LARUNBAT

Codificación de video utilizando técnicas de cuantificación vectorial

Documentación del proyecto de grado

Adriana Piazza
Flavio Caldeiro
Federico Lecumberry

29 de diciembre de 2000

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Versión corregida por los integrantes
de la mesa examinadora.

Tabla de contenido

Índice de Figuras

Parte I

Introducción

Hoy en día, las necesidades de transmisión y almacenamiento de la información visual (la imagen y el video) son cada vez más importantes. Existen actualmente muchos canales de muy bajo ancho de banda (como por ejemplo: la red telefónica pública, la telefonía celular, los canales de radio, etc), la red más difundida a nivel mundial es la red telefónica cuyo ancho de banda es de 3.4 kHz. La transmisión de video más difundida, que es la señal de televisión, requiere un ancho de banda aproximado de 6 MHz, lo que lo hace prohibitivo para la mayoría de las aplicaciones de interés.

Entonces, es necesario desarrollar algoritmos de compresión para que la transmisión de video sobre redes existentes sea factible a costos razonables.

Los objetivos del proyecto son:

- el estudio de los conceptos básicos y las técnicas de codificación de video existentes. En particular *Transformada Discreta del Coseno*, *Codificación de longitud variable*, estándar *MPEG*.
- características de la cuantificación vectorial aplicada a imágenes fijas y secuencias de video. Para esto será necesario un estudio del estado del arte en cuantificación vectorial.
- estudio e implementación de diferentes esquemas de estimación de movimiento.
- estudio de la compresión de imágenes (fijas y en movimiento) usando una combinación de *DCT* y *VQ*. Es de interés investigar como interaccionan estos métodos.

Para finalizar se procedería a la integración y adaptación del trabajo hecho anteriormente para implementar un sistema de codificación de video para bajo bit rate, sujeto a otros requerimientos a definir según la aplicación que se le quiera dar (calidad de imagen, operación en tiempo real, etc).

La idea es utilizar las técnicas de cuantificación vectorial como innovación fundamental respecto de los sistemas de codificación de video convencionales (MPEG, H.263, etc.).

Hoy se plantea un objetivo: lograr una aplicación que obtenga imágenes de buena calidad con fines de almacenamiento o de transmisión vía web, sin requerimientos de codificación en tiempo real aunque sí se exigirá decodificación en tiempo real.

Nuestro codificador se basará en el esquema de bloques utilizado por el estándar MPEG. Tendrá como base de su funcionamiento las *Técnicas*

de Cuantificación Vectorial, *Transformada Discreta del Coseno* (DCT), la *Compensación de Movimiento*(MC) y la *Codificación de Entropia*.

Para reducir la redundancia espacial de las imágenes naturales utilizamos la cuantificación vectorial. Para cuantificar vectorialmente la representación de una imagen se divide la misma en vectores. Éstos luego se codifican eligiendo el vector más parecido dentro de un conjunto finito de vectores previamente elegidos (diccionario o codebook). La idea es que tendremos regiones dentro de la imagen donde habrá poca variación, entonces es probable que con un codebook relativamente chico podremos lograr codificar la imagen sin sacrificar mucha calidad (si los vectores que lo integran están bien elegidos). Las técnicas para diseñar este codebook y para codificar los vectores de la imagen de forma rápida y eficiente han sido profundamente investigados en las últimas dos décadas y ocupan una parte importante de este trabajo. Su idoneidad se basa principalmente en los altos índices de compresión que se alcanzan y en su sencilla decodificación (que consiste simplemente en la extracción de un valor de la tabla que contiene al diccionario).

Por otra parte, en una secuencia de video podemos observar que dos cuadros consecutivos son, en general, muy parecidos (una clara excepción a esto se da cuando se produce un cambio de escena. Una primera aproximación para aprovechar esta redundancia temporal podría ser transmitir solamente las diferencias entre una imagen y la anterior.

Pero podemos hacer algo mejor que esto: si estimamos cuanto se han movido los objetos de la imagen (con respecto al cuadro anterior) y transmitimos esta información, el decodificador podrá obtener una buena reconstrucción de la imagen. De este modo sólo será necesario transmitir las diferencias entre la imagen verdadera y la reconstrucción que ha podido obtener el decodificador. Las técnicas para lograr una buena estimación/compensación de movimiento serán objeto de amplio estudio en este trabajo.

Es sabido que las bajas frecuencias son percibidas mucho mejor que las altas frecuencias. De ahí que, sea posible recortar el envío de información de alta frecuencia afectando mínimamente la calidad de la imagen decodificada. Esto justifica la aparición de la transformada discreta del coseno en el codificador de video (ya está presente en el estándar MPEG). Esta transformada discrimina la información en frecuencias espaciales lo que nos permite decidir con que precisión se transmitirá cada banda de frecuencia. Las imágenes naturales presentan una gran concentración de la energía de la imagen en las bajas frecuencias y muy poca información en las altas frecuencias. Estas altas frecuencias son las primeras que se sacrifican en aras de lograr una mejor compresión pues son las que se detectan con mayor dificultad por el sistema visual humano.

Para terminar la introducción de este trabajo, se destaca la importancia de los métodos de codificación de entropía que se aplican a la salida del codificador. Este tipo de compresión, a diferencia de los anteriores es aplicable a cualquier tipo de datos (muestras de audio, video, etc.)

La cuantificación vectorial (VQ), la transformada discreta del coseno (DCT) y la estimación/compensación de movimiento (MC) son los tres grandes pilares en que se basa la reducción de la información de una secuencia de imágenes que se debe transmitir o almacenar. Resulta interesante, destacar que los bloques de la DCT y de la estimación/compensación de movimiento no reducen la información, solamente la preparan para que la codificación vectorial y de entropía puedan efectuar una importante compresión sin perder calidad de imagen.

Un poco de historia

Al momento de empezar nuestro proyecto, en el Instituto de Ingeniería Eléctrica se estaba desarrollando una línea de investigación basada en la codificación de imágenes fijas vectorialmente. Se basaba en dividir la imagen en bloques no solapados, ordenar los píxeles de cada uno en un vector y codificar éstos utilizando la cuantificación vectorial. Si bien los resultados obtenidos fueron buenos no se logró superar al estándar JPEG.
Los trabajos llevados a cabo siguiendo esa línea de investigación fueron fundamentales durante los primeros meses de trabajo.
Los resultados obtenidos y la documentación de la investigación llevada cabo hasta el momento en que empezamos nuestro proyecto se pueden encontrar en el apéndice A.

Líneas de Trabajo a Desarrollar

El presente proyecto se dividió en dos grandes partes:

- Codificación de imágenes fijas
  - *VQ*
    Estudio e implementación del algoritmo de *Lloyd* utilizado para hallar *codebooks* óptimos y también de los algoritmos que proporcionan *codebooks* que sirven como puntos de partida para éste.
  - *DCT + VQ*
    Se trata de codificar la imagen utilizando la DCT y emplear VQ para cuantificar los coeficientes de la misma. Esto implica un trabajo con información perceptual y estudio de la importancia de los coeficientes de la DCT para la evaluación subjetiva.
- Codificación de Video
  - *Compensación de Movimiento (MC)* Estudio de las diferentes estrategias para la estimación y la compensación del movimiento. Búsquedas rápidas, optimizaciones de la velocidad, algoritmos de *block matching*, algoritmos con tamaño de Macrobloque variables, estimación independiente de la iluminación. Además se deja abierta la posibilidad de futuros estudios en el área de la compensación de movimiento (nuevos esquemas que mejoren el"block matching").
  - *Cuantificación de la Continua*
    Algoritmos de codificación escalares que compriman sin pérdidas o con muy poca degradación de la imagen.
  - *Codificación adaptativa*
    Dado que en el tiempo la estadística de la señal podría modificarse, es de interés estudiar cómo modificar paulatinamente el codificador y el decodificador de forma de seguir la estadística de la señal sin tener que inicializar el sistema periódicamente con el retardo y la carga de información a transmitir que esto requeriría.
  - *Codificación de entropía*
    Codificación con largo de código variable (VLC), algoritmos de compresión sin pérdidas para señales en general.
  - *MC + DCT + VQ*
    Se trata en este punto de utilizar lo aprendido en puntos anteriores en la búsqueda de un esquema final de codificación de video. Ahora aplicaríamos la DCT a la diferencia entre la imagen original y la recuperada después de compensar el movimiento, y son los coeficientes de la DCT los que se codifican vectorialmente.

La primera etapa sirvió de base para poder atacar luego la codificación de video, objetivo principal de nuestro proyecto. Fue fundamental para entender las características de la codificación vectorial y para optimizar algunos de los algoritmos más importantes de la misma que nos permitieron lograr una buena calidad de imágenes en movimiento más rápidamente que si hubiéramos intentado atacar todas las facetas del problema al mismo tiempo.

Resumen de la documentación

En el capítulo 1 presentaremos una introducción general a la teoría de la información y los límites de la compresión de las señales en general.

En el capítulo 2 nos extendemos sobre la cuantificación vectorial, base principal de nuestro proyecto. Se presenta un estudio teórico de la cuantificación vectorial y se describen algunos de los algoritmos más interesantes que son específicos de la compresión de imágenes y video.

En los capítulo 3 y 4 se tratan métodos para simplificar la señal a comprimir. Más precisamente en el capítulo 3 se trata sobre la estimación y la compensación de movimiento. Se describen algunos de los algoritmos más comunes y también algoritmos de reciente desarrollo para la estimación de movimiento, y se presentan varias optimizaciones que se implementaron en ellos. En el capítulo 4 se trata la Transformada Discreta del Coseno.

En el capítulo 5 se presenta la estructura del codificador/decodificador implementado en el presente trabajo.

En los capítulos 6, 7, 8 y 9 se desarrollan las implementaciones particulares de los diferentes bloques constitutivos del codificador/decodificador. En el capítulo 6 se hace un estudio comparativo de los resultados que se obtuvieron con los diferentes algoritmos de estimación de movimiento. En el capítulo 7 se trata la codificación vectorial aplicada a los coeficientes de la DCT. En los capítulos 8 y 9 se tratan la codificación de los coeficientes de continua y la codificación de entropía, respectivamente.

El capítulo 10 presenta las pruebas que se realizaron para evaluar la performance del codificador de video y las conclusiones que se desprenden de ellas y de nuestra experiencia en estos dos años de trabajo en la codificación de video.

En el capítulo 11 se describe el ambiente de desarrollo, las herramientas de software que se utilizaron para su implementación, así como la interfaz gráfica del decodificador, el uso del codificador y la estructura de los archivos de entrada y salida.

Finalmente, se presentan las líneas de trabajo que se abren a partir de este estudio, base para posibles futuras investigaciones en el tema.

Los antecedentes de este proyecto se encuentran en el apéndice A donde se presenta la investigación que se había desarrollado en el Instituto de Ingeniería Eléctrica en el momento en que se comenzó el presente trabajo.

Parte II

Bases Teóricas

Capítulo 1

Teoría de la información

1.1 La medida de la información

Una parte crucial dentro de la Teoría de la Información[3] es la medida de la *información*. Entendiendo por *información* los datos que son producidos por la fuente para ser transferidos al destino. Esto implica que antes de la transferencia la *información* no estaba disponible en el destino.

Supongamos, entonces, que la fuente puede enviar N mensajes distintos, que puede seleccionar de un alfabeto, $X = \{x_0, x_1, ...x_{N-1}\}$, dado, donde cada uno de los mensajes tiene una probabilidad $P(x_i) = P_i$, de ser transmitido. La cantidad de *información* asociada con el mensaje x_i, será una función de P_i; más precisamente C.E.Shannon definió, en su trabajo *"A Mathematical Theory of Communications"* en 1948, la medida de la *información* como:

$$I_i = -\log_b P_i = \log_b \frac{1}{P_i},$$

donde la unidad en que se mide depende de la base, b, en que se tome el logaritmo. En caso de que la base sea $b = 2$, la unidad de medida es el *bit*,

$$I_i = -\log_2 P_i = \log_2 \frac{1}{P_i} \quad \text{bits.}$$

La cantidad I_i es llamada la *auto-información* del mensaje x_i.

Es de notar que con esta definición, la *información* asociada a un mensaje x_i es mayor que la *información* asociada con el mensaje x_j, si y sólo si la probabilidad de transmitir el mensaje x_i es menor que la probabilidad de transmitir el mensaje x_j, es decir:

$$I_i > I_j \text{ sii } P_i < P_j.$$

1.2 Entropía

Asumiendo que la fuente de información es estacionaria, es decir que las probabilidades no cambian con el tiempo, que los símbolos (mensajes) que envía son estadísticamente independientes y que se los envía con una tasa de r símbolos por segundo, la cantidad de *información* que produce la fuente en un intervalo cualquiera, es una variable aleatoria discreta que toma valores del conjunto $I = \{I_0, I_1, ..., I_{N-1}\}$.

La *información* transmitida por símbolo promedio viene dada por el primer momento estadístico:

$$H(X) = \sum_{i=0}^{N} P_i I_i = \sum_{i=0}^{N} P_i \log_2 \frac{1}{P_i} \quad \text{bits/símbolo},$$

lo que también es llamado *entropía* de la fuente.

El valor de la *entropía* de la fuente, depende de las probabilidades de los símbolos (mensajes), P_i, y del tamaño, N, del alfabeto, X, de la fuente.

Sin embargo, el valor de la *entropía* de la fuente, siempre varía en el rango,

$$0 \leq H(X) \leq \log N.$$

La cota inferior corresponde al caso en que uno de los mensajes tiene probabilidad $P_j = 1$ de ser enviado y todos los demás mensajes tiene probabilidad $P_i = 0$ con $i \neq j$.

La cota superior corresponde al caso en que todos los símbolos tienen igual probabilidad de ser enviados $P_i = 1/N \ \forall i$, donde tenemos la mayor incertidumbre.

1.3 Tasa de Información

En el caso en que una fuente transmita, con una tasa de r símbolos por segundo, una secuencia de n símbolos, con $n >> 1$, la *información* total que se envía se aproxima a $nH(X)$. Y el tiempo de transmisión de esta secuencia es n/r. Entonces, se deduce, que la *información* se transmite con una tasa promedio de $(nH(X))(r/n) = rH(X)$.

Formalmente, se define la *tasa de transmisión de información* de la fuente como

$$R = rH(X) \quad \text{bits/segundo}.$$

Para una tasa de transmisión de símbolos fija r, y un alfabeto X, fijo, la *tasa de transmisión de información* de la fuente se maximiza cuando se maximiza la *entropía* de la fuente. Y esto sucede cuando los símbolos (mensajes), que transmite la fuente son equiprobables . En este caso la *tasa de transmisión de información* de la fuente queda

$$R = rH(X) = r \log N \quad \text{bits/segundo}.$$

1.4 Transmisión de información sobre un canal continuo

Dadas determinadas características (ancho de banda y relación señal a ruido) de un canal de transmisión se verá un límite teórico para la capacidad máxima de transmisión de información sin errores sobre dicho canal. Este resultado es debido a Hartley y a Shannon y es muy importante como medida de referencia para cualquier sistema de transmisión de información ya que impone un límite superior teórico en la tasa de transferencia de información.

1.4.1 Información

Se supondrá que la señal emitida por la fuente $x(t)$ es de ancho de banda limitado, o sea que queda perfectamente caracterizada por sus muestras; y en forma análoga al caso discreto se define la entropía de la fuente como:

$$H(X) = \int_{-\infty}^{\infty} p(x) log\Big(\frac{1}{p(x)}\Big) dx$$

En un canal continuo la fuente emite una señal $x(t)$ que luego de ser corrompida por el ruido del canal llega al destino como una señal $y(t)$. Análogamente al caso discreto se define el promedio de información mutua como

$$I(X;Y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p_{XY}(x,y) \log\Big(\frac{p(x|y)}{p_X(x)}\Big) dxdy$$

donde $p_X(x)$ es la función densidad de probabilidad, $p_{XY}(x,y)$ es la función de densidad de probabilidad conjunta entre x e y, y $p(x|y)$ es la probabilidad condicional de x dado y. La máxima cantidad de información transferida por muestra de $y(t)$ se define como

$$C_s = \max I(X;Y) \text{ bits/muestra.}$$

Si se supone que el canal tiene un ancho de banda fijo e igual a B, entonces la señal $y(t)$ queda completamente determinada por sus muestras a una frecuencia de muestreo igual $f_s = 2B$. Por lo tanto la máxima tasa de transferencia de información es $C = 2BC_s$ bits/seg.

1.4.2 Teorema fundamental de Shannon

A continuación se enuncia el teorema fundamental de Shannon para canales con ruido, fundamental para la teoría de la información.

Dado un canal de capacidad C y una fuente con una tasa de transferencia de información $R \leq C$, existe un código tal que la salida de la fuente puede ser transmitida sobre el canal con una probabilidad de errores tan

pequeña como se desee. Si por el contrario $R > C$, no es posible transmitir información sin errores.

En el caso particular de un canal con ruido blanco, aditivo y Gaussiano (AWGN) se obtiene la fórmula de Shannon-Hartley:

$$C = B \log \left(1 + SNR_R\right)$$

siendo B el ancho de banda del canal y S/N la relación señal a ruido. Este resultado junto con el Teorema Fundamental de Shannon establece un limite superior para la transmisión confiable de datos sobre un canal AWGN,

$$R <= B \log \left(1 + SNR_R\right)$$

Capítulo 2

Cuantificación vectorial

2.1 Introducción

En esta sección se estudia la cuantificación vectorial haciendo especial énfasis en sus analogías y diferencias con la cuantificación escalar, por eso primero se realiza un breve repaso del caso escalar donde además se plantean muchas definiciones válidas en ambos contextos.
En esta primera sección se dan a conocer una serie de definiciones básicas para el entendimiento de las secciones subsiguientes. Se define un cuantificador escalar como una función que para cada valor de entrada selecciona un número de un conjunto predeterminado de valores posibles. Generalmente los valores de entrada son analógicos y los de salida digitales. Más formalmente se define como una función

$$Q : I\!R \rightarrow \mathcal{C}$$

donde $I\!R$ es el conjunto de los números reales y

$$\mathcal{C} = \{y_1, y_2, \ldots, y_N\} \subset I\!R$$

es el conjunto de salida, denominado *codebook*. Se dice que el tamaño del cuantificador o del *codebook* es N. Se define la resolución o *code rate* del cuantificador escalar como $r = \log_2 N$; este parámetro mide el número necesario de bits para especificar en forma única el valor cuantificado. Además, es una medida de la capacidad de reproducción de una entrada analógica. Si la entrada al cuantificador es una señal muestreada cada T segundos se obtiene que la tasa de bits por segundo a la salida del mismo es $R = \frac{r}{T}$ (bit rate). Cada cuantificador de tamaño N tiene asociado un conjunto de N celdas o intervalos de cuantificación $\mathcal{R}_i$, siendo

$$\mathcal{R}_i \equiv \{x \in I\!R / Q(x) = y_i\}$$

Claramente $\bigcup_i \mathcal{R}_i = I\!R$ y $\mathcal{R}_i \bigcap \mathcal{R}_j = \phi$ para $i \neq j$.
A partir de las definiciones anteriores se nota que un cuantificador queda

determinado por los conjuntos y_i y $\mathcal{R}_i$ con $i = 1 \dots N$. Un cuantificador se dice regular si cada una de sus celdas es un intervalo y además $y_i \in (x_{i-1}, x_i)$.Todo cuantificador escalar puede descomponerse en dos funciones más elementales, un codificador $E : I\!R \to \mathcal{I}$ donde $\mathcal{I} = \{1, 2, \dots, N\}$ y un decodificador $D : \mathcal{I} \to \mathcal{C}$. Por lo tanto, si $Q(x) = y_i$, entonces $E(X) = i$ y $D(i) = y_i$, además, $Q(x) = D(E(x))$.

2.2 Condiciones para la optimalidad

2.2.1 Introducción

En esta sección se trata de diseñar un cuantificador escalar en forma óptima. En primer lugar se ven dos condiciones necesarias que deben cumplir los cuantificadores para ser óptimos. Luego se usan dichas condiciones para obtener un algoritmo que genere el cuantificador (*codebook* y regiones de cuantificación).

2.2.2 Condiciones para la optimalidad

El objetivo fundamental del diseño del cuantificador es seleccionar los niveles de reproducción (*codebook*) y la partición de tal forma que la distancia media sea mínima para una cantidad fija de N niveles.

En general, este problema no tiene una solución cerrada. En particular, para el error cuadrático medio deben hallarse $(y_i, \mathcal{R}_i)$ con $i = 1 \dots N$ para minimizar

$$D = \sum_{i=1}^{N} \int_{\mathcal{R}_i} (x - y_i)^2 f_X(x) dx$$

siendo $f_X(x)$ la distribución de probabilidad de la entrada al cuantificador. En este caso no existe un método para resolver el problema de hallar el cuantificador óptimo, sin embargo, existen dos condiciones necesarias para la optimalidad que se derivan de la descomposición del cuantificador en un codificador y un decodificador. A partir de ellas se desarrollará un algoritmo de diseño. Primero se asume que el decodificador no es modificable y se halla el codificador óptimo, luego se halla el decodificador óptimo para un codificador dado utilizando como medida de distorsión el error cuadrático medio.

2.2.3 El codificador óptimo dado un decodificador

Dado un decodificador, la tarea de hallar el codificador óptimo es equivalente a encontrar la partición óptima dado un *codebook*. Claramente, el codificador óptimo debe ser aquel que mapee las entradas en los niveles de reproducción que tienen menor distorsión respecto de la entrada. Esto quiere decir que la

i-ésima región de la partición debe consistir de todos los valores de entrada más cercanos a cualquier otro valor de salida. Un codificador que cumple esta condición se dice que satisface la condición del vecino más cercano. Formalmente se puede escribir dicha condición de la siguiente forma: dado un *codebook*, $\mathcal{C}$, las regiones de la partición deben ser tales que,

$$\mathcal{R}_i = \{x/\ d(x, y_i) \leq d(x, y_j)\ \forall j \neq i\}$$

o sea, $Q(x) = y_i$ sí y solo sí

$$d(x, y_i) \leq d(x, y_j)\ \forall j \neq i$$

donde $d(\alpha, \beta)$ es la distancia entre los elementos α y β obtenida con una medida adecuada.

Entonces, dado un decodificador, el codificador es de distorsión mínima si $d(x, Q(x)) = \min_{y_i \in \mathcal{C}} d(x, y_i)$. Debe aclararse que la condición vale para cualquier medida de distorsión. Ahora se prueba que la condición del vecino más cercano es suficiente para una codificación óptima dado el decodificador.

Prueba: Dado un cuantificador $Q(x)$ la distorsión es:

$$D = \int d(x, Q(x)) f_X(x) dx \geq \int \min_{y_i \in \mathcal{C}} d(x, y_i) f_X dx$$

y el límite inferior se obtiene cuando se cumple la condición del vecino más cercano. Cuando los valores de entrada equidistan de dos valores del *codebook*, se debe elegir uno de los dos para que no existan ambigüedades. Para resolverlo se usa la convención de asignarle al valor de entrada, el valor de cuantificación que se encuentra a su izquierda o sea, $\mathcal{R}_i = \{x/x_{i-1} < x \leq x_i\}$

Para las medidas de distorsión MAD y MSE, la condición del vecino más cercano implica que, dada una entrada x, el valor de cuantificación se elige para minimizar $|x - y_i|$. Dicho de otra forma, si x cae entre y_{i-1} e y_i , la regla es elegir el nivel más próximo. Esto se consigue eligiendo x_{i-1} como el punto medio entre dos niveles de salida adyacentes.

2.2.4 El decodificador óptimo dado un codificador

Ahora se analiza la segunda condición necesaria de optimalidad. Cuando la medida de distorsión usada es el error cuadrático medio se encuentra que la condición del centroide, que será explicada luego, es necesaria y suficiente para la optimalidad de un decodificador dado un codificador.

La condición del centroide dice que: Dada una partición no degenerada (i.e. ninguna región tiene probabilidad nula) el *codebook* óptimo

para una variable aleatoria X con respecto al error cuadrático medio es $y_i = E[X|X \in \mathcal{R}_i]$.

