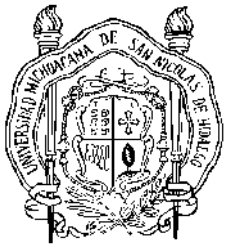


UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

DIVISIÓN DE ESTUDIOS DE POSGRADO DE LA FACULTAD DE INGENIERÍA ELÉCTRICA

RESTAURACIÓN DE IMÁGENES DIGITALES MEDIANTE UNA MARCA DE AGUA BASADA EN LA TRANSFORMADA WAVELET DISCRETA

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

Maestro en Ciencias en Ingeniería Eléctrica

PRESENTA:

Larissa Lizbeth Malváez Álvarez

DIRECTOR DE TESIS:

Dr. Félix Calderón Solorio

CO-DIRECTOR DE TESIS:

Dr. Juan José Flores Romero

Morelia, Michoacán. Noviembre de 2020.

Resumen

Hoy en día, las imágenes digitales son uno de los medios de expresión más comunes y se propagan rápida y ampliamente a través de internet. Este tipo de contenido digital puede ser copiado y modificado fácilmente por personas no autorizadas, lo cual puede llegar a causar problemas de violación a los derechos de autor o incluso tener implicaciones legales. Las marcas de agua digitales surgen como una solución para la autenticación de los pixeles de la imagen y algunas incluso permiten restaurar el contenido manipulado de la imagen.
En esta tesis se presenta un algoritmo que inserta una marca de agua frágil en una imagen en escala de grises y permite verificar el contenido de la imagen, en caso de que haya sido manipulado, logra restaurar una versión comprimida de la imagen original. Este esquema divide la imagen en bloques de 8×8 y comprime la imagen por bloques utilizando una sub-banda de la Transformada Wavelet Discreta y a partir de la información de la imagen comprimida se generan dos conjuntos distintos de recuperación, esto da lugar a que se genere redundancia. La marca de agua se conforma de los dos conjuntos de recuperación generados y un hash criptográfico, este último es el que permite saber si la imagen fue manipulada o no. Además, dicho esquema tiene como parámetro el nivel de compresión, el cual toma valores en $\{1, 2, 3\}$, que son los niveles de la Transformada Wavelet en los que se puede comprimir un bloque. El nivel de compresión de la imagen determina dos aspectos importantes de la marca, uno es el porcentaje máximo capaz de restaurar, esto debido a que entre más comprimida esté la imagen, más copias de la misma es posible insertar en la marca y, por lo tanto, mayor es el porcentaje de restauración; el segundo aspecto es la calidad de la imagen restaurada, ya que a mayor compresión, se obtiene menor calidad al descomprimirla. Por lo tanto, cada uno de los tres niveles ofrece distinto porcentaje máximo de restauración y distinta calidad en la imagen recuperada, de tal manera que sea el usuario quien determine el nivel que es adecuado para su aplicación. En el primer nivel, se logra recuperar una imagen que esté manipulada a lo más en un 64 % del total de los pixeles que la conforman; en el segundo nivel, se recupera hasta un 86 % y en el tercer nivel se restaura hasta el 95 %. La contribución de este trabajo es que la marca de agua propuesta ha demostrado tener mejor desempeño respecto a los otros esquemas del estado del arte revisado, los cuales reportan porcentajes de restauración menores al 90 %.

Palabras clave: compresión, información, recuperación, autenticación de imágenes.

Abstract

Nowadays, digital images are one of the most common ways of expression, propagating rapidly and throughout the web. This type of digital content can be easily copied and modified by non authorized users, which can lead to violating copyrights or even legal repercussions. Digital watermarks come as a solution for authenticating the pixels of an image, some of them even allowing to restore the tampered content of the image.

This thesis presents an algorithm that inserts a fragile watermark in a gray-scale image, allowing to verify the content of the image and, in case of tampering, restoring a compressed version of the original image. This scheme divides the image in 8×8 blocks and compresses the image by blocks utilizing a Discrete Wavelet Transform sub-band, generating two different recovery sets from the compressed image information, giving place to redundancy. The watermark is made from the two recovery sets and a cryptographic hash, the latter being responsible for telling if the image was tampered or not. Additionally, the aforementioned scheme has a compression level as a parameter, taking values in $\{1, 2, 3\}$, corresponding to the levels of the Wavelet Transform in which a block can be compressed. The image compression level determines two important aspects of the watermark, one is the maximum restoration percentage, because having a more compressed image means that more copies of it can fit in the watermark, which makes the restoration percentage bigger; the second aspect is the quality of the restored image, because with greater compression comes lesser quality when decompressing. Ergo, each of the three levels offers a different maximum restoration percentage and quality of the restored image, being the user the one that determines which one of them is the right one for their application. In the first level, it is possible to restore an image that has at most 64 % of its total pixeles modified; in the second level, it is possible to restore up to 86 %, and in the third level it is possible to restore up to 95 %. The contribution of this work is that the proposed watermarking scheme has shown to have a better performance compared to other state of the art schemes, that have less than 90 % restoring percentages.

Key words: compression, information, recovery, image authentication.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Acrónimos y Símbolos

DWT	Transformada wavelet discreta.
IDWT	Transformada wavelet inversa discreta.
DCT	Transformada coseno discreta.
IDCT	Transformada coseno inversa discreta.
DFT	Transformada discreta de Fourier.
MSE	Error cuadrático medio.
PSNR	Valor pico de la relación señal-ruido.
MSB	Bits más significativos.
LSB	Bits menos significativos.
RGB	Rojo Verde Azul.
SDV	Descomposición en valores singulares.
NSCT	Transformada contourlet no submuestreada.
SDV	Descomposición en valores singulares.
dB	Decibel.
2D	Bidimensional.
SHA	Algoritmo seguro de hash.
α	Porcentaje de bits manipulados.
I	Imagen original.
I_W	Imagen marcada.
I_T	Imagen manipulada.
I_R	Imagen restaurada.
I_C	Imagen comprimida.
I_D	Imagen descomprimida.
d	Vector de datos.
c	Vector de códigos de paridad.
A	Matriz binaria de incidencia.
B_k	Bloque de 8×8 pixeles.
N_B	Número total del bloques de la imagen.
L	Tamaño de los segmentos en que se divide el vector d.
Q_s	Número de segmentos de tamaño L.

k	Proporción entre los renglones y columnas de A.
$Permutar(., llave)$	Función de permutación pseudo-aleatoria.
$Permutar^{-1}(., llave)$	Inversa de la función de permutación pseudo-aleatoria.
$Hash32(.)$	Función hash de 32 bits.
$ToBin(.)$	Función para convertir un número entero a binario.
$ToInt(.)$	Función para redondear un número a entero.
$BinToInt(.)$	Función para convertir un número binario a entero.
$5MSB(.)$	Función para obtener los 5 MSB de un arreglo.
$3LSB(.)$	Función para obtener los 3 LSB de un arreglo.
$[x\|y]$	Concatenación de vectores.
$v[n:m]$	Segmento del vector v que va desde el elemento n hasta el $m-1$.
$M[n:m, k:l]$	Submatriz de M del renglón n al $m-1$ y de la columna k al $l-1$.
$LL^{(s)}$	Sub-banda de la transformada wavelet de nivel s.

Capítulo 1

Introducción

Con el fácil acceso que se tiene hoy en día a las imágenes digitales y la gran cantidad de herramientas que existen para su edición, la creación y reproducción de este contenido se ha hecho cada vez más sencilla y frecuente, por tal motivo, el desarrollo técnicas de protección de contenido y de derechos de autor juega un papel muy importante en el campo de la computación. Las marcas de agua se utilizan para proteger la integridad y autenticidad del documento o imagen digital que la contiene. Éstas han ido cambiando a lo largo del tiempo, en la siguiente sección se habla brevemente de la evolución de las marcas de agua.

1.1. Antecedentes

El termino *marca de agua* se deriva de la historia de los fabricantes de papel de la Edad Media. La marca de agua en papel más antigua data del año 1292 y fue encontrada en Italia [Bloom, 1999]. La marca de agua era creada introduciendo un alambre con cierto patrón en los moldes del papel, esto creaba una diferencia de grosor y, con ello, una transparencia, en ese tiempo las marcas eran usadas para identificar la procedencia del papel [Yusof y Khalifa, 2007]. En la Figura 1.1 se muestra un ejemplo de una marca de agua en papel antiguo, vista a contraluz.

Figura 1.1: Marca de agua en papel del año 1721.

Una vez mejorada y perfeccionada la técnica, en el siglo XIX, se transforman en las marcas de agua sombreadas, que vistas a contraluz dejan ver imágenes con relieve e incluso varias tonalidades. Hoy en día, las marcas de agua en papel se utilizan para evitar la falsificación de documentos oficiales y para demostrar su autenticidad, por ejemplo, en los billetes. En la Figura 1.2, se puede apreciar dentro del recuadro rojo la marca de agua en un billete, la cual sólo puede ser vista a contraluz.

Figura 1.2: Marca de agua en billete 20 euros.

Más tarde, con el desarrollo de los sistemas de cómputo y el inicio de la era digital, se generalizó el uso de marcas de agua para señales digitales, tales como: audio, video e imágenes. El término de marca de agua digital apareció por primera vez en 1993, cuando Tirker *et al.* presentaron una técnica de marcado en imágenes [Tirkel *et al.*, 1993]. En particular, en esta tesis nos vamos a centrar únicamente en marcas de agua en imágenes. Para entender cómo funciona una marca de agua digital, primero debe quedar clara la representación de una imagen digital.

1.2. Imágenes Digitales

Una imagen digital I se representa mediante un arreglo bidimensional de tamaño $N \times M$, donde N son los renglones y M las columnas. A cada elemento del arreglo se le denomina pixel y se denota por $I[i, j]$ donde $0 \leq i \leq N-1$ y $0 \leq j \leq M-1$, siendo el pixel la unidad básica de una imagen digital. Cada pixel contiene información, dependiendo del modelo de color que se utilice. Algunos modelos de color son: binario, escala de grises, RGB, CMYK, HSL, HSV, NCS, entre otros. Por ejemplo, en una imagen en escala de grises, el 0 es el color negro y 255 representa el color blanco, entonces cualquier tono de gris es un número entre 0 y 255. En la Figura 1.3 se muestran tres imágenes: la de arriba es una fotografía de Albert Einstein en tonos de grises, la imagen de la izquierda es un fragmento cuadrado de tamaño 12×12 tomado de la fotografía y escalado; a la derecha se pueden observar los valores numéricos que corresponden al tono de gris del fragmento escalado de la fotografía.

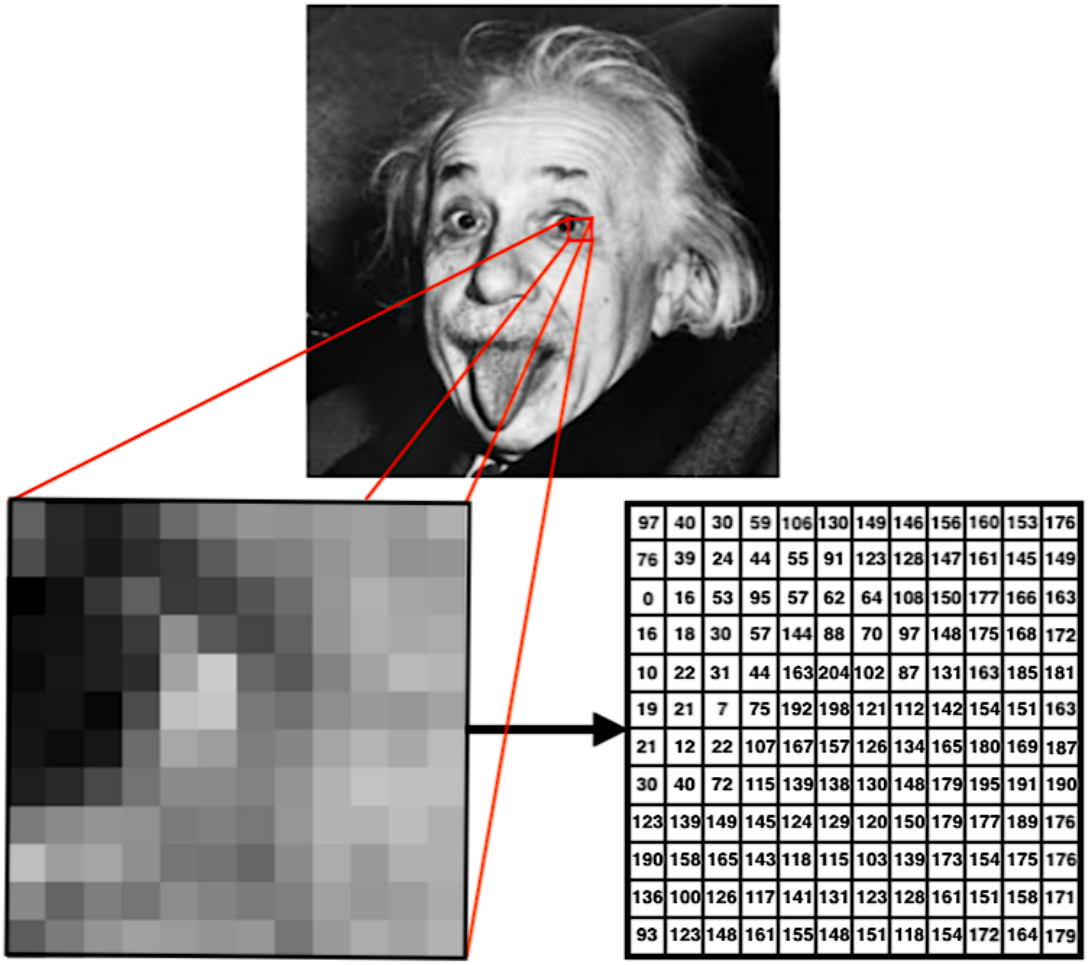


Figura 1.3: Imagen en escala de grises y su representación.

En el modelo RGB (por sus siglas en inglés: Red Green Blue) se tienen tres canales: el rojo (R), el verde (G) y el azul (B); en este modelo cada pixel tiene tres números entre 0 y 255 que refieren a la cantidad de cada color. Convertir una imagen RGB a una imagen en escala de grises es sencillo, basta con realizar el promedio (el cual puede ser ponderado) de los valores de los tres colores y eso asignarlo como el valor del pixel. En esta tesis se trabaja únicamente con imágenes en escala de grises por simplicidad, puesto que para una imagen en RGB bastaría con aplicarle el procedimiento a cada uno de los canales por separado.

Como cada pixel $I[i,j] \in [0,255]$, entonces en notación binaria puede ser representado por 8 bits, $b_7[i,j], \ldots, b_0[i,j]$, donde:

$$b_l[i,j] = \left\lfloor \frac{I[i,j]}{2^l} \right\rfloor \quad (\text{mód } 2), \quad l = 0, \ldots, 7. \tag{1.1}$$

Entonces, los bits $b_7[i,j], b_6[i,j], b_5[i,j], b_4[i,j], b_3[i,j]$ son los bits más significativos y se denotan por MSB (del inglés, More Significant Bits), los bits $b_2[i,j], b_1[i,j]$ y $b_0[i,j]$ corresponden a los bits menos significativos y se denotan como LSB (del inglés, Less Significant Bits).

1.3. Marcas de Agua

Una marca de agua digital es un patrón de bits que se inserta en la imagen, este patrón de bits puede ser otra imagen, números pseudoaleatorios generados por una llave privada, la misma imagen comprimida, un mensaje secreto, etc. De acuerdo con el propósito que se le de, las marcas de agua pueden ser visibles o invisibles:

- **Visibles**: se refiere a que el ojo humano es capaz de apreciar la marca a simple vista, la marca suele ser una imagen secundaria traslúcida sobrepuesta a la imagen original, tal como el logo de una institución.

- **Invisibles**: es aquella que es imperceptible al ojo humano. Para este tipo de marcas se busca que al introducirla no degrade a la imagen que se está marcando.

Las marcas de agua pueden ser clasificadas en: robustas, frágiles y semi-frágiles, según su resistencia a los ataques, entendiendo por ataque cualquier procesamiento de la imagen. Dichos ataques pueden incluir: compresión, rotación, traslación, escalamiento, cortado, adición de ruido, etc.

- **Robusta**: la marca de agua permanece intacta aún después de ciertas alteraciones y degradaciones de la imagen. Este tipo de marca de agua puede ser usada para proteger los derechos de autor.

- **Frágil**: está diseñada para modificarse o destruirse ante cualquier alteración de la imagen que la contenga. Este tipo de marcado suele ser utilizado para la detección del contenido manipulado.

- **Semi-frágil**: la marca de agua tiene robustez ante cierto conjunto de ataques, pero es vulnerable (frágil) ante otros. Este marcado es comúnmente empleado para detectar ataques maliciosos, la marca es robusta ante ataques que preserven el contenido de la imagen (como compresión JPEG), pero frágil ante ataques que alteren el contenido de la imagen.

Basados en lo que requiere el sistema de marcado durante la etapa de extracción y/o detección, las marcas se pueden clasificar como:

- **Ciega**: este tipo de marca no requiere a la imagen original y ninguna información acerca de ésta para detectar la marca.

- **No ciega**: precisa de la imagen original para realizar el proceso de detección de la marca.

- **Semi-ciega**: necesita la llave secreta y la imagen marcada para detectar la marca.

Finalmente, según el dominio en el que el algoritmo realice el proceso de inserción y extracción, se clasifican en:

- **Marca en el dominio espacial**: es cuando la marca de agua se inserta modificando directamente los valores de los pixeles de la imagen original.

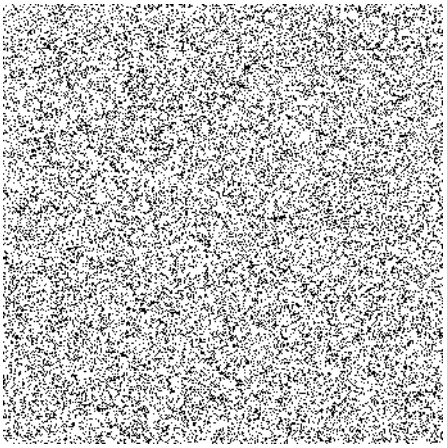
- **Marca en el dominio transformado**: la marca es introducida modificando los coeficientes de la transformada de la imagen. Algunas de las transformadas más utilizadas son: las Transformadas Discretas Coseno [Nilchi y Taheri, 2008], Wavelet [Joo *et al.*, 2002], de Fourier [Poljicak *et al.*, 2011, Solachidis y Pitas, 1999], entre otras.

Un ejemplo sencillo de un esquema de marcado en el dominio espacial, es reemplazar los 3 LSB de la imagen por cualquier información que permita autentificarla, mientras que los 5 MSB de la imagen se mantienen intactos. Como los 3 LSB son sensibles a cualquier tipo de procesamiento de la imagen, este esquema de marcado se considera frágil y es muy útil para la detección de contenido manipulado. Se ha demostrado que si se insertan patrones pseudo-aleatorios en los 3 LSB, la marca de agua se vuelve imperceptible (ver Figura 1.4(d)) [Zhang *et al.*, 2011c]. La ecuación (1.2) describe este esquema de marcado, en donde dada una imagen I de dimensiones $N \times M$ y una imagen secundaria W (la marca de agua) de las mismas dimensiones, tal que $W[i,j] \in [0,7]$ con $0 \leq i \leq N-1$ y $0 \leq j \leq M-1$, la imagen W es introducida en los 3 LSB de la imagen I, mientras los otros 5 MSB de I permanecen intactos. Sea I_w una imagen de tamaño $N \times M$ que guarda el resultado de marcar la imagen I, es decir, I_w es la imagen marcada y se calcula como sigue:

$$I_w[i,j] = 8 \left\lfloor \frac{I[i,j]}{8} \right\rfloor + W[i,j] \quad \forall [i,j] \in N_r \times N_c \tag{1.2}$$

donde $N_r = \{0, \ldots, N-1\}$ y $N_c = \{0, \ldots, M-1\}$. Para saber si la imagen marcada I_w ha sido modificada, se comparan los 3 LSB de I_w con la marca de agua W, si no son iguales, entonces la imagen fue alterada. El proceso de detección descrito no requiere de la imagen original, por lo tanto, se puede considerar que es una marca de agua ciega. En las Figuras 1.4(c) y 1.4(d), se utilizó (1.2) para marcar la Figura 1.4(e). En la Figura 1.4(c), se muestra una imagen marcada con el logo de la UMSNH (Figura 1.4(a)), este es un ejemplo de una marca de agua visible, puesto que es posible apreciar el logo. En cambio, en la Figura 1.4(d), se presenta una marca de agua invisible, lo que se hizo fue permutar los pixeles del logo de la UMSNH de manera pseudo-aleatoria, utilizando una semilla y el resultado se insertó como marca de agua, el logo permutado se puede ver en la Figura 1.4(b).

(a) ***Marca de agua.***



(b) ***Marca de agua pseudo-aleatoria.***

(c) ***Marca de agua visible.***

(d) ***Marca de agua invisible.***

(e) ***Imagen Original.***

Figura 1.4: Ejemplos de marcas de agua visibles e invisibles.

Como alguna porción del contenido podría ser modificado y reemplazado con información falsa (ya sea de manera maliciosa o no) o incluso debido a una mala transmisión de datos, se han desarrollado diversos esquemas de marcado que, además de verificar si ha sido manipulado o no, permiten recuperar el contenido original, siempre y cuando la imagen haya sido manipulada hasta cierto porcentaje establecido por cada método de marcado. En estos esquemas, la marca de agua consiste en introducir una representación de la propia imagen, de tal manera que la marca brinde información de la imagen original, aún cuando una porción del contenido ha sido manipulado. Dicha representación puede ser la información comprimida de los 5 MSB de la imagen, en este caso, se dice que es de recuperación *aproximada*. En cambio, si se restaura de manera directa la información de los 5 MSB de la imagen, se denomina de recuperación *exacta*. En el capítulo 2, se explica más a detalle esta clasificación y se citan algunos ejemplos que se encuentran en la literatura.

1.4. Planteamiento del problema

Una de las aplicaciones más importantes de las marcas de agua digitales es el marcado de imágenes tomadas por una cámara de seguridad. Como alguna(s) de las imágenes captadas por la cámara podrían contener información que tenga implicaciones legales, es decir, que comprueben la inocencia o culpabilidad ante algún acto ilícito, es de suma importancia que se pueda verificar la integridad de las imágenes y, más aún, que si fueron modificadas, sea posible restaurar el contenido de la imagen original. Considere la siguiente situación hipotética: en una empresa se ha cometido un robo, se sabe el día y la hora aproximada en que fue cometido este acto ilícito, sin embargo, al revisar las imágenes tomadas por la cámara de seguridad, descubren que no hay evidencia del robo.

Figura 1.5: Imagen que se quiere verificar.

En la Figura 1.5 se ejemplifica lo que podría ser la imagen captada por la cámara de seguridad de la entrada a la presunta hora del robo, en dicha imagen se observa que no hay movimiento alguno de personas o vehículos. Afortunadamente, dicha empresa tiene la precaución de marcar las fotografías captadas por las cámaras de seguridad y deciden corroborar la integridad de la imagen tomada a la hora aproximada del robo. Utilizando el algoritmo de marcado, descubren que la imagen fue alterada en las áreas que se muestran de color negro en la Figura 1.6.

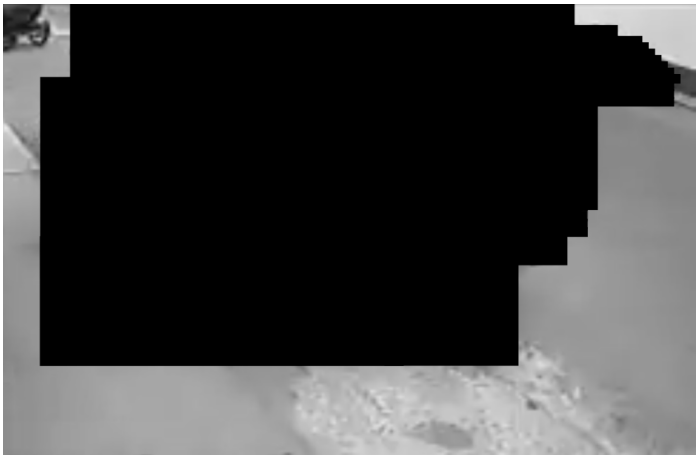


Figura 1.6: Área manipulada de la imagen.

La empresa sabe que su algoritmo de marcado además de verificar la integridad de las imágenes, les permite restaurar una imagen manipulada hasta en un 63 % y si la imagen está modificada más porcentaje, no es posible restaurarla. Como la imagen está manipulada en un 61.208 %, se disponen a recuperar el contenido manipulado y logran descubrir que hubo una camioneta implicada en el robo y uno de sus empleados fue cómplice, tal como se muestra en la Figura 1.7. Esta información les será útil a la hora de empezar el proceso legal en contra de los implicados.

Figura 1.7: Imagen Restaurada.

Los esquemas de marcado estudiados permiten recuperar una imagen manipulada hasta en un 80 %, con una calidad aceptable [Lee y Lin, 2008]. En esta tesis se plantea el problema de restaurar un mayor porcentaje (hasta 95 %) de la imagen manipulada, aunque esto conlleve una disminución en la calidad de la imagen restaurada.

1.5. Hipótesis

Es posible proponer un esquema de marca de agua frágil basado en la DWT, que permita restaurar hasta el 95 % de una imagen manipulada, dependiendo de la calidad de la imagen restaurada que el usuario requiera.

1.6. Objetivos de la Tesis

1.6.1. Objetivo General

Plantear un esquema de marca de agua frágil para imágenes en escala de grises, que sea capaz de restaurar una imagen manipulada a lo más en un 95 %, de acuerdo al porcentaje de manipulación que presente la imagen y a la calidad de la imagen restaurada que el usuario necesite.

