MARCAS DE AGUA BASADAS EN EL ESTÁNDAR JPEG PARA LA RESTAURACIÓN DE IMÁGENES

TESIS

Que para obtener el grado de
Maestro en ciencias en ingeniería eléctrica

PRESENTA
Josué Espinosa Romero

ASESOR
Dr. Félix Calderón Solorio

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

DIVISIÓN DE ESTUDIOS DE POSGRADO DE LA FACULTAD DE INGENIERÍA ELÉCTRICA

Morelia, Michoacán agosto del año 2019

MARCAS DE AGUAS BASADAS EN EL ESTÁNDAR JPEG PARA LA RESTAURACIÓN DE IMÁGENES

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

presenta

Josué Espinosa Romero

Dr. Félix Calderón Solorio

Director de Tesis

Universidad Michoacana de San Nicolás de Hidalgo

Julio 2019

Lista de Publicaciones

"Parallel Mining of Frequent Patterns for School Records Analytics at the Universidad Michoacana"
Juan J. Flores, Luis Garcia-Nava, Monserrat A. Castro-Coria, Victor M. Telles, Emanuel Huerta B., Josue Espinosa-Romero y Felix Calderon
Publicado en ROPEC 2017

"Watermarks based on DCT for Digital Images Restoration"
Felix Calderón, Josue Espinosa-Romero, Juan Flores y Sergio Bravo-Solorio
Publicado en ROPEC 2018

Resumen

En esta tesis se proponen los algoritmos TDC1 y TDC2 para incrustar una marca de agua en una imagen digital, que permite verificar el contenido de la imagen y en caso de que éste haya sido manipulado, permite restaurar el contenido original. Verificar el contenido en una imagen es importante, sobre todo si dicha imagen tiene implicaciones legales. Por ejemplo, considere el caso hipotético en el que una imagen digital es presentada como evidencia en un juicio y que además dicha imagen, es determinante para que una persona sea declarada como inocente o culpable. En este caso, es muy importante confirmar que dicha imagen no haya sido alterada de ninguna manera. Para ello, algunos esquemas de marcas de agua permiten autenticar los píxeles en la imagen y que incluso van más allá, permitiendo restaurar los píxeles originales de la imagen.

Los algoritmos TDC1 y TDC2 propuestos, hacen uso de un algoritmo de compresión basado en el estándar JPEG, para comprimir la imagen a marcar. Luego generan un conjunto de códigos de paridad, a partir del resultado de la compresión. En el caso del algoritmo TDC1, los códigos de paridad que se generaron, se almacenan junto con el resultado del algoritmo de compresión y un conjunto de hash criptográficos, como marca de agua en la imagen digital. Para el caso del algoritmo TDC2, sólo los códigos de paridad y el conjunto de hash criptográficos, se almacenan como marca de agua, dejando de lado la información del resultado del algoritmo de compresión. La imagen con la marca de agua incrustada, se almacena en un formato sin perdida de información, como por ejemplo: PNG, BMP y TIF, entre otros.

Para una imagen con marca de agua creada por el algoritmo TDC1, se logra restaurar el contenido manipulado en la imagen, siempre y cuando ésta presente un porcentaje de manipulación de hasta un 45% del total de píxeles de la imagen. Para una imagen con marca de agua creada con el algoritmo TDC2, se logra restaurar el contenido manipulado en la imagen, si ésta presenta un porcentaje de manipulación de hasta un 63% del total de píxeles de la imagen.

Palabras clave: Calidad de imágenes, sistemas de ecuaciones lineales, manipulación de imágenes, transformada discreta coseno, códigos de paridad

Abstract

In this thesis the TDC1 and TDC2 algorithms are proposed; those algorithms embed a watermark in a digital image. That watermark allows verifying if the content of the image has been manipulated; if so, it allows the original content to be restored. Verifying the content in an image is important, especially if that image has legal implications. For example, consider the hypothetical case in which a digital image is presented as evidence in a trial and that image is also decisive for a person to be declared innocent or guilty. In this case, it is very important to confirm that the image has not been altered in any way. To do this, some watermark schemes allow to authenticate the image pixels and restore its original pixels' content.

Algorithms TDC1 and TDC2 make use of a compression algorithm based on the JPEG standard, to compress the image to be marked. They then generate a set of parity codes, from the result of the compression. The TDC1 algorithm stores the parity codes that were generated together with the result of the compression algorithm and a set of cryptographic hash values, as a watermark. The TDC2 algorithm stores only the parity codes and the cryptographic hash set are as a watermark, leaving aside the information of the result of the compression algorithm. The image with the embedded watermark is stored in a lossless format, such as: PNG, BMP and TIF, among others.

It is possible to restore the manipulated content of a watermarked image created by the TDC1 as long as it presents a manipulation percentage of up to 45% of the total image pixels. The TDC2 algorithm allows a restoration of the content of an image where up to 63% of the image pixels have been manipulated.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Algoritmos

Lista de Símbolos

I	Imagen
I_m	Imagen con marca de agua
I_M	Imagen empleada como marca de agua
I_t	Imagen manipulada, con los píxeles identificados mediante el valor -1
N	Número de filas en la imagen
M	Número de columnas en la imagen
$\leftarrow$	Asignación
&	Operación lógica and, realizada bit a bit
$0X$	Indica que se trata de un valor numérico en hexadecimal
$\times$	Operación de multiplicación
$\sum$	Operador de sumatoria
$\log_{10}$	Logaritmo base 10
R	Valor de fluctuación máxima
$\approx$	Valor aproximado
β	Tasa de decaemiento en el rango $0 < \beta < 1$
α	Tasa de manipulación
μ	Media
σ	Desviación estándar
X	Variable aleatoria
d	Vector de datos
A	Matriz de codificación
c	Vector de códigos de paridad
h	Vector de hash criptográficos
L	Tamaño de segmento
N_d	Número de elementos en el vector d
N_c	Número de elementos en el vector c
$N_{c,d}$	Número de elementos en el vector d_c
d_c	Vector de datos comprimidos
N_B	Número de bloques de 8×8 en la imagen
B	Bloque de 8×8 píxeles
k	Factor de compresión
$\in$	Elemento de

Q	Matriz de cuantización del estándar JPEG
q	Valor cuantizado de un coeficiente de la DCT
C	Conjunto de coordenadas
$[a\|b]$	Concatenación de vectores
$a : b$	Rango a a b
key	Clave secreta
$ToDec(.)$	Convierte un número binario a decimal
$ToBin(.)$	Convierte un número decimal a binario
$Round(.)$	Redondea al entero más cercano
$Matriz(.)$	Reserva memoria para una matriz de dimensiones dadas
$Vector(.)$	Reserva memoria para un vector de tamaño dado
$Shuffle(.)$	Permuta de manera pseudoaleatoria los elementos del vector dado
$Shuffle^{-1}(.)$	Deshace la permutación pseudoaleatoria del vector dado
$Hash32(.)$	Función criptográfica de 32 bits
MSB	More Significant Bits
LSB	Least Significant Bits
DCT	Discrete Cosine Transform
$IDCT$	Inverse Discrete Cosine Transform
PSNR	Peak Signal to Noise Ratio
MSE	Mean Square Error
JPEG	Join Photographic Expert Group
dB	Decibel o Decibelio

Capítulo 1

Introducción

En la actualidad existen herramientas que permiten editar medios gráficos digitales, tales como las imágenes. Entre estas destacan Adobe Photoshop, The Gimp, Pixelmator y Paintshop Pro, entre otras. Además, en Internet se encuentran muchos vídeos tutoriales y foros, que permiten convertirse en un experto en manejo de dichas herramientas.

En Internet se encuentran muchas imágenes de la naturaleza, las cuales son impresionantes. Imagine la idea de estar en medio de un paisaje como el que se muestra en la Figura 1.1(a). Disfrutando del colorido de los árboles, oyendo un leve susurro provocado por el correr del viento, y a lo lejos, el sonido del agua corriendo. Por otra parte, también en la Web podemos encontrar imágenes un tanto extrañas, algunas son reales y otras que parecen ser reales pero que no lo son. Un ejemplo de estas imágenes extrañas, se muestra en la Figura 1.1(b) donde se presenta a una mujer que tiene piernas de rana, la cual está esperando a ser atendida en el mostrador. Sin embargo, aunque la imagen mostrada en Figura 1.1(b) parece ser real, más de alguno la miraría dos veces, dudando de su veracidad.

Lamentablemente, no todo lo que circula en Internet es real y el paisaje de ensueño es uno de ellos. En la Figura 1.2(a) se muestra el paisaje real, que aunque sigue siendo muy hermoso, la idea de pensar que los árboles no tienen ese color morado, provoca un cierto desinterés en ir a conocer dicho lugar. Por otra parte, los memes han invadido Internet y hoy en día cualquiera puede ser víctima de ellos. La mujer mostrada en la Figura 1.1(b) no fue la excepción. Ella se descuidó un instante tal y como se muestra en la Figura 1.2(b),

(a) ***Paisaje de ensueño***

(b) ***Mujer rana***

Figura 1.1: Imágenes que circulan en Internet

lamentablemente para ella, alguien no desaprovechó la oportunidad y le tomó una foto. El resto fue trabajo de los expertos en el manejo de herramientas de edición de imágenes digitales, los cuales la convirtieron en algo divertido.

(a) ***Paisaje real***

(b) ***Fotografia original***

Figura 1.2: Imágenes reales de las Figura 1.1

Alterar y/o modificar imágenes tiene un carácter divertido y/o entretenido en el que se puede dejar volar la imaginación y sirven para pasar un rato de entretenimiento. Sin embargo, no todo es "miel sobre hojuelas", ya que en un juicio legal una imagen se puede presentar como evidencia acusatoria. En esta situación, se debe tener certeza de que lo que

muestra la imagen es una evidencia confiable y real. En la actualidad, algunos esquemas de marcas de agua se utilizan para validar el contenido de una imagen y en caso de que éste haya sido alterado, permiten recuperar el contenido original antes de su manipulación [Zhang08]. Para ello, se suele guardar cierta información en el propio contenido de la imagen de tal manera que resulte imperceptible al ojo humano [Zhang11c].

1.1. Planteamiento del Problema

Consideremos el siguiente escenario hipotético, en donde se presume el robo de un equipo de cómputo en una oficina y que además los hechos ocurrieron de la siguiente manera. El encargado de la seguridad llega al trabajo y nota que falta un equipo de cómputo, así que supone que lo han tomado prestado. De cualquier modo, decide asegurarse y pregunta al personal, para saber quién tiene el equipo faltante. Después de preguntarles resulta que nadie lo tomó y no saben nada al respecto. Sin embargo, el encargado sabe que el ilícito ocurrió el 6 de junio del 2017, así que recure a las cámaras de seguridad para averiguar qué pasó con ese equipo. Al recorrer las imágenes, nota algo sospechoso en una de ellas (ver Figura 1.3), la cual presenta distintas tonalidades en la base de las escaleras y en la pared, como si tuviera una imagen sobrepuesta de esa misma cámara, pero tomada en otro momento del día.

Figura 1.3: Imagen sospechosa

El encargado de seguridad sabe que las imágenes tomadas por esa cámara, agregan

una marca de agua al momento de guardarlas, la cual es imperceptible al ojo humano y que permite detectar y recuperar la imagen original, en el caso de que la imagen marcada haya sido alterada. Así que procede a verificar la integridad de la imagen de la Figura 1.3, haciendo uso de una clave y de la computadora. Después de hacerle el análisis de integridad, se topa con el resultado mostrado en la Figura 1.4(a), en donde de color negro se muestra toda el área manipulada de la imagen en cuestión. El resultado del análisis, no sólo permite determinar si la imagen ha sido alterada, sino que también permite detectar el área exacta de la manipulación, el cual corresponde a un 25% del total de píxeles, para esta imagen.

(a) ***Análisis de integridad*** **(b)** ***Imagen restaurada***

Figura 1.4: Validación y restauración a partir de la marca de agua

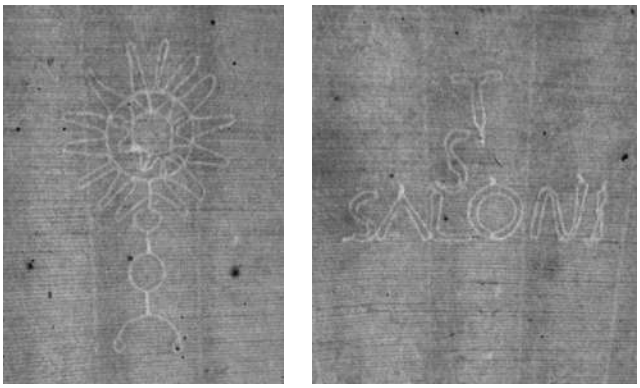
Una vez determinado que efectivamente la imagen está manipulada, se procede a restaurar el contenido original de la imagen, que fue manipulado. La imagen restaurada, se muestra en la Figura 1.4(b), donde se aprecia que alguien ocultó el hecho de que llevaba una caja, donde posiblemente vaya oculto el equipo de cómputo. Además, se observa claramente al culpable y por lo tanto se puede tomar acciones pertinentes para lidiar con el problema. La marca de agua que se utilizó en esta imagen, permite recuperar el contenido manipulado en la imagen, siempre y cuando éste no exceda el 26% del total de píxeles, ya que si se sobre pasa dicho porcentaje, la marca de agua no podrá restaurar el contenido manipulado. Para solventar este problema, se hará una propuesta de marca de agua que permita recuperar el contenido original, cuando la imagen presenta hasta un 66% del total de píxeles manipulados.

1.2. Antecedentes

En esta sección se habla brevemente del uso que se les ha dado a las marcas de agua en el papel. Luego se trata de manera sencilla y clara como se representa una imagen por computadora y por último, se abordan brevemente las marcas de agua en medios digitales.

1.2.1. Marcas de agua en el papel

En el siglo XIII los italianos introdujeron la filigrana o marca de agua, como signo de identificación de un fabricante de papel, de una zona o de una ciudad determinada [Martínez Leal15]. En la Figura 1.5(a) se muestra un documento con un cáliz como marca de agua, en la Figura 1.5(b) se muestra el texto "TS SALONI" como marca de agua, ambas sólo se pueden apreciar a trasluz.



(a) ***Caliz*** **(b)** ***Texto***

Figura 1.5: Marcas de agua en documentos antiguos

En la actualidad, los historiadores han podido obtener datos importantes tales como la fecha y la localización geográfica de documentos antiguos, gracias al estudio de las marcas de agua. Esto debido a que han podido establecer datos relevantes tales como fechas de elaboración, el molino en el que se produjo, su procedencia e incluso han podido trazar las rutas comerciales que tuvieron dichos materiales [Martínez Leal15].

1.2.2. Imágenes

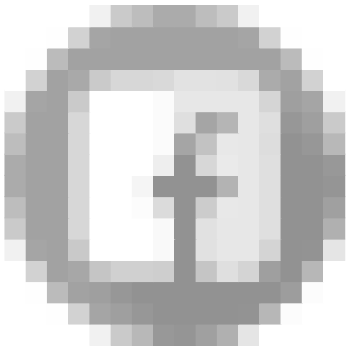
Las imágenes por computadora se representan como un arreglo bidimensional de píxeles de dimensiones $N \times M$, donde N indica el número de filas y M el número de columnas. Además, cada píxel puede tener uno o más valores, dependiendo del modelo del color empleado. Algunos de estos modelos son: escala de grises, RGB, CMYK, HSL y HSV, entre otros y definen el espacio de color que tendrá la imagen. Por ejemplo, para el modelo RGB el cual emplea el espacio de color Rojo, Verde y Azul (en inglés Red, Green, and Blue), se tendrá una matriz de dimensiones $N \times M \times 3$, debido a que se tienen 3 valores por píxel uno por cada color base del modelo. En la Figura 1.6(a) se muestra una imagen a color en el modelo RGB, para la misma imagen en la Figura 1.6(b) se muestra el canal rojo, en la Figura 1.6(c) se muestra el canal verde y en la Figura 1.6(d) se muestra el canal azul.

(a) ***Imagen a color*** **(b)** ***Canal rojo (R)*** **(c)** ***Canal verde (G)*** **(d)** ***Canal azul (B)***

Figura 1.6: Imagen a color y sus respectivos canales en el modelo RGB

Las computadoras solo manejan números binarios, por lo tanto, la cantidad de cada color, se debe representar mediante un valor numérico binario. En los modelos RGB y escala de grises, esta cantidad se representa mediante 8 bits, el cual en notación decimal se encuentra en el rango $[0, 255]$. La Figura 1.7(a) muestra el logo de Facebook de dimensiones 16×16 empleando el modelo de color escala de grises. En la Figura 1.7(b) se muestra una matriz cuyos elementos son los valores numéricos de la imagen mostrada en la Figura 1.7(a).

A lo largo de este documento, se hará referencia a los 5 bits más significativos (MSB por sus siglas en inglés) y los 3 bits menos significativos (LSB por sus siglas en inglés) de una imagen o un píxel, los cuales corresponden a los bits de mayor orden y de



255	255	255	255	230	173	139	107	107	139	173	230	255	255	255	255
255	255	254	183	97	91	91	91	91	91	91	97	183	254	255	255
255	254	138	91	91	91	91	91	91	91	91	91	91	138	254	255
255	183	91	101	173	189	189	190	201	201	200	190	107	91	183	255
230	97	91	173	255	255	255	252	209	209	209	209	186	88	97	230
173	91	91	189	255	255	255	252	197	97	129	209	197	71	88	173
139	91	91	189	255	255	255	252	97	157	210	209	196	67	71	132
107	91	91	189	255	255	255	222	67	189	212	209	197	67	67	73
107	91	91	189	255	255	238	67	67	67	127	209	197	67	67	70
139	91	91	189	255	255	253	194	67	174	210	209	197	67	67	89
173	91	91	189	255	255	255	250	67	200	209	209	197	67	67	119
230	97	91	172	255	255	255	250	67	200	209	209	185	67	68	206
255	183	91	100	163	179	179	176	67	163	197	185	88	67	131	255
255	254	138	91	88	71	67	67	67	67	67	67	67	87	253	255
255	255	254	183	97	88	71	67	67	67	67	68	131	253	255	255
255	255	255	255	230	173	132	73	70	89	119	206	255	255	255	255

(a) ***En escala de grises*** **(b)** ***Valores numéricos***

Figura 1.7: Logo de Facebook

menor orden, respectivamente, del valor de 8 bits por píxel. Además, se considerará que las imágenes están en escala de grises.

1.2.3. Marcas de agua digitales

Las marcas de agua en medios digitales se propusieron como solución para problemas de derechos de copia y propiedad de archivos de datos multimedia, posibilitando la identificación de la fuente, autor, propietario, distribuidor o consumidor autorizado, de medios digitales [Orúe12]. Sin embargo, éstas han ido evolucionando con el tiempo, llegando a tal grado de permitir recuperar información perdida o manipulada de manera maliciosa [M. Vargas16]. Además, algunas otras aplicaciones de las marcas de agua son: monitorizado de transmisiones de radiodifusión o comunicaciones secretas entre otras [Orúe12].

Las marcas de agua en imágenes digitales se clasifican en robusta, frágil o semi-frágil. Se considera robusta si perdura después de operaciones tales como la compresión, cambios de color y/o transformaciones geométricas entre otros. Por otra parte, las marcas frágiles son aquellas que quedan eliminadas o modificadas y dejan de cumplir su función en caso de ataque [M. Vargas16]. El objetivo de estas últimas, es determinar si la imagen marcada ha sido alterada o no, ya que en caso de cualquier manipulación por mínima que sea, la marca queda parcial o completamente eliminada. Una marca semi-frágil, es una marca que conserva características tanto de las robustas como de las frágiles [Liu02].

