UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

**INGENIERO TÉCNICO DE TELECOMUNICACIONES,
ESPECIALIDAD EN SISTEMAS DE TELECOMUNICACIÓN**

Compresor Lower Tree Wavelet (LTW) sobre Android 5.0

PROYECTO FIN DE CARRERA

Septiembre - 2015

AUTOR: María Martínez Ferrández

DIRECTOR/ES: Otoniel M. López Granado

Miguel O. Martínez Rach

Escuela Politécnica Superior de Elche
Universidad Miguel Hernández

ACTA DE EVALUACIÓN PROYECTO FIN DE CARRERA

Título proyecto:
COMPRESOR LOWER TREE WAVELET (LTW) SOBRE ANDROID 5.0

Proyectante:
MARÍA MARTÍNEZ FERRÁNDEZ

Director/es:
MIGUEL O. MARTÍNEZ RACH Y OTONIEL M. LÓPEZ GRANADO

CALIFICACIÓN NUMÉRICA ☐ **MATRÍCULA DE HONOR** ☐

Méritos justificativos en el caso de conceder Matrícula de Honor.

Conforme presidente:	Conforme secretario:	Conforme vocal:
Fdo.:	Fdo.:	Fdo.:

Lugar y Fecha: _____

Resumen

En este proyecto se pretende implementar un codificador de imágenes basado en Lower Tree Wavelets (LTW) sobre sistema operativo Android, usando las características de captura de imágenes RAW (crudas) de la versión 5.0. Actualmente existen muchos codificadores de imagen como por ejemplo JPEG, los cuales pueden ejecutarse en equipos con recursos limitados como lo son los dispositivos móviles. Sin embargo, estos dispositivos no poseen las mismas prestaciones que un computador, y ejecutar estos codificadores podría conllevar un gran consumo de memoria y procesado. Por ello es necesario el desarrollo de codificadores que reduzcan dicho consumo en sistemas operativos móviles, como sería el caso del codificador LTW. Se ha escogido la tecnología Android ya que a día de hoy su cuota de mercado es superior al 70 %.

Como primer paso en este proyecto, realizaremos un estudio exhaustivo del codificador Lower Tree Wavelets con el fin de realizar una librería Java acorde al mismo. El uso de dicho lenguaje radica en la pretensión de obtener el mejor soporte de implementación en el Smartphone o Tablet, ya que se trataría del lenguaje que la gente de Google ha escogido para dar soporte a Android. Con la librería Java desarrollada, se implementará la aplicación Android que realizará la compresión de imágenes haciendo uso de dicha librería. Finalmente, se estudiará el rendimiento del codificador LTW y se hará la comparación de tasas de compresión y calidad con uno de los codificadores de imágenes más extendidos, JPEG.

Palabras clave: Android, Lower Tree Wavelets, JPEG, codificador, compresión, imagen.

Índice general

Índice de figuras

Capítulo 1

Introducción

Contents

1.1. Fundamentos de la codificación de imágenes

El principal inconveniente de las imágenes digitales es, sin duda, la cantidad de información que requiere y el tamaño de los archivos que genera. La capacidad de trabajo y de almacenamiento van en aumento, pero también nos vamos habituando a disponer de mayor resolución. De ahí que se mantenga el uso y el interés por la evolución de las técnicas de compresión.

La aparición de los actuales smartphones ha incrementado el volumen de imágenes capturadas. Esto junto a la menor capacidad de almacenamiento de dichos dispositivos hace necesario optimizar el proceso de compresión de imágenes. Por ello antes de almacenar o transmitir dicha imagen, ésta debe ser comprimida.

En cualquier caso, y volviendo a la Teoría de la Codificación, debemos señalar que el principal objetivo de la compresión, en cualquiera de sus formas, es la eliminación de la redundancia, concepto que definiremos a continuación. Partiendo de esta idea básica, estudiaremos las principales clases de algoritmos de compresión: compresión sin pérdida de información (lossless) y compresión con pérdida de información (lossy).

Lo que se propone en este primer capítulo es dar una visión global de las técnicas de compresión más importantes de la actualidad y del pasado.

1.2. Algoritmos de compresión

La compresión de datos se basa fundamentalmente en buscar repeticiones en series de datos para después almacenar solo el dato junto al número de veces que se repite. Así, por ejemplo, si en un fichero aparece una secuencia como .ªAAAAA", ocupando 6 bytes se podría almacenar simplemente "6A"que ocupa solo 2 bytes. En realidad, el proceso es mucho más complejo, ya que raramente se consigue encontrar patrones de repetición tan exactos.

El objetivo de la compresión es, por tanto, reducir siempre el tamaño de la información, intentando siempre que esta reducción de tamaño no afecte al contenido. No obstante, aunque en multitud de aplicaciones sólo es aceptable la compresión desde el punto de vista de la reducción de datos sin error (imágenes médicas, satélite, documentos legales?), existen otras en las que pequeñas pérdidas de información pueden llevar a una gran eficiencia de compresión (internet, multimedia, FAX, ...).

Distinguiremos así dos técnicas de compresión: compresión sin pérdida y compresión con pérdidas.

Pero antes de profundizar en estas técincas de compresión, hay que tener presentes dos conceptos: redundancia y entropía.

1.2.1. Definición de redundancia

La redundancia es el exceso de información transmitida por unidad de datos. Precisamente una de las aplicaciones de la Teoría de la Información es la compresión de datos, que simplemente trata de eliminar la información superflua o repetitiva dentro de un archivo de forma que se reduzca el flujo binario sin perder la información necesaria para recuperar la secuencia.

La información redundante que suelen presentar las imágenes es del tipo espacial. Ésta viene asociada al hecho de que la naturaleza está llena de objetos sólidos con superficies y texturas uniformes; los decorados, los paisajes, e incluso los rostros no varían significativamente la información de píxel a píxel, sino que encontraremos generalmente grandes superficies sin variación.

El hecho de que varios píxeles adyacentes sean prácticamente iguales nos va a permitir, en vez de transmitirlos todos o almacenarlos todos, transmitir un píxel representativo del conjunto, y las diferencias de cada uno respecto a éste. Dichas diferencias, por ser general-

mente pequeñas, pueden codificarse con menos bits.

1.2.2. Definición de entropía

En el ámbito de la teoría de la información, la entropía mide la incertidumbre de una fuente de información. La entropía también se puede considerar como la cantidad de información promedio que contienen los símbolos usados, siendo los símbolos con menor probabilidad los que aportan mayor información.

La entropía nos indica el límite teórico para la compresión de datos ya que denota el mínimo número de bits por símbolo necesarios para representar una cadena. Su cálculo se realiza mediante la siguiente fórmula:

$$H = \sum_{k=1}^{m} p_k log \frac{1}{p_k}$$

donde H es la entropía, p son las probabilidades de que aparezcan los diferentes símbolos y m el número total de símbolos.

Si se utiliza el logaritmo en base 2, la entropía se mide en bits.

Por ejemplo: Si se transmiten mensajes que están formados por combinaciones aleatorias de las 26 letras del alfabeto inglés, el espacio en blanco y cinco signos de puntuación, la entropía sería:

$$\begin{aligned} H &= \frac{1}{32} log_2 32 + \frac{1}{32} log_2 32 + \ldots + \frac{1}{32} log_2 32 \\ &= \left(\frac{1}{32} + \frac{1}{32} + \cdots + \frac{1}{32}\right) log_2 32 = log_2 32 = 5 bits \end{aligned}$$

Esto significa que se necesitan 5 bits para codificar cada carácter o mensaje: 00000, 00001, 00010, 11111. Una transmisión y almacenamiento eficiente de la información exige la reducción del número de bits utilizados en su codificación.

1.3. Algoritmos de compresión sin pérdida

La compresión sin pérdidas es aquella que permite recuperar exactamente la calidad original de la imagen. La compresión sin pérdida de datos, es utilizada para comprimir archivos o información que contienen datos que no pueden ser degradados o perdidos, como pueden ser documentos de texto, archivos ejecutables, etc. Se distingue entre sistemas adaptativos, no adaptativos y semiadaptativos, según tengan en cuenta o no las características del archivo a comprimir.

- Los **no adaptativos** (se les relaciona con el código Huffman que explicaremos en detalle a continuación) establecen a priori una tabla de códigos con las combinaciones de bits que más se repiten estadísticamente. A estas secuencias se asignan códigos cortos, y a otras menos probables claves más largas. El problema que presentan es que un diccionario de claves único tiene resultados muy diferentes en distintos originales

- Un sistema es **semiadaptativo** si se analiza primero la cadena de datos a comprimir y se crea una tabla a medida. Se logra mayor compresión, pero introduce dos inconvenientes: la pérdida de velocidad al tener que leer el original dos veces, por un lado, y la necesidad de incrustar en el archivo comprimido el índice de claves, por el otro.

- Entre los **métodos adaptativos**,el más simple es el Run Lengh Encode (cuyas siglas significan Ejecutar Longitud Codificar) o RLE, que consiste en sustituir series de valores repetidos por una clave con indicador numérico.

Los compresores de uso general más populares utilizan métodos como éste, por eso tardan más en empaquetar los datos que en descomprimirlos.

1.3.1. Codificación Huffman

El algoritmo de codificación Huffman fue creado por David A. Huffmann en 1952. Como hemos dicho ya anteriormente, este algoritmo de codificación es de tipo no adaptativo y se trata de un ejemplo de codificación basado en la entropía.

David Huffman propuso un método estadístico que permite asignar un código binario a diversos símbolos a comprimir, tales como píxeles o caracteres, por ejemplo.

Este algoritmo toma un alfabeto de n símbolos, junto con sus frecuencias de aparición asociadas y produce un código de Huffman para ese alfabeto y esas frecuencias.

Para comprimir cada símbolo de la cadena, simplemente debemos usar el código que se ha calculado mediante Huffman. La forma de conseguir esta asignación óptima es representar los símbolos con códigos cuya longitud es inversamente proporcional a la probabilidad del símbolo. De esta forma, los símbolos menos probables se representan con códigos más largos, y los más probables con códigos más cortos.

El proceso de asignación de códigos se lleva a cabo mediante la construcción de un árbol binario, desde las hojas hacia la raíz, de manera que los nodos hoja son los símbolos del alfabeto. En la construcción del árbol, los nodos menos probables se unen sucesivamente para formar otro nodo de mayor probabilidad, de forma que cada uno de los enlaces añade un bit al código de los símbolos que estamos juntando. Este proceso termina cuando sólo se dispone de un nodo, de forma que éste representa la raíz del árbol.

A partir de este árbol podemos conocer el código asociado a un símbolo, así como obtener el símbolo asociado a un determinado código.

Supongamos que se tiene la oración: `COMMENT_CA_MARHE`; cuya frecuencia de aparición por carácter es la que se muestra en la figura 1.1:

M	A	C	E	_	H	O	N	T	R
3	2	2	2	2	1	1	1	1	1

Figura 1.1: *Frecuencia de aparición de las letras del ejemplo citado para explicación del código Huffman.*

En este ejemplo, su árbol de Huffman correspondiente sería el que vemos en la figura 1.2:

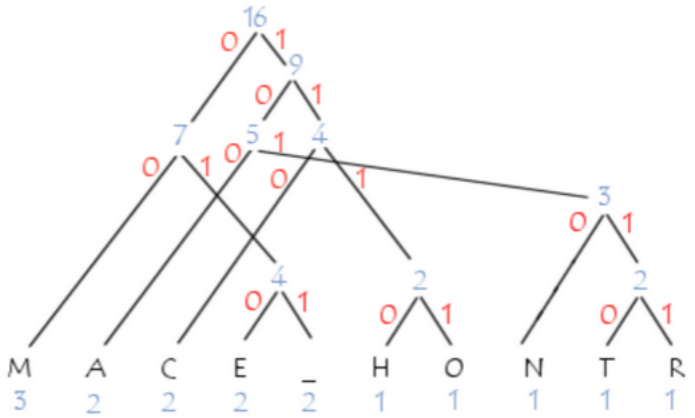


Figura 1.2: *Árbol correspondiente al ejemplo del Figura 1.1.*

Los códigos correspondientes a cada carácter son tales que los códigos para los caracteres más frecuentes son cortos y los correspondientes a los símbolos menos frecuentes son largos. Esto se puede apreciar mejor si se ve la figura 1.3:

M	A	C	E	_	H	O	N	T	R
00	100	110	010	011	1110	1111	1010	10110	10111

Figura 1.3: *Paso final del código Huffman.*

Esto significa que para la utilización del algoritmo de Huffman es necesario conocer de antemano las frecuencias de aparición de cada símbolo. Su eficiencia dependerá de lo próximas a las frecuencias reales que sean las estimadas. Ésta depende también del balance que exista entre los hijos de cada nodo del árbol, siendo más eficiente conforme menor sea la diferencia de frecuencias entre los dos hijos de cada nodo.

Una vez ha reconstruido el árbol, el decodificador, puede decodificar la cadena original fácilmente. Para ello, debe recorrer el árbol desde la raíz hacia las hojas, usando los bits de la cadena codificada para avanzar en el recorrido hacia las hojas.

Este código puede aplicarse también de modo semiadaptativo, si se analiza primero la cadena de datos a comprimir y se crea una tabla a medida. Se logra mayor compresión,

pero introduce dos inconvenientes: la pérdida de velocidad al tener que leer el original dos veces, por un lado, y la necesidad de incrustar en el archivo comprimido el índice de claves, por el otro.

Para el caso de compresión de imágenes el algoritmo se utiliza de la misma manera excepto que en lugar de caracteres nos referimos a niveles de gris. Dada la distribución estadística de los niveles de gris, el algoritmo de Huffman genera un código lo más próximo posible al límite teórico mínimo (entropía).

1.4. Algoritmos de compresión con pérdida

Se refiere a técnicas de compresión de datos donde cierta cantidad de datos es desechada con el objetivo de reducir su tamaño original. Estas técnicas de compresión eliminan información redundante o innecesaria.

La compresión con pérdida produce así una relación de compresión mucho mayor que la compresión sin pérdida. Sin embargo, con esta técnica de compresión, una vez realizada la misma, no se puede obtener la señal original, aunque sí una aproximación cuya semejanza con la original dependerá de la técnica empleada.

Esta técnica de compresión se da principalmente en imágenes, vídeos y sonidos, aprovechando así las limitaciones de percepción de los sentidos humanos.

A continuación, tas estudiar las etapas de un compresor genérico con pérdidas, se van a estudiar, como preámbulo a LTW, dos métodos para reducir la redundancia espacial de las imágenes. Para ello, se transforma la imagen a otro dominio, en el cual sólo unos pocos de los coeficientes contienen la mayor parte de la información, y los otros coeficientes tienen valores despreciables. En el nuevo dominio, la imagen tendrá una representación mucho más compacta, y podrá ser representada básicamente por unos pocos coeficientes de la transformada.

1.4.1. Etapas de un compresor genérico con pérdidas

Antes de profundizar en los algoritmos de compresión con pérdidas en los que se centra este proyecto, se va a realizar una breve introducción a los sistemas de compresión.

La mayoría de los codificadores de imagen consisten en transformación, cuantificación y codificación entrópica. Esto podemos verlo en la figura 1.4. En dicha figura, T y T^{-1} son la transformada y la inversa de la transformada. Q y Q^{-1} representan la cuantificación y su proceso inverso, respectivamente. El conjunto original de píxeles está representado por P.

