i][j];
                                        k++;
                                }
                        }
                }
        }
        while(m<tamano){
                for(i=0;i<alto;i++){
                        for(j=0;j<ancho;j++){
                                if(im2[i][j]>T){
                                        linealr2[m]=im2[i][j];
                                        m++;
                                }
                        }
                }
        }

        temp = Correlaciona(linealr1,linealr2,tamano);
        //delete lineal1;
        //delete lineal2;
        ////////////////////

        im1 = cdec1.GetMatrix('v');
        im2 = cdec2.GetMatrix('v');
```

```
cdec1.DWT2(im1,alto,ancho);
cdec2.DWT2(im2,alto,ancho);

arr = GetUmbral(im1,im2,alto,ancho);

T = arr[0];
N = arr[1];
tamano = (unsigned int)N;

linealv1 = new double[tamano];
linealv2 = new double[tamano];

k=0;
m=0;
while(k<tamano){
      for(i=0;i<alto;i++){
            for(j=0;j<ancho;j++){
                  if(im1[i][j]>T){
                        linealv1[k]=im1[i][j];
                        k++;
                  }
            }
      }
}
while(m<tamano){
      for(i=0;i<alto;i++){
            for(j=0;j<ancho;j++){
                  if(im2[i][j]>T){
                        linealv2[m]=im2[i][j];
                        m++;
                  }
            }
      }
}
temp += Correlaciona(linealv1,linealv2,tamano);
//delete lineal1;
//delete lineal2;
/////////////////

im1 = cdec1.GetMatrix('a');
im2 = cdec2.GetMatrix('a');

cdec1.DWT2(im1,alto,ancho);
cdec1.DWT2(im2,alto,ancho);

arr = GetUmbral(im1,im2,alto,ancho);

T = arr[0];
N = arr[1];
tamano = (unsigned int)N;

lineala1 = new double[tamano];
lineala2 = new double[tamano];

k=0;
m=0;
while(k<tamano){
      for(i=0;i<alto;i++){
            for(j=0;j<ancho;j++){
                  if(im1[i][j]>T){
                        lineala1[k]=im1[i][j];
                        k++;
                  }
```

```
                        }
                    }
                }
                while(m<tamano){
                    for(i=0;i<alto;i++){
                        for(j=0;j<ancho;j++){
                            if(im2[i][j]>T){
                                lineala2[m]=im2[i][j];
                                m++;
                            }
                        }
                    }
                }
                temp += Correlaciona(lineala1,lineala2,tamano);
                //delete linea11;
                //delete linea12;
                ////////////////////

                r = temp/3;
        }

        return r;
}


/*****************************************************************************

 *****************************************************************************/
double* Detector::GetUmbral(double **im1,
                                                    double **im2,
                                                    unsigned long alto,
                                                    unsigned long ancho)
{
        double media=0, sumaSqr=0, std=0, T=0,*arr;
        unsigned long N=0,M=0;
        unsigned int i,j;

        arr = new double[2];

        for(i=0;i<alto;i++){
                for(j=0;j<ancho;j++){
                        media += im1[i][j]/(alto*ancho);
                        sumaSqr += pow(im1[i][j],2)/(alto*ancho);
                }
        }

        std=sqrt(sumaSqr-pow(media,2));
        T=abs(media+std/2);

        for(i=0;i<alto;i++){
                for(j=0;j<ancho;j++){
                        if(im1[i][j]>T)
                                N++;
                        if(im2[i][j]>T)
                                M++;
                }
        }

        (N <= M) ? M=N : N=M;

        arr[0] = T;
        arr[1] = N;
```

```
	return arr;

}
```

Clase Watermarker

La clase Watermarker se implementa por medio de los archivos Watermarker.h y Watermarker.cpp, en los cuales se definen la clase Watermarker y la implementación de sus métodos respectivamente.

Watermarker.h

```
class Watermarker{
private:
	unsigned short *llaveNum;
	char *llave;
public:
	Watermarker(char *);
	unsigned int Watermark(double**,DWORD,DWORD);
};
```

Watermarker.cpp

```
#include <windows.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "Watermarker.h"
#include "stdafx.h"