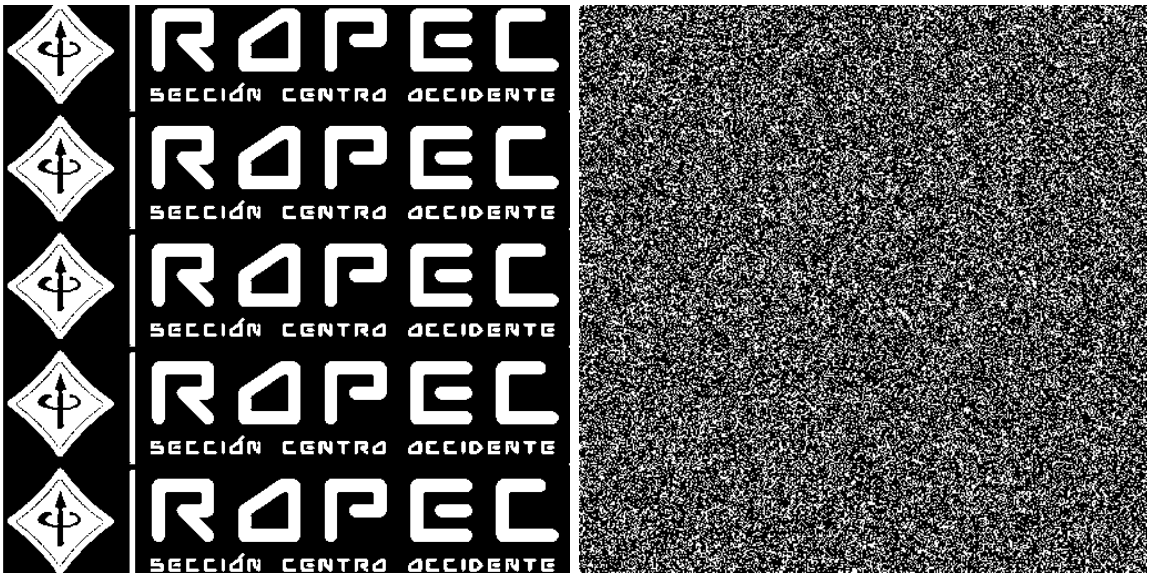
Otra cuestión importante en las marcas de agua, es su visibilidad. Se considera visible si es apreciable a simple vista, dicho de otro modo, ésta resalta a la vista. Por otra parte, se considera invisible si es imposible de apreciar por el ojo humano, pero fácil de detectar usando un algoritmo dado y una clave secreta, en una computadora [M. Vargas16]. Dependiendo de la aplicación se decidirá entre visible e invisible, en el caso de aplicaciones donde se requiere validar la integridad del contenido de una obra, es preferible que sea invisible, debido a que, si alguien quiere manipularlo de manera maliciosa, éste no sea capaz de identificar los mecanismos de seguridad incrustados en ella. Existen diversos métodos que generan marcas invisibles en imágenes, sin embargo, se pueden clasificar en dos principales; marcado en el dominio transformado y marcado en el dominio espacial. El primero, consiste en modificar ligeramente los coeficientes resultantes al aplicar alguna transformación. Algunos ejemplos de éstas son, la transformada discreta coseno (DCT por sus siglas en inglés) [Yadav14] y la transformada discreta Wavelet [Lu01]. Por otra parte, el marcado en el dominio espacial, consiste en guardar la información referente a la marca de agua, directamente en los píxeles de la imagen. Múltiples trabajos que emplean este método han demostrado que se puede conseguir una marca invisible, incrustando información de la marca de agua en los 3 LSB de cada píxel de la imagen, siempre y cuando ésta presente un patrón aleatorio.

Una manera muy sencilla de detectar si el contenido de una imagen ha sido alterado, se consigue guardando en el dominio espacial una imagen I_M, que servirá como marca de agua. La detección se realiza comparando el contenido almacenado en los 3 LSB de la imagen marcada I_m, con la imagen original I_M. En el Algoritmo 1 se describe este mecanismo de marcado en el dominio espacial, la cual almacena la marca de agua en los 3 LSB de la imagen, considerando que la imagen a marcar I tiene las mismas dimensiones que la imagen I_M y que el píxel $I_M[i, j]$ está en un rango entre 0 y 7.

La Figura 1.8 muestra dos imágenes que se emplearán como marca de agua. En la Figura 1.8(a), se muestra una imagen con el logo de la ROPEC, el cual se repitió para cubrir toda la imagen. La imagen de la Figura 1.8(b) se creó permutando de manera aleatoria los píxeles de la imagen mostrada en la Figura 1.8(a), empleando una clave como semilla para generar números aleatorios. Con esto el contenido de la imagen que será empleado como marca de agua, presenta un patrón aleatorio, lo cual resultará en una marca invisible.

Algoritmo 1 $MarcaLSB(I, I_M)$

1: $N \leftarrow Filas(I), M \leftarrow Columnas(I)$
2: $I_m \leftarrow Matriz(N, M)$
3: **para** $n = 0$ **hasta** N **hacer**
4: **para** $m = 0$ **hasta** M **hacer**
5: $I_m[n, m] \leftarrow (I[n, m]$ & 0XF8) $+ (I_M[n, m]$ & 0X7)
6: **fin para**
7: **fin para**
8: **devolver** I_m



(a) ***Logo ROPEC*** **(b)** ***Logo aleatorio***

Figura 1.8: Imágenes empleadas como marcas de agua

En la Figura 1.9, se muestran dos imágenes de Lena marcadas empleando el Algoritmo 1. En la Figura 1.9(a) se muestra la imagen marcada empleando la imagen de la Figura 1.8(a) como marca de agua, en ésta se puede apreciar casi por completo la palabra "ROPEC" en la parte superior de la imagen (marcado con una línea blanca) y las letras "OP" en la espalda y parte del brazo derecho (enmarcadas en un cuadro blanco). La imagen de la Figura 1.9(b) emplea la imagen de la Figura 1.8(b) como marca de agua, en esta se observa que el contenido que era perceptible a simple vista del logo de la ROPEC, ahora es imperceptible, de este modo se evita que se note que la imagen tiene una marca de agua.

(a) ***Marca de agua visible*** **(b)** ***Marca de agua invisible***

Figura 1.9: Imágenes de Lena marcadas en los 3 LSB

1.2.4. Nivel de calidad en imágenes

Para medir la calidad en una imagen de manera cuantitativa, se suele emplear la relación señal a ruido pico (PSNR por sus siglas en inglés). El cálculo de esta medida se hace en decibeles y se muestra en la Ecuación (1.1), donde R es la máxima fluctuación que se puede presentar en los valores de los píxeles de la imagen y además el error cuadrático medio (MSE por sus siglas en inglés) se calcula tal y como se indica en la Ecuación (1.2).

$$\text{PSNR} = 10 \log_{10} \left(\frac{R^2}{\text{MSE}} \right) \tag{1.1}$$

$$MSE = \frac{\sum_{N,M}(I_1[n,m] - I_2[n,m])^2}{N \times M} \tag{1.2}$$

Para imágenes en escala de grises R es igual a 255, ya que los valores de los píxeles están en el rango entre 0 y 255. Considere dos imágenes I_1 e I_2 de dimensiones $N \times M$ y considere que todos los píxeles en I_1 valen 255 y que todos los píxeles de I_2 valen 0. El MSE de I_1 e I_2 es MSE $= N \times M \times 255^2/(N \times M) = 255^2$, con lo que el PSNR de I_1 e I_2 es PSNR $= 10 \log_{10}(255^2/255^2) = 0$ dB. Ahora considere que todos los píxeles de I_1 y de I_2, tienen el mismo valor o sea I_1 e I_2 son idénticas. Por lo tanto, se tiene que el MSE $= \sum_{N,M}(0)^2/(N \times M) = 0$ y por consiguiente: PSNR $= 10(\log_{10}(255^2) - \log_{10}(0)) = \infty$ dB. De lo anterior se puede observar, que entre más similares sean dos imágenes, entonces el valor del PSNR será mayor. En la práctica, valores por encima de 25 dB indican una calidad aceptable [Li07]. Por último, en una imagen con marca de agua en los 3 LSB del domino espacial, se tendrá un PSNR $\approx$ 37.9 dB, si la información que se emplea como marca de agua, presenta una distribución uniforme [Zhang11a].

1.3. Objetivos de la Tesis

1.3.1. Objetivo general

Proponer un esquema de marca de agua digital para imágenes en escala de grises, basado en los esquemas de marcado y restauración, que permita restaurar la información contenida en una imagen mediante una versión comprimida de la misma, cuando el porcentaje de píxeles dañados sea a lo más un 66% y donde la imagen marcada, se almacenará en formato PNG, que es un formato de compresión sin pérdida de información.

1.3.2. Objetivos particulares

- Utilizar un hash criptográfico por cada bloque de 8×8, como mecanismo para detectar manipulaciones en un imagen digital.
- Proponer un esquema de compresión basado en el estándar JPEG, que permita comprimir una imagen lo suficiente, sin que ésta presente grandes degradaciones, para ser usada en los procesos de marcado, recuperación y restauración de una imagen digital.
- Adaptar los esquemas de marcado y restauración exacta basados en sistemas de ecuaciones lineales, para que permita recuperar el contenido de la imagen manipulada a través de la imagen comprimida de la misma, cuando el porcentaje de píxeles manipulados sea a lo más en un 66%.
- Plantear el esquema de restauración de la imagen dañada, basado en la imagen original y la versión comprimida de la misma.

1.4. Descripción de Capítulos

En el Capítulo 2 se exponen los fundamentos para este trabajo, los cuales están basados en sistemas de ecuaciones y además, se describen de manera resumida y concreta algunos trabajos que emplean marcas de agua frágiles, en la literatura. En el Capítulo 3 se describe la implementación realizada de compresión basada en el estándar JPEG, se plantea el uso de la imagen comprimida en lugar de la imagen real y cómo esto ayuda a cumplir

con el objetivo general de ampliar el porcentaje máximo permisible de píxeles dañados, basado en un esquema de recuperación que emplea sistemas de ecuaciones, describiendo los algoritmos implementados para lograrlo. En el Capítulo 4 se muestran y analizan los resultados obtenidos y en el Capítulo 5 se expresan las conclusiones a las que se llegó tras analizar los resultados y se plantean posibles estrategias para el trabajo futuro.

Capítulo 2

Marco teórico

Considere una imagen I de dimensiones $N \times M$ que ha sido manipulada y la cual tiene una marca de agua frágil invisible almacenada en los 3 LSB. Para restaurar la información de los píxeles manipulados a partir de la información de los píxeles no manipulados de la imagen, primeramente, se requerirá de un proceso que determine cuales píxeles han sido manipulados y cuáles no. En la literatura existen diversos mecanismos que permiten llevar a cabo la tarea de detección de manipulaciones, algunos emplean 1 bit por píxel [Fridrich99], en otros se emplean 2 bits por cada 4 píxeles [Lee08] y en algunos otros, 32 bits por cada 64 píxeles [Zhang11b].

Con referencia a la restauración, en la literatura también se encuentran diversos procedimientos que realizan esta labor. A aquellos que se avocan a recuperar de manera directa la información de los 5 MSB, a partir de información almacenada en los 3 LSB, les denominaremos restauración exacta [Zhang08, Zhang11b, Bravo-Solorio18, Qin16], mientras que otros que utilizan compresión para representar los 5 MSB les denominaremos restauración aproximada [Lee08, He09, Qin12, Dadkhah14, Zhang15, Qin17]. En cualquiera de los dos casos, se tiene el problema de que se está tratando de recuperar el contenido de los 5 MSB a partir de los 3 LSB, con lo que indudablemente habrá perdida de información y por consiguiente, el proceso de restauración de la imagen estará limitado.

Este capítulo, describe el proceso de extracción y detección de manipulaciones basada en un hash criptográfico de 32 bits, por bloques de 8×8 (64 píxeles). Posteriormente, se

describen algunas codificaciones que permiten la detección y recuperación de la información que se transmite a través de un canal, incluyendo un análisis de alcances y limitaciones y cómo se puede aplicar a marcas de agua. También, se describen de manera resumida algunos trabajos en la literatura que emplean restauración exacta y/o aproximada. Por último, se plantea una propuesta de marca de agua aproximada que permita restaurar los píxeles manipulados, cuando la imagen está manipulada hasta en un 66% del total de píxeles.

2.1. Extracción de información y detección de manipulaciones basada en hash criptográfico

Considere la imagen I de dimensiones $N \times M$, la cual se divide en bloques sin traslape de 8×8. Por cada bloque, 320 bits corresponden a los 5 MSB y 192 bits restantes a los 3 LSB. Los 192 bits serán la marca de agua, de los cuales 160 corresponden a un segmento de un vector de información que permitirá recuperar la información manipulada correspondiente a los 5 MSB y los 32 bits restantes para un hash criptográfico [Zhang11b]. En la Figura 2.1 se muestra de manera gráfica como una imagen de dimensiones $N \times M$ (N y M múltiplos de 8) se divide en bloques de 8×8 y como a su vez de cada bloque se extraen 320 bits correspondientes a los 5 MSB d_i, 160 bits para información c_i y 32 bits para el hash criptográfico h_i.

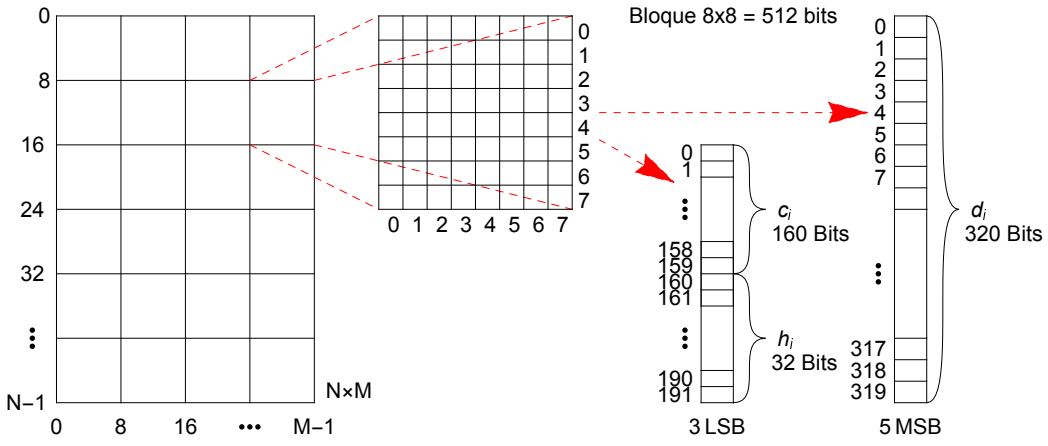


Figura 2.1: Extracción de la marca de agua en una imagen

Si para todos los bloques de 8×8 en la imagen, los 320 bits se concatenan a un único vector $d = [d_1, d_2, d_3, \cdots, d_{N_B}]$, los 160 bits de información a un único vector $c = [c_1, c_2, c_3, \cdots, c_{N_B}]$ y los 32 bits del hash, a un único vector $h = [h_1, h_2, h_3, \cdots, h_{N_B}]$, entonces se tienen 3 vectores formados por exactamente N_B (2.1) segmentos. Para el cálculo del hash criptográfico del $i-$ésimo bloque h_i, se utilizará la función $HASH32([d_i|c_i])$, que permitirá validar si el bloque fue manipulado o no. La función $HASH32(.)$ es una función criptográfica cualesquiera, como por ejemplo: MD4, MD5, SHA-0, SHA-1, SHA-256 y SHA-512 [Stevens17, Zhang11b]. Además, la expresión $[a|b]$ indica la concatenación de vectores.

$$N_B = \frac{N \times M}{64} \tag{2.1}$$

Para la detección de bloques manipulados, se extrae d_i, c_i y h_i de cada bloque y se calcula el hash $\hat{h}_i = HASH32([d_i|c_i])$. Si el hash extraído h_i y el hash calculado $\hat{h}_i$ son iguales, consideraremos que el $i-$ésimo bloque no está manipulado y en caso contrario asumiremos que los vectores d_i y c_i fueron alterados y los marcaremos asignando el valor de -1 al bit correspondiente, para un procedimiento posterior de restauración.

Si en la imagen hay N_B bloques de 8×8, entonces para almacenar información (vector c) que permitirá recuperar la información manipulada, se tendrán para toda la imagen $160N_B = 160 \times N \times M/64 = 5 \times N \times M/2$ bits. El vector c se empleará en el proceso de restauración para recuperar los 5 MSB de la imagen (vector d). Por ejemplo, a través de una codificación que permita recuperar datos manipulados, ya sea de manera directa o a través de una compresión con perdida de información, tales como la DCT [He09] o promedios de píxeles [Lee08].

2.2. Análisis de la distribución de los bits dañados

Consideremos que el vector de datos d para toda la imagen, está manipulado en una tasa α. Éste lo permutaremos de manera pseudoaleatoria, con la finalidad de que segmentos completos de 320 bits marcados como manipulados (un bloque de 8×8), no queden contiguos. El vector d permutado, lo dividiremos en segmentos sin traslape de tamaño L, entonces en cada segmento se espera tener en promedio αL bits manipulados y $(1-\alpha)L$ bits no

manipulados. Sin embargo, esta condición no siempre se cumple y nos interesa para cada segmento de tamaño L calcular la distribución de probabilidad de que n bits se encuentren dañados.

Consideremos un experimento cuyo resultado es tomar un bit manipulado o no y que dicho resultado no dependa de experimentos anteriores. La variable aleatoria discreta X cuenta la cantidad de veces que ocurrieron manipulaciones, cuando el experimento se repite n veces. Si la probabilidad de obtener un bit manipulado es α y la probabilidad de que no esté manipulado es $(1-\alpha)$, entonces, la distribución de probabilidad de obtener exactamente n bits manipulados sigue la distribución binomial (2.2), donde $0 \leq \alpha \leq 1$, con media (2.3) y desviación estándar (2.4).

$$P(X = n|L, \alpha) = \begin{pmatrix} L \\ n \end{pmatrix} \alpha^n (1-\alpha)^{L-n} \tag{2.2}$$

$$\mu = \alpha L \tag{2.3}$$

$$\sigma = \sqrt{\alpha(1-\alpha)L} \tag{2.4}$$

En la Figura 2.2 se muestra la distribución de probabilidad de tomar exactamente n bits manipulados, para un segmento de tamaño L el cual tiene una probabilidad de estar manipulado en una tasa $\alpha = 0.3$. Dicho de otro modo, al dividir el vector de datos d en segmentos de tamaño $L = 32$, por cada segmento se se espera tener una distribución de datos manipulados como la que muestra la Figura 2.2, siendo $\mu = 0.3 \times 32 = 9.6$.