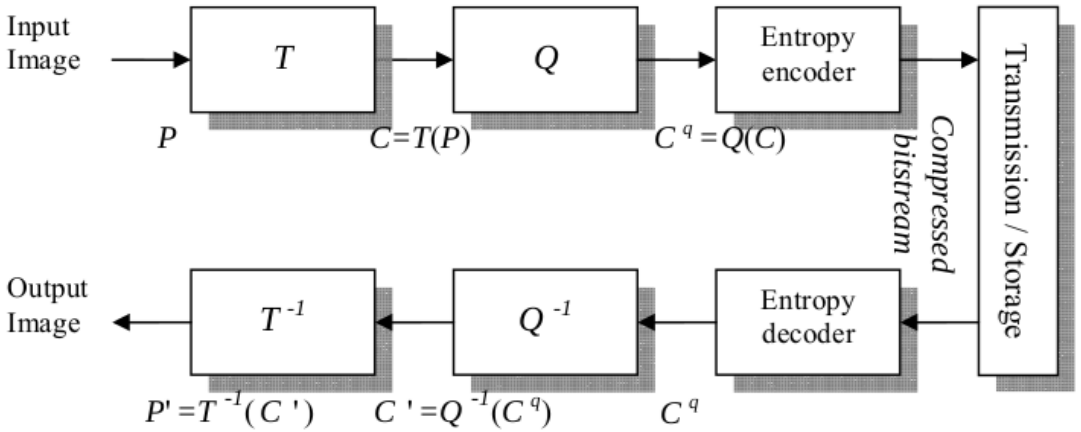


Figura 1.4: *Diagrama de bloques de un codificador y decodificador de imagen.*

El bloque de la transformada es por norma general una operación reversible. Esto quiere decir que $T \cdot T^{-1} \cdot (constante) = T^{-1} \cdot T \cdot (constante) = (constante)$.

Por otra parte, la cuantificación es una técnica de compresión habitualmente con pérdida que agrupa un rango de valores a un único valor. Si el número de símbolos discretos en un flujo dado se reduce, dicho flujo se vuelve más comprensible. Esto quiere decir que $Q \cdot Q^{-1} \cdot (constante) \neq (constante)$.

El codificador entrópico es el bloque responsable de la compresión. Mapea los datos recibidos y transforma los datos más repetidos en palabras símbolo cortas y los menos repetidos en palabras símbolo más largas. Así, los que más se repiten necesitarán menos bits. La ventaja de esto es que son algoritmos reversibles y sabiendo el símbolo que ha llegado, se la información que contiene.

La parte más compleja y computacionalmente costosa de un sistema de compresión es el codificador entrópico.

1.4.1.1. Transformada Discreta del Coseno (DCT)

La transformada discreta del coseno (DCT), también denominada transformada del coseno, es la más ampliamente utilizada en la compresión de imágenes. Esta transformada cuenta con una buena propiedad de compactación de energía, que produce coeficientes incorrelados. La decorrelación de coeficientes es muy importante para la compresión, ya que, el posterior tratamiento de cada coeficiente se puede realizar de forma independiente, sin pérdida de eficiencia de compresión.

La DCT (al igual que las demás transformadas discretas) puede aplicarse a codificación de imagen, desde un punto de vista de reducción de ancho de banda o compresión de datos. EL objetivo es conseguir que una imagen (dominio espacial), se traslade a un dominio transformado con el objetivo de reducir el ancho de banda para la transmisión o los requerimientos para el almacenamiento; de tal forma que la subsiguiente recuperación de la imagen mediante la transformada inversa, no presente distorsión perceptible.

El concepto de la DCT se basa sintéticamente en tomar cada píxel de un bloque de 8x8 píxel como se puede ver en la figura 1.5. Ese píxel es una ?Muestra? (sample) de una señal variable en el tiempo, proporcional a la luminancia, y de otra señal variable en el tiempo, proporcional a la cromancia.

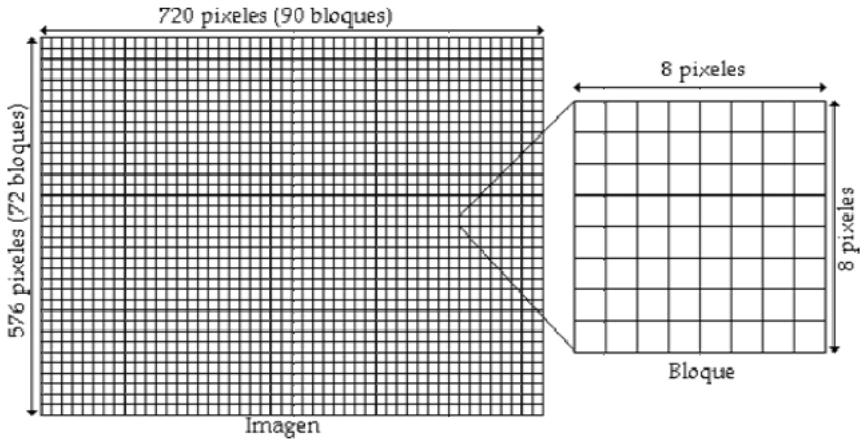


Figura 1.5: *División en bloques o subimágenes de 8x8 píxeles.*

Estas dos señales son las que se pasarán, separadamente, al dominio de las frecuencias, obteniéndose los coeficientes en frecuencia y transmitiéndolos, en lugar de transmitir las funciones del tiempo.

Una vez realizada la DCT, para su decodificación se utiliza la transformada inversa.

Supongamos que a la entrada del codificador, la imagen original de agrupa en bloques de 8x8, se traslada de enteros sin signo en el rango $[0, 2^{p-1}]$ a enteros con signo con rango $[-2^{p-1}, 2^{p-1} - 1]$ y se le da como la entrada de la transformada DCT. En la salida del codificador, la transformada DCT inversa devuelve los bloques 8x8 para formar la imagen reconstruida.

Las siguientes ecuaciones son las definiciones matemáticas idealizadas de la DCT 8x8 (1.1) y su inversa 8x8 (1.2):

$$F(u,v) = \frac{C(u)}{2}\frac{C(v)}{2}\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) cos\frac{(2x+1)u\pi}{16} cos\frac{(2x+1)v\pi}{16}\right] \quad (1.1)$$

$$f(x,y) = \sum_{u=0}^{7}\frac{C(u)}{2}\sum_{v=0}^{7}\frac{C(v)}{2}F(u,v)\cos\frac{(2x+1)u\pi}{16} cos\frac{(2x+1)v\pi}{16} \quad (1.2)$$

1: F es el valor del coeficiente de la linea 'u' y la columna 'v' (dominio frecuencia), recordemos que 'u' y 'v' varían de 0 a 7.

2: Estas constantes valen 0.707 ($\frac{1}{\sqrt{2}}$) para cuando 'u' y/o 'v' valen 0, y valen 1 en cualquier otro lugar del tablero.

3: Sumatorio de todos los términos de 'x' entre 0 y 7.

4: Sumatorio de todos los términos de 'y' entre 0 y 7.

5: Valor del píxel en una columna y línea dadas (dominio de tiempo).

6: Generador de funciones vertical, produce la forma de onda cosenoidal vertical: 'x' es el numero de la columna de los píxeles y 'u' es la frecuencia. La unidad es en radianes.

7: Generador de funciones horizontal, produce la forma de onda cosenoidal horizontal: 'y' es el numero de la línea de los píxeles y 'v' es la frecuencia.

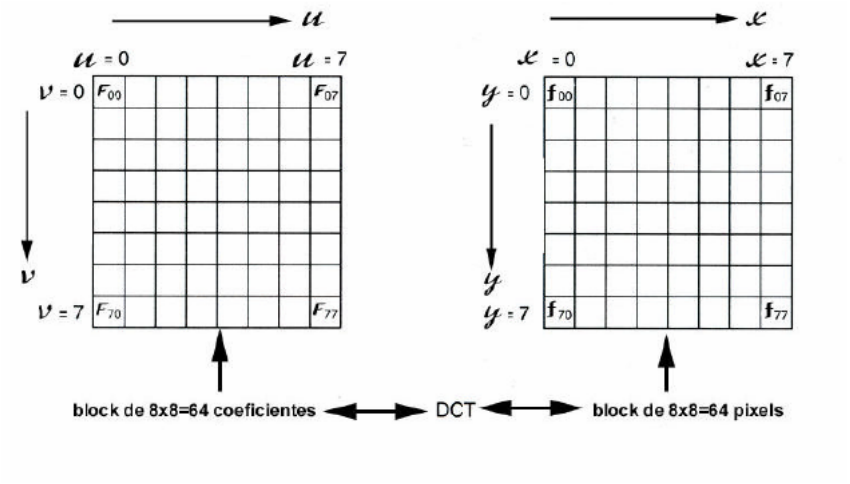


Figura 1.6: *Representación gráfica de las definiciones matemáticas de la DCT.*

Con esto, un bloque de píxeles es transformado a un bloque de coeficientes de frecuencias como se puede ver gráficamente en la figura 1.7.

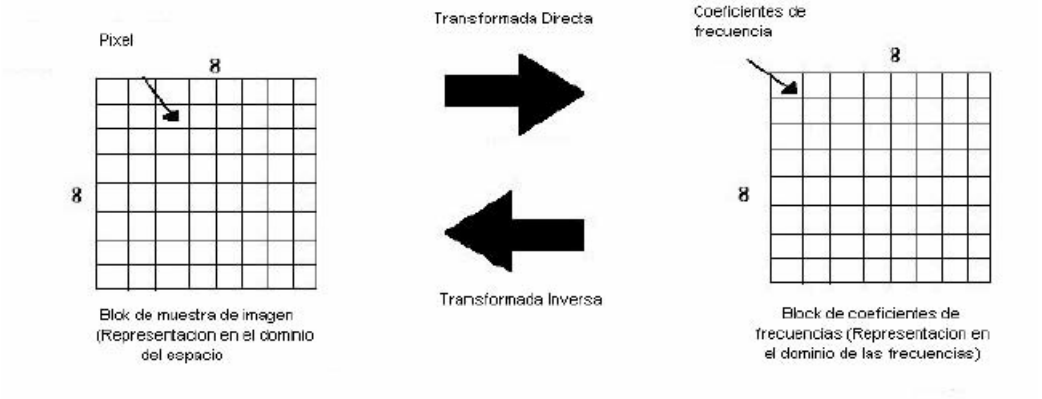


Figura 1.7: *Representación gráfica de la DCT y su inversa.*

La DCT toma cada bloque 8x8 y lo descompone en 64 señales de una base ortogonal. Cada una de estas señales contiene una de las 64 frecuencias espaciales únicas que comprenden el espectro de la señal de entrada. La salida de la DCT es un conjunto de 64 amplitudes de señal de la base o coeficientes cuyos valores son únicamente determinados por una señal de 64 puntos de entrada en particular.

Ya que los valores de la imagen varían lentamente entre los píxel consecutivos, el pro-

ceso de la DCT realiza la compresión de los datos que concentran la mayor parte de la señal en las frecuencias espaciales bajas. Para un bloque típico de 8x8 de una imagen típica, la mayoría de frecuencias espaciales tienen una amplitud cero o casi cero y no serán codificadas.

En el decodificador se realiza el proceso inverso. Toma 64 coeficientes DCT (que para entonces ya han sido cuantificados) y reconstruye una imagen de 64 puntos de salida sumando las señales base. Si la DCT y su transformada pudieran ser calculadas con precisión absoluta y los coeficientes de la DCT no se cuantificaran se podría reconstruir perfectamente la imagen original.

1.4.1.2. Transformada Wavelet Discreta (DWT)

Dado que este proyecto se centra en la utilización de Wavelets para la compresión de imágenes, se hará una explicación más en profundidad de su funcionamiento.

La Transformada Wavelet (WT) permite conocer qué frecuencias componen una señal en cada instante de tiempo ya que permite el análisis dentro de intervalos grandes de tiempo en aquellos segmentos en los que se requiere mayor precisión en baja frecuencia, y regiones más pequeñas donde se requiere información en alta frecuencia.

Para muchas señales la información más importante se encuentra en las frecuencias bajas, mientras que en las altas frecuencias se encuentran los detalles o matices de la señal. Por ejemplo, en el caso de la voz humana, si eliminamos los componentes con altas frecuencias, la voz suena diferente pero se sigue entendiendo su mensaje. En cambio, si lo que se elimina son las componentes de bajas frecuencias, el mensaje se vuelve irreconocible. Por eso el análisis Wavelet permite descomponer la señal en aproximaciones y detalles, a éste proceso se le conoce con el nombre de análisis.

En el análisis de imágenes por medio de la transformada Wavelet es necesario hacer una discretización de la misma surgiendo por esto el método DWT (Discreet Wavelet Transform ó Transformada Wavelet Discreta) que es una técnica que permite el procesamiento digital de señales discretas e imágenes.

La Transformada Wavelet Discreta está recibiendo mucha atención en el campo del procesamiento de imágenes debido a su flexibilidad en la representación de señales no estacionarias y a su adecuación por adaptarse a las características del sistema de visión humano. Su aplicación realiza fielmente un análisis multiresolución y una descomposición subbanda, por lo cual su utilización en los últimos 10 años está siendo fundamental en el

procesamiento digital de imágenes y en todas sus disciplinas.

En el proceso de análisis Wavelet, las señales son representadas utilizando un grupo de funciones básicas producidas por el desplazamiento y el escalado de una función madre o función principal. La transformada Wavelet es una descomposición de una señal en frecuencias.

La Transformada Wavelet Discreta unidimensional descompone recursivamente la señal de entrada, $S_0(n)$, en una parte de detalle y otra de promedio en cada iteración. Sea $S_i(n)$ y $W_i(n)$ el promedio y el detalle respectivamente, y sea i el nivel de aplicación de la transformada. La aproximación de la señal en el nivel i+1 se calcula usando

$$S_{i+1}(n) = \sum_k g(k) S_i(2n-k) \tag{1.3}$$

Y el detalle de la señal en el nivel i+1 se calcula usando

$$W_{i+1}(n) = \sum_k h(k) S_i(2n-k) \tag{1.4}$$

Las ecuaciones (1.3) y (1.4) describen la computación de la DWT. En la siguiente figura se muestran los tres niveles de cálculo de la DWT 1-D.

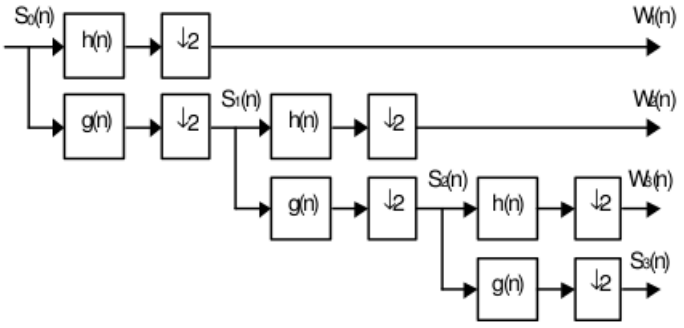


Figura 1.8: *Diagrama de bloques del banco de filtros de la DWT.*

La DWT bidimensional trabaja sobre una señal 2-D como puede ser una imagen, que es el caso que nos ocupa. Tal como los filtros 1-D se utilizan para computar la DWT 1-D, para computar la DWT 2-D utilizamos filtros 2-D calculados a partir de la convolución de los filtros 1-D sobre ellos mismos. Estos filtros 2-D pueden ser separables o no separables, donde un filtro 2-D $f(n_1,n_2)$ es separable si es expresable como $f(n_1,n_2) = f_1(n_1)f_2(n_2)$.