Prueba: La distorsión media puede escribirse como

$$D = \sum_{i=1}^{N} \int_{\mathcal{R}_i} (x - y_i)^2 f_X(x) dx$$

Se observa que los términos de la suma son independientes, por lo tanto, el mínimo de la suma es la suma de los mínimos de cada uno de los términos. Además

$$\int_{\mathcal{R}_i} (x - y_i)^2 f_X(x) dx = P_j \int_{-\infty}^{\infty} (x - y_i)^2 f_{X/\mathcal{R}_j}(x) dx =$$

$$P_j E\left[(x - y_j)^2 |\ X \in \mathcal{R}_j\right]$$

donde $f_{X/\mathcal{R}_j}$ es la función de distribución condicional de X dado que pertenece a la región $\mathcal{R}_j$ y P_j es la probabilidad de que X pertenezca a $\mathcal{R}_j$. De aquí se puede demostrar que el valor de y_j que minimiza la distorsión es el centroide $E[X|\ X \in \mathcal{R}_j]$. O sea,

$$y_j = \int_{\mathcal{R}_j} x f_{X/\mathcal{R}_j}(x) dx = \frac{\int_{\mathcal{R}_j} x f_X(x) dx}{\int_{\mathcal{R}_j} f_X(x) dx}$$

2.2.5 Condición generalizada del centroide

La condición del vecino más cercano es aplicable tanto al error cuadrático medio como a cualquier otro tipo de medida de distorsión. Sin embargo, para que la condición del centroide sea aplicable a cualquier medida de distorsión se deben hacer algunas modificaciones. Se define el centroide generalizado $cent(\mathcal{R})$ de una variable aleatoria X en una región $\mathcal{R}$ con respecto a una medida de distorsión $d(x, y)$ como el valor de y que minimiza $E[d(x, y)|\ X \in \mathcal{R}]$. A veces esto se escribe como $cent(\mathcal{R}) = \min_y E[X|X \in \mathcal{R}_i]$.

Prueba: $E[d(X, Q(X))] = \sum_j P_j E[D(X, y_i)|\ X \in \mathcal{R}_j]$ y el valor límite se consigue cuando y_i es el centroide. Como observación final se puede decir que para algunas medidas de distorsión el centroide generalizado no es único y pueden haber infinitos valores que satisfagan la definición del centroide.

2.3 Introducción a la Codificación Vectorial

La cuantificación vectorial puede verse como una generalización de la cuantificación de un escalar a la cuantificación de un vector (conjunto ordenado de escalares). Existen interesantes paralelos entre la cuantificación escalar

y la vectorial y muchas de las técnicas de análisis y de diseño son generalizaciones naturales de aquellas ya vistas para el caso escalar.

Sin embargo, el salto de una dimensión a varias dimensiones es en realidad un importante cambio de enfoque y descubre gran cantidad de nuevas ideas, conceptos, técnicas y aplicaciones.

Una diferencia importante entre la cuantificación escalar y la vectorial es la siguiente: mientras que la cuantificación escalar es usada principalmente para la conversión A/D, la cuantificación vectorial es usada junto a un procesamiento digital de señales sofisticado donde en general la señal de entrada ya tiene algún tipo de representación digital y la salida deseada es una versión comprimida de la señal original.
Un vector puede ser usado para describir casi cualquier tipo de patrón, como podría ser un segmento de una señal de voz o de una imagen, simplemente formando un vector de muestras de la forma de onda o de la imagen.
Pero también podría formarse un vector con un conjunto de parámetros usados para representar, por ejemplo, la envolvente espectral de una señal de voz. La cuantificación vectorial puede verse como una forma de reconocimiento de patrones donde un patrón es apareado con un patrón seleccionado de un conjunto almacenado de codewords.
De ahí que la cuantificación vectorial es mucho más que una generalización formal de la cuantificación escalar. En los últimos años se ha convertido en una importante técnica en reconocimiento de voz como, también en compresión de audio y video, y su importancia y sus aplicaciones van en aumento.

2.4 Definiciones Básicas

2.4.1 Cuantificador

Es un mapeo desde un vector en un espacio Euclídeo k-dimensional en un conjunto finito $\mathcal{C}$, conteniendo N puntos de salida (llamados vectores de código o palabras de código).
Esto es:

$$Q : I\!R^k \rightarrow \mathcal{C}$$

donde $\mathcal{C} = \{y_1, y_2, y_3, \ldots, y_N\}$ e $y_i \in I\!R^k$ para cada $i \in J$, con $J = 1, 2 \ldots, N$.
El conjunto $\mathcal{C}$ es llamado *codebook* o código (o diccionario) y tiene tamaño N, significando que tiene N elementos diferentes, siendo cada uno de ellos un vector en $I\!R^k$. También se define la dimensión del espacio de los vectores que lo componen, k como la *dimensión* del *codebook*.
La resolución (o *code rate* o simplemente *rate*) de un cuantificador vectorial es $r = \frac{\log_2 N}{k}$ lo que mide el número de bits por cada componente de vector usado para representar el vector de entrada. El *code rate* nos da

una indicación de la precisión que es alcanzable si el cuantificador está bien diseñado.
Asociado a cada uno de los N puntos del *codebook* tenemos una *región* o *celda*. La i-ésima celda esta definida por:

$$\mathcal{R}_i = \{x \in I\!R^k : Q(x) = y_i\}$$

De la definición de celda, se sigue que $\bigcup_i \mathcal{R}_i = I\!R^k$ y $\mathcal{R}_i \bigcap \mathcal{R}_j \neq \phi$ para $i \neq j$ de manera que las celdas forman una *partición* de $I\!R^k$.

Definición: Un cuantificador vectorial es llamado *regular* si:

1. Cada celda es un conjunto convexo
2. $y_i \in \mathcal{R}_i, \quad \forall i$.

También es conveniente definir un cuantificador vectorial politópico como un cuantificador vectorial regular cuyas celdas consisten en la intersección finita de semiespacios de la forma: $\{x \in \mathcal{R};\ u_\nu \cdot x + \beta_\nu \geq 0\}$ Este tipo de cuantificadores permite la implementación de algoritmos rápidos pues las operaciones a realizar se simplifican notoriamente.
Al igual que los cuantificadores escalares, los vectoriales pueden ser descompuestos en 2 funciones más elementales: la codificación vectorial y la decodificación vectorial.

2.4.2 Codificador

El codificador ε es el mapeo de $I\!R^k$ en el conjunto de índices J:

$$\varepsilon : I\!R^k \rightarrow J$$

Es importante notar que una partición dada de $I\!R^k$, determina completamente la forma como el codificador asignará un índice a cada entrada dada. El codificador no necesita conocer el *codebook* para cumplir su función.

2.4.3 Decodificador

El decodificador D mapea el conjunto de índices J en el conjunto de representaciones $\mathcal{C}$:

$$D : J \rightarrow \mathcal{C}.$$

Análogamente al caso del codificador, dado un *codebook* tenemos perfectamente determinado como el decodificador generará la salida a partir de un índice dado. El procedimiento de decodificación es simplemente un *table-lookup*, no es necesario conocer la geometría de la partición para llevarla a cabo.

En el contexto de un sistema de comunicación digital, el codificador de un cuantificador vectorial selecciona un *codevector* apropiado, el índice i de este *codevector* es transmitido como una palabra binaria al receptor donde el decodificador realiza una búsqueda en su tabla y genera la reproducción y_i, aproximación cuantificada del vector original de entrada.

2.4.4 Generalidad de la cuantificación vectorial

La cuantificación vectorial no es una mera generalización de la cuantificación escalar. De hecho es la mejor solución para la cuantificación de una señal vectorial.
Ninguna técnica puede superar a la cuantificación vectorial, como se demuestra en el siguiente teorema.

Teorema: Para cualquier sistema de codificación dado que mapee una señal vectorial dada en N palabras binarias y reconstruya el vector aproximación desde alguna de estas palabras binarias, existe un cuantificador vectorial con tamaño de *codebook* N que tiene exactamente el mismo rendimiento, esto es, para cualquier entrada produce la misma salida que el sistema de codificaciń dado.

Prueba: Enumere el conjunto de las palabras binarias producidas por nuestro sistema de codificación por los índices 1,2, . . . ,N. Defina el vector y_i como la salida correspondiente a la palabra i-ésima. Defina el *codebook* $\mathcal{C}$ como el conjunto ordenado de *codevectors* y_i. Entonces un decodificador de un cuantificador vectorial cuyo *codebook* sea $\mathcal{C}$ y que mapee la palabra binaria i-ésima en el codevector y_i alcanzará un rendimiento equivalente al decodificador de nuestro sistema de codificación. El codificador del cuantificador vectorial también se define para ser idéntico al codificador de nuestro sistema de codificación.

2.5 Cuantificadores del vecino más cercano

Una clase especial de cuantificadores vectoriales, muy importante, es la de los llamados cuantificadores *Voronoi* o de *vecino más cercano*. Veremos más adelante que un cuantificador vectorial para ser óptimo en el sentido de minimizar la distorsión promedio debe pertenecer a esta clase.
Se define un cuantificador de vecino más cercano como uno cuya partición en celdas viene dada por

$$\mathcal{R}_i = \{x/\ d(x, y_i) \leq d(x, y_j)\ \forall j \in J\}$$

donde el *codebook* está dado por $\mathcal{C} = \{(y_j)\}$.

Una ventaja de estos cuantificadores es que el proceso de codificación no necesita un almacenamiento explícito de la descripción geométrica de las celdas. En lugar de eso, se puede codificar haciendo referencia solamente al *codebook* almacenado.
Uno de los algoritmos más sencillos para codificar es el siguiente:

Regla de codificación del vecino más cercano

Paso 0: Inicializo $d = d_0$, $i = 1$ y $j = 1$.

Paso 1: Calculo $D_j = d(x, y_j)$.

Paso 2: Si $D_j \leq d$: $d = D_j$ y $i = j$.

Paso 3: Si $j \leq N$: $j = j + 1$ y va a paso 1.

Paso 4: Stop. Resultado es el índice i.

El valor de i que resulta del algoritmo nos da la salida del codificador y el valor final de d es la distorsión resultante de codificar x por medio de y_i. El valor inicial, d_0, debe ser mayor que cualquier distorsión esperada y generalmente se hace igual al mayor número posible que puede ser representado por el procesador. En el importante caso particular donde la medida utilizada es el error cuadrático medio, las celdas son politópicas, pero en el caso general, las celdas definidas por este algoritmo no son politópicas ni convexas.

2.6 Condiciones de optimalidad

En esta sección se estudiarán las propiedades de optimalidad de los cuantificadores vectoriales, es decir las extensiones de las propiedades ya estudiadas para el caso escalar.

Estas propiedades son de gran ayuda para el diseño de cuantificadores pues proporcionan condiciones simples que un cuantificador vectorial debe cumplir para ser óptimo y de ellas se deduce una técnica iterativa sencilla para mejorar un cuantificador dado (*Algoritmo de Lloyd*).

La meta principal en el diseño de un cuantificador vectorial es encontrar un *codebook* y una partición que minimicen una medida de distorsión que considere la secuencia completa de vectores a ser codificados durante todo el tiempo de vida del cuantificador.

Esta medida del rendimiento será un promedio estadístico de una medida de distorsión adecuada que, como ya se ha presentado, puede ser expresado como,

$$D = E(X, Q(X)) = \int d(X, Q(X)) f_X(X) dX$$

donde $f_X(X)$ es la *pdf* conjunta del vector X y la integral es sobre todo el espacio k-dimensional.

En el caso discreto se tiene,$D = E(X, Q(X)) = \sum_i d(x_i, Q(x_i)) p_X(x_i)$ donde x_i son los valores cuya probabilidad es distinta de cero.

El objetivo es determinar las condiciones necesarias para que un cuantificador vectorial sea óptimo en el sentido que minimice la distorsión promedio para un tamaño de *codebook* (N), una estadística de la señal de entrada y una medida de la distorsión dados.
Al igual que en el caso escalar encontramos las condiciones necesarias para que un codificador sea óptimo para un determinado decodificador y viceversa. Recuérdese que el codificador está completamente especificado una vez dada la partición de $\mathbb{R}^k$ y el decodificador queda completamente determinado por el *codebook*.

2.6.1 Condición del vecino más cercano

Primero se considera la optimización del codificador para un decodificador fijo.

Para un *codebook* dado, una partición óptima es la que satisface la condición de vecino más cercano: es decir para cada i, todos los puntos más cercanos al vector del código y_i, que a cualquier otro vector del código deben ser asignados a la región $\mathcal{R}_i$.

Para un conjunto de niveles de salida, C, la partición óptima satisface, $Q(x) = y_i$ sólo si $d(x, y_i) \leq d(x, y_j) \forall j$.

Se observa que esta condición es la misma que para los cuantificadores escalares. Si una entrada x es equidistante de dos o más vectores del *codebook*, debe ser asignada a aquel de los vectores con menor subíndice. Esto es una convención pues la medida de la distorsión será la misma para la asignación de x a cualquiera de sus vecinos más cercanos.

Prueba: Para un *codebook* dado, la distorsión promedio (para el caso continuo) puede ser acotada de la siguiente manera

$$D = \int d(x, Q(x)) f_X(x) dx \geq \int \min_{i \in I} d(x, y_i) f_X(x) dx$$

donde I es el conjunto de índices. Este número se alcanza si $Q(x)$ es el vector del código con el cual x genera la mínima distorsión, es decir, si la condición de vecino más cercano es satisfecha.

2.6.2 Condición de centroide

Ahora se estudia el decodificador óptimo dado el codificador.

Se define el centroide, $cent(\mathcal{R})$, de cualquier conjunto con probabilidad distinta de cero como el vector y, si existe, que minimiza la distorsión entre un punto X e y, promediada sobre la distribución de probabilidad de X, dado que X pertenece a R. Esto es,

$$y^* = cent(\mathcal{R})$$

si

$$E(d(X, y^*)/X \in \mathcal{R}) \leq E(d(X, y)/X \in \mathcal{R}) \forall y \in I\!R^k.$$

De ahí que, el centroide es en algún sentido, un representante natural para el conjunto $\mathcal{R}$ y la distorsión de probabilidad asociada a $\mathcal{R}$. Debe hacerse notar en este punto que para algunas medidas de la distorsión el centroide puede no estar definido o no ser único.

Para la medida de distorsión asociada al error cuadrático medio, el centroide de un conjunto $\mathcal{R}$ es simplemente la estimación de mínimos cuadrados de X dado que $X \in \mathcal{R}$.
También es fácil ver que $cent(\mathcal{R}) = E(X/X \in \mathcal{R})$ de manera que la definición de centroide concuerda con la definición de centro de gravedad vista en Mecánica, y en este caso, el centroide es único.

Para el caso discreto, la definición de centroide sigue siendo válida y se aplica si se conoce la función de probabilidad de masa (pmf) para la distribución de entrada. El interés se centrará en el caso en que cada vector tiene la misma probabilidad y la medida de distorsión es la asociada al error cuadrático medio. En este caso el cálculo de centroide se reduce al promedio aritmético:

$$cent(\mathcal{R}) = \frac{1}{\|\mathcal{R}\|} \sum_{i=1}^{\|\mathcal{R}\|} x_i$$

donde $\mathcal{R} = x_i, i = 1, \ldots, \|\mathcal{R}\|$, ($\|\mathcal{R}\|$ es la cardinalidad del conjunto $\mathcal{R}$, es decir, el número de elementos en $\mathcal{R}$).
Para una partición dada el *codebook* optimal satisface $y_i = cent(\mathcal{R}_i)$. Este resultado es una generalización de la condición de centroide hallada para el caso escalar.
Prueba: La distorsión promedio es dada por

$$D = \sum_i \int_{\mathcal{R}_i} d(x, y_i) f_X(x) dx = \sum_{i=1}^{N} p_i \int_{I\!R^k} d(x, y_i) f_{X/i}(x) dx$$

donde $f_{X/i}(x)$ es la función de densidad de probabilidad condicional para x dado que $x \in \mathcal{R}$ y $p_i = P(x \in \mathcal{R})$.

Dado que la partición es fija, minimizamos cada término por separado para minimizar la suma. Se debe hallar el vector y_i que minimiza la distorsión esperada, es decir, el vector que minimiza:

$$E(d(x, y_i)/x \in \mathcal{R}_i) = \int_{\mathbb{R}^k} D(x, y_i) f_{X/i}(x) dx.$$

Por definición, el centroide minimiza esta distorsión condicional.

Estos resultados asumen que la partición no es degenerada en el sentido de que todos las regiones tienen probabilidad distinta de cero de contener al vector de entrada. En el caso degenerado ($P_i = 0$ para algún i), se tiene lo que se ha dado en llamar el "problema de la celda vacía". Para una región con probabilidad cero el centroide no está definido y además no tiene sentido dedicar un vector del *codebook* a representar esta región.
Este problema no sólo presenta interés teórico sino que es un problema que se presenta a menudo en los algoritmos de diseño o adaptación de *codebooks*. Evidentemente cualquier cuantificador con una celda vacía es sub-óptimo. La solución que se toma generalmente, consiste en eliminar la celda vacía y partir la celda con mayor distorsión en dos. De esta manera el tamaño del *codebook* se mantiene constante y la distorsión disminuye.

2.6.3 Condición de probabilidad cero en los bordes

Existe una tercera condición necesaria para la optimalidad de un cuantificador (debida al *Algoritmo de Lloyd*, quién la planteó para el caso escalar) que es útil en el caso discreto.

Esta condición es: $P(\bigcup_{j=1}^{N} B_j) = 0$ donde B_j es el borde de la región $\mathcal{R}_j$, definido como los puntos que están en algún $\mathcal{R}_i$ con $i \neq j$ pero que son tan cercanos a y_j como a y_i (no tienen un vecino más cercano único). En otras palabras la condición de probabilidad cero en los bordes implica que:

$$P(x, d(x, y_i) = d(x, y_j) \text{ para } i \neq j) = 0.$$

Prueba: Se supone que $P(B_j) \neq 0$ para algún j por lo tanto existe por lo menos un punto de entrada (x_0), que es codificado con algún vector y_i, pero es igual de cercano a y_j. Entonces, una nueva partición se puede formar asignando x_0 a y_j en lugar de a y_i. Esta nueva partición tendrá asociada la misma distorsión promedio. Pero, sin embargo, estamos cambiando de celda un punto de entrada con probabilidad distinta de cero, lo cual mueve los centroides de $\mathcal{R}_i$ y $\mathcal{R}_j$, lo que implica que el *codebook* ya no es óptimo para la nueva partición.

La tercera condición se cumple siempre cuando la entrada es una variable aleatoria continua pues para todas las medidas que se han considerado, el

borde tiene volumen cero (y por lo tanto probabilidad cero). Esta condición es útil para el caso discreto pues la probabilidad puede ser colocada en puntos del borde, esto es, un vector de la secuencia de entrenamiento puede ser equidistante de dos vectores del código.

2.7 Diseño de un cuantificador vectorial

Las condiciones necesarias estudiadas para la optimalidad proporcionan las bases para un algoritmo iterativo que mejore un cuantificador vectorial dado. Se ha visto que las condiciones de optimalidad no aseguran que el cuantificador sea globalmente optimal. Por lo tanto, la condición inicial se torna un aspecto importante a tener en cuenta para obtener un buen resultado final: si se parte de un cuantificador bien diseñado la probabilidad de converger al óptimo será mayor.

2.7.1 Técnicas para diseñar cuantificadores

Se empezará estudiando algunas formas de obtener un buen *codebook* inicial. De hecho si éste es lo suficientemente bueno, no valdrá la pena correr algoritmos de mejora.

Random Coding La idea más simple para encontrar un *codebook* de tamaño N es elegir aleatoriamente los vectores del código de acuerdo con la distribución de probabilidad de la fuente, lo que puede ser visto como un diseño *Monte Carlo*. La opción más simple cuando se diseña el *codebook* basándose en una secuencia de entrenamiento es elegir los primeros N vectores. Si la secuencia de entrenamiento es muy correlacionada es mejor elegir entonces, uno de cada K vectores.
Desafortunadamente, este *codebook* no tendrá ninguna estructura útil y podría comportarse bastante mal.

Pruning (podar) Esta técnica se basa en la idea de comenzar con todos los vectores de la secuencia de entrenamiento como candidatos a integrar el *codebook*; y eliminarlos uno a uno según cierto criterio hasta que el conjunto final resulte ser el *codebook*.
Un método posible podría ser el siguiente: se considera el primer vector de la secuencia como el primer vector del *codebook*. Luego se calcula la distorsión entre éste y el siguiente vector de entrenamiento. Si la distorsión es mayor que cierto umbral el vector siguiente se incluye también, sino es desechado. Con cada vector de entrenamiento nuevo se busca su vecino más cercano entre los vectores ya integrados al *codebook*, y, si la distorsión entre ambos es mayor que cierto umbral, el vector se integra al *codebook* sino se desecha. Este paso se repite hasta que se completa el *codebook*. Para una secuencia

de entrenamiento finita puede resultar que no se encuentre el número de vectores necesario; en este caso se debe reducir el umbral de decisión y recomenzar.

Diseño de vecino más cercano dos a dos o clustering Se verá en esta sección otra forma de encontrar el *codebook* partiendo de una secuencia de entrenamiento, que es bastante más complicada que las vistas hasta el momento pero proporciona mucho mejores resultados. Este algoritmo también es una forma de *pruning* desde el punto de vista de que se parte de la totalidad de la secuencia de entrenamiento y se llega a un *codebook* con un tamaño mucho menor, sin embargo, se diferencia en que los vectores que finalmente permanecen en el *codebook* no tienen por qué ser algunos de los originales. Los k vectores de la secuencia de entrenamiento se consideran como vectores de *codebook*, cada uno de ellos con su celda asociada en la partición del espacio de entrada. La meta del algoritmo es agrupar los vectores hasta obtener el número deseado de grupos. El *codebook* contendrá los centroides de estos grupos. Si se logra esto, se tiene una partición de la secuencia de entrenamiento en el correcto número de celdas y tenemos el *codebook* óptimo para esta partición.

La partición se obtiene de la siguiente manera. Primero, se calcula la distorsión entre todos los vectores de la secuencia dos a dos. El par de vectores que tenga entre si la menor distorsión conformarán un solo grupo que será representado por su centroide. Se tiene ahora $k-1$ grupos. En el paso i-ésimo se tendrá $k-i$ grupos cada uno de ellos con uno o varios vectores y se desea unir dos de estos grupos para obtener una partición con $k-(i+1)$ grupos.

La forma de hacerlo de manera óptima es la siguiente: se calcula el incremento en la distorsión que resultaría si dos de los grupos se uniesen y fuesen representados por el correspondiente centroide; este cálculo se realiza para todas las uniones posibles dos a dos. El par de grupos que se elige para unir es el que genera la menor contribución a la distorsión promedio general. Se tiene ahora, un *codebook* con $k-(i+1)$ elementos. Este *codebook* no cumple con la regla del vecino más cercano si bien cumple (por construcción) con la regla del centroide. Se podría entonces reemplazar la partición por la partición inducida por el *codebook* según la regla del vecino más cercano. Se observa que si bien cada unión es óptima, el proceso en conjunto no tiene por qué serlo.

Códigos producto En algunos casos un código producto como punto de partida puede ser una buena elección. Por ejemplo, para diseñar un *codebook* para un cuantificador vectorial k-dimensional, de tamaño 2^{kR} para una

resolución entera R, entonces se podría utilizar el cuantificador producto de k cuantificadores escalares con 2^R palabras cada uno.
En general otras estructuras pueden ser utilizadas, por ejemplo, se podría diseñar un cuantificador escalar $q_1(x)$, después para hallar el cuantificador en dos dimensiones $q_2(x_o, x_1)$ se podría usar $(q_1(x), q_1(x))$ como punto de partida y así sucesivamente hasta obtener el *codebook* de la dimensión deseada.

Splitting Esta técnica se parece a la inicialización por medio de código producto en el sentido de que genera *codebook*s grandes a partir de otros más pequeños, pero difiere en que no requiere un número entero de bits por símbolo y en que produce *codebook* cada vez más grandes pero de la misma dimensión.
Inicialmente se elige el centroide de la secuencia de entrada (y_0) como *codebook* de resolución 0 (con un solo elemento). Esta única palabra puede ser divida en dos palabras de código: y_0 e $y_0 + \epsilon$ donde ϵ es un vector de módulo pequeño.
Este nuevo *codebook* tiene dos elementos por lo cual no puede ser peor que el original. El algoritmo iterativo de mejora (el cual se discutió brevemente al comienzo de esta sección) puede ahora aplicarse a este *codebook* para obtener un buen *codebook* de resolución 1.
Como siguiente paso de divide cada uno de los vectores del *codebook* en dos, repitiendo lo anterior. Se continua de ésta manera, hallando un *codebook* de resolución $r+1$ partiendo de un buen *codebook* de resolución r. Esta técnica provee un algoritmo completo de diseño de *codebook* desde la secuencia de entrenamiento.