1.6.2. Objetivos Particulares

- Detectar las regiones de la imagen que fueron manipuladas utilizando un hash criptográfico de 32 bits para bloques de 8×8 de la imagen.
- Adecuar los esquemas de marca de agua de recuperación exacta redundantes, para que logren restaurar la imagen utilizando la Transformada Discreta Wavelet y además, con distintos niveles de compresión.
- Agregar al algoritmo de restauración un filtrado de la imagen obtenida, que sea adecuado a cada nivel, con el fin de mejorar la calidad de la imagen recuperada.

1.7. Justificación

Actualmente, las técnicas de marcado aplicadas para la restauración de imágenes digitales, que se basan en resolver sistemas de ecuaciones lineales, permiten recuperar hasta el 80 % de manera aproximada, pero con una calidad visual aceptable. El uso de la Transformada Discreta Wavelet, denotada por DWT (del inglés, Discrete Wavelet Transform), se presenta como una solución factible para poder recuperar más porcentaje de la imagen manipulada, esto debido a que dada una imagen I de tamaño $N \times N$, donde $N = 2^s$, entonces es posible hacer $\log_2(N) = s$ niveles de la DWT. Entonces, una imagen se puede comprimir hasta $\frac{1}{2^{2s}}$ de su tamaño original, al

comprimir más, es posible introducir más copias de la imagen en la marca, lo cual da lugar a que se pueda restaurar mayor porcentaje de la imagen.

1.8. Descripción de Capítulos

El presente trabajo de tesis consta de 5 capítulos, los cuales están organizados de la siguiente manera. En el capítulo 2, se exponen de manera breve los fundamentos teóricos en los que se basa este trabajo de investigación y que son necesarios para entender el funcionamiento del esquema de marcado que se presenta. Además, se describe el estado del arte de los trabajos relacionados con marcas de agua frágiles. En el capítulo 3, se explica de manera detallada en qué consiste el algoritmo que se plantea en esta tesis y se describe la implementación. En el capítulo 4, se exponen los resultados de los experimentos realizados y se hace un análisis de los mismos. Finalmente, en el capítulo 5 se presentan las conclusiones que se obtuvieron luego del desarrollo este trabajo de investigación, así como algunas sugerencias de trabajo que podrían hacerse a futuro para mejorar el desempeño y/o funcionamiento del esquema de marcado presentado.

Capítulo 2

Fundamentos Teóricos

Considere una imagen I, la cual tiene una marca de agua frágil en los 3 LSB. Suponga que dicha imagen ha sido manipulada. Para que la marca de agua permita restaurar información perdida de la imagen, se requiere que la marca contenga información de ésta. Es común que en esquemas de marcas de agua, para la información de la imagen I únicamente se consideren los 5 MSB de sus pixeles [Zhang *et al.*, 2011c, Bravo-Solorio *et al.*, 2018, Zhang *et al.*, 2011a]. Para lograr recuperar los pixeles de I que han sido manipulados, se requieren dos procesos: uno que permita detectar las áreas manipuladas y otro que permita restaurar los pixeles manipulados a partir de la información que proporcionan los que no fueron manipulados. En la literatura existen diversos métodos que permiten llevar a cabo la detección de los pixeles manipulados, uno de ellos consiste en dividir la imagen en bloques de tamaño fijo y sin traslape, luego, por cada bloque emplear un mecanismo que les permita saber si el bloque fue alterado o no. Algunos esquemas dividen la imagen en bloques de 4×4 pixeles y emplean 4 bits para verificar la integridad del bloque [Tai y Liao, 2018], otros dividen la imagen en bloques de 8×8 y, por cada bloque, insertan un hash criptográfico de 32 bits para realizar la detección de bloques manipulados [Zhang *et al.*, 2011c, Zhang *et al.*, 2011b, Calderon *et al.*, 2018, Romero, 2019], incluso otros dividen la imagen en bloques de 2×2 pixeles y utilizan 2 bits por bloque

para saber si el bloque fue alterado [Lee y Lin, 2008].

En cuanto al proceso de restauración de la imagen también se han desarrollado diversos esquemas, algunos de ellos se centran en una recuperación exacta, es decir, restauran de manera directa la información de los 5 MSB de la imagen [Zhang *et al.*, 2011c, Zhang y Wang, 2008, Bravo-Solorio *et al.*, 2018], mientras que otros utilizan algún tipo de compresión para representar la información de los 5 MSB, este tipo de recuperación se le denomina aproximada [Zhang *et al.*, 2011a, Qin *et al.*, 2012, Qin *et al.*, 2017, Calderon *et al.*, 2018]. En ambos casos, se pretende recuperar la información de los 5 MSB de la imagen a partir de la información almacenada en los 3 LSB, por lo tanto, el proceso de restauración tiene una limitación de memoria que conlleva una pérdida de información.

En este capítulo se describe el proceso de detección de las áreas que han sido manipuladas y el de recuperación en el esquema que utiliza un hash criptográfico de 32 bits por cada bloque de 8×8 de la imagen, este esquema fue planteado en [Zhang *et al.*, 2011b]. Posteriormente, se explica el proceso de restauración de los bits manipulados. Para concluir el capítulo, se hace una revisión del estado del arte de algunas de las marcas de agua frágiles que hay en la literatura.

2.1. Generación de la marca de agua

En esta sección se va a explicar de manera general, cómo se hace la marca de agua en el esquema planteado por Zhang et al. en [Zhang *et al.*, 2011b], puesto que la propuesta de marca de agua que se presenta en esta tesis está basada principalmente en dicho esquema.

2.1.1. Extracción del vector de datos

Dada una imagen I en escala de grises, de dimensiones $N \times M$, donde N y M son múltiplos de 8. Se divide la imagen I en bloques B_k de tamaño 8×8, sin traslape,

$1 \leq k \leq (\frac{N}{8} \times \frac{M}{8})$. El número total de bloques, será denotado por N_B y está dado por:

$$N_B = \frac{NM}{64} \tag{2.1}$$

Cada bloque B_k tiene un total de 512 bits, de los cuales 320 bits corresponden a los 5 MSB y 192 a los 3 LSB. En estos 192 bits se inserta la marca de agua, la cual se conforma de 160 bits que corresponden a la información que permite restaurar los pixeles manipulados y 32 bits designados para introducir un hash criptográfico. Se va a denotar por d_k al vector que contiene los 320 bits correspondientes a los 5 MSB del bloque B_k, por c_k a los 160 bits de información y por h_k al hash criptográfico correspondiente.

$$B_k \rightarrow 64 \times 8 = 512 \text{ bits} \begin{cases} 320 & \text{MSB} \\ 192 & \text{LSB} \begin{cases} 160 & \text{bits de información} \\ 32 & \text{hash criptográfico} \end{cases} \end{cases} \tag{2.2}$$

Para todos los bloques B_k de la imagen, sus vectores d_k se concatenan en un sólo vector al que denotaremos por d, tal como se muestra en (2.3). El vector de datos d contiene la información de los 5 MSB de la imagen original, así, el vector d tiene tamaño $320N_B$ o $5NM$ bits.

$$d = [d_1|d_2|\ldots|d_{N_B}] \tag{2.3}$$

donde la expresión $[x|y]$ denota la concatenación de los vectores x y y. Si bien la concatenación de vectores no es una operación definida matemáticamente, en los lenguajes de programación es ampliamente utilizada. Lenguajes de programación como python cuentan con funciones como *numpy.concatenate*() para realizar esta operación [VanderPlas, 2016].

Una vez que se calculan los vectores c_k y h_k de cada bloque, se procede a la inserción de la marca en los 3 LSB. En las siguientes subsecciones se explica cómo obtener estos vectores.

Como el vector de datos d contiene la información de los 5 MSB de la imagen, no se inserta directamente a en la marca, si no que se se codifica para que sea difícil descifrar la información de la imagen. Para ello se generan códigos de paridad, los cuales sí forman parte de la marca y además permiten restaurar los bits que hayan sido manipulados. Estos códigos de paridad se calculan utilizando una semilla que sirve como llave secreta, pues de acuerdo a dicha llave se genera la marca de agua y únicamente la posee la persona que marca la imagen, esto le brinda al algoritmo la seguridad de que otra persona no puede generar la misma marca de agua en una imagen, a menos de que tenga acceso a los mismos parámetros.

2.1.2. Cálculo de los códigos de paridad

Existen distintos códigos que permiten verificar si la información que ha sido transmitida por un canal fue manipulada y, en caso de ser así, son capaces de restaurar la información original. Una de las codificaciones más utilizadas es el código Hamming [Hamming, 1950], que en seguida se explica más a detalle.

Códigos Hamming

Considere un mensaje que se transmite a través de un canal con ruido, puede pasar que debido a la mala transmisión de datos, el mensaje llegue erróneo, el código Hamming permite detectar error en un bit y corregirlo, para ello, utiliza algunos *bits de paridad*, que son enviados junto con el mensaje. De acuerdo con Hamming, si suponemos que el mensaje que se quiere transmitir es de tamaño n bits y m es el número de bits de paridad que se deben agregar, entonces la siguiente desigualdad se debe satisfacer:

$$2^m \geq n + m + 1 \tag{2.4}$$

Es decir, $m + n$ son todas las posiciones en las que se puede presentar error, el uno representa cuando no hay error y como m bits pueden generar 2^m combinaciones

distintas, entonces 2^m debe ser por lo menos igual a $m+n+1$ para que pueda indicar si hay error y la posición en la que está. Entonces, de acuerdo con (2.4), para m bits de paridad se pueden transmitir a lo más $n = 2^m - m - 1$ bits de datos y el mensaje que se transmite es de tamaño $n+m = 2^m - 1$ bits [Hamming, 1950]. La codificación Hamming tiene la desventaja de que si más de un bit es erróneo, entonces no se puede hacer la corrección.

Suponga que se quiere transmitir una palabra de 7 bits $d = [d_1, d_2, d_3, d_4, d_5, d_6, d_7]^T$, de acuerdo con Hamming, se requieren 4 bits de paridad $p = [p_1, p_2, p_3, p_4]^T$. Para la codificación, hay varias reglas que se tienen que seguir:

- Los bits de paridad deben ir en las posiciones que son potencias de 2, es decir: $\{1, 2, 4, 8, \ldots\}$.
- Los bits de datos ocupan el resto de las posiciones: $\{3, 5, 6, 7, \ldots\}$.
- El bit de paridad p_i se calcula sumando módulo 2 los bits de datos que están en las posiciones cuyas representaciones binarias tienen un 1 en el $i-ésimo$ lugar (partiendo de la derecha), excluyendo el i. Por ejemplo, p_1 es el bit de paridad de los bits de datos que están en las posiciones: 3, 5, 7, 9, 11,... pues todos esos números tienen un 1 en el bit menos significativo (posición 1 de su representación binaria). Entonces, conforme a la Tabla 2.1, los bits de datos que se van a utilizar para calcular el bit de paridad p_1 son: d_1, d_2, d_4, d_5 y d_7.

Posición	1	2	3	4	5	6	7	8	9	10	11
Bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7

Tabla 2.1: Posiciones para la codificación de Hamming.

Primero se calculan los bits de paridad $p = [p_1, p_2, p_3, p_4]^T$ mediante (2.5), de

acuerdo con lo explicado anteriormente.

$$\begin{aligned} p_1 &= d_1 \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_7 \\ p_2 &= d_1 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_7 \\ p_3 &= d_2 \oplus d_3 \oplus d_4 \\ p_4 &= d_5 \oplus d_6 \oplus d_7 \end{aligned} \tag{2.5}$$

donde $\oplus$ denota la suma módulo 2. De forma matricial, (2.5) se puede expresar como sigue:

$$p = Ad \pmod 2 \tag{2.6}$$

donde

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{2.7}$$

Una vez calculados los bits de paridad, el mensaje que se va a transmitir es: $c = [p_1, p_2, d_1, p_3, d_2, d_3, d_4, p_4, d_5, d_6, d_7]^T$, de acuerdo con las posiciones que le corresponden a los bits de paridad.

$$E = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.8}$$

Sea $\hat{c}$ el mensaje recibido, para detectar si hay error y corregirlo, se multiplica la matriz E por $\hat{c}$. Si $E\hat{c} = [0, 0, 0, 0]^T$, entonces el mensaje fue recibido sin errores y, por lo tanto, $c = \hat{c}$. Si no, $E\hat{c} = [p_1, p_2, p_3, p_4]^T$, entonces el número binario $p_4p_3p_2p_1$ convertido a decimal da la posición del bit en $\hat{c}$ que tiene el error.

A manera de ejemplificar el método descrito anteriormente, considere que se quiere transmitir la palabra $d = [1, 0, 1, 0, 1, 0, 1]^T$, entonces los bits de paridad están dados por $p = Ad \pmod 2 = [1, 1, 1, 0]^T$, por lo tanto, la codificación Hamming queda como sigue:

$$c = [1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1]^T \tag{2.9}$$

Si se realiza la multiplicación Ec el resultado es $[0, 0, 0, 0]^T$ lo cual indica que el mensaje no tiene error. Ahora, suponga que al transmitir el mensaje se dañó el bit de datos d_5, entonces lo que llega es:

$$\hat{c} = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1]^T \tag{2.10}$$

Si utilizamos (2.8), se tiene que $E\hat{c} = [1, 0, 0, 1]^T$, entonces el número binario 1001 en decimal nos indica que el error está en la posición 9 de $\hat{c}$, la cual corresponde a d_5. Finalmente, para corregir el error basta con cambiar d_5 de 0 a 1.

De manera análoga a los códigos Hamming, para los códigos de paridad se puede definir una matriz de incidencia A, de tal manera que si se tiene un vector de datos d, se pueda calcular el vector de códigos de paridad c, utilizando la formulación (2.11).

$$c = Ad \pmod 2 \tag{2.11}$$

donde la matriz A es una matriz binaria, generada de manera pseudo-aleatoria con una semilla y las operaciones se realizan utilizando aritmética módulo 2.

El vector c_k contiene los bits de paridad del bloque B_k. Para calcular el vector c_k del bloque B_k, Zhang *et al.* en [Zhang *et al.*, 2011b] utilizan la codificación (2.11). El vector de datos d se divide en segmentos $\hat{d}_i$ sin traslape de tamaño L con $L > 0$ y múltiplo de 2. Entonces, se genera pseudo-aletoriamente una matriz binaria A de tamaño $\frac{L}{2} \times L$, esto para que la matriz A guarde la proporción entre el vector c y d (c es de la mitad de la longitud de d) y se calcula:

$$\hat{c}_i = A\hat{d}_i \pmod 2. \tag{2.12}$$

Luego, los vectores $\hat{c}_i$ se concatenan en un sólo vector.

$$\hat{c} = [\hat{c}_1|\hat{c}_2|\ldots|\hat{c}_i]. \tag{2.13}$$

El vector $\hat{c}$ se permuta de manera pseudo-aleatoria, utilizando una llave secreta (2.14), a la función para permutar el vector la vamos a denotar por $Permutar(\cdot, llave)$ y recibe dos parámetros: el vector que se desea permutar y la llave secreta. Esta función tiene la propiedad de que es reversible, es decir, teniendo la llave correcta, el vector puede regresar a su orden original, a la función inversa la vamos a denotar por $Permutar^{-1}(\cdot, llave)$ y recibe el vector antes permutado y la llave secreta.

$$c = Permutar(\hat{c}, llave). \tag{2.14}$$

Esta permutación de $\hat{c}$ garantiza que la información de un bloque, quede distribuida de manera pseudo-aleatoria en todos los bloques de la imagen y, con ello, que si un bloque está manipulado, entonces la información que permite restaurarlo no se pierda. De esta manera, los bits que están guardados en los demás bloques no manipulados, permiten restaurar el contenido del bloque manipulado.

Finalmente, el vector c se divide en segmentos sin traslape de tamaño 160 bits, a estos vectores los vamos a denotar por c_k. Dado que cada c_k tiene un tamaño de 160 bits, entonces el vector c tiene un tamaño de $160N_B$ bits, esto es $\frac{160NM}{64} = \frac{5NM}{2}$ y la cantidad de bits correspondientes a los 5 MSB de la imagen es el tamaño del vector d, es decir, $5NM$, entonces la información para restaurar la imagen es la mitad de la cantidad de bits del vector d. Debido a esta limitación de memoria, es común que se utilice una codificación, dicha codificación se puede hacer utilizando los bits de los 5 MSB de manera directa o mediante una compresión con pérdida de los mismos.

2.1.3. Cálculo del Hash Criptográfico

Dado el $k-$ésimo bloque B_k, el hash criptográfico de 32 bits correspondiente denotado por h_k se calcula utilizando una función hash. Una función hash transforma

un conjunto de datos (que por lo regular son cadenas) en una cadena de longitud fija, entre las funciones hash más utilizadas destaca la familia de las SHA (Secure Hash Algorithm, por sus siglas en inglés), algunas de esta familia son: SHA-0, SHA-1, SHA-256, SHA-512; SHA-0 y SHA-1 regresan 160 bits, mientras que SHA-256 y SHA-512 regresan 256 y 512 bits, respectivamente [Burrows, 1995, Eastlake y Jones, 2001]. Entonces, para calcular h_k se aplica una función hash (que puede ser cualquiera de las mencionadas) al vector $[d_k|c_k]$ y de su salida únicamente se consideran 32 bits. La expresión para calcular h_k está dada por la siguiente ecuación:

$$h_k = Hash32([d_k|c_k]) \tag{2.15}$$

donde $Hash32(\cdot)$ denota la operación de aplicar una función hash y quedarse sólo con un arreglo de 32 bits. Una función hash tiene la propiedad de que cualquier cambio en la entrada genera una salida diferente, por esta razón se utiliza para verificar la integridad de los bloques.

Una vez que ya se calcularon los vectores c_k y h_k correspondientes de cada bloque, la marca se forma concatenando los vectores $[c_k|h_k]$. Luego, se toman segmentos de 3 bits sin traslape de $[c_k|h_k]$ para concatenarlos con los 5 MSB de cada pixel del bloque y así tener 8 bits, estos 8 bits se convierten a un número entero y esto forma cada pixel de la imagen marcada.

2.2. Detección de los bloques manipulados basada en un Hash Criptográfico

Sea I_T la imagen marcada que ha sido manipulada. Lo primero que se requiere para restaurar la información manipulada es verificar la integridad de los bloques de la imagen. Denotaremos por $\tilde{B}_k$ a los bloques de la imagen manipulada. Para cada bloque $\tilde{B}_k$ de I_T, se extraen c_k y h_k de los 3 LSB de la imagen (de la marca) y se obtiene el vector $\tilde{d}_k$, que son los 5 MSB del bloque $\tilde{B}_k$. Utilizando los vectores

extraídos, se calcula un nuevo hash $\tilde{h}_k$, tal como se muestra en (2.16).

$$\tilde{h}_k = Hash32([\tilde{d}_k|c_k]). \tag{2.16}$$

Para saber si el bloque $\tilde{B}_k$ ha sido manipulado, se comparan el hash extraído h_k y el hash calculado $\tilde{h}_k$. Si son iguales, significa que el bloque $\tilde{B}_k$ no fue manipulado; si resultan ser distintos, el bloque fue manipulado y cada elemento de $\tilde{d}_k$ y c_k se cambia por un -1, así se podrán diferenciar los bits dañados de los bits no dañados.

2.3. Proceso de Restauración de la imagen

Para resolver los sistemas de ecuaciones equivalentes planteados en el proceso de creación de la marca de agua, Zhang et al. en [Zhang *et al.*, 2011b] reescriben al vector $d = [d_T, d_R]^T$, donde d_T representa los bits marcados como manipulados y d_R los bits no manipulados. De manera análoga, el vector c se reescribe como $c = [c_T, c_R]$, donde c_T son los bits de paridad marcados como manipulados y c_R los no manipulados. Entonces, el sistema de ecuaciones descrito en (2.11) se puede representar según (2.17).

$$\left[\begin{array}{c|c} A_{TT} & A_{TR} \\ \hline A_{RT} & A_{RR} \end{array}\right] \begin{bmatrix} d_T \\ d_R \end{bmatrix} = \begin{bmatrix} c_T \\ c_R \end{bmatrix} \tag{2.17}$$

En el sistema de ecuaciones (2.17), la matriz A se conoce, dado que sabemos la llave secreta para generarla. La información que nos interesa recuperar es d_T, pues son los bits que fueron manipulados de la imagen, además, es posible utilizar los bits de paridad c_R, puesto que de acuerdo con el proceso de detección realizado anteriormente, podemos asegurar que no fueron alterados. A simple vista se pueden observar dos ecuaciones que se obtienen del sistema (2.17): (2.18) y (2.19).

$$A_{TT}d_T = c_T - A_{TR}d_R \tag{2.18}$$

$$A_{RT}d_T = c_R - A_{RR}d_R \tag{2.19}$$

No es posible resolver (2.18), puesto que c_T no se conoce, está marcado como manipulado. En cambio, (2.19) sí se puede resolver, pues c_R y d_R son vectores conocidos. Para resolver el sistema, Zhang et al. utilizan el método de Eliminación Gaussiana, entonces hay otras dos cuestiones que se deben considerar en la solución de este sistema. La primera es que la capacidad de restauración está limitada por el tamaño de la matriz A. Para observar esto, primero se debe analizar la distribución de probabilidad que tienen los bits dañados. La segunda es la dependencia lineal de las columnas de la matriz A_{RT}.

2.3.1. Distribución de los bits dañados

Sea I_T la imagen manipulada y suponga que al extraer su vector d y detectar los bloques alterados, d tiene una tasa de bits manipulados α, con $0 \leq \alpha \leq 1$. Sea $\hat{d} = Permutar(d, llave)$ el vector que resulta de permutar de manera pseudo-aleatoria el vector d, esto para que los segmentos de 320 bits marcados como dañados correspondientes a los bloques manipulados se separen y queden distribuidos.

Para analizar la distribución de los bits dañados en $\hat{d}$, considere el experimento aleatorio que consiste en extraer un bit manipulado o no. La probabilidad de que el bit extraído esté manipulado es α y de que el bit no esté manipulado es $1 - \alpha$. Si se repite el ensayo L veces y cada ensayo es independiente de los demás, entonces, la probabilidad de que en una muestra de tamaño L se tengan n bits manipulados obedece a la distribución binomial (2.20), cuya media y desviación estándar están expresadas en (2.21) y (2.22), respectivamente.

$$P(n|L,\alpha) = \binom{L}{n}\alpha^n(1-\alpha)^{L-n} \tag{2.20}$$

$$\mu = L\alpha \tag{2.21}$$

$$\sigma = \sqrt{L\alpha(1-\alpha)} \tag{2.22}$$

En la Figura 2.1 se muestran las gráficas de dos distribuciones de probabilidad de los bits manipulados, si se toman segmentos de tamaño $L = 64$ de $\hat{d}$, la roja muestra la distribución Binomial considerando una tasa de bits manipulados $\alpha = 0.5$ y la azul considerando que $\alpha = 0.2$, con medias $\mu = 32$ y $\mu = 12.8$ respectivamente.

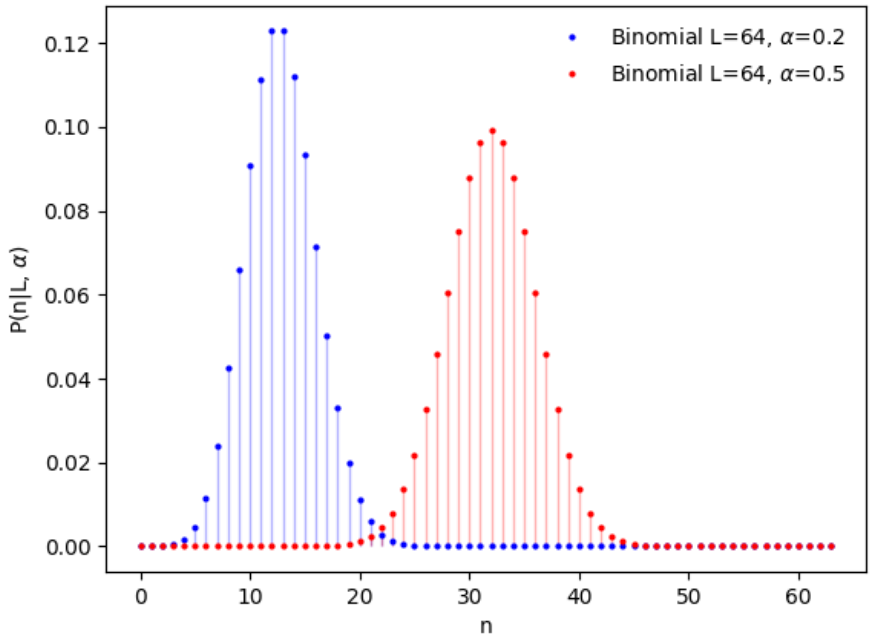


Figura 2.1: Distribución probabilidad de los bits manipulados.

De acuerdo con lo anterior, para un porcentaje de bits dañados α, la cantidad promedio de bits manipulados que puede haber en un segmento de tamaño L es la media αL, esto implica que la longitud promedio del vector d_T, denotada por l_{d_T}, es αL. Si se considera una matriz A de tamaño $\frac{L}{k} \times L$, donde k es la relación que existe entre el número de columnas y renglones de A, entonces para la matriz A_{RT} de (2.19) el número de columnas promedio es l_{d_T} y el número de renglones promedio (que es equivalente al tamaño promedio de los bits de paridad no alterados c_R) es $l_{c_R} = \frac{(1-\alpha)L}{k}$. Así, para que el sistema de ecuaciones (2.19) tenga solución se debe cumplir que el número de renglones de A_{RT} sea igual que el número de columnas

$l_{c_R} = l_{d_T}$, es decir, que:

$$(1-\alpha)\frac{L}{k} = \alpha L \tag{2.23}$$

Haciendo un despeje en (2.23), finalmente se llega a (2.24), lo que nos indica que el porcentaje máximo de restauración está en función de la proporción k.

$$\alpha = \frac{1}{k+1} \tag{2.24}$$

No obstante, estos valores aún están sujetos a la dependencia lineal de la matriz A, por lo tanto, estos son valores teóricos y no es posible obtenerlos en la experimentación. Entonces, el porcentaje máximo de restauración α_{max} está dado por (2.25) y es una cota superior del procentaje de restauración real.

$$\alpha_{max} = \frac{1}{k+1} \tag{2.25}$$

En la Tabla 2.2 se señalan los porcentajes máximos de restauración de acuerdo con una k dada.

k	Tamaño de A	α_{max} (%)
$\frac{1}{4}$	$4L \times L$	80
$\frac{1}{2}$	$2L \times L$	66.666666
1	$L \times L$	50
2	$\frac{L}{2} \times L$	33.333333
3	$\frac{L}{3} \times L$	25
4	$\frac{L}{4} \times L$	20

Tabla 2.2: Porcentaje máximo de restauración según el tamaño de la matriz A.