La DWT 2-D separable descompone una imagen $S_i(n_1,n_2)$ en una imagen promedio y tres imágenes detalles, de acuerdo con las expresiones

$$S_{i+1}(n_1,n_2) = \sum_{k1}\sum_{k2} g(k_1)g(k_2)S_i(2n_1-k_1,2n_2-k_2) \tag{1.5}$$

$$W^1_{i+1}(n_1,n_2) = \sum_{k1}\sum_{k2} g(k_1)h(k_2)S_i(2n_1-k_1,2n_2-k_2) \tag{1.6}$$

$$W^2_{i+1}(n_1,n_2) = \sum_{k1}\sum_{k2} h(k_1)g(k_2)S_i(2n_1-k_1,2n_2-k_2) \tag{1.7}$$

$$W^3_{i+1}(n_1,n_2) = \sum_{k1}\sum_{k2} h(k_1)h(k_2)S_i(2n_1-k_1,2n_2-k_2) \tag{1.8}$$

donde H(z) y G(z) son los filtros Wavelet 1-D. La señal $S_{i+1}(n_1,n_2)$ es un suavizado de baja resolución de la imagen $S_i(n_1,n_2)$. Este suavizado se calcula desde $S_i(n_1,n_2)$ mediante un filtro paso-bajo y diezmado por 2 a lo largo de filas y columnas. Las señales $W^1_{i+1}(n_1,n_2)$, $W^2_{i+1}(n_1,n_2)$ y $W^3_{i+1}(n_1,n_2)$ contienen el detalle de $S_i(n_1,n_2)$. El nivel 1 de la DWT 2-D separable se muestra en la figura 1.9.

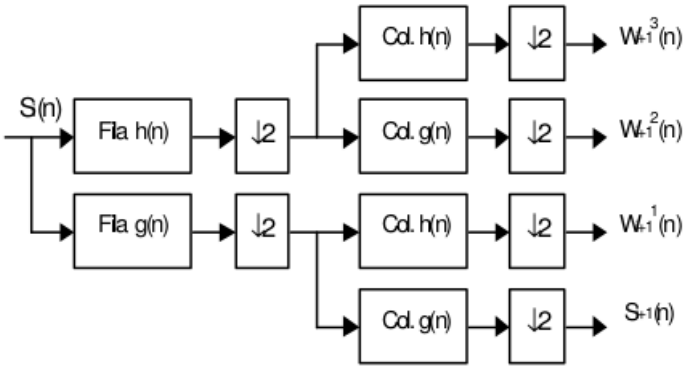


Figura 1.9: *Diagrama de bloques del banco de filtros de análisis utilizado para calcular la DWT 2-D separable.*

En contraste con los filtros separables, los filtros no separables descomponen directamente una imagen en cuatro subimágenes resultado de la aplicación de la DWT por filas y posteriormente por columnas. La figura 1.10 muestra la arquitectura de la DWT no separable con dos niveles de resolución. Notar que cada bloque en esta arquitectura es un filtro

2-D submuestreado por 2.

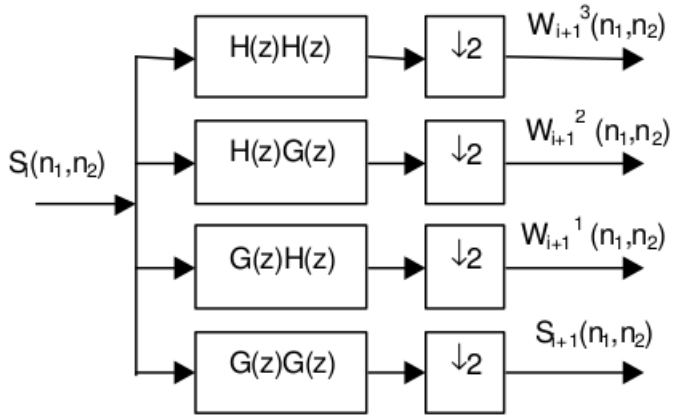


Figura 1.10: *Diagrama de bloques del banco de filtros de análisis utilizado para calcular la DWT 2-D no separable.*

Una realización sencilla de la DWT 2-D no separable es el resultado de aplicar el algoritmo piramidal. En esencia consiste en 4 filtros 2-D modulados (HH, HG, GH y GG) los cuales son usados repetidamente de la forma que muestra la figura 1.11.

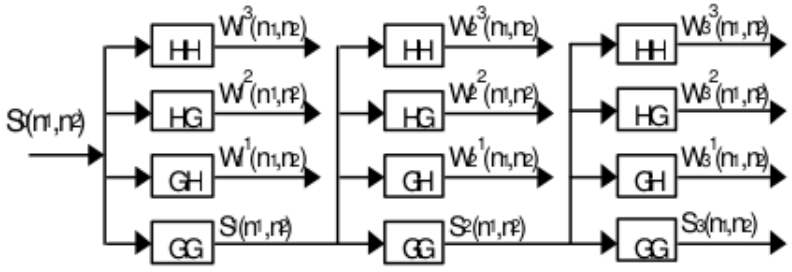


Figura 1.11: *Arquitectura de la DWT 2-D con tres niveles.*

Como se puede intuir con esto, la elección de los filtros (wavelets) influye notablemente en los resultados finales. Los filtros que se aplican los podemos ver en la figura 1.12 que son del tipo CDF 9/7.

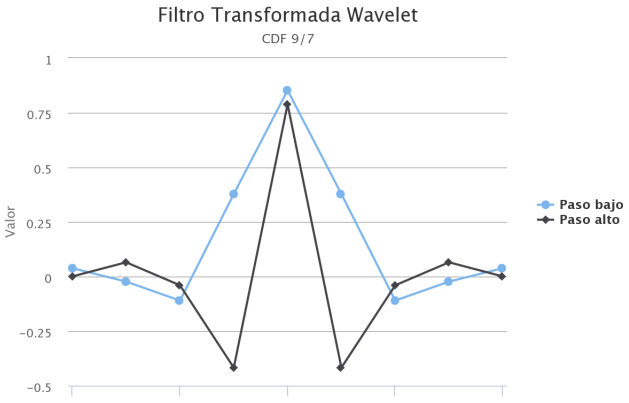


Figura 1.12: *Filtros utilizados para la Transformada Wavelet.*

La DWT aplicada a imágenes proporciona una matriz de coeficientes, conocidos como coeficientes Wavelet. Si a una imagen le aplicamos la DWT obtenemos cuatro tipos de coeficientes: aproximaciones, detalles horizontales, detalles verticales y detalles diagonales. La aproximación contiene la mayor parte de la energía de la imagen, es decir, la información más importante, mientras que los detalles tienen valores próximos a cero.

La figura 1.13 ilustra la descomposición en 8 bandas de una imagen utilizando la DWT 2-D en la que se puede observar el esquema de organización de los coeficientes Wavelet. Vamos a fijarnos en el nivel 1. Tenemos un cuadrado dividido en 4 partes que se obtienen del proceso de descomposición de una imagen en 2-D, tal y como vemos en la figura 1.13 HH son los coeficientes diagonales, GH son los coeficientes horizontales, HG son los coeficientes verticales y GG es la parte en la que se concentra la mayor parte de la información de la imagen, es decir, la aproximación, que como ya se ha mencionado anteriormente suele estar en las bajas frecuencias.

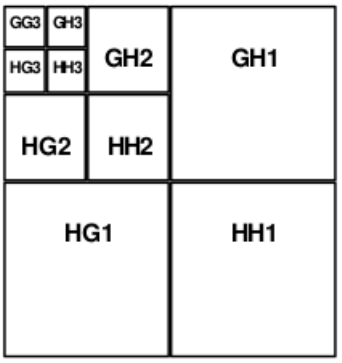


Figura 1.13: *Arquitectura de la DWT 2-D con tres niveles.*

En la figura 1.14 podemos ver el análisis de nivel 1 de la imagen Lena de tamaño 512x512 pixels.

Figura 1.14: *Descomposicion de primer orden. Lena (512x512).*

Entre las características de DWT cabe resaltar las siguientes:

- Aísla y manipula distintos tipos de patrones específicos ocultos en grandes cantidades de datos.
- Comprime o elimina ruido sin degradación apreciable.

1.4.2. JPEG

El JPEG (Joint Photographic Experts Group) es el método de compresión más utilizado actualmente para la compresión de imágenes con pérdida. Este método utiliza la transformada discreta del coseno (DCT), que se calcula empleando números enteros, por lo que se aprovecha de algoritmos de computación veloces. El JPEG consigue una compresión ajustable a la calidad de la imagen que se desea reconstruir.

A fin de proporcionar un estándar universal para la compresión mínima, Joint Photographic Experts Group (JPEG) desarrolló un formato de almacenamiento de la imagen digital basado en estudios de la percepción visual humana. El estándar JPEG describe una familia de técnicas de compresión de imágenes fijas de tonalidad continua en escala de grises o color (24 bits). Sin embargo, numerosas aplicaciones han usado la técnica también para compresión de video, porque proporciona descompresión de imagen de calidad bastante alta a una razón de compresión muy buena, y requiere menos poder de cálculo que la compresión MPEG (Motion Pictures Experts Group).

Debido a la cantidad de datos involucrada y la redundancia psicovisual en las imágenes, JPEG emplea un esquema de compresión con pérdidas basado en la codificación por transformación. El estándar resultante tiene tantas alternativas como sean necesarias para servir a una amplia variedad de propósitos y hoy día es reconocido por la Organización Internacional de Estándares con el nombre de ISO 10918.

El estándar JPEG define tres sistemas diferentes de codificación:

- Un sistema de codificación básico, con pérdidas, que se basa en la Transformada Discreta del Coseno y es apropiado para la mayoría de las aplicaciones de compresión.
- Un sistema de codificación extendida, para aplicaciones de mayor compresión, mayor precisión, o de reconstrucción progresiva.
- Un sistema de codificación independiente sin pérdidas, para la compresión reversible.

Tras la supremacía de JPEG como estándar de compresión de imágenes durante varios años, aparece un nuevo competidor al mismo. Debido al incremento en el uso de las tecnologías multimedia, y a los grandes avances técnicos en informática de los últimos años, la compresión de imágenes requiere mayor potencia así como nueva funcionalidad.

Es por ello que se desarrolla JPEG 2000. No solo se ha pretendido que este estándar ofrezca una mejor calidad subjetiva que JPEG y una mayor tasa de compresión, sino que

además ofrezca una rica gama de nuevas características que consigan el mismo éxito para el nuevo estándar que el que tuvo su predecesor.

1.4.2.1. Ventajas de JPEG

El estándar JPEG tiene una fuerte aceptación en el mundillo informático, de hecho con JPEG2000 no se pretende sustituirlo, solamente complementarlo. Esto se debe a varios factores clave en JPEG:

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

1.4.2.2. Inconvenientes de JPEG

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

- JPEG solo permite una única resolución y calidad.
- En compresión con pérdida la tasa de compresión es baja.
- En imágenes fuertemente comprimidas aparecen los famosos artefactos con forma de cuadrados.
- Es un formato poco resistente a errores, por ejemplo, en transmisión de un JPEG a través de una red inalámbrica propensa a errores.
- No ha sido pensado para la compresión de imágenes sintéticas, solamente para imágenes naturales.

- No admite tener zonas de la imagen codificadas con mayor nivel de detalle que otras.
- No ofrece buena calidad en compresión de imágenes de dos niveles (B/N).

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

Capítulo 2

Lower Tree Wavelet

Contents

2.1. Fundamentos de LTW

Los codificadores de los que hemos ido hablando (Huffman, JPEG, codificadores basados en DCT) están diseñados para obtener un alto rendimiento R/D, pero desafortunadamente otros parámetros como la complejidad o recursos de memoria no se consideran tan importantes.

La principal contribución del codificador LTW es la eficiencia demostrada en la agrupación de coeficientes y su rapidez de codificación, reduciendo el ancho de banda o la cantidad de memoria necesaria para transmitir o almacenar una imagen comprimida. Su complejidad computacional es menor que la de JPEG2000 y funciona hasta 15 veces más rápido con una mejor calidad de imagen.

LTW no necesita una gran cantidad de memoria ni procesador para ejecutarse debido a una de sus principales virtudes: inplace computation (?computación en el sitio?, traducción literal). Esta virtud dota a LTW de necesitar el mismo espacio de memoria para codificar la imagen original que la que ocuparía dicha imagen. Pero esto no significa que LTW deje a un lado el factor R/D, ya que es capaz de obtener un buen rendimiento R/D con las necesidades de recursos de computación reducidos.

2.2. Algoritmo de codificación LTW

Las imágenes digitales están representadas por un conjunto de pixels, P. El codificador LTW puede aplicarse a un conjunto de coeficientes C resultantes del conjunto de pixels.

Un elemento $C_{i,j} \in C$ es denominado coeficiente Wavelet, o coeficiente de la transformada. En una transformada Wavelet, obtenemos ciertas subbandas de frecuencias resultantes de la descomposición de primer nivel de la imagen: LH_1 (horizontal), HL_1 (vertical) y HH_1 (diagonal). El resto de la transformada se realiza mediante una transformada Wavelet recursiva sobre la subbanda de baja frecuencia hasta que se llega al nivel N de descomposición deseado (LL_N es la subbanda restante).

En LTW, el proceso de cuantización se realiza en dos fases. La primera consiste en aplicar una cuantización escalar uniforme, Q, a los coeficientes Wavelet. La otra se basa en quitar los bits menos significativos, $rplanes$, de los coeficientes Wavelet.

La estructura en árbol se usa para reducir la redundancia de datos entre subbandas y también para agrupar rápidamente los coeficientes. Como consecuencia, el número total de símbolos necesarios para codificar la imagen se ve reducido, disminuyendo el tiempo total de la ejecución del algoritmo. Esta estructura se denomina *lower-tree* y es un árbol de coeficientes que son mayores que $2^{rplanes}$.

El algoritmo LTW consta de dos fases. En la primera se construye el mapa de símbolos tras hacer la cuantificación de los coeficientes Wavelet, proceso que depende de Q y $2^{rplanes}$. El conjunto de símbolos empleados en LTW es el siguiente: el símbolo $LOWER(L)$ representa un coeficiente que es la raíz de un *lower-tree*. El resto de coeficientes en un *lower-tree* son $LOWER\_COMPONENT$ $(*)$y nunca son codificados porque ya están representados en el coeficiente raíz (L). Si un coeficiente es insignificante (menor que $2^{rplanes}$) pero no forma parte de un *lower-tree* porque tiene al menos un descendiente significativo se le asigna el símbolo $ISOLATED\_LOWER(I)$. Para un coeficiente significativo, el símbolo utilizado es el número de bits que se necesitan para representarlo (en el caso de 51 se utilizaría el símbolo 6). Para un coeficiente significativo que es raíz de un *lower-tree* se utiliza un símbolo especial que indica el número de bits necesarios para representarlo con superíndice L; por ejemplo 4^L (ver figura 2.1).

