UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

SISTEMA DE CODIFICACION DE IMAGENES DIGITALES UTILIZANDO LA MODULACION POR PULSOS CODIFICADOS DIFERENCIAL (DPCM)

T E S I S

Y PARA OBTENER EL TITULO DE
INGENIERO MECANICO ELECTRICISTA

P R E S E N T A N
GERARDO SANTOS PAZ
SALVADOR RUIZ CORREA

DIRECTOR DE TESIS
DR. FRANCISCO J. GARCIA UGALDE

MEXICO, D. F. JULIO DE 1990

INDICE.

INTRODUCCION

INTRODUCCION.

El presente trabajo tiene como objetivo estudiar la factibilidad de implantación de un algoritmo de codificación de fuente, utilizando un esquema predictivo, en un sistema digital basado en el microprocesador TMS320C25 . El marco teórico del estudio (Fig. 1) incluye los siguientes temas: La representación numérica de la información visual (Cap. I), el sistema de comunicaciones de televisión monocromática (Cap. II), métodos de compresión (Cap. III), el sistema visual humano (Cap. IV) y los fundamentos de la codificación de fuente DPCM (Modulación por pulsos codificados diferencial) (Cap. V).

En el capitulo VI, se presenta un estudio comparativo de diversos algoritmos de codificación DPCM. Los criterios de análisis son fundamentalmente subjetivos y las simulaciones de los algoritmos se llevaron a cabo en un sistema digital de cómputo.

El conjunto de programas que se utilizaron en las simulaciones de los algoritmos, se integraron en un paquete de simulación MICD (DPCM) (Cap. VII), con el objeto de servir como material didáctico de apoyo para el Laboratorio de Procesamiento de Imágenes Digitales, de la División de Estudios de Posgrado de la Facultad de Ingeniería, UNAM.

El capitulo VIII, está consagrado al estudio de la factibilidad de implantación del algoritmo seleccionado en el capitulo VI en una arquitectura (DSP), construida alrededor del microprocesador TMS320C25. Las conclusiones se presentan en el capitulo IX .

En la parte final del trabajo, se incluyen cuatro apéndices que complementan los tópicos tratados en los capitulos VI, VII y VIII. En particular, en el apéndice C se incluyen todos los documentos de la codificación de los módulos de los algoritmos de simulación, en lenguaje de alto nivel.

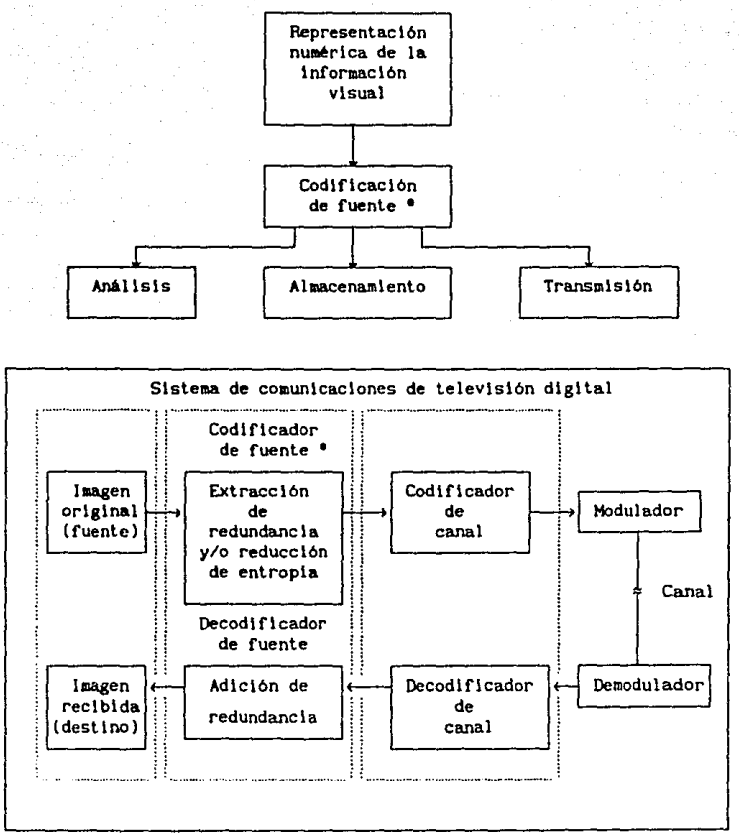


Fig. 1 Diagrama de bloques del marco teórico.

CAPITULO 1

REPRESENTACION NUMERICA DE LA INFORMACION VISUAL

CAPITULO I. REPRESENTACION NUMERICA DE LA INFORMACION VISUAL.

I.1 La información visual.

La capacidad de ver es una de las características de muchos seres vivientes. Les permite percibir imágenes y asimilar en muy poco tiempo una gran cantidad de conocimiento del mundo que les rodea. El hombre ha inventado dispositivos que le permiten detectar ondas de radiación electromagnética que están fuera del rango de visión normal, a niveles de energía inferiores que el ojo no es capaz de detectar por sí mismo. Es mediante estos dispositivos que hoy es posible ver infinidad de objetos y lugares desconocidos.

El concepto de *información visual* está intimamente relacionado con estos comentarios, sin embargo, la formalización del mismo no resulta sencilla. Descartes decía que una imagen dice más que mil palabras, desafortunadamente, al igual que Shannon, no dijo como lo hace. Además, cualquier intento de formalización debe contemplar la capacidad tecnológica actual para el manejo de la información. Por estas razones, se deben establecer una serie de restricciones que permitan esclarecer y manejar el concepto adecuadamente.

La primera restricción es que la información de las imágenes percibidas por los observadores, es obtenida mirando a través de una ventana rectangular de dimensiones finitas, es decir, *el tamaño de la imagen es finito*. Las cámaras, los telescopios y los microscopios, por ejemplo, tienen campos de visión de tamaño definido y la restricción de finitud es usualmente necesaria para crear sistemas que puedan procesar solamente cantidades ponderables de información.

Se asume también que el observador es incapaz de percibir la profundidad por sí mismo. Esto es, en la escena que está observando no puede decir que tan lejos están los objetos los unos de los otros, usando la visión binocular o cambiando el enfoque de sus ojos. Sin embargo, es capaz de inferir la profundidad de otra forma. Por ejemplo, si se observa a través de la ventana que un objeto se mueve, éste pasará frente a otros objetos más lejanos en el fondo de la escena. Si el observador se mueve en un automóvil, entonces su

ángulo de visión cambiará con el tiempo, mientras observa a través de la ventana. Y finalmente, si una lente óptica es colocada entre la ventana y la escena, entonces la profundidad puede ser inferida modificando el foco de la lente. De este modo, el mundo fuera de la ventana puede cambiar y la casa o el automóvil del observador se pueden mover, pero él mismo, debe aceptar que cualquier información visual llega a través de la ventana.

Este modelo describe una gran cantidad de sistemas que reciben información visual incluyendo, la televisión, la fotografía, los rayos x, etc. En este contexto, la información visual está completamente determinada por las longitudes de onda y las amplitudes de la luz que pasan a través de cada punto en la ventana y alcanzan el ojo del observador. De este modo, si el mundo exterior desapareciera y un proyector reprodujera la distribución de luz en la ventana, el observador no sería capaz de notar la diferencia.

Es así que el problema de la representación numérica de la información visual, se ve reducido a especificar la distribución de la energía luminosa y las longitudes de onda sobre el área finita de la ventana. Si se considera que la imagen que percibe el observador es *monocromática*, entonces, la imagen estará determinada por la energía luminosa percibida (la suma ponderada de energía de las longitudes de onda distinguibles), que pasa a través de cada punto de la ventana hasta el ojo del observador. Si se incorpora a la ventana un sistema de ejes cartesianos, es posible representar la energía luminosa o intensidad de luz en el punto (x,y) mediante una función bidimensional de intensidad de luz, $f(x,y)$. Aunque es posible considerar el parámetro de tiempo para definir f , en lo que sigue, sólo se usará como una función de las coordenadas espaciales x e y . En la siguiente sección se formalizan los conceptos hasta aquí presentados.

1.2 El modelo matemático de una imagen.

El término *imagen monocromática* o simplemente *imagen*, se refiere a una función bidimensional de intensidad de luz, $f(x,y)$, donde x e y denotan las coordenadas espaciales y el valor de f en cualquier punto (x,y), es proporcional a la brillantez (o nivel de gris) de la imagen en ese punto. La

figura I.1 muestra la convención de ejes que comúnmente se utiliza. Debido a que la luz es una forma de energía, $f(x,y)$ debe ser mayor que cero y finita, esto es,

$$0 < f(x,y) < \infty \qquad (I.1)$$

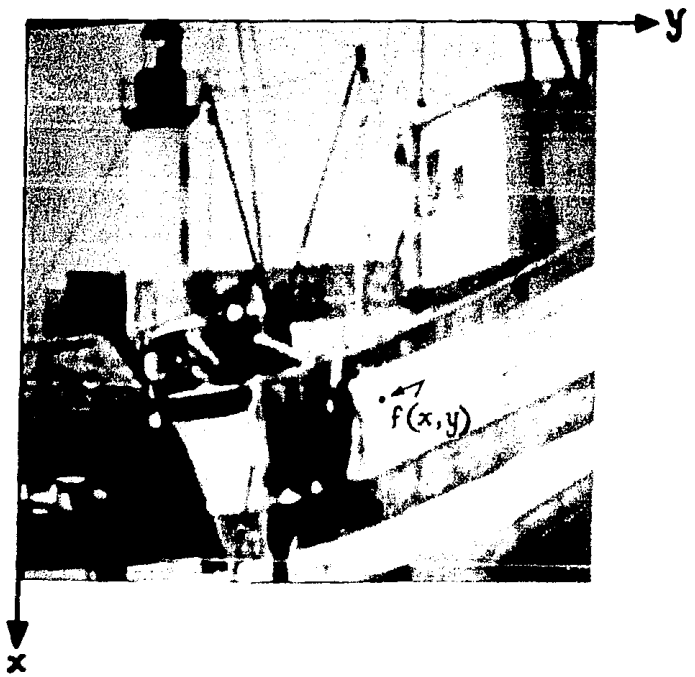


Fig. I.1 Sistema de referencia utilizado en el presente trabajo.

Las imágenes que percibimos normalmente consisten en la luz que los objetos reflejan, por ello la naturaleza básica de $f(x,y)$ puede ser caracterizada por dos componentes. Una componente es la cantidad de luz de la fuente que incide en la escena observada, mientras que la otra es la cantidad de luz reflejada por los objetos en la escena. Estas componentes se denominan respectivamente *componente de iluminación* y *componente de reflectancia* y se

denotan por $i(x,y)$ y $r(x,y)$. Las funciones $i(x,y)$ y $r(x,y)$ se combinan como un producto para formar $f(x,y)$, es decir,

$$f(x,y) = i(x,y) \cdot r(x,y) \tag{I.2}$$

donde

$$0 < i(x,y) < \infty \tag{I.3}$$

y

$$0 < r(x,y) < 1 \tag{I.4}$$

La ecuación I.4 muestra el hecho de que la reflectancia está restringida entre el 0 (absorción total) y el 1 (reflexión total). La naturaleza de $i(x,y)$ está determinada por la fuente de luz, mientras que $r(x,y)$ está determinada por las características de los objetos en la escena. Los valores dados en las ecuaciones I.3 y I.4 son límites teóricos.

La intensidad de una imagen monocromática, f , en las coordenadas (x,y) se denomina nivel de gris (l) de la imagen en ese punto. De las ecuaciones I.1, I.2, I.3 y I.4, es evidente que :

$$L_{min} \leq l \leq L_{max} \tag{I.5}$$

En teoría, el único requerimiento de L_{min} es que sea positiva y L_{max} , que sea finita. En la práctica $L_{min} = i_{min} \cdot r_{min}$ y $L_{max} = i_{max} \cdot r_{max}$. En aplicaciones de procesamiento de imágenes en interiores, $L_{min} \approx 0.005$ y $L_{max} \approx 100$.

El intervalo $[L_{min}, L_{max}]$ se denomina *escala de gris*. Es practica común recorrer este intervalo numéricamente hacia el rango $[0,L]$, donde $l = 0$ se considera negro y $l = L$ se considera blanco en la escala. Todos los valores intermedios son tonos de gris que varian continuamente del negro al blanco.

I.3 El muestreo y la cuantificación de una imagen.

La función bidimensional $f(x,y)$ de una imagen debe ser digitalizada espacialmente y en amplitud, para poder ser procesada digitalmente. A la digitalización de las coordenadas espaciales (x,y) se le denomina *muestreo de la imagen*, mientras que la digitalización en amplitud se denomina *cuantificación del nivel de gris.*

Supongamos que una imagen continua, $f(x,y)$, es aproximada por un arreglo cuadrado de $N \times N$ muestras igualmente espaciadas como se muestra a continuación, donde cada elemento del arreglo es una cantidad discreta.

El lado derecho de esta ecuación representa lo que comúnmente se denomina imagen digital, en donde los índices de renglón y columna identifican un punto en la imagen y el correspondiente elemento de la matriz identifica el nivel de gris en ese punto. Los elementos de tal arreglo digital se denominan *elementos de la imagen, pixeles, peals* o *puntos.*

$$\hat{f}(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(N-1,0) & f(N-1,1) & \cdots & f(N-1,N-1) \end{bmatrix} \qquad (I.6)$$

El proceso de digitalización, requiere la determinación del valor N, así como el número de niveles de gris para la representación de los puntos. Es práctica común en el procesamiento de imágenes digitales, asignar a estas cantidades números enteros potencias de 2 , esto es :

$$N = 2^n \qquad (I.7)$$

y

$$G = 2^m \qquad (I.8)$$

donde G denota el número de niveles de gris. Asumiendo que los valores discretos están igualmente espaciados entre 0 y L , en la escala de gris y

utilizando las ecuaciones 1.7 y 1.8 , el número de bits, b , requerido para almacenar una imagen digitalizada está dado por:

$$b = N \times N \times m \qquad (1.9)$$

Por ejemplo, una imagen de 256 x 256 elementos, digitalizada con 256 niveles de gris, requiere 65,536 bytes para ser almacenada. La tabla (1.1) indica los valores de b para algunos rangos típicos de N y m . La tabla (1.2) muestra el número de bytes, de 8 bits, necesarios para el

Tabla 1.1 Número de bits de almacenamiento para varios valores de N y m .

N \ m	1	2	3	4	5	6	7	8
32	1,024	2,048	3,072	4,096	5,120	6,144	7,168	8,192
64	4,096	8,192	12,288	16,384	20,480	24,576	28,672	32,768
128	16,384	32,768	49,152	65,536	81,920	98,304	114,688	131,072
256	65,536	131,072	196,608	262,144	327,680	393,216	458,752	524,288
512	262,144	524,288	786,432	1,048,576	1,310,720	1,572,864	1,835,008	2,097,152

Tabla 1.2 Número de bytes (de 8 bits) de almacenamiento para varios valores de N y m .

N \ m	1	2	3	4	5	6	7	8
32	128	256	512	512	1,024	1,024	1,024	1,024
64	512	1,024	2,048	2,048	4,096	4,096	4,096	4,096
128	2,048	4,096	8,192	8,192	16,384	16,384	16,384	16,384
256	8,192	16,384	32,768	32,768	65,536	65,536	65,536	65,536
512	32,768	65,536	131,072	131,072	262,144	262,144	262,144	262,144

almacenamiento. Generalmente, no es práctico desde el punto de vista de la programación, llenar un byte completamente, si esto implica el traslape de los puntos entre un byte y otro. De este modo, la tabla (I.2) muestra el número mínimo de bytes que se requieren para cada valor de N y m, cuando el traslape no se permite.

Debido a que la ecuación I.6 es una aproximación a una imagen continua, una pregunta razonable es, cuántas muestras y cuántos niveles de gris se requieren para una buena aproximación ? La *resolución* (es decir, el grado de discriminación de los detalles) de una imagen digital, depende en gran parte de N y m. Entre más grandes sean estos parámetros, más fiel es la aproximación a la imagen. Sin embargo, la ecuación I.9 muestra claramente que la capacidad de almacenamiento y consecuentemente los requerimientos de procesamiento, se incrementan rápidamente como una función de N y m.

En vista de los comentarios anteriores, es necesario considerar el efecto que tienen las variaciones de N y m en la calidad de una imagen. Para ello, se define el *proceso de muestreo* como la adquisición de los valores de intensidad definidos dentro de *celdas*, en las que se divide la imagen. Cada celda corresponde a un punto (considerando que dicho punto se ubica en la parte central de la celda). Y su forma, constituye el *motivo* de base del enrejado de muestreo para dividir el plano de la imagen. Los motivos más comunes son el rectangular, el triangular y el hexagonal. A su vez, de estos tres motivos, el rectangular es el más usado.

Los requerimientos de calidad varían de acuerdo a la aplicación. La figura I.2 muestra el efecto de la reducción del tamaño del enrejado de muestreo en la imagen. De la figura I.2(a) a la I.2(d) se muestra la misma imagen, pero con N = 256, 128, 64 y 32 muestras por renglón, respectivamente. En todos los casos el número máximo de niveles de gris permitidos se mantuvo en 32 . Debido a que el área de despliegue usada para cada imagen es la misma (es decir, 256 x 256 puntos), los puntos en las imágenes de menor resolución, fueron duplicados de forma tal que se llenara completamente dicha área. Esto produce un efecto de cuadriculado, que se aprecia claramente en las imágenes de poca resolución. Se nota, además, que la imagen I.2(a) es muy parecida a la figura I.2(b), pero la calidad de la imagen se deteriora rápidamente para los demás valores de N .