2.8 El *Algoritmo de Lloyd* generalizado

Se discute ahora en profundidad un algoritmo iterativo de mejora de *codebook*, este algoritmo generaliza el *Algoritmo de Lloyd* visto para la codificación escalar. Si la iteración continúa hasta la convergencia, un buen cuantificador vectorial (se desea que óptimo) se alcanza.
La iteración comienza con un *codebook* que cumpla la condición de vecino más cercano hallado con alguna de las técnicas ya vistas. Entonces se encuentra un nuevo *codebook* (usando la condición del centroide) que sea optimal para la partición dada, luego encuentra una nueva partición para el nuevo *codebook* (usando la condición de vecino más cercano).
Este nuevo cuantificador vectorial tiene una distorsión menor o igual al original. La aplicación repetitiva de este paso proporciona un algoritmo que reduce o no cambia la distorsión en cada paso. Si bien cada uno de los pasos es óptimo y directo, nada nos asegura que se hallará el cuantificador vectorial optimal, ni siquiera uno que cumpla ambas condiciones a la vez.

Aun así, las aplicaciones prácticas han demostrado ser muy efectivas.

Si se conoce la *pdf* conjunta del vector de entrada y ésta es continua, el *Algoritmo de Lloyd* sería el siguiente:

La iteración de Lloyd para un *codebook* de estadística conocida

1. Dado el *codebook*, $C_m = \{y_j,\, j \in 1, \ldots, N\}$, encuéntrese la partición en celdas, usando la condición de vecino más cercano:

   $$\mathcal{R}_j = \{x/\ d(x, y_j) < d(x, y_i); \forall j \neq i\}$$

   si $d(x, y_j) = d(x, y_i)$ para $j \neq i$ asígnese x a la celda $\mathcal{R}_j$ correspondiente al menor j.
2. Usando la condición de centroide, determine $\mathcal{C}_{m+1} = \{cent(\mathcal{R}_j), j = 1, \ldots, N\}$ *codebook* optimal para las celdas halladas en el paso 1.

Esta forma del *Algoritmo de Lloyd* requiere que la *pdf* y la geometría de la partición sean conocidas. Además, el cálculo de los centroides con una integral múltiple de la *pdf* sobre una región complicada de $\mathbb{R}^k$, es en general, imposible usando métodos analíticos.

Obs: si la *pdf* de entrada tiene regiones con probabilidad cero, es posible que algunos *codebook* enfrenten el problema de la celda vacía y, por consiguiente, que no se puedan calcular algunos centroides.
Por otro lado, en la mayoría de las aplicaciones no se dispone de una descripción analítica de la *pdf* de la entrada, en su lugar se utiliza una distribución de muestras basada en observaciones empíricas del vector de entrada. De echo, el método que se describirá a continuación es equivalente al método *Monte Carlo* para evaluar las integrales que determinarán el centroide. Para este método se necesita un conjunto de entrenamiento compuesto por observaciones de la señal a ser cuantificada: $\tau = \{\nu_1, \nu_2, \ldots, \nu_M\}$.
Se define la relación $\beta = \frac{M}{N}$ donde N, como siempre, es el tamaño del *codebook*.
Este parámetro indica qué tan bien el conjunto de entrenamiento puede describir a la *pdf* real.

La secuencia de entrenamiento puede ser usada para definir un vector aleatorio U por medio de su función de acumulación de probabilidad (no se define a través de la *pdf* pues ésta no está bien definida para variables discretas, a menos que se recurra a la delta de Dirac).

Concretamente, la función de acumulación de probabilidad empírica de U es dada por:

$$F^{(M)}(x) = \frac{1}{M}\sum_{i=1}^{M}\hat{u}(x-\nu_i).$$

Donde $\hat{u}(w)$ representa la función de Heavyside para varias variables definida como:

$$\hat{u}(w) = \begin{cases} 1 & \text{si } \omega_i \geq 0 \ \forall i \\ 0 & \text{en otro caso} \end{cases}$$

La ley fuerte de los grandes números implica que nuestra función $F^{(M)}(x)$ converge con probabilidad 1 a la función de acumulación de probabilidad de la variable de entrada (en todos los puntos en que ésta es continua) cuando M tiende a infinito. Si se encuentra un cuantificador vectorial de tamaño N que es óptimo para un vector de entrada con la $F^{(M)}(x)$ dada por el conjunto de entrenamiento y si β es lo suficientemente grande entonces se puede esperar que el cuantificador vectorial sea cercano al óptimo para el vector de entrada real.

Se definen las funciones de decisión binarias como

$$S_j(x) = \begin{cases} 1 & \text{si } x \in \mathcal{C}_j \\ 0 & \text{en otro caso} \end{cases}$$

Una vez que se han definido estas funciones, puede verse que la condición de centroide de una celda está dada por:

$$Y_j = E(x/x \in \mathcal{R}_j) = \frac{E(x\ S_j(x))}{E(S_j(x))}$$

Por lo tanto para una partición de τ se tiene:

$$Y_j = \frac{\frac{1}{M}\sum_{i=1}^{M}\nu_i S_j(\nu_i)}{\frac{1}{M}\sum_{i=1}^{M}S_j(x)} \quad j = 1,\ldots,N$$

El índice i cuenta los vectores de la secuencia de entrada y el índice j cuenta las celdas de la partición. Nuevamente se observa que el problema de la celda vacía puede darse si una celda $\mathcal{R}_j$ no tiene ningún vector de entrenamiento. La condición de vecino más cercano para una entrada discreta, una vez dada la partición es:

$$\mathcal{R}_j = \{\nu \in \tau/\|\nu - y_j\| \leq \|\nu - y_i\| \ \forall i \neq j\}.$$

Y la distorsión promedio es:

$$D = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{n} d(\nu_i, y_j)S_j(\nu_i)$$

donde la entrada al cuantificador (ν_i) está restringida a los valores pertenecientes a τ y $\mathcal{C} = \{(y_i)\}$ es el *codebook*.

Luego de haber establecido estas definiciones el *Algoritmo de Lloyd* puede ser aplicado directamente a la distribución discreta definida por τ para obtener un cuantificador localmente optimal para esta distribución de probabilidad.

La iteración de Lloyd para datos empíricos

1. Dado el *codebook*, $C_m = \{y_j,\, j \in 1, \ldots, N\}$, se particiona el conjunto de entrenamiento en celdas $\mathcal{R}_j$ usando la condición del vecino más cercano

$$\mathcal{R}_j = \{\nu \in \tau / \parallel \nu - y_j \parallel \leq \parallel \nu - y_i \parallel \forall j \neq i\}$$

   y una regla adecuada para resolver los casos de igualdad.

2. Usando la condición de centroide, se calculan los centroides para la partición anterior para obtener el nuevo *codebook*. Si una celda vacía es generada en el paso 1, una asignación alternativa debe realizarse.

Una gran variedad de soluciones han sido propuestas para solucionar el problema de la celda vacía. Uno de ellos es eliminar la celda vacía y asignarle a la celda de mayor distorsión una segunda palabra de código por la vía de dividir su centroide en 2 vectores como ya se ha visto (splitting). Si hay k celdas vacías se eligen otras k celdas para aplicar este método.

Ahora que se ha definido la iteración de Lloyd se verá como queda el algoritmo:

El algoritmo de Lloyd para datos empíricos

1. **Paso 1-** Se comienza con un *codebook*, C_1. Se impone $m = 1$.
2. **Paso 2-** Dado el *codebook*, C_m, se le aplica la iteración de Lloyd para generar el C_{m+1}.
3. **Paso 3-** Se calcula la distorsión promedio para C_{m+1}. Si el cambio desde la última iteración es menor que cierto umbral $\rightarrow$ **FIN**. Si no es asi, se impone $m = m + 1$ y se vuelve al **paso 2**.

Se observa que el paso 2 podría incluir algún método para lidiar con el problema de la celda vacía. También podría chequearse, antes de salir

del algoritmo, que ningún vector de entrenamiento sea equidistante de dos vectores de código (condición de borde con probabilidad cero).
Si esto sucede el *codebook* resultante puede mejorarse reasignándose ese vector de entrenamiento y corriendo una iteración de Lloyd adicional. La mejora introducida por este último paso es despreciable si un β suficientemente grande es usado, pero si β es muy pequeño este paso es importante.

Muchos criterios de parada pueden ser usados. Uno de los más comunes es chequear si $(1 - \frac{D_{m+1}}{D_m})$ es menor que cierto umbral adecuado.
Si el umbral se fija como cero, se tiene una sucesión de *codebook* cuyos valores de distorsión asociados son no crecientes. Si el algoritmo converge a un *codebook* en el sentido de que sucesivas iteraciones no producen cambios en él, entonces éste debe satisfacer las dos primeras condiciones necesarias de optimalidad.

Para un conjunto de entrenamiento finito, el *Algoritmo de Lloyd* converge en un número finito de pasos. Esto es fácil de ver dado que hay sólo un número finito de particiones del τ posibles, y la distorsión nunca crece, por lo cual, el algoritmo no puede volver a una partición que entregue un valor mayor de distorsión. De ahí que, la distorsión promedio asociada a la sucesión de cuantificadores vectoriales producidos por el *Algoritmo de Lloyd* debe converger en un número finito de pasos.

Si bien, existen muchas otras técnicas de diseño, ésta que hemos estudiado es muy importante pues siempre puede ser utilizada después de cualquier otra técnica de diseño, dado que nunca empeora el rendimiento del *codebook* dado.

Es importante enfatizar que el diseño de un cuantificador vectorial óptimo de vecino más cercano es equivalente a la minimización de una función de muchas variables. El *Algoritmo de Lloyd* es un algoritmo de descenso (pues cada iteración reduce o no modifica la distorsión promedio) y además cada iteración induce un cambio local en el *codebook*, esto es, el nuevo *codebook* no es demasiado diferente al anterior. Esto muestra que una vez que el *codebook* inicial es elegido, el algoritmo nos llevará al mínimo local más cercano a este en el espacio de todos los *codebooks* posibles.

Dado que la función de distorsión es muy compleja, seguramente tendrá muchos mínimos locales por lo que es claro que el *Algoritmo de Lloyd* no localiza el óptimo global y que para obtener un buen resultado es esencial disponer de un buen punto de partida.

Capítulo 3

Métodos para simplificar la señal a comprimir: estimación y compensación de movimiento

3.1 Introducción

En este capítulo se hará una revisión teórica de algunos algoritmos de estimación de movimiento. Un estudio comparativo de los mismos se realiza en el capítulo 6.

El término estimación de movimiento se refiere a la identificación de los movimientos que hay entre dos cuadros mientras que la compensación de movimiento se refiere al uso de esta información para la reconstrucción de las imágenes en la codificación y decodificación de una secuencia de video. Estos movimientos identificados se representan con los vectores de movimiento.

En los sistemas de codificación de video, la estimación de movimiento reduce la redundancia espacial por la vía de explotar la correlación entre imágenes. La hipótesis que se realiza es que los píxeles de una imagen pueden ser modelados como traslaciones de los píxeles de alguna imagen anterior.

Sin embargo, en la realidad ocurren otros tipos de movimientos; en un cuerpo rígido existen movimientos de traslación y de rotación, siendo estos últimos movimientos difíciles de encontrar. También existen cuerpos deformables cuyo análisis es aun más complicado. En la práctica se encuentra que si los movimientos son pequeños, la hipótesis de movimientos traslacionales funciona bastante bien, evitando la carga computacional elevada de otros modelos menos adecuados para aplicaciones de tiempo real.

Si bien muchas medidas de distorsión pueden ser usadas, la más popular de ellas es la suma de los valores absolutos de las diferencias (SAD: sum of absolute differences) debido a su simplicidad y a los buenos resultados que se obtienen. Su expresión (para bloques de $m \times n$) es la siguiente:

$$SAD(x,y) = \sum_{i=0}^{n} \sum_{j=0}^{m} |I_1(x+i, y+j) - I_2(x+i+dx, y+j+dy)|.$$

A veces también se utiliza MAD (MAD: mean of absolute differences) que es igual a SAD/256 (siempre en el caso de bloques 16×16) aunque SAD, al permitir una representación entera, es computacionalmente más eficiente. SAD es la medida que se utiliza en todos los métodos implementados para este estudio.

Los métodos de estimación de movimiento más populares son los que dividen la imagen en bloques y estiman el movimiento de cada uno de estos, conocidos como algoritmos de *block matching*. Estos métodos generalmente predicen el movimiento de bloques de tamaño 16×16 píxeles (macrobloques) que no se solapan a partir de macrobloques en la imagen anterior. Estos métodos asumen que todos los píxeles en el bloque se trasladan la misma cantidad en la misma dirección.

El esquema de todos los algoritmos de estimación de movimiento es calcular la distorsión (definida según cierta medida) entre el bloque a codificar y cierto conjunto de candidatos dentro de un área de búsqueda dada y elegir el candidato que minimiza esta distorsión.

El método más simple de todos es el conocido como búsqueda exhaustiva (full search), que evalúa la SAD en todas las posibles posiciones en una determinada área de búsqueda. Para las implementaciones en software, la carga computacional de este algoritmo es tan grande que es comparable o incluso mayor que la suma de todos los demás pasos necesarios para la codificación. De ahí que, se encuentren en la bibliografía muchos métodos de estimación rápida que reducen la carga computacional. Vale la pena aclarar que cualquier búsqueda que no sea la exhaustiva, si bien será mucho más rápida, puede quedar atrapada en un mínimo local, lo que por supuesto va en contra de la calidad de la imagen recuperada.

Las búsquedas que se han implementado para este proyecto son las siguientes:

- Búsqueda Exhaustiva
- Búsqueda En Tres Pasos

- Búsqueda En Tres Pasos Rápida
- Búsqueda Logarítmica
- Búsqueda En La Dirección Conjugada
- Búsqueda En Capas
- Búsqueda Por El Camino Del Vecino Más Cercano

Todas las búsquedas pueden combinarse con métodos de predicción de los vectores de movimiento. Éstos consisten en estimar el vector de movimiento tomando como base la historia pasada. Si la predicción es buena, será posible restringir el área de búsqueda (que ahora se centrará en el bloque estimado) y seguir obteniendo buenos resultados.

Cabe aclarar, que a los efectos de la compensación, no importa la búsqueda ni el método de predicción que se utilice.

Dentro de los algoritmos de bloques además de la técnica convencional se probaron dos técnicas un poco más sofisticadas: compensación de movimiento con bloques de tamaño variable y compensación de movimiento independiente del valor de la media (que se explicarán con detalle más adelante). Estas técnicas pueden implementarse con cualquiera de las búsquedas y de las predicciones mencionadas, y la elección de éstas dependerá del objetivo perseguido.

Macrobloques Intra

Si la mejor distorsión lograda es mayor que cierto umbral el bloque se codifica como *intra*. Es decir no se refiere su codificación a un bloque anterior sino que se transmite toda la información.

En nuestro caso la elección del umbral se hizo siguiendo el mismo criterio que usa el estándar MPEG, es decir, se eligió como umbral al máximo entre MAD = 16 y el valor que deja un octavo de la distribución de la MAD en la imagen, por arriba. La cota fija para el umbral en 16 evita que se utilice la codificación intra en niveles de distorsión que son bien manejados por la codificación inter. La cota variable impide que se dispare la cantidad de bpp's, al clasificarse muchos macrobloques como intras.

Una diferencia fundamental con el estándar MPEG es que los macrobloques intra se codifican de manera totalmente independiente de los macrobloque diferencia. Por lo tanto, no es necesario restarle al macrobloque intra un macrobloque uniforme de color gris neutro como se hace en el estándar MPEG a fin de mantener la uniformidad en los coeficientes de continua de forma de lograr mayor compresión.

3.2 Búsquedas rápidas

Como se mencionó en la sección anterior, el algoritmo que obtiene la menor distorsión es el de búsqueda exhaustiva, sin embargo es demasiado costoso del punto de vista de las operaciones necesarias para llevarlo a cabo. Esta búsqueda consiste en calcular la distorsión entre el bloque a codificar y todos los posibles bloques dentro de un área de búsqueda dada y quedarse con el candidato que proporcione la menor de ellas. Dicho área es en general un cuadrado con centro en la posición del bloque a codificar (o en la posición estimada por algún método de predicción adecuado). Esta es la búsqueda por excelencia para aplicaciones *off line*. Para aplicaciones en tiempo real, como ya se dijo, el número de cuentas que implica la hace prohibitiva , pero es estándar utilizarla como patrón para evaluar las restantes técnicas. A continuación se presentan algunos de los algoritmos rápidos más conocidos que fueron implementados en el presente proyecto.

Búsqueda en tres pasos

El algoritmo de la Búsqueda En Tres Pasos[2][6], uno de los recomendados por el estándar MPEG, es el más popular de los basados en UESA (unimodal search error asuumption, se hace la suposición de que el error residual de block matching crece monótonamente cuando nos alejamos del mínimo global de este error), debido a su simplicidad y a su performance razonable. En este algoritmo se realizan búsquedas en el centro de cada macrobloque de la imagen de referencia y en ocho posiciones alrededor de su centro en forma de cuadrado como se ve en la Figura 3.1. En cada paso de la búsqueda se reduce el tamaño del cuadrado hasta que el tercer paso se realiza a una distancia de un pixel. En cada paso, la posición de menor distorsión se transforma en el centro del cuadrado para el próximo paso.

Búsqueda en tres pasos rápida

Este algoritmo[7] se basa en el recién descrito y fue propuesto para reducir el número de puntos de chequeo de nueve a un número entre cinco y siete, obteniendo una performance similar. La estrategia de búsqueda divide cada paso de la búsqueda anterior en dos o tres sub-pasos para reducir el número de puntos de chequeo eficientemente.
La estrucutura de cada uno de los pasos es chequear primero sólo tres puntos y según el resultado de este chequeo decidir el segundo sub-paso. La estrategia de búsqueda se representa en la siguiente tabla y se muestra en la Figura 3.2.

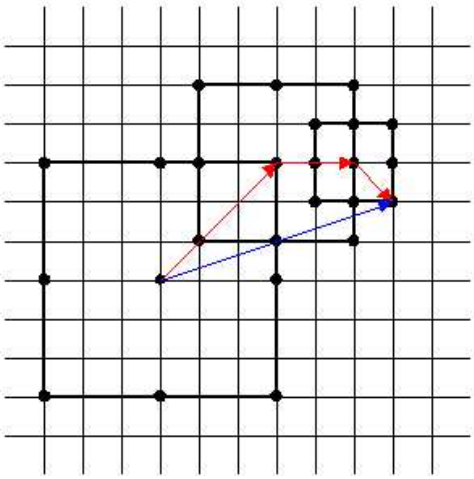

Figura 3.1: Búsqueda en tres pasos

Estrategia de búsqueda en tres pasos rápida.

- **Primer sub-paso:** Calcular SAD(5), SAD(6) y SAD(8).
- **Segundo sub-paso:**
  - Si SAD(6) es mínimo → calcular SAD(3) y SAD(9) y elegir el mínimo entre $\{3, 6$ y $9\}$ **Fin**.
  - Si SAD(8) es mínimo → calcular SAD(7) y SAD(9) y elegir el mínimo entre $\{7, 8$ y $9\}$ **Fin**.
  - Si SAD(5) es mínimo → calcular SAD(2) y SAD (4)
    - Si SAD(2) es mínimo → calcular SAD(1) y SAD(3) y elegir el mínimo entre $\{1, 2$ y $3\}$ **Fin**.
    - Si SAD(4) es mínimo → calcular SAD(1) y SAD(7) y elegir el mínimo entre $\{1, 4$ y $7\}$ **Fin**.
    - Si SAD(5) es minimo → el mínimo es 5. **Fin**.

SAD(i) representa la distorsión entre el bloque de referencia y el candidato en la posición i.

Búsqueda logarítmica

En el algoritmo búsqueda logarítmica[2][6] se realizan búsquedas en la posición de cada Macrobloque y en cuatro posiciones a una distancia de d píxeles del centro (siendo d potencia de dos); d a la derecha, d a la izquierda, d hacia arriba y d hacia abajo como puede verse en la Figura 3.3. En cada paso, la posición de menor distorsión se transforma en el centro del cuadrado para

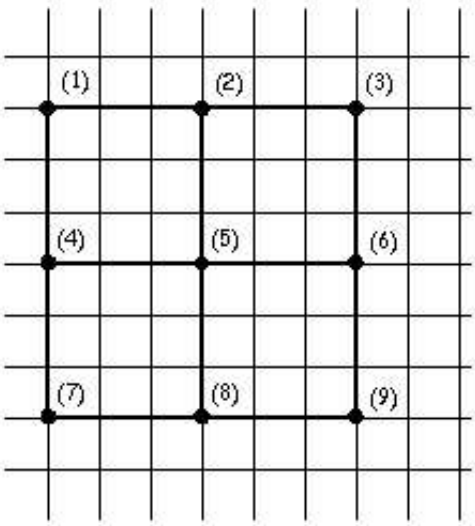

Figura 3.2: Búsqueda en tres pasos rápida

el próximo paso. Cuando la posición de menor distorsión es el centro, se reduce la cantidad de píxeles d a la mitad, y la figura donde se realiza la nueva búsqueda es el cuadrado conformado por los puntos medios de las aristas del cuadrado anterior. El algoritmo termina cuando, siendo $d = 1$, se elige el centro del cuadrado.

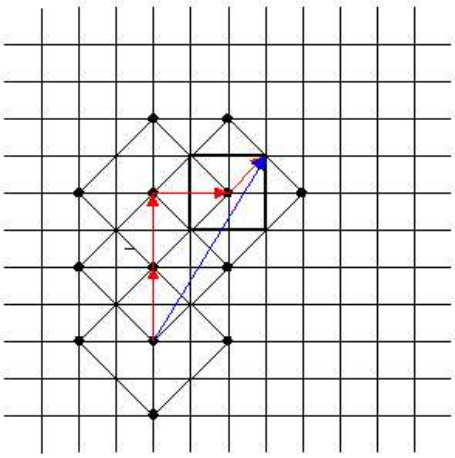

Figura 3.3: Búsqueda logarítmica

Búsqueda de dirección conjugada

En la búsqueda de la dirección conjugada[2][6] se realiza una búsqueda de la posición de menor distorsión en la dirección horizontal para cada mac-

robloque de la imagen de referencia, luego se hace lo mismo en la dirección vertical y con estas dos posiciones se encuentra el vector de movimiento buscado como se muestra en la Figura 3.4.

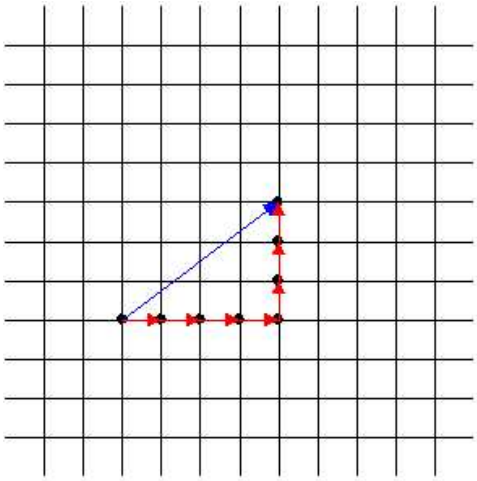

Figura 3.4: Búsqueda conjugada

Búsqueda en capas

Esta es una búsqueda con soporte variable. Se divide el área de búsqueda en capas concéntricas (en forma de diamante) y se chequean todas las posiciones de una capa a la vez, de adentro hacia afuera (ver Figura 3.5). Se ha observado que generalmente hay más movimiento en la dirección horizontal que en la vertical, por lo que podría resultar más conveniente utilizar áreas de búsqueda asimétricas que fueran más grandes en esta dirección. Sin embargo los resultados experimentales consultados en la bibliografía muestran que las mejoras son generalmente insignificantes por lo que se decidió utilizar áreas de búsquedas simétricas en la dirección vertical y horizontal. Si la mínima distorsión obtenida en la capa l es mayor que la obtenida en la capa $l-1$ se suspende la búsqueda pues se considera poco probable que se encuentre un mejor vector de movimiento por más que se continue la búsqueda hacia afuera. También debe ponerse una restricción al número de capas que pueden recorrerse para evitar que el algoritmo se estanque en uno de los bloques. Algunas implementaciones utilizan criterios de parada más estrictos, lo que lleva a un mejor resultado en cuanto a calidad pero insume mayor cantidad de cuentas. Esta es otra búsqueda basada en UESA.

Búsqueda por el camino de la menor distorsión

Esta búsqueda[8] se parece a la anterior en que divide su región de búsqueda en capas. La diferencia es que en esta búsqueda todas las capas tienen el

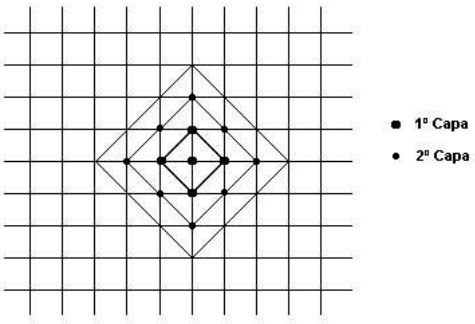

Figura 3.5: Búsqueda en capas

mismo tamaño y la posición de una capa depende de la posición del mínimo en la capa anterior. Es una búsqueda altamente localizada por lo que es conveniente combinarla con alguna técnica de predicción que de buenos resultados.

