

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Compresión de imágenes utilizando el estándar JPEG con la cuantificación óptima de Lloyd-Max

Mateo Sebastian Ortiz Higuera

Universidad Nacional de Colombia
Facultad de Ciencias - Departamento de Matemáticas
Teoría de Códigos
2023

Índice general

1. Introducción	4
2. Descripción del Problema	5
3. Objetivos	6
3.1. Objetivo General	6
3.2. Objetivos Específicos	6
4. Marco Teórico	7
4.1. Digitalización de una Imagen	7
4.2. Estándar JPEG para la compresión de imágenes	8
4.2.1. Transformación del Color	9
4.2.2. Construcción de subimágenes y DCT	10
4.2.3. Cuantificación	12
4.2.4. Codificación	15
4.2.5. Decodificación	16
4.3. Cuantificación de Imágenes	17
4.3.1. Error de Cuantificación	21
4.3.2. Cuantificación Óptima (Lloyd-Max)	22
5. Implementación del algoritmo	25
5.1. Cuantificación sobre el DCT	25
5.1.1. Cuantificación Lloyd-Max	25
5.1.2. Codificación	28
5.1.3. Análisis comparativo	29
5.1.4. Ventajas y limitaciones sobre esta implementación	33

5.2.	Cuantificación sobre los bloques de imagen	34
5.2.1.	Cuantificación Lloyd-Max	34
5.2.2.	Análisis comparativo	35
5.2.3.	Ventajas y limitaciones sobre esta implementación	38
5.3.	Cuantificación sobre la imagen	39
5.3.1.	Cuantificación Lloyd-Max	39
5.3.2.	Análisis comparativo	39
5.3.3.	Ventajas y limitaciones sobre esta implementación	41
6.	Programación del algoritmo	43
6.1.	Programación de los algoritmos del estándar JPEG	43
6.2.	Programación de la primera implementación del algoritmo Lloyd-Max	46
6.3.	Programación de la segunda implementación del algoritmo Lloyd-Max	48
6.4.	Programación de la tercera implementación del algoritmo Lloyd-Max	50
6.5.	Observaciones Finales	51
7.	Conclusiones	52

1 | Introducción

En este proyecto se realizará una descripción del popular estándar JPEG y su algoritmo de cuantificación convencional. Posteriormente, se presentará el algoritmo óptimo para la cuantificación y se explicará cómo se puede implementar en el estándar JPEG. Se llevarán a cabo diversas pruebas que se centrarán principalmente en el nivel de compresión de imagen, entendido como la relación entre la calidad de la imagen y el tamaño de información que ocupa en memoria.

Para respaldar estos análisis, se utilizarán varios programas desarrollados en Octave. Estos programas mostrarán paso a paso la implementación del código basado en los diferentes algoritmos explicados para el estándar JPEG.

2 | Descripción del Problema

En este proyecto se abordará la primera versión del estándar JPEG, que es ampliamente conocido y utilizado para la codificación de imágenes digitales. Sin embargo, debido a su simplicidad, una de sus características más destacadas, su algoritmo de cuantificación uniforme, conlleva a diversas implicaciones, entre las cuales se encuentra el aumento del error de cuantificación, lo que provoca distorsión en la imagen resultante. No obstante, es posible optimizar este algoritmo de cuantificación con el fin de minimizar el error en relación a la cantidad de información perdida, utilizando el enfoque de Lloyd-Max. De esta manera, se puede obtener imágenes con una menor distorsión y aún conservar un nivel de compresión bastante bueno.

3 | Objetivos

3.1. Objetivo General

- Implementar la cuantificación óptima de Lloyd-Max dentro del estándar de codificación de imágenes JPEG.

3.2. Objetivos Específicos

- Explicar el estándar JPEG y su algoritmo de compresión con pérdida de información.
- Explicar el algoritmo de Lloyd-Max para cuantificación óptima en el contexto de compresión de una imagen.
- Comparar los métodos de cuantificación uniforme con los métodos de cuantificación no uniforme en el proceso de compresión de una imagen.

4 | Marco Teórico

4.1. Digitalización de una Imagen

Para hablar de la digitalización de una imagen, debemos definir unos cuantos términos:

Definición:

1. Una imagen es una matriz de tres dimensiones $M_{m,n,c}$, en donde m representa el ancho de la imagen, n el largo y c el número de canales.
2. Un elemento p con índices i, j de una imagen lo llamaremos *píxel*, el cual representa la unidad mínima de color sobre enteros de 8 bits, lo cual corresponde a valores en el intervalo $[0, 255]$.
3. Un canal de una imagen es un componente C de una imagen, el cual contiene la información codificada de un tono. Si existe más de un canal en una imagen, se dice que es una imagen a color.¹ Por el contrario, si la imagen solo tiene un canal, se dice que la imagen es en escala de grises.



Figura 4.1: Imagen en escala de grises

¹Habitualmente se utiliza la representación RGB, en donde un canal contiene el nivel de intensidad del color rojo (RED), otro del verde (GREEN) y uno último del azul (BLUE). Cabe decir que con la superposición de cada color podemos crear prácticamente cualquier color del espectro visible para el humano.

En las imágenes a color convencionales, como las que utilizan el modelo de color RGB, los canales se superponen para generar un color determinado. Por lo tanto, en realidad, podemos observar que un píxel de una imagen a color ocupa 24 bits de información, lo que corresponde a la suma de los 8 bits de cada canal. De esta manera, una imagen puede tener hasta 16.777.216 variaciones de color.

Aunque cada canal funciona de forma independiente, es decir, en cada píxel tenemos valores que van desde 0 hasta 255, los cuales representan el nivel de intensidad del color.

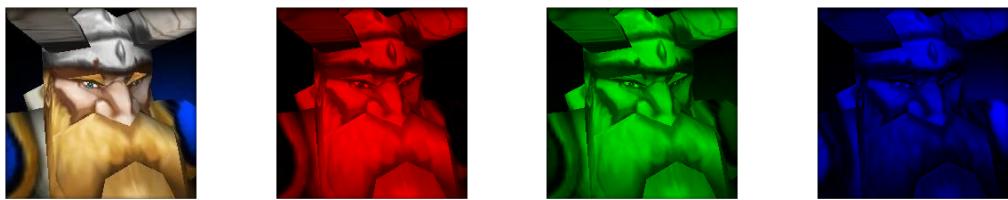


Figura 4.2: Imagen con sus respectivos canales de color RGB

De aquí podemos ver uno de los mayores problemas que aparecieron en la década de los 80, pues digitalizar una imagen ocupa demasiado espacio en memoria de un computador. Tomemos como ejemplo la imagen utilizada en la figura 4.1, la imagen tiene un tamaño de 215x215px, es decir, tenemos tres matrices de 215x215, en donde en cada índice tenemos 8 bits de información, al final tenemos $215 \times 215 \times 3 \times 8 = 1109400$ bits, es decir, 138 kilobytes, que en su tiempo sí podría representar bastante espacio de memoria dada las limitaciones de su época.

Por lo tanto, las apariciones de estándares de compresión de imágenes fue un paso bastante importante en la digitalización de imágenes, y quizás la más importante fue la llamada JPEG que explicaremos en este proyecto.

4.2. Estándar JPEG para la compresión de imágenes

Cuando hablamos de JPEG nos referimos al estándar creado por el Joint Photographic Experts Group (JPEG) para la compresión y la codificación de imágenes. Este grupo es derivado de la International Organization for Standardization (ISO) y demás agrupaciones, quienes desde 1983 había comenzado a investigar sobre la digitalización de imágenes.

En 1992, se lanzó la primera versión del estándar de codificación de imágenes fijas, formalmente llamado JPEG (ISO/IEC-10918, 1994), basado en la transformada de coseno discreta de tipo II (DCT-II). A lo largo de los años han sido lanzadas más versiones más sofisticadas, particularmente implementando algoritmos para la compresión sin pérdida de información: la versión JPEG LS (ISO/IEC-14495, 1999) y JPEG 2000 (ISO/IEC-15444, 2004).

Sin embargo, la versión más popular sigue siendo la primera, y es que su algoritmo de compresión resulta ser bastante simple y efectivo. A continuación se explicará cada uno de los pasos que realiza este estándar.

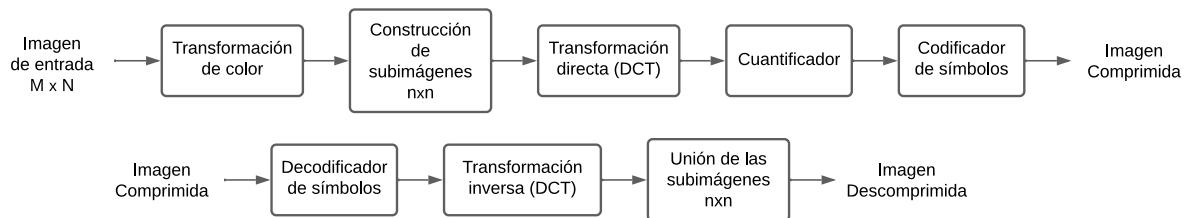


Figura 4.3: Diagrama de bloques del sistema de codificación JPEG

4.2.1. Transformación del Color

El algoritmo JPEG inicia transformando los canales de color RGB a YCbCr, en los cuales se separa la luminancia de la imagen en el canal Y y la información del color en los canales Cb y Cr.

Existen varias razones por las que se realiza este cambio de espacio de color; la principal tiene que ver con dos efectos visuales presentes en la vista del ser humano, y es que el ojo humano es más sensible al cambio de luminancia que al cambio de color y además nota con más facilidad pequeños cambios de brillo en zonas homogéneas que en zonas donde la variación es grande. Por esto, podemos atacar la compresión por separado en la cuantificación de los canales de luz y de color, pudiendo así tener más perdida de información (distorsionar más la imagen) sin tener cambios tan susceptibles para la vista.



Figura 4.4: Imagen con sus respectivos canales de colores transformados en YCbCr

Para las distintas versiones de JPEG se han implementado diversas ecuaciones para transformar los espacios de color, sin embargo, la ecuación más usada comúnmente es:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0,257 & 0,504 & 0,098 \\ -0,148 & -0,291 & 0,439 \\ 0,439 & -0,368 & -0,071 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

El algoritmo JPEG continua con los canales codificados en YCbCr.

4.2.2. Construcción de subimágenes y DCT

Después de crear los canales para la luminancia y la cromancia, el algoritmo de JPEG utiliza la Transformada Discreta de Coseno (DCT) para separar la imagen en sus componentes de frecuencia.

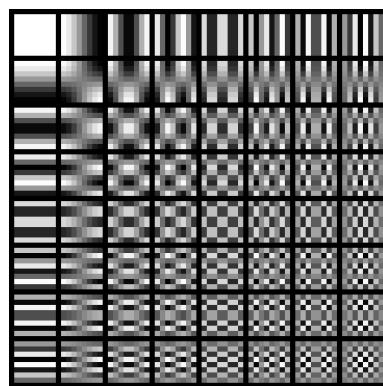


Figura 4.5: Representación visual de las componentes frecuenciales que utiliza la DCT

Para ello subdividimos la imagen en subimágenes de 8x8 y aplicamos la transformada en cada una de estas subimágenes. Hay dos observaciones para decidir el tamaño de estas subimágenes, la primera es que es computacionalmente más costoso realizar menos trans-

10

formaciones en subimágenes más grandes que hacer más transformaciones en subimágenes más pequeñas, la segunda es que si tomamos subimágenes más pequeñas vamos a tener más distorsión. Por lo tanto el tamaño de 8x8 píxeles parece ser un punto intermedio para que el algoritmo no sea tan costoso computacionalmente y al mismo tiempo no se produzca tanta distorsión de imagen.

