División de Estudios de Posgrado
Facultad de Ingeniería

Compresor de Señal Video para Transmisión por Fibra Optica

Juventino Delfino Rosas Espejel

TESIS

PRESENTADA A LA DIVISION DE ESTUDIOS DE
POSGRADO DE LA
FACULTAD DE INGENIERIA
DE LA
Universidad Nacional Autonoma de Mexico

COMO REQUISITO PARA OBTENER
EL GRADO DE
MAESTRO EN INGENIERIA ELECTRICA

CIUDAD UNIVERSITARIA
Febrero de 1991.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE

ABREVIATURAS USADAS EN ESTE TRABAJO.

ADPCM	Modulación por pulsos codificados diferencial adaptable.
CAD-CAE	Diseño asistido por computadora-ingeniería asistida por computadora.
DCT	Transformada coseno discreta.
DPCM	Modulación por pulsos codificados diferencial.
DVI	Video digital interactivo.
FFT	Transformada rápida de Fourier.
HVS	Función del sistema visual humano.
NTSC	Comite de sistemas de televisión nacional.
PCM	Modulación por pulsos codificados.
RDSI	Redes digitales de servicios integrados.
Run-length	Codificación por corridas de ceros.
SNR	Relación señal a ruido.

INDICE FIGURAS

INDICE TABLAS

CAPITULO I INTRODUCCION

1.1.-SISTEMAS DE COMUNICACION.

Mucho del progreso de la sociedad actual ha ocurrido desde el advenimiento de los sistemas de comunicación electrónicos. La introducción del radio, teléfono y televisión posibilitaron la comunicación masiva de grandes volumenes de información a lugares distantes de manera casi instantánea. Las técnicas de comunicación electrónica han tenido un gran impacto en el comercio, relaciones internacionales, ciencia y entretenimiento del mundo. Sin los sistemas de comunicación actuales muchas de las actividades del quehacer humano serían engorrosas y algunas imposibles.

En forma básica un sistema de comunicación consta de tres bloques principales: transmisor, receptor y canal de comunicación. Este modelo básico se muestra en la figura 1.1.

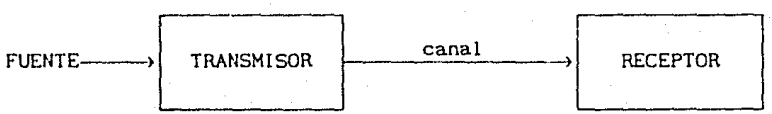


Figura 1.1 Modelo básico de comunicación

Generalmente, el mensaje que produce la fuente no es de naturaleza eléctrica, por lo tanto, es necesario utilizar un transductor a la entrada para convertir la señal de la fuente en

una señal eléctrica variable en el tiempo, equivalente a la variable física observada. Utilizando otro transductor, el mensaje original se reconstruye en el extremo de recepción.

El propósito del transmisor es conformar la señal mensaje en una forma adecuada para su transmisión por el canal. Esta conformación se realiza por medio de un proceso conocido como modulación, el cuál comprende la variación de algún parámetro de la señal portadora (por ejemplo amplitud, frecuencia o fase) en concordancia con la señal del mensaje. [4]

El canal de comunicación puede ser una línea de transmisión, una fibra óptica o el espacio libre. Al propagarse a tráves del canal , la señal se distorsiona por las no linealidades e imperfecciones espectrales del canal y adicionalmente en el curso de la transmisión, la señal se afecta por ruido e interferencia, distorsionando aún más la señal. En los sistemas de comunicación, el transmisor y receptor se diseñan para minimizar estos efectos.

El proposito del receptor es reconstruir la señal original transmitida, después de su propagación a tráves del canal, este proceso se conoce como demodulación, proceso inverso a la modulación del transmisor. Sin embargo, debido a los problemas mencionados anteriormente, el receptor no puede reconstruir la señal original con exactitud.

En general, en el diseño de los sistemas de comunicación existe un compromiso entre dos parámetros, potencia de transmisión y ancho de banda, y se emplean en forma tan eficiente como sea posible; para algunos casos podría considerarse un recurso más importante que el otro.

1.2.-COMUNICACIONES DIGITALES.

En este tipo de sistemas la información se transmite en patrones de señales binarias, independientemente del tipo de

modulación utilizada AM, FM, QAM, PSK u otra. El esquema básico de este sistema se muestra en la figura 1.2. Para este sistema es necesario sumar convertidores analógico/digital y digital/analógico si la fuente y el usuario necesitan comunicarse a través de señales de tipo analógico.

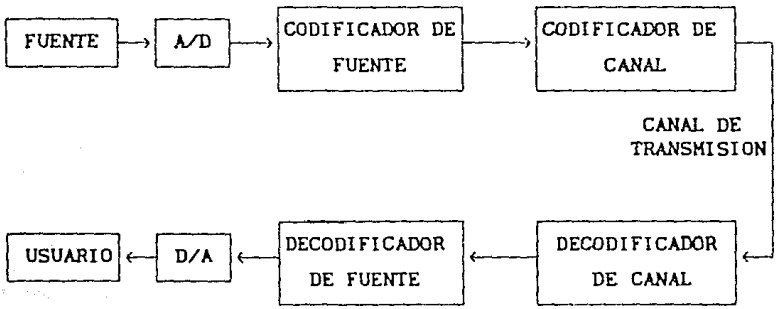


Figura 1.2 Esquema de comunicación digital básico.

Este esquema es apropiado para la transmisión de señales codificadas en formato digital a grandes distancias con varios repetidores .[3]

Para este esquema, el bloque fuente representa cualquier persona o aparato generador de información con la necesidad de transmitir esta a gran distancia, hacia otro(s) usuario(s). Si la información generada es analógica, es necesario realizar una digitalización previa a la transmisión.

El codificador de fuente es la sección encargada de aprovechar algunas de las características de la señal (como la entropía), para eliminar redundancias entre valores adyacentes, reduciendo así, el número de bits necesarios para representar la información de la señal, dando en consecuencia una reducción en el ancho de banda necesario en el canal para la transmisión.

Este bloque no siempre existe en un sistema de comunicación

digital, cuando el ancho de banda del canal o de manera equivalente la capacidad del canal, lo permite, se prefiere realizar la transmisión con el método de modulación por pulsos codificados PCM, el cuál transmite señales codificadas digitalmente sin reducción de redundancias.

El codificador de canal se encarga de preparar la señal de manera que en el receptor, el decodificador de canal pueda detectar y corregir errores generados en la transmisión. Esto se realiza utilizando diferentes técnicas que adicionan bits redundantes al flujo original de bits, con el fin de que el receptor tenga una detección libre de errores, bajo la presencia de ruido de canal.

En el receptor se realiza el proceso inverso al transmisor, en cada uno de sus bloques, presentandose al usuario la información transmitida.

1.2.1.-TEOREMA DE MUESTREO.

Sea x(t) una señal, la cuál esta limitada en banda, de tal manera que el componente espectral de más alta frecuencia es f_x. Si la señal x(t) se muestrea periodicamente en tiempos regulares separados por intervalos nTs, donde $Ts \leq 1/2\, f_x$, entonces las muestras x(nTs) permiten que la señal pueda ser reconstruida sin ninguna distorsión. El tiempo Ts se conoce como tiempo de muestreo, y la minima velocidad de muestreo ($2f_x$) como velocidad de Nyquist.

Demostración: la señal en banda base x(t) se muestrea con el tren periodico de pulsos S(t) de amplitud unitaria, duración τ y periodo Ts, figura 1.3 a y b.
Expresando la señal S(t) como una serie de Fourier tenemos:

$$S(t) = Co + \sum_{n=1}^{\alpha} 2Cn \cos(n\omega t)$$

donde:

$$Co = \tau/Ts \ , \ Cn = fs\ \tau\ sen(nfs\tau) \ , \ \omega s = 2\pi fs$$

Multiplicando x(t) y S(t):

$$xs(t) = x(t)\ S(t)$$

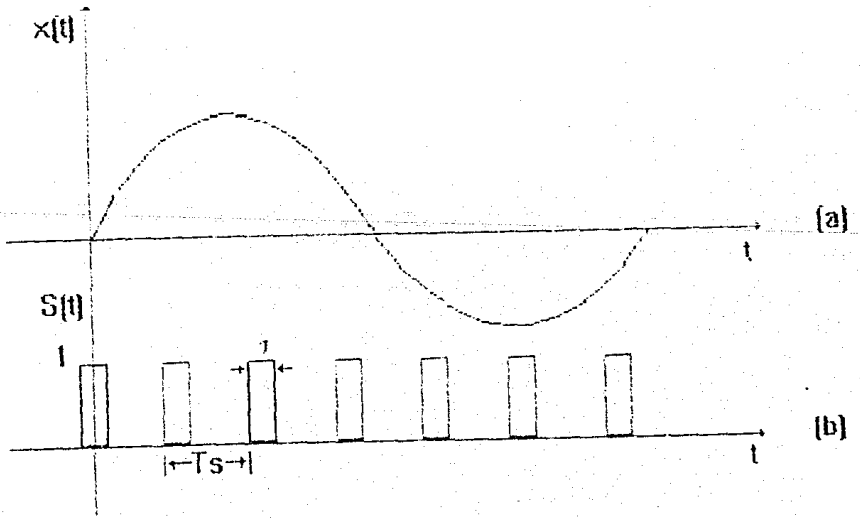
$$xs(t) = Co\ x(t) + 2C_1\ x(t)\ \cos(\omega st) + C_2\ x(t)\ \cos(2\omega st) + \ldots$$

cuya transformada de Fourier es:

$$Xs(f) = Co\ X(f) + C_1[X(f-fs) + X(f+fs)] + C_2[X(f-2fs) + X(f+2fs).$$

$$= CoX(f) + \sum_{n=-\alpha}^{+\alpha} Cn\ X(f-nfs)$$

A partir de esta ecuación obtenemos el expectro de xs(t). En la figura 1.4 se muestra el espectro de la señal muestreada cuando la entrada x(t) esta limitada en banda por fx hertz.



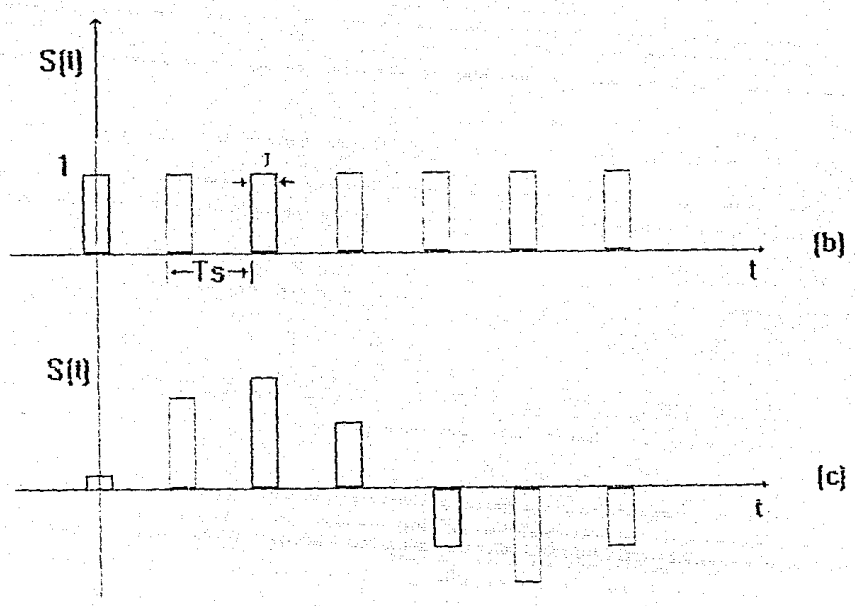


Figura 1.3 Muestreo (a) Señal x(t). (b) Función de muestreo S(t). (c) Señal muestreada.

Se nota que si fs > 2fx, la operación de muestreo deja el mensaje intacto, y lo repite en el dominio de la frecuencia con un periódo fs. De esta forma, el mensaje x(t) no se altera y es posible reconstruirlo a partir de $x_s(t)$ por medio de un filtro pasa bajas.

Cuando la velocidad de muestreo fs < 2fx, las bandas laterales de la señal se traslapan y x(t) no se puede recuperar sin distorsión a partir de $x_s(t)$. Esta distorsión se conoce como traslape. De esta manera la frecuencia de muestreo debe de satisfacer: $fs \geq 2fx$, o $Ts \leq 2Tx$. [1,4]

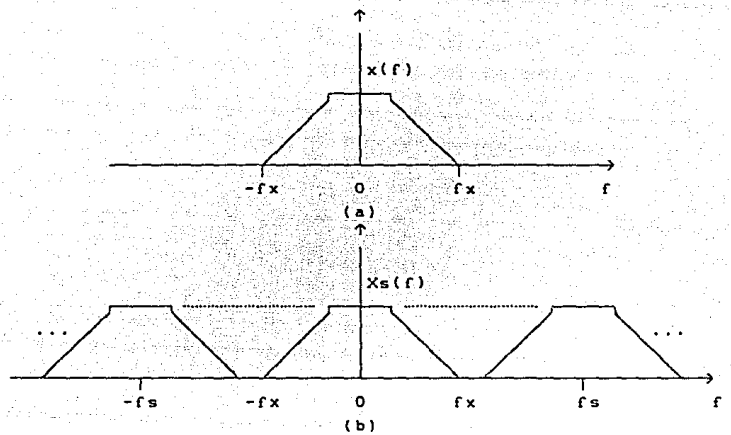


Figura 1.4 Espectro de señales idealmente muestreadas.
(a) Mensaje. (b) Mensaje idealmente muestreado.

1.3.-TELEVISION.

1.3.1.- FUNDAMENTOS

Los sistemas de televisión, son sistemas de transmisión de imágenes, con un mensaje de patrón bidimensional, y por lo tanto, función de dos variables.[6]

Los sistemas de televisivos están basados en el traslado de una escena, a una serie de pequeños puntos llamados pixeles, los cuales se obtienen al barrer sucesivamente una imagen con un haz de electrones con intensidad modulada por el brillo relativo de la imagen. Como resultado, se obtiene una variación de voltaje proporcional al brillo instantáneo del pixel, siendo este voltaje función de la variable tiempo.

Para que la transmisión de imágenes se realice de manera satisfactoria, un sistema de televisión requiere adquirir y reproducir fielmente ciertas características de los objetos observados como son su forma, brillo relativo, movimiento, sonido

y color.

En el mundo actual existen varias normas para la transmisión de televisión, cada una con sus propias características e incompatibilidad con los demás sistemas, siendo los principales el norteamericano y el europeo.[1]

El sistema norteamericano emplea para las transmisiones en blanco y negro la norma de la FCC (Federal Communications Commission) y para color la norma NTSC (National Television Standars Committe), adicionalmente, esta norma se utiliza en Latino América (con excepción de Argentina, y Brasil), y Japón.

En México el sistema utilizado es el norteamericano, el cuál también es el más poderoso en el mundo, debido al número de receptores existentes, calculado en 230 millones de receptores en 1988.

Por su parte, el sistema europeo utiliza las normas CCIR (Comité Consultivo Internacional de Radio) y PAL (Phase Alternation Line) para blanco y negro, y color respectivamente.

Esta norma se emplea en Europa exceptuando Francia y parte de Bélgica, que emplean el sistema SECAM (Sequentiel Couleur avec Memoire or sequential color with Memory) para color.

Por lo que corresponde al bloque socialista, estos paises utilizan un sistema casi identico al CCIR para blanco y negro, y en color, la norma SECAM.

Para el futuro, debido a su alta eficiencia, se espera que los sistemas de comunicación televisivos sean digitales. Para una transmisión de video, las señales digitales requieren tener una menor relación señal a ruido y presentan mejor inmunidad al ruido e interferencia, que las señales analógicas. Sin embargo, debido a que estos sistemas actualmente requieren tasas de transmisión muy altas, 100 ~ 135 Mbits/seg, hacer uso común de estos sistemas no parece cercano.[2]

1.3.2.-SEÑAL VIDEO.

La señal video se obtiene a partir de una imagen óptica formada en el fotocátodo del tubo de una cámara. La conversión hacia una señal x(t), y viceversa, requiere un proceso discontinuo de mapeo, realizado por medio de un haz de electrones, dando como resultado una variación de voltaje proporcional al brillo instantáneo del pixel, esto se puede observar en la figura 1.5.

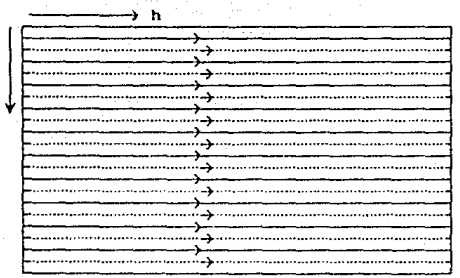


Figura 1.5 Trama de exploración.

Como lo muestra la figura, la imagen es barrida por el haz de electrones , con un movimiento que inicia en la esquina superior izquierda moviéndose hacia la derecha y ligeramente hacia abajo, con velocidad horizontal y vertical constantes (ver tabla 1.1).

Cuando se alcanza el extremo derecho, el haz regresa rápidamente a una posición inferior a la inicial y de nuevo se repite el movimiento hacia la derecha y ligeramente hacia abajo. Este proceso se repite hasta alcanzar la parte inferior de la imagen; en este punto, el haz regresa a la parte superior de la imagen (que puede ser el mismo punto anterior u otro diferente), reiniciándose el barrido.

Una serie de imágenes cambiando a un ritmo lento, por ejemplo 10/seg, producen en el observador la ilusión de movimiento, pero acompañado de un parpadeo pronunciado; aumentando la velocidad de despliegue de las imágenes, el parpadeo disminuye hasta que, a un ritmo de 60/seg, se tiene movimiento sin un parpadeo perceptible.

En televisión, para eliminar el parpadeo, se utiliza la técnica conocida como entrelazado, en la cuál se subdivide la imagen en dos partes, cada una llamada campo. El campo par, compuesto de todas las líneas pares de la pantalla y el campo impar por todas las líneas impares.

El proceso de barrido se realiza en dos partes, primero sobre el campo de líneas pares de la imagen, y después sobre el campo de líneas impares. De esta manera el ojo humano no percibe las líneas individuales. Esta técnica despliega 30 imágenes por segundo con 60 campos/seg, en un ancho de banda correspondiente a las 30 imágenes/seg. En la norma NTSC se tienen 30 cuadros y 60 campos por segundo, dando una relación de entrelazado de 2:1.[1]

En el receptor, la imagen se reconstruye en el cinescopio, por medio de un haz de electrones que se mueve de manera sincronizada con el haz en el tubo de la cámara. La pantalla brilla con una intensidad que depende del poder del haz de electrones, manteniéndose este brillo durante un tiempo determinado. Este efecto, combinado con la persistencia de visión del ojo humano, dá al observador la impresión de ver en la pantalla una imagen completa. Si además se muestra rápidamente una sucesión de imágenes, alterada cada una de ellas ligeramente respecto a la anterior, aparece la ilusión de movimiento.[1]

Lo visto anteriormente corresponde a una transmisión televisiva en blanco y negro, para la televisión en color, se

requiere la transmisión de una mayor cantidad de información. Además del brillo o luminancia, debe incluirse información de tinte y saturación, que al combinarse forman la señal de crominancia o croma. Así, para cada pixel, la señal de reconstrucción debe contener información de luminancia y crominancia.

Las tres señales de color primario (A)zul, (V)erde y (R)oJo, se pueden representar por medio de tres señales cualquiera que sean combinaciones lineales independientes. Con una elección apropiada de coeficientes, se puede obtener una combinación que sea igual a la señal de luminancia, esto es:[1]

$$Y = 0.30R + 0.59V + 0.11A$$

Con las señales de crominancia:

$$I = 0.60R - 0.28V - 0.32A$$
$$Q = 0.21R - 0.52V + 0.31B$$

A partir de estas expresiones se define el vector de color:

$$C = I + JQ$$

cuya magnitud es la intensidad de color o saturación y cuyo ángulo es el tinte.

En el tubo de la cámara transmisora se producen señales separadas para cada uno de estos colores. En el cinescopio del receptor, esas señales se aplican a tres diferentes cañones de electrones, cuya pantalla consiste de puntos adyacentes de los colores básicos, los cuales brillan en su color correspondiente

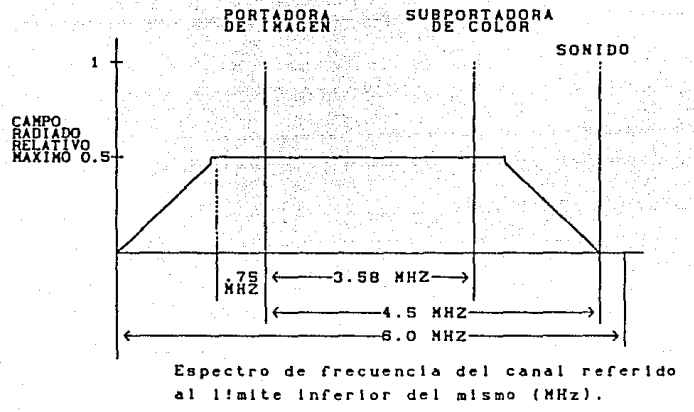
cuando el haz de electrones incide sobre ellos, recreandose el color de la imagen captada por la cámara del transmisor.[1,4]

1.3.3.- ESPECTRO DE LA SEÑAL VIDEO.

En la transmisión de la señal de video de televisión se utiliza la modulación en amplitud con banda lateral vestigial, el sonido se transmite en modulación de FM de banda amplia con frecuencia cercana a la señal de imagen, de modo que al momento de ser radiadas, ambas señales se combinan para poder utilizar la misma antena.[4]

Al llegar al aparato receptor, se utiliza un mismo mecanismo de sintonía y amplificación, separándose posteriormente el audio del video, para ir a sus respectivas secciones de tratamiento. El espectro de esta señal se muestra en la figura 1.6.

El ancho de banda del canal para la norma NTSC es de 6 MHz y acomoda la banda lateral superior, el vestigio de la banda lateral inferior, el espectro de la señal de sonido y la subportadora de color con sus dos bandas laterales. En cada canal de televisión la frecuencia portadora de imagen está a 1.25 MHz sobre el extremo inferior del canal y la subportadora de color 3.58 MHz más arriba.



Espectro de frecuencia del canal referido al límite inferior del mismo (MHz).

Figura 1.6 Espectro de la señal video NTSC.

Se enlistan algunas de las características de la norma televisiva NTSC.

Ancho de banda del canal de transmisión	6 MHz
Ancho de banda de la señal video	4.2 MHz
Desviación de audio en FM	25 KHz
Localización de la portadora de imagen	1.25 MHz desde el extremo inferior.
Localización de la portadora de color	3.58 MHz desde la portadora de imagen.
Relación de aspecto	4/3
Líneas de barrido	525 por cuadro
Entrelazado	2 : 1
Frecuencia de barrido vertical	15,750 Hz
Frecuencia de barrido horizontal	
Tiempo de línea	63.5 μs
Tiempo de retrazo horizontal	10.0 μs

Tabla 1.1 Norma NTSC.

1.3.4.-LA TELEVISION COMO FUENTE DE INFORMACION.

Bajo este punto de vista, la televisión es una fuente de varias clases de información:[5]

-Información bidimensional de cada campo
-Información del movimiento existente entre campos
-Información de luminancia (intensidad o contraste)
-Información de color

Podemos considerar una imagen digitalizada como una matriz, en donde el valor de cada uno de los elementos representa la intensidad instantánea de la radiación que incide en el punto. Por otro lado, los tonos de gris de las imágenes digitalizadas, al ser bidimensionales, presentan redundancias entre los valores de elementos circunvecinos, o en otras palabras, el valor en intensidad de un elemento de imagen puede ser cálculado a partir de los valores de los elementos vecinos.

Como se vió en este capítulo, la norma NTSC utiliza 525 líneas por imagen, a una tasa de 30 imágenes por segundo.

Convirtiendo estos datos al sistema digital más cercano (2^9=512), la norma de la NTSC se apróxima a una imagen digital con una resolución de 512 x 512 pixeles, códificada a 8 bits por pixel y 30 imágenes por segundo , dando como resultado[5] una tasa de transmisión de:

$$512 \times 512 \times 8 \times 30 = 63 \text{ Mbits/seg}$$

únicamente considerando la información de luminancia, en una transmisión de televisión en blanco y negro. Para color, las señales de luminancia y crominancia, se codifican de manera separada.

En el caso de tener que enviar una sola de estas imágenes

sobre una línea telefónica, usando un modem de 2400 bauds, se requiere un tiempo de poco menos de 15 minutos. A partir de este resultado, se ve la necesidad de reducir o comprimir la cantidad de datos para representar la imagen, con el fin de acortar el tiempo necesario para efectuar la transmisión.

Básicamente, la compresión, es la representación de la imagen mediante un conjunto de datos, menor al original, que nos proporciona información sobre alguna propiedad única de la imagen, única en el sentido de que esa propiedad no se puede predecir a partir de los otros datos, y que nos permite posteriormente ,a partir de este conjunto de datos comprimidos, reconstruir la imagen, aunque con algunas diferencias (distorsiones) con respecto a la imagen original.

En general, las imágenes presentan tres tipos de redundancias:[3]

1.-Redundancia espacial.- debido a la correlación entre los valores de pixeles circunvecinos.
2.-Redundancia espectral.-es la correlación entre los diferentes planos de color o bandas espectrales, como es el caso de imágenes de color o las fotografías aereas.
3.-Redundancia temporal.-corresponde a la correlación entre campos consecutivos en una sucesión de imágenes.

En 1956, trabajando únicamente con la información de luminancia códificada a 6 bits/pixel, W.F. Schreiber midió la entropía o información promedio de las imágenes de televisión, obteniendo los siguientes resultados:[3]

Entropía de orden cero, $H(y) = 4.4$ bits/pixel
Entropía de primer orden, $H(y/x) = 1.9$ bits/pixel
Entropía de segundo orden, $H(y,x_1,x_2) = 1.5$ bits/pixel

Donde de acuerdo con los resultados obtenidos, la tasa máxima de compresión posible es 6:1.5 o 4:1. Sin embargo, la estimación hecha por Schreiber no toma en cuenta que el video realmente es una sequencia de imágenes donde existe una considerable redundancia entre las imágenes sucesivas, permitiendo obtener mayores tasas de compresión a las calculadas por él.[7]

Las técnicas de compresión utilizadas en televisión pueden utilizar códificación entre-campos, que utiliza dos o más campos para realizar la compresión, o la codificación intra-campos que trabaja la compresión sobre un solo campo a la vez.

Estas técnicas producen cierta cantidad de distorsión, pero como el receptor final es el sistema visual humano, y no una computadora, se toleran algunos tipos de distorsiones.[2] Por esto, independientemente de las pruebas objetivas que se puedan aplicar a las imágenes resultantes de las técnicas de compresión, frecuentemente se utiliza a un grupo de seres humanos para realizar evaluaciones subjetivas de la calidad de las imágenes reconstruidas.

Para la evaluación subjetiva de imágenes televisivas el CCIR, ha establecido algunas condiciones, con respecto a como deben de ser los observadores, las pruebas que, y como, se deben de realizar, todo esto en su recomendación número 500-3.[2]

Esta recomendación fue fórmulada para transmisiones televisivas analógicas, pero los mismos métodos son aplicables a la evaluación de las imágenes digitales. Los puntos principales de esta recomendación, se muestran en la tabla I.

En la tabla II, la escala de calidad de la imagen consta de cinco grados de evaluación, junto a estos valores se muestra la escala de percepción de daños, o en otras palabras, las

diferencias de la imagen reconstruida, con respecto a la imagen original.

En la tabla III se muestra una escala de comparación de siete grados de calidad.

TABLA II

PUNTOS PRINCIPALES DE LA RECOMENDACION CCIR 500-3

RUBRICA	CONTENIDO
Observadores	Al menos diez observadores, no expertos, deben de estar en la prueba de evaluación subjetiva.
Escala de graduación	Se debe de utilizar la escala mostrada en la tabla II.
Imágenes	Mínimo cinco imágenes o secuencias de imágenes.
Ambiente	En los resultados se deben de especificar la razón de la distancia de vista hacia la imagen contra la altura de la misma, valor de la luminancia pico en la pantalla y las condiciones de iluminación en el cuarto de prueba.
Presentación	La presentación de las imágenes deberá de ser de manera aleatoria.

TABLA III

ESCALA DE CALIDAD-DAÑO DE LAS IMAGENES

CALIDAD	DAÑO
5 Exelente	5 Imperceptible
4 Buena	4 Perceptible, sin causar molestias
3 Regular	3 Causa molestias ligeras
2 Pobre	2 Molesta
1 Mala	1 Muy molesta

TABLA IV

ESCALA DE COMPARACION

+3	Muy bien
+2	Bien
+1	Ligeramente bien
0	Lo mismo
-1	Ligeramente mal
-2	Mal
-3	Muy mal

1.4.-REFERENCIAS.

1.-Freeman, R.L., Reference Manual for Telecommunications Engineering, John Wiley, New York, USA, 1985.

2.- Hitomi Murakami, Quality of band compressed TV Services, IEEE Communications Magazine, October 1988.

3.-Hisashi Kaneko and Tatsuo Ishiguro, Digital Television Transmission Using Bandwidth Compression Techniques, IEEE Communications Magazine, July 1980.

4.-Kennedy, G., Electronic Communication System, McGraw-Hill, New York, USA, 1985.

5.-Majid Rabbani, IISG-research Laboratories, Imagin Science Lab, Eastman Kodak Company, Curso: Digital Image Compression, Abril 1990, San Diego, CA.

6.-Santiago, S.J., Digitalizador de Señales de Televisión, Tesis Profesional, Facultad Ingeniería, UNAM, 1990.

7.-Schreiber, W.F., The measurement of Third Order Probability Distributions of Television Signals, IRE Transactions on Information Theory, vol. IT-2, September 1956.

CAPITULO II TEORIA Y TECNICAS DE COMPRESION DE SEÑALES DE BANDA ANCHA.

2.1.-COMPRESION DE DATOS.

La idea de comprimir datos no es nueva. Por ejemplo, las personas tendemos a utilizar, cuando es posible, abreviaciones escritas u orales para comunicar una idea. En el caso de los viejos sistemas de comunicaciones, el telegráfo, es un caso representativo de un sistema que usa técnicas de compresión de datos, optimizando estadisticamente la longitud del código usado para cada una de las letras y números que se tienen que transmitir.

De esta manera, a la letra que tiene uso más frecuente se le asigna el código de longitud más corta, y por lo contrario a la letra de menor uso se le asigna el código de longitud más larga.

Así, al transmitir muchos mensajes, el tiempo promedio que se emplea el canal telegráfico, para enviar cada mensaje, es el mínimo posible. Y como la compañía telegráfica cobra por mensaje transmitido, esta técnica de compresión le permite enviar más mensajes en un espacio de tiempo determinado, dando como consecuencia la obtención de mayores ingresos económicos.

En el tiempo presente, las comunicaciones digitales están reemplazando casi todas las formas de comunicaciones analógicas, debido a las ventajas que tiene en términos de la flexibilidad para el procesamiento, fácil acceso aleatorio para almacenamiento, alta relación señal a ruido, la posibilidad de menores errores en la transmisión, etc.

Sin embargo, el proceso de hacer la conversión de una señal analógica a una señal digital, resulta en el incremento del ancho de banda requerido para transmisión. Para solucionar esta desventaja de las señales digitales, las técnicas de compresión

buscan reducir el ancho de banda, ó en forma equivalente, la cantidad de datos necesario para representar la señal digital.

Aunque no existe una clasificación única de las técnicas de compresión, estas se pueden enmarcar en dos tipos: reversibles e irreversibles. Estos dos tipos pueden ser:

Reversibles	Irreversibles
Codificación sin pérdidas	Codificación reducción de fidelidad
Reducción de redundancias	Reducción de entropia

Para este trabajo, se utilizarán los términos de reducción de entropia y redundancia. El término entropia, se define como el promedio de información del mensaje o grupo de datos.[6] Así, la operación de reducción de entropia produce una disminución de la cantidad de información en los datos, la información perdida es irrecuperable y el proceso es irreversible.

Por otro lado, en la operación de reducción de redundancia se remueven, o al menos se reducen, las redundancias de modo que estas puedan subsecuentemente ser reinsertadas en los datos, de manera que los valores reconstruidos sean idénticos a los valores originales.

2.2.-FUENTES DE ENTROPIA DISCRETAS.

La información se define en términos de la medida de incertidumbre, por esto, el mensaje con la menor probabilidad de ocurrencia contiene la mayor cantidad de información, y en contraparte el mensaje más probable tiene la menor cantidad de información.[6]

Matemáticamente, se dice que el contenido de información de un mensaje o evento a ser transmitido es una función monotónica decreciente de su probabilidad de ocurrencia. Esta función puede ser modelada a tráves del logaritmo negativo de la probabilidad de ocurrencia del mensaje, y con la particularidad de que cuando la probabilidad es igual a uno, la función se iguala a cero.[4,6]

El contenido de información del i_{simo} evento de una fuente

discreta sin memoria (esto es, la fuente no presenta dependencia entre dos eventos sucesivos) esta dada por:

$$I_i = -\log_2 P_i \quad [\text{ bits }] \tag{2.1}$$

donde P_i es la probabilidad del i simo evento, $0 \leq P_i \leq 1$.

El logaritmo puede ser de cualquier base, pero se prefiere utilizarlo en base dos por la facilidad para relacionarlo con los esquemas de codificación binaria. Esta función se muestra en la figura 2.1.

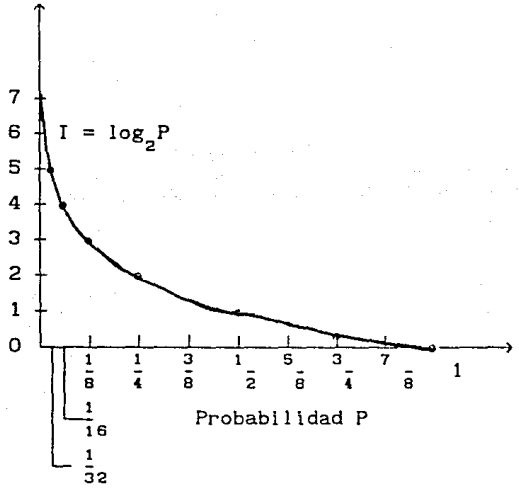


Figura 2.1 Función de información.

Si se promedia el contenido de información de todos los posibles eventos de una fuente, se obtiene la entropía de la fuente. Para el caso de una fuente discreta sin memoria la entropía se define como: [6]

$$H = -\sum_{i=1}^{M} P_i \log(P_i) \tag{2.2}$$

donde M es el número total de valores discretos de la fuente.

Para una fuente discreta de tipo general, esto es, una fuente

donde los eventos sucesivos presentan cierta dependencia, , se considera un vector aleatorio X compuesto de n eventos discretos sucesivos generados por una fuente estacionariamente aleatoria. Una fuente estacionaria se define como sigue: sea t un vector de tiempo discreto dado por $t = (t_1, t_2, \ldots, t_n)$, y para todos los k enteros $(t + k) = (t_1 + k, t_2 + k, \ldots, t_n + k)$. Entonces si $P_{t+k}(x) = P_t(x)$ la fuente es estacionaria.[6]

Sea P(x) la probabilidad de que el vector X tome el valor x (una secuencia especifica de n eventos), entonces la entropia total se define como:

$$H(X) = -\sum_{\substack{\text{todas} \\ \text{las } x}} P(x) \log_2 P(x) \qquad (2.3)$$

y la entropia por evento es el limite de (2.3)

$$H = \lim_{n \to \alpha} \{ n^{-1} H(x) \} \qquad (2.4)$$

donde H se conoce como tasa de entropia debido a que sus unidades de medida son bits/muestra o bits/simbolo.

2.3.-TASA DE DISTORSION

El concepto de tasa de distorsión es completamente diferente al concepto de codificación de canal manejado hasta ahora (ver capitulo 1.2). Mientras que en la codificación de canal se trata de tener una comunicación libre de errores, para la función de tasa de distorsión se pretende encontrar la tasa minima de la fuente , que mantenga el promedio de distorsión de la transmisión menor o igual a una distorsión especificada. Esto es:[2,6]

$$R(D) \leq C \qquad (2.5)$$

donde:

C = Capacidad del canal de transmisión o tasa de codificación
D = Distorsión del canal
R(D) = Función de la tasa de distorsión

además R(D) y C deberán utilizar unidades equivalentes: bits/evento, bits/muestra o bits/segundo.
Matemáticamente, la función tasa de distorsión se define como el promedio mínimo de información mutua entre la fuente y el receptor sujeto a un promedio fijo de distorsión D. Donde el promedio de distorsión es función de la medida de distorsión d(x,y), la cuál mide el costo de no reproducir exactamente el valor original **x** en la muestra de salida y. En forma matemática:

$$R(D) = \underset{\text{para toda } P(y/x)}{\text{mínima}}\ I(X;Y) = \min \sum_X \sum_Y P(X)P(Y/X)\ \frac{\log P(Y/X)}{Q(Y)} \qquad (2.6)$$

$$Q(Y) = \sum_X P(X)P(Y/X)$$

con la condición:

$$\text{Distorsión promedio} = \sum_X \sum_Y P(X)P(Y/X)D(X,Y) \leq D \qquad (2.7)$$

2.4.-CUANTIZACION

Como se demostró las señales analógicas, una vez digitalizadas a frecuencia $fs \geq 2fx$, pueden recuperarse en base a sus muestras. Pero el proceso de muestreo, por si solo, proporciona señales continuas en amplitud sobre cada Ts.
Cuando estas muestras son transmitidas sobre un canal ruidoso, el receptor no puede encontrar el valor exacto.

Este problema de ruido, puede ser minimizado si se representa a las muestras con un número finito de niveles de amplitud predeterminados. Al proceso, de representar los valores de las muestras analógicas por un conjunto finito de niveles, se le conoce como cuantización. Un ejemplo de este proceso se muestra en la figura 2.2 a y b.[4]

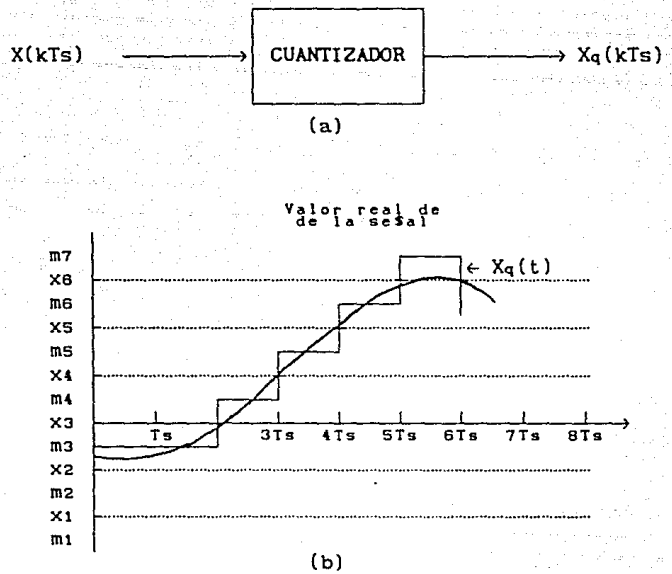


Figura 2.2. Cuantización.

En la entrada del cuantizador existe un proceso x(t) que representa la salida de una fuente de información analógica. Esta forma de onda se muestrea a una tasa apropiada y los valores x(kTs) toman uno de los Q valores posibles $m_1, m_2, \ldots, m_Q$, de acuerdo con una regla predeterminada

$$X_q(kTs) = m_i \qquad si \qquad X_{i-1} \leq X(kTs) \leq X_i$$

$$X_0 = -\alpha, \qquad X_Q = +\alpha$$

A la salida del cuantizador se tendrá una secuencia de niveles, como la mostrada en la figura 2.2, en una forma de onda $X_q(t)$ escalonada, con

$$X_q(t) = X_q(kTs), \qquad kTs \leq t < (k+1)Ts$$

La medida de la calidad de la aproximación de $X_q(t)$ contra $X(t)$ es la relación de señal de salida v.s. ruido de cuantización,

definido como

$$\frac{S_q}{N_q} = \frac{E[\ [X_q(kTs)]^2]}{E[\ [X(kTs) - X_q(kTs)]^2]} \qquad (2.8)$$

Existen dos caminos en el diseño de un cuantizador: [5,6]

(1) Dado un número fijo de niveles de cuantización, se desea minimizar el ruido de cuantización.

(2) Dado un ruido de cuantización, se minimiza el promedio de bits por muestra necesarios para no superar el ruido de cuantización especificado.

