UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

ESTUDIO DE LA CODIFICACIÓN PREDICTIVA DE IMÁGENES BASADA EN CÓDIGOS RICE

María Manglano Bermejo

Tutor: José Prades Nebot

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 3 de Julio de 2020

Resumen

El proceso de digitalización en el que vivimos obliga a que cada imagen, audio, vídeo, etc. que queramos transmitir o almacenar sea previamente codificada para transformarla en una secuencia de bits. En algunos casos, es necesario que esta codificación sea lo más precisa posible, y no introduzca error. El trabajo se centra en la codificación de imágenes sin error basada en predicción lineal y códigos Rice. El uso de códigos Rice en codificación predictiva se utiliza en muchas técnicas de comprensión de audio como Shorten y MPEG-4 ALS, y de imágenes como FELICS.

En este trabajo, se han estudiado los fundamentos teóricos de la codificación predictiva y de los códigos Rice. En MATLAB se han programado una serie de funciones que permiten implementar distintas configuraciones de predictor y codificador Rice. Con estas funciones y un conjunto de imágenes test se ha estudiado la eficiencia de cada una de las configuraciones consideradas. Los resultados de este análisis han permitido ver cuales son las ventajas y desventajas de cada una de las técnicas de predictor y codificador utilizadas.

Resum

El procés de digitalització en què vivim obliga que cada imatge, àudio, vídeo, etc. que vulguem transmitir o almatzenar siga prèviament codificat per transformar-lo en una seqüència de bits. En alguns casos, cal que aquesta codificació siga la més precisa posible, sense errors. Aquest treball es centra en la codificació d'imatges sense errors basada en predicció lineal i codis Rice. L'ús de codis Rice en codificació predictiva es utilizat en moltes tècniques de compressió d'àudio com Shorten i MPEG-4 ALS, i d'imatges com FELICS.

En aquest treball s'han estudiat els fonaments teòrics de la codificació predictiva i dels codis Rice. En MATLAB s'han programat una sèrie de funcions que permeten implementar diverses configuracions de predictor i codificador Rice. Amb aquestes funcions i un conjunt d'imatges test s'ha estudiat l'eficiència de cadascuna de les configuracions considerades. Els resultats d'aquest anàlisi ha permès veure quins són els avantatges i desavantatges de cadascuna de les tècniques de predictor i codificador utilitzats.

Abstract

The process of digitalization that we are currently living makes us compress the images, audios, videos, etc before transmitting or saving them. In this way, we transform them into a sequence of bits that are easier to transmit or save. In some cases, it is also necessary a lossless compression. This project focuses on the lossless predictive compression of images with Rice coding. This codification is used in multiple compression techniques of audio such as Shorten y MPEG-4 ALS, and images as FELICS.

In this project, we studied the basis of lossless predictive compression and Rice coding. In MATLAB we have programmed some functions that allow us to implement different configurations of predictor and Rice encoder. We have studied the efficiency of these configurations by using different images. The results have revealed the advantages and disadvantages of each technique used.

Índice general

Bibliografía **37**

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Objetivos

El proceso de digitalización en el que vivimos obliga a que cada imagen, audio, vídeo, etc. que queramos transmitir o almacenar sea previamente codificada para transformarla en una secuencia de bits. En algunos casos, es necesario que esta codificación sea lo más precisa posible, y no introduzca error. La finalidad de este trabajo es el estudio de la codificación de imágenes sin error basada en predicción lineal y códigos Rice. El uso de códigos Rice en codificación predictiva se utiliza en muchas técnicas de comprensión de audio como Shorten y MPEG-4 ALS, y de imágenes como FELICS.

Para conseguir la finalidad perseguida se plantearon dos objetivos principales. El primer objetivo es conseguir una tasa binaría de envío mínima codificando imágenes predictivamente con códigos Rice, y el segundo es ver si la tasa binaria difiere respecto a otras técnicas de codificación.

Estos objetivos generales se han plasmado en los siguientes objetivos específicos:

- Estudiar los fundamentos de codificación de fuente sin pérdidas, en concreto, la codificación con predictor lineal usando códigos Rice.
- Implementar una serie de funciones de MATLAB que permitan la codificación y decodificación de imágenes utilizando dos tipos de predicción y varios tipos de codificación Rice.
- Estudiar la eficiencia de cada una de las posibles opciones de este código, y compararla con otros tipos de codificación.
- Ver cuales son las ventajas y desventajas de cada configuración de predictor y codificador implementada.

1.2. Metodología empleada

A grandes rasgos, la realización de este trabajo fin de grado se ha dividido en tres grandes tareas: Estudio de los fundamentos teóricos de codificación, programación de codificador y predictor, y estudio de resultados.

Más concretamente, se han realizado las siguientes actividades:

1. Estudiar los fundamentos de la codificación. (60 horas)
2. Estudiar el entorno de programación de MATLAB. (30 horas)
3. Elegir un conjunto de imágenes test de diferentes tamaños. (5 horas)
4. Programar en MATLAB una serie de funciones que permiten implementar el codificador con distintos predictores y configuraciones de códigos Rice. (115 horas)
5. Estudiar los resultados obtenidos y compararlos con otras técnicas de codificación. (20 horas)
6. Redactar la memoria. (70 horas)

Como puede observarse, gran parte del tiempo se ha empleado en el estudio de la codificación y en la programación de las distintas configuraciones de predictor y codificador. Además, algunas de las actividades han sido transversales a la realización del trabajo, como estudio del entorno de MATLAB.

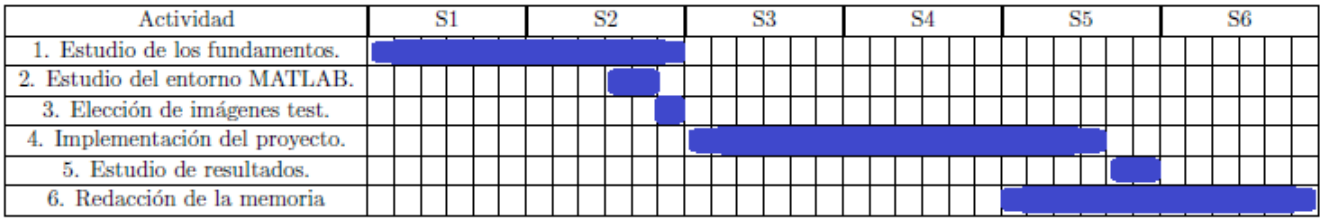


Figura 1: Diagrama temporal de actividades por semana.

Capítulo 2

Fundamentos teóricos

En este capítulo se explica la codificación de fuente, y las magnitudes necesarias para medir la eficiencia del codificador. Desarrollamos el concepto de código, y profundizamos en códigos Rice. También veremos como se representa matemáticamente una imagen y como esta se puede codificar predictivamente.

2.1. Codificación de fuente

La mayor parte de los sistemas de comunicación actuales son digitales y, consecuentemente, requieren que la información a transmitir (voz, música, imagen, vídeo,...) esté representada mediante una secuencia de bits. En muchas ocasiones, la información está contenida en una secuencia $\mathbf{x}[\mathbf{n}]$ que no es digital. Esto hace que sea necesario transformar $\mathbf{x}[\mathbf{n}]$ en una secuencia de bits, proceso que se conoce como *codificación de fuente*. A tal efecto, el transmisor dispone de un *codificador de fuente* que transforma $\mathbf{x}[\mathbf{n}]$ en una secuencia de bits. Dicha secuencia de bits se transforma en una señal analógica que se transmite a través de un medio de transmisión. En el otro extremo, la señal analógica recibida se demodula y la secuencia de bits obtenida se introduce en un *decodificador de fuente* que transforma la secuencia de bits en la secuencia decodificada $\mathbf{y}[\mathbf{n}]$. Estas operaciones se muestran en las figura 2. En la práctica, la secuencia de bits recibida puede no coincidir con la transmitida debido a la distorsión introducida por el medio de transmisión y el ruido del receptor. Para reducir o eliminar los bits erróneos suele usarse una técnica conocida como *codificación de canal*. En este trabajo supondremos que la secuencia de bits demodulada no contiene ningún bit erróneo.

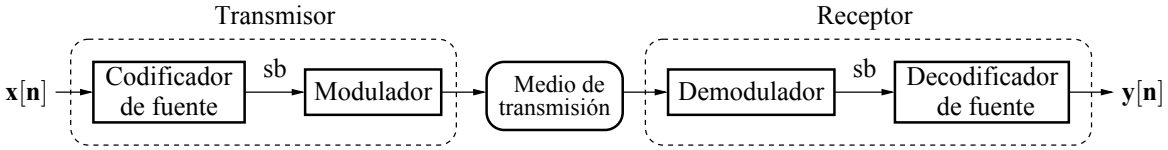


Figura 2: Diagrama de bloques básico de un sistema de comunicación digital.

Algo similar ocurre cuando hay que *almacenar* la información en un dispositivo. Dado que los dispositivos de almacenamiento actuales son digitales, cualquier señal a almacenar debe primero convertirse en una secuencia de bits usando un codificador de fuente. Cuando un usuario quiere

recuperar la secuencia, este debe usar un decodificador de fuente para decodificar la secuencia de bits almacenada en el dispositivo.

Aunque en ingeniería el término «codificación» se usa en muchos contextos (codificación de canal, codificación de fuente, codificación de un algoritmo, ...), en este trabajo únicamente hablaremos de codificación de fuente. En aras de la brevedad, a partir de ahora obviaremos decir «de fuente» cada vez que nos refiramos a la codificación o decodificación.

Mediremos la eficiencia de la codificación de una secuencia $\mathbf{x}[\mathbf{n}]$ mediante la distorsión y la tasa binaria. La *distorsión*, D, es una medida global del error introducido en el proceso de codificación-decodificación de una secuencia. Cuando $\mathbf{x}[\mathbf{n}]$ e $\mathbf{y}[\mathbf{n}]$ son iguales, no hay distorsión y se dice que la codificación es *sin pérdidas*. En caso contrario, se dice que la codificación es *con pérdidas*. La tasa binaria del codificador, R, es el número de bits que se invierten en promedio en la codificación de cada muestra de $\mathbf{x}[\mathbf{n}]$. Otra aspecto importante a tener en cuenta es el *retardo* introducido al codificar $\mathbf{x}[\mathbf{n}]$. En servicios conversacionales, como videoconferencia o videotelefonía, es necesario que el retardo de codificación sea pequeño.

Aparte de la distorsión, la tasa binaria y el retardo la distorsión, es importante considerar la complejidad del proceso de codificación (y decodificación). En general, la codificación de una secuencia la realiza un algoritmo, implementado en software o en hardware. Por esta razón, la complejidad computational suele medirse mediante el número de operaciones aritméticas que requiere en promedio codificar cada muestra de $\mathbf{x}[\mathbf{n}]$ y la cantidad de memoria que utiliza el codificador.

2.1.1. Codificación de imágenes

Dado que en este trabajo nos centraremos en la codificación de imágenes, vamos a ver en esta sección cómo representar una imagen matemáticamente. Un sensor de imagen proporciona una secuencia $\mathbf{x}[\mathbf{n}]$ en la que $\mathbf{n} = [n_1, n_2]^T$. Las variables discretas n_1 y n_2 representan las coordenadas espaciales de cada muestra o *píxel* de la imagen. La coordenada vertical es n_1 y la coordenada horizontal es n_2. Por convención, el origen de la imagen es el punto $\mathbf{n} = (0, 0)$, n_1 crece hacia abajo y n_2 crece hacia la derecha (como se muestra en la figura 3)

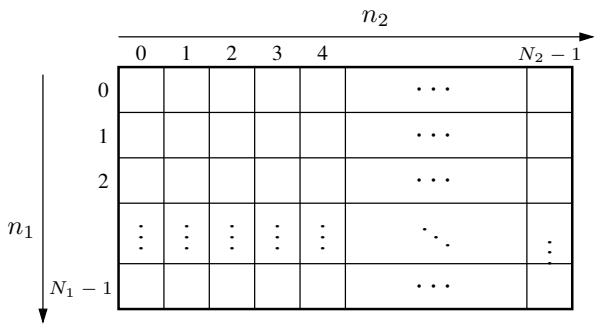


Figura 3: Coordenadas en una imagen.

De acuerdo con lo anterior, podemos representar una imagen como la secuencia 2D

$$\mathbf{x}[n_1, n_2], \quad 0 \leq n_1 \leq N_1,\ 0 \leq n_2 \leq N_2 \tag{1}$$

donde N_1 y N_2 son el número de píxeles en horizontal y vertical, respectivamente. La variable

dependiente **x** es vectorial y el número de componentes depende del espacio de color utilizado, esto es, de cómo se representa el color en la imagen.