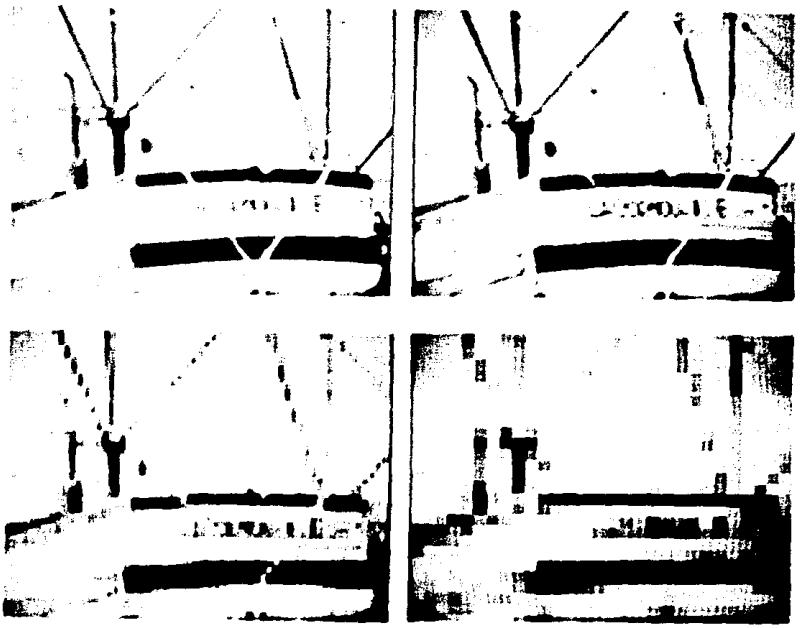


Fig. I.2 Efectos de la reducción del tamaño en el enrejado de muestreo de una imagen. Arriba a la izquierda (a) $N = 256$, a la derecha (b) $N = 128$. Abajo a la izquierda (c) $N = 64$ y a la derecha (d) $N = 32$.

La figura I.3 ilustra los efectos producidos, debido a la reducción del número de bits utilizados para representar los niveles de gris en una imagen. La figura I.2(a) es la imagen de un barco, que ha sido digitalizada usando 32 niveles de gris ($m = 5$ en la ecuación I.8). De la figura I.3(a) a la I.3(d) se muestran las imágenes obtenidas al reducir el número de bits de

$m = 4$ a $m = 1$ respectivamente, manteniendo el tamaño del enrejado constante. La imagen de 32 niveles de gris (2^5) mantiene una calidad aceptable. La imagen de 16 niveles (2^4), sin embargo, tiene falsos contornos en el fondo. Este efecto es considerablemente más notable en la imagen desplegada con 8 niveles (2^3), y se incrementa rápidamente en las imágenes subsiguientes.

El número de muestras y los niveles de gris requeridos para producir una reproducción fiel de la imagen original, depende de la imagen misma. Como base de comparación, los requerimientos para obtener una calidad aceptable,

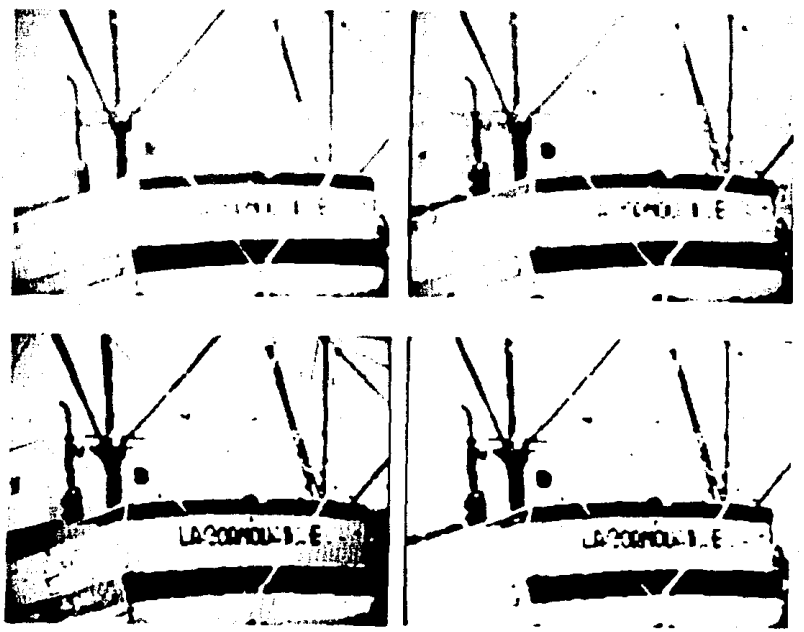


Fig. 1.3 Efectos de la variación de niveles de gris. Arriba a la izquierda (a) $m = 4$, a la derecha (b) $m = 3$. Abajo a la izquierda (c) $m = 2$ y a la derecha (d) $m = 1$.

semejante a la de las imágenes monocromáticas de televisión (dentro de un rango amplio de imágenes diferentes), implica arreglos del orden de 512 x 512 puntos con 128 niveles de gris. Como regla general, un sistema mínimo para el procesamiento de imágenes digitales, debe ser capaz de desplegar 256 x 256 puntos con al menos 64 niveles de gris.

Los resultados mostrados en las figuras I.2 y I.3 ilustran los efectos producidos en la calidad de la imagen variando independientemente, N y m. Sin embargo, estos resultados sólo responden parcialmente la pregunta inicial, debido a que nada se ha expuesto respecto a la relación entre estos parámetros.

1.3.1 Muestreo uniforme y cuantificación.

Para cuantificar experimentalmente los efectos en la calidad de una imagen, producidos al variar N y m simultáneamente, se realizan un conjunto de pruebas subjetivas. Si tomamos tres imágenes, la primera con relativamente pocos detalles, la segunda con una cantidad media de detalles y la tercera con una gran cantidad de información en los detalles, es posible generar gráficas de isopreferencia, al preguntar a distintos observadores sobre la calidad (subjetiva) de las imágenes.

Para analizar los resultados, se emplean las curvas de isopreferencia en un plano N *vs.* m, en donde los puntos representan imágenes de igual calidad subjetiva. Los resultados proporcionan conclusiones empíricas: (1) Como es de esperarse, la calidad de las imágenes tiende a mejorar conforme N y m son incrementados. Pero hay pocos casos en los cuales, para un N fijo, la calidad mejora al decrementar m. Esto es debido probablemente al hecho de que un decremento en m, generalmente incrementa el contraste aparente de una imagen. (2) Las curvas tienden a ser más verticales conforme los detalles en la imagen se incrementan. Esto sugiere que para imágenes con una gran cantidad de detalles se requieren solamente pocos niveles de gris. (3) Las curvas de isopreferencia se desvían marcadamente de las curvas de b constantes (bits requeridos para almacenar una imagen digitalizada), lo que indica que se requieren más bits para ser almacenadas.

1.3.2 Muestreo no uniforme y cuantificación.

Para un valor fijo de N, es posible en muchos casos, mejorar la apariencia de una imagen usando un esquema adaptable, donde el proceso de muestreo depende de las características de la imagen. En general, el muestreo fino es requerido en la vecindad de las transiciones de nivel de gris muy grandes, mientras que un muestreo burdo puede ser empleado en regiones relativamente uniformes. Considérese, por ejemplo, una sola imagen que consiste en un rostro sobrepuesto en un fondo uniforme. Claramente el fondo aporta poca información de los detalles y puede ser representado adecuadamente mediante un muestreo burdo. El rostro, por otro lado, contiene considerablemente más detalles. Si las muestras adicionales no usadas en el fondo son usadas en esta última región de la imagen, el resultado final tenderá a mejorar, particularmente si N es pequeño. Distribuyendo las muestras, la concentración más grande podría ser usada en las fronteras de transición de niveles de gris, como lo es la frontera entre el rostro y el fondo en el ejemplo anterior. Este hecho es importante, ya que implica la necesidad de identificar fronteras, aún sobre bases aproximadas. Esto constituye una desventaja en el muestreo no uniforme. Este método no es práctico para imágenes que contienen, relativamente, pequeñas regiones uniformes. Por ejemplo, un muestreo no uniforme es difícil de justificar para una imagen en donde aparece un denso grupo de personas.

Cuando el número de niveles de gris disponible es pequeño, usualmente es deseable usar niveles desigualmente espaciados en el proceso de cuantificación. Un método similar a la técnica de muestreo no uniforme, discutida anteriormente, puede ser usada para la distribución de los niveles de gris en una imagen. Sin embargo, ya que el ojo es relativamente ineficiente al estimar niveles de gris cerca de cambios bruscos, la mejor aproximación en este caso se obtiene usando pocos niveles de gris en la vecindad de las fronteras. Los niveles restantes pueden ser usados entonces, en regiones donde las variaciones de niveles de gris son pequeñas, con lo que se evitan o reducen los falsos contornos que aparecen frecuentemente en estas regiones, si éstos se cuantifican burdamente.

Este método, está sujeto a las mismas observaciones hechas antes, con respecto a la detección de fronteras y a la cantidad de detalles en la imagen. Una técnica alternativa, que es particularmente atractiva para la distribución de niveles de gris, consiste en calcular la frecuencia de ocurrencia de todos los niveles permitidos. Si algunos niveles de gris ocurren frecuentemente en un cierto rango, los niveles de cuantificación deberán ser espaciados finamente, mientras que los que ocurren rara vez requerirán una cuantificación burda. A este método se le conoce comúnmente con el nombre de *cuantificación gradual*.

I.4 Consideraciones respecto a la digitalización de una imagen monocromática.

Una vez que se ha decidido la densidad de muestreo para digitalizar una imagen, puede suceder que existan algunas variaciones de intensidad en $f(x,y)$, demasiado rápidas para ser representadas adecuadamente utilizando

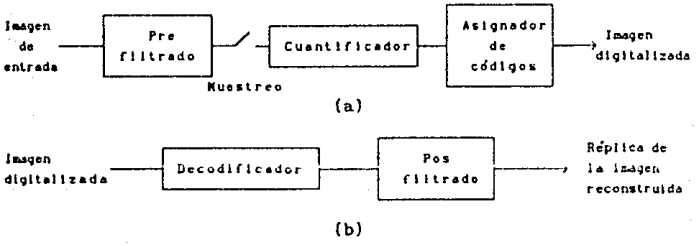


Fig. I.4 Las imágenes muestreadas y cuantificadas requieren generalmente un prefiltrado (a), antes del muestreo y un posfiltrado (b) de la imagen reconstruida. Un prefiltrado inadecuado resulta en un pretraslape mientras que un posfiltrado inadecuado provoca postraslape.

la tasa de muestreo seleccionada. A este fenómeno se le conoce como *pretraslape*.

Con el objeto de que este fenómeno no se presente, resulta más adecuado no muestrear la imagen en los puntos (x,y) . En lugar de ello, se promedia la intensidad de luz en una pequeña región alrededor de cada punto (x,y) . Con esta aproximación, las transiciones de intensidad no son muy pequeñas o demasiado rápidas y ello permite una mejor representación de la información visual. A este proceso se le denomina *prefiltrado* (Fig. I.4(a)).

Una vez que la imagen ha sido prefiltrada, se procede a muestrear y a cuantificar para obtener la imagen prefiltrada digital $\hat{f}(x,y)$. Si se desea ahora obtener la imagen analógica $f'(x,y)$, a partir de $\hat{f}(x,y)$, la principal dificultad es decidir qué valores de intensidad se deberán asignar a los puntos (x,y), que no corresponden a los puntos de muestreo. Se podría, por ejemplo, escoger el valor cero o asignar el valor de intensidad correspondiente al punto de muestreo más cercano. En ambos casos, la imagen f' presentará patrones artificiales que no existen en f . A este fenómeno se le conoce como *postraslape*. El postraslape puede ser sensiblemente reducido, interpolando (promediando), utilizando varios puntos. Esta interpolación puede introducirse en el sistema de despliegue. Entre mayor sea el polinomio de interpolación, se obtienen mejores resultados (Fig I.4(b)).

I.5 El barrido de líneas.

El barrido de líneas mostrado en la figura I.5, es la forma más utilizada para la representación de una imagen. El barrido de líneas convierte la señal bidimensional de intensidad $f(x,y)$, en una forma de onda unidimensional. Básicamente, la imagen $f(x,y)$ se segmenta en N_y líneas horizontales, iniciando en la parte superior de la imagen. La intensidad es medida a lo largo del centro de cada línea para obtener una forma de onda unidimensional que es función del tiempo (t) . Esto es, la imagen es barrida una línea a la vez, secuencialmente de izquierda a derecha y de arriba a abajo. Si se requieren T_L segundos para barrer cada línea, entonces el barrido completo de la imagen requiere $N_y \cdot T_L$ segundos.

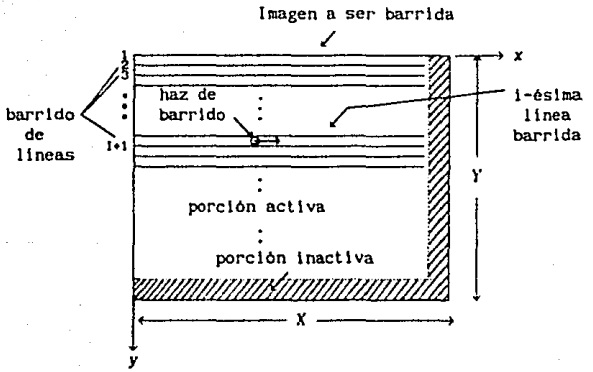


Fig. I.5 Barrido de líneas de una imagen. El haz de barrido se mueve a través de cada línea de izquierda a derecha y en cierta posición, determina un promedio espacial para dar la intensidad de luz en ese punto.

I.6 El dominio de la frecuencia.

El muestreo espacial y la cuantificación de la intensidad de luz de una imagen, no es el único método para representar la información visual a partir de un conjunto finito de datos. Algunas veces resulta más conveniente transformar la imagen a otro dominio de representación.

I.6.1 Series de Fourier.

Por ejemplo, supóngase que la frontera de las imágenes a representar es rectangular, es decir:

$$0 \le x \le X$$
$$0 \le y \le Y \qquad (I.10)$$

y para x e y, en este rango, se define la función exponencial compleja:

$$\phi_{mn}(x,y) = \frac{1}{\sqrt{XY}} \exp 2\pi \sqrt{-1} \left[\frac{nx}{X} + \frac{my}{Y} \right] \qquad (I.11)$$

Entonces, es posible representar la imagen $f(x,y)$ en términos de una serie de Fourier:

$$f(x,y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} c_{mn} \phi_{mn}(x,y) \qquad (I.12)$$

donde

$$c_{mn} = \int_0^X \int_0^Y f(x,y)\, \phi_{kl}^{*}(x,y)\, dy\, dx \qquad (I.13)$$

Aquí, $\{\phi_{mn}(x,y)\}$ es un conjunto ortonormal de funciones bidimensionales de base, es decir,

$$\int_0^X \int_0^Y \phi_{mn}(x,y)\, \phi_{kl}^{*}(x,y)\, dy\, dx = \delta_{mk} \cdot \delta_{nl} \qquad (I.14)$$

La representación de la imagen en términos de esas funciones bidimensionales, descompone la imagen en sus componentes de frecuencia espacial $\{c_{mn}\}$, esto es, se genera una representación espectral. Para cada función de base $\phi_{mn}(x,y)$, las frecuencias espaciales en las direcciones x e y son:

$$f_x = \frac{n}{X} \quad \text{ciclos por unidad de longitud}$$
$$f_y = \frac{m}{Y} \quad \text{ciclos por unidad de longitud} \qquad (I.15)$$

La frecuencia espacial total del patrón periódico se define como:

$$f_s = \sqrt{f_x^2 + f_y^2} \qquad (I.16)$$

En general, las componentes a frecuencias espaciales altas representan cambios rápidos de intensidad en la imagen. Las componentes a frecuencias bajas representan cambios tenues en la intensidad.

1.6.2 Imágenes limitadas en banda. El teorema del muestreo.

Una imagen que posee componentes de frecuencia en un rango finito, se denomina *limitada en banda*. Por ejemplo,

$$\bar{f}(x,y) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} c_{mn} \phi_{mn}(x,y) \qquad (I.17)$$

está limitada al rango de frecuencias:

$$\begin{aligned} |f_x| &< \frac{M + 0.5}{X} = F_x \quad \text{ciclos/longitud} \\ |f_y| &< \frac{N + 0.5}{Y} = F_y \quad \text{ciclos/longitud} \end{aligned} \qquad (I.18)$$

Para esta imagen, el número de coeficientes c_{mn} está dado por $N_s = (2M + 1)(2N + 1)$. Con el objeto de evaluar estos coeficientes, la integral de la ecuación I.13 debe ser evaluada. Sin embargo, en muchas situaciones, esta aproximación es difícil de llevarse a cabo.

Métodos prácticos para evaluar los coeficientes de Fourier (c_{mn}), se basan en las N_s $\{(x_k, y_k),\ k = 1 \ldots N_s\}$ muestras espaciales de la imagen limitada en banda $\bar{f}(x,y)$. Para cada una de las muestras (x_k, y_k), se tiene de la ecuación I.17, que permite expresar:

$$\bar{f}(x_k, y_k) = \sum_{|m| \le M} \sum_{|n| \le N} c_{mn} \phi_{mn}(x_k, y_k) \qquad (I.19)$$

Ahora, si las muestras se eligen de tal modo que las N_s ecuaciones de I.19 sean linealmente independientes, entonces es posible resolver el sistema para los N_s coeficientes desconocidos, c_{mn}.