Ahora se va a analizar la dependencia lineal de las columnas de la matriz binaria A_{RT}. Para una matriz binaria de tamaño $n \times m$, una condición necesaria y suficiente para que sus m columnas sean linealmente independendientes es que su rango sea m,

además, es importante recordar que cualquier conjunto que contiene al vector nulo es linealmente dependiente. Entonces, de acuerdo con [Zhang y Wang, 2008], la probabilidad de que las columnas de una matriz de $n \times m$ sean linealmente dependientes es denotada por $q(n, m)$ y se calcula con las siguientes ecuaciones:

$$q(n,m) = \begin{cases} \frac{1}{2^n} & \text{si } m = 1 \\ q(n,m-1) + (1 - q(n,m-1))\frac{2^{m-1}}{2^n} & \text{si } m = 2, \ldots, n \\ 1 & \text{si } m > n. \end{cases} \tag{2.26}$$

En (2.26) se puede observar que la probabilidad de que una matriz de una columna sea linealmente dependiente es equivalente a la probabilidad de que dicha columna esté conformada únicamente por ceros.

El número de incógnitas (bits manipulados), que en este caso es el vector d_T obedece a la distribución binomial (2.27).

$$P_I(m) = \binom{L}{m} \alpha^m (1-\alpha)^{L-m}, \quad \text{con } m = 0, 1, \ldots, L. \tag{2.27}$$

El número de ecuaciones, que corresponde a la longitud del vector de los códigos de paridad que no están manipulados c_R, tiene la distribución binomial (2.28).

$$P_E(n) = \binom{L/k}{n} (1-\alpha)^n \alpha^{\frac{L}{k}-n}, \quad \text{con } n = 0, 1, \ldots, L/k. \tag{2.28}$$

Por lo tanto, la probabilidad de que el sistema sea linealmente independiente está dada por (2.29).

$$P_{LI} = \sum_{n=0}^{\frac{L}{k}} \sum_{m=0}^{L} P_E(n) P_I(m) (1 - q(n,m)) \tag{2.29}$$

En la Tabla 2.3 se muestran las probabilidades de que una matriz de tamaño $\frac{L}{k} \times L$ sea linealmente independiente, dados los parámetros $k = 2$, distintos valores de L y el porcentaje de bits manipulados α. En la Tabla 2.4 se muestran las probabilidades de que una matriz de tamaño $L \times L$ sea linealmente independiente, para distintos

valores de L. Los valores de L que se consideraron son los sugeridos por los autores del artículo.

α \ $\frac{L}{2} \times L$	128×256	256×512	512×1024
0.1	1	1	1
0.2	1	1	1
0.25	0.999725	1	1
0.3	0.898976	0.971173	1
0.4	0.002759	0.00005	0.010749

Tabla 2.3: Probabilidad de que un sistema de $\frac{L}{2} \times L$ sea linealmente independiente.

α \ $L \times L$	128×128	256×256	512×512	1024×1024
0.1	1	1	1	1
0.2	1	1	1	1
0.25	1	1	1	1
0.3	1	1	1	1
0.4	0.998689	0.999995	1	1
0.5	0.446906	0.46175	0.472686	0.480359
0.6	0	0	0	0

Tabla 2.4: Probabilidad de que un sistema de $L \times L$ sea linealmente independiente.

Como se puede observar en las Tablas 2.3 y 2.4, a medida que L crece, mayor es el porcentaje de bits manipulados que es posible restaurar, es decir, la capacidad de restauración aumenta. Sin embargo, como L determina el tamaño de los sistemas de ecuaciones que se van a resolver, utilizar una L grande del orden del tamaño del vector de datos d que es NM pixeles por 5 bits de cada pixel, de decir, $L = 5NM$ no es factible, puesto que en [Zhang *et al.*, 2011b] los sistemas de ecuaciones son re-

sueltos utilizando el método de Eliminación Gaussiana, el cual tiene una complejidad computacional de $O(L^3)$ para una matriz de tamaño $L \times L$, esto es: $O((5NM)^3)$. Suponga que se tiene una imagen de tamaño $N = 256 = 2^8$, $M = 256 = 2^8$ y se considera $L = 5 * 2^{16}$, entonces se requieren $5^3 * 2^{48}$ operaciones para resolver el sistema de ecuaciones, lo cual no es computacionalmente viable para esta aplicación. Por lo tanto, aunque tamaños de L muy grandes aumentan la probabilidad de que el sistema sea linealmente independiente, no son factibles. Si el tamaño de la L es pequeño, se requieren menos operaciones para resolver el sistema y el tiempo se resolución disminuye, entonces los valores para L que sugieren algunos autores que utilizan el método de Eliminación Gaussiana para recuperación de información son menores o iguales a 1024.

Los resultados de ambas tablas coinciden con lo expuesto en la Tabla 2.2, pues para $\alpha > 0.3333$ en el caso de $k = 2$ y $\alpha > 0.5$ para $k = 1$ la probabilidad de que el sistema sea linealmente independiente es cero y, en ambos casos, la probabilidad indica que no es posible restaurar el 100 % de los bits para el α_{max} teórico.

2.4. Medida de desempeño

En el desarrollo del capítulo 4, se muestran los resultados obtenidos de las pruebas realizadas y para medir el desempeño del esquema propuesto, se emplea el siguiente parámetro que es comúnmente utilizado en la literatura para este fin.

2.4.1. PSNR

El valor pico de la relación señal-ruido (PSNR, por sus siglas en inglés), es una medida relativa de la calidad de la imagen y se expresa en decibeles (dB). En este caso, el PSNR brinda una medida cuantitativa de la calidad de la imagen reconstruida con respecto a la de la imagen marcada. Sean I_1 la imagen marcada e I_2 la imagen restaurada, ambas de dimensiones $N \times M$, para calcular el PSNR primero se calcula

el error cuadrático medio de las imágenes, tal y como se muestra en (2.30), una vez que se tiene el MSE (por sus siglas en inglés, Mean Squared Error), para calcular el PSNR se utiliza (2.31).

$$MSE(I_1, I_2) = \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (I_1[i,j] - I_2[i,j])^2 \tag{2.30}$$

$$PSNR(I_1, I_2) = 10 \log_{10} \frac{MAX_I^2}{MSE(I_1, I_2)} \tag{2.31}$$

donde MAX_I es el máximo valor posible que pueden tomar los pixeles de la imagen. Si los pixeles son representados utilizando 8 bits, $MAX_I = 255$. Considere el caso donde la diferencia $I_1[i,j] - I_2[i,j]$ siempre es 255, entonces el MSE es:

$$MSE = \frac{255^2 NM}{NM} = 255^2 \tag{2.32}$$

Así, el PSNR toma el valor de:

$$PSNR = 10 \log_{10} \frac{255^2}{255^2} = 0 \quad dB \tag{2.33}$$

Por otro lado, si las imágenes I_1 e I_2 son idénticas, se tiene que el $MSE = 0$, entonces:

$$PSNR = 10(\log_{10}(255^2) - \log_{10}(0)) = \infty \tag{2.34}$$

Un bajo MSE significa menor diferencia entre la imagen restaurada y la imagen original y según (2.31), esto se traduce en un valor grande de PSNR, así que entre más grande sea el valor del PSNR se considera mejor calidad en la imagen. Los valores típicos que adopta este parámetro están entre 20 y 50 dB. En general, se considera que si el PSNR es de más de 20 dB, entonces se obtuvo una restauración con calidad aceptable [Thomos *et al.*, 2005].

2.5. Compresión de Imágenes

Existen diversos esquemas de marcado y restauración en la literatura, entre ellos destacan los esquemas de marcado de restauración aproximada, éstos utilizan métodos

de compresión de imágenes para que sea posible almacenar más información de la imagen en la marca y lograr restaurar un mayor porcentaje o también para que la marca de agua se almacene en una menor cantidad de bits, lo cual puede aumentar la calidad de la imagen marcada y de la restaurada.

Compresión es el proceso mediante el cual se reduce la cantidad de datos que se necesitan para representar una imagen digital. La compresión en imágenes puede ser con pérdida o sin pérdida. Compresión sin pérdida se refiere a que no se pierde información de los datos originales durante el proceso, de tal manera que los datos originales pueden ser reconstruidos de manera exacta si se descomprime. Ésta se puede lograr eliminando información redundante del archivo. Algunos ejemplos de formatos de compresión sin pérdida son: JBIG, GIF, Photo CD, PNG y JPEG-LS [Yang y Bourbakis, 2005]. La gran ventaja de este tipo de compresión es que la calidad de la imagen permanece intacta y aún así se logra disminuir el tamaño de la imagen. Compresión con pérdida es el proceso en el cual se disminuye el tamaño de la imagen eliminando información de la misma. Este proceso es irreversible, una vez que se comprime la imagen, no es posible reconstruirla de manera exacta. Una de las principales ventajas de utilizar este tipo de compresión es que se reduce significativamente el tamaño de la imagen, sin embargo, esto implica una disminución en la calidad de la misma. Existen diversas maneras de hacer compresión con pérdida, entre ellas figuran: promedios, la DCT (del inglés, Dicrete Cosine Transform), la DWT y el estándar JPEG (del inglés, Joint Photographic Experts Group), el cual hace uso la DCT para comprimir. A continuación, se explica en qué consisten estos últimos tres métodos de compresión.

2.5.1. Transformada Discreta Coseno

La DCT es equivalente a aplicar la Transformada Discreta de Fourier (DFT, por sus siglas en inglés) a la señal real y convertida en par. La DCT también transforma una señal del dominio del tiempo al dominio de la frecuencia, pero está dada

únicamente en terminos de funciones coseno y, por lo tanto, no tiene parte compleja. Calcular la DCT en dos dimensiones es equivalente a aplicar la DCT de una dimensión a los renglones de la imagen y luego aplicarle la DCT a las columnas del resultado anterior. Sea $I[n,m]$ una imagen de tamaño $N \times M$, la DCT en 2D y su inversa utilizada en esta tesis se calculan según (2.35) y (2.36), respectivamente.

$$DCT[k,l] = \frac{2}{N} w(k)w(l) \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} I[n,m] \cos\left(\frac{(2n+1)k\pi}{2N}\right) \cos\left(\frac{(2m+1)l\pi}{2N}\right) \tag{2.35}$$

$$k = 0, \ldots, N-1, \quad l = 0, \ldots, M-1.$$

$$I[n,m] = \frac{2}{N} \sum_{k=0}^{N-1} \sum_{l=0}^{M-1} w(k)w(l)DCT[k,l] \cos\left(\frac{(2n+1)k\pi}{2N}\right) \cos\left(\frac{(2m+1)l\pi}{2N}\right) \tag{2.36}$$

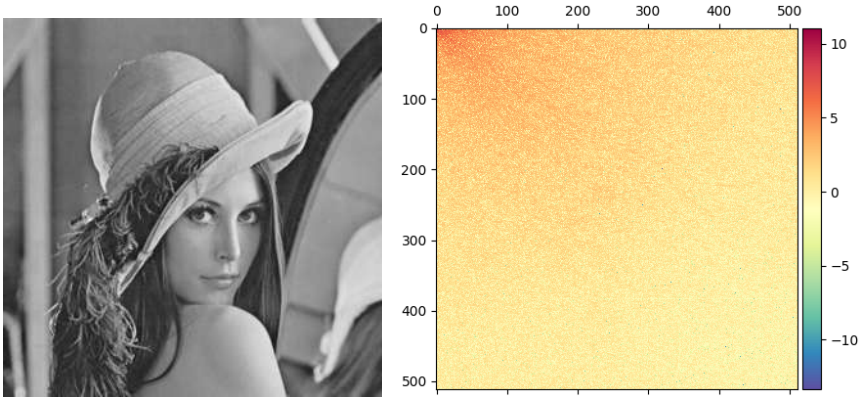
$$n = 0, \ldots, N-1, \quad m = 0, \ldots, M-1,$$

donde

$$w(t) = \begin{cases} \frac{1}{\sqrt{2}} & \text{si } t = 0 \\ 1 & \text{si } t > 0. \end{cases}$$

Cuando se transforma una imagen en sus componentes de frecuencia usando la DCT, se ha visto que la información más importante de la imagen se encuentra en las bajas frecuencias, por lo tanto, es posible desechar algunos de los componentes de altas frecuencias, de esta manera se puede reducir la cantidad de coeficientes que se requieren para representar la imagen, sin sacrificar significativamente la calidad de ésta. Los componentes de las más bajas frecuencias se encuentran en la esquina superior izquierda de la transformada, mientras que los de altas frecuencias están concentrados en la esquina inferior derecha. En la Figura 2.2(a) se puede observar la imagen de Lena en escala de grises con tamaño 512×512 pixeles, en la Figura 2.2(b) se muestra el logaritmo del valor absoluto de la DCT de la Figura 2.2(a), esto nos dice en dónde se encuentran los coeficientes más significativos. En la Figura 2.2(d),

se puede ver el resultado de obtener la DCT de Lena, luego quedarse con el 20 % de las más bajas frecuencias, hacer 0 los demás coeficientes y al resultado de esto sacarle la IDCT; en la Figura 2.2(c), se representan de color blanco los coeficientes de la DCT que se conservaron y de negro los que se hicieron 0 (el filtro). De acuerdo con lo obtenido, la información importante de la imagen en efecto se mantiene en las bajas frecuencias y, por lo tanto, es posible deshacerse de un gran porcentaje de las altas frecuencias, sin degradar significativamente la imagen, esto permite que la imagen se pueda representar con menos coeficientes de su DCT.



(a) ***Imagen Original.*** **(b)** ***Energía de los coeficientes de la DCT.***

(c) ***Máscara*** 20 % ***de los coeficientes DCT.*** **(d)** ***Imagen utilizando sólo el*** 20 % ***de los coeficientes DCT.***

Figura 2.2: Ejemplo de compresión utilizando la DCT.

El estándar JPEG

Es estándar JPEG es un método de compresión basado en la DCT. Este formato de compresión con pérdida es ampliamente utilizado, tanto para imágenes comunes que circulan en la web, como para aplicaciones computacionales específicas, por ejemplo: las marcas de agua. En las siguientes líneas se describe la manera en la que se lleva a cabo el estándar JPEG.

1. La imagen I se divide en bloques de 8×8, sin traslape.

2. Se le resta 128 al valor de cada pixel de los bloques.

3. Se calcula la DCT de cada bloque.

4. Los valores de la DCT se cuantizan utilizando una matriz de cuantización Q, como la que se muestra en (2.37). Esto se hace dividiendo el valor de la DCT entre el valor correspondiente a la misma posición en la matriz Q y lo obtenido se convierte a números enteros, ya sea mediante la función techo o piso.

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{2.37}$$

5. Los coeficientes cuantizados que se consideren para la compresión son tomados en forma de zigzag, siguiendo el orden de las flechas, tal y como lo muestra la Figura 2.3. Por ejemplo, en [Calderon *et al.*, 2018, Romero, 2019] guardan únicamente los primeros 10 coeficientes para representar a la imagen.

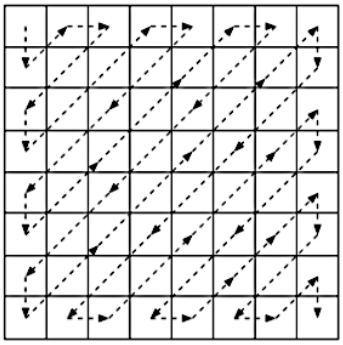


Figura 2.3: Orden de los coeficientes.

6. Hay una última etapa de codificación, para ello se pueden utilizar RLE (del inglés, Run-length Encoding) o Huffman, el cual se realiza en base a unas tablas pre-calculadas de acuerdo a las probabilidades de aparición de cada coeficiente.

En la Figura 2.4(b), se muestra el resultado de comprimir con el método utilizado en [Calderon *et al.*, 2018, Romero, 2019], en el cual se sigue el proceso descrito anteriormente, pero sólo se conservan los primeros 10 coeficientes. Como se utilizaron 10 coeficientes de los 64 por bloque, entonces la tasa de compresión es de $\frac{10}{64} = 0.15625$, el PSNR de la imagen obtenida es 30.864015 dB. En la Figura 2.4(a) se observa la imagen original.

(a) *Imagen Original.* **(b) *Imagen comprimida al* 15.625 %*.***

Figura 2.4: Compresión de una imagen usando 10 coeficientes de la DCT.

2.5.2. Transformada Discreta Wavelet

La Transformada Discreta Wavelet se calcula aplicándole una serie de filtros a la señal. Primero, la señal se pasa a través de un filtro pasa bajas y se sub-muestrea por mitad. Luego, la señal se hace pasar por un pasa altas y se vuelve a sub-muestrear a la mitad. En el caso de una señal de dos dimensiones, se aplica la DWT a los renglones de la imagen y al resultado se le aplica la DWT, pero ahora por columnas. La DWT separa la imagen en una imagen de menor resolución (LL) y componentes de detalles: horizontal (HL), vertical (LH) y diagonal (HH), estas iniciales hacen referencia a los filtros que se aplican para determinar cada sub-banda. La DWT es eficiente y simple de implementar utilizando un filtro de convolución. Dada una señal bidimensional I y l, h filtros pasa bajas y pasa altas, respectivamente. La DWT de una señal de dos dimensiones está dada por (2.38) y (2.39).

$$LL[i,j] = \sum_{n=-\infty}^{\infty} \left[\sum_{m=-\infty}^{\infty} I[2i-n, 2j-m]\ l[m] \right] l[n] \tag{2.38}$$

$$HH[i,j] = \sum_{n=-\infty}^{\infty} \left[\sum_{m=-\infty}^{\infty} I[2i-n, 2j-m]\ h[m] \right] h[n] \tag{2.39}$$

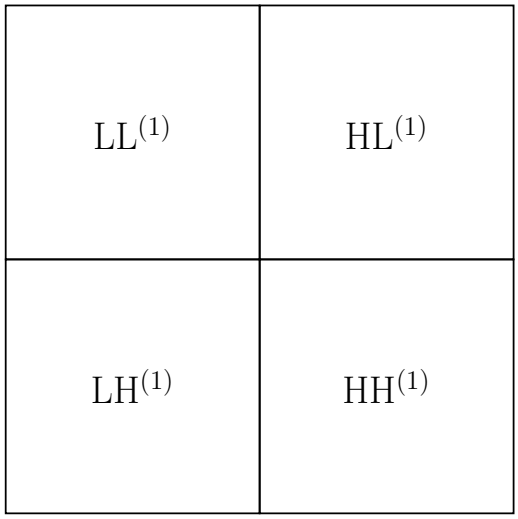
Los coeficientes de la DWT de mayor en magnitud están en la sub-banda LL de cada nivel de descomposición y cada sub-banda ofrece distinta información de la imagen.

- LL: el cuadrante superior izquierdo consta de todos los coeficientes obtenidos de pasar por un filtro pasa bajas a los renglones de la imagen y luego, las columnas del resultado pasarlas nuevamente por el filtro pasa bajas. Este cuadrante representa una versión aproximada de la imagen original, pero con menor resolución.

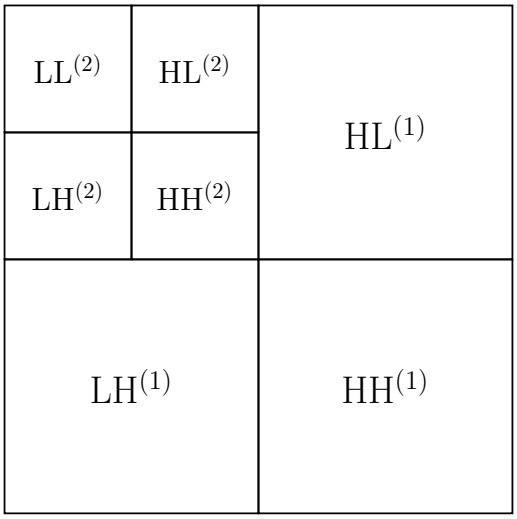
- LH y HL: son los cuadrantes que se calculan filtrando los renglones y las columnas con el filtro pasa bajas y el filtro pasa altas, de manera alternada. El bloque LH muestra los bordes en la dirección vertical, en cambio, el HL muestra los bordes con dirección horizontal de la imagen.

- HH: para obtener este cuadrante, los renglones y las columnas son pasados por el filtro pasa altas. Aquí, se resaltan los bordes de la imagen en una dirección diagonal.

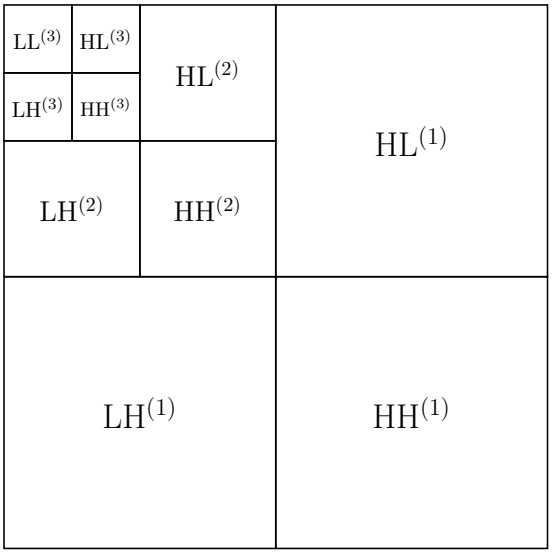
Si se considera la restricción de que las imágenes sean cuadradas, con dimensiones $N \times N$, donde $N = 2^s$ para $s \in \mathbb{N}$, entonces a lo más se pueden calcular $s = \log_2(N)$ niveles de la trasformada. En la Figura 2.5 se muestran las sub-bandas para los primeros tres niveles de descomposición de la DWT en una señal de dos dimensiones. Entre más grande sea la magnitud del coeficiente, más significativo es, por este motivo, una manera de comprimir una imagen es sólo quedarse con la sub-banda LL, que es la que tiene los coeficientes más significativos y desechar los demás. Las partes de las altas frecuencias de la DWT representan información detallada, como los bordes, contornos y texturas. En el caso de las marcas de agua en el dominio transformado que utilizan la DWT, suelen insertar la marca en estos cuadrantes pues esto la vuelve prácticamente imperceptible [Dubolia *et al.*, 2011]. En las Figuras 2.6(a), 2.6(d) y 2.6(g) se observan el primero, segundo y tercer nivel de la DWT de la imagen de Lena, respectivamente. Para obtener la DWT se utilizaron los kerneles de convolución $g = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$ y $h = \left[\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right]$. Luego, para cada nivel de la DWT únicamente se conservaron los coeficientes de la sub-banda $LL^{(nivel)}$, donde $nivel = 1, 2, 3$, y los demás coeficientes se hicieron cero. Este proceso se muestra en la Figura 2.6(b) para nivel 1, en la Figura 2.6(e) para nivel 2 y en la Figura 2.6(h) para el nivel 3. Finalmente, al resultado de hacer cero las demás sub-bandas se le calculó la IDWT y así se obtienen la Figuras 2.6(c), 2.6(f) y 2.6(i). Así, se evidencia que utilizando la DWT es posible representar a la imagen con una menor cantidad de muestras. En el nivel 1, se utilizó un cuarto de la imagen para representarla, por lo tanto, la tasa de compresión es del 25 % y su PSNR es de 29.157720 dB. Para el nivel 2, sólo se reservó $\frac{1}{16}$ de sus coeficientes, entonces la tasa de compresión es del 6.25 % y su PSNR es de 24.786019 dB. Para el nivel 3, sólo se reservó $\frac{1}{64}$ de todos sus coeficientes, entonces la tasa de compresión es de 1.5625 % y su PSNR es de 21.484410 dB.