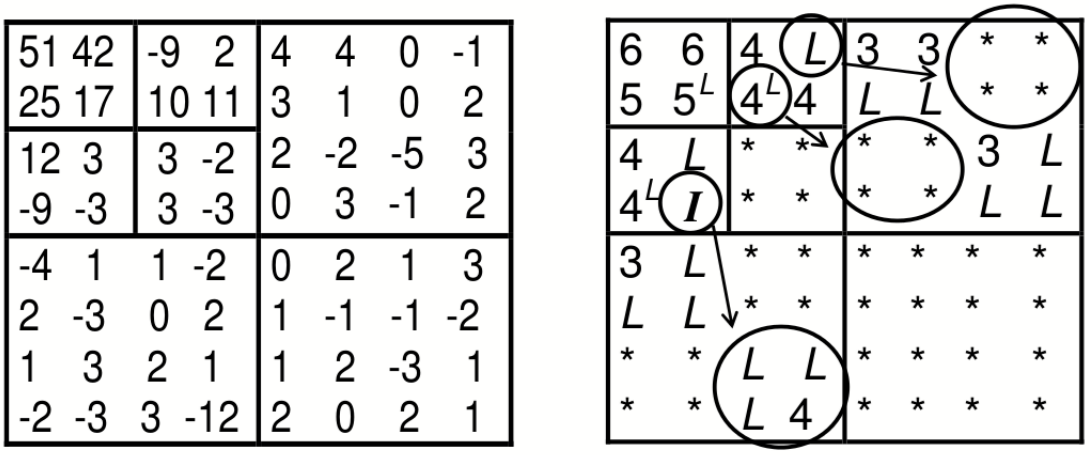


Figura 2.1: *Mapa de coeficientes Wavelet de nivel 2 (izquierda) y su correspondiente mapa de símbolos (derecha) de una imagen de* 8×8.

En la primera parte del algoritmo, todas las subbandas Wavelet son escaneadas en bloques de coeficientes de 2×2, desde el primer nivel de descomposición hasta el N-ésimo. En la subbanda de primer nivel, si los cuatro coeficientes de un bloque de 2×2 son no significativos (menores que $2^{rplanes}$, estos son considerados como $LOWER\_COMPONENT(*)$. Entonces, cuando se escanean subbandas más altas, si el bloque descendiente de 2×2 de coeficientes está marcado como $LOWER\_COMPONENT(*)$, todos se marcan como $LOWER\_COMPONENT$.

Sin embargo, cuando al menos un coeficiente en el bloque es significante, se asigna un símbolo a cada coeficiente individualmente. Los coeficientes insignificantes son marcados como $LOWER$ si sus descendientes son $LOWER\_COMPONENT$. Si no lo son, se marcan como $ISOLATED_LOWER$. Por otro lado, para los coeficientes significativos (mayores que 2 rplanes), se utiliza un símbolo que informe de la cantidad de bits necesarios para su representación. En caso de que todos los descendientes de un coeficiente significativo sean insignificantes ($LOWER\_COMPONENT$), se usa un símbolo especial que indica el número de bits con una L en como superíndice.

Finalmente, en la segunda parte las subbandas son codificadas desde LL^N hasta el primer nivel, como se muestra en la figura 2.2, donde se observa el orden en el que el decodificador necesita saber los símbolos. En cada subbanda, por cada bloque de 2×2 los símbolos son tratados por un codificador aritmético adaptativo. Ningún símbolo $LOWER\_COMPONENT(*)$ es codificado. Además, se añade el signo para cada coeficiente.

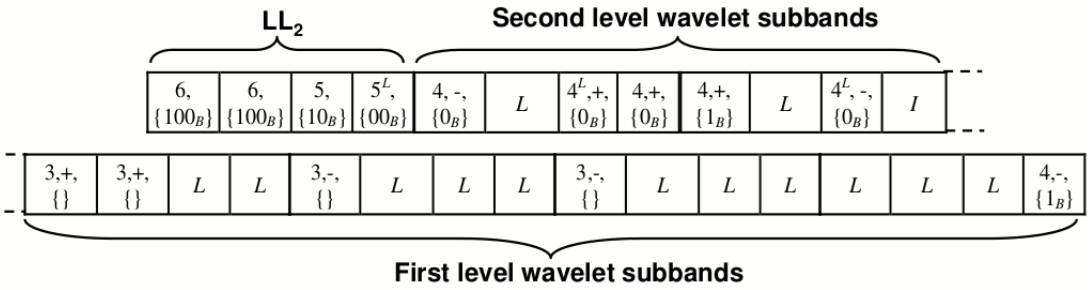


Figura 2.2: *Codificación LTW resultante del ejemplo de la Figura 2.1.*

Como se puede observar en los siguientes algoritmos, el primero de ellos describe la función que realiza la computación de los símbolos de LTW (ver pseudocódigo LTWCalculateSymbols()). Éste está diseñado de forma recursiva, formando *lower-trees* desde las hojas hasta la raíz. En las subbandas de primer nivel, los coeficientes se analizan en bloques de 2×2 y si los 4 coeficientes son no significativos (menores que 2×2), son considerados parte del mismo *lower-tree* y son marcados como $LOWER\_COMPONENT$. Tras esto, cuando se es- canean subbandas más altas, si un bloque de 2×2 tiene sus 4 coeficientes no significativos, todos sus descendientes son $LOWER\_COMPONENT$. Sin embargo, cuando al menos un coeficiente en un bloque es significativo, cada coeficiente del bloque se marca como $LOWER$ si todos sus descendientes son $LOWER\_COMPONENT$. Si no, son marcados como $ISOLATED\_LOWER$.

LTWCalculateSymbols()

Scan the level subbands(HH_1, LH_1 and HL_1) in 2×2 blocks.

For each block B_n

  if $|c_{i,j}| < 2^{rplanes}$ $\forall c_{i,j} \in B_n$

    set $c_{i,j} = LOWER\_COMPONENT$ $\forall c_{i,j} \in B_n$

  else

    for each $c_{i,j} \in B_n$

      if $|c_{i,j}| < 2^{rplanes}$

        set $c_{i,j}$ =LOWER

Scan the rest subbands(from level 2 to N) in 2×2 blocks.

For each block B_n

if $(|c_{i,j}| < 2^{rplanes} \wedge descendant\left(c_{i,j}\right) = LOWER\_COMPONENT)\ \forall c_{i,j} \in B_n$
set $c_{i,j} = LOWER\_COMPONENT\ \forall c_{i,j} \in B_n$
else
for each $c_{i,j} \in B_n$
if $|c_{i,j}| < 2^{rplanes} \wedge descendant\left(c_{i,j}\right) = LOWER\_COMPONENT$
set $c_{i,j} = LOWER$
if $|c_{i,j}| < 2^{rplanes} \wedge descendant\left(c_{i,j}\right) \neq LOWER\_COMPONENT$
set $c_{i,j} = ISOLATED\_LOWER$

En el segundo paso (función LTWOutputCoefficients())), todas las subbandas son exploradas desde el nivel N hasta el 1. Este es el orden en el que el decodificador necesita los coeficientes. Para cada coeficiente en una subbanda, si es $LOWER$ o $ISOLATED_LOWER$, se codifica como tal. Si el coeficiente ha sido marcado como
$LOWER\_COMPONENT$, no se necesita una salida porque dicho coeficiente ya ha sido representado en el árbol al que pertenece. Un coeficiente significativo se codifica de dos maneras: (a) con un número entero representando el número de bits necesarios ($nbits_{i,j}$) o (b), con un símbolo especial ($nbits_{i,j}^{LOWER}$) que no sólo representa el número de bits requeridos, sino también que sus descendientes son $LOWER\_COMPONENT$.

LTWOutputCoefficients()
Scan the subbands(from N to 1, in 2×2 blocks)
For each $c_{i,j}$ in a subband
if $c_{i,j} \neq LOWER\_COMPONENT$
if $c_{i,j} = LOWER$
arithmetic_output $LOWER$
else if $c_{i,j} = ISOLATED\_LOWER$
arithmetic_output $ISOLATED\_LOWER$
else
$nbits_{i,j} = \lceil \log_2\left(|c_{i,j}|\right) \rceil$
if $descendant\left(c_{i,j}\right) \neq LOWER\_COMPONENT$
arithmetic_output $nbits_{i,j}$
else
arithmetic_output $nbits_{i,j}^{LOWER}$

output $bit_{nbits_{i,j}-1}\left(|c_{i,j}|\right)\ldots bit_{rplane+1}\left(|c_{i,j}|\right)$
output $sign\left(c_{i,j}\right)$

2.3. Algoritmo de decodificación LTW

El algoritmo de decodificación de LTW realiza el mismo proceso que la codificación, pero a la inversa y de manera más rápida, ya que no necesita calcular los símbolos de cada coeficiente sino que ya se le dan. Este decodificador viene descrito mediante el siguiente pseudocódigo:

LTWCalculateSymbols()
Scan the level subbands(HH_1, LH_1 and HL_1) in 2×2 blocks.
For each block B_n
  if $|c_{i,j}| < 2^{rplanes}\ \forall c_{i,j} \in B_n$
    set $c_{i,j} = LOWER\_COMPONENT\ \forall c_{i,j} \in B_n$
  else
    for each $c_{i,j} \in B_n$
      if $|c_{i,j}| < 2^{rplanes}$
        set $c_{i,j}$ =LOWER

Scan the rest subbands(from level 2 to N) in 2×2 blocks.
For each block B_n
  if $(|c_{i,j}| < 2^{rplanes} \wedge descendant\left(c_{i,j}\right) = LOWER\_COMPONENT)\ \forall c_{i,j} \in B_n$
    set $c_{i,j} = LOWER\_COMPONENT\ \forall c_{i,j} \in B_n$
  else
  for each $c_{i,j} \in B_n$
    if $|c_{i,j}| < 2^{rplanes} \wedge descendant\left(c_{i,j}\right) = LOWER\_COMPONENT$
      set $c_{i,j}$ =*LOWER*
    if $|c_{i,j}| < 2^{rplanes} \wedge descendant\left(c_{i,j}\right) \neq LOWER\_COMPONENT$
      set $c_{i,j}$ =*ISOLATED_LOWER*

Capítulo 3

Herramientas de trabajo

Contents

3.1. Introducción

Antes de pasar al capítulo dedicado al análisis e implementación de la aplicación, vamos a realizar, en primer lugar, un análisis de la plataforma sobre la que se ha implementado la aplicación y, en segundo lugar, se detallarán las herramientas de las que se ha hecho uso para llevar a cabo su análisis e implementación.

3.2. Android

Android es el sistema operativo de Google orientado a dispositivos móviles y que actualmente supera el 70 % de cuota de mercado. Este sistema operativo fue lanzado al mercado en Octubre de 2008 en su versión 1.0 y, en la actualidad, se encuentra en su versión 5.0.

Figura 3.1: *Logo Android.*

Android está basado en una versión modificada del kernel de Linux 2.6 el cual se utiliza para controlar servicios del núcleo del sistema como pueden ser la seguridad, gestión de memoria o gestión de procesos. Actualmente, podemos encontrarlo presente en teléfonos móviles, PDAs, Tablets e incluso en Netbooks.

Es una plataforma de código abierto distribuida bajo la licencia Apache 2.0 por lo que su distribución es libre y posibilita el acceso y modificación de su código fuente. Inicialmente fue desarrollado por Google, para más tarde unirse a la Open Handset Alliance (de la cual, Google también forma parte) que está integrada por T-Mobile, Intel, Samsung, HTC o Nvidia entre otros. Sin embargo, Google ha sido la compañía que ha publicado la mayor parte del código fuente bajo la licencia Apache.

En cuanto al ámbito de desarrollo de software para esta plataforma, inicialmente a los programadores se les proporcionó de manera gratuita el SDK (donde se incluyen todas las APIs) y un plugin que se integra dentro del entorno de desarrollo de Eclipse, además de un potente emulador integrado para facilitar las pruebas de la aplicación (Android Virtual Devices, AVD). Actualmente, existe un entorno de desarrollo propio llamado Android Studio con el SDK y su simulador nativo (AVD) integrado. A este simulador, también le han salido grandes competidores como podría ser GenyMotion. Este simulador se caracteriza por su velocidad y su diversidad de imágenes para emular dispositivos reales, además de contar

con una fantástica integración tanto en el entorno de desarrollo Eclipse como en Android Studio.

3.2.1. Máquina Virtual Dalvik

Dalvik es la máquina virtual utilizada por Android que permite la ejecución de aplicaciones Android programadas en Java. La Máquina Virtual Dalvik (DVM), al contrario que la Máquina Virtual Java (JVM), ha escogido reducir la portabilidad de las aplicaciones para que tengan un mejor rendimiento y un menor consumo de energía.

Esta mejora aportada por DVM radica en el hecho de que los dispositivos móviles disponen de una menor autonomía. Por ello ha sido diseñada para requerir poca memoria y permitir ejecutar varias aplicaciones al mismo tiempo.

3.2.2. Arquitectura

Android es un sistema diseñado por capas. Como se ha dicho anteriormente, utiliza el kernel de Linux 2.6 que le da acceso a la parte hardware de los dispositivos a la par que le permite ser compatible con muchos de los drivers creados para Linux. Esta arquitectura está perfectamente plasmada en el diagrama que encontramos en la siguiente figura 3.2:

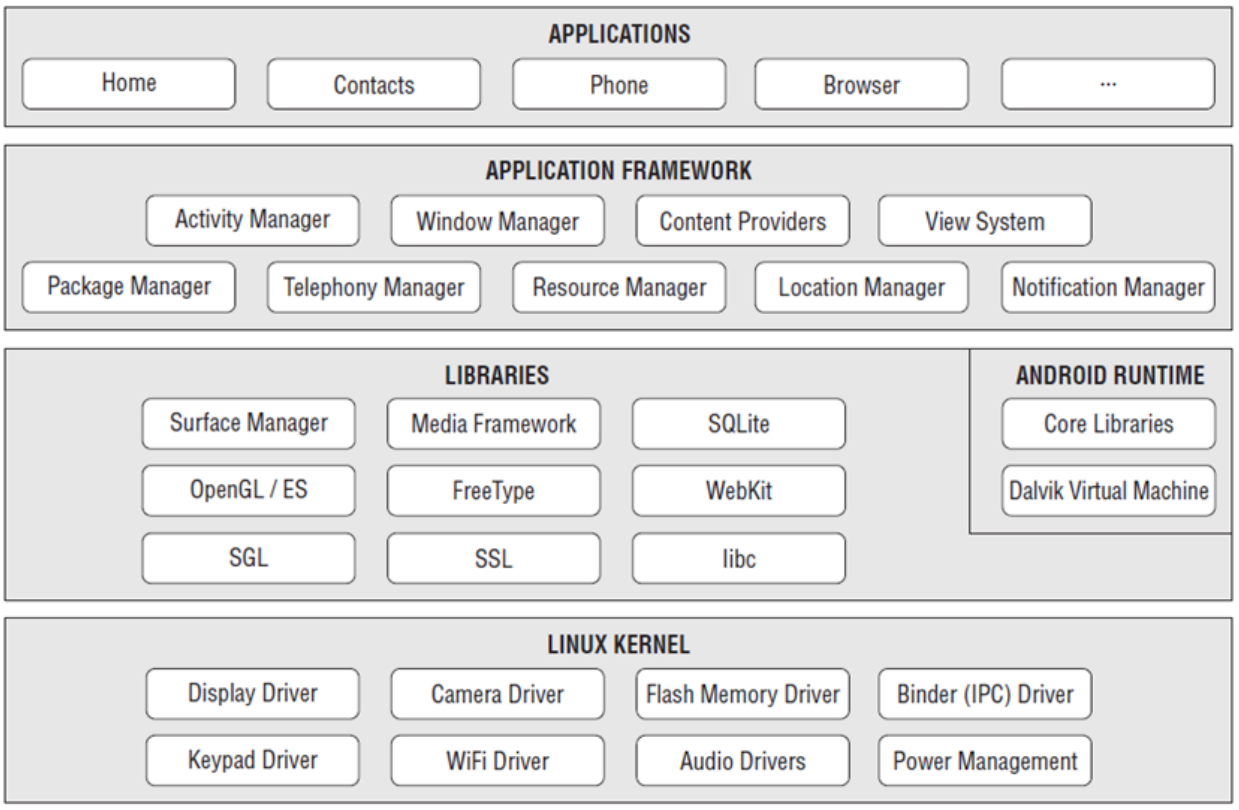


Figura 3.2: *Arquitectura Android.*

A partir de la versión 5.0 de Android, se permite el acceso sin restricciones al hardware de la cámara, accediendo así a las capturas de tipo RAW como podría ser YUV. Esto permite sacar el máximo partido a la cámara ya que dispondríamos de la imagen tal cual es capturada.