2.3. Códigos Hamming

Los códigos Hamming se emplean para detectar y corregir un bit erróneo en un canal de transmisión de datos. La manera en que corrigen y detectan errores, es a través de códigos de paridad los cuales son transmitidos con el mensaje. Estos siguen una nomenclatura definida por Hamming [Hamming50], la cual se describe a continuación. Para m bits de paridad, se transmitirán a lo más $n = 2^m - m - 1$ bits de datos, en un bloque de tamaño

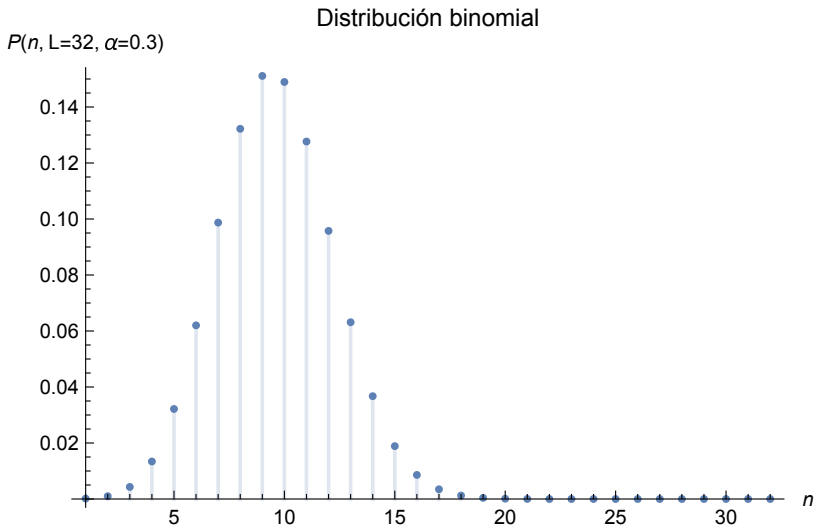


Figura 2.2: Distribución de probabilidad de tomar n bits manipulados en un segmento d_i de tamaño $L = 32$, con una tasa de manipulación $\alpha = 0.3$ en d

$L = 2^m - 1$. En la Tabla 2.1 se muestran cuatro columnas, la primera indica el número de bits de paridad m, la segunda es el número de datos n, la tercera es el tamaño del mensaje $L = n + m$ y en la última se indica el porcentaje de datos con respecto al tamaño del mensaje.

m	n	L	% datos
1	0	1	0%
2	1	3	33.33%
3	4	7	57.14%
4	11	15	73.33%
5	26	31	83.87%
6	57	63	90.47%

Tabla 2.1: Porcentaje de datos transmitidos por bloque

De la Tabla 2.1 se puede ver que a medida que se tienen más bits de paridad, se puede enviar una mensaje con un porcentaje mayor de datos. Sin embargo, esta codificación tiene el inconveniente de que si se altera más de un bit fallará al hacer la recuperación.

Los códigos Hamming se pueden codificar y decodificar mediante operaciones matriciales en aritmética módulo dos [Hamming50]. Por ejemplo, si se quiere transmitir un mensaje de 7 bits de datos, de acuerdo a la Tabla 2.1 se necesitarán 4 bits de paridad or-

ganizados de la siguiente manera $c = [p_1, p_2, d_1, p_4, d_2, d_3, d_4, p_8, d_5,\ d_6, d_7]^T$. Para calcular el mensaje c que se transmitirá, se utiliza la matriz de codificación A_G (2.5) en el cálculo matricial y para la detección de errores, se empleará la matriz A_H (2.6).

$$A_G = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5) \qquad A_H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}^T \quad (2.6)$$

Como ejemplo consideremos el vector de datos $d = [1, 0, 1, 0, 0, 1, 1]^T$, para calcular c hacemos $c = A_G d = [0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1]^T$. Si el destinatario recibe al vector c de manera íntegra, entonces la operación de detección $[p_1, p_2, p_4, p_8]^T = A_H c$ dará el resultado $[0, 0, 0, 0]^T$, lo cual indica que el mensaje ha llegado correctamente. En caso de un error y fuese el caso de que el quinto bit c_5 cambia de 0 a 1, el vector que recibe es $c' = [0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1]^T$. Al hacer la operación de detección $A_H c'$ dará como resultado $e = [1, 0, 1, 0]^T$. Si hacemos la conversión del vector e a decimal considerando que los bits van de izquierda a derecha, se tiene que $e = [1, 0, 1, 0][1, 2, 4, 8]^T = 5$. Por lo tanto para corregir, simplemente cambiamos c_5 de 1 a 0.

2.4. Códigos Reed-Solomon

Los sistemas de almacenamiento distribuido en red, consisten en repartir grandes cantidades de datos en múltiples nodos en la red. Para garantizar el correcto acceso a los

datos, ya que los fallos en la red son inminentes, se emplean los códigos Reed-Solomon. Estos códigos son una codificación convencional basada en los códigos de chequeo de paridad de baja densidad (LDPC por sus siglas en ingles), planteados en [Gallager62]. Los códigos Reed-Solomon, dividen los archivos en j símbolos $d = [d_1, d_2, \cdots, d_j]^T$ de tamaño L y se codifican en n símbolos $c = [c_1, c_2, \cdots, c_n]^T$ del mismo tamaño. Así los n símbolos codificados se almacenan en los distintos nodos de la red, uno en cada nodo. Esta codificación al igual que los códigos Hamming, se puede ver como la multiplicación matricial en aritmética módulo dos (2.7) [Tian14].

$$c = Ad \tag{2.7}$$

Donde la matriz A es binaria y se obtiene de alguna de las codificaciones que existen en la literatura, tales como: minimum storage regeneration MSR, minimum bandwidth regeneration MBR, locally repairable codes LRC y simple regeneration codes SRC. La restauración de los datos se hace de acuerdo a cada codificación [Tian14].

2.5. Códigos Tornado

En las grandes empresas que requieren de enviar software a sus usuarios, generalmente, lo hacían a través de transmisiones unicast. Sin embargo, cuando se trata de un archivo enviado a múltiples usuarios a la vez, la trasmisión unicast genera una gran carga en la red [Byers02]. Como solución a este problema, se utilizan los códigos Tornado planteados en [Luby97]. Estos se basan en un grafo bipartito disperso, que se puede modelar a través de una matriz de incidencias [Ai06].

La codificación en los códigos Tornado se hace en niveles (ver Figura 2.3) a través de la operación matricial (2.7), en aritmética módulo dos. Para ello la matriz A_i de cada nivel debe ser de dimensiones $\beta n \times n$, con $0 < \beta < 1$ y donde n indica el número de bits a codificar. Los βn bits de paridad generados por $c^{(1)} = A_1 d$, constituyen el primer nivel de códigos. Un segundo nivel se obtiene de la operación $c^{(2)} = A_2 c^{(1)}$, con A_2 de dimensiones $\beta(\beta n) \times \beta n = \beta^2 n \times \beta n$. Posterior a esto, los $\beta^2 n$ bits de paridad del segundo nivel, se emplean para obtener los $\beta^3 n$ bits de códigos de paridad del tercer nivel. Este mismo procedimiento se

repite hasta el m-ésimo nivel, donde se debe cumplir que $\beta^{m+1}n \approx \sqrt{n}$. Una vez obtenido el m-ésimo nivel, se utiliza una codificación convencional para obtener el último nivel de códigos de paridad $\hat{c}$. El código completo que se transmitirá es $c = [d, c^{(1)}, c^{(2)}, \cdots, c^{(m)}, \hat{c}]$, el cual contiene los n bits de datos d, los $n\beta/(1-\beta)$ bits de paridad $c^{(i)}$ y el código convencional $\hat{c}$ [Luby97]. La Figura 2.3 muestra gráficamente, cada uno de los niveles. En ésta se puede apreciar la tasa de decaimiento entre cada nivel, así como el hecho de que, para generar un nuevo nivel de códigos, se debe conocer el nivel previo.

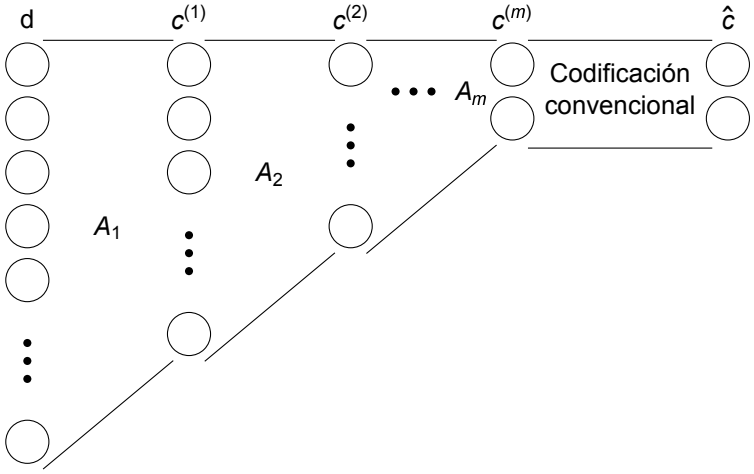


Figura 2.3: Representación gráfica de los códigos tornado

2.6. Recuperación de información

Los códigos para recuperación de información, analizados en las secciones previas, se modelan mediante la multiplicación matricial de una matriz A con un vector de datos d de tamaño N_d, dando como resultado un vector de códigos c de tamaño N_c. En el caso particular de una imagen, el vector d se obtiene de los 5 MSB y el vector c se almacenará como marca de agua en los 3 LSB, por esta razón N_d es mayor que N_c. La matriz A es una matriz aleatoria binaria, la cual se obtiene a partir de un generador de números aleatorios.

Para la codificación, el vector de datos $d = [d_1, d_2, \cdots, d_j, \cdots]$ se divide en vectores d_j de tamaño L. Los vectores que tendrán los bits de paridad c_j, se calculan mediante $c_j = Ad_j$ (2.7) y este vector tendrá un tamaño de L/k, donde k lo llamaremos un factor de

compresión y debe ser de tal magnitud que permita que el vector $c = [c_1, c_2, \cdots, c_j, \cdots]$ se pueda almacenar en los 3 LSB. Para la recuperación, considere que tanto el vector de bits d y el vector de códigos de paridad c, son alterados en una tasa α. Con lo que la Ecuación (2.7), la reformularemos como (2.8). Para (2.8) el vector d_j se separó en bits manipulados $d_{j,t}$ y bits no manipulados $d_{j,r}$ y el vector de códigos de paridad c_j, se separó en códigos manipulados $c_{j,t}$ y códigos no manipulados $c_{j,r}$.

$$\begin{bmatrix} c_{j,r} \\ c_{j,t} \end{bmatrix} = \begin{bmatrix} A_{r,r} & A_{r,t} \\ A_{t,r} & A_{t,t} \end{bmatrix} \begin{bmatrix} d_{j,r} \\ d_{j,t} \end{bmatrix} \tag{2.8}$$

Dado que se pretende recuperar los datos originales de $d_{j,t}$, entonces, se debe de resolver el sistema de ecuaciones mostrado en (2.9), mediante eliminación Gaussiana en aritmética módulo dos.

$$A_{r,t}d_{j,t} = c_{j,r} - A_{r,r}d_{j,r} \tag{2.9}$$

Considerando que la matriz A tiene dimensiones $L/k \times L$, la cantidad de bits manipulados $d_{j,t}$ en promedio es $N_t = \alpha L$ (2.3) y que el número de códigos de paridad no alterados en promedio es $N_r = (1-\alpha)L/k$ (2.3), para resolver el sistema se deberá cumplir $N_r \geq N_t$ (2.10).

$$(1-\alpha)\frac{L}{k} \geq \alpha L \tag{2.10}$$

Con lo que se podrán recuperar todos los bits originales en $d_{j,t}$, si se cumple (2.11).

$$\alpha \leq \frac{1}{k+1} \tag{2.11}$$

De la ecuación anterior (2.11), se desprende que solamente se podrá recuperar información hasta $1/(k+1)$, donde k relaciona el tamaño del vector de datos d con el vector de códigos de paridad c, en una relación $N_d = kN_c$. Por ejemplo, para un valor $k = 1/2$ el número de códigos de paridad será del doble del número de datos y de acuerdo con (2.11) se podrá recuperar el 100% de los datos en $d_{j,t}$, si éste está manipulado en hasta un 66%. En los casos de que k sea igual a $1, 2, 3, 4$, se podrá recuperar el 100% de los datos, si $d_{j,t}$ está manipulado en hasta 50%, 33%, 25%, 20%, respectivamente.

En el caso particular de almacenar una marca de agua en los 3 LSB de datos obtenidos de los 5 MSB, tendremos un factor de compresión $k = 3/5$ que tendrá la posibilidad de recuperar hasta un 62% de acuerdo con (2.11). Si llegáramos a utilizar este factor de compresión $k = 3/5$ no tendremos oportunidad de guardar el hash criptográfico en la marca de agua y en consecuencia no tenemos manera de detectar que la imagen fue alterada. Con $k = 2$, el vector de códigos de paridad será de la mitad que el vector de datos, entonces los bits más significativos de la imagen $320N_B$ producirán un vector de paridad de tamaño $160N_B$, que se almacenarán junto con el hash de tamaño $32N_B$ en los 3 LSB que tiene un tamaño de $192N_B$ $(160 + 32 = 192)$.

En [Zhang11b], aunque emplean una codificación como (2.7), para la recuperación de bits manipulados con un factor de compresión $k = 2$, reportan que recuperan la imagen original con tasas de manipulación de a lo más $\alpha = 0.28$. Esto es debido a que en la matriz aleatoria binaria A, cabe la posibilidad de que al momento de resolver el sistema (2.8), se obtenga una dependencia lineal que impida llegar a la solución. Con lo que, para conocer la capacidad real de recuperación, se hará el análisis de independencia lineal en la matriz A.

Se dice que las columnas de una matriz son linealmente dependientes, si existe al menos una columna, que se pueda obtener como la combinación lineal de las otras. De acuerdo con Zhang *et al.* en [Zhang08], para una matriz aleatoria binaria A de dimensiones $n \times m$, la probabilidad $D(n, m)$ de que sus columnas sean linealmente dependientes, está dada por (2.12).

$$D(n,m) = \begin{cases} 1 & m > n \\ 2^{-n} & m = 1 \\ D(n,m-1) + (1 - D(n,m-1))2^{m-n} & 2 \leq m \leq n \end{cases} \tag{2.12}$$

Por otra parte, los códigos de paridad no manipulados $c_{j,r}$, sigue la distribución binomial dada en (2.13).

$$P_c(n) = \begin{pmatrix} L/k \\ n \end{pmatrix} (1-\alpha)^n \alpha^{L/k-n} \tag{2.13}$$

El número de bits desconocidos $d_{j,t}$, sigue la distribución binomial dada por (2.14).

$$P_d(m) = \begin{pmatrix} L \\ m \end{pmatrix} \alpha^m (1-\alpha)^{L-m} \tag{2.14}$$

Por último, la probabilidad de que todas las columnas de la matriz A, sean linealmente independientes está dada por (2.15).

$$P_{LI} = \sum_{n=0}^{L/k} \sum_{m=0}^{L} P_c(n) \cdot P_d(m) \cdot (1 - D(n,m)) \tag{2.15}$$

En la Tabla 2.2, se muestra la probabilidad de independencia lineal P_{LI} de la matriz A, de dimensiones $L/2 \times L$, con tamaños $L = 128$, $L = 256$ y $L = 512$. En ésta se puede apreciar que la capacidad de recuperación, no sólo depende del factor de compresión k, sino que también influye el valor de L empleado para las dimensiones de la matriz A.

α \ Dimensiones	64×128	128×256	256×512
0	1	1	1
0.05	1	1	1
0.10	1	1	1
0.15	1	1	1
0.20	0.99	1	1
0.25	0.98	0.99	1
0.30	0.79	0.89	0.97
0.35	0.26	0.21	0.14
0.40	0.02	2.75×10^{-3}	5.45×10^{-5}

Tabla 2.2: Probabilidad de independencia lineal de la matriz A de dimensiones $L/2 \times L$

En la tabla anterior se observa, que a medida que L se incrementa, se mejora la recuperación de los datos. Esto debido a que al aumentar las dimensiones de la matriz A, entonces, existe una mayor cantidad combinaciones que se pueden dar en las columnas de A. Con lo que se disminuye la probabilidad de que dos o más columnas sean linealmente dependientes.

2.7. Trabajos previos de marcas de agua

En la literatura, existen múltiples trabajos de marcas de agua que permiten restauración de la imagen manipulada, sin embargo, estos se pueden dividir en dos principales categorías, las cuales denominaremos como: restauración exacta y restauración aproximada. A continuación, se describen de manera concreta algunos trabajos, separándolos en estas categorías.

2.7.1. Restauración exacta

Los trabajos de marcas de agua que realizan restauración exacta, consisten en construir una nueva imagen que tenga exactamente la misma información, que la imagen marcada. Estos tienen como principal ventaja, que pueden recuperar el contenido principal de la imagen marcada original. A continuación, se describen algunos trabajos que realizan este tipo de restauración.

Zhang *et al.* en [Zhang11b], presentan dos esquemas de marcado y restauración. El primero permite restauración exacta al emplear una formulación $c = Ad$ (2.7). Para ello, emplean la información de los 5 MSB de todos los píxeles de la imagen, como vector de datos d, para generar el vector de códigos de paridad c. Posterior a esto, el vector c es almacenado dentro de los 3 LSB de bloques de 8×8, junto con un hash criptográfico de 32 bits, que servirá para detectar las manipulaciones. La restauración de la imagen, se lleva a cabo de manera correcta al resolver el sistema de ecuaciones equivalente, cuando la imagen no presenta una manipulación mayor al 28%.

Bravo *et al.* en [Bravo-Solorio18], plantean un esquema similar al planteado en [Zhang11b]. En éste, plantean dos formulaciones $c_1 = A_1 d_1$ y $c_2 = A_2 d_2$ (2.7), en lugar de una única, como en [Zhang11b]. El resultado es un esquema de marcado y recuperación, con tiempos de ejecución más cortos que en [Zhang11b]. Además, la restauración se obtiene de manera correcta, si la imagen no presenta una manipulación mayor al 26%. La detección de manipulaciones se hace de igual manera que en [Zhang11b], a través de un hash criptográfico.

Qin *et al.* en [Qin16], proponen una formulación $c = Ad$ (2.7) para recuperar información, de manera similar a como se hace en [Zhang11b]. Sin embargo, para autenticar

el contenido proponen dos métodos basados en hash criptográfico de bloques de tamaño $b \times b$. En el primero, la imagen se divide en bloques sin traslape y en el segundo la imagen se divide en bloques con traslape. Otra cuestión interesante de este trabajo, es que no restringen a que la marca de agua se almacena en los 3 LSB de la imagen, en su lugar emplean los parámetros m y l. El parámetro m indica que en la marca se emplearán los m MSB como contenido principal de la imagen y el parámetro l indica que la marca se almacenará en los l LSB de la imagen. Con todo esto reportan que pueden recuperar de manera exacta el contenido manipulado de la imagen, con hasta un 28% de manipulación empleando $m = 5$ y $l = 3$.

2.7.2. Restauración aproximada

Para la restauración aproximada, generalmente se emplea una compresión del contenido de la imagen en los 5 bits más significativos, para la restauración de la imagen manipulada. Algunos trabajos dividen la imagen en bloques y emplean los promedios de los bloques para aproximar el contenido de los 5 MSB, otros emplean la DCT, o la transformada Wavelet, o alguna otra transformada. En general todo trabajo que emplee algún método que, al cambiar de representación, la imagen presente degradaciones al regresar a la representación inicial, se le considerará como restauración aproximada. En este tipo de trabajos, no solo se suele comparar el porcentaje máximo de manipulación, al que puede restaurar la imagen dañada, sino que también se comparan los niveles de calidad en PSNR de los píxeles restaurados. A continuación, se describen algunos de éstos.

Lee and Lin en [Lee08] emplean un esquema en el que guardan dos copias de la marca de agua en toda la imagen. Este consiste en subdividir la imagen en bloques de 2×2, de los cuales guardan los promedios de los 5 MSB de dos bloques en otros dos bloques distintos, seleccionados a partir de una tabla de mapeo. Como validación de la integridad de la imagen, emplean 2 bits de paridad que se calculan y almacenan por cada bloque. El contenido manipulado de la imagen original, se aproxima a través de los promedios almacenados en los 3 LSB de los bloques, generando así, la imagen restaurada. En su trabajo, presentan que pueden restaurar una imagen con un PSNR mayor a 20 dB, cuando la imagen presenta porcentajes de manipulación, hasta en un 85% de su contenido.