La ecuación para cada subimagen M corresponde a

$$T_{u,v} = \frac{1}{4} \alpha(u) \alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 M_{x,y} \cos \left[\frac{\pi}{8} \left(x + \frac{1}{2} \right) u \right] \cos \left[\frac{\pi}{8} \left(y + \frac{1}{2} \right) v \right]$$

Donde

- u es la frecuencia espacial horizontal, para $0 \leq u \leq 7$.
- v es la frecuencia espacial vertical, para $0 \leq v \leq 7$.
- $\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{si } u = 0 \\ 1 & \text{en otro caso} \end{cases}$ es el factor de escala de normalización para hacer la transformación ortonormal.



Figura 4.6: Bloque 8×8 de la figura 4.1 correspondiente a los píxeles de los intervalos 85:92,85:92

Podemos ver como ejemplo la siguiente matriz que representa los valores de la figura 4.6:

$$M = \begin{bmatrix} 210 & 204 & 210 & 35 & 54 & 52 & 49 & 52 \\ 230 & 57 & 55 & 71 & 164 & 79 & 41 & 223 \\ 217 & 75 & 54 & 104 & 151 & 93 & 63 & 51 \\ 42 & 117 & 84 & 59 & 71 & 69 & 97 & 76 \\ 104 & 131 & 119 & 82 & 42 & 90 & 130 & 72 \\ 92 & 90 & 104 & 110 & 102 & 108 & 112 & 73 \\ 148 & 133 & 108 & 105 & 130 & 94 & 79 & 80 \\ 146 & 139 & 137 & 135 & 135 & 147 & 131 & 148 \end{bmatrix}$$

Aplicándole la transformada tenemos:

$$T = \begin{bmatrix} 845,625 & 116,190 & 70,658 & 42,872 & 32,875 & -18,604 & 20,657 & 36,405 \\ -41,047 & 80,773 & 78,809 & 18,661 & 58,421 & -27,511 & 41,041 & 41,245 \\ 111,223 & 70,611 & 42,880 & -2,724 & 42,613 & -35,254 & 52,486 & 23,250 \\ -26,247 & 57,194 & 13,780 & -53,947 & -92,299 & 10,096 & 3,773 & 9,557 \\ -3,374 & 29,634 & 18,931 & -78,638 & -143,625 & 0,9392 & -17,989 & -8,603 \\ -54,401 & 75,108 & -36,120 & -9,224 & -70,674 & -11,725 & -29,999 & 9,117 \\ 8,184 & 39,651 & -42,763 & 24,725 & -40,516 & 39,198 & -5,380 & -0,926 \\ 16,458 & 81,425 & 5,636 & 34,588 & -14,394 & 30,495 & -2,681 & 2,399 \end{bmatrix}$$

4.2.3. Cuantificación

El algoritmo de cuantificación del estándar JPEG es uniforme, es decir, para todo coeficiente de una subimagen 8×8 el nivel de reconstrucción es la misma. Lo que supone que el nivel de cuantificación es igual en todos los coeficientes sin importar que el valor del mismo tenga alta o baja frecuencia, lo que puede producir pérdidas en detalles de la imagen. Primero se utiliza la tabla de cuantificación de Losheller para crear una matriz de cuantización Q_1

para la luminancia:

$$Q_1 = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \quad (4.1)$$

Y la matriz de cuantización Q_2 para la cromancia:

$$Q_2 = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix} \quad (4.2)$$

Y con ellas se realiza la ecuación de cuantificación para obtener las matrices cuantificadas \hat{T}_1 y \hat{T}_2 :

$$\hat{T}_{1u,v} = \left[\frac{T_{1u,v}}{Q_{1u,v}} \right] \quad \hat{T}_{2u,v} = \left[\frac{T_{2u,v}}{Q_{2u,v}} \right]$$

Donde $T_{1u,v}$ es un coeficiente de la matriz transformada del canal de luminancia, $T_{2u,v}$ es un coeficiente la matriz transformada de los canales de cromancia, $Q_{1u,v}$ es un índice de la matriz de cuantificación para la luminancia y $Q_{2u,v}$ es un índice de la matriz de cuantificación para la cromancia.

De esta forma se cuantizan las tres matrices de forma distinta.

Continuando con el ejemplo, aplicamos la ecuación de cuantificación a todos los

coeficientes y obtenemos nuestra matriz cuantificada:

$$\hat{T} = \left[\frac{T}{Q_1} \right] = \begin{bmatrix} 53 & 11 & 7 & 3 & 1 & 0 & 0 & 1 \\ -3 & 7 & 6 & 1 & 2 & 0 & 1 & 1 \\ 8 & 5 & 3 & 0 & 1 & -1 & 1 & 0 \\ -2 & 3 & 1 & -2 & -2 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & -2 & 0 & 0 & 0 \\ -2 & 2 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

De esta forma cuantificaríamos los bloques transformados (notemos que en el ejemplo estamos cuantificando una imagen en escala de grises, por lo tanto tan solo cuantificamos con Q_1), sin embargo, existen varias implementaciones en donde se puede elegir el nivel de compresión de la imagen, y esto corresponde a multiplicar por un escalar $\lambda > 1$ la matriz de cuantificación:

$$\hat{T}'_{u,v} = \left[\frac{T_{u,v}}{\lambda Q_{u,v}} \right]$$

Si por ejemplo tomamos $\lambda = 3$ obtenemos:

$$\hat{T}' = \begin{bmatrix} 18 & 4 & 2 & 1 & 0 & 0 & 0 & 0 \\ -1 & 2 & 2 & 0 & 1 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Podemos observar así que más coeficientes tendrán un valor de 0, que en el siguiente paso representa la mayor compresión del estándar JPEG.

Como observación podemos ver que en Q_2 la mayoría de coeficientes tienen valor 99, por lo que, en general, el nivel de compresión va a ser mayor que en Q_1 ; esto aprovechando uno de los efectos en la visión del ojo humano comentados anteriormente (Sección 4.2.1.).

4.2.4. Codificación

El último paso es la codificación de símbolos, para cada matriz cuantificada se realiza una secuencia en zig-zag como se ve en la figura 4.7 y se recoge la información de cada coeficiente.

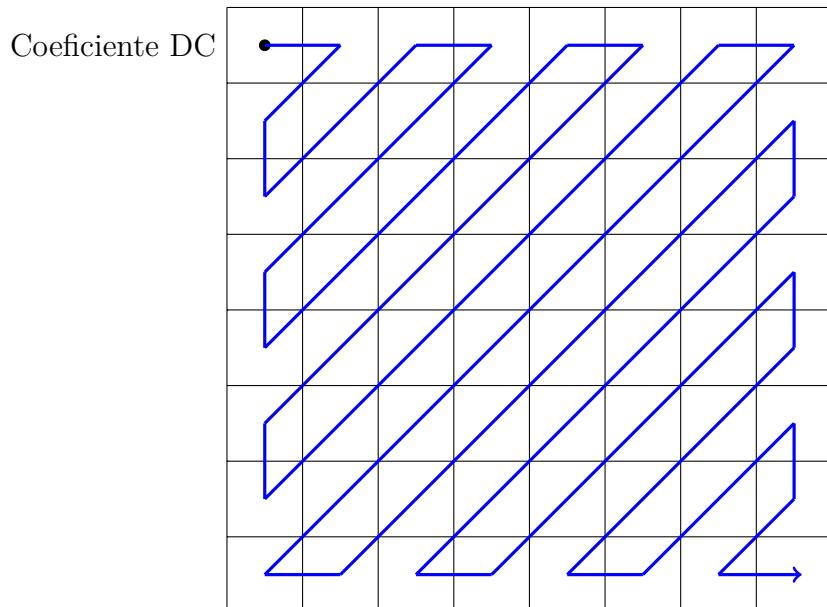


Figura 4.7: Secuencia en zig-zag de la matriz cuantificada

Primero realiza codificación de longitud de ejecución (RLE)², de esta forma se reduce una buena cantidad de coeficientes repetidos. Esta parte es fundamental y aprovecha la cuantificación realizada anteriormente, ya que hay una gran cantidad de coeficientes que se convierten a 0, especialmente en los canales de cromancia. Esta es, de hecho, la razón por la que se realiza la secuencia en zig-zag.

Así, el RLE reemplaza los valores por pares (n, m) , en donde n es el número de ceros que hay hasta el próximo valor m

²La codificación de longitud de ejecución comprime la información contando el número de símbolos contiguos de la secuencia, por ejemplo la secuencia *aaaaabbacc* se codificaría *5a2b1a2c*.

$(7, -1), (0, 0)\}$. En donde $(0, 0)$ es la señal que indica que el resto de coeficientes tienen el valor de 0.

Podemos notar que el coeficiente DC no se codificó con el RLE, y es que los coeficientes CC de cada bloque utilizan un esquema de codificación predictivo, en donde codificamos la diferencia $DIFF$ entre el valor del coeficiente DC del bloque actual y el valor del coeficiente DC del bloque anterior, lo que es:

$$DIFF = DC_i - D_{i-1}$$

Donde DC_i es el coeficiente DC del bloque 8×8 i -ésimo de la imagen.

Después de codificar todos los coeficientes en cada bloque, se realiza codificación Huffman por separado para las diferencias $DIFF$ y para el resto de pares para el resto de coeficientes.

4.2.5. Decodificación

El proceso de decodificación es el inverso al de codificación, primero decodifica con Huffman, luego devuelve los valores de cuantificación:

$$T' = \hat{T} \cdot Q = \left\lfloor \frac{T}{Q} \right\rfloor Q$$

En este paso vamos a ver pérdida de información, ya que muchos de los coeficientes serán 0, por lo que en la reconstrucción de los valores de los píxeles éstos serán distintos. La ecuación para cada índice corresponde a la transformada inversa:

$$M'_{x,y} = \frac{1}{4} \sum_{u=0}^7 \sum_{v=0}^7 \alpha(u) \alpha(v) F_{u,v} \cos \left[\frac{\pi}{8} \left(x + \frac{1}{2} \right) u \right] \cos \left[\frac{\pi}{8} \left(y + \frac{1}{2} \right) v \right]$$

A continuación se unen todas las subimágenes para reconstruir la imagen original y finalmente cambian los canales de color a RGB utilizando la siguiente ecuación:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1,164 & 0 & 1,596 \\ 1,164 & -0,391 & -0,813 \\ 1,164 & 2,018 & 0 \end{bmatrix} \cdot \begin{bmatrix} Y - 16 \\ Cb - 128 \\ Cr - 128 \end{bmatrix}$$

Así podemos obtener resultados como el siguiente:

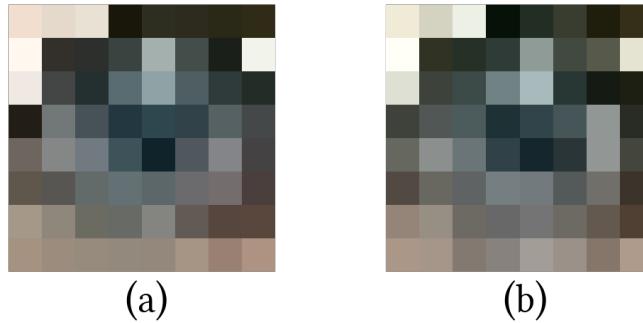


Figura 4.8: Diferencia entre la imagen original (a) y la imagen resultante del algoritmo (b)

4.3. Cuantificación de Imágenes

Definición:

Un cuantizador es una tripla $Q = \{f, q, T\}$, donde f es una señal en un intervalo $[t_1, t_L]$ para $L > 1$, T es una partición sobre el dominio de f y q una función

$$\begin{aligned} q : f &\longrightarrow f' \\ x &\longmapsto x' \end{aligned}$$

Donde $x \subset T$ y es llamado *nivel de decisión*, $x' \in [t_1, t_L]$ y es llamado *nivel de reconstrucción*, y f' es una variable discreta.

Una imagen es una señal f discreta en el intervalo $[0, 255]$, que al ser enviada a un cuantizador Q obtenemos otra imagen f' con un número reducido de valores de píxeles. En efecto, cuantizando una imagen, los píxeles con valores pertenecientes a un subconjunto x se convertirán todos a x' , el cual es un valor fijo. Así aumentamos la redundancia en los valores, sin embargo, perdemos información. Y es que como podemos notar, esta operación de mapeo no tiene inversa, por lo tanto es una operación con pérdida.