/*****************************************************************************
 * Constructor de la clase Watermarker                                       *
 *****************************************************************************/
Watermarker::Watermarker(char *key){

	unsigned int i;

	llave = key;

	llaveNum= new unsigned short[strlen(llave)];

	for(i=0;i<strlen(llave);i++)
		llaveNum[i]=(unsigned short)llave[i];
}


/*****************************************************************************
 * void Watermark (double **Im)                                              *
 * Inserta la marca de agua en los coeficientes wavelet de 1, 2 y 3 nivel    *
 *****************************************************************************/
unsigned int Watermarker::Watermark(double **Im,DWORD alto, DWORD ancho){

	unsigned short i,j,offKey,m,cota;
	int numcoefs=0;
	long double media,std,sumaSqr,T1,T2,T3;
	short al,an;


	// Dependiendo el tamaño del tile a marcar se calcula el nivel de
transformaciones
	// wavelet a las que fue sometido, y en consecuencia, el nivel de la marca

	if ((alto <= 8)||(ancho <= 8))
		m = 1;
	else if(((alto >= 16)&&(alto <= 32))||((ancho >= 16)&&(ancho <= 32)))
```

```
        m = 2;
else if((alto > 32)&&(ancho > 32))
        m = 3;


while(m>0){

        al = (short)(alto/pow(2,m));
        an = (short)(ancho/pow(2,m));

        media=0;
        sumaSqr=0;
        for(i=0;i<al;i++){
                for(j=an;j<an*2;j++){
                        media += Im[i][j]/(al*an);
                        sumaSqr += pow(Im[i][j],2)/(al*an);
                }
        }

        std=sqrt(sumaSqr-pow(media,2));

        T1=abs(media+std/2);


        media=0;
        sumaSqr=0;
        for(i=al;i<al*2;i++){
                for(j=0;j<an;j++){
                        media+=Im[i][j]/(al*an);
                        sumaSqr+=pow(Im[i][j],2)/(al*an);
                }
        }


        std=sqrt(sumaSqr-pow(media,2));

        T2=abs(media+std/2);


        media=0;
        sumaSqr=0;
        for(i=al;i<al*2;i++){
                for(j=an;j<an*2;j++){
                        media += Im[i][j]/(al*an);
                        sumaSqr += pow(Im[i][j],2)/(al*an);
                }
        }

        std=sqrt(sumaSqr-pow(media,2));


        T3=abs(media+std/2);

        ////////////////////////////////////////////////////////////////////////////
        cota = strlen(llave)-1;
        offKey=0;
        for(i=0;i<al;i++){
                for(j=an;j<an*2;j++){
                        if((i>0 && i<al-2) && (j>an && j<an*2-2)){
                                if(Im[i][j]>T1){
                                        srand(llaveNum[offKey]);
```

```
                                        media=(Im[i-1][j-1]+Im[i-1][j]+Im[i-
1][j+1]+Im[i][j-1]+
                                                Im[i][j+1]+Im[i+1][j-
1]+Im[i+1][j]+Im[i+1][j+1])/8;
                                        Im[i][j]+=0;
                                        Im[i][j]+=.079*media*(rand()/0x7fff);
                                        numcoefs++;
                                        offKey++;
                                }
                                if (offKey==cota) offKey=0;
                        }
                }
        }

        offKey=0;
        for(i=al;i<al*2;i++){
                for(j=0;j<an;j++){
                        if((i>al && i<al*2-2) && (j>0 && j<an-2)){
                                if(Im[i][j]>T2){
                                        srand(llaveNum[offKey]);
                                        media=(Im[i-1][j-1]+Im[i-1][j]+Im[i-
1][j+1]+Im[i][j-1]+
                                                Im[i][j+1]+Im[i+1][j-
1]+Im[i+1][j]+Im[i+1][j+1])/8;
                                        Im[i][j]+=0;
                                        Im[i][j]+=.079*media*(rand()/0x7fff);
                                        numcoefs++;
                                        offKey++;
                                }
                                if (offKey==cota) offKey=0;
                        }
                }
        }

        offKey=0;
        for(i=al;i<al*2;i++){
                for(j=an;j<an*2;j++){
                        if((i>al && i<al*2-2) && (j>an && j<an*2-2)){
                                if(Im[i][j]>T3){
                                        srand(llaveNum[offKey]);
                                        media=(Im[i-1][j-1]+Im[i-1][j]+Im[i-
1][j+1]+Im[i][j-1]+
                                                Im[i][j+1]+Im[i+1][j-
1]+Im[i+1][j]+Im[i+1][j+1])/8;
                                        Im[i][j]+=0;
                                        Im[i][j]+=.079*media*(rand()/0x7fff);
                                        numcoefs++;
                                        offKey++;
                                }
                                if (offKey==cota) offKey=0;
                        }
                }
        }
        ////////////////////////////////////////////////////////////////////////

        m--;
    }

    return numcoefs;