El algoritmo consiste en la búsqueda secuencial a través de capas en forma de diamante de igual tamaño (ver Figura 3.6). La regla básica es que la capa $l+1$ está centrada en el mínimo de la capa l, (observar que cada capa contiene una o dos posiciones que ya fueron chequeadas).

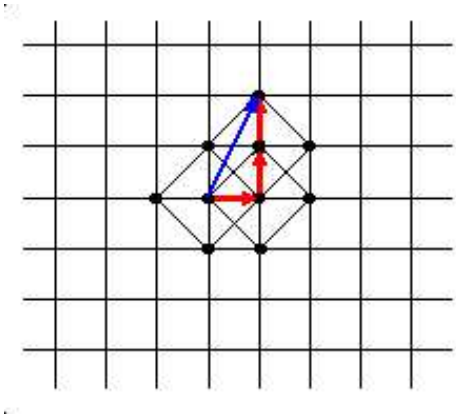

Figura 3.6: Búsqueda por el camino de menor distorsión

Este algoritmo incorpora también un nuevo elemento: tiene en cuenta el número de operaciones que se realizan en el criterio de parada. La búsqueda continua hasta que:

$$J_l - J_{l+1} \leq \tau, \quad J_l = D_l^* + \lambda C_l$$

Se elige el vector correspondiente a la menor SAD calculada hasta el momento. En las ecuaciones anteriores:

- J_l es la función de costo.
- D_l^* es la mínima distorsión de la capa l.
- C_l es el número de operaciones que se realizan en la capa l[1].
- el parámetro λ controla la relación entre la performance en cuanto a distorsión y la carga computacional.
- el umbral τ es un parámetro de relajación que permite pequeños incrementos en el costo entre dos capas sucesivas. Responde al hecho que en el camino al vector de movimiento *óptimo* pueden existir ligeros aumentos en la función de costo.

3.3 Estimación de movimiento con bloques de tamaño variable

En un principio cuando las técnicas de compresión no eran tan buenas, los vectores de movimiento ocupaban un lugar despreciable frente a la cantidad de información a transmitir; hoy que se han logrado niveles de compresión mayores se observa que los vectores de movimiento empiezan a ocupar un ancho de banda comparable al resto de la información a mandar. Trabajar con bloques de tamaño mayor, evidentemente disminuye la cantidad de vectores a enviar, pero va en contra de la calidad de la compensación. Inspirada en estas ideas surge la técnica de estimación de movimiento con bloques de tamaño variable que mejora la compresión por la vía de disminuir la cantidad de vectores de movimiento que debe ser transmitida sin perjudicar la calidad de la imagen recuperada.

Este algoritmo empieza por estimar el movimiento de bloques del máximo tamaño permitido; si la mejor distorsión que puede lograr para un bloque en particular es mayor que cierto umbral; se parte este bloque en cuatro subbloques y se repite el procedimiento por separado para cada uno de ellos. Si un subbloque es del menor tamaño permitido y la distorsión óptima que se consigue es mayor que cierto umbral, el subbloque es codificado como *intra*.

Esto permite trabajar con bloques de tamaño grande cuando la distorsión se mantiene acotada (esto implica pocos vectores de movimiento), y hacer una buena estimación de movimiento cuando el mismo es irregular por la vía de achicar el tamaño de los bloques. En este caso en particular

[1] una operación consiste de una sustracción, una operación de tomar valor absoluto y una adición de valor absoluto

se trabaja con dos tamaños de bloques (16×16 y 8×8). Dado que en un siguiente paso se aplicará la transformada DCT a bloques de 8×8, es conveniente que el menor tamaño de bloque permitido sea precisamente 8×8.

El algoritmo sería entonces el siguiente:

Codificación con tamaño de bloque variable

1. **Paso 1:** Parto el bloque en cuatro subbloques. Para $i = 1, \ldots, 4$ codifico B_i. Si la distorsión es menor que 16 $\rightarrow$ **Fin**
2. **Paso 2:** Parto el subbloque B_i en cuatro subbloques. Para $j = 1, \ldots, 4$ codifico B_{ij}. Si la distorsión es mayor que 4 marco B_{ij} como *intra*.

Este algoritmo es de por sí, más lento que el de tamaño de bloque fijo, pues en el peor de los casos pasa dos veces por cada pixel de la imagen actual.

Esto no lo hace apropiado para aplicaciones de tiempo real, sólo se aplicará en la compresión de video para almacenamiento (tipo *web*). Su gran ventaja es que al necesitar, en el caso general, menor cantidad de vectores de movimiento alcanza mejores niveles de compresión. Este algoritmo es una solución intermedia entre trabajar con bloques de tamaño fijo 16×16 y trabajar con bloques de tamaño fijo 8×8.

3.4 Estimación de movimiento independiente de la media

Los cambios de iluminación pueden hacer fallar al mejor algoritmo de estimación de movimiento si no se han tomado las precauciones necesarias para tenerlos en cuenta.

Para evitar este problema lo que se hace es calcular la media de cada bloquecito que se considera y a continuación se resta este valor de cada pixel, de manera de obtener un bloquecito con media cero que mantiene la misma forma. Las medias deben transmitirse al decodificador por separado.

3.5 Optimizaciones

3.5.1 Eliminación por distorsión parcial (half-stop)

Esta técnica consiste en detener el cálculo de la distorsión entre el bloque a codificar y un determinado candidato cuando ésta supera la mínima distorsión alcanzada hasta el momento. Es evidente que no hace falta terminar de calcular esta distorsión pues el candidato será descartado de todos modos.

La forma en que se implementa es efectuando la comparación entre el resultado parcial de la distorsión que se está calculando y la distorsión alcanzada con el mejor vector de movimiento que se ha encontrado hasta ese punto. Si el recorrido dentro del bloque para calcular la distorsión se realiza de arriba a abajo y de izquierda a derecha, la comparación mencionada se realiza al terminar de sumar cada fila.
Es importante destacar que esta optimización, a diferencia de todas las otras técnicas utilizadas para acelerar el encuentro del mejor vector de movimiento, no repercute sobre la calidad de la imagen recuperada.

3.5.2 Recorrido adaptativo según la norma del gradiente

Este algoritmo[9] también cae dentro del grupo de técnicas que mejoran la velocidad sin afectar la calidad de la imagen recuperada. Se basa en la técnica de eliminación por distorsión parcial, la mejora depende de que tan rápido se abandone el cálculo del error en un bloque que será desechado.

Se demuestra, por medio del desarrollo de Taylor, que el error de matcheo entre el bloque de referencia y el bloque candidato es aproximadamente proporcional a la norma del gradiente del bloque de referencia. Entonces para desechar más rápidamente un bloque (lo que introduciría una mejora), no hay más que empezar a calcular el error donde el gradiente es mayor.

Se divide el bloque en cuatro subbloques y se calcula la norma del gradiente de cada uno de ellos. Se ve qué mitad del cuadrado tiene suma de gradientes mayores (la mitad superior, inferior, izquierda o derecha) y se empieza a recorrer el macrobloque por ésta.

(i)	(ii)
(iii)	(iv)

Recorrido según la norma del gradiente

1. **Si (i)+(ii) es máximo:** Recorrido de arriba a abajo y de izquierda a derecha.
2. **Si (iii)+(iv) es máximo:** Recorrido de abajo a arriba y de izquierda a derecha.
3. **Si (i)+(iii) es máximo:** Recorrido de izquierda a derecha y de arriba a abajo.
4. **Si (ii)+(iv) es máximo:** Recorrido de derecha a izquierda y de arriba a abajo.

Las comparaciones entre todas estas técnicas y estrategias se hacen en el capítulo 6

Capítulo 4

Métodos para simplificar la señal a comprimir: transformada discreta del coseno

Una imagen natural tiene, generalmente, una variación complicada de su amplitud (o color) vista como función de la posición dentro de la imagen. A pesar de eso, es posible expresar esta función como una suma de cosenos de la frecuencia y amplitud correspondientes. De eso se trata la transformada discreta del coseno. Cuando trabajamos en dimensión 2 la transformada toma series de 8×8 datos (bloques de la imagen de 8×8 píxeles) y los transforma en un bloque de 8×8 coeficientes en el espacio de las frecuencias espaciales. La manera más práctica de pensar en ella es tal como si fuera una transformada de Fourier.

La aplicación de esta transformada es completamente reversible, lo único que hace es pasar del dominio del espacio al dominio de la frecuencia sin pérdida alguna de información, salvo la pérdida de información debida a la representación de la máquina).

Se puede ver que los coeficientes de la DCT están mucho menos correlacionados que los coeficientes de una imagen natural (las imágenes naturales presentan una gran redundancia espacial). Esto hace que sea más eficiente la compresión de los coeficientes de la transformada que la compresión de los coeficientes de la imagen original.

4.1 Procedimiento de cálculo

La 2D-DCT se define como [2]:

$$F(\mu,\nu) = \frac{C(\mu)}{2}\frac{C(\nu)}{2}\sum_{y=0}^{7}\sum_{x=0}^{7} f(x,y)\cos\left(\frac{(2x+1)\mu\pi}{16}\right)\cos\left(\frac{(2y+1)\nu\pi}{16}\right)$$

donde μ y ν son los índices de la frecuencia horizontal y vertical respectivamente y las constantes $C(\mu)$ y $C(\nu)$ vienen dadas por:

$$\begin{cases} C(\kappa) = 1/\sqrt{(}2) & \text{para } \kappa = 0 \\ C(\kappa) = 1 & \text{para } \kappa > 0 \end{cases} \tag{4.1}$$

Puede verse fácilmente que la 2D-DCT no es otra cosa que la aplicación sucesiva de la 1D-DCT en una y otra dirección espacial.

Lo mismo puede decirse para la transformada inversa del coseno (IDCT) que se expresa como:

$$f(x,y) = \sum_{\mu=0}^{7}\sum_{\nu=0}^{7} \frac{C(\mu)}{2}\frac{C(\nu)}{2}F(\mu,\nu)\cos\left(\frac{(2x+1)\mu\pi}{16}\right)\cos\left(\frac{(2y+1)\nu\pi}{16}\right)$$

Esto simplifica en gran medida la implementación de estos cálculos pues no hay más que aplicar sucesivamente las versiones 1D de la DCT y de la IDCT que, por cierto, son bastante sencillas de implementar.

Se quiere resaltar que el cálculo de la DCT es costoso, por ejemplo, trabajando con bloques de 8×8 y una imagen de 352×288, como en este caso, la DCT debe ejecutarse un total de 1584 veces. Debido a esto, desde el primer momento se buscó que los algoritmos implementados fuesen algoritmos rápidos, tanto para la DCT como para la IDCT.

Otra observación importante es que, si bien, la DCT recibe una matriz de enteros y devuelve otra matriz de enteros en sus cálculos intermedios utiliza coeficientes entre 0 y 1 que no pueden redondearse sin riesgo de perder completamente la información de la imagen.

Esto llevó a implementar dos tipos de algoritmos para calcular la DCT y su inversa, IDCT (todos ellos en versiones rápidas como ya se dijo).

Primer tipo Algoritmos que utilizan números flotantes para sus cálculos intermedios y que sólo realizan el *casting* a enteros después de dividir el resultado entre los coeficientes de la máscara de cuantización.

Segundo tipo Algoritmos que trabajan exclusivamente con enteros. Para salvar el problema de la pérdida de precisión, básicamente lo que se hace es multiplicar los coeficientes por un número k, grande (potencia de 2) y realizar todos los cálculos con los nuevos coeficientes. Como último paso se realiza un corrimiento hacia la derecha de n bits ($n = n(k)$) para recuperar el resultado correcto con la precisión suficiente.

Los resultados obtenidos fueron prácticamente idénticos en términos de relación señal a ruido (PSNR)[1]. El tiempo insumido, por el contrario, es sensiblemente menor en el primero de los algoritmos implementados, por lo que éste es el que se elige.

4.2 Cuantificación

Se mencionó en la introducción que la DCT permite discriminar la información en sus distintas componentes frecuenciales y, por lo tanto, representar cada frecuencia con diferente precisión en consonancia con la curva de respuesta del Sistema Visual Humano. Esto es una herramienta poderosa para lograr compresiones importantes sin deteriorar demasiado la imagen.

El proceso de *cuantificación* es lo que permite modificar con elegancia las precisiones utilizadas para cada frecuencia espacial. Este proceso lo que hace es dividir cada uno de los coeficientes de la DCT por un valor (llamado valor de cuantificación) redondeando luego el resultado obtenido para obtener un entero (la matriz de los valores de cuantificación se llama máscara de cuantificación).

Se aplica la DCT conjuntamente con la cuantificación tanto en los bloques intra como en los inter. Las máscaras que se utilizan en cada caso son netamente diferentes, dado que la información se concentra de diferente forma en cada caso. Como ejemplo, en la Figura 4.1 se presentan las máscaras sugeridas por el estándar MPEG que son las utilizadas en este proyecto.

4.3 Comentarios sobre la DCT aplicada a bloques 4×4

Durante las primeras pruebas se observó que la imagen recuperada quedaba afectada por un efecto de bloque importante. Esto llevó a pensar que trabajar con bloques de tamaño 4×4 (que seguramente solucionaría este problema) podría ser una buena opción.

[1] Peak Signal to Noise Relation $= 10 \log_{10} \frac{255^2}{MSE}$

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8
8	8	8	8	8	8	8	8

Figura 4.1: **Izquierda:** Máscara para bloques intra **Derecha:** Máscara para bloques inter

Se implementaron algoritmos rápidos para la DCT y la IDCT (del primer tipo) para bloques de 4×4 cuyas fórmulas son:

$$F(\mu,\nu) = \frac{C(\mu)}{2}\frac{C(\nu)}{2}\sum_{y=0}^{3}\sum_{x=0}^{3} f(x,y)\cos\left(\frac{(2x+1)\mu\pi}{8}\right)\cos\left(\frac{(2y+1)\nu\pi}{8}\right)$$

$$f(x,y) = \sum_{\mu=0}^{3}\sum_{\nu=0}^{3}\frac{C(\mu)}{2}\frac{C(\nu)}{2}F(\mu,\nu)\cos\left(\frac{(2x+1)\mu\pi}{8}\right)\cos\left(\frac{(2y+1)\nu\pi}{8}\right)$$

donde $C(\mu)$ y $C(\nu)$ cumplen con la ecuación 4.1.

Los resultados obtenidos fueron muy buenos en cuanto a la calidad de la imagen recuperada, pero la compresión que se podía alcanzar y el tiempo insumido eran mucho menos satisfactorios. Esto llevó al abandono definitivo de esta línea de trabajo.

En cuanto al efecto de bloque, se puede implementar algún *filtro cosmético*, que mejore la calidad de la imagen que recupera el decodificador, dado que, como es sabido, éste tiene una carga significativamente menor que el codificador.

Parte III

Codificador-Decodificador del presente proyecto

Capítulo 5

Estructura del Codificador-Decodificador

5.1 Codificador de Video

En la figura 5.1 se ve el diagrama de bloques del Codificador de Video que se implementó. Este codificador hereda muchas características del estándar MPEG siendo su principal innovación la introducción de la cuantificación vectorial. Son los resultados logrados por el estándar MPEG los que se tomarán como referencia para evaluar la performance del codificador/decodificador.

Cada uno de los bloques se estudiará en detalle en el presente trabajo pero a continuación se presenta, como forma de introducción al tema, una breve descripción del trabajo de los bloques y de la interacción entre ellos.

Estimador/Compensador de Movimiento Este es el bloque que intenta explotar la redundancia temporal presente en toda secuencia de video. La mayor parte de las imágenes de una secuencia son similares a las imágenes anteriores. El bloque de estimación/compensación de movimiento, estima el movimiento que se ha producido entre 2 cuadros consecutivos, luego con la misma información de movimiento que se transmite al decodificador, reconstruye la imagen a partir de la imagen anterior. Esta operación de reconstrucción es la misma que realiza el decodificador pues éste tiene toda la información para hacerla: conoce la imagen anterior y se le ha transmitido la información de movimiento. Por lo tanto, sólo será necesario transmitir la imagen diferencia entre la imagen original y la reconstruida o compensada. Es de esperar que esta imagen diferencia contenga mucha menos información que la imagen original, lo que justifica el tiempo que se emplea en realizar estas operaciones y el envio de la información de movimiento. Es importante destacar que el tamaño de la imagen diferencia es exactamente el mismo

que el de la imagen original, y que en principio estas operaciones no están ahorrando ancho de banda, son las operaciones posteriores las que se verán beneficiadas por esta transformación de la información a transmitir y que permiten la compresión.

Transformada Discreta del Coseno La transformada Discreta del Coseno discrimina la información según sus componentes de frecuencia espacial. Las frecuencias espaciales son muy representativas para el sistema visual humano, pues es sabido que no se perciben con igual intensidad las altas frecuencias que las bajas frecuencias, que las líneas horizontales o verticales son más importantes que las líneas diagonales. La aplicación de esta transformada, lleva bloques de 8×8 de niveles de gris en bloques de 8×8 de coeficientes de frecuencia y es completamente reversible. Es decir, nuevamente se tiene un bloque que no reduce la información a transmitir sino que la expresa de diferente manera para que la reducción de la información se realice sin afectar demasiado la calidad de la imagen.

Codificación de la Continua El primer coeficientes de la DCT es la información de continua. Para el sistema visual humano ésta información es mucho más importante que la información aportada por los demás coeficientes de la DCT. Por eso es que se utiliza una codificación sin pérdidas para este coeficiente. Para el mismo se implementó un codificador escalar que codifica las diferencias entre la continua de un macrobloque y la continua del macrobloque anterior, pues es de esperar que éstas estén altamente correlacionadas a menos que haya un cambio brusco de iluminación.

Cuantificación Vectorial Este bloque toma algunos de los coeficiente de la DCT y los ordena en diferentes vectores. La cuantificación se realiza buscando el vector más parecido en un codebook o diccionario y transmitiendo el índice del representante elegido. Se utiliza un codebook diferente para cada vector de coeficientes, que obviamente es conocido por el decodificador. La forma en que se ordenan estos coeficientes en los distintos vectores puede influir en el resultado de la cuantificación, lo que se pretende es agrupar los coeficientes que están más correlacionados entre sí. Algunos coeficientes no se codifican pues se supone a priori que representan poca información por lo que se sustituyen directamente por cero en el decodificador. Este es uno de los dos bloques en el que se tiene **compresión de la información**, y es el único punto de todo el proceso donde se tiene pérdida de información. Dentro de este bloque tenemos la importante función de **Adaptación de Codebooks** que se encarga de actualizar los codebooks que se utilizan tanto en el codificador como en el decodificador. Es necesario para la codificación de video pues la estadística de la señal puede variar en el tiempo haciendo que los codebooks que son óptimos al principio de la secuencia

dejen de serlo algunos cuadros más adelante o incluso que se vuelvan completamente inútiles estropeando la codificación. Esta función, se encarga de incluir nuevos vectores en el codebook cuando aparecen vectores de la imagen que no pueden codificarse satisfactoriamente y de eliminar los vectores del codebook que han dejado de utilizarse. Esto aumenta el overhead de la transmisión pero apunta a mejorar la calidad de la imagen recuperada en el decodificador.

Codificación de Entropía En nuestro codificador se implementó el algoritmo de Huffman, se trata de una codificación sin pérdidas de largo de código variable. Básicamente lo que hace es asignar palabras de código más cortas a los símbolos más frecuentes, apuntando a disminuir el largo promedio de los símbolos transmitidos y con ellos el ancho de banda necesario para la transmisión. También aquí tenemos **compresión de la información** a transmitir, al igual que en el bloque de la cuantificación vectorial, con la diferencia importante de que en este bloque no tenemos pérdida de información.

Decodificación de Entropía El decodificador recibe información de frecuencia de los símbolos con la que puede recuperar los símbolos que se transmiten de la cadena de bits que recibe del codificador.

Decodificación Vectorial La decodificación es sencilla y rápida. Se limita a una búsqueda en tabla, dado que el decodificador también conoce los *codebooks* con los que el codificador realiza la cuantificación de los vectores.

Decodificación de Continua La decodificación simplemente recupera los coeficientes de continua.

Transformada Discreta del Coseno Inversa Después de decodificar cada uno de los bloques de la DCT, se realiza la DCT inversa, donde se hacen cero todos los coeficientes de la DCT que no fueron cuantificados. Luego de realizar la DCT inversa, nuevamente se obtiene una matriz donde cada uno de los elementos corresponde al nivel de gris del pixel respectivo.

5.2 Codificador de Imágenes fijas

En el principio de este proyecto se trabajó con imágenes fijas y no con secuencias de video. Esto se hizo pues la interacción entre los distintos bloques es muy alta y fue necesario simplificar el trabajo. Esta fue una etapa muy importante para entender las posibilidades de la cuantificación vectorial, así como sus limitaciones. En la Figura 5.2 vemos un diagrama de bloques del codificador que se utilizó para estudiar las imágenes fijas. Como

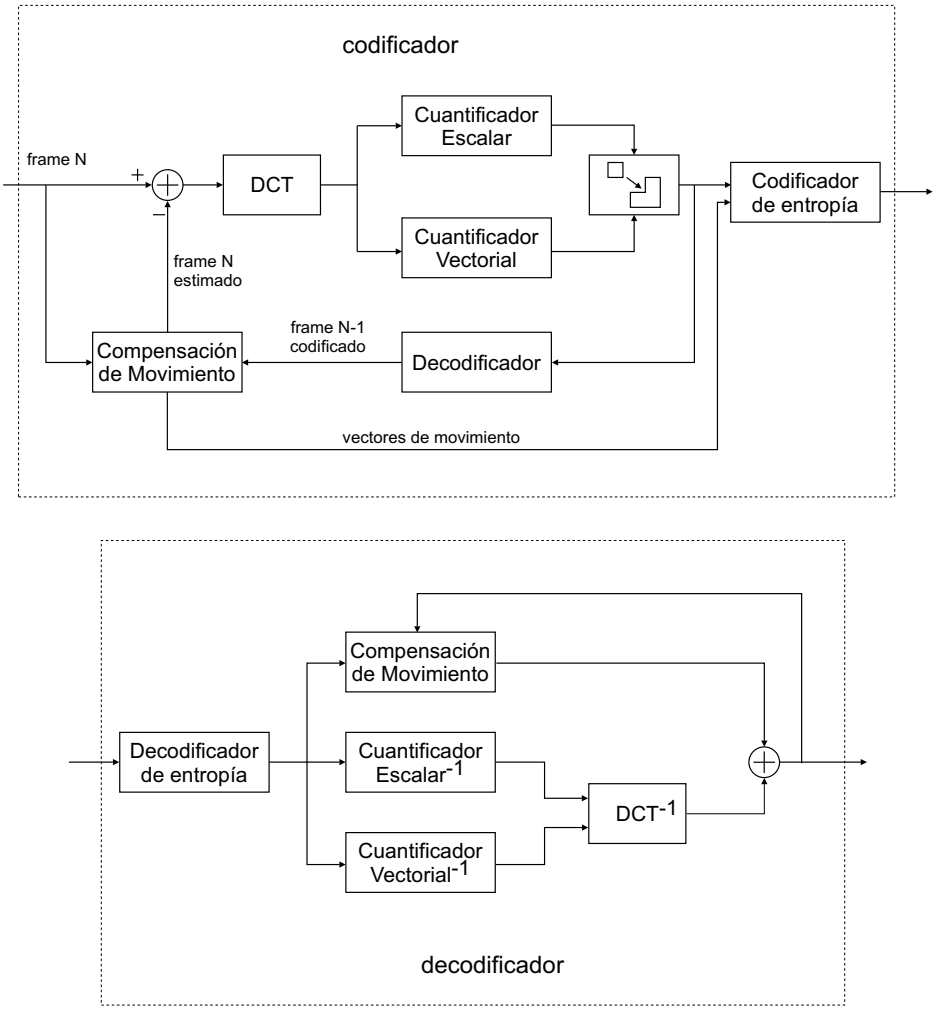

Figura 5.1: Diagrama del codificador/decodificador.

se ve, este codificador está compuesto por algunos de los bloques presentes en el codificador de video que ya se describió, siendo por cierto, mucho más sencillo.