En forma básica existen tres formas de realizar estos cuantizadores:

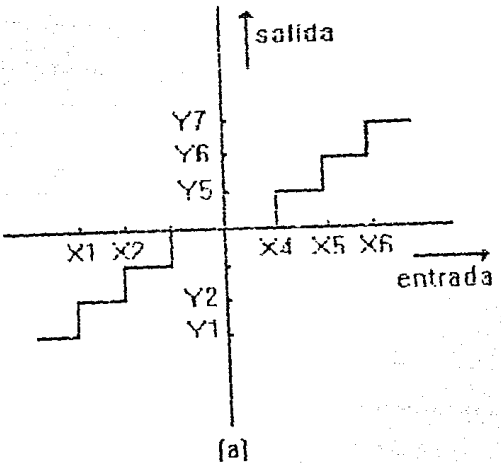
1.-Cuantización sin memoria.
2.-Cuantización de bloque.
3.-Cuantización secuencial.

Donde, la cuantización sin memoria no es más que la representación discreta de una muestra analógica en un tiempo t. En la cuantización de bloque, un conjunto de muestras analógicas se representa por un bloque de valores de salida, seleccionados a partir de un grupo finito de bloques de salida. Y finalmente, en la cuantización de tipo secuencial, esta, se realiza sobre una secuencia de datos utilizando información de las muestras circunvecinas.

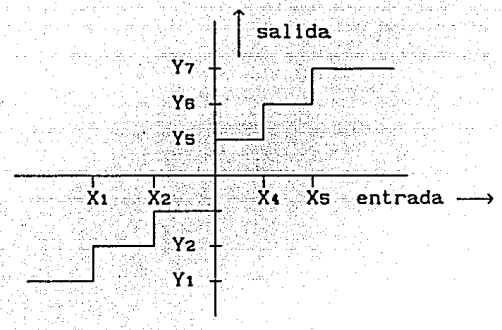
2.4.1.-CUANTIZACION SIN MEMORIA

En este tipo de representación, se trata de minimizar el ruido de cuantización. Dado un número específico de niveles de cuantización, se representa una sola muestra en cada espacio de tiempo , utilizando el mismo cuantizador característico para todas las muestras. El modelo entrada-salida para este cuantizador se muestra en la figura 2.3.

Como puede observarse, la diferencia entre estos dos tipos de cuantizadores reside en el manejo de la región cero. En (a), un cero como valor de entrada produce un cero a la salida, mientras que en (b), existen dos intervalos de entrada en la vecindad del cero, el intervalo positivo produce como valor de salida un nivel positivo y en contraparte el intervalo negativo presenta como salida un nivel negativo, esto quiere decir que en la salida de este cuantizador no existe un nivel cero.



(a)



(b)

Figura 2.3 Cuantizadores característicos.
a) "Midtread" b) "Midriser"

En forma usual, los cuantizadores se caracterizan por el número de niveles de salida, M, y todos presentan un error de cuantización, definido como la diferencia entre el valor de entrada y el correspondiente nivel discreto de salida.

El error de salida instantáneo entre la entrada x y la salida y(x) se define como

$$Q_E = y(x) - x \tag{2.9}$$

La distorsión promedio en términos de la medida de la distorsión media cuadrática es

$$D = \int_{-\alpha}^{+\alpha} [y(x) - x]^2 p(x)\, dx \tag{2.10}$$

Si se conoce la varianza de la señal de entrada σ^2 entonces la relación señal a ruido será: [6]

$$SNR = 10\log_{10}\left[\frac{\sigma^2}{D}\right] \text{ [dB]} \tag{2.11}$$

2.4.1.1.-CUANTIZACION UNIFORME

Para el siguiente análisis x(t) es un proceso aleatorio, estacionario, con media cero y función de densidad de probabilidad $f_x(x)$. Adicionalmente, se abrevian los términos X(kTs) y Xq(kTs) por X y Xq respectivamente.

Suponiendo una cuantización uniforme, esto es, el rango continuo de la variable aleatoria X es dividido en Q intervalos de longitud constante Δ. Si el valor de X cae en el i-esimo intervalo de cuantización, entonces el valor cuantizado de X es el valor medio de ese i-esimo intervalo (fig 2.4).

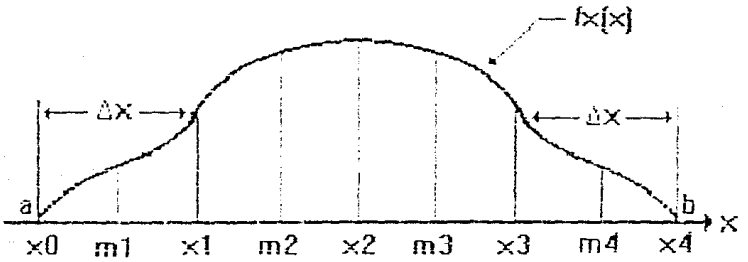


Figura 2.4 Ejemplo de cuantización uniforme

Si a y b son el mínimo y máximo valores de X respectivamente, entonces la medida del escalón o longitud del intervalo Δ esta dado por

$$\Delta = (b-a) / Q$$

La salida cuantizada Xq se genera de acuerdo a

$$Xq = m_i \qquad si \qquad X_{i-1} < X \leq X_i$$

donde $X_i = a + i\Delta$ y $m_i = \frac{X_{i-1} + X_i}{2}$, $i = 1,2,\ldots,Q$

La potencia de ruido de cuantización Nq para el cuantizador uniforme, está dada por:

$$Nq = E[\ (X-Xq)^2] = \int_a^b (X-Xq)^2 f_X(x)\ dx$$

$$= \sum_{i=1}^{Q} \int_{X_{i-1}}^{X_i} (X-m_i)^2 f_X(x) dx$$

donde $\quad X_i = a + i\Delta \qquad m_i = a + i\Delta - \Delta/2$

La potencia de la señal Sq, a la salida del cuantizador puede obtenerse con:

$$Sq = E[\ (Xq)^2] = \sum_{i=1}^{Q} (m_i)^2 \int_{X_{i-1}}^{X_i} f_X(x) dx$$

Para un cuantizador uniforme, con Q niveles y entrada con función de densidad de probabilidad uniforme sobre un intervalo [-a,a], aplicando las fórmulas anteriores obtenemos:

$$Nq = \sum_{i=1}^{Q} \int_{X_{i-1}}^{X_i} (X-m_i)^2 \left[\frac{1}{2a} \right] dx$$

$$= \sum_{i=1}^{Q} \int_{-\alpha+(2-1)\Delta}^{-\alpha+2\Delta} (x+a-i\Delta+\frac{\Delta}{2})^2 \left[\frac{1}{2a} \right] dx = \sum_{i=1}^{Q} \left[\frac{1}{2a} \right] \left[\frac{\Delta^3}{12} \right]$$

$$= \frac{Q\ \Delta^2}{(2a)12} = \frac{\Delta^2}{12} \quad , \text{ con lo que } Q\Delta = 2a$$

para Sq se tiene:

$$Sq = \sum_{i=1}^{Q} (m_i)^2 \left[\frac{1}{2a} \right] = \frac{Q^2 - 1}{12} (\Delta^2)$$

por lo que la relación señal promedio-ruido de cuantización es:

$$\frac{Sq}{Nq} = Q^2 - 1 \cong Q^2 \quad \text{cuando} \quad Q >> 1$$

De este desarrollo se puede concluir, que el valor de la relación señal promedio-ruido de cuantización, depende únicamente del número de niveles Q del cuantizador, y no varía por cambiar la potencia de la señal de entrada.

2.4.1.2.-CUANTIZACION NO-UNIFORME.

Debido a que la mayoria de las fuentes de datos presentan funciones de densidad de probabilidad no-uniformes, muchas veces es necesario optimizar el arreglo, en tamaño, de los intervalos de entrada del cuantizador, para producir la menor distorsión posible en la señal de salida.

No existe un solo modelo para este tipo de cuantizador, cada investigador ha tratado de modelar la función de densidad de probabilidad en consideración a diferentes parámetros y con diferentes pesos de estos. En este trabajo se presentará el modelo de cuantizador no-uniforme desarrollado por W.R. Bennett.[1]

Bennett, modeló un cuantizador para una densidad de probabilidad no-uniforme a tráves de una función no-lineal monotónica creciente, E(x), seguida de un cuantizador uniforme de M niveles. A partir de este modelo, desarrollo una fórmula que apróxima la distorsión media cuadrática por:

$$D = \frac{1}{12M^2} \int_{L1}^{L2} \frac{p(x)}{[\lambda(x)]^2}\, dx \qquad (2.12)$$

donde:

p(x) es la densidad de probabilidad de la señal de entrada

$\lambda(x) = E'(x)/(L2 - L1)$

L2-L1 es el rango del cuantizador

E'(x) es la pendiente de la función alineal

2.4.2.-CUANTIZACION DE BLOQUE.

Como se vió en 2.4, en este tipo de cuantización, un conjunto de muestras de entrada, es representada por otro conjunto de muestras de salida, seleccionados de un grupo finito de bloques de salida. Para realizar la representación de las muestras de salida, una de las técnicas más empleadas, es aplicar una transformación ortogonal. Definiendo: sean dos vectores diferentes de cero $X = [x_1, x_2, \ldots, x_n]$, y $Y = [y_1, y_2, \ldots, y_n]$, se dice que los vectores son perpendiculares u ortogonales, si y sólo si ellos satisfacen la condisión: [10]

$$X \cdot Y = \sum_{i=1}^{n} x_i y_i = 0$$

Así, en el caso de las transformaciones ortogonales, una secuencia de datos muestreados se divide en bloques de N muestras, donde cada bloque ,la transformación ortogonal, la trata como un vector independiente. Reduciendo la distorsión promedio de los elementos cuantizados del vector, para un número dado de niveles de cuantización. [12]

Este tipo de cuantización es realmente un proceso de dos pasos. Primero se aplica una transformación ortogonal a las muestras de una secuencia y después cada una de las muestras transformadas se cuantiza, con un cuantizador de diferente número de niveles para cada muestra, diseñado para minimizar la distorsión promedio de cada una de estas, a una tasa de codificación prefijada.

Esta tasa de codificación utiliza un número fijo de bits, menor al número de bits requeridos para codificar el bloque original, para la codificación del bloque transformado. Dando como resultado una disminución en el número de bits requeridos para codificar el bloque.

En algunos casos, antes de cuantizar las muestras transformadas, estas son reordenadas en términos del valor de su varianza promedio. Tipicamente, los valores de las varianzas de

las muestras reordenadas caen rapidamente, debido a que la transformación coloca la mayor parte de la información del bloque en los valores de algunos componentes, permitiendo hacer una cuantización más eficiente, y resultando en una compresión de los datos originales.[4,8]

Este tema se trata a profundidad en el capítulo dedicado a la transformada coseno.

2.4.3.-CUANTIZACION SECUENCIAL.

En la cuantización de bloque se optimiza la codificación de fuente únicamente en un bloque determinado e independientemente de los bloques previos o subsecuentes.

Sin embargo, en la cuantización secuencial la codificación se realiza permitiendo el traslape de los vectores de datos para procesar, en lugar de un bloque, una muestra en cada instante de tiempo, como ejemplo se pueden mencionar los codificadores DPCM de orden superior, donde se utilizan un grupo de muestras previas para hacer la predicción de la muestra siguiente. [2,3,4,9]

En forma básica, existen dos tipos de cuantizadores secuenciales: codificadores predictivos y codificadores de busqueda en multitrayectoria.

2.4.3.1.- CODIFICADORES PREDICTIVOS.

En este tipo, se incluyen las técnicas de modulación delta y modulación diferencial por pulsos codificados (DPCM), donde la modulación delta utiliza un cuantizador de un solo bit, mientras que DPCM usa un cuantizador de k bits, con $k > 1$.[2,4,6,7]

En estas técnicas se trata de predecir el valor de la muestra de entrada subsecuente, y entonces se cuantiza la diferencia entre el valor resultante de la predicción y el valor real de la muestra de entrada; esta predicción se basa en una combinación de los valores resultantes de las predicciones calculadas con antelación.

La modulación delta y DPCM serán analizadas en un capítulo posterior.

2.4.3.2.-CODIFICADORES DE BUSQUEDA EN MULTITRAYECTORIA (MSC).

A diferencia del tipo anterior, estos codificadores utilizan los valores de las muestras previas y futuras, para seleccionar una versión cuantizada de salida. La compresión de los datos se obtiene porque la versión cuantizada, seleccionada a partir de las muestras de entrada, se codifica en una secuencia binaria, donde cada muestra se representa por un único digito binario.

Existen tres tipos de técnicas MSC de uso actual: libro-código, codificación de árbol y codificación Trellis.[3]

CODIFICACION LIBRO-CODIGO

Para esta codificación se utiliza una lista con los códigos de las 2^n secuencias, de n muestras, que estadisticamente tienen la más alta probabilidad de ocurrencia.[6]

Así, para una secuencia de entrada, se selecciona el indice que representa en la lista la secuencia que minimiza la distorsión entre sí misma y la secuencia de entrada. Después, el indice de la secuencia seleccionada se códifica en una palabra de n bits, y se envia al receptor; donde se tiene una lista de indices igual a la del transmisor para realizar la decodificación.

Si cada muestra tiene M posibles niveles, entonces, existen M^n secuencias posibles, pero únicamente 2^n secuencias de alta probabilidad, dando como resultado una tasa de compresión de:

$$R = \frac{\log_2 M^n}{\log_2 2^n} = \log_2 M \qquad (2.13)$$

CODIFICACION DE ARBOL Y TRELLIS.

Estas técnicas hacen uso de estructuras geométricas normalmente usadas con propósitos de control de error en los canales de comunicación. La diferencia, en este caso, es que las estructuras son utilizadas en el cuantizador para codificar, y no para decodificación como es el caso de los códigos convolucionales.[6,9]

Así mismo, en vez de guardar las 2^n secuencias de más alta probabilidad en una lista, cada secuencia típica se guarda como una serie de ramas en una estructura de tipo árbol.

Cuando se selecciona una secuencia, su trayectoria que le corresponde dentro del árbol, se transmite como una serie binaria, donde cada uno de los bits transmitidos corresponde a un nodo en las ramas de la estructura del árbol.

En la codificación Trellis se utiliza un procedimiento similar al anterior, pero con la estructura del árbol truncada. En la figura 2.5 se muestra la estructura típica de un árbol, con profundidad l, α ramas por nodo y β digitos por rama conteniendo α^l palabras código de longitud $l\beta$.

De esta manera la tasa de codificación en base a las ramas del árbol esta dada por:

$$r = \frac{\log_2 \alpha^l}{l} = \log_2 \alpha \quad \text{[bits/muestra]} \qquad (2.14)$$

Donde:

α = número de ramas/nodo (2 para la figura 2.5)
β = número de digitos/rama
l = profundidad del árbol en ramas

la cuál es independiente de la profundidad del árbol, y para α=2 es igual a un bit por muestra.

En el diseño del árbol existen tres puntos a considerar:

1.- Número de ramas del árbol.
2.- Codificación por búsqueda dentro de las ramas del árbol.
3.- Decodificación, por generación de las ramas correctas, a partir de la trayectoria transmitida.

El número de ramas es el punto más difícil de diseñar, dado que es necesario predecir las secuencias típicas para una fuente de información dada. No existe un método óptimo para realizar esta tarea, aunque las técnicas de codificación aleatoria son las más usadas para seleccionar estas secuencias típicas.

Para la búsqueda, de la codificación dentro de las ramas del árbol, se utiliza un algoritmo que busca secuencialmente M trayectorias, moviendose de un nodo a otro, y dando por terminada la búsqueda cuando alcanza el nodo de nivel L.

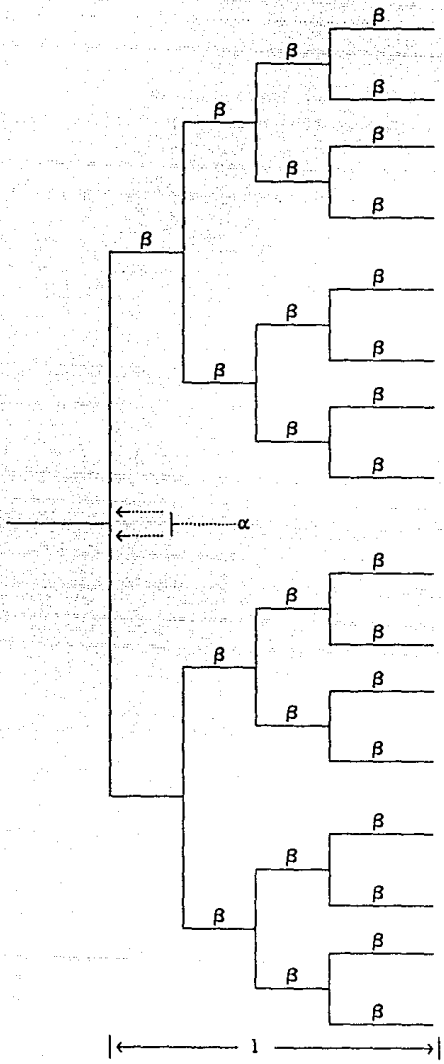
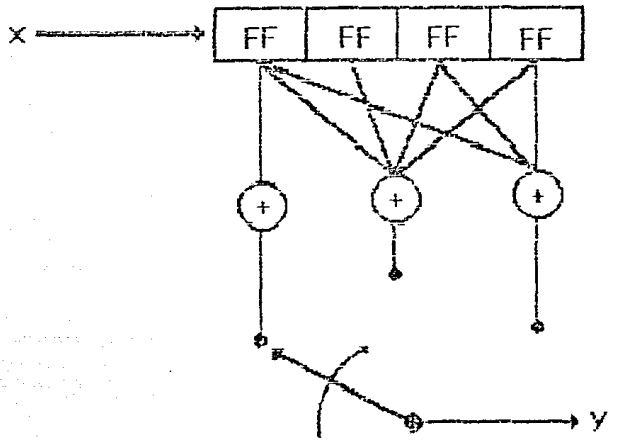


Figura 2.5 Estructura tipica de un árbol.

Para esta técnica, la decodificación es la tarea más fácil de realizar, la implementación se puede llevar a cabo a tráves de un

registro de corrimiento y un sumador de modulo dos. En la figura 2.6 se muestra un ejemplo de este tipo de arreglo.



x = bits del transmisor
y = datos decodificados

Figura 2.6 Decodificador del compresor de árbol

2.5.- REFERENCIAS.

1.- Bennet, W.R., Spectra of Quantized Signals, Bell Systems Technology, vol. 27, July 1948.

2.- Camana, P., Video Bandwidth Compression: A Study in Tradeoffs, IEEE Spectrum, June 1979.

3.- Fehn, H.G., and Noll, P., Multipath Search Coding of Stationary Signals With Applications to Speech, IEEE Transactions on Communications, vol. COM-30, No. 4, april 1982.

4.- Jain, A.K., Image Data Compression: A Review, Proceedings of the IEEE, vol. 69, No. 3, March 1981.

5.- Limb, J.O., Rubinstein, C.B., and Thompson, J.E., Digital Coding of Color Video Signals - A Review, IEEE Transactions on Communications, vol. COM-25, No 11, November 1977.

6.- Lynch, T.J., Data Compression, Von Nostrand Reinhold Company, New York, 1985.

7.- Majid Rabbani, Eastman Kodak Co., SPIES's Short Course Notes, San Diego, California, USA, July 1990.

8.- Pearl, J., Andrews, H. and Pratt, W.K., Performance Measures for Transform Data Coding, IEEE Transactions on Communications, vol. COM-20,No. 3, June 1972.

9.- Stewart, L.C., Gray, R.M., and Yinde, Y., The Design of Trellis Waveform Coders, IEEE Transactions on Communications, vol. COM-30, No 4, April 1982.

10.-Wiley, C.R., Advanced Engineering Mathematics, Editorial McGraw-Hill, 1975.

CAPITULO III ESQUEMAS DE COMPRESION DE VIDEO

3.1.-TECNICAS DE COMPRESION PREDICTIVAS

TEORIA GENERAL.

La modulación diferencial por pulsos codificados DPCM, ha sido desde hace tiempo, un medio atractivo para la codificación de imágenes debido a sus esquemas de baja complejidad. [2, 11, 15, 16]

En las señales de video hay una considerable correlación entre muestras adyacentes, esto quiere decir, que en promedio la señal no presenta cambios bruscos entre muestras circunvecinas; por lo que la varianza entre muestras adyacentes es menor a la varianza del total de la imagen.

Si estas muestras, altamente correlacionadas, se codifican por técnicas de modulación por pulsos códificados (PCM), la señal resultante contiene una gran cantidad de información redundante.

Pero si en lugar del codificador PCM se utiliza un esquema diferencial, como el mostrado en la figura 3.1, el total de información redundante disminuye considerablemente. [15, 18]

En este esquema, en vez de transmitir la muestra de entrada x(n) directamente al receptor, como es el caso en PCM, se transmite la diferencia cuantizada entre la señal de entrada x(n) y el valor de la señal estimada $\hat{x}(n)$. Así, la señal de entrada al cuantizador es:

$$d(n) = x(n) - \hat{x}(n) \qquad (3.1)$$

donde el valor de la señal estimada es la salida de un sistema de predicción P, cuya entrada es una versión cuantizada de la señal de entrada x(n). La señal d(n), conocida como error de predicción, es la señal que se cuantiza en lugar de la señal de entrada x(n). Adicionalmente, la varianza de esta señal es mucho menor que x(n), dando como consecuencia una compresión de la

imagen, al requerirse menos bits para transmitir la misma información.

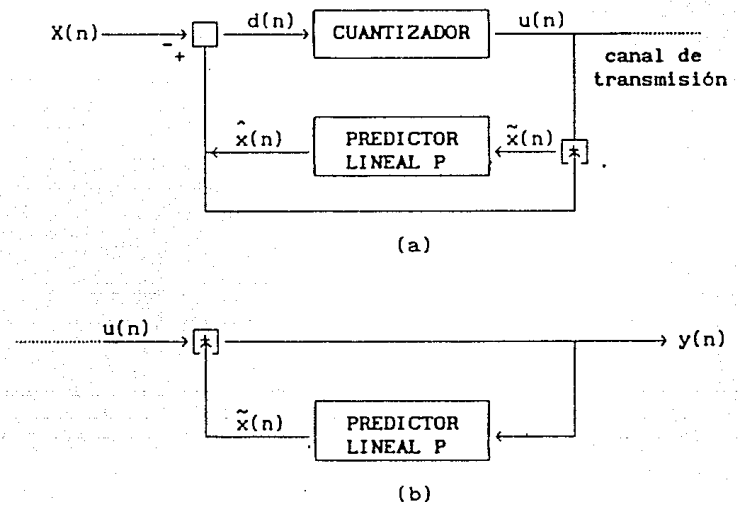


Figura 3.1 Sistema DPCM. a) Transmisor b) Receptor

En este esquema, los parámetros del cuantizador se ajustan en función de la varianza de la señal d(n). La diferencia cuantizada u(n) está dada por:

$$u(n) = d(n) - e(n) \tag{3.2}$$

donde e(n) es el error de cuantización. De acuerdo con la figura 3.1a la señal de diferencia cuantizada se agrega al valor de predicción $\hat{x}(n)$, para dar una versión cuantizada de la señal de entrada, es decir:

$$\tilde{x}(n) = \hat{x}(n) + u(n) \tag{3.3}$$

sustituyendo 3.1 y 3.2 en 3.3 tenemos:

$$\tilde{x}(n) = x(n) + e(n) \qquad (3.4)$$

Por esto, independientemente de las propiedades del sistema de predicción P, la muestra cuantizada $\tilde{x}(n)$ difiere de x(n) sólo por el error de cuantización de la señal de diferencia.

Por lo tanto, si la predicción ha sido elegida correctamente la varianza de la señal d(n) será más pequeña que la varianza de x(n), así, un cuantizador con un cierto número de niveles puede ser ajustado para dar un error de cuantización más pequeño que el que resulta cuando la transmisión de x(n) es directa.

La reconstrucción de la muestra cuantizada se consigue empleando el mismo esquema de predicción de la transmisión. Esto es:

$$y(n) = \hat{x}(n) + u(n) = \tilde{x}(n) \qquad (3.5)$$

donde y(n) es la señal reconstruida y u(n) es la diferencia cuantizada.

En un esquema DPCM, el error de reconstrucción r(n), por muestra n, es idéntica al error de cuantización e(n). El error de reconstrucción esta dado por:

$$r(n) = e(n) = x(n) - y(n) \qquad (3.6)$$

Por lo tanto, el error cuadrático medio será:

$$\sigma_r^2 = \sigma_e^2 \qquad (3.7)$$

donde σ_r^2 es la varianza del error de reconstrucción, y σ_e^2 la varianza de error de cuantización. El término σ_e^2 depende de las condiciones de entrada al cuantizador.

3.1.1 RUIDO GRANULAR Y PENDIENTE DE SOBRECARGA

La operación de un codificador DPCM, se ilustra por medio de las formas de onda en la figura 3.2, donde se utiliza una predicción de primer orden.

En esta figura también se muestran los tipos de errores asociados al proceso de cuantización: distorsión de sobrecarga y ruido granular.[2,11,16] Sin embargo, en esta caso no se cuantiza x(n), como en PCM, sino la señal diferencia d(n).

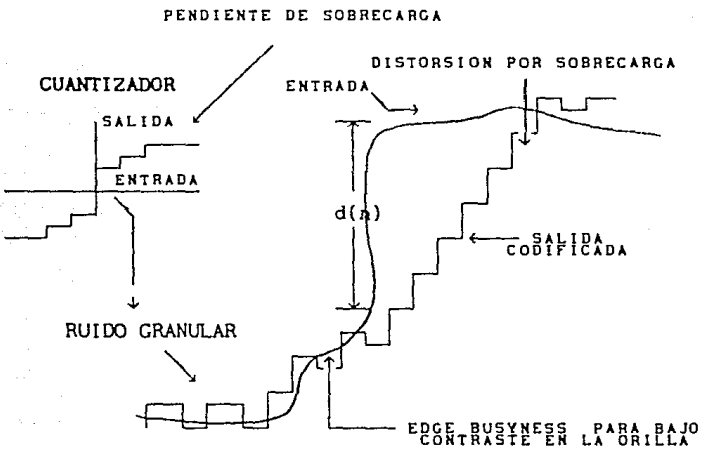


Figura 3.2 Formas de onda de los tipos de errores de reconstrucción en sistemas DPCM. Ruido granular, pendiente de sobrecarga y Edge Busyness.

Los errores de sobrecarga en DPCM se caracterizan no como sobrecarga en amplitud, sino sobrecarga por pendiente, tal como se

muestra en la figura 3.2.

La sobrecarga por pendiente, ocurre siempre que el paso Δ(n) más grande del cuantizador es pequeño para los valores de la señal diferencia d(n), presentandose un alejamiento considerable respecto a los valores de la función de escalera y(n), la cuál presenta cambios rápidos en la entrada. Este problema se presenta, por ejemplo, en imágenes con cambios rápidos de intensidad o con bordes de alto contraste.

La situación opuesta es el ruido granular, que ocurre cuando las variaciones en la entrada son pequeñas en comparación con el mínimo tamaño de paso del cuantizador, este problema se presenta en areas constantes en gris oscuro de la imagen.

Un tercer tipo de errores en DPCM, ocurre en la codificación de orillas o bordes con poco contraste, donde la salida del cuantizador oscila de manera aleatoria, y toma valores ligeramente diferentes entre líneas y de cuadro a cuadro. Este efecto espacio-temporal se conoce en el idioma Ingles como Edge-Busyness. [2, 11, 15, 16]

3.2 PREDICCION LINEAL

Los predictores que se describen a continuación, han sido la base de la baja complejidad en los sistemas DPCM y proporcionan una alta calidad en video, con tasas de transmisión en el orden de R = 4 ó 3 bits/ muestra. [2, 5, 6, 11]

Los sistemas DPCM son de baja complejidad por dos razones: primero, el bajo orden de la predicción implica una correspondiente baja cantidad de memoria para fines de almacenamiento, y segundo, estos sistemas utilizan coeficientes predictivos invariantes en el tiempo, cuyo valor es cálculado a partir de la estadística de la señal, promediada sobre largos períodos de tiempo.

Esta característica simplifica el diseño del esquema de predicción, proporcionando adicionalmente un factor de ganancia de predicción Gp, que se traduce en un mejoramiento significativo en

la calidad de la imagen reconstruida, con respecto a la conseguida por un sistema PCM, el cuál necesita R = 8 ó 9 bits/muestra para digitalización de video de alta calidad.

El esquema de predicción consiste de la suma ponderada de un número de muestras previas de la señal. Específicamente, si x(n-j), es la muestra previa, $\hat{x}(j-1)$ es una estimación, $\hat{x}(j-2)$ la estimación de la segunda muestra anterior, etc.

La expresión general de un predictor lineal se escribe como:

$$\hat{x}(n) = \sum_{j=1}^{N} hj\ x(n-j) \qquad (3.8)$$

Los coeficientes h = { hj }, j = 1,2,...,N, son los factores de ponderación. El predictor más simple, es el que realiza la estimación de la muestra actual empleando únicamente la muestra previa.

$$\hat{x}(n) = hj\ x(n-1) \qquad (3.9)$$

Los coeficientes se calculan para reducir los errores de la estimación bajo criterios de error mínimo. Una medida común del desempeño es la minimización del error cuadrático medio (mse), calculado según la estadística seguida por la muestras de entrada x(n).

3.2.1 PREDICCION DE PRIMER ORDEN

Considerando la predicción $\hat{x}(n) = h_1 x(n-1)$. La señal diferencia será:

$$d(n) = x(n) - h_1 x(n-1) \qquad (3.10)$$

y su varianza:

$$\sigma_d^2 = (1 + h_1^2 - 2\rho_1 h_1)\ \sigma_x^2 \qquad (3.11)$$

donde ρ_1 es el coeficiente de correlación de primer orden. Calculando $\partial(\sigma_d^2)/\partial h_1 = 0$, se encuentra la h_1 óptima y la correspondiente varianza mínima de error de predicción:

$$h_{1\,opt} = \rho \quad ; \quad \min\{\sigma_d^2\} = (1 - \sigma_1^2)\sigma_x^2 \qquad (3.12)$$

donde ρ es el coeficiente de correlación entre muestras. La máxima ganancia de predicción para N = 1, predicción de primer orden, es por lo tanto:

$$\max\{{}^1Gp\} = (1 - \rho_1^2)^{-1} \qquad (3.13)$$

si hacemos $\rho = 0.95$, la máxima ganancia Gp = 10.1 dB, un valor óptimo para h_1 a partir de 3.11 minimizará en forma explícita 3.10.

3.2.2 PREDICCION BIDIMENSIONAL DE ORDEN SUPERIOR

En el tratamiento de imágenes una aplicación importante, es la predicción de segundo orden, la cuál basa la predicción de un valor de intensidad x(i,j), en una muestra previa x(i,j-1) de la misma línea de pixeles, y los valores x(i-1,j) de la línea previa. En la figura 3.3 se muestran los pixeles A,B,C y D sobre los que se basa la predicción.[2]

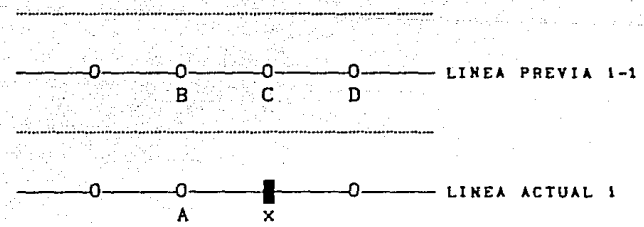


Figura 3.3 Pixeles utilizados en la predicción bidimensional

Existen varios predictores fijos de orden superior basados en la distribución de pixeles mostrada en la figura 3.3. Entre los que mejores resultados producen se encuentran los predictores desarrollados por Zschunke:

(1) $\hat{x} = A$ Primer orden

(2) $\hat{x} = A + D/2$ Segundo orden

(3) $\hat{x} = A + (C-B)/2$ Tercer orden

donde $\hat{x}$ es el valor estimado para la muestra x(n).

Algunas de las razones por las que se utilizan predictores bidimensionales de orden superior en la compresión de imágenes, es para mejorar la relación señal a ruido, aumentar la calidad subjetiva de la imagen reconstruida y dar una tolerancia mayor contra los efectos de los errores generados en la transmisión.

3.3 RELACION SEÑAL A RUIDO EN DPCM

En un sistema de modulación diferencial por pulsos codificados, por definición la relación señal a ruido esta dada por:[16]

$$SNR = \frac{E[X^2(n)]}{E[e^2(n)]} = \frac{\sigma_x^2}{\sigma_e^2} \qquad (3.14)$$

La cuál se puede escribir como:

$$SNR = \frac{\sigma_x^2}{\sigma_d^2} * \frac{\sigma_d^2}{\sigma_e^2} = Gp * SNRQ \qquad (3.15)$$

donde:

$$SNRQ = \frac{\sigma_d^2}{\sigma_e^2} \qquad (3.16)$$

es la relación señal a ruido del cuantizador típico en un esquema PCM, y la cantidad:

$$Gp = \frac{\sigma_x^2}{\sigma_d^2} \qquad (3.17)$$

se define como la ganancia de predicción debido a la configuración diferencial.

En la relación señal a ruido, la cantidad SNRQ es dependiente del cuantizador específico que se utilice, y de las propiedades de d(n). Si la cantidad Gp es mayor que la unidad, representa una ganancia en SNR debida al esquema diferencial.

Para una señal dada, σ_x^2 es una cantidad fija, Gp se puede maximizar, minimizando la varianza de error de predicción o σ_d^2 de la expresión 3.17.

3.4 SISTEMAS ADAPTIVOS (ADPCM)

Debido a la falta de estacionariedad de la estadística de las imágenes, es imposible diseñar un cuantizador o predictor simple que siempre sea óptimo. Existen técnicas adaptivas que utilizan un rango de cuantización y/o predicción característico que por lo general es óptimo, y se seleccionan de acuerdo a un grupo de reglas basadas en las propiedades de las imágenes. Por lo tanto,

al diseñar sistemas adaptivos se deben utilizar parámetros variables que cambien con las variaciones de las señales, esto es, se considera que las señales son estacionarias por intervalos.

En esquemas DPCM la adaptabilidad puede ser tanto en un predictor fijo como en un cuantizador fijo ó utilizando predictores y cuantizadores adaptivos. [2, 11, 15]

3.4.1 CUANTIZACION ADAPTIVA

En los esquemas DPCM descritos anteriormente se utilizó un cuantizador uniforme con tamaño de paso $\Delta(n)$ fijo.

Sin embargo, es deseable tener un cuantizador con el tamaño de paso suficientemente grande para acomodar el máximo rango pico a pico de la señal de entrada con el mismo número de niveles. En contraparte, para algunos casos es conveniente que el tamaño de paso de cuantización sea lo suficientemente pequeño para minimizar la varianza del ruido de cuantización.

Una alternativa, para resolver este problema, es el uso de un cuantizador adaptivo; el cuál adapta el tamaño de paso de cuantización al valor del nivel de la señal de entrada.

Basicamente existen dos tipos de cuantizadores adaptivos: el primero con estimación hacia-adelante (feed-forward) o directa y el segundo con estimación realimentada (feed-back). [16, 18]

Un tipo de cuantizador adaptivo que utiliza la estimación directa, basandose en muestras $x(n)$ sin cuantizar, se muestra en la figura 3.4. En este esquema el tamaño de paso $\Delta(n)$ del cuantizador es proporcional a la señal de entrada, y la señal diferencia $d(n)$ será también proporcional a la entrada, siendo posible controlar el tamaño de paso del cuantizador ya sea desde $d(n)$ ó la entrada $x(n)$.

En la figura 3.5 se ilustra un cuantizador adaptivo realimentado, una ventaja del control realimentado, es que la información del tamaño de paso es derivada desde una secuencia de palabras código $c(n)$, así que no es necesario transmitir o

almacenar información adicional sobre el tamaño de paso Δ(n). Esto hace que la calidad de la señal reconstruida sea más sensitiva a los errores generados en la transmisión.

Con el control de alimentación directo, la palabra código c(n) y el tamaño de paso Δ(n) son necesarios para reconstruir la señal original, por lo que debe realizarce su transmisión al receptor.

Con los esquemas ADPCM es posible tener una mejoría en la relación señal a ruido (SNR) de entre 10 y 12 dB, con respecto a los sistemas de cuantización fija que utilizan el mismo número de niveles.

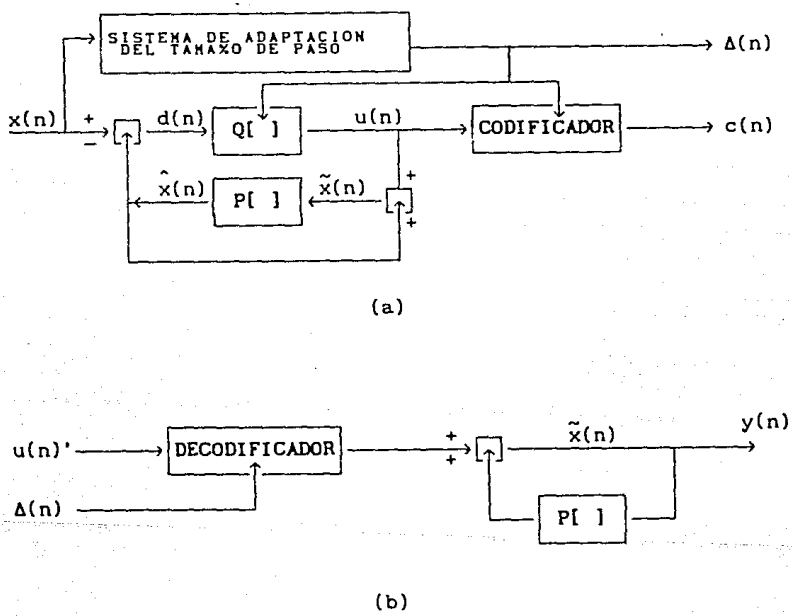
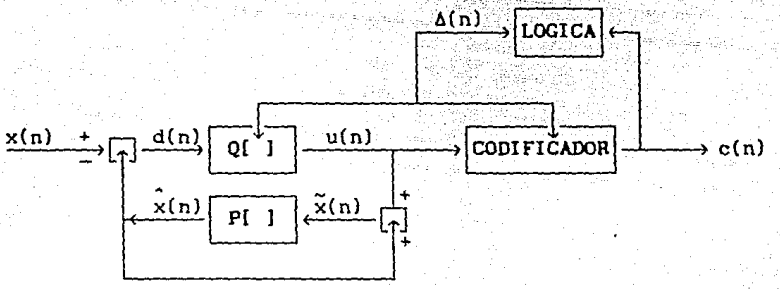
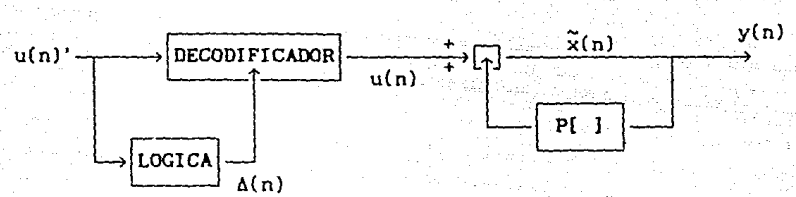


Figura 3.4 Sistema ADPCM con predicción directa
a) Transmisor b) Receptor



(a)



(b)

Figura 3.5 Sistema ADPCM con predicción realimentada
a) Transmisor b) Receptor

En la figura 3.6 se ilustra un cuantizador adaptivo con R = 3 bits/muestra, 8 niveles y L/2 = 4 multiplicadores de tamaño de paso M. Sólo se muestran cuatro valores distintos, debido a la simetría del cuantizador. Los niveles de salida del cuantizador son:

$$y(n-1) = H(n-1)\ \Delta(n-1)/2 \qquad (3.18)$$

donde: $H(n-1) = 1,3,5,\ldots,2^{R-1}, \quad R \geq 2$

La lógica de adaptación requiere que el tamaño de paso del cuantizador $\Delta(n)$, se incremente como el producto de $\Delta(n-1)$ y el multiplicador $M(*)$, que es una función invariante con el tiempo del último indice de magnitud $|H(n-1)|$:

$$\Delta(n) = M\,|H(n-1)| * \Delta(n-1) \qquad (3.19)$$

donde $\Delta(n)$ es el tamaño de paso del cuantizador en un instante n, $\Delta(n-1)$ es el tamaño de paso en otro instante n-1, y $M|H(n-1)|$ es una constante de multiplicación dependiente de la magnitud de $H(n-1)$.

En el ejemplo, de la figura 3.6, se muestran cuatro posibles valores de $H(n-1) = 1,3,5,7$, con sus respectivos multiplicadores M_1, M_2, M_3 y M_4.