Ahora bien, veamos cómo diseñar un cuantizador uniforme, esto nos servirá, en principio, para diseñar un cuantizador óptimo:

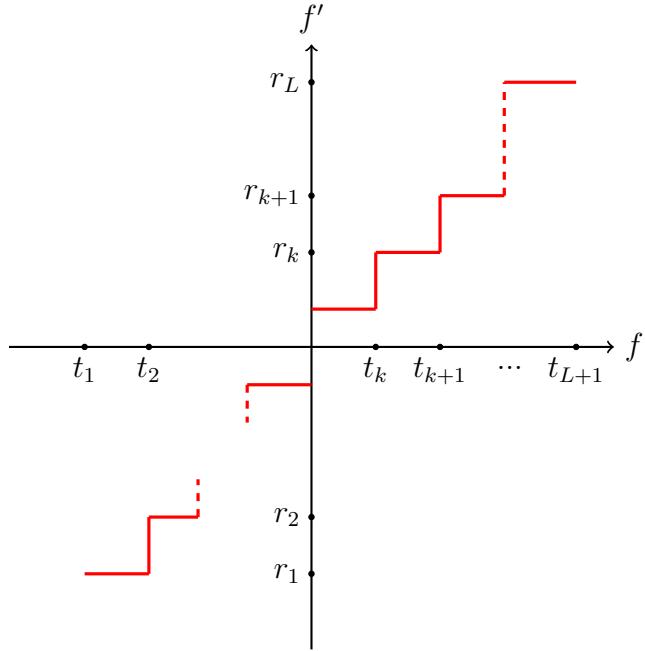


Figura 4.9: Cuantificador *Midrise*

Debemos tener la siguiente información:

- El rango de f , el cual comúnmente será $[0, 256]$ al trabajar en imágenes de 8bits.
- El número de bits (B) del cuantificador para calcular el número de niveles de reconstrucción $L = 2^B$.
- Una expresión para los niveles de decisión.
- Una expresión para los niveles de reconstrucción.

De esta forma, si asumimos una distribución de probabilidad uniforme en el intervalo $[t_1, t_{L+1}]$, podemos construir un cuantificador cuyos intervalos en los niveles de decisión $[t_1, t_k]$ sean iguales, como por ejemplo el cuantificador *Midrise* (ver figura 4.9).

Así podemos diseñar nuestro cuantificador uniforme óptimo, para ello utilizamos la distribución uniforme:

$$p_f(f) = \begin{cases} \frac{1}{t_{L+1}-t_1}, & t_1 \leq f \leq t_{L+1} \\ 0, & \text{otro caso} \end{cases}$$

Y el valor de la distancia de los intervalos de los niveles de decisión está determinado por:

$$q = \frac{t_{L+1} - t_1}{L}$$

Luego, los niveles de desición y de reconstrucción son espaciados así:

$$t_k = t_{k-1} + q, \quad r_k = t_k + \frac{q}{2}$$

Por ejemplo podríamos cuantificar la señal de un canal de una imagen f de 8bits, determinando una tabla de 2 bits para cuantificar. Sabemos que el rango de información de una imagen de 8 bits corresponde a $[t_1, t_{L+1}] = [0, 256]$, así tenemos los siguientes resultados:

$$B = 2 \quad L = 4 \quad q = \frac{256 - 0}{4} = 64 \quad t_k = t_{k-1} + 64 \quad r_k = r_{k-1} + 32$$

Y así la tabla de cuantificación sería:

Niveles de Decisión	Niveles de Reconstrucción
$t_1 = 0$	$r_1 = 32$
$t_2 = 64$	$r_2 = 96$
$t_3 = 128$	$r_3 = 160$
$t_4 = 192$	$r_4 = 224$
$t_5 = 256$	224

Por lo tanto, la función cuantificadora sería:

Lo que, tomando como ejemplo la imagen de la figura 4.1, sería:

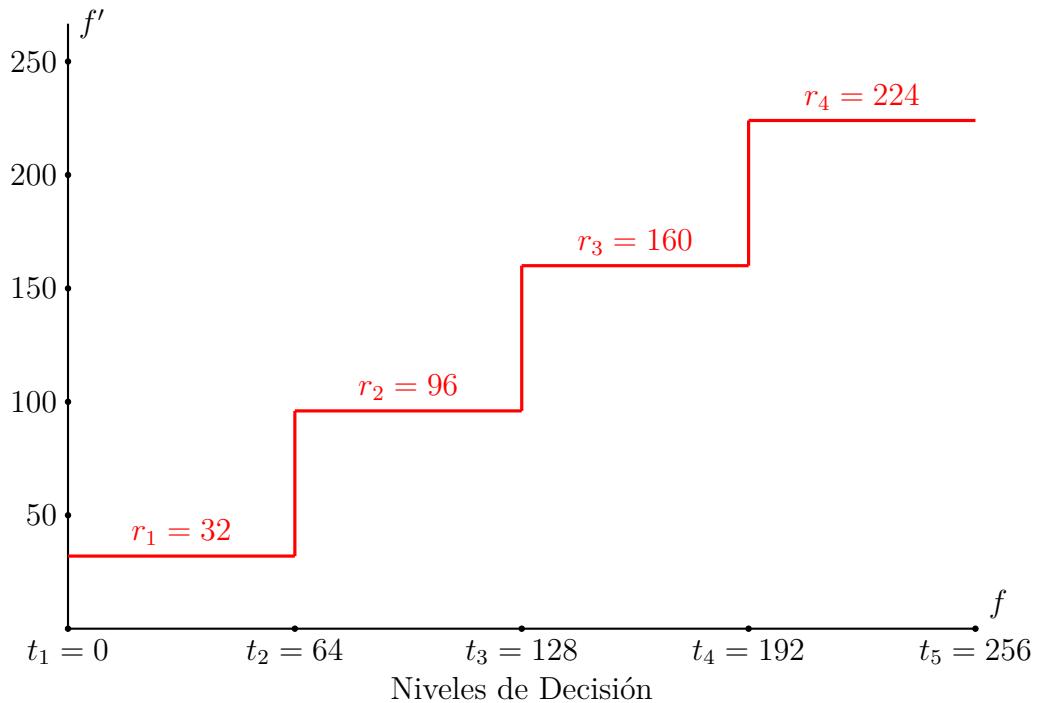


Figura 4.10: Función del cuantificador uniforme para $B = 2$, $L = 4$ en el cuantificador *midrise*

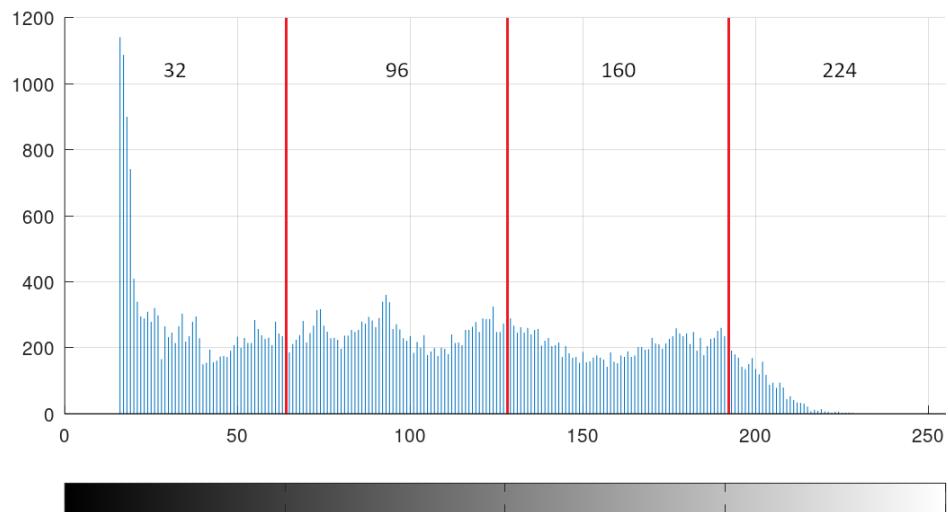


Figura 4.11: Histograma de frecuencias de la figura 4.1. Las líneas rojas denotan los límites de los niveles de decisión y los número en ellos los niveles de reconstrucción para un cuantificador de 2 bits uniforme

De aquí podemos observar que entre mayor sea el número de bits B de nuestro cuantificador <https://es.overleaf.com/projectador>, el nivel de reconstrucción va a ser mejor y por lo tanto no tendremos tanta distorsión en la imagen.

Finalmente, también podemos notar que no es una regla estricta construir un cuantizador utilizando un número de bits B , podemos definir nosotros mismos los niveles de decisión y sobre ellos definir los niveles de reconstrucción. Sin embargo, un cuantizador uniforme óptimo tiene por regla que el número de niveles de decisión L sea de la forma 2^B .

4.3.1. Error de Cuantificación

Para medir el nivel de distorsión (o de error) calculamos el error cuadrático medio (MSE por sus siglas en inglés) entre la señal no cuantificada f y la señal cuantificada f' .

$$E[(f - f')^2] = \frac{1}{M \times N} \sum_{v=0}^{M-1} \sum_{w=0}^{N-1} [(f(v, w) - f'(v, w))^2]$$

En Octave basta con introducir el comando

```
err = immse(img1,img2);
disp(err);
```

En donde `img1`, `img2` son, en este caso, las imágenes.

Tomando como ejemplo la imagen utilizada para la figura 4.11. El error o nivel de distorsión corresponde a

$$E[(f - f')^2] = \frac{1}{215 \times 215} \sum_{v=0}^{214} \sum_{w=0}^{214} [(f(v, w) - f'(v, w))^2] = 640,64$$

como se ve en la figura 4.12.

Ahora bien, como ya se ha explicado en la sección 4.2.3., en el estándar JPEG la cuantización es uniforme; para el canal de luminancia se utiliza la matriz de cuantificación (4.1) y para los canales de cromancia se utiliza la matriz de cuantificación (4.2), que, aunque es bastante mejor (en términos de pérdida de calidad de imagen) que las funciones

cuantificadas uniformemente con B bits, sigue teniendo un error considerable. En efecto:

$$\begin{aligned} E[(f - f')^2] &= \frac{1}{215 \times 215} \sum_{w=0}^{214} \sum_{w=0}^{214} [f(v, w) - f'(v, w)]^2 \\ &= 44,613 \end{aligned}$$

Donde f representa la imagen original, f' la imagen comprimida con el estándar JPEG y $v, w \in [0, 215]$ son los píxeles de cada imagen.