El proceso de codificación de una imagen (considerándola como una matriz donde cada elemento es el nivel de gris de cada pixel) tiene tres etapas claramente separables: **DCT**, **Cuantificación Escalar** del coeficiente de continua de la DCT y **Cuantificación Vectorial** de los coeficientes restantes.

Primero se calcula la transformada discreta del coseno de la imagen obteniendo una matriz de igual tamaño que la imagen, pero con una distribución de la información totalmente distinta. Luego se realizan la cuantificación escalar de los coeficientes de continua y la cuantificación vectorial del resto de

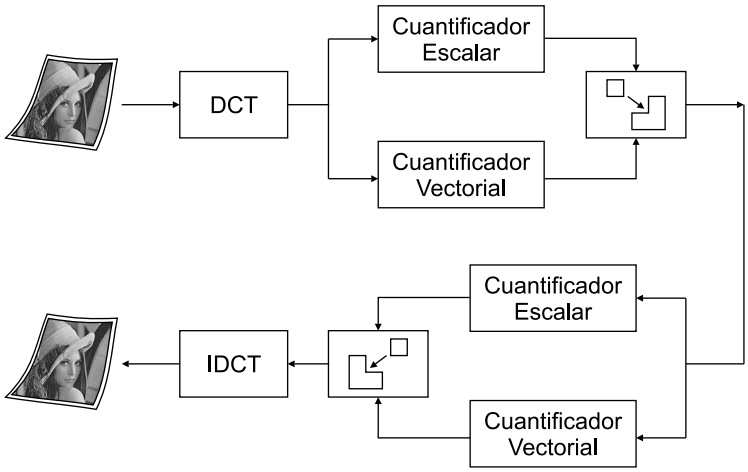

Figura 5.2: Diagrama del codificador/decodificador.

los coeficientes; estos procesos son independientes uno del otro y los datos que se extraen de cada uno de ellos forman los datos de la imagen codificada.

Capítulo 6

Estimación de movimiento

En este capítulo se presentarán los resultados obtenidos con las técnicas implementadas (que ya fueron descritas en el capítulo 3) y la justificación técnica de las elecciones que se realizaron.

A los efectos de comparar las distintas búsquedas y estrategias implementadas no se realizó la clasificación de los bloques como *intra* en las pruebas que se presentan en esta sección, para que éstos no afectaran el valor del PSNR obtenido. En todos los casos se utiliza el criterio de parada por distorsión parcial.

6.1 Comparación de las búsquedas rápidas

Se utilizaron cuatro parejas de imágenes para realizar estas comparaciones correspondientes a las secuencias estándares `foreman.cif` y `trainera.cif`. De cada una de estas secuencias se eligieron dos parejas de imágenes, una pareja de imágenes consecutivas y una pareja de imágenes ligeramente alejadas en la secuencia (concretamente separadas 12 cuadros lo que representa medio segundo).

Se eligieron estas dos secuencias a los efectos de tener un ejemplo de una secuencia con movimiento de cámara y con un rostro en primer plano (esto último es típico de una transmisión de video telefonía) como es la secuencia Foreman y otro ejemplo de una compensación de movimiento difícil de lograr como es la secuencia Trainera pues las grandes extensiones de mar, hacen difícil la compensación exitosa. Las búsquedas se hacen en un área de ± 7 pixeles.

Como ya se mencionó la mejor calidad es de esperarse cuando se utiliza la búsqueda exhaustiva. Por lo tanto las evaluaciones de las búsquedas rápidas se realizarán comparándolas con ésta.

Los tres parámetros que nos interesarán son:

- Calidad de imagen alcanzada, medida según el PSNR en dB. El PSNR se presenta en su valor absoluto y como un porcentaje del PSNR logrado con la búsqueda exhaustiva para esa misma pareja de imágenes.

- Tiempo de búsqueda. Este parámetro se presenta sólo como un porcentaje del tiempo empleado en la búsqueda exhaustiva para el mismo par de imágenes. No se toman en cuenta los valores absolutos pues éstos dependen fuertemente del equipo utilizado para realizar la compensación de movimiento.

- Bits necesarios para la transmisión de los vectores de movimiento después de aplicada una compresión de entropía.

Búsqueda en tres pasos

Imagen de referencia	Imagen a reconstruir	Tamaño de Macrobloque	PSNR (en dB)	PSNR (en %)	Bpps	Tiempo en porcentaje
Foreman000	Foreman001	8×8	32.0	94 %	0.110	19.7 %
Foreman010	Foreman022	8×8	24.6	94 %	0.123	20.2 %
Trainera000	Trainera001	8×8	26.7	98 %	0.114	18.2 %
Trainera010	Trainera022	8×8	24.2	97 %	0.125	16.4 %
Foreman000	Foreman001	16×16	29.8	98 %	0.027	23.5 %
Foreman010	Foreman022	16×16	23.6	97 %	0.030	24.3 %
Trainera000	Trainera001	16×16	25.2	98 %	0.024	16.0 %
Trainera010	Trainera022	16×16	22.4	99 %	0.031	17.5 %

Búsqueda en tres pasos rápida

Imagen de referencia	Imagen a reconstruir	Tamaño de Macrobloque	PSNR (en dB)	PSNR (en %)	Bpps	Tiempo en porcentaje
Foreman000	Foreman001	8×8	31.4	92 %	0.103	12.4 %
Foreman010	Foreman022	8×8	21.5	82 %	0.120	12.3 %
Trainera000	Trainera001	8×8	26.0	95 %	0.111	14.8 %
Trainera010	Trainera022	8×8	23.4	94 %	0.118	13.3 %
Foreman000	Foreman001	16×16	29.5	97 %	0.0259	18.2 %
Foreman010	Foreman022	16×16	20.7	85 %	0.0246	19.1 %
Trainera000	Trainera001	16×16	24.8	97 %	0.0234	11.7 %
Trainera010	Trainera022	16×16	21.8	96 %	0.0300	18.2 %

Búsqueda logarítmica

Imagen de referencia	Imagen a reconstruir	Tamaño de Macrobloque	PSNR (en dB)	PSNR (en %)	Bpps	Tiempo en porcentaje
Foreman000	Foreman001	8×8	31.3	92 %	0.100	16.0 %
Foreman010	Foreman022	8×8	21.3	81 %	0.112	20.3 %
Trainera000	Trainera001	8×8	26.1	96 %	0.105	10.8 %
Trainera010	Trainera022	8×8	23.7	95 %	0.112	16.4 %
Foreman000	Foreman001	16×16	29.8	98 %	0.0240	13.9 %
Foreman010	Foreman022	16×16	20.4	84 %	0.0287	19.1 %
Trainera000	Trainera001	16×16	24.9	97 %	0.0215	11.7 %
Trainera010	Trainera022	16×16	22.1	98 %	0.0281	14.3 %

Valor de la distancia inicial = 4 pixeles

Búsqueda en la dirección conjugada

Imagen de referencia	Imagen a reconstruir	Tamaño de Macrobloque	PSNR (en dB)	PSNR (en %)	Bpps	Tiempo en porcentaje
Foreman000	Foreman001	8×8	23.2	68 %	0.112	11.7 %
Foreman010	Foreman022	8×8	19.4	74 %	0.114	11.6 %
Trainera000	Trainera001	8×8	25.0	92 %	0.105	11.5 %
Trainera010	Trainera022	8×8	22.3	89 %	0.110	9.70 %
Foreman000	Foreman001	16×16	23.3	76 %	0.0294	13.9 %
Foreman010	Foreman022	16×16	18.8	77 %	0.0287	13.9 %
Trainera000	Trainera001	16×16	24.2	95 %	0.0205	8.03 %
Trainera010	Trainera022	16×16	21.5	95 %	0.0274	11.0 %

Búsqueda en capas

Imagen de referencia	Imagen a reconstruir	Tamaño de Macrobloque	PSNR (en dB)	PSNR (en %)	Bpps	Tiempo en porcentaje
Foreman000	Foreman001	8×8	32.3	95 %	0.0950	55.5 %
Foreman010	Foreman022	8×8	21.6	82 %	0.114	84.1 %
Trainera000	Trainera001	8×8	26.5	97 %	0.105	37.2 %
Trainera010	Trainera022	8×8	23.8	95 %	0.111	46.7 %
Foreman000	Foreman001	16×16	30.0	98 %	0.0234	95.7 %
Foreman010	Foreman022	16×16	20.4	84 %	0.0281	100 %
Trainera000	Trainera001	16×16	25.0	98 %	0.0199	40.1 %
Trainera010	Trainera022	16×16	22.0	97 %	0.0278	50.0 %

Búsqueda por el camino de menor distorsión

Los valores utilizados para los parámetros son algunos de los sugeridos en la bibliografía. Se presentan sólo algunas de las pruebas realizadas, concretamente las que obtuvieron mejores resultados.

Imagen de referencia	Imagen a reconstruir	Tamaño de Macrobloque	PSNR (en dB)	PSNR (en %)	Bpps	Tiempo en porcentaje
Foreman000	Foreman001	8×8	30.7	90 %	0.101	24.1 %
Foreman010	Foreman022	8×8	21.0	80 %	0.119	23.9 %
Trainera000	Trainera001	8×8	25.8	95 %	0.105	18.2 %
Trainera010	Trainera022	8×8	23.4	94 %	0.112	50.3 %
Foreman000	Foreman001	16×16	29.7	98 %	0.0262	24.3 %
Foreman010	Foreman022	16×16	20.4	84 %	0.0300	24.3 %
Trainera000	Trainera001	16×16	24.4	96 %	0.0249	16.1 %
Trainera010	Trainera022	16×16	21.8	96 %	0.0290	17.5 %

Valores de los parámetros: lambda = 2, tolerancia = 1600.

Imagen de referencia	Imagen a reconstruir	Tamaño de Macrobloque	PSNR (en dB)	PSNR (en %)	Bpps	Tiempo en porcentaje
Foreman000	Foreman001	8×8	30.7	90 %	0.101	27.7 %
Foreman010	Foreman022	8×8	21.0	80 %	0.119	23.9 %
Trainera000	Trainera001	8×8	25.8	95 %	0.105	18.2 %
Trainera010	Trainera022	8×8	23.4	94 %	0.112	20.0 %
Foreman000	Foreman001	16×16	29.7	98 %	0.0262	28.7 %
Foreman010	Foreman022	16×16	20.4	84 %	0.0300	28.7 %
Trainera000	Trainera001	16×16	24.4	96 %	0.0249	16.1 %
Trainera010	Trainera022	16×16	21.8	96 %	0.0290	14.3 %

Valores de los parámetros: lambda = 0, tolerancia = 1600.

Se puede ver que los resultados conseguidos con los dos juegos de parámetros son idénticos en lo que tiene que ver con el PSNR obtenido, no así con el tiempo de estimación de movimiento.

En la figura 6.1 vemos una comparación del PSNR promedio obtenido con los distintos algoritmos implementados contra el tiempo promedio invertido en realizar la estimación. Ambos valores se calculan como fracción de la búsqueda exhaustiva.

De la comparación entre todas las búsquedas son varias las conclusiones que se pueden extractar. Lo primero que se destaca es que todas las búsquedas logran valores de PSNR superiores al 80 % del valor que se alcanzaría con la búsqueda exhaustiva. La búsqueda que obtiene la mejor performance en este sentido es sin duda alguna la búsqueda en tres pasos, por un margen por demás amplio en el valor de PSNR. Esta búsqueda aumenta la velocidad de la estimación en aproximadamente cinco veces. Las búsquedas en Tres Pasos Rápida y la Logarítmica también logran buenos resultados de PSNR disminuyendo aun más el tiempo necesario para la estimación.

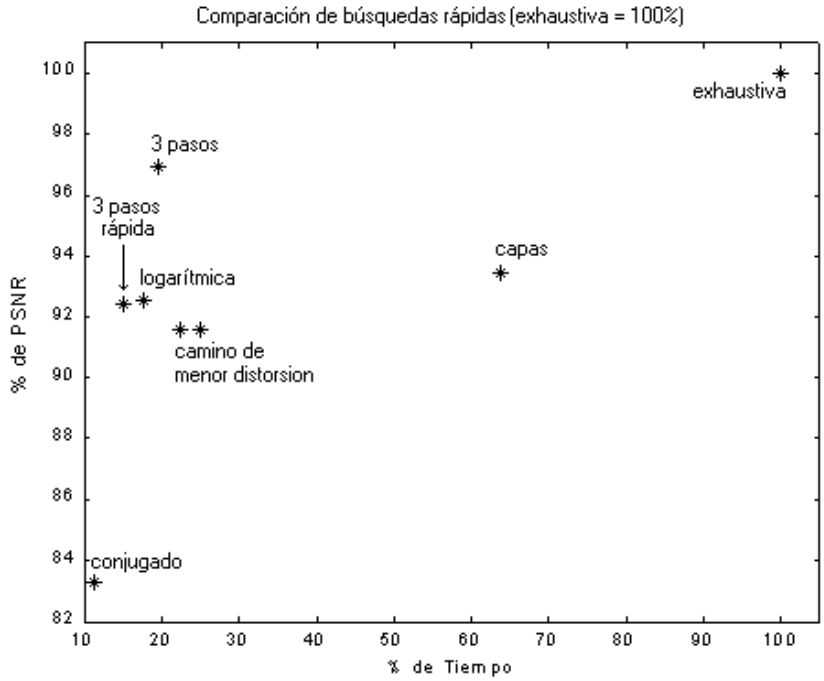

Figura 6.1: Comparación entre las distintas búsquedas rápidas

La búsqueda por el camino de menor distorsión no sería recomendable pues no supera a ninguna de las tres búsquedas ya mencionadas ni en calidad ni en velocidad, si bien sus resultados son aceptables. Esto último, no puede decirse de la búsqueda en capas pues esta es claramente inferior en cuanto al tiempo de ejecución. Por último, la búsqueda en la dirección conjugada sería apropiada para una aplicación en la cual el tiempo fuera la limitante principal, pues si bien sus resultados de calidad son los peores de todos los vistos aquí, es la búsqueda más rápida llegando casi a la décima parte del tiempo necesario para la búsqueda exhaustiva.

Por otra parte e independientemente de la búsqueda utilizada se puede ver que los resultados obtenidos con `tamaño de macrobloque = 16` son muy parecidos en calidad a los obtenidos con `tamaño de macrobloque = 8` y del orden del 20% en los bits necesarios para codificar los vectores de movimiento. Esto es totalmente razonable pues los vectores de movimiento necesarios son cuatro veces menos. Se puede ver en la Figura 6.2 los resultados obtenidos con todas las búsquedas para los distintos tamaños de Macrobloque.

Como se vió al principio de esta parte, no se intenta la codificación en tiempo real, por lo que *se elige la búsqueda exhaustiva para utilizar de aquí*

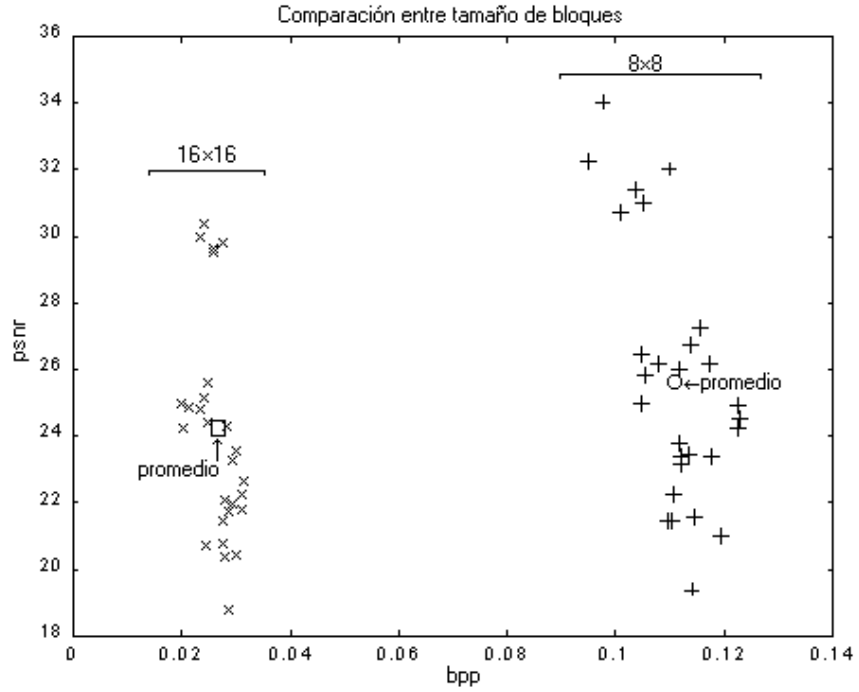

Figura 6.2: Comparación entre tamaños de Macrobloque

en más en el presente proyecto.

6.2 Estimación con tamaño de bloque variable

A continuación se presentan los resultados obtenidos con la Estimación De Tamaño De Bloque Variable. Dado que ésta es una técnica para aplicaciones fuera de línea, el tiempo insumido no es una restricción por lo que se la combinará con la búsqueda exhaustiva en una región de ± 7. La comparación se hará con la búsqueda exhaustiva para tamaño de bloque fijo (8×8 y 16×16). Los parámetros de interés son la calidad lograda y los bpp que ocupan los vectores de movimiento.

Comparación contra la estimación de movimiento con búsqueda exhaustiva con tamaño de bloque 8x8.

Imagen de referencia	Imagen a reconstruir	PSNR (en dB)	PSNR (en %)	Bpps	Bpps en porcentaje
Foreman000	Foreman001	31.6	93 %	0.0265	27 %
Foreman010	Foreman022	25.5	97 %	0.0391	33 %
Trainera000	Trainera001	26.6	98 %	0.0366	32 %
Trainera010	Trainera022	23.8	95 %	0.0467	38 %

Comparación contra la estimación de movimiento con búsqueda exhaustiva con tamaño de bloque 16x16.

Imagen de referencia	Imagen a reconstruir	PSNR (en dB)	PSNR (en %)	Bpps	Bpps en porcentaje
Foreman000	Foreman001	31.6	104%	0.0265	109 %
Foreman010	Foreman022	25.5	105%	0.0391	137 %
Trainera000	Trainera001	26.6	104%	0.0366	147 %
Trainera010	Trainera022	23.8	104%	0.0467	149 %

Los resultados obtenidos no son muy alentadores al mirar los números resultantes de la comparación de la búsqueda con bloques variables con la búsqueda exhaustiva con Tamaño de Bloque = 16. Se puede ver, que el PSNR no mejora más allá del 105% a pesar de que los bpp's se disparan más allá del 40 % más.

La mejora del PSNR podría modificarse si se cambia el criterio utilizado para cambiar el tamaño de bloque. El que se utilizó en éstas pruebas fue partir solamente aquellos bloques que hubieran sido codificados como intras en el caso de tamaño de bloque fijo.

Los bpp's obtenidos podrían mejorarse si se permiten bloques más grandes para empezar la estimación, por ejemplo 32×32. En este caso se tendrían tres tamaños posibles de bloques 32×32, 16×16 y 8×8.

Mejorar esta técnica por alguno de estos dos caminos hubiera sido posible pero aun así no hubiera resultado conveniente para el presente proyecto. La razón es muy sencilla: el tiempo necesario para la compensación de movimiento.

Como ya se dijo, uno de los objetivos es lograr la decodificación en tiempo real. La compensación de movimiento después de una estimación con tamaño de bloque variable es bastante más complicada que la compensación de una estimación tradicional y por lo tanto necesita más tiempo, tiempo que no pudo permitirse en el decodificador.

6.3 Estimación independiente de la media

Para estas pruebas cambiaremos las imágenes a utilizar. Se buscó una secuencia donde los cambios de iluminación fueran importantes (Ori) de manera de poder evaluar si esta técnica trae consigo algún beneficio. Se pudo observar que las imágenes obtenidas eran mejores con esta estimación que

con la búsqueda exhaustiva tradicional pero de todas formas la calidad de las mismas era bastante mala.

Tamaño de Bloque	Margen de búsqueda	PSNR de búsq. exhaustiva (bpp)	PSNR de estim. indep. de la Media (bpp)
8×8	7	12.1	16.2
8×8	15	12.3	16.5
16×16	7	12.1	17.0
16×16	15	12.2	15.7

6.4 Optimizaciones

Criterio de parada por distorsión parcial

Esta es una optimización aplicable a cualquier estrategia de estimación y a cualquier búsqueda utilizada. Las pruebas que se realizaron miden el tiempo que lleva la búsqueda exhaustiva con esta optimización como porcentaje del tiempo que lleva esta misma búsqueda sin aplicar este criterio.

Imagen de referencia	Imagen a reconstruir	Tamaño de Macrobloque	Tiempo en (en %)
Foreman000	Foreman001	8×8	58.8 %
Foreman010	Foreman022	8×8	57.8 %
Trainera000	Trainera001	8×8	64.0 %
Trainera010	Trainera022	8×8	70.2 %
Foreman000	Foreman001	16×16	46.9 %
Foreman010	Foreman022	16×16	53.5 %
Trainera000	Trainera001	16×16	63.6 %
Trainera010	Trainera022	16×16	83.9 %

Como se puede ver el tiempo empleado en la estimación de movimiento se reduce aproximadamente un 40 % en promedio, y como se vio en el capítulo 3 esta reducción en el tiempo de cálculo se logra sin afectar en absolutamente nada el PSNR alcanzado.

Recorrido adaptativo según la norma del gradiente

Los resultados no fueron buenos, comprobándose que algunas veces el tiempo ahorrado al desechar rápidamente un Macrobloque era inferior al invertido en calcular los gradientes de cada subbloque.

Capítulo 7

Transformada Discreta del Coseno

7.1 Elección de los coeficientes

Un aspecto importante de la codificación de los coeficientes de la DCT es como se eligen y ordenan los mismos para la cuantificación vectorial. Este es el momento en que se decide, principalmente, que coeficientes se codifican y cuales simplemente se sustituyen por cero en el decodificador. También se dispone, cual será el agrupamiento que se realizará con los coeficientes que se decide cuantificar.

En este punto de la decodificación es que se comienza a hablar de *bandas de frecuencia espacial*, concepto de fundamental importancia para este esquema de codificación de video. Los coeficientes que se han de codificar se dividen en varios vectores y la codificación de cada uno de estos vectores es completamente independiente. Cada vector tiene distinta dimensión y se codifica con un codebook específico.

Si cambiamos la composición de un vector cambiaremos su estadística y, por lo tanto, se necesitará diseñar un nuevo *codebook* óptimo y transmitirlo al decodificador junto con una indicación de qué coeficientes componen ahora el vector. Esta estrategia es completamente inapropiada por lo que se debe definir a priori que coeficientes se codificarán y como se ordenarán en los vectores.

Los posibles caminos a seguir en este punto son muchos. La primera decisión a tomar es cuales coeficientes deben cuantificarse, lo siguiente es cómo se agrupan en los distintos vectores. Relacionado con estas decisiones está el tema de qué tamaño tendrá cada codebook (y por lo tanto, cuantos bits se necesitarán para representar cada índice) y no es posible evaluar lo anterior

sin tener en cuenta que se tienen varios grados de libertad al determinar el tamaño de los codebooks que se le asignan a cada banda de frecuencia.

Sobre el primer punto, los estudios sobre el sistema visual humano indican que es fundamental dar prioridad a las bajas frecuencias frente a las altas frecuencias para obtener una calidad de imagen razonable frente a niveles de compresión importantes. Es decir, se elegirán principalmente los coeficientes de la esquina superior izquierda de la transformada de la DCT.

Sobre lo segundo, no se tiene una respuesta cerrada y definitiva. Es claro que para obtener los mejores resultados de la cuantificación vectorial en cuanto a compresión, lo mejor sería agrupar todos los coeficientes en un sólo vector. Esto llevaría a enormes retardos en la codificación y a una difícil construcción de un codebook óptimo como ya se vio en el capítulo 2.1.

Dado que esta solución se descarta se intenta una solución de compromiso: agrupar los coeficientes en más de un vector de forma que los coeficientes más correlacionados entre sí, formen parte del mismo vector.

7.2 Ordenamiento

La implementación que se realizó es completamente flexible en este aspecto, pues el camino a seguir para ordenar los coeficientes viene dado en una matriz que se pasa como parámetro a la función que calcula la DCT.

En el estándar MPEG los coeficientes del bloque son recorridos en forma de zig-zag, como se ve en la Figura 7.1.