}
```

Componentes de Watermarking.exe

La aplicación Watermarking.exe es demasiado sencilla como para que sea ilustrativo incluir aquí su código fuente: constituye el "cascarón" del sistema de marcas de agua; una interfaz de usuario amigable.

No obstante, existen algunos módulos de ésta que sí son de importancia en este anexo pues constituyen la interfaz tanto a la API de MS Windows como a la biblioteca mrkrdll.dll, por esta razón se incluirá el código realiza lo anterior.

MDIFrmMain.frm

Es la ventana principal que alberga a todas las demás ventanas y cuadros de diálogo de la aplicación. Hace uso de los módulos de información y detección por medio de llamadas a la biblioteca mrkrdll.dll.

Llamadas de MDIFrmMain

```
#If (DevMode) Then      'Path explícito
Private Declare Function InstImBMP Lib _
"C:\Documents and Settings\enroiv\Mis
documentos\Stuff\progs\CProgs\Tesis\mrkrdll\Debug\mrkrdll.dll" _
(ByVal ruta As String, ByRef datos As Byte) As Long

Private Declare Function DoDetect Lib _
"C:\Documents and Settings\enroiv\Mis
documentos\Stuff\progs\CProgs\Tesis\mrkrdll\Debug\mrkrdll.dll" _
(ByVal c1 As String, ByVal c2 As String) As Double

#Else                   'Path relativo

Private Declare Function InstImBMP Lib "mrkrdll.dll" _
(ByVal ruta As String, ByRef datos As Byte) As Long

Private Declare Function DoDetect Lib "mrkrdll.dll" _
(ByVal c1 As String, ByVal c2 As String) As Double

#End If
```

frmAttacks.frm

Es el cuadro de dialogo que pregunta el soporte del filtro a emplear para realizar el ataque. Hace uso del módulo de ataques por medio de llamadas a la biblioteca mrkrdll.dll.

Llamadas de frmAttacks

```
#If (DevMode) Then      'Path explícito
Private Declare Function Ataques Lib _
"C:\Documents and Settings\enroiv\Mis
documentos\Stuff\progs\CProgs\Tesis\mrkrdll\Debug\mrkrdll.dll" _
(ByVal ruta As String, ByRef rutaAtacada As Byte, ByVal soporte As String, ByVal
tipoFiltro As String) As Long
```

```
#Else                    'Path relativo
Private Declare Function Ataques Lib "mrkrdll.dll" _
(ByVal ruta As String, ByRef rutaAtacada As Byte, ByVal soporte As String, ByVal
tipoFiltro As String) As Long
#End If
```

frmParams.frm

Es el cuadro de dialogo que pregunta las dimensiones del tile y la llave de marcado. Hace uso del módulo de marcado por medio de llamadas a la biblioteca mrkrdll.dll.

Llamadas de frmParams

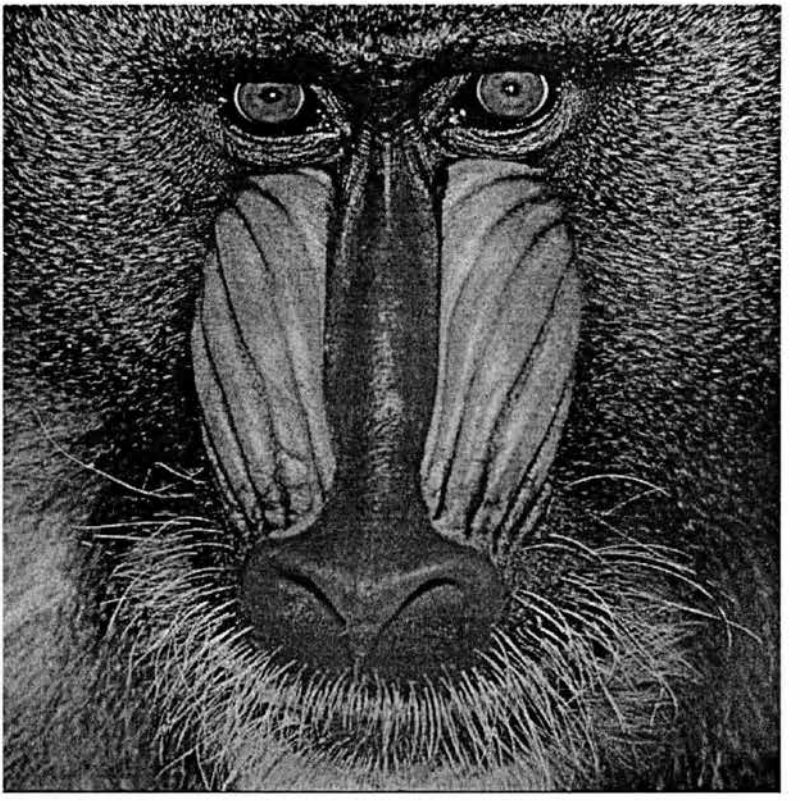
```
#If (DevMode) Then       'Path explícito