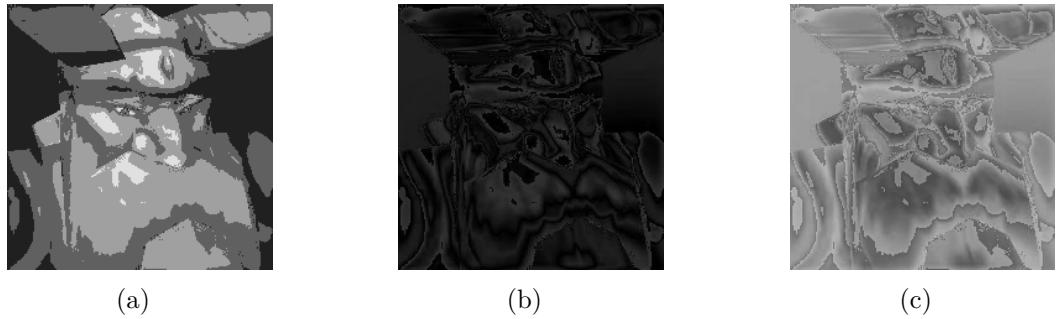


Figura 4.12: (a) Imagen de la figura 4.1 cuantificada uniformemente con 2 bits, (b) diferencia absoluta entre la imagen original y la imagen cuantificada, (c) diferencia del granulado entre la imagen original y la imagen final

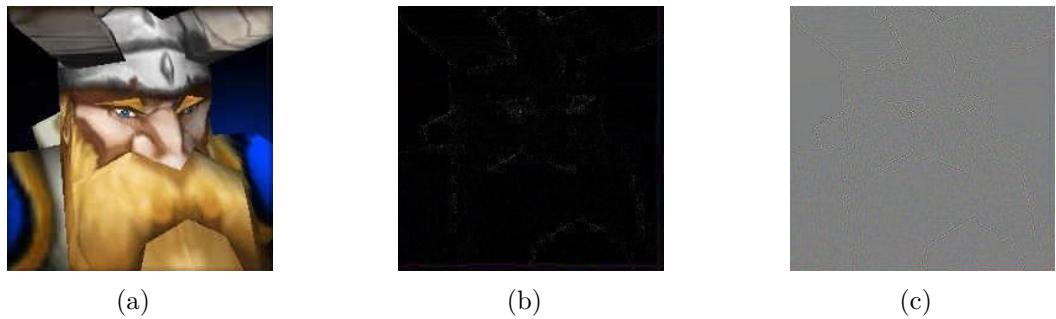


Figura 4.13: Imagen comprimida con el estándar JPEG,(b) la diferencia absoluta entre la imagen original y la imagen final,(c) diferencia del granulado entre la imagen original y la imagen final

4.3.2. Cuantificación Óptima (Lloyd-Max)

Ahora podemos preguntarnos ¿podemos reducir el error de cuantificación teniendo un nivel de decisión fijo?. Para visualizar esto podemos ver el histograma de la figura 4.11, en

ese caso estamos tomando los niveles de reconstrucción al valor que está justo en medio de cada nivel de decisión, pero esto no asegura que estamos tomando los mejores valores, de hecho, podemos ver que el último nivel de reconstrucción corresponde a 224, el cual es un valor que casi ningún píxel toma, es más, todo el intervalo de dicho nivel de decisión toma una cantidad bastante más pequeña de píxeles en comparación con los otros tres (esto se puede notar en la figura 4.12 (a) en donde hay muy pocos píxeles con el valor más bajo). De esta forma podemos ver que sí se puede reducir el error de cuantificación, ya que la concurrencia de un valor (la probabilidad de que un píxel tome dicho valor) en la mayoría de los casos será distinto del resto (como se evidencia en la figura 4.11 y en la figura 4.12, en donde hay mayor píxeles que tienden a tener valores cercanos al 0).

Por lo tanto, podemos preguntarnos:

¿Cómo encontrar un cuantificador con valores de reconstrucción tal que minimice el error para una señal f con una función de probabilidad $p_f(f)$? Esto es:

$$MSE = E[(f - \hat{f})^2] \rightarrow \min$$

Es así como Lloyd y Max presentan su solución, construida con el siguiente algoritmo:

Primer paso: Elegimos un conjunto de r_k niveles de reconstrucción para la una señal f con L niveles de decisión, donde $k = 1, 2, 3, \dots, L - 1$.

Segundo paso: Calculamos los niveles de decisión:

$$t_k = \frac{1}{2}(r_{k-1} + r_k) \quad k = 1, 2, 3, \dots, L - 1$$

Tercer paso: Calculamos los nuevos niveles de reconstrucción:

$$r_k = \frac{\int_{t_k}^{t_{k+1}} f \cdot p_f(f) df}{\int_{t_k}^{t_{k+1}} p_f(f) df} \quad k = 1, 2, 3, \dots, L - 1$$

Cuarto paso: Repetir los pasos **2** y **3** hasta que ya no halla más reducción de distorsión.

Para $p_f(f)$ es común usar la distribución Gaussiana:

$$p_f(f) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(f-\mu)^2}{2\sigma^2}}$$

De esta forma, utilizando un número de niveles de decisión podemos obtener los mejores resultados; un mejor nivel de compresión sin tener un gran nivel de distorsión.

Siguiendo con el ejemplo, si realizamos este algoritmo para la figura 4.1, tenemos el siguiente nivel de error:

$$E[(f - f')^2] = \frac{1}{215 \times 215} \sum_{v=0}^{214} \sum_{w=0}^{214} [(f(v, w) - f'(v, w))^2] = 256,43$$

El cual es más de la mitad que el obtenido realizando cuantificación uniforme.

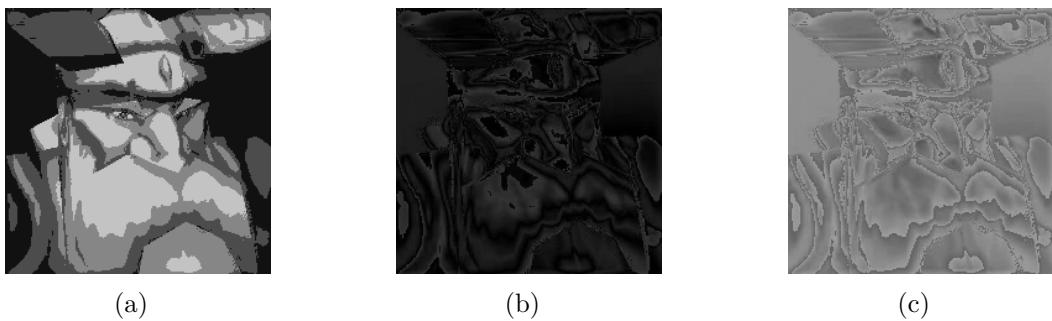


Figura 4.14: (a) Imagen cuantificada con el algoritmo Lloyd-Max con 4 niveles de decisión, (b) diferencia absoluta entre la imagen original y la imagen cuantificada, (c) diferencia de granulado entre la imagen original y la imagen cuantificada

Finalmente, gracias a la figura 4.14 podemos notar que no tan solo los valores de los colores cambian (los niveles de reconstrucción), si no que también la cantidad de píxeles que toman esos valores (los niveles de decisión) ya que, como se dijo al inicio de esta sección, en este ejemplo la cantidad de píxeles que entraban en el último nivel de reconstrucción eran bastante menos que el resto de píxeles en los otros niveles de reconstrucción, de esta forma se ha “balanceado” el número de píxeles en por cada nivel de decisión.

5 | Implementación del algoritmo

Se han propuesto tres implementaciones del algoritmo de Lloyd-Max para la compresión de imágenes, cada una satisfaciendo alguna limitación que pudiese tener la implementación inmediatamente anterior.

5.1. Cuantificación sobre el DCT

5.1.1. Cuantificación Lloyd-Max

Antes de elegir los niveles de reconstrucción para iniciar con el algoritmo de cuantización óptima debemos notar que, a diferencia de los ejemplos de la sección (4.3.2), aquí estamos cuantificando sobre los bloques transformados, por lo tanto, no tenemos un intervalo fijo sobre el cual cuantificar las matrices, como lo era en los ejemplos de la sección (4.3), cuyo intervalo a particionar correspondía a $[0, 255]$. Por lo tanto, el intervalo a particionar para cada bloque estará dado entre la parte entera del coeficiente más pequeño y la parte entera del coeficiente más grande de la matriz (excluyendo al coeficiente DC que no se cuantificará).

Retomando así con el ejemplo de la sección (4.2), tenemos el siguiente bloque transformado:

$$T = \begin{bmatrix} 845,625 & 116,190 & 70,658 & 42,872 & 32,875 & -18,604 & 20,657 & 36,405 \\ -41,047 & 80,773 & 78,809 & 18,661 & 58,421 & -27,511 & 41,041 & 41,245 \\ 111,223 & 70,611 & 42,880 & -2,724 & 42,613 & -35,254 & 52,486 & 23,250 \\ -26,247 & 57,194 & 13,780 & -53,947 & -92,299 & 10,096 & 3,773 & 9,557 \\ -3,374 & 29,634 & 18,931 & -78,638 & -143,625 & 0,9392 & -17,989 & -8,603 \\ -54,401 & 75,108 & -36,120 & -9,224 & -70,674 & -11,725 & -29,999 & 9,117 \\ 8,184 & 39,651 & -42,763 & 24,725 & -40,516 & 39,198 & -5,380 & -0,926 \\ 16,458 & 81,425 & 5,636 & 34,588 & -14,394 & 30,495 & -2,681 & 2,399 \end{bmatrix}$$

Para este caso se guardaría el valor del coeficiente DC = 845,625 y los otros 63 coeficientes se cuantificarán entre [-144, 117].

Ahora bien, para aprovechar que tenemos un canal con la información de la luminancia y dos de la cromancia, podemos diseñar cuantificadores con distinto número de bits (B); para cuantificar la matriz de la luminancia podemos usar más bits para tomar mayor número de niveles de reconstrucción, mientras que para las otras dos podemos usar menos y así comprimir más.

Después de varias pruebas, se ha decidido diseñar la matriz de cuantización con 5 bits, así tenemos los siguientes datos:

$$B = 5 \quad L = 32 \quad q = \frac{\lceil \max T \rceil - \lfloor \min T \rfloor}{32}$$

donde T es la matriz transformada y $\max T$ el coeficiente con el número más grande (exceptuando el coeficiente DC) y $\min T$ el coeficiente con el número más pequeño. A continuación creamos una tabla de cuantificación uniforme, tomando los niveles de decisión y de reconstrucción así

$$t_k = t_{k-1} + q \quad r_k = t_k + \frac{q}{2}$$

y ejecutamos el algoritmo de Lloyd-Max iniciando desde el tercer paso. Adicionalmente, elegimos la parte entera del nivel de reconstrucción final, para que después, al codificar con Huffman, poder tener concurrencia entre los números (algo que con números decimales es muy difícil de tener). Lo que es

$$r_k = \lfloor r_k \rfloor \quad \text{para todo } k = 1, 2, \dots, 32$$

Siguiendo con el ejemplo de la matriz T , tendríamos

$$B = 5 \quad L = 32 \quad q = \frac{|117 - (-144)|}{32} = \frac{261}{32} = 8,156$$

Para ahorrar espacio y tiempo describiendo la tabla de cuantificación inicial y todo el proceso del algoritmo, a continuación tan solo se mostrará la matriz resultante \hat{T} :

$$\hat{T} = \begin{bmatrix} 845,625 & 116 & 70 & 41 & 33 & -22 & 21 & 33 \\ -42 & 79 & 79 & 21 & 58 & -32 & 41 & 41 \\ 111 & 71 & 41 & -4 & 41 & -32 & 52 & 21 \\ -22 & 58 & 12 & -54 & -92 & 12 & 5 & 12 \\ -4 & 33 & 21 & -79 & -143 & 5 & -15 & -4 \\ -54 & 79 & -32 & -4 & -71 & -15 & -32 & 5 \\ 5 & 41 & -42 & 21 & -42 & 41 & -4 & -4 \\ 12 & 79 & 5 & 33 & -15 & 33 & -4 & 5 \end{bmatrix}$$

Ahora bien, para los otros dos canales, diseñaremos un cuantificador de 4 bits, siguiendo de forma similar los pasos para el cuantificador para el canal de luminancia:

$$B = 4 \quad L = 16 \quad q = \frac{\lceil \max T \rceil - \lfloor \min T \rfloor}{16}$$

donde T es la matriz transformada y $\max T$ el coeficiente con el número más grande (exceptuando el coeficiente DC) y $\min T$ el coeficiente con el número más pequeño. Después seguimos exactamente los mismos pasos que tomamos en el cuantificador para el canal de luminancia: creamos una tabla de cuantificación uniforme, tomando los niveles de decisión y de reconstrucción así

$$t_k = t_{k-1} + q \quad r_k = t_k + \frac{q}{2}$$

y ejecutamos el algoritmo de Lloyd-Max iniciando desde el tercer paso. Finalmente elegimos la parte entera del nivel de reconstrucción final, para que después codificar con Huffman. Lo que es

$$r_k = \lfloor r_k \rfloor \quad \text{para todo } k = 1, 2, \dots, 16$$

Es importante destacar que si reducimos el número de bits, tendríamos una mayor redundancia entre los valores del bloque, lo que permitiría comprimir más información. Sin embargo, esto resultaría en una mayor distorsión de la imagen. En mi opinión personal,

recomendaría reducir hasta alcanzar 14 niveles de decisión para la matriz de cuantificación de luminancia y hasta 6 niveles de decisión para la matriz de cuantificación de cromancia. Reducir aún más los niveles de decisión nos acercaría a niveles de distorsión similares a los generados por el estándar JPEG.