Los espacios de color más utilizados son el espacio RGB y el espacio YCbCr. En el espacio RGB, un color se representa mediante tres componentes

$$\mathbf{x} = [\mathrm{R} \quad \mathrm{G} \quad \mathrm{B}]^T \tag{2}$$

llamadas componentes *roja* (R), *verde* (G) y *azúl* (B). Cada componente representa la energía de color en una banda de longitudes de onda. El espacio RGB se utiliza en *captación de imágenes* (el sensor de una cámara representa el color de cada píxel de una imagen en el espacio RGB) y en *visualización de imágenes* (para visualizar una imagen en una pantalla es necesario especificar el valor de R, G y B de cada uno de sus píxeles).

El espacio de color YCbCr también tiene tres componentes:

$$\mathbf{x} = [\mathrm{Y} \quad \mathrm{Cb} \quad \mathrm{Cr}]^T \tag{3}$$

La componente Y o *luminancia* representa el brillo del color. Las componentes Cb y Cr se llaman *componentes de crominancia* azúl y roja, respectivamente. Las componentes de crominancia no tienen significado físico pero son necesarias porque es imposible representar el color con menos de tres componentes. El espacio YCbCr se utiliza en transmisión y almacenamiento de imágenes.

Un color **x** en el espacio RGB puede transformarse al espacio YCbCr mediante una transformación afín. De forma similar, podemos transformar un color YCbCr en el correspondiente color RGB utilizando la transformación afín inversa.

La mayor parte de los algoritmos de codificación permiten codificar secuencias 1D de escalares. Sin embargo, las imágenes son secuencias 2D de vectores. La solución habitual a este problema consiste en realizar dos pasos previos en el codificador:

1. Descomponer la imagen a codificar $\mathbf{x}[n_1, n_2]$ en tres secuencias 2D:

$$x_1[n_1, n_2], \quad x_2[n_1, n_2] \quad \text{y} \quad x_3[n_1, n_2] \tag{4}$$

   donde x_1, x_2 y x_3 son las componentes de color de $\mathbf{x}[n_1, n_2]$. Cada $x_i[n_1, n_2]$ o *imágen componente* puede considerarse una matriz $N_1 \times N_2$. Cada imagen componente tiene dos varibles independientes (n_1 y n_2).

2. Trasformar cada imagen componente $x_i[n_1, n_2]$ en una secuencia con una única variable independiente, n, utilizando un proceso llamado *exploración* (*scanning*). Lo habitual es utilizar una *exploración ráster* que consiste en concatenar todas las filas de la imagen componente.

Tras realizar estos dos pasos, la imagen original se ha convertido en tres secuencias escalares 1D, $x_1[n]$, $x_2[n]$ y $x_3[n]$, que en general se codifican de manera independiente utilizando un mismo algoritmo (figura 4). Los dos pasos realizados son reversibles y, consecuentemente, pueden invertirse en el codificador. La codificación de cada $x_i[n]$ se realiza de forma secuencial (se codifica por orden, empezando por la primera muestra y acabando en la última).

Resumamos todo lo anterior. El sensor de la cámara captura una imagen en el espacio RGB. Dicha imagen se transforma en una imagen YCBCr, $x_{\mathrm{YCbCr}}[n_1, n_2]$, utilizando una transformación afín. Posterioremente, el codificador transforma $x_{\mathrm{YCbCr}}[n_1, n_2]$ en tres imágenes componente ($x_i[n_1, n_2]$, $i = 1, 2, 3$). Cada imagen componente $x_i[n_1, n_2]$ se explora y se codifica, generánose así tres secuencias de bits que se multiplexan y transmiten. En el receptor, la decodificación invierte cada uno de estos pasos.

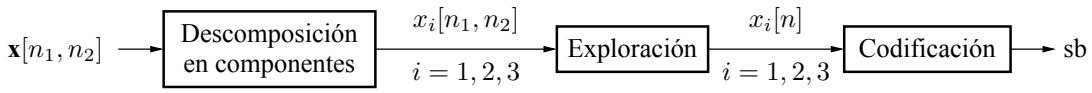


Figura 4: Desomposición en conponentes, exploración y codificación.

2.2. Fuentes de información y codificación

En esta sección revisaremos un modelo probabilístico para las secuencias a codificar: la fuente de información. Basándonos en este modelo, definiremos la entropía, que es fundamental para conocer cuál es la menor tasa binaria con la que podemos codificar una fuente discreta sin memoria (FDE).

2.2.1. Fuentes de información

En la sección anterior, hemos considerado la codificación de una *secuencia determinista*, esto es, una secuencia en la que se conoce la amplitud en cada **n**. En general, la codificación debe ser eficiente con *cualquier* secuencia de un determinado tipo (el tipo de secuencia para el que ha sido diseñado el codificador). Por ejemplo, un codificador de imágenes debería codificar eficientemente cualquier imagen que se pueda capturar con una cámara (aunque es posible que la codificación sea más eficiente en algunas imágenes que en otras). Por tanto, al diseñar un codificador no deberíamos considerar una única secuencia (determinista) sino el conjunto de infinitas secuencias de un determinado tipo que potencialmente pueden codificarse. Para contemplar este conjunto infinito de potenciales secuencias, consideraremos que en la entrada del codificador hay una *secuencia aleatoria* y que cada secuencia que en la práctica se codifica es una *realización* de longitud finita de dicha secuencia. En codificación, se llama fuente de información a las secuencias aleatorias que modelan las señales a codificar.

Definimos una *fuente de información* $X[n]$ ($n = 0, 1, \ldots$) como una secuencia de variables aleatorias con un alfabeto (o rango) común $\mathcal{X}$. Los elementos de $\mathcal{X}$ se llaman *símbolos*. Cada realización de la fuente, $x[n]$, es una secuencia determinista de infinitas muestras. La codificación de una fuente $X[n]$ genera una secuencia de bits aleatoria y su correspondiente decodificación da como resultado una secuencia aleatoria decodificada $Y[n]$.

Dependiendo del tipo de variable aleatoria, podemos clasificar las fuentes de información en *fuentes discretas* (cuando $\mathcal{X}$ es un conjunto finito o infinito numerable) o *fuentes continuas* (cuando $\mathcal{X}$ es infinito no numerable). En general, caracterizar una fuente de información es complicado porque hay que especificar cualquier función de probabilidad conjunta que se pueda definir entre sus variables aleatorias. Dicha caracterización es más sencilla cuando la fuente es *estacionaria*.

En una fuente discreta estacionaria (FDE) tenemos que

$$p_{X_0,\ldots,X_{n-1}}(x_0, \ldots, x_{n-1}) = p_{X_i,\ldots,X_{i+n-1}}(x_0, \ldots, x_{n-1}) \tag{5}$$

para todo $i > 0, n > 0$, y $x_0, \ldots, x_{n-1} \in \mathcal{X}$. Observamos que en cualquier función de probabilidad conjunta, al desplazar en el tiempo todas sus variables aleatorias, dicha función no cambia. Una definición similar aplica a una fuente estacionaria continua (FCE) cambiando las funciones de probabilidad por funciones de densidad de probabilidad en (5).

Si en (5) especificamos $n = 1$, tenemos

$$p_{X_0}(x_0) = p_{X_i}(x_0), \tag{6}$$

para todo $i > 0$ y $x_0 \in \mathcal{X}$. De esto concluimos que en una FDE, todas sus variables aleatorias están idénticamente distribuidas (estoe es, todas ellas tienen la misma función de probabilidad $p_X(\cdot)$). La simplicidad de las fuentes estacionarias hace que sean los modelos más utilizados en codificación.

También podemos clasificar las fuentes según la relación que hay entre sus variables aleatorias. Cuando las variables son independientes decimos que la fuente es *sin memoria* (*con memoria* en caso contrario). De acuerdo con todo lo anterior, una fuente discreta estacionaria sin memoria (FDEsM) está formada por infinitas variables aleatorias independientes y todas ellas tienen la misma función de probabilidad que denotaremos como $p_X(x)$ o simplemente $p(x)$ cuando no haya posibilidad de confusión, esto es,

$$p(x) = p\{X = x\} \tag{7}$$

Una FDE puede codificarse sin distorsión. Algunos resultados de la *teoría de la información* permiten conocer cuál es la menor tasa binaria con la que se puede codificar sin pérdidas una FDE. A continuación veremos algunas magnitudes que permiten determinar dichas tasas binarias.

2.2.2. Entropía

Una maginitud de gran importancia en la codificación de una FDE es la *entropía*. Sea X una variable aleatoria con alfabeto $\mathcal{X}$ y función de probabilidad $p(x)$. Definimos la *entropía* de X como

$$H(X) = -\sum_{x \in \mathcal{X}} p(x)\ \log_2 p(x). \tag{8}$$

La unidad de la entropía es el bit. Por brevedad, en el resto de este trabajo nos referiremos a la entropía de una variable aleatoria $X[n]$ de una FDE simplemente como «la entropía de $X[n]$».

Si disponemos de una realización de longitud finita $x[n]$ ($n = 0, \ldots, N-1$) de una FDE, podemos estimar los valores de $p(x)$ mediante frecuencias relativas $f(x)$:

$$f(x) = \frac{\#x}{N} \tag{9}$$

donde $\#x$ es el número de veces que ha aparecido el símbolo x en la secuencia $x[n]$. A veces hablaremos de *la entropía de una secuencia determinista* $x[n]$, queriendo decir que es la entropía de una FDE en la que la probabilidad de cada símbolo coincide con la frecuencia relativa de dicho símbolo en $x[n]$ ($p(x) = f(x)$). Esto es, definimos la *entropía de* $x[n]$ como

$$H(x) = -\sum_{x \in \mathcal{X}} f(x)\ \log_2 f(x). \tag{10}$$

En el resto de este trabajo el término «entropía» hará referencia a (8) cuando estemos considerando la secuencia de variables aleatorias generada por una FDE, y hará referencia a (10) cuando estemos considerando una secuencia determinista de longitud finita.

2.2.3. Fuentes geométricas y TSGD

La función de probabilidad es una de las características más importantes a tener en cuenta a la hora de codificar una FDE de manera eficiente. En esta sección, veremos dos distribuciones de gran importancia en este trabajo.

Una variable aleatoria X se dice que es *geométrica* de parámetro ρ si su alfabeto es $\mathcal{X} = \{0, 1, 2, \ldots\}$ y su función de probabilidad es

$$p(x) = (1-\rho)\,\rho^x, \quad x = 0, 1, 2, \ldots \tag{11}$$

donde $0 < \rho < 1$. Cuanto mayor es ρ, más lentamente decrece $p(x)$.

La media de X es

$$\mu_X = \frac{\rho}{1-\rho} \tag{12}$$

y su varianza es $\sigma^2 = \frac{\rho}{(1-\rho)^2}$. La entropía de X es

$$H(X) = \frac{-(1-\rho)\log_2(1-\rho) - \rho\log_2\rho}{1-\rho}. \tag{13}$$

La figura 5 muestra la función de probabilidad de una distribución geométrica con $\rho = 0{,}25$ en el intervalo $x \in [0, 50]$.

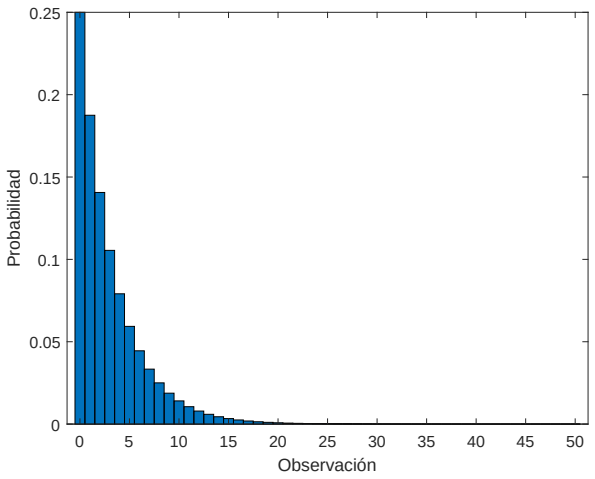


Figura 5: Función de probabilidad de una distribución geométrica con parámetro $\rho = 0{,}25$.

Una variable aleatoria X se dice que sigue una distribución TSGD (*two-sided geometric distribution*) de parámetro ρ ($0 < \rho < 1$), si $\mathcal{X} = \mathbb{Z}$ y su función de probabilidad es

$$p(x) = \frac{1-\rho}{1+\rho}\,\rho^{|x|}. \tag{14}$$

En esta distribución, la función de probabilidad decrece monótonamente conforme nos alejamos de $x = 0$. La entropía de una variable aleatoria TSGD de parámetro ρ es

$$H(X) = -\log_2\left(\frac{1-\rho}{1+\rho}\right) - \frac{2\rho\log_2(\rho)}{1-\rho^2}. \tag{15}$$

La figura 6 muestra la distribución geométrica con parámetro $\rho = 0{,}5$ en $x \in [-20, 20]$.