He *et al.* en [He09], plantean un esquema basado en la DCT. Para ello, subdividen la imagen en bloques de 8×8, luego cada bloque es caracterizado a través de 11 coeficientes de la DCT. Los 11 coeficientes se representan mediante 64 bits, los cuales se codifican y se almacenan en otro bloque, seleccionado aleatoriamente de acuerdo a una clave secreta. El método de validación de integridad que emplean, es un método estadístico el cual emplea las características del bloque en cuestión y las de sus vecinos. Por último, la construcción de la imagen restaurada se hace a partir de los 11 coeficientes que caracterizan a cada bloque. En su trabajo indican que la imagen restaurada, tiene un PSNR aceptable, siempre y cuando el contenido manipulado sea menor al 51% del total de la imagen.

Zhang *et al.* en [Zhang11c], presentan un esquema basado en los promedios de parejas de píxeles. Para ello, emparejan todos los píxeles dentro de la imagen y calculan el promedio de los 5 MSB de cada pareja. Una vez calculados todos los promedios, éstos se subdividen en grupos de 160 bits, los cuales se almacenarán en los 3 LSB por bloques de 8×8. Como método de validación, emplean un hash criptográfico de 32 bits por cada bloque, el cual se calcula a partir de los 5 MSB del bloque y los 160 bits del grupo de promedios. Así de este modo, el contenido manipulado de la imagen, se puede aproximar a partir de los promedios almacenados en la misma. En su trabajo presentan que la restauración se puede llevar a cabo de manera correcta, si el porcentaje de manipulación está por debajo del 54%, obteniendo una imagen con un PSNR en el rango entre 22 y 23 dB.

Zhang et al. en [Zhang11b], presentan dos esquemas de marcado y restauración. El segundo es un esquema de restauración que emplea los promedios de bloques de 4×4 y de 2×2, para aproximar el contenido principal de la imagen. Primeramente, dividen la imagen en bloques de 8×8, luego, cada bloque se subdivide en bloques de 4×4, de los cuales se calculan los promedios de cada bloque de 4×4. Luego, cada bloque de 4×4 a su vez se subdivide en bloques de 2×2 y se calculan sus promedios. Posterior a esto, se emplea una formulación como (2.7) para generar un vector de códigos de paridad c, donde todos los promedios obtenidos, se emplean como el vector de datos d. Por último, el vector c es almacenado dentro de los 3 LSB de bloques de 8×8, junto con un hash criptográfico de 32 bits. La imagen restaurada tiene un PSNR en el rango entre 22 y 40 dB. La restauración se lleva acabo de manera correcta al resolver el sistema de ecuaciones equivalente, cuando la

imagen no presenta una manipulación mayor al 66%.

Zhang *et al.* en [Zhang11a], presentan un esquema basado en la DCT. Para ello, subdividen la imagen en bloques de 8×8 y le calculan la DCT a cada bloque. Generan grupos con los coeficientes de 16 bloques, lo cual resulta en 1024 coeficientes por grupo. Posterior a esto, generan 368 referencias a partir de una formulación como (2.7), donde d es un vector con los 1024 coeficientes de la DCT y donde cada referencia es posteriormente representada mediante 7 bits. Por último, las referencias son agrupadas en grupos de 23 (161 bits), para ser almacenadas dentro de los 3 LSB de bloques de 8×8, junto con un hash criptográfico que servirá como método de validación. Para la recuperación plantean dos métodos; el primero consiste en resolver el sistema de ecuaciones, siempre y cuando el sistema equivalente sea de rango completo, el segundo consiste en un proceso iterativo el cual aproximará la solución, cuando el primer método no puede resolverlo. La restauración del contenido manipulado, se logra al aplicar la transformada discreta coseno inversa (IDCT por sus siglas en inglés), a los coeficientes correspondientes del bloque manipulado. En su trabajo presentan, que pueden restaurar la imagen de manera correcta con un PSNR mayor a 27 dB, siempre y cuando el porcentaje de manipulación no sea mayor al 60%.

Qin *et al.* en [Qin12], presentan un esquema de marcado y restauración basado en la transformada de contorno no submuestreada (NSCT por sus siglas en ingles). En este esquema, los bits de información que permitirá restaurar el contenido manipulado, se almacena en el dominio transformado, mientras que como método de validación se emplea el bit menos significativo (1 LSB) de cada píxel. En su trabajo, presentan que se puede restaurar la imagen con un PSNR mayor a 40 dB, si el porcentaje de manipulación es a lo más de un 27%.

Dadkhah *et al.* en [Dadkhah14], exploran las características de la descomposición en valores singulares (SVD por sus siglas en inglés) como mecanismo de detección y restauración. En este la restauración se puede llevar a cabo siempre y cuando, el porcentaje de manipulación no exceda al 55%. Además, la imagen restaurada tiene un PSNR mayor a 30 dB.

Zhang *et al.* en [Zhang15], emplean la compresión fractal como mecanismo de detección y restauración. A través de la compresión, logran almacenar tres marcas de agua

que permitirán tener dos posibilidades más de recuperar la información manipulada, en caso de que una falle. El resultado es un esquema que permite restaurar la información manipulada de manera correcta, cuando la imagen presenta a lo más un 80% de manipulación. El PSNR que obtienen está en el rango entre 22 y 44 dB.

Qin *et al.* en [Qin17], dividen la imagen en bloques de 3×3 con traslape de 3 píxeles. De cada bloque el píxel que se encuentra en el centro, tendrá los bits que permitirán detectar manipulaciones. Los píxeles de las esquinas de cada bloque, tendrán dos bits de referencia y los 4 píxeles restantes solo 1 bit de referencia. Esos bits de referencia son los que permitirán recuperar la información manipulada, a través de una formulación $c = Ad$ (2.7) similar que en [Zhang11b]. La restauración se consigue con ayuda de los promedios de cada bloque de 3×3. Reportan que pueden restaurar la imagen cuando ésta presenta tasas de manipulación de hasta un 45%, con niveles de calidad en PSNR entre el rango de 29 y 41 dB.

2.8. Conclusiones

Como se observa en la literatura, los esquemas de restauración aproximada permiten recuperar la información original, cuando se tienen porcentajes de manipulación mayor que los esquemas de restauración exacta. Con lo que se puede decir, que es una buena estrategia a seguir si se quiere restaurar el contenido manipulado, cuando se tiene un mayor porcentaje de manipulación.

En la literatura, algunos trabajos guardan información que permitirá restaurar un segmento manipulado de la imagen, dentro de otro segmento diferente con la esperanza de que ambos no sean manipulados. Ya que, de ser así no se podrá restaurar el contenido manipulado del primero. Por otra parte, esquemas que emplean un proceso de recuperación de datos como en [Zhang11b], tienen la certeza de que podrán recuperar el 100% de los datos, si el porcentaje de manipulación está por debajo del límite máximo permisible (28% para [Zhang11b]). Por lo que, en este trabajo, se empleará un proceso de recuperación de datos, para garantizar la correcta restauración de la imagen.

Si nuestro objetivo es lograr una recuperación cercana al 66% y si solamente te-

nemos $160 \times N_B$ bits para códigos de paridad c, será necesario tener un vector de datos d de tamaño $80 \times N_B$. Para lograr este cometido proponemos aplicar el estándar JPEG para compresión de imágenes, con la finalidad de reducir de $320 \times NB$ a $80 \times NB$, del vector de datos d. Con esto no solo lograremos aumentar el porcentaje de manipulación al que se podrá restaurar la imagen, sino que la formulación $c = Ad$ (2.7) permitirá garantizar la correcta restauración, si el porcentaje de manipulación está por debajo del límite máximo.

Capítulo 3

Uso de imágenes comprimidas como marcas de agua

En la actualidad existen diferentes métodos de compresión de imágenes con pérdida de información. Algunos de estos proporcionan altas tasas de compresión con niveles de calidad en PSNR mayores a 30 dB. En la Figura 3.1(a) se muestra la imagen de Lena original de 512×512 y en la Figura 3.1(b) se muestra la misma imagen de Lena, pero comprimida con el estándar JPEG en una calidad del 50%, con un PSNR de 34.92 dB. Sin embargo, para la imagen original se necesitan 256 KB para su almacenamiento en disco, mientras que para la imagen comprimida en formato JPEG, sólo requiere de 24 KB, lo cual resulta en una tasa de compresión de 0.0937. En la Figura 3.1(c) de igual manera, se muestra un paisaje dibujado a mano de 512×512 y en la Figura 3.1(d) se muestra ese paisaje pero en compresión JPEG con un 50% de calidad y con PSNR de 31.11 dB. En disco la imagen original requiere de 369.75 KB, mientras que la imagen comprimida solo requiere de 64 KB, lo cual resulta en una tasa de compresión de 0.173.

La marca de agua a construir será en el dominio espacial, almacenándola dentro de los 3 LSB de cada píxel de la imagen y tendrá la capacidad de restaurar los píxeles manipulados. Para ello, se implementará un mecanismo de recuperación de datos basado en una formulación $c = Ad$ (2.7). Como mecanismo para detectar manipulaciones, se implementará un hash criptográfico de 32 bits por cada uno de los bloques de 8×8. La restauración de la

(a) ***Lena original*** **(b)** ***Lena comprimida***

(c) ***Paisaje original*** **(d)** ***Paisaje comprimido***

Figura 3.1: Comparativa visual de una imagen comprimida con JPEG

imagen se hará a través de una versión comprimida de la misma; en consecuencia, se tendrá un esquema de restauración aproximado. Sin embargo, se espera que el resultado visual sea bastante similar al original. Además, dado que la compresión mediante JPEG no presenta las mismas tasas de compresión, se propondrá un algoritmo de compresión basado en el estándar JPEG que tenga una tasa de compresión constante. Entonces, para una imagen de dimensiones $N \times M$, de los $192N_B$ (3 LSB) bits se necesitarán $32N_B$ para la detección de manipulaciones basada en hash criptográfico. Con lo que para almacenar información que permitirá recuperar los datos comprimidos originales, se tendrán $160N_B$ bits para toda la imagen y a los cuales representaremos mediante el vector v.

En este capítulo, primeramente, se describe el algoritmo de compresión del estándar JPEG. Posterior a esto, se describen las modificaciones que se le harán a dicho algoritmo, con la finalidad de que siempre presente las mismas tasas de compresión. A continuación, se describe el algoritmo TDC1 para el marcado de imágenes, el cual almacena información de la imagen comprimida dentro de la marca de agua. Luego, se describe el algoritmo TDC2 para marcado de imágenes, el cual surgió durante el desarrollo del algoritmo TDC1. Por último, se describen los algoritmos para la extracción y restauración de imágenes, de los algoritmos TDC1 y TDC2.

3.1. Estándar JPEG

En el estándar de compresión JPEG, la imagen I se divide en bloques B_i sin traslape de tamaño 8×8 y se le restan 128 a cada píxel. Posteriormente a cada bloque se le aplica la DCT, $\hat{B}_i = DCT(B_i - 128)$ y los valores resultantes se cuantizan utilizando (3.1), empleando la matriz Q de cuantización (3.2), la cual corresponde a una calidad del 50% [Tuba14] y donde $Round(.)$ redondea al entero más cercano (3.3).

$$q[k,l] = Round\left(\frac{\hat{B}_i[k,l]}{Q[k,l]}\right) \qquad k,l = 1,2,\cdots,8 \tag{3.1}$$

$$Q = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 130 & 99 \end{pmatrix} \tag{3.2}$$

$$Round(a) = \lfloor a + 0.5 \rfloor \tag{3.3}$$

Los coeficientes cuantizados $q[k, l]$ de cada bloque, se ordenan en forma de zigzag [Wallace92] para luego ser seleccionados. Durante la selección, si un coeficiente cae por debajo de un umbral, el resto de los coeficientes de ese bloque se considerarán 0. El ordenamiento en zigzag, permite tomar primero los coeficientes de baja frecuencia espacial (valores que tras la cuantización son distantes de 0) y luego los de alta frecuencia (valores que tras la cuantización son cercanos a 0). En la Figura 3.2 se muestra dicho ordenamiento.

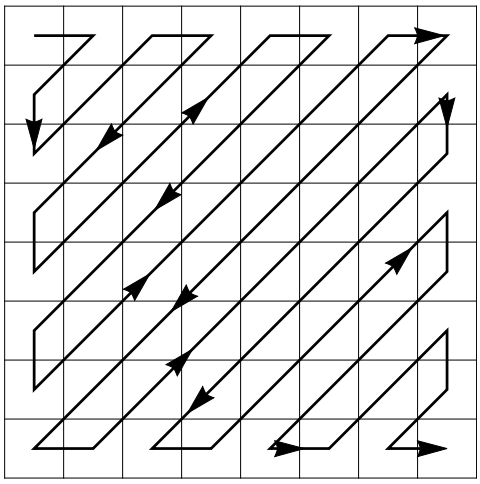


Figura 3.2: Orden de selección de los coeficientes de la DCT

Por último, todos los coeficientes resultantes de los bloques de 8×8, se codifican mediante algún método de compresión sin pérdida de información. Algunos de los métodos

de compresión sin perdida que se suelen emplear son: run-lenght encoding RLE, Huffman y Codificación Aritmética [La Serna09]. En el caso de Huffman, se toman todos los coeficientes cuantizados de la imagen y se ordenan de acuerdo a cuantas veces se repiten. Al coeficiente que tiene una mayor ocurrencia, se le asigna un símbolo pequeño que lo remplazará en el archivo final. Al siguiente con más ocurrencia se le asigna un símbolo mayor. Al tercer coeficiente con más ocurrencia, se le asigna un símbolo más grande y así sucesivamente.

Como ejemplo para la compresión sin perdida de información de Huffman, considere la secuencia de coeficientes $[15, 7, 10, 15, 7, 7, 15, 1, 7, 1, 0, 0, \cdots]$, obtenida de la ordenación en zigzag en un bloque de 8×8 (64 coeficientes). En la secuencia el coeficiente 0 se repite 54 veces, el coeficiente 7 se repite 4 veces, el coeficiente 15 se repite 3 veces, el coeficiente 1 se repite 2 veces y el coeficiente 10 se repite 1 vez. Si se utilizan los símbolos $01, 011, 0111, 01111, \cdots$, entonces, el tamaño de la cadena de símbolos resultante que se almacenará en lugar de los coeficientes será $2(54) + 3(4) + 4(3) + 5(2) + 6(1) = 148$. Si en lugar de la secuencia de coeficientes anterior, se hubiera obtenido la secuencia $[10, 9, 2, 10, 2, 2, 0, \cdots]$, entonces, el tamaño de la secuencia de símbolos hubiera sido $2(58) + 3(3) + 4(2) + 5(1) = 138$. Como se observa en el ejemplo, esta codificación provoca que dos secuencias con la misma cantidad de coeficientes, varíen en cuanto al tamaño final de la cadena de símbolos que remplazará a los coeficientes, dependiendo de su ocurrencia, lo que resulta en diferentes tasas de compresión. Si se pretende emplear el resultado de la compresión como vector de datos d, será problemático que el tamaño del vector d, varié entre imágenes con las mismas dimensiones. Ya que el espacio para la marca de agua en la imagen, es proporcional a las dimensiones.

3.1.1. Compresión JPEG10

Dado que se requiere aproximar únicamente el contenido principal de la imagen, se truncarán a 0 los 3 LSB y se trabajará únicamente con los 5 MSB. Ya que se busca una compresión constante, entonces la compresión sin pérdida de información del estándar JPEG no se realizará, ya que como se mencionó, ésta provoca diferentes tasas de compresión. En [Fridrich99] se indica que al utilizar la matriz de cuantización (3.2) y redondear al entero más cercano (3.3), se consigue de manera general que los primeros 11 coeficientes del ordenamiento en zigzag, se puedan representar con tan solo 64 bits. Sin embargo, para

ello, los primeros 4 coeficientes se deben codificar con 7 bits, los siguientes 2 con 6, los 4 siguientes con 5 bits y el último coeficiente con 4 bits. [Fridrich99] también indica que existen casos en los que 64 bits no bastan, así que para evitar este problema se representará cada coeficiente con 8 bits y en lugar de 11, se tomarán solo 10 coeficientes. Los 10 coeficientes que se tomarán, se localizan en las coordenadas dadas por (3.4). Por lo tanto, al tomar únicamente 10 coeficientes en lugar de los 64 de un bloque de 8×8, se consigue una tasa de compresión de $10/64$ y si además cada coeficiente se representa con la misma cantidad de bits, se consigue que la compresión sea constante. A esta propuesta de compresión le denominaremos JPEG10, JPEG por el estándar y 10 porque en un lugar de utilizar los 64 coeficientes de un bloque de 8×8, sólo se toman los primeros 10 del ordenamiento en zigzag.

$$C = \left[\ (0,0),(0,1),(1,0),(2,0),(1,1),(0,2),(0,3),(1,2),(2,1),(3,0) \ \right] \tag{3.4}$$

El Algoritmo 2 describe el proceso de compresión propuesto, éste recibe como parámetro la imagen I de dimensiones $N \times M$ y devuelve un único vector d_c, con los $10N_B$ coeficientes de toda la imagen en representación binaria (un vector de $80N_B$ bits). En el algoritmo, de la línea 5 a la línea 18 se analiza la imagen en bloques de 8×8. En la línea 7 se truncan a 0 los 3 LSB quedando únicamente los 5 MSB y se crea el bloque B_i de 8×8. En la línea 8 se le resta 128 al bloque B_i y se le aplica la DCT para generar el bloque $\hat{B}_i$. De la línea 9 a la línea 16, se toman los 10 coeficientes situados en las coordenadas indicadas en C (3.4). En la línea 12, cada coeficiente seleccionado es cuantizado y redondeado al entero más cercano, quedando en el rango $[-128, 127]$. Por último, en la línea 13 se le suma 128 al coeficiente (para que queden en el rango $[0, 255]$), se representa mediante 8 bits y se concatena al vector d_c.

La descompresión de la imagen I_c de dimensiones $N \times M$, parte del vector binario d_c el cual debe tener $80N_B$ elementos. Primero, se genera una matriz de dimensiones $N \times M$, la cual representará a la imagen I_c. Posterior a esto, el vector d_c se divide en segmentos sin traslape de 80 bits, cada uno de estos segmentos contiene a 10 coeficientes en binario de un bloque sin traslape de 8×8. Con cada segmento de 80 bits se generará un bloque temporal $\hat{B}_i$, al cual se le aplicará la IDCT (DCT inversa), $B_i = IDCT(\hat{B}_i)$. Por último, la imagen

Algoritmo 2 $JPEG10(I)$

1: $Q \leftarrow$ Matriz de cuantización del estándar
2: $d_c \leftarrow []$
3: $N \leftarrow Filas(I)$
4: $M \leftarrow Columnas(I)$
5: **para** $n = 0$ **hasta** N **paso** 8 **hacer**
6: **para** $m = 0$ **hasta** M **paso** 8 **hacer**
7: $B_i \leftarrow I[n : n + 8, m : m + 8]$ & 0XF8
8: $\hat{B}_i \leftarrow DCT((B_i - 128))$
9: **para** $k = 0$ **hasta** 8 **hacer**
10: **para** $l = 0$ **hasta** 8 **hacer**
11: **si** $(k, l) \in C$ **entonces**
12: $q \leftarrow Round(\hat{B}_i[k, l]/Q[k, l])$
13: $d_c \leftarrow [d_c|ToBin(q + 128)]$
14: **fin si**
15: **fin para**
16: **fin para**
17: **fin para**
18: **fin para**
19: **devolver** d_c

I_c se construye con el contenido de los bloques B_i sumándoles 128, ya que la IDCT los deja en el rango $[-128, 127]$, $I_c[n : n + 8, m : m + 8] = B_i + 128$.