Un conjunto posible de muestras consiste en un arreglo rectangular de $N_x = 2N + 1$ columnas y $N_y = 2M + 1$ renglones. En este caso $N_s = N_x N_y$. Es fácil mostrar que para $|m|, |k| \leq M$ y $|n|, |l| \leq N$, se tiene:

$$\sum_{i=0}^{Ny-1} \sum_{j=0}^{Nx-1} \phi_{mn}(x_j, y_i)\, \phi_{kl}(x_j, y_i) = \frac{N_s}{XY}\, \delta_{mk}\, \delta_{nl} \qquad (I.20)$$

donde,

$$x_j = \frac{X}{N_x}\, j \qquad y \qquad y_i = \frac{Y}{N_y}\, i$$

Aplicando este resultado en la ecuación I.20 se tiene que:

$$c_{mn} = \frac{XY}{N_s} \sum_{i=0}^{Ny-1} \sum_{j=0}^{Nx-1} \tilde{f}(x_j, y_i)\, \phi^{*}(x_j, y_i) \qquad (I.21)$$

Por lo tanto se puede observar que al muestrear $f(x,y)$ utilizando un arreglo rectangular es posible especificar completamente la imagen limitada en banda. Las $N_x = 2N + 1$ muestras deben de ser tomadas horizontalmente y las $N_y = 2M + 1$, verticalmente. Por lo tanto las densidades de muestreo respectivas son:

$$D_x = \frac{2N + 1}{X} = 2F_x \quad \text{muestras/longitud}$$
$$D_y = \frac{2M + 1}{Y} = 2F_y \quad \text{muestras/longitud} \qquad (I.22)$$

Comparando las ecuaciones I.18 y I.22 se puede observar que las densidades de muestreo son, al menos, dos veces mayores que las frecuencias espaciales más altas contenidas en la imagen. Esta es la esencia del bien conocido *teorema de muestreo de Nyquist* que se aplica a una señal de potencia. El teorema establece que :

Si $g(t)$ está limitada en banda en un rango de frecuencia W, esto es, la transformada de Fourier $G(f)$ es cero para las frecuencias $f \geq W$, entonces

$g(t)$ puede ser completamente especificada a partir de las muestras tomadas a la tasa de muestreo de Nyquist de $2W$ muestras por unidad de tiempo. Específicamente:

$$g(t) = \sum_{k=-\infty}^{\infty} g(\frac{k}{2W}) \; \mathrm{sinc}(2Wt - k) \qquad (I.23)$$

La versión bidimensional de tamaño finito del teorema de muestreo se escribe como:

$$\bar{f}(x,y) = \sum_{i=0}^{2M} \sum_{j=0}^{2N} \bar{f}\,(x_j, y_i) \; X\,(x - x_j,\; y - y_i) \qquad (I.24)$$

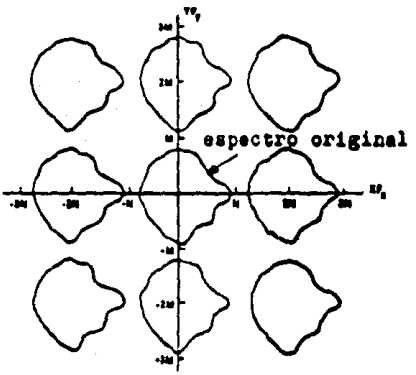


Fig. I.6 Espectro de frecuencia de Fourier (planta) de la imagen muestreada $\hat{f}(x,y)$. Xf_x y Yf_x son ciclos verticales y horizontales respectivamente. Esta es periódica e infinita. La porción de baja frecuencia es el espectro original limitado en banda de la imagen $\bar{f}(x,y)$. Las porciones restantes son espectros réplicas causados por el proceso de muestreo.

donde,

$$X(u,v) = \frac{1}{N_s} \sum_{\substack{|m|\le M \\ |n|\le N}} \exp 2\pi\sqrt{-1}\left\{\frac{mu}{x} + \frac{nv}{y}\right\} \qquad (I.25)$$

En la figura I.6 se muestra el espectro de una imagen limitada en banda muestreada de acuerdo a la ecuación I.22 . En ella se muestra cómo el el espectro se reproduce centrado en las coordenadas múltiplos de N y M .

Si la imagen $\bar{f}(x,y)$ contiene componentes de frecuencia arriba de la mitad de la frecuencia de muestreo, entonces no es posible reconstruir la imagen $f(x,y)$ puesto que se han traslapado las réplicas del espectro con el espectro original de $f(x,y)$ (Fig. I.7). Este es el mismo fenómeno señalado

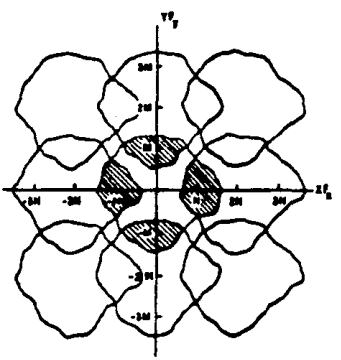


Fig. I.7 Espectro de la imagen muestreada $\hat{f}(x,y)$ donde la imagen original $\bar{f}(x,y)$ contiene componentes de frecuencia arriba de la mitad de la frecuencia de muestreo. Las componentes en las áreas sombreadas están traslapadas por los espectros réplicas. Esta distorsión es llamada pretraslape y es debida a un prefiltrado insuficiente de $\bar{f}(x,y)$, o bien a una frecuencia de muestreo inadecuada.

en la sección 1.4 : el *pretraslape*.

El fenómeno del *postraslape* se muestra en la figura 1.8 . Se origina por el hecho de que el filtro de reconstrucción $X(u,v)$ no pudo ser implantado y se utilizó algún método de reconstrucción subóptimo.

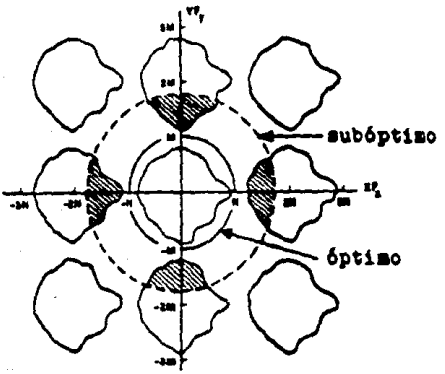


Fig. 1.8 Posfiltrado no lineal. La linea punteada indica la extensión de la frecuencia de un posfiltro subóptimo. La linea continua indica un filtro óptimo. Las réplicas del espectro original en las áreas sombreadas aparecen en la imagen reconstruida. Esta distorsión es llamada postraslape y es debida a un posfiltrado inadecuado de $\hat{f}(x,y)$.

I.6.3 Espectro de frecuencia de las imágenes en las que se ha utilizado el barrido de lineas.

La señal $f(x,y)$ que ha sido convertida en una señal en función del tiempo, $f(t)$, puede ser muestreada y reconstruida utilizando el teorema de Nyquist. Sin embargo, debido a que existen discontinuidades cada T_L segundos,

cuando $f(t)$ es filtrada se producen efectos indeseables. Por otro lado posee un alto contenido armónico debido a la periodicidad ($1/T_L$) del barrido de líneas (Fig. I.9) .

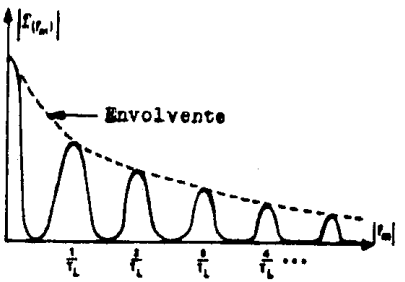


Fig. I.9 Representación en el dominio de la frecuencia de una señal de video que ha sido barrida unidimensionalmente. $|f(r_m)|$ tiene un gran contenido armónico en frecuencias múltiplos de $1/T_L$. Los anchos de los lóbulos están determinados por las componentes verticales de frecuencia espacial f_y . El ancho de banda completo de $F(t)$ está determinado por las componentes de frecuencia espacial f_x .

C A P I T U L O I I

EL SISTEMA DE COMUNICACION DE TELEVISION MONOCROMATICA

CAPITULO II. EL SISTEMA DE COMUNICACION DE TELEVISION MONOCROMATICA.

La mayoria de las señales de televisión (para grabación y transmisión), son analógicas y su transmisión, utilizando como medio el aire o un cable coaxial, se lleva a cabo mediante la modulación en amplitud (AM) con portadoras de radiofrecuencia en las bandas VHF o UHF . Las transmisiones via satélite o microondas se llevan a cabo mediante la modulación en frecuencia (FM). Un gran porcentaje de las transmisiones por fibra óptica se llevan a cabo mediante la modulación de pulsos en frecuencia, es decir, la frecuencia de un tren de pulsos ópticos es modulada mediante la señal de video.

Las señales digitales de televisión, pueden ser encontradas en muchos de los equipos de procesamiento de señales de un estudio de televisión, por ejemplo, en correctores de base de tiempo, sincronizadores de marco y generadores de efectos especiales. Sin embargo, la transmisión digital de señales de televisión, ha comenzado a desarrollarse recientemente y su uso generalizado podrá esperarse en los años siguientes.

Un sistema típico de comunicación de televisión se muestra en la figura II.1 . Virtualmente todos los sistemas de televisión monocromática emplean un barrido de lineas entrelazadas en una proporción 2:1 . Existen muchos estándares internacionales y algunos de ellos se muestran en la tabla II.1 . La relación entre el ancho de la imagen y su altura se denomina *razón de aspecto* y en todos los sistemas de la tabla II.1 es de 4:3 . Sin embargo, para la televisión de alta definición †, la razón de aspecto es de 5.3 .

Muchas cámaras de televisión y casi todos los tubos de rayos catódicos, tienen una relación no lineal inherente entre la señal de voltaje y la intensidad de luz (B). Una ley exponencial de la forma:

$$B = c \cdot v^{\gamma} + B_0 \qquad \text{(II.1)}$$

---

† En los sistemas de alta definición (HDTV) el ancho de banda y la tasa de transmisión es al menos cuatro veces mayor que en cualquier otro sistema de televisión estándar.

se asume generalmente como la relación entre la intensidad de luz y el voltaje. En la ecuación II.1 B representa la intensidad de luz, v el voltaje sobre el corte, c es un factor de ganancia constante, B_0 es el nivel de corte o nivel de negro y γ está en el rango de 1.0 a 3 . Por ejemplo, el tubo de la cámara de televisión puede tener una $\gamma \approx 0.7$. El tubo de rayos catódicos de un monitor de televisión tiene una γ del orden de 2.5 a 3 .

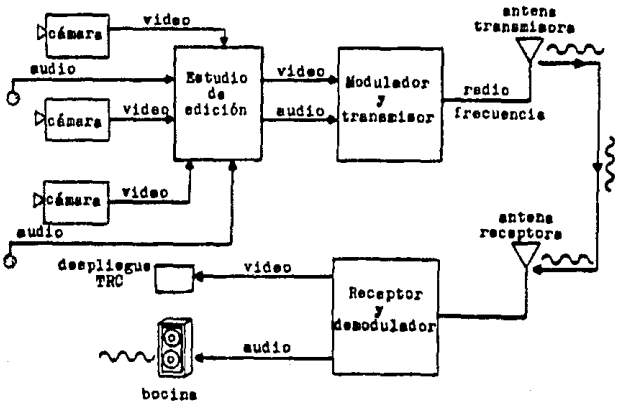


Fig II.1 Sistema de comunicación típico de televisión.

Con el objeto de evitar una circuitería de conexión dentro de cada uno de los millones de receptores de televisión, se utiliza la *corrección gama* dentro de cada cámara de televisión antes de la transmisión. Esto es, si γ_c es la gama de la cámara y γ_d corresponde al tubo de rayos catódicos (TRC) del monitor de televisión, entonces el voltaje v_c es corregido a la salida mediante la relación:

$$v_d = v_c^{\gamma_c/\gamma_d} \qquad \text{(II.2)}$$

para obtener un voltaje v_d que pueda ser transmitido y aplicado directamente al TRC del monitor de televisión. El voltaje corregido v_d , es menos susceptible al ruido de transmisión.

TABLA II.1 Estándares internacionales para la televisión monocromática. El sistema M se utiliza en EUA y Japón, los sistemas A e I se utilizan en el Reino Unido; Francia utiliza los sistemas E y L. El resto de los países de Europa occidental utilizan los sistemas B, C, G y H. Los países de Europa oriental utilizan los sistemas D y K.

Estándar	A	M	B,C,G,H	I	D,K,L	E
Líneas/Marco	405	525	625	625	625	819
Campos/Segundo	50	60	50	50	50	50
Marcos/Segundo	25	30	25	25	25	25
Líneas/Segundo	10125	15750	15625	15625	15625	20475
Ancho de banda (MHz)	3.0	4.2	5.0	5.5	6.0	10.0
Razón-Bit Mbs (8 bits PCM)†	48	67.2	80	88	96	160
Modulación		AM		FM		FM

Aproximadamente el 18% de cada línea de barrido es eliminada, para permitir un retraso horizontal y que sea posible reiniciar el barrido cada que se finaliza una línea. De igual forma, entre el 6 y 10 % de cada campo es eliminado para permitir el retraso vertical al inicio de cada campo. En todas las líneas eliminadas se introducen pulsos de sincronización como se muestra en la figura II.2 , con el objeto de proveer información de tiempo para los circuitos de sincronía vertical y horizontal a los aparatos receptores. La señal de información se transmite junto con los pulsos de sincronización.

† Capítulo V.

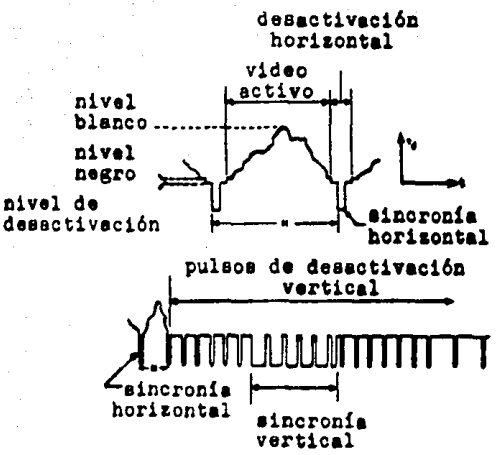


Fig. II.2 Pulsos de sincronización que se suman a la señal de video.

La señal de televisión monocromática puede ser muestreada a la frecuencia de Nyquist, es decir, dos veces el ancho de banda de dicha señal (13.5 MHz en Francia) y las muestras cuantificadas con 8 bits, con el objeto de representar adecuadamente la información visual. El método más usado para lograr esta representación se conoce comúnmente como PCM (Modulación por Pulsos Codificados).

CAPITULO III

METODOS DE COMPRESION

CAPITULO III. METODOS DE COMPRESION.

En los dos primeros capítulos se anotó la forma en que se puede representar la información visual utilizando una cantidad finita de datos digitales. Sin embargo, los datos producidos por estas técnicas, normalmente contienen una cantidad considerable de información redundante, que puede ser removida utilizando los métodos que se mencionan en éste y los siguientes capítulos.

Generalmente, es posible hablar de dos tipos de información redundante. La primera es la *redundancia estadística*, que se relaciona con la similaridad, la correlación y la predictibilidad de los datos. La redundancia estadística tiene la propiedad de poder ser removida de los datos sin destruir la información esencial. El segundo tipo de información superflua es la *redundancia subjetiva*, que tiene que ver con las características de los datos que pueden ser removidas sin que el observador humano lo note. Pero a diferencia de la redundancia estadística, la eliminación de la redundancia subjetiva es un proceso irreversible. Los datos originales no pueden ser recuperados una vez que se ha removido la redundancia subjetiva. El proceso para lograr la representación de la información libre de redundancia estadística se denomina *codificación de la información*.

En este capítulo se hará referencia a uno de los métodos de eliminación de redundancia estadística: *la codificación predictiva*. En el capítulo siguiente se tratarán brevemente algunas de las características de la redundancia subjetiva.

En lo que sigue, se asumirá que las imágenes poseen propiedades estadísticas medibles, aunque en la realidad este hecho rara vez es cierto. Bajo esta asunción cada punto de la imagen es considerado una variable aleatoria.

En cuanto a la notación que se utilizará se deberá distinguir entre una variable aleatoria y el conjunto de sus posibles valores. Así, cuando sea necesario, se denotará una variable aleatoria con letras mayúsculas y uno de sus posibles valores con minúsculas. El conjunto de sus posibles valores se escribirá entre llaves. Se representará la distribución de probabilidad (*PD*) de una variable aleatoria, B , mediante $\{P(b)\}$. Se denotarán los valores de

varios puntos (usualmente adyacentes) de la imagen como $(b_1 b_2 \ldots b_N)$. Se hará referencia a ellos como vectores b de longitud N. El correspondiente vector aleatorio es B, o algunas veces $(B_1 B_2 \ldots B_N)$ y su PD se denota como $\{P(b)\}$ o también como $\{P(b_1 b_2 \ldots b_N)\}$.