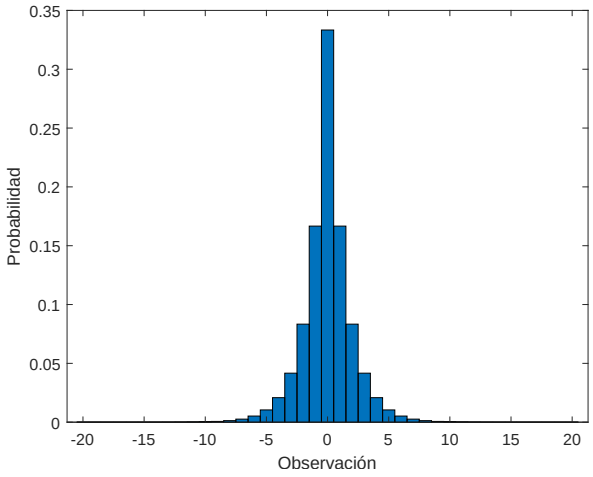


Figura 6: Distribución TSGD con $\rho = 0{,}5$ en $x \in [-20, 20]$.

Como veremos posteriormente, la distribución TSGD es fundamental en el diseño de codificadores predictivos para imagen.

2.3. Códigos

Muchas aplicaciones requieren que la codificación realizada sea simple computacionalmente. Podemos realizar la codificación sin pérdidas de una fuente de forma muy sencilla utilizando un código. Un *código* es una función uno a uno, $c(x)$, que asigna a cada símbolo de la fuente una palabra binaria. Para codificar una secuencia de símbolos, un código concatena las palabras que corresponde a cada símbolo de la secuencia. El resultado de esta concatenación es la secuencia de bits que se transmite al decodificador. Para que el decodificador pueda separar las palabras que hay en la secuencia de bits, es necesario que el código utilizado sea *separable*. Además, para que el decodificador puede decodificar una palabra tan pronto como dicha palabra se ha recibido, el código separable utilizado debe ser *instantáneo*, esto es, ninguna palabra debe ser prefijo de otra [1].

Un código puede implementarse con una tabla de $|\mathcal{X}|$ entradas (cada vez que hay que codificar un símbolo, el codificador busca en la tabla la palabra correspondiente). La siguiente tabla especifica un código instantáneo para una fuente discreta con alfabeto $\mathcal{X} = \{a, b, c\}$.

x	$p(x)$	$c(x)$
a	0.5	0
b	0.25	10
c	0.25	11

Denotaremos la longitud (en bits) de la palabra de un símbolo x como $l(x)$. Supongamos que codificamos una FDE $X[n]$ con un código $c(x)$. La tasa binaria, o número de bits invertidos en promedio en codificar cada variables aleatoria $X[n]$, es la longitud media

$$R = \mathbb{E}\{l(X)\} = \sum_{x \in \mathcal{X}} p(x)\, l(x) \tag{16}$$

donde $\mathbb{E}$ denota el operador esperanza. Si las palabras de un código tienen todas la misma longitud se dice que es un *código de longitud fija* (CLF). Un código que no es un CLF se dice que es un *código de longitud variable* (CLV).

En la práctica, siempre codificaremos una secuencia determinista de longitud finita $x[n]$ ($n = 0, \ldots, N-1$). Pero si $x[n]$ es una realización de una FDE $X[n]$ y N es grande, el número de bits invertidos en codificar cada símbolo de $x[n]$ coincidirá aproximadamente con la que proporciona (16). Por esta razón, se intenta codificar una FDE $X[n]$ con un código que proporcione la menor tasa binaria. Un código para una FDE $X[n]$ se dice que es *óptimo* si no existe otro código para $X[n]$ que proporcione una tasa binaria inferior.

Si codificamos sin pérdidas una FDE $X[n]$ con un código, se cumple [1]:

$$H(X) \leq R < H(X) + 1. \tag{17}$$

El valor concreto que toma R dentro del intervalo $[H(X), H(X)+1)$ depende de la función de probabilidad de la fuente y del código utilizado. Observamos en (17) que no es posible codificar sin pérdidas una FDE con un código a una tasa binaria inferior a su entropía. De ahí que la entropía sea una magnitud tan importante en codificación sin pérdidas.

Un código codifica los símbolos que genera una FDE $X[n]$ de manera independiente (la codificación de un símbolo no depende de símbolos anteriores o posteriores). Por esta razón, un código no explota la posible dependencia que pueda haber entre las variables de una fuente. Hay métodos de codificación más sofisticadas que sí que explotan dicha dependencia y en los que la tasa binaria puede ser inferior a $H(X)$. En la sección 2.4 veremos un ejemplo de este tipo de métodos.

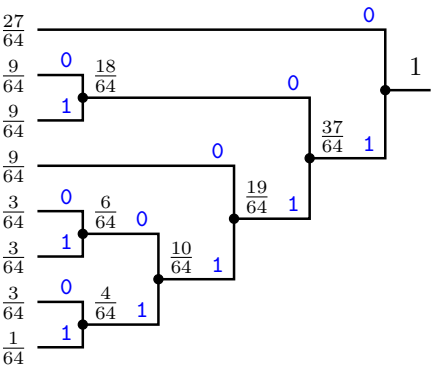
2.3.1. El código Huffman

El *código Huffman* es un código instantáneo óptimo. La construcción de un código Huffman para una FDE $X[n]$ requiere realizar los siguientes pasos:

1. Ordena los símbolos según sus probabilidades.
2. Considera los dos símbolos de menor probabilidad.
3. Combina los dos símbolos en un único símbolo.
   Fija la probabilidad del símbolo creado como la suma de las probabilidades de símbolos unidos.
   Asigna un `0` y un `1` a cada rama que confluyen en el símbolo creado.
4. Vuelve a 2 si la fuente resultante del paso anterior tiene más de un símbolo.
5. Partiendo del nodo raíz, recorre el árbol y obtén la palabra de cada símbolo.

La siguiente figura muestra la construcción de un código Huffman para una fuente con alfabeto $\mathcal{X} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f}, \mathrm{g}, \mathrm{h}, \mathrm{i}\}$. En las dos columnas de la derecha se muestran los símbolos del alfabeto y las probabilidades de cada símbolo. Las dos columnas de la izquierda especifican las palabras y sus correspondientes longitudes. En el árbol, los bits que se asignan a cada rama son de color azul y los símbolos generados se han representados con puntos (su probabilidad está en su derecha).

$c(x)$	$l(x)$	$\mathcal{X}$	$p(x)$
0	1	a	$\frac{27}{64}$
100	3	b	$\frac{9}{64}$
101	3	c	$\frac{9}{64}$
110	3	d	$\frac{9}{64}$
11100	5	e	$\frac{3}{64}$
11101	5	f	$\frac{3}{64}$
11110	5	h	$\frac{3}{64}$
11111	5	i	$\frac{1}{64}$



En general, el código de Huffman de una fuente discreta es un *código de longitud variable*. Para minimizar R, la longitud de las palabras que asigna un código Huffman depende de la probabilidad del símbolo (en general, a mayor probabilidad, menor longitud).

Para construir el código de Huffman debemos conocer las probabilidades de cada símbolo de la fuente. Una vez construido el código, es necesario almacenarlo en tablas que se usan al codificar y decodificar. Esto supone un gran problema cuando la fuente tiene un alfabeto muy grande.

2.3.2. Códigos de longitud fija

Los códigos de longitud fija permiten codificar una FDE de la manera más simple posible. Un *código de longitud fija* (CLF) asigna palabras de $\lceil \log_2 |\mathcal{X}| \rceil$ bits a cada uno de los símbolos de la fuente. Por tanto, la tasa binaria en un CLF es

$$R_{\mathrm{CLF}} = \lceil \log_2 |\mathcal{X}| \rceil. \tag{18}$$

Para codificar con un CLF una fuente no es necesario conocer la probabilidad de sus símbolos. Ni siquiera es necesario una tabla puesto que podemos asignar a cada símbolo de la fuente un entero del conjunto en $\{0, 1, \ldots, 2^{R_{\mathrm{CLF}}}\}$ y codificar dicho entero en binario con R_{CLF} bits. En contrapartida, un CLF es generalmente un código ineficiente (en tasa binaria) puesto que la codificación con un CLF no tiene en cuenta la distribución de la fuente.

2.3.3. Códigos unarios

Sea $\mathcal{X} = \mathbb{N}$ (o un subconjunto de $\mathbb{N}$) y sea $X[n]$ una fuente con alfabeto $\mathcal{X}$. Un *código unario* para $\mathcal{X}$ asigna a cada $x \in \mathcal{X}$ la palabra

$$c(x) = \underbrace{0\,0 \cdots 0}_{x \text{ ceros}}\,1$$

Por tanto, la palabra código de un símbolo x consiste en una palabra de x ceros consecutivos terminada con un `1`, que funciona como separador de símbolos. Por ejemplo, si $\mathcal{X} = \{0, 1, 2\}$, las palabras de su código unario son: $c(0) = 1$, $c(1) = 01$ y $c(2) = 001$. Si $X[n]$ sigue una distribución geométrica, el código unario es un código instantáneo óptimo si $\rho \leq 0{,}618$ [2].

2.3.4. Codigos Golomb

Si $X[n]$ es geométrica con $\rho > 0{,}618$, entonces el código unario no es óptimo. En muchas aplicaciones es interesante tener un código que sea óptimo para fuentes geométricas con cualquier valor de ρ. El código de Golomb cumple esta propiedad.

La codificación Golomb con parámetro $m > 0$, Golomb(m), representa x mediante la descomposición única

$$x = x_q\, m + x_r \tag{19}$$

donde $x_q \in \{0, 1, \ldots\}$ y $x_r \in \{0, 1, \ldots, m-1\}$. Los enteros x_q y x_r son el *cociente* y el *resto* de la división de x con m y pueden obtenerse mediante:

$$x_q = \left\lfloor \frac{x}{m} \right\rfloor \tag{20}$$

$$x_r = x - x_q \times m. \tag{21}$$

La codificación Golomb codifica x_q y x_r con dos códigos, $c_q(\cdot)$ y $c_r(\cdot)$, y concatena las palabras resultantes. El código $c_q(\cdot)$ es un código unario. El código $c_r(\cdot)$ es más complejo y se describe a continuación. Primero fijamos

$$k = \lceil \log_2 m \rceil \quad \text{y} \quad m_r = 2^k - m \tag{22}$$

y con estas cada símbolo x_r con la siguiente regla:

- si $x_r < m_r$, codifica x_r con un CLF de $k-1$ bits
- si no, codifica $x_r + m_r$ con un CLF de k bits.

Es fácil ver que cada uno de los pasos dados al codificar es invertible y que el código resultante es un código instantáneo [2]. El valor de m óptimo es el menor entero que cumple $\rho^m \geq \frac{1}{2}$, esto es,

$$\rho^m \gtrapprox \frac{1}{2}. \tag{23}$$

2.3.5. Códigos Rice para números naturales

Cuando $m = 2^k$, la codificación Golomb se llama *codificación Rice* y lo denotaremos como RN(k). Esta codificación es muy sencilla puesto que c_r es siempre un CLF de k bits.

Para codificar x con un RN(k) damos los siguientes pasos:

1. Obtén x_q utilizando (20) y codifica x_q con un código unario ($c_q(x_q)$).

2. Obtén x_r utilizando (21) y codifica x_r con un CLF de k bits ($c_r(x_r)$).

3. Concatena las palabras $c_q(x_q)$ y $c_q(x_q)$.

Por ejemplo, si $x = 21$ y $m = 8$ ($k = 3$), daremos los siguientes pasos:

$$x_q = 2 \to c_q(x_q) = \texttt{001}$$
$$x_r = 5 \to c_r(x_r) = \texttt{101}$$
$$c(21) = c_q(x_q) \parallel c_r(x_r) = \boxed{\texttt{001101}}$$

Si $m = 1$, entonces $k = 0$, $x_q = x$ y por tanto el código Rice es un código unario.

El algortimo 1 muestra la implementación de un decodificador RN(k).

Algoritmo 1: Decodificador RN(k)

Data: Secuencia de bits, k
while *(queden bits)* **do**
  Cuenta el número de ceros que preceden al primer uno $\to x_q$
  Decodifica los m bits siguientes $\to x_r$
  Saca $x = x_r\, 2^k + x_q$ y elimina los bits decodificados
end

La codificación Rice de una fuente geométrica no puede ser óptima a no ser que m^* sea una potencia de dos. Sin embargo, eligiendo un valor de k adecuado, el código Rice permite codificar una fuente geométrica de forma eficiente y sencilla.