El bloque temporal $\hat{B}_i$, se obtiene dividiendo cada segmento de 80 bits del vector d_c, en segmentos sin traslape de 8 bits. Cada uno de estos subsegmentos corresponden a un coeficiente binario, de los 10 que fueron tomados de las coordenadas dadas por (3.4). Los 54 coeficientes restantes que forman a $\hat{B}_i$ se establecen a 0, (3.5). En (3.5), la función $ToDec(.)$ realiza la conversión del segmento de 8 bits en un número entero en el rango $[0, 255]$. Por último, a cada coeficiente obtenido por $ToDec(.)$, se le restan 128 y se multiplica por su correspondiente en la matriz de cuantización (3.2). Con esto queda construido el bloque

temporal $\hat{B}_i$, permitiendo proceder a la aplicación de la IDCT para generar el bloque B_i.

$$\hat{B}_i[k,l] = \begin{cases} \textbf{si } (k,l) \in C & (ToDec(d_c[j:j+8]) - 128) \times Q[k,l] \\ \textbf{de otro modo} & 0 \end{cases} \tag{3.5}$$

El Algoritmo 3 describe el proceso de descompresión, éste recibe como parámetros, el vector binario d_c, las dimensiones N y M para la imagen resultante y regresa la imagen descomprimida I_c. En el algoritmo, de la línea 4 a la línea 20, se construye la imagen I_c recorriéndola en bloques de 8×8. De la línea 6 a la línea 16, se construye el bloque temporal $\hat{B}_i$ con los 10 coeficientes extraídos del segmento de 80 bits del vector d_c, revirtiendo la cuantización y rellenando con 0 los 54 coeficientes restantes (3.5). En la línea 17 se calcula la IDCT para generar el bloque B_i. Por último, en la línea 18 se sustituye el bloque de 8×8 final de I_c, con el contenido de B_i sumándole 128 a cada píxel, $I_c[n:n+8, m:m+8] = B_i + 128$.

En la Figura 3.3, se comparan los dos métodos de compresión empleando dos imágenes. En la Figura 3.3(a) se muestra a Lena comprimida con JPEG y en la Figura 3.3(b) comprimida con JPEG10. La imagen comprimida con JPEG tiene un PSNR de 34.92 dB, mientras que la comprimida con JPEG10 tiene un PSNR de 29.92 dB. De igual forma, en las Figuras 3.3(c) y 3.3(d) se presentan la imagen de un paisaje dibujado a mano, comprimido con JPEG y con JPEG10, respectivamente. El PSNR de la imagen comprimida con JPEG es de 31.11 dB y el PSNR de la imagen comprimida con JPEG10 es de 25.10 dB.

El algoritmo de compresión JPEG10, consigue mantener la misma tasa de compresión (10/64) sin importar la imagen a comprimir. Sin embargo, si la imagen a comprimir presenta muchos cambios en el color (altas frecuencias), el PSNR de la imagen comprimida decae a valores apenas aceptables PSNR $\geq$ 25 dB [Li07]. Aun así, las imágenes resultantes tienen una calidad visual bastante buena (se pueden apreciar pequeñas diferencias si se presta atención a los detalles), con lo que se considera apto para ser empleado en la restauración de la imagen manipulada.

Algoritmo 3 $JPEG10^{-1}(d_c, N, M)$

1: $Q \leftarrow$ Matriz de cuantización del estándar
2: $I_c \leftarrow Matriz(N, M)$
3: $j \leftarrow 0$
4: **para** $n \leftarrow 0$ **hasta** N **paso** 8 **hacer**
5: **para** $m \leftarrow 0$ **hasta** M **paso** 8 **hacer**
6: $\hat{B}_i \leftarrow Matriz(8, 8)$
7: **para** $k \leftarrow 0$ **hasta** 8 **hacer**
8: **para** $l \leftarrow 0$ **hasta** 8 **hacer**
9: **si** $(k, l) \in C\{(3.4)\}$ **entonces**
10: $\hat{B}_i \leftarrow (ToDec(d_c[j : j + 8]) - 128) \times Q[k, l]$
11: $j \leftarrow j + 8$
12: **si no**
13: $\hat{B}_i \leftarrow 0$
14: **fin si**
15: **fin para**
16: **fin para**
17: $B_i \leftarrow DCT^{-1}(\hat{B}_i)$
18: $I_c[n : n + 8, m : m + 8] \leftarrow B_i + 128$
19: **fin para**
20: **fin para**
21: **devolver** I_c

(a) ***Lena JPEG***

(b) ***Lena a partir de 10 coeficientes de la DCT***

(c) ***Paisaje JPEG***

(d) ***Paisaje a partir de 10 coeficientes de la DCT***

Figura 3.3: Comparativa visual de la compresión propuesta

3.2. Algoritmo TDC1 para el marcado de la imagen

Dada la imagen I de dimensiones $N \times M$ en escala de grises (8 bits por píxel), ésta se comprime a través del algoritmo de compresión JPEG10, generando al vector d_c, el cual tiene $N_{d,c} = 80N_B$ bits. Si de los 3 LSB ($192N_B$) de toda la imagen, se tiene que $32N_B$ se emplearán para los hash criptográficos, entonces para el vector de información v solo se tendrán $160N_B$ bits, en toda la imagen. Para poder realizar la restauración de la imagen, será necesario conocer la información referente a la imagen comprimida, con lo que ésta se tendrá que almacenará en el vector v. Ahora, si de los $160N_B$ bits del vector v se emplean $80N_B$ bits para almacenar al vector d_c, entonces quedarán $N_c = 80N_B$ para códigos de paridad c, que se emplearán en un proceso de recuperación de información basado en la formulación $c = Ad$ (2.7).

En la formulación $c = Ad_c$ (2.7), si el vector de códigos c tiene el mismo tamaño que el vector de datos comprimidos d_c, se tiene que, para la relación $N_c = kN_{d,c}$ propia de la formulación, se tendrá el valor de $k = 1$. Por lo que, las dimensiones de la matriz A que se emplearán en la formulación deben ser $L/k \times L = L \times L$. De la Ecuación (2.11), se puede decir que esta codificación garantizará recuperar los datos comprimidos originales, si la tasa de manipulación que presentan los vectores d_c y c es $\alpha \leq 0.5$.

En general, el algoritmo de marcado TDC1 consistirá en generar un vector de códigos de paridad c a partir del vector de datos comprimidos d_c, obtenido por el algoritmo de compresión JPEG10. Para ello se utilizará la codificación $c = Ad_c$ (2.7), donde el número de códigos a generar será el mismo que el de los datos comprimidos $N_c = N_{d,c}$; además, ambos vectores d_c y c, se almacenarán en los 3 LSB, como un único vector de información v. Éste será el primer algoritmo propuesto para el marcado de la imagen, el cual denominaremos TDC1, las letras TDC por que se emplea la transformada discreta coseno en su compresión y 1 porque se están generando la misma cantidad de códigos que de datos.

En la Figura 3.4, se muestra el diagrama general del proceso de marcado que se realizará para el algoritmo TDC1. En el diagrama, la imagen I se comprime para obtener el vector de datos comprimidos d_c. A continuación, el vector d_c se divide en segmentos de tamaño L y cada segmento se codifica empleando la formulación $c = Ad_c$ (2.7), la

cual genera a los segmentos de tamaño L que formarán al vector c. Posteriormente, los vectores d_c y c se concatenan y se les aplica una permutación pseudoaleatoria, para generar el vector de información v. La permutación tiene como finalidad, distribuir la información de los segmentos empleados en la codificación, a lo largo de toda la imagen. Al distribuir la información de esta manera, se logra que cuando el algoritmo de detección marque un bloque como manipulado, este marque como dañados unos cuantos bits de varios segmentos de tamaño L a la vez, en lugar de que marque muchos bits de un único segmento. Lo anterior debido a que como se mencionó, para que la codificación garantice la correcta recuperación de los datos originales, tanto el i-ésimo segmento del vector d_c como el i-ésimo segmento del vector c correspondiente, deben presentar una tasa de manipulación inferior a $\alpha \leq 0.5$. Continuando con el diagrama, una vez generado el vector de información v, se establecen a 0 los 3 LSB de la imagen I, quedando únicamente los 5 MSB. Con los 5 MSB y el vector v, se procede a generar los hash criptográficos de 32 bits que permitirán la detección de manipulaciones y que se almacenarán junto con el vector v en los 3 LSB de bloques de 8×8. Con esto concluye la construcción de la marca de agua.

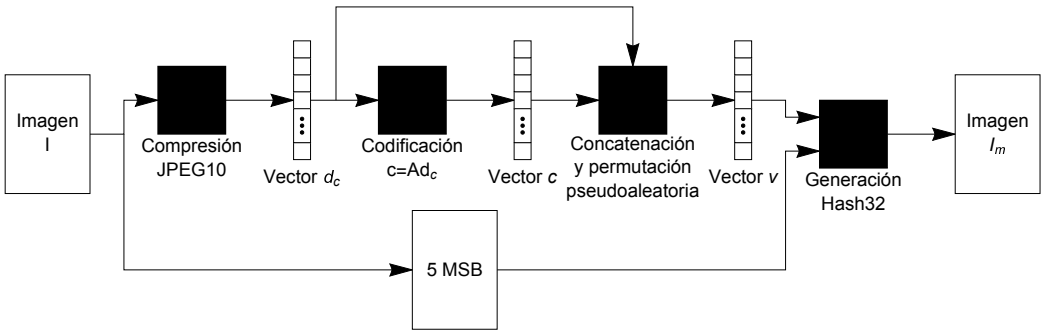


Figura 3.4: Diagrama general del proceso de marcado, para el algoritmo TDC1

El Algoritmo 4 describe el proceso de marcado de una imagen, empleando el algoritmo TDC1. Éste recibe como parámetros la imagen I, las dimensiones N y M de la imagen, el tamaño de segmento L, la clave secreta key y devuelve la imagen marcada I_m. En el algoritmo, en la línea 3 se copian los 5 MSB de la imagen I a la imagen I_m, recordando que los 5 MSB de la imagen original permanecen inalterables en la imagen marcada. En la línea 4 se genera la matriz aleatoria binaria A de dimensiones $L \times L$, empleando la

clave secreta *key* como semilla para generar números aleatorios. En la línea 5 se obtiene el vector de datos d_c a partir del algoritmo de compresión $JPEG10(.)$. De la línea 6 a la línea 8, se analiza en segmentos sin traslape de tamaño L al vector d_c y se generan los códigos de paridad c (también en segmentos de tamaño L). En la línea 9, se construye el vector v uniendo y permutando de manera pseudoaleatoria a los vectores d_c y c. Luego de la línea 10 a la línea 16, se construye la marca de agua final. Primeramente, en la línea 12 se genera el hash criptográfico de 32 bits, con los 5 MSB del bloque de 8×8 y un segmento v_i de 160 bits del vector v. Por último en la línea 13, se guardan en los 3 LSB del bloque, el segmento v_i y el hash h_i.

Algoritmo 4 $TDC1MarcaImagen(I, N, M, L, key)$

1: $c \leftarrow Vector(80N_B)$
2: $i \leftarrow 0$
3: $I_m = MSB5(I)$
4: $A \leftarrow MatrizAleatoriaBinaria(L, L, key)$
5: $d_c \leftarrow JPEG10(I)$
6: **para** $j \leftarrow 0$ **hasta** $80N_B$ **paso** L **hacer**
7: $\quad c[j : j + L] = A.d_c[j : j + L]$
8: **fin para**
9: $v \leftarrow Shuffle([d_c|c], key)$ //Al finalizar se divide en segmentos sin traslape de 160 bits
10: **para** $n \leftarrow 0$ **hasta** N **paso** 8 **hacer**
11: $\quad$ **para** $m \leftarrow 0$ **hasta** M **paso** 8 **hacer**
12: $\quad\quad h_i \leftarrow Hash32([MSB5(I[n : n + 8, m : m + 8])|v_i])$
13: $\quad\quad$ El vector $[v_i|h_i]$, se almacena en los 3 LSB del bloque $I_m[n : n + 8, m : m + 8]$
14: $\quad\quad i \leftarrow i + 1$
15: $\quad$ **fin para**
16: **fin para**
17: **devolver** I_m

3.3. Algoritmo TDC2 para el marcado de la imagen

Zhang *et al.* muestran que no es necesario almacenar al vector de datos d_c dentro de la marca de agua, que basta con que, en el proceso de detección de manipulaciones, no solo se marquen los bloques manipulados, sino que también sus coeficientes respectivos en el vector d_c [Zhang11c]. Si ya no es necesario almacenar al vector d_c en la marca de agua, los 160 bits de cada uno de los N_B bloques, se podrán emplear como códigos de paridad c. Dado que el vector d_c, sigue teniendo $80N_B$ bits, entonces, se podrá emplear un factor de compresión $k = 1/2$, para la codificación $c = Ad_c$ (2.7), generando ahora el doble de códigos de paridad que de datos comprimidos, $N_c = 2N_{d,c}$. Este nuevo algoritmo lo denominaremos TDC2 y ya no almacenará información del vector d_c en el vector v, además, tendrá la capacidad de restaurar la imagen a partir de la imagen comprimida, cuando la imagen presente una tasa de manipulación $\alpha \leq 0.66$.

En la Figura 3.5 se muestra el diagrama general del proceso de marcado, para al algoritmo TDC2. En el diagrama, primeramente, el vector d_c se obtiene a partir de la compresión JPEG10 y se permuta de manera pseudoaleatoria. Posterior a esto, el vector d_c se codifica en segmentos sin traslape de tamaño L, para generar el vector de códigos de paridad c (en segmentos de tamaño $2L$). Luego el vector c es permutado de manera pseudoaleatoria para formar al vector de información v. De igual forma que en la marca TDC1, estas permutaciones tienen como finalidad evitar que segmentos completos en los vectores d_c y c sean marcados como manipulados. Por último, la construcción de la marca de agua final, se realiza de igual manera que en el esquema anterior, a partir del vector v y los 5 MSB de la imagen I.

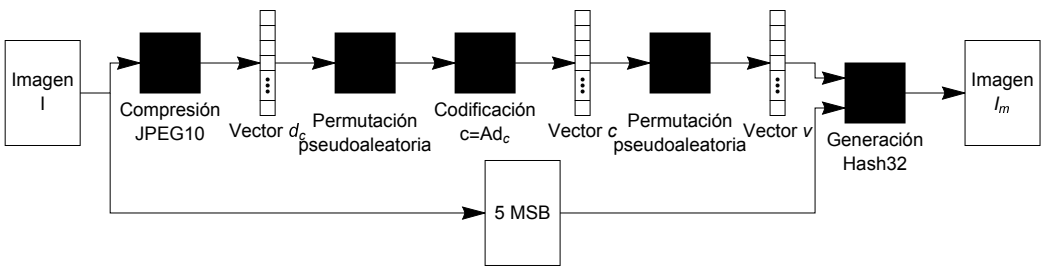


Figura 3.5: Diagrama general del proceso de marcado, para el algoritmo TDC2

En el Algoritmo 5 se describe el proceso de marcado para al algoritmo TDC2, este en general es el mismo que el del algoritmo TDC1 (Algoritmo 4), salvo algunos pequeños cambios. El primero se encuentra en la línea 2, donde para el vector c ahora se reserva memoria para $160N_B$ elementos, ya que ahora se requiere memoria para el doble de códigos de paridad. El segundo se muestra en la línea 4, donde la matriz aleatoria binaria A se genera de dimensiones $2L \times L$ en lugar de $L \times L$. El tercero está en la línea 5, donde el vector d_c (obtenido por el algoritmo de compresión JPEG10) es permutado de manera pseudoaleatoria antes de generar los códigos de paridad c, a través del vector d. El cuarto está en la línea 7, ya que por cada segmento de tamaño L en d_c, se genera un segmento de tamaño $2L$ en los códigos de paridad c. El último se muestra en la línea 9, donde se aprecia que el vector de información v, se obtiene solamente permutando de manera pseudoaleatoria al vector c.

Algoritmo 5 $TDC2MarcaImagen(I, N, M, L, key)$

1: $I_m = MSB5(I)$
2: $c \leftarrow Vector(160N_B)$
3: $i \leftarrow 0$
4: $A \leftarrow MatrizAleatoriaBinaria(2L, L, key)$
5: $d \leftarrow Shuffle(ZipJPEG(I), key)$
6: **para** $j \leftarrow 0$ **hasta** $80N_B$ **paso** L **hacer**
7: $\quad c[2j : 2(j + L)] = A.d[j : j + L]$
8: **fin para**
9: $v \leftarrow Shuffle(c, key)$ //Al finalizar se divide en segmentos sin traslape de 160 bits
10: **para** $n \leftarrow 0$ **hasta** N **paso** 8 **hacer**
11: $\quad$ **para** $m \leftarrow 0$ **hasta** M **paso** 8 **hacer**
12: $\quad\quad h_i \leftarrow Hash32([MSB5(I[n : n + 8, m : m + 8])|v_i])$
13: $\quad\quad$ El vector $[v_i|h_i]$, se almacena en los 3 LSB del bloque $I_m[n : n + 8, m : m + 8]$
14: $\quad\quad i \leftarrow i + 1$
15: $\quad$ **fin para**
16: **fin para**
17: **devolver** I_m

3.4. Extracción de la marca de agua y restauración de la imagen manipulada

La restauración de la imagen I para los algoritmos TDC1 y TDC2 se puede ver a través de tres pasos. El primero consiste en extraer la información de la marca de agua, de la cual $32N_B$ bits se emplean para detectar manipulaciones y los $160N_B$ bits restantes se emplean en el segundo paso. El segundo paso, recupera la mayor cantidad de bits manipulados en el vector d_c a partir de la información no manipulada, en el vector c. Por último, en el tercer paso, la restauración se hace a partir del vector de coeficientes binarios d_c, el cual se pasa como parámetro a la función $RestauraImagen(.)$. Esta función se detalla en la Subsección 3.4.3. Al finalizar el tercer paso, la imagen estará restaurada.

3.4.1. Extracción de la marca de agua para el algoritmo TDC1

La Figura 3.6 muestra el diagrama de extracción y restauración para el algoritmo de marcado TDC1. Dada la imagen I, se realiza el proceso de extracción y detección de manipulaciones. La extracción de la información se hace de los 3 LSB toda la imagen, de los cuales una parte se emplea en la detección de manipulaciones y el resto, en el proceso de recuperación de información (vector v). La detección de manipulaciones, se hace a través de los hash criptográficos de 32 bits, tal y como se mencionó en el Capítulo 2. Como resultado de la detección se obtiene un vector v manipulado y la imagen I_t. La imagen I_t es una copia de la imagen I, pero con los píxeles detectados como manipulados, identificados mediante el valor -1. Esto con la finalidad de facilitar la restauración de la imagen manipulada. Dado que en el proceso de marcado, el vector v se construyó como la permutación pseudoaleatoria de la concatenación de los vectores d_c y c, entonces hay que revertir dicha permutación, para poder extraer los vectores d_c manipulado y c manipulado. El siguiente paso es entonces, recuperar la mayor cantidad de bits en el vector d_c manipulado, para obtener el vector d_c. Una vez finalizada la recuperación de información, se procede a la construcción de la imagen restaurada I_r, a partir de la imagen I_t y el vector d_c, a través de la función $RestauraImagen(.)$.