(a) ***Descomposición de Nivel 1.***



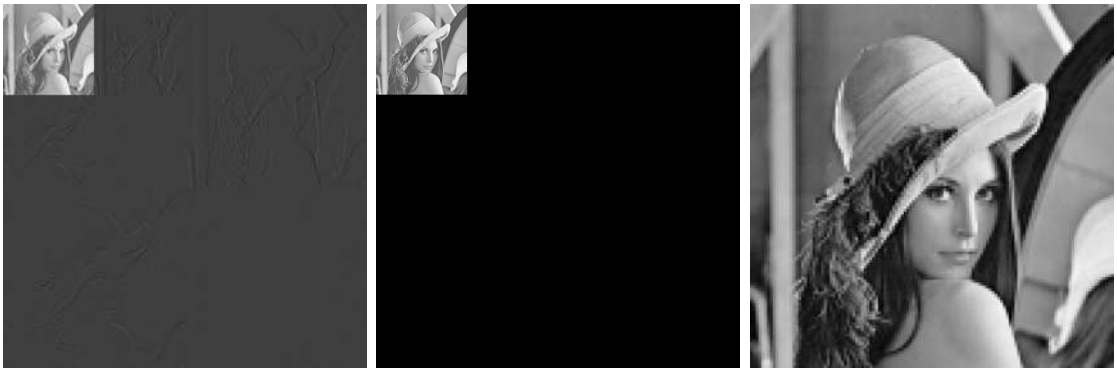
(b) ***Descomposición de Nivel 2.***



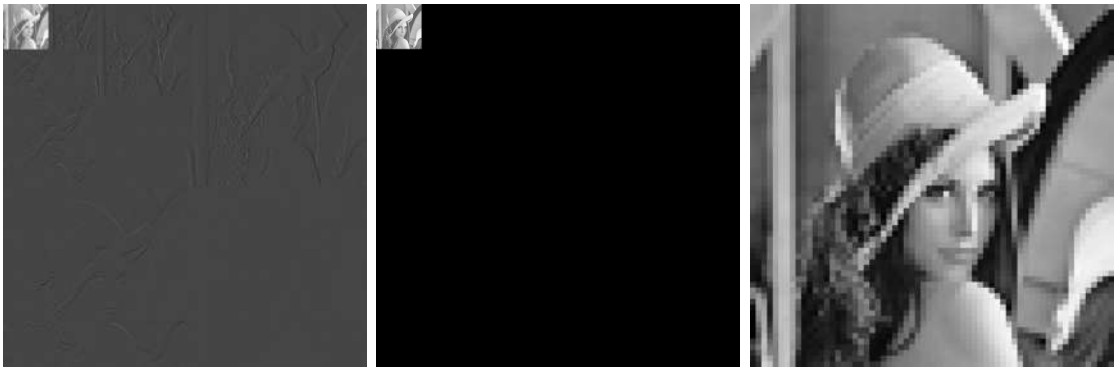
(c) ***Descomposición de Nivel 3.***

Figura 2.5: Distribución de las sub-bandas según el nivel de la DWT.

(a) ***DWT de nivel 1.*** (b) 25 % ***de los coeficientes de DWT de nivel 1.*** (c) ***IDWT de la imagen (b).***



(d) ***DWT de nivel 2.*** (e) 6.25 % ***de los coeficientes de DWT de nivel 2.*** (f) ***IDWT de la imagen (e).***



(g) ***DWT de nivel 3.*** (h) 1.5625 % ***de los coeficientes de DWT de nivel 3.*** (i) ***IDWT de la imagen (h).***

Figura 2.6: Ejemplo de compresión utilizando la DWT.

Además de las tasas altas de compresión, la DWT presenta ciertas ventajas en comparación con otras transformadas comúnmente usadas en la literatura para marcas de agua en imágenes, como la Transformada Discreta Coseno [Dubolia *et al.*, 2011]. En el caso de la DCT, lo que se inserta en la marca son frecuencias, entonces, si una o varias de las más bajas frecuencias no se recuperan, la imagen se degrada significativamente o incluso se puede volver prácticamente imperceptible. En cambio, en la transformada Wavelet, lo que se guarda en la marca es una imagen de más baja resolución, entonces, si algunos coeficientes no se logran restaurar, sólo se pierden determinados pixeles de la imagen y no toda la imagen. Para observar lo anterior, considere la imagen de Lena de tamaño 512×512 pixeles y la sub-banda de baja frecuencia de su DWT, el resultado de esto es una imagen de menor resolución de tamaño un cuarto de la imagen original, esto es, 65536 coeficientes de más baja frecuencia, en la Figura 2.7(a) se muestra el resultado de aplicar la DWT, tomar dichos coeficientes y luego manipular aleatoriamente 2000 de estos coeficientes colocando un 0 en su lugar y, finalmente, aplicar la inversa de la Transformada Discreta Wavelet (por sus siglas en inglés, IDWT). De manera análoga, en la Figura 2.7(b) se aplicó la DCT a la imagen y de los 65536 coeficientes de más baja frecuencia se manipularon 2000 coeficientes colocando un 0 en su lugar, para luego obtener la inversa.

(a) *Imagen DWT.* **(b) *Imagen DCT.***

Figura 2.7: Comparación entre la recuperación con DWT y DCT.

2.6. Filtros en Imágenes

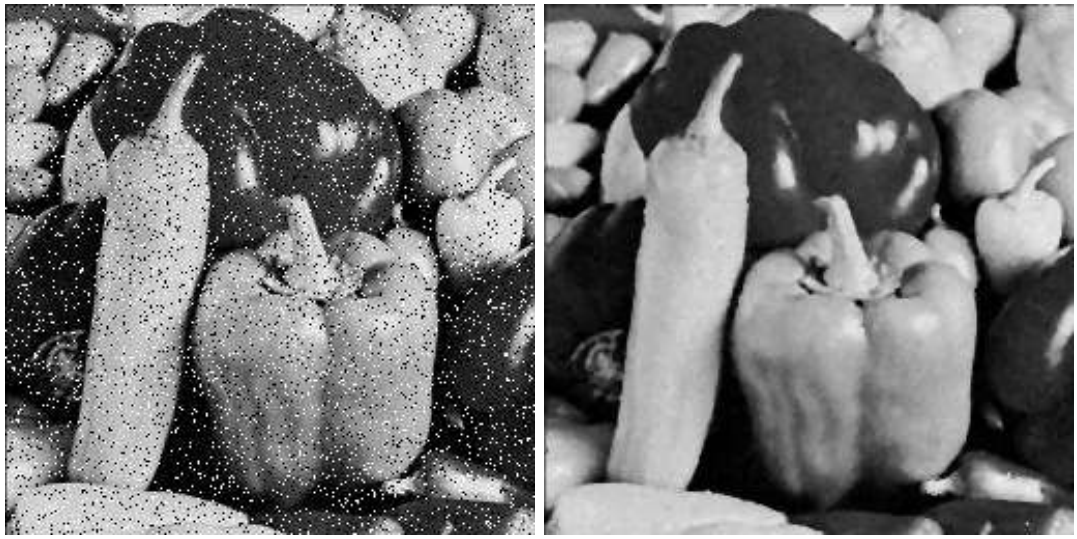
El filtrado en imágenes es una técnica de procesamiento digital de señales comúnmente utilizada para modificar o mejorar apariencia de la imagen. Los filtros se pueden dividir en Lineales y No Lineales. El esquema que se propone en esta tesis hace uso de un filtro de mediana para mejorar la apariencia de la imagen restaurada, dado que cuando una poca cantidad de bits no se logran recuperar, en la imagen restaurada se ven como ruido, específicamente como ruido de tipo sal y pimienta, entonces un filtro puede ayudar a quitar este tipo de ruido y mejorar la calidad de la imagen.

2.6.1. Filtros Lineales

Los filtros lineales son aquellos en los que la salida es una función lineal de los pixeles de entrada y son comúnmente utilizados para eliminar ciertas frecuencias o seleccionar ciertas frecuencias. Un filtro Gaussiano es lineal, éste es muy usado para emborronar una imagen y reducir el contraste.

2.6.2. Filtros No Lineales

En un filtro no lineal la salida es el resultado de aplicarle una función no lineal a la entrada. Un ejemplo de esta clase de filtros es el *filtro de mediana*, éste consiste en una ventana (que se desliza por toda la imagen) y el pixel del centro es reemplazado por la mediana de los otros pixeles que se encuentran en la ventana. Este filtro es útil para suprimir el ruido de tipo "sal y pimienta" en las imágenes, puesto que ayuda a eliminar los valores extremos (pixeles blancos y negros) en una vecindad [Brownrigg, 1984]. En la Figura 2.8, se muestra una imagen con ruido de tipo "sal y pimienta" (a) y en (b) se muestra la imagen resultante de usar un filtro de mediana, con una ventana de tamaño 3×3.



(a) ***Imagen con ruido "sal y pimienta".*** **(b)** ***Imagen Filtrada.***

Figura 2.8: Filtro de Mediana.

2.7. Estado del Arte

A continuación se describen algunos de los esquemas de marcado de agua y restauración que se encuentran en la literatura, éstos se pueden dividir en dos categorías: los de recuperación exacta y los de recuperación aproximada.

2.7.1. Restauración Exacta

Una ventaja de los esquemas de restauración exacta es que la imagen restaurada tiene mejor calidad, pues se reconstruye con los bits de la imagen original. La desventaja de estos esquemas es que no se puede recuperar un gran porcentaje de la imagen, debido a las limitaciones de memoria que tienen las marcas de agua.

Zhang *et al.* en [Zhang *et al.*, 2011b] presentan un esquema de marcado en el cual la imagen se divide en bloques sin traslape de tamaño 8×8. Entonces, por cada bloque se tienen 320 bits que corresponden a los 5 MSB y 192 bits de los 3 LSB. Estos últimos se dividen en 160 bits en los que se insertan los bits de paridad y 32

bits de un hash criptográfico que permite verificar si el bloque fue manipulado o no. Concatenando los 5 MSB de cada bloque se crea el vector de datos d y se genera de forma pseudo-aleatoria una matriz binaria de incidencia A, con d y A se generan los códigos de paridad c mediante la multiplicación matricial $c = Ad$ (mód 2). Luego, se divide el vector de datos d en segmentos sin traslape de tamaño L, así la matriz A resulta de tamaño $L/2 \times L$. La marca de agua consiste en insertar c y h en los 3 LSB. La restauración de la imagen se realiza resolviendo sistemas de ecuaciones. Los resultados de este esquema exponen que es posible restaurar hasta el 28 % de una imagen, si se considera un tamaño de L grande, esto es $L \geq 512$.

Zhang *et al.* en [Zhang *et al.*, 2011c] proceden como en [Zhang *et al.*, 2011b], dividen la imagen en $\frac{N \times M}{64}$ bloques sin traslape de tamaño 8×8, hacen uso de un hash criptográfico de 32 bits para determinar si un bloque fue alterado o no; el vector de datos d lo calculan con los 5 MSB de cada bloque, sin embargo, para calcular los bits de paridad del vector c se permutan con una llave todos los pixeles de la imagen y se forman $\frac{N \times M}{2}$ parejas de pixeles. Para cada pareja de pixeles p_i y p_j se calculan 5 bits aplicando la operación XOR entre el mayor de los 5 MSB de un pixel y el menor de los 5 MSB del otro, es decir $\hat{c}_m = b_{i,7-m} \oplus b_{j,m+3}$, con $m = 0, 1, \ldots, 4$, $b_{i,m}$ y $b_{j,m}$ los 5 MSB de cada pixel. De esta manera se obtiene un total de $\frac{5N \times M}{2}$ de bits de paridad, los cuales se dividen en $\frac{N \times M}{64}$ vectores y así cada uno de éstos es c_i. Para este esquema, los autores reportan una tasa de restauración de 54 %.

Bravo *et al.* en [Bravo-Solorio *et al.*, 2018] proponen una marca de agua basada en [Zhang *et al.*, 2011b], la diferencia radica en que al utilizar la formulación $Ad = c$ para generar los códigos de paridad, emplean dos matrices de tamaño $L/4 \times L$, con lo cual se originan dos vectores de códigos de paridad c_1 y c_2, esto da lugar a un esquema redundante en el cual se espera que la información que no se logre recuperar a partir de c_1 se pueda recuperar con c_2. Para la restauración de la imagen se emplea un método iterativo que consiste en resolver un sistema equivalente $A_1 d = c_1$ y luego $A_2 d = c_2$ iteradamente hasta que se cumpla el criterio de parada. Este esquema per-

mite restaurar el total de la imagen si su porcentaje de manipulación es de hasta un 26 %, con $L = 52$.

2.7.2. Restauración Aproximada

Los esquemas de recuperación aproximada utilizan algún tipo de compresión con pérdida para representar la información de los 5 MSB de la imagen, por ejemplo: la DCT, DWT, SVD (por sus siglas en inglés, Singular Value Decomposition) o promedios de los pixeles. Dado que se habla de una compresión con pérdida de los datos, en estos esquemas suele ser de interés la calidad de la imagen restaurada. La ventaja de estos esquemas es que permite que se recupere más porcentaje de la imagen, pues es posible almacenar más información comprimida de la imagen original.

Calderón *et al.* en [Calderon *et al.*, 2018] presentan un esquema de marcado basado en el estándar de compresión JPEG. Dividen la imagen en bloques de 8×8, a cada pixel le restan 128 y se aplica la transformada DCT a los 5 MSB de los bloques, posteriormente se hace una cuantización para que los coeficientes de la DCT sean valores enteros. El vector de datos d está dado por los primeros 10 coeficientes (convertidos a binario) de la DCT de cada bloque, los cuales corresponden a las bajas frecuencias, tomados en forma de zigzag. Los códigos de paridad c se calculan con la formulación $c = Ad \pmod 2$, el tamaño de la matriz A es de $L \times L$. Para este esquema la marca de agua consiste concatenar d, c y un hash criptográfico de 32 bits. De acuerdo con los resultados, con este método se logra restaurar una imagen que esté manipulada en a lo más 47 % utilizando $L = 1024$.

Zhang *et al.* en [Zhang *et al.*, 2015] plantean un esquema de marcado basado la compresión fractal, la cual a su vez se basa en la DCT. Este esquema emplea tres tipos de bloques, para cada tipo de bloque se utiliza un método distinto para generar la información de la marca de agua para la autenticación y restauración de la imagen. Para cada bloque de 4×4, se tienen tres versiones de la marca de agua, todas

en diferentes cuadrantes, por lo tanto, se tienen tres posibilidades para restaurar la información del bloque. Con este esquema se logra restaurar una imagen modificada hasta en un 80 % con un PSNR que varía entre 22 y 44 dB.

Qin *et al.* proponen un esquema basado en la Transformada Contourlet no Submuestreada NSCT (por sus siglas en inglés, Nonsubsampled Contourlet Transform) [Qin *et al.*, 2012]. La imagen se divide en bloques sin traslape de $n \times n$. Los bits de autenticación se generan utilizando un algoritmo de hashing basado en la DCT y los bits de restauración se calculan codificando los coeficientes de la NSCT de los bloques de la imagen original. De acuerdo con los autores de este artículo, la NSCT es más eficiente capturando y representando las principales características de la imagen, comparada con trasformadas como la DCT. La marca de agua, que consiste en los bits de autenticación y los bits de restauración se insertan en 1 LSB de la imagen. Los resultados de este esquema indican que es posible restaurar una imagen manipulada en un 27.3 % con un $PSNR = 38.78dB$.

Qin *et al.* en [Qin *et al.*, 2017] presentan un esquema de marcado en el cual la imagen se divide en bloques de tamaño 3×3 y sí se traslapan. Existen dos maneras de insertar la marca, dependiendo del lugar donde se encuentre el bloque, puede ser en los 1 LSB o 2 LSB. Los bits de autenticación se generan de acuerdo a la complejidad de cada bloque. Y los bits de información se calculan utilizando los 6 LSB del promedio de los bloques. De acuerdo con los resultados reportados en el artículo, este esquema es capaz de restaurar una imagen que tiene un porcentaje de manipulación menor a 45 % con un PSNR entre 29 y 41 dB.

Zhang *et al.* en [Zhang *et al.*, 2013] plantean un esquema basado en la DCT. Dividen la imagen en bloques de tamaño 2×2. La marca de agua se forma codificando los coeficientes de la DCT de cada bloque y se inserta en otro bloque, de acuerdo con un mapeo de bloques que se genera a partir de una secuencia caótica no-lineal. Los resultados experimentales señalan que es posible restaurar hasta el 34 % de una imagen manipulada, con un $PSNR = 26.05$ dB.

Josué Espinosa en [Romero, 2019], presenta un esquema de marcado y restauración parecido al que se propone en [Calderon *et al.*, 2018]. En este caso, el vector de datos d se forma también con los primeros 10 coeficientes (convertidos a binario) de la DCT de cada bloque, tomados en zigzag, pero la diferencia radica en la codificación, puesto que la matriz A es de tamaño $2L \times L$. Entonces, la marca de agua se conforma del vector de códigos de paridad $c = Ad$ y un hash de 32 bits. Con este esquema se permite restaurar una imagen modificada a lo más en un 63 %.

Lee *et al.* en [Lee y Lin, 2008], dividen la imagen en bloques sin traslape de 2×2 y utilizan 2 bits de paridad para verificar la integridad de cada bloque. Cada uno de estos bloques contiene los promedios de los 5 MSB (la marca de agua) de dos bloques distintos a él, la asignación se realiza mediante una secuencia de mapeos. De esta manera, se forman dos copias de la marca de agua de la imagen y si una de las copias es destruida, la otra servirá para restaurar la información de la imagen. Los resultados obtenidos con este esquema muestran que es posible restaurar hasta un 90 % de la imagen con un $PSNR \approx 20\ dB$.

Zhang *et al.* en [Zhang *et al.*, 2011a], utilizan los métodos de *Proyección de Gradiente* y *Compressive Sensing* para hacer la restauración de la imagen. En esta marca, la imagen también se divide en bloques de 8×8 y se calcula la DCT de los 5 MSB de cada bloque. Con una llave secreta, se permutan los bloques de manera pseudo-aleatoria y se crean grupos de 16 bloques. Los 64 coeficientes de la DCT de los 16 bloques se concatenan en un vector de datos d_i. Luego, con otra llave secreta, se genera de manera pseudo-aleatoria una matriz A de dimensiones 368×1024, de tal manera que la norma Euclideana de cada uno de sus renglones sea 1 y sus elementos sigan una distribución Gaussiana con $\mu = 0$. Con esta matriz, se calcula un vector de recuperación c, usando que $Ad = c$. A diferencia de otros esquemas, $A_{i,j} \in \mathbb{R}$, $d_i \in \mathbb{R}$ y $c_i \in \mathbb{R}$, es decir, son matrices y vectores de números reales. El vector de recuperación c se cuantiza mediante una función no lineal para convertir sus elementos a números enteros. Estos valores del vector de recuperación en binario y un hash criptográfico de

31 bits constituyen la marca de agua. Para la reconstrucción, se utilizan los métodos antes mencionados para resolver los sistemas de ecuaciones. Si el número de ecuaciones es mayor o igual que el número de incógnitas se utiliza el método de compressive sensing [Donoho, 2006], en caso contrario, se utiliza el método de proyección de gradiente [Figueiredo *et al.*, 2007]. Los resultados obtenidos con este esquema exponen que es posible restaurar hasta un 60 % de la imagen.

El esquema de marca de agua frágil que presentan Wei-Liang Tai y Zi-Jun Liao en [Tai y Liao, 2018], dividen la imagen en bloques de 4×4 y utilizan la DWT para comprimir los 6 MSB de los bloques. Para autentificar el contenido de cada bloque utilizan 4 bits y la marca de agua se inserta en los 2 LSB. Utilizan el wavelet de Haar para calcular la DWT y una secuencia de mapas caóticos para que la información de un bloque quede dispuesta en otro. Para la reconstrucción de la imagen presentan un sistema jerárquico de dos niveles, el primero utiliza la inversa de la DWT y el segundo nivel es para los pixeles que no se lograron restaurar, básicamente hace un promedio de la vecindad del pixel.

2.8. Conclusiones del capítulo

De acuerdo con los esquemas de marcas de agua que se encuentran en la literatura, los de restauración aproximada permiten restaurar un mayor porcentaje de una imagen manipulada, en contraste con los esquemas de restauración exacta. Entonces, el uso de una trasformada para representar a la imagen es una buena estrategia si lo que se desea es restaurar un mayor porcentaje de la imagen. Por tal motivo, la marca que se propone en esta tesis es de restauración aproximada y hace uso de la DWT.

La inserción de la marca en los 3 LSB logra que la imagen marcada no se degrade significativamente, además, permite almacenar la cantidad de información suficiente para restaurar la imagen y más si se trata de una versión comprimida de ésta. Por lo tanto, la marca que se plantea en esta tesis se va a insertar en los 3 LSB.

En cuanto al proceso de detección de pixeles manipulados, se va a utilizar el mecanismo planteado en [Zhang *et al.*, 2011b], que consiste en dividir la imagen en bloques de 8×8 y utilizar un hash criptográfico de 32 bits, puesto que este mecanismo logra localizar de manera efectiva los bloques que fueron manipulados.

Respecto a la recuperación de información, se pretende utilizar el mecanismo redundante propuesto en [Bravo-Solorio *et al.*, 2018], el cual permite la recuperación del 100 % de los bits manipulados de manera correcta, para tasas de manipulación α por encima de la que corresponde a la máxima para la formulación (2.11), debido a que utiliza esta formulación para crear dos conjuntos de recuperación y lo que no se restaura con uno, se espera que se restaure con el otro.

Capítulo 3

Desarrollo de la marca de agua basada en la DWT

En el Capítulo 2 se explicó la manera en la que se puede representar una imagen utilizando algunos coeficientes de su DWT y las ventajas que ofrece con respecto a los métodos más comúnmente utilizados en la literatura como la DCT y el estándar JPEG, en este caso, la ventaja que más nos interesa es la tasa de compresión.

La marca de agua que aquí se propone, a la cual se le denotará a partir de ahora como *MarcaDWT*, está basada principalmente en los esquemas presentados en [Zhang *et al.*, 2011b] y [Bravo-Solorio *et al.*, 2018], el primero de ellos se explica en el Capítulo 2. El esquema *MarcaDWT* es una marca de agua frágil y en el dominio espacial, pues se va a insertar en los 3 LSB de los pixeles de la imagen, mientras los otros 5 MSB permanecen intactos. Para el proceso de inserción de la marca, la imagen se divide en bloques sin traslape de tamaño 8×8 y se calcula el vector de datos d utilizando una versión comprimida de los bloques, la compresión se realiza con un método basado en la DWT. Para verificar la integridad de la imagen, se calcula un hash criptográfico de 32 bits, por cada bloque. En cuanto al proceso de recuperación de la imagen, *MarcaDWT* utiliza la formulación redundante propuesta en [Bravo-Solorio *et al.*, 2018]. Dicha formulación consiste en generar dos códigos

de paridad c_1 y c_2 a partir del mismo vector de datos d, utilizando dos matrices pseudo-aleatorias distintas obtenidas a partir de dos claves secretas, es decir, $c_1 = A_1 d$ (mód 2) y $c_2 = A_2 d$ (mód 2). De acuerdo con los resultados experimentales reportados en [Bravo-Solorio *et al.*, 2018], el procedimiento de inserción de la marca resultó cuatro veces más rápido que en [Zhang *et al.*, 2011b], lo cual puede llegar a ser muy útil en ciertos escenarios. Respecto a la capacidad de restauración, utilizando un tamaño de segmento $L < 64$ se reportaron porcentajes de restauración similares al esquema propuesto en [Zhang *et al.*, 2011b] con $L = 256$. Entonces, la formulación redundante propuesta en [Bravo-Solorio *et al.*, 2018] ofrece ventajas de tiempo respecto a la utilizada en [Zhang *et al.*, 2011b], por este motivo se usa esta formulación. Uno de los parámetros de entrada del esquema *MarcaDWT* es el **nivel**, el cual puede tomar cualquier valor del conjunto $\{1, 2, 3\}$, este parámetro hace referencia al nivel de compresión de la DWT que se utiliza para generar la marca y para restaurar la imagen. Es claro que la imagen restaurada tendrá menor calidad conforme el nivel se incremente, pero tendrá mayor capacidad de restauración.

En este capítulo, primero se describe el algoritmo utilizado para el cálculo de la DWT en las imágenes, puesto que únicamente se va a considerar una sub-banda y los demás coeficientes no se calculan. Además, dado que se utiliza un wavelet específico, las operaciones necesarias para calcular dicha sub-banda de la DWT de la imagen se simplifican considerablemente. Posteriormente, se explican de manera detallada los algoritmos de marcado y restauración que se plantean en esta tesis.

3.1. Algoritmo de submuestreo basado en la DWT

Dada una imagen I en escala de grises, de tamaño $N \times M$, se quiere calcular su DWT y reducir la cantidad de bits que se requieren para representarla. Para ello, únicamente se van a considerar los coeficientes de la sub-banda $LL^{(s)}$, donde $s \in \{1, 2, 3\}$ y denota el nivel de la DWT. En seguida, se va a analizar una manera

eficiente en la que se puede realizar este cálculo, para un tipo de kernel de convolución, esto permite que el tiempo de marcado de la imagen y la reconstrucción de ésta, sean menores respecto a otras marcas que utilizan otras transformadas y cuyo cálculo involucra más operaciones.