Siguiendo con el ejemplo, la figura 5.1 muestra el bloque de la imagen con esta implementación comparado con el bloque original y el bloque con el estándar JPEG.

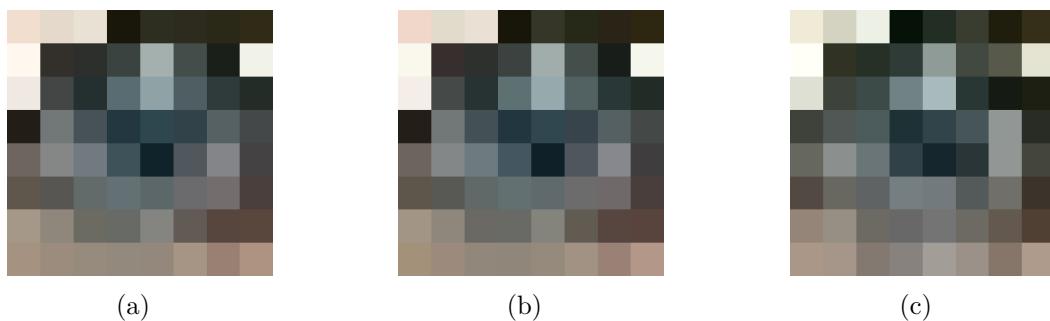


Figura 5.1: Diferencia entre la imagen original (a), la imagen resultante de la implementación (b) y la imagen resultante del estándar JPEG

5.1.2. Codificación

Para la codificación de los símbolos debemos notar que la codificación RLE aplicada en JPEG está basada en el número de coeficientes que se convierten en 0. Por lo tanto además de que no tendría sentido utilizar los pares (n, m) que utilizábamos en el estándar, tampoco lo tiene hacer un rastreo en zig-zag, pues no se genera ningún patrón particular.

Para esta implementación se realiza codificación predictiva como se hacía con los coeficientes DC en el estándar JPEG

$$\text{Diff}_{u,v} = T_{u,v} - T_{u-1,v}$$

donde $0 \leq u \leq n - 1$ y $0 \leq v \leq m$, y n es el ancho de la imagen y m la altura de la misma. De esta forma podemos tener más redundancia en los valores.

Finalmente, estos valores se codificarán utilizando árboles de Huffmann, para así obtener la longitud promedio de palabra mínima.

En el ejemplo tendríamos:

$$\text{Diff} = \begin{bmatrix} 729 & 46 & 29 & 8 & 58 & -43 & -12 & 33 \\ -121 & 0 & 58 & -37 & 90 & -73 & 0 & 41 \\ 40 & 30 & 45 & -45 & 9 & -84 & 31 & 21 \\ -80 & 46 & 66 & 38 & -104 & 7 & -7 & 12 \\ -37 & 12 & 100 & 64 & -148 & -20 & -11 & -4 \\ -133 & 47 & -28 & 67 & -56 & 17 & -37 & 5 \\ -36 & 83 & -63 & 63 & -83 & 45 & 0 & -4 \\ -67 & 74 & -28 & 48 & -48 & 37 & 1 & 5 \end{bmatrix}$$

De esta forma se leería la información por filas y así se codificarían las imágenes.

5.1.3. Análisis comparativo

En esta sección vamos a analizar los resultados obtenidos utilizando esta implementación con la imagen original y el estándar JPEG.

Podemos empezar para el bloque que hemos tomado de ejemplo en esta sección (véase figura 5.1), podemos calcular el error de cuantificación y también hallar las diferencias absolutas:

$$E[(M - \hat{M})^2] = \frac{1}{8 \times 8} \sum_{v=0}^7 \sum_{w=0}^7 [(M(v, w) - \hat{M}(v, w))^2] = 9,2083$$

Este resultado se puede comprobar observando la figura 5.3, en donde podemos comprobar que el error cuadrático medio de esta implementación, es bastante menor que el obtenido en el estándar.

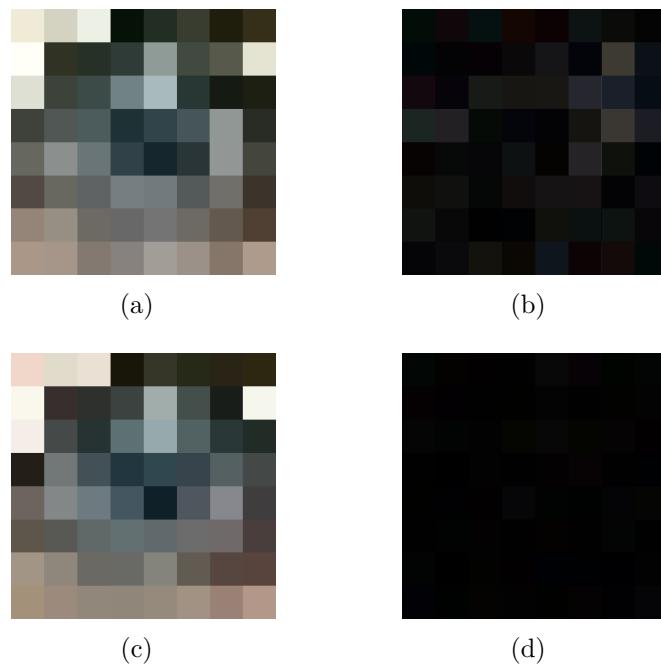


Figura 5.2: (a) Imagen resultante del estándar JPEG, (b) diferencia absoluta con la imagen original, (c) Imagen resultante de la implementación y (d) diferencia absoluta con la imagen original

Si tomamos la imagen completa tenemos los siguientes resultados:

$$E[(M - \hat{M})^2] = \frac{1}{215 \times 215} \sum_{v=0}^{214} \sum_{w=0}^{214} [(M(v, w) - \hat{M}(v, w))^2] = 3,3885$$

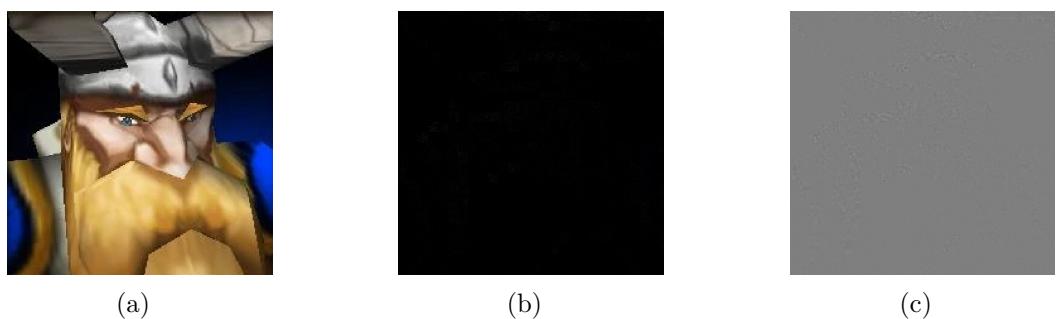


Figura 5.3: (a) Imagen resultante del estándar JPEG, (b) diferencia absoluta con la imagen original, (c) Imagen resultante de la implementación y (d) diferencia absoluta con la imagen original

Hay que notar que en la implementación, la imagen resultante presenta un pequeño escalamiento con respecto a la imagen original. Por tal motivo, para lograr hacer la diferencia absoluta entre las imágenes, se realiza un escalamiento a la imagen original.

Otros resultados que tenemos



Figura 5.4: Segunda imagen tomada para analizar el algoritmo

Con el estándar JPEG tenemos

$$E[(M - \hat{M})^2] = \frac{1}{1920 \times 1080} \sum_{v=0}^{1919} \sum_{w=0}^{1079} [(M(v, w) - \hat{M}(v, w))^2] = 6,6625$$

Mientras que con la implementación propuesta tenemos

$$E[(M - \hat{M})^2] = \frac{1}{1920 \times 1080} \sum_{v=0}^{1919} \sum_{w=0}^{1079} [(M(v, w) - \hat{M}(v, w))^2] = 0,4386$$

Dichos resultados se pueden observar en la figura 5.5

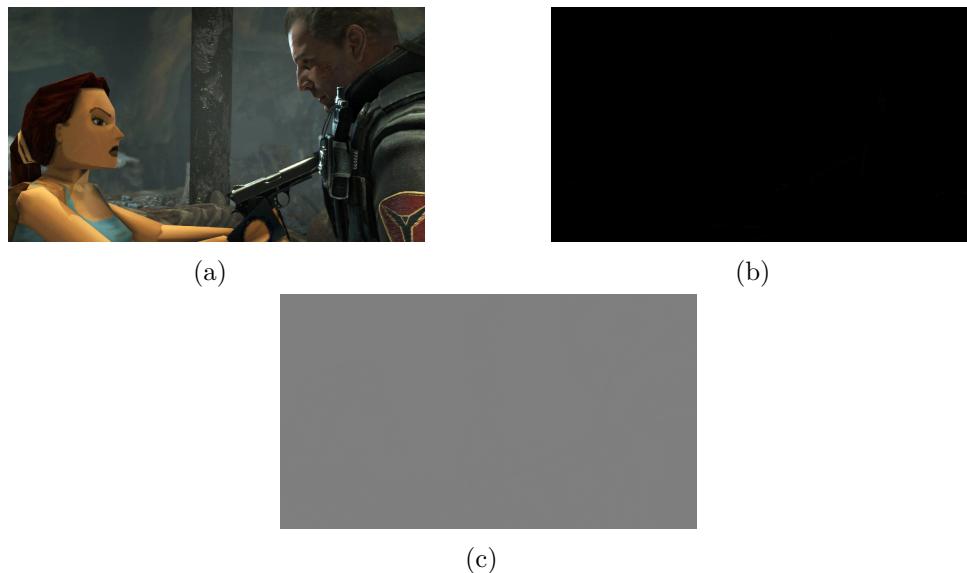


Figura 5.5: (a) Imagen resultante de la implementación, (b) diferencia absoluta con la imagen original y (c) diferencia de granulado con la imagen original



Figura 5.6: Segunda imagen tomada para analizar el algoritmo

Con el estándar JPEG tenemos

$$E[(M - \hat{M})^2] = \frac{1}{600 \times 636} \sum_{v=0}^{599} \sum_{w=0}^{635} [(M(v, w) - \hat{M}(v, w))^2] = 57,679$$

Mientras que con la implementación propuesta tenemos

$$E[(M - \hat{M})^2] = \frac{1}{600 \times 636} \sum_{v=0}^{599} \sum_{w=0}^{635} [(M(v, w) - \hat{M}(v, w))^2] = 2,7276$$

Dichos resultados se pueden observar en la figura 5.5

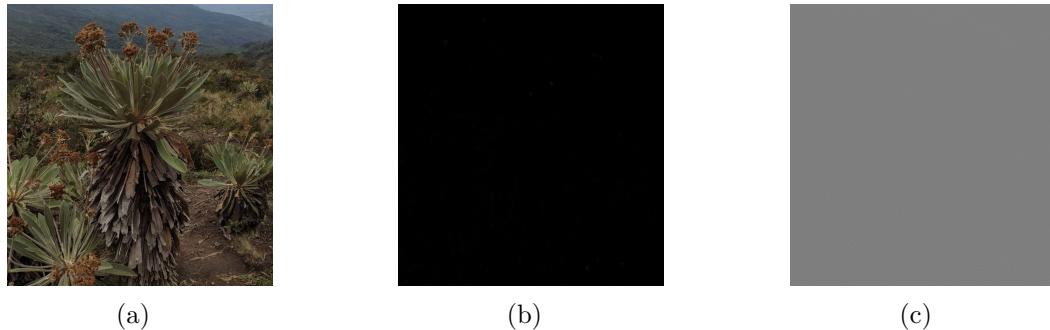


Figura 5.7: (a) Imagen resultante de la implementación, (b) diferencia absoluta con la imagen original y diferencia de granulado con la imagen original

Computacionalmente se ve una gran diferencia, en la tabla 5.1 se puede ver los tiempos que tardaron los ejemplos anteriores

Imagen	Tamaño	TE JPEG (s)	TE Implementación (s)
	[215 × 215] 46225px	04,406s	69,402s
	[1920 × 1080] 2073600px	139,265s	3028,881s
	[600 × 636] 381600px	28,137s	196,272s

Cuadro 5.1: Comparativa de Tiempo de Ejecución entre el estándar JPEG y la implementación con las imágenes anteriores

5.1.4. Ventajas y limitaciones sobre esta implementación

A pesar de que el objetivo del algoritmo de Lloyd-Max sea reducir el error de cuantización y lograr una imagen con menos distorsión, es importante tener en cuenta las siguientes ventajas y desventajas asociadas al mismo:

- El nivel de distorsión de la imagen es bastante pequeño.