Se asumirá también que la estadística de la imagen es estacionaria, es decir, la estadística de los puntos es la misma en todas las partes de la imagen y no cambia con el tiempo. Esta suposición rara vez es completamente cierta en la realidad.

III.1 La redundancia en las imágenes digitales.

Supógase que se tiene una imagen monocromática que tiene $N \cdot N$ puntos, cuantificados con m bits, de acuerdo a las técnicas del capitulo I. Supóngase además, que los valores de luminancia cuantificados no son igualmente probables, teniendo el histograma de la figura III.1.

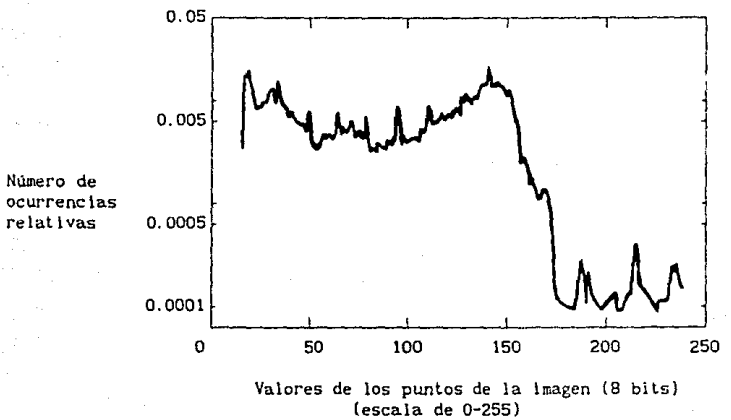


Fig. III.1 Ejemplo de un histograma de luminancia. Los valores de luminancia están cuantificados con 8 bits en el rango de 0 a 255.

En este caso, es posible la reducción del número de bits requeridos para especificar la imagen, si en lugar de asignar palabras de m bits a los 2^m posibles niveles de gris, se asignan palabras de *longitud variable*. De este modo, si se asignan palabras más cortas a aquellos niveles con mayor probabilidad de ocurrencia y palabras más largas a los niveles con menor probabilidad, entonces se puede reducir el número promedio de bits por palabra de código. A este procedimiento, en el que se utilizan palabras de código con longitud variable se le denomina *codificación entrópica*.

Por ejemplo, supóngase que el nivel de cuantificación de luminancia b tiene la probabilidad de ocurrencia $P(b)$ y que le es asignada una palabra de código de longitud $L(b)$, en bits. Entonces, el promedio de longitud de palabras de código para la imagen en consideración es:

$$\bar{L} = \sum_{b} L(b)P(b) \qquad \text{bits/punto} \qquad \text{(III.1)}$$

en donde la sumatoria se hace sobre los 2^m posibles valores de los puntos de la imagen. Sería deseable encontrar códigos para los que $\bar{L}$ es lo más pequeña posible.

La longitud de palabra promedio $\bar{L}$, no puede ser arbitrariamente pequeña, sin embargo, puede ser correctamente decodificada por un receptor. En efecto, el límite inferior de $\bar{L}$ ha sido determinado en la teoría de la información. Este límite se denomina *entropía* de la variable aleatoria B y está dada por:

$$H(B) = -\sum_{b} P(b) \log_2 P(b) \qquad \text{(III.2)}$$

La entropía nunca es negativa ya que $P(b)$ está en el rango de $[0,1]$. Como ya se estableció:

$$H(b) \leq \bar{L} \qquad \text{(III.3)}$$

para todos los códigos de longitud de palabra variable decodificables, que pueden codificar puntos independientemente unos de otros.

La entropía $H(b)$ es una medida de la cantidad de información que posee la variable aleatoria B . Por ejemplo, si el valor de B es altamente predecible*, es decir, $P(b) = \delta_{bb_o}$, entonces la entropía es cero. Por otra parte, si todos los valores de B son igualmente probables, es decir, $P(b) = 2^{-k}$, entonces la entropía es máxima y $H(b) = m$ bits/punto. Nótese que en el último ejemplo no tiene objeto codificar con longitud de palabra variable.

De esta forma, conforme $\{P(b)\}$ está más concentrada, la entropía es más pequeña y el beneficio de una codificación con longitud de palabra variable, aumenta.

III.2 Códigos de Huffman.

Un *código compacto* es aquel, que en promedio, tiene una longitud de palabra menor o igual que el promedio de longitud de todos los códigos individuales, univocamente decodificables, para el mismo conjunto de probabilidades de entrada, esto es, tiene un mínimo de longitud de palabra. Dado un conjunto de probabilidades de entrada es posible generar un código compacto utilizando el algoritmo de Huffman [1952]. Un código de Huffman puede ser construido ordenando las probabilidades de entrada con respecto a su magnitud, como se ilustra en el ejemplo de la figura III.2 , para seis posibles valores de entrada. Las dos probabilidades más pequeñas se suman para formar un nuevo conjunto de probabilidades. El nuevo conjunto de probabilidades, tiene un valor menos de probabilidad que el conjunto original. Se ordena nuevamente de acuerdo a su orden de magnitud: las probabilidades iguales pueden ser ordenadas de cualquier forma (por ejemplo, el 0.1 obtenido de la suma de las probabilidades 0.06 y 0.04 puede ser colocado en cualquiera de las tres últimas posiciones de la columna 1). El proceso se repite hasta que se obtiene una columna con dos valores. Hasta aquí, se ha terminado la primera parte del algoritmo.

---

* $\delta_{bb_o} = 1$ si $b = b_o$; $\delta_{bb_o} = 0$ si $b \neq b_o$

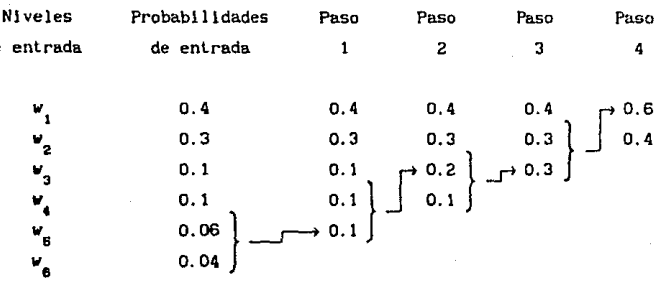


Fig. III.2 Generación de un código de Huffman. Primera etapa en la construcción.

Las palabras del código se generan de la forma siguiente: comenzando en la última columna (la 4) hacia la primera. Se asigna un 0 a uno de los dos valores de la columna 4 y un 1 al otro valor, como se ilustra en la figura III.3 , en donde se ha colocado un 0 a la izquierda del valor de probabilidad 0.6 y un 1 al lado del valor de 0.4 . Se propagan entonces el 1 y el 0 hacia la columna 3 tomando en cuenta la suma de las probabilidades llevada a cabo

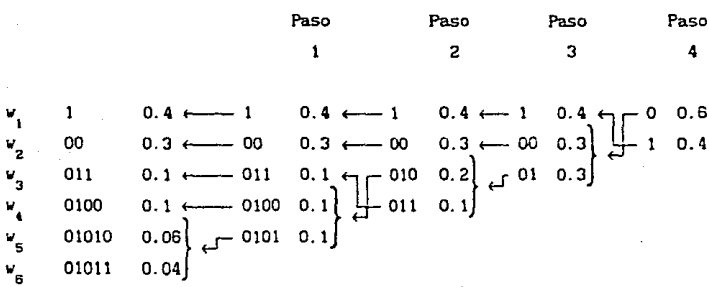


Fig. III.3 Construcción de un código de Huffman. Segunda etapa.

para obtener la columna 4 . Por ejemplo, el 0 asociado al 0.6 obtenido en la columna 4 , se retiene en la columna 3 como el primer bit de las palabras de código de esta columna, tomando como referencia las probabilidades que dieron origen al 0.6 . El 1 asociado al 0.4 se retiene de la misma manera en la columna 3 . Un segundo bit, un 0 y un 1 respectivamente, es asignado a cada una de las palabras de código asociadas con las probabilidades reconstruidas para obtener las palabras de código de la columna 3 . El mismo procedimiento se repite hasta que se llega a la columna 1 y finalmente a las probabilidades de entrada. En esta etapa, se tienen ya las palabras de código asignadas a cada nivel de entrada w_i . Es posible demostrar que el procedimiento bosquejado genera un código compacto [3].

Para las probabilidades de entrada listadas en la figura III.2 la entropía es:

$$\begin{aligned} H &= (-0.4)log\ (0.4) - (0.3)log\ (0.3) - (0.1)log\ (0.1) \\ &\quad - (0.1)log\ (0.1) - (0.6)log\ (0.6) - (0.04)log(0.04) \\ &= 2.14 \text{ bits.} \end{aligned}$$

La longitud de palabra del código de Huffman de este ejemplo es:

$$\begin{aligned} \bar{L} &= 1(0.4) + 2(0.3) + 3(0.1) + 4(0.1) + 5(0.06) + 5(0.04) \\ &= 2.2 \text{ bits.} \end{aligned}$$

Si bien un código de Huffman minimiza el promedio de la longitud de palabra, otra clase de códigos también son usados.

Los códigos de palabras de longitud fija requieren más bits, pero tienen ventajas cuando se envían por un canal de transmisión. Los codigos de longitud de palabra variable, por otra parte y los códigos de Huffman en general, pierden sincronización en el caso de que se produzca un error de transmisión. Esto es, muchas palabras del código son decodificadas erróneamente antes de que la decodificación correcta sea reestablecida.

III.3 Codificación predictiva.

Existen además de la codificación entrópica, otros esquemas que permiten extraer la redundancia, cada uno con características particulares:

a) *Codificación por bloques.*
b) *Codificación condicional.*
c) *Codificación predictiva.*

En la *codificación por bloques*, los puntos de una imagen son codificados en bloques de N muestras en lugar de punto por punto. La dificultad de este esquema estriba en que generalmente los bloques son de gran tamaño. Una tabla o un árbol con 2^{Nm} palabras de código debe ser incluido con el objeto de codificar los vectores **b** de los puntos cuantificados, cada uno con m bits.

En la *codificación condicional*, se asume que las componentes $b_1 b_2 \ldots b_{N-1}$ del vector **b** han sido transmitidas y son conocidas en el receptor. La componente b_N , puede ser codificada más eficientemente utilizando esta información (asumiendo que existe dependencia entre los puntos de la imagen). Específicamente para un vector particular $(b_1 b_2 \ldots b_{N-1})$, un código de Huffman para b_N puede ser construido basado en la *PD* condicional:

$$\{P(b_N | b_1 b_2 \ldots b_{N-1})\}. \qquad \text{(III.4)}$$

El mayor problema al emplear un esquema de codificación condicional es el gran número de conjuntos de códigos de longitud de palabra variable, que deben generarse durante la codificación y la decodificación. Este número es $2^{m(N-1)}$ y resulta demasiado grande para permitir una implantación práctica.

Una forma de reducir el número de códigos a utilizar es emplear el concepto de *espacio de estado*. Con esta aproximación el conjunto de $N - 1$ vectores:

$$(b_1 b_2 \ldots b_{N-1}) \qquad \text{(III.5)}$$

es particionado en J subconjuntos o estados $\{s_j , j = 1 \ldots J\}$ de tal forma que la *PD* condicional:

$$\{P(b_N | b_1 b_2 \ldots b_{N-1})\} \qquad \text{(III.6)}$$

es aproximadamente invariante para los $(N-1)$ vectores $\mathbf{b}$, en cada subconjunto. De acuerdo a esto, los códigos de longitud de palabra variable J se construyen con base en la probabilidad de J:

$$\{P(b_N | s_1)\} \qquad \text{(III.7)}$$

Cada uno de los códigos J, tiene 2^m palabras de código, para dar como total $J \cdot 2^m$ palabras de código que, con una partición apropiada, es considerablemente más pequeño, que el número de 2^{Nm} palabras de código, requeridas en la codificación por bloques, o condicional.

Todas las técnicas de codificación que se han mencionado, se utilizan en los casos en que la estadística de los datos no es inicialmente conocida en el codificador. Básicamente, estas técnicas, llamadas *algoritmos universales de codificación*, tratan de medir la estadística durante la operación de codificación actual y adaptarse con el objeto de maximizar la compresión.

La *codificación predictiva* (Fig. III.4) está relacionada con la codificación condicional. La principal diferencia, estriba en que, en lugar de codificar y transmitir b_N, se transmite otra cantidad, $b_N - \hat{b}_N$, en donde $\hat{b}_N$ es una predicción de b_N y es usualmente una función determinística de los valores previamente decodificados $b_1 b_2 \ldots b_{N-1}$. Se denominará a $b_N - \hat{b}_N$ la *señal de diferencia* o *error de predicción*.

Debido a que la diferencia está disponible en el receptor, b_N puede ser recuperada en el decodificador, sumando a la diferencia, la predicción.

Una de las ventajas de la codificación predictiva es que la *PD* condicional:

$$\{P(b_N - \hat{b}_N \mid b_1 b_2 \ldots b_{N-1})\} \qquad \text{(III.8)}$$

usualmente presenta mucho menos variaciones respecto a $\{b_1 b_2 \ldots b_{N-1}\}$ que la variación que presenta la *PD* condicional de b_N. En efecto, con una codificación predictiva en el espacio de estado, el número de estados puede ser reducido significativamente en comparación al espacio de estado de la codificación condicional. De hecho, sólo un estado es utilizado, es decir, no

existe codificación condicional y solamente un código de longitud de palabra variable necesita ser diseñado con base en la ecuación (III.9):

$$\{P(b_N - \hat{b}_N)\} \qquad \text{(III.9)}$$

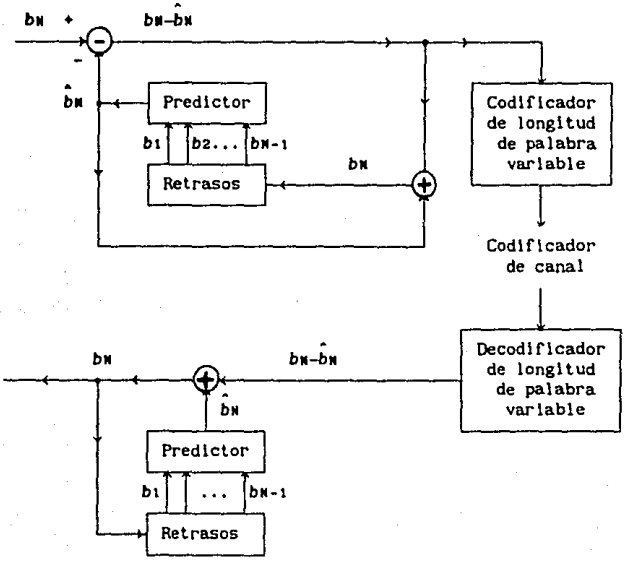


Fig. III.4 Diagrama de bloques de una codificación predictiva.

Otra ventaja de un sistema de codificación predictiva, es que las señales sucesivas de diferencia, usualmente tienen una dependencia estadistica menor, es decir, menos redundancia, que los puntos sucesivos. En efecto, un codificador predictivo óptimo tiene salidas estadisticamente independientes.

El principal objetivo de la codificación predictiva es producir señales de diferencia, en promedio pequeñas y grandes ocasionalmente, es decir, la entropía es pequeña. De este modo, entre mejor sea el predictor, la entropía es menor. Por ejemplo, un predictor que minimiza el error de predicción cuadrático medio es:

$$\hat{b}_N = \sum b_N \; P(b_N | b_1 b_2 \ldots b_{N-1}) \qquad \text{(III.10)}$$

y se denomina *predictor de media condicional*.

Un predictor *lineal* se escribe como:

$$b = \sum_{i=1}^{N-1} \alpha_i b_i \qquad \text{(III.11)}$$

donde los coeficientes de ponderación $\{\alpha_i\}$ se escogen para que su implantación sea sencilla. Un predictor lineal que minimiza el error cuadrático medio se obtiene al diferenciar $E(b_N - \hat{b}_N)$.

CAPITULO IV

EL SISTEMA VISUAL HUMANO

CAPITULO IV. EL SISTEMA VISUAL HUMANO.

El presente capitulo tiene como propósito introducir algunos conceptos relacionados con el funcionamiento del sistema visual humano. Se hace referencia también a una serie de resultados, producto del estudio de este sistema y su aplicación a la codificación de imágenes digitales.

IV.1 Aspectos generales del sistema visual humano.

En esta sección, se presenta una síntesis breve del funcionamiento del sistema visual humano y algunas de las características consideradas relevantes en el contexto del procesamiento de imágenes digitales.

IV.1.1 El mecanismo de visión humano.

El elemento final de la mayoría de los sistemas de procesamiento de imágenes digitales es el ojo humano. Por lo tanto es conveniente tener una idea de qué es capaz de ver el sistema visual humano y cómo lo hace.

El sistema visual humano es una parte del sistema nervioso. Este último es, sin duda alguna, la red de comunicaciones más complicada que existe y es controlado por el cerebro. La comunicación en esta red se lleva a cabo a través de las células nerviosas denominadas neuronas. El cerebro contiene alrededor de 10^{11} neuronas.