3.3. Herramientas de trabajo

En este punto vamos a realizar un estudio de los distintos entornos de desarrollo y entornos de análisis empleados para el análisis e implementación de la aplicación.

3.3.1. Entorno de desarrollo Android

En la programación de las aplicaciones Android existen diferentes entornos de desarrollo. Los más destacados son Eclipse y Android Studio, siendo el primero de ellos el más utilizado.

Android Studio es propiedad de Google Inc. y se basa en IntelliJ Idea, apareció en Mayo de 2013, revolucionando el mundo de la programación Android. A día de hoy se encuentra en su versión estable 1.2.

A pesar de que en algunos casos Eclipse es algo más inestable que Android Studio, por experiencia de uso se ha trabajado con Eclipse (v4.2) para el desarrollo de la aplicación. Además, Android Studio al ser un IDE (Entorno de Desarrollo Integrado) relativamente nuevo, carece de tanta documentación como Eclipse. Todo esto irá sobre la versión 7 de Java.

En este punto, recordamos el capítulo anterior donde se indicaba que para el desarrollo de aplicaciones Android era estrictamente necesario la herramienta Android SDK (Android System Development Kit), donde se encuentran todas las APIs y herramientas de desarrollo e implementación.

3.3.2. Entorno de análisis del codificador LTW 

Para el desarrollo de la aplicación Android será necesario analizar la versión de escritorio del codificador LTW ya existente e implementado en C++. Esta versión de escritorio está optimizada para entornos Windows, por lo que se necesitará otro entorno de desarrollo para este lenguaje. Este entorno de desarrollo será Visual Studio 2013 Ultimate.

El código C++ es simplemente un método de análisis de funcionamiento del codificador. En ningún momento se desarrollará nada en C++.

3.3.3. Diagrama UML. Definición.

Puesto que en el siguiente capítulo se va a detallar el desarrollo de la aplicación Android a través de su diagrama UML, a partir del cual se ha realizado la implementación de la misma, se va a dedicar este apartado a realizar una breve definición del mismo.

UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh.

Además, el desarrollo del diagrama UML ha sido posible gracias a la instalación de un plugin existente para Eclipse que nos ha permitido tener actualizados tanto el diagrama UML como el código de manera automatizada. Junto con esta característica, a continuación se van a presentar tanto los objetivos de UML como el resto de ventajas destacadas.

3.3.3.1. Objetivos

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:

- **Visualizar**
  UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.

- **Especificar**
  UML permite especificar cuáles son las características de un sistema antes de su construcción.

- **Construir**
  A partir de los modelos especificados se pueden construir los sistemas diseñados.

- **Documentar**
  Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

3.3.3.2. Ventajas

UML es un método formal de modelado que aporta la siguientes ventajas:

- Mayor rigor en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y a partir del código fuente generar los modelos, permitiendo así que el modelo y el código estén actualizados.

Capítulo 4

Análisis e implementación de la aplicación

Contents

4.1. Introducción

Una vez asimilados los conceptos expuestos a lo largo de los capítulos anteriores para así entender mejor el objetivo de la aplicación, es el momento de pasar a la fase en la que se detalla el desarrollo que ha sido planteado para este proyecto.

Este capítulo está dedicado a profundizar la parte de desarrollo de la aplicación Android que implementa el codificador LTW. Se abordarán temas como la fase de diseño del proyecto, con una aproximación a la fase del diseño arquitectónico dentro del ciclo de desarrollo de software, o los requisitos de uso, entre otros.

4.1.1. Objetivos

Los objetivos de esta aplicación son varios. Uno de ellos es acceder al hardware de la cámara desde un dispositivo Android con una versión mínima 5.0 y tomar la captura en formato YUV (formato RAW sin pérdida), para posteriormente, con el plano de luminancia de dicha captura se haga un estudio de la eficiencia real del codificador LTW sobre la plataforma Android. Otro de ellos es el de hacer una comparativa de compresión con el formato de imágenes más extendido en la actualidad, JPEG. La aplicación ofrece la posibilidad gráfica de conocer las tasas de compresión tanto de JPEG como de LTW de las capturas tomadas desde nuestro dispositivo móvil.

Además, con la compresión LTW se le ofrece al usuario la posibilidad de almacenar fotografías con un consumo de memoria flash mucho menor del habitual.

4.2. Análisis

En este apartado vamos a definir los requisitos de usuario y las funciones de la aplicación.

4.2.1. Funciones de la aplicación

Lo más importante en la implementación de un aplicación es saber cual es su objetivo y cual deben ser las funciones de dicha aplicación para conseguir dicho objetivo. A continuación vamos a detallar cual son esas funciones para nuestra aplicación:

- **Capturar imágenes:**
  Este requisito es indispensable para la aplicación. La aplicación debe estar dotada de una interfaz que permita acceder a la cámara del dispositivo y capturar imágenes.

- **Modificar/Consultar la configuración de compresión:**
  Se podrá acceder a la configuración de las propiedades de compresión LTW y modificarlas para determinar cuál deben ser sus valores a la hora de capturar una imagen.

- **Consultar las tasas de compresión:**
  Desde la interfaz gráfica de la aplicación se podrá acceder a las tasas de compresión de LTW, junto con una comparativa de compresión de las mismas imágenes capturas en formato de compresión JPEG.

- **Borrar/Copiar/Renombrar los archivos (imágenes):**
  Será posible realizar una copia de los archivos (LTW), cambiar el nombre de los mismos e incluso eliminarlos si se desea.

4.2.2. Requisitos de la aplicación

En este punto vamos a detallar los requisitos de la aplicación, tanto a nivel de plataforma como de permisos.

4.2.2.1. Plataforma Android

El dispositivo en el que se instale/ejecute la aplicación debe contar con una versión Android igual o superior a la 5.0 (Lollipop). Este requisito es indispensable ya que es a

partir de esta versión en la que se permite acceder al hardware de la cámara y obtener el RAW de la imagen, que en nuestro caso será en formato YUV. A partir de este formato, se obtendrá la compresión LTW de la luminancia (Y) de dicha imagen (YUV).

4.2.2.2. Permisos

En la instalación de la aplicación, se le debe dar permisos al usuario a los siguientes recursos del dispositivo:

- Montar la unidad de almacenamiento externo
- Leer de la unidad de almacenamiento externo.
- Escribir en la unidad de almacenamiento externo.
- Acceso total a la cámara del dispositivo móvil.

4.3. Análisis de implementación

Como ya se ha dicho, existe una implementación del codificador LTW desarrollada en C++ para escritorio. Para desarrollar la aplicación Android, es necesario conocer el funcionamiento de esta implementación, ya que nos facilitará la etapa de diseño de la arquitectura de la aplicación.

A continuación se detallan los pasos a seguir para la implementación del codificador:

1. Análisis de la versión de escritorio (C++) del codificador LTW

En este proyecto se va a contar con la ventaja de disponer de una implementación ya desarrollada del codificador, aunque esté en otro lenguaje de programación. Con el análisis de esta implementación ya desarrollada, lo que se pretende es conocer la estructura de la aplicación. Realizando un *debug* de esta aplicación se puede conocer su traza de ejecución para plasmarla en nuestra aplicación.

2. Desarrollo de una librería para la aplicación Android

Una vez realizado un exhaustivo análisis del codificador en C++, se implementará una librería Java que imite la traza de ejecución de la aplicación ya implementada con el fin de que para una luminancia (Y) de entrada al codificador LTW, junto con los parámetros de configuración, se obtenga su correspondiente archivo LTW comprimido.

3. Maquetación de la aplicación Android

En el proceso de maquetación, además de importar la librería Java del codificador LTW implementada en el punto anterior, se desarrollará la interfaz de usuario. En esta interfaz se desarrollará tanto la apariencia gráfica de la aplicación como la lógica necesaria para interactuar con el usuario y así recabar la información de entrada necesaria al codificador LTW.

4. Uso de la librería Java (codificador LTW)

Como ya hemos indicado en el punto anterior, la interfaz gráfica interactuará con el usuario de manera que se obtengan la información necesaria de entrada a la codificación LTW para, con ella, obtener el archivo LTW de salida.

4.3.1. Librería Java. Implementación del codificador LTW.

Esta librería se encargará de la codificación LTW. Se accederá a ella a través de una llamada a la clase LTW desde el propio código dónde le pasaremos los siguientes parámetros:

- **LEVEL**:
  Niveles de descomposición de la Transformada Wavelet.
- **ancho**:
  Valor del ancho de la imagen en píxeles.
- **Alto**:
  Valor del alto de la imagen en píxeles.
- **Imagen**:
  Plano de luminancia de la imagen capturada representada por un array con valores de tipo double.
- **Rplanes**:
  Bits que quitaremos a los coeficientes Wavelet.
- **q**:
  Nivel de cuantización (quant). El rango de posibles valores oscila entre $(0 \mapsto 1]$.
- **filename**:
  Ruta del archivo dónde se almacenará la imagen comprimida en formato LTW.

A continuación, se va a detallar la funcionalidad de la clase LTW junto con el resto de clases que componen el codificador:

- **LTW**
  Esta clase es la primera que se va a ejecutar. Sirve para recoger los datos más importantes de la imagen que se va a tratar: nombre del fichero dónde se va almacenar la imagen, rplanes, nivel de cuantización, ancho, alto, niveles de descomposición y por supuesto, la propia imagen. Dentro de ella se crearán dos objetos de las clases:
  - *Wavelet*:
    Objeto para realizar la Transformada Wavelet de la imagen.

- *Cuantificador*
  Objeto que crea el mapa de símbolos y los escribe en el fichero que hemos pasado a la clase LTW.

- **Wavelet**
  En esta clase se realiza la transformada Wavelet de la imagen. El método principal de esta clase es:

  - *DirectTransform*:
    Este método aplica la DWT (Discrete Wavelet Transform) a la imagen capturada. Se aplicará tantas veces como niveles de descomposición haya seleccionado el usuario. Al finalizar, tendremos un array del mismo tamaño que el anterior, pero con floats en lugar de bytes debido a que se aplican los filtros CDF 9/7 (vistos en el apartado 1.4.1.2. Transformada Wavelet Discreta (DWT)).

- **Cuantificador**
  Esta clase calcula los símbolos correspondientes al conjunto de coeficientes obtenidos anteriormente. Cabe destacar la importancia de esta clase ya que a partir de ella obtenemos el mapa de símbolos y también el nivel de compresión.

  El problema que tenemos aquí es que, al contrario que en C++, necesitamos crear un nuevo array con los símbolos, que serán enteros, por lo que estamos utilizando más memoria que en la implementación para escritorio. Esta es una pequeña limitación que tenemos al utilizar Java.

  Los métodos principales de esta clase son:

  - *CalculaSimbolos*:
    Este método realiza una pasada hacia arriba de la matriz aplicando el algoritmo visto en la figura **??**.

  - *CodificaSimbolos*
    Tras haber calculado los símbolos, ahora hacemos el recorrido inverso, hacia abajo, de la matriz en Z para escribir los símbolos en un fichero. Para ello, se hace uso de la clase *CodificadorAritmetico*.

- **Codificador Arimético**

  Esta clase se encarga de gestionar los bits que se escriben en el fichero según las probabilidades y frecuencia de aparición de cada símbolo. Para ello, se hace uso de la clase *TbitStream*.

- **TbitStream**

  Esta clase gestiona la escritura y lectura de bits en fichero con la librería RandomAccessFile, es decir, es la clase que escribirá en el fichero los bits que le indique el objeto CodificadorAritmetico.

A grandes rasgos, esta sería la librería Java desarrollada para la implementación del codificador LTW.

4.4. Desarrollo de la aplicación Android

Una vez creada la librería Java para la implementación del codificador LTW, vamos a pasar al detalle del desarrollo de la aplicación Android, y lo vamos a hacer a través del diagrama UML de la misma, como se comentó en el capítulo anterior.

4.4.1. Diagrama UML. Descripción, clases y métodos de la aplicación.

Esta subsección está dedicada a la explicación de la estructura de la aplicación a través de su diagrama UML.

Este diagrama lo hemos separado en dos partes, en el diagrama de la figura 4.1 se muestran las relaciones de la actividad principal,*CameraActivity*, y en la figura 4.2 se muestran las relaciones de la librería LTW.