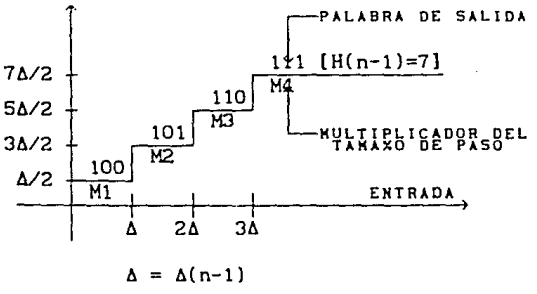


Figura 3.6 Sección de un cuantizador adaptivo midriser, R = 3.

En el diseño de los multiplicadores es conveniente seguir la siguiente relación:

$$1 > M_1 \leq M_2 \leq M_3 \leq M_4 > 1 \qquad (3.20)$$

En este cuantizador cuando las salidas son $\pm 7\ \Delta(n-1)/2$, el

cuantizador esta en sobrecarga y el tamaño de paso debe incrementarse por un factor de $M_4 > 1$.

Por otro lado, una salida de $\pm \Delta(n-1)/2$ representa una entrada pequeña y el tamaño de paso debe disminuirse por un factor de $M_1 < 1$. En condiciones intermedias el cuantizador esta sujeto a los valores intermedios M_2 y M_3. El tamaño de paso se incrementa y se decrementa, implicando que el cuantizador se expande ó contrae de acuerdo con el nivel del valor de la señal de entrada. En la figura 3.6 las características del cuantizador se mantienen , en cada caso, para un espacio de tiempo n.

En el caso más simple, la adaptibilidad comprende únicamente los valores máximo y mínimo del tamaño de paso. Esto es:

$$\Delta_{min} \leq \Delta(n) \leq \Delta_{m\,x} \qquad (3.21)$$

El valor de Δ_{min} controla el nivel de ruido del canal desocupado (el ruido granular generado por entradas cero o cercanas a cero), mientras que $\Delta_{m\,x}$ controla la distorsión del nivel de sobrecarga, los valores de Δ_{min} y $\Delta_{m\,x}$ estan en función de R.

Por definición, el rango dinámico del cuantizador, es el rango de variación en σ_x , dentro del cuál el cuantizador puede ser eficiente. Una ventaja de los sistemas adaptivos esta en el incremento del rango dinámico del cuantizador para un valor máximo de SNR; donde la relación $\Delta_{m\,x}/\Delta_{min}$ controla este rango.

3.4.2 PREDICCION ADAPTIVA

Los sistemas adaptivos proporcionan una mejoría subjetiva en la calidad de las imágenes, especialmente en los contornos. Esto se puede observar en los sistemas PCM y DPCM de primer orden, cuando se transmite a tasas muy bajas y por consiguiente la calidad de las imágenes es muy deficiente.[11,18]

Una técnica común ,para la adaptibilidad de parámetros, es utilizar predictores que toman una o varias muestras previas, como

se ilustra en la figura 3.3. Otra forma , es seleccionar para cada pixel una predicción de acuerdo con el contorno de la imagen.

En la figura 3.3 se muestran las posiciones de los pixeles cercanos a la muestra $\hat{x}(n)$. Estos pixeles son utilizados por los algoritmos de predicción fija, a tráves de combinaciones de muestras previas, para realizar su trabajo.[10,18] Sin embargo, se ha observado que estas combinaciones de pixeles favorecen la predicción de los valores en ciertas direcciones de contorno de la imagen, ,pero dan resultados pobres para otras.

Esta dificultad puede superarse si la predicción se modifica conforme al contorno entre los pixeles A, B, C y D, y con una resolución más fina, si se interpolan valores adicionales.

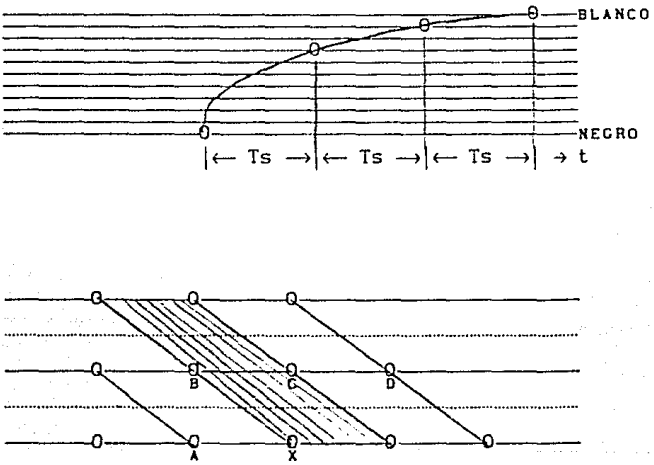


Figura 3.7 Transición del contorno a lo largo de una línea , y su mapeo de líneas de contorno en el plano de la imagen.

En la parte superior de la figura 3.7 se ilustra un paso largo del contorno de la linea examinada. Como lo muestra la parte inferior de la misma figura, para representar el contorno sobre todo el plano de la imagen se prefiere hacer el mapeado por lineas de contorno de igual amplitud o igual altitud.

Para esta representación es conveniente que la dirección del contorno atraviese el pixel actual $\tilde{x}(n)$. Así, la dirección del contorno puede tomarse desde cada uno de los pixeles cercanos y puede utilizarce para predecir la muestra presente $\tilde{x}(n)$ por medio de alguno(s) de los pixeles A, B , C o D.

La dirección del contorno puede conocerse calculando las diferencias de los pixeles cercanos; la determinación de la dirección del contorno es muy importante para el buen funcionamiento de la predicción adaptiva. Uno de los problemas de esta técnica, esta en el cálculo de la dirección de contorno en el caso de áreas uniformes de la imagen, pudiendose obtener desiciones diferentes a la correcta.

Una de las técnicas que ha producido buenos resultados, es la determinación de la dirección de contorno a tráves de un "punto de contorno", esto es, si la diferencia absoluta entre la muestra actual x(n) y una muestra previa x(n-1) exceden un umbral Lc, se tiene el inicio de un contorno en la dirección calculada, pudiendose utilizar el mejor predictor para esta dirección.

$$|X_i - X_{i-1}| > Lc \qquad (3.22)$$

De otra manera, el pixel esta dado en la dirección "horizontal", que apunta hacia la muestra previa.

El umbral utilizado deberá ser pequeño con el objeto de detectar el inicio de un contorno, tan rápido como sea posible. Pero esto, en un momento puede generar errores de contorno debido a que el umbral de Lc es demasiado pequeño.

Si la señal procesada es ruidosa, ya sea por ruido de

cuantización o ruido de la fuente de la señal, una buena elección de Lc es un valor del 10% del valor de la amplitud de la señal.

Una vez calculada la dirección del contorno, un acercamiento del pixel a un valor interpolado entre dos valores adyacentes es usada como un valor de predicción en los extremos de transmisión y recepción.

En las áreas planas de la imagen donde solamente las direcciones "horizontales" estan presentes, se emplea una predicción simple $\hat{x} = A$.

En general, los esquemas adaptivos, sobre los fijos, presentan una mejoría en la relación señal a ruido SNR de entre 10 y 14 dB, con una buena calidad en la imagen reconstruida en tasas de 2 - 3 bits/pixel. Los errores, generados en el canal de transmisión, se presentan en la imagen como lineas de tono diferente cuando se usan predictores de primer orden, y como rayos en los predictores de segundo orden. Si se utiliza una codificación con palabras de longitud variable, el efecto de un sólo bit erroneo puede ser catástrofico.

3.5 CODIGOS POR TRANSFORMACION

Una alternativa para la codificación predictiva es el empleo de los códigos por transformación. Este tipo de codificador utiliza transformaciones lineales para mapear un vector de entrada de N muestras en un espacio transformado, donde las muestras transformadas son comprimidas para transmitirse o almacenarse. En el receptor las muestras comprimidas son decodificadas y reconstruidas aplicándoles la transformación inversa. Este proceso se muestra en la figura 3.8.[11]

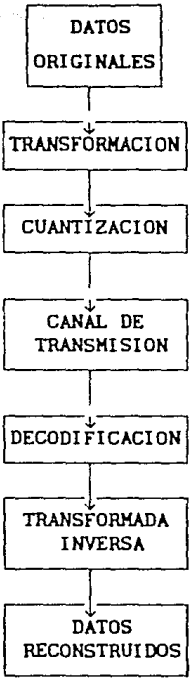


Figura 3.8 Esquema de un sistema de compresión de datos usando transformaciones.

La idea de codificar imágenes empleando métodos por transformación, es el de transformar un conjunto de datos dentro de otro conjunto "menos correlacionado", antes de codificarlo para su transmisión.[10,11]

Existen diversas estrategias en códigos por transformación, una estrategia común, es dividir la imagen en sub-imágenes de NxN para reducir los cálculos de la transformada, ya que el número de operaciones se incrementa cuadráticamente con el tamaño del bloque a transformar. El tamaño de los bloques se selecciona preferentemente en valores de potencias de dos y no existe

traslape entre bloques. Esto se muestra en la figura 3.9:

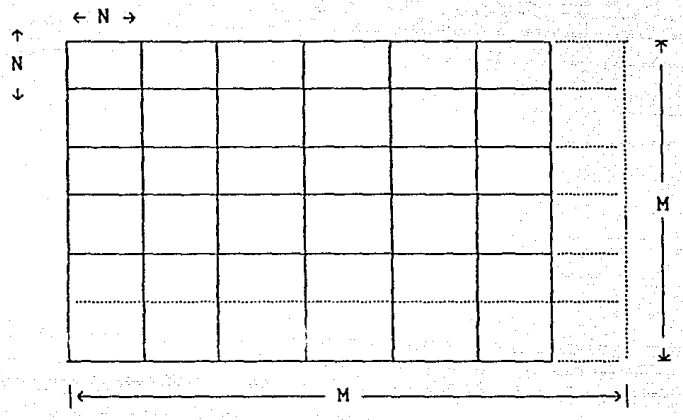


Figura 3.9 Sub-división de una imagen de MxM pixeles en bloques de NxN.

Los códigos por transformación se desarrollan como sigue: suponiendo un vector **u**, con media cero y covarianza **R**, este se transforma en una matriz **A** de orden **N** x **N**, que produce un vector (complejo) **v**, tal que sus componentes {v(k)} estan descorrelacionados, o en otras palabras, los valores de los tonos de los pixeles v(k), no son predecibles a partir de los valores de sus puntos vecinos. [1,2,5,11]

Cada componente v(k) se cuantiza independiente y en concordancia con su contribución al contenido de información y calidad subjetiva de la imagen, generandose el vector $\mathbf{v}^*$. Los componentes con mayor importancia en la reconstrucción de la imagen se cuantizan con más precisión, y los menos importantes se desprecian, lograndose con esto una compresión de los datos.

La salida del vector $\mathbf{v}^*$ es linealmente transformado por una matriz **B** para producir un vector $\mathbf{u}^*$ (ver figura 3.10). Esto se

desea para encontrar la matriz óptima descorrelacionada **A**, la reconstrucción de **B**, y un cuantizador óptimo, tal que la distorsión media cuadrática será:

$$D = \frac{1}{N} E\{\sum_{k=1}^{N} [u(k) - u^{*}(k)]^2\} \qquad (3.23)$$

La solución de este problema requiere que $B = A^{-1}$, que **A** sea la matriz descorrelacionadora óptima (transformada de Karhunen-Loeve) de **u**, en otras palabras las filas de **A** son eigenvectores ortonormales de **R**, y que el cuantizador sea óptimo.[11] Por definición, los componentes resultantes v(k) se encuentran descorrelacionados.[5]

Considerando de ahora en adelante que las filas de **A** son ortonormales, resulta:

$$A A^{*T} = I \qquad (3.24)$$

"

$$A^{-1} = A^{*T} \qquad (3.25)$$

y **A** es unitaria. Adicionalmente, la varianza de los coeficientes transformados es diferente para cada uno de ellos. La matriz descorrelacionadora óptima desarrollada anteriormente, se conoce como la transformada Karhunen-Loeve (KL).

La transformada KL tiene la propiedad de que para cualquier $M \leq N$, empaca la mayor cantidad de energía en algunas M componentes de v. Sin embargo, aunque la transformada KL es la transformación óptima, por su capacidad de empacar energía (usando el criterio de la energía media cuadrática), es difícil de calcular y adicionalmente no existe un algoritmo rápido asociado a ella, le toma muchas operaciones efectuar la transformación (N^2 operaciones para transformar un vector de N datos).

Es por esto, que la mayor parte de los trabajos se enfocan en transformadas seno, coseno, Fourier, etc, que son sub-óptimas con respecto a la transformada KL, pero si tienen transformaciones

rápidas asociadas a ellas. [1,7,8,11,13,17,19]

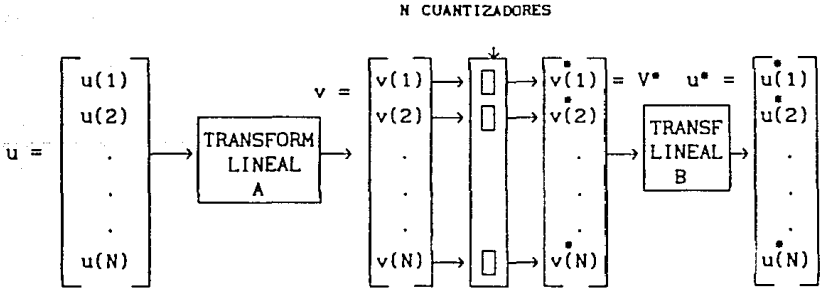
En dos dimensiones, se considera la clase de transformaciones de una imagen N x M dada por:

$$V = A_N \, U \, A_M^T \qquad (3.26)$$

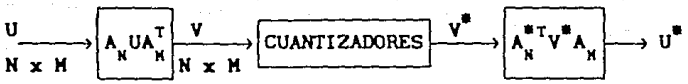
donde A_N es una transformada rápida unitaria N x N, de los tipos mencionados en el párrafo anterior. La transformación de 3.26, se dice separable, porque puede ser realizado por una transformación modo-columna ($A_N U$) seguida por una transformación modo-renglón.

Para que la codificación por transformación sea práctica, la imagen es sub-dividida en pequeños bloques. Así por ejemplo, si una imagen de 256 x 256 se códifica en bloques de 16 x 16, en lugar del bloque original de 256 x 256, el almacenamiento se reduce en un factor de 256 y la velocidad de ejecución del algoritmo se acelera por un factor de dos, para una transformada que requiera N $\log_2 N$ operaciones (coseno discreta) en transformar un vector de N x 1 datos. [11]

La figura 3.10 muestra en diagrama a bloques los esquemas básicos de los códigos por transformación unidimensional y bidimensional.



(a)



(b)

Figura 3.10 a) Código por transformación unidimensional
b) Código por transformación bidimensional

3.5.1 TRANSFORMADA COSENO DISCRETA

Como se ha visto, la transformada Karhunen-Loeve aunque óptima, no tiene un algortimo rápido asociada a ella. Una de las transformadas sub-óptimas que tienen una ejecución más parecida a la transformada KL es la transformada coseno discreta, con la ventaja de tener asociados a ella, algoritmos de cálculo rápido a tráves de la FFT, y evitando cálculos con algebra compleja.
La transformada coseno discreta bidimensional se define como: [1,5]

$$F(u,v) = \frac{4C(u)C(v)}{n^2}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} f(j,k)\cos\left[\frac{(2j+1)u\pi}{2n}\right]\cos\left[\frac{(2k+1)v\pi}{2n}\right] \quad (3.27)$$

con $u,v = 0,1,2,\ldots,N-1$

$$c(\omega) = \begin{cases} \frac{1}{\sqrt{2}} & ; \text{ para } \omega = 0 \\ 1 & ; \text{ para } \omega = 1,2,\ldots,N-1 \end{cases}$$

Adicionalmente la transformada coseno discreta inversa está definida por:

$$f(j,k) = \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} C(u)C(v)F(u,v)\cos\left[\frac{(2j+1)u\pi}{2n}\right]\cos\left[\frac{(2k+1)v\pi}{2n}\right] \quad (3.28)$$

con $u, v = 0, 1, 2, \ldots, N - 1$

$$c(\omega) = \begin{cases} \frac{1}{\sqrt{2}} ; & \text{para } \omega = 0 \\ 1 ; & \text{para } \omega = 1, 2, \ldots, N - 1 \end{cases}$$

El proceso de cálculo de la transformada es como sigue: primero se subdivide la imagen en $(M/N)^2$ bloques de N x N pixeles tal como lo ilustra la figura 3.9. Haciendo:

$$j = 0, 1, 2, \ldots, (M/N)^2 = \text{número de bloque en la imagen}$$

entonces, las componentes transformadas de un bloque se pueden representar por $F_j(u, v)$, y el arreglo matricial de las componentes será:

$$\begin{bmatrix} F_j(0,0) & F_j(0,1) & F_j(0,2) \ldots & F_j(0,N-1) \\ F_j(1,0) & F_j(1,1) & F_j(1,2) \ldots & F_j(1,N-1) \\ \vdots & \vdots & \vdots & \vdots \\ F_j(N-1,0) & F_j(N-1,1) & F_j(N-1,2) \ldots & F_j(N-1,N-1) \end{bmatrix}$$

Figura 3.11 Arreglo matricial de los componentes transformados en el bloque j de la imagen.

En la figura 3.12 se muestra como se comportan las varianzas de las componentes de varios tipos de transformadas.[1, 17] Dado que la varianza es una medida del contenido de información en los componentes transformados, esta gráfica nos demuestra lo dicho en 3.5, la transformada KL tiene el mejor desempeño para empacar la

energía de los componentes, distribuyendo la mayor parte de la energía en los primeros componentes y menos energía, que las otras transformadas, en los otros componentes.

Es de notarse, que la transformada coseno discreta (DCT), tiene un desempeño para empacar energía aproximadamente idéntico a la KL para $\rho = 0.95$.

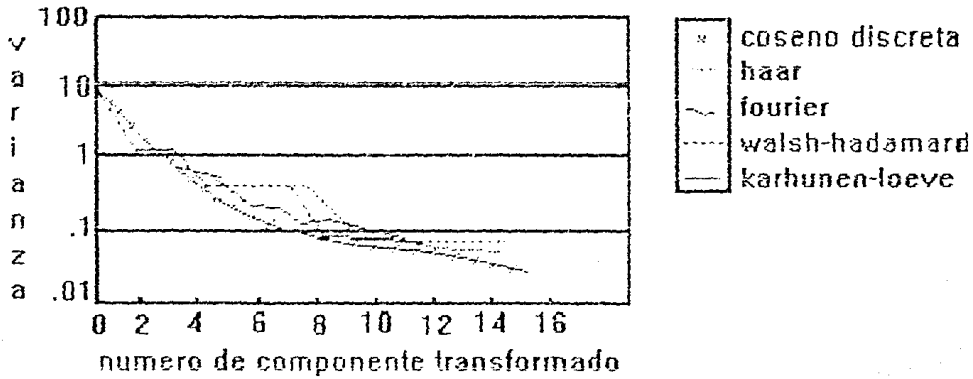


Figura 3.12 Varianza de las transformadas; M = 16, $\rho = 0.95$

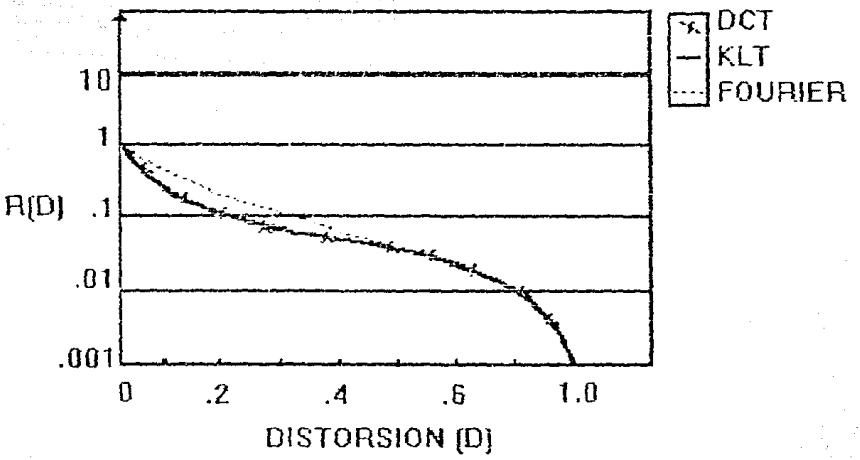


Figura 3.13 Tasa contra distorsión, M = 16, ρ = 0.9

En 3.13 se puede ver como la transformada coseno discreta y la transformada Karhunen-Loeve, muestran un desempeño similar ante la distorsión.

En la transformada coseno discreta, las componentes transformadas tienen una distribución Gaussiana, exepción de la componente de DC con una distribución Rayleigh.[3]

3.5.2 IMPLANTACION RAPIDA DE LA DCT VIA LA FFT.

Sea x(t) una señal en el dominio del tiempo, y $\{\phi_k\}$, k = 1, ...,α, un grupo de funciones base en el mismo espacio de la señal, entonces x(t) se puede representar como:

$$x(t) = \sum_{k=1}^{\alpha} X_k \phi_k(t) \qquad (3.29)$$

Similarmente si la señal x(t) es muestreada en intervalos regulares Δt, la i sima muestra se puede representar por:

$$x(i\Delta t) = \sum_{k=1}^{\alpha} X_k \phi_k(i\Delta t) \qquad (3.30)$$

Si la señal es de duración finita, con N muestras de longitud:

$$x_i = \sum_{k=1}^{N} \phi_{ik} X_k, \qquad i = 1,\ldots,N \tag{3.31}$$

donde $x_i = x(i\Delta t)$, $\phi_{ik} = \phi_k(i\Delta t)$, y X_k los coeficientes de expansión.[5]

Expresando 3.31 en notación matricial:

$$\mathbf{x} = \Phi X \tag{3.32}$$

donde $\mathbf{x} = \{x_1,\ldots,x_N\}^T$, $X = \{X_1,\ldots,X_N\}^T$, Φ es la matriz de N x N dada por:

$$\Phi = \begin{matrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1N} \\ \vdots & \vdots & & \vdots \\ \phi_{N1} & \phi_{N2} & & \phi_{NN} \end{matrix} \tag{3.33}$$

La ecuación 3.32 indica que la señal temporal x(t) , se puede representar como una combinación de las funciones "frecuenciales" ϕ_{ik}, con los coeficientes dados por el vector X. X se dice es la transformada de $\mathbf{x}$, con todas las propiedades de la transformada contenidas en la matriz Φ. Si:

$$\Phi^{-1}\Phi = \Phi^{*}\Phi = I \tag{3.34}$$

la matriz es no-singular (el signo * indica que la matriz compleja Φ se conjuga y transpone) y se dice que la transformada es unitaria. Si Φ es real, entonces, la transformada es unitaria si:

$$\Phi^{T}\Phi = I \tag{3.35}$$

donde T denota la matriz traspuesta.

Las transformadas unitarias tienen un papel muy importante para las transformadas discretas, porque invertirlas es fácil y preservan la energía de sus componentes. El vector $\mathbf{X} = \Phi^{-1}\mathbf{x}$ se conoce como la transformada de $\mathbf{x}$.

Es necesario aclarar que no todas las transformadas discretas son unitarias, dado que para esto es necesario que la matriz Φ tenga vectores fila linealmente independientes, pero la transformada coseno discreta si pertenece a esta clase de matrices.

Si las funciones base de la ecuación 3.31 son funciones cosenoidales, a partir de ellas se pueden obtener transformadas coseno con diferentes arreglos para su implementación. La primera fué desarrollada en 1974 por Ahmed[1], después, en 1980 Kitajima[12] desarrollo una versión simétrica, y en 1984 Wang[20], variando los corrimientos de fase de las funciones base e invirtiendo los indices i y k, estudio de manera sistemática las implementaciones rápidas de la transformada coseno discreta. Las cuatro versiones básicas de DCT's desarrolladas se definen como:[5]

DCT tipo I $$\phi_{ik} \equiv [C^{I}]_{ik} = \sqrt{\frac{2}{N}} \left[c_i c_k \cos\left(\frac{ik\pi}{N} \right) \right] \quad i,k=0,\ldots,N$$

DCT tipo II $$\phi_{ik} \equiv [C^{II}]_{ik} = \sqrt{\frac{2}{N}} \left[c_i \cos\left(\frac{i(k+\frac{1}{2})\pi}{N} \right) \right] \quad i,k=0,\ldots,N-1$$

DCT tipo III $$\phi_{ik} \equiv [C^{III}]_{ik} = \sqrt{\frac{2}{N}} \left[c_k \cos\left(\frac{k(i+\frac{1}{2})\pi}{N} \right) \right] \quad i,k=0,\ldots,N-1$$

DCT tipo IV $$\phi_{ik} \equiv [C^{IV}]_{ik} = \sqrt{\frac{2}{N}} \left[\left[i+\frac{1}{2} \right] \cos\left(\frac{i(k+\frac{1}{2})\pi}{N} \right) \right] \quad i,k=0,\ldots,N-1$$

donde:

$$c_i = \begin{cases} 1 ; & \text{si } i \neq 0 \text{ " } N \\ \frac{1}{\sqrt{2}} ; & \text{si } i = 0 \text{ " } N \end{cases} \qquad (3.36)$$

Para cada uno de estos tipos se han desarrollado algoritmos de cálculo rápido, básicamente a tráves de decimaciones temporales (rearreglando y combinando las muestras de entrada antes del procesamiento) y las que utilizan las técnicas de la Transformada rápida de Fourier (FFT).

A continuación se describe el algoritmo rápido de la DCT implementado en esta tesis.[4,5]

La DCT tipo II de una secuencia de datos x(m), m=0,1,...,N-1, y su DCT^{-1} pueden ser definidas como:

$$X_c(k)=\sqrt{\frac{2}{N}}\, c(k) \sum_{m=0}^{N-1} x(m)\cos\left[\frac{(2m+1)k\pi}{2N}\right], \qquad k = 0,1,\ldots,N-1 \quad (3.37)$$

$$X(m)=\sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} c(k)X_c(k)\cos\left[\frac{(2m+1)k\pi}{2N}\right], \qquad m = 0,1,\ldots,N-1 \quad (3.38)$$

donde:

$$c(k) = \begin{cases} 1/\sqrt{2} , & k = 0 \\ 1 , & k = 1,2,\ldots,N-1 \end{cases}$$

La ecuación 3.30, también se puede expresar como:

$$X_c(k)=\sqrt{\frac{2}{N}}\, c(k)\,\mathrm{Re}\left[e^{-jk\pi/2N} \sum_{m=0}^{2N-1} x(m)W_{2N}^{mk}\right], \qquad k = 0,1,\ldots,N-1 \quad (3.39)$$

donde $W_{2N} = \exp(-j2\pi/2N)$ y $x(m)=0$, $m=N, N+1,\ldots,2N-1$.

Que en palabras, nos indica que el cálculo de la DCT sobre una sucesión de N muestras se puede implantar al sumar N ceros a la secuencia original, y posteriormente aplicarle una FFT de 2N puntos a la secuencia resultante. La manera de implantar esta técnica se muestra en el diagrama a bloques de la figura 3.14. Es necesario notar que las operaciones $\exp(-jk\pi/2N)$ y la parte real, son necesarias en la configuración de este algoritmo.

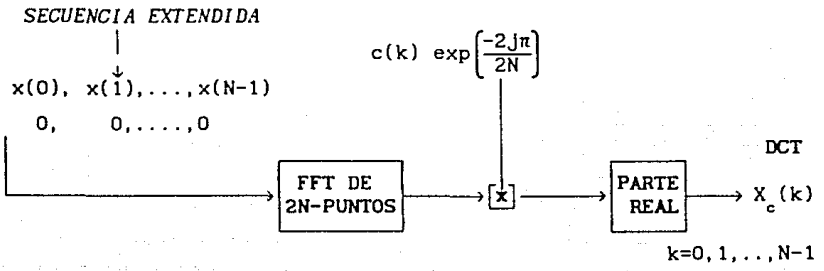


Figura 3.14 Cálculo de una DCT, a tráves de una FFT

3.6 CODIFICACION HUFFMAN.

Desarrollada por D.A. Huffman, este tipo de codificación reversible (ver capitulo 2.1), siempre produce el grupo óptimo de palabras código, en el sentido de que ningún otro tipo de codificación reversible presenta un promedio más pequeño de simbolos por mensaje.[6]

Considerese un conjunto de M mensajes $u_1, u_2, \ldots, u_M$ que ocurren con probabilidades $P(u_1)$, $P(u_2), \ldots, P(u_M)$, y un alfabeto de D simbolos. El procedimiento siguiente, dá como resultado el código de palabras Huffman de un grupo de mensajes:

Paso 1 Los M mensajes se areglan en orden decreciente de probabilidades, esto se muestra en el ejemplo de la figura 3.15

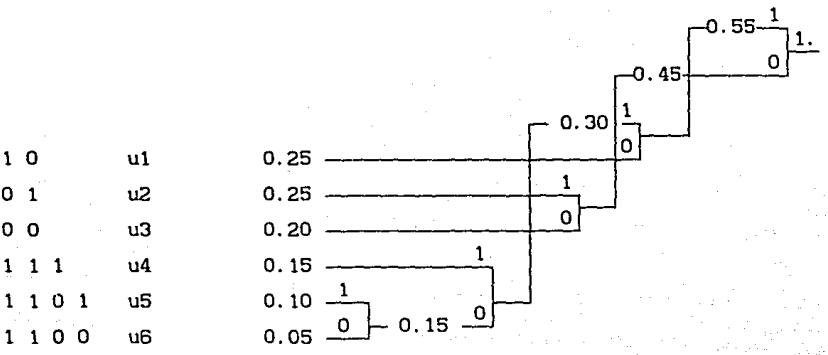


Figura 3.15 Codificación Huffman

Paso 2 Sea m_o un entero que satisface:

$$2 \leq m_o \leq D \qquad (3.40)$$

$$\frac{M - m_o}{D - 1} = \text{entero positivo} \qquad (3.41)$$

Note como cuando $m_o = 2$ también $D = 2$.

Una los dos mensajes m_o menos probables , sumando las probabilidades de esta pareja de mensajes.

Paso 3 Se construye un nuevo mensaje, que es la suma de los mensajes generados en el paso 2. Este nuevo mensaje tendrá una probabilidad equivalente a la suma de las probabilidades de los mensajes del paso 2. Rearregle los mensajes nuevamente en orden decreciente de probabilidades, ver figura 3.15.

Paso 4 Se forma una nueva pareja de los D mensajes menos probables, a partir del arreglo auxiliar generado en el paso 3, y se suma de nuevo el valor de las probabilidades de los dos mensajes con la menor probabilidad

Paso 5 Se construye un nuevo arreglo auxiliar, a partir del primero considerando la pareja de D mensajes, formados en el paso 4, como un sólo mensaje con probabilidad igual a la probabilidad total de la pareja. Se rearreglan los mensajes en orden de probabilidades decrecientes.

Paso 6 Se forman arreglos auxiliares, por repetir los pasos 4 y 5, hasta que se tiene un sólo mensaje con probabilidad unitaria, en la parte izquierda del árbol; tal como lo muestra la figura 3.15.

Paso 7 Unas vez terminados los pasos anteriores, se produce un árbol con un número determinado de nodos. Ahora, las palabras código se forman por asignar diferentes simbolos del alfabeto predeterminado a las ramas salientes en cada nodo, tal como se muestra en la figura 3.15. Por ejemplo, para este caso D = {0,1}, y la primera palabra código asignada "10" se obtiene siguiendo los dos nodos superiores del árbol.

El grupo de palabras código que resultan del procedimiento anterior, tienen la característica de que ninguna de las palabras generadas es idéntica a algún prefijo de otra palabra más larga.

Esta última característica de los códigos Huffman, hace que las palabras código generadas , al arrivar al receptor, puedan ser decodificadas secuencialmente.[6]

3.7 BIBLIOGRAFIA

1.- Ahmed, N., Natarajan, T. and Rao, K.R., Discrete Cosine Transform, IEEE Transactions on Computers, January 1974.

2.- Andrews, C.H., Tutorial and Selected Papers in Digital Image Processing, IEEE Computer Society, 1978.

3.- Berger, T., Rate Distortion Theory, Prentice-Hall, 1971.

4.- Chen, W., Smith, H., and Fralick, S.C., A Fast Computational Algorithm for the Discrete Cosine Transform, IEEE Transactions on Communications, vol. COM-25, No. 9, September 1975.

5.- Elliott, D.F., Handbook of Digital Signal Processing, Academic Press, 1987.

6.- Fano, R.M., Transmission of Information, The M.I.T. Press, 1961.

7.- Gi Lee, B., A New Algorithm to Compute the Discrete Cosine Transform, IEEE Transactions on Acoustics, Speech and Signal Processing, vol. ASSP-32, No. 6, December 1984.

8.- Hamidi, M., Pearl, J., Comparation of the Cosine and Fourier Transform of Markov-1 Signals, IEEE Transactions on Acoustics, Speech, and Signal Processing, October 1976.

9.- Hsieh, S.H., A Fast Recursive Algorithm for Computing the Discrete Cosine Transform, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-35, No. 10, October 1987.

10.-Huchim, D.A., Procesamiento de Señales Bidimensionales para Transmisión-recepción, Tesis Profesional, Instituto Tecnológico de Puebla, 1989.

11.-Jain, A.K., Image Data Compression: A Review, Proceedings of

the IEEE, vol. 69, No. 3, March 1981.

12.-Kitajima, H., A Simetric Cosine Transform, IEEE Transactions on Computation C-29, 1980.

13.-Madihally, J., Narasimha and Peterson, A.M., On the Computation of the Discrete Cosine Transform, IEEE Transactions on Communications, vol. COM-26, No. 6, June 1978.

14.-Modestino, J.W., Daut, D.G., Vickers, A.L., Combined Source-Channel Coding of Images Using the Block Cosine Transform, IEEE Transactions on Communications, vol. COM-29, No 9, September 1981.

15.-Oppenheim, A.V., Schafer, R.W., Digital Signal Processing, Prentice-Hall Signal Processing Series, 1975.

16.-Rabiner, L.R., Schafer, R.W., Digital Processing of Speech Signals, Prentice-Hall Signal Processing Series, 1978.

17.-Shanmugan, K.S., Comments on "Discrete Cosine Transform", IEEE Transactions on computers, July 1975.

18.-Venkat, D., Rao, K.R., DPCM Coders with Adaptive Prediction for NTSC Composite TV Signal, IEEE Transactions on Communications, Vol COM-28, No. 7, July 1980.

19.-Wang, Z., On Computing the Discrete Fourier and Cosine Transforms, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-33, No. 4, October 1985.

20.-Wang, Z., and Hunt, B.R., Comparative Performance of Two Different Versions of Discrete Cosine Transforms, IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-32, 1984.

CAPITULO IV DESARROLLO Y SIMULACION DE ALGORITMOS PARA COMPRESION VIDEO.

Como se comentó en el capítulo I, el paso previo a todo proceso de codificación es la conversión analógico-digital, con una adecuada frecuencia de muestreo, para obtener la representación número rica de la imagen.

Las imágenes con que se contó para simular los esquemas de compresión se pueden representar por un arreglo de 256 x 256 pixeles, donde cada elemento de la imagen puede tomar uno de los 256 posibles niveles de gris, en el rango de 0 a 255, ó en correspondencia de negro a blanco (cada pixel tiene una resolución de ocho bits).

A diferencia de un monitor de televisión, donde las tonalidades van desde el negro, tonos de gris hasta el blanco, un monitor de computadora VGA únicamente despliega 16 tonos de gris (en modo no entrelazado), pero visualmente resultan suficientes para desplegar una imagen mono-cromática de buena calidad. En la figura 4.1 se muestra el proceso de formación de la imagen digital.[7]

En la parte superior izquierda de la figura se tiene la imagen original, a tráves de un proceso de muestreo, se obtiene la representación continua de cada uno de los puntos de la imagen en un tiempo t_m (figura en la parte superior derecha). Cada pixel se procesa analógicamente (no siempre es necesaria esta parte) y se cuantiza, para finalmente obtener un valor número ico que representa el nivel discreto del pixel en el tiempo de muestreo t_m. El total de los valores obtenidos es la representación número ica de la

imagen, y generalmente se guarda en un archivo para procesos posteriores.

Como esquema de comparación adicional, en este trabajo se simulo un codificador PCM a 6,4,3 bits/muestra. Los demás esquemas simulados fueron: modulador diferencial por pulsos codificados (DPCM) de primero, segundo y tercer orden, a 4, 3, 2 bits/muestra; transformada coseno discreta con cuantización uniforme simple, y una de las técnicas de compresión de imágenes, que parece ser la indicada a convertirse en norma mundial: la transformada coseno discreta adaptable (ADCT), utilizando el modelo del sistema visual humano (HVS). Esta última técnica de compresión en realidad es un esquema hibrido que engloba transformada coseno discreta, arreglo de componentes transformados en zig-zag (figura 4.6), normalización a tráves de la función del sistema visual humano (HVS), cuantización, y una codificación reversible y conjunta entre Huffman y corridas de ceros (Run-length). Como complemento a este último esquema, la DCT se implementó via la FFT para hacerla rápida.[1,5,6,7]

Para todos los esquemas simulados, se obtuvieron gráficas en amplitud-tiempo, para visualizar el comportamiento de la señal temporal reconstruida. Adicionalmente a las imágenes reconstruidas, se obtuvieron las imágenes diferencia, esto es, la imagen resultado de efectuar la diferencia entre la imagen original y reconstruida.

En algunas imágenes diferencia los valores de los pixeles resultantes , dado su pequeño valor, tuvieron que ser escalados por un factor número mayor a la unidad para poder hacerlos visibles en el monitor de la computadora. Dentro de las evaluaciones objetivas se utilizó como criterio de distorsión, la relación señal a ruido (SNR).[2,8]

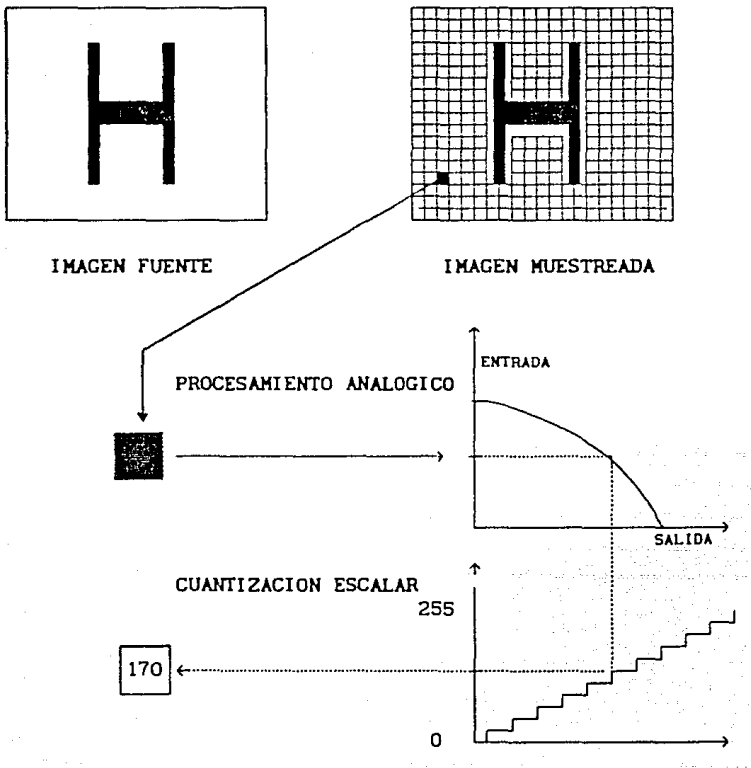


Figura 4.1 Formación de una imagen digital.

4.1 ESQUEMA PCM

Usualmente, en los sistemas PCM, para obtener una buena calidad de video en las transmisiones se códifica a 8 bits/muestra, sin embargo, cuando es necesario realizar varias conversiones analógico/digital, digital/analógico es preferible

utilizar 9 bits/muestra, con el fin de conservar video de buena calidad.[2]

El ruido de cuantización, el cuál engloba a los tipos de ruido mostrados en la figura 3.2, es el principal causante del deterioro de las señales en un esquema PCM, por esto, es necesario mantenerlo dentro de ciertos límites, que van a depender de la aplicación dada al video reconstruido (tabla 5.1).

En este trabajo, PCM se simulo únicamente para un cuantizador uniforme.

Para el cálculo de la relación señal a ruido, tomando la señal video al 70% del valor pico a pico de la señal de entrada, se usa la expresión:[4]

$$SNR = 6b + 7.7 \qquad (4.1)$$

donde: b = número de bits usados en la cuantización.