2.3.5.1. Códigos Rice para números enteros

Supongamos que $\mathcal{X} = \mathbb{Z}$ y que una FDE $X[n]$ sigue una distribución TSGD. Si transformamos $X[n]$ mediante la aplicación reversible

$$\hat{x} = \begin{cases} 2x, & x \geq 0 \\ 2|x| - 1, & x < 0 \end{cases}, \tag{24}$$

obtenemos una fuente $\hat{X}[n]$ con alfabeto $\hat{\mathcal{X}} = \{0, 1, 2, \ldots\}$ que cumple las siguientes propiedades:

- $p_{\hat{X}}(n) \geq p_{\hat{X}}(n+1)$ $(n \geq 0)$;
- $p_{\hat{X}}(2n) = p_{\hat{X}}(2n-1)$ $(n \geq 1)$;
- $p_{\hat{X}}(2n)$ es una distribución geométrica.

Podemos codificar $X[n]$ de la siguiente manera: primero se transforma el símbolo x a codificar en $\hat{x}$ utilizando (24) y posteriormente se codifica $\hat{x}$ usando un código Rice para números naturales.

Para decodificar, primero se decodifica la palabra código y después se aplica la transformación inversa a la definida en (24):

$$x = \begin{cases} \hat{x}/2, & \hat{x} \text{ par} \\ -(\hat{x}+1)/2, & \hat{x} \text{ impar} \end{cases}. \tag{25}$$

Este tipo de codificación se llama *codificación Rice para enteros* y lo denotaremos con RE(k). Gracias a las tres propiedades mencionadas anteriormente, esta codificación es eficiente si k se elige adecuadamente (de acuerdo con el valor de ρ de la distribución TSGD).

2.3.5.2. Codificación Rice adaptativa

Como la codificación Rice para enteros depende un único parámetro (k), podemos obtener fácilmente el k óptimo de una FDE geométrica (mismo k en todas los símbolos) o realizar una adaptación sencilla cuando la fuente geométrica no es estacionaria (esto es, cuando el parámetro ρ de la distribución geométrica puede cambiar con n) [2].

En una FDE geométrica, la media es $\mu_X = \frac{\rho}{1-\rho}$. Si suponemos que $1-\rho \ll 1$ (esto es, que $\rho \approx 1$), tenemos que

$$\rho^m \approx 1 - \frac{m(1-\rho)}{\rho} = 1 - \frac{m}{\mu_X} \tag{26}$$

Como queremos que $\rho^m \gtrsim \frac{1}{2}$ para que el código sea óptimo, tenemos que $m = 2^k \lesssim \frac{1}{2}\mu_X$ y, de ahi, el valor de k óptimo es

$$k^* = \text{máx}\left\{0, \left\lceil \log_2\left(\frac{1}{2}\mu_X\right)\right\rceil\right\} \tag{27}$$

Aunque esta expresión para k^* se ha obtenido suponiendo que $\rho \approx 1$, también proporciona buenos resultados cuando ρ es pequeño.

El valor de k se puede estimar en cada símbolo con el fin de realizar una *codificación Rice adaptativa* [2]. En este tipo de codificación, el k estimado en cada n se utiliza para codificar el símbolo x_n de la fuente. Un ejemplo de esta codificación adaptativa lo constituye el algoritmo 2 que tiene los siguientes argumentos de entrada:

- x_n es la secuencia de símbolos a estimar;
- $\hat{\mu}_X$ es una estima inicial de μ_X
- $N_{\text{máx}}$ es un entero que permite controlar la estabilidad de la estima.

En el algoritmo 2, las variables N y A representan un número de símbolos y la suma de dichos símbolos, respectivamente. De ese modo, la estima de μ_X en cualquier instante se aproxima por A/N. Cada $N_{\text{máx}}$ muestras, A y N se dividen por dos reduciéndose de este modo la influencia de muestras pasadas en la estima. Cuanto mayor es $N_{\text{máx}}$, mayor es la estabilidad en la estima de μ_X pero menor es el grado de adaptabilidad a cambios en la estadística de la fuente. Para decodificar

cada símbolo, el decodificador primero estima k (de forma idéntica a como lo hace el codificador) y posteriormente realiza la decodificación de la correspondiente palabra código.

Algoritmo 2: Codificador Rice adaptativo

Data: x_n, $\hat{\mu}_X$ y $N_{\text{máx}}$
$A = \hat{\mu}_X$ y $N = 1$
for *i = 1,2,...* **do**
  $k = \text{máx}\left\{0, \left\lceil \log_2 \frac{A}{2N} \right\rceil\right\}$
  Codifica x_n con RN(k)
  if $N = N_{\text{máx}}$ **then**
    $A = \lceil A/2 \rceil$ y $N = \lfloor N/2 \rfloor$
  end
  $A = A + x_n$
  $N = N + 1$
end

Podemos realizar una codificación adaptativa de una $X[n]$ en la que $\mathcal{X} = \mathbb{Z}$ (o un subconjunto de $\mathbb{Z}$) si transformamos $X[n]$ en $\hat{X}[n]$ utilizando (24) y codificamos los símbolos de $\hat{X}[n]$ con el algoritmo 2.

2.4. Codificación predictiva

La codificación predictiva es una alternativa más eficiente a la codificación directa con códigos. En esta sección veremos brevemente los fundamentos de la codificación predictiva sin pérdidas.

La figura 7 muestra el diagrama de bloques de un codificador predictivo para una secuencia unidimensional $x[n]$ ($n = 0, \ldots, N-1$). En cada instante n, en el codificador:

1. El sistema *predictor* genera la *predicción (lineal)* de $x[n]$, $\tilde{x}[n]$, mediante

$$\tilde{x}[n] = \sum_{i=1}^{P} a_i \, x[n-1] \tag{28}$$

   donde P es el *orden de predicción* y los coeficientes $a_1, \ldots, a_P$ son los *coeficientes de predicción*.

2. Se obtiene el *error de predicción*, $e[n]$, restando a cada muestra de $x[n]$ su predicción:

$$e[n] = x[n] - \tilde{x}[n]. \tag{29}$$

3. Se codifica $e[n]$ sin pérdidas. *En este trabajo supondremos que esta codificación sin pérdidas se realiza utilizando un código*.

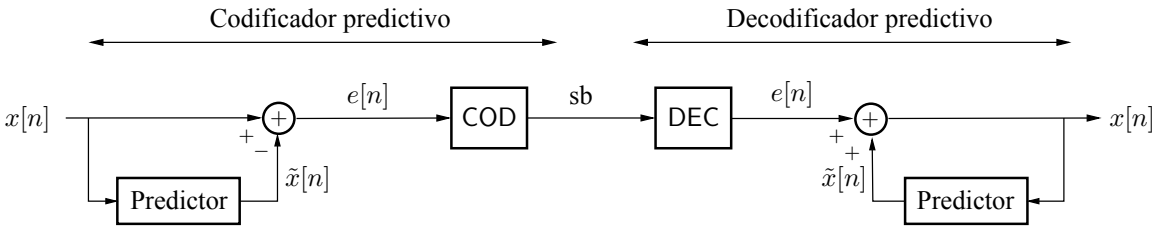


Figura 7: Esquema de un sistema de codificación predictiva.

Para decodificar la n-ésima palabra código de la secuencia de bits generada por el codificador, el decodificador predictivo realiza los siguientes pasos (figura 7):

1. Decodifica la palabra código obteniendo el error de predicción $e[n]$.
2. Genera la predicción $\tilde{x}[n]$ (usando el mismo predictor que el codificador).
3. Reconstruye $x[n]$ sumando $e[n]$ y $\tilde{x}[n]$.

Para predecir las $P-1$ primeras muestras de $x[n]$ se necesitan las muestras $x[-P+1], x[-P+1], \ldots, x[-1]$. Como estas muestras son desconocidas, codificador y decodificador deben asumir que estas muestras toman unos determinados valores. Una elección habitual en señales de media nula es considerar que son nulos:

$$x[-P+1] = x[-P+2] = \cdots = x[-1] = 0.$$

En general, cuanto mayor es el orden de predicción P, más precisa es la predicción (si se eligen los coeficientes adecuadamente). Sin embargo, cuanto mayor es P, más memoria requiere el predictor y más operaciones aritméticas debe realizar. En la práctica, la elección de P es un compromiso entre eficiencia en codificación y complejidad computacional.

Supongamos que se quiere codificar una FDE $X[n]$ predictivamente. De acuerdo con lo anterior, un codificador predictivo transforma $X[n]$ en una nueva FDE, el error de predicción $E[n]$, y codifica sin pérdidas $E[n]$. Si la codificación de $E[n]$ requiere menos bits que la de $X[n]$, entonces la codificación predictiva de $E[n]$ será más eficiente que una codificación directa de $X[n]$ con un código.

Dado que en este trabajo supondremos que $E[n]$ se codifica con un código, es importante que la entropía $H(E)$ sea la menor posible. En general, cuanto menor sea $H(E)$, menor será la tasa binaria de la codificación predictiva. La entropía $H(E)$ depende tanto de las características de $X[n]$ como del predictor utilizado. Por tanto, es razonable elegir el predictor que minimiza $H(E)$. Otro criterio de diseño muy utilizado consiste en elegir el predictor que minimiza la varianza del error de predicción σ_E^2 (en general, cuanto menor es σ_E^2, menor es la tasa binaria) [3].

Para codificar predictivamente una imagen $x[n_1, n_2]$ (figura 8), dicha imagen se transforma en una secuencia 1D, $x[n]$, que se codifica tal y como se ha explicado anteriormente (figura 7). En el decodificador, se decodifica predictivamente la secuencia de bits y finalmente se transforma $x[n]$ en $x[n_1, n_2]$ (operación raster inversa).

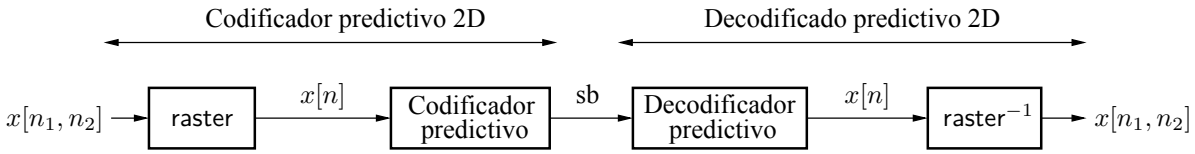


Figura 8: Esquema de un sistema de codificación predictiva para imágenes.

En el codificador de la figura 8, la imagen $x[n_1, n_2]$ es monocromática. Como se ha explicado en la sección 2.1.1, la codificación de imágenes de tres o más componentes puede realizarse de forma sencilla aplicando por separado la codificación anteriormente descrita a cada una de las imágenes componente.

Capítulo 3

Codificación predictiva de imágenes

En este capítulo describiremos la codificación predictiva sin pérdidas de imágenes utilizando códigos Rice. También se analizará la distribución que sigue el error de predicción. Implementación MATLAB

3.1. Predicción y codificación Rice

En esta sección describiremos las imágenes, los predictores y los codificadores Rice que se utilizan en la codificación predictiva considerada en este trabajo.

Respecto a las imágenes a codificar, consideramos imágenes monocromáticas de 8 bits por píxel. Por tanto, cada píxel toma un valor entero entre el 0 y 255 ($\mathcal{X} = \{0, 1, \ldots, 255\}$). En concreto, utilizaremos la componente de luminancia de imágenes RGB que se utilizan para probar la eficiencia de algoritmos de codificación (la sección 4.1 muestra dichas imágenes).

Respecto a los predictores, se han considerado dos: un predictor fijo y un predictor adaptativo. El predictor fijo usado es el de *primera diferencia*. En este predictor, cada muestra $x[n]$ que resulta de la exploración raster se predice con un predictor de primer orden con $a_1 = 1$:

$$\tilde{x}[n] = x[n-1] \tag{30}$$

siendo $\tilde{x}[0] = 0$. La precisión de este predictor es baja en aquellos píxeles que contienen contornos no horizontales porque en dichos píxeles suelen haber grandes diferencias entre $x[n]$ y $x[n-1]$

El predictor adaptativo utilizado en este trabajo es el predictor de JPEG-LS. Este es un standard definido por la ITU y la ISO para la codificación predictiva sin pérdidas de imágenes. Si definimos x_a, x_b y x_c como

$$\begin{aligned} x_a &= x[n_1, n_2 - 1] \\ x_b &= x[n_1 - 1, n_2] \\ x_c &= x[n_1 - 1, n_2 - 1] \end{aligned}$$

la predicción de un píxel $x[n_1, n_2]$ que no pertenece a la primera fila o la primera columna de la

imagen es

$$\tilde{x}[n_1, n_2] = \begin{cases} \text{mín}(x_a, x_b), & \text{si } x_c = \text{máx}(x_a, x_b, x_c) \\ \text{máx}(x_a, x_b), & \text{si } x_c = \text{mín}(x_a, x_b, x_c) \\ x_a + x_b - x_c, & \text{en otro caso} \end{cases} \tag{31}$$

En este predictor, $\tilde{x}[0, 0] = 0$. La predicción de un píxel de la primera fila de la imagen es el píxel que está en la misma fila pero en la columna anterior ($\tilde{x}[0, n_2] = x(0, n_2 - 1]$). Similarmente, la predicción de un píxel de la primera columna de la imagen es el píxel de la misma columna pero de la fila superior ($\tilde{x}[n_1, 0] = x[n_1 - 1, 0]$). Este predictor intenta predecir correctamente un posible contorno horizontal o vertical que pase a través de $x[n_1, n_2]$ [2].