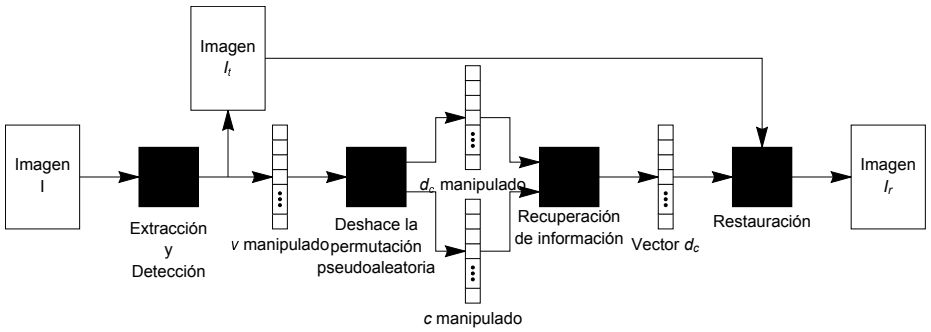


Figura 3.6: Diagrama de extracción y restauración de la marca de agua, para el algoritmo TDC1

El Algoritmo 6 describe el proceso mostrado en el diagrama. Éste recibe como parámetros la imagen I, sus dimensiones N y M, el tamaño de segmento L, la clave secreta key y regresa la imagen restaurada I_r. En el algoritmo, la detección de manipulaciones se realiza desde la línea 5 hasta la línea 16, primeramente, en la línea 7 se extraen los 3 LSB de un bloque de 8×8. Luego en la línea 8, de los 192 bits correspondientes a los 3 LSB del bloque, 160 remplazarán a un segmento del vector v, mientras que los 32 restantes se almacenan en el segmento h_i. En la línea 9 se calcula el hash temporal $\hat{h}_i$, que servirá para verificar que ese bloque no haya sido manipulado. De la línea 10 a la línea 13, se marcan como manipulados mediante el valor -1, los datos en v y los píxeles en I_t correspondientes a un bloque de 8×8. La recuperación de información en el vector de datos d_c, se realiza desde la línea 17 hasta la línea 24. Primeramente, en la línea 17, se deshace la permutación aleatoria en el vector v, para poder extraer los vectores d_c y c. Para la recuperación de información, ésta se hace en segmentos de tamaño L para los vectores d_c y c. Cuando se hace referencia a que las operaciones se realizan en aritmética módulo dos, esto indica que al terminar cada operación aritmética se le debe aplicar al resultado la operación módulo dos. Así por lo tanto, eliminación Gaussiana en aritmética modulo dos, es el algoritmo que conocemos, pero al finalizar el cálculo de un valor, a este se le debe aplicar la función modulo dos. Por último, la restauración de la imagen se realiza en la línea 25, a través de la función $RestauraImagen(.)$.

Algoritmo 6 $ExtraeMarcaTDC1(I, N, M, L, key)$

$i \leftarrow 0$
$v \leftarrow Vector(160N_B)$
$I_t \leftarrow I$
$A \leftarrow MatrizAleatoriaBinaria(L, L, key)$
para $n \leftarrow 0$ **hasta** N **paso** 8 **hacer**
  para $m \leftarrow 0$ **hasta** M **paso** 8 **hacer**
    $lsb3 \leftarrow LSB3(I[n : n + 8, m : m + 8])$
    $[v[160i : 160i + 160], h_i] \leftarrow [lsb3[0 : 160], lsb3[161 : 192]]$
    $\hat{h}_i \leftarrow Hash32([MSB5(I[n : n + 8, m : m + 8])|v[160i : 160i + 160]])$
    si $h_i \neq \hat{h}_i$ **entonces**
      $v[160i : 160i + 160] \leftarrow -1$
      $I_t[n : n + 8, m : m + 8] \leftarrow -1$
    fin si
    $i \leftarrow i + 1$
  fin para
fin para
$\hat{v} \leftarrow Shuffle^{-1}(v, key)$
$[d_c, c] \leftarrow [\hat{v}[0 : 80N_B], \hat{v}[80N_B : 160N_B]]$
para $i \leftarrow 0$ **hasta** $80N_B$ **paso** L **hacer**
  Separar $d_c[i : i + L]$ en datos manipulados d_t y no manipulados d_r.
  Separar $c[i : i + L]$ en códigos manipulados c_t y no manipulados c_r.
  Resolver (2.9) para d_t con eliminación Gaussiana en aritmética módulo dos.
  Sustituir d_t en el segmento original $d_c[i : i + L]$
fin para
$I_r \leftarrow RestauraImagen(I_t, d_c, N, M)$
devolver I_r

3.4.2. Extracción de la marca de agua para el algoritmo TDC2

Dado que ya no se tiene en la marca de agua información del vector de datos comprimidos d_c, dicho vector se debe obtener de la imagen que se presume manipulada; sin embargo, ésta contiene información falsa, la cual se debe descartar en el vector de datos comprimidos d_c. Para ello, primeramente, se deben de marcar como manipulados los bits en un vector temporal $d_{c,t}$, que correspondan a los bloques marcados como manipulados en la imagen. En nuestro caso, dado que el algoritmo de compresión JPEG10 genera el vector de datos comprimidos d_c, concatenando 80 bits por cada bloque de 8×8 a dicho vector, entonces, esta tarea no será complicada debido a que la detección también está basada en dividir la imagen en bloques de 8×8. Ya con el vector $d_{c,t}$ marcado como manipulado, se procede de manera similar a como se hace en la marca de agua TDC1.

El diagrama de la Figura 3.7 muestra el proceso de restauración de la imagen I, para el algoritmo de marcado TDC2. En el diagrama, primeramente se comprime la imagen I al vector $d_{c,t}$. Posterior a esto, el vector $d_{c,t}$ y la imagen I son empleados en la extracción y detección de manipulaciones. Al finalizar la extracción y detección, se tiene la imagen I_t, el vector de información manipulado v_m y el vector $d_{c,t,m}$ manipulado. A continuación, al vector $d_{c,t,m}$ se le aplica una permutación pseudoaleatoria y en el vector v_m se deshace la permutación pseudoaleatoria, obteniendo así los vectores manipulados d_m y c_m, respectivamente. Ya con los vectores d_m y c_m se hace la recuperación de datos manipulados y se obtiene el vector d. El vector d contiene la información comprimida de la imagen pero se encuentra desordenada, con lo que hay que ordenarla deshaciendo la permutación pseudoaleatoria que se hizo al inicio de la restauración, con esto se obtiene el vector d_c. La restauración de la imagen I_r, se hace a partir de la imagen I_t y el vector de datos d_c a través de la función $RestauraImagen(.)$, descrita en la siguiente sección.

El Algoritmo 7 describe el proceso de extracción de la marca de agua TDC2, éste recibe como parámetros la imagen I, las dimensiones N y M, el tamaño de segmento L, la clave secreta key y devuelve la imagen restaurada I_r. En el algoritmo, la detección de manipulaciones tanto en la imagen como en el vector $d_{c,t}$, se realiza de la línea 5 a la línea 18. Para ello, en la línea 5 se obtiene el vector de datos comprimidos temporal $d_{c,t}$, el cual

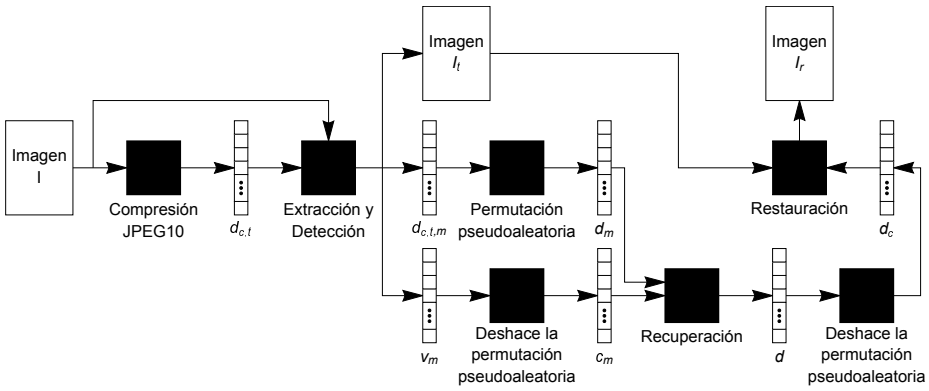


Figura 3.7: Diagrama de restauración para la marca de agua TDC2

contiene la información comprimida de la imagen I, ya sea que este manipulada o no. Ya en el proceso de detección de manipulaciones por bloques de 8×8, se realiza la extracción de los 3 LSB y se separan en 160 bits que sustituirán a un segmento del vector v y 32 bits que se almacenarán en el segmento h_i. A continuación se calcula el hash temporal $\hat{h}_i$, que se empleará para determinar si el bloque ha sido manipulado o no. Luego de la línea 11 a la línea 15, se marcan los datos del vector v, los píxeles de la imagen I_t y los datos comprimidos del vector $d_{c,t}$ con el valor -1, si el bloque en cuestión fue detectado como manipulado. La recuperación de información en el vector d, se realiza de la línea 19 a la línea 26. Primeramente, para obtener al vector d, en la línea 19 se le aplica al vector $d_{c,t}$ una permutación pseudoaleatoria. Para el vector de códigos c, al vector v se le deshace la permutación pseudoaleatoria hecha antes de almacenarlo en los 3 LSB. La recuperación de información, se realiza en segmentos de tamaño L para el vector d y en segmentos de tamaño $2L$ para el vector c. Al finalizar la recuperación, para obtener el vector de datos comprimidos d_c, al vector d se le deshace la permutación pseudoaleatoria. Por último, la restauración de la imagen se realiza en la línea 28.

Algoritmo 7 $ExtraeMarcaTDC2(I, N, M, L, key)$

$v \leftarrow Vector(160N_B)$
$I_t \leftarrow I$
$i \leftarrow 0$
$A \leftarrow MatrizAleatoriaBinaria(2L, L, key)$
$d_{c,t} \leftarrow JPEG10(I)$
para $n \leftarrow 0$ **hasta** N **paso** 8 **hacer**
  para $m \leftarrow 0$ **hasta** M **paso** 8 **hacer**
    $lsb3 \leftarrow LSB3(I[n : n + 8, m : m + 8])$
    $[v[160i : 160i + 160], h_i] \leftarrow [lsb3[0 : 160], lsb3[161 : 192]]$
    $\hat{h}_i \leftarrow Hash32([MSB5(I[n : n + 8, m : m + 8])|v[160i : 160i + 160]])$
    si $h_i \neq \hat{h}_i$ **entonces**
      $v[160i : 160i + 160] \leftarrow -1$
      $d_{c,t}[80i : 80i + 80] \leftarrow -1$
      $I_t[n : n + 8, m : m + 8] \leftarrow -1$
    fin si
    $i \leftarrow i + 1$
  fin para
fin para
$d \leftarrow Shuffle(d_{c,t}, key)$
$c \leftarrow Shuffle^{-1}(v, key)$
para $i \leftarrow 0$ **hasta** $80N_B$ **paso** L **hacer**
  Separar $d[i : i + L]$ en datos manipulados d_t y no manipulados d_r.
  Separar $c[2i : 2i + 2L]$ en códigos manipulados c_t y no manipulados c_r.
  Resolver (2.9) para d_t con eliminación Gaussiana en aritmética módulo dos.
  Sustituir d_t en el segmento original $d[i : i + L]$
fin para
$d_c \leftarrow Shuffle^{-1}(d, key)$
$I_r \leftarrow RestauraImagen(I_t, d_c, N, M)$
devolver I_r

3.4.3. Restauración de la imagen manipulada

La restauración de la imagen manipulada para los algoritmos TDC1 y TDC2, se realiza primeramente extrayendo la imagen comprimida I_c del vector de datos d_c, para luego analizar cada píxel de la imagen I_t. Para ello, si un píxel en I_t está marcado como manipulado, entonces en la imagen I_r se copiará el píxel de la imagen I_c correspondiente, en caso contrario se copiará el píxel de la imagen I_t (3.6).

$$I_r[n,m] = \begin{cases} \textbf{si } I_t[n,m] = -1 & I[n,m] \\ \textbf{si no} & I_t[n,m] \end{cases} \tag{3.6}$$

En el Algoritmo 8 se describe el proceso de restauración, el cual recibe como parámetros la imagen manipulada I_t (donde los píxeles manipulados se marcaron con -1), el vector d_c, las dimensiones N y M de la imagen final y regresa la imagen restaurada I_r. En el algoritmo, en la línea 1 se genera una matriz de dimensiones $N \times M$, que representará a la imagen I_r. En la línea 2 se extrae la imagen comprimida en el vector d_c. Por último, de la línea 3 a la línea 11, se revisa cada píxel en I_t, donde si éste está marcado como manipulado, en I_r se le asigna el píxel de I_c, en caso contrario se le asigna el píxel de la imagen I_t.

Algoritmo 8 *Restaura*(I_t, d, N, M)

```
1:  I_r ← Matriz(N, M)
2:  I_c ← ZipJPEG^{-1}(d, N, M)
3:  para n ← 0 hasta N hacer
4:     para m ← 0 hasta M hacer
5:        si I_t[n, m] = −1 entonces
6:           I_r[n, m] = I_c[n, m]
7:        si no
8:           I_r[n, m] = I_t[n, m]
9:        fin si
10:    fin para
11: fin para
12: devolver I_r
```

3.5. Conclusiones

Las algoritmos TDC1 y TDC2, tienen como ventaja que se puede recuperar la imagen manipulada en tasas de hasta como máximo un $\alpha = 0.5$ para el algoritmo TDC1 y de hasta como máximo un $\alpha = 0.66$ para el algoritmo TDC2. Sin embargo, dado que la restauración es en base a una imagen comprimida, la calidad de la imagen restaurada dependerá directamente del algoritmo de compresión JPEG10.

Dado que los algoritmos TDC1 y TDC2, almacenan la marca de agua en los 3 LSB del dominio espacial de la imagen, entonces estas caen en la categoría de marca de agua frágil. Por consiguiente, cualquier alteración por mínima que sea, alterará parcial o totalmente la marca de agua, en nuestro caso es parcialmente (bloques de 8×8).

Si las dimensiones de la imagen N y M no son múltiplos de 8, entonces, los algoritmos TDC1 y TDC2, ignorarán los píxeles que no caigan dentro de un bloque completo de 8×8. Con lo que no habrá manera de restaurarlos en caso de que sean manipulados.

Para la recuperación de datos manipulados, se usará eliminación Gaussiana con valores de $L = 1024$ como máximo, ya que como se menciona en [Bravo-Solorio18], este genera tiempos de marcado del orden de 25 segundos para un $L = 512$ lo cual es impráctico, ya que se requiere marcar una imagen cuando está es capturada.

Capítulo 4

Resultados

En este capítulo, primeramente, se describen algunas pruebas para mostrar el funcionamiento de los algoritmos de marcado propuestos. Posterior a esto, se compararán con el trabajo de [Zhang11b], el cual fue la base para este trabajo y por último se compararán con los trabajos en el estado del arte citados.

4.1. Pruebas individuales

Las implementaciones de los algoritmos, se realizaron en C++, con los siguientes parámetros de compilación: -Wall, -Werror, -Wextra, -march=native, -mtune=native, -pipe y -O2. La computadora empleada para las pruebas, es una iMac de finales del 2013, con procesador Intel i5 a 2.7 GHz, 8 GB de RAM a 1600 MHz y gráficos Intel Iris Pro 1536 MB.

En la Figura 4.1 se muestra la imagen marcada con el algoritmo TDC1 correspondiente a un paisaje. Como manipulación, se tapó con color blanco algunas áreas de la Figura 4.1, la imagen ya manipulada se muestra la Figura 4.2. En la Figura 4.3, se muestran en gris los píxeles detectados como manipulados, en esta imagen se aprecia claramente que el algoritmo de detección de manipulaciones, marca bloques completos de 8×8, al momento de hacer la detección de manipulaciones. En la Figura 4.4 se muestra la imagen restaurada a partir de la imagen de la Figura 4.2. Como se puede apreciar, el algoritmo pudo restaurar de manera correcta los píxeles manipulados, lo cual se refleja en una imagen donde se aprecia claramente las áreas en blanco de Figura 4.2, que fueron restauradas.

Figura 4.1: Imagen de un paisaje, marcada con el algoritmo TDC1

Figura 4.2: Imagen manipulada con color blanco, en algunas de las regiones de la imagen de la Figura 4.1

Figura 4.3: Imagen donde en gris se muestran los píxeles detectados como manipulados, en la imagen de la Figura 4.2

Figura 4.4: Imagen restaurada a partir de la imagen de la Figura 4.2

En la Figura 4.5 se muestra una imagen de una catarata, que se marcó con el algoritmo TDC2. En la Figura 4.6 se muestra la imagen de la Figura 4.5, la cual se manipuló rellenando con color blanco, algunas regiones de la imagen. En la Figura 4.7 se muestran en color gris los píxeles detectados como manipulados en la imagen de la Figura 4.6. En la Figura 4.8 se muestra la imagen restaurada a partir de la marca de agua incrustada en la misma imagen de la Figura 4.6. De igual manera que en el ejemplo anterior, se aprecia de manera clara, todo el contenido que se había tapado con el color blanco, incluso visualmente, no se llegan a apreciar diferencias entre la imagen restaurada de la Figura 4.8 y la imagen con la marca de agua de la Figura 4.5.

Figura 4.5: Imagen correspondiente a una catarata marcada con el algoritmo TDC2

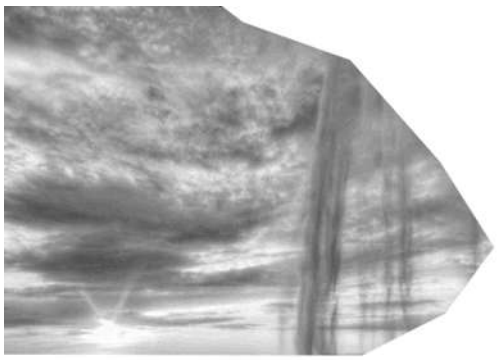


Figura 4.6: Imagen manipulada correspondiente a la Figura 4.5, donde se relleno de color blanco algunas de las regiones de la imagen

Figura 4.7: Imagen en donde en gris se muestran los píxeles detectados como manipulados, de la Figura 4.6

Figura 4.8: Imagen restaurada a partir de la imagen de la Figura 4.6

En un ejemplo un poco más real, en el que se busca alterar la percepción de un lugar, ya se agregando contenido inexistente o simplemente, cambiando de posición algunos objetos. Considere la imagen de dimensiones $1,976 \times 1,312$ mostrada en la Figura 4.9, la cual se marcó mediante el algoritmo TDC1 con $L = 1040$. La imagen de la Figura 4.10 se manipulo con una transformación del tipo espejo, en la que a simple vista se puede alcanzar a apreciar donde se ha hecho la manipulación indicada. En la Figura 4.11, se muestran en gris los píxeles detectados como manipulados, por el algoritmo de detección de manipulaciones implementado, estos corresponden a un 44.1296% del total de píxeles de la imagen. En la Figura 4.12, se muestra la imagen restaurada a partir de la marca de agua incrustada en la imagen. La restauración tardo 2 minutos y 57 segundos y tiene un PSNR de 33.7319 dB. Comparando a simple vista las imágenes de la Figura 4.12 y de la Figura 4.9, se alcanzan a apreciar pequeñas diferencias en los detalles causadas por el algoritmo de compresión JPEG10.