Una neurona es una célula cuyo tamaño varia entre 5 y 100 μm. Posee una fibra central denominada axón y un número considerable de ramas llamadas dendritas. La transferencia de información de una neurona a otra se hace electroquímicamente. La unión entre dos neuronas se denomina sinapsis. Las neuronas transmisoras y receptoras se llaman, respectivamente: presinápticas y postsinapticas. La información que se genera en una neurona presináptica viaja a través de su axón como una señal eléctrica en un cable. Las ramas terminales del axón transmiten esta señal a las dendritas de la neurona

postsináptica. Durante la transmisión, la señal eléctrica provoca la secreción de un mediador químico en las ramas conectadas a la neurona. A través del mediador, el impulso eléctrico viaja a otras neuronas. Una neurona puede recibir señales de cientos de neuronas presinápticas y puede transmitir a cientos de neuronas postsinápticas. Una sola neurona puede contar hasta con 200,000 sinapsis. La acción de una neurona puede ser de dos tipos: excitadora o inhibidora. La primera genera pulsos en la neurona postsináptica, mientras que la segunda inhibe los pulsos existentes.

Para tener una idea de la complejidad del sistema nervioso, es posible imaginar una red de 10^{12} neuronas conectadas en cascada, en paralelo y con realimentación. Una vista esquemática de esta organización se muestra en la figura IV.1 .

El estudio del cerebro y su funcionamiento se ha dado, gracias a que es posible estudiar una sola neurona o un grupo pequeño de ellas. Es decir, existen características que pueden simplificar el estudio de una red tan complicada. Una de estas características, es que sólo existen dos tipos de señales en el sistema nervioso, a saber, las que se propagan a grandes

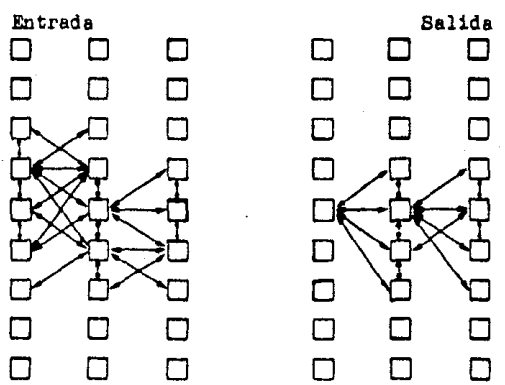


Fig. IV.1 Diagrama esquemático del sistema nervioso.

distancias y las que se propagan a distancias pequeñas. La segunda característica es que estas señales son casi idénticas sea cual fuere el tipo de información que contenga (visual, táctil, audible, etc.). Más aún, la forma de la señal no varía significativamente entre diferentes seres vivos, de tal forma que la señal proveniente del sistema nervioso de un gato es muy similar a la señal que se observa en el sistema nervioso humano. Las señales recibidas y procesadas por el cerebro son símbolos que representan eventos externos. El sistema nervioso ha sido analizado por los neurofisiólogos de tres formas: la microscopía electrónica, las tinturas selectivas y por medio de microelectrodos. La señal medida de una neurona es un tren de pulsos de 100 mV y de 1 ms de duración. La tasa de repetición de los pulsos es proporcional a la intensidad del estímulo. El sistema nervioso se comunica por frecuencia modulada. Ello permite al cerebro distinguir la ruta específica de dos señales idénticas. Esto implica la existencia de un conjunto específico de neuronas que reaccionan a cada tipo de excitación. Desde el punto de vista matemático existe una relación uno a uno entre las diferentes partes del cerebro y del cuerpo.

IV.1.1.1 La estructura del ojo humano.

Una sección transversal simplificada del ojo humano se muestra en la figura IV.2 . El ojo tiene una forma aproximadamente esférica con un diámetro promedio de unos 20 mm . El ojo se encuentra encerrado por tres membranas: la córnea y la esclerótica que lo cubren exteriormente, el coroides y la retina. La córnea es un tejido transparente y flexible que cubre la superficie anterior del ojo. La esclerótica es una membrana adyacente a la córnea que cubre la parte restante del globo óptico.

El coroides se encuentra dentro de la esclerótica. Esta membrana contiene una red de venas que nutren al ojo de sangre. El recubrimiento del coroides está muy pigmentado y por ello es posible reducir la cantidad de luz externa que entra al ojo y que se difunde dentro del ojo mismo. En su extremo anterior, el coroides está dividido en el *cuerpo ciliar* y el *diafragma del iris*. Este último se contrae o expande para controlar la cantidad de luz que

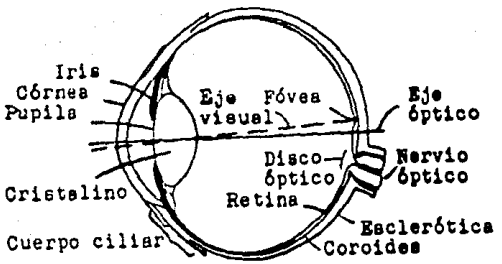


Fig. IV.2 Diagrama simplificado de una sección transversal del ojo humano.

puede penetrar al ojo. La abertura central del iris (la pupila) tiene un diámetro variable de entre 2 y 8 mm . El frente del iris contiene el pigmento visible del ojo, mientras que la parte posterior contiene pigmento negro.

La membrana más interna del ojo es la retina, la cual se alinea en la parte posterior del ojo, cuando éste enfoca adecuadamente. La luz del objeto que se observa incide en la retina y ahí se forma la imagen. La visión de patrones se lleva a cabo mediante la distribución discreta de receptores de luz sobre la superficie de la retina. Hay dos clases de receptores: los *conos* y los *bastones*. El numero de conos en cada ojo es de aproximadamente 6.5 millones; están localizados principalmente en la porción central de la retina denominada fóvea y son muy sensitivos al color. El ojo es capaz de observar muy finos detalles puesto que cada cono está conectado a una terminal nerviosa. Los músculos del ojo controlan su movimiento para que el objeto de interés caiga en la fóvea. La visión mediante los conos se denomina *fotópica*.

El número de bastones es mucho mayor que el número de conos, siendo del orden de entre 75 y 150 millones distribuidos sobre toda la superficie de la retina. La amplia distribución y el hecho de que muchos bastones estan conectados a una sola terminal nerviosa reduce la resolución de los detalles discernibles por estos receptores. Los bastones dan un campo general de

visión, no se relacionan con la visión de los colores y son muy sensitivos a los niveles de iluminación. Por ejemplo, los objetos brillantes que aparecen a la luz de la luna estimulan a los bastones. A este tipo de visión se le denomina *escotópica*.

Como se ve en la figura IV.3, los fotoreceptores hacen un contacto sináptico con las células bipolares que se extienden a través de las capas nerviosas de la retina más cercanas al cristalino. Una segunda sinapsis interna conecta a las células bipolares con las células ganglionares. También

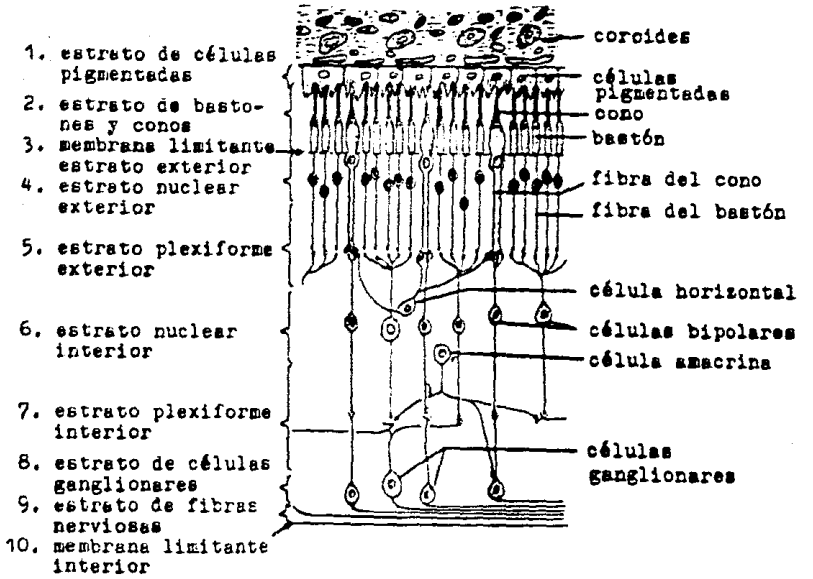


Fig. IV.3 Diagrama esquemático de la retina en donde se muestran las interconexiones entre los receptores (bastones y conos) y las células ganglionares, horizontales y amacrinas.

ocurren interacciones laterales por medio de las células horizontales y amacrinas. Los axones de las células ganglionares forman las fibras del nervio óptico por medio del cual la información es transmitida al cerebro. Las conexiones laterales entre las células horizontales y amacrinas son responsables de la *inhibición lateral*.

Si bien, la transmisión de la señal desde los fotoreceptores hasta las células se lleva a cabo mediante una reacción química y la conducción de ciertos químicos, los potenciales eléctricos permiten la transmisión entre las células fotoreceptoras, horizontales, amacrinas y bipolares. Los potenciales en las células ganglionares controlan la tasa de generación de pulsos eléctricos que se propagan a través del nervio óptico. Se cree que la codificación de información está relacionada con la tasa de disparo de estos pulsos.

El cristalino se encuentra formado por capas concéntricas de células fibrosas que se sujetan al cuerpo ciliar. Contiene entre 60 y 70 % de agua, cerca del 6 % de nutrientes y más proteínas que ningún otro tejido en el ojo. El cristalino tiene una pigmentación amarilla tenue que se incrementa con la edad. Absorbe aproximadamente el 8 % del espectro de luz visible, con mayor absorción para menores longitudes de onda. La luz ultravioleta e infrarroja son absorbidas apreciablemente por las proteínas dentro de la estructura, cabe señalar, que el exceso de esta clase de radiación puede dañar al ojo.

Tanto el cristalino como la córnea pueden tener defectos que provocan aberraciones ópticas. Si la curvatura de la córnea no decrece suficientemente, los rayos luminosos que pasan por el borde forman un foco más lejano que los que pasan por el centro y ello, produce la denominada *aberración esférica*. Por otro lado, debido a que el cristalino refracta con mayor intensidad en el centro, también se produce la aberración esférica.

La *aberración cromática* es otra aberración importante en la que una lente refracta los rayos luminosos de longitud de onda corta más fuertemente que los de onda larga. Por tanto, los rayos luminosos de diferentes colores forman su foco a distintas distancias detrás del cristalino. El resultado es una imagen borrosa con bordes irrizados. El ojo corrige la aberración cromática mediante el pigmento amarillo del cristalino que absorbe, como se dijo, los rayos de luz de longitud de onda menor a 365 μm, región del

espectro en donde la aberración cromática es mayor. Existe además una acción filtrante en la pigmentación amarilla de mácula lútea que atenúa la visión en las regiones violeta y azul del espectro, en las cuales la aberración cromática es elevada. Protege así a los receptores de la fóvea que están más expuestos.

La aberración esférica y la acción de la pupila al abrirse y cerrarse pueden ser modeladas matemáticamente como filtros bidimensional y unidimensional, respectivamente. Por ejemplo, la frecuencia de corte máxima de la acción de la pupila al regular la cantidad de luz se presenta cuando el diámetro de la misma es de 2 mm y decrece conforme la pupila alcanza un diámetro máximo de 8 mm .

Además de las degradaciones producidas a la imagen debido a las imperfecciones del sistema óptico humano, existe otra fuente que degrada la imagen: el movimiento del ojo. El movimiento voluntario es necesario y permite perseguir objetos en movimiento o desviar la mirada de un objeto a otro. Los movimientos involuntarios ocurren aún durante una estado de fijación, e introducen ciertas variaciones temporales a cualquier imagen. Si bien los movimiento involuntarios del ojo degradan la imagen en general, son importantes para mantener una visibilidad continua del campo visual, ya que una imagen estacionaria en la retina tiende a desvanecerse y eventualmente a desaparecer.

La figura IV.4 muestra una representación esquemática del camino que sigue la información visual hasta la corteza cerebral. Un procesamiento espacial importante ocurre más allá de la retina. Las fibras de cada nervio óptico se dividen en el quiasma óptico. La mitad de las fibras correspondientes a la porción derecha de la pared retinal van al núcleo geniculado lateral en el lado derecho del cerebro y la otra mitad van al núcleo geniculado lateral en el lado izquierdo del cerebro. En el núcleo geniculado lateral se establece la comunicación con la parte de la corteza cerebral encargada de procesar la información visual (corteza visual).

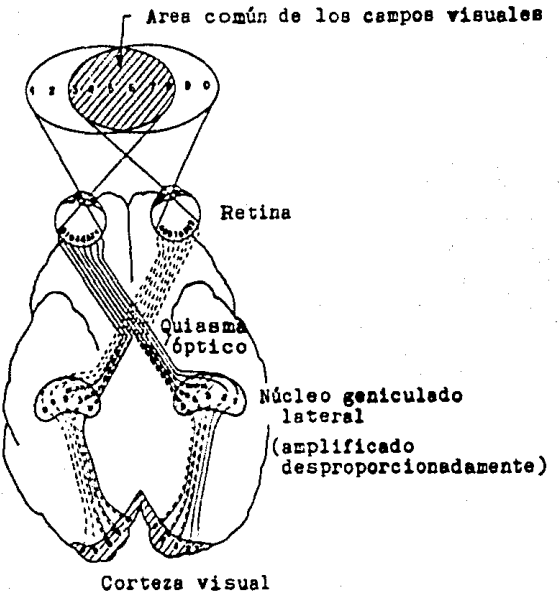


Fig. IV.4 Representación esquemática de la trayectoria que sigue la información visual hasta la corteza cerebral, vía el quiasma óptico y el núcleo geniculado lateral.

IV.1.1.2 La formación de una imagen en el ojo.

Un objeto externo es afocado por la córnea y el cristalino para formar la imagen del objeto sobre la retina en la parte trasera del glóbulo ocular. La diferencia principal entre las lentes ordinarias y el cristalino del ojo es la flexibilidad de este último. El radio de curvatura de la superficie anterior es mayor que el radio de la cara posterior. La orientación del cristalino está controlada por la tensión en las fibras del cuerpo ciliar (Fig. IV.2). Para enfocar objetos distantes, el control de los músculos

provoca que el cristalino sea ligeramente aplanado. De igual forma, estos músculos permiten que el cristalino se vuelva delgado para poder enfocar objetos cercanos al ojo.

La distancia entre el centro focal del cristalino y la retina varía aproximadamente entre 17 y 14 mm , conforme el cristalino incrementa su capacidad de refractar la luz de un mínimo a un máximo. Cuando el ojo enfoca a un objeto que está a más de 3 m de distancia, el cristalino exhibe su menor poder refractivo y cuando enfoca objetos muy cercanos exhibe su máxima capacidad de refracción. Con esta información, es tarea sencilla calcular el tamaño de la imagen retinal en el ojo. En la figura IV.5 , por ejemplo, el

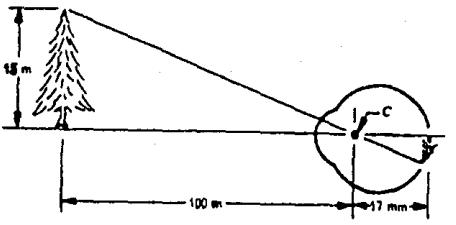


Fig. IV.5 Representación óptica del ojo observando un árbol.
El punto C es el centro óptico del cristalino.

observador está viendo un árbol de 15 m de altura a una distancia de 100 m . Si hacemos que x sea el tamaño de la imagen sobre la retina, en mm , tenemos por la geometría de la figura IV.5 que $15 / 100 = x / 17$ o $x = 2.75$ mm . Como se indicó en la sección previa, la imagen retinal es reflejada inicialmente en el área de la fóvea. La percepción tiene lugar debido a la relativa excitación de los receptores de luz, los cuales transforman la energía radiante en impulsos eléctricos que son finalmente decodificados por el cerebro.

IV.2 Umbral de visión.

El estudio de las características del sistema visual humano es muy importante debido a que en las aplicaciones de la codificación de imágenes el interés fundamental está relacionado con la visibilidad de las degradaciones introducidas por el procesamiento. Las degradaciones no son detectadas siempre y por ello es importante determinar el *umbral de visibilidad* de ciertos estímulos. La psicofísica se encarga de determinar estos umbrales y para ello considera al sistema visual humano como un sistema de entrada-salida en donde el estímulo visual es la excitación y las sensaciones percibidas son la respuesta. El objetivo es entonces determinar la función de transferencia del sistema visual, caracterizada por la capacidad del mismo para discriminar entre diferentes estímulos. La determinación de estos umbrales permite depurar las técnicas de codificación, sobretodo cuando se desea gran calidad en las imágenes que han sido procesadas.

El umbral de visibilidad se define como la magnitud para la cual el estímulo comienza a ser visible. También es importante considerar "cómo se ven las imágenes procesadas" y qué tan grande es el efecto subjetivo de las distorsiones visibles introducidas por el procesamiento. Para ello, es necesario establecer alguna metodología de evaluación.

En esta sección, se describirán brevemente los resultados de algunos experimentos llevados a cabo para evaluar los niveles de visibilidad de algunos estímulos útiles. Posteriormente se hará referencia a algunos de los criterios subjetivos para la evaluación de la calidad de una imagen.

IV.2.1 Umbral de visión en el dominio espacial.

El nivel de visibilidad de un estímulo en particular depende de muchos factores. Considerando únicamente estímulos acromáticos, los factores principales son:

a) Nivel de luminancia promedio de un fondo sobre el cual un estímulo es presentado.

b) Nivel para el cual los cambios de luminancia adyacentes a un estímulo de prueba son visibles.

c) Gradiente espacial de luminancia.

Cada factor se analiza independientemente de los otros, si bien, en realidad, no existe tal independencia entre ellos. En la vida real, en las imágenes complejas el nivel de visibilidad de un estímulo puede ser una función complicada de estos tres factores.