Pasemos a ver ahora cómo se codifica la secuencia error de predicción $e[n]$. El valor $e[0]$ se codifica siempre usando un CLF de 8 bits. El resto de muestras de $e[n]$ toman valores enteros (positivos y negativos) y se codifican utilizando codificación Rice para enteros RE(k). Se consideran dos posibilidades: la codificación Rice fija de $e[n]$ (descrita en la sección 2.3.5.1) y la codificación Rice adaptativa de $e[n]$ (descrita en la sección 2.3.5.2). En la fija, el parámetro k permanece constante en todas las muestras de $e[n]$ mientras que en la adaptativa k cambia según el algoritmo 2. En cualquier caso, hay que recalcar que lo que se codifica son las muestras del error de predicción ($e[n]$) y no las muestras de la imagen ($x[n]$). Por tanto, cada valor de error e primero se transforma en un número natural, $\hat{e}$, de acuerdo con

$$\hat{e} = \begin{cases} 2e, & e \geq 0 \\ 2|e| - 1, & e < 0 \end{cases}, \tag{32}$$

y posteriormente se codifica $\hat{e}$ con un código Rice para números naturales.

Un problema de la codificación Rice fija es que se debe proporcionar un valor adecuado del parámetro k (la eficiencia de la codificación puede variar mucho dependiendo del k elegido). Como solución a este problema se puede realizar una *codificación de dos pasos*. En el primer paso se obtiene el error de predicción y a partir de él, se obtiene k utilizando la expresión 27. En el segundo paso, se codifica el error de predicción utilizado un código Rice fijo con el valor de k obtenido.

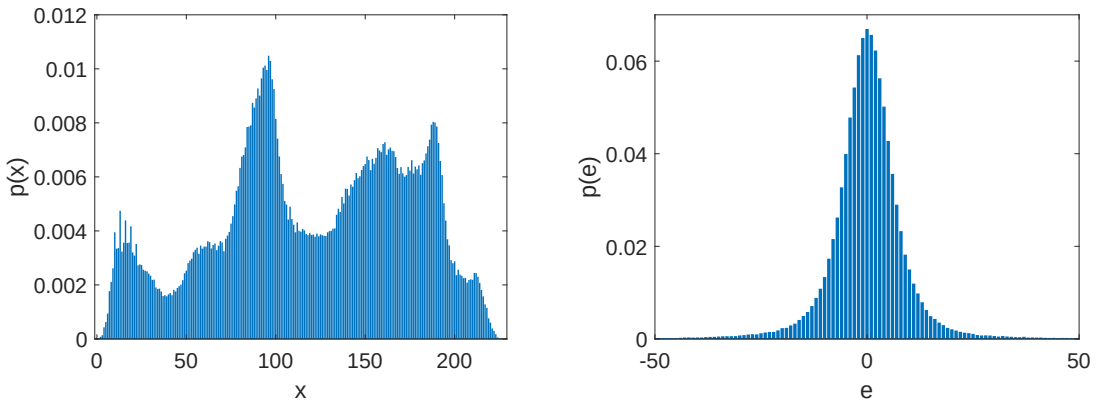
Sin embargo, esta codificación requiere tener toda la imagen en memoria para poder obtener el valor de k y no puede empezar a codificar hasta que el codificador haya obtenido el valor de *todos* los píxeles de la imagen. Este problema no existe en la codificación Rice adaptativa, puesto que el valor de k se estima píxel a píxel. Además, la codificación Rice adaptativa permite que el codificador sin pérdidas se adapte a los posibles cambios que pueda haber en la distribución de la secuencia error de predicción. Este tipo de cambios son frecuentes en imágenes, señales en las que suelen coexistir regiones muy uniformes (en las que $e[n]$ toma valores cercanos a cero) y regiones con grandes y frecuentes variaciones (en las que $|e[n]|$ toma valores grandes).

En la codificación Rice adaptativa, se utilizan los parámetros $\hat{\mu} = 16$ y $N_{\text{máx}} = 10$. La elección de $\hat{\mu} = 16$ está justificado porque, como se verá en un capítulo posterior, el valor óptimo de k en muchas imágenes cuando se utiliza un código Rice fijo es 3. Respecto a la elección $N_{\text{máx}} = 10$, se ha comprobado que el uso de dicho valor proporciona generalmente tasas binarias menores al codificar las imágenes test.

3.2. Estadística del error de predicción

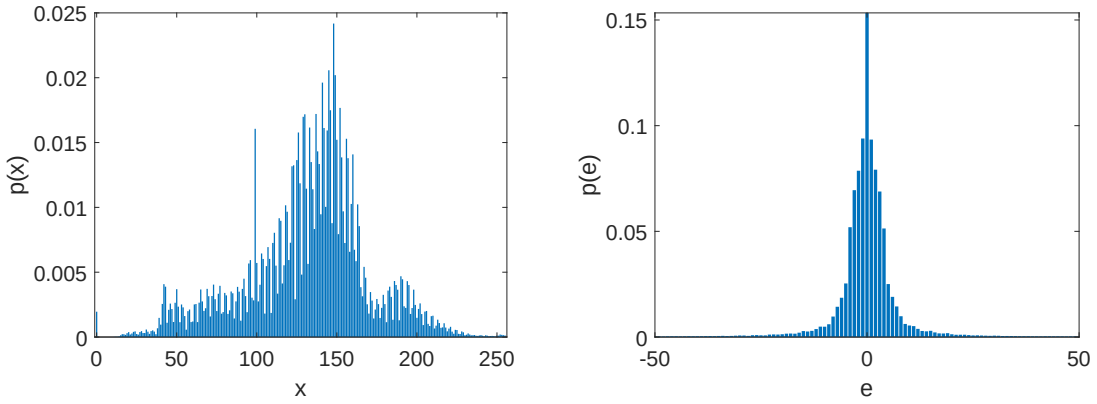
A continuación estudiaremos la diferencia que hay entre la distribución de $x[n]$ y la de $e[n]$. Para ello, analizaremos el histograma de ambas secuencias $x[n]$, y $e[n]$ que se generan al codificar varias imágenes utilizando tanto el predictor fijo como el adaptativo. También consideraremos los valores de entropía y varianza de $e[n]$ puesto que, en general, cuanto más pequeñas son estas magnitudes, menor es la tasa binaria generada al codificar.

Primero consideramos los resultados obtenidos usando el predictor fijo. La figuras 9, 9, 11, 12 y 13 muestran los histogramas de $x[n]$, y $e[n]$ de las imágenes Peppers, Kodim09, Artificial, Cafe y P22, respectivamente. En estas figuras observamos que los histogramas de las cuatro imágenes ($x[n]$) son completamente distintos. Esto imposibilita que sea posible codificar de manera eficiente las cuatro imágenes con un único código (un código que sea eficiente para una imagen no lo será para el resto). En cambio, los histogramas de $e[n]$ tienen *formas* similares en las cuatro imágenes. En concreto, los histogramas de $e[n]$ generalmente decaen de manera monótona cuando $|e|$ crece. Esta característica hace que una codificación Rice fija de $e[n]$ pueda ser eficiente puesto que dichos códigos proporcionan buenos resultados cuando las fuentes son TSGD. No obstante, la rapidez con la que decaen estos histogramas depende mucho de la imagen. Así, en la imagen Artificial (que es muy uniforme y contiene pocos contornos) el decaimiento del histograma de $e[n]$ es muy rápido mientras que en el resto de imágenes (menos uniformes y con un mayor número de contornos), el decaimiento del histograma de $e[n]$ es significativamente más lento. Esto sugiere que el parámetro k utilizado en la codificación Rice fija de $e[n]$ deberá ser muy distinto en la imagen Artificial y en el resto.



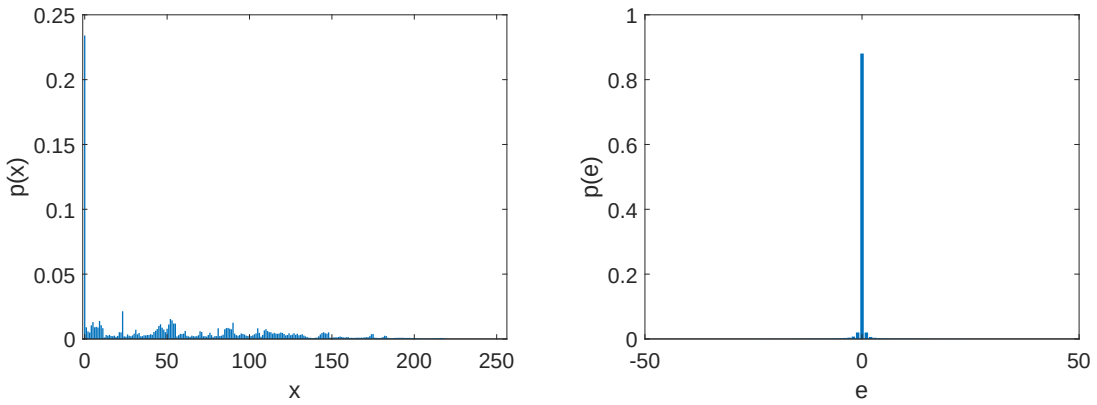
(a) (b)

Figura 9: Histogramas de: (a) luminancia y (b) error de predicción de la imagen Peppers



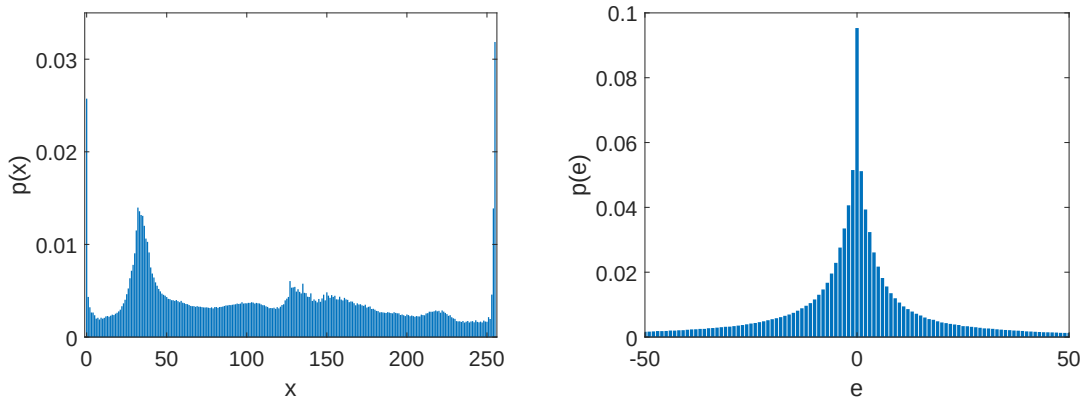
(a) (b)

Figura 10: Histogramas de: (a) luminancia y (b) error de predicción de la imagen Kodim09



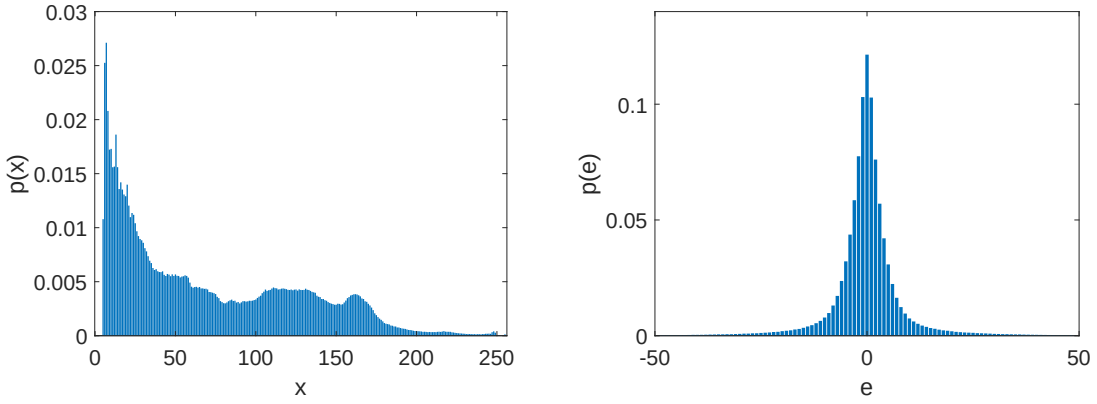
(a) (b)