Sea $g = \left[\frac{1}{c}, \frac{1}{c}\right]$ la respuesta al impulso unitario del filtro pasa bajas, es decir, el kernel de convolución. Bajo esta condición, la sub-banda de baja frecuencia de la DWT de una señal unidimensional x de tamaño N se puede expresar como:

$$Low[i] = x[2i]\left(\frac{1}{c}\right) + x[2i+1]\left(\frac{1}{c}\right), \tag{3.1}$$

con $i = 0, \ldots, \left\lfloor \frac{N}{2} \right\rfloor$ y como se puede observar en (3.1), el submuestreo a la mitad ya está incluido. Se puede considerar a una imagen como un conjunto de señales unidimensionales. Denotemos por I_n al renglón n de la imagen I, entonces $I_n[j]$ con $j = 0, \ldots, M-1$ son los elementos de la señal unidimensional formada por dicho renglón. Recordemos del capítulo 2, que para calcular la sub-banda $LL^{(1)}$ de la imagen, los renglones de I se pasan a través del filtro pasa bajas y se realiza un submuestreo, a la matriz resultante se le vuelve a pasar por el filtro pasa bajas y se vuelve a realizar el submuestreo. A la señal que resulta de pasar el **renglón n** de la imagen por el filtro pasa bajas y submuestrear, la vamos a denotar por $Low_n[m]$ y se calcula de acuerdo a la fórmula (3.2).

$$\begin{aligned} Low_n[m] &= I_n[2m]\left(\frac{1}{c}\right) + I_n[2m+1]\left(\frac{1}{c}\right) \\ &= \frac{I[n,2m] + I[n,2m+1]}{c}. \end{aligned} \tag{3.2}$$

Ahora, se va a considerar la matriz que se forma con las señales Low_n como renglones, esta matriz se convoluciona con el kernel g, sin embargo, esta convolución se realiza sobre las columnas de dicha matriz. A esta operación se le va a denotar por $LL^{(1)}$ puesto que es calcular la sub-banda de baja frecuencia de la DWT de la imagen.

$$\begin{aligned} LL^{(1)}[n,m] &= Low_{2n}[2m]\left(\frac{1}{c}\right) + Low_{2n+1}[2m]\left(\frac{1}{c}\right) \\ &= \frac{Low_{2n}[2m] + Low_{2n+1}[2m]}{c} \end{aligned} \tag{3.3}$$

Sustituyendo (3.2) en (3.3), se obtiene (3.4).

$$\begin{aligned} LL^{(1)}[n,m] &= \frac{\left(\frac{I[2n,2m]+I[2n,2m+1]}{c}\right)+\left(\frac{I[2n+1,m]+I[2n+1,2m+1]}{c}\right)}{c} \\ &= \frac{I[2n,2m]+I[2n,2m+1]+I[2n+1,2m]+I[2n+1,2m+1]}{c^2}. \end{aligned} \tag{3.4}$$

Entonces, si se considera el wavelet de Haar, donde $c = 2$, la ecuación anterior resulta ser el promedio de cada 4 pixeles contiguos de la imagen, tal como se muestra en (3.5).

$$LL^{(1)}[n,m] = \frac{I[2n,2m]+I[2n,2m+1]+I[2n+1,2m]+I[2n+1,2m+1]}{4} \tag{3.5}$$

Como se ha mencionado anteriormente, el esquema *MarcaDWT* utiliza tres niveles de compresión de la DWT, entonces, la sub-banda $LL^{(2)}$ se calcula aplicando (3.5) a la sub-banda $LL^{(1)}$ y sub-banda $LL^{(3)}$ se calcula a partir de $LL^{(2)}$. Si el resultado de (3.5) resulta un número racional, entonces se redondea a entero utilizando la función piso. El wavelet de Haar es el que se va a emplear en el esquema de marcado propuesto en esta tesis $g = \left[\frac{1}{2}, \frac{1}{2}\right]$. Cabe señalar que el propósito de esta tesis no es realizar una comparación entre wavelets y determinar cuál es el mejor para la compresión de la imagen. En este caso, únicamente se consideraron kerneles de convolución de la forma $g = \left[\frac{1}{c}, \frac{1}{c}\right]$, puesto que proporcionan características de compresión de la imagen suficientes para lograr el objetivo de esta tesis y, en particular, el wavelet de Haar resulta muy conveniente, pues el hecho de que se trate de un promedio de los pixeles de la imagen facilita la reconstrucción de la imagen, pues es posible hacer un sobremuestreo para recontruir la imagen. Entonces, visto de esta manera, la compresión de la imagen consiste en un *submuestreo* de los pixeles la imagen.

A pesar de que el algoritmo, tal como está, se puede aplicar a una imagen en general, en este esquema se va a aplicar a los bloques de tamaño 8×8 de la imagen. Se va a considerar que el nivel 0 corresponde a los 5 MSB del bloque original, es decir, $LL^{(0)} = 5MSB(B_k)$. En un primer nivel, la sub-banda $LL^{(1)}$ del bloque tiene una resolución de 4×4 pixeles; en un segundo nivel, la sub-banda correspondiente $LL^{(2)}$

se obtiene a partir de $LL^{(1)}$ y tiene tamaño 2×2 pixeles; finalmente, en el tercer nivel, la sub-banda $LL^{(3)}$ consta de un sólo pixel. En la Figura 3.1 se muestra la resolución de los tres niveles de la sub-banda $LL^{(nivel)}$ de la DWT para un bloque B_k de tamaño 8×8. Considerando que únicamente se toman en cuenta los 5 MSB de la imagen, entonces para representar B_k se requieren 320 bits, en el nivel 1 se necesitan 80 bits, en el nivel 2 se necesitan 20 bits y para el nivel 3 se precisan 5 bits.

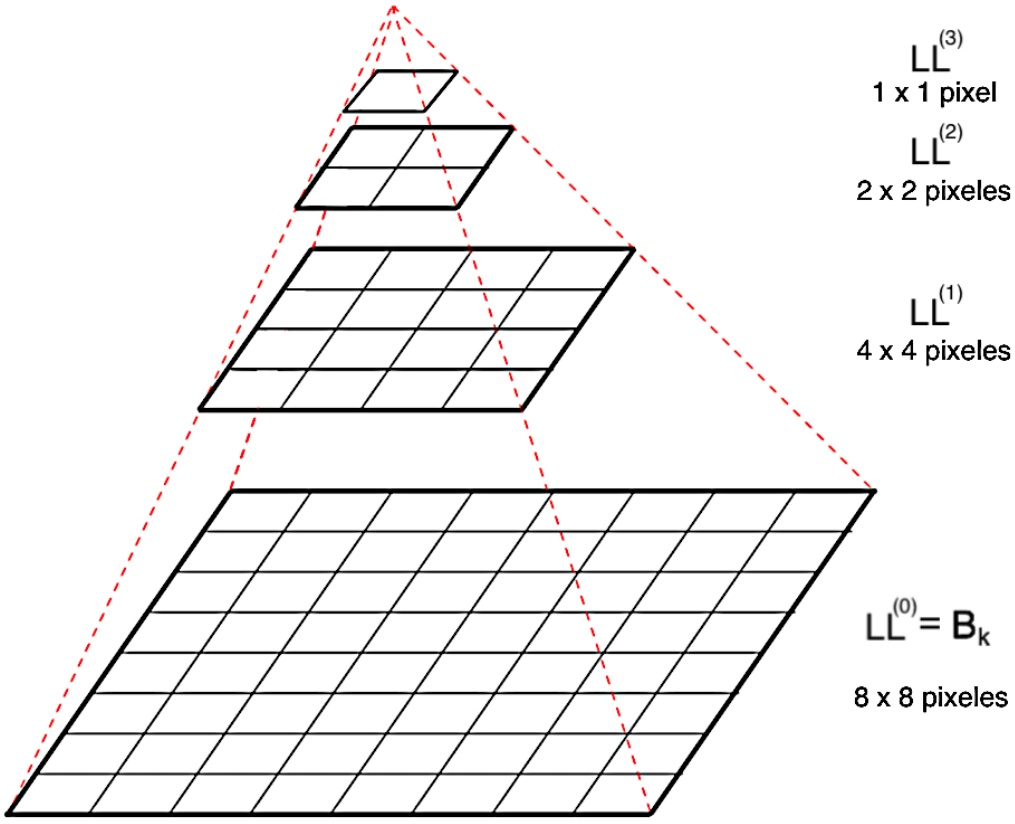


Figura 3.1: Tamaño de las sub-bandas $LL^{(s)}$ de la DWT de un bloque.

Todo el proceso de cálculo de las sub-bandas $LL^{(s)}$, con $s \in \{1, 2, 3\}$ se puede ver de manera análoga a las pirámides Gaussianas, en las cuales, en la base de la pirámide se encuentra la imagen original y cada nivel de la pirámide se obtiene suavizando la imagen con un kernel Gaussiano y submuestreando el nivel anterior. En este caso, para subir un nivel en la pirámide se obtiene el promedio cada cuatro pixeles contiguos del

nivel anterior.

El Algoritmo 1 calcula la sub-banda $LL^{(nivel)}$ de la DWT de una imagen utilizando el wavelet de Haar y fue nombrado *Submuestrear*. Este algoritmo describe el proceso de submuestreo de una imagen utilizando el método descrito anteriormente y sólo calcula la sub-banda $LL^{(nivel)}$ de la DWT de la imagen. El algoritmo recibe como parámetros una imagen I, sus dimensiones $N \times M$ (ambos múltiplos de 8) y el *nivel* hasta el cual se quiere comprimir; regresa la imagen de menor resolución I_C de acuerdo al nivel dado, pero sin los 3 LSB. Si el nivel elegido es 0, entonces regresa los 5 MSB de la imagen original. Las lineas $3, 4$ y 5 calculan las dimensiones de la imagen submuestreada en cada nivel. En las lineas $6 - 13$ se calculan los pixeles correspondientes a cada nivel a partir del nivel anterior, cada nivel se va sustituyendo en el arreglo bidimensional P. Finalmente, el algoritmo regresa la imagen submuestreada.

Algoritmo 1: Algoritmo de Submuestreo.

Entrada: imagen original I, dimensiones de la imagen N y M, nivel de submuestreo *nivel*.
Salida: imagen submuestreada I_C.

SUBMUESTREAR($I, N, M, nivel$)

1 $LL \leftarrow$ Matriz(N, M)
2 **para** $l \leftarrow 0$ **hasta** *nivel* **hacer**:
3 $t \leftarrow 2^l$
4 $R \leftarrow \left\lfloor \frac{N}{t} \right\rfloor$
5 $C \leftarrow \left\lfloor \frac{M}{t} \right\rfloor$
6 **para** $i \leftarrow 0$ **hasta** $(R-1)$ **hacer**:
7 **para** $j \leftarrow 0$ **hasta** $(C-1)$ **hacer**:
8 **si** $l = 0$ **entonces**:
9 $LL[i,j] \leftarrow \left\lfloor \frac{I[i,j]}{8} \right\rfloor$
10 **si no** :
11 $LL[i,j] \leftarrow \frac{LL[2i,2j]+LL[2i,2j+1]+LL[2i+1,2j]+LL[2i+1,2j+1]}{4}$
12 **fin para**
13 **fin para**
14 **fin para**
15 $I_C \leftarrow$ Matriz(R, C)
16 $I_C \leftarrow LL[0:R, 0:C]$
17 **regresar** I_C

3.1.1. Algoritmo de sobremuestreo

El siguiente algoritmo se va a definir para imágenes en general, sin embargo, se va a aplicar a los bloques de la imagen, que a su vez, son imágenes. Para la reconstrucción de la imagen hay que tener en cuenta que los 5 MSB de la imagen original corresponden al nivel 0. Además, se hace la consideración de que partir del nivel s se quiere reconstruir el nivel $s-1$, con $s \in \{1,2,3\}$, bajo estas condiciones, los pixeles del nivel $s-1$ están dados por la siguiente ecuación:

$$\begin{aligned} LL^{(s-1)}[2i,2j] &= LL^{(s)}[i,j] \\ LL^{(s-1)}[2i,2j+1] &= LL^{(s)}[i,j] \\ LL^{(s-1)}[2i+1,2j] &= LL^{(s)}[i,j] \\ LL^{(s-1)}[2i+1,2j+1] &= LL^{(s)}[i,j], \end{aligned} \tag{3.6}$$

donde $i,j = 0,\ldots,(2^{(3-s)}-1)$. Es decir, a los cuatro pixeles del nivel anterior con los que se calculó el promedio se les asigna su promedio. Entonces, para reconstruir el bloque B_k a partir de un nivel s de submuestreo, se reconstruyen todos los niveles anteriores hasta llegar al nivel 0. Se puede notar que si del nivel 3 se quiere reconstruir el bloque B_k, entonces todo el bloque tendrá el mismo valor: $LL^{(3)}[0,0]$.

En la Figura 3.2, se muestra un ejemplo de la reconstrucción por niveles. Se tienen los coeficientes del la sub-banda $LL^{(2)}$ de un bloque y a partir de ésta se quiere reconstruir el nivel 0 (el bloque), para ello, primero se va a reconstruir el nivel $LL^{(1)}$. Entonces, se tiene que la sub-banda $LL^{(2)}$ se conforma de los siguientes valores:

$$\begin{aligned} LL^{(2)}[0,0] &= a \\ LL^{(2)}[0,1] &= b \\ LL^{(2)}[1,0] &= c \\ LL^{(2)}[1,1] &= d, \end{aligned}$$

donde a,b,c y d son los valores de los pixeles. De acuerdo con (3.6), para $i=1$ y

$j = 0$, se obtiene lo siguiente:

$$LL^{(1)}[2,0] = LL^{(2)}[1,0] = c$$
$$LL^{(1)}[2,1] = LL^{(2)}[1,0] = c$$
$$LL^{(1)}[3,0] = LL^{(2)}[1,0] = c$$
$$LL^{(1)}[3,1] = LL^{(2)}[1,0] = c.$$

Y eso se realiza para los demás pixeles restantes $[i,j]$ de $LL^{(2)}$, es decir, $\{[0,0],[0,1],[1,1]\}$, de esta manera se logra reconstruir el nivel 1. Se procede de igual manera para reconstruir el nivel 0, pero ahora considerando los valores de $LL^{(1)}$. En la Figura 3.2, se muestra el resultado de reconstruir el bloque B_k a partir del nivel 2 de compresión para el ejemplo dado. Como se puede observar, el resultado es una matriz en la cual se pueden distinguir cuatro submatrices de dimensiones $2^2 \times 2^2 = 4 \times 4$, cada una con valor único en todos sus elementos: a, b, c y d. Esta observación fue utilizada para realizar el algoritmo de la descompresión de la imagen que se explica a continuación.

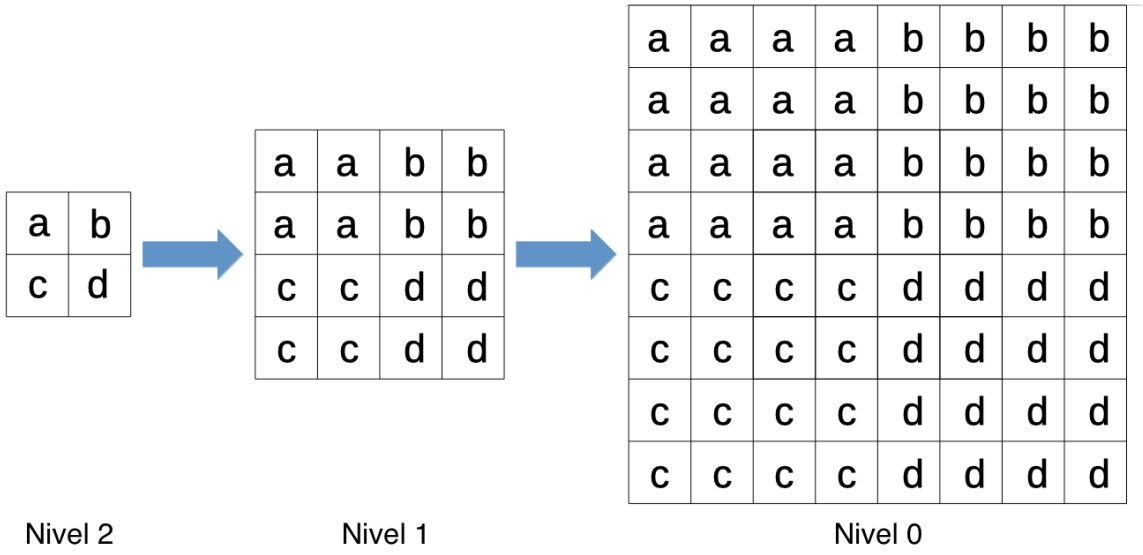


Figura 3.2: Sobremuestreo de un bloque a partir del nivel 2 de submuestreo.

En el Algoritmo 2 llamado *Sobremuestrear*, se presenta el pseudo-código para

reconstruir una imagen a partir de la imagen submuestreada I_C. Este algoritmo recibe como parámetros la imagen previamente submuestreada I_C, el *nivel* hasta el cual se submuestreó y sus dimensiones $N_s \times M_s$, las cuales están definidas como $N_s = \left\lfloor \frac{N}{2^{nivel}} \right\rfloor$ y $M_s = \left\lfloor \frac{M}{2^{nivel}} \right\rfloor$, donde N y M son las dimensiones de la imagen original. La salida de este algoritmo es la imagen reconstruida I_D (nivel 0). Las líneas $1-3$ se encargan de calcular las dimensiones que debe tener la imagen I_D. La línea 4 crea la matriz de I_D. De la linea 5 a la linea 15, se le asignan los valores correspondientes a la imagen reconstruida; dados $i \in 0, \ldots, N-1$ y $j \in 0, \ldots, M-1$ se calculan cuáles elementos de I_D van a tener el valor de $I_C[i, j]$. Note que la imagen I_D que se regresa tiene 8 bits, aunque los 3 LSB son cero, pues en la línea 11 al hacer la asignación, el valor de I_C se multiplica por 8.

Algoritmo 2: Algoritmo de Sobremuestreo.

Entrada: imagen submuestreada I_C, dimensiones de la imagen submuestreada N_s y M_s, nivel de submuestreo *nivel*.
Salida: imagen reconstruida I_D.

```
SOBREMUESTREAR(I_C, N_s, M_s, nivel)
 1  t ← 2^nivel
 2  N ← N_s * t
 3  M ← M_s * t
 4  I_D ← Matriz(N, M)
 5  para i ← 0 hasta (N_s − 1) hacer:
 6     para j ← 0 hasta (M_s − 1) hacer:
 7        t_i ← i * t
 8        t_j ← j * t
 9        para k ← t_i hasta (t_i + t − 1) hacer:
10           para l ← t_j hasta (t_j + t − 1) hacer:
11              I_D[k, l] ← (I_C[i, j] * 8)
12           fin para
13        fin para
14     fin para
15  fin para
16  regresar I_D
```

En las Figuras 3.3, 3.4 y 3.5 se muestra el resultado de submuestrear y sobremuestrear por bloques la imagen de Lena de 512×512 utilizando los algoritmos 1 y 2. Para ello, la imagen de Lena se divide en bloques de 8×8, a cada bloque se le aplica el algoritmo 1 con $nivel = 1$, esto da como resultado la imagen comprimida I_C, luego, a partir de I_C se reconstruye por bloques la imagen de Lena usando el algoritmo 2, en la Figura 3.3 se puede ver el resultado, el PSNR de la imagen obtenida es de 28.9882 dB. En la Figura 3.4, se puede observar la imagen descomprimida, pero ahora utilizando el nivel 2 de los algoritmos 1 y 2 y procediendo igual que para la figura anterior, la imagen descomprimida tiene un $PSNR = 24.4063$ dB. En la Figura 3.5, se muestra la imagen resultante para el nivel 3, la cual tiene un $PSNR = 21.1571$ dB. Dado que sólo se utilizan los 5 MSB de la imagen, la imagen original tiene $5NM$ bits, aplicando el nivel 1 de compresión del algoritmo 1, la imagen se logra representar con $\frac{5NM}{4}$ bits, mientras que para el nivel 2 se utilizan $\frac{5NM}{16}$ bits y $\frac{5NM}{64}$ bits para el nivel 3.

Imagen submuestreada I_C

Imagen sobremuestreada I_D

Figura 3.3: Imagen reconstruida a partir de la sub-banda $LL^{(1)}$ de los bloques.

Sobremuestrear

Imagen submuestreada I_C

Imagen sobremuestreada I_D

Figura 3.4: Imagen reconstruida a partir de la sub-banda $LL^{(2)}$ de los bloques.

Sobremuestrear

Imagen submuestreada I_C

Imagen sobremuestreada I_D

Figura 3.5: Imagen reconstruida a partir de la sub-banda $LL^{(3)}$ de los bloques.

Además, en las Figuras 3.3, 3.4 y 3.5 es posible ver cómo la calidad de la imagen descomprimida decrece a medida que el nivel de compresión aumenta. Ésta es una de las claras diferencias entre los niveles que ofrece el esquema propuesto.

3.2. Esquema de marca de agua: MarcaDWT

Una vez descrita la parte de cómo se va a comprimir la imagen para poder representarla con una menor cantidad de bits, se puede proceder a explicar de manera detallada el proceso de creación e inserción de la marca. Posteriormente, se expone el proceso de restauración de la imagen utilizado en el esquema de marcado *MarcaDWT* que aquí se propone.

3.2.1. Proceso de inserción de la marca

La imagen I se divide en bloques B_n de tamaño 8×8 sin traslape, con $n \in \{0, 1, \ldots, N_B - 1\}$. Dado el nivel s, con $s \in \{1, 2, 3\}$, el vector de datos d se calcula con las versiones comprimidas de los bloques, para ello se utiliza el algoritmo 1 y se obtiene la sub-banda $LL^{(s)}$ de la DWT de los 5 MSB de cada bloque, el resultado se convierte a binario y se coloca en un vector al que se le va a denotar por $\hat{d}_n$.

$$\hat{d}_n = ToBin(Submuestrear(B_n, s)) \tag{3.7}$$

donde $ToBin(.)$ denota una función auxiliar que convierte a binario los elementos de un arreglo. Para todos los bloques, los vectores $\hat{d}_n$ se concatenan y forman el vector $\hat{d}$, tal como se muestra en (3.8).

$$\hat{d} = [\hat{d}_0 | \hat{d}_1 | \ldots | \hat{d}_{N_B - 1}] \tag{3.8}$$

El vector $\hat{d}$ se permuta de manera pseudo-aleatoria con una *llave* secreta, esto para que la información de los bloques quede dispersa, al vector que resulta le vamos a denotar por d y está dado por (3.9).

$$d = Permutar(\hat{d}, llave) \tag{3.9}$$

El nivel de compresión determina el tamaño del vector d. Si la compresión es de nivel $s = 1$, entonces la sub-banda $LL^{(1)}$ de un bloque tiene tamaño 4×4 y requiere (16)(5) bits para almacenarse, por lo tanto, el vector d tiene tamaño $(16)(5)N_B = 80N_B$ bits; si la compresión es de nivel $s = 2$, entonces d tiene tamaño $(4)(5)N_B = 20N_B$; si es de nivel 3, entonces d es de tamaño $5N_B$. Dichas diferencias de tamaño se van a utilizar más adelante para determinar ciertos aspectos de los niveles. Siguiendo con la creación de la marca, el vector d se divide en segmentos sin traslape $d^{(i)}$ de tamaño L, tal como se muestra en (3.10).

$$d^{(i)} = d[iL : (i+1)L] \tag{3.10}$$

La notación $v[p : q]$ denota el segmento del vector v que va de la posición p hasta la $q - 1$. El número de segmentos $d^{(i)}$ que hay en d es $Q_1 = \frac{80N_B}{L}$, para el nivel 1; $Q_2 = \frac{20N_B}{L}$, para nivel 2; $Q_3 = \frac{5N_B}{L}$ para el nivel 3. En general, se va a denotar por Q_s al número de segmentos de tamaño L que hay en d, para el nivel s.

En cuanto a los códigos de paridad, se generan dos códigos de paridad c_1 y c_2, para ello, se crean dos matrices binarias y pseudo-aleatorias distintas A_1 y A_2, luego, cada una se multiplica por un segmento $d^{(i)}$ del vector d, todas las operaciones se realizan con aritmética módulo dos [Bravo-Solorio *et al.*, 2018], de manera general, se calculan como se muestra en (3.11).

$$\begin{aligned} c_1^{(i)} &= A_1 d^{(i)} \pmod 2 \\ c_2^{(i)} &= A_2 d^{(i)} \pmod 2, \end{aligned} \tag{3.11}$$

con $i = 0, \ldots, (Q_1 - 1)$ para generar los códigos en el nivel 1, $i = 0, \ldots, (Q_2 - 1)$ para el nivel 2 y $i = 0, \ldots, (Q_3 - 1)$ para el nivel 3. Los códigos de paridad calculados mediante (3.11) se concatenan para formar los vectores $c_1 = [c_1^{(0)}|c_1^{(1)}| \ldots |c_1^{(Q_s-1)}]$ y $c_2 = [c_2^{(0)}|c_2^{(1)}| \ldots |c_2^{(Q_s-1)}]$. Cabe señalar que el tamaño de las matrices A_1 y A_2 varía dependiendo el nivel, esto debido a que el vector de datos d tiene distinto tamaño con cada nivel y los códigos de paridad c_1 y c_2 tienen el mismo tamaño independientemente del nivel, pues es la información que se inserta en la marca, entonces,

se debe adecuar el tamaño de las matrices para que el producto se pueda efectuar, sin embargo, la fórmula para calcular los códigos de paridad (3.11) es la misma para todos los niveles. En seguida se calcula el tamaño que deben tener las matrices para que el producto matricial se pueda llevar a cabo en cada nivel.

La marca de agua se va a insertar en los 3 LSB de la imagen y por cada bloque de 8×8 se va a utilizar un hash criptográfico de 32 bits para verificar su integridad. Dado que los 3 LSB de un bloque corresponden a 192 bits y 32 de ellos serán usados para el hash criptográfico, entonces para almacenar los códigos de paridad c_1 y c_2 se van a utilizar los 160 bits restantes, cada uno de ellos se va a almacenar en 80 bits por bloque. Entonces, c_1 debe tener una longitud de $80N_B$ bits para almacenar los códigos de todos los bloques y lo mismo para c_2, esto es independiente del nivel.

El esquema *MarcaDWT* permite elegir entre tres niveles de marcado, de acuerdo con las necesidades del usuario y cada uno de ellos tiene una diferencia substancial. Los vectores c_1 y c_2 tienen tamaño $80N_B$ (independientemente del nivel) y, en el nivel 1, el vector d tiene una longitud de $80N_B$, entonces la proporción entre las dimensiones de d y c_r, con $r \in \{1, 2\}$, es $k_1 = 1$ y esta proporción debe ser la misma que entre las columnas y los renglones de A_1 y A_2 para poder llevar a cabo la formulación (3.11), por lo tanto, las matrices A_1 y A_2 deben tener dimensiones $L \times L$. Para el nivel 2, el vector d tiene una longitud de $20N_B$, así que la proporción entre las columnas y los renglones de A_1 y A_2 es $k_2 = \frac{20N_B}{80N_B} = \frac{1}{4}$, por lo tanto, las dimensiones de A_1 y A_2 son $\frac{L}{k_2} \times L = 4L \times L$. Finalmente, en el nivel 3 el vector d tiene una longitud de $5N_B$, así que la proporción que se debe conservar es $k_3 = \frac{5N_B}{80N_B} = \frac{1}{16}$ y por lo tanto, las matrices A_1 y A_2 deben tener tamaño $\frac{L}{k_3} \times L = 16L \times L$. De acuerdo con (2.24), la tasa de restauración máxima depende de la proporción k_s, esto implica que cada nivel va a tener distinta tasa de restauración.