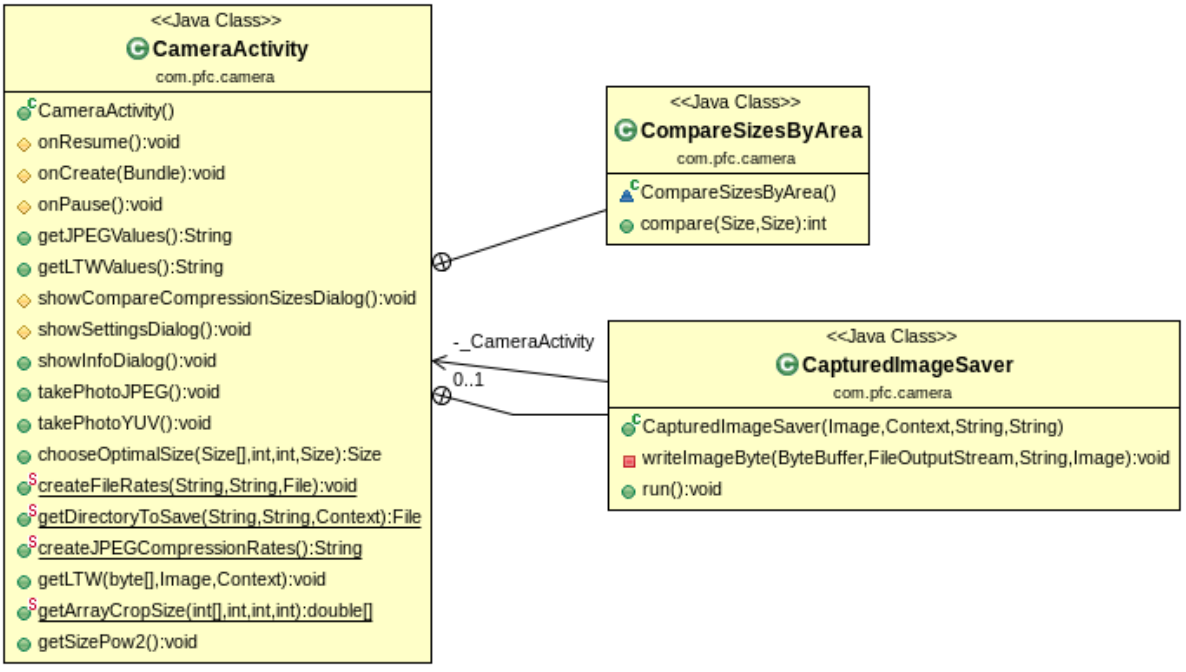


Figura 4.1: *Diagrama UML (CameraActivity).*

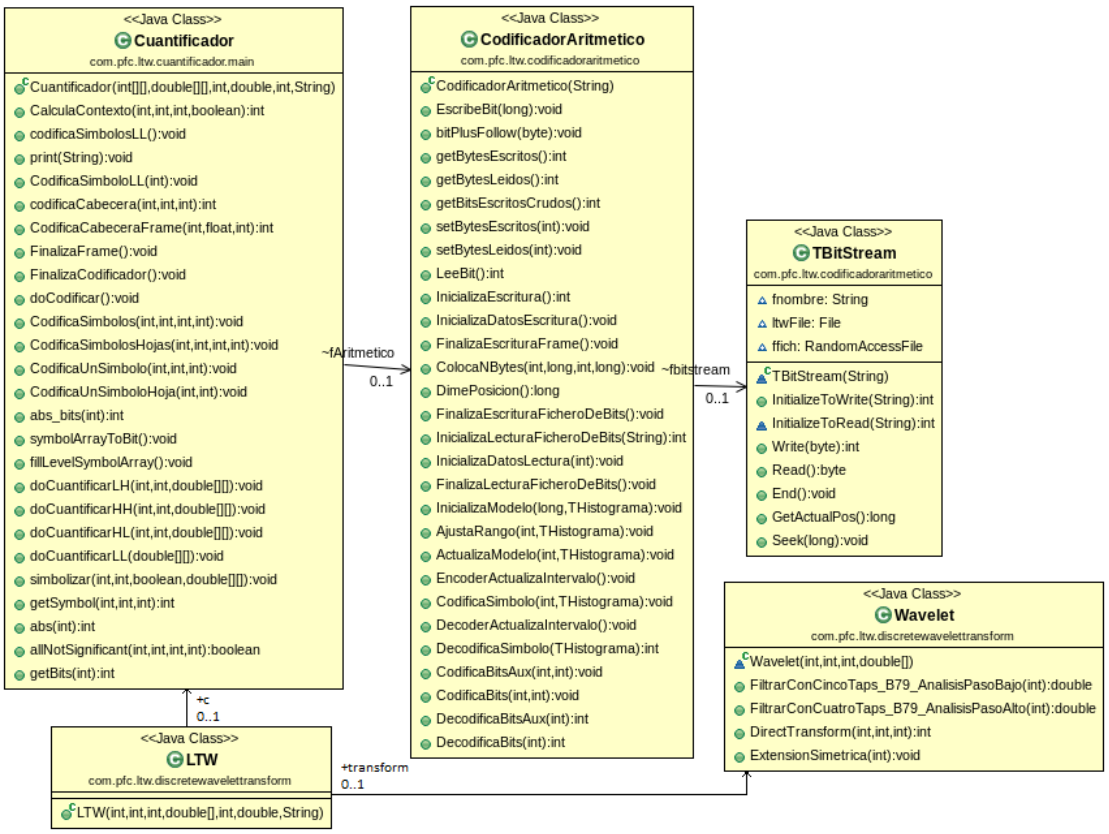


Figura 4.2: *Diagrama UML (LTW).*

4.4.1.1. Definición de la aplicación a través de su diagrama UML

Como se muestra en el diagrama UML de la figura 4.1, existen cuatro clases relacionadas con la clase principal, CameraActivity. Esta clase será la que se ejecute al iniciarse la aplicación puesto que es el núcleo de la aplicación. Cuando la aplicación se inicie, CameraActivity controlará la interfaz de usuario junto con su lógica. Al crearse dicho Activity, el primer método al que se llamará será el método onCreate() que hace las veces de constructor de clase y es desde donde se llama a setContentView. Por ello, en este método será el lugar donde se instancien los objetos del layout junto con la lógica para cada uno de ellos, que en el caso de nuestros cuatro botones se encontrará en el evento 'onClickListener'. Será aquí también donde creemos la vista de la cámara en nuestro SurfaceView. De aquí la relación entre CameraActivity y CompareSizesByArea, clase la cual se utiliza para establecer el tamaño óptimo del SurfaceHolder. En el siguiente punto se detallará cómo el paquete android.hardware.camera2 proporciona una interfaz para dispositivos de cámara

Android a partir del API21. Al capturar una imagen, CameraActivity recurrirá a la clase CapturedImageSaver para proceder a guardarla, una vez haya sido esta imagen comprimida a través de la librería LTW. Una vez explicado el diagrama UML de la figura 4.1 y sus relaciones, pasamos al diagrama UML de la figura 4.2 donde se presentan las relaciones entre las clases que componen la compresión de imágenes. Dentro de la clase LTW hacemos la Transformada Wavelet de la imagen con los niveles de descomposición que haya seleccionado el usuario o, en caso de no haberlo hecho, con los niveles por defecto. Tras esto creamos un objeto de la clase Cuantificador para que cree el mapa de símbolos y escriba estos a un fichero a través de una instancia de la clase Codificador Aritmético, que a su vez hace uso de TBitStream.

4.5. Paquete android.hardware.camera2

En este punto vamos a estudiar las limitaciones del paquete predecesor a android.hardware.camera2 que nos servirá de introducción para presentar las características y mejoras de este nuevo paquete. Con esto, estudiaremos la arquitectura, la configuración empleada y relataremos las conclusiones extradias en este punto.

4.5.1. Limitaciones de android.hardware.camera

Antes de profundizar en las características del nuevo paquete android.hardware.camera2 vamos a enumerar las principales limitaciones con las que contaba su predecesor, android.hardware.camera, y que llevan a la implementación de este nuevo paquete que nos ocupa este punto:

- Solo cuenta con tres modos de operación: vista preliminar, captura y grabación de vídeo.
- Nuevas características difíciles de implementar: modo ráfaga, zero shutter lag (factor de retardo entre pulsar el botón y el disparo efectivo del objetivo), multi disparo HDR, imágenes panorámicas, entre otras.
- No es posible el control por trama.
- Metadatos mínimos.
- Ajustes personalizados primitivos.

4.5.2. Características de android.hardware.camera2

Conocidas estas limitaciones, vamos a profundizar en las características aportadas por el paquete android.hardware.camera2 que vamos a agrupar en cuatro categorías:

1. POINT AND CLICK
    - Modos de operación:
        - Vista preliminar
        - Fotograma

- Grabación de vídeo

2. PROFESSIONAL CAMERA

   - Control de alta calidad de
     - Sensor
     - Flash
     - Lente
   - Canalización de procesamiento de señal de imagen
   - Procesar imágenes a máxima resolución y frecuencia (30 fps)
   - Control sobre una base por trama y comportamiento determinista
   - Salida RAW del sensor con metadatos de captura: cada trama se devuelve con la configuración real con que se tomó y para la que se hizo la petición.

3. COMPUTATIONAL PHOTOGRAPHY

   - Permite fotografía de calidad profesional (DSLR en lugar de apuntar y disparar)
   - Permite el acceso a imágenes de tipo RAW
   - Permite el procesamiento en el dispositivo de datos de la cámara: High Dynamic Range (HDR), focus stacking
   - Permite nuevos casos de uso combinando imágenes con
     - Abundantes entradas del sensor: sensores inerciales, de altitud, etc.
     - Potencia de computación: multi-core CPU & GPU
     - Conectividad: cloud & proximity (BLE, NFC)
     - Contexto: localización & historial de usuario
   - Multi disparo HDR: múltiples disparos en diferentes exposiciones y luego mezclados juntos
   - Panoramic Stitching: múltiples imágenes con información de orientación y exposición fija
   - Fotografía con flash y sin flash

- Focus stacking para macro fotografía. También conocido como fotografía all-in-focus
- Se permite editar la trama en el visor mediante la selección de desenfoque, brillo, etc., y mantenerlo durante el modo de captura

4. INNOVATIVE MOBILE CAMERAS

- Sensores
- Localización
- Conectividad/Cloud
- Reconocimiento de gestos
- Reconocimiento facial
- Seguimiento de objetos
- Búsqueda visual
- Mapas en 3D
- Realidad aumentada

4.5.3. Arquitectura de android.hardware.camera2

Una vez enumeradas las principales características y aportaciones de camera2, se va a mostrar una síntesis de la arquitectura sobre la que está implementado:

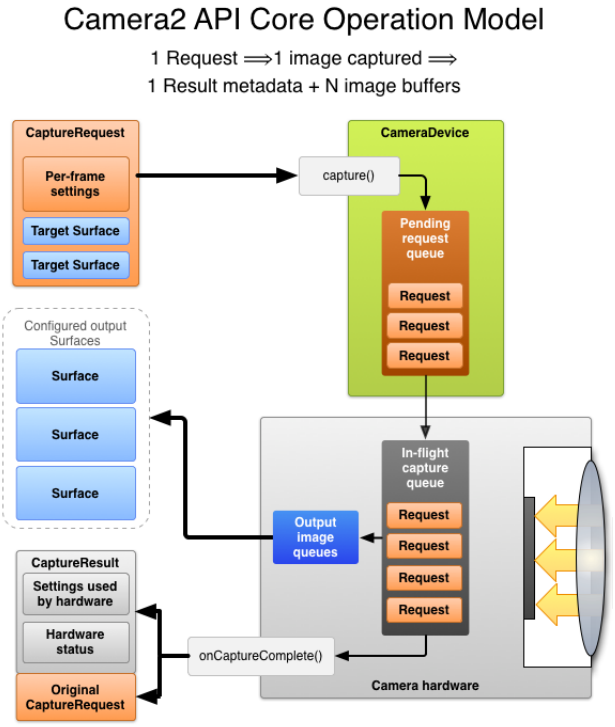


Figura 4.3: *Core Operation Model de android.hardware.camera2*

Puntos a destacar:

1. Se establece la configuración por trama.
2. Esta configuración viaja con la petición individual y ya no se aplica de forma global.
3. Se entrega el buffer de la imagen a todos los Surfaces configurados y solicitados.
4. Se devuelven los metadatos a la aplicación, correspondientes a las peticiones individuales.
5. Las solicitudes y los resultados múltiples se encuentran en cola simultáneamente.

Como se puede ver en la arquitectura, este paquete modela un dispositivo de cámara como una fuente de información, la cual coge una petición de entrada para capturar un solo fotograma, captura esa única imagen solicitada, y luego envía un paquete de metadatos como resultado de la captura, además de un conjunto de buffers de imagen de salida para la solicitud. Las solicitudes se procesan en orden, y múltiples solicitudes pueden ser

lanzadas simultáneamente. Dado que el dispositivo de la cámara es una fuente de información con múltiples etapas, que tiene múltiples solicitudes lanzadas simultáneamente, se requiere mantener la máxima frecuencia de procesamiento de imágenes en la mayoría de los dispositivos Android.

Con esto, vamos a pasar a profundizar los conceptos y términos vistos en la arquitectura y necesarios para el empleo del paquete android.hardware.camera2.

- CameraDevice
  - Consulta el sistema de la cámara para conocer sus capacidades (propiedades, ajustes disponibles, parámetros de salida, etc.)
    - CameraManager proporciona información acerca del número de cámaras disponibles o CameraDevices.
    - CameraCharacteristics proporciona metadatos estáticos para un CameraDevice dado. Esta información es inmutable para una cámara dada.
- Capture Session
  - Operaciones costosas
  - Todos los modos no pueden ser soportados simultáneamente
    - Demasiadas sesiones pueden hacer que se vengan abajo y necesitar así realizar de nuevo la configuración
  - Un fallo al crear una sesión puede dar excepciones al ser lanzada la sesión
- Target Surfaces
  - Diferentes Target Streams de salida:
- Capture Request
  - Elección de la Capture Request Template
    - TEMPLATE_MANUAL
      - ◇ TEMPLATE básico para el control de la aplicación directo de los parámetros de captura.
    - TEMPLATE_PREVIEW
      - ◇ Solicitud para una vista previa.

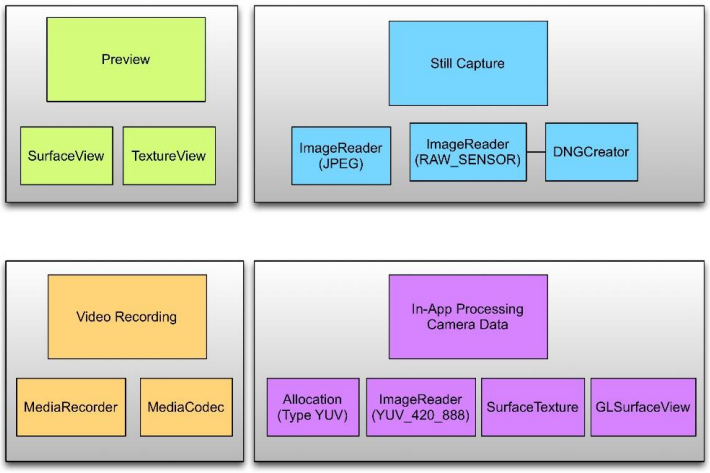


Figura 4.4: *Target Surfaces de android.hardware.camera2*

  - TEMPLATE_RECORD
    - Solicitud para la grabación de vídeo.
  - TEMPLATE_STILL_CAPTURE
    - Solicitud para la captura de imágenes fijas.
  - TEMPLATE_VIDEO_SNAPSHOT
    - Solicitud para la captura de imágenes fijas durante la grabación de vídeo.
  - TEMPLATE_ZERO_SHUTTER_LAG
    - Solicitud para la captura de fotogramas zero shutter lag.
- Escoger un Target apropiado de salida
- Seleccionar la frecuencia de las Capture Request
  - capture
    - Solicitud de una imagen para ser capturada por la cámara del dispositivo.
  - captureBurst
    - Lista de peticiones para ser capturada una secuencia, como por ejemplo una ráfaga.
  - setRepeatingRequest

        - ◇ Repetir sin cesar la captura de imágenes de la actual Capture Session.
      - setRepeatingBurst
        - ◇ Repetir sin cesar la captura de una secuencia de imágenes de la actual Capture Session.
      - stopRepeating
        - ◇ Cancelar cualquier repetición de captura establecida o bien por setRepeatingRequest o bien por setRepeatingBurst.
      - abortCaptures
        - ◇ Desecha todas las capturas que se encuentren pendientes o en curso lo más rápido posible.