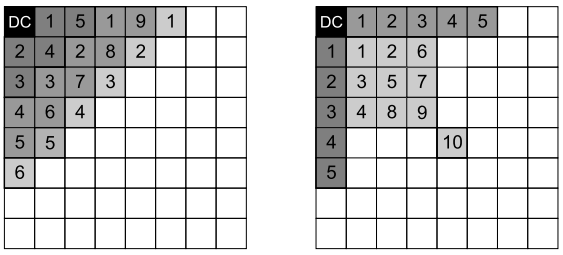

Figura 7.1: Recorridos de la DCT. **Izq.** Estándar MPEG. **Der.** Propuesto

Este ordenamiento hace más efectivo el proceso de compresión sin pérdidas pues así los coeficientes aparecen ordenados en forma creciente según la frecuencia, de esta forma los coeficientes de frecuencias altas (susceptibles de ser eliminadas en la cuantificación) se agrupan al final de la cadena.

Se probaron varios ordenamientos alternativos: el que dio mejores resultados fue el de agrupar las frecuencias horizontales en un vector, las frecuencias verticales en otro vector y las frecuencias diagonales en un tercero como se ve también en la Figura 7.1.

Los resultados obtenidos para las imágenes intra indican que el ordenamiento propuesto en este proyecto consiguió mejores resultados, si bien el PSNR alcanzado no fue muy diferente al alcanzado con el ordenamiento por defecto de MPEG, la evaluación subjetiva fue más favorable para las imágenes obtenidas con el primer ordenamiento que para las imágenes obtenidas con el segundo.

Para las imágenes diferencia la situación es diferente, dado que los coeficientes se encuentran bastante menos correlacionados. Se pudo comprobar que las bajas frecuencias seguían siendo fundamentales frente a las altas frecuencias y que la forma de agrupar los coeficientes no era de gran importancia. Por lo tanto se decide utilizar el ordenamiento por defecto de MPEG en este caso.

Capítulo 8

VQ sobre los coeficientes de la DCT

8.1 Cuantificación Vectorial

La cuantificación de los coeficientes en cada una de las bandas se realiza de forma muy sencilla, se toma cada uno de los vectores a cuantificar y se determina el vector del *codebook* que mejor lo representa, calculando la distancia del vector a cada uno de los elementos del *codebook* y eligiendo el que dista menos. El índice de este vector dentro del *codebook* es el índice que se le asocia. Este proceso se repite para cada uno de los vectores de los bloques de la DCT, con el *codebook* respectivo.

En caso de que no exista un *codebook* con el cual cuantificar las bandas, también se genera en este bloque. Se toma una imagen, se extrae cada una de las bandas y se utilizan como secuencia de entrenamiento para correr el *Algoritmo de Lloyd*, con algún *codebook* inicial. Para obtener un *codebook* inicial se ejecuta el *Algoritmo de Pruning* con cada una de las bandas.

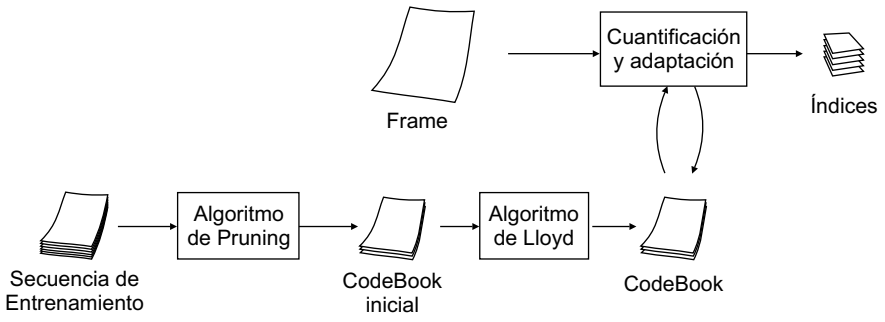

Figura 8.1: El proceso de cuantificación, con el cálculo de un *codebook* óptimo.

8.1.1 Algoritmo de Lloyd

Recordemos que este algoritmo construye un *codebook* óptimo para una secuencia de entrenamiento dada a partir de un *codebook* inicial cualquiera. La forma en que se obtiene este *codebook* inicial importa para la optimalidad del *codebook* final, porque el algoritmo podría converger hacia un mínimo local que no sea el mínimo global. También influye en la velocidad de convergencia.

El algoritmo se implementa como un algoritmo iterativo que como ya se vio en 2.8 reduce o mantiene constante la distorsión global con respecto a la secuencia de entrenamiento disponible. La condición de parada de esta iteración, entonces, puede ser la disminución relativa de esta distorsión.

En las iteraciones el procedimiento que se sigue es: calcular qué elementos de la secuencia de entrenamiento pertenecen a cada región, es decir, con cuál de los vectores del *codebook* se representa cada uno de los vectores de la secuencia de entrenamiento. Luego se recorre región por región calculando en cada una de ellas el centroide de los vectores que pertenecen a ella. Este centroide sustituye al vector del *codebook* que representa a esta región. Luego se calcula la distorsión con el *codebook* recién creado y en caso de que la disminución relativa sea menor que un cierto umbral, la iteración se da por finalizada.

Un problema grave que enfrenta el *Algoritmo de Lloyd*, es que en las sucesivas iteraciones, algunas regiones pueden quedar vacías, es decir que algunos de los vectores del *codebook* calculado no sean representantes de ningún elemento de la secuencia de entrenamiento.

En estos casos se debe realizar alguna estrategia para solucionar este problema. Esta implementación del algoritmo realiza el procedimiento que se describe a continuación para el caso que en la última iteración realizada haya quedado alguna región vacía.

8.1.2 Estrategia para las regiones vacías.

La estrategia que se implementó elimina cada uno de los vectores del *codebook* que genera una región vacía y aprovecha el lugar libre que queda en el *codebook* para dividir la región de mayor distorsión.

Este nuevo vector se obtiene al perturbar el vector asociado a la región de mayor distorsión en la última iteración. Para el cálculo de la perturbación se obtiene el vector propio asociado al mayor valor propio de la matriz de autocorrelación generada con los vectores de la secuencia de entrenamiento

que pertenecen a la región de mayor distorsión. Se perturba el vector asociado a la región de mayor distorsión en la dirección del vector propio recién calculado.

Se deben actualizar las distorsiones y se debe dividir nuevamente los vectores de la secuencia de entrenamiento en las nuevas regiones que se formaron. Luego se procede con la siguiente región vacía realizando el mismo proceso.

Una vez que se terminó con todas las regiones vacías, no se chequea la condición de salida y se fuerza a realizar una iteración más antes de chequear esta condición. Obviamente, en esta iteración se pueden generar nuevas regiones vacías.

Se realizaron pruebas para chequear su funcionamiento variando el número de elementos del *codebook* entre 64 y 256 elementos; tomando como secuencia de entrenamiento matrices de 1024 vectores de dimensiones entre 5 y 10 (bandas en imágenes de 256×256). De estas corridas se puede sacar una conclusión importante, el tiempo que toma la ejecución completa del *Algoritmo de Lloyd* es prohibitivo si se desea ejecutar este algoritmo en tiempo real, extendiéndose aún más con la búsqueda del *codebook* inicial (*Algoritmo de Pruning* o de *Clustering*).

8.1.3 Obtención del Codebook Inicial

Para realizar el cálculo de los *codebooks*, como primer paso se obtiene un *codebook* inicial y luego se le aplica el *Algoritmo de Lloyd*. El *codebook* inicial es imprescindible ya que es la entrada al *Algoritmo de Lloyd* y por lo general se obtiene a partir de un algoritmo sencillo. Las diferentes alternativas estudiadas para elegir el *codebook* inicial fueron las siguientes:

- *Codebook* con vectores en la diagonal
- Algoritmo de Pruning
- Algoritmo de Clustering

Estos mismos algoritmos fueron probados utilizando diferentes *codebooks* para diferentes zonas de frecuencia (la continua se codificó aparte). En este caso se eligieron 3 bandas de 5×1024 cada una, o sea, la secuencia de entrenamiento para cada una de la bandas fue de 1024 vectores de dimensión 5.

Codebook con vectores de la diagonal

En este caso se pudo notar que el *codebook* inicial es eficiente para imágenes naturales pero no para otras estadísticas como ser la transformada discreta del coseno de la imagen. En esta última clase de estadística, la imagen recuperada luego del proceso de cuantificación vectorial no fue satisfactoria. Como ventaja, este método no consume tiempo de codificación.

Algoritmo de Pruning

Este método es rápido y da buenos resultados para diferentes estadísticas, ya que el algoritmo intenta hallar un *codebook* adecuado a la estadística de entrada, a diferencia del algoritmo anterior que mantiene un *codebook* inicial constante, independientemente de la entrada. Como ya se vio, en este algoritmo se define un *umbral* que acota el tamaño mínimo de las celdas. Si este umbral es muy grande cubriremos el espacio de entrada con muy pocas celdas sin obtener un buen *codebook* inicial. Y viceversa, si el umbral es muy pequeño podría pasar que las celdas no cubriesen todo el espacio de entrada; en este caso tampoco se obtendría un buen *codebook* inicial. Vale la pena aclarar, que cuando se hace referencia al espacio de entrada sólo interesa la región del espacio de entrada donde está soportada la función de densidad de probabilidad de la señal de entrada. Este soporte es aproximado por medio del conocimiento de la secuencia de entrenamiento.

Se definen dos *coeficientes de actualización* que sirven para aumentar y/o achicar el tamaño mínimo de las celdas según sea necesario. Se define un umbral inicial (que solamente influirá en la velocidad de convergencia) y se corre una iteración del algoritmo. Si el umbral resulta grande (es decir, se puede cubrir toda la secuencia de entrenamiento con un codebook menor del que se está buscando) se multiplica por un coeficiente de actualización menor que uno que ha sido previamente definido. Si el umbral resulta chico (es decir, se utilizan todos los vectores del codebook sin que se pueda cubrir toda la secuencia de entrenamiento) se multiplica por un coeficiente de actualización mayor que la unidad y se itera nuevamente. Para evitar que el algoritmo entre en un ciclo infinito, se establece que el umbral no puede aumentarse después de haberse disminuido al menos una vez.

Obviamente la velocidad de convergencia estará dada por los coeficientes de actualización. Valores parecidos a la unidad aseguran una buena aproximación pero prometen una lenta convergencia.

Estas pruebas se realizaron utilizando el ordenamiento por defecto de MPEG (*zig-zag*) y se codificaron 15 coeficientes de la DCT divididos en tres vectores de 5 componentes cada uno. Se pudo observar la fuerte dependencia

Imagen	(a)	(b)	(c)	(d)
Umbral inicial	1500/1000/1000			
Tamaño Codebook	128/16/8	128/16/8	128/64/32	128/64/32
Coef. de Actualización	0.7/0.7/0.7	0.9/0.9/0.9	0.7/0.7/0.7	0.9/0.9/0.9
Umbral final	1050/490/490	1220/590/590	1050/490/170	1215/590/185
Tiempo de ejec. (en ref. a la prueba (a))	1	2.1	0.94	2.8

Tabla 8.1: Resultados para las pruebas del *Algoritmo de Pruning*.

del PSNR con respecto a las bajas frecuencias, lo que llevó a utilizar un *codebook* de mayor tamaño para la primera banda. En la determinación del umbral óptimo se observa claramente la dependencia de este con respecto al tamaño del *codebook*, a las diferentes estadísticas, etc. En la Tabla 8.1 se muestran algunos resultados obtenidos y los parámetros correspondientes.

Algoritmo de Clustering

El *Algoritmo de Clustering*, también conocido como el *Algoritmo de vecino más cercano dos a dos* es sin duda el algoritmo de mayor carga computacional de los presentados aquí. También es el que obtiene los mejores resultados, pero la mejora en la calidad de la imagen no justifica el enorme aumento en el tiempo que tarda este algoritmo en correr.

Dado que en este algoritmo se comienza con toda la secuencia de entrenamiento, el tiempo que demora es muy dependiente del largo de la misma. Para secuencias demasiado largas es inaplicable. También depende del tamaño del *codebook* que se pretende diseñar, para menores *codebooks* mayor será el tiempo que tarda el algoritmo en terminar. Por ejemplo, partiendo de una secuencia de entrenamiento de 1024 vectores para obtener un codebook de 256 vectores el tiempo insumido es aproximadamente 5 horas. Es más que evidente que estas demoras son prohibitivas para aplicaciones en tiempo real pero incluso pueden llegar a ser problemáticas para aplicaciones fuera de línea.

Las ventajas de este algoritmo son, además de su mejor rendimiento, que no dependen de parámetros fijados por el usuario sino que el *codebook* obtenido sólo depende de la secuencia de entrenamiento que se utilice.

8.2 Adaptación de codebooks

La estadística de la señal de video puede variar en el tiempo. Sería muy costoso recalcular y reenviar los codebooks al decodificador cada vez que el cambio de estadística sea tal que lo haga necesario. Pero, mantener el mismo *codebook* llevaría a la rápida degradación de la calidad de la imagen.

Es necesario entonces llegar a una solución de compromiso que impida la pérdida de calidad de imagen sin insumir gran cantidad de ancho de banda. La solución que se elige es adaptar cada uno de los codebooks paulatinamente a medida que la estadística de la señal de cada banda varía.

La técnica que se implementó es chequear continuamente la distancia entre cada vector que se codifica y su representante elegido en el codebook. Si esta distancia es mayor que cierto umbral (más adelante se explica como se fija éste), enviamos un símbolo no válido e incluimos el vector de la secuencia en el codebook. Para esto enviamos al decodificador un símbolo no válido e inmediatamente transmitimos el vector a agregar.

Para mantener constante el tamaño del codebook (y por lo tanto, los bpps) es necesario eliminar un vector del codebook por cada uno que se agregue. Elegimos para eliminar el vector del codebook que hace más tiempo que no se utiliza. Esta decisión, obliga a que la adaptación deba comenzar una vez que se ha avanzado en la secuencia lo suficiente como para poder tener estadística relativamente buena del uso de los vectores. El número de imágenes que se utilizan para hacer esta estadística es un parámetro configurable.

Una modificación que se utiliza para disminuir aun más el ancho de banda necesario es no transmitir el vector que se debe incluir, sino codificar este vector con un codebook bastante más grande que el que se utiliza para la codificación de la imagen y transmitir el índice del vector que mejor lo representa dentro de este codebook. Como es obvio, el codebook que se utiliza para actualizar debe ser calculado previamente y conocido por el decodificador. Este codebook se entrena con una colección amplia de imágenes naturales y es siempre el mismo, es decir, no será necesario transmitirlo al decodificador, o se transmite una sola vez fuera de línea. Es importante resaltar que la sustitución sólo se realiza si el vector del codebook grande aproxima mejor al vector de la imagen que con el codebook original; esto se hace para evitar algún caso especial en que no haya un buen matcheo en el codebook grande. En este caso, de realizarse la sustitución sólo se estaría empeorando la calidad de la imagen, malgastando ancho de banda.

Cabe indicar que después de la transmisión de una imagen intra los codebooks se calculan nuevamente y se reenvían al decodificador.

La determinación de los umbrales se hizo con el objetivo de no recargar la transmisión. Se chequea continuamente la cantidad de actualizaciones que se realizan por imagen pues éstas repercuten directamente en los bits transmitidos. Cuando las actualizaciones superan más del 20% de los índices que se transmiten, se considera que son demasiadas y se aumentan los umbrales de comparación de manera de disminuir las actualizaciones (se aumenta el tamaño máximo permitido a las celdas asociadas a cada vector del codebook). De igual forma, si las actualizaciones son menos que el 10% de los índices que se transmiten, entonces se disminuye el umbral de forma de mejorar la calidad de la imagen al disminuir el tamaño de las celdas asociadas a los vectores del codebook. Esto aumentará las cantidad de actualizaciones (y con ello los bits transmitidos) pero redundará en una mejora de la calidad de la imagen.

Capítulo 9

Codificación de continua

9.1 Cuantificador Escalar

Este cuantificador es el encargado de realizar la cuantificación de los coeficientes de *continua* (DC) de la DCT.

Como se comenta en la sección 4, estos coeficientes se tratan por separado debido a que corresponden al valor de luminancia o color de cada uno de los bloques de la DCT. Siendo portadores de una gran cantidad de información para el Sistema Visual Humano, se impone realizar una cuantificación lo más exacta posible, debido a que un error en la recuperación de estos coeficientes produce efectos que se aprecian claramente en la imagen recuperada.

Los problemas de la *continua* (DC)

En la mayoría de las imágenes, bloques vecinos en la DCT suelen tener niveles de iluminación o de color semejantes. De esta forma los coeficientes de *continua* de sus respectivas DCT tienen valores que difieren poco, o lo que es más interesante, la diferencia entre ellos es menor que el valor de los coeficientes. Para lograr una mayor compresión, entonces, se cuantificarán las diferencias entre un coeficiente y el coeficiente inmediatamente anterior. El primer coeficiente, no se cuantifica, y se transmite su valor sin ningún tipo de tratamiento.

Influye también el orden en que se toman los bloques de la DCT, dentro de la imagen. Si los bloques de la DCT se toman haciendo un recorrido por filas de la imagen, en el momento en que se llega al fin de una fila, se salta al comienzo de la siguiente fila y estos bloques de la DCT no quedan vecinos, pudiendo existir entre ellos una diferencia de iluminación o de color muy importante, con la consiguiente diferencia en los coeficientes de *continua* de la DCT, con un mayor número de bits necesarios para cuantificarla;

o con una *saturación* del cuantificador escalar, provocando un error en la recuperación del coeficiente.

Codificación implementada

El algoritmo que se implementó es un algoritmo de compresión sin pérdidas (la imagen puede ser recuperada correctamente). Básicamente se calculan las diferencias con respecto al valor de continua del bloque anterior y se las codifica con un cierto número de bits (`nroBits`), pero si el valor a codificar excede el máximo representable con `nroBits`, se envía un código no válido (`noValido`)[1], e inmediatamente después se envía la diferencia sin ningún tipo de cuantificación, igual que el primer coeficiente.

Este procedimiento se hace variando el `nroBits` de 1 a 8, y calculando el tamaño total (incluyendo los `noValidos` y los coeficientes sin codificar) en cada caso; finalmente se elige el `nroBits` que minimiza este tamaño.

El tamaño total se calcula como,

$$\text{Tamaño total} = \frac{nm}{64}\texttt{nroBits} + 9\,\texttt{nroFallos}$$

donde `nroFallos` representa la cantidad de coeficientes que fueron enviados sin cuantificación debido a que superaban el máximo valor posible de cuantificación.

Luego de que se elige el número de bits con el que se va a codificar la *continua*, y se obtienen las diferencias codificadas, se realiza una etapa de compresión de entropía (ver Capítulo 10), en esta etapa se logra disminuir considerablemente el tamaño que ocupan los coeficientes de *continua*.

En el decodificador se procede de la siguiente manera; mientras que no sea recibido el símbolo `noValido` la decodificación se realiza actualizando el valor del último coeficiente a partir de la diferencia que es enviada, pero si se recibe este símbolo, simplemente se utiliza el siguiente coeficiente sin codificación, y el bloque donde se produciría un error, se recupera sin problemas.

Esta última implementación logra niveles de compresión máximos cercanos al 25% en imágenes con un cierto tipo de estadística, por ejemplo Claire, Bird, Camera donde hay grandes regiones con una luminancia casi constante, de manera que al disminuir el número de bits la diferencia entre ellos puede ser codificada; y el número de los bloques donde falla la codificación son pocos, manteniendo un alto porcentaje de compresión (ver Figura

[1] Se aprovecha la existencia del '-0' en la representación signo-mantisa; se representa el rango 2^{N-1} .

9.1).

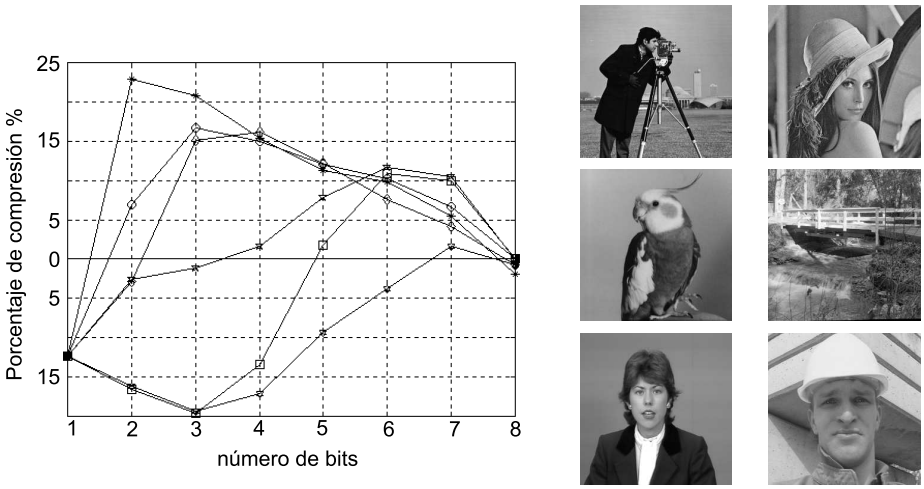

Figura 9.1: `Izquierda:` Porcentaje de compresión para el algoritmo propuesto. `Derecha:` Las imágenes con que se probó.

Para secuencias de imágenes en movimiento, donde se codifican las diferencias entre imágenes consecutivas, la estadística que presentan es muy semejante a las primeras, mientras que no existan movimientos muy rápidos, con grandes zonas donde la diferencia se hace cero (negro) o toma valores muy pequeños, permitiendo ser codificadas con pocos bits y que el número de bloques donde falla la codificación sea bajo.

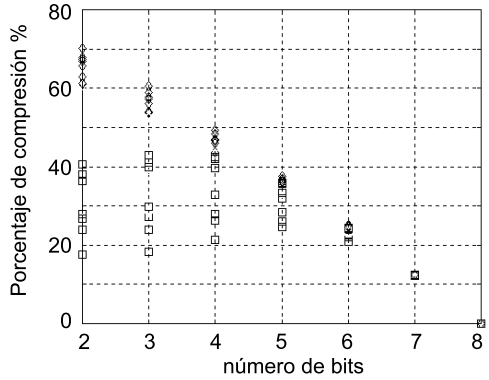

Figura 9.2: Porcentaje de compresión para el algoritmo propuesto. Con □ las diferencias de la secuencia Foreman; con ◇ las diferencias de la secuencia Claire

En la Figura 9.2 se muestran los porcentajes de compresión para varias imágenes diferencia entre dos cuadros de las secuencias Claire y Foreman. Podemos observar que se pueden obtener porcentajes de compresión cercanos al 50%, con un número de bits bajo, 3 ó 4.

El estándar MPEG plantea una forma de codificación de los coeficientes de *continua* de la DCT basado también en la diferencia entre los coeficientes, pero codificándolos en base a una tabla con una codificación de entropía, es decir dándole código de palabras mas cortas a los coeficientes más probables y código de palabras mas largas a los coeficientes menos probables.

Se compararon los resultados que se obtienen con el método presentado[2] y con MPEG; en más del 98% de los casos probados el tamaño final fue inferior (mayor compresión) al obtenido con MPEG, siendo en promedio un 14,8% menor, con valores máximos de hasta un 27%.

[2]Luego de realizar la etapa de codificación de entropía

Capítulo 10

Codificación de entropía

10.1 Introducción

En esta sección vamos a tratar los codificadores de entropía. Estos se definen como técnicas para codificar datos discretos sin ruido en palabras de código de longitud variable, siendo este proceso reversible.

Como ejemplo se pueden citar el código de Morse, y el código de Huffman que asignan palabras más cortas a los símbolos (letras en el caso del código Morse) más frecuentes del alfabeto, y el código RLE (Run Length Encoding) que codifica las corridas de símbolos iguales. Más adelante se profundizará en el código de Huffman.

Un problema que se presenta en la transmisión de datos generados por un codificador de palabras de longitud variable (e.g. codificadores de entropía) es que si se quiere un bit rate constante a la salida del codificador deben usarse buffers. Esto hace al codificador más complejo ya que pueden ocurrir errores debido a un overflow o un underflow por falta de datos a transmitir.

10.2 Condición de optimalidad

El objetivo de un codificador de longitud variable es minimizar la longitud media, $\bar{l}$, de los símbolos a transmitir.

Veamos más formalmente esta última afirmación. Si se tiene una secuencia estacionaria X_n y un alfabeto $A = \{a_0, \ldots, a_{N-1}\}$ tal que

$$P(X = a) = p(a) \text{ si } a \in A,$$

nuestro objetivo será minimizar

$$\bar{l} = E(l(X_n)) = \sum_{a \in A} p(a)l(a).$$

Sería muy bueno entonces, construir códigos que se aproximen a este mínimo y tener una forma de cuantificarlo. El teorema de Kraft nos proporciona un método para hacerlo. Antes de pasar a enunciarlo hay que aclarar que además de minimizar el largo medio de las palabras codificadas, las mismas deben poder ser decodificables para que el codificador sea de utilidad. El termino decodificable significa que dada una secuencia de símbolos codificados, solo existe una única secuencia de símbolos de entrada que la podría haber generado.