- Las tablas de cuantización obtenidas y la codificación propuesta comprimen bastante menos que el estándar.
- Dado que se realiza la DCT y el algoritmo de Lloyd-Max para cada uno de los bloques de la imagen, el costo computacional es bastante más grande que el utilizado en el estándar JPEG.

5.2. Cuantificación sobre los bloques de imagen

En esta implementación omitimos el paso de la DCT y pasamos a cuantificar los bloques con sus píxeles.

5.2.1. Cuantificación Lloyd-Max

Dado que no transformamos los bloques, podríamos tomar el intervalo $[0, 255]$ para cuantificar las subimágenes, sin embargo, es mejor utilizar el intervalo entre el píxel con menor valor y píxel con mayor valor.

Tomando el ejemplo anterior tenemos la matriz M asociada a la subimagen descrita en la figura 4.6.

$$M = \begin{bmatrix} 210 & 204 & 210 & 35 & 54 & 52 & 49 & 52 \\ 230 & 57 & 55 & 71 & 164 & 79 & 41 & 223 \\ 217 & 75 & 54 & 104 & 151 & 93 & 63 & 51 \\ 42 & 117 & 84 & 59 & 71 & 69 & 97 & 76 \\ 104 & 131 & 119 & 82 & 42 & 90 & 130 & 72 \\ 92 & 90 & 104 & 110 & 102 & 108 & 112 & 73 \\ 148 & 133 & 108 & 105 & 130 & 94 & 79 & 80 \\ 146 & 139 & 137 & 135 & 135 & 147 & 131 & 148 \end{bmatrix}$$

De igual forma como lo hemos hecho en la implementación anterior, vamos a diseñar dos cuantificadores con distinto número de bits, uno correspondiente para el canal de luminancia y el otro para los dos de cromancia.

Siguiendo el algoritmo tenemos para el ejemplo

$$M' = \begin{bmatrix} 210 & 204 & 210 & 35 & 54 & 54 & 49 & 54 \\ 230 & 54 & 54 & 73 & 164 & 80 & 42 & 223 \\ 217 & 73 & 54 & 105 & 148 & 91 & 61 & 54 \\ 42 & 118 & 84 & 61 & 73 & 69 & 96 & 80 \\ 105 & 130 & 118 & 80 & 42 & 91 & 130 & 73 \\ 91 & 91 & 105 & 110 & 105 & 110 & 110 & 73 \\ 148 & 135 & 105 & 105 & 130 & 96 & 80 & 80 \\ 142 & 142 & 135 & 135 & 135 & 148 & 130 & 148 \end{bmatrix}$$

En este caso, se han elegido 8 los niveles de decisión para la matriz de cuantificación de luminancia y 4 los niveles de decisión para la matriz de cromancia. Ya que el número de píxeles es de 64 valores, luego tomar 32 valores, como se hacía en la implementación anterior, no aumentaría la redundancia de los valores por bloque.

Siguiendo con el ejemplo, la figura 5.8 muestra el bloque de la imagen con esta implementación comparado con el bloque original y el bloque estándar JPEG.

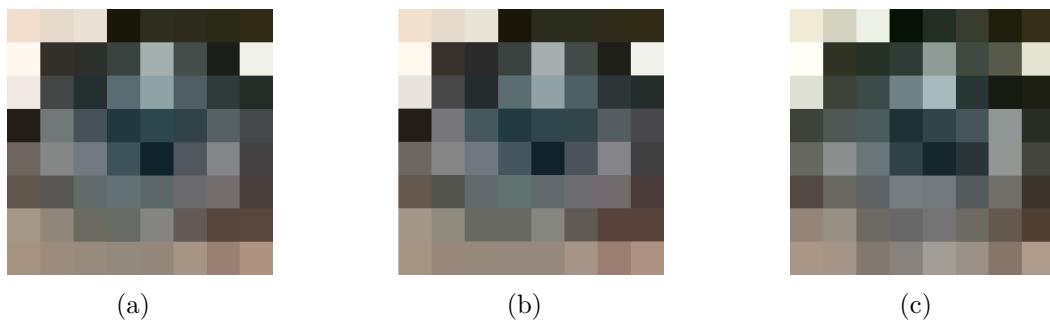


Figura 5.8: Diferencia entre la imagen original (a), la imagen resultante de la implementación (b) y la imagen resultante del estándar JPEG (c)

5.2.2. Análisis comparativo

Igual que en la implementación anterior, vamos a analizar los resultados obtenidos utilizando esta implementación con la imagen original y el estándar JPEG. Empecemos con el bloque que hemos tomado en esta sección (véase figura 5.8), podemos calcular el error de

cuantificación y también hallar las diferencias absolutas:

$$E[(M - \hat{M})^2] = \frac{1}{8 \times 8} \sum_{v=0}^7 \sum_{w=0}^7 [(M(v, w) - \hat{M}(v, w))^2] = 6,8249$$

Este resultado se puede comprobar observando la figura 5.3, en donde podemos comprobar que el error cuadrático medio de esta implementación, es bastante menor que el obtenido en el estándar.

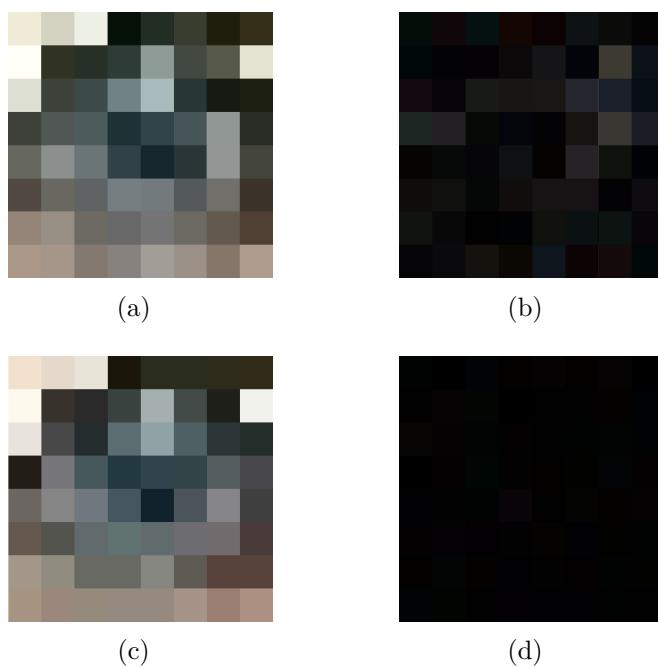


Figura 5.9: (a) Imagen resultante de la implementación, (b) diferencia absoluta con la imagen original y (c) diferencia de granulado con la imagen original

Si tomamos la imagen completa tenemos los siguientes resultados:

$$E[(M - \hat{M})^2] = \frac{1}{215 \times 215} \sum_{v=0}^{214} \sum_{w=0}^{214} [(M(v, w) - \hat{M}(v, w))^2] = 16,840$$

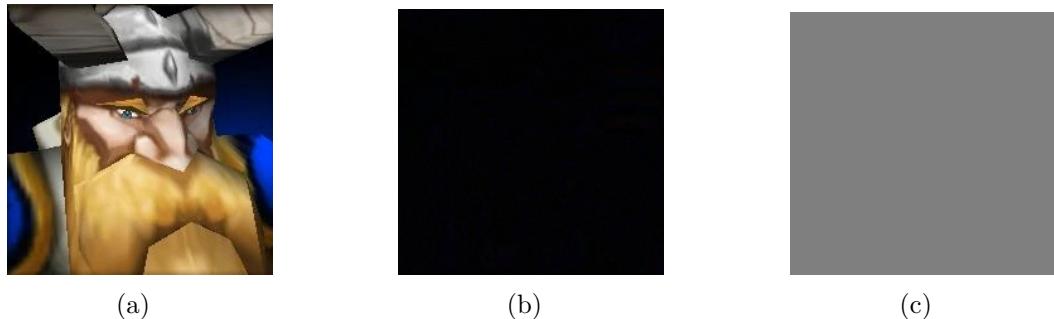


Figura 5.10: (a) Imagen resultante de la implementación, (b) diferencia absoluta con la imagen original y (c) diferencia de granulado con la imagen original

Con los otros dos ejemplos tenemos

Para la imagen de la figura 5.4 con esta implementación tenemos

$$E[(M - \hat{M})^2] = \frac{1}{1920 \times 1080} \sum_{v=0}^{1919} \sum_{w=0}^{1079} [(M(v, w) - \hat{M}(v, w))^2] = 0,4449$$

Dichos resultados se pueden observar en la figura 5.11

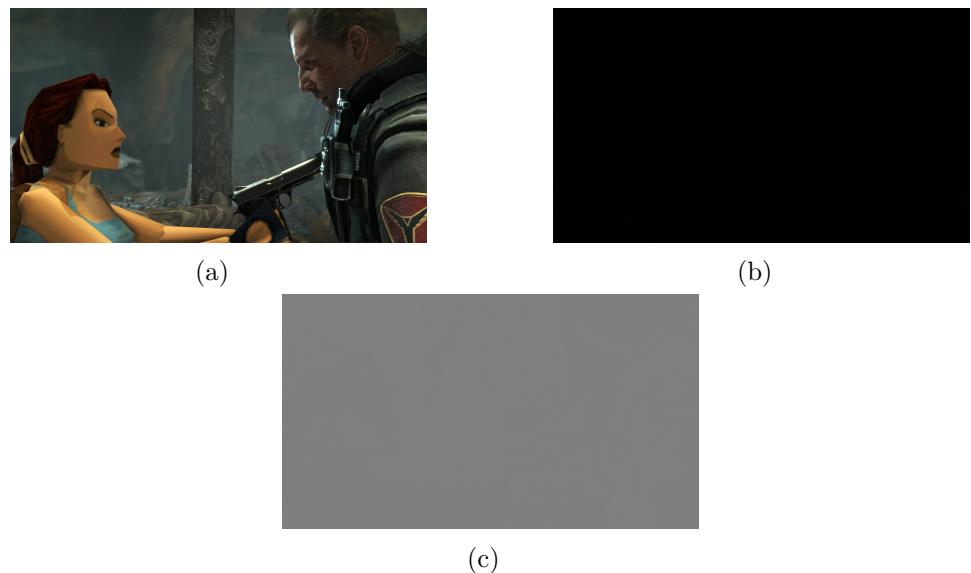


Figura 5.11: (a) Imagen resultante del estándar JPEG, (b) diferencia absoluta con la imagen original, (c) Imagen resultante de la implementación y (d) diferencia absoluta con la imagen original