Figura 11: Histogramas de: (a) luminancia y (b) error de predicción de la imagen Artificial



(a) (b)

Figura 12: Histogramas de: (a) luminancia y (b) error de predicción de la imagen Cafe



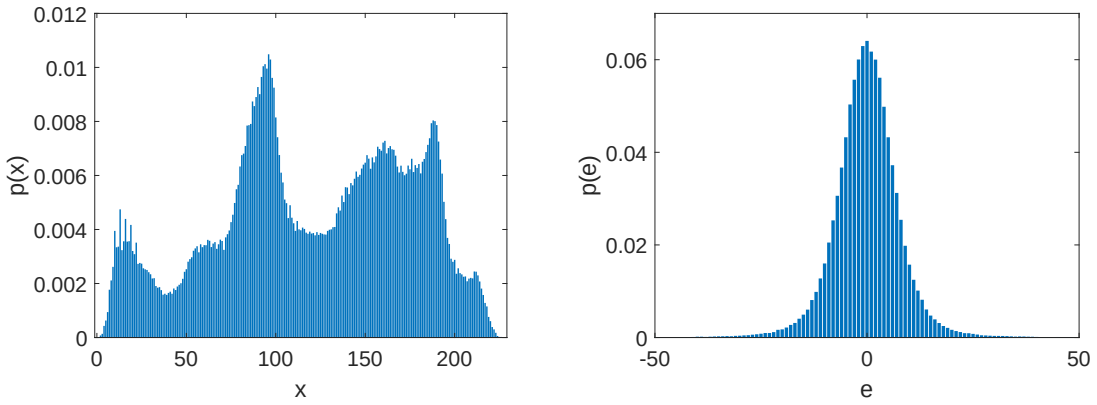
(a) (b)

Figura 13: Histogramas de: (a) luminancia y (b) error de predicción de la imagen P22

Respecto a la entropía, se cumple que $H(e) < H(x)$ en todas las imágenes. Por ejemplo, en la imagen Peppers $H(x) = 7{,}59$ bits mientras que $H(e) = 5{,}0$ bits. Por tanto, la codificación predictiva de esta imagen puede proporcionar una tasa binaria significativamente inferior que su codificación directa con código. Algo similar ocurre con la varianza. Así, en la imagen Cafe tenemos $\sigma_x^2 = 4050$ mientras que $\sigma_e^2 = 588$. Dado que σ_e^2 es mucho menor que σ_x^2, cabe esperar que la codificación predictiva de Cafe sea mucho más eficiente que su codificación directa con un código.

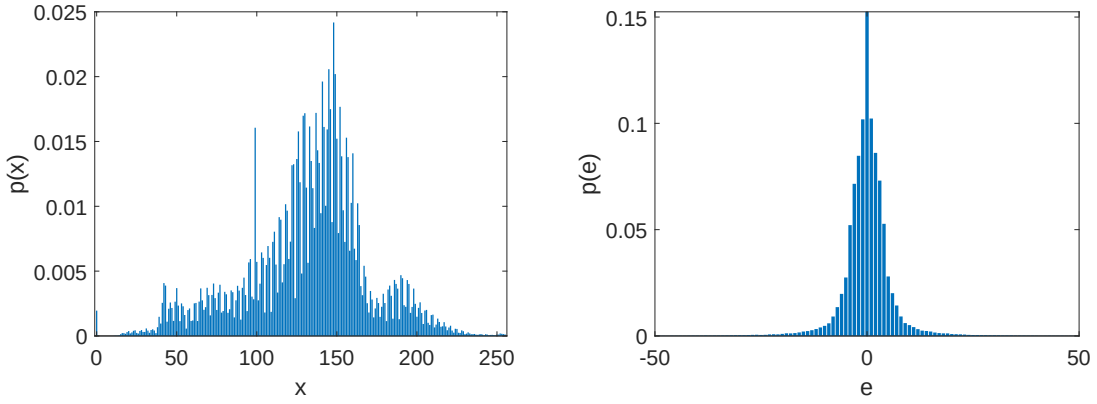
Consideremos a continuación el predictor adaptativo. La figuras 14, 15, 16, 17 y 18 muestran los histogramas de $x[n]$ y $e[n]$ de las imágenes Peppers, Kodim09, Artificial, Cafe y P22, respectivamente. En cada imagen, los histogramas de las secuencias error decrecen más rápidamente que los obtenidos al usar predicción fija. Esto es debido a que el predictor adaptativo proporciona predicciones más precisas que el fijo, sobre todo en los contornos verticales.

La entropía y la varianza del error también disminuye respecto a los valores obtenidos con predicción fija. Por ejemplo, en Cafe $\sigma_e^2 = 1645$ (con predicción fija es 4050) y mientras que en Kodim09 es $\sigma_e^2 = 128$ (con predicción fija es 588). Respecto a la reducción de entropía del error, en Peppers tenemos $H(e) = 4{,}93$ bits (5.09 bits con predicción fija) y en P22 es tenemos $H(e) = 4{,}4$ bits (4.678 bits con predicción fija).



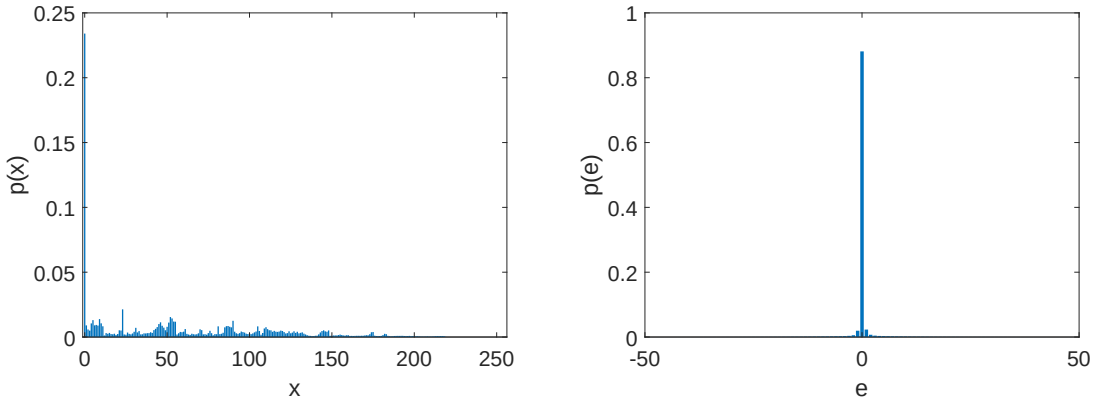
(a) (b)

Figura 14: Histogramas de: (a) luminancia y (b) error de predicción de la imagen Peppers



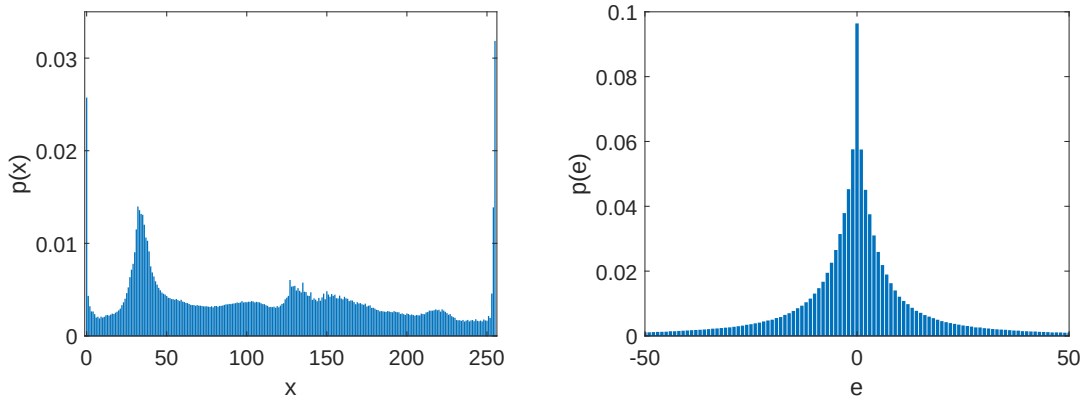
(a) (b)

Figura 15: Histogramas de: (a) luminancia y (b) error de predicción de la imagen Kodim09



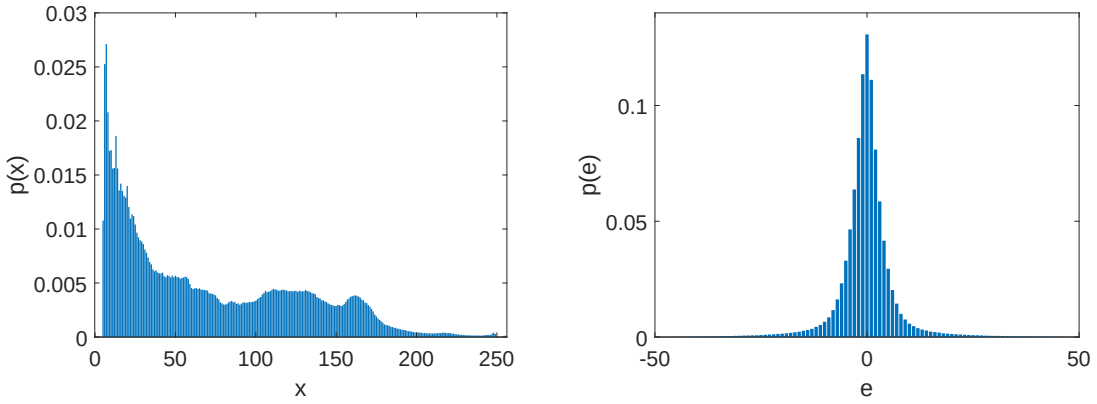
(a) (b)

Figura 16: Histogramas de: (a) luminancia y (b) error de predicción de la imagen Artificial



(a) (b)

Figura 17: Histogramas de: (a) luminancia y (b) error de predicción de la imagen Cafe



(a) (b)

Figura 18: Histogramas de: (a) luminancia y (b) error de predicción y (c) error transformado de la imagen P22

3.3. Implementación en MATLAB

En esta sección se van a explicar las funciones implementadas en MATLAB para la codificación sin pérdidas con códigos Rice de las imágenes test seleccionadas.

Para la codificación fija, es decir, codificación Rice con parámetro k constante, hemos implementado una función para el codificador y otra para el decodificador.

La función del codificador se define como:

[nBits, R] = codIm(nombreI,nombreS,p,k)

donde cada variable de entrada es:

- `nombreI`: nombre del fichero con la imagen a codificar.
- `nombreS`: nombre del fichero con la secuencia de bits generada.
- `p`: variable que indica el tipo de predictor a utilizar.
- `k`: valor del parámetro del código Rice.

y las variables de salida son

- `nBits`: número de Bits enviados.
- `R`: tasa binaria de envío expresada en bpp.

El codificador a partir del fichero `nombreI` calcula el vector del error de predicción, codifica los valores con código Rice a partir del valor del parámetro k introducido, y escribe la secuencia de

bits generada en el fichero `nombreS`. Una vez tiene todos los valores del error codificados, calcula el número total de bits y la tasa binaria.

A partir del parámetro `p` , se elige el predictor que queremos utilizar: `p=0` indica el predictor fijo, y `p=1` el predictor adaptativo. Este valor es escrito por el codificador en la cabecera del fichero `nombreS`.

La función del decodificador, `decodIm`, a partir del fichero `nombreS` genera de nuevo la imagen, para ello tiene implementados ambos predictores. Por último, se comprueba que la imagen resultante es igual a la imagen enviada, calculando el error entre ellas, y confirmando que este es cero.

Si en lugar de utilizar codificación fija, donde el parámetro k es introducido, deseamos codificar la imagen con una $k*$ calculada por el codificador, instanciaríamos las funciones de codificador y decodificador adaptativo.

El codificador adaptativo se define como

[nBits, R] = codAdapt(nombreI,nombreS,p,c)

Las variables de entrada y salida son iguales que en el caso de utilizar un codificador fijo, pero como es de esperar, en este caso la k no es una variable introducida. El valor del parámetro c selecciona si se quiere utilizar una $k*$ para todos los valores del error, o si por el contrario, se desea utilizar una $k*$ para cada valor del error.