Teorema de Kraft (sin demostración)

Dado un alfabeto de N símbolos $A = \{a_0, \ldots, a_{N-1}\}$ con largo de palabra de código $l_k = l(a_k)$, $k = 0, \ldots, N-1$, se cumple que $\sum_{k=0}^{N-1} 2^{-l_k} \leq 1$ sii existe un código decodificable con alfabeto A y largo de palabra l_k. $\diamond$

A partir de este teorema veremos una condición de optimalidad para un codificador.

$$\bar{l} = \sum_{a \in A} p(a) l(a) = -\sum_{a \in A} p(a) \log_2 2^{-l(a)} \overset{(1)}{\geq} -\sum_{a \in A} p(a) \frac{2^{-l(a)}}{\sum_{b \in A} 2^{-l(b)}} =$$

$$\sum_{a \in A} p(a) \frac{1}{\frac{2^{-l(a)}}{\sum_{b \in A} 2^{-l(b)}}} \overset{(2)}{\geq} \sum_{a \in A} p(a) \frac{1}{p(a)} = H(p),$$

De aquí se desprende que $l_{min} = H(p)$, y esto es válido cuando se cumple (1) y (2), o sea que:

(1) $\sum_{b \in A} 2^{-l(b)} = 1$

(2) $p(a) = \frac{2^{-l(a)}}{\sum_{b \in A} 2^{-l(b)}}$

De estas dos condiciones se desprende que el codificador es óptimo cuando $p(a) = 2^{-l(a)}$.

Entonces, dado un codificador sin ruido de longitud variable, $\bar{l} \geq H(p)$, o sea que la longitud media del código no puede ser menor que la entropía. Es por esta razón que esta clase de codificadores son llamados codificadores de entropía. El mínimo se alcanza cuando $p(a) = 2^{-l(a)}\ \forall a \in A$, esto quiere decir que la optimalidad se obtiene cuando las probabilidades de los símbolos son potencias de $\frac{1}{2}$.

10.3 Códigos de prefijos

Este tipo de códigos son un caso especial de códigos decodificables donde cada palabra de código no es prefijo de ninguna otra. A partir de esta propiedad se deduce que cada palabra de código se decodifica tan pronto como se descubre una palabra válida, no se deben decodificar más símbolos para resolver ambigüedades. A pesar de ser un caso particular de codificadores se puede demostrar que no hay pérdida de optimalidad al utilizar códigos de prefijos. A continuación se muestran dos propiedades interesantes de los codificadores de prefijos óptimos:

1. Si la palabra de código para un símbolo de entrada a tiene largo $l(a)$, entonces $p(a) > p(b)$ implica que $l(a) \leq l(b)$.

2. Los dos símbolos menos probables tienen igual largo de palabra y difieren solamente en el símbolo final.

Los códigos de Huffman son códigos de prefijo y por lo tanto estas propiedades le serán aplicables. En la próxima sección se utilizaran estas propiedades para construir códigos de Huffman.

10.4 Codificador de Huffman

Este codificador genera una longitud media de palabras de código muy cercana a la entropía y si las probabilidades de los símbolos a codificar son potencia de $\frac{1}{2}$ se tiene que el codificador es optimo. A continuación se presenta una breve descripción de cómo generar las palabras de código utilizando propiedades de códigos de prefijo [11]. El objetivo es la construcción de un árbol binario cuyas hojas sean las palabras válidas del código, de esta forma ninguna palabra es prefijo de otra. Los códigos generados con este algoritmo se obtienen recorriendo el árbol desde la hoja hasta el nodo raíz, agregando un uno por cada rama que se recorre por la derecha y un cero por cada rama que se recorre por la izquierda. Sin embargo, la palabra de código debe generarse desde el nodo raíz hacia la hoja para evitar que haya palabras de código que sean prefijo de otras.

Supongamos que ordenamos los N símbolos de entrada del alfabeto $A = \{a_0, ..., a_{N-1}\}$ en función de su probabilidad, o sea, $p(a_0) \geq ... \geq p(a_{N-1})$ y construimos la siguiente tabla:

a_0	$p(a_0)$
a_1	$p(a_1)$
:	:
a_{N-1}	$p(a_{N-1})$

La construcción se hará desde las hojas (del árbol a construir) hacia el nodo raíz utilizando la propiedad que dice que los dos símbolos de menor probabilidad tienen el mismo largo y solamente difieren en el ultimo símbolo (bit en este caso). Por lo tanto, se comienza el árbol con dos nodos hoja que corresponden a los dos símbolos de menor probabilidad (i.e. mayor longitud) y un nodo que es padre de ellos, luego se eliminan los dos símbolos de menor probabilidad de la tabla (ya agregados al árbol) y se agrega un nuevo símbolo

$$(a_{N-1}, a_{N-2})$$

con probabilidad

$$p(a_{N-1}) + p(a_{N-1}).$$

Se puede ver que si el código de prefijos para el alfabeto

$$A' = \{a_i / i = 0, ..., N-3; (a_{N-1}, a_{N-2})\}$$

es óptimo, también lo es el código para el alfabeto A.

Iterando de esta forma se obtiene el árbol de Huffman, donde todas las hojas son palabras de código válidas; el proceso termina cuando queda un solo símbolo en la tabla. Este codificador escalar, sin ruido, de largo variable y decodificable, es óptimo.

Se podría mejorar el rendimiento del codificador considerando alfabetos no binarios o realizando codificaciones no escalares. En el siguiente cuadro se presenta un resumen del algoritmo para generar el árbol binario para el codificador de Huffman.

Algoritmo para generar el árbol binario para el codificador de Huffman

PASO 1: Sea L una lista de todas las probabilidades de los símbolos a codificar.

PASO 2: Tomar los dos símbolos de menor probabilidad de la lista L y hacerlos nodos siblings (hermanos). Generar un nodo intermedio como su padre y etiquetar la rama del padre hacia uno de sus hijos con un uno y la otra con un cero.

PASO 3: Reemplazar los dos nodos de la lista L que tienen menor probabilidad por el nuevo nodo intermedio creado en el **PASO 2** cuya probabilidad es la suma de las probabilidades de los dos nodos reemplazados. Si la lista contiene un elemento, terminar, sino ir al **PASO 2**.

En cuanto a la trasmisión de los datos existen dos formas de que el receptor pueda decodificar los datos comprimidos por el algoritmo de Huffman. En ambos casos se debe enviar la información comprimida (obviamente) más la información necesaria para poder decodificarla (overhead). El overhead puede consistir en todos los símbolos junto con sus códigos, en cuyo caso el decodificador debería solamente decodificar buscando en una tabla con los símbolos y códigos. Otra posibilidad es enviar únicamente las probabilidades de los símbolos y que el decodificador calcule el árbol de Huffman y a partir de ahí realice la decodificación de la misma forma que el método anterior. La segunda opción tiene la ventaja de que necesita enviar menor información (comprime más), aunque necesita realizar más cálculos para realizar la decodificación (más lento).

Códigos adaptativos

El algoritmo recién presentado supuso el conocimiento de las probabilidades de los símbolos de entrada, pero éstas en general no son conocidas. Más aun, estas probabilidades varían con el tiempo, lo que lleva a que los códigos de Huffman (y por lo tanto el árbol) deban recalcularse para adecuarse a estas variaciones.

La primer estrategia que se podría pensar sería ir calculando la cantidad de veces que se repite cada símbolo y a partir de ahí, calcular los códigos de Huffman. O sea, la estimación de la probabilidad del símbolo a, dados N símbolos de entrada, sería

$$p(a) = \frac{n_N(a)}{N}$$

siendo $n_N(a)$la cantidad de veces que se repite a en la secuencia de N símbolos de entrada.

Siguiendo esta metodología, la adaptación se podría hacer recalculando las probabilidades de los símbolos cada M símbolos. En este caso se tiene un compromiso, cuanto mayor es M, mayor es el retardo que se genera para el calculo de los códigos, pero el overhead de la información adicional sobre los símbolos codificados es menor.

Existe una forma más eficiente de realizar la adaptación, la idea de este método es ir modificando el árbol de Huffman en vez de recalcularlo cada una determinada cantidad de símbolos.

En este caso se utilizan pesos proporcionales a las probabilidades, estos pesos pueden ser la cantidad de ocurrencias de cada símbolo. Entonces, una vez que se tiene el árbol de Huffman, por cada nuevo símbolo que se codifica

se va adaptando el árbol. Dado un símbolo de entrada, a_i, por cada ocurrencia se le aumenta su peso en una unidad y se verifica su posición en la lista ordenada L, si no supera ningún símbolo de la lista, se sigue recorriendo el árbol y se opera de la misma manera, aumentando en una unidad el peso del nodo padre de a_i. Si en cambio, se supera un símbolo de la lista L, se deben intercambiar dicho símbolo y a_i.

De esta forma se recorre el árbol hasta la raíz y se obtiene el código para el símbolo y a la vez se adapta el árbol.

Hay que observar que con esta técnica el decodificador puede realizar la adaptación sin necesidad de overhead.

10.5 Codificador RLE

Otro tipo de codificador de entropía es el RLE (Run Length Encoder). A diferencia del codificador de Huffman, este no debe conocer las probabilidades de ocurrencia de los símbolos de entrada.

En este caso se van leyendo los símbolos de entrada y se buscan corridas de símbolos iguales (ceros y unos en el caso binario) enviando solamente el símbolo que generó la corrida junto con la cantidad de veces que se repitió.

En general este codificador alcanza grados de compresión aceptables si hay grandes corridas de símbolos y este no es el caso de símbolos de entrada generados por vectores de un codebook. Una mejor aplicación de este método es para comprimir los datos que transmite un fax ya que generalmente tiene grandes corridas de ceros.

10.6 Pruebas y conclusiones

En esta ultima parte se presentan algunas comparaciones entre el algoritmo de Huffman y el conocido programa para comprimir archivos,WinZip.

Para estas pruebas se tomaron algunos archivos que representan cada uno de ellos un cuadro codificada vectorialmente, su formato (`vqg`) se explica en §12.2.3.

Es importante aclarar que para realizar la compresión se utilizaron diferentes árboles de Huffman ya que los datos a enviar al decodificador contienen diferentes estadísticas. En particular, para este codificador se utilizaron 3 árboles de Huffman diferentes para la compresión de imágenes

inter, uno para los vectores de movimiento, otro para los índices de los codebooks y otro para los coeficientes de continua.

En el caso de las imágenes *intra* se utilizaron dos árboles, uno para los índices de los codebooks y otro para los coeficientes de *continua*.

En la siguiente tabla se muestran los tamaños de las imágenes sin comprimir, luego de ser comprimidas con el algoritmo de Huffman y comprimidas con el programa WinZip.

Las secuencias utilizadas fueron Foreman y Carphone.

Foreman

Número de *frame*	Tamaño del archivo sin comprimir	Tamaño del archivo `vqg`	Tamaño del archivo `zip`
0	12906	4838	4785
1	11724	5109	4241
2	11397	4085	3930
3	11261	3971	3778
4	11248	4008	3778
5	11185	3966	3702

Nota: Los tamaños son medidos en bytes.

Carphone

Número de *frame*	Tamaño del archivo sin comprimir	Tamaño del `vqg`	Tamaño del `zip`
0	9823	4349	3934
1	12196	5165	4085
2	12046	4148	4029
3	11539	4082	3778
4	12155	4135	3863
5	12296	4168	3948

Nota: Los tamaños son medidos en bytes.

Se puede notar que el algoritmo WinZip comprime más la información a transmitir que el algoritmo de Huffman y que dados los niveles de compresion que se obtienen es muy importante la etapa de codificación de entropía. Como se puede observar en las tablas, el tamaño de los archivos comprimidos es del orden de 2 ó 3 veces menor que el original para las secuencias utilizadas (Carphone y Foreman).

Capítulo 11

Análisis de desempeño en secuencias

11.1 Pruebas y Resultados

Ha llegado el momento de integrar cada uno de los bloques desarrollados para formar un sistema de codificación-decodificación de secuencias.

Lo primero a destacar es que la optimización conjunta de todos los bloques del codificador/decodificador es un problema de muy difícil solución pues existe mucha interacción entre cada bloque y los restantes. Por más que durante todo el desarrollo del proyecto se intentó estudiar cada bloque independientemente, se vio que a la hora de unir todos los bloques para construir el codificador, la interacción de cada bloque con los demás empieza a pesar fuertemente y los valores óptimos que se habían hallado con anterioridad (estudiando los bloques por separado) pueden dejar de ser las mejores opciones.

Se creó una plataforma para realizar las pruebas, fácilmente configurable desde archivos, que permitió modificar los valores de los muchos parámetros del codificador y con ello la interacción entre los distintos bloques.

Las pruebas que se realizaron no sólo sirvieron para evaluar la performance del codificador/decodificador, sino también para evaluar el desempeño de cada uno de los bloques por separado y, producto de esta evaluación, se pudo repensar muchos de los algoritmos implementados.

Se presentan los resultados obtenidos primero para la codificación de imágenes fijas (intra) y luego se verán los resultados que se obtuvieron para la codificación de secuencias.

11.1.1 Codificación de imágenes

Se probaron dos esquemas distintos para tomar las bandas de frecuencias y se variaron las dimensiones de cada una de ellas y el tamaño de los *codebooks* (ver Figura 11.1). El primer esquema se realizó siguiendo el recorrido *zig-zag* del estándar MPEG (§7.2), quedándose con los coeficientes de las frecuencias bajas [1].

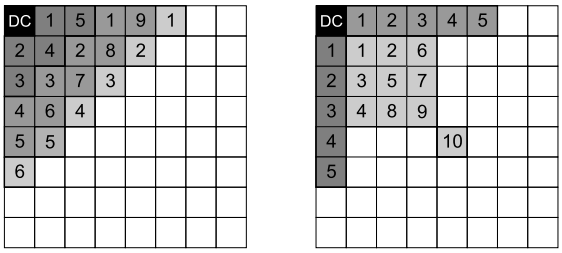

Figura 11.1: Forma en que se dividieron las bandas. **Izquierda:** Según la *frecuencia* (recorrido *zig-zag*). **Derecha:** Según la orientación.

En la Tabla 11.1, se presentan los valores de los parámetros que se utilizaron y los datos obtenidos con el primer esquema planteado.

En la Tabla 11.2, se presentan los valores de los parámetros que se utilizaron y los datos obtenidos con el segundo esquema planteado.

Imagen	Bird	Lena	Claire	Foreman	Bridge	Goldhill
Tamaño del *Codebook* de la 1 Banda	128	128	128	128	128	128
Tamaño del *Codebook* de la 2 Banda	64	64	64	64	64	64
Tamaño del *Codebook* de la 3 Banda	32	32	32	32	32	32
PSNR (dB)	34.5	27.3	34.9	32.2	24.8	27.1
Bits por pixel	0.322	0.443	0.296	0.396	0.512	0.455

Tabla 11.1: Resultados de las pruebas para el ordenamiento en zig-zag.

Teniendo varias bandas que cubren los coeficientes, se tiene la posibilidad de tener distintos *codebooks*, especialmente creados para cada una de ellas, pudiendo darles mayor importancia a ciertas bandas, asignándoles, por ejemplo, mayor número de vectores al *codebook* correspondiente. Esto se ve claramente en la primera de las estrategias, dándole mayor tamaño a la primera banda con respecto a las otras dos.

Imagen	Bird	Lena	Claire	Foreman	Bridge	Goldhill
Tamaño del *Codebook* de la 1 Banda	64	64	64	64	64	64
Tamaño del *Codebook* de la 2 Banda	64	64	64	64	64	64
Tamaño del *Codebook* de la 3 Banda	64	64	64	64	64	64
PSNR (dB)	34.8	27.3	35.0	32.4	24.7	27.1
Bits por pixel	0.326.	0.427	0.290	0.386	0.485	0.432

Tabla 11.2: Resultados de las pruebas para el ordenamiento con orientaciones.

El segundo esquema presentado demostró tener mejores resultados; si bien la mejora en la calidad de la imagen es reducida, el aumento en el PSNR nunca supera los 0.5 dB, el resultado perceptual (algo que no puede medir el PSNR) es bastante mejor.

11.1.2 Codificación de Secuencias

Para evaluar el desempeño del decodificador se utilizaron secuencias estándares de formato CIF (352×288). Se elige este formato a los efectos de poder realizar comparaciones con el estándar MPEG1 el cual sólo soporta los formatos CIF y QCIF. Además el hecho de elegir secuencias estándares nos permite comparar nuestros resultados con los presentados en las publicaciones del área.

En las pruebas se mide en particular la calidad obtenida (por medio del PSNR) y la compresión alcanzada (a través de los bits por pixel). Presentaremos los promedios alcanzados en la secuencia para las distintas configuraciones del codificador.

Para todas las pruebas que se presentan a continuación se utilizó la búsqueda exhaustiva con un área de búsqueda de ±7 píxeles. Para todas las imágenes intra se calculan los codebooks y se transmiten al decodificador, pues se pudo comprobar que un mala imagen intra degrada de forma importante la calidad de las imágenes siguientes. Después de cada cuadro intra, se calculan los *codebooks* para los cuadros diferencia una sola vez (es decir, para la primer imagen consecutiva a la intra) y los siguientes cuadros se codifican utilizando éstos. La razón por la que se restringe tanto el cálculo de *codebooks* es, obviamente, que estos deben transmitirse al decodificador. Los bloques intra dentro de las imágenes diferencia se codifican utilizando los *codebooks* correspondientes a la última imagen intra codificada.

En la Figura 11.2 se muestra la gráfica de PSNR vs bits por pixel para las distintas dimensiones y tamaños de *codebooks*. Las regiones más interesantes a destacar de esta gráfica son:

(1) una zona de bajos bpp's y calidad baja

(2) una zona de bpp's altos y una calidad intermedia

(3) una zona de bpp's altos y calidad alta

Un aspecto interesante a observar es que la zona (2) es menos conveniente que la zona (3), la cuál con otro juego de parámetros logra una calidad superior con una relación de bits por píxel similar.

La zona (1), es una zona de bpp's bajos, con la consiguiente reducción del PSNR. Estos valores fueron logrados al utilizar solamente una banda de coeficientes de la DCT (en lugar de tres como en los otros casos), lo cuál nos lleva a considerar muy pocos coeficientes. Las dimensiones consideradas para estos vectores variaron entre 5 y 9 coeficientes.

El tamaño de los codebooks utilizados en estos casos también fue muy reducido, del orden de unos 16 vectores. El tamaño de macrobloque con que realizó la estimación de movimiento es de 16×16 píxeles.

Las pruebas de las regiones (2) y (3) se realizaron con tres bandas de coeficientes (de dimensiones 5, 9 y 6) y tamaño de codebook mayores que en la región (1), llegando a valores de 128 vectores en alguno de los codebooks. La principal diferencia entre estas dos regiones, fue el tamaño de los macrobloques en la estimación de movimiento, siendo de tamaño 16×16 píxeles en la región (2) y de 8×8 píxeles en la región (3).

De la comparación con MPEG resulta, que esta codificación no logra alcanzar los niveles obtenidos por éste. Sin embargo, los resultados obtenidos no están alejados de los del estándar.

11.2 Conclusiones

Los resultados obtenidos nos parecen aceptables y dejan abierta la posibilidad de introducir cambios que mejoran el desempeño del sistema de codificación y decodificación.

Nos parece importante destacar la función que la estimación de movimiento cumple en la codificación de secuencias de video. Es difícil pensar en enfrentar una codificación de video sin tener en cuenta mecanismos que exploten la redundancia temporal. Esto queda en evidencia al comparar los

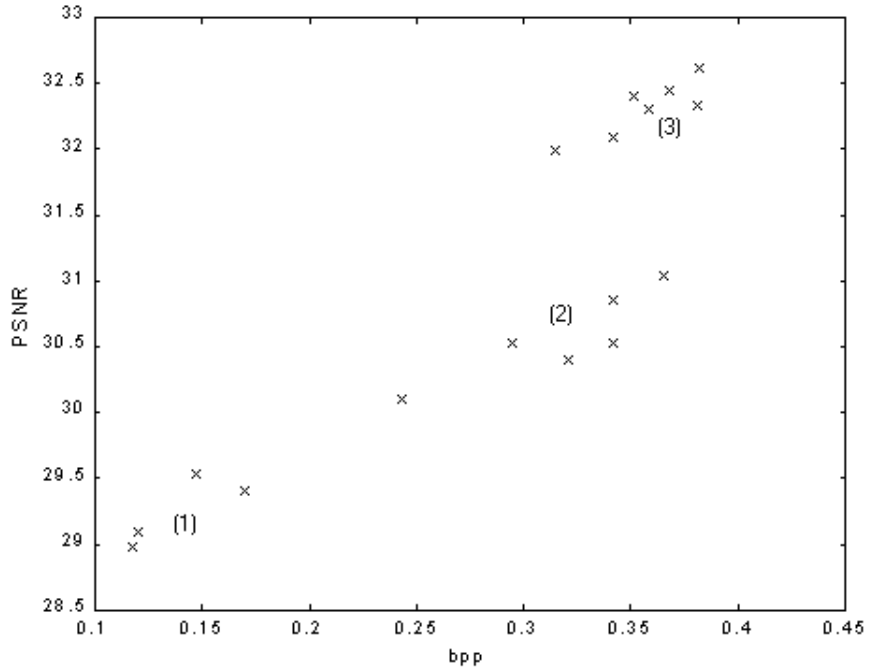

Figura 11.2: PSNR vs. *bpp's* para la secuencia Carphone.

resultados de calidad de imagen obtenidos al comprimir imágenes fijas con los obtenidos al codificar secuencias de video: la calidad de los distintos cuadros de la secuencia de video es muy superior para un mismo nivel de compresión.

La cuantificación vectorial es una técnica poderosa de compresión de señales en general; siendo fundamental para el éxito de la compresión el diseño de codebooks adaptados a la estadística de la señal a comprimir.

Se pudo comprobar que en la codificación de imágenes naturales es posible utilizar codebooks que fueron entrenados para otras imágenes, y aun así obtener resultados satisfactorios. Para la codificación de secuencias de video, esta característica se mantiene incambiada.

La cuantificación vectorial tiene una carga computacional importante. Esto y el hecho de que se trabaja sobre una plataforma de propósito general no adecuada para la realización de cálculos pesados hacen que sea imposible implementar un codificador de video en tiempo real, con las severas exigencias temporales que esto implicaría.

A pesar de esto, se logra un decodificador vectorial que trabaja en tiempo

real debido a que, como ya se mencionó, la decodificación vectorial consiste en una sencilla búsqueda en tabla.

La implementación de nuestro cuantificador vectorial es estática, dificultando la adaptación frente a cambios de estadística. La adaptación de los *codebooks* es un primer acercamiento a resolver esta cuestión pero, claramente, no fue suficiente. El principal problema radica en que muchas veces es conveniente desechar algunos coeficientes y otras veces cuantificar otros. Esto no es modificable sobre la marcha, la decisión de cuales son los coeficientes a cuantificar se debe tomar a priori basándose en la compresión que se pretende lograr. Los cambios de las componentes frecuenciales en la estadística de los cuadros, no son tomados en cuenta para definir qué coeficientes se han de cuantificar.

Es por esto que aparece en las secuencias decodificadas el fenómeno de *ringing* o ruido de alta frecuencia. Este problema está siempre presente, incluso cuando se utilizan *codebooks* de dimensión alta y la secuencia se decodifica con altos valores de PSNR.

A nuestro juicio los objetivos del proyecto han sido cubiertos, pues se ha estudiado en profundidad un abanico de técnicas de cuantificación vectorial aplicadas a la codificación de video. Esto nos proporcionó un amplio horizonte de las posibilidades que se abren con la aplicación de esta técnica a los coeficientes de la DCT, problema que según nuestro conocimiento no había sido estudiado hasta el momento.

Capítulo 12

Implementación del paquete de software

12.1 Detalles de implementación

Como lenguaje de programación se optó por utilizar el C++[12] y como ambiente de desarrollo el Microsoft Developper Studio 6.0. Este ambiente de desarrollo contiene varias utilidades integradas, como ser el compilador, el depurador (debugger), ayuda en línea, la biblioteca de plantillas estándar (STL), etc. Para la implementación de la interfaz gráfica[10] se optó por utilizar Visual C++, ya que puede ser usado dentro del mismo ambiente de desarrollo y permite, entre otras cosas la implementación de interfaces amigables al usuario brindando las facilidades de botones de comando, cuadros de diálogo, y el manejo del programa en el ambiente Windows9x.

La implementación del sistema se llevó a cabo siguiendo el paradigma de la Programación Orientada a Objetos (OOP), fundamentalmente por las facilidades de reutilización de código que debe proporcionar una plataforma desarrollada para la investigación, dado el cambio permanente que debe soportar.