Figura 4.9: Imagen de un conjunto de edificios, marcada con el algoritmo TDC1

Figura 4.10: Imagen manipulada correspondiente a la Figura 4.9, con una transformación del tipo espejo

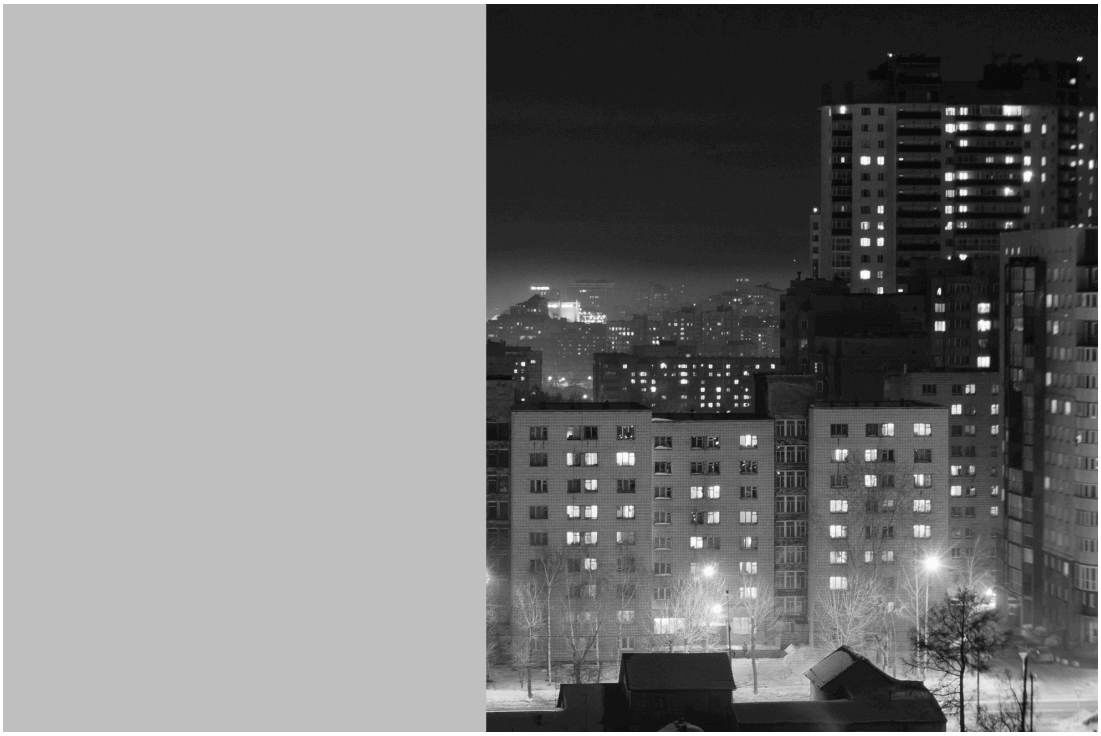


Figura 4.11: Imagen donde en gris se muestran los píxeles detectados como manipulados, de la imagen de la Figura 4.10

Figura 4.12: Imagen restaurada a partir de la imagen de la Figura 4.10

Ahora considere la imagen de la Figura 4.13 de dimensiones $1,976 \times 1,312$, la cual fue marcada mediante el algoritmo TDC2, con $L = 1040$. La imagen de la Figura 4.14 se manipuló con una transformación del tipo espejo, sin embargo, cuesta más trabajo detectar donde se ha hecho la manipulación. La Figura 4.15 muestra en gris los píxeles detectados como manipulados, por el algoritmo de detección de manipulaciones, los cuales corresponden al 62.9036% del total de píxeles en la imagen. La imagen restaurada a partir de la marca de agua incrustada se muestra en la Figura 4.16, esta tiene un PSNR de 32.1267 dB y se tardó 8 minutos y 13 segundos, en realizar dicha restauración. Al comparar visualmente la imagen con la marca original y la imagen restaurada, a simple vista es muy difícil detectar las diferencias entre una y otra.

Cabe resaltar, que las diferencias en tiempo de restauración, es debido a que al momento de recuperar los bits originales, se emplea eliminación Gaussiana para resolver el sistema de ecuaciones, a cada segmento de tamaño L, en el vector de datos comprimidos d_c. El algoritmo de eliminación Gaussiana es de orden $\mathcal{O}(N^3)$, con lo que, sí el algoritmo TDC1 en promedio tiene que recuperar $0.441296(1040) \approx 459$ bits dañados, mientras que el algoritmo TDC2 en promedio recupera $0.629036(1040) \approx 654$ y sí además, cada uno tiene que aplicar eliminación Gaussiana en $(80(1,976 \times 1,312/64))/1040 = 3,116$ segmentos de tamaño L, entonces, la diferencias de tiempos de computo son evidentes. El algoritmo de marcado TDC2 mejora en cuanto al porcentaje de manipulación máximo al que puede restaurar los píxeles manipulados, pero también para lograrlo se requerirá de mucho mayor tiempo de computo que el algoritmo TDC1. Así que, si se requiere que la restauración de las imagen se obtengan de manera rápida, entonces hay emplear valores de L más pequeños, en especial para el algoritmo TDC2.

Figura 4.13: Imagen marca con el algoritmo TDC2, correspondiente a un edificio

Figura 4.14: Imagen manipulada a partir de la imagen de la Figura 4.13

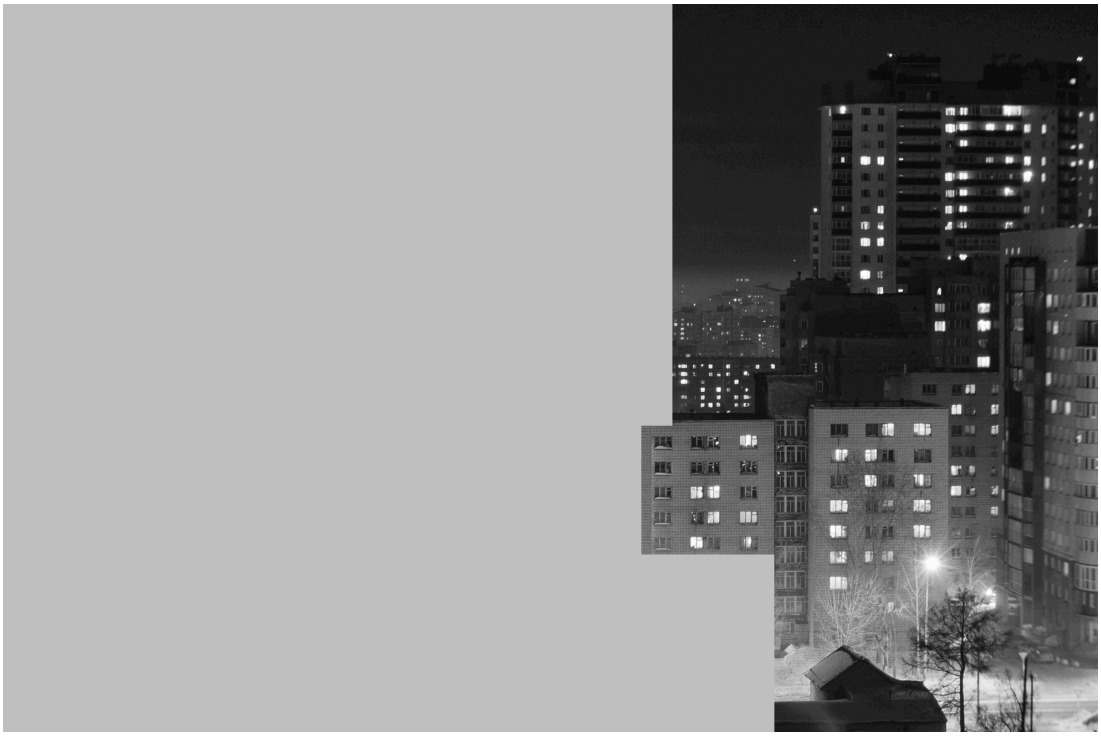
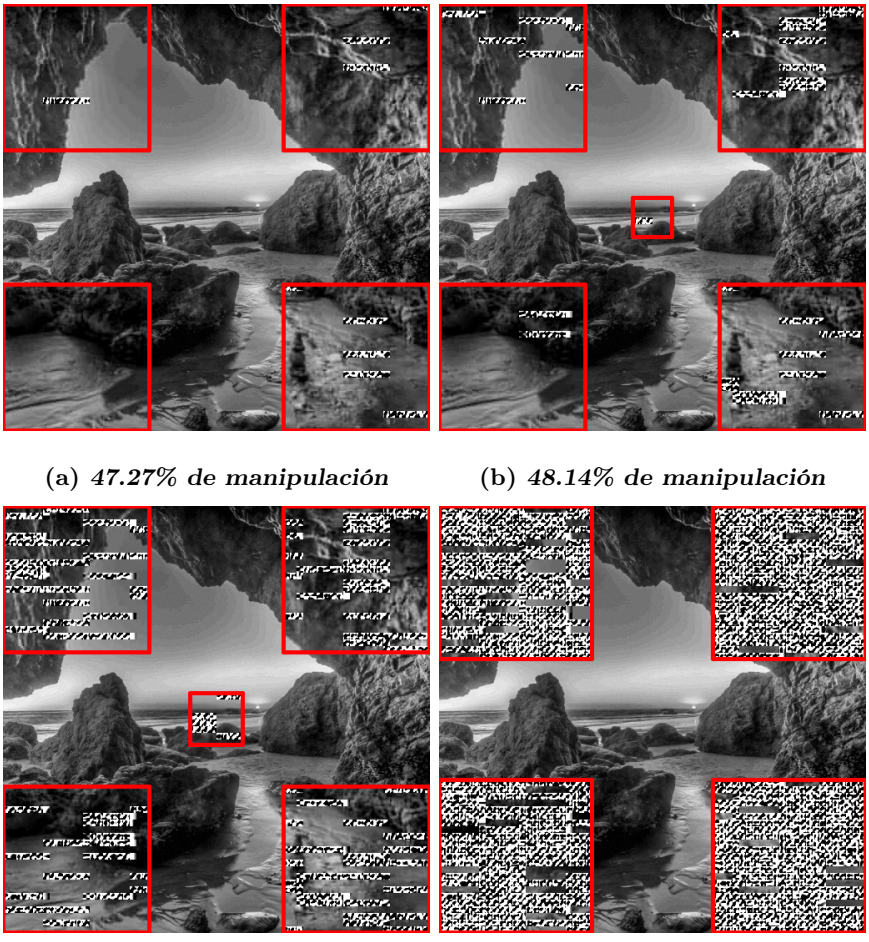


Figura 4.15: Píxeles detectados como manipulados en la imagen de la Figura 4.14

Figura 4.16: Imagen restaurada con el algoritmo TDC2, a partir de la imagen manipulada de la Figura 4.14

La formulación $c = Ad$ (2.7), garantiza la correcta recuperación de los datos siempre y cuando se presente una taza de manipulación por debajo de un límite máximo. Este límite depende principalmente de la relación en las dimensiones de la matriz A (el valor de k en $L/k \times L$) y en menor medida, el valor de L, empleadas en la formulación. Las imágenes mostradas en la Figura 4.17, muestran que es lo que pasa cuando se supera el límite de manipulación que permite el algoritmo de marcado TDC1. A medida que el porcentaje de manipulación va aumentado, empiezan a resaltar los segmentos de tamaño L, que no se pudieron recuperar por la formulación, lo que resulta en una imagen distorsionada. Las área remarcadas en rojo, indican donde se realizaron las manipulaciones para las pruebas.

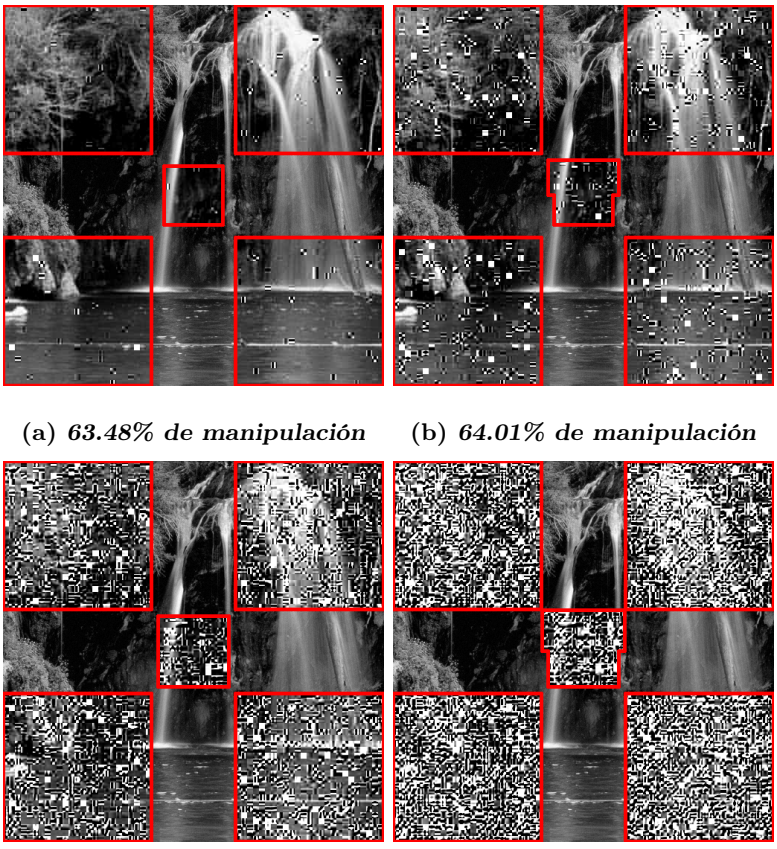


(a) ***47.27% de manipulación*** **(b)** ***48.14% de manipulación***

(c) ***48.83% de manipulación*** **(d)** ***51.66$ de manipulación***

Figura 4.17: Restauración en los límites del algoritmo TDC1

En la Figura 4.18 se muestra lo que pasa, cuando el porcentaje de manipulación supera al límite máximo, que permite el algoritmo de marcado TDC2. En rojo, se muestran las áreas manipuladas para las pruebas. En las imágenes, se presentan problemas de restauración en bloque de 8×8 aislados, a diferencia del algoritmo TDC1 que parece como si 5 o 6 bloques contiguos, fallaran a la vez. Esto es, debido a como se maneja la información en la marca de agua. Para el algoritmo TDC1, el vector de datos comprimidos d_c, se concatena al vector de códigos c y se permutan, para luego ser almacenados en la marca de agua. En las imágenes restauradas esos 5 o 6 bloques de 8×8, reflejan que cuando se hace la recuperación de datos en el vector d_c (después de deshacer la permutación), un segmento de tamaño $L = 512$ no se pudo recuperar. Lo que implica que los $512/80 = 6.4$ segmentos contiguos de 80 bits (10 coeficientes) en el vector d_c, aun tengan datos dañados.



(a) ***63.48% de manipulación*** **(b)** ***64.01% de manipulación***

(c) ***64.55% de manipulación*** **(d)** ***65.19% de manipulación***

Figura 4.18: Restauración en los límites del algoritmo TDC2

4.2. Pruebas generales

Para las pruebas generales, se probaron los algoritmos TDC1, TDC2 y el algoritmo de recuperación exacta de [Zhang11b], que fue la base de este trabajo. Para cada uno de los algoritmos, se emplearon 50 imágenes de 512×512 píxeles, como tamaños de segmento para la codificación se emplearon 5 valores, los cuales fueron $L = 64$, $L = 128$, $L = 256$, $L = 512$ y $L = 1024$, para cada imagen y como clave secreta se empleó un valor aleatorio. En resumen, por cada algoritmo se crearon 250 imagen con marca de agua, dando un total de 750 imágenes por los tres algoritmos. A cada imagen marcada, se le manipuló en la tasa $\alpha = 0.2$, luego a esa misma imagen se le manipuló en la tasa $\alpha = 0.2625$ y así sucesivamente para las tasas de manipulación $\alpha = 0.325$, $\alpha = 0.3875$, $\alpha = 0.45$, $\alpha = 0.5125$, $\alpha = 0.575$, $\alpha = 0.6375$ y $\alpha = 0.7$, del total de píxeles de cada imagen.

Para medir la capacidad de recuperación de cada uno de los algoritmos, en la Tabla 4.1 se muestran los porcentajes de recuperación de los datos manipulados y en la Tabla 4.2 se muestra el tiempo en segundos que tardo en restaurar las imágenes manipuladas. En ambas tablas, en la primera columna se muestra los valores de L, en la segunda el algoritmo que se empleó y en los títulos de las columnas se muestran las tasas de manipulación α que se probaron.

L	Marca\α	0.2	0.26	0.32	0.39	0.45	0.51	0.57	0.64	0.70
	Zhang	99	86	43	8	0	0	0	0	0
64	TDC1	100	100	100	99	90	64	44	36	30
	TDC2	100	100	100	100	100	100	99	72	12
	Zhang	100	97	49	4	0	0	0	0	0
128	TDC1	100	100	100	100	96	63	43	36	30
	TDC2	100	100	100	100	100	100	100	84	6
	Zhang	100	100	56	0	0	0	0	0	0
256	TDC1	100	100	100	100	99	60	43	36	30
	TDC2	100	100	100	100	100	100	100	94	2
	Zhang	100	100	64	0	0	0	0	0	0
512	TDC1	100	100	100	100	100	58	42	36	30
	TDC2	100	100	100	100	100	100	100	99	0
	Zhang	100	100	72	0	0	0	0	0	0
1024	TDC1	100	100	100	100	100	54	43	36	30
	TDC2	100	100	100	100	100	100	100	100	0

Tabla 4.1: Porcentaje recuperado de los datos manipulados en los 3 algoritmos

L	Marca\α	0.2	0.26	0.32	0.39	0.45	0.51	0.57	0.64	0.70
	Zhang	0.32	0.34	0.3	0.23	0.2	0.2	0.2	0.2	0.19
64	TDC1	0.23	0.25	0.27	0.29	0.28	0.22	0.17	0.17	0.17
	TDC2	0.33	0.38	0.43	0.47	0.5	0.52	0.52	0.45	0.24
	Zhang	0.61	0.72	0.58	0.26	0.22	0.22	0.21	0.21	0.2
128	TDC1	0.39	0.47	0.55	0.61	0.62	0.35	0.18	0.17	0.17
	TDC2	0.65	0.84	1.02	1.19	1.32	1.41	1.42	1.25	0.29
	Zhang	1.6	2.06	1.64	0.3	0.26	0.25	0.24	0.23	0.22
256	TDC1	0.96	1.27	1.57	1.81	1.95	0.73	0.19	0.19	0.19
	TDC2	1.87	2.57	3.28	3.92	4.44	4.78	4.87	4.47	0.32
	Zhang	5.43	7.24	6.08	0.35	0.33	0.31	0.3	0.28	0.27
512	TDC1	3.14	4.42	5.65	6.61	7.2	1.78	0.21	0.21	0.21
	TDC2	6.33	9.14	12.06	14.72	16.9	18.31	17.61	17.61	0.32
	Zhang	19.44	27.11	25.01	0.51	0.48	0.45	0.42	0.39	0.36
1024	TDC1	11.53	16.43	21.1	25.07	27.55	3.91	0.28	0.27	0.26
	TDC2	25.64	37.39	49.1	59.88	68.99	74.25	74.67	69.83	0.37

Tabla 4.2: Tiempo en segundos para la restauración de la imagen en los 3 algoritmos

De los resultados mostrados en la Tabla 4.1, se puede decir que en una marca de agua codificada mediante una formulación como (2.7), sí se aumenta la cantidad de códigos de paridad que genera dicha codificación, entonces, se puede aumentar la tasa de manipulación α en la que se podrá restaurar la imagen de manera correcta. En el caso de Zhang *et al.*, los datos se extraen de los 5 MSB y generan solo la mitad de códigos de paridad, dado que no hay más espacio para almacenar más códigos, obteniendo una tasa máxima de manipulación $\alpha \approx 0.26$, en la que se puede recuperar el 100% de los datos. En los algoritmos TDC1 y TDC2, los 5 MSB se cambiaron por un vector de datos comprimidos, obtenido por el algoritmo de compresión JPEG10. En el algoritmo TDC1 se generaron la misma cantidad de códigos de paridad que de datos, consiguiendo resultados en la tasa de manipulación en la que puede restaurar, de casi el doble que Zhang *et al.*, $\alpha \approx 0.45$. Para el algoritmo TDC2, se consiguió generar el doble de códigos de paridad alcanzando una tasa de manipulación de poco más del doble que Zhang *et al.*, $\alpha \approx 0.64$. En general, en una codificación como (2.7), entre más códigos de parida se tengan, mayor será la tasas de manipulación en las que se podrá recuperar la información.