IV.2.1.1 Adaptabilidad y discriminación a la brillantez.

Debido a que las imágenes digitales son desplegadas como un conjunto discreto de puntos brillantes, la habilidad del ojo para discriminar entre diferentes niveles de brillantez, es una consideración importante cuando se observan resultados del procesamiento de imágenes digitales.

El rango de la intensidad de luz de los niveles a los cuales el sistema visual humano puede adaptarse es enorme, siendo del orden de 10^{10} para un umbral escotópico, hasta el límite de destello.

Existe una cantidad considerable de evidencia experimental que indica que la brillantez subjetiva (es decir, la brillantez como la percibe el sistema visual humano) es una función logarítmica de la intensidad de luz incidente sobre el ojo. Esta característica se ilustra en la figura IV.6. En ella se muestra una gráfica de la intensidad de luz contra la brillantez subjetiva. La curva de mayor longitud representa el rango de intensidades para las cuales el sistema visual se puede adaptar. En la visión fotópica el rango de adaptación es de alrededor de 10^6. La transición de la visión escotópica a fotópica es gradual en el rango aproximado de 0.001 - 0.1 mL (de -3 a -1 mL en la escala logarítmica), como se ilustra en las dos ramas de la curva de adaptación en este rango.

El punto clave para interpretar el rango dinámico de excitación mostrado en la figura IV.6 , es que el sistema visual no puede operar sobre este rango simultáneamente, es decir, el sistema opera en subrangos llevando a cabo cambios en toda su sensitividad. A este fenómeno se le conoce con el nombre

de *adaptación a la brillantez*. El rango total de niveles de intensidad que se pueden discriminar simultáneamente es pequeño comparado con el rango de adaptación total. Para cualquier conjunto dado de condiciones, el nivel de sensibilidad actual del sistema visual se denomina nivel brillantez - adaptación, el cual corresponde, por ejemplo a la brillantez B_a en la figura IV.6 . La curva pequeña que intersecta, representa el rango de brillantez subjetiva que el ojo puede percibir cuando se adapta a este nivel.

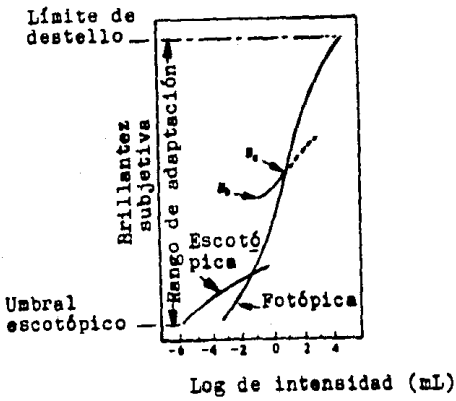


Fig. IV.6 Rango de sensaciones de brillantez subjetiva en donde se muestra un nivel de adaptación.

Se puede notar que el rango está restringido teniendo un nivel B_b abajo del cual todos los estímulos son percibidos como negro. La porción superior punteada de la curva no está acotada pero, si se prolonga demasiado, pierde utilidad porque muchas intensidades mayores podrían simplemente aumentar el nivel de adaptación a un valor mayor que B_a . La sensibilidad del ojo al contraste puede ser medida mostrando a un observador un campo de luz uniforme de brillantez B , en donde en el centro de observación se muestra un blanco circular de brillantez $B + \Delta B$, como se muestra en la figura IV.7(a). Se

incrementa ΔB desde 0 hasta que se alcanza a percibir. La diferencia en el momento en que se observa, es medida como una función de B . La cantidad $\Delta B / B$ se denomina razón de Weber y es aproximadamente constante alrededor del 2% sobre un amplio rango de niveles de brillantez como se muestra en la figura IV.7(b). Este fenómeno ha dado lugar a la idea de que el ojo humano tiene un rango dinámico mucho más amplio que los sistemas de observación de imágenes hechos por el hombre. Sin embargo, esto no corresponde a cualquier observación ordinaria y resultados más útiles se obtienen utilizando el patrón de la figura IV.8(a). La cantidad $\Delta B / B$ es medida, pero la brillantez del fondo (adaptación) es un parámetro. Los resultados se muestran en la

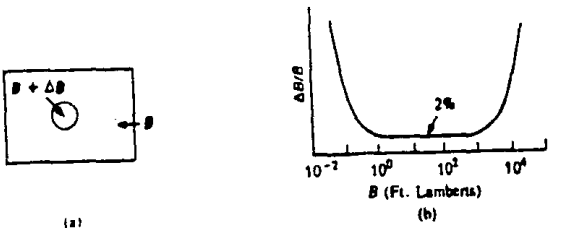


Fig IV.7 Sensibilidad al contraste con un fondo fijo.

figura IV.8(b). El rango dinámico es alrededor de 2.2 unidades logaritmicas entorno al valor de la brillantez de adaptación, el cual es comparable al que puede ser logrado con sistemas electrónicos de imágenes si se ajusta correctamente la brillantez del fondo. La facilidad y rapidez con la cual el ojo se adapta -en formas diferentes en cada parte de la retina- es realmente la caracteristica más importante, aún más que el rango dinámico completo. Lo que significa un rango de 2.2 unidades logaritmicas es que $\Delta B/B$ permanece relativamente constante en este intervalo. Conforme B cambia a partir de la brillantez B_0 , la apariencia también cambia, asi, una brillantez de alrededor de 1.5 unidades logaritmicas arriba o abajo de B_0 , se traduce a la vista del observador como blanco o negro respectivamente. Si el blanco central es puesto a un nivel constante mientras B_0 varia en un

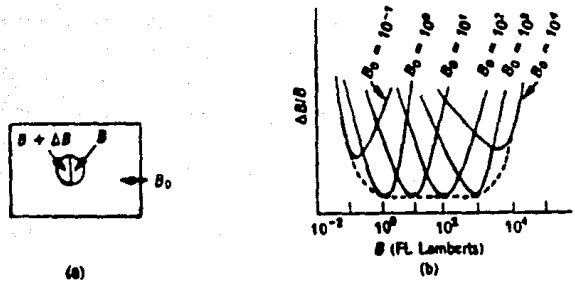


Fig. IV.8 Sensibilidad al contraste con un fondo variable.

amplio rango, el blanco parece cambiar de completamente blanco a completamente negro.

En el caso de una imagen compleja el sistema visual no se adapta a un solo nivel de intensidad. En lugar de ello, se adapta a un nivel promedio que depende de las propiedades de la imagen. Como el ojo vaga alrededor de la escena, el nivel de adaptación instantáneo fluctúa alrededor de este promedio. Para cualquier punto o área pequeña en la imagen, la razón de Weber es generalmente más grande que la obtenida en sus circunstancias experimentales, debido a la pérdida de una frontera bien definida y a las variaciones de intensidad en el fondo.

El resultado es que el ojo puede detectar solamente de una a dos docenas de niveles de intensidad en cualquier punto en una imagen compleja. Esto no significa, sin embargo, que una imagen necesita ser desplegada con dos docenas de niveles de intensidad para lograr resultados visuales satisfactorios. Este pequeño rango de discriminación se modifica conforme se varía el nivel de adaptación de tal forma que se puedan observar diferentes niveles de intensidad conforme el ojo se mueve alrededor de la escena. Esto permite un rango mayor de discriminación de intensidad total. Para desplegar una imagen que parezca adecuada al ojo, para una gran cantidad de imágenes, generalmente se requiere un rango de más de 100 niveles de intensidad.

La brillantez de una región, como es percibida por el ojo, depende de otros factores más que simplemente la radiación luminosa de esta imagen. En términos de aplicaciones de procesamiento de imágenes, uno de los fenómenos más interesantes relacionados a la percepción de la brillantez es que la

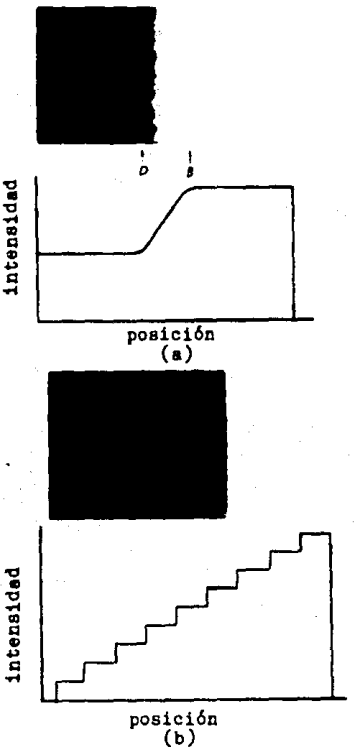


Fig. IV.9 Un ejemplo del efecto de las bandas de Mach.

respuesta del sistema visual humano tiende a "ir más allá" de los límites de las regiones de diferente intensidad. El resultado de ir más allá es hacer que áreas de intensidad constante aparezcan como si hubieran variado de brillantez. En la figura IV.9(a), por ejemplo, la imagen mostrada fue creada variando la intensidad de acuerdo al perfil de intensidad mostrado bajo la fotografía. Aunque la variación de intensidad es perfectamente uniforme, el ojo percibe una franja más brillante en la región marcada B y una franja más obscura en la región marcada con D . Estas franjas son llamadas *bandas de Mach*, en honor a Ernst Mach quien fué el primero en describirlas en 1865. Un ejemplo más sorprendente de las bandas de Mach se muestra en la figura IV.9(b) . Como se indicó para el perfil de intensidad, cada banda en la fotografía fue creada usando una intensidad constante. Para el ojo, sin embargo, el patrón de brillantez en la imagen aparece fuertemente escalonado particularmente alrededor de los límites marcados.

IV.2.1.2 Enmascaramiento visual.

El nivel de visibilidad de un estímulo de prueba cambia cuando es observado en la vecindad de transiciones visibles de luminancia. Es sabido que hay una reducción en la visibilidad del estímulo, es decir, un incremento en el nivel de visibilidad del mismo, debido a un fondo no uniforme. A este fenómeno se le conoce como *enmascaramiento* del estímulo de prueba, debido a un fondo no uniforme. El estímulo de prueba es en general pequeño, y su valor es cercano al umbral de visión del mismo, mientras que el patrón de enmascaramiento está sobre el nivel de visibilidad del estímulo. El *enmascaramiento espacial*, es decir, la reducción de la visibilidad del estímulo de prueba en ambos lados de un cambio brusco en el nivel de luminancia del fondo, ha sido estudiado por algún tiempo. El enmascaramiento visual no debe ser confundido con el *efecto de Mach*. Este efecto se refiere a cambios en la brillantez percibida en un contorno y se manifiesta como un aparente incremento de la brillantez en el lado más brillante y un decremento en el lado más obscuro. Por lo tanto, hay un realce del contraste en las transiciones de un nivel de luminancia a otro.

Algunos de los resultados en el estudio del enmascaramiento visual son los siguientes:

a) El nivel de visibilidad no se eleva significativamente si el estímulo de enmascaramiento se presenta por unos instantes. Esto indica que al menos algún enmascaramiento espacial está relacionado con los movimientos involuntarios del ojo.

b) El efecto de enmascaramiento decrece considerablemente si la imagen permanece en la misma posición sobre la retina.

c) Conforme el estímulo de enmascaramiento es más grande en la vecindad del estímulo de prueba, el nivel de visibilidad del mismo se incrementa.

Como se verá posteriormente, el enmascaramiento espacial ha recibido considerable atención en el campo de la codificación de imágenes. Se ha reconocido que el cuantificador de un sistema predictivo DPCM puede ser diseñado considerando el efecto de enmascaramiento, gracias a que los errores que produce el cuantificador pueden ser enmascarados por las transiciones espaciales en el nivel de luminancia.

IV.2.1.3 Sensibilidad al constraste.

Bajo la consideración de linealidad del sistema visual humano, el nivel de visibilidad de un estímulo de prueba puede ser calculado, si la función de transferencia de la respuesta a un impulso es conocida. Si el estímulo de prueba es una sinusoidal, entonces el nivel de visibilidad puede ser dibujado como una función de la frecuencia espacial. Esta función se denomina *función de sensibilidad al contraste*. La función tiene un valor máximo (un nivel mínimo de contraste) para frecuencias espaciales de 3 a 4.5 ciclos por grado (Fig. IV.10).

Esta respuesta puede ser aproximada adecuadamente utilizando el modelo de la figura IV.11 , donde el estímulo de prueba es filtrado linealmente y si la salida del filtro excede un cierto umbral, el estímulo es considerado visible. El modelo funciona bien para varias clases de estímulos pero falla para otras. Sin embargo, el modelo puede ser extendido para tomar en cuenta algunas otras situaciones. Por ejemplo, un *modelo multicanal*, que utiliza un

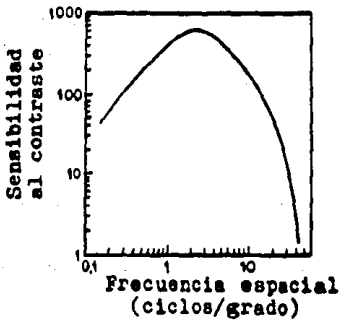


Fig. IV.10 Sensibilidad al contraste para un enrejado sinusoidal con una intensidad de 500 cd/m^2.

banco de filtros lineales, puede predecir las respuestas a estímulos que son una mezcla de varias señales sinusoidales.

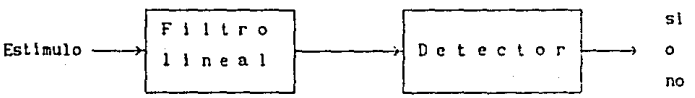


Fig. IV.11 Modelo simple de un sólo canal para simular el umbral de visión del ojo cuando se le presenta un fondo constante.

C A P I T U L O V

APLICACION DE LA TECNICA DE CODIFICACION DPCM A IMAGENES MONOCROMATICAS DE TELEVISION

CAPITULO V. APLICACION DE LA TECNICA DE CODIFICACION DPCM A IMAGENES MONOCROMATICAS DE TELEVISION.

En el capítulo III se abordaron algunos conceptos relacionados con la extracción de redundancia de las imágenes digitalizadas, aprovechando la gran correlación que existe entre los puntos adyacentes que definen una cierta *vecindad local*. Los métodos presentados en ese capítulo constituyen una herramienta en la construcción de sistemas prácticos de codificación. Por ejemplo, si al sistema de codificación predictivo (Fig III.4) se le agrega un bloque de cuantificación, es posible aprovechar al máximo el menor rango dinámico de variación del error de predicción (señal de diferencia). De este modo la asignación de códigos se hace sobre las diferencias cuantificadas. En este contexto, a la razón entre los bits con que se representa la imagen original y los bits que se asignan a las diferencias cuantificadas, se le denomina *razón de compresión*.

En este capítulo se hace la descripción de uno de los métodos de codificación de fuente más utilizados: el DPCM, del inglés *Differential Pulse Code Modulation* (Modulación por Pulsos Codificados Diferencial). En particular se hace referencia a él considerando los requerimientos de codificación de señales provenientes de un sistema de televisión monocromática.

V.1. La compresión de imágenes.

La representación digital de imágenes requiere una gran cantidad de bits (§ Cap. III). En muchas aplicaciones es importante considerar técnicas para representar una imagen, o la información contenida en la imagen, con pocos bits. En la terminología de la teoría de la información esto es denominado *codificación de fuente* o *compresión* de la imagen. En otras palabras, una imagen digitalizada puede ser caracterizada como una secuencia de mensajes. Hay muchas formas de seleccionar los mensajes. El único requerimiento, es poder reconstruir un duplicado de la imagen original a partir de la secuencia

de mensajes. Para obtener esta secuencia, que contiene la información esencial (libre de redundancia) de la imagen original, se aplica una técnica de codificación de fuente. La forma particular de seleccionar los mensajes, define la técnica o método de compresión de la imagen.

Una primera clasificación de los métodos de codificación de fuente puede hacerse considerando si existe pérdida de información durante el proceso de codificación. Es decir, si el método permite reconstruir de manera exacta la imagen original, se le denomina *preservador de información*. En contraparte, si durante el proceso de codificación hay una pérdida de información, el método se denomina *no preservador de información*. La pérdida de información se traduce en una distorsión de la imagen reconstruida.

Una segunda clasificación de los métodos de codificación se puede hacer al considerar el espacio de representación donde se aplica el método. Un método que combina los valores de los puntos de alguna forma apropiada, se denomina *método espacial*. En contraste, un método que utiliza el conjunto de coeficientes de alguna transformada, se denomina *método por transformada*. Finalmente, los métodos que combinan el dominio espacial y el de la transformada se denominan *métodos híbridos*.

A continuación, se anotan algunas de las técnicas de compresión más estudiadas y frecuentemente utilizadas para la codificación de fuente. Se indica también la razón de compresión máxima que se alcanza con cada método.

Métodos espaciales:

1) *Modulación por pulsos codificados, PCM* (Pulse Code Modulation). Puede obtenerse una calidad aceptable en las imágenes con 3 bits/punto. La razón de compresión es por tanto C = 2.6 (considerando que una imagen codificada con PCM y ocho bits por punto tiene una calidad subjetiva excelente). La técnica "Diethering" aplicada a la codificación PCM mejora la calidad subjetiva de la imagen.

2) *Codificación predictiva*. Los métodos predictivos alcanzan una razón de compresión de 4 . Puede ser adaptable si los parámetros se modifican de acuerdo a los datos en una forma adecuada. Por ejemplo, puede ser definida una medida del estado local en cada punto a codificar y los parámetros de la

predicción pueden cambiarse conforme cambia el estado local. De esta forma la razón de compresión puede ser incrementada entre un 10 y 20 % .