Para la imagen de la figura 5.6 con esta implementación tenemos

$$E[(M - \hat{M})^2] = \frac{1}{600 \times 636} \sum_{v=0}^{599} \sum_{w=0}^{635} [(M(v, w) - \hat{M}(v, w))^2] = 2,7276$$

Dichos resultados se pueden observar en la figura 5.12

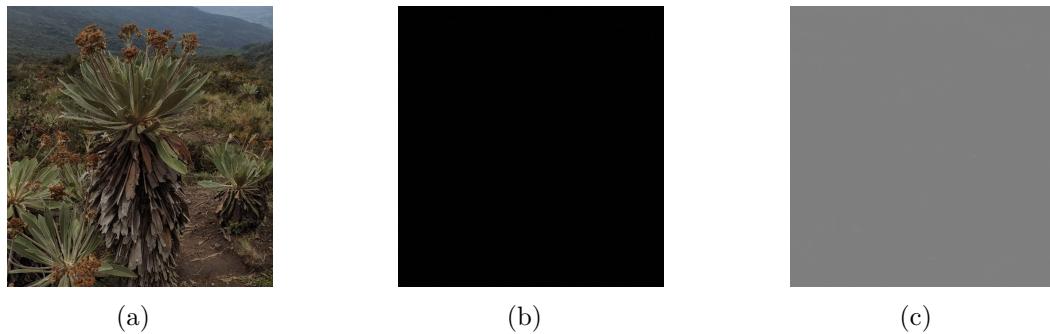


Figura 5.12: (a) Imagen resultante del estándar JPEG, (b) diferencia absoluta con la imagen original, (c) Imagen resultante de la implementación y (d) diferencia absoluta con la imagen original

Imagen	Tamaño	TE JPEG (s)	TE Implementación (s)
	[215 × 215] 46225	04,406s	17,641s
	[1920 × 1080] 2073600	139,265s	859,104s
	[600 × 636] 381600	28,137s	149,368s

Cuadro 5.2: Comparativa de Tiempo de Ejecución entre el estándar JPEG y la implementación con las imágenes anteriores

5.2.3. Ventajas y limitaciones sobre esta implementación

Para esta implementación hay que tener en cuenta las siguientes ventajas y desventajas:

- El nivel de distorsión es un poco mayor en comparación con la implementación anterior, sin embargo, sigue siendo menor en comparación con el estándar JPEG.
- La codificación propuesta comprime la información de forma bastante similar que la primera implementación.
- El costo computacional se reduce en gran medida en comparación con la primera implementación, sin embargo, sigue siendo mayor que la del estándar.

5.3. Cuantificación sobre la imagen

En esta implementación omitimos dividir la imagen en bloques de 8×8 píxeles y cuantificamos sobre toda la imagen, el resultado será un mayor número de ocurrencia en la valor de los píxeles y, en consecuencia, mayor distorsión de la imagen.

5.3.1. Cuantificación Lloyd-Max

Podemos seguir tomando el intervalo desde el valor más alto de píxeles y el más pequeño.

Para esta implementación debemos notar que no tiene sentido tomar de ejemplo la matriz M asociada a la imagen descrita en la figura 4.6, ya que se cuantifica sobre toda la imagen y no sobre las subimagenes.

El diseño de los cuantificadores es igual al de las dos implementaciones anteriores. Sin embargo, reducir los niveles de decisión sí afectarán en gran medida el nivel de error, por lo tanto se aconseja dejar en 32 y 16 los niveles de decisión para la matriz de cuantificación de luminancia y de cromancia respectivamente.

Los resultados de esta implementación se mostrarán en la siguiente sección.

5.3.2. Análisis comparativo

Tomando la imagen completa (figura 4.4) tenemos los siguientes resultados:

$$E[(M - \hat{M})^2] = \frac{1}{215 \times 215} \sum_{v=0}^{214} \sum_{w=0}^{214} [(M(v, w) - \hat{M}(v, w))^2] = 75,401$$

Este resultado se puede comprobar observando la figura 5.3, en donde podemos comprobar que el error cuadrático medio de esta implementación, es bastante menor que el obtenido

en el estándar.

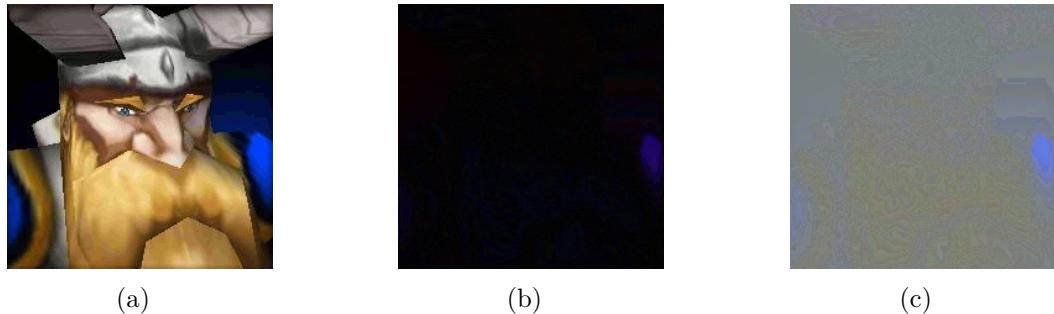


Figura 5.13: (a) Imagen resultante de la implementación, (b) diferencia absoluta con la imagen original y (c) diferencia de granulado con la imagen original

Con los otros dos ejemplos tenemos

Para la imagen de la figura 5.4 con esta implementación tenemos

$$E[(M - \hat{M})^2] = \frac{1}{1920 \times 1080} \sum_{v=0}^{1919} \sum_{w=0}^{1079} [(M(v, w) - \hat{M}(v, w))^2] = 11,118$$

Dichos resultados se pueden observar en la figura 5.11

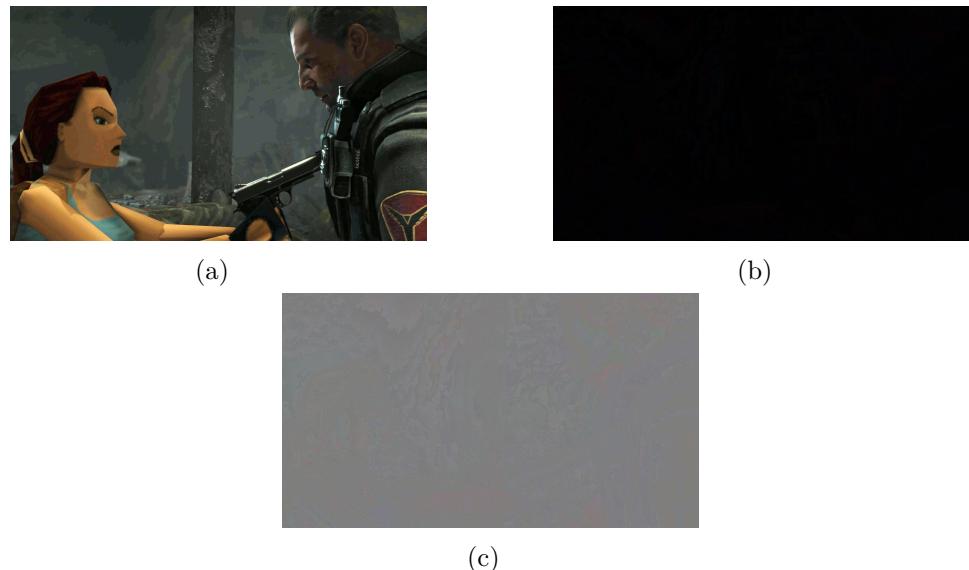


Figura 5.14: (a) Imagen resultante del estándar JPEG, (b) diferencia absoluta con la imagen original, (c) Imagen resultante de la implementación y (d) diferencia absoluta con la imagen original

Para la imagen de la figura 5.6 con esta implementación tenemos

$$E[(M - \hat{M})^2] = \frac{1}{600 \times 636} \sum_{v=0}^{599} \sum_{w=0}^{635} [(M(v, w) - \hat{M}(v, w))^2] = 2,7276$$

Dichos resultados se pueden observar en la figura 5.12

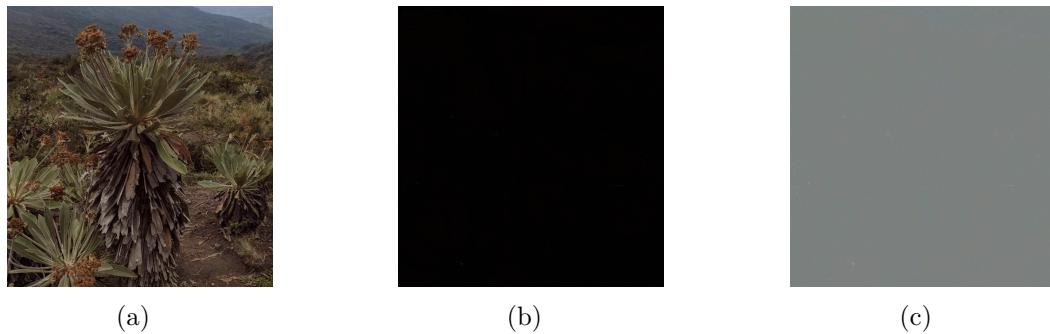


Figura 5.15: (a) Imagen resultante de la implementación, (b) diferencia absoluta con la imagen original y (c) diferencia de granulado con la imagen original

Imagen	Tamaño	TE JPEG (s)	TE Implementación (s)
	[215 × 215] 46,225 píxeles	04,406s	14,618s
	[1920 × 1080] 2,073,600 píxeles	139,265s	1921,940s
	[600 × 636] 381,600 píxeles	28,137s	131,285s

Cuadro 5.3: Comparativa de Tiempo de Ejecución entre el estándar JPEG y la implementación con las imágenes anteriores

5.3.3. Ventajas y limitaciones sobre esta implementación

Para esta implementación hay que tener en cuenta las siguientes ventajas y desventajas con respecto a las dos implementaciones anteriores y el estándar JPEG:

- El nivel de distorsión es mayor en comparación con las dos implementaciones anteriores, aunque sigue siendo un poco menor en comparación con el estándar JPEG.
- La codificación propuesta comprime mucho mejor la información con respecto a las dos implementaciones anteriores, acercándose al nivel de compresión del estándar JPEG.
- El costo computacional se reduce un poco en comparación con la implementación anterior, sin embargo, sigue siendo mayor que la del estándar.

6 | Programación del algoritmo

6.1. Programación de los algoritmos del estándar JPEG

Para implementar el algoritmo Lloyd-Max de cuantificación se ha programado el estandar de JPEG en *Octave*, siguiendo los pasos explicados en la sección 4.2.