Así se obtuvo la siguiente tabla:

Número de bits	SNR teórica	SNR cálculada
6	43.7 dB	31.05 dB
4	31.7 dB	22.14 dB
3	25.7 dB	17.36 dB
2	19.7 dB	9.85 dB

Tabla 4.1 SNR esquema PCM

Independientemente de las diferencias mostradas entre la SNR teórica y cálculada, debidas principalmente a la suma de 7.7 de la expresión 4.1, se puede ver como la relación señal a ruido disminuye rápidamente conforme se van utilizando menos bits para la codificación de las imágenes. Visualmente, el deterioro

comienza a ser notable en los contornos, extendiendose gradualmente al resto de la imagen.

Como ejemplo, en la figura 4.2, se muestra el comportamiento de una señal cuantizada de manera uniforme, con R = 3 bits/muestra. En el siguiente capítulo, se muestra la imagen reconstruida correspondiente.

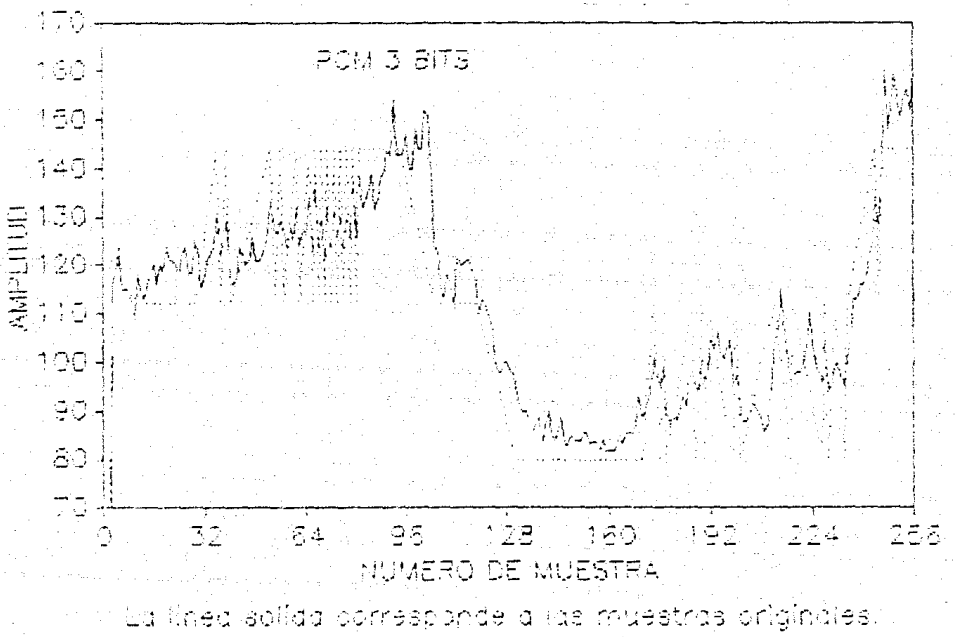


Figura 4.2 Señal temporal PCM para una línea de imagen de 256 pixeles, R = 3 bits/muestra.

4.2 ESQUEMA DPCM.

A diferencia de un codificador PCM, un esquema DPCM no

transmite la señal presente directamente al receptor, sino que previo a la transmisión, cálcula la diferencia entre la muestra actual x(n), y el valor de predicción $\hat{x}(n)$, la cuantiza, y la transmite usando un menor número de bits para una calidad de video equivalente.[5,7,8]

Como ya se vió en el capitulo tres, uno de los elementos que influyen sustancialmente en el desempeño y complejidad del esquema DPCM es el predictor. En pruebas con predictores de diferente orden, se ha encontrado que el incremento del orden en el predictor no dá un mejoramiento apreciable en la calidad de la imagen reconstruida, y con tres o cuatro muestras de predicción se obtienen resultados satisfactorios.[5,7]

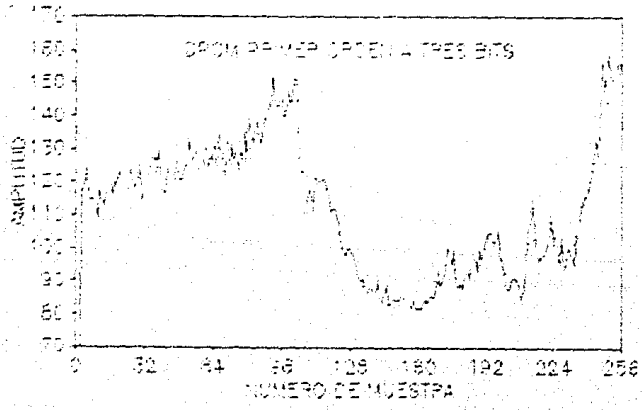
ESQUEMA DPCM						
	Primer orden [bits]		Segundo orden [bits]		Tercer orden [bits]	
	2	3	2	3	2	3
Varianza de la se$al de entrada	7.178 bits	7.178 bits	7.178 bits	7.178 bits	7.178 bits	7.178 bits
Varianza del error de predicción.	1.294 bits	1.288 bits	1.613 bits	1.606 bits	1.196 bits	1.169 bits
Varianza del error de cuantización.	.0427 bits	.0300 bits	.0367 bits	.0248 bits	.0333 bits	.0214 bits
SNR del cuantizador.	30.29	42.88	43.87	64.68	35.86	54.64
Ganancia del predictor.	5.544	5.572	4.449	4.467	5.999	6.137
SNR total.	22.25 dB	23.78 dB	22.90 dB	24.60 dB	23.32 dB	25.25 dB

Tabla 4.2 Comportamiento de los esquemas DPCM de primero, segundo y tercer orden.

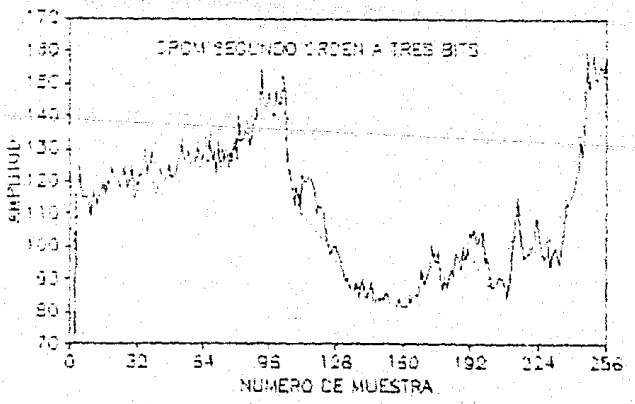
Comparando las tablas 4.1 y 4.2, se puede ver como a diferencia de un esquema PCM, cuando se disminuye el número de bits para codificar la imagen, el esquema DPCM, muestra una relación señal a ruido más alta y con valores de poca variación.

Aunque la relación señal a ruido parece relativamente baja, el mejoramiento visual en la calidad de la imagen reconstruida es muy grande para esquemas DPCM, en relación a PCM, para valores de SNR similares.

En la figura 4.3 se muestran las señales de linea para R = 3 bits, con predictores de primero, segundo y tercer orden.



La linea solida corresponde a las muestras originales.



La linea solida corresponde a las muestras originales

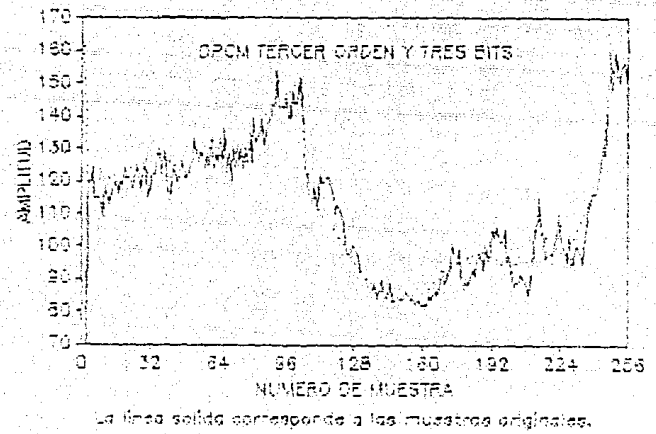


Figura 4.3 Señales temporales DPCM para una línea de imagen de 256 pixeles.

4.3 ESQUEMA DCT CON HVS.

4.3.1.-TENDENCIAS EN LA ESTANDARIZACION DE ALGORTIMOS DE COMPRESION.

En 1982, la organización sobre normas internacionales (International Standars Organization "ISO") creo un grupo especial "WG8" (Working Group 8), para comenzar a trabajar sobre la identificación de esquemas óptimos para la codificación de imágenes de video, gráficas e imágenes fotográficas. En 1985, el Consejo Consultivo Internacional en Telegráfo y telefonía (CCITT), formó un grupo para caracterizar los requisitos de los diferentes servicios en comunicación de imágenes.

Así, en 1986, la ISO y CCITT convinieron en formar el grupo conjunto de expertos fotográficos (JPEG), bajo el auspicio del WG8, con la tarea de seleccionar un algoritmo de compresión de alto desempeño, costo de implantación bajo, y con la capacidad de comprimir imágenes mono-crómaticas y en color.

Inicialmente, doce esquemas fueron seleccionados para someterlos a prueba, estos incluían: codificación predictiva, transformada coseno discreta, codificación de bloque truncado, cuantización vectorial y combinaciones de estas técnicas. Después de las evaluaciones técnicas y subjetivas, tres esquemas fueron seleccionados para pasar a la etapa final: transformada coseno discreta adaptable (ADCT) del grupo ESPRIT, codificación predictiva por IBM y codificación de bloque truncado por Japón. [6,7]

En la etapa final se hizo una revisión en detalle de los algoritmos contendientes, dejando en claro que el algoritmo ADCT producía imágenes de mejor calidad que los otros dos esquemas, siendo seleccionado como norma mundial.

Se espera que en lo que resta de este año, o principios del siguiente se publique la norma completa. [6,7]

4.3.2.- DISEÑO DEL ESQUEMA DCT CON HVS.

El diagrama a bloques de este esquema se muestra en la figura 4.4.[1,7]

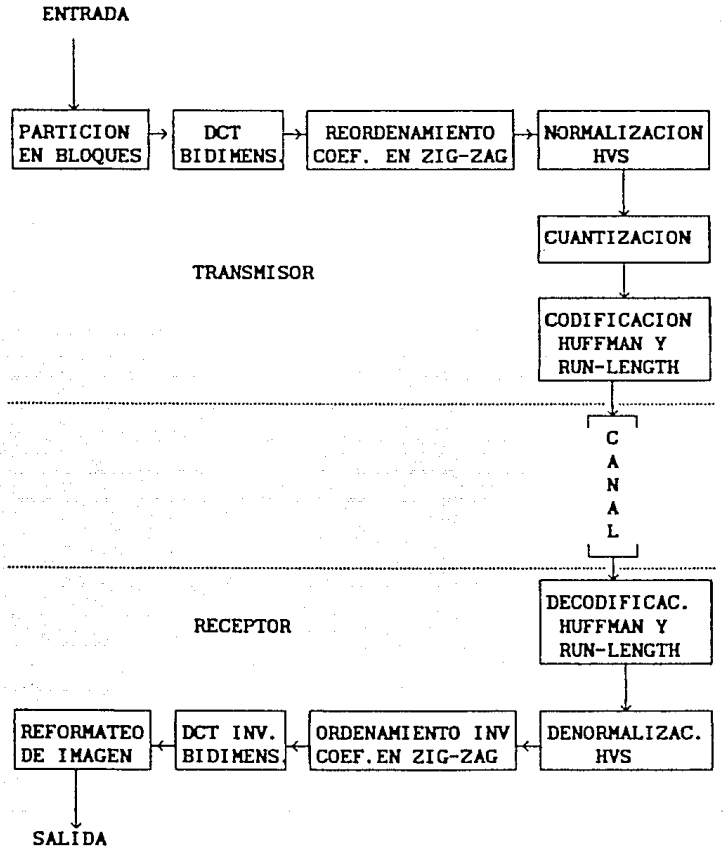


Figura 4.4 Diagrama a bloques de un esquema DCT con HVS.

La imagen utilizada como ejemplo en este trabajo, "Teresa", tiene la distribución de muestras que se ve en el histograma de la

figura 4.5.

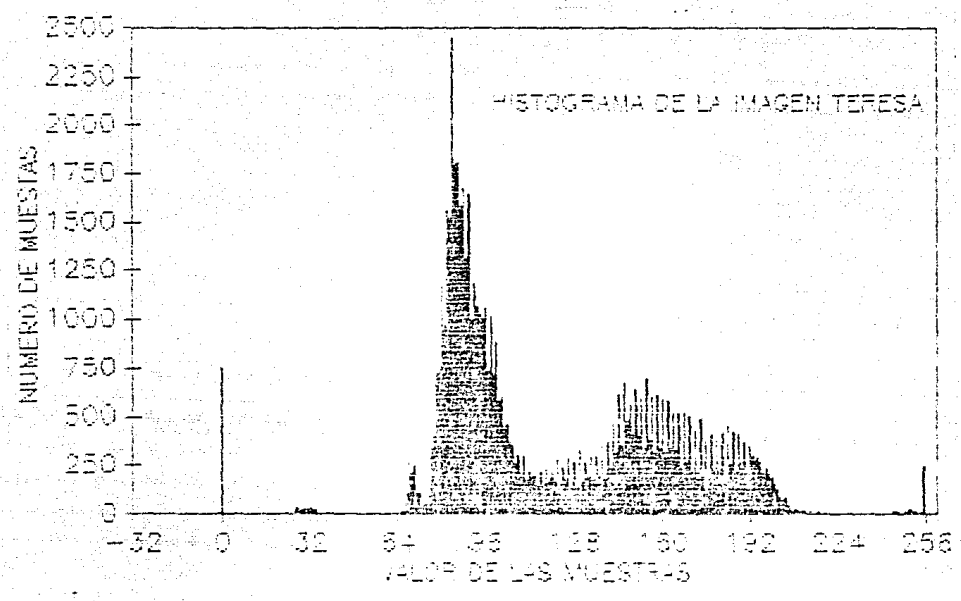


Figura 4.5 Histograma de la imagen Teresa.

Para la implantación de este esquema, se siguieron los siguientes pasos:

Paso 1.- La imagen original se particiona en bloques de 8 x 8 pixeles, cada bloque es transformado por la DCT de manera independiente.

Paso 2.- Se realiza la transformada coseno discreta sobre el bloque de sesenta y cuatro datos.

Paso 3.- Los coeficientes bidimensionales transformados se arreglan en un nuevo vector unidimensional, utilizando el reordenamiento zig-zag que se muestra en la figura 4.6.

Esto es equivalente a arreglar los coeficientes transformados en orden decreciente conforme a su contenido de energía. El objetivo de este arreglo es hacer que los coeficientes con bajo contenido de energía, queden contiguos, para que en un subsecuente proceso de cuantización se produzcan cadenas de ceros que puedan ser codificadas eficientemente por un algoritmo de corridas de ceros.

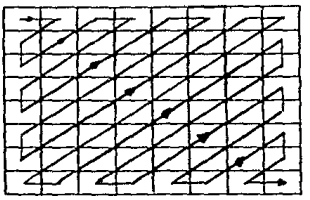


Figura 4.6 Secuencia de arreglo en zig-zag.

Paso 4.- Dado que el primer coeficiente, después del arreglo zig-zag (el que se encuentra en la esquina superior izquierda despues de la transformación DCT), contiene energía para producir una brillantez al doble del promedio total de los 8 x 8 coeficientes del bloque. Este coeficiente se conoce como DC, en analogía de una corriente de directa, porque cuando se reconstruye el bloque original, por aplicar la DCT inversa, distribuye su energía uniformemente en el bloque, a diferencia del resto de los coeflentes, AC.

Y como el valor de este coefiente afecta significativamente la calidad de la imagen reconstruida, este se codifica usando un esquema DPCM de primer orden a 9 bits, que utiliza como muestra de predicción el coeficiente de DC del bloque anterior.

Paso 5.- El resto de los coeficientes, conocidos como valores AC, se cuantizan de acuerdo con su importancia visual, usando la función de sensitividad de contraste HVS. Esta función se muestra en la figura 4.7. Escalando los valores de esta función, se puede variar la tasa de transmisión.

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

Figura 4.7 Matriz de normalización de luminancia en DCT HVS.

Paso 6.- Los coeficientes normalizados, son uniformemente cuantizados, por redondear los valores al entero más cercano. La normalización previa sirve para escalar de manera efectiva el cuantizador utilizado, de acuerdo a la respuesta del sistema visual humano.

Paso 7.- La codificación de los coeficientes de AC , diferentes de cero,se lleva a cabo por identificar, a cuál de las ocho categorías, de la tabla mostrada en la tabla 4.3, pertenece el coeficiente. Los valores en cada categoría k; k = 1, ,8, están en el rango de $(2^k,\ldots,2^{k+1}-1)$, o $(-2^{k+1}+1,\ldots,-2^k)$.

CATEGORIA	RANGO DE LOS COEFICIENTES DE AC.
1	-1,1
2	-3,-2,2,3
3	-7,...,-4,4,...,7
4	-15,...,-8,8,...,15
5	-31,...,-16,16,...,31
6	-63,...,-32,32,...,63
7	-127,...,-64,64,...,127
8	-255,...,-128,128,...,255

Tabla 4.3 Valores asignados a cada categoría de coeficientes de AC.

TABLA MODIFICADA DE HUFFMAN (TERESA 1.44 BITS/PIXEL)					
CORRIDAS DE CEROS	CATEGORIA	HISTOGRAMA DE FRECUENCIAS	BITS USAD.	CODIGO	BITS x NUMERO DE MUESTRAS
0	1	4592	2	00	9184
1	1	1410	3	110	4230
2	1	569	5	01010	2845
3	1	505	5	01111	2525
4	1	583	4	1111	2332
5	1	530	5	01110	2650
6	1	468	5	10111	2340
7	1	272	6	011001	1632
8	1	129	7	0110001	903
9	1	38	9	011000001	342
10	1	24	9	111000001	216
11	1	16	10	0110000111	160
12	1	45	8	11100111	360
13	1	56	8	11100001	448
14	1	47	8	11100110	376
15	1	33	9	111000000	297
0	2	2104	3	100	6312
1	2	342	5	11101	1710
2	2	95	7	1110001	665
3	2	114	7	1011011	798
4	2	121	7	1011010	847
5	2	62	8	10110011	496
0	3	1141	4	0100	4564
1	3	127	7	1011000	889
2	3	36	9	011000010	324
3	3	20	10	0110000110	200
0	4	567	5	01011	2835
1	4	41	9	011000000	369
0	5	270	6	011010	1620
0	6	94	7	1110010	658
0	7	65	8	10110010	520
fin de bloque todas las otras		1024	4	1010	4096
combinaciones		264	6	011011	1584
total:		15804			59327
BITS POR MUESTRA		bits para el signo			14780
		bits para el componente de DC			9216
$\frac{94421}{65536} = 1.44075$		bits para representar magnitud en todas las otras combinaciones			11098
		Total de bits transmitidos			94421

Tabla 4.4 Ejemplo del codificador Huffman utilizado en el esquema DCT HVS.

Paso 8.- Una vez identificada la categoría, en la tabla 4.4, se selecciona el código de Huffman correspondiente. Estos códigos fueron obtenidos siguiendo los pasos descritos en el sección 3.6, del capítulo 3. Por ejemplo, si el coeficiente tiene un valor de 1, y tiene 6 ceros anticipándole, de acuerdo a la tabla su código será: 10111.

Paso 9.- Enseguida, es necesario especificar el signo de la mágnitud transmitida. Para este trabajo se utilizó la asignación de signo de la tabla 4.5.

	SIGNO DEL COEFICIENTE	
	NEGATIVO	POSITIVO
CODIGO	0	1

Tabla 4.5 Código del signo de los coeficientes.

Paso 10.- Finalmente es necesario transmitir el código de la mágnitud. Es obvio que para la categoría 1 no es necesario transmitir información concerniente a la mágnitud (sólo se tiene una mágnitud).

Para este trabajo, en el resto de las categorías, se utilizó la codificación de mágnitud mostrada en la tabla 4.6. Se observa como conforme se incrementa el valor de la categoría, también se incrementa el número de bits para codificarla. No se muestra la tabla completa, pero el completarla es fácil. El código asignado a la mágnitud es de longitud fija, dentro de cada una de las categorías.

Como ejemplo de una codificación completa; sea una secuencia de coeficientes ordenados en zig-zag: 2 0 0 0 -3 4 2 0, codificando el -3 resulta: (3,2), donde 3 es el número de ceros anticipandole y 2 es la categoría.
El código de Huffman para esta combinación es: 1011011 (categoría 2 precedida por 3 ceros), 0 (signo negativo), 1 (magnitud 3).

CODIFICACION DE LA MAGNITUD		
CATEGORIA	VALOR DEL COEFICIENTE	CODIGO ASIGNADO
2	2	0
	3	1
3	4	00
	5	01
	6	10
	7	11
4	8	000
	9	001
	10	010
	11	011
	12	100
	13	101
	14	110
	15	111
5	16	0000
	17	0001
	18	0010
	19	0011
	20	0100
	21	0101
	22	0110
	23	0111
	24	1000
	25	1001
	26	1010
	27	1011
	28	1100
	29	1101
	30	1110
	31	1111
6	32	00000
	33	00001
	34	00010
	35	00011
	36	00100
	37	00101
	38	00110
	39	00111
	40	01000
	41	01001
	42	01010
	43	01011
.	.	.
.	.	.
.	.	.

Tabla 4.6 Codificación de la mágnitud.

Paso 11.- Una vez codificados todos los coeficientes diferentes de cero, es necesario insertar una palabra especial que le indique al receptor que se terminó de codificar el bloque, en la tabla 4.4, se ve que el código para el fin de bloque usado en este trabajo es: 1010.

Paso 12.- Todo el proceso anterior se invierte en el receptor, con el fin de recuperar la imagen original. En este esquema, el deterioro de la imagen reconstruida se controla en la etapa de cuantización, y esta varia de acuerdo con la tasa de transmisión, la cuál es controlada escalando la función de normalización de la figura 4.7.

En las siguientes figuras se muestra como se comporta un bloque de la imagen "Teresa", a tráves de las diferentes etapas del algoritmo. La etapa de normalización HVS y cuantización se realizan en un sólo paso.

84	84	85	87	87	88	87	86
84	84	82	84	84	82	84	85
83	87	88	84	84	85	86	83
83	83	87	88	85	86	86	85
82	83	85	86	84	85	88	86
86	85	87	89	84	85	85	83
83	86	85	84	86	88	84	86
85	90	84	85	88	84	86	87

a

170.374985	-0.671459	0.310930	0.977012	0.687500	0.602320	0.750652	-0.027039
-0.531556	-0.267076	1.107300	-0.326425	-0.315294	-0.148953	0.346908	-0.552107
-0.667296	-0.088724	0.599803	-0.355823	-0.140526	-0.691522	-0.241497	0.194686
-0.573130	-0.096489	0.501250	0.123571	-0.133253	0.121643	-0.179282	0.047529
-0.375991	0.096667	0.505671	0.430183	0.312500	-0.395522	-0.412405	-0.733938
0.194268	0.610943	-1.145166	0.319444	0.159483	-0.979377	0.354416	-0.427296
-0.467746	0.091127	-0.553398	0.339766	-0.346454	0.733488	-0.197303	0.005067
-0.139659	0.261693	0.050013	-0.270241	0.137736	-0.103897	0.136225	0.222883

b

170.374985 -0.671459 -0.531556 -0.667235 -0.267076 0.310930 0.077012 0.197300

-0.988724 -0.578130 -0.375001 -0.996429 0.509803 -0.326425 0.687500 0.602329

-0.815234 -0.355823 0.501350 0.996667 0.104268 -0.467746 0.610943 0.505671

0.123571 -0.143506 -0.148953 0.750652 -0.927039 0.845908 -0.691622 -0.133253

0.430163 -1.145166 0.091127 -0.139959 0.251633 -0.553998 0.810444 0.312500

0.121543 -0.241497 -0.552107 0.194696 -0.179262 -0.395522 0.156463 0.333766

0.950013 -0.270241 -0.346454 -0.079377 -0.412405 0.047529 -0.739398 0.354416

0.733488 0.137786 -0.109387 -0.197303 -0.427236 0.035067 0.136225 0.222983

c)

0.316406 -1.000000 -1.000000 -1.000000 -0.000000 0.000000 0.000000 0.000000

-0.000000 -1.000000 -0.000000 -0.000000 0.000000 -0.000000 0.000000 0.000000

-1.000000 -0.000000 1.000000 0.000000 0.000000 -0.000000 0.000000 0.000000

0.000000 -0.000000 -0.000000 0.000000 -0.000000 0.000000 -0.000000 -0.000000

0.000000 -1.000000 0.000000 -0.000000 0.000000 -0.000000 0.000000 0.000000

0.000000 -0.000000 -0.000000 0.000000 -0.000000 -0.000000 0.000000 0.000000

0.000000 -0.000000 -0.000000 -0.000000 -0.000000 0.000000 -0.000000 0.000000

0.000000 0.000000 -0.000000 -0.000000 -0.000000 0.000000 0.000000 0.000000

d)

m= 100

DC--> 0.316406 00 0 00 0 00 0 01110 0 10111 0 110 1 11100110 0 1010

e)

0.316406 -1.000000 -1.000000 -1.000000 0.000000 0.000000 0.000000 0.000000

0.000000 -1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

-1.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 -1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

f)

170.621094 -1.000000 -1.000000 -1.000000 0.000000 0.000000 0.000000 0.000000

0.000000 -1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

-1.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 -1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

g)

170.621094 -1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

-1.000000 0.000000 0.000000 0.000000 -1.000000 0.000000 0.000000 0.000000

-1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

-1.000000 0.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 -1.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

h)

82	84	84	84	86	86	85	86
84	85	85	85	86	85	84	85
84	84	86	86	85	84	85	85
82	83	86	87	84	84	87	87
83	83	87	88	85	85	87	87
84	85	87	87	85	85	86	86
85	86	86	86	87	86	85	86
84	85	85	85	88	87	86	87

i)

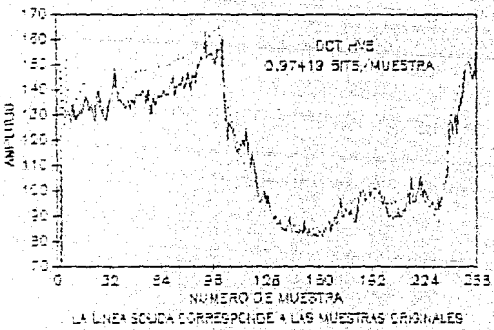
Figura 4.8 Transformación de un bloque en DCT HVS.
a) Bloque original de la imagen Teresa. b) Bloque transformado con la DCT. c) Bloque reordenado en zig-zag. d) Bloque normalizado y cuantizado. El componente de DC es códificado con DPCM de primer orden. e) Bits transmitidos, después de la codificación conjunta Huffman y corridas de ceros. f) Bloque reconstruido en el receptor, aplicando conjuntamente decodificación de Huffman y corridas de ceros. g) Bloque reconstruido, después de la desnormalización HVS. h) Bloque reordenado, aplicando zig-zag inversa. i) Bloque reformado, después de la DCT inversa.

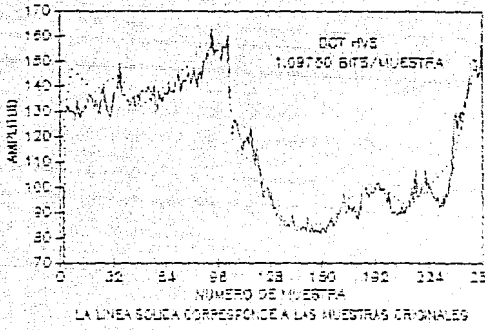
Comparando las tablas 4.1, 4.2, y 4.7, se observa como el esquema DCT HVS tiene un desempeño muy superior a los otros dos esquemas, aunque por otro lado implica una mayor complejidad para su implantación. En el siguiente capítulo se verá, como para este esquema, con incrementos en fracciones de bits/muestra, la imagen reconstruida mejora notablemente.

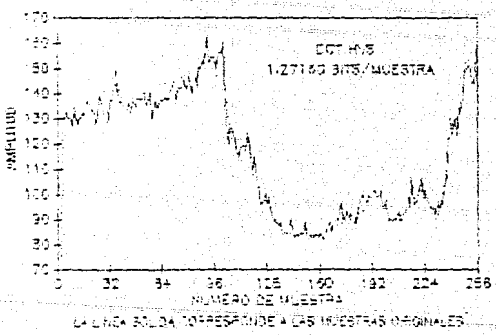
BITS/MUESTRA	SNR CALCULADO [dB]
0.97419	19.1534339
1.09750	22.6454350
1.27160	24.1069510
1.44075	25.3315430

Tabla 4.7 SNR esquema DCT HVS.

En la figura 4.9 se muestran las señales temporales de línea para este esquema, es obvio el superior desempeño de este esquema con respecto a PCM y DPCM.







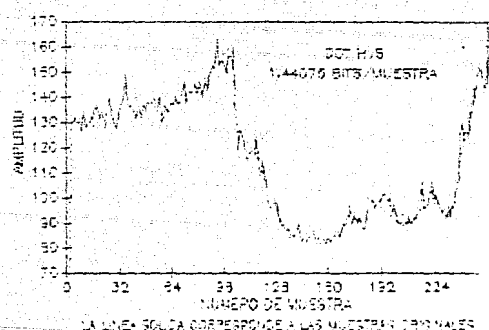


Figura 4.9 Señales temporales DCT HVS para una línea de imagen de 256 pixeles.

4.4.- BIBLIOGRAFIA.

1.- Ahmed, N., Natarajan, T., and Rao, K.R., Discrete Cosine Transform, IEEE Transactions on Computers, January 1974.

2.- Andrews, C.H., Tutorial and Selected Papers in Digital Image Processing, IEEE Computer Society, 1978.

3.- Gi Lee, B., A new Algorithm to Compute the Discrete Cosine Transform, IEEE Transactions on Acoustics, Speech and Signal Processing, vol. ASSP-32, No. 6, December 1984.

4.-Hitomi, M., Hashimoto, H., Hatori, Y., Quality of Band-Compressed TV Services, IEEE Communications Magazine, October 1988.

5.- Jain, A.K., Image Data Compression: A Review, Proceedings of the IEEE, vol. 69, No. 3, March 1981.

6.- Mitchell, J.L., and Pennebaker, W.B., Standardization of Color Image Data Compression II. Progressive Coding, Electronic Imaging'89 East, Boston, October 2-5, 1989.

7.- Rabbani, M., Eastman Kodak Co., SPIES's Short Course Notes, San Diego, California, USA, July 1990.

8.- Rabiner, L.R., Schafer, R.W., Digital Processing of Speech Signals, Prentice-Hall Signal Processing Series, 1978.

CAPITULO V RESULTADOS Y APLICACIONES

En este trabajo, se utilizó como criterio de distorsión objetivo la relación señal a ruido, con el fin de conocer la cantidad de distorsión introducida en el proceso de codificación. Sin embargo, dada la escasa correlación en el mejoramiento visual de una imagen con relación al aumento de la relación señal a ruido, y tomando en consideración que el usuario final es una persona, es recomendable hacer las evaluaciones de manera subjetiva (comparando las imágenes reconstruidas), y sólo tomar las evaluaciones objetivas como referencia de ejecución del algoritmo; dado que estas no toman en cuenta los parámetros de observación del sistema visual humano.[1]

Los resultados de cada esquema comprenden:

1.- Gráficas temporales de línea (capítulos IV y V).
2.- Gráficas de error temporal (diferencia entre la señal de una línea original y una reconstruida.
3.- Imágenes reconstruidas a diferentes valores bits/muestra.
4.- Imágenes de error (diferencia entre la imagen original y la imagen reconstruida). Todas estas imágenes fueron escaladas para ser observables, dado el pequeño valor de los pixeles.
5.- Tablas de la relación señal a ruido (capítulo IV).

Cada uno de estos resultados dan una idea del comportamiento y limitaciones de los esquemas de compresión de imágenes. Aunque en esta tesis únicamente se muestran los resultados para una imagen, estos resultados fueron similares para otras imágenes diferentes.

Las imágenes fueron obtenidas por un adquisidor de imágenes fijas, hecho en un trabajo previo del mismo grupo, cuando este laboraba en el Instituto de Investigaciones Eléctricas, en Cuernavaca, Morelos.[7] Actualmente se cuenta con un adquisidor en tiempo real, mono-crómatico de la empresa Metrabyte.

5.1.- RESULTADOS PCM.

En la figura 5.1 se muestran las señales temporales de error absoluto para el codificador PCM a R = 2, 3, 4, 6 bits/muestra, se puede notar como la señal de error se incrementa notablemente conforme disminuyen los bits/muestra utilizados en el codificador. Por otro lado,para el codificador, la SNR cálculada (tabla 4.1) se incrementa de manera aproximadamente logaritmica en función del número de bits utilizados.

En las figuras 4.2 y 5.2, se observa el comportamiento de la señal reconstruida. Para PCM con R = 3 bits/muestra, se ve como el cuantizador trata, sin conseguirlo, de seguir la señal original. Las diferencias con respecto a la señal original son grandes, aún en las secciones con menos detalle.

Con 6 bits/muestra, figura 5.2, la señal reconstruida es más parecida a la señal original, únicamente para los detalles más finos el codificador se muestra incapaz de reconstruirlos, debido principalmente al tamaño del escalón Δ utilizado en el cuantizador.

Visualmente , tal como lo muestra la secuencia de imágenes de la figura 5.3, el mejoramiento subjetivo en la calidad de las imágenes reconstruidas sigue una secuencia lógica, en correspondencia al número de bits/muestra del codificador. En la imagen 5.3.d, se observa como los detalles más finos no son recuperados, corroborado por la señal temporal de la figura 5.2.

Las imágenes diferencia, resultado de hacer la diferencia absoluta entre la imagen original y las imágenes reconstruidas (figura 5.4), nos muestran la cantidad de información pérdida en

la codificación. La cara, que es una zona de grandes contrastes, presenta las pérdidas más grandes, sobre todo en los contornos de la misma.

Conforme se aumenta el número de bits/muestra en la codificación las pérdidas de información son menores, extendidas en toda la imagen, y menos marcadas en los contornos.

Es necesario aclarar que todas las imágenes diferencia fueron escaladas para hacerlas observables, en realidad los valores de las diferencias son relativamente pequeños, menores a siete. Sin embargo, el monitor VGA únicamente puede desplegar 16 tonos de gris, con cada tono agrupando 16 valores. De esta manera, el primer tono agrupa los valores de 0 a 15, el segundo de 16 a 31, el tercero de 32 a 47, y así sucesivamente.

Por esto, dado que todos los valores diferencia caerían en el primer rango de valores, la imagen diferencia presentaría un sólo tono, el más cercano al negro, y esto haría invisibles los errores de las imágenes reconstruidas. Para resolver este problema, los valores diferencia se multiplican por 16 , lo cuál equivale a escalarlos en los diferentes tonos de gris, resultando en una imagen diferencia resaltada.

5.2.- RESULTADOS COMPRESOR DPCM.

Los esquemas simulados para el compresor DPCM fueron de primero ($\hat{x} = A$), segundo ($\hat{x} = [A + D] / 2$), y tercer orden ($\hat{x} = A + [C - B] / 2$) (ver figura 3.3), a 2 y 3 bits/muestra.[3,4]

Como era de esperarse, el comportamiento de este esquema es mejor que PCM. En la figura 5.5 se muestran las señales temporales , sobre una línea de 256 muestras, reconstruidas y originales para R = 2, 3 bits/muestra con codificadores de primero, segundo y tercer orden.

En los esquemas DPCM de primer orden a dos y tres bits, se observa como el ruido de sobrecarga es considerable, debido a que el predictor utiliza una sola muestra para su trabajo.

En la figura 5.7 se ve como el ruido de sobrecarga, en este

esquema , visualmente produce franjas de diferentes tonos de gris, haciendo que la calidad subjetiva de la imagen sea baja. La figura 5.8, de las imágenes diferencia, corrobora la gran cantidad de información pérdida por este esquema, sobre todo en el lado izquierdo de la imagen y la cara.

No se denota una mejora visual apreciable con el esquema codificando a dos y tres bits/muestra. En la tabla 4.2 se tiene una mejora en la relación señal a ruido de poco más de 1.5 dB entre codificadores de dos y tres bits/muestra, que subjetivamente no es observable; aunque si muy superior al de un sistema PCM usando el mismo número de bits/muestra.

El esquema DPCM de segundo orden, presenta una mejora sustancial ante el ruido de sobrecarga, esto se muestra en la figura 5.5, y aunque de acuerdo con la tabla 4.2 la mejora del SNR en dB para el DPCM de segundo orden, con respecto al primer orden, en dos y tres bits es menor a uno, la mejora visual es enorme.

Esto se puede ver en las imágenes reconstruidas de la figura 5.7, las franjas de ruido de sobrecarga desaparecieron haciendo las imágenes reconstruidas más aceptables a la vista. En las imágenes diferencia de la figura 5.8, se nota como existen pérdidas de información en los contornos de alto contraste, aunque mucho menores que en el DPCM de primer orden.

En el cambio del número de bits/muestra, dos a tres, se observa una pequeña mejora en la calidad de la imagen reconstruida, sobre todo en los detalles finos.

Finalmente, el esquema DPCM de tercer orden, muestra en la figura 5.5 una respuesta al ruido de sobrecarga ligeramente superior al DPCM de segundo orden, en dos y tres bits/muestra.

La relación señal a ruido, presenta una mejora menor a 1 dB con respecto al DPCM de segundo orden, en dos y tres bits/muestra. Las imágenes recontruidas de la figura 5.7, para DPCM de tercer

orden, tampoco presentan franjas de sobrecarga, y comparadas con las obtenidas por el DPCM de segundo orden presentan, un poco más de detalle, sin hacerlas visualmente de mejor calidad.

Sin embargo, cuando se comparan las imágenes diferencia, de la figura 5.8, entre DPCM de segundo y tercer orden, se observa que aunque el codificador DPCM de tercer orden presenta mayores pérdidas de información que DPCM de segundo orden, este distribuye las pérdidas de manera más equilibrada en toda la imagen y no únicamente en las zonas de alto contraste.

Por lo que respecta a las señales temporales de error de la figura 5.6, es necesario remarcar, como estas señales de error en el codificador DPCM de tercer orden, presentan un comportamiento de amplitud aproximadamente constante en toda la línea, redundando en pérdidas de información distribuidas a tráves de toda la imagen reconstruida, tal como se explicó en el párrafo anterior.

5.3.-RESULTADOS COMPRESOR DCT HVS.

Este esquema presenta un comportamiento muy diferente a los esquemas anteriores. En la figura 5.9 se muestran las señales temporales, sobre líneas de 256 pixeles, para R = 0.97 , 1.09, 1.27, y 1.44 bits/muestra. Para 0.97 bits/muestra, se nota como el esquema sigue la señal original con una cierta compensación, sin tomar en cuenta las variaciones de gran pendiente. La pérdida de información es muy superior al esquema DPCM, esto se puede ver al comparar las figuras de las señales de error temporales 5.10 y 5.6.

En la imagen reconstruida para 0.97 bits/muestra de la figura 5.11, se nota la gran cantidad de detalles pérdidos, así como la estructura a bloques de la imagen reconstruida. Este último efecto se observa en este codificador cuando se usan valores bits/muestra relativamente pequeños, siendo el efecto más notorio en las zonas de gran contraste debido a la pérdida de frecuencias altas.

Los coeficientes transformados $F_j(u,v)$ con valores pequeños de u y v, en la figura 3.11, representan los componentes de frecuencias bajas de la imagen, por el contrario, cuando los coeficientes $F_j(u,v)$ tienen valores grandes de u y v, se tienen las frecuencias altas, responsables de los detalles con más resolución de la imagen.[6]
La imagen diferencia 5.12, muestra como la información pérdida se encuentra distribuida en toda la imagen, aunque resaltada en los contornos.

Conforme aumentan en fracciones de unidad los valores bits/muestra R, las señales temporales reconstruidas presentan mayor fidelidad, la señal de error disminuye y las imágenes reconstruidas mejoran de calidad, eliminándose el efecto de bloques.

A 1.44 bits/muestra la imagen recontruida con DCT HVS presenta una calidad subjetiva equivalente a la imagen reconstruida con DPCM de tercer orden y tres bits/muestra, siendo también la relación señal a ruido ,de las dos imágenes reconstruidas, equivalente (tablas 4.1 y 4.7).