Dentro de los métodos predictivos se pueden señalar:

3) *Modulación por pulsos codificados diferencial DPCM* (Differential Pulse Code Modulation). Esta técnica permite alcanzar una razón de compresión de 2.5 . Además, introduciendo un esquema adaptable la razón de compresión se puede incrementar hasta 3.5 .

4) *Modulación delta*. El promedio de la razón de compresión no es muy alto pero la técnica es muy simple.

5) *Codificación interpolativa*. Los interpoladores más frecuentemente utilizados son de orden cero y de primer orden, dando una razón de compresión de 4 . Se pueden utilizar polinomios de orden mayor, pero la complejidad computacional no justifica los resultados.

6) *Codificación "Bit-Plane"*. Con este tipo de codificación puede ser obtenida una razón de compresión de 4 .

Métodos por transformada:

La motivación básica de este tipo de codificación es transformar el conjunto de datos (puntos) de la imagen a otro conjunto "menos correlacionado" o "más independiente" de coeficientes antes de codificarlos. La transformación inversa permite recuperar la imagen original. Las transformaciones más comúnmente utilizadas son las lineales, implantadas con algoritmos rápidos para lograr eficiencia computacional.

1) *La transformada Karhunen-Loève*. Esta transformada es la transformación lineal más eficiente en el sentido de que permite obtener un conjunto de coeficientes descorrelacionados completamente. Sin embargo, es poco utilizada en la práctica debido a su complejidad computacional. Permite conocer un límite teórico superior para evaluar la eficacia de cualquier otra transformación más sencilla y rápida que permita descorrelacionar los puntos de una imagen.

2) *Transformadas rápidas*. Existen muchas transformaciones lineales que pueden ser calculadas a partir de un algoritmo rápido con un número $N \log_2 N$ de operaciones, comparado con N^2 operaciones requeridas para calcular las

transformadas a partir de algoritmos poco eficientes en cuanto a velocidad de procesamiento. Las más importantes son las transformadas rápidas de: Fourier, Hadamard, Haar, seno, coseno y Slant. Una diferencia importante entre estas transformaciones es que no dependen de la estadística de la imagen de entrada.

Existen muchas formas de asignar palabras de código a los coeficientes, que se obtienen al aplicar una transformada a una imagen. En primera instancia la dimensionalidad de la transformada debe ser determinada. Una imagen puede ser procesada mediante una transformada bidimensional o unidimensional, para ser codificada línea por línea (renglón por renglón). El siguiente parámetro es fijar el tamaño de la transformación. Una estrategia comúnmente utilizada es subdividir la imagen en subimágenes de tamaño $M \times M$ siendo M mucho más pequeña que la N de la imagen original (por ejemplo, $M = 32$ y $N = 512$) y transformar entonces cada subimagen por separado. La característica fundamental de estas transformadas es que los coeficientes que se obtienen a partir de ellas se agrupan en una sola área del dominio de transformación. Pueden obtenerse razones de compresión hasta de 10 a 1, dependiendo del número de coeficientes tomados en una cierta área. Otra posibilidad es establecer un umbral a los coeficientes de la transformada de tal forma que sólo se toman aquellos que sobrepasan al umbral y el resto de los coeficientes se iguala a cero. Con ello pueden obtenerse razones de compresión de 15 a 1 , e imágenes reconstruidas con buena calidad.

La codificación por transformada puede ser adaptable haciendo que los parámetros del sistema de codificación se adapten a la estadística de la subimagen que está siendo codificada. La adaptabilidad se puede introducir al obtener la transformada, cuando se lleva a cabo la asignación de bits, o la asignación de niveles de cuantificación. En comparación al caso no adaptable, la eficiencia en la codificación se incrementa entre un 25 y 30 % .

Métodos híbridos:

Es de uso común referirse a la codificación híbrida cuando se combinan la codificación predictiva y la codificación por transformada. La codificación DPCM y la codificación por transformada ofrecen características

atractivas y algunas limitaciones. Combinando estas dos técnicas, podemos obtener una codificación hibrida que mejorará la razón de compresión de 8 a 1, teniendo las ventajas de simplicidad en hardware (DPCM) y robustez (codificación por transformada), con respecto a los errores de transmisión.

Otra clasificación puede hacerse considerando si el método es o no adaptable, es decir, si los parámetros cambian como una función de los datos locales en la imagen, o permanecen fijos.

Las aplicaciones de la codificación de fuente en el campo del procesamiento de imágenes, caen generalmente en una de las tres categorías: (1) almacenamiento de los datos de la imagen, (2) transmisión de la imagen y (3) extracción de características.

Las aplicaciones del almacenamiento de los datos de las imágenes están motivadas por la necesidad de reducir los requerimientos de memoria para almacenar la imagen. Es muy importante emplear por ello, técnicas que permitan la reconstrucción perfecta de los datos codificados. En las aplicaciones de transmisión de imágenes el interés principal es que las técnicas logren una reducción máxima de la cantidad de datos a ser transmitidos, con la restricción de que cierta fidelidad debe ser preservada. Por lo tanto, la técnica de codificación no requiere preservar toda la información, mientras las imágenes resultantes sean aceptables para el análisis visual o de máquina. Las aplicaciones de extracción de características son utilizadas primordialmente para el reconocimiento de patrones mediante un sistema digital de cómputo.

V.2 Criterios objetivos tradicionales de fidelidad.

Existen dos criterios objetivos de fidelidad frecuentemente utilizados. A saber, el valor cuadrático medio (root mean square) entre la imagen de entrada y la imagen de salida y la razón señal a ruido *rms* de la imagen de salida. Supongamos que la imagen de entrada consiste en un arreglo de $N \times N$ puntos $f(x,y)$, donde x, $y = 0, 1, \ldots, N-1$. Como se ha mencionado, cada punto es una palabra binaria de m bits que corresponde a uno de los 2^m posibles valores de los niveles de gris. El codificador reduce el total de

$N \times N \times m$ bits a un número menor. El decodificador procesa estos bits para reconstruir la imagen de salida, que consiste igualmente en un arreglo de $N \times N$ elementos de imagen $g(x,y)$, donde x, $y = 0, 1, \ldots, N-1$, y cada punto es también una palabra binaria que corresponde a uno de los 2^m valores de niveles de gris. Para cada valor de x e y en el rango de 0, 1, ..., $N-1$, el error entre un pixel de entrada y su correspondiente punto de salida es:

$$e(x,y) = g(x,y) - f(x,y)$$

El error cuadrático promediado sobre el arreglo de la imagen es:

$$e^2 = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} e^2(x,y)$$

$$e^2 = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} [g(x,y)-f(x,y)]^2$$

y el error *rms* está definido como:

$$e_{rms} = [e^2]^{1/2} \qquad (V.1)$$

Podemos también considerar la diferencia entre las imágenes de entrada y salida como "ruido", de tal forma que cada señal de salida (punto) consiste en una señal de entrada (que corresponde al punto de entrada), más ruido (el error), esto es:

$$g(x,y) = f(x,y) + e(x,y)$$

La razón señal a ruido cuadrática media de la imagen de salida está definida como el promedio de $g^2(x,y)$ dividida por el promedio de $e^2(x,y)$ sobre el arreglo de la imagen. En otras palabras:

$$(SNR)_{rms} = \frac{\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} g^2(x,y)}{\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} e^2(x,y)}$$

El valor *rms* de (SNR) está entonces dado por:

$$(SNR)_{rms} = \left[\frac{\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} g^2(x,y)}{\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} [g(x,y)-f(x,y)]^2} \right]^{1/2} \qquad (V.2)$$

donde el término variable en el denominador es el ruido expresado en términos de las imágenes de entrada y salida.

Una definición alternativa de la razón señal a ruido, es la raiz cuadrada del valor pico de $g(x,y)$ al cuadrado, entre el valor e_{rms} del ruido, esto es:

$$(SNR)_p = \left\{ [\text{valor pico de } g(x,y)]^2 / e_{rms} \right\}^{1/2} \qquad (V.3)$$

donde e_{rms} está dado por la ecuación (V.1) . El valor pico de $g(x,y)$ es el rango dinámico total de la imagen de salida. De aquí que, $(SNR)_{rms}$ y $(SNR)_p$ difieren entre si por una constante de escalamiento, igual a la razón del máximo nivel de la señal, al nivel de señal promedio.

V.3 El proceso de compresión.

Cada imagen digital $f_i(x,y)$ puede ser expresada en la forma de un vector $\mathbf{x}_i$, de dimensión N^2, de la manera siguiente:

$$\mathbf{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \cdot \\ \cdot \\ \cdot \\ x_{ij} \\ \cdot \\ \cdot \\ \cdot \\ x_{iN^2} \end{bmatrix} \qquad (V.4)$$

donde x_{ij} representa la j-ésima componente del vector $\mathbf{x}_i$. Una forma de construir tal vector es hacer que las primeras N componentes de $\mathbf{x}_i$ estén formadas por el primer renglón de $f_i(x,y)$ (es decir, $x_{i1} = f(0,0)$, $x_{i2} = f(0,1)$, ..., $x_{iN} = f(0,N-1)$); el segundo conjunto de las N componentes, forma el segundo renglón, etc. También es posible utilizar las columnas de $f_i(x,y)$, en lugar de los renglones. Ambas representaciones son válidas y son las más utilizadas.

Para simplificar la notación, denotaremos al vector $\mathbf{x}_i$ simplemente como $\mathbf{x}$, en donde:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \qquad (V.5)$$

de tal forma que $N^2 = n$.

Esta representación se obtiene prácticamente utilizando el método de barrido de líneas del capítulo 1.

El proceso de codificación puede ser modelado como una secuencia de operaciones que transforman al conjunto de elementos de la imagen, expresados en la forma de la ecuación (V.5), en otro conjunto de elementos, tal y como se muestra en la figura V.1 . La primera operación transforma los datos de entrada del dominio de los puntos a otro dominio,en el cual el bloque de *cuantificación* y el bloque de *asignación de códigos* pueden ser utilizados más eficientemente. El cuantificador redondea cada dato de la salida del bloque de *transformación*, de tal forma que el nuevo conjunto requiere un número menor de palabras de código (cada una formada por pocos bits) para ser representado. El bloque de asignación de códigos determina qué palabra de código corresponde a cada salida del bloque de cuantificación.

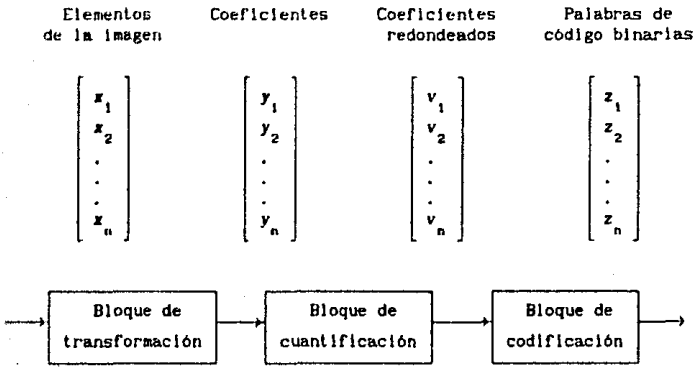
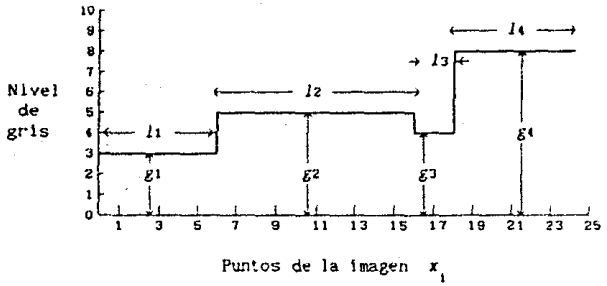


Fig. V.1 Un modelo de un sistema de codificación.

V.3.1 El bloque de transformación.

Este bloque tiene la función de transformar al conjunto de números de entrada en otro conjunto de números. El procedimiento básico se puede explicar mejor mediante algunos ejemplos.

En un sistema de codificación *run length* (longitud de corrida), la secuencia de elementos de una imagen a lo largo de la línea (renglón) de barrido $x_1, x_2, \ldots, x_N$ es transformada en una secuencia de pares (g_1, l_1), (g_2, l_2), ... , (g_k, l_k) , donde g_i representa el nivel de gris y l_i la longitud de la i-ésima corrida, como se ilustra en la figura V.2 . Por ejemplo, en las imágenes meteorológicas, se requiere una cantidad significativamente más pequeña para representar la secuencia de longitud de corrida la imagen, que para la secuencia original. Esta transformación es reversible porque la secuencia de elementos de la imagen puede ser reconstruida de la secuencia de corridas.



i	gi, li
1	3 ; 6
2	5 ;10
3	4 ; 2
4	8 ; 6

Fig. V.2 Ejemplo de una transformación por longitud de corrida.

Otro tipo de transformación útil en el proceso de codificación es la lineal:

$$\begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \cdot & & \cdot \\ & \cdot & & \cdot \\ & \cdot & & \cdot \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \qquad (V.6)$$

o

$$y = A\,x \qquad (V.7)$$

Esta transformación puede, o no ser reversible, dependiendo de la elección de A . En este caso el vector de elementos de imagen $\mathbf{x}$, se transforma en un vector de coeficientes $\mathbf{y}$. Para un conjunto de vectores $\mathbf{x}$ y alguna transformación A , se requieren menos bits para codificar los n coeficientes de $\mathbf{y}$ que los n puntos de $\mathbf{x}$. En particular, si los elementos $x_1, x_2, \ldots, x_n$ están muy correlacionados y la matriz de transformación A se elige de tal forma que los coeficientes $y_1, y_2, \ldots, y_n$ estén menos correlacionados, entonces las y_i pueden ser codificadas individualmente de manera más eficiente que las x_i .

La transformación de diferencia (*difference mapping*) se obtiene si se utiliza la matriz:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \qquad (V.8)$$

en la ecuación (V.7) . El primer elemento de $\mathbf{y}$ es $y_1 = x_1$. Sin embargo, todos los coeficientes subsecuentes están dados por $y_i = x_{i-1} - x_i$. Si los niveles de gris de los puntos adyacentes son similares, entonces las diferencias $y_i = x_{i-1} - x_i$, en promedio serán más pequeñas que los niveles de gris, por lo que se requieren menos bits para representarlos. La transformación también es reversible.

Los ejemplos anteriores son procedimientos típicos de transformación utilizados en algunos sistemas de codificación de imágenes.

V.3.2 El bloque de cuantificación.

El bloque de cuantificación tiene como función proporcionar una salida con un número limitado de valores. Dependiendo del rango al que pertenezca, a cada entrada le es asignado uno de los valores permitidos de salida. Existen varios criterios para clasificar a los cuantificadores. Tres de ellos son:

1. *Cuantificación de memoria cero o escalar.*
2. *Cuantificación por bloques.*
3. *Cuantificación secuencial.*

En la *cuantificación de memoria cero* la cuantificación se lleva a cabo sobre una muestra a la vez. En la *cuantificación por bloques*, la representación de un conjunto de muestras de entrada se lleva a cabo mediante un bloque de salida elegido a partir de un conjunto discreto de posibles bloques de salida, de tal modo que la distancia entre los bloques de entrada y salida es mínima. En la *cuantificación secuencial*, se cuantifica una secuencia de muestras utilizando la información de las muestras vecinas, sobre la base de la utilización o no utilización de bloques.

En lo que resta de esta sección se tratará solamente la cuantificación de memoria cero.

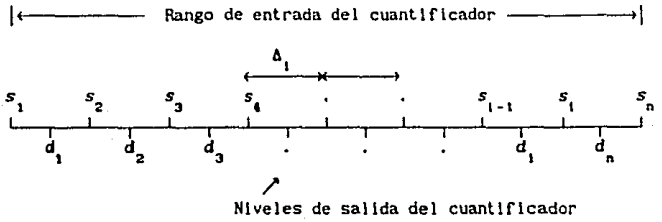
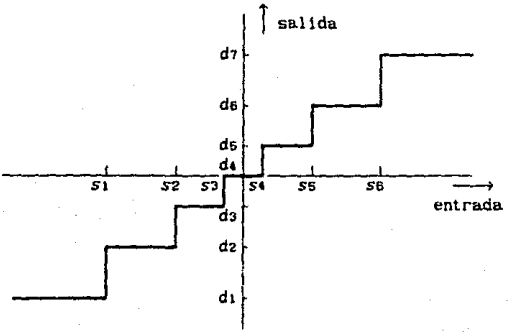


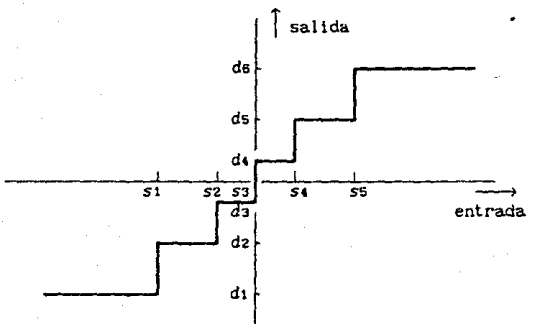
Fig. V.3 Relaciones de entrada-salida para un cuantificador.