Private Declare Function Marca Lib _
"C:\Documents and Settings\enroiv\Mis
documentos\Stuff\progs\CProgs\Tesis\mrkrdll\Debug\mrkrdll.dll" _
(ByVal ruta As String, rutaMarcada As Byte, ByVal llave As String, ByVal alt As
String, ByVal ant As String) As Long

#Else                    'Path relativo

Private Declare Function Marca Lib "mrkrdll.dll" _
(ByVal ruta As String, ByRef rutaMarcada As Byte, ByVal llave As String, ByVal alt As
String, ByVal ant As String) As Long

#End If
```

Anexo 2
Imágenes de prueba e imágenes marcadas

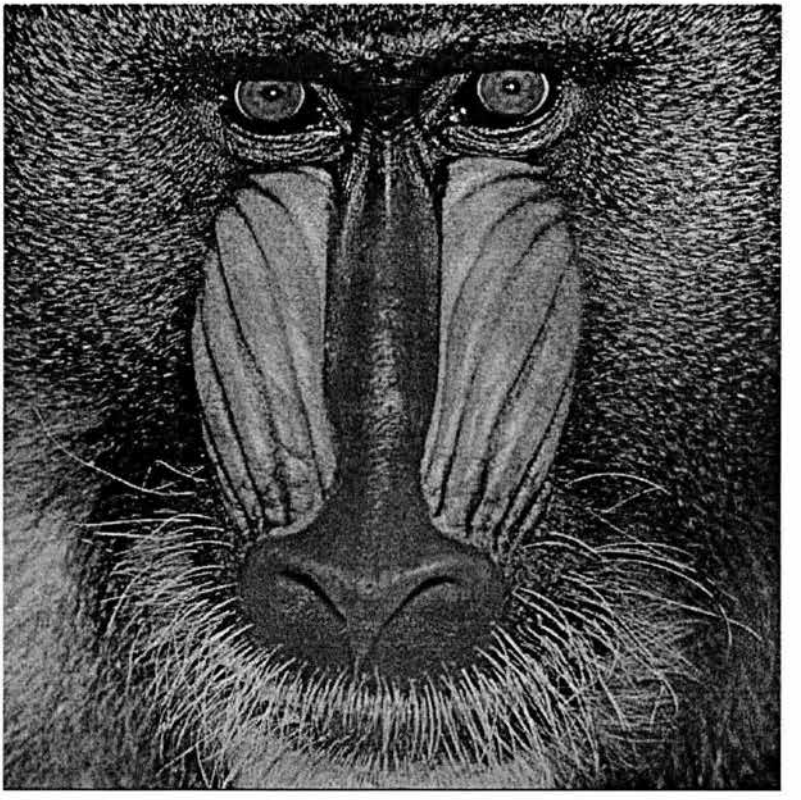
Aerial.bmp

WAerial.bmp

Baboon.bmp



WBaboon.bmp



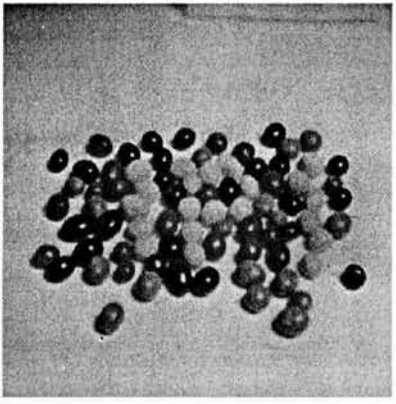
Boat.bmp

WBoat.bmp

Ivanka2.bmp

WIvanka2.bmp

Jellybeans.bmp



WJellybeans.bmp