En el caso de $c = 0$, utilizamos la $k*$ para toda la imagen implementando la codificación de dos pasos (3.1). Si $c = 1$ el codificador implementa el algoritmo 2 para calcular la $k*$ de cada valor del error (sección 2.3.5.2).

El `decodAdapt` a partir del fichero `nombreS` calcula de nuevo la imagen. Esta función además de los predictores, tiene implementados los algoritmos para calcular la $k*$ en ambos casos.

Capítulo 4

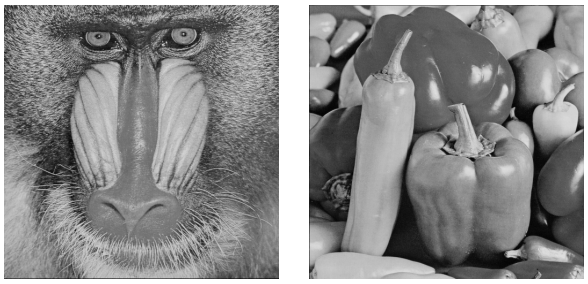
Resultados

En este capítulo se muestran y analizan los resultados obtenidos al codificar un conjunto de imágenes test con los algoritmos descritos en el capítulo anterior. Se consideran tanto el uso de predicción fija como adaptativa, y de codificación Rice tanto fija como adaptativa.

4.1. Imágenes test

Los resultados mostrados en este capítulo se han obtenido al codificar un conjunto de imágenes test con los algoritmos propuestos en el tema 2. Las imágenes test utilizadas se muestran a continuación.

Imágenes de la clase A. Dos imágenes de 512×512 obtenidas de la base de datos de la http://sipi.usc.edu/database y seis imágenes (tres 768×512 y tres de 512×768) obtenidas de http://r0k.us/graphics/kodak.



Baboon *Peppers*

Figura 19: Imágenes del grupo A de 512×512


Kodim04


Kodim09


Kodim19

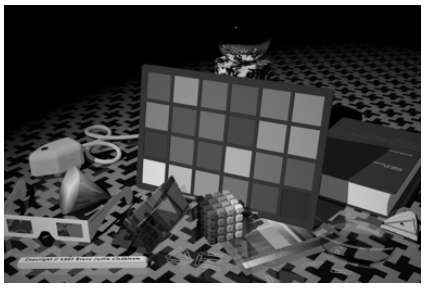

Kodim03


Kodim08


Kodim23

Figura 20: Imágenes del grupo A de 768×512 **y** 512×768**.**

Imágenes de la clase B. Cuatro imágenes de alta resolución y grandes tamaños obtenidas de http://imagecompression.info/test_images/.


Artificial 2048×3072

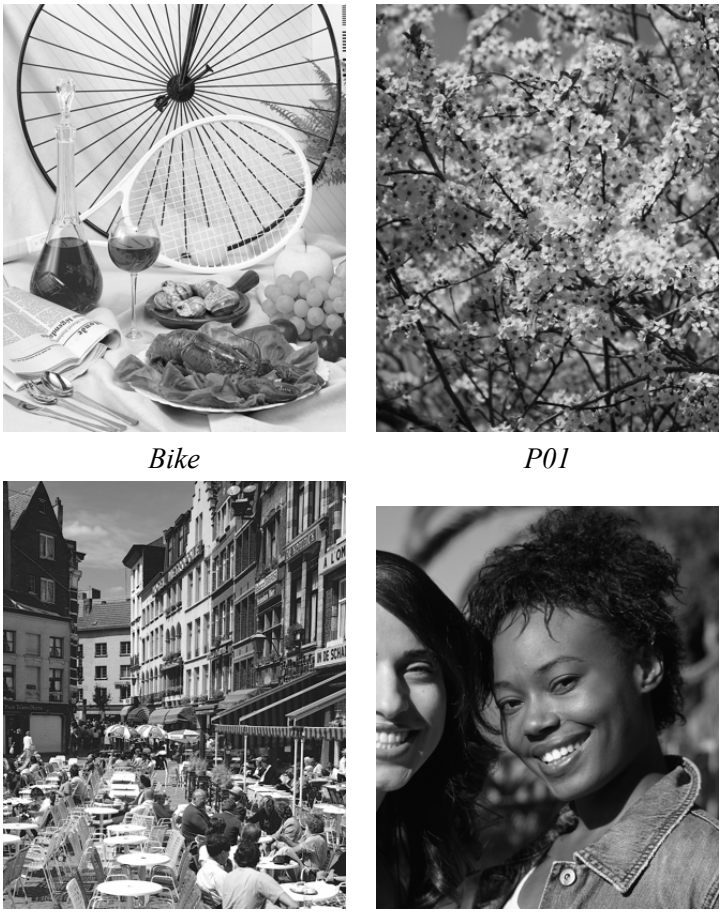

Big building 5412×7216


Big tree 4550×6088


Bridge 4049×2749

Figura 21: Imágenes del grupo B

Imágenes de la clase C. Cuatro imágenes de alta resolución de tamaño alrededor de 1600×1280 obtenidas de http://mmspg.epfl.ch/iqa.



Bike *P01*

Cafe *P22*

Figura 22: Imágenes del grupo C de 1600×1280

Estas imágenes se utilizan habitualmente para evaluar la eficiencia de algoritmos de codificación de imágenes. Todas ellas son imágenes monocromáticas (componente Y) y están codificadas a una tasa de 8 bpp.

Para que los resultados obtenidos sean representativos, el conjunto de imágenes test incluye una gran variedad de escenas y resoluciones espaciales. También incluye imágenes con distinto grado de variabilidad. Por ejemplo, las imágenes Baboon, Kodim08 y Cafe tienen una gran variabilidad (contienen muchos contornos o regiones con textura) mientras que Kodim03, Artificial o Bridge tienen una baja variabilidad (tienen pocos contornos o grandes regiones uniformes).

4.2. Resultados con codificador Rice fijo

En este sección se analizan los resultados obtenidos al usar el codificador predictivo con codificación Rice fija. La tabla 1 muestra la tasa binaria obtenida para cada imagen test y cada valor de k del código Rice ($0 \leq k \leq 7$). En el resto de este capítulo, la menor tasa binaria obtenida en cada imagen se denota como R^* (mostrada en negrita en las tablas) y el valor de k en el que se obtiene dicha tasa se denota como k^*.

k	0	1	2	3	4	5	6
Baboon	30.73	16.62	10.07	7.30	**6.43**	6.54	7.15
Peppers	13.96	8.25	5.89	**5.23**	6.44	6.12	7.03
Kodim03	7.75	5.19	**4.41**	4.56	5.21	6.06	7.01
Kodim04	12.05	7.32	5.45	**5.02**	5.38	6.11	7.03
Kodim08	33.53	18.03	10.79	7.67	**6.64**	6.68	7.26
Kodim09	11.65	7.11	5.35	**4.98**	5.38	6.14	7.05
Kodim19	18.50	10.53	7.05	5.82	**5.77**	6.29	7.10
Kodim23	8.31	5.44	**4.52**	4.59	5.21	6.07	7.02
Artificial	3.23	**3.09**	3.51	4.24	5.10	6.04	7.01
Big Building	10.23	6.41	5.00	**4.84**	5.31	6.09	7.01
Big Tree	8.27	5.42	**4.49**	4.56	5.17	6.04	7.01
Bridge	10.45	6.50	5.02	**4.80**	5.24	6.05	7.01
Bike	19.10	10.82	7.19	5.89	**5.80**	6.30	7.10
Cafe	40.22	21.37	12.46	8.52	7.07	**6.12**	7.02
P01	12.81	5.63	5.61	**5.10**	5.40	6.17	7.02
P22	10.70	5.68	5.09	**4.86**	5.30	6.09	7.02

Tabla 1: Tasa binaria (en bpp) de cada imagen al usar predicción fija y codificación Rice fija.

Observamos en la tabla 1 que, como cabía esperar, dada una resolución espacial, cuanto mayor es la variabilidad, mayor es el valor de R^*. Así, entre las dos imágenes de 512×512 píxeles, $R^* = 6{,}43$ bpp en Baboon (que es la que tiene mayor variabilidad de las dos) mientras que $R^* = 5{,}23$ bpp en Peppers. Algo similar ocurre con las imágenes Cafe y P22, ambas de 1600×1280 píxeles. Así, $R^* = 6{,}12$ bpp en Cafe (que es la que tiene mayor variabilidad) mientras $R^* = 4{,}86$ bpp en P22.

Otro aspecto a resaltar es que cuanto mayor es la variabilidad, mayor es generalmente el valor de k^*. Por ejemplo, $k^* = 5$ en Cafe y $k^* = 4$ en Kodim08 (ambas con gran variabilidad), mientras que $k^* = 1$ en Artificial y $k^* = 2$ en Kodim03 (que tienen baja variabilidad). La razón es que cuanto mayor es la variabilidad, mayor dispersión hay en los valores del error de predicción $e[n]$ (como vimos en la sección 2.4), y consecuentemente, mayor tiene que ser el valor del parámetro k del código Rice para que la codificación del error sea eficiente.

Es interesante observar que conforme k aumenta, las tasas binarias obtenidas tienden al valor $k + 1$. Por ejemplo, en $k = 5$ los valores de R varían entre 6.04 bpp y 6.68 bpp mientras que en $k = 6$ varían entre 7.01 bpp y 7.26 bpp. Como en un código RE(k), los enteros $-2^{k-1}, -2^{k-1} + 1, \ldots, 2^{k-1} - 1$ se codifican con palabras de $k+1$ bits, cuando k es grande hay un gran número de valores de error e entorno a cero que se representan con palabras de $k + 1$ bits, y dado que dichos valores acapararán casi toda la probabilidad en *todas* las imágenes, las tasas binarias obtenidas también son similares.

Esta tabla también muestra que la elección de un valor de k próximo a k^* es importante a la hora de obtener una codificación eficiente. Esto es especialmente crítico en imágenes con gran variabilidad, en las que el uso de una k ligeramente distinta a k^* puede proporcionar tasas muy superiores a R^* (e incluso superiores a la tasa binaria de la imagen original). Tal es el caso de Baboon y Kodim08, en las que $k^* = 4$ y el uso de $k = 2$ proporciona tasas superiores a 10 bpp.

La tabla 2 muestra la tasa binaria obtenida para cada imagen test y cada valor de k del código Rice ($0 \leq k \leq 7$) cuando se utiliza el predictor adaptativo. Como cabía esperar, la tasa binaria obtenida para cada imagen y cada k es generalmente inferior a la obtenida con el predictor fijo (tabla 1). Comparando los valores de R^* de las tablas 1 y 2, observamos que el uso de predicción fija proporciona reducciones en R^* que van de 0.13 bpp en Cafe hasta 0.96 bpp en Kodim08. Como la predicción adaptativa proporciona errores de predicción de menor varianza que la predicción fija, en cada imagen el k^* obtenido con predicción adaptativa es menor o igual que el k^* obtenido con predicción fija. De hecho, mientras con predicción fija el valor de k^* más frecuente es $k^* = 3$, en la predicción adaptativa es $k^{=}2$. Gran parte de los comentarios sobre la tabla 1 son igualmente aplicables a la tabla 1. Por ejemplo, al igual que ocurría con predicción fija, el uso de un k muy distinto al de k^* puede reducir considerablemente la eficiencia de la codificación.

k	0	1	2	3	4	5	6
Baboon	29.17	15.84	9.68	7.11	**6.34**	6.49	7.13
Peppers	11.99	7.26	5.40	**4.98**	5.31	6.06	7.01
Kodim03	7.02	4.82	**4.22**	4.46	5.17	6.05	7.01
Kodim04	9.03	5.80	4.69	**4.66**	5.22	6.05	7.01
Kodim08	17.79	10.17	6.86	5.72	**5.68**	6.23	7.06
Kodim09	7.93	5.26	**4.42**	4.52	5.15	6.04	7.01
Kodim19	11.45	7.01	5.28	**4.94**	5.34	6.10	7.02
Kodim23	6.37	4.48	**4.04**	4.37	5.11	6.04	7.01
Artificial	**2.77**	2.86	3.40	4.18	5.07	6.02	7.01
Big Building	6.64	4.62	**4.11**	4.42	5.13	6.02	7.00
Big Tree	6.05	4.31	**3.94**	4.30	5.06	6.01	7.00
Bridge	7.65	5.10	**4.24**	4.47	5.10	6.01	7.00
Bike	15.13	8.84	6.20	**5.40**	5.56	6.20	7.06
Cafe	26.92	14.73	9.14	6.86	**6.25**	6.49	7.16
P01	7.93	5.25	**4.41**	4.53	5.17	6.04	7.01
P22	8.76	5.65	4.61	**4.26**	5.20	6.05	7.01

Tabla 2: Tasa binaria (en bpp) de cada imagen al usar predicción adaptativa y codificación Rice fija.