En la Tabla 3.1 se muestran las tasas máximas de restauración de cada nivel cuando se utiliza una formulación del tipo $Ad = c$ (sin redundancia), sin embargo, en este esquema se está utilizando una formulación redundante y de acuerdo con los resultados presentados por Bravo-Solorio *et al.* se esperan tasas de restauración mayores a las presentadas en la tabla, pues se tienen dos formulaciones y lo que no restaura una, podría restaurarse con la otra, además, en cada iteración la tasa de daño disminuye.

Nivel s	k_s	Tamaño de A_1 y A_2	Tasa máxima de restauración α
1	1	$L \times L$	0.5
2	$\frac{1}{4}$	$4L \times L$	0.8
3	$\frac{1}{16}$	$16L \times L$	0.941176

Tabla 3.1: Tasa de restauración máxima para cada nivel.

En la Tabla 3.2, se muestra la probabilidad de que el sistema sea linealmente independiente, calculada con (2.29), para $L = 32$ y para distintas tasas de daño. Se puede observar que conforme el sistema es más sobre-determinado (tiene más renglones que columnas) la probabilidad de que el sistema sea linealmente independiente aumenta, por lo tanto, conforme crece el nivel, más porcentaje de pixeles alterados es capaz de restaurar el esquema propuesto.

Tasa de daño \ Nivel	1	2	3
0.5	0.4032	1	1
0.6	0.0372	1	1
0.7	0.0002	0.9936	1
0.8	0	0.4135	1
0.9	0	0	0.9989

Tabla 3.2: Probabilidad de que los sistemas sean linealmente independientes para $L = 32$.

La Figura 3.6 explica cómo se calculan los códigos de paridad para cada nivel, cabe señalar que las operaciones matriciales que se indican en la imagen se hacen con aritmética módulo 2. Se va a denotar por $A_1^{(s)}$ y $A_2^{(s)}$ a las matrices binarias y aleatorias, cuyo tamaño depende del nivel s, con $s \in \{1, 2, 3\}$, el tamaño se presenta en la Tabla 3.1. Sea $P_s = \frac{1}{k_s}$, donde k_s es la proporción que se calculó para cada nivel, así $P_1 = 1$, $P_2 = 4$ y $P_3 = 16$. Entonces, de acuerdo con la imagen, cada segmento $c_1^{(i)}$ tiene tamaño $P_s L$ e igual para $c_2^{(i)}$. Por ejemplo, para el nivel 3, dichos segmentos van a tener tamaño $16L$. Además, el vector d previamente calculado se utiliza para ambos códigos de paridad, lo que hace que sean códigos distintos son las matrices aleatorias generadas con diferentes claves.

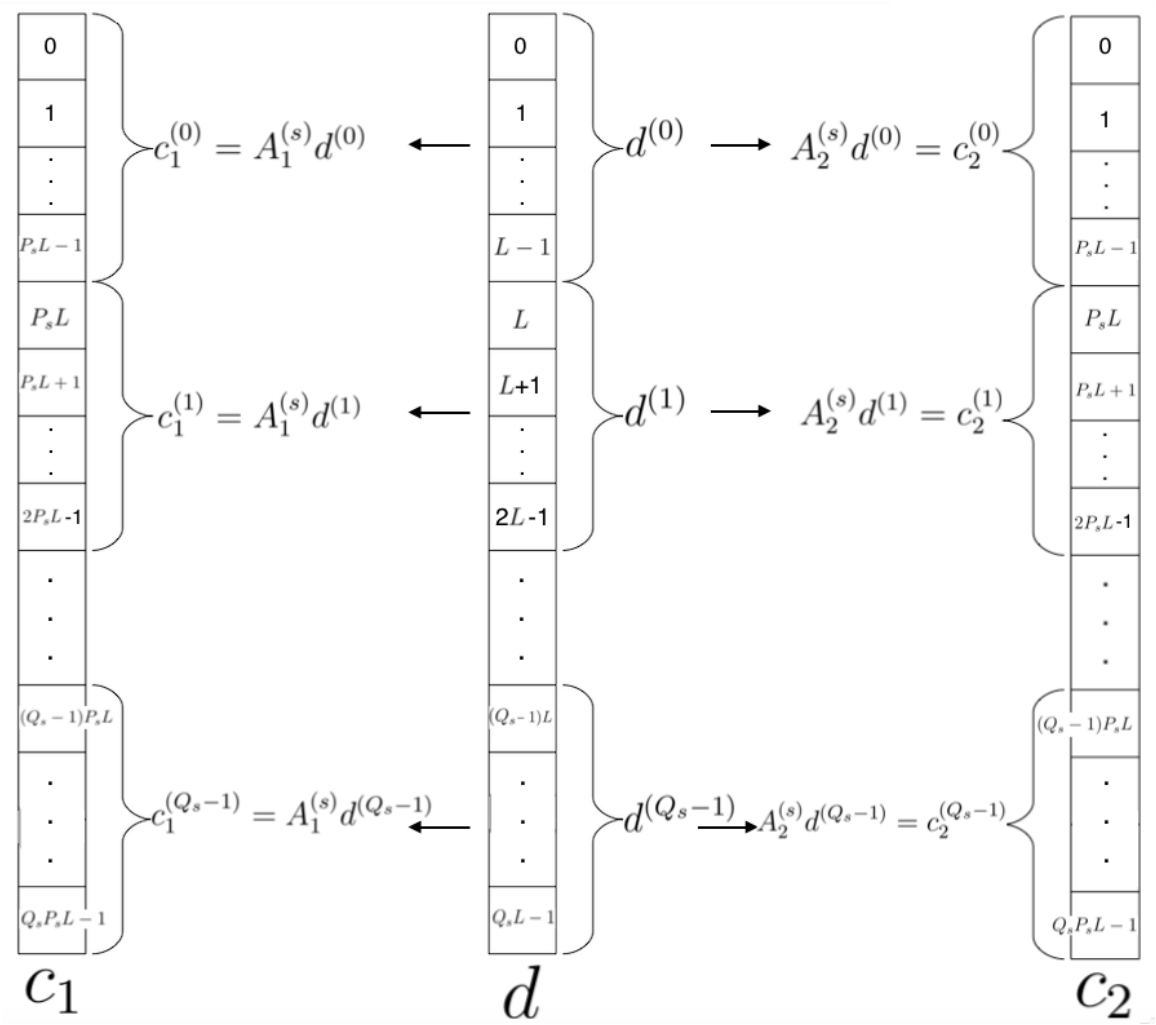


Figura 3.6: Cáculo de los códigos de paridad.

Continuando con el procedimiento de marcado, los vectores c_1 y c_2 se concatenan en el vector c y se permutan pseudo-aleatoriamente para que al insertar la marca la información de un bloque quede distribuida en toda la imagen, tal como se muestra en (3.12).

$$c = Permutar([c_1|c_2], llave) \tag{3.12}$$

El vector c se divide en segmentos $\hat{c}_n$ de tamaño 160. Dichos segmentos se van a usar para calcular el hash criptográfico de cada bloque y su vez van a conformar la marca

que se va a insertar.

$$\hat{c}_n = c[160n : 160(n+1)] \quad \text{con } n = 0, \ldots, (N_B - 1). \tag{3.13}$$

Para generar el hash criptográfico de 32 bits h_n del bloque B_n, se concatenan los 5 MSB del bloque con el vector $\hat{c}_n$ y a eso se le aplica la función $Hash32(.)$, cuyo algoritmo está descrito en 6.

$$h_n = Hash32([5MSB(B_n)|\hat{c}_n]) \tag{3.14}$$

Para insertar la marca, los códigos de paridad $\hat{c}_n$ y el hash criptográfico h_n del bloque B_n se concatenan para formar el vector $u_n = [\hat{c}_n|h_n]$, el cual tiene una longitud de 192 bits (160 de $\hat{c}_n$ y 32 de h_n) y de este vector se van tomando de tres en tres elementos, se convierten a número entero y se suma a los 5 MSB de cada pixel del bloque.

El Algoritmo 5 *InsertaMarcaDWT* describe el pseudo-código del procedimiento de creación e inserción de la marca de agua. En las líneas 3 y 4 se generan dos matrices binarias y pseudo-aleatorias, utilizando la función auxiliar $MatrizBinariaRandom()$, la cual está definida en el Algoritmo 4. Esta función recibe como parámetros L, la *llave* secreta y el *nivel*, puesto que el tamaño de las matrices depende del *nivel*. En las lineas 5 y 6 se obtiene el vector d. Luego, con el vector d se generan los códigos de paridad c_1 y c_2, para ello se utiliza la función descrita en el Algoritmo 3, la cual realiza la multiplicación en módulo 2 de una matriz A que se le pasa como parámetro con cada segmento de longitud L del vector d, el resultado de las multiplicaciones lo va almacenando en un vector, para posteriormente regresarlo. En la línea 10, los códigos de paridad se concatenan y permutan para formar c. En las líneas $11 - 25$, para cada bloque se calcula su hash h_k y se forma el vector $u_k = [\hat{c}_k|h_k]$; finalmente, los elementos de u_k se insertan en los 3 LSB de cada bloque para formar la imagen marcada I_W.

Algoritmo 3: Algoritmo de generación de códigos de paridad.
Entrada: matriz binaria pseudo-aleatoria A, vector de datos d y tamaño de segmento L.
Salida: códigos de paridad c.

GeneraCodigos(A, d, L)
1 $Q \leftarrow \frac{Longitud(d)}{L}$
2 $R \leftarrow$ Número de renglones de A
3 $c \leftarrow$ Vector de tamaño Q*R
4 **para** $i \leftarrow 0$ **hasta** $Q - 1$ **hacer**:
5 $c[iR : (i+1)R] \leftarrow Ad^{(i)} \pmod 2$ \\Donde $d^{(i)} = d[iL : (i+1)L]$
6 **fin para**
7 **regresar** c

Algoritmo 4: Algoritmo para generar una matriz binaria pseudo-aleatoria.
Entrada: tamaño de segmento L, nivel de submuestreo *nivel* y clave secreta *llave*.
Salida: matriz binaria pseudo-aleatoria A.

MatrizBinariaRandom$(L, nivel, llave)$
1 **si** $nivel = 1$ **entonces:**
2 $Ren \leftarrow L$
3 **si no si** $nivel = 2$ **entonces:**
4 $Ren \leftarrow 4L$
5 **si no si** $nivel = 3$ **entonces:**
6 $Ren \leftarrow 16L$
7 **fin si**
8 $A \leftarrow Matriz(Ren, L)$
9 $Semilla \leftarrow llave$
10 **para** $n \leftarrow 0$ **hasta** $Ren - 1$ **hacer**:
11 **para** $m \leftarrow 0$ **hasta** $L - 1$ **hacer**:
12 $A[n][m] \leftarrow GeneradorDeEnteroAleatorio() \,\%2$
13 **fin para**
14 **fin para**
15 **regresar** A

Algoritmo 5: Algoritmo de inserción de la marca de agua.

Entrada: imagen original I, sus dimensiones N y M, el tamaño de segmento L, el nivel de submuestreo *nivel* y las claves secretas $llave1$ y $llave2$.
Salida: imagen marcada I_W.

INSERTAMARCADWT$(I, N, M, L, nivel, llave1, llave2)$

$I_W \leftarrow Matriz(N, M), N_B \leftarrow \frac{NM}{64}, d \leftarrow$ Vector
$c_1 \leftarrow$ Vector , $c_2 \leftarrow$ Vector , $CB \leftarrow \frac{M}{8}$
$A_1 \leftarrow MatrizBinariaRandom(L, nivel, llave1)$
$A_2 \leftarrow MatrizBinariaRandom(L, nivel, llave2)$
Se calcula el vector $\hat{d}$ de acuerdo a (3.7) y (3.8).
$d \leftarrow Permutar(\hat{d}, llave1)$
Se calculan los códigos de paridad c_1 y c_2 .
$c_1 \leftarrow GeneraCodigos(A_1, d, L)$
$c_2 \leftarrow GeneraCodigos(A_2, d, L)$
$c \leftarrow Permutar([c_1|c_2], llave1)$
para cada bloque B_k **hacer**:
  El vector c se divide en segmentos $\hat{c}_k \leftarrow c[160k : 160(k+1)]$
  Se calcula el hash criptográfico $h_k \leftarrow Hash32(5MSB(B_k), \hat{c}_k)$
  $u_k \leftarrow [\hat{c}_k|h_k]$
  $l \leftarrow 0$
  para $n \leftarrow 0$ **hasta** 7 **hacer**:
    para $m \leftarrow 0$ **hasta** 7 **hacer**:
      $i \leftarrow \lfloor \frac{k}{CB} \rfloor$
      $j \leftarrow k\,\%CB$
      $I_W[i+n, j+m] \leftarrow B_k[n, m] + ToInt(u_k[l : l+3])$
      \\$ToInt()$ es una función que convierte un arreglo binario a entero.
      $l \leftarrow l + 3$
    fin para
  fin para
fin para
regresar I_W

Algoritmo 6: Algoritmo del cálculo del hash de 32 bits.
Entrada: vector de datos d y códigos de paridad c.
Salida: hash criptográfico h.

HASH32(d, c)
1 $h \leftarrow$ Vector de tamaño 32
2 $sha \leftarrow$ Vector de tamaño 4
3 $aux \leftarrow [d|c]$
4 $sha \leftarrow SHA32(aux)$ \\SHA32() regresa arreglo de 4 enteros de 8 bits tomados de SHA256($[d|c]$).
5 **para** $i \leftarrow 0$ **hasta** 3 **hacer**:
6 $h[8i : 8(i+1)] \leftarrow ToBin(sha[i])$
7 **fin para**
8 **regresar** h

3.2.2. Proceso de restauración de la imagen

Sea I_T la imagen que resulta de haber manipulado la imagen marcada I_W, para que el proceso de restauración de la imagen I_T se pueda llevar a cabo, se debe tener el *nivel* y las dos claves secretas con los cuales se realiza el marcado de la imagen. La primera parte del proceso de restauración consiste en extraer la marca e información de la imagen manipulada.

Extracción del vector de datos $\tilde{d}$ y de la marca de agua

Primero, se divide I_T en bloques $\tilde{B}_n$ de 8×8 pixeles, con $n \in \{0, \ldots, N_B - 1\}$. Luego, para cada $\tilde{B}_n$ se calcula la sub-banda $LL^{(nivel)}$, con el Algoritmo 1, el resultado se convierte a binario y se coloca en el vector $\tilde{d}_n$.

$$\tilde{d}_n = ToBin(Submuestreo(\tilde{B}_n, nivel)). \tag{3.15}$$

También, para cada bloque se extrae la marca, es decir, los 3 LSB de cada bloque.

Esto se coloca en el vector u_n.

$$u_n = 3LSB(\tilde{B}_n) \quad \forall n \in \{0, \ldots, N_B - 1\} \tag{3.16}$$

De acuerdo con el procedimiento de marcado, u_n contiene los códigos de paridad $\tilde{c}_n = u_n[0 : 160]$ y el hash criptográfico $h_n = u_n[160 : 192]$. Con la información extraída, se procede a la autenticación de los bloques de la imagen.

Detección de bloques manipulados

Con el vector $\tilde{c}_n$ previamente calculado y los 5 MSB de cada bloque bloque se calcula el hash temporal:

$$\tilde{h}_n = Hash32([5MSB(\tilde{B}_n)|\tilde{c}_n]) \tag{3.17}$$

El hash $\tilde{h}_n$ se compara con h_n y si son diferentes, entonces se considera que el bloque $\tilde{B}_n$ está manipulado y se marcan como dañados (con un -1) los bits de $\tilde{c}_n$ y de $\tilde{d}_n$. Una vez que se realiza la detección de los bloques alterados, es posible determinar el porcentaje de bits manipulados, para ello, se determina la cantidad total de bits B_t, la cual está dada por (3.18).

$$B_t = 5NM \tag{3.18}$$

La cantidad de pixeles manipulados se calcula multiplicando la cantidad de bloques manipulados B_m por 320 que es el número total de bits que hay en un bloque. Entonces, el poncentaje de bits manipulados está dado por (3.19).

$$pbm = \frac{320 * B_m}{B_t} * 100 \tag{3.19}$$

Recuperación de los bits manipulados

Una vez que los vectores $\tilde{d}_n$ manipulados ya están marcados, se concatenan en uno solo, al que se va a denotar por $\tilde{d}$

$$\tilde{d} = [\tilde{d}_0|\tilde{d}_1| \ldots |\tilde{d}_{N_B-1}] \tag{3.20}$$

Los códigos de paridad $\tilde{c}_n$ de todos los bloques (manipulados o no) se concatenan en un vector, el cual se va a denotar por $\tilde{c}$ y está expresado en (3.21).

$$\tilde{c} = [\tilde{c}_0|\tilde{c}_1|\ldots|\tilde{c}_{N_B-1}] \tag{3.21}$$

Dado que el vector $\tilde{c}$ contiene ambos conjuntos de códigos de paridad, lo que sigue es deshacer la permutación que se aplicó al insertar la marca y separar dichos conjuntos. Entonces, $Permutar^{-1}(\tilde{c}, llave) = [\bar{c}_1|\bar{c}_2]$, la notación utilizada pretende hacer notar que no son los mismos que se insertaron en la marca, puesto que algunos de sus bits pueden estar marcados como manipulados. Entonces, según el proceso de inserción de la marca, se tiene que:

$$\bar{c}_1 = [\bar{c}_1^{(0)}|\bar{c}_1^{(1)}|\ldots|\bar{c}_1^{(Q_s-1)}]$$
$$\bar{c}_2 = [\bar{c}_2^{(0)}|\bar{c}_2^{(1)}|\ldots|\bar{c}_2^{(Q_s-1)}]$$

El vector $\tilde{d}$ se permuta pseudo-aleatoriamente, puesto que para generar los códigos de paridad en la marca se permutó, así se obtiene el vector $\bar{d} = Permutar(\tilde{d}, llave)$, el cual también contiene bits manipulados. Este vector se divide en segmentos de tamaño L que vamos a denotar por $\bar{d}^{(i)}$.

$$\bar{d}^{(i)} = \bar{d}[iL : (i+1)L] \quad \text{con } i = 0, \ldots, (Q_s - 1) \tag{3.22}$$

Puesto que se conocen las *llaves* secretas con las cuales se generaron las matrices A_1 y A_2 para marcar la imagen, entonces éstas se pueden volver a generar, de tal manera que con $\bar{d}$, $\bar{c}_1$ y $\bar{c}_2$, ya se tienen los sistemas de ecuaciones con los que se generaron los códigos de paridad (con algunos bits manipulados) y, por lo tanto, se puede proceder utilizando (2.19) en los sistemas (3.23) y (3.24).

$$A_1\bar{d}^{(i)} = \bar{c}_1^{(i)} \tag{3.23}$$
$$A_2\bar{d}^{(i)} = \bar{c}_2^{(i)} \tag{3.24}$$

Para solucionar los sistemas de ecuaciones (3.23) y (3.24), primero se hace la separación de los sistemas de acuerdo con lo descrito en (2.17), una vez que se tiene la

separación, el sistema a resolver es el que se presenta en (2.19). El método de solución utiliza la información dada por ambos conjuntos de sistemas de ecuaciones para recuperar los bits dañados. Primero se resuelven los del primer conjunto de sistemas, dados por (3.23) $\forall i \in \{0, 1, \ldots, (Q_s - 1)\}$, los bits recuperados con este conjunto de sistemas se sustituyen en el vector $\bar{d}$. Luego, utilizando el nuevo vector $\bar{d}$, se resuelven los sistemas usando los sistemas descritos en (3.24). Los bits recuperados mediante (3.24) también se sustituyen en $\bar{d}$ y el proceso se repite hasta que el vector $\bar{d}$ ya no presente ningún cambio, pues esto significa que ya no se han recuperado bits manipulados. Al vector $\bar{d}$ que resulta del proceso se va a denotar por $\hat{d}_R$. Como este vector se había permutado previamente, se debe regresar a su orden original, entonces $d_R = Permutar^{-1}(\hat{d}_R, llave)$.

El método utilizado para la solución de los sistemas de ecuaciones se presenta en el Algoritmo 7, los sistemas (3.23) y (3.24) ya separados como en (2.19) se pasan como parámetros al Algoritmo 7, el cual realiza el método de Gauss-Jordan para sistemas de ecuaciones binarios (línea 4), al hacer las operaciones modifica directamente la matriz y el vector del sistema y calcula el rango de la matriz.

Sea $Ax = b$ el sistema que se va a resolver, donde el número de renglones de A es F y las columnas de A es G, si el rango de la matriz A es igual al número de columnas de la misma, entonces las ecuaciones son linealmente independientes y el vector x tiene solución única, por lo tanto, es posible restaurar todos los bits manipulados, en este caso, se dice que A es de rango completo. Al realizar el método de Gauss-Jordan, el sistema queda de la forma (3.25), donde $\mathbf{I}$ denota la matriz identidad y $\hat{b}$ es el vector solución.

$$\mathbf{I}\hat{x} = \hat{b} \tag{3.25}$$

En la línea 33 del Algoritmo 7, se contempla ese caso. Si el rango de A es menor que las columnas de A, entonces los renglones del sistema son linealmente dependientes y

al aplicar Gauss-Jordan, queda una matriz de la forma:

$$\left[\begin{array}{c|c} \mathbf{I} & C \\ \hline 0's & 0's \end{array}\right] \left[\begin{array}{c} \hat{x} \\ \hline \tilde{x} \end{array}\right] = \left[\begin{array}{c} \hat{b} \\ \hline 0's \end{array}\right] \tag{3.26}$$

donde $\hat{x}$ son las variables pivote, $\tilde{x}$ son las variables libres y $\hat{b}$ denota el vector b después de las operaciones de Gauss-Jordan. A pesar de que el sistema (3.26) es linealmente dependiente, es posible recuperar algunos de los bits manipulados, utilizando (3.27).

$$\hat{x} + C\tilde{x} = \hat{b} \tag{3.27}$$

Lo anterior se puede despejar y queda (3.28).

$$\hat{x} = \hat{b} - C\tilde{x} \tag{3.28}$$

Dado el sistema (3.28), se puede generar un conjunto de soluciones si se le dan valores a las variables libres $\tilde{x}$, cada una de ellas toma valores en el conjunto $\{0, 1\}$. Entonces, si se tienen $t = F - rango(A)$ ecuaciones linealmente dependientes, se pueden generar 2^t soluciones posibles para $\tilde{x}$ y a partir de ellas calcular $\hat{x}$ con (3.28). Las variables del vector $\hat{x}$ que permanecen iguales con cada una de las soluciones generadas para $\tilde{x}$, son solución del sistema, pues satisfacen el sistema y no dependen del valor que tomen las variables libres. En las líneas $10 - 30$ del Algoritmo 7 se considera este caso. Cabe recalcar que esta parte del algoritmo crece de manera exponencial, entonces para tamaños de L grandes, el problema se vuelve ineficiente en tiempo. También se puede volver ineficiente si se consideran porcentajes de α que sobrepasen por mucho el α máximo encontrado de manera experimental para este método redundante.

Algoritmo 7: Algoritmo de resolución del sistema de ecuaciones.
Entrada: la matriz del sistema A, el vector de términos independientes b y el tamaño del sistema ren y col.
Salida: la solución del sistema sol.

```
ResolverSistema(A, b, ren, col)
 1  rango ← 0, bin ← Vector
 2  orden, xsol, b2, sol2, sol ← Vectores de tamaño col
 3  Se crea el vector orden tal que: orden[i] ← i
 4  Se aplica el método de Gauss-Jordan binario al sistema.
 5  Se obtiene el rango de la matriz A usando lo anterior.
 6  Nd ← col − rango
 7  Ns ← 2^Nd
 8  Se obtiene la matriz C de tamaño rango × Nd, de acuerdo a (3.26).
 9  C ← A[0 : rango, rango : col]
10  si Ns > 1 entonces:
11    para iter ← 0 hasta Ns hacer:
12      bin ← ToBin(iter)
13      xsol[rango : col] ← bin
14      xsol[0 : rango] ← Cbin
15      para k ← 0 hasta col hacer:
16        xsol[k] ← (xsol[k] + b[k]) %2
17      fin para
18      si iter = 0 entonces:
19        para k ← 0 hasta col hacer:
20          sol[orden[k]] ← xsol[k]
21        fin para
22      si no :
23        para k ← 0 hasta col hacer:
24          sol2[orden[k]] ← xsol[k]
25        fin para
26        para j ← 0 hasta col hacer:
27          si sol[j] ≠ sol2[j] entonces:
28            sol[j] ← −1
29        fin para
30    fin para
31  si no :
32    para k ← 0 hasta col hacer:
33      sol[k] ← b[k]
34    fin para
35  fin si
36  regresar sol
```

Restauración de la imagen

Para la restauración de la imagen, si un bloque no fue marcado como manipulado, entonces se copia igual de la imagen I_T a la imagen restaurada I_R. Si un bloque ha sido marcado como manipulado, recordemos que d_R contiene los bits que corresponden a los bloques submuestreados de la imagen, entonces se tienen tres posibilidades, según el nivel que se haya elegido:

- **Nivel 1:** dado que cada bloque comprimido corresponde a un segmento de tamaño 80 de d_R, entonces para sustituir el bloque n por su versión comprimida, se genera un bloque auxiliar de 4×4, se extraen de d_R sus bits correspondientes $v_n = d_R[80n : 80(n+1)]$ y los pixeles se forman tomando de 5 en 5 bits de v_n y convirtiéndolos a un número entero, es decir, $BinToInt(v_n[5n : 5(n+1)])$ donde $n = 0, 1, \ldots, 15$. Una vez que el bloque de 4×4 tiene todos sus valores, se le aplica la función *Sobremuestrear* 2 y lo obtenido se sustituye en el bloque manipulado.