Para manejar distintos formatos de imagen, se decidió utilizar la biblioteca conocida como Image Magick, que proporciona las facilidades necesarias para leer y escribir archivos de imágenes (imágenes fijas) en un gran número de formatos estándar (JPEG y BMP ente otros). Además de esto, permite rotar imágenes y convertir secuencias de imágenes al formato GIF.

12.1.1 La Standard Template Library (STL)

Para la manipulación de grandes cantidades de datos, hecho muy usual en el tratamiento de imágenes, se utilizó en varias oportunidades la biblioteca

de plantillas estándar (STLs). Estas bibliotecas están compuestas de los siguientes elementos:

- **Contenedores** Son objetos que se utilizan para guardar otros objetos, por ejemplo, un vector de enteros. Existen contenedores secuenciales y asociativos (se ordenan o recuperan mediante una clave) como ser listas, vectores, etc.
- **Algoritmos** Se utilizan para realizar operaciones sobre objetos de un contenedor, por ejemplo, ordenar una lista de enteros.
- **Funciones** Se aplican sobre cualquier tipo de objeto, usualmente para determinar si determinada condición se satisface. Por ejemplo, puede emplearse para comparar dos enteros y decidir cual es mayor, de esta forma, junto con un algoritmo de ordenamiento se puede ordenar un vector de enteros en forma creciente.
- **Iteradores** Se utilizan para recorrer los contenedores

Un aspecto fundamental de todo esto es la reutilización del código, a través de la independencia entre los contenedores, algoritmos y funciones.

Las características más importantes de la biblioteca son:

- Posee contenedores genéricos que se pueden aplicables a cualquier tipo de objeto.
- Es software que ha sido verificado.
- Es fácil de usar, la forma de programar sobre diferentes contenedores es muy parecida ya que poseen muchas funciones miembro en común.
- Facilita el trabajo en grupo y la lectura del software para otras personas ya que las clases que se utilizan son estándar.

Esto fue de gran ayuda ya que no se tuvieron que programar clases para utilizar vectores que aumentan su tamaño en forma dinámica, esto hubiera significado programar todas las funciones miembro necesarias para la clase, tampoco se programaron algoritmos ni funciones para ordenarlos.

12.2 Codificador

En esta sección se presenta el software que se encarga de la codificación de una secuencia, su configuración, la forma de ejecutarlo, y las características de los archivos de entrada y salida.

12.2.1 Interfaz

El codificador corre bajo el ambiente DOS, y se ejecuta desde su línea de comandos.

La forma de codificar una secuencia de entrada es la siguiente:

```
$ larunbat nombre_secuencia_in nombre_secuencia_out
```

En la sentencia anterior, `nombre_secuencia_in` es el prefijo del nombre de los archivos que contienen los datos de cada uno de los cuadros de la secuencia *a codificar* (ver §12.2.2). `nombre_secuencia_out` es el prefijo del nombre de los archivos que contienen los datos de cada uno de los cuadros de la secuencia *codificados* (ver §12.2.3).

Por ejemplo, el comando:

```
$ larunbat for foreman
```

codificará la secuencia que tiene los mapas de bits de los distintos cuadros en los archivos `forXXX.bmp`; y los archivos de la salida `foremanXXX.vqg`. El número de cuadros que se codificarán se establece en los archivos de configuración (ver §12.2.2).

12.2.2 Secuencia de entrada

Los archivos correspondientes a cada uno de los cuadros de la secuencia a codificar deben encontrarse en el directorio `entrada/`. Las imágenes deben ser archivos de mapa de bits de 256 niveles de gris (8 bits), de dimensiones CIF, 352×288 píxeles.

Archivos de entrada

Dado el nombre de entrada (`nombre_secuencia_in`) y el número de frames a codificar, se codificarán secuencialmente cada una de las imágenes.

El nombre del archivo de mapa de bits correspondiente al *frame* n-ésimo debe ser, `nombre_secuencia_inXXX.bmp`, donde `XXX` indica el número del cuadro (n en este caso).

Por ejemplo el archivo de mapa de bits correspondiente al *frame* 81 de la secuencia Carphone, lleva como nombre, `carphone081.bmp`.

Configuración

La configuración de las distintas variantes con las cuales se puede realizar la codificación de una secuencia, se realiza principalmente desde dos archivos.

La configuración del codificador es difícil y pesada para un usuario sin un conocimiento profundo del mismo. El motivo de esta dificultad radica en que, a los efectos de realizar una gran cantidad de pruebas, el codificador debía proporcionarnos una gran libertad para cambiar todos sus parámetros y estrategias.

Los archivos de configuración son:

- `configVQ.cnfg` - Tiene los parámetros relativos a la cuantificación vectorial, dimensiones de los vectores, tamaño de los codebooks, tanto para los *frames inter* como los *frames intra*. Estos parámetros son:
  - `[Numero de cuadros]`: número de cuadros en la secuencia.
  - `[dim1]`, `[dim2]`, `[dim3]`: dimensiones de las bandas en los bloques *inter*
  - `[dim1 Intra]`, `[dim2 Intra]`, `[dim3 Intra]`: dimensiones de las bandas en los bloques *intra*
  - `[nroVect1]`, `[nroVect2]`, `[nroVect3]`: número de vectores de los codebooks de cada una de las bandas de los bloques *inter*
  - `[nroVect1 Intra]`, `[nroVect2 Intra]`, `[nroVect3 Intra]`: número de vectores de los codebooks de cada una de las bandas de los bloques *intra*
  - `[Calcular Codebook]`: estrategia para calcular los codebooks de los bloques *inter*. Este parámetro puede ser:
    - 0: no se calculan codebooks, se utilizan codebooks ya existentes
    - 1: se calculan los codebooks en cada caso
    - 2: se calculan codebooks a partir de codebooks existentes más grandes
    - 3: se calcula codebook para el primer cuadro, y luego se utiliza este fijo.
  - `[Calcular Codebook Intra]`: estrategia para calcular los codebooks de los bloques *intra*. Este parámetro puede ser el mismo que en el caso anterior.
  - `[Los inadaptados de siempre]`: indica el número de cuadros que se codificarán utilizando *algún codebook* (dependiendo de la estrategia utilizada), después de ese cuadro la codificación se hará basándose en la adaptación del último *codebook* utilizado.

Este archivo permitía también la determinación de algunos otros parámetros que luego de realizadas varias series de pruebas fueron fijados con lo que el manejo del codificador resulta más sencillo. Estos parámetros fueron: los tres umbrales de pruning y sus correspondientes coeficientes de actualización tanto para los bloques intra como para los bloques inter y las columnas donde se iniciaban cada una de las tres bandas de frecuencia para los bloques intra y los inter.

- `configMC.cnfg` - Tiene los parámetros relativos a la configuración de la estimación y compensación de movimiento. Estos son:
  - `[Movimiento]`: determina el algoritmo de estimación a utilizar.
  - `[Margen de busqueda]`: determina el margen de búsqueda (en píxeles) en todos los sentidos.

12.2.3 Secuencia codificada

Los datos generados por el codificador, que serán necesarios para decodificar la secuencia, se encuentran distribuídos en diferentes archivos.

Los diferentes archivos se distinguen por su extensión, siendo posibles los siguientes casos:

- **vqg** - contiene los datos específicos de la codificación de un cuadro cualquiera
- **cnfg** - archivo de configuración, contiene los datos con que fue configurada la codificación de la secuencia (dimensión de las bandas, tamaño de los codebooks, etc.).
- **cdbk** - codebook.

Archivos de salida

Estructura del archivo de salida para un frame `inter`:

- un bit indicando si el frame es intra o inter
- número de bloques intra
- `nroBitsDC`, numero de bits con que se cuantifican los coeficientes de continua
- cadena de caracteres obtenidos al codificar los siguientes datos: coeficientes de continua de los bloques intra, índices de las tres bandas de los bloques intra, los índices de los codebooks grandes utilizados para la adaptación de los codebooks de los bloques intra.

- caracteres obtenidos al codificar los vectores de movimiento, con el algoritmo de Huffman
- caracteres obtenidos al codificar los índices de las bandas de los bloques inter, con el algoritmo de Huffman
- caracteres obtenidos al codificar los coeficientes de continua, con el algoritmo de Huffman

Estructura del archivo de salida para un frame `intra`:

- un bit indicando si el frame es intra o inter.
- `nroBitsDC`, numero de bits con que se cuantifican los coeficientes de continua
- cadena de caracteres obtenidos al codificar los coeficientes de continua que no fueron codificados con `nroBitsDC` (`noValidos`)
- caracteres obtenidos al codificar los índices de las bandas de los bloques inter, con el algoritmo de Huffman
- caracteres obtenidos al codificar los coeficientes de continua, con el algoritmo de Huffman

Codificación de imágenes fijas

La codificación de imágenes fijas se realiza en forma similar a como se codifica una imagen *intra* en una secuencia.

Para codificar una imagen se debe incluir en la línea de comandos la opción `-m`, el nombre de la imagen de entrada y el prefijo de la imagen codificada.

Por ejemplo, para codificar la imagen `lena.bmp` y que la salida vaya en el archivo `lenaCod.vqg`, se debe ejecutar la siguiente sentencia:

```
$ larunbat -m lena.bmp lenaCod
```

En los archivos de configuración, importarán los datos correspondientes a los cuadros *intra*

12.3 Interfaz Gráfica

El decodificador presenta una interfaz gráfica donde se despliega la secuencia que se desea decodificar y se muestra al mismo tiempo el progreso en la decodificación.

12.3.1 Decodificación y despliegue de una secuencia

Estructura de los directorios

Para decodificar una secuencia dada, deben estar presentes los archivos correspondientes en los directorios, como se especifica a continuación.

En el directorio `secuencias/`

- `nombreSecuencia.cnfg`

En el directorio `secuencias/nombreSecuencia/`

- `nombreSecuenciaXXX.vqg`
- `nombreSecuenciaXXX.cdbk`

estos archivos corresponden a los datos de la secuencia codificada.

Además pueden ser utilizados archivos correspondientes a codebooks "universales"; estos se deben encontrar como se especifica a continuación.

En el directorio `codebooks/`

- `codebookUniversalX.cdbk`

Decodificación y reproducción de una secuencia

Para que se decodifique la secuencia `nombreSecuencia` deben seguirse los siguientes pasos:

1. Abrir el cuadro *Decodificar secuencia* (ver Figura 12.2) de alguna de las siguientes formas (ver Figura 12.1):
   - Presionar `Ctrl+D`.
   - Seleccionando en el menú `Decodificar`, la opción `Decodificar secuencia`.
   - Presionar el botón `Play` en la barra de herramientas.
2. Escribir el nombre de la secuencia (`nombreSecuencia`)
3. Dar `Aceptar`.

El archivo elegido (`nombreSecuencia.cnfg`) debe estar en el directorio antes mencionado (`secuencias/`).

Al elegirse una secuencia para decodificar, se crea un nuevo *thread*, el cual se encarga exclusivamente de la decodificación, y se destruye al terminar la misma.

Figura 12.1: Decodificar una secuencia.

Figura 12.2: Cuadro de diálogo *Decodificar secuencia*.

Detener la decodificación y reproducción de la secuencia.

Es posible detener la decodificación y el depliegue de la secuencia. Para llevar a cabo esta acción hay tres posibilidades (ver Figura 12.3).

1. Presionar `Ctrl+C`.
2. Seleccionando en el menú `Decodificar`, la opción `Detener decodificación`.
3. Presionar el botón `Stop` en la barra de herramientas.

Pausar la decodificación y reproducción de la secuencia.

Es posible realizar una pausa en la decodificacion y despliegue de la secuencia. También hay tres formas de lograr esto,

Figura 12.3: Detener y/o pausar la decodificación de la secuencia.

1. Presionar `Ctrl+P`.
2. Seleccionando en el menú `Decodificar`, la opción `Pausar decodificación`.
3. Presionar el botón de `Pausa` en la barra de herramientas.

Para continuar con la decodificación y el despliegue debe hacerse alguna de las tres acciones recién comentadas.

12.3.2 Decodificación y guardado de una secuencia

Esta variante en la ejecución del decodificador permite guardar cada uno de los cuadros a medida que van siendo decodificados. Mientras se decodifica la secuencia y se guardan cada uno de los distintos cuadros la secuencia no se despliega.

La estructura de directorios en que se encuentran los archivos es similar al caso comentado en 12.3.1, y los pasos que se deben seguir son los mismos que se comentan en 12.3.1, excepto que debe seleccionarse la casilla de *Guardar cuadros como bmp's* en el cuadro de diálogo *Decodificar secuencia* correspondiente(ver Figura 12.2).

Cada uno de los cuadros decodificados se guardará como un archivo con extensión *bmp*, correspondiendo con el nombre de la secuencia y con el número de cuadros dentro de la misma.

Parte IV

Líneas de Trabajo a Seguir

Capítulo 13

Líneas de Trabajo a Seguir

Este proyecto podría extenderse mucho más aun. Algunos de los posibles temas a estudiar son:

- *VQ + Máscaras de cuantificación*
  Estudio de la influencia de las máscaras de cuantificación aplicadas a los coeficientes de la DCT en la cuantificación vectorial. Por medio de la aplicación de éstas se podría dar más peso en la cuantificación a algunos coeficientes del vector con respecto a otros coeficientes pertenecientes al mismo vector.

- *VQ + DCT*
  Implementar un esquema de codificación en el cual el número de bandas a codificar pueda variar dinámicamente de manera de llegar a cubrir todos los coeficientes de la DCT para eliminar el efecto del *ringing*.

- *VQ + DCT en tres dimensiones*
  La DCT tridimensional es una técnica que consiste en codificar las imágenes en bloques de ocho, aplicando la DCT en el tiempo a las transformadas de las ocho imágenes. La VQ se utiliza para codificar los coeficientes de la 3D-DCT. Esto permite eliminar la compensación de movimiento y se obtiene un esquema de codificador/decodificador radicalmente distinto al que se obtiene en el punto anterior.

- *Wavelets + VQ*
  Eventualmente se podría sustituir la DCT por la transformada wavelets con la intención de reducir los efectos de bloque característicos de la anterior.

- *Búsquedas rápidas*
  Técnicas de búsquedas rápidas que permitan la codificación de señales en tiempo real.

- *Estimación de Movimiento*
  Técnicas de estimación de movimiento no basadas en *block matching*, sino basadas en regiones o con identificación de objetos dentro de la imagen.

- VQ + Codificación de Entropía
  Estudiar a fondo la interacción entre la cuantificación vectorial y la codificación de entropía. Por ejemplo, analizar la variación del nivel de compresión logrado por la codificación de entropía al variar el tamaño de los codebooks con que se cuantifica una misma señal.

Parte V

Apéndices

Apéndice A

Codificación Vectorial de Imágenes

En este capítulo se abordarán dos líneas de trabajo relacionadas con la codificación vectorial directa de la imagen, a saber:

- *Compresión de imágenes*
  La idea es simplemente dividir una imagen fija (mapa de bits) en bloques no solapados y codificar éstos utilizando cuantificación vectorial. Esto nos permitirá evaluar las posibilidades de la VQ como herramienta de compresión de video.

- *Compresión de imágenes con media removida*
  En este esquema, para cada bloque se codifican por separado su medio y su *forma*. La codificación vectorial se aplica a la forma del bloque mientras que la media simplemente se codifica escalarmente.

A.1 Compresión de imágenes fijas

Tratemos de ver cuál es la capacidad de compresión de la cuantificación vectorial aplicada a imágenes fijas.

Supongamos una imagen de $x \times y$ píxeles dividida en bloques cuadrados de lado l y un *codebook* de N vectores con $N = 2^n$. Consideremos además que la imagen viene dada en tonos de grises con 256 valores posibles (2^8).

En estas condiciones la razón de bits por pixel (bpp) resulta:

$$bpp = \frac{\text{bits empleados}}{\text{píxeles codificados}} = \frac{\text{imagen codificada + } \textit{codebook}}{\text{píxeles codificados}} = \frac{\frac{x}{l}\frac{y}{l}n + 8Nl^2}{xy}$$

Si se supone, como es usual, que el número de bits empleados para codificar la imagen es mucho mayor que el número de bits empleados para

representar el *codebook* (esto es $\frac{x}{l}\frac{y}{l}n \gg 8Nl^2$) tenemos que $bpp = \frac{n}{l^2}$.

Los resultados obtenidos en la práctica muestran que tomar $l > 4$ produce un efecto cuadriculado demasiado notorio en la imagen (para imágenes en formato CIF de 352x288). Por otra parte utilizar $n = 4$ (o sea trabajar con 16 vectores) proporciona en general una calidad poco aceptable. Tomando estos valores para los parámetros de cuantificación resulta que en una compresión que no sacrifique la calidad de la imagen, un valor de *bpp* podría ser: $bpp = \frac{4}{16} = 0.25$.

Esta primer aproximación, si bien es útil para tener una idea de cómo VQ por sí solo es una herramienta de compresión interesante, no es representativa de las verdaderas posibilidades en tanto no tiene en cuenta la codificación de entropía a la que se debería someter a los índices del *codebook*, que son los resultados de salida del codificador.

Para tener esto en cuenta se debe utilizar algún esquema de codificación de entropía. Para esto se opta por guardar los índices de la cuantificación en un archivo y actuar sobre ellos con un compresor de archivos estándar(por ejemplo *WinZip*).

Como fuente de comparación se decide tomar el estándar JPEG y su curva de PSNR en función de los *bpp*. La figura siguiente muestran una comparación de los resultados obtenidos por nuestro codificador y por un codificador JPEG.

- Comparación de resultados(2×2 y 4×4)

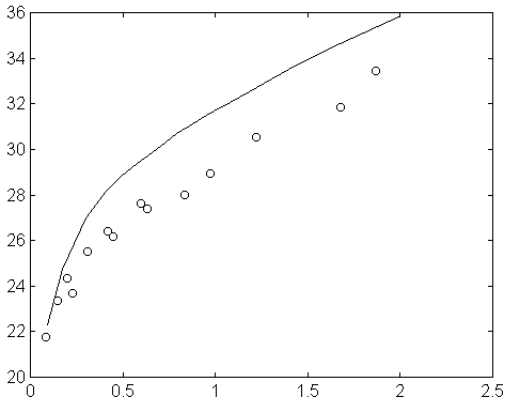

Figura A.1: PSNR vs *bpp*, sin media removida 2×2 y 4×4, con $\circ$ los resultados obtenidos, con línea llena los resultados de JPEG.

A.2 Compresión de imágenes fijas con media removida

Veamos cual es la capacidad de compresión de un cuantificador vectorial con media removida.

En este esquema, antes de codificar la imagen se debe preprocesar cada uno de los vectores que la componen de la siguiente manera: se calcula la media del vector (la que se codifica escalarmente) y luego se hace la resta entre éste y un vector uniforme cuyos componentes sean iguales a la media. Es decir, obtenemos un vector de media cero, pero que mantiene la forma del vector original (al decir mantiene la forma, se quiere decir que mantiene todas las transiciones presentes en el vector original).

Si se tiene un sistema con las mismas características del sistema de la sección anterior (imagen en tonos de gris de $x \times y$ píxeles, un *codebook* de N vectores con $N = 2^{n_1}$ con vectores cuadrados de lado l), y se dedican además n_0 bits para codificar la media de cada bloque, la razón de bits por pixel resulta:

$$bpp = \frac{\text{bits empleados}}{\text{píxeles codificados}} = \frac{\text{imagen codificada + } \textit{codebook}}{\text{píxeles codificados}} =$$

$$\frac{\frac{x}{l}\frac{y}{l}(n_0 + n_1) + 8Nl^2}{xy}$$

Si se desprecian los bits empleados para representar el *codebook* frente a los empleados para codificar la imagen se obtiene: $bpp = (n_0 + n_1)/l^2$.

Los resultados obtenidos se muestran en las Figuras A.2 y A.3 comparados con los que obtiene el estándar JPEG.

Es claro que esta técnica sólo superará a la anterior si la extracción de la media permite achicar el *codebook* tanto que resulte $n_0 + n_1 < n$. En el caso en que se de la igualdad, ambas técnicas serán idénticas en cuanto a los *bpp* obtenidos; en este caso se preferirá la primera de ellas debido a su mayor simplicidad y rapidez.

A.3 Conclusiones

Estos resultados nos indican que si bien la cuantificación vectorial es una herramienta poderosa para la compresión de imágenes, no se logran a priori mejores resultados que los que se obtienen con el estándar JPEG. Para poder

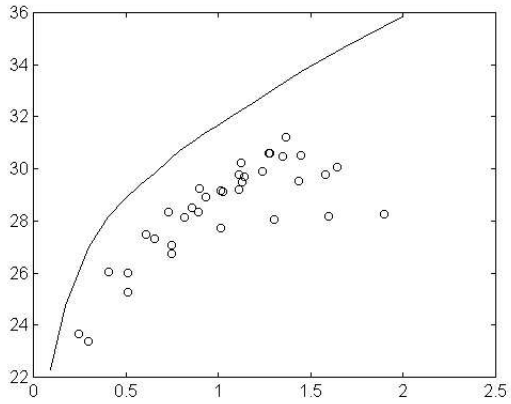

Figura A.2: PSNR vs *bpp*: comparación de resultadps con media removida 2×2 (con $\circ$) con JPEG (con línea llena).

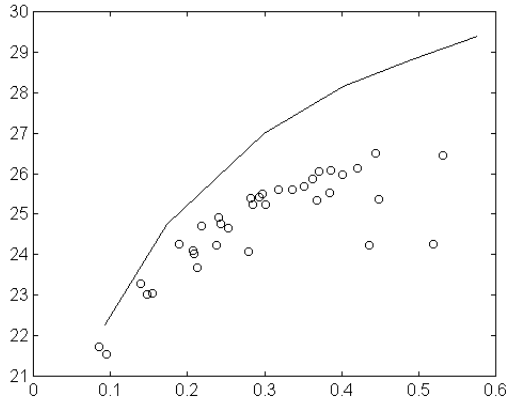

Figura A.3: PSNR vs *bpp*: comparación de resultadps con media removida 4×4 (con $\circ$) con JPEG (con línea llena).

seguir avanzando resulta necesario desarrollar un esquema de compresión que combine la cuantificación vectorial con otras técnicas para seguir disminuyendo los bits por pixel necesarios.

Parte VI

Bibliografía

Bibliografía

[1] A. Gersho; R. M. Gray. *Vector quantization and signal compresion.* 4th ed. Kluwer, 1995. ISBN 0-7923-9181-0.

[2] J. L. Mitchel; W. Pennebaker; C. E. Fogg; D. J. LeGall. *MPEG video compression standard.* International Thompson Publishing, 1996. ISBN 0-412-08771-5.

[3] A. Bruce Carlson. *Communication systems.* 3rd. ed. McGraw-Hill, 1996. ISBN 0-07-009960-X.

[4] K. Panusopone; K. R. Rao. *VQ based on a main feature classification in images.* Journal of Visual Communication and Image Representation, vol.11, pp. 1-16, 2000.

[5] T.D.Tran; R.Safranek. *A locally adaptive perceptual masking treshold model for image coding.* AT&T Bell Laboratories, Signal Processing Research Department, University of Wisconsin.

[6] A. Ribeiro; A. Gonzalez; A. Pardo. *ActivZ.* Facultad de Ingeniería. Documentación de proyecto de graduación. Montevideo, 1999.

[7] Jong-Nam Kim; Tae-Sun Choi. *Real-time video coding.* IEEE Transactions on Consumer Electronics, vol.45, nro.2, pp.417-425, 1999.

[8] M. Gallant; G. Côtè; F. Kossentini. *An efficient computation-constrained block-based motion estimation algorithm for low bit rate video coding.* IEEE Transactions on Image Processing, vol.8, nro.12, pp.1816-1823, 1999.

[9] Jong-Nam Kim; Tae-Sun Choi. *Adaptive matching scan algorithm based on gradient magnitude for fast full search in motion estimation.* IEEE Transactions on Consumer Electronics, vol.45, nro.3, pp. 762-772, 1999.

[10] Charles Petzold. *Programming Windows 95.* Microsoft Press, 1996. ISBN: 1199000558.

[11] Alfred V. Aho; John E. Hopcroft; Jeffrey D. Ullman. *Estructuras de Datos y Algoritmos.* Addison-Wesley, 1983, ISBN: 0201000237.

[12] Bjarne Stroustrup. *C++ Programming Language.* 3ra. ed. Addison-Wesley, 1997, ISBN:0201889544

.

Versión corregida por los integrantes de la mesa examinadora.

Las imágenes aparecen con efecto de *ringing* debido a que el conversor a pdf codifica las imágenes en formato JPEG.