Por tanto, como acabamos de demostrar es necesario elegir un valor de k cercano al valor óptimo para obtener una codificación eficiente. Una opción para hallar este valor es la ya mencionada en el capítulo 3 codificación de dos pasos.

Estudiemos ahora la eficiencia de este tipo de codificador. La tabla 3 muestra para cada imagen: la entropía del error de predicción $H(e)$, el valor de k obtenido por el algoritmo, la tasa binaria obtenida cuando $e[n]$ se codifica un código Rice (R_R) y la tasa binaria obtenida cuando se codifica $e[n]$ con un código Huffman (R_H). Los valores de la parte izquierda de la tabla son los obtenidos al usar el predictor fijo y los de la parte derecha son los obtenidos al usar el predictor adaptativo.

	Predictor fijo				Predictor adaptativo			
Imagen	$H(e)$	k	R_R	R_H	$H(e)$	k	R_R	R_H
Baboon	6.35	4	6.43	6.38	6.27	4	6.34	6.30
Peppers	5.10	3	5.23	5.12	4.94	3	4.98	4.96
Kodim03	4.01	2	4.41	4.03	3.91	2	4.22	3.94
Kodim04	4.82	3	5.02	4.85	4.46	3	4.66	4.49
Kodim08	6.26	5	6.68	6.30	5.45	4	5.68	5.49
Kodim09	4.56	3	4.98	4.60	4.24	2	4.42	4.27
Kodim19	5.31	4	5.77	5.34	4.82	3	4.94	4.85
Kodim23	4.19	2	4.52	4.23	3.83	2	4.04	3.85
Artificial	1.15	1	3.08	1.63	1.11	0	2.77	1.59
Big Building	4.56	3	4.84	4.60	3.95	2	4.11	3.98
Big Tree	4.33	2	4.49	4.36	4.27	2	4.33	4.30
Bridge	4.74	3	4.80	4.77	4.27	2	4.33	4.30
Bike	5.35	4	5.80	5.36	5.08	3	5.40	5.11
Cafe	6.45	5	6.89	6.48	5.95	4	6.25	6.01
P01	5.00	3	5.10	5.03	4.25	2	4.41	4.28
P22	4.68	3	4.86	4.70	4.41	2	4.61	4.43

Tabla 3: Codificación en dos pasos. En cada imagen se muestra: $H(e)$, k, R_R (código Rice) y R_H (código Huffman). Los valores de la izquierda son los obtenidos con el predictor fijo y los de la derecha los obtenidos con el predictor adaptativo.

Al comparar el valor de k obtenido con el algoritmo de dos pasos, con el valor de k óptimo de las tablas 1 y 2 comprobamos que ambos coinciden excepto en dos casos: en la imagen Kodim08 cuando el predictor es fijo ($k = 5$ en el algoritmo de dos pasos mientras que $k^* = 4$) y en la imagen P22 cuando el predictor es adaptativo ($k = 2$ en el algoritmo de dos pasos mientras que $k^* = 3$). Dado que $|k - k^*| = 1$ en estos dos casos, podemos concluir que el algoritmo de dos pasos obtiene de manera automática y con precisión el valor de k de la imagen a codificar.

Como cabría esperar, para cada tipo de predictor e imagen, la tasa obtenida al usar el código Huffman es mayor que la entropía, y la tasa binaria al usar el código Rice es mayor que la que proporciona el código Huffman. En ambos predictores, la diferencia entre el valor de R_H y $H(e)$ es muy pequeño en todas las imágenes excepto en Artificial. En cambio, el valor de $R_R - R_H$ fluctúa mucho dependiendo de la imagen considerada. Aquellas imágenes en las que la distribución de $e[n]$ puede modelarse bien mediante una distribución TSGD con un valor adecuado de ρ, el código Rice (con un k adecuado) se parecerá al código Huffman y, consecuentemente, $R_R - R_H$ será pequeña. En todo caso, salvo en la imagen Artifical, la pérdida de eficiencia al usar codificación Rice en lugar de codificación Huffman no es muy grande. En contrapartida, la codificación Rice solo necesita obtener el valor de un parámetro (k) y codifica cada valor de $e[n]$ utilizando una aritmética muy simple. Por el contrario, la codificación Huffman necesita obtener la frecuencia relativa de los 511 posibles valores que puede tomar el error de predicción; a partir de ellos, construir el código; y finalmente, mantener en memoria el código obtenido para realizar la codificación. De ahí que en aplicaciones en las que la simplicidad sea importante, se prefiera la codificación Rice.

4.3. Eficiencia con codificación Rice adaptativa

El problema de la codificación de dos pasos es, como se ha mencionado anteriormente, que requiere tener toda la imagen en memoria para poder obtener el valor de k (primer paso). Aparte de esto, no se puede empezar a codificar (segundo paso) hasta que el codificador ha obtenido el valor de *todos* los píxeles de la imagen. La alternativa que requiere una cantidad ínfima de memoria y que no introduce retardo como se menciono en el capítulo 3 consiste en utilizar codificación Rice adaptativa (sección 2.3.5.2). En este tipo de codificación, solo se necesita conocer el valor de un píxel para obtener su correspondiente palabra código. Otra ventaja de la codificación Rice adaptativa es que se adapta automáticamente a los cambios en la función de probabilidad del error de predicción al recorrer la imagen.

La tabla 4 muestra los resultados obtenidos al utilizar codificación Rice adaptativa. Para cada imagen, la tabla muestra la tasa binaria utilizando el codificador adaptativo (R_a), la entropía del error de predicción $H(e)$, la tasa binaria al utilizar codificación Rice fija con el valor óptimo de k y la tasa binaria al utilizar el código Huffman. Los valores de la parte izquierda de la tabla son los obtenidos al usar el predictor fijo y los de la parte derecha son los obtenidos al usar el predictor adaptativo. Los valores de $H(e)$ y R_H son los mismos que los de la tabla 3 puesto que estos valores dependen exclusivamente del error de predicción.

En la tabla 4 observamos que la codificación Rice adaptativa proporciona tasas binarias inferiores a las obtenidas con el codificador Rice fijo con parámetro óptimo y, generalmente, inferiores a los proporcionados por la codificación Huffman. En algunas imágenes, la tasa binaria con codificación Rice adaptativa es inferior a $H(e)$. Esto es debido a que $H(e)$ es una cota inferior en la codificación de una fuente estacionaria cuando se utiliza un código. La codificación adaptativa no codifica la secuencia error con *un* código; implementa un código distinto para cada posible valor de k y codifica cada muestra de $e[n]$ con aquel código que es más eficiente.

	Predictor fijo				Predictor adaptativo			
Imagen	R_A	$H(e)$	$R_R(k*)$	R_H	R_A	$H(e)$	$R_R(k*)$	R_H
Baboon	6.24	6.35	6.43	6.38	6.16	6.27	6.34	6.30
Peppers	5.18	5.10	5.23	5.12	4.97	4.94	4.98	4.96
Kodim03	3.97	4.01	4.41	4.03	3.79	3.91	4.22	3.94
Kodim04	4.79	4.82	5.02	4.85	4.32	4.46	4.66	4.45
Kodim08	6.23	6.26	6.68	6.30	5.43	5.45	5.68	5.49
Kodim09	4.61	4.56	4.98	4.60	4.19	4.24	4.42	4.27
Kodim19	5.08	5.31	5.77	5.34	4.66	4.82	4.94	4.85
Kodim23	4.19	4.19	4.52	4.23	3.75	3.83	4.04	3.86
Artificial	2.21	1.15	3.08	1.63	1.97	1.11	2.77	1.60
Big Building	4.30	4.56	4.84	4.50	3.73	3.95	4.11	3.98
Big Tree	4.33	4.33	4.49	4.36	3.96	3.89	3.94	3.92
Bridge	4.71	4.74	4.80	4.77	4.30	4.27	4.33	4.30
Bike	5.44	5.35	5.80	5.36	5.07	5.08	5.40	5.11
Cafe	6.49	6.45	6.89	6.48	5.91	5.95	6.25	6.01
P01	4.94	4.50	5.10	5.04	4.19	4.25	4.41	4.28
P22	4.50	4.68	4.86	4.70	4.24	4.41	4.61	4.43

Tabla 4: Comparación entre la entropía de $e[n]$ ($H(e)$), la tasa binaria obtenida con el codificador adaptativo (R_a), la obtenida con el codificador fijo con el valor de k óptimo de cada imagen $R_R(k*)$, y la obtenida con el código Huffman (R_H). Los valores de la izquierda son los obtenidos con el predictor fijo y los de la derecha los obtenidos con el predictor adaptativo.

Capítulo 5

Conclusiones y líneas futuras

A través de los distintos capítulos de este documento se ha proporcionado un estudio de distintas configuraciones de predictor y codificador Rice. Ahora vamos a analizar las ventajas y desventajas respecto a la tasa binaria y la complejidad de implementación de cada una de las configuraciones.

En primero lugar, se ha comentado la implementación del codificador fijo con un predictor de primera diferencia. Utilizando una k no óptima para una imagen la codificación no es eficiente. En caso contrario, la eficiencia del codificador es mayor pero seguimos obteniendo unos resultados de tasa binaria mayores a los que se obtendría con el codificador Huffman. Si utilizamos el codificador de dos pasos, como elige la $k*$ para cada imagen, la diferencia de tasas entre el codificador fijo y Huffman se reduce. Sin embargo, la tasa binaria sigue siendo mayor y por tanto, la complejidad de implementación que conlleva debido a que tiene que leer la imagen completa antes de codificarla no compensa.

Analizando los resultados obtenidos tras utilizar el predictor de JPEG-LS observamos que la tasa binaria es menor que utilizando un predictor fijo. Es más, en algunas de las imágenes (Baboon, Peppers, Big Tree, y Bridge) esta tasa es prácticamente igual a la tasa binaria del codificador Huffman . Por tanto, si no quisiéramos gastar recursos de memoria en el hardware o software donde el codificador se implementase, sería una buena opción utilizar un codificador de dos pasos (puesto que si no utilizamos la $k*$ el codificador no es eficiente) con un predictor adaptativo como el utilizado en el proyecto.

Finalmente, se ha analizado el codificador adaptativo tanto con el predictor fijo como con el adaptativo. En ambos casos hemos obtenido valores de tasa binaria menores a la tasa binaria del código Huffman. Si empleamos el predictor adaptativo la tasa binaria se reduce pero la implementación de este es un poco más compleja. Sin embargo, la reducción es significativa en la mayoría de imágenes como en Kodim19, Artifcial, y P01.

Por todo ello, podemos concluir que si queremos codificar imágenes sin pérdidas es una buena opción utilizar un predictor adaptativo y codificación Rice adaptativa, siempre y cuando nuestra prioridad sea el mínimo valor de tasa binaria, y la complejidad del algoritmo no sea relevante. Además, cabe recordar que no necesita tablas de memoria a diferencia del código Huffman.

Como opinión personal, ha sido un trabajo complejo debido al gran tiempo que se ha dedicado en el estudio de la codificación y los predictores, además de la implementación de estos en MATLAB. Pero los resultados obtenidos son relevantes y dan pie a nuevos estudios.

La codificación Rice, como cualquier CLV que sea eficiente, asigna palabras muy largas a aquellos símbolos que son altamente improbables. En algunas aplicaciones es imprescindible que la longitud máxima del código sea inferior a un determinado valor. Una manera habitual de conseguirlo es utilizando un código especial para los símbolos muy poco probables en el que la longitud de sus palabras sea similar. Una posible línea futura de este TFG consistiría en incluir este código especial en el código Rice y evaluar la pérdida de eficiencia que sufre dicho código respecto a una codificación Rice pura.

Otra opción es la implementación de este codificador predictivo en otro lenguaje como C++, o en un hardware específico, ya que el tiempo de ejecución de los algoritmos en MATLAB es muy elevado.

La codificación predictiva es muy vulnerable a los errores que introduce un canal de comunicaciones porque la codificación de un error no sólo afecta a esa palabra código si no también a las siguientes palabras código. Por ello, se podría hacer un estudio en el impacto de los errores que introduce la codificación del canal en este tipo de codificación.

Bibliografía

[1] Thomas M. Cover y Joy A. Thomas. *Elements of Information Theory*. Wiley, 2006.

[2] Micahel Taubman y Michael W. Marcellin. *JPEG2000. Image compression fundamentals, standards and practice*. Kluwer, 2002.

[3] N. S. Jayant y Peter Noll. *Digital Coding of waveformins*. Prentice-Hall, 1984.