- **Nivel 2:** en este caso, cada bloque comprimido corresponde a un segmento de 20 bits de d_R, entonces se genera un bloque auxiliar de 2×2, se extraen de d_R sus bits correspondientes $v_n = d_R[20n : 20(n+1)]$, los pixeles se forman tomando de 5 en 5 bits y convirtiéndolos a un número entero, es decir, $BinToInt(v_n[5n : 5(n+1)])$, donde $n = 0, 1, \ldots, 3$. Cada número se coloca en el bloque de 2×2 y se le aplica la función *Sobremuestrear*, de acuerdo al Algoritmo 2. Luego, el bloque obtenido se sustituye en el bloque manipulado.

- **Nivel 3:** para este nivel, un bloque comprimido corresponde a un segmento de 5 bits de d_R, así que se requiere un sólo pixel, se extraen de d_R sus bits correspondientes $v_n = d_R[5n : 5(n+1)]$ y se convierte a número entero: $BinToInt(v_n)$, este número se coloca en un bloque de 1×1 y se aplica la función *Sobremuestrear* 2. Lo obtenido se reemplaza por el bloque manipulado.

En el Algoritmo 8, llamado *RestauraImagenDWT* se muestra el pseudo-código para la restauración de la imagen manipulada. Este algoritmo recibe la imagen posiblemente manipulada I_T, L, el *nivel*, la *llave*1 y la *llave*2, estos últimos deben ser iguales a los que se utilizaron para la inserción de la marca. El algoritmo devuelve la imagen restaurada I_R. En la línea 6 se calculan los vectores $\tilde{d}_k$, de la línea 7 a la 9 se extrae la marca de los 3 LSB de los bloques y se genera el nuevo hash. Luego, en las líneas $10-12$ se hace la detección de bloques manipulados y se marcan los segmentos correspondientes de $\tilde{d}_n$ y $\tilde{c}_n$. De la línea 17 a la 23 corresponde a la recuperación de los bits dañados y se resuelven los sistemas de ecuaciones correspondientes, la función $ResolverSistema()$ recibe los argumentos A, d, c, *nivel* y L, regresa el vector d_R con los bits restaurados ya incluidos. Finalmente, las líneas $26-37$ hacen la reconstrucción de la imagen. Si un bloque fue alterado se obtiene su segmento correspondiente del vector d_R, los bits se convierten a enteros y se colocan en un bloque auxiliar B_{aux}, este se reconstruye con la función $Descompresión(B_{aux}, nivel)$ y se coloca en el lugar del bloque manipulado. Si el bloque no fue manipulado, se copia de la imagen I_T.

Para los niveles 1 y 2, una vez que el proceso de restauración termina, si los porcentajes de restauración son mayores a 90 y menores estrictamente a 100, se aplica un filtro de mediana, puesto que para estos porcentajes es posible mejorar la apariencia de las imágenes restauradas. Cuando se logra restaurar un porcentaje de bits manipulados mayor a 90 y menor a 100, los bits que no se restauraron se perciben como ruido de tipo sal y pimienta, entonces, es posible quitarlo con un filtro de mediana. Para menores porcentajes de restauración, las imágenes lucen casi de color constante, como se verá en el capítulo de resultados, y el filtro puede empeorar la calidad de la imagen restaurada.

Algoritmo 8: Algoritmo de restauración de la imagen.
Entrada: imagen posiblemente manipulada I_T, el tamaño de segmento L, el nivel $nivel$ y las claves secretas $llave1$ y $llave2$.
Salida: imagen restaurada I_R.

RestauraImagenDWT$(I_T, L, nivel, llave1, llave2)$

$I_R \leftarrow Matriz(N, M), N_B \leftarrow \frac{NM}{64}, CB \leftarrow \frac{M}{8}, d_R \leftarrow$ Vector
$\bar{c}_1 \leftarrow$ Vector , $\bar{c}_2 \leftarrow$ Vector
$A_1 \leftarrow MatrizBinariaRandom(L, nivel, llave1)$
$A_2 \leftarrow MatrizBinariaRandom(L, nivel, llave2)$
para cada bloque $\tilde{B}_k$ **hacer**:
  Se calcula el vector $\tilde{d}_k$, de acuerdo con (3.15).
  Se extrae la marca de agua $u_k \leftarrow 3LSB(\tilde{B}_k)$
  Se obtienen $\tilde{c}_k \leftarrow u_k[0:160]$ y $h_k \leftarrow u_k[160:192]$
  Se calcula el nuevo hash $\tilde{h}_k$ de acuerdo con (3.17)
  si $h_k \neq \tilde{h}_k$ **entonces:**
    $\tilde{B}_k, \tilde{c}_k, \tilde{d}_k$ se marcan como manipulados.
  fin si
  $\tilde{c} \leftarrow [\tilde{c}|\tilde{c}_k], \quad \tilde{d} \leftarrow [\tilde{d}|\tilde{d}_k]$
fin para
$\bar{d} \leftarrow Permutar(\tilde{d}, llave1)$
$[\bar{c}_1|\bar{c}_2] \leftarrow Permutar^{-1}(\tilde{c}, llave1)$
mientras $\bar{d} \neq \hat{d}_R$ **hacer:**
  para $i \leftarrow 0$ **hasta** $(Q_s - 1)$**hacer**:
    Se separan los sistemas de acuerdo a (2.19).
    $\hat{d}_R^{(i)} \leftarrow ResolverSistema(A_1, \bar{d}^{(i)}, \bar{c}_1^{(i)}, nivel, L)$
    $\bar{d}^{(i)} \leftarrow ResolverSistema(A_2, \hat{d}_R^{(i)}, \bar{c}_2^{(i)}, nivel, L)$
  fin para
fin mientras
$d_R \leftarrow Permutar^{-1}(\hat{d}_R, llave1), r \leftarrow \frac{320}{2^{2*nivel}}$
Se calcula el porcentaje de bits restaurados pbr
para cada bloque $\tilde{B}_k$ **hacer**:
  si $\tilde{B}_k$ fue manipulado **entonces:**
    $v_k \leftarrow d_R[rk : r(k+1)]$ \\El tamaño del vector v_k depende del nivel.
    Los bits de v_k (tomados de 5 en 5) se convierten a entero y se colocan en B_{aux}
    $B_k \leftarrow Sobremuestreo(B_{aux}, nivel)$
    Se sustituye B_k en el bloque correspondiente de I_R
  si no:
    Se sustituye el bloque correspondiente de I_T en I_R
  fin si
fin para
Si $nivel = 1, 2$ y $90 \leq pbr < 100$, se aplica un filtro de mediana.
regresar I_R

3.3. Conclusiones del capítulo

Al utilizar el wavelet de Haar para calcular la sub-banda $LL^{(nivel)}$ de los bloques de la imagen y, con ello, representarla con una menor cantidad de bits, el cálculo se reduce a obtener los promedios de cada cuatro pixeles del bloque, es decir, a realizar un submuestreo de la imagen. Así mismo, la reconstrucción de los bloques se puede ver como un sobremuestreo. Esto representa una ventaja de eficiencia respecto a otras transformadas, lo cual influye tanto en el tiempo de marcado como en el tiempo de restauración de la imagen. Por otra parte, el hecho de aplicar el submuestreo a los bloques de 8×8 de la imagen, explica porqué únicamente se utilizan tres niveles de la DWT.

Dado que el esquema de marcado *MarcaDWT* se planteó como una marca de agua frágil, esto significa que cualquier alteración de la imagen destruye parcial (en bloques de 8×8) o totalmente la marca de agua.

Para la recuperación de los pixeles manipulados, el método con redundancia propuesto en [Bravo-Solorio *et al.*, 2018] está planteado para utilizar tamaños de segmento L pequeñas, del orden de $L < 64$, puesto que una parte de la solución de los sistemas es de orden exponencial y una L mayor representaría un aumento significativo en el tiempo de recuperación.

En el primer nivel de marcado, para llevar a cabo la formulación redundante, ambas matrices A_1 y A_2 tienen dimensiones $L \times L$. Para este nivel se espera que se pueda restaurar más del 50 % de una imagen manipulada. Para el segundo nivel, se tiene que las matrices A_1 y A_2 deben tener dimensiones $4L \times L$ y se espera que se logre restaurar una imagen que tenga más del 80 % de manipulación. En el tercer nivel, ambas matrices A_1 y A_2 tienen dimensiones de $16L \times L$. Para este nivel, se puede esperar una restauración de más del 94 % de los pixeles de una imagen manipulada, entonces este esquema puede restaurar un mayor porcentaje de bits manipulados, respecto a los métodos de marcado citados en el estado del arte de esta tesis, lo

cual representa una contribución importante en el área de restauración de imágenes digitales con marcas de agua.

Conforme se incrementa el nivel de compresión, la calidad de la imagen restaurada va a decrecer, pues se utiliza una menor cantidad de bits para representar la imagen y, por ello, la pérdida de información de la imagen es mayor.

Capítulo 4

Pruebas y resultados

Para mostrar el funcionamiento del esquema *MarcaDWT* propuesto en esta tesis, se realizaron dos tipos de pruebas, las primeras consisten en aplicar el esquema de marcado y restauración a imágenes de manera individual, utilizando un nivel distinto para cada una. El otro tipo de pruebas se realizaron con 100 imágenes y se obtuvieron datos generales como el porcentaje promedio de bits recuperados, PSNR promedio de las imágenes restauradas, entre otros. Las pruebas se realizaron en una computadora MacBook Air, con procesador Intel Core i5 a 2.9 GHz y memoria RAM de 8 GB 1600 MHz DDR3. La implementación de los algoritmos fue hecha en `C++`.

En la Figura 4.1(a) se muestra una imagen de dimensiones 512×512, marcada con el **nivel 1** del esquema *MarcaDWT* y utilizando $\mathbf{L = 32}$, el PSNR de la imagen marcada es de 37.9867 dB y el tiempo de inserción de la marca fue de 0.29 segundos. Ésta se manipuló pegando la imagen de un oso visto de cerca, en la Figura 4.1(b), se presenta la imagen manipulada. La Figura 4.1(c) señala en color gris los bloques que fueron detectados como manipulados. El porcentaje de manipulación de la imagen es de $\alpha = \mathbf{61.0352}\,\%$ del total de sus pixeles. En la Figura 4.1(d) se puede ver el resultado de la restauración de la imagen manipulada para el nivel 1 del esquema, se logró restaurar el $\mathbf{100}\,\%$ de los bits marcados como manipulados. La imagen obtenida tiene un PSNR= 30.3924 dB y el tiempo de restauración fue de 84.80 segundos.

(a) ***Imagen marcada.***

(b) ***Imagen manipulada.***

(c) ***Pixeles detectados como manipulados.***

(d) ***Imagen restaurada.***

Figura 4.1: Restauración de la imagen con el esquema *MarcaDWT* de nivel 1 para una tasa de manipulación $\alpha = 0.61$ y $L = 32$.

Dado que en el nivel 1, para distintos tamaños de L se obtuvieron buenos resultados, se va a hacer una discusión sobre cuál L es mejor. Entonces, para el **nivel 1** del esquema, pero ahora utilizando $\mathbf{L = 20}$, se marcó la misma imagen para comparar, la cual se muestra en la Figura 4.2. El PSNR de la imagen marcada es de 37.9877 dB, y el tiempo de inserción de la marca fue de 0.27 segundos. La Figura fue manipulada de igual manera que la anterior, pero con mayor porcentaje de daño y se puede observar en 4.3(a). La Figura 4.3(b) presenta en color gris los bloques que fueron detectados como manipulados. El porcentaje de manipulación de la imagen es de $\alpha = \mathbf{63.9648}\,\%$. En la Figura 4.3(c) se puede ver el resultado de la restauración de la imagen manipulada, se logró restaurar el 99.9652 % de los bits marcados como dañados. La imagen obtenida tiene un PSNR= 30.0329 dB y el tiempo de restauración fue de 19.25 segundos. Dado que no se restauró el 100 % de los bits manipulados, se puede observar en la imagen restaurada un ruido de tipo sal y pimienta, pero como $90 < 99.9652 < 100$, entonces el esquema aplica un filtro de mediana, cuyo resultado se ve en 4.3(d). Por lo tanto, tomando en cuenta que para $\mathbf{L = 20}$, el tiempo de restauración fue cuatro veces menor y que es posible restaurar más porcentaje de la imagen manipulada, se puede considerar que $L = 20$ otorga mejores resultados.

Figura 4.2: Imagen marcada con el esquema *MarcaDWT* de nivel 1 con $L = 20$.

(a) ***Imagen manipulada.***

(b) ***Pixeles detectados como manipulados.***

(c) ***Imagen restaurada.***

(d) ***Imagen restaurada filtrada.***

Figura 4.3: Restauración de la imagen con el esquema *MarcaDWT* de nivel 1 para una tasa de manipulación $\alpha = 0.64$ y $L = 20$.

La Figura 4.4(a), cuyas dimensiones son 512×512 pixeles, fue marcada con el **nivel 2** del esquema *MarcaDWT* y se utilizó un tamaño de segmento de $L = 20$. El tiempo de marcado de la imagen fue de 0.26 segundos y tiene un PSNR de $= 37.9256$ dB. Como manipulación, se pegó la cara de un koala, la imagen alterada se puede ver en la Figura 4.4(b). En la Figura 4.4(c) se observa en color gris los bloques de la imagen que fueron detectados como manipulados, dicha área gris representa el 85.8887 % de los pixeles de la imagen. En la Figura 4.4(d) se puede ver el resultado de la restauración de la imagen manipulada para el nivel 2 del esquema; para el 85.8887 % de manipulación, se logró restaurar el 100 % de los bits marcados como dañados. La imagen restaurada tiene un PSNR= 26.2328 dB y el tiempo de restauración fue de 4.27 segundos.

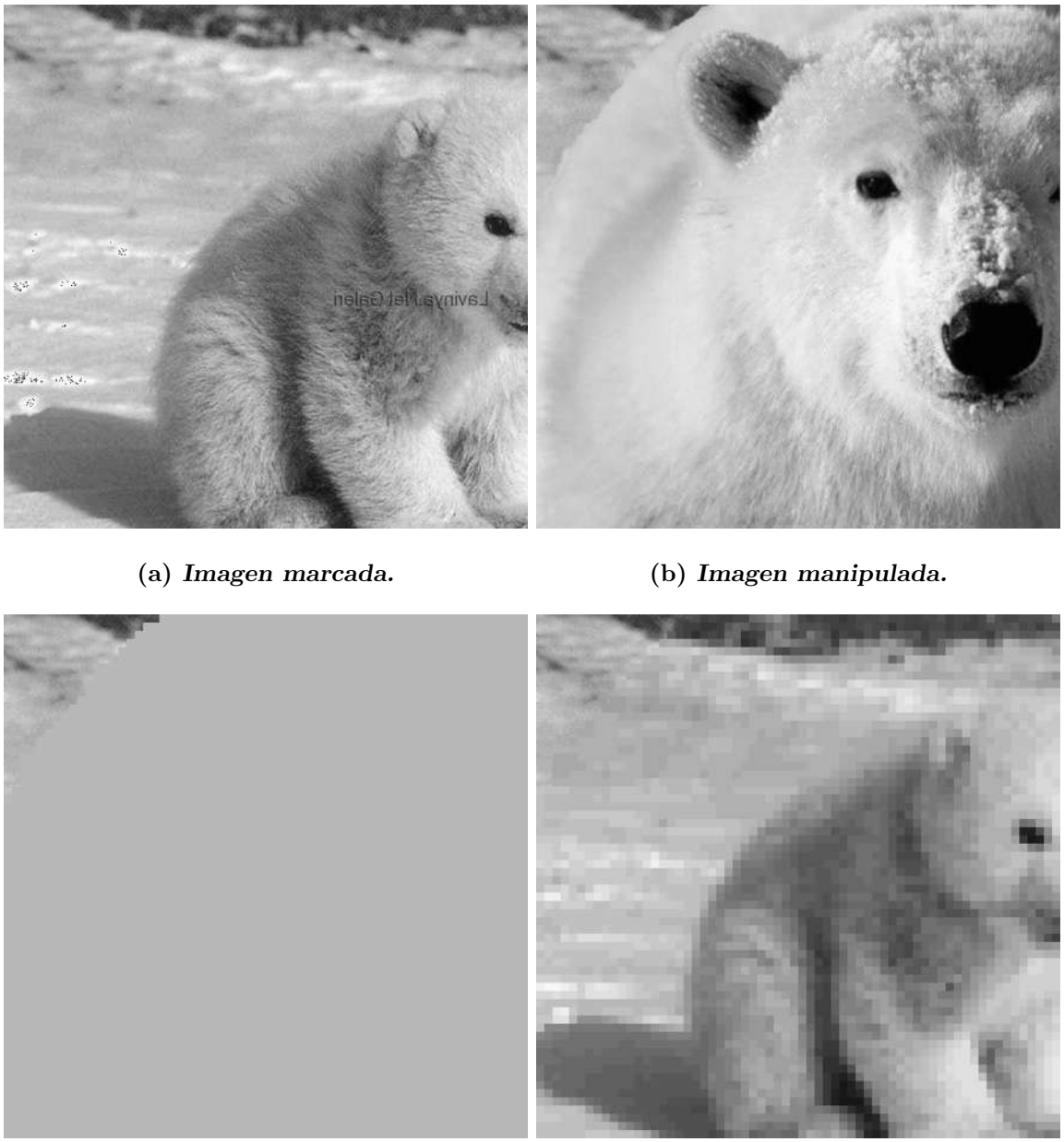
En la Figura 4.5(a), se presenta una imagen de tamaño 512×512 pixeles que fue marcada con el esquema *MarcaDWT* usando el **nivel 3** y $L = 16$. El tiempo de marcado de la imagen fue de 0.28 segundos y tiene un PSNR= 37.9269 dB. En la Figura 4.5(b), se muestra la imagen manipulada, a la cual se le pegó la cara de un oso polar. En la Figura 4.5(c) se puede observar en color gris los bloques de la imagen que fueron detectados como manipulados, dicha área gris representa el 95.4102 % de los pixeles de la imagen. En la Figura 4.5(d) se puede ver el resultado de la restauración de la imagen manipulada para el nivel 3 del esquema, para dicho porcentaje de manipulación, se logró restaurar el 100 % de los bits marcados como dañados. La imagen restaurada tiene un PSNR= 25.3667 dB y el tiempo de restauración fue de 0.67 segundos.

Como se puede apreciar, en cada nivel, el método propuesto hizo la detección de los bloques manipulados de manera acertada y, más aún, recuperó correctamente los pixeles manipulados de la imagen. Los PSNR de las imágenes restauradas son mayores a 20 dB, por lo tanto, su calidad es aceptable. Dependiendo el nivel, la calidad de la imagen restaurada varía, entre mayor es el nivel de compresión, la calidad de la imagen restaurada disminuye, pero la capacidad de restauración del método aumenta.

(a) ***Imagen marcada.*** (b) ***Imagen manipulada.***

(c) ***Pixeles detectados como manipulados.*** (d) ***Imagen restaurada.***

Figura 4.4: Restauración de la imagen con el esquema *MarcaDWT* de nivel 2 para una tasa de manipulación $\alpha = 0.86$ y $L = 20$.



(a) ***Imagen marcada.*** **(b)** ***Imagen manipulada.***

(c) ***Pixeles detectados como manipulados.*** **(d)** ***Imagen restaurada.***

Figura 4.5: Restauración de la imagen con el esquema *MarcaDWT* de nivel 3 para una tasa de manipulación $\alpha = 0.95$ y $L = 16$.

4.1. Pruebas Generales

Para las pruebas generales se utilizaron 100 imágenes de tamaño 512×512 pixeles, en escala de grises. Para cada nivel del esquema, se marcaron todas las imágenes variando el tamaño de segmento L, se utilizaron los valores $L = 8, 16, 20, 32$, de tal manera que se generaron 400 imágenes marcadas para cada nivel, es decir, en total 1200 imágenes. Cada una de las imágenes marcadas se manipularon con las siguientes tasas: $\alpha = 0.3, 0.4, 0.5, 0.6, 0.61, 0.64, 0.65, 0.7, 0.8, 0.85, 0.86, 0.9, 0.95, 0.96$. La manipulación consistió en obtener el número de pixeles de la imagen que corresponde al porcentaje de daño y colocar el valor de 0 en esa cantidad de pixeles, recorriendo la imagen por renglones. Posteriormente, se procedió a la restauración de las imágenes manipuladas. Los aspectos de la marca que se van a considerar para medir su desempeño son: el tiempo de marcado, el PSNR de las imágenes marcadas, el porcentaje de bits recuperados, el tiempo de restauración de la marca y el PSNR de las imágenes restauradas. Para calcular el PSNR de las imágenes restauradas se consideró toda la imagen, no únicamente el área manipulada.

En la Tabla 4.1 se muestra el tiempo promedio de marcado de las 100 imágenes, dado el nivel y L. El tiempo de marcado es importante para cierto tipo de aplicaciones como vídeos, en los que se requiere marcar varias imágenes por segundo. Se puede observar que el tiempo de marcado, en general, no rebasa los 0.30 segundos.

L	Nivel 1	Nivel 2	Nivel 3
8	0.239	0.217	0.208
16	0.256	0.229	0.219
20	0.268	0.241	0.232
32	0.280	0.258	0.247

Tabla 4.1: Tiempo promedio de marcado en segundos.

En la Tabla 4.2 se señalan los PSNR promedio las imágenes marcadas para cada nivel, para calcularlo se utilizó la imagen original. El PSNR promedio de cada nivel es aproximadamente de 37.9 dB, lo cual coincide con el análisis hecho en [Zhang *et al.*, 2011b], en el cual establecen que para una imagen con marca de agua que se inserta en los 3 LSB del dominio espacial, se tiene un $PSNR \approx 37.8$ dB, si la distribución de la información empleada como marca es uniforme. Como se puede evidenciar en las Figuras 4.1(b), 4.2, 4.4(b) y 4.5(b) la marca es imperceptible al ojo humano y no degradan significativamente la calidad de la imagen.

L	Nivel 1	Nivel 2	Nivel 3
8	37.8832	37.8890	37.8824
16	37.8397	37.8959	37.8845
20	37.8409	37.8864	37.8850
32	37.8396	37.8960	37.8842

Tabla 4.2: PSNR promedio de las imágenes marcadas en dB.

En las Tablas 4.3 y 4.4 se presentan los promedios de los porcentajes de bits restaurados para cada nivel, dados los valores de L y de α. Se resaltaron en rojo los mejores resultados que se obtuvieron en relación con el porcentaje de bits restaurados y el tiempo de restauración, para cada nivel. En las Tablas 4.5 y 4.6 se presentan los promedios de los tiempos de restauración dados el nivel, L y α. Se muestran en color rojo los tiempos correspondientes a los porcentajes de restauración seleccionados en las tablas anteriores. Los resultados de las cuatro tablas se analizan a continuación.

L	Nivel	Tasa de manipulación α						
		0.3	0.4	0.5	0.6	0.61	0.64	0.65
8	1	99.95 %	99.76 %	98.70 %	85.98 %	77.90 %	65.58 %	57.78 %
	2	100 %	100 %	100 %	100 %	100 %	100 %	100 %
	3	100 %	100 %	100 %	100 %	100 %	100 %	100 %
16	1	100 %	100 %	99.99 %	99.89 %	99.86 %	99.66 %	37.06 %
	2	100 %	100 %	100 %	100 %	100 %	100 %	100 %
	3	100 %	100 %	100 %	100 %	100 %	100 %	100 %
20	**1**	100 %	100 %	100 %	99.98 %	99.97 %	**99.95 %**	19.82 %
	2	100 %	100 %	100 %	100 %	100 %	100 %	100 %
	3	100 %	100 %	100 %	100 %	100 %	100 %	100 %
32	1	100 %	100 %	100 %	100 %	**100** %	8.79 %	4.55 %
	2	100 %	100 %	100 %	100 %	100 %	100 %	100 %
	3	100 %	100 %	100 %	100 %	100 %	100 %	100 %

Tabla 4.3: Porcentajes promedio de los bits recuperados para $0.3 \leq \alpha \leq 0.65$.

L	Nivel	Tasa de manipulación α						
		0.7	0.8	0.85	0.86	0.9	0.95	0.96
8	1	37.15 %	12.06 %	7.27 %	6.85 %	4.15 %	1.87 %	1.22 %
	2	99.99 %	99.64 %	97.37 %	95.84 %	27.94 %	4.69 %	2.38 %
	3	100 %	100 %	100 %	100 %	100 %	99.45 %	68.96 %
16	1	9.64 %	0 %	0 %	0 %	0 %	0 %	0 %
	2	100 %	100 %	99.97 %	99.95 %	3.43 %	0 %	0 %
	3	100 %	100 %	100 %	100 %	100 %	**100 %**	9.54 %
20	1	5.05 %	0 %	0 %	0 %	0 %	0 %	0 %
	2	100 %	100 %	100 %	**100 %**	1.22 %	0 %	0 %
	3	100 %	100 %	100 %	100 %	100 %	**100** %	4.30 %
32	1	0 %	0 %	0 %	0 %	0 %	0 %	0 %
	2	100 %	100 %	100 %	9.58 %	0 %	0 %	0 %
	3	100 %	100 %	100 %	100 %	100 %	**100** %	0 %

Tabla 4.4: Porcentajes promedio de los bits recuperados para $0.7 \leq \alpha \leq 0.96$.