- Capture Result
  - Image Data
    - Los datos de imagen suelen ser recibido en un listener asociado a la salida del Surface.
  - Metadata
    - Los metadatos se reciben en el método onCaptureCompleted callback de CameraCaptureSession.CaptureCallback a través del objeto TotalCaptureResult.
  - Total Capture Result

4.5.4. Configuración de android.hardware.camera2

En este apartado se va a analizar la configuración establecida del paquete android.hardware.camera2 para la parte de acceso a la cámara de la implementación de la aplicación.

Se va a detallar la configuración en los siguientes puntos principales:

1. Se selecciona SurfaceView como target
2. Se usa TEMPLATE_STILL_CAPTURE para CaptureRequest
3. Se crea un ImageReader con ImageFormat.YUV_420_888
4. Se escoge como frecuencia de CaptureRequest CameraCaptureSession.capture()
5. Se obtiene ImageData en ImageReader.onImageAvailableListener

4.5.5. Conclusiones

Todas estas propiedades implementadas por este paquete disponible a partir de la versión 5.0 de Android hacen posible la implementación del codificador LTW. Hasta esta versión, con la inclusión de android.hardware.camera2, no era posible el acceso sin restricciones al hardware de la cámara que permite, entre otras, la capacidad de obtener imágenes en formato RAW, entre los que cabe destacar el formato YUV.

Destacamos la propiedad de obtener imágenes en formato RAW y en concreto el formato YUV debido a que es este formato sin pérdidas el que hace posible la implementación real del codificador LTW, que comprime la luminancia de la imagen (Y) del formato YUV.

Además, permitirá el perfeccionamiento y ampliación futura de la aplicación que veremos en el capítulo 6.

4.6. Diseño de la aplicación

En este apartado vamos a aludir a la popular frase de ”Las interfaces de usuario son como los chistes, si tienes que explicarlas, seguro no son muy buenas”. Efectivamente, este es un punto crítico en el diseño de una aplicación. Es un punto importante para que los usuarios lleven tu App en ”el bolsillo”.

En el caso que nos compete, si se quiere extender un nuevo algoritmo de compresión de imágenes, un método efectivo sería implementarlo en una aplicación móvil que puedan llevar siempre encima y usarla en cualquier momento, pero que además no les limite la eficiencia de procesamiento de su dispositivo y que les resulte cómoda, que no sea necesario explicarla.

La interfaz de la aplicación desarrollada se puede observar en la figura 4.5.

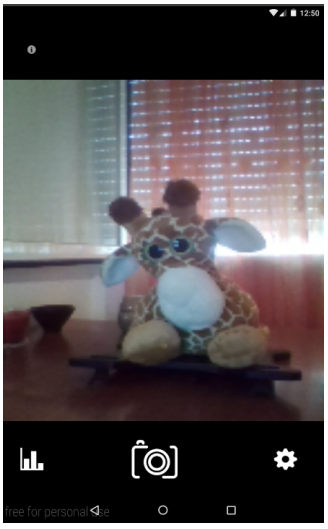


Figura 4.5: *Interfaz gráfica de la aplicación*

Esta aplicación posee una barra de herramientas en la parte inferior de la pantalla con tres botones. El primero de ellos empezando por la izquierda nos muestra una comparativa gráfica entre LTW y JPEG, como puede apreciarse en la figura 4.6.

Figura 4.6: *Comparativa gráfica entre LTW y JPEG*

El botón central captura la imagen que deberá ser comprimida con LTW. Y con el último botón de esta barra de herramientas se accedería a la configuración de la aplicación como se puede ver en la figura 4.7.

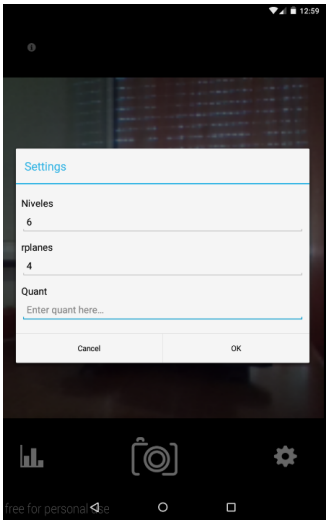


Figura 4.7: *Menú configuración de la aplicación*

En este menú de configuración se podrán establecer los siguientes parámetros de la aplicación:

Niveles

Niveles de descomposición de la Transformada Wavelet.

Rplanes

Bits que quitaremos a los coeficientes Wavelet.

Quant

Nivel de cuantización. El rango de posibles valores oscila entre $(0 \mapsto 1]$.

Además, la aplicación cuenta con un botón superior izquierdo de información dónde se describiría el propósito de la misma. Podemos ver el resultado de pulsar dicho botón en la figura 4.8.

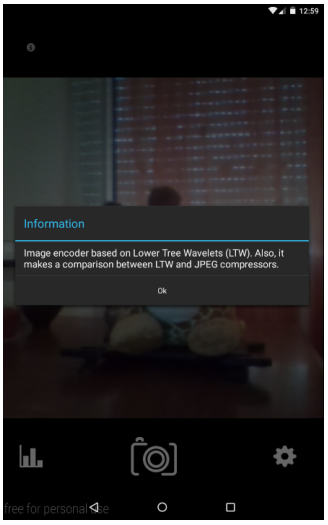


Figura 4.8: *Botón información de la aplicación.*

Capítulo 5

Pruebas y resultados

Contents

5.1. Introducción

En capítulos anteriores se ha visto el funcionamiento del codificador LTW y la arquitectura de la aplicación Android, justificando teóricamente los métodos de implementación para obtener los mejores resultados posibles en la compresión de imágenes. En este capítulo vamos a testear la eficiencia de la aplicación.

La dinámica de testeo la vamos a establecer en dos bloques. En el primero de ellos se va a testear la eficiencia de compresión frente a JPEG. En el segundo, se van a analizar los tiempos de codificación sin escritura y con escritura en disco desde distintos dispositivos con distintas configuraciones del codificador LTW (distintos niveles de descomposición, rplanes y q).

Una vez realizadas las pruebas pertinentes para ambos bloques, se va a concluir el capítulo con un estudio mas exhaustivo de los resultados obtenidos.

5.2. Test de compresión. Comparativa entre LTW y JPEG.

En la tabla 5.1 se muestran los resultados de las pruebas realizadas. La relación de compresión para JPEG se realiza en base al tamaño de los tres planos de la imagen YUV, sin embargo, el porcentaje de compresión LTW se realiza en base al tamaño del plano de luminancia de la imagen (Y). Esto se debe a que JPEG realiza la compresión de los tres planos, plano de luminancia (Y) mas los dos planos de cromas(UV), y LTW solo realiza la compresión del plano de luminancia (Y).

Con esto, las relaciones de compresión para ambos formatos se calcularán en base a las siguientes fórmulas:

- Relación de compresión para JPEG:

$$RC_{jpeg} = \frac{tamanoficheroYUV}{tamanoficheroJPEG}$$

- Relación de compresión para LTW:

$$RC_{ltw} = \frac{tamanoficheroY}{tamanoficheroLTW}$$

Esta relación de compresión (RC) nos indica en qué proporción ha sido reducida la información. Para entender los resultados de la tabla 5.1, supongamos que se obtiene una *RC* igual a $10:1$. Esto indicaría que por cada 10 bits del fichero original solamente tenemos 1 bit en el fichero comprimido, es decir, el tamaño del fichero se habrá reducido en veces.

Niveles	rplanes	Quant	Bytes YUV	Bytes Y	Bytes		RC	
					LTW	JPEG	LTW	JPEG
6	2	0,573	460800	307200	10901	29662	28:1	16:1
		0,873	460800	307200	16388	29646	19:1	16:1
		0,973	460800	307200	18022	29426	17:1	16:1
6	3	0,573	460800	307200	3424	26022	90:1	18:1
		0,873	460800	307200	5462	26103	56:1	18:1
		0,973	460800	307200	6102	25963	50:1	18:1
6	4	0,573	460800	307200	1532	26174	201:1	18:1
		0,873	460800	307200	2465	26293	125:1	18:1
		0,973	460800	307200	2628	25917	117:1	18:1
6	5	0,573	460800	307200	733	26152	419:1	18:1
		0,873	460800	307200	1115	25990	276:1	18:1
		0,973	460800	307200	1228	26194	250:1	18:1
6	6	0,573	460800	307200	342	26063	898:1	18:1
		0,873	460800	307200	551	26224	558:1	18:1
		0,973	460800	307200	593	25870	518:1	18:1
5	2	0,573	460800	307200	13014	29737	24:1	15:1
		0,873	460800	307200	19260	29474	16:1	16:1
		0,973	460800	307200	21600	29656	14:1	16:1
5	3	0,573	460800	307200	4797	29595	64:1	16:1
		0,873	460800	307200	7842	29895	39:1	15:1
		0,973	460800	307200	8819	29161	35:1	16:1
5	4	0,573	460800	307200	2210	29696	139:1	16:1
		0,873	460800	307200	3373	29745	91:1	15:1
		0,973	460800	307200	3746	29811	82:1	15:1
5	5	0,573	460800	307200	1116	30106	275:1	15:1
		0,873	460800	307200	1636	29840	188:1	15:1
		0,973	460800	307200	1809	29635	170:1	16:1
5	6	0,573	460800	307200	577	29881	532:1	15:1
		0,873	460800	307200	821	29982	374:1	15:1
		0,973	460800	307200	922	29836	333:1	15:1

Cuadro 5.1: *Bytes escritos y RC LTW frente a JPEG*

5.3. Test tiempos de codificación

Como se ha visto en la introducción, este segundo bloque de testeo se dividirá en un estudio de codificación LTW sin escritura en disco y otra de dicha codificación con escritura en disco.

En este último test, a priori se puede deducir que el proceso sin escritura en disco va a ser más rápido y que, además, la escritura en disco dependerá de las características hardware del dispositivo desde el que se ejecute. Es por ello que este segundo test de tiempos se ha realizado desde diversos dispositivos.

Los dispositivos con los que se ha realizado el testeo son:

- **Google Nexus 7**
  - *Versión Android:* 5.1.0 (Lollipop)
  - *Cámara:* 8MP, autofocus, flash LED
  - *RAM:* 2G.
  - *Procesador:* Qualcomm Snapdragon 800 MSM8974 a 2.5GHz
  - *GPU:* Adreno 330 (450 MHz)
  - *Pantalla:* 5”1080x1920px
- **Google Nexus 9**
  - *Versión Android:* 5.1.0 (Lollipop)
  - *Cámara:* 8MP, autofocus, flash LED
  - *RAM:* 2G.
  - *Procesador:* NVIDIA Tegra K1 dual-core de 64 bits a 2.3GHz
  - *GPU:* Kepler de 192 núcleos
  - *Pantalla:* 8.9? 2048 x 1536px

5.3.1. Test sin escritura en disco

En la tabla 5.2 se muestran los resultados de los tiempos de codificación sin escritura en disco con distintos valores para los parámetros de entrada del codificador LTW desde los distintos dispositivos.

Niveles	rplanes	Quant	Bytes	Tiempo (ms)	
				Nexus 9	Nexus 7
6	2	0,573	10901	273	352
		0,873	16388	293	206
		0,973	18022	347	346
6	3	0,573	3424	298	398
		0,873	5462	182	296
		0,973	6102	341	227
6	4	0,573	1532	163	225
		0,873	2465	195	234
		0,973	2628	254	280
6	5	0,573	733	211	361
		0,873	1115	257	275
		0,973	1228	270	254
6	6	0,573	342	202	300
		0,873	551	211	254
		0,973	593	298	296
5	2	0,573	13014	189	248
		0,873	19260	335	354
		0,973	21600	324	513
5	3	0,573	4797	171	445
		0,873	7842	268	499
		0,973	8819	271	383
5	4	0,573	2210	323	311
		0,873	3373	259	296
		0,973	3746	263	372
5	5	0,573	1116	255	339
		0,873	1636	387	355
		0,973	1809	244	455
5	6	0,573	577	253	518
		0,873	821	284	210
		0,973	922	216	299

Cuadro 5.2: *Tiempo de ejecución en milisegundos sin escritura en disco*

5.3.2. Test con escritura en disco

En la tabla 5.3. se muestran los resultados de los tiempos de codificación y posterior escritura en disco del archivo comprimido en formato LTW con distintos valores para los parámetros de entrada del codificador LTW desde los distintos dispositivos.

Niveles	rplanes	Quant	Bytes	Tiempo (ms)	
				Nexus 9	Nexus 7
6	2	0,573	10901	8016	4444
		0,873	16388	11540	5943
		0,973	18022	12563	6639
6	3	0,573	3424	2701	1519
		0,873	5462	4284	2554
		0,973	6102	5252	3121
6	4	0,573	1532	840	1026
		0,873	2465	1806	1087
		0,973	2628	2079	1051
6	5	0,573	733	692	439
		0,873	1115	950	651
		0,973	1228	1190	665
6	6	0,573	342	532	337
		0,873	551	650	431
		0,973	593	613	408
5	2	0,573	13014	9904	5036
		0,873	19260	15326	7363
		0,973	21600	17853	8541
5	3	0,573	4797	4216	2025
		0,873	7842	6224	3202
		0,973	8819	6840	3335
5	4	0,573	2210	1863	1090
		0,873	3373	2622	1597
		0,973	3746	2929	1838
5	5	0,573	1116	920	618
		0,873	1636	1463	880
		0,973	1809	1677	870
5	6	0,573	577	737	397
		0,873	821	920	572
		0,973	922	852	599

Cuadro 5.3: *Tiempo de ejecución en milisegundos con escritura en disco*

5.4. Estudio de los resultados obtenidos

Se va a empezar por estudiar la relacion de compresion LTW. En primer lugar, se va a analizar el comportamiento de compresion de rplanes a partir de un valor fijo de niveles de descomposicion y de quant. En la figura 5.1 se muestra el tamaño del archivo comprimido LTW en el eje Y y el valor de rplanes en el eje X. Claramente existe una mayor compresion a mayor valor de rplanes. Esto se debe a que rplanes elimina los bits menos significativos, lo cual hace que la compresion sea mayor, haciendo mar arboles de coeficientes nulos ($LOWER\_COMPONENT$) con menos calidad pero con un tiempo de ejecucion mucho mejor como se puede observar en las 5.2 y 5.3.