1. Se carga la imagen de muestra, la cual es guardada en un arreglo de dos o tres dimensiones (dependiendo si la imagen es a color o no) la cual contiene variables de tipo `uint8`¹

```
muestra = imread('muestra.bmp');
```

2. Se divide la imagen en sus tres canales de color RGB y a continuación se convierten a los canales de color YCbCr.

```
R = muestra(:,:,1);
G = muestra(:,:,2);
B = muestra(:,:,3);

Y = ((R * 0.257) + (G * 0.504) + (B * 0.098)) + 16;
Cb = ((R * -0.148) + (G * -0.291) + (B * 0.439)) + 128;
Cr = ((R * 0.439) + (G * -0.368) + (B * -0.071)) + 128;
```

¹La variable `uint8` en matlab y en octave se almacena como un entero sin signo de 1 byte (8 bits), es decir, un número del intervalo [0,255]

3. Se pasan las matrices a `double()` para realizar la DCT-II, después se crea una función que realiza la transformada discreta de coseno II y se utiliza la función `blockproc()` para realizar la DCT-II a cada bloque 8×8 para cada canal.

```

Y = double(Y);
Cb = double(Cb);
Cr = double(Cr);

fun1 = @(bloque) dct2(bloque);

bloquesY = blockproc(Y, [8 8], fun1);
bloquesCb = blockproc(Cb, [8 8], fun1);
bloquesCr = blockproc(Cr, [8 8], fun1);

```

4. Se crean las matrices de cuantificación Q_1 y Q_2 para los canales de Y y de Cb y Cr respectivamente, y en seguida se definen otras dos funciones, las cuales cuantifican cada bloque 8×8 para cada canal.

```

Q1 = [16 11 10 16 24 40 51 61;
       12 12 14 19 26 58 60 55;
       14 13 16 24 40 57 69 56;
       14 17 22 29 51 87 80 62;
       18 22 37 56 68 109 103 77;
       24 35 55 64 81 104 113 92;
       49 64 78 87 103 121 120 101;
       72 92 95 98 112 100 103 99];

Q2 = [17 18 24 47 99 99 99 99;
       18 21 26 66 99 99 99 99;
       24 26 56 99 99 99 99 99;
       47 66 99 99 99 99 99 99;
       99 99 99 99 99 99 99 99;
       99 99 99 99 99 99 99 99;
       99 99 99 99 99 99 99 99];

```

```

99 99 99 99 99 99 99 99 99] ;

fun2 = @(bloque) round(bloque(:, :) ./ Q1);
fun3 = @(bloque) round(bloque(:, :) ./ Q2);

bloquesY = blockproc(bloquesY, [8 8], fun2);
bloquesCb = blockproc(bloquesCb, [8 8], fun3);
bloquesCr = blockproc(bloquesCr, [8 8], fun3);

```

5. Se descantiza y se realiza el inverso del DCT-II creando nuevas funciones:

```

fun4 = @(bloque) bloque(:, :) .* Q1;
fun5 = @(bloque) bloque(:, :) .* Q2;

bloquesY = blockproc(bloquesY, [8 8], fun4);
bloquesCb = blockproc(bloquesCb, [8 8], fun5);
bloquesCr = blockproc(bloquesCr, [8 8], fun5);

fun6 = @(bloque) idct2(bloque);

bloquesY = blockproc(bloquesY, [8 8], fun6);
bloquesCb = blockproc(bloquesCb, [8 8], fun6);
bloquesCr = blockproc(bloquesCr, [8 8], fun6);

```

6. Ahora, antes de unir nuevamente todos los canales, debemos tener en cuenta de que la función `blockproc()` añade el número de coeficientes necesarios para que podamos crear los bloques de 8×8 . Por ejemplo, si nuestra imagen tiene originalmente unas dimensiones de 215×215 , la función la extenderá hasta 216×216 , ya que 216 es múltiplo de 8 . Por lo tanto se restaurará el tamaño original de la imagen:

```

[filas, columnas] = size(bloquesY);
[filasIniciales, columnasIniciales] = size(muestra);

```

```

filasFinales = filas - mod(filasIniciales, 8);
columnasFinales = columnas - mod(columnasIniciales, 8);

bloquesY = bloquesY(1:filasFinales, 1:columnasFinales);
bloquesCb = bloquesCb(1:filasFinales, 1:columnasFinales);
bloquesCr = bloquesCr(1:filasFinales, 1:columnasFinales);

```

7. Finalmente las matrices son convertidas nuevamente a `uint8` y las transformamos al espacio de color RGB, así por último son unidas para obtener la imagen final.

```

Y = uint8(bloquesY);
Cb = uint8(bloquesCb);
Cr = uint8(bloquesCr);

R = (Y-16) * 1.164 + (Cr - 128) * 1.596;
G = (Y-16) * 1.164 - (Cr -128) * 0.813 - (Cb - 128) * 0.391;
B = (Y-16) * 1.164 + (Cb - 128) * 2.018;

imagenFinal = cat(3,R,G,B);

```

6.2. Programación de la primera implementación del algoritmo Lloyd-Max

Esta implementación parte del código programado en la sección anterior y se le introduce las funciones para encontrar cuantizar los canales.

Declaramos la función `qLM1`, la cual recibe la matriz de 8×8 transformado y devuelve la matriz `bloqueCuantizado`. Primeramente toma el tamaño de la matriz, el valor del primer coeficiente y el valor del resto de coeficientes, así convierte el coeficiente DC en 0 y calcula el máximo y el mínimo valor de los coeficientes AC. Declaramos los niveles de decisión, el cual para esta primera función corresponde a 32 (para el canal de iluminación) y creamos la tabla cuantizadora, así calculamos el número de niveles de reconstrucción y

finalmente creamos la matriz cuantizada.

La segunda parte de la función es la implementación del algoritmo de Lloyd-Max propiamente, hacemos 8 iteraciones (se espera que en ese número el nivel de error sea mínimo) y se empieza a iterar sobre el tamaño del bloque. Cada coeficiente es tomado y es aproximado al valor más cercano en los niveles de reconstrucción y se actualiza el bloque. Después iteramos para actualizar los niveles de decisión, aquí realizamos una operación equivalente a la propuesta en la sección 4.3.2. pues generamos la distribución gaussiana con la desviación estándar y la media de los valores actuales y con ella actualizamos la distribución de probabilidad, así actualizamos los niveles de reconstrucción realizando el cociente entre la sumatoria del producto del valor actual del bloque con su probabilidad y la sumatoria de la probabilidad del valor.

```

function bloqueCuantizado = qLM1(block)
    tamaño = size(block, 1);
    coeficienteDC = block(1, 1);
    coeficientesAC = block;
    coeficientesAC(1,1) = 0;
    maxAC = max(abs(coeficientesAC(:)));
    minAC = min(coeficientesAC(:));
    nivelesDecision = 32;
    tablaCuantizadora = zeros(tamaño-1);
    nReconstruccion = linspace(minAC, maxAC, nivelesDecision);
    bloqueCuantizado = block;

    for iter = 1:8
        for i = 1:tamaño
            for j = 1:tamaño
                coeficienteActual = bloqueCuantizado(i, j);
                [~, index] = min(abs(coeficienteActual - nReconstruccion));
                bloqueCuantizado(i, j) = round(nReconstruccion(index));
                tablaCuantizadora(i, j) = index - 1;
            end
        end
    for k = 1:nDecision-1

```

```

        muestras = bloqueCuantizado(tablaCuantizadora == k-1);
        mu = mean(muestras);
        sigma = std(muestras);
        peso = mu + sigma * randn(size(muestras));
        nReconstruccion(k) = sum(muestras .* peso) / sum(peso);
    end
end
bloqueCuantizado(1,1)=coeficienteDC;
end

```

El programa sigue llamando las funciones para cada bloque como se ve

```

bloquesY = blockproc(bloquesY, [8 8], @qLM1);
bloquesCb = blockproc(bloquesCb, [8 8], @qLM2);
bloquesCr = blockproc(bloquesCr, [8 8], @qLM2);

fun6 = @(bloque) idct2(bloque);

bloquesY = blockproc(bloquesY, [8 8], fun6);
bloquesCb = blockproc(bloquesCb, [8 8], fun6);
bloquesCr = blockproc(bloquesCr, [8 8], fun6);

...

```

La función qLM2 es casi en su totalidad igual a qLM1, con la única diferencia de que la variable nivelesDecision toma el valor 16.

6.3. Programación de la segunda implementación del algoritmo Lloyd-Max

En esta implementación se suprimen las funciones de DCT que se realizaban en el estándar JPEG y una vez que se separan los canales, cada subImagen 8×8 llama a la función, la

cual es prácticamente igual a la descrita en el código anterior.

```

function bloqueCuantizado = qLMP(block)
    tamaño = size(block);
    max = max(abs(block(:)));
    lower = min(block(:));
    tablaCuantizadora = zeros(n-1,m-1);
    nDecision = 12;
    nReconstruccion = linspace(lower, max, nDecision);
    bloqueCuantizado = block;

    for iter = 1:8
        for i = 1:tamaño
            for j = 1:tamaño
                indiceActual = bloqueCuantizado(i, j);
                [~, index] = min(abs(indiceActual - nReconstruccion));
                bloqueCuantizado(i, j) = round(nReconstruccion(index));
                tablaCuantizadora(i, j) = index - 1;
            end
        end
        for k = 1:nDecision-1
            muestras = bloqueCuantizado(tablaCuantizadora == k-1);
            mu = mean(muestras);
            sigma = std(muestras);
            peso = mu + sigma * randn(size(muestras));
            nReconstruccion(k) = sum(muestras .* peso)
                / sum(peso);
        end
    end
end

```

6.4. Programación de la tercera implementación del algoritmo Lloyd-Max

En esta implementación realizamos la cuantización sobre toda la imagen, por lo tanto podemos tener tan solo una función para la cuantización de ambos canales.

La única diferencia con el código de la implementación anterior es la entrada del parámetro `nDecision` con el cual definimos el número de niveles de decisión para cada canal.

```

function bloqueCuantizado = qLMP(block,nDecision)
    n = size(block, 1);
    m = size(block, 2);
    max = max(abs(block(:)));
    lower = min(block(:));
    tablaCuantizadora = zeros(n-1,m-1);
    nReconstruccion = linspace(lower, max, nDecision);
    bloqueCuantizado = block;

    for iter = 1:8
        for i = 1:n
            for j = 1:m
                indiceActual = bloqueCuantizado(i, j);
                [~, index] = min(abs(indiceActual - nReconstruccion));
                bloqueCuantizado(i, j) = round(nReconstruccion(index));
                tablaCuantizadora(i, j) = index - 1;
            end
        end
        for k = 1:nDecision-1
            muestras = bloqueCuantizado(tablaCuantizadora == k-1);
            mu = mean(muestras);
            sigma = std(muestras);
            peso = mu + sigma * randn(size(muestras));
            nReconstruccion(k) = sum(muestras .* peso)
                / sum(peso);
    
```

```
    end
  end
end
```

6.5. Observaciones Finales

En esta última sección se darán algunas observaciones que se consideran significativas en la implementación.

- La complejidad computacional del algoritmo programado en `Octave` corresponde a $O(n^4)$, lo cual representa un gran problema al momento de manipular imágenes grandes como se evidencia en las tablas 5.1, 5.2, 5.3.
 - Dicho valor podría reducirse si se encuentra una forma más óptima de manipular los índices.
- En todas las implementaciones los niveles de decisión se han tomado pensando en reducir al mínimo el error, lo cual produce menos compresión. Estos niveles pueden reducirse si se desea mejorar la compresión.
- Una posible cuarta implementación se podría hacer sin transformar los canales RGB a YCbCr.

7 | Conclusiones

- El estándar JPEG es una gran introducción al entendimiento de compresión de datos gracias a su algoritmo de procesamiento de imagen y a la codificación presentada en el mismo.
- Aunque el estándar de cuantización utilizado en en JPEG no sea óptimo, ha sido estudiado y en general logra un buen balance entre distorsión de imagen y su gran forma de codificar la información.
- Como cualquier método de compresión de imágenes, el uso de las implementaciones del algoritmo de Lloyd-Max para la cuantización en la compresión de imágenes se pueden decidir dependiendo de las necesidades del usuario. Aunque, en general, éstas no son las mejores tomando en cuenta la relación nivel de compresión y costo computacional.
 - Sin embargo, puede diseñarse, a nivel general, un mejor algoritmo que pueda implementar el algoritmo de Lloyd-Max para reducir el costo computacional y aumentar el nivel de compresión.

Bibliografía

- [1] Rafael C. Gonzalez y Richard E, Woods, *Digital Image Processing* 4th Edition, Pearson, 2018.
- [2] K.R. Rao, Humberto Ochoa Domínguez y Shreyanka Subbarayappa, *JPEG Series*, River Publishess, 2021.
- [3] Yun-Qing Shi y Huifang Sun, *Image and Video Compression for Multimedia Engineering, Fundamentals, Algorithms and Standards* Third Edition, Crc Press, 2019.
- [4] Ben Girod, *EE398A Image and Video Compression*, Leland Stanford Junior University. Obtenido de: <https://web.stanford.edu/class/ee398a/handouts/lectures/05-Quantization.pdf>