L	Nivel	Tasa de manipulación α						
		0.3	0.4	0.5	0.6	0.61	0.64	0.65
8	**1**	1.18	1.35	1.84	1.89	3.04	4.31	2.96
	2	0.38	0.42	0.46	0.49	0.52	0.68	0.73
	3	0.31	0.32	0.33	0.30	0.30	0.34	0.34
16	**1**	0.86	1.16	1.38	3.14	3.30	3.79	5.96
	2	0.41	0.42	0.41	0.51	0.55	0.55	0.54
	3	0.34	0.35	0.37	0.36	0.37	0.41	0.44
20	**1**	0.89	0.90	1.23	4.66	5.21	**21.09**	49.84
	2	0.49	0.51	0.48	0.59	0.60	0.59	0.59
	3	0.35	0.36	0.38	0.41	0.43	0.43	0.44
32	**1**	1.01	1.03	1.91	103.58	**136.05**	187.94	227.81
	2	0.56	0.57	0.59	0.57	0.63	0.63	0.62
	3	0.60	0.60	0.54	0.54	0.61	0.62	0.65

Tabla 4.5: Tiempos promedio de restauración en segundos para para $0.3 \leq \alpha \leq 0.65$.

L	Nivel	Tasa de manipulación α						
		0.7	0.8	0.85	0.86	0.9	0.95	0.96
8	**1**	2.99	3.78	4.17	4.40	6.15	6.07	7.02
	2	0.75	0.71	1.38	1.43	1.51	1.76	1.72
	3	0.29	0.33	0.34	0.35	0.35	0.37	0.59
16	**1**	14.43	26.53	30.25	35.19	29.46	27.58	26.35
	2	0.46	0.56	1.58	1.99	4.80	17.60	34.43
	3	0.38	0.29	0.28	0.34	0.33	**0.39**	3.74
20	**1**	71.38	82.09	88.84	90.20	94.79	96.99	111.35
	2	0.75	2.34	4.16	**6.35**	34.29	128.48	938.7
	3	0.44	0.46	0.47	0.53	0.59	0.61	16.72
32	**1**	475.65	756.56	1346.84	1358.96	1446.84	1635.89	2068.6
	2	0.96	2.94	10.24	39.97	372.47	706.26	987.96
	3	0.61	0.64	0.62	0.69	0.77	9.82	105.15

Tabla 4.6: Tiempos promedio de restauración en segundos para para $0.7 \leq \alpha \leq 0.96$.

De acuerdo con lo presentado en las Tablas 4.3 y 4.4, para el **Nivel 1** con $L = 32$, se logra restaurar el 100 % de los bits dañados si la imagen está manipulada a lo más un $\alpha = 0.61$, sin embargo, si se utiliza $L = 20$, es posible restaurar hasta el 99.95 % de los bits manipulados para una tasa de daño de $\alpha = 0.64$. Los pixeles que no se restauran se solucionan con el filtro de mediana, dando una imagen prácticamente idéntica a la que se obtiene con $L = 32$. En el **Nivel 2**, para $L = 20$, si la tasa de daño es de a lo más 0.86 es posible restaurar el 100 % de los bits manipulados. Los demás valores de L ofrecen un menor porcentaje de restauración. Para el **Nivel 3**, utilizando $L = 16, 20$ y 32, para una tasa de manipulación de $\alpha = 0.95$ es posible restaurar el 100 % de los bits dañados, sin embargo, $L = 16$ es el que hace la restauración en el menor tiempo.

Respecto a los tiempos de restauración que se presentan en las Tablas 4.5 y 4.6, se puede notar cómo el tiempo se incrementa significativamente cuando se acerca a las tasas máximas de manipulación de cada nivel, de manera práctica se observó que el método realiza más iteraciones cuanto más se acerca a dichos valores. Además, entre más grande es la L, el tiempo de restauración aumenta de manera, de hecho, el tiempo fue el motivo por el cual no se consideraron L más grandes. Para el nivel 1, el esquema tarda 21.09 segundos en restaurar el 99.95 % de los bits dañados, para una tasa de manipulación $\alpha = 0.64$ y $L = 20$, lo cual es aproximadamente una sexta parte del tiempo de restauración con los parámetros $\alpha = 0.61$ y $L = 32$; para el nivel 2, le toma 6.35 segundos restaurar el total de los bits dañados si se tiene una tasa de $\alpha = 0.86$ y una $L = 20$. Finalmente, para el nivel 3, con $L = 16$ le toma 0.39 segundos restaurar todos los bits dañados de una imagen manipulada en un 95 %.

Se puede observar en ambas tablas el efecto que causa el tamaño de segmento L en el proceso de restauración cuando se utiliza la formulación redundante (3.11), propuesta en Bravo-Solorio *et al.*, 2018, si la L es pequeña se logra recuperar un menor porcentaje de bits dañados, comparado con las otras L, incluso, en el caso del nivel 1 con $\alpha = 0.3$ no fue posible restaurar más del 99.95 %. Por otro lado, para

la L más grande considerada $L = 32$, tampoco se obtuvieron los mejores resultados, pues para los niveles 1 y 2, se puede restaurar el 100 % de los bits, pero para tasas ligeramente inferiores a la máxima considerada en cada nivel, 0.61 en el caso del nivel 1 y 0.85 en el caso del nivel 2. En cuanto al tiempo, sí se aprecia un incremento en el tiempo de restauración, conforme L crece.

En la Figura 4.6, se muestra la gráfica de los PSNR promedio de las imágenes restauradas. Para ello, únicamente se tomaron en cuenta los porcentajes de α en donde el esquema es capaz de restaurar un porcentaje aceptable de los bits manipulados. Los valores de L son los que dieron mejores resultados de manera experimental para cada nivel. En el nivel 1 (línea roja), se consideró $L = 20$ y $\alpha = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.64$. Para el nivel 2 (línea azul), se utilizó $L = 20$ y y $\alpha = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.64, 0.7, 0.8, 0.86$. Finalmente, para el nivel 3 (línea negra), se utilizó $L = 16$ y $\alpha = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.64, 0.7, 0.8, 0.86, 0.9, 0.95$.

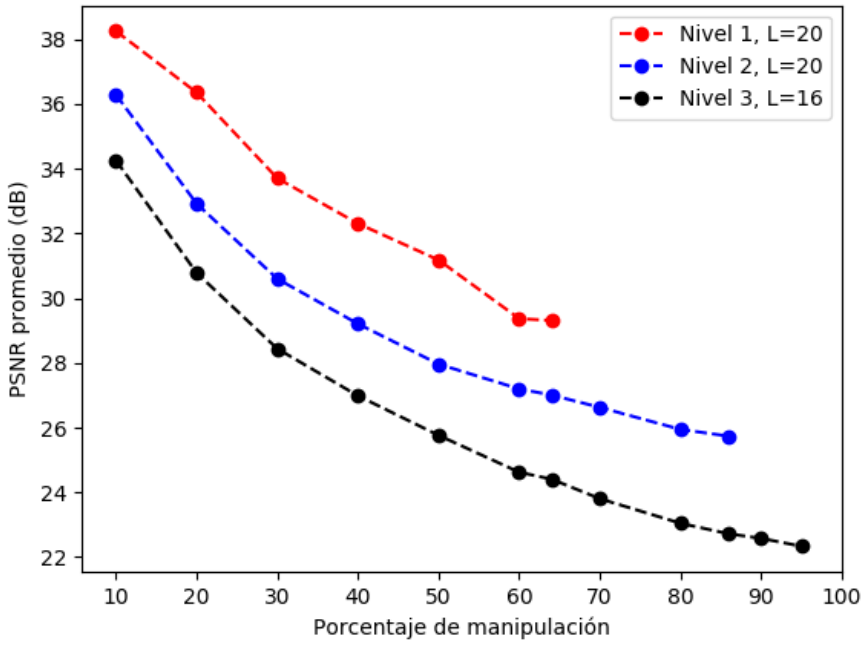


Figura 4.6: PSNR promedio de las imágenes restauradas.

En la Figura 4.6, se hace evidente el efecto del nivel utilizado en la marca en la calidad de las imágenes restauradas, en este caso, la línea correspondiente al PSNR promedio de las imágenes restauradas en el nivel 1 tiene valores por encima del nivel 2 y 3, y el nivel 2 tiene valores mayores que los obtenidos para el nivel 3. En la Tabla 4.7 se resumen los resultados obtenidos de manera experimental con las pruebas individuales y generales para el esquema de marcado propuesto.

Nivel	L	Tasa máxima de manipulación α	PSNR máximo	PSNR mínimo
1	20	0.64 %	38 dB	29 dB
2	20	0.86 %	36 dB	26 dB
3	16	0.95 %	34 dB	21 dB

Tabla 4.7: Resumen de los resultados obtenidos de forma experimental para el esquema *MarcaDWT*.

4.2. Límites del Esquema MarcaDWT

Para exhibir los límites del esquema aquí propuesto, se pretende explorar un porcentaje de bits manipulados por encima del máximo encontrado en las pruebas generales. Para ello, se consideró la L que dio mejores resultados en cada nivel en las pruebas individuales y generales. Se manipuló una imagen con tasas ligeramente mayores a los porcentajes máximos de restauración que se presentan en la Tabla 4.7. De las Tablas 4.3 y 4.4 se puede ver que los porcentajes de restauración decrecen significativamente al superar el máximo. Para estas pruebas, con el nivel 1 se consideró un $\alpha = 0.65$ y $L = 20$; para el nivel 2 se tomó $\alpha = 0.87$ y $L = 20$; en el nivel 3 los parámetros utilizados fueron $\alpha = 0.96$ y $L = 16$.

En la Figura 4.7(a), se puede ver la imagen manipulada en un 65 % y en 4.7(b) se observa la imagen restaurada, dado que la detección de bits manipulados se hace por bloques, la tasa real de manipulación fue de $\alpha = 0.65625$, el porcentaje de bits

restaurados fue 19.8238 %, sin embargo, con un sólo bit correspondiente a un pixel que no se restaure, pues no hay manera de saber el valor del pixel, sin embargo, los bits restaurados dan una idea del valor del pixel, por eso se puede observar una ligera similitud con la imagen original. El PSNR de la imagen obtenida es de 6.70533 dB.

(a) ***Imagen manipulada*** 65 %. **(b)** ***Imagen restaurada.***

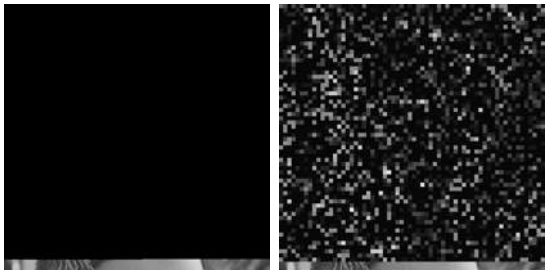
Figura 4.7: Límites del esquema para nivel 1, $\alpha = 0.65$ y $L = 20$.

(a) ***Imagen manipulada*** 87 %. **(b)** ***Imagen restaurada.***

Figura 4.8: Límites del esquema para nivel 2, $\alpha = 0.87$ y $L = 20$.

En la Figura 4.8(a), se puede ver la imagen manipulada en un 87 %, la tasa real de manipulación es de $\alpha = 0.875$. La Figura 4.8(b) presenta la imagen restaurada, el porcentaje de bits restaurados fue 17.0452 %. En esta imagen, aún es posible observar patrones similares a los de la imagen original. El PSNR de la imagen obtenida es de 5.56383 dB. En la Figura 4.9(a), se observa la imagen manipulada en un 96 %, el

porcentaje real de manipulación es de 96.875 %. La Figura 4.9(b) presenta la imagen restaurada, el porcentaje de bits restaurados fue 11.3861 %. En este caso, no es posible observar rasgo alguno de la imagen original. El PSNR de la imagen obtenida es de 5.17612 dB.



(a) ***Imagen manipulada*** 96 %**.** **(b)** ***Imagen restaurada.***

Figura 4.9: Límites del esquema para nivel 3, $\alpha = 0.96$ y $L = 16$.

Entonces, con este esquema de marcado, cuando se tienen tasas de manipulación por encima del máximo, el porcentaje de bits restaurados disminuye al grado de prácticamente no distinguir la imagen de la cual se trata, sin embargo, las tasas máximas están por encima de algunos esquemas que hay en la literatura, en seguida se hace una comparación de la marca aquí propuesta con las citadas en el estado del arte.

4.3. Comparación de *MarcaDWT* con el estado del arte

En la Tabla 4.8 es muestra la comparación entre los métodos citados en el estado del arte y el método planteado aquí MarcaDWT con cada uno de sus niveles. **MarcaDWT1** refiere a que se utilizó $nivel = 1$ como parámetro, de manera análoga para los demás niveles. Para la comparación se consideraron los siguientes aspectos: el

PSNR de las imágenes marcadas (columna 2), el PSNR de las imágenes restauradas (columna 3) y el porcentaje de manipulación máximo reporta cada trabajo que es capaz de restaurar el esquema, α_{max} (columna 4). Los PSNR que están entre corchetes denotan el intervalo entre en cual se encuentran los resultados obtenidos.

Esquema	PSNR imagen marcada (dB)	PSNR imagen restaurada (dB)	α_{max}
MarcaDWT3	37.9	21	95 %
Lee y Lin, 2008	38	> 20	90 %
MarcaDWT2	37.9	26	86 %
Zhang *et al.*, 2015	44	$[22, 44]$	80 %
MarcaDWT1	37.9	29	64 %
Romero, 2019	37.9	22	63 %
Zhang *et al.*, 2011a	37.9	$[24, 42]$	60 %
Zhang *et al.*, 2011c	37.9	$[22, 38]$	54 %
Calderon *et al.*, 2018	37.9	$[22, 39]$	48 %
Qin *et al.*, 2017	$[38, 46]$	$[29, 41]$	45 %
Tai y Liao, 2018	44	> 20	40 %
Zhang *et al.*, 2011b	37.9	$+\infty$	28 %
Qin *et al.*, 2012	51	> 40	27.3 %
Bravo-Solorio *et al.*, 2018	37.9	$+\infty$	26 %

Tabla 4.8: Comparación del desempeño del esquema propuesto con el estado del arte.

En lo que respecta al porcentaje de restauración máximo, el esquema planteado MarcaDWT con el nivel 1, supera apenas en un 1 % al esquema propuesto en Romero, 2019 y por 36 % y 38 % a los esquemas presentados en Zhang *et al.*, 2011b y Bravo-Solorio *et al.*, 2018, respectivamente, que es en los que está basado este trabajo. El nivel 2 del esquema MarcaDWT, se encuentra por encima de Zhang *et al.*, 2015 y de los recién mencionados. El nivel 3 del esquema MarcaDWT es el que muestra mayor porcentaje de restauración respecto al estado del arte citado en esta tesis y a los otros dos niveles.

En cuanto al PSNR de la imagen restaurada, el nivel 1 está entre 22 y 38 dB, el

nivel 2 presenta valores entre 26 y 36 dB y el nivel 3 tiene valores que varían entre 21 y 34 dB. Como dichos valores son mayores a 20 dB, se consideran dentro de los estándares de calidad aceptables. En general, los resultados obtenidos con el esquema MarcaDWT demostraron ser capaces de superar a los resultados del estado del arte, en cuanto a capacidad de restauración.

4.4. Conclusiones del capítulo

De acuerdo con los resultados experimentales, el esquema propuesto utilizando el $nivel = 1$ permite restaurar una imagen manipulada hasta en un 64 % del total de sus pixeles, esto se logra con un tamaño se segmento $L = 20$ y en un tiempo promedio de 21.09 segundos. La calidad de la imagen restaurada está entre 22 dB y 38 dB.

Para el $nivel = 2$ del esquema de marca de agua, los resultados demuestran que con $L = 20$, es posible recuperar el 100 % de los bits dañados, siempre y cuando el porcentaje de manipulación no sea mayor a 86 %. Para este nivel, el PSNR está entre 26 y 36 dB. En este nivel, el tiempo promedio de restauración fue de 6.35 segundos.

Con el $nivel = 3$, los experimentos señalan que con $L = 16$, es posible restaurar el 100 % de los bits dañados, para una imagen manipulada a lo más en un 95 %. La calidad de la imagen restaurada en este nivel es menor a la de los otros niveles, pero aún está dentro de los estándares de calidad aceptables para una imagen digital, el PSNR está entre 21 y 34 dB. El tiempo promedio para restaurar las imágenes fue de 0.39 segundos.

Lo anterior indica que lo desarrollado en la teoría coincide con lo obtenido de manera práctica. Los porcentajes de restauración y la calidad de las imágenes restauradas de cada nivel fueron los esperados. Por otra parte, los tiempos de marcado y

de restauración resultaron ser adecuados para una aplicación real.

Capítulo 5

Conclusiones

En este capítulo se presentan las conclusiones generales obtenidas de la interpretación de los resultados experimentales que se muestran en el Capítulo 4. Por otra parte, sugiere algunas ideas para trabajos futuros, basadas en este trabajo de tesis.

5.1. Conclusiones Generales

En esta tesis se propuso el esquema de marcado *MarcaDWT*, la cual consiste en una marca de agua frágil, que se inserta en los 3 LSB de la imagen. Para calcular el vector de datos hace uso de la sub-banda de baja frecuencia de la DWT con el wavelet de Haar, lo cual es equivalente a calcular los promedios de cada cuatro pixeles contiguos de la imagen, es decir, a submuestrear la imagen, por lo tanto, es un esquema de restauración aproximada. Este método maneja un parámetro llamado *nivel*, el cual toma valores en el conjunto $\{1, 2, 3\}$, cada nivel hace referencia al nivel de compresión de la DWT utilizado, entre mayor sea el nivel de compresión usado en el marcado, mayor es el porcentaje de restauración. El nivel de compresión también influye en la calidad de la imagen restaurada, pues entre más comprimida esté una imagen, menor es la calidad que se obtiene al descomprimirla. Para el proceso de recuperación se utilizó una formulación redundante en donde se generan dos conjuntos de códigos de

paridad $c_1 = A_1 d$ y $c_2 = A_2 d$ y se utilizan de manera iterada para restaurar los bits manipulados de la imagen.

Gracias a dichas características, fue posible plantear un esquema de marca de agua que para $nivel = 1$ es capaz de restaurar una imagen manipulada a lo más un 64 % de sus pixeles, para $nivel = 2$ logra restaurar una imagen manipulada hasta en un 86 % y con $nivel = 3$ es capaz de restaurar una imagen manipulada hasta en un 95 % de sus pixeles, manteniendo en todos los niveles una calidad aceptable. Más aún, el esquema propuesto mejora los resultados de los esquemas presentados en el estado del arte, en cuanto a capacidad de restauración, esto representa una contribución importante en el área de restauración de imágenes digitales con marcas de agua.

5.2. Trabajos Futuros

A continuación se listan algunas sugerencias acerca de qué se puede mejorar en el esquema propuesto, de acuerdo con lo que se observó a lo largo de su realización.

1. Dado que el proceso de marcado de la imagen se puede realizar en tiempos del orden de 0.20 segundos, se propone optimizar el esquema de marcado para que sea posible utilizarse en aplicaciones en donde el marcado debe hacerse en tiempo real, por ejemplo: los vídeos de vigilancia y vídeos tomados a pocos cuadros por segundo. Esto podría lograrse utilizando más redundancia.

2. En el método aquí propuesto se usa un mecanismo de detección de bloques manipulados basado en un hash criptográfico de 32 bits por cada bloque de 8×8. Sin embargo, en la literatura existen mecanismos igual de efectivos que emplean 4 bits por bloque [Tai y Liao, 2018]. Entonces, se propone cambiar el mecanismo de detección de pixeles manipulados, de tal manera que utilice una menor cantidad de bits, esto permitiría guardar más información de la imagen en la marca y, por lo tanto, aumentar la capacidad de restauración.

3. El esquema de marcado planteado utiliza como parámetro el *nivel* de compresión, sin embargo, este se tiene que elegir de forma previa al marcado, dependiendo si el usuario requiere más calidad o más porcentaje de restauración. Por este motivo, se propone un esquema que utilice los tres niveles de compresión para formar el vector de datos e implementar un sistema jerárquico de restauración, de tal manera que si no es posible restaurar un bloque con los bits del bloque comprimido de nivel 1, entonces se pueda restaurar con los bits del bloque comprimido de nivel 2 y, si aún no es posible restaurarlo, se utilicen los bits del nivel 3.

Referencias

[Bloom, 1999] Bloom, J. M. (1999). Revolution by the ream: A history of paper. *Saudi Aramco World*, 50:26–39.

[Bravo-Solorio *et al.*, 2018] Bravo-Solorio, S., Calderon, F., Li, C.-T., y Nandi, A. K. (2018). Fast fragile watermark embedding and iterative mechanism with high self-restoration performance. *Digital signal processing*, 73:83–92.

[Brownrigg, 1984] Brownrigg, D. R. (1984). The weighted median filter. *Communications of the ACM*, 27(8):807–818.

[Burrows, 1995] Burrows, J. H. (1995). Secure hash standard. Technical report, Department of Commerce Washington DC.

[Calderon *et al.*, 2018] Calderon, F., Espinosa, J., Flores, J., y Bravo, S. (2018). Watermarks based on dct for digital images restoration. *IEEE International Autumn Meeting on Power, Electronics and Computing*.

[Donoho, 2006] Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306.

[Dubolia *et al.*, 2011] Dubolia, R., Singh, R., Bhadoria, S. S., y Gupta, R. (2011). Digital image watermarking by using discrete wavelet transform and discrete cosine transform and comparison based on psnr. En *2011 International Conference on Communication Systems and Network Technologies*, pp. 593–596. IEEE.

[Eastlake y Jones, 2001] Eastlake, D. y Jones, P. (2001). Us secure hash algorithm 1 (sha1).

[Figueiredo *et al.*, 2007] Figueiredo, M., Nowak, R., y Wright, S. (2007). Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE Selected Topics in Signal Processing*, 1(4).

[Hamming, 1950] Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160.

[Joo *et al.*, 2002] Joo, S., Suh, Y., Shin, J., y Kikuchi, H. (2002). A new robust watermark embedding into wavelet dc components. *ETRI journal*, 24(5):401–404.

[Lee y Lin, 2008] Lee, T.-Y. y Lin, S. D. (2008). Dual watermark for image tamper detection and recovery. *Pattern recognition*, 41(11):3497–3506.

[Nilchi y Taheri, 2008] Nilchi, A. R. N. y Taheri, A. (2008). A new robust digital image watermarking technique based on the discrete cosine transform and neural network. En *2008 International Symposium on Biometrics and Security Technologies*, pp. 1–7. IEEE.

[Poljicak *et al.*, 2011] Poljicak, A., Mandic, L., y Agic, D. (2011). Discrete fourier transform-based watermarking method with an optimal implementation radius. *Journal of Electronic Imaging*, 20(3):033008.

[Qin *et al.*, 2012] Qin, C., Chang, C.-C., y Chen, P.-Y. (2012). Self-embedding fragile watermarking with restoration capability based on adaptive bit allocation mechanism. *Signal Processing*, 92(4):1137–1150.

[Qin *et al.*, 2017] Qin, C., Ji, P., Zhang, X., Dong, J., y Wang, J. (2017). Fragile image watermarking with pixel-wise recovery based on overlapping embedding strategy. *Signal Processing*, 138:280–293.

[Romero, 2019] Romero, J. E. (2019). Marcas de Agua basadas en el estándar JPEG (Tesis de maestría). *Universidad Michoacana de San Nicolás de Hidalgo*.

[Solachidis y Pitas, 1999] Solachidis, V. y Pitas, I. (1999). Circularly symmetric watermark embedding in 2-d dft domain. En *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volumen 6, pp. 3469–3472. IEEE.

[Tai y Liao, 2018] Tai, W.-L. y Liao, Z.-J. (2018). Image self-recovery with watermark self-embedding. *Signal Processing: Image Communication*, 65:11–25.

[Thomos *et al.*, 2005] Thomos, N., Boulgouris, N. V., y Strintzis, M. G. (2005). Optimized transmission of jpeg2000 streams over wireless channels. *IEEE Transactions on image processing*, 15(1):54–67.

[Tirkel *et al.*, 1993] Tirkel, A. Z., Rankin, G., Van Schyndel, R., Ho, W., Mee, N., y Osborne, C. F. (1993). Electronic watermark. *Digital Image Computing, Technology and Applications (DICTA'93)*, pp. 666–673.

[VanderPlas, 2016] VanderPlas, J. (2016). *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media.

[Yang y Bourbakis, 2005] Yang, M. y Bourbakis, N. (2005). An overview of lossless digital image compression techniques. En *48th Midwest Symposium on Circuits and Systems, 2005.*, pp. 1099–1102. IEEE.

[Yusof y Khalifa, 2007] Yusof, Y. y Khalifa, O. O. (2007). Digital watermarking for digital images using wavelet transform. En *2007 IEEE International Conference on Telecommunications and Malaysia International Conference on Communications*, pp. 665–669. IEEE.

[Zhang *et al.*, 2013] Zhang, J., Zhang, Q., y Lv, H. (2013). A novel image tamper

localization and recovery algorithm based on watermarking technology. *Optik*, 124(23):6367–6371.

[Zhang *et al.*, 2011a] Zhang, X., Qian, Z., Ren, Y., y Feng, G. (2011a). Watermarking with flexible self-recovery quality based on compressive sensing and compositive reconstruction. *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY*, 6(4).

[Zhang *et al.*, 2011b] Zhang, X., Qian, Z., y Wang, S. (2011b). Reference sharing mechanism for watermark self-embedding. *IEEE Transactions on Image Processing*, 20(2).

[Zhang y Wang, 2008] Zhang, X. y Wang, S. (2008). Fragile watermarking with error-free restoration capability. *IEEE Transactions on Multimedia*, 10(8):1490–1499.

[Zhang *et al.*, 2011c] Zhang, X., Wang, S., Qian, Z., y Feng, G. (2011c). Self-embedding watermark with flexible restoration quality. *Multimedia Tools and Applications*, 54(2):385–395.

[Zhang *et al.*, 2015] Zhang, X., Xiao, Y., y Zhao, Z. (2015). Self-embedding fragile watermarking based on dct and fast fractal coding. *Multimedia Tools and Applications*, 74(15):5767–5786.