En la figura 5.13 se muestran dos imágenes DCT HVS reconstruidas con errores en la codificación de signo (5.13.a), y la codificación de magnitud (5.13.b). Se nota inmediatamente , en este caso, como el error en la transmisión que afectó el signo de la mágnitud visualmente, generó un bloque erróneo menos notable, que el del error cometidó en la mágnitud.

Sin embargo, no siempre los resultados de códigos erróneos en signo y mágnitud mantienen esta relación visual; para errores en la mágnitud, el efecto negativo del bloque reconstruido depende en lugar ocupado por el bit afectado dentro de la palabra código, así como su importancia, para el sistema visual humano (figura 4.7), dentro del arreglo de los componentes transformados.

En cambio, el efecto visual en el bloque reconstruido de un

error cometido en el signo, únicamente cambia los niveles de gris de los planos afectados por el componente erróneo.

Cuando, en este trabajo, se simuló el efecto de un error en la palabra del código Huffman_corridas de ceros el efecto fué catastrófico, y el programa correspondiente al decodificador no pudo reconstruir la imagen.

Inicialmente, el tiempo de ejecución del algoritmo era de aproximadamente 1 hora 45 minutos en una computadora personal AT. Con la incorporación del esquema descrito en la sección 3.5.2, para hacer la DCT rápida, el tiempo de ejecución final se redujo a 15 minutos, en una computadora personal AT, sin coprocesador y con un tiempo de acceso a disco duro de 28 milisegundos.

5.4.-APLICACIONES.

La codificación de alta eficiencia se aplica a diferentes tipos de transmisión de imágenes digitales, televisión de alta definición (HDTV), video-conferencia, video-telefóno, etc. La calidad de estas transmisiones varian de acuerdo a la tasa de bits asignada o en correspondencia al número de bits/muestra utilizados en la transmisión. [1,4]

La HDTV requiere una alta calidad, el sistema de codificación utilizado no debe causar deterioro perceptible en la calidad de la imagen reconstruida. Por esto, las tasas requeridas para HDTV oscilan entre 100 y 135 Mbits/seg.

Para video-conferencia las tasas de transmisión tienen rangos del orden de 32 a 45 Mbits/seg para movimiento completo sobre una televisión estandar, y de 2 Mbits/seg a 48 Kbits/seg para video de mivimiento limitado.

Los sistemas de video-telefóno, actualmente, manejan tasas de transmisión en el rango de 48 a 64 Kbits/seg.[1]

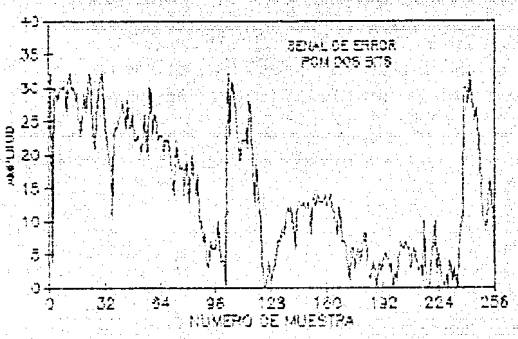
Consecuentemente con el decremento en las tasas de transmisión, la calidad de las imágenes reconstruidas presentan algún grado de deterioro, pero éste, siempre dentro de los limites dictados para el uso especifico del video.

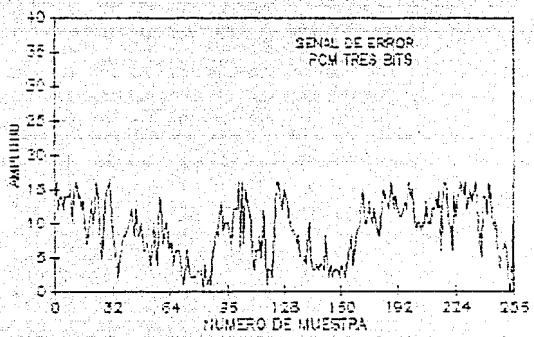
Las señales de televisión pueden ser transmitidas a una tasa de la tercera jerarquía digital de la CCITT, mientras se tenga una calidad aceptable en la red. La tasa estándar Europea, es de 34.368 Mbits/seg (537 canales de voz PCM a 64 Kbits/seg por canal), 44.736 Mbits/seg (672 canales de voz PCM) para los Estados Unidos y 44.736 Mbits/seg para el Japón.[2]

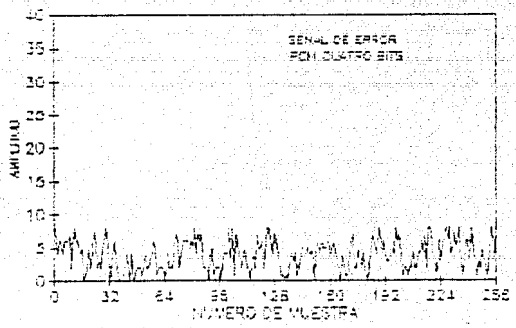
En la tabla 5.1 se muestran algunos ejemplos de servicios donde se aplica la codificación de la señal video.[1]

SISTEMAS DE CODIFICACION				
SEÑAL CODIFICADA	CARACTERISTICAS DE LOS SISTEMAS	TASAS DE TRANSMISION	APLICACION	NIVEL DEL PRODUCTO
TV	4 fsc de muestreo NTSC predicción adaptiva intra/inter-campo DPCM	45 Mb/s	Transmisión de televisión	experimental
	3 fsc de muestreo NTSC predicción adaptiva inter-campo DPCM	45 Mb/s	video-conferencia	comercial
	4 fsc de muestreo predicción intra -campo DPCM	32 Mb/s	video-conferencia y video-distribución	comercial
	13.5 MHz de muestreo predicción (intra/inter campo, intra-cuadro) DPCM	15 ~ 30 Mb/s	difusión	en práctica
HDTV	Filtrado espacio-temporal, predicción por extrapolación e interpolación	100~135 Mb/s	distribución	experimental
TV MOV. LIM.	Reemplazo condisional y DPCM en las áreas con movimiento	2 Mb/s	video-con ferencia	comercial
	Codificación híbrida DCT y codificación adaptable de escena	384~1544 Mb/s	video-conferencia	comercial
	DCT compensada en movimiento	64 Kb/s	video-conferencia	comercial
	predicción entrecuadro, con cuantización vectorial para las componentes de luminancia y crominancia	48 Kb/s	videofóno	comercial

Tabla 5.1 Ejemplos de los sistemas de codificación.







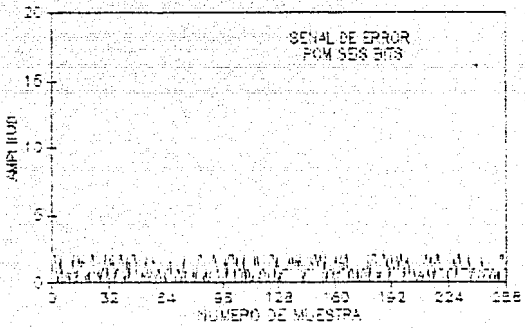
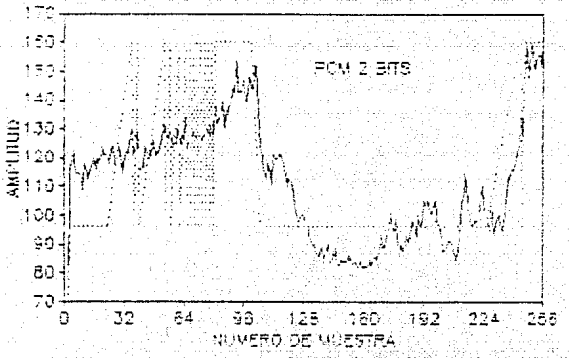
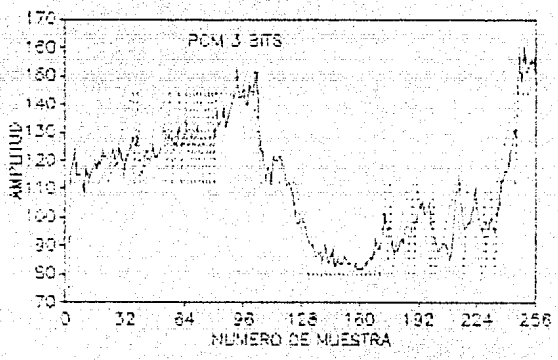


Figura 5.1 Señales temporales de error absoluto PCM.



La línea sólida corresponde a las muestras originales.



La línea sólida corresponde a las muestras originales.

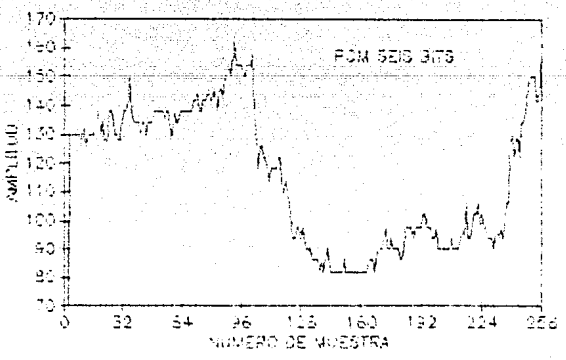
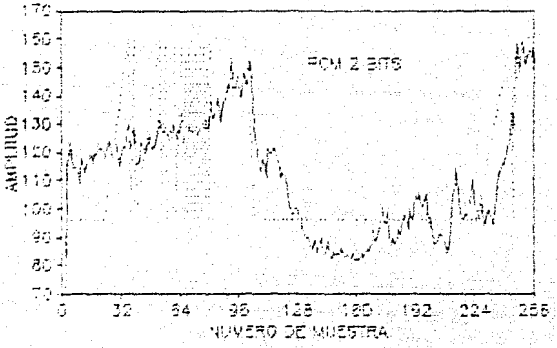
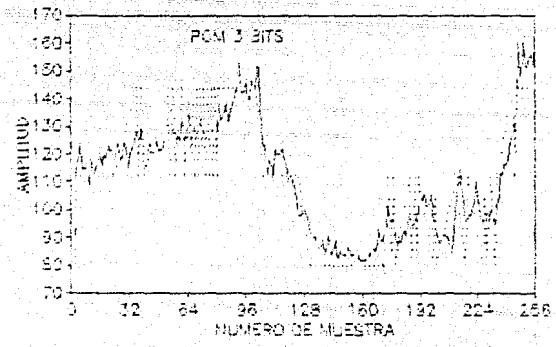


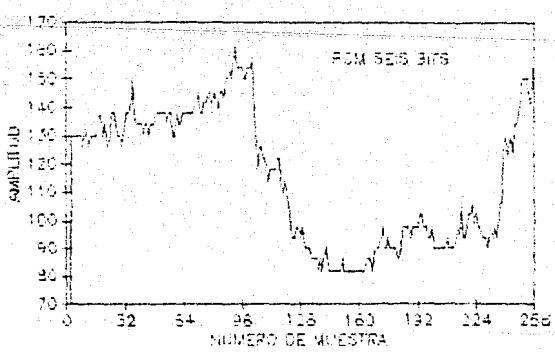
Figura 5.2 Señales temporales reconstruidas en PCM.



La línea sólida corresponde a las muestras originales.



La línea sólida corresponde a las muestras originales.



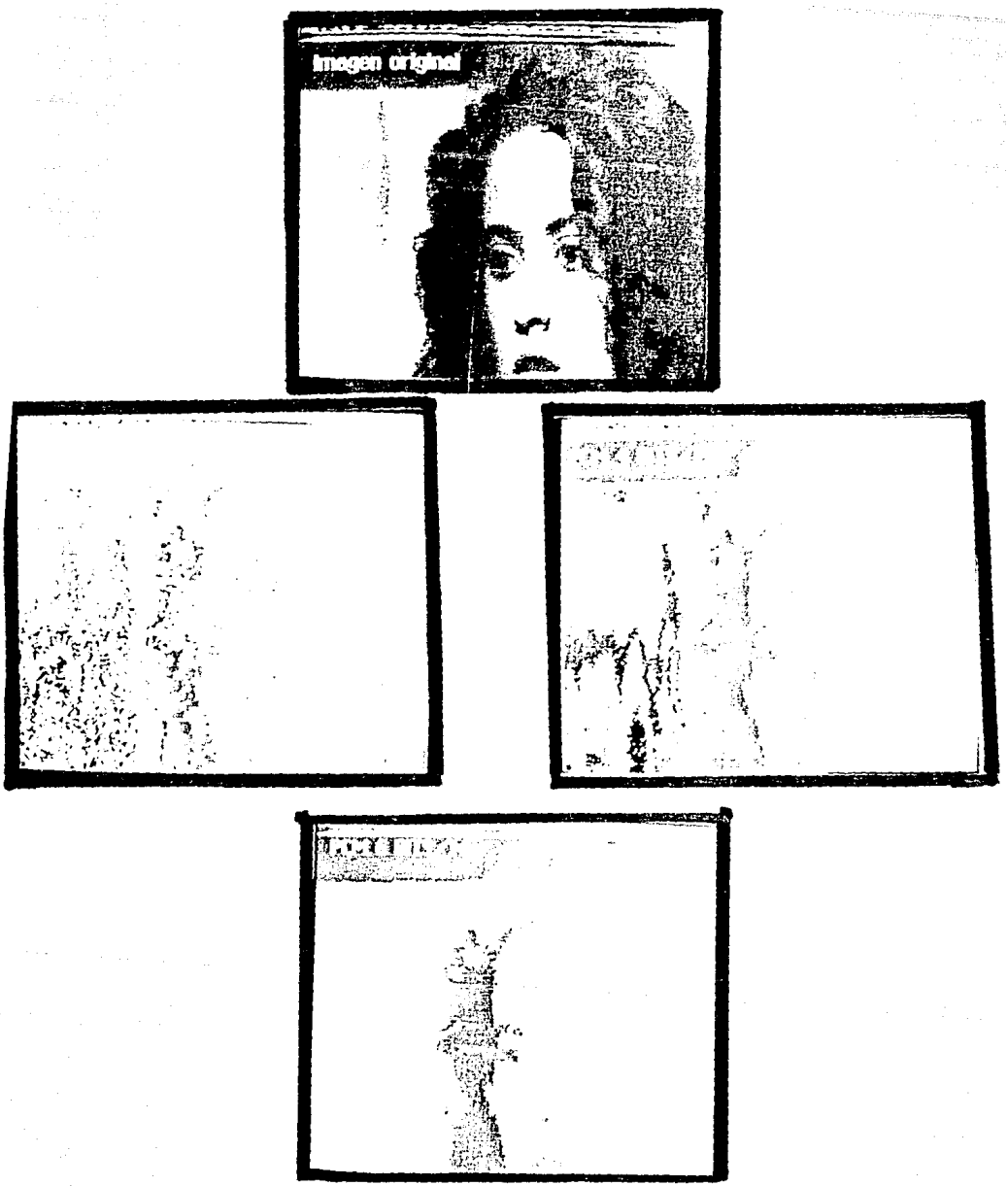
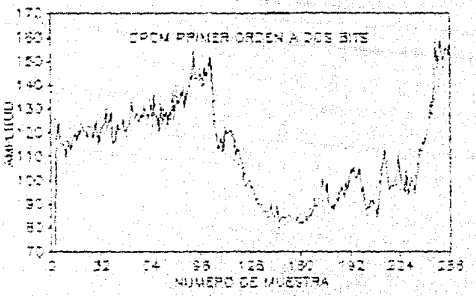
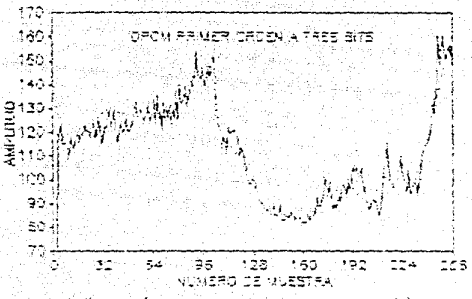
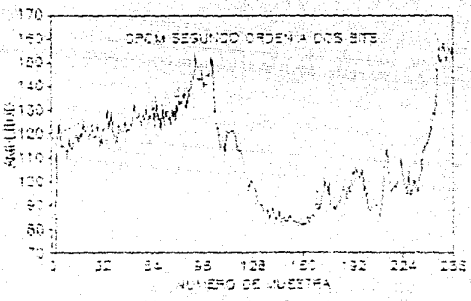


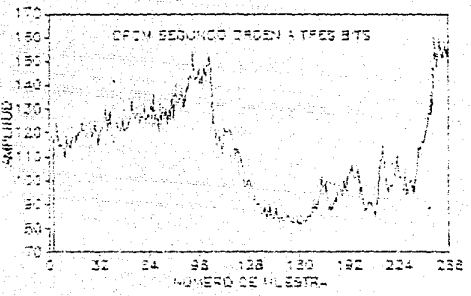
Figura 5.3 Imágenes reconstruidas PCM.
a) R = 2 b) R = 3 c) R = 6 bits/muestra.

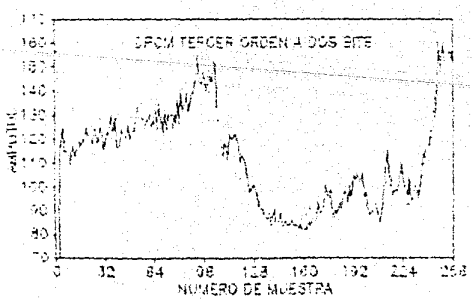
Figura 5.4 Imágenes diferencia PCM.
a) R = 2 b) R = 3 c) R = 6 bits/muestra.











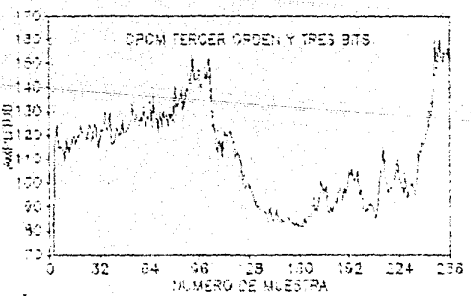


Figura 5.5 Señales temporales reconstruidas en DPCM.

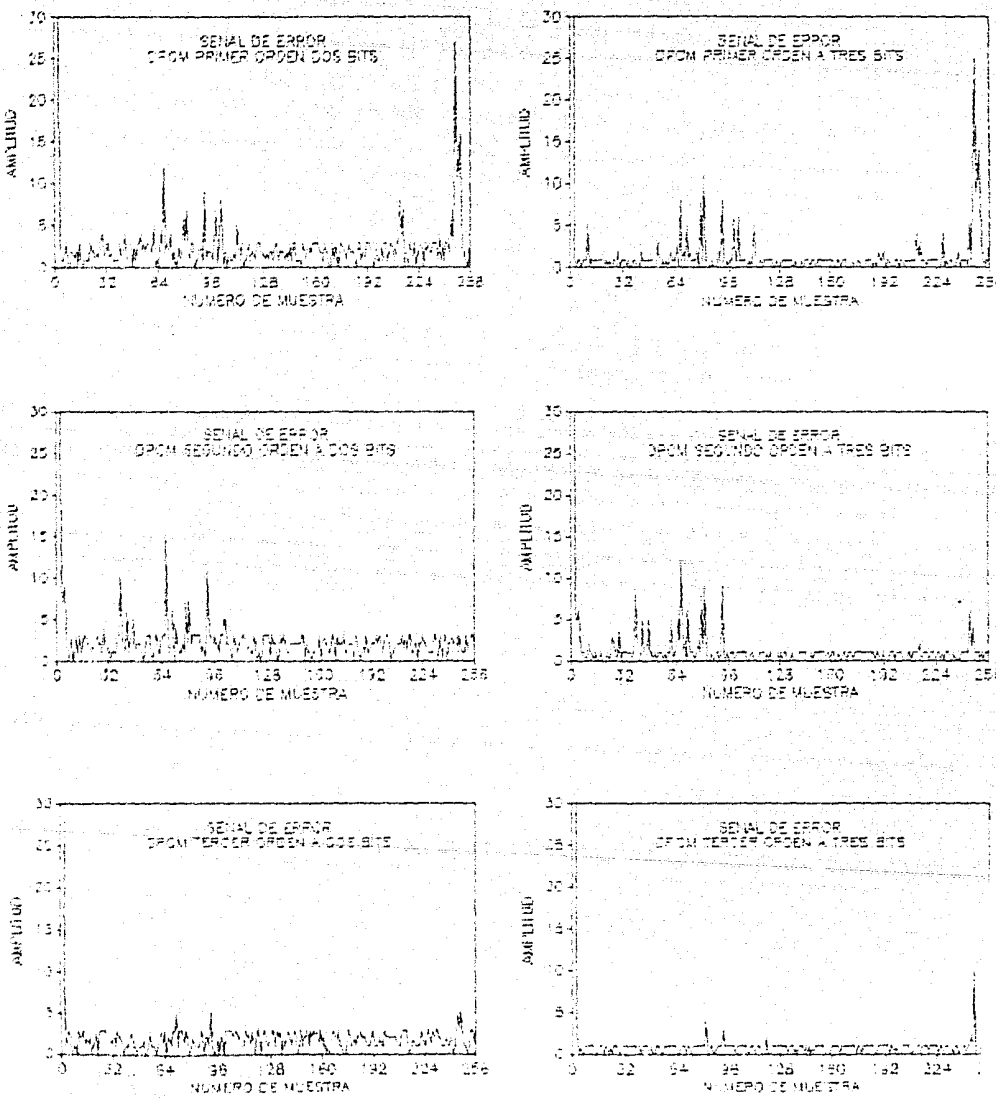


Figura 5.6 Señales temporales de error absoluto DPCM.

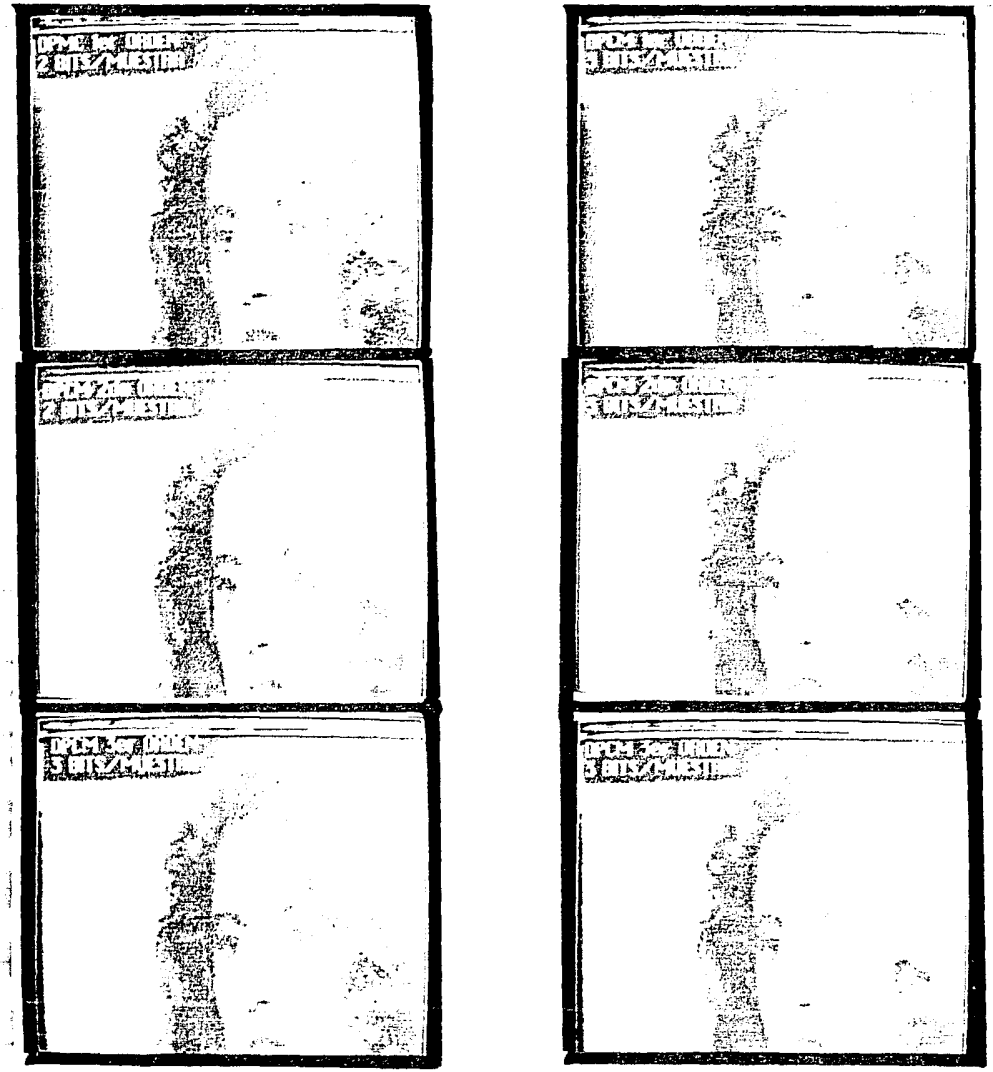


Figura 5.7 Imágenes reconstruidas DPCM.

a) Primer orden, R = 2. b) Primer orden, R = 3.

c) Segundo orden, R = 2. d) Segundo orden, R = 3.

e) Tercer orden, R = 2. f) Tercer orden, R = 3 bits/muestra.

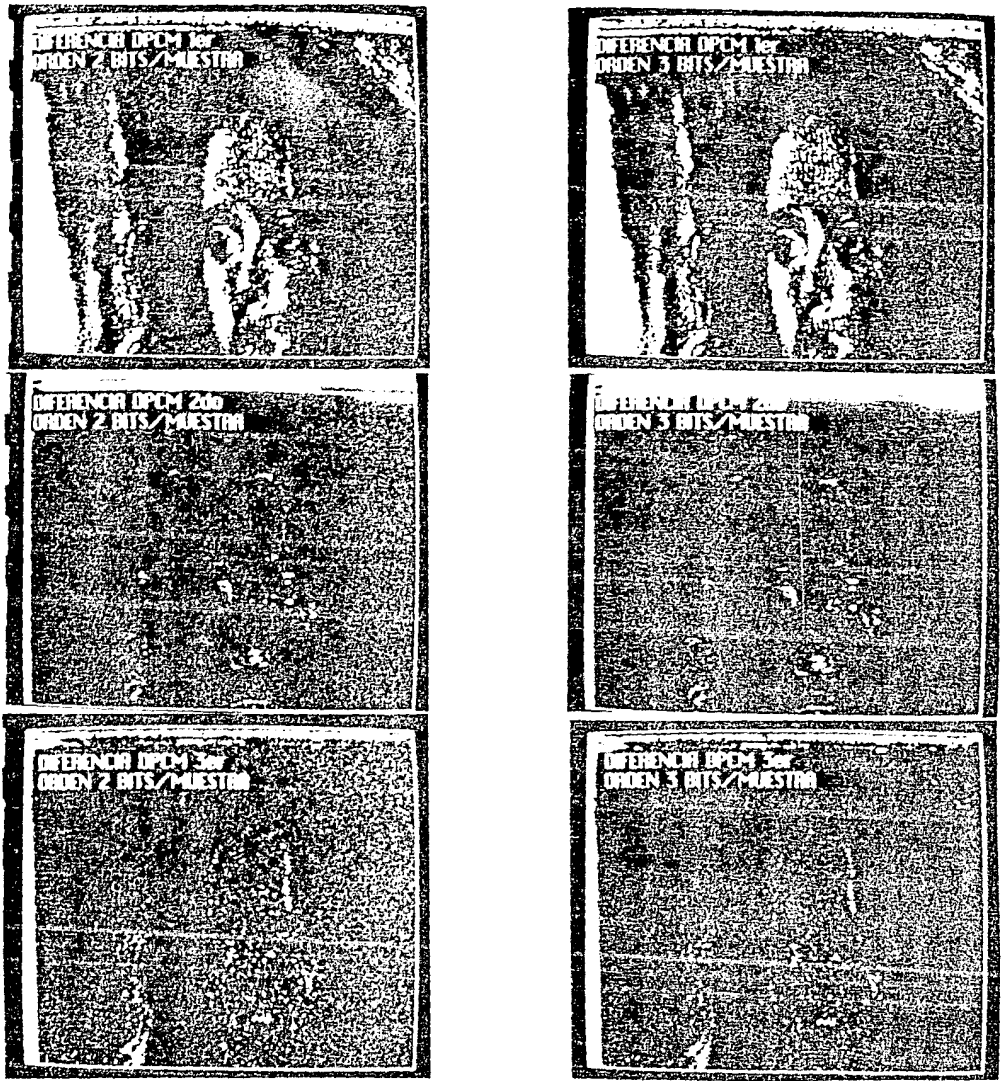
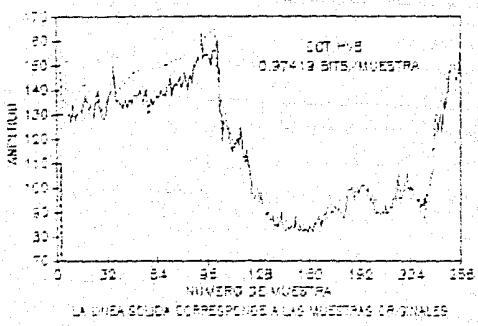


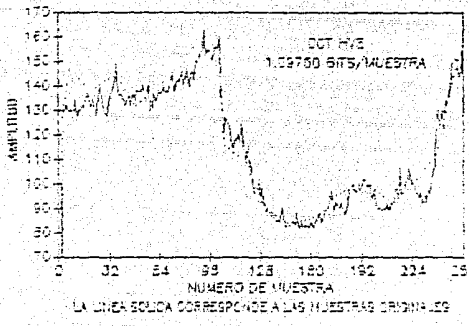
Figura 5.8 Imágenes diferencia DPCM.

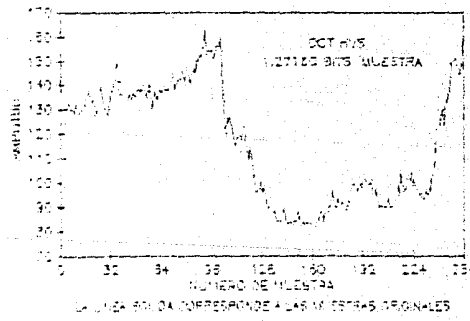
a) Primer orden, R = 2. b) Primer orden, R = 3.

c) Segundo orden, R = 2. d) Segundo orden, R = 3.

e) Tercer orden, R = 2. f) Tercer orden, R = 3 bits/muestra.







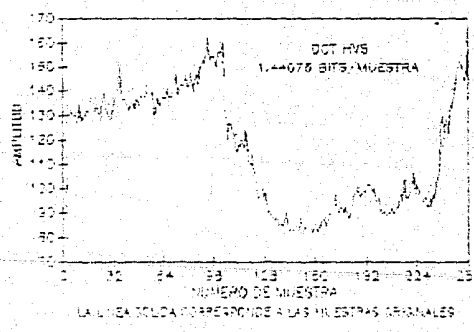
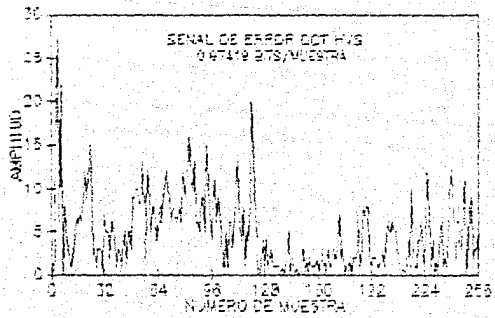
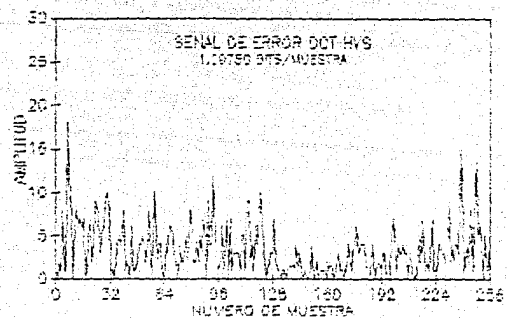
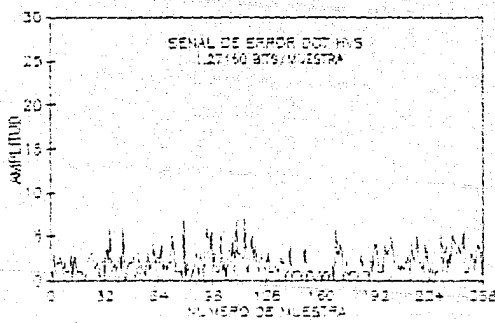


Figura 5.9 Señales temporales reconstruidas DCT.







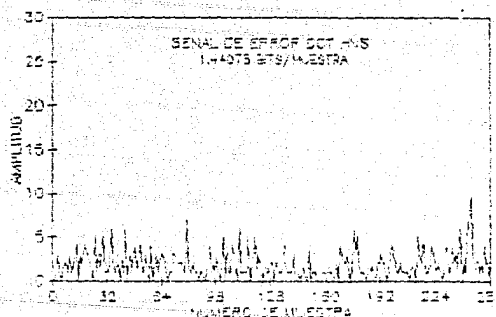


Figura 5.10 Señales temporales de error absoluto DCT.

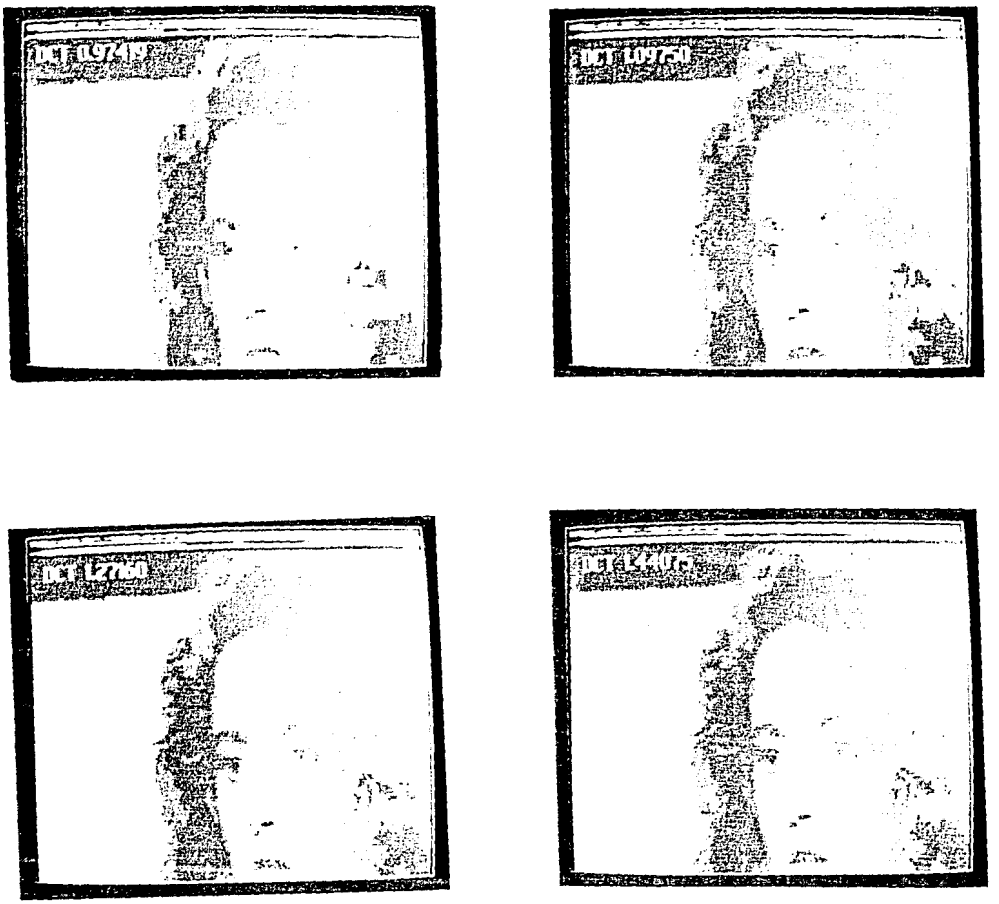


Figura 5.11 Imágenes reconstruídas DCT HVS.
a) R = 0.97419 b) R = 1.09750
c) R = 1.27160 d) R = 1.44075 bits/muestra.

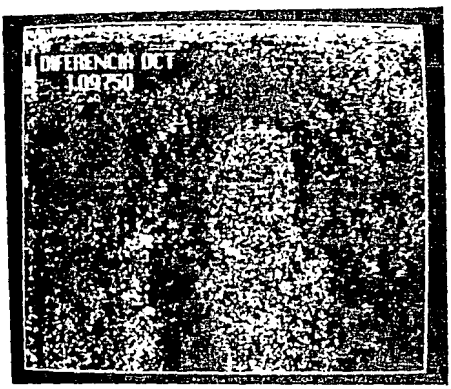


Figura 5.12 Imágenes Diferencia DCT HVS.
a) R = 0.97419 b) R = 1.09750
c) R = 1.27160 d) R = 1.44075 bits/muestra.

5.5.-BIBLIOGRAFIA.

1.- Murakami, H., Hashimoto, H., Hatori, Y., Quality of Band-Compressed TV Services, IEEE Communications Magazine, October 1988.

2.- Feher, C., Digital Communications: Microwave Applications, Prentice-Hall, 1984.

3.- Huchim, D.A., Procesamiento de Señales Bidimensionales para Transmisión-recepción, Tesis Profesional, Instituto Tecnológico de Puebla, 1989.

4.- Jayant, N.S., Noll, P., Digital Coding of Waveforms, Principles and Applications to Speech and Video, Prentice-Hall, 1984.

5.- Jurgen, R.K., High-definition Television Update, IEEE Spectrum, Abril 1988.

6.- Rosenfeld, A., Digital Picture Processing, Academic Press, 1982.

7.- Santiago, S.J., Digitalizador de Señales de Televisión, Tesis Profesional, Facultad de Ingenieria, UNAM, 1990.

CAPITULO VI CONCLUSIONES

Este trabajo comprende actividades de desarrollo y simulación de algoritmos para compresión video, con la potencialidad de ser implantados en circuito, para operar en tiempo real, en un canal de comunicación digital.

Los esquemas desarrollados y simulados comprendieron, modulación diferencial por pulsos codificados (DPCM), de primero, segundo y tercer orden; transformada coseno discreta con la función del sistema visual humano (DCT HVS), y como esquema de comparación secundario, se presentó el esquema PCM a tasas bajas de codificación. De acuerdo con los resultados obtenidos, para una transmisión de imágenes con buena calidad, el esquema DPCM de tercer orden a tres bits/muestra y DCT HVS a 1.44 bits/muestra mostraron el mejor desempeño.

Cuando el número de bits/muestra para codificar usando DCT HVS fueron menores a uno, se presentó el efecto de bloques o falso contorno. Existen técnicas para eliminar este efécto, una de ellas es permitir un ligero traslape entre bloques adyacentes antes de efectuar la transformada coseno discreta, para posteriormente, en la reconstrucción de la imágen, promediar los valores de cada pixel reconstruido que haya sido codificado más de una vez, debido al traslape de bloques.[3] Sin embargo, estas técnicas suman complejidad al esquema, redundando en tiempos de cálculo mayores, que alejan al esquema de la posibilidad de operar en tiempo real, cuando se implanten circuitalmente.

El tipo de imágenes usadas en este trabajo, fueron seleccionada debido a que presenta tanto zonas de bajo como de

alto contraste, permitiendo mostrar los efectos de los esquemas utilizados, al efectuar la compresión/ descompresión. Esto es importante, porque en las comunicaciones de video digital no se tiene control sobre el tipo de imágenes transmitidas, la varianza, contraste, textura, etc., entre imágenes varía considerablemente.

Por otro lado, como el objetivo del esquema es transmitir imágenes en tiempo real, entonces con las tecnologías actuales, sería extremadamente difícil y económicamente insatisfactorio optimizar dinámicamente el algoritmo para cada imágen que se transmite. Aunque se debe de procurar, que los errores generados en la compresión/decompresión se mantengan imperceptibles para el usuario final, por ello, se espera que en un futuro inmediato, el esquema DCT HVS se imponga como una primera norma mundial en la compresión de video, ya sea para fines de transmisión o almacenamiento de las imágenes, e incluso en la tecnología de DVI (digital video interactive) que actualmente emerge en el campo del procesamiento y despliegue de la información.

La traducción de los esquemas simulados hacia circuitos electrónicos operando en tiempo real es realizable sin gran complejidad, dado que en el mercado existen integrados (ASIC's) que realizan cada una de las operaciones necesarias para la ejecución de los algoritmos, incluso para ejecutar la Transformada coseno discreta, directa e inversa, existen integrados a un nivel comercial.