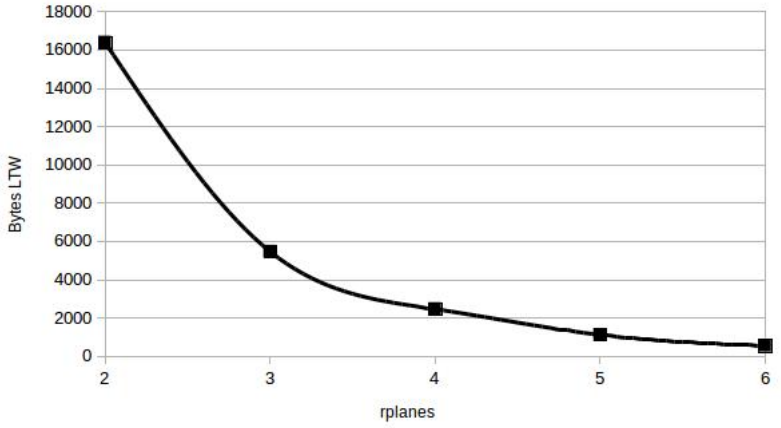


Figura 5.1: *Relación tamaño archivo comprimido LTW - rplanes*

Ahora vamos a pasar a estudiar la relacion de compresion LTW con el valor de quant. Esta relacion la tenemos en la figura 5.2, donde se muestra el tamaño del archivo comprimido LTW en el eje Y y el valor de quant en el eje X. Esta figura denota que a mayor quant menor compresion LTW.

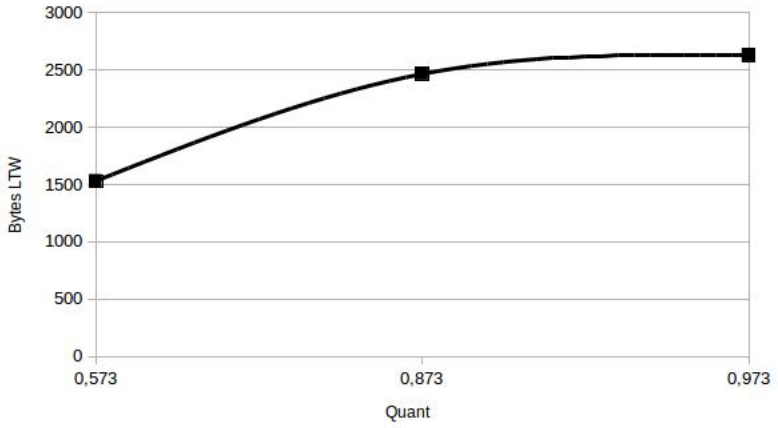


Figura 5.2: *Relación tamaño archivo comprimido LTW - quant*

Finalmente, para un quant y rplanes fijo, se va a estudiar la relacion de compresion LTW frente al nivel de descomposicion. En la Figura 5.3 se muestra el el tamaño del archivo comprimido LTW en el eje Y y los niveles de compresion en el eje X. Esta grafica refleja que a mayor nivel de compresion menor compresion LTW. Esto se debe a que cuanto mayor es el nivel de descomposicion, aás información quitamos, ya que se realiza el filtrado mas veces. Sin embargo, no existe una gran diferencia cuando se llega a valores de 5 o 6 niveles, por lo que se podria decir que serian los niveles optimos.

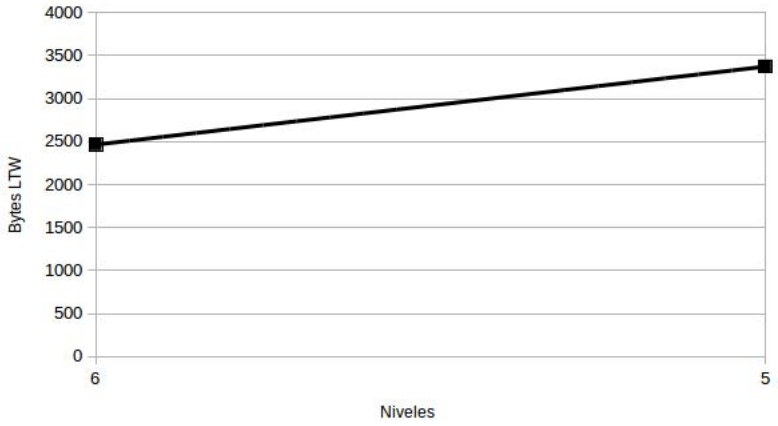
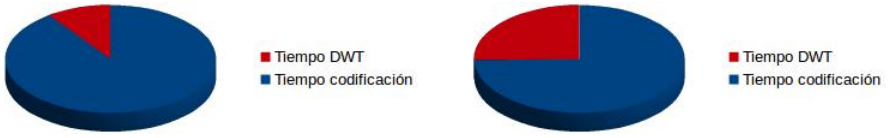


Figura 5.3: *Relación tamaño archivo comprimido LTW - niveles de compresión*

Con el estudio de compresion realizado, se va a efectuar el analisis de los tiempos de ejecucion desde distintos dispositivos. Al principio del punto anterior ya se intuyó que debido a las características de ambos dispositivos la DWT y su posterior codificación iba a

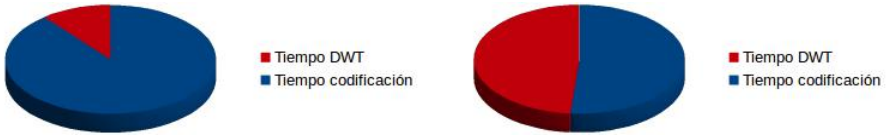
ser mas eficiente en el dispositivo Nexus 9 que en el Nexus 7.

En las figuras 5.4, 5.5 y 5.6 se presentan diversas gráficas obtenidas con los datos de las talas 5.2 y 5.3 con las que gráficamente se clarifica que el tiempo de escritura en disco es el proceso más costoso. Sin la escritura en disco la codificación LTW contaría con una gran eficiencia, pero sin él no se obtendría el fichero LTW por lo que no podemos prescindir del proceso de escritura.



(a) Niveles = 6, rplanes = 4, Quant = 0.873 (b) Niveles = 6, rplanes = 6, Quant = 0.973

Figura 5.4: *Comparación de tiempos entre la Transformada Wavelet y la codificacion en Nexus 7.*



(a) Niveles = 6, rplanes = 4, Quant = 0.873 (b) Niveles = 6, rplanes = 6, Quant = 0.973

Figura 5.5: *Comparación de tiempos entre la Transformada Wavelet y la codificacion en Nexus 9.*

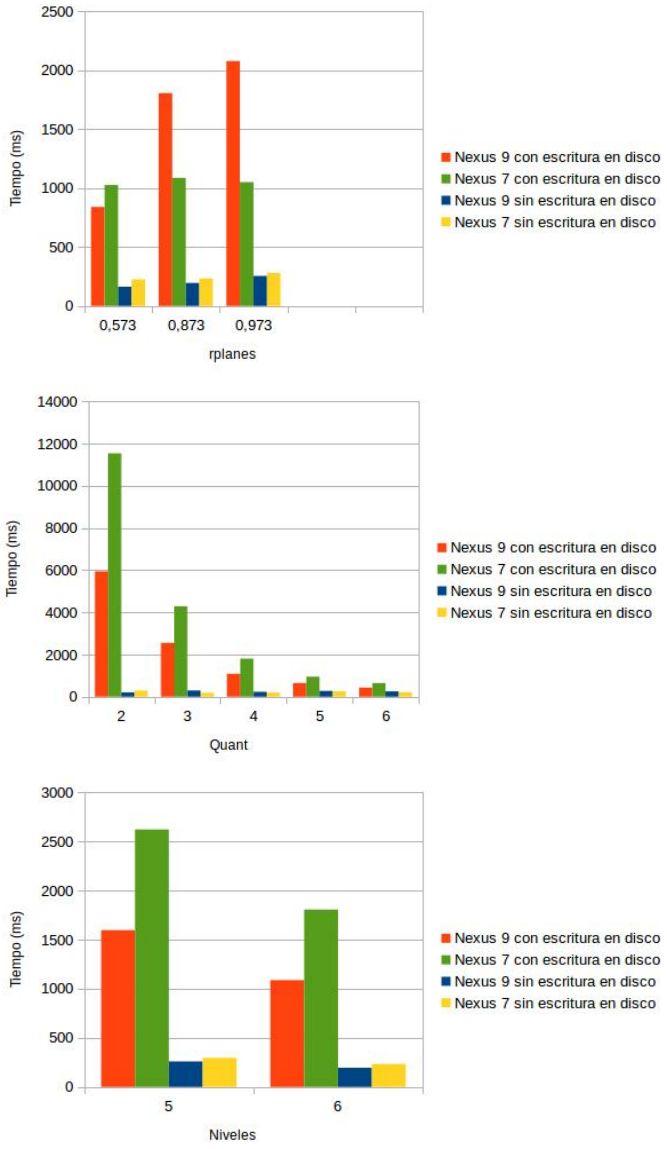


Figura 5.6: *Comparación de tiempos con y sin escritura.*

Capítulo 6

Conclusiones y líneas futuras

Contents

6.1. Conclusiones

En este último punto se va a recapitular los temas abarcados para el desarrollo del proyecto redactados en esta memoria.

En el primer capítulo se ha hecho un estudio del arte de los métodos de compresión, para así entender las ventajas de la Transformada de Wavelet, en la que se basa el codificador LTW, para la compresión de imágenes. Junto con esto, se ha enfatizado JPEG ya que es el estándar con mayor uso en la actualidad y principal competidor para el resto de codificadores.

Con todos estos conceptos ya en la mente, se ha realizado la presentación y explicación del algoritmo LTW (Lower Tree Wavelets) junto con sus algoritmos de codificación y decodificación en pseudocódigo. Con esto, se han presentado las grandes ventajas que ofrece este codificador, entre las que destaca un alto rendimiento con una buena calidad de imagen.

Conocida la dinámica de trabajo del codificador LTW, se han establecido las bases para el desarrollo de la aplicación Android, pero no sin antes hacer un breve repaso de las herramientas de desarrollo empleadas. Con esto, el primer paso ha sido el análisis de codificación LTW para la implementación de una librería Java que realice dicha codificación en nuestra aplicación.

Una vez implementada la librería para la codificación LTW, se ha profundizado en el desarrollo de la aplicación Android, descripción, clases y métodos a través de un diagrama UML. Además, se han tratado temas como permisos y requisitos de implementación y características, mejoras y proceso de configuración del paquete android.hardware.camera2, el cual nos permite la adquisición de capturas de tipo RAW.

Para finalizar, hemos testeado los resultados obtenidos en la compresión de imágenes con nuestra aplicación, con los que hemos podido concluir que las tasas de compresión del codificador LTW son mayores que las de JPEG (estándar pionero en compresión de imágenes en la actualidad) con una mejor calidad de imagen. También se ha podido concluir con el testeo de tiempos de compresión con escritura y sin escritura en disco que en el proceso de compresión la parte más costosa radica en la escritura del archivo LTW.

6.2. Líneas futuras

A partir del actual proyecto y con el resto de nuevas características aportadas por el paquete hardware.camera2 se pueden crear diversas líneas de mejora.

Por un lado, se podría permitir no sólo la captura de imágenes sino la grabación de vídeo.

Esta aplicación, también podría desarrollarse en los lenguajes nativos del resto de sistemas operativos móviles pioneros en la actualidad, como son iOS o Windows Phone.

Además, si se quisiera pensar en un ámbito más comercial y no sólo de investigación, el color sería un punto importante a tratar ya que actualmente sólo se realiza la compresión de la luminancia, dejando a un lado el color de la imagen capturada.

Sería interesante, además, incluir la parte de decodificación de manera que en el mismo dispositivo móvil se pudiera decodificar la imagen para poder visualizarla.

Apéndice A

Manual de uso de la aplicación

Este apéndice está dedicado a dar una pequeña asistencia funcional de la aplicación.

Empecemos por mostrar la apariencia de la aplicación al ejecutarla. Ésta es puede ver en la figura A.1.

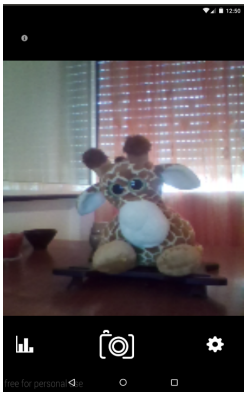


Figura A.1: *Apariencia principal de la aplicación.*

En la barra de herramientas inferior, se encuentra la siguiente botonera:

- **Botón configuración**
  Establece los parámetros de compresión LTW. Si no se indican otros, estos parámetros serán los valores por defecto ($Niveles = 6$, $rplanes = 4$ y $Quant = 0,873$). En la figura A.2 se puede ver el botón de configuración y en la figura A.3 el resultado de pulsar dicho botón.

Figura A.2: *Botón configuración.*

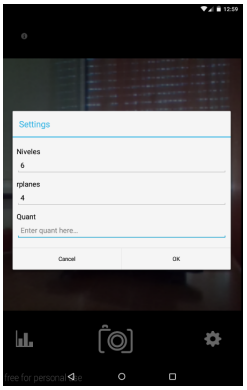


Figura A.3: *Menú de configuración de parámetros de compresión LTW.*

- **Botón comparación de compresiones**
  En este botón se hace una comparación de compresiones entre JPEG y LTW y muestra gráficamente qué estándar de compresión es el más eficiente. En la figura A.4 vemos el botón de comparación de compresiones y en la figura A.5 se muestra un ejemplo de comparación al pulsar dicho botón.

Figura A.4: *Botón comparación de compresiones.*

Figura A.5: *Comparativa gráfica entre LTW y JPEG.*

- **Botón captura**
  Al pulsar este botón se realiza una captura que es comprimida y almacenada con formato LTW. En la figura A.6 se puede ver el botón captura.

Figura A.6: *Botón captura.*

Además, en la esquina superior izquierda se encuentra un botón adicional. Se trataría del botón información que se describe a continuación:

- **Botón información**
  Al pulsar este botón se muestra una descripción del propósito de la aplicación. En la figura A.7 se puede ver el botón información y en la figura A.8 el resultado de pulsar dicho botón.

Figura A.7: *Botón información.*

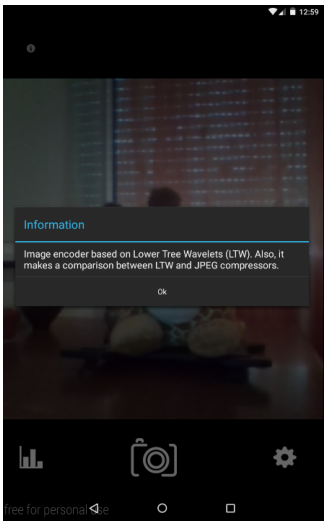


Figura A.8: *Información de la aplicación.*

Apéndice B

Conjunto de imágenes de prueba

(a) Lena

(b) jappy

Figura B.1: *Imágenes utilizadas para pruebas.*

Bibliografía

[1] Arnaiz, G.(1969). *Introducción a la Estadística Teórica.* Ed. Lex Nova,S.A.

[2] Casella, G. y Berger, R. L. (1990). *Statistical Inference.* Ed. Jenny Greenwood.

[3] Lindgren, B. W. (1993). *Statistical Theory.* Ed. Chapman and Hall/CRC.

[4] Maurandi, A. Del Río, L. y Balsalobre, C. (2013). *Fundamentos estadísticos para investigación. Introducción a R.* Ed. Bubok Publishing S.L.

[5] Mood, A. M. y Graybill,F. A. (1978). *Introducción a la Teoría de la Estadística.* Ed. Aguilar.

[6] Zoroa, P. y Zoroa, N. (2008). *Elementos de Probabilidades.* Ed. Diego Marín.