De los resultados mostrados en la Tabla 4.2, se observa que toma más tiempo para valores grandes de L. También se observa que, en cada una los tiempos de restauración son altos solo cuando se acercan a su límite de recuperación, en el caso de Zhang *et al.* cuando la imagen está manipulada cerca del 26 %, para el algoritmo TDC1 cuando la manipulación está cerca del 45 % y en el caso del algoritmo TDC2, cuando se tiene una manipulación cercana al 64 %.

Para conocer los límites, de los algoritmos TDC1 y TDC2 de manera individual, se realizó una nueva prueba con las mismas 50 imágenes, pero ahora marcando las imágenes con un único valor de $L = 1024$, para ambos algoritmos. Como el límite de manipulación en el algoritmo TDC1 anda alrededor del 45%, se decidió utilizar las tasas de manipulación $\alpha = 0.42$, $\alpha = 0.43$, $\alpha = 0.44$, $\alpha = 0.45$, $\alpha = 0.46$, $\alpha = 0.47$ y $\alpha = 0.48$. Como el límite para el algoritmo TDC2 anda al rededor del 63%, se utilizaron las tasas de manipulación $\alpha = 0.62$, $\alpha = 0.63$, $\alpha = 0.64$, $\alpha = 0.65$, $\alpha = 0.66$, $\alpha = 0.67$ y $\alpha = 0.68$. Los datos que se obtuvieron son: el porcentaje recuperado de los datos manipulados, el PSNR máximo y mínimo en toda la imagen restaurada y el PSNR máximo y mínimo en el área manipulada,

de la imagen restaurada. Los resultados obtenidos se muestran en la Tabla 4.3, donde, el porcentaje de recuperación es el promedio de todas las pruebas y los PSNR son valores máximos o mínimos de las imágenes. En la tabla, T indica que es en toda la imagen y A indica que solo es en el área detectada como manipulada.

Marca	α	0.42	0.43	0.44	0.45	0.46	0.47	0.48
TDC1	% REC	100	100	100	100	100	99.77	97.48
	PSNR MAX T	38.92	38.79	38.65	38.46	38.34	35.79	24.85
	PSNR MIN T	26.52	26.41	26.29	26.19	26.09	23.62	18.71
	PSNR MAX A	35.16	35.12	35.09	35.00	34.97	32.51	21.67
	PSNR MIN A	22.75	22.75	22.72	22.73	22.72	20.34	15.52
Marca	α	0.62	0.63	0.64	0.65	0.66	0.67	0.68
TDC2	% REC	100	100	99.88	96.56	76.20	31.50	4.96
	PSNR MAX T	36.19	36.08	33.86	16.53	8.74	7.96	7.85
	PSNR MIN T	24.99	24.95	18.37	10.88	7.12	6.32	5.92
	PSNR MAX A	34.11	34.07	31.93	14.66	6.94	6.22	6.18
	PSNR MIN A	22.92	22.94	16.43	9.01	5.32	4.58	4.25

Tabla 4.3: Alcances y límites de las marcas de agua TDC1 y TDC2

De los resultados de la Tabla 4.3, se puede decir que para el algoritmo TDC1 la tasa de manipulación máxima en que se pueden recuperar los datos está en $\alpha = 46$ y que para el algoritmo TDC2 está en $\alpha = 0.63$. De toda la tabla, si se consideran únicamente las tasas donde se recuperó el 100% de los datos. Entonces, podemos decir que los algoritmos TDC1 y TDC2, obtienen niveles de calidad en la recuperación en el rango entre 22 y 39 dB para el PSNR.

4.3. Comparativa con el estado del arte

En la Tabla 4.4 se comparan los algoritmos TDC1 y TDC2, con los algoritmos en el estado del arte citado. En la primera columna se muestran los trabajos de marcas de agua, en la segunda columna se muestra la tasa máxima en la que reportan que pudieron recuperar y en la tercera columna se muestra los PSNR que reportan cada trabajo. Con referente al algoritmo TDC1, este apenas supera a [Qin12] que tiene una tasa de manipulación máxima del 27% y está a la par de [Qin17] en la tasa de manipulación, pero en calidad de PSNR, está por debajo de ambos. Con lo que respecta al algoritmo TDC2, conseguimos un esquema de recuperación que solamente está por debajo de [Lee08] que tiene un 85%, [Zhang11b] (esquema con promedios) que tiene 66% y [Zhang15] que tiene 80%. En cuanto a los valores de PSNR obtenidos en la recuperación, estos se encuentran entre los mismos rangos reportados por los demás trabajos.

Marca	Porcentaje	PSNR dB
[Lee08]	85%	>20
[Zhang15]	80%	[22, 44]
[Zhang11b] Promedios	66%	[22, 40]
TDC2	63%	[22, 39]
[Zhang11a]	60%	>27
[Dadkhah14]	55%	>30
[Zhang11c]	54%	[22, 38]
[He09]	51%	No se indica
TDC1	45%	[22, 39]
[Qin17]	45%	[29, 41]
[Qin12]	27%	>40

Tabla 4.4: Comparativa de nuestras propuestas con el estado del arte

En general, de la tabla se puede decir que la calidad que se obtiene en las imágenes restauradas, está dentro de los rangos de calidad que se manejan en el estado del arte. También, en cuanto al porcentaje máximo de manipulación al que se puede restaurar la imagen, podemos decir que estamos en una buena posición comparado con el estado del arte.

El PSNR en imágenes mide que tanto se parecen dos imágenes, donde, el valor del PSNR aumenta si las imágenes son más parecidas entre si, llegando hasta infinito, si las imágenes son iguales. En la Figura 4.19 se muestran dos imágenes restauradas, con el algoritmo de [Zhang11a] para la imagen de la Figura 4.19(a) y con el algoritmo TDC2 para la imagen de la Figura 4.19(b). La imagen de la Figura 4.19(a) tiene un PSNR de 23.75 dB con respecto a la imagen marcada con el algoritmo de [Zhang11a] y la imagen de la Figura 4.19(b) tiene un PSNR de 30.05 dB, con respecto a la imagen marcada con el algoritmo TDC2. Las imágenes manipuladas de donde se restauraron, presentaban un porcentaje de manipulación del 60%. Aunque en el algoritmo de [Zhang11a] se emplean todos los coeficientes de la DCT para la restauración, sin embargo, el PSNR de la imagen restaurada está por debajo de la imagen restaurada por el del algoritmo TDC2, que solo emplea 10 coeficientes de la DCT por cada bloque de 8×8.

(a) ***Restaurada con [Zhang11a]*** **(b)** ***Restaurada con TDC2***

Figura 4.19: Imágenes restauradas con los algoritmos [Zhang11a] y TDC2

4.4. Conclusiones

El algoritmo de compresión JPEG10, demostró ser apto para ser usado en la restauración de imágenes manipuladas, al obtener una buena aproximación de la imagen original, de manera visual y con un nivel de calidad en PSNR, dentro de los rangos que se manejan en el estado del arte.

El utilizar una formulación $c = Ad(2.7)$ en los algoritmos TDC1 y TDC2 para el marcado de la imagen. Permite garantizar la correcta restauración de la imagen manipulada, con tasas de a lo más del 46% para el algoritmo TDC1 y de a lo más del 63%, para el algoritmo TDC2.

Debido a que se está utilizando eliminación Gaussiana como método para recuperar la información original, esto genera que el proceso de restauración sea lento en cuanto a tiempo. Una medida para solventar este problema consistirá en reducir el tamaño del segmento L, sacrificando un poco el porcentaje máximo de recuperación.

Capítulo 5

Conclusiones

5.1. Conclusiones Generales

Se realizaron dos algoritmos para el marcado de imágenes que permiten recuperar la información original, en una imagen marcada cuando ésta presenta tasas de manipulación mayores a $\alpha = 0.26$. Para el algoritmo TDC1 con un tamaño de segmento $L = 1024$ en la codificación $c = Ad(2.7)$, se puede recuperar el 100% de los datos manipulados para una imagen de 512×512, cuando está presenta tasas de manipulación $\alpha \leq 0.46$. Para el algoritmo TDC2, empleando un tamaño de segmento $L = 1024$ en la codificación $c = Ad(2.7)$, se pudo recuperar el 100% de los datos manipulados en tasas de manipulación $\alpha \leq 0.63$.

En cuanto al objetivo general de proponer una marca de agua que permita restauración en imágenes con manipulaciones no mayores a 66%, no se alcanzó de manera práctica. Esto debido a que la recuperación de datos mediante la formulación $c = Ad(2.7)$, está sujeta a que la matriz A sea linealmente independiente.

La codificación (2.7) es una buena herramienta para recuperar de manera exacta la información original en un vector de datos manipulado, sin embargo, cuando se aplica a marcas de agua, ésta se ve limitada por el reducido espacio de almacenamiento que se tiene en la imagen. Para solventar este problema, se hizo uso de una de las herramientas de compresión de imágenes del estándar JPEG.

Continuando con la codificación (2.7), en ésta se tienen dos factores que modifican la tasa de manipulación máxima a la que pueden recuperar los datos manipulados. El primero y más importante es la relación que existe entre el número de códigos de paridad que se generan con respecto al número de datos, ya que este establece la tasa máxima de recuperación teórica. El segundo es el tamaño de segmento L en el que se divide al vector de datos, el cual al incrementarse incrementa también la tasa manipulación máxima, pero en menor medida y sin sobrepasar el límite teórico. Aunque en la práctica, sería una buena opción aumentar el tamaño de L para mejorar los resultados en la restauración, sin embargo, esto no es conveniente debido a que para solucionar el sistema de ecuaciones se utiliza un algoritmo de orden $\mathcal{O}(N^3)$. Con lo que el problema ahora sería que la restauración tomará horas o incluso días para una imagen.

Dado que los algoritmos TDC1 y TDC2, son esquemas de restauración aproximada, entonces, la calidad de la imagen restaurada depende directamente del algoritmo empleado para la aproximación. En nuestro caso, se consiguió una buena calidad en el PSNR ([22, 39] dB), debido a que se tomó como base el algoritmo de compresión del estándar JPEG.

5.2. Trabajos Futuros

1. Considerando que el algoritmo de compresión JPEG10 reduce a 80 bits cada bloque de 8×8, lo cual permite codificarlos con 160 bits (el doble de tamaño). Se propone cambiar el método de compresión basado en la DCT del estándar JPEG, al método de compresión fractal del estándar JPEG. Este cambio se propone debido a que como se menciona en [Zhang15], la compresión fractal permite almacenar hasta 3 copias de la marca de agua en los 3 LSB. Con lo que se espera que los datos comprimidos se codifiquen con un factor de compresión $k = 1/3$, lo cual derivará en el aumento del porcentaje máximo permisible, hasta un $\alpha = 0.75$.

2. Dado que los algoritmos propuestos, tienen un cuello de botella en el proceso de recuperación de información causado por eliminación Gaussiana. Se propone probar algún otro método de recuperación de datos, que permita obtener tiempos de recuperación más cortos, tales como en el caso de [Bravo-Solorio18].

Referencias

[Ai06] Ai, D., Chang, Y., Luo, Z., y Wang, J. Practical algorithms for tornado codes. *Journal of Electronics (China)*, 23(2):274–276, 2006.

[Bravo-Solorio18] Bravo-Solorio, S., Calderon, F., Li, C.-T., y Nandi, A. K. Fast fragile watermark embedding and iterative mechanism with high self-restoration performance. *Digital Signal Processing*, 73:83–92, 2018.

[Byers02] Byers, J. W., Luby, M., y Mitzenmacher, M. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1528–1540, Oct 2002. ISSN 0733-8716. doi:10.1109/JSAC.2002.803996.

[Dadkhah14] Dadkhah, S., Manaf, A. A., Hori, Y., Hassanien, A. E., y Sadeghi, S. An effective svd-based image tampering detection and self-recovery using active watermarking. *Signal Processing: Image Communication*, 29(10):1197 – 1210, 2014. ISSN 0923-5965. doi: https://doi.org/10.1016/j.image.2014.09.001.

[Fridrich99] Fridrich, J. y Goljan, M. Images with self-correcting capabilities. *En Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, tomo 3, págs. 792–796 vol.3. Oct 1999. doi: 10.1109/ICIP.1999.817228.

[Gallager62] Gallager, R. Low-density parity-check codes. *IRE Transactions on*

Information Theory, 8(1):21–28, January 1962. ISSN 0096-1000. doi: 10.1109/TIT.1962.1057683.

[Hamming50] Hamming, R. W. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950. ISSN 0005-8580. doi:10.1002/j.1538-7305.1950.tb00463.x.

[He09] He, H.-J., Zhang, J.-S., y Chen, F. Adjacent-block based statistical detection method for self-embedding watermarking techniques. *Signal Processing*, 89(8):1557 – 1566, 2009. ISSN 0165-1684. doi: https://doi.org/10.1016/j.sigpro.2009.02.009.

[La Serna09] La Serna, N., Concepción, L. P., y Durán, C. Y. Compresión de imágenes: Fundamentos, técnicas y formatos. *Revista de investigación de Sistemas e Informática*, 6(1):21–29, 2009.

[Lee08] Lee, T.-Y. y Lin, S. D. Dual watermark for image tamper detection and recovery. *Pattern Recognition*, 41(11):3497 – 3506, 2008. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2008.05.003.

[Li07] Li, X. y Cai, J. Robust transmission of jpeg2000 encoded images over packet loss channels. *En 2007 IEEE International Conference on Multimedia and Expo*, págs. 947–950. July 2007. ISSN 1945-7871. doi: 10.1109/ICME.2007.4284808.

[Liu02] Liu, T. y Qiu, Z.-d. The survey of digital watermarking-based image authentication techniques. *En Signal Processing, 2002 6th International Conference on*, tomo 2, págs. 1556–1559. IEEE, 2002.

[Lu01] Lu, C.-S. y Liao, H.-Y. Multipurpose watermarking for image authentication and protection. *IEEE transactions on image processing*, 10(10):1579–1592, 2001.

[Luby97] Luby, M. G., Mitzenmacher, M., Shokrollahi, M. A., Spielman, D. A., y Stemann, V. Practical loss-resilient codes. *En Proceedings of the twenty-*

ninth annual ACM symposium on Theory of computing, págs. 150–159. ACM, 1997.

[M. Vargas16] M. Vargas, L., Vera de Payer, E., y Di Gionantonio, A. Marcas de agua: una contribución a la seguridad de archivos digitales. *FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES*, 3(1):49–54, March 2016.

[Martínez Leal15] Martínez Leal, L. Marcas de agua en el papel. *En* UNAM, ed., *El giro visual en bibliotecología: prácticas cognoscitivas de la imagen*, cap. Vías del conocimiento hacia la imagen, págs. 39–51. Universidad Nacional Autónoma de México, 1ª ed$^{\text{ón}}$., 2015.

[Orúe12] Orúe, A. B. Marcas de agua en el mundo real. 2002-03-12. URL `http://hdl.handle.net/10261/8864`

[Qin12] Qin, C., Chang, C.-C., y Chen, P.-Y. Self-embedding fragile watermarking with restoration capability based on adaptive bit allocation mechanism. *Signal Processing*, 92(4):1137 – 1150, 2012. ISSN 0165-1684. doi: https://doi.org/10.1016/j.sigpro.2011.11.013.

[Qin16] Qin, C., Wang, H., Zhang, X., y Sun, X. Self-embedding fragile watermarking based on reference-data interleaving and adaptive selection of embedding mode. *Information Sciences*, 373:233 – 250, 2016. ISSN 0020-0255. doi:https://doi.org/10.1016/j.ins.2016.09.001.

[Qin17] Qin, C., Ji, P., Zhang, X., Dong, J., y Wang, J. Fragile image watermarking with pixel-wise recovery based on overlapping embedding strategy. *Signal Processing*, 138:280 – 293, 2017. ISSN 0165-1684. doi: https://doi.org/10.1016/j.sigpro.2017.03.033.

[Stevens17] Stevens, M., Bursztein, E., Karpman, P., Albertini, A., y Markov, Y. The first collision for full sha-1. *En* J. Katz y H. Shacham, eds., *Advances*

in Cryptology – CRYPTO 2017, págs. 570–596. Springer International Publishing, Cham, 2017. ISBN 978-3-319-63688-7.

[Tian14] Tian, C. A c library of repair-efficient erasure codes for distributed data storage systems. *En 2014 IEEE International Conference on Big Data (Big Data)*, págs. 21–26. Oct 2014. doi:10.1109/BigData.2014.7004379.

[Tuba14] Tuba, M. y Bacanin, N. Jpeg quantization tables selection by the firefly algorithm. *En 2014 International Conference on Multimedia Computing and Systems (ICMCS)*, págs. 153–158. April 2014. doi: 10.1109/ICMCS.2014.6911315.

[Wallace92] Wallace, G. K. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, Feb 1992. ISSN 0098-3063. doi:10.1109/30.125072.

[Yadav14] Yadav, U., Sharma, J. P., Sharma, D. C., y Sharma, P. K. Different watermarking techniques & its applications: A review. 2014.

[Zhang08] Zhang, X. y Wang, S. Fragile watermarking with error-free restoration capability. *IEEE Transactions on Multimedia*, 10(8):1490–1499, Dec 2008. ISSN 1520-9210. doi:10.1109/TMM.2008.2007334.

[Zhang11a] Zhang, X., Qian, Z., Ren, Y., y Feng, G. Watermarking with flexible self-recovery quality based on compressive sensing and compositive reconstruction. *IEEE Transactions on Information Forensics and Security*, 6(4):1223–1232, Dec 2011. ISSN 1556-6013. doi: 10.1109/TIFS.2011.2159208.

[Zhang11b] Zhang, X., Wang, S., Qian, Z., y Feng, G. Reference sharing mechanism for watermark self-embedding. *IEEE Transactions on Image Processing*, 20(2):485–495, 2011.

[Zhang11c] Zhang, X., Wang, S., Qian, Z., y Feng, G. Self-embedding watermark with

flexible restoration quality. *Multimedia Tools and Applications*, 54(2):385–395, 2011.

[Zhang15] Zhang, X., Xiao, Y., y Zhao, Z. Self-embedding fragile watermarking based on dct and fast fractal coding. *Multimedia Tools and Applications*, 74(15):5767–5786, 2015.