Recientemente se ha introducido un nuevo algoritmo, totalmente diferente a todos los desarrollados para compresión hasta ahora, basado en la transformación fractal, desarrollado por Michel Barnsley[2], que permite potencialmente una tasa de compresión de 1/2000, lo que implica que de cada dos mil datos de la imagen original, se necesita únicamente un dato para representarlos, y a partir del cuál, es posible reconstruir de

nuevo el vector de datos original. Actualmente, el algoritmo presentado, puede alcanzar como máximo una tasa de compresión de 1/36 para una buena calidad de video[4]. Se espera que los primeros chips basados en este algoritmo aparezcan durante 1991.

Dado el avance tecnológico de los componentes utilizados en el procesamiento digital de señales, es probable que este nuevo algoritmo pueda ser fuertemente considerado en la normatividad internacional.

Actualmente, la transmisión, distribución, almacenamiento y despliegue de imágenes constituyen actividades de gran interés científico, tecnológico y comercial a nivel internacional en los escenarios de: a) televisión digital multimedia (CATV, satelite de difución directa, etc.), b) televisión de alta definición, c) videoconferencia, d) DVI, e) almacenamiento masivo y recuperación rápida de imágenes, f) RDSI de banda ancha, g) videófono, h) CAD-CAE interactivo, y en el futuro mediato televisión holográfica.

Es, por lo tanto, importante efectuar una continuación de este trabajo abordando aspectos conceptuales y tecnológicos asociados a la compresión por transformación fractal. Asimismo, las características del sistema de percepción humano, tanto para imágenes fijas como en movimiento, deberán ser explotadas en las técnicas de compresión. Finalmente, las nociones y resultados de la teoría de los sistemas expertos deben ser consideradas en el planteamiento de algoritmos y estructuras de compresión de video en tiempo real.

6.1.-BIBLIOGRAFIA.

1.- Barnsley, M.F., The Desktop Fractal Design Systems, Academic Press Inc., 1989.

2.- Barnsley, M.F., Fractal Research: Payoff at Last. Mathematician Turns Fractal Geometry into Image-compression products, EDN News Edition, vol. 35, No. 19A, September 20, 1990.

3.- Kunt, M., and Ikonomopoulos, A., Second-Generation Image-Coding Techniques, Proceedings IEEE, vol.73, April 1985.

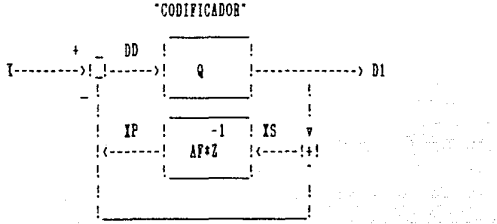
4.- MItchell, J.L., Pennebaker, W.B., Standardization of Color Image Data Compression, II Progressive Coding, Electronic Imaging'89 East, Boston, October, 1989.

```
PROGRAM PCM(INPUT,OUTPUT);
{
*****************************************************************
*     PROGRAMA PARA REALIZAR LA SIMULACION DE UN SISTEMA PCM     *
*                                                               *
*     REALIZADO POR: JUVENTINO ROSAS ESPEJEL.                   *
*     CENTRO DE INVESTIGACION CIENTIFICA Y EDUCACION SUPERIOR   *
*     DE ENSENADA (CICESE).                                     *
*     ABRIL 1990.                                               *
*                                                               *
*****************************************************************
}


VAR
   X1,I,Z,R             :INTEGER;
   C,DD,D1,E            :REAL;
   B                    :ARRAY[1..1024] OF REAL;
   ARCH,ARCHI           :FILE OF BYTE;
   NOM,NAME             :STRING;
   BY,X                 :BYTE;
CONST
    DOS=2;


{
*****************************************************************
 SIGNIFICADO DE LAS VARIABLES UTILIZADAS:

            X1 = NUMERO DE NIVELES DEL CUANTIZADOR
             C = TAMAÑO DEL CUANTIZADOR
          B[I] = SALIDAS DEL CUANTIZADOR
          DD,X = SEÑAL DE ENTRADA
            D1 = SALIDA DEL CUANTIZADOR
*****************************************************************
}

{
*****************************************************************
*                  CUANTIZADOR PCM                              *
*****************************************************************
}
PROCEDURE CUANT;

LABEL 30;
VAR
   R              :INTEGER;

BEGIN

  IF(DD<0) THEN
  BEGIN
    R:=-1;
    DD:=ABS(DD);
  END
  ELSE
  BEGIN
    R:=1;
  END;
  FOR I:=1 TO (X1) DO
    BEGIN
      IF DD<=B[I] THEN
        BEGIN
          IF I=1 THEN
            BEGIN
              D1:=B[I]/2*R;
              DD:=DD*R;
            END
            ELSE
            BEGIN
              D1:=(B[I]+B[I-1])/2*R;
              DD:=DD*R;
            END;
          GOTO 30;
        END;
    END;
D1:=(B[X1]+B[(X1)-1])/2*R;
DD:=DD*R;
30 : END;


{
*****************************************************************
*     ESTA SECCION SE ENCARGA DEL CALCULO DE LOS VALORES DEL    *
*       CUANTIZADOR QUE SE UTILIZARA ATRAVEZ DEL PROGRAMA       *
*****************************************************************
}
PROCEDURE CUANTIZADOR;

CONST
    DOS=2;

BEGIN
WRITELN;WRITELN;
    WRITELN('DAME EL NUMERO DE BITS QUE DESEAS');
    WRITE('PARA EL CUANTIZADOR (Z) >>>>>> ');READLN(Z);
    WRITELN;WRITELN;WRITELN;
    WRITELN('     ********** ! MAQUINA TRABAJANDO ! **********');
{    DADO QUE EN PASCAL NO EXISTE EXPONENCIACION ESTA SE
     IMPLEMENTA DE MANERA ALTERNA EN EL SIGUIENTE "FOR"}

   X1:=2;
   FOR I:=2 TO Z DO
     BEGIN
       X1:=X1*DOS;
     END;
   C:=256/X1;
   FOR I:=1 TO (X1) DO
     BEGIN
       B[I]:=C*I;
     END;
   WHILE NOT EOF(ARCH) DO
     BEGIN
       READ(ARCH,X);
       DD:=X;
       CUANT;
       X:=D1;
       R:=ROUND(X);
       BY:=LO(R);
       WRITE(ARCH1,BY);
```

```
    END;
END;


{
*************************************************************************
*   SECCION ENCARGADA DE SOLICITAR LOS NOMBRES DE LOS ARCHIVOS    *
*************************************************************************
}

PROCEDURE NOMBRES;
BEGIN
  WRITELN;WRITELN;WRITELN;WRITELN;
  WRITE('NOMBRE DEL ARCHIVO PARA LEER (BYTE) >>> ');
  READLN(NAME);
  ASSIGN(ARCH,NAME);
  WRITE('NOMBRE DEL ARCHIVO DE SALIDA (BYTE) >>> ');
  READLN(NOM);
  ASSIGN(ARCHI,NOM);
  RESET(ARCH);
  REWRITE(ARCHI);
END;


{
*************************************************************************
*                    PROGRAMA PRINCIPAL                                 *
*************************************************************************
}
BEGIN

WRITELN;WRITELN;
WRITELN('                          CICESE                              ');
WRITELN('    DEPARTAMENTO DE ELECTRONICA Y TELECOMUNICACIONES           ');
WRITELN('                SISTEMA DE VIDEO PCM                          ');
WRITELN('                   MENU PRINCIPAL                             ');
NOMBRES;
CUANTIZADOR;
CLOSE(ARCH);
CLOSE(ARCHI);
WRITE('         ********** FIN DEL PROCESO **********                 ');
END.
```

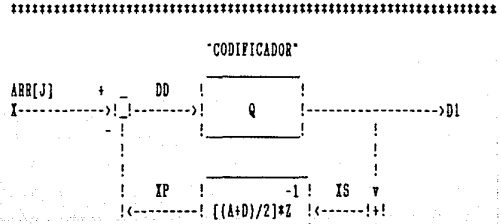
```
PPROGRAM VIDEO(INPUT,OUTPUT);
{
*****************************************************************
*  PROGRAMA PARA REALIZAR LA COMPRESION DE IMAGENES POR MEDIO DE  *
*  TRES TECNICAS PREDICTIVAS: DPCM PRIMER ORDEN, SEGUNDO ORDEN Y  *
*  TERCER ORDEN.                                                  *
*                                                                 *
*  PROGRAMA REALIZADO POR: JUVENTINO BOSAS ESPEJEL.               *
*  CENTRO DE INVESTIGACION CIENTIFICA Y EDUCACION SUPERIOR        *
*  DE ENSENADA (CICESE).                                          *
*  ABRIL DE 1990.                                                 *
*                                                                 *
*****************************************************************
}


VAR
   K1,I,L,M,Z,H,J,K,N   :INTEGER;
   C,DD,D1              :REAL;
   X                    :BYTE;
   FLAG                 :BOOLEAN;
   B                    :ARRAY[1..1024] OF REAL;
   ARCH,ARCH1           :FILE OF BYTE;
   NOM,NOM1,NAME,NOM2   :STRING;
CONST
     AF=0.95;
     DOS=2;


{
*****************************************************************
 SIGNIFICADO DE LAS VARIABLES UTILIZADAS:

            K1 = NUMERO DE NIVELES DEL CUANTIZADOR
             C = TAMAÑO DEL CUANTIZADOR
          B[I] = SALIDAS DEL CUANTIZADOR
        X,A[N] = SERALES DE ENTRADA
            XP = PREDICION
          D,DD = SERAL DE DIFERENCIA
            D1 = SALIDA DEL CUANTIZADOR
            IS = SERAL DE SALIDA CUANTIZADA AL RECEPTOR
             Y = PREDICCION EN EL RECEPTOR
            XR = SERAL RECONSTRUIDA
*****************************************************************
}


{
-----------------------------------------------------------------
-    ESTA SECCION SE ENCARGA DEL CALCULO DEL CUANTIZADOR QUE SE  -
-              UTILIZARA ATRAVEZ DEL PROGRAMA                    -
-----------------------------------------------------------------
}
PROCEDURE CUANTIZADOR;

CONST
     DOS=2;

BEGIN
WRITELN;WRITELN;
     WRITELN('          SECCION ENCARGADA DEL CALCULO EN EL CUANTIZAI
     WRITELN;
     WRITE('QUE ABARQUE EL CUANTIZADOR (L) >>>  ');


     READLN(L);
     WRITELN('DAME EL VALOR MAXIMO POSITIVO QUE QUIERAS');
     WRITE('QUE ABARQUE EL CUANTIZADOR (M), >>>  ');

{ PARA LOS ARCHIVOS MANEJADOS EL MAXIMO VALOR DE  LOS  DATOS  }
{ ES 256 DEBIDO A QUE EL ADQUISIDOR UTILIZA 8 BITS SIN SIGNO  }
{ Y EL MINIMO VALOR ES 0                                      }

     READLN(M);
     WRITELN('DAME EL NUMERO DE BITS QUE DESEAS');
     WRITE('PARA EL CUANTIZADOR (Z) >>>>>> ');READLN(Z);

{ DADO QUE EN PASCAL NO EXISTE EXPONENCIACION ESTA SE
  IMPLEMENTA DE MANERA ALTERNA EN EL SIGUIENTE "FOR"}

     K1:=2;
     FOR I:=2 TO Z DO
         BEGIN
             K1:=K1*DOS;
         END;
    C:=(M-L)/K1;
    FOR I:=1 TO (K1 DIV 2) DO
        BEGIN
            B[I]:=C*I;
        END;

END;


{
-----------------------------------------------------------------
-                      CUANTIZADOR FIJO                         -
-----------------------------------------------------------------
}
PROCEDURE CUANT;

LABEL 30;
VAR
   R          :INTEGER;

BEGIN
     IF (DD<0) THEN
       BEGIN
         R:=-1;
         DD:=Abs(DD);
       END
       ELSE
       BEGIN
         R:=1;
       END;
     FOR I:=1 TO (K1 DIV 2) DO
        BEGIN
            IF DD<=B[I] THEN
               BEGIN
                  IF I=1 THEN
                     BEGIN
                        D1:=B[I]/2*R;
                        DD:=DD*R;
```

```
                    END
                    ELSE
                    BEGIN
                        D1:=(B[I]+B[I-1])/2*R;
                        DD:=DD*R;
                    END;

                GOTO 30;
            END;
        END;
    D1:=(B[X1 DIV 2]+B[(X1 DIV 2)-1])/2*R;
    DD:=DD*R;
30 : END;
{
*************************************************************
*                PREDICCION DE PRIMER ORDEN                 *
*************************************************************
```

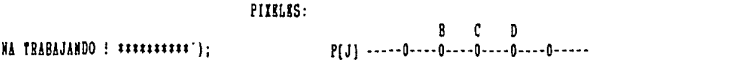



```
PROCEDURE PRIMERO;

VAR
  XS,XR,IP          :REAL;
  RO,COUNT          :INTEGER;
  BY                :BYTE;
BEGIN
    WRITELN;WRITELN;
    WRITELN('      ********** ! MAQUINA TRABAJANDO ! **********');
    IP:=0;
    COUNT:=0;
    WHILE NOT EOF(ARCH) DO
        BEGIN
            COUNT:=COUNT+1;
            READ(ARCH,X);
            DD:=X-IP;
            IF (COUNT=32896) THEN DD:=10;
            CUANT;
            XS:=D1+IP;{SE ESCRIBE XS AL ARCHIVO RESULTANTE}
            IP:=XS*AF;
            RO:=ROUND(XS);
            BY:=LO(RO);
            WRITS(ARCHI,BY);
        END;
    CLOSE(ARCH);
    CLOSE(ARCHI);
END;
```

```
{
*************************************************************
*                PREDICCION DE SEGUNDO ORDEN                *
*************************************************************
```

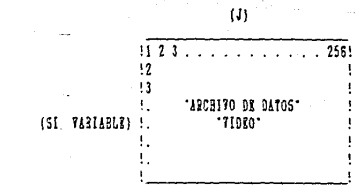


EL METODO DE PREDICCION SE HACE A PARTIR DEL SIGUIENTE ESQUEMA DE PIXELES:



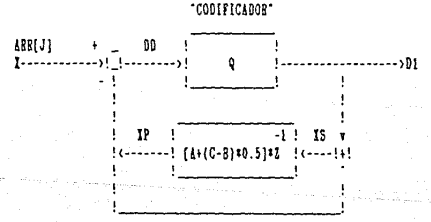
EL ARCHIVO, DONDE SE ENCUENTRA GRABADA LA IMAGEN DE 256X256, ES

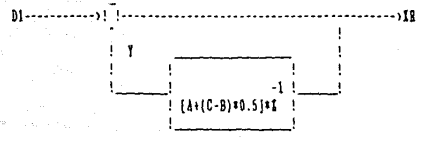
MANEJADO DE ACUERDO CON LAS SIGUIENTES VARIABLES:




```
}
P2C  DURE SEGUNDO;

VAR
    BB                  :ARRAY[1..256] OF BYTE;
    S,IR                :ARRAY[1..256] OF REAL;
    P                   :ARRAY[1..257] OF REAL;
    C,Y,E,F             :REAL;
    O                   :INTEGER;
    BY                  :BYTE;
BEGIN
    WRITELN;WRITELN;
    WRITELN('         ********** ! MAQUINA TRABAJANDO ! **********');
    S:=0;               { INICIALIZACION }
    P:=0;               { INICIALIZACION }
    FOR J:=1 TO 256 DO { INICIALIZACION }
    BEGIN               { INICIALIZACION }
        P[J]:=0;        { INICIALIZACION }
    END;                { INICIALIZACION }

    WHILE NOT EOF(ARCH) DO
    BEGIN
        FOR J:=1 TO 256 DO
          BEGIN
            READ(ARCH,I);
            ARR[J]:=I;
          END;
        FOR J:=1 TO 256 DO
           BEGIN
              IF J=1 THEN
                BEGIN
                  IP:=(F+P[J+1])*0.5;
                  DD:=ARR[J]-IP;
                  COANT;
                  IS[J]:=D1+IP;
                  Y:=(E+P[J+1])*0.5;
                  IR[J]:=D1+Y;
                  RO:=ROUND(IR[J]);
                  BY:=LO(RO);
                  P[J]:=IR[J];
                END
              ELSE
              BEGIN
                IF J=256 THEN P[J+1]:=P[J];
                IP:=(IS[J-1]+P[J+1])*0.5;
                DD:=ARR[J]-IP;
                COANT;
                IS[J]:=D1+IP;
                Y:=(IR[J-1]+P[J+1])*0.5;        { DECODIFICADOR
                IR[J]:=D1+Y;                    { DECODIFICADOR
                RO:=ROUND(IR[J]);
                BY:=LO(RO);
                P[J]:=IR[J];{ ACTUALIZACION DE LINEA DE TRABAJO
              END;
            WRITE(ARC2I,BY);
          END;
        E:=IR[J];
        F:=IS[J];
    END;
    CLOSE(ARCH);
    CLOSE(ARCHI);
END;
```

```
{
*****************************************************************
*                    PREDICCION DE TERCER ORDEN                 *
*****************************************************************
```





 L ESQUEMA DE PIXELES Y LAS VARIABLES UTILIZADAS PARA EL MANEJO DEL

```
ARCHIVO DE DATOS DE LA IMAGEN ES  EL MISMO  QUE EL  UTILIZADO EN LA
 REDICCION DE SEGUNDO ORDEN.

}
ROCEDURE TERCERO;

AR
   ARR                 :ARRAY[1..256] OF BYTE;
   XS,XR               :ARRAY[1..256] OF REAL;
   P                   :ARRAY[1..257] OF REAL;
   XP,Y                :REAL;
   RO                  :INTEGER;
   BY                  :BYTE;

EGIN
    WRITELN;WRITELN;
    WRITELN('      ********** ! MAQUINA TRABAJANDO ! **********');
    XP:=0;              { INICIALIZACION }
     Y:=0;              { INICIALIZACION }
    FOR J:=1 TO 256 DO  { INICIALIZACION }
       BEGIN            { INICIALIZACION }
           P[J]:=0;     { INICIALIZACION }
       END;             { INICIALIZACION }
    WHILE NOT EOF(ARCH) DO
      BEGIN
          FOR J:=1 TO 256 DO
             BEGIN
                 READ(ARCH,X);
                 ARR[J]:=X;
             END;
          FOR J:=1 TO 256 DO
             BEGIN
               IF J=256 THEN
                 BEGIN
                   P[J+1]:=P[J];
                   P[J]:=P[J-1];
                 END;
               DD:=ARR[J]-XP;
               CUANT;
               XS[J]:=D1+XP;
               XR[J]:=D1+Y;
               RO:=ROUND(XR[J]);
               BY:=LO(RO);
               Y:=XR[J]+(P[J+1]-P[J])*0.5;
               XP:=XS[J]+(P[J+1]-P[J])*0.5;
               P[J]:=XR[J];
               WRITE(ARCH1,BY);
             END;
      END;
      CLOSE(ARCH);
      CLOSE(ARCH1);
ND;


{
 ------------------------------------------------------------
   SECCION ENCARGADA DE SOLICITAR LOS NOMBRES DE LOS ARCHIVOS  -
 ------------------------------------------------------------
}
PROCEDURE NOMBRES;
BEGIN
  WRITELN;WRITELN;WRITELN;WRITELN;
  WRITE('NOMBRE DEL ARCHIVO PARA LEER (byte) >>> ');READLN(NAME);
  ASSIGN(ARCH,NAME);
  WRITE('NOMBRE DEL ARCHIVO DE SALIDA (byte) >>> ');READLN(NOM);
  ASSIGN(ARCH1,NOM);
  RESET(ARCH);
  REWRITE(ARCH1);
END;

{
 ------------------------------------------------------------------
 -   SECCION QUE SE ENCARGARA DE HACER LA SELECCION DEL TIPO DE   -
 -               TECNICA  QUE SE DESEA UTILIZAR                   -
 ------------------------------------------------------------------
}
PROCEDURE SELECCION;
LABEL 10,20;
BEGIN

10 : CUANTIZADOR;
     NOMBRES;
     WRITELN;WRITELN;
     WRITELN('TECLEA UNA TECLA DE ACUERDO CON LA TECNICA QUE QUIERAS
     WRITELN('SIMULAR >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> <H>
     WRITELN;
     WRITELN('<1> PARA DPCM DE PRIMER ORDEN');
     WRITELN('<2> PARA DPCM DE SEGUNDO ORDEN');
     WRITELN('<3> PARA DPCM DE TERCER ORDEN');
     WRITELN('<4> PARA TERMINAR EL PROGRAMA');
     FLAG:=TRUE;
     REPEAT
     WRITELN;
     READLN(H);
          IF ((H<>1) AND (H<>2) AND (H<>3)) THEN GOTO 20;
                CASE H OF
                   1 : PRIMERO;
                   2 : SEGUNDO;
                   3 : TERCERO;
                END;
             FLAG:=FALSE;
     UNTIL FLAG=FALSE;
     WRITELN;WRITELN;
     WRITELN('          QUIERES SIMULAR OTRA TECHICA, <SI> o <NO> ?'
     WRITELN;
     READLN(NOM1);
     IF ((NOM1='SI') OR (NOM1='si')) THEN GOTO 10;
     WRITELN;WRITELN;
20 : WRITELN('          ********* FIN DE LA SIMULACION *********');
END;

{
*****************************************************************
*                    PROGRAMA PRINCIPAL                         *
*****************************************************************
```

```
}
BEGIN

WRITELN;WRITELN;
WRITELN('                          CICESE                          ');
WRITELN('    DEPARTAMENTO DE ELECTRONICA Y TELECOMUNICACIONES       ');
WRITELN('             SISTEMA DE COMPRESION DE VIDEO               ');
WRITELN('                  POR TECNICAS DPCM                       ');
WRITELN('                    MENU PRINCIPAL                        ');
SELECCION;
END.
```

```
/*
 ************************************************************
 *  PROGRAMA PARA REALIZAR EL CALCULO DE LA TRANSFORMADA COSENO *
 *  DISCRETA VIA LA TRANSFORMADA RAPIDA DE FOURIER.          *
 *                                                           *
 *  PROGRAMA REALIZADO POR: JUVENTINO ROSAS ESPEJEL          *
 *  CENTRO DE INVESTIGACION CIENTIFICA Y EDUCACION           *
 *  SUPERIOR DE ENSENADA (CICESE).                           *
 *  1990.                                                    *
 ************************************************************
*/

   #include<stdio.h>
   #include<stdlib.h>
   #include<conio.h>
   #include<math.h>

   #define pi 3.141592654

   main(int argc,char *argv[])
   {
   FILE *in;
   FILE *out;
   float dct[8][8],dcti[8][8];
   float tdct[8][8],tdcti[8][8];
   float iorg[8][8];
   float trorgi[8][8],trorg[8][8];
   int suma,bloque,a;
   int m,j,i,k,n,linea,dato;
   float rm,lee[8][256];
   unsigned char ch;
   clrscr();

   if(argc!=3)
    {
    gotoxy(5,10);
    printf("Te olvidaste de teclear el nombre del archivo de entrada");
    gotoxy(5,11);
    printf("ó de salida: program  archivo entrada  archivo salida");
    exit(1);
    }

   if((in=fopen(argv[1],"rb"))==NULL)
    {
    gotoxy(5,10);
    printf("no fue posible abrir el archivo de entrada");
    exit(1);
    }

   if((out=fopen(argv[2],"wb"))==NULL)
    {
    gotoxy(5,11);
    printf("no fue posible abrir el archivo de salida ");
    exit(1);
    }

gotoxy(14,12);
printf("************ ! MAQUINA TRABAJANDO ! ************");

/*
 ------------------------------------------------------------
 -  Sección del programa para generar la matriz DCT[8][8]  y la  -
 -          transpuesta de esta matriz TDCT[8][8]            -
 ------------------------------------------------------------
*/

n=8;            /* orden de las matrices a procesar n*n */
rm=n;           /* y por lo tanto el orden de la matriz */
                /*   es el tamaño del bloque procesado  */

for(j=0;j<n;j++)
 {
 dct[0][j]=sqrt(1.0/rm);
 tdct[j][0]=dct[0][j];
 }

for(i=1;i<n;i++)
 {
 for(j=0;j<n;j++)
  {
  dct[i][j]=(sqrt(2.0/rm))*cos(2.0*pi*( (2.0*(j+1))-1 )*(i)/(4.0*rm));
  tdct[j][i]=dct[i][j];
  }
 }

for(i=0;i<n;i++)
 {
 for(j=0;j<n;j++)
  {
  dcti[i][j]=dct[i][j]/(sqrt(2.0/rm));
  tdcti[i][j]=tdct[i][j]/(sqrt(2.0/rm));
  dct[i][j]=dct[i][j]*sqrt(2.0/rm);
  tdct[i][j]=tdct[i][j]*sqrt(2.0/rm);
  }
 }

/*
 ------------------------------------------------------------
 -  Finaliza sección de generación de DCT[8][8]  y TDCT[8][8]  -
 ------------------------------------------------------------
*/

/*
 ------------------------------------------------------------
 -  Esta sección ejecuta la transformada coseno directa de acuerdo -
 -       con la fórmula: [DCT]*[IORG]*[TDCT]=[TRORG]         -
 -          Sección de cambio de línea de bloques            -
 ------------------------------------------------------------
*/

for(linea=0;linea<32;linea++)
 {

/*-------------------------------------------------------------
                    Lectura de datos
```

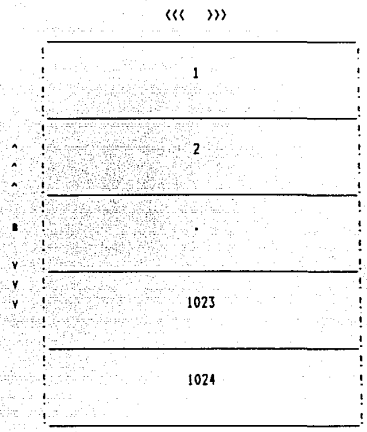
```
----------------------------------------------------------------*/

or(i=0;i<8;i++)
{
for(j=0;j<256;j++)
 {
 ch=getc(in);
 dato=ch;
 lee[i][j]=dato;
 }
}

/*-------------------------------------------------------------
            Finaliza la lectura de los datos
-------------------------------------------------------------*/

,*
-------------------------------------------------------------
     Sección de ejecución de la transformada coseno directa -
                    cambio de bloque -
-------------------------------------------------------------
*/

suma=0;
for(bloque=0;bloque<32;bloque++)
 {
 if(bloque!=0) suma=suma+8;
 a=linea*32+bloque+1;
 gotoxy(34,15);
 printf("a= %d",a);

/*
 -------------------------------------------------------------
   Asignación de datos leidos para la matriz a procesar
-------------------------------------------------------------
 */

for(i=0;i<8;i++)

 for(j=0;j<8;j++)
  {
  iorg[i][j]=lee[i][j+suma];
  }
 }

/*-------------------------------------------------------------
   Finaliza asignación de datos leidos a la matriz a procesar
-------------------------------------------------------------*/

for(k=0;k<m;k++)

 for(n=0;n<m;n++)
  {
  trorg1[k][n]=0.0;
  for(j=0;j<m;j++)
   {
    trorg1[k][n]=trorg1[k][n]+dct[k][j]*iorg[j][n];
   }
  }
 }

for(k=0;k<m;k++)
 {
 for(n=0;n<m;n++)
  {
  trorg[k][n]=0.0;
  for(j=0;j<m;j++)
   {
   trorg[k][n]=trorg[k][n]+trorg1[k][j]*tdct[j][n];
   }
  }
 }

/*-------------------------------------------------------------
        Se escriben los datos en el archivo de salida
-------------------------------------------------------------*/

for(i=0;i<m;i++)
 {
 for(j=0;j<m;j++)
  {
  fwrite(&trorg[j][i],4,1,out);
  }
 }

/*-------------------------------------------------------------
          Termina escritura en el archivo de salida
-------------------------------------------------------------*/

}      /* end for(bloque) */

/*
-------------------------------------------------------------
-          Termina sección de cambio de bloque             -
-------------------------------------------------------------
*/

}     /* end for(linea) */

/*
-------------------------------------------------------------
-  Finaliza sección que ejecuta la transformada coseno directa -
-              y cambio de linea de bloques                 -
-------------------------------------------------------------
*/


fclose(in);
fclose(out);

   }     /* end main() */
```

```
/*
*****************************************************************
  PROGRAMA PARA REALIZAR EL CALCULO DE LA TRANSFORMADA COSENO   *
*  DISCRETA INVERSA VIA LA TRANSFORMADA RAPIDA DE FOURIER.      *
*                                                               *
  PROGRAMA REALIZADO POR: JUVENTINO ROSAS ESPEJEL.             *
*  CENTRO DE INVESTIGACION CIENTIFICA Y EDUCACION               *
*  SUPERIOR DE ENSENADA (CICESE).                               *
  1990.                                                        *
*                                                               *
*****************************************************************
*/


  #include<stdio.h>
  #include<stdlib.h>
  #include<conio.h>
  #include<math.h>

  #define pi 3.141592654

  main(int argc,char *argv[])
  {
  FILE *in;
  FILE *out;
  float dct[8][8],dcti[8][8];
  float tdct[8][8],tdcti[8][8];
  float trorg[8][8];
  float tro[8][8],tro1[8][8];
  float lee[128][64];
  int m,j,i,k,n,bloque,linea,a,b;
  float rm,s,z1,z2;
  unsigned char ch;
  clrscr();

  if(argc!=3)
   {
   gotoxy(5,10);
   printf("Te olvidaste de teclear el nombre del archivo de entrada");
   gotoxy(5,11);
   printf("ó de salida: program    archivo entrada   archivo salida");
   exit(1);
   }

  if((in=fopen(argv[1],"rb"))==NULL)
   {
   gotoxy(5,11);
   printf("no fue posible abrir el archivo de entrada inv.dat");
   exit(1);
   }

  if((out=fopen(argv[2],"wb"))==NULL)
   {
   gotoxy(5,11);
   printf("no fue posible abrir el archivo de salida 3");
   exit(1);
   }

   gotoxy(14,12);
   printf("************ ! MAQUINA TRABAJANDO ! ************");

/*
*****************************************************************
*   Sección del programa para generar la matriz DCT[8][8]  y la *
*           transpuesta de esta matriz TDCT[8][8]               *
*****************************************************************
*/

  m=8;              /* orden de las matrices a procesar m*m */
  rm=m;             /* y por lo tanto el orden de la matriz */
                    /*   es el tamaño del bloque procesado  */

  for(j=0;j<m;j++)
   {
   dct[0][j]=sqrt(1.0/rm);
   tdct[j][0]=dct[0][j];
   }

  for(i=1;i<m;i++)
   {
   for(j=0;j<m;j++)
    {
    dct[i][j]=(sqrt(2.0/rm))*cos(2.0*pi*( (2.0*(j+1))-1 )*(i)/(4.0*rm))
    tdct[j][i]=dct[i][j];
    }
   }

  for(i=0;i<m;i++)
   {
   for(j=0;j<m;j++)
    {
    dcti[i][j]=dct[i][j]/(sqrt(2.0/rm));
    tdcti[i][j]=tdct[i][j]/(sqrt(2.0/rm));
    dct[i][j]=dct[i][j]*sqrt(2.0/rm);
    tdct[i][j]=tdct[i][j]*sqrt(2.0/rm);
    }
   }

/*
*****************************************************************
*   Finaliza sección de generación de DCT[8][8]  y TDCT[8][8]   *
*****************************************************************
*/

/*
*****************************************************************
*   Sección del programa que ejecuta la transformada inversa por *
*           medio de la expresión: [TDCT]*[TRORG]*[DCT]         *
*                       cambio de linea                         *
*****************************************************************

*/

  for(linea=0;linea<8;linea++)
  {

  /*
  ----------------------------------------------------------------
```

```
                    Lectura de los datos
-------------------------------------------------------------------

for(i=0;i<128;i++)
 {
 for(j=0;j<64;j++)
  {
   fread(&lee[i][j],4,1,in);
  }
 }


-------------------------------------------------------------------
                Finaliza la lectura de los datos
-------------------------------------------------------------------
*/


-------------------------------------------------------------------
                Sección de cambio de bloque
-------------------------------------------------------------------


for(bloque=0;bloque<128;bloque++)

 b=linea*128+bloque+1;
 gotoxy(34,15);
  printf("b= %d",b);

/*
-------------------------------------------------------------------
      Asignación de datos leidos para la matriz a procesar
-------------------------------------------------------------------
*/

 for(i=0;i<m;i++)
  {
  for(j=0;j<m;j++)
  {
    a=(i*8)+j;
   trorg[i][j]=lee[bloque][a];
  }
 }


-------------------------------------------------------------------
   Finaliza asignación de datos leidos a la matriz a procesar
-------------------------------------------------------------------


 for(k=0;k<m;k++)

  for(n=0;n<m;n++)
   {
   trol[k][n]=0.0;
   for(j=0;j<m;j++)
    {
    trol[k][n]=trol[k][n]+tdcti[k][j]*trorg[j][n];
    }
   }
  }

 for(k=0;k<m;k++)
  {
  for(n=0;n<m;n++)
   {
   tro[k][n]=0.0;
   for(j=0;j<m;j++)
    {
    tro[k][n]=tro[k][n]+trol[k][j]*dcti[j][n];
    }
   }
  }

/*
-------------------------------------------------------------------
        Se escriben los datos en el archivo de salida
-------------------------------------------------------------------
*/

 for(i=0;i<m;i++)
  {
  for(j=0;j<m;j++)
   {
   x1=ceil(tro[i][j]);
   x2=fabs(x1-tro[i][j]);
   if(x2<=0.5) s=x1;
   else
   s=floor(tro[i][j]);
   putc(s,out);
   }
  }

/*
-------------------------------------------------------------------
        Termina la escritura en el archivo de salida
-------------------------------------------------------------------
*/

 }       /* end for(bloque) */

/*
-------------------------------------------------------------------
            Finaliza sección de cambio de bloque
-------------------------------------------------------------------
*/

 }       /* end for(linea) */

/*
*******************************************************************
*     Finaliza sección de ejecución de la transformada inversa    *
*                       y cambio de linea                         *
*******************************************************************
*/

 fclose(in);
 fclose(out);
 }      /* end main */
```

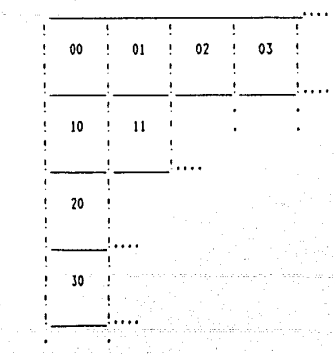
```
/*
*****************************************************************
*  PROGRAMA PARA REORDENAR LOS COEFICIENTES TRANSFORMADOS       *
*    T HVS EN ORDEN DECRECIENTE DE SU IMPORTANCIA VISUAL.       *
*  ESTE PROGRAMA CORRESPONDE AL BLOQUE NUMERO TRES DE LA        *
*  FIGURA 4.4, EN LA TESIS.                                     *
*                                                               *
*  PROGRAMA REALIZADO POR:  JUVENTINO ROSAS ESPEJEL.            *
*  CENTRO DE INVESTIGACION CIENTIFICA Y EDUCACION               *
*    PERIOR DE ENSENADA (CICESE).                               *
*    90.                                                        *
*                                                               *
*****************************************************************
*/



    #include<stdio.h>
    #include<stdlib.h>
    #include<conio.h>
    #include<math.h>

    main(int argc, char *argv[])
    {
      FILE *in;
      FILE *out;
      float s1,s2,dato,s;
      float e[8][8];
      float n[8][8];
      long int a;
      int m,j,k;
      clrscr();

      if(argc!=3)
      {
      gotoxy(5,10);
      printf("te olvidaste de teclear el  nombre  de  alguno");
      gotoxy(5,11);
      printf("de los archivos para procesar (entrada-salida)");
      exit(1);
      }

      if((in=fopen(argv[1],"rb"))==NULL)
      {
      gotoxy(5,10);
      printf("no fue posible abrir el archivo de entrada");
      exit(1);
      }

      if((out=fopen(argv[2],"wb"))==NULL)
      {
      gotoxy(5,11);
      printf("no fue posible abrir el archivo de salida");
      exit(1);
      }

/*
      La busqueda de los datos a procesar se hace de acuerdo con la
      siguiente estrategia:

      1.-Dado que el programa    DCTHVS.c  coloca los datos de los bloques
```

resultantes de manera secuencial ya no es necesario utilizar la funcion fseek() con tanta frecuencia, unicamente se utiliza para realizar el cambio de bloques. La variable utilizada es: m, que es la variable encargada del cambio del bloque a procesar.



2.-Adicionalmente cada uno de los bloques contiene 64 pixeles ,en la figura siguiente se muestra la ubicacion de los pixeles en un bloque cualquiera de los 1024 posibles.

3.- Los puntos del bloque de 8*8 que este programa esta transmitiend



```
      El reordenamiento de los datos es de acuerdo a la siguiente
      secuencia:
      originalmente se tiene: 00 01 02 03 04 05 06 07
                              10 11 ...          17
                              20                 .
                              .                  .
                              .       jk         .
                              .       ||         .
                              .       ||         .
                              .       vv         .
                              70 71 72 73 74 75 76 77

      y con el cambio la secuencia queda:

                              00 01 10 20 11 02 03 12
                              21 30 40 31 22 13 04 05
                              14 23 32 41 50 60 51 42
                              33 24 15 06 07 16 25 34
                              43 52 61 70 71 62 53 44
                              35 26 17 27 36 45 54 63
                              72 73 64 55 46 37 47 56
                              65 74 75 66 57 67 76 77

      Esto se hace para poder generar corridas grandes de ceros.

      Nota: No se muestra el bloque de 8*8 completo.

*/
  gotoxy(14,12);
  printf("************ ! MAQUINA TRABAJANDO ! ************");

 /*************************************************************
 *                    Inicio del zig-zag                     *
 *                 sección de cambio de bloque               *
 *************************************************************/

 for(m=0;m<1024;m++)

 gotoxy(35,16);
  printf("m= %d",m);
  if(m==0) a=0;     /* cambio de bloque */
 else
 a=a+256;
  seek(in,a,0);

 /*
 *************************************************************
 *              Sección para la lectura de los datos         *
 *************************************************************/
  for(j=0;j<8;j++)
  {
  for(k=0;k<8;k++)
   {
   fread(&dato,4,1,in);
   s1=dato;
   e[j][k]=s1;
   }
  }
 /*
 *************************************************************
 *           Finaliza la sección de lectura de los datos     *
 *************************************************************/

 /*************************************************************
 *   La siguiente sección hace el reordenamiento de los puntos *
 *************************************************************/

 n[0][0]=e[0][0];              /* linea uno */
 n[0][1]=e[0][1];
 n[0][2]=e[1][0];
 n[0][3]=e[2][0];
 n[0][4]=e[1][1];
 n[0][5]=e[0][2];
 n[0][6]=e[0][3];
 n[0][7]=e[1][2];

 n[1][0]=e[2][1];              /* linea dos */
 n[1][1]=e[3][0];
 n[1][2]=e[4][0];
 n[1][3]=e[3][1];
 n[1][4]=e[2][2];
 n[1][5]=e[1][3];
 n[1][6]=e[0][4];
 n[1][7]=e[0][5];

 n[2][0]=e[1][4];              /* linea tres */
 n[2][1]=e[2][3];
 n[2][2]=e[3][2];
 n[2][3]=e[4][1];
 n[2][4]=e[5][0];
 n[2][5]=e[6][0];
 n[2][6]=e[5][1];
 n[2][7]=e[4][2];

 n[3][0]=e[3][3];              /* linea cuatro */
 n[3][1]=e[2][4];
 n[3][2]=e[1][5];
 n[3][3]=e[0][6];
 n[3][4]=e[0][7];
 n[3][5]=e[1][6];
 n[3][6]=e[2][5];
 n[3][7]=e[3][4];

 n[4][0]=e[4][3];              /* linea cinco */
 n[4][1]=e[5][2];
 n[4][2]=e[6][1];
 n[4][3]=e[7][0];
 n[4][4]=e[7][1];
 n[4][5]=e[6][2];
 n[4][6]=e[5][3];
 n[4][7]=e[4][4];

 n[5][0]=e[3][5];              /* linea seis */
 n[5][1]=e[2][6];
```

```
n[5][2]=e[1][7];
n[5][3]=e[2][7];
n[5][4]=e[3][6];
n[5][5]=e[4][5];
n[5][6]=e[5][4];
n[5][7]=e[6][3];

n[6][0]=e[7][2];                /* línea siete */
n[6][1]=e[7][3];
n[6][2]=e[6][4];
n[6][3]=e[5][5];
n[6][4]=e[4][6];
n[6][5]=e[3][7];
n[6][6]=e[4][7];
n[6][7]=e[5][6];

n[7][0]=e[6][5];                /* línea ocho */
n[7][1]=e[7][4];
n[7][2]=e[7][5];
n[7][3]=e[6][6];
n[7][4]=e[5][7];
n[7][5]=e[6][7];
n[7][6]=e[7][6];
n[7][7]=e[7][7];

 /*
 ********************************************************************
 *       Finaliza la sección de reordenamiento de los puntos        *
 ********************************************************************
 */

 /*
 ********************************************************************
 *           Sección de la reescritura de los puntos                *
 ********************************************************************
 */

for(j=0;j<8;j++)
 {
 for(k=0;k<8;k++)
  {
  s2=n[j][k];
  fwrite(&s2,4,1,out);
  }
 }
 /*
 ********************************************************************
 *          Termina sección de reescritura de puntos                *
 ********************************************************************
 */

 }
/*
*********************************************************************
*                  Fin de la sección zig-zag                        *
*                 Sección de cambio de bloque                       *
*********************************************************************
*/
   fclose(in);
   fclose(out);
 }
```

```
/*
*****************************************************************
*  PROGRAMA PARA REORDENAR LOS  COEFICIENTES  TRANSFORMADOS     *
*  DCT HVS, EN SUS POSICIONES ORIGINALES QUE TENIAN DESPUES     *
*  DE LA TRANSFORMADA COSENO DISCRETA.                          *
*  ESTE PROGRAMA CORRESPONDE AL BLOQUE NUMERO DIEZ DE LA        *
*  FIGURA 4.4, EN LA TESIS.                                     *
*                                                               *
*  PROGRAMA REALIZADO POR:  JUVENTINO ROSAS ESPEJEL.            *
*  CENTRO DE INVESTIGACION CIENTIFICA Y EDUCACION               *
*  SUPERIOR DE ENSENADA (CICESE).                               *
*  1990.                                                        *
*                                                               *
*****************************************************************
*/



#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>

main(int argc, char *argv[])
{
FILE *in;
FILE *out;
float dato,s;
float s1,s2;
float e[8][8];
float n[8][8];
long int a;
int m,j,k;
clrscr();

if(argc!=3)
 {
 gotoxy(5,10);
 printf("te olvidaste de teclear el  nombre  de  alguno");
 gotoxy(5,11);
 printf("de los archivos para procesar (entrada-salida)");
 exit(1);
 }

if((in=fopen(argv[1],"rb"))==NULL)
 {
 gotoxy(5,10);
 printf("no fue posible abrir el archivo de entrada");
 exit(1);
 }

if((out=fopen(argv[2],"wb"))==NULL)
 {
 gotoxy(5,11);
 printf("no fue posible abrir el archivo de salida");
 exit(1);
 }

/*
```

El reordenamiento de los datos en el receptor consiste en reordenar los bloques recibidos, los cuales tienen la siguiente secuencia de componentes transformados:

```
00 01 10 20 11 02 03 12
21 30 40 31 22 13 04 05
14 23 32 41 50 60 51 42
33 24 15 06 07 16 25 34
43 52 61 70 71 62 53 44
35 26 17 27 36 45 54 63
72 73 64 55 46 37 47 56
65 74 75 66 57 67 76 77
```

Para que queden ordenados como estaban originalmente:

```
00 01 02 03 04 05 06 07
10 11 ...           17
20                  .
.                   .
.       jk          .
.       ;;          .
.       ;;          .
.       vv          .
70 71 72 73 74 75 76 77
*/
 gotoxy(14,12);
 printf("************ ! MAQUINA TRABAJANDO ! ************");

/*
*****************************************************************
*                    Inicio del zig-zagi                        *
*                  sección de cambio de bloque                  *
*****************************************************************
*/

for(a=0;a<1024;a++)
 {
 gotoxy(35,16);
 printf("a= %d",a);
 if(a==0) a=0;     /* cambio de bloque */
 else
 a=a+256;
 fseek(in,a,0);

 /*
 *****************************************************************
 *             Sección para la lectura de los datos              *
 *****************************************************************
 */
 for(j=0;j<8;j++)
  {
  for(k=0;k<8;k++)
   {
   fread(&dato,4,1,in);
   s1=dato;
   e[j][k]=s1;
   }
  }
 /*
```

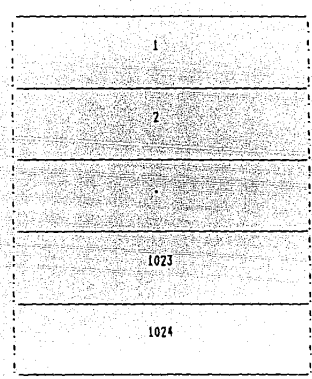
```
/*****************************************************************
*          Finaliza la sección de lectura de los datos          *
*****************************************************************/

/*****************************************************************
*   La siguiente sección hace el reordenamiento de los puntos   *
*****************************************************************
*/

 ,0][0]=e[0][0];              /* linea uno */
n[0][1]=e[0][1];
n[1][0]=e[0][2];
 ,2][0]=e[0][3];
 ,1][1]=e[0][4];
n[0][2]=e[0][5];
 ,0][3]=e[0][6];
 ,1][2]=e[0][7];

n[2][1]=e[1][0];              /* linea dos */
 ,3][0]=e[1][1];
n[4][0]=e[1][2];
n[3][1]=e[1][3];
 ,2][2]=e[1][4];
 ,1][3]=e[1][5];
n[0][4]=e[1][6];
 ,0][5]=e[1][7];

n[1][4]=e[2][0];              /* linea tres */
n[2][3]=e[2][1];
 ,3][2]=e[2][2];
n[4][1]=e[2][3];
n[5][0]=e[2][4];
 ,6][0]=e[2][5];
 ,5][1]=e[2][6];
n[4][2]=e[2][7];

 ,3][3]=e[3][0];              /* linea cuatro */
n[2][4]=e[3][1];
n[1][5]=e[3][2];
 ,0][6]=e[3][3];
 ,0][7]=e[3][4];
n[1][6]=e[3][5];
 ,2][5]=e[3][6];
 ,3][4]=e[3][7];

 ,4][3]=e[4][0];              /* linea cinco */
n[5][2]=e[4][1];
n[6][1]=e[4][2];
n[7][0]=e[4][3];
 ,7][1]=e[4][4];
n[6][2]=e[4][5];
n[5][3]=e[4][6];
 ,4][4]=e[4][7];

n[3][5]=e[5][0];              /* linea seis */
n[2][6]=e[5][1];
 ,1][7]=e[5][2];
n[2][7]=e[5][3];
n[3][6]=e[5][4];
n[4][5]=e[5][5];
n[5][4]=e[5][6];
n[6][3]=e[5][7];

n[7][2]=e[6][0];              /* linea siete */
n[7][3]=e[6][1];
n[6][4]=e[6][2];
n[5][5]=e[6][3];
n[4][6]=e[6][4];
n[3][7]=e[6][5];
n[4][7]=e[6][6];
n[5][6]=e[6][7];

n[6][5]=e[7][0];              /* linea ocho */
n[7][4]=e[7][1];
n[7][5]=e[7][2];
n[6][6]=e[7][3];
n[5][7]=e[7][4];
n[6][7]=e[7][5];
n[7][6]=e[7][6];
n[7][7]=e[7][7];

 /*
 *****************************************************************
 *       Finaliza la sección de reordenamiento de los puntos     *
 *****************************************************************
 */

 /*
 *****************************************************************
 *           Sección de la reescritura de los puntos             *
 *****************************************************************
 */

for(j=0;j<8;j++)
 {
 for(k=0;k<8;k++)
  {
  s2=n[j][k];
  fwrite(&s2,4,1,out);
  }
 }
 /*
 *****************************************************************
 *          Termina sección de reescritura de puntos             *
 *****************************************************************
 */

 }
/*
*****************************************************************
*                  Fin de la sección zig-zagi                   *
*                 Sección de cambio de bloque                   *
*****************************************************************
*/
   fclose(in);
   fclose(out);
   }                          /* Fin del programa */
```

```
/*
*****************************************************************
*  E  IGRAMA PARA HACER LA NORMALIZACION DE LOS COEFICIENTES     *
*  )...NFORMADOS Y REORDENADOS, DE ACUERDO CON LA FUNCION DEL    *
*  "SISTEMA VISUAL HUMANO".                                      *
*  E  E PROGRAMA CORRESPONDE A LOS BLOQUES NUMERO CUATRO Y       *
*  (  ICO DE LA FIGURA 4.4, EN LA TESIS.                         *
*                                                                *
*  PROGRAMA REALIZADO POR:  JUVENTINO ROSAS ESPEJEL.             *
*  (  NTRO DE INVESTIGACION CIENTIFICA Y EDUCACION               *
*  SUPERIOR DE ENSENADA (CICESE).                                *
*  1990.                                                         *
*                                                                *
*****************************************************************
*/


#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>

int s2,s,dd,d1,b[256];

main(int argc, char *argv[])
/
FILE *in;
FILE *out;
FILE *out1;
int subrutina();
float cuant();
float t[8][8];
int s1,xp,ys,dato,s3;
long int a;
int m,j,k,i;
clrscr();

xp=0.0;
ys=0.0;

if(argc!=3)
 {
 gotoxy(5,10);
 printf("te olvidaste de teclear el  nombre  de  alguno");
 gotoxy(5,11);
 printf("de los archivos para procesar (entrada-salida)");
 exit(1);
 }

if((in=fopen(argv[1],"rb"))==NULL)
 {
 gotoxy(5,10);
 printf("no fue posible abrir el archivo de entrada");
 exit(1);
 }

if((out=fopen(argv[2],"wb"))==NULL)
 {
 gotoxy(5,11);
 printf("no fue posible abrir el archivo de salida");
 exit(1);
 }

if((out1=fopen("dos.dat","wb"))==NULL)
 {
 gotoxy(5,11);
 printf("no fue posible abrir el archivo de salida");
 exit(1);
 }
```

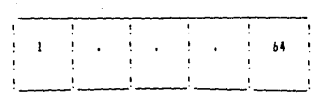
/*

La busqueda de los datos a procesar se hace de acuerdo con la siguiente estrategia:

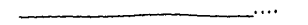
1.- La variable utilizada es: m, que es la variable encargada del cambio del bloque a procesar.

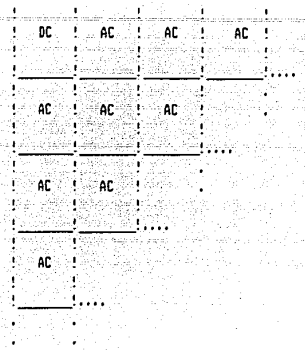


2.-Adicionalmente cada uno de los bloques contiene 64 pixeles ,en la figura siguiente se muestra la ubicacion de los pixeles en un bloque cualquiera de los 1024 posibles.



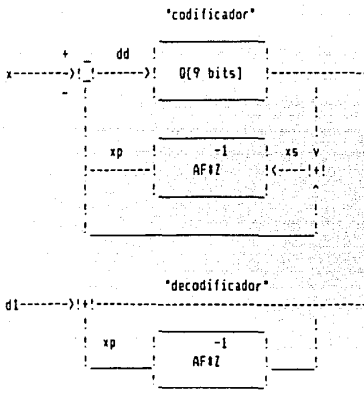
3.- Los puntos del bloque de 8*8 se nombran como componente de CD al primer punto, y los restantes 63 puntos son conocidos como componentes de AC. Esto se muestra en la siguiente figura.





Nota: No se muestra el bloque de 8*8 completo.

4.- Debido a la importancia del primer punto para la transformación coseno inversa, este es codificado en DPCM de primer orden a 9 bits tal como se muestra.



5.- Los restantes 63 puntos, componentes AC, son normalizados de acuerdo con la función de sensitividad de contraste HVS. Y los resultados son redondeados. Se muestra la matriz de normalización HVS(luminancia).

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

./

```
  gotoxy(14,12);
  printf("************ ! MAQUINA TRABAJANDO ! ************");



/*
*****************************************************************
*                    Inicio de la normalización                 *
*                  sección de cambio de  bloque                 *
*****************************************************************
*/
 for(i=1;i<257;i++)
 {
 b[i]=i*0.6328125; /* asignación del tamaño del escalón del cuantiza
 }

for(a=0;a<1024;a++)
 {
 gotoxy(35,16);
 printf("a= %d\n",a);
 if(a==0) a=0;     /* cambio de bloque */
 else
 a=a+256;
 fseek(in,a,0);

 /*
 *****************************************************************
 *              Sección para la lectura de los datos             *
 *****************************************************************
 */
 for(j=0;j<8;j++)
  {
  for(k=0;k<8;k++)
   {
   fread(&dato,4,1,in);
   s1=dato;
   t[j][k]=s1;
   }
  }
 /*
 *****************************************************************
 *           Finaliza la sección de lectura de los datos         *
 *****************************************************************
 */
 /*
 *****************************************************************
 *             Sección de normalización del primer punto         *
 *                    con el codificador DPCM                    *
 *****************************************************************
 */
  dd=t[0][0]-xp;
  d1=cuant();
  xs=d1+xp;
  t[0][0]=d1;
  xp=xs;
 /*
 *****************************************************************
 *      Finaliza la sección de normalización del primer punto    *
 *****************************************************************
```

```
/*****************************************************************
      Sección de normalización de los 63 puntos restantes      *
*****************************************************************
*/
 0][1]=t[0][1]/  1.0;          /* primer linea */
t[0][2]=t[0][2]/  1.0;
t[0][3]=t[0][3]/  1.0;
 0][4]=t[0][4]/  2.0;
.0][5]=t[0][5]/  3.0;
t[0][6]=t[0][6]/  3.0;
 0][7]=t[0][7]/  4.0;

t[1][0]=t[1][0]/  1.0;          /* segunda linea */
*1][1]=t[1][1]/  1.0;
 1][2]=t[1][2]/  1.0;
t[1][3]=t[1][3]/  1.0;
t[1][4]=t[1][4]/  2.0;
 1][5]=t[1][5]/  4.0;
.1][6]=t[1][6]/  4.0;
t[1][7]=t[1][7]/  3.0;

 2][0]=t[2][0]/  1.0;          /* tercer linea */
t[2][1]=t[2][1]/  1.0;
*2][2]=t[2][2]/  1.0;
 2][3]=t[2][3]/  2.0;
t[2][4]=t[2][4]/  3.0;
t[2][5]=t[2][5]/  4.0;
 2][6]=t[2][6]/  4.0;
.2][7]=t[2][7]/  4.0;

 3][0]=t[3][0]/  1.0;          /* cuarta linea */
 3][1]=t[3][1]/  1.0;
t[3][2]=t[3][2]/  1.0;
*3][3]=t[3][3]/  2.0;
 3][4]=t[3][4]/  3.0;
t[3][5]=t[3][5]/  5.0;
t[3][6]=t[3][6]/  5.0;
 3][7]=t[3][7]/  4.0;

t[4][0]=t[4][0]/  1.0;          /* quinta linea */
 4][1]=t[4][1]/  1.0;
.4][2]=t[4][2]/  2.0;
t[4][3]=t[4][3]/  4.0;
*4][4]=t[4][4]/  4.0;
 4][5]=t[4][5]/  7.0;
t[4][6]=t[4][6]/  6.0;
t[4][7]=t[4][7]/  5.0;

 5][0]=t[5][0]/  2.0;          /* sexta linea */
t[5][1]=t[5][1]/  2.0;
 5][2]=t[5][2]/  3.0;
 5][3]=t[5][3]/  4.0;
t[5][4]=t[5][4]/  5.0;
*5][5]=t[5][5]/  7.0;
 5][6]=t[5][6]/  7.0;
t[5][7]=t[5][7]/  6.0;

 6][0]=t[6][0]/  3.0;          /* septima linea */
*6][1]=t[6][1]/  4.0;
t[6][2]=t[6][2]/  5.0;
t[6][3]=t[6][3]/  5.0;
t[6][4]=t[6][4]/  6.0;
t[6][5]=t[6][5]/  8.0;
t[6][6]=t[6][6]/  8.0;
t[6][7]=t[6][7]/  6.0;

t[7][0]=t[7][0]/  5.0;          /* octava linea */
t[7][1]=t[7][1]/  6.0;
t[7][2]=t[7][2]/  6.0;
t[7][3]=t[7][3]/  6.0;
t[7][4]=t[7][4]/  7.0;
t[7][5]=t[7][5]/  6.0;
t[7][6]=t[7][6]/  6.0;
t[7][7]=t[7][7]/  6.0;
/*
*****************************************************************
*  Finaliza sección de normalización de los 63 puntos restantes  *
*****************************************************************
*/

 /*
 *****************************************************************
 *                Sección de escritura de los datos              *
 *****************************************************************
 */
for(j=0;j<8;j++)
 {
 for(k=0;k<8;k++)
  {
  if( (j==0)&&(k==0) )
   {
    s2=t[0][0];
    fwrite(&s2,4,1,out);
    if( (m==545)||(m==768) ) fwrite(&s2,4,1,out1);
   }
  else
   {
   s2=t[j][k];
   s3=subrutina(s2);
   fwrite(&s3,4,1,out);
   }
  }
 }

/*
*****************************************************************
*           Finaliza sección de escritura de los datos          *
*****************************************************************
*/

 }
/*
*****************************************************************
*                    Fin de la normalización                    *
*****************************************************************
*/
   fcloseall();
   fclose(out);
```

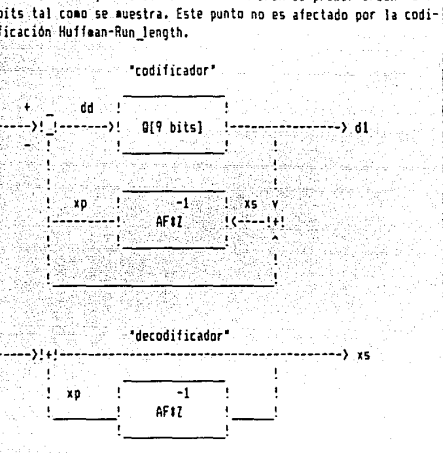
```
   fclose(out1);
   }                        /* Fin del programa */

/*
*************************************************************************
*            Subrutina para el redondeo de los puntos                   *
*************************************************************************
*/


float subrutina()
 {
    float sal,z1,z2;

    sal=s2;
    z1=ceil(sal);
    z2=fabs(z1-sal);
    if(z2<=0.5) s=z1;
    else
    s=floor(sal);
    return s;
 }                          /* Fin de subrutina */

 /*
 *************************************************************************
 *                    Subrutina "Cuantizador fijo "                      *
 *************************************************************************
 */
 float cuant()
  {
  int r,i;

  if(dd<0)
   { r=-1; dd=fabs(dd); }
   else
   { r=1; }
  for(i=1;i<257;i++)
   {
   if(dd<=b[i])
    {
    if(i==1)
     { d1=b[i]/2.0*r; }
     else
     { d1=(b[i]+b[i-1])/2.0*r; }
     goto sal;
    }
   }               /* fin del for */
  d1=255.5*r;
sal: return d1;
 }            /* fin del cuantizador fijo */
```

```
                                                        //*
*************************************************************
   PROGRAMA PARA HACER LA NORMALIZACION INVERSA DE  LOS    *
   COEFICIENTES DECODIFICADOS Y NORMALIZADOS EN LA SEC-    *
*  CION CORRESPONDIENTE AL TRANSMISOR.                     *
*  PARA ESTE CASO SE APLICA EL RECIPROCO DE LA FUNCION     *
   DEL "SISTEMA VISUAL HUMANO".                            *
*  ESTE PROGRAMA CORRESPONDE AL BLOQUE NUMERO NUEVE DE     *
*  LA FIGURA 4.4, EN LA TESIS.                             *
                                                          *
.  PROGRAMA REALIZADO POR:  JUVENTINO ROSAS ESPEJEL.       *
*  CENTRO DE INVESTIGACION CIENTIFICA Y EDUCACION          *
   SUPERIOR DE ENSENADA (CICESE).                          *
   1990.                                                  *
*                                                         *
**************************************************************




   #include<stdio.h>
   #include<stdlib.h>
   #include<conio.h>
   #include<math.h>

   float s;

   main(int argc, char *argv[])
   {
   FILE *in;
   FILE *out;
   float t[8][8];
   float s1,xp,xs,dato,s2,d1;
   long int a;
   int m,j,k,i;
   clrscr();

   xp=0.0;
   xs=0.0;

   if(argc!=3)
    {
    gotoxy(5,10);
    printf("te olvidaste de teclear el  nombre  de  alguno");
    gotoxy(5,11);
    printf("de los archivos para procesar (entrada-salida)");
    exit(1);
    }

   if((in=fopen(argv[1],"rb"))==NULL)
    {
    gotoxy(5,10);
    printf("no fue posible abrir el archivo de entrada");
    exit(1);
    }

   if((out=fopen(argv[2],"wb"))==NULL)
    {
    gotoxy(5,11);
    printf("no fue posible abrir el archivo de salida");
    exit(1);
    }

    }

    gotoxy(14,12);
    printf("************ ! MAQUINA TRABAJANDO ! ************");



   /*
   ************************************************************
   *                    Inicio de la normalización            *
   *                  sección de cambio de  bloque            *
   ************************************************************
   */

   for(m=0;m<1024;m++)
    {
    gotoxy(35,16);
    printf("m= %d\n",m);
    if(m==0) a=0;     /* cambio de bloque */
    else
    a=a+256;
    fseek(in,a,0);

    /*
    ************************************************************
    *              Sección para la lectura de los datos        *
    ************************************************************
    */
    for(j=0;j<8;j++)
     {
     for(k=0;k<8;k++)
      {
      fread(&dato,4,1,in);
      s1=dato;
      t[j][k]=s1;
      }
     }
    /*
    ************************************************************
    *          Finaliza la sección de lectura de los datos     *
    ************************************************************
    */
   /*
   ************************************************************
   *             Sección de normalización del primer punto    *
   *                    con el codificador DPCM               *
   ************************************************************
   */
    d1=t[0][0];
    xs=d1+xp;
    xp=xs;
    t[0][0]=xs;
   /*
   ************************************************************
   *     Finaliza la sección de normalización del primer punto *
   ************************************************************


   ************************************************************
```

```
/*      Sección de normalización de los 63 puntos restantes      */
/*****************************************************************/
/*
t[0][1]=t[0][1]*  1.0;          /* primer linea */
t[0][2]=t[0][2]*  1.0;
t[0][3]=t[0][3]*  1.0;
t[0][4]=t[0][4]*  2.0;
t[0][5]=t[0][5]*  3.0;
t[0][6]=t[0][6]*  3.0;
t[0][7]=t[0][7]*  4.0;

t[1][0]=t[1][0]*  1.0;          /* segunda linea */
t[1][1]=t[1][1]*  1.0;
t[1][2]=t[1][2]*  1.0;
t[1][3]=t[1][3]*  1.0;
t[1][4]=t[1][4]*  2.0;
t[1][5]=t[1][5]*  4.0;
t[1][6]=t[1][6]*  4.0;
t[1][7]=t[1][7]*  3.0;

t[2][0]=t[2][0]*  1.0;          /* tercer linea */
t[2][1]=t[2][1]*  1.0;
t[2][2]=t[2][2]*  1.0;
t[2][3]=t[2][3]*  2.0;
t[2][4]=t[2][4]*  3.0;
t[2][5]=t[2][5]*  4.0;
t[2][6]=t[2][6]*  4.0;
t[2][7]=t[2][7]*  4.0;

t[3][0]=t[3][0]*  1.0;          /* cuarta linea */
t[3][1]=t[3][1]*  1.0;
t[3][2]=t[3][2]*  1.0;
t[3][3]=t[3][3]*  2.0;
t[3][4]=t[3][4]*  3.0;
t[3][5]=t[3][5]*  5.0;
t[3][6]=t[3][6]*  5.0;
t[3][7]=t[3][7]*  4.0;

t[4][0]=t[4][0]*  1.0;          /* quinta linea */
t[4][1]=t[4][1]*  1.0;
t[4][2]=t[4][2]*  2.0;
t[4][3]=t[4][3]*  4.0;
t[4][4]=t[4][4]*  4.0;
t[4][5]=t[4][5]*  7.0;
t[4][6]=t[4][6]*  6.0;
t[4][7]=t[4][7]*  5.0;

t[5][0]=t[5][0]*  2.0;          /* sexta linea */
t[5][1]=t[5][1]*  2.0;
t[5][2]=t[5][2]*  3.0;
t[5][3]=t[5][3]*  4.0;
t[5][4]=t[5][4]*  5.0;
t[5][5]=t[5][5]*  7.0;
t[5][6]=t[5][6]*  7.0;
t[5][7]=t[5][7]*  6.0;

t[6][0]=t[6][0]*  3.0;          /* septima linea */
t[6][1]=t[6][1]*  4.0;
t[6][2]=t[6][2]*  5.0;
t[6][3]=t[6][3]*  5.0;
t[6][4]=t[6][4]*  6.0;
t[6][5]=t[6][5]*  8.0;
t[6][6]=t[6][6]*  8.0;
t[6][7]=t[6][7]*  6.0;

t[7][0]=t[7][0]*  5.0;          /* octava linea */
t[7][1]=t[7][1]*  6.0;
t[7][2]=t[7][2]*  6.0;
t[7][3]=t[7][3]*  6.0;
t[7][4]=t[7][4]*  7.0;
t[7][5]=t[7][5]*  6.0;
t[7][6]=t[7][6]*  6.0;
t[7][7]=t[7][7]*  6.0;
/*
*****************************************************************
*  Finaliza sección de normalización de los 63 puntos restantes  *
*****************************************************************
*/

 /*
 *****************************************************************
 *               Sección de escritura de los datos               *
 *****************************************************************
 */
for(j=0;j<8;j++)
 {
 for(k=0;k<8;k++)
  {
  s2=t[j][k];
  fwrite(&s2,4,1,out);
  }
 }

/*
*****************************************************************
*          Finaliza sección de escritura de los datos           *
*****************************************************************
*/

 }
/*
*****************************************************************
*                    Fin de la normalización                    *
*****************************************************************
*/
   fclose(in);
   fclose(out);
   }                        /* Fin del programa */
```

```
//*
***  ************************************************************
*  ...OGRAMA PARA REALIZAR LA CODIFICACION CONJUNTA  HUFFMAN    *
*  RUN-LENGTH DE LOS COEFICIENTES REORDENADOS, NORMALIZADOS    *
*    CUANTIZADOS.                                              *
*    TE PROGRAMA CORRESPONDE AL BLOQUE NUMERO SEIS DE LA       *
*  FIGURA 4.4, EN LA TESIS.                                    *
*                                                              *
*    OGRAMA REALIZADO POR:  JUVENTINO ROSAS ESPEJEL.           *
*  ENTRO DE INVESTIGACION CIENTIFICA Y EDUCACION               *
*  SUPERIOR DE ENSENADA (CICESE).                              *
*    90.                                                       *
*                                                              *
**************************************************************
*/


   #include<stdio.h>
    nclude<stdlib.h>
   #include<conio.h>
   #include<math.h>
    nclude<string.h>

   main(int argc, char *argv[])
   {
    LE *in;
   FILE *out;
   char num[80],signo[80],mag[80],eob[80];
    t t[8][8],tjk,tjk1;
   float dato,s1,s2;
   long int b,byte;
    t m,j,k,i,count,count1;
    t m1,flag,flag1;
   clrscr();

    (argc!=3)
    {
   gotoxy(5,10);
    rintf("te olvidaste de teclear el  nombre  de  alguno");
   gotoxy(5,11);
   printf("de los archivos para procesar (entrada-salida)");
    xit(1);
    }

    ((in=fopen(argv[1],"rb"))==NULL)
    {
   gotoxy(5,10);
   printf("no fue posible abrir el archivo de entrada");
    xit(1);
    }

    ((out=fopen(argv[2],"w"))==NULL)
    {
   gotoxy(5,11);
   printf("no fue posible abrir el archivo de salida");
    xit(1);
    }

/*
```

1.- Debido a la importancia del primer punto para la transformación coseno inversa, este es codificado en DPCM de primer orden a 9 bits tal como se muestra. Este punto no es afectado por la codificación Huffman-Run_length.



2.- Los restantes 63 puntos, componentes AC, son normalizados de acuerdo con la función de sensitividad de contraste HVS. Y los resultados son redondeados. Se muestra la matriz de normalización HVS(luminancia).

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

3.-En este programa se hace la codificación de Huffman conjuntamente con la codificación Run-lengt (corridas de ceros) de los datos resultantes de la HVS. Para esto es necesario que cada número pertenezca a cuatro parámetros:

1.- Posición relativa con respecto al último número diferente de cero.
2.- Categoria del número (rango).
3.- Signo del número.
4.- Magnitud del número.

```
*/
   gotoxy(14,12);
   printf("************ ! MAQUINA TRABAJANDO ! ************");


/*
**************************************************************
```

```
/*                    Inicio de la codificación                    */
/*                  sección de cambio de bloque                    */
/*****************************************************************/
b=0;
byte=0;
for (a=0;a<1024;a++)
 {
gotoxy(35,16);
printf("a= %d\n",a);

 /*
 /*****************************************************************/
 /*              Sección para la lectura de los datos             */
 /*****************************************************************/
 for (j=0;j<8;j++)
  {
  for(k=0;k<8;k++)

    read(&dato,4,1,in);
   s1=dato;

    if( (j==0)&&(k==0) )
     { s2=s1; }          /* se conserva flotante el número */
    else
     {
      if(s1>=128)
       { s1=127;  b=b+1; }
      if(s1<=-128)
       { s1=-127; b=b+1; }
      t[j][k]=s1;       /*     se hace entero el número     */
     }

  }
 /*
 /*****************************************************************/
 /*          Finaliza la sección de lectura de los datos          */
 /*****************************************************************/


/*
/*****************************************************************/
/*            Sección de cálculo de la codificación              */
/*****************************************************************/
 count=0;
 count1=0;
 flag=0;
 flag1=0;
 for(j=0;j<8;j++)
  {
  for(k=0;k<8;k++)

   if( (j==0)&&(k==0) )        /* if(1) */
    {
     fwrite(&s2,4,1,out);     /* se escribe el componente de DC */
     printf("DC--> %f ",s2);
     byte=byte+4;
    }
   else
    {
     a1=t[j][k];              /* a1 es de tipo entero */
     count1=count;            /* count1 guarda la contabilidad */
                              /* de ceros anticipatorios       */

     switch(a1)               /* switch(1) */
      {
      case 0:
       count=count+1;         /* contabilidad del número de ceros */
       break;
      default:                /* se detecta un número diferente de cero */
       count=0;

       tjk1=t[j][k];          /* tjk1 es de tipo entero */
       if(tjk1<0)
         { strcpy(signo,"0"); }
        else
         { strcpy(signo,"1"); }

       tjk=abs(t[j][k]);
/*
/*****************************************************************/
/*                    Sección categoria 1                        */
/*****************************************************************/
       if(tjk==1)
         {
         switch(count1)
          {
          case 0:                      /* caso 0 ceros anticipatorios */
            strcpy(num,"00");
            break;
          case 1:                      /* caso de un cero anticipatori
            strcpy(num,"110");
            break;
          case 2:                      /* caso 2 ceros anticipatorios
            strcpy(num,"01010");
            break;
          case 3:                      /* caso 3 ceros anticipatorios
            strcpy(num,"01111");
            break;
          case 4:                      /* caso 4 ceros anticipatorios
            strcpy(num,"1111");
            break;
          case 5:                      /* caso 5 ceros anticipatorios
            strcpy(num,"01110");
            break;
          case 6:                      /* caso 6 ceros anticipatorios
            strcpy(num,"10111");
            break;
          case 7:                      /* caso 7 ceros anticipatorios
            strcpy(num,"011001");
            break;
          case 8:                      /* caso 8 ceros anticipatorios
            strcpy(num,"01100001");
```

```
      break;
    case 9:                       /* caso 9 ceros anticipatorios *
      strcpy(num,"011000001");
      break;
    case 10:                      /* caso 10 ceros anticipatorios
      strcpy(num,"111000001");
      break;
    case 11:                      /* caso 11 ceros anticipatorios
      strcpy(num,"0110000111");
      break;
    case 12:                      /* caso 12 ceros anticipatorios
      strcpy(num,"11100111");
      break;
    case 13:                      /* caso 13 ceros anticipatorios
      strcpy(num,"11100001");
      break;
    case 14:                      /* caso 14 ceros anticipatorios
      strcpy(num,"11100110");
      break;
    case 15:                      /* caso 15 ceros anticipatorios
      strcpy(num,"111000000");
      break;
    default:
      flag=1;
      break;
    }
   }
/*
*****************************************************************
*               Finaliza sección categoria 1                    *
*****************************************************************
*/

/*
*****************************************************************
*                    Sección categoria 2                        *
*****************************************************************
*/
if( (tjk==2)||(tjk==3) )
 {
 switch(count1)
  {
    case 0:                       /* caso 0 ceros anticipatorios *
      strcpy(num,"100");
      break;
    case 1:                       /* caso de un cero anticipatori
      strcpy(num,"11101");
      break;
    case 2:                       /* caso 2 ceros anticipatorios *
      strcpy(num,"1110001");
      break;
    case 3:                       /* caso 3 ceros anticipatorios *
      strcpy(num,"1011011");
      break;
    case 4:                       /* caso 4 ceros anticipatorios *
      strcpy(num,"1011010");
      break;
    case 5:                       /* caso 5 ceros anticipatorios *
      strcpy(num,"10110011");
      break;
    default:
      flag=1;
      break;
  }
 switch(tjk)                      /* selección de la magnitud */
  {
    case 2:
      strcpy(mag,"0");
      break;
    case 3:
      strcpy(mag,"1");
      break;
  }
 }

/*
*****************************************************************
*               Finaliza sección categoria 2                    *
*****************************************************************
*/

/*
*****************************************************************
*                    Sección categoria 3                        *
*****************************************************************
*/
if( (tjk>=4)&&(tjk<=7) )
 {
 switch(count1)
  {
    case 0:                       /* caso 0 ceros anticipatorios *
      strcpy(num,"0100");
      break;
    case 1:                       /* caso de un cero anticipatorio
      strcpy(num,"1011000");
      break;
    case 2:                       /* caso 2 ceros anticipatorios *
      strcpy(num,"011000010");
      break;
    case 3:                       /* caso 3 ceros anticipatorios *
      strcpy(num,"0110000110");
      break;
    default:
      flag=1;
      break;
  }

 switch(tjk)                      /* selección de la magnitud */
  {
    case 4:
      strcpy(mag,"00");
      break;
    case 5:
      strcpy(mag,"01");
```

```
        break;
      case 6:
        strcpy(mag,"10");
        break;
      case 7:
        strcpy(mag,"11");
        break;
  }

 }

 /*
 ************************************************************************
 *                Finaliza sección categoria 3                          *
 ************************************************************************
 */


 /*
 ************************************************************************
                        Sección categoria 4                             *
 ************************************************************************
 */
 if( (tjk>=8)&&(tjk<=15) )
 {
 switch(count1)
  {
        case 0:                         /* caso 0 ceros anticipatorios
          strcpy(num,"01011");
          break;
        case 1:                         /* caso de un cero anticipatori
          strcpy(num,"011000000");
          break;
        default:
          flag=1;
          break;
  }

 switch(tjk)                            /* selección de la magnitud */
  {
        case 8:
          strcpy(mag,"000");
          break;
        case 9:
          strcpy(mag,"001");
          break;
        case 10:
          strcpy(mag,"010");
          break;
        case 11:
          strcpy(mag,"011");
          break;
        case 12:
          strcpy(mag,"100");
          break;
        case 13:
          strcpy(mag,"101");
          break;
        case 14:
          strcpy(mag,"110");
          break;
        case 15:
          strcpy(mag,"111");
          break;
  }

 }

 /*
 ************************************************************************
 *                Finaliza sección categoria 4                          *
 ************************************************************************
 */

 /*
 ************************************************************************
 *                      Sección categoria 5                             *
 ************************************************************************
 */
 if( (tjk>=16)&&(tjk<=31) )
 {
  switch(count1)
  {
        case 0:                         /* caso 0 ceros anticipatorio
          strcpy(num,"011010");
          break;
        default:
          flag=1;
          break;
  }


 switch(tjk)                            /* selección de la magnitud */
  {
        case 16:
          strcpy(mag,"0000");
          break;
        case 17:
          strcpy(mag,"0001");
          break;
        case 18:
          strcpy(mag,"0010");
          break;
        case 19:
          strcpy(mag,"0011");
          break;
        case 20:
          strcpy(mag,"0100");
          break;
        case 21:
          strcpy(mag,"0101");
          break;
        case 22:
          strcpy(mag,"0110");
```

```
          break;
        case 23:
          strcpy(mag,"0111");
          break;
        case 24:
          strcpy(mag,"1000");
          break;
        case 25:
          strcpy(mag,"1001");
          break;
        case 26:
          strcpy(mag,"1010");
          break;
        case 27:
          strcpy(mag,"1011");
          break;
        case 28:
          strcpy(mag,"1100");
          break;
        case 29:
          strcpy(mag,"1101");
          break;
        case 30:
          strcpy(mag,"1110");
          break;
        case 31:
          strcpy(mag,"1111");
          break;
  }


 }
/*
*************************************************************************
*                  Finaliza sección categoria 5                         *
*************************************************************************
*/

/*
*************************************************************************
*                       Sección categoria 6                             *
*************************************************************************
*/
if( (tjk)=32)&&(tjk(=63) )
 {
  switch(count1)
  {
        case 0:                         /* caso 0 ceros anticipatorios */
          strcpy(num,"1110010");
          break;
        default:
          flag=1;
          break;
  }


 switch(tjk)                            /* selección de la magnitud */
  {
        case 32:
          strcpy(mag,"00000");
          break;
        case 33:
          strcpy(mag,"00001");
          break;
        case 34:
          strcpy(mag,"00010");
          break;
        case 35:
          strcpy(mag,"00011");
          break;
        case 36:
          strcpy(mag,"00100");
          break;
        case 37:
          strcpy(mag,"00101");
          break;
        case 38:
          strcpy(mag,"00110");
          break;
        case 39:
          strcpy(mag,"00111");
          break;
        case 40:
          strcpy(mag,"01000");
          break;
        case 41:
          strcpy(mag,"01001");
          break;
        case 42:
          strcpy(mag,"01010");
          break;
        case 43:
          strcpy(mag,"01011");
          break;
        case 44:
          strcpy(mag,"01100");
          break;
        case 45:
          strcpy(mag,"01101");
          break;
        case 46:
          strcpy(mag,"01110");
          break;
        case 47:
          strcpy(mag,"01111");
          break;
        case 48:
          strcpy(mag,"10000");
          break;
        case 49:
          strcpy(mag,"10001");
          break;
        case 50:
          strcpy(mag,"10010");
          break;
        case 51:
          strcpy(mag,"10011");
```

```
        break;
      case 52:
        strcpy(aag,"10100");
        break;
      case 53:
        strcpy(aag,"10101");
        break;
      case 54:
        strcpy(aag,"10110");
        break;
      case 55:
        strcpy(aag,"10111");
        break;
      case 56:
        strcpy(aag,"11000");
        break;
      case 57:
        strcpy(aag,"11001");
        break;
      case 58:
        strcpy(aag,"11010");
        break;
      case 59:
        strcpy(aag,"11011");
        break;
      case 60:
        strcpy(aag,"11100");
        break;
      case 61:
        strcpy(aag,"11101");
        break;
      case 62:
        strcpy(aag,"11110");
        break;
      case 63:
        strcpy(aag,"11111");
        break;


 }
/*
** !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
*                 Finaliza secció
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
*/

/*
** !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
*                   Sección cat
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
*/
if (tjk>=64)&&(tjk<=127) )
 {
  switch(count1)

        case 0:
                strcpy(num,"10110010");
                break;
        default:
                flag=1;
                break;
 }


switch(tjk)
 {
        case 64:
          strcpy(aag,"000000");
          break;
        case 65:
          strcpy(aag,"000001");
          break;
        case 66:
          strcpy(aag,"000010");
          break;
        case 67:
          strcpy(aag,"000011");
          break;
        case 68:
          strcpy(aag,"000100");
          break;
        case 69:
          strcpy(aag,"000101");
          break;
        case 70:
          strcpy(aag,"000110");
          break;
        case 71:
          strcpy(aag,"000111");
          break;
        case 72:
          strcpy(aag,"001000");
          break;
        case 73:
          strcpy(aag,"001001");
          break;
        case 74:
          strcpy(aag,"001010");
          break;
        case 75:
          strcpy(aag,"001011");
          break;
        case 76:
          strcpy(aag,"001100");
          break;
        case 77:
          strcpy(aag,"001101");
          break;
        case 78:
          strcpy(aag,"001110");
          break;
        case 79:
          strcpy(aag,"001111");
          break;
        case 80:
          strcpy(aag,"010000");
          break;
        case 81:
          strcpy(aag,"010001");
          break;
        case 82:
          strcpy(aag,"010010");
          break;
        case 83:
          strcpy(aag,"010011");
          break;
        case 84:
          strcpy(aag,"010100");
          break;
        case 85:
          strcpy(aag,"010101");
          break;
        case 86:
          strcpy(aag,"010110");
          break;
        case 87:
          strcpy(aag,"010111");
          break;
        case 88:
          strcpy(aag,"011000");
          break;
        case 89:
          strcpy(aag,"011001");
          break;
        case 90:
          strcpy(aag,"011010");
          break;
        case 91:
          strcpy(aag,"011011");
          break;
        case 92:
          strcpy(aag,"011100");
          break;
        case 93:
          strcpy(aag,"011101");
          break;
        case 94:
          strcpy(aag,"011110");
          break;
        case 95:
          strcpy(aag,"011111");
          break;
        case 96:
          strcpy(aag,"100000");
          break;
        case 97:
          strcpy(aag,"100001");
          break;
        case 98:
          strcpy(aag,"100010");
          break;
        case 99:
          strcpy(aag,"100011");
          break;
        case 100:
          strcpy(aag,"100100");
          break;
        case 101:
          strcpy(aag,"100101");
          break;
        case 102:
          strcpy(aag,"100110");
          break;
        case 103:
          strcpy(aag,"100111");
          break;
        case 104:
          strcpy(aag,"101000");
          break;
        case 105:
          strcpy(aag,"101001");
          break;
        case 106:
          strcpy(aag,"101010");
          break;
        case 107:
          strcpy(aag,"101011");
          break;
        case 108:
          strcpy(aag,"101100");
          break;
        case 109:
          strcpy(aag,"101101");
          break;
        case 110:
          strcpy(aag,"101110");
          break;
        case 111:
          strcpy(aag,"101111");
          break;
        case 112:
          strcpy(aag,"110000");
          break;
        case 113:
          strcpy(aag,"110001");
          break;
        case 114:
          strcpy(aag,"110010");
          break;
        case 115:
          strcpy(aag,"110011");
          break;
        case 116:
          strcpy(aag,"110100");
          break;
        case 117:
          strcpy(aag,"110101");
          break;
        case 118:
          strcpy(aag,"110110");
          break;
        case 119:
          strcpy(aag,"110111");
```

```
          break;
        case 120:
          strcpy(mag,"111000");
          break;
        case 121:
          strcpy(mag,"111001");
          break;
        case 122:
          strcpy(mag,"111010");
          break;
        case 123:
          strcpy(mag,"111011");
          break;
        case 124:
          strcpy(mag,"111100");
          break;
        case 125:
          strcpy(mag,"111101");
          break;
        case 126:
          strcpy(mag,"111110");
          break;
        case 127:
          strcpy(mag,"111111");
          break;
  }

 }
/*
*************************************************************************
*                  Finaliza sección categoria 7                         *
*************************************************************************
*/

/*
*************************************************************************
*       Sección categoria de todas las otras combinaciones              *
*************************************************************************
*/
if( (flag==1)||(tjk==128) )       /* se checa si no hubo codificación */
 (                                /* para entrar dentro de esta categori

          strcpy(num,"011011");
          flag=0;                 /* se reactiva la bandera  */
          flag1=1;                /* bandera para hacer la   */
                                  /* escritura de magnitud 1 */
                                  /* de esta sección.        */

 switch(count1)                      /* selección de la magnitud */
  {
        case 1:
          strcpy(mag,"0000000");
          break;
        case 2:
          strcpy(mag,"0000001");
          break;
        case 3:
          strcpy(mag,"0000010");
          break;
        case 4:
          strcpy(mag,"0000011");
          break;
        case 5:
          strcpy(mag,"0000100");
          break;
        case 6:
          strcpy(mag,"0000101");
          break;
        case 7:
          strcpy(mag,"0000110");
          break;
        case 8:
          strcpy(mag,"0000111");
          break;
        case 9:
          strcpy(mag,"0001000");
          break;
        case 10:
          strcpy(mag,"0001001");
          break;
        case 11:
          strcpy(mag,"0001010");
          break;
        case 12:
          strcpy(mag,"0001011");
          break;
        case 13:
          strcpy(mag,"0001100");
          break;
        case 14:
          strcpy(mag,"0001101");
          break;
        case 15:
          strcpy(mag,"0001110");
          break;
        case 16:
          strcpy(mag,"0001111");
          break;
        case 17:
          strcpy(mag,"0010000");
          break;
        case 18:
          strcpy(mag,"0010001");
          break;
        case 19:
          strcpy(mag,"0010010");
          break;
        case 20:
          strcpy(mag,"0010011");
          break;
        case 21:
          strcpy(mag,"0010100");
          break;
        case 22:
          strcpy(mag,"0010101");
```

```
    break;
  case 23:
    strcpy(mag,"0010110");
    break;
  case 24:
    strcpy(mag,"0010111");
    break;
  case 25:
    strcpy(mag,"0011000");
    break;
  case 26:
    strcpy(mag,"0011001");
    break;
  case 27:
    strcpy(mag,"0011010");
    break;
  case 28:
    strcpy(mag,"0011011");
    break;
  case 29:
    strcpy(mag,"0011100");
    break;
  case 30:
    strcpy(mag,"0011101");
    break;
  case 31:
    strcpy(mag,"0011110");
    break;
  case 32:
    strcpy(mag,"0011111");
    break;
  case 33:
    strcpy(mag,"0100000");
    break;
  case 34:
    strcpy(mag,"0100001");
    break;
  case 35:
    strcpy(mag,"0100010");
    break;
  case 36:
    strcpy(mag,"0100011");
    break;
  case 37:
    strcpy(mag,"0100100");
    break;
  case 38:
    strcpy(mag,"0100101");
    break;
  case 39:
    strcpy(mag,"0100110");
    break;
  case 40:
    strcpy(mag,"0100111");
    break;
  case 41:
    strcpy(mag,"0101000");
    break;
  case 42:
    strcpy(mag,"0101001");
    break;
  case 43:
    strcpy(mag,"0101010");
    break;
  case 44:
    strcpy(mag,"0101011");
    break;
  case 45:
    strcpy(mag,"0101100");
    break;
  case 46:
    strcpy(mag,"0101101");
    break;
  case 47:
    strcpy(mag,"0101110");
    break;
  case 48:
    strcpy(mag,"0101111");
    break;
  case 49:
    strcpy(mag,"0110000");
    break;
  case 50:
    strcpy(mag,"0110001");
    break;
  case 51:
    strcpy(mag,"0110010");
    break;
  case 52:
    strcpy(mag,"0110011");
    break;
  case 53:
    strcpy(mag,"0110100");
    break;
  case 54:
    strcpy(mag,"0110101");
    break;
  case 55:
    strcpy(mag,"0110110");
    break;
  case 56:
    strcpy(mag,"0110111");
    break;
  case 57:
    strcpy(mag,"0111000");
    break;
  case 58:
    strcpy(mag,"0111001");
    break;
  case 59:
    strcpy(mag,"0111010");
    break;
  case 60:
    strcpy(mag,"0111011");
    break;
  case 61:
    strcpy(mag,"0111100");
    break;
  case 62:
    strcpy(mag,"0111101");
```

```
        break;
      case 63:
        strcpy(mag,"0111110");
        break;
      case 64:
        strcpy(mag,"0111111");
        break;
  }

 /
/*
  ************************************************************************
    Finaliza sección categoria de todas las otras combinaciones   *
************************************************************************
..

  ******************** se escriben los datos **********************/
                          fprintf(out,"%s",num);
                          printf("%s ",num);
                          byte=byte+strlen(num);
                          fprintf(out,"%s",signo);
                          printf("%s ",signo);
                          byte=byte+strlen(signo);
  if( (tjk!=1)&&(flag1==1) )
                         {
                          fprintf(out,"%s",mag);
                          printf("%s  ",mag);
                          byte=byte+strlen(mag);
                         }
                          flag1=0;    /* se reinicializa flag1 */
..************************************************************/

     break;                 /* break del switch(1) */

    } /* end del switch(1) */
   }  /* end del if(1)     */

  }   /* end del for "k"   */
 }    /* end del for "j"   */


..
************************************************************************
  Como ya no se detectan números diferentes de cero, se escribe  *
                  el final del bloque                            *
************************************************************************
*/

  !
  strcpy(eob,"1010");
  fprintf(out,"%s",eob);
  printf("%s\n\n\n",eob);
  byte=byte+strlen(eob);


 ************************************************************************
*        Finaliza sección de cálculo de la codificación          *
**************************************************************************


 }
 intf("\n\n\n\n");
 intf("          El número de datos con valor superior                a 128 fue:
                                                                      %d\n",b);
/*
************************************************************************
*                  Fin de la codificación                              *
*               y sección de cambio de bloque                          *
************************************************************************
*/
   fclose(in);
   fclose(out);
   }                    /* Fin del programa */

/*
************************************************************************
* PROGRAMA PARA REALIZAR LA DECODIFICACION CONJUNTA  HUFFMAN   *
* RUN-LENGTH DE LOS DATOS GENERADOS EN LA SECCION CORRESPON-   *
* DIENTE AL TRANSMISOR.                                        *
* ESTE PROGRAMA LEE UN SOLO DATO (0 ó 1) A LA VEZ Y SE COMPARA *
* CON LOS CODIGOS HUFFMAN RESPECTIVOS, SI EL DATO NO SE APAREA *
* CON NINGUN CODIGO, SE LEE UN NUEVO DATO, SE ADICIONA AL  AN- *
```

```
/*****************************************************************
* PROGRAMA PARA REALIZAR LA DECODIFICACION CONJUNTA  HUFFMAN     *
  RUN-LENGTH DE LOS DATOS GENERADOS EN LA SECCION CORRESPON-     *
  DIENTE AL TRANSMISOR.                                          *
* ESTE PROGRAMA LEE UN SOLO DATO (0 ó 1) A LA VEZ Y SE COMPARA   *
  CON LOS CODIGOS HUFFMAN RESPECTIVOS, SI EL DATO NO SE APAREA   *
  CON NINGUN CODIGO, SE LEE UN NUEVO DATO, SE ADICIONA AL  AN-   *
* TERIOR Y SE VUELVE A REALIZAR LA COMPARACION CON LOS CODIGOS   *
* HUFFMAN.  SI NO HAY APAREAMIENTO EL PROCESO SE REPITE HASTA    *
  OBTENERLO.                                                     *
*                                                                *
* ESTE PROGRAMA CORRESPONDE AL BLOQUE NUMERO OCHO DE LA          *
  FIGURA 4.4, DE LA TESIS.                                       *
*                                                                *
* PROGRAMA REALIZADO POR: JUVENTINO BOSAS ESPEJEL.               *
  CENTRO DE INVESTIGACION CIENTIFICA Y EDUCACION                 *
  SUPERIOR DE ENSENADA (CICESE).                                 *
* 1990.                                                          *
*                                                                *
*****************************************************************
*/


  #include<stdio.h>
  #include<stdlib.h>
  #include<conio.h>
  #include<math.h>
  #include<string.h>


  int ceros,num,num1,count,flag,num2,flag2,signo;
  char d1[2],d2[80];
  unsigned char ch;
  FILE *in;
  FILE *out;
  FILE *in1;
  char *p;
  float mag;
  long int byte;

  main(int argc, char *argv[])
  {
  int sub();
  int mag1();
  int signo1();
  float dato,a2;
  int m,j,k,i;
  clrscr();

  if(argc!=3)
   {
   gotoxy(5,10);
   printf("te olvidaste de teclear el  nombre  de  alguno");
   gotoxy(5,11);
   printf("de los archivos para procesar (entrada-salida)");
   exit(1);
   }

  if((in=fopen(argv[1],"rb"))==NULL)
   {
   gotoxy(5,10);
   printf("no fue posible abrir el archivo de entrada");
   exit(1);
   }

  if((in1=fopen("dos.dat","rb"))==NULL)
   {
   gotoxy(5,10);
   printf("no fue posible abrir el archivo de entrada dos.dat");
   exit(1);
   }

  if((out=fopen(argv[2],"wb"))==NULL)
   {
   gotoxy(5,11);
   printf("no fue posible abrir el archivo de salida");
   exit(1);
   }

/*

     En este programa se hace la codificación de Huffman con-
     juntamente con la codificación Run-lengt (corridas de ce-
     ros) de los datos resultantes  de  la HVS.  Para esto es
     necesario que cada número pertenezca a cuatro parámetros:

        1.- Posición relativa con respecto al último  número
            diferente de cero.
        2.- Categoría del número (rango).
        3.- Signo del número.
        4.- Magnitud del número.

*/
   gotoxy(14,12);
   printf("************ ! MAQUINA TRABAJANDO ! ************");


  /*
  ****************************************************************
  *                  Inicio de la decodificación                 *
  *                  sección de cambio de bloque                 *
  ****************************************************************
  */

  byte=0;
  for(m=0;m<1024;m++)          /* for (1) */
   {
   gotoxy(35,16);
   printf("m= %d",m);

  /*
  ****************************************************************
  *             Sección de cálculo de la codificación            *
  ****************************************************************
  */
    flag2=0;
    a2=0;
```

```
count=0;
s  :py(d1,"0");              /* se inicializa d1 */

do                           /*    do-while(1)   */


 if(count==0)                /* if(1) */
  {
   fwrite(&s2,4,1,out);      /* se escribe el componente de DC */
   count=count+1;
  }
   se
  {
   ch=getc(in);      byte=byte+1;
   p=&ch;
   d1[0]=*p;

   if(flag==0)
    {
     strcpy(d2,d1);
     flag=1;
    }
   else
    { strcat(d2,d1); }

      if(!(strcmp(d2,"00")))              /* caso 0 ceros anticipato
        {
        ceros=0;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"110")))        /* caso 1 cero   anticipat
        {
        ceros=1;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"01010")))      /* caso 2 ceros anticipato
        {
        ceros=2;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"01111")))      /* caso 3 ceros anticipato
        {
        ceros=3;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"1111")))       /* caso 4 ceros anticipato
        {
        ceros=4;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"01110")))      /* caso 5 ceros anticipato
        {
        ceros=5;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"10111")))      /* caso 6 ceros anticipato
        {
        ceros=6;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"011001")))     /* caso 7 ceros anticipato
        {
        ceros=7;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"0110001")))    /* caso 8 ceros anticipato
        {
        ceros=8;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"0110000001")))  /* caso 9 ceros anticipato
        {
        ceros=9;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"1110000001")))  /* caso 10 ceros anticipat
        {
        ceros=10;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"01100000111")))/* caso 11 ceros anticipat
        {
        ceros=11;
        mag=1.0*signo1();
        sub();
        goto fin;
        }
      else if(!(strcmp(d2,"111000111")))   /* caso 12 ceros anticipat
        {
        ceros=12;
        mag=1.0*signo1();
        sub();
        goto fin;
```

```
  }
else if(!(strcmp(d2,"11100001")))   /* caso 13 ceros anticipato
  {
  ceros=13;
  mag=1.0*signo1();
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"11100110")))   /* caso 14 ceros anticipato
  {
  ceros=14;
  mag=1.0*signo1();
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"111000000")))  /* caso 15 ceros anticipato
  {
  ceros=15;
  mag=1.0*signo1();
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"100")))        /* caso 0 ceros anticipator
  {
  ceros=0;
  num=1;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"11101")))      /* caso  de  1  anticipator
  {
  ceros=1;
  num=1;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"1110001")))    /* caso 2 ceros anticipator
  {
  ceros=2;
  num=1;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"1011011")))    /* caso 3 ceros anticipator
  {
  ceros=3;
  num=1;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"1011010")))    /* caso 4 ceros anticipator
  {
  ceros=4;
  num=1;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"10110011")))   /* caso 5 ceros anticipato
  {
  ceros=5;
  num=1;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"0100")))       /* caso 0 ceros anticipato
  {
  ceros=0;
  num=2;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"1011000")))    /* caso  1  cero anticipat
  {
  ceros=1;
  num=2;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"011000010")))  /* caso 2 ceros anticipato
  {
  ceros=2;
  num=2;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"0110000110"))) /* caso 3 ceros anticipat
  {
  ceros=3;
  num=2;
  mag=magl(num);
  sub();
  goto fin;
  }
else if(!(strcmp(d2,"01011")))      /* caso 0 ceros anticipat
  {
  ceros=0;
  num=3;
  mag=magl(num);
  sub();
  goto fin;
  }

else if(!(strcmp(d2,"011000000")))  /* caso  1  cero anticipa
  {
  ceros=1;
  num=3;
  mag=magl(num);
  sub();
  goto fin;
  }
```

```
    else if(!(strcmp(d2,"011010")))     /* caso 0 ceros anticipato
      {
      ceros=0;
      num=4;
      mag=mag1(num);
      sub();
      goto fin;
      }
    else if(!(strcmp(d2,"1110010")))    /* caso 0 ceros anticipato
      {
      ceros=0;
      num=5;
      mag=mag1(num);
      sub();
      goto fin;
      }
    else if(!(strcmp(d2,"10110010")))   /* caso 0 ceros anticipato
      {
      ceros=0;
      num=6;
      mag=mag1(num);
      sub();
      goto fin;
      }
    else if(!(strcmp(d2,"011011")))     /* caso del resto de combi
      {
      num=7;
      ceros=mag1(num);

      if(ceros==1)
       { mag=30*signo; }
      else
       {
       if( (ceros<10)&&(ceros>1) )
        { mag=12*signo; }               /* se asigna magnitud sig
       else                             /* de manera arbitraria,
        { mag=2*signo;  }               /*   con bases estadistic
       }

      sub();
      goto fin;
      }
    else if(!(strcmp(d2,"1010")))       /*     fin del block (EOB)
      {
      flag2=1;
      if(count<64)
       {
       ceros=64-count;
       mag=0;
       sub();
       }
      strcpy(d2,"");                    /*  se reinicializa d2  */
      flag=0;                           /* se reinicializa flag */
      goto fin;
      }
  }                                     /*       end del if(1)

fin: continue;          /* linea para continuar el programa sin cambio
    } while(flag2==0);    /*    end del do-while(1)        */
/*
*****************************************************************
*         Finaliza sección de cálculo de la codificación        *
*****************************************************************
*/

  }      /* end del for(1) */
/*
*****************************************************************
*                    Fin de la decodificación                   *
*                 y sección de cambio de bloque                 *
*****************************************************************
*/
   fclose(in);
   fclose(out);
   fclose(in1);
   }                        /* Fin del programa */

/*
*****************************************************************
*      Subrutina para la escritura de los ceros y la magnitud   *
*****************************************************************
*/
int sub()
{
  int i;
  float cero;

  cero=0.0;

  for(i=0;i<ceros;i++)
   {
   if(count>=64) goto ya;
   fwrite(&cero,4,1,out);     /* se escriben los ceros de la    */
   count=count+1;             /* categoria correspondiente      */
   }

  if(mag!=0)                  /* no se escribe si es fin de bloque */
   {
   if(count>=64) goto ya;
   fwrite(&mag,4,1,out);      /* se escribe la magnitud signada */
   count=count+1;
   }
ya:strcpy(d2,"");             /* se reinicializa d2   */
   flag=0;                    /* se reinicializa flag */
}
/*
*****************************************************************
* Fin de la subrutina para la escritura de los ceros y la magnitud *
*****************************************************************
*/

/*
*****************************************************************
*       Subrutina para la lectura del signo de números "1"      *
*****************************************************************
```

```
*/
int signo1()

 t signo;

  trcpy(d2,"");              /* se reinicializa d2 */

 ch=getc(in);  byte=byte+1;/* se lee codigo de signo */
 p=&ch;
  1[0]=*p;

 if(!(strcmp(d1,"0")))      /* se asigna signo    */
  { signo=-1; }
  lse
  { signo=1;  }

  um2=signo;
 return num2;



/*
*  ************************************************************************
*  Fin de la subrutina para la lectura del signo de números "1"  *
************************************************************************
*/


/*
************************************************************************
*        Subrutina para el cálculo de la magnitud signada          *
*  ************************************************************************
*/
i   magl(int num)

 int i;

/  ****** subsección de lectura y asignación del signo **********/

  strcpy(d2,"");             /* se reinicializa d2 */

  f(flag2!=1)                /* no se lee signo cuando es EOB */
   {
  ch=getc(in); byte=byte+1; /* se lee codigo de signo */
  p=&ch;
  d1[0]=*p;

  if(!(strcmp(d1,"0")))      /* se asigna signo    */
   { signo=-1; }
  else
   { signo=1;  }

   }                         /*  fin if(EOB)  */

/  **** finaliza subsección de lectura y asignación del signo *****/

/*
*  ************************************************************************
*        Subsección de lectura del código de la magnitud           *
************************************************************************
*/

 strcpy(d2,"");             /* se reinicializa d2 */
 strcpy(d1,"0");            /* se inicializa d1   */

 for(i=0;i<num;i++)         /*     for(11)        */
  {
  ch=getc(in); byte=byte+1; /* se leen los datos para la magnitud */
  p=&ch;
  d1[0]=*p;

  if(i==0)                  /* se asigna el codigo de magnitud --> d2
   { strcpy(d2,d1); }
  else
   { strcat(d2,d1); }
   }                        /* end del for(11) */

 if(flag2==1) return num1;  /* no se recupera magnitud en el caso */
                            /*      del fin de bloque EOB         */
/*
************************************************************************
*    Finaliza subsección de lectura del codigo de la magnitud     *
************************************************************************
*/

/*
************************************************************************
*     Subsección de recuperación codigo --> magnitud signada      *
************************************************************************
*/

  if(!(strcmp(d2,"0")))      /* 2 */
   {
   num1=2*signo;
   return num1;
   }

  if(!(strcmp(d2,"1")))      /* 3 */
   {
   num1=3*signo;
   return num1;
   }

  if(!(strcmp(d2,"00")))     /* 4 */
   {
   num1=4*signo;
   return num1;
   }

  if(!(strcmp(d2,"01")))     /* 5 */
   {
   num1=5*signo;
   return num1;
   }

  if(!(strcmp(d2,"10")))     /* 6 */
   {
   num1=6*signo;
```

```
 return num1;
 }

if(!(strcmp(d2,"11")))      /* 7 */
 {
 num1=7*signo;
 return num1;
 }

if(!(strcmp(d2,"000")))     /* 8 */
 {
 num1=8*signo;
 return num1;
 }

if(!(strcmp(d2,"001")))     /* 9 */
 {
 num1=9*signo;
 return num1;
 }

if(!(strcmp(d2,"010")))     /* 10 *
 {
 num1=10*signo;
 return num1;
 }

if(!(strcmp(d2,"011")))     /* 11 *
 {
 num1=11*signo;
 return num1;
 }

if(!(strcmp(d2,"100")))     /* 12 *
 {
 num1=12*signo;
 return num1;
 }

if(!(strcmp(d2,"101")))     /* 13
 {
 num1=13*signo;
 return num1;
 }

if(!(strcmp(d2,"110")))     /* 14
 {
 num1=14*signo;
 return num1;
 }

if(!(strcmp(d2,"111")))     /* 15
 {
 num1=15*signo;
 return num1;
 }

if(!(strcmp(d2,"0000")))    /* 16
 {
 num1=16*signo;
 return num1;
 }

if(!(strcmp(d2,"0001")))    /* 1
 {
 num1=17*signo;
 return num1;
 }

if(!(strcmp(d2,"0010")))    /* 1
 {
 num1=18*signo;
 return num1;
 }

if(!(strcmp(d2,"0011")))    /* 1
 {
 num1=19*signo;
 return num1;
 }

if(!(strcmp(d2,"0100")))    /* 2
 {
 num1=20*signo;
 return num1;
 }

if(!(strcmp(d2,"0101")))    /* 2
 {
 num1=21*signo;
 return num1;
 }

if(!(strcmp(d2,"0110")))    /* 2
 {
 num1=22*signo;
 return num1;
 }

if(!(strcmp(d2,"0111")))    /* 2
 {
 num1=23*signo;
 return num1;
 }

if(!(strcmp(d2,"1000")))    /* 2
 {
 num1=24*signo;
 return num1;
 }

if(!(strcmp(d2,"1001")))    /* 2
 {
 num1=25*signo;
 return num1;
 }

if(!(strcmp(d2,"1010")))    /* 2
 {
 num1=26*signo;
 return num1;
 }

if(!(strcmp(d2,"1011")))    /* 2
 {
 num1=27*signo;
 return num1;
 }

if(!(strcmp(d2,"1100")))    /* 2
 {
 num1=28*signo;
 return num1;
 }

if(!(strcmp(d2,"1101")))    /* 2
 {
 num1=29*signo;
 return num1;
 }

if(!(strcmp(d2,"1110")))    /* 3
 {
 num1=30*signo;
 return num1;
 }

if(!(strcmp(d2,"1111")))    /* 3
 {
 num1=31*signo;
 return num1;
 }

if(!(strcmp(d2,"00000")))   /*
 {
 num1=32*signo;
 return num1;
 }

if(!(strcmp(d2,"00001")))   /*
 {
 num1=33*signo;
 return num1;
 }

if(!(strcmp(d2,"00010")))   /*
 {
 num1=34*signo;
 return num1;
 }

if(!(strcmp(d2,"00011")))   /*
 {
 num1=35*signo;
 return num1;
 }

if(!(strcmp(d2,"00100")))   /*
 {
 num1=36*signo;
 return num1;
 }

if(!(strcmp(d2,"00101")))
 {
 num1=37*signo;
 return num1;
 }

if(!(strcmp(d2,"00110")))
 {
 num1=38*signo;
 return num1;
 }

if(!(strcmp(d2,"00111")))
 {
 num1=39*signo;
 return num1;
 }

if(!(strcmp(d2,"01000")))
 {
 num1=40*signo;
 return num1;
 }

if(!(strcmp(d2,"01001")))
 {
 num1=41*signo;
 return num1;
 }

if(!(strcmp(d2,"01010")))
 {
 num1=42*signo;
 return num1;
 }

if(!(strcmp(d2,"01011")))
 {
 num1=43*signo;
 return num1;
 }

if(!(strcmp(d2,"01100")))
 {
 num1=44*signo;
 return num1;
 }

if(!(strcmp(d2,"01101")))
 {
 num1=45*signo;
 return num1;
 }

if(!(strcmp(d2,"01110")))
```

```
{
num1=46*signo;
return num1;
}

if(!(strcmp(d2,"01111")))    /* 4
{
num1=47*signo;
return num1;
}

if(!(strcmp(d2,"10000")))    /* 4
{
num1=48*signo;
return num1;
}

if(!(strcmp(d2,"10001")))    /* 4
{
num1=49*signo;
return num1;
}

if(!(strcmp(d2,"10010")))    /* 5
{
num1=50*signo;
return num1;
}

if(!(strcmp(d2,"10011")))    /* 5
{
num1=51*signo;
return num1;
}

if(!(strcmp(d2,"10100")))    /* 5
{
num1=52*signo;
return num1;
}

if(!(strcmp(d2,"10101")))    /* 5
{
num1=53*signo;
return num1;
}

if(!(strcmp(d2,"10110")))    /* 5
{
num1=54*signo;
return num1;
}

if(!(strcmp(d2,"10111")))    /* 5
{
num1=55*signo;
return num1;
}

if(!(strcmp(d2,"11000")))    /*
```

```
{
num1=56*signo;
return num1;
}

if(!(strcmp(d2,"11001")))    /*
{
num1=57*signo;
return num1;
}

if(!(strcmp(d2,"11010")))    /*
{
num1=58*signo;
return num1;
}

if(!(strcmp(d2,"11011")))    /*
{
num1=59*signo;
return num1;
}

if(!(strcmp(d2,"11100")))    /*
{
num1=60*signo;
return num1;
}

if(!(strcmp(d2,"11101")))    /*
{
num1=61*signo;
return num1;
}

if(!(strcmp(d2,"11110")))    /*
{
num1=62*signo;
return num1;
}

if(!(strcmp(d2,"11111")))    /*
{
num1=63*signo;
return num1;
}

if(!(strcmp(d2,"000000")))    /*
{
num1=64*signo;
return num1;
}

if(!(strcmp(d2,"000001")))    /*
{
num1=65*signo;
return num1;
}

if(!(strcmp(d2,"000010")))    /*
```

```
{
num1=66*signo;
return num1;
}

if(!(strcmp(d2,"000011")))    /*
{
num1=67*signo;
return num1;
}

if(!(strcmp(d2,"000100")))    /*
{
num1=68*signo;
return num1;
}

if(!(strcmp(d2,"000101")))    /*
{
num1=69*signo;
return num1;
}

if(!(strcmp(d2,"000110")))    /*
{
num1=70*signo;
return num1;
}

if(!(strcmp(d2,"000111")))    /*
{
num1=71*signo;
return num1;
}

if(!(strcmp(d2,"001000")))    /*
{
num1=72*signo;
return num1;
}

if(!(strcmp(d2,"001001")))    /*
{
num1=73*signo;
return num1;
}

if(!(strcmp(d2,"001010")))    /*
{
num1=74*signo;
return num1;
}

if(!(strcmp(d2,"001011")))    /*
{
num1=75*signo;
return num1;
}

if(!(strcmp(d2,"001100")))    /*
{
num1=76*signo;
```

```
return num1;
}

if(!(strcmp(d2,"001101")))
{
num1=77*signo;
return num1;
}

if(!(strcmp(d2,"001110")))
{
num1=78*signo;
return num1;
}

if(!(strcmp(d2,"001111")))
{
num1=79*signo;
return num1;
}

if(!(strcmp(d2,"010000")))
{
num1=80*signo;
return num1;
}

if(!(strcmp(d2,"010001")))
{
num1=81*signo;
return num1;
}

if(!(strcmp(d2,"010010")))
{
num1=82*signo;
return num1;
}

if(!(strcmp(d2,"010011")))
{
num1=83*signo;
return num1;
}

if(!(strcmp(d2,"010100")))
{
num1=84*signo;
return num1;
}

if(!(strcmp(d2,"010101")))
{
num1=85*signo;
return num1;
}

if(!(strcmp(d2,"010110")))
```

```
{
num1=86*signo;
return num1;
}

if(!(strcmp(d2,"010111")))    /*
{
num1=87*signo;
return num1;
}

if(!(strcmp(d2,"011000")))    /*
{
num1=88*signo;
return num1;
}

if(!(strcmp(d2,"011001")))    /*
{
num1=89*signo;
return num1;
}

if(!(strcmp(d2,"011010")))    /*
{
num1=90*signo;
return num1;
}

if(!(strcmp(d2,"011011")))    /*
{
num1=91*signo;
return num1;
}

if(!(strcmp(d2,"011100")))    /*
{
num1=92*signo;
return num1;
}

if(!(strcmp(d2,"011101")))    /*
{
num1=93*signo;
return num1;
}

if(!(strcmp(d2,"011110")))    /*
{
num1=94*signo;
return num1;
}

if(!(strcmp(d2,"011111")))    /*
{
num1=95*signo;
return num1;
}

if(!(strcmp(d2,"100000")))    /*
```

```
{
num1=96*signo;
return num1;
}

if(!(strcmp(d2,"100001")))    /*
{
num1=97*signo;
return num1;
}

if(!(strcmp(d2,"100010")))    /*
{
num1=98*signo;
return num1;
}

if(!(strcmp(d2,"100011")))    /*
{
num1=99*signo;
return num1;
}

if(!(strcmp(d2,"100100")))    /*
{
num1=100*signo;
return num1;
}

if(!(strcmp(d2,"100101")))    /*
{
num1=101*signo;
return num1;
}

if(!(strcmp(d2,"100110")))    /*
{
num1=102*signo;
return num1;
}

if(!(strcmp(d2,"100111")))    /*
{
num1=103*signo;
return num1;
}

if(!(strcmp(d2,"101000")))    /*
{
num1=104*signo;
return num1;
}

if(!(strcmp(d2,"101001")))    /*
{
num1=105*signo;
return num1;
}

if(!(strcmp(d2,"101010")))
```

```
/*
{
num1=106*signo;
return num1;
}

if(!(strcmp(d2,"101011")))    /*
{
num1=107*signo;
return num1;
}

if(!(strcmp(d2,"101100")))    /*
{
num1=108*signo;
return num1;
}

if(!(strcmp(d2,"101101")))    /*
{
num1=109*signo;
return num1;
}

if(!(strcmp(d2,"101110")))    /*
{
num1=110*signo;
return num1;
}

if(!(strcmp(d2,"101111")))    /*
{
num1=111*signo;
return num1;
}

if(!(strcmp(d2,"110000")))    /*
{
num1=112*signo;
return num1;
}

if(!(strcmp(d2,"110001")))    /*
{
num1=113*signo;
return num1;
}

if(!(strcmp(d2,"110010")))    /*
{
num1=114*signo;
return num1;
}

if(!(strcmp(d2,"110011")))    /*
{
num1=115*signo;
return num1;
}

if(!(strcmp(d2,"110100")))    /*
```

```
{
num1=116*signo;
return num1;
}

if(!(strcmp(d2,"110101")))
{
num1=117*signo;
return num1;
}

if(!(strcmp(d2,"110110")))
{
num1=118*signo;
return num1;
}

if(!(strcmp(d2,"110111")))
{
num1=119*signo;
return num1;
}

if(!(strcmp(d2,"111000")))
{
num1=120*signo;
return num1;
}

if(!(strcmp(d2,"111001")))
{
num1=121*signo;
return num1;
}

if(!(strcmp(d2,"111010")))
{
num1=122*signo;
return num1;
}

if(!(strcmp(d2,"111011")))
{
num1=123*signo;
return num1;
}

if(!(strcmp(d2,"111100")))
{
num1=124*signo;
return num1;
}

if(!(strcmp(d2,"111101")))
{
num1=125*signo;
return num1;
}

if(!(strcmp(d2,"111110")))
```

```
    {
    num1=126*signo;
    return num1;
    }

  if(!(strcmp(d2,"111111")))    /* 127 */
    {
    num1=127*signo;
    return num1;
    }

/*
*************************************************************************
* Sección de búsqueda de la magnitud para todas las otras combinaciones
*************************************************************************
*/

    if(!(strcmp(d2,"0000000")))   /* 1 */
    {
    num1=1;
    return num1;
    }

   if(!(strcmp(d2,"0000001")))    /* 2 */
    {
    num1=2;
    return num1;
    }

   if(!(strcmp(d2,"0000010")))    /* 3 */
    {
    num1=3;
    return num1;
    }

   if(!(strcmp(d2,"0000011")))    /* 4 */
    {
    num1=4;
    return num1;
    }

   if(!(strcmp(d2,"0000100")))    /* 5 */
    {
    num1=5;
    return num1;
    }

   if(!(strcmp(d2,"0000101")))    /* 6 */
    {
    num1=6;
    return num1;
    }

   if(!(strcmp(d2,"0000110")))    /* 7 */
    {
    num1=7;
    return num1;
    }

   if(!(strcmp(d2,"0000111")))    /* 8 */
    {
    num1=8;
    return num1;
    }

   if(!(strcmp(d2,"0001000")))    /* 9 */
    {
    num1=9;
    return num1;
    }

   if(!(strcmp(d2,"0001001")))    /* 10 */
    {
    num1=10;
    return num1;
    }

   if(!(strcmp(d2,"0001010")))    /* 11 */
    {
    num1=11;
    return num1;
    }

   if(!(strcmp(d2,"0001011")))    /* 12 */
    {
    num1=12;
    return num1;
    }

   if(!(strcmp(d2,"0001100")))    /* 13 */
    {
    num1=13;
    return num1;
    }

   if(!(strcmp(d2,"0001101")))    /* 14 */
    {
    num1=14;
    return num1;
    }

   if(!(strcmp(d2,"0001110")))    /* 15 */
    {
    num1=15;
    return num1;
    }

   if(!(strcmp(d2,"0001111")))    /* 16 */
    {
    num1=16;
    return num1;
    }

   if(!(strcmp(d2,"0010000")))    /* 17 */
    {
    num1=17;
    return num1;
    }

   if(!(strcmp(d2,"0010001")))    /* 18 */
```

```
{
num1=18;
return num1;
}

if(!(strcmp(d2,"0010010")))   /*
{
num1=19;
return num1;
}

if(!(strcmp(d2,"0010011")))   /*
{
num1=20;
return num1;
}

if(!(strcmp(d2,"0010100")))   /*
{
num1=21;
return num1;
}

if(!(strcmp(d2,"0010101")))   /*
{
num1=22;
return num1;
}

if(!(strcmp(d2,"0010110")))   /*
{
num1=23;
return num1;
}

if(!(strcmp(d2,"0010111")))   /*
{
num1=24;
return num1;
}

if(!(strcmp(d2,"0011000")))   /*
{
num1=25;
return num1;
}

if(!(strcmp(d2,"0011001")))   /*
{
num1=26;
return num1;
}

if(!(strcmp(d2,"0011010")))   /*
{
num1=27;
return num1;
}

if(!(strcmp(d2,"0011011")))   /*
```

```
{
num1=28;
return num1;
}

if(!(strcmp(d2,"0011100")))   /
{
num1=29;
return num1;
}

if(!(strcmp(d2,"0011101")))   /
{
num1=30;
return num1;
}

if(!(strcmp(d2,"0011110")))   /
{
num1=31;
return num1;
}

if(!(strcmp(d2,"0011111")))   /
{
num1=32;
return num1;
}

if(!(strcmp(d2,"0100000")))   /
{
num1=33;
return num1;
}

if(!(strcmp(d2,"0100001")))   /
{
num1=34;
return num1;
}

if(!(strcmp(d2,"0100010")))   /
{
num1=35;
return num1;
}

if(!(strcmp(d2,"0100011")))   /
{
num1=36;
return num1;
}

if(!(strcmp(d2,"0100100")))   /
{
num1=37;
return num1;
}

if(!(strcmp(d2,"0100101")))   /
{
num1=38;
```

```
return num1;
}

if(!(strcmp(d2,"0100110")))   /
{
num1=39;
return num1;
}

if(!(strcmp(d2,"0100111")))   /
{
num1=40;
return num1;
}

if(!(strcmp(d2,"0101000")))   /
{
num1=41;
return num1;
}

if(!(strcmp(d2,"0101001")))   /
{
num1=42;
return num1;
}

if(!(strcmp(d2,"0101010")))   /
{
num1=43;
return num1;
}

if(!(strcmp(d2,"0101011")))   /
{
num1=44;
return num1;
}

if(!(strcmp(d2,"0101100")))   /
{
num1=45;
return num1;
}

if(!(strcmp(d2,"0101101")))   /
{
num1=46;
return num1;
}

if(!(strcmp(d2,"0101110")))   /
{
num1=47;
return num1;
}

if(!(strcmp(d2,"0101111")))   /
{
num1=48;
```

```
return num1;
}

if(!(strcmp(d2,"0110000")))
{
num1=49;
return num1;
}

if(!(strcmp(d2,"0110001")))
{
num1=50;
return num1;
}

if(!(strcmp(d2,"0110010")))
{
num1=51;
return num1;
}

if(!(strcmp(d2,"0110011")))
{
num1=52;
return num1;
}

if(!(strcmp(d2,"0110100")))
{
num1=53;
return num1;
}

if(!(strcmp(d2,"0110101")))
{
num1=54;
return num1;
}

if(!(strcmp(d2,"0110110")))
{
num1=55;
return num1;
}

if(!(strcmp(d2,"0110111")))
{
num1=56;
return num1;
}

if(!(strcmp(d2,"0111000")))
{
num1=57;
return num1;
}

if(!(strcmp(d2,"0111001")))
{
num1=58;
```

```
    return num1;
    }

   if(!(strcmp(d2,"0111010")))    /* 59 */
    {
    num1=59;
    return num1;
    }

   if(!(strcmp(d2,"0111011")))    /* 60 */
    {
    num1=60;
    return num1;
    }

   if(!(strcmp(d2,"0111100")))    /* 61 */
    {
    num1=61;
    return num1;
    }

   if(!(strcmp(d2,"0111101")))    /* 62 */
    {
    num1=62;
    return num1;
    }

   if(!(strcmp(d2,"0111110")))    /* 63 */
    {
    num1=63;
    return num1;
    }

   if(!(strcmp(d2,"0111111")))    /* 64 */
    {
    num1=64;
    return num1;
    }

  /*
  ********************************************************************************
  *    Fin búsqueda de la magnitud para todas las otras combinaciones     *
  ********************************************************************************
  */


  /*
  ********************************************************************************
  * Fin de la subsección de la recuperación codigo --> magnitud signada *
  ********************************************************************************
  */

   }
  /*
  ********************************************************************************
  *    Fin de la subrutina para el cálculo de la magnitud signada    *
  ********************************************************************************
  */
```