En la cuantificación de memoria cero, a cada entrada le es asignada uno de los valores permitidos de salida. Una forma de llevar a cabo esta operación es dividiendo el rango de entrada en un número de niveles de decisión, s_i , que definen intervalos de decisión Δ_i , como se muestra en la figura V.3 . De este modo si la entrada cae en el i-ésimo intervalo, Δ_i, entonces se obtiene la salida correspondiente al i-ésimo nivel de reconstrucción *di* . Es posible hacer que el valor de reconstrucción, *di* , corresponda

al valor central del i-ésimo intervalo de decisión Δ_i , de tal forma que cada entrada se redondea respecto al centro del intervalo de decisión en que cae.



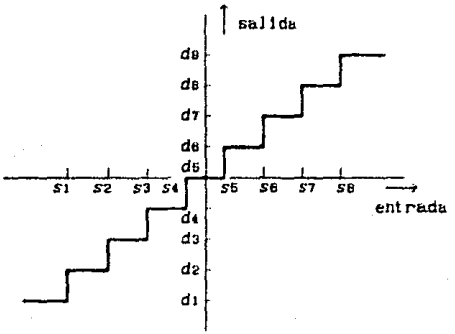
(a)



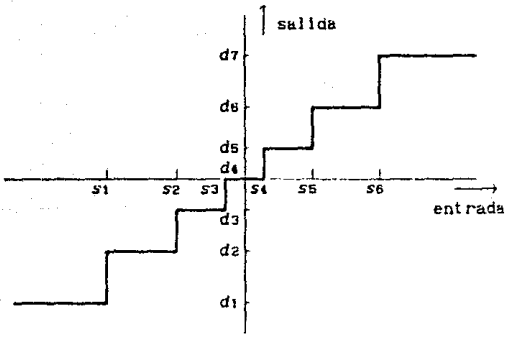
(b)

Fig. V.4 Caracteristicas de los cuantificadores fijos.

De acuerdo a sus caracteristicas de entrada y salida, los cuantificadores pueden ser del tipo *"midtread"* (Fig. 4.5 (a)),en los que una entrada en el intervalo cero produce una salida cero y los del tipo *"midriser"* (Fig. V.4(b)) en los que existen dos intervalos de entrada alrededor del



(a)



(a)

Fig. V.5 Cuantificadores uniforme y no uniforme.

cero. El intervalo positivo produce un nivel positivo y el intervalo negativo produce un nivel negativo a la salida.

De acuerdo a las caracteristicas de los intervalos de decisión, los cuantificadores pueden ser *uniformes* (Fig. V.5(a)), en los que todos los niveles de decisión están equidistantes los unos de los otros (intervalos de decisión del mismo tamaño) y cuantificadores *no uniformes* (Fig. V.5(b)) en los que los intervalos de decisión tienen tamaños diferentes.

La operación de cualquier cuantificador no es reversible puesto que para un cierto valor de salida, no es posible determinar el valor de entrada. Haciendo que y represente cualquier valor del vector y (de entrada al bloque de cuantificación en la figura V.1) y v_i la correspondiente salida del cuantificador. Entonces, el error de cuantificación es la diferencia entre la salida y la entrada del cuantificador (Fig. V.6), esto es:

$$e_q = v - y \qquad (V.9)$$

Es claro que el error mínimo es cero cuando la entrada es igual a alguno de los valores permitidos de la salida. Por lo tanto si Δ es igual a la diferencia entre dos niveles de decisión de un cuantificador uniforme, entonces $e_{qmax} = \Delta / 2$, es el error máximo.

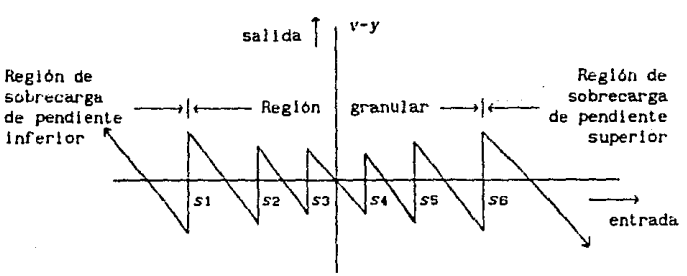


Fig. V.6 Error de cuantificación para el cuantificador de la figura V.4(a).

El error *rms* es la raiz cuadrada del error cuadrático medio promediado sobre todos los posibles valores de la entrada **y** . Si la entrada y tiene la misma probabilidad de ser cualquier valor dentro del intervalo que definen dos niveles de decisión, entonces el error cuadrático medio es:

$$e_q^2 = \int_{v-\frac{\Delta}{2}}^{v+\frac{\Delta}{2}} (v - y)^2 \, dy \qquad (V.10)$$

Si todos los intervalos tienen el mismo ancho Δ y para cada uno, los valores de entrada tienen la misma probabilidad de ser cualquier valor dentro del intervalo, entonces el error e_{qrms} es el mismo para todos los intervalos, por lo que para todos ellos, el error de cuantificación *rms* está dado por la raiz cuadrada de la ecuación (V.10), aún para entradas y que tienen mayor probabilidad de caer en unos intervalos que en otros.

Si el valor *rms* de la señal de entrada es:

$$y_{rms} = \sqrt{\int y^2 \, dy} \qquad (V.11)$$

la relación señal a ruido de cuantificación está dada por:

$$Q_{SNR} = y_{rms} / e_{q\ rms} \qquad (V.12)$$

Si todos los valores de y dentro de los intervalos de decisión no son igualmente probables, entonces, el error cuadrático $(v - y)^2$ debe ser ponderado por la función de densidad de probabilidad $p(y)$:

$$e_q^2 = \int_{v-\frac{\Delta}{2}}^{v+\frac{\Delta}{2}} (v - y)^2 \, p(y) \, dy \qquad (V.13)$$

y el error de cuantificación es un promedio ponderado de todos estos términos. En otras palabras, el término de error para cada intervalo de decisión debe ser ponderado por la probabilidad:

$$\int_{v-\frac{\Delta}{2}}^{v+\frac{\Delta}{2}} p(y)\, dy \qquad (V.14)$$

para la y que cayó en este intervalo.

En el procesamiento de imágenes digitales, el diseño de cuantificadores de memoria cero puede hacerse desde dos puntos de vista complementarios. Por un lado los cuantificadores son especificados utilizando los resultados de pruebas de caracter subjetivo. Por otro, en el marco de la teoria de la información, se especifica un cuantificador tomando como límite teórico mínimo, la función de razón de distorsión para cierto modelo de fuente.

V.3.3 El bloque de asignación de códigos.

Las entradas al bloque de asignación de códigos son n elementos del vector v de la figura V.1 .Suponiendo que cada elemento v_i tiene un rango dinámico de M valores (w_1, w_2, ..., w_M). Para cada entrada v_i , la salida del bloque de asignación de códigos es una palabra binaria cuyo valor depende del valor w_k de la entrada. La relación entrada-salida del bloque de asignación de códigos es uno a uno, en el sentido de que una sola palabra de código c_k es asignada a cada posible valor de w_k .

De aquí, el proceso es reversible porque para una cierta palabra de código c_k , se conoce w_k . El bloque de asignación de código no introduce ningún error en el proceso de codificación.

Un código de longitud de palabra constante es un conjunto de palabras de código que tienen el mismo número de bits, que mediante alguna regla, se asignan a cada nivel de salida del cuantificador. Una posible regla de asignación para el código natural, es ordenar las palabras de código de acuerdo a su valor binario. Por ejemplo, supongamos que hay ocho posibles valores de entrada al bloque de asignación de códigos (niveles de salida del cuantificador) w_1, w_2, ..., w_8 ; entonces el código natural es c_1 = 000, c_2 = 001, c_3 = 010, ..., c_8 = 111, como se ilustra en la tabla V.1 . El *código binario reflejado* o *código Gray*, también se ilustra en la tabla V.1 y tiene

la propiedad de que dos palabras de código adyacentes en el conjunto, se diferencian solamente en un bit.

Tabla V.1 Algunos códigos típicos.

Entrada	Natural	Gray	Código B_1	B_2	S_2
w_1	000	111	C0	C00	00
w_2	001	110	C1	C01	01
w_3	010	100	C0C0	C10	10
w_4	011	101	C0C1	C11	1100
w_5	100	001	C1C0	C00C00	1101
w_6	101	000	C1C1	C00C01	1110
w_7	110	010	C0C0C0	C00C10	111100
w_8	111	011	C0C0C1	C00C11	111101

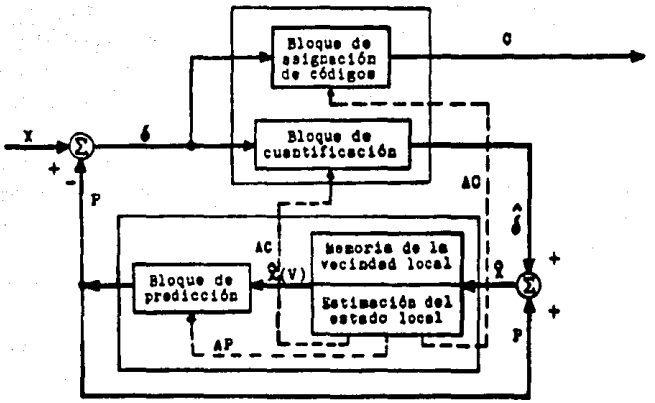
Un código *univocamente decodificable* es aquel que tiene la propiedad de que la secuencia de palabras de código sólo se puede hacer de una sola forma. El código $c_1 = 0$, $c_2 = 1$, $c_3 = 01$, $c_4 = 10$ no es único. Porque la secuencia de bits 0011 puede ser decodificada como $c_1c_1c_2c_2$ o $c_1c_3c_2$. Todos los códigos que se presentan en la tabla V.1 son univocamente decodificables.

Un *código instantáneo* es aquel que puede ser decodificado instantáneamente. Esto es, si se observa la secuencia de los bits uno a la vez, se conocerá el valor de la entrada una vez que llega el último bit de la misma. Por ello, no es necesario conocer alguna característica de los bits que llegarán para decodificar la secuencia de bits actual. Todos los códigos en la tabla V.1 son instantáneos excepto los códigos B . Estos requieren que la secuencia actual de bits a decodificar, contenga información respecto a la cadena de bits que llegará después.

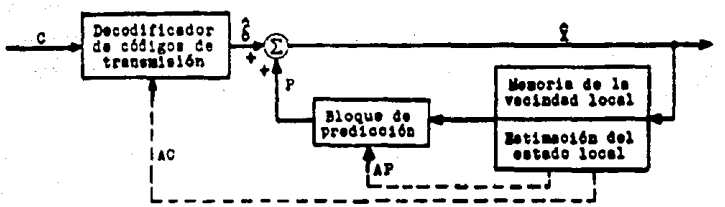
V.4 Fundamentos de la codificación DPCM (Modulación por pulsos codificados diferencial).

La codificación DPCM es una de las primeras estructuras de codificación predictiva que se haya propuesto (Fig V.7). En ella, la señal analógica a procesar es muestreada inicialmente a una frecuencia ligeramente superior a la frecuencia de Nyquist. En el esquema más simple de predicción, se utiliza el valor horizontal previamente decodificado. Esto equivale a cuantificar una aproximación del gradiente horizontal de la señal. Otros predictores más elaborados pueden alcanzar mayor eficiencia haciendo un mejor uso de la correlación que existe en la señal, utilizando una mayor cantidad de elementos de la imagen en el campo actual (incluyendo la línea actual y las precedentes) así como el campo o el marco precedente. Estos predictores se denominan *intercampo* o *intermarco* respectivamente y los sistemas de codificación de fuente que los utilizan llevan el mismo nombre. Los sistemas intermarco requieren una cantidad suficiente de memoria para almacenar un campo o un marco completo y son, en general, mucho más complejos que los sistemas intercampo.

En las secciones siguientes se presentarán los cambios que, pregresivamente, se han introducido para optimizar los sistemas de codificacción DPCM. Los cambios conciernen a las estructuras que constituyen el sistema al introducirles parámetros adaptables. La primera estructura del sistema, el predictor, apoya su adaptabilidad en las propiedades no estacionarias de la fuente. El cuantificador, que es la segunda estructura, es diseñado tomando en cuenta las propiedades visuales del receptor (humano) para lograr que sus parámetros se adapten a ellas. La adaptabilidad puede ser introducida en el predictor, en el cuantificador o en ambos. En lo que sigue solamente se hablará de sistemas intercampo (una imagen de televisión tiene dos campos entrelazados espacialmente que son presentados secuencialmente uno después de otro (§ I.5)) y la señal a codificar es la intensidad de luz o luminancia utilizada en los sistemas monocromáticos de televisión.



(a)



(b)

Fig. V.7 Sistema de codificación DPCM.

V.4.1 Notación general para el sistema DPCM intercampo.

A continuación se presenta la notación utilizada frecuentemente en un sistema de codificación DPCM intercampo: en el sistema, $X_{m,n}$ es el punto actual a codificar, $P_{m,n}$ la predicción determinada para este punto, $\delta_{m,n}$ el error de predicción, $\hat{\delta}_{m,n}$ el error de predicción cuantizado y $\hat{X}_{m,n}$ el valor decodificado del punto actual. Para simplificar la notación suprimiremos los subíndices, con lo que los símbolos son: X, P, δ, $\hat{\delta}$ y $\hat{X}$ (Fig. V.7).

Una de las propiedades fundamentales dentro de la codificación DPCM es su recursividad: la predicción es determinada de una manera causal, es decir, a partir de los puntos previamente codificados y no a partir de los puntos de la imagen original. Los sistemas DPCM adaptables son más eficaces que aquellos que utilizan parámetros fijos, sin embargo, respetan esta condición de causalidad. La adaptabilidad del predictor y del cuantificador se apoya entonces en algoritmos que usan exclusivamente los puntos previamente decodificados. Por lo tanto, los parámetros de adaptación no son transmitidos; el codificador y el decodificador, funcionan paralelamente y producen las mismas señales P y $\hat{X}$, siempre y cuando la transmisión esté libre de errores ($C' = C$ entre las figuras V.7(a) y V.7(b)). Hay que señalar que la adaptabilidad modifica sensiblemente el comportamiento del sistema bajo condiciones de ruido.

La causalidad de la codificación DPCM nos conduce a definir una vecindad local causal del punto actual $X_{m,n}$, que denotaremos como $V_{m,n}$ o simplemente V. Esta será un conjunto de puntos previamente decodificados $\hat{X}$, es decir:

$$V_{m,n} \subset \{ \hat{X}_{m',n'} , (m' < n \text{ y } n' = n) \text{ o } (n' < n \ \forall \ m') \} \qquad (V.15)$$

Los $\hat{X}_{m,n}$ más próximos al punto actual $X_{m,n}$, son denominados clásicamente como A, B, C, ... ; la figura V.8 es un ejemplo de una vecindad V. La predicción P del punto actual X será una combinación lineal, con coeficientes eventualmente variables de algunos puntos $\hat{X}$ de V.

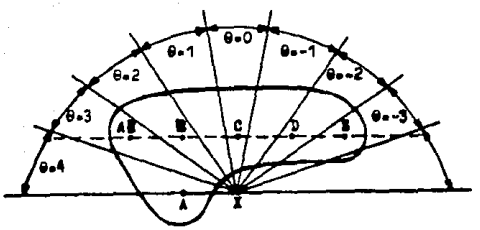


Fig. V.8 Vecindad del punto X .

La adaptabilidad se introduce generalmente mediante la determinación de un *estado local* del punto actual, a partir de la observación de valores previamente decodificados. Este estado local puede ser determinado a partir de *observaciones locales* en toda, o en parte de la vecindad, V . El estado local, de igual forma, puede ser determinado a partir de observaciones recursivas. Este estado local, E , podrá servir como criterio de conmutación del predictor adaptable (parámetros de adaptación AP) y como criterio de conmutación para el cuantificador adaptable (parámetros AQ). Las ecuaciones que rigen al sistema de la figura V.7 son las siguientes:

- predicción (adaptable):

$$P_{m,n} = \sum_{v \in V_{m,n}} a_v^E \hat{X}_v \quad ; \tag{V.16}$$

- diferenciación (error de predicción):

$$\delta_{m,n} = X_{m,n} - P_{m,n} \tag{V.17}$$

- cuantificación (adaptable):

$$\hat{\delta}_{m,n} = Q^E(\delta_{m,n}) \tag{V.18}$$

- decodificación:

$$\hat{X}_{m,n} = P_{m,n} + \hat{\delta}_{m,n} \quad . \tag{V.19}$$

Un cuantificador fijo Q de N niveles, se define por su conjunto de niveles de decisión $\{ s_i , i = 0 \text{ a } N \}$ y su conjunto de niveles de reconstrucción $\{ d_i , i = 0 \text{ a } N \}$. La dinámica de la señal a cuantificar (aquí $\delta_{m,n}$), [A,B], permite establecer $s_0 = A$ y $s_N = B$ (en nuestro caso, A = -255, B = +255) ; en efecto, la señal de entrada se representa con 8 bits y la dinámica de $\hat{X}$, X y P es [0,255]. La cuantificación fija se define entonces como:

$$\hat{\delta}_{m,n} = Q(\delta_{m,n}) = d_i \qquad \text{si} \quad s_i - 1 < \delta_{m,n} \leq s_i \qquad (V.20)$$

A cada d_i se le asocia un código de transmisión C_i ($i = 1$ a N). Este codificador (Fig. V.7(a)), efectúa la operación siguiente:

$$\text{si} \qquad s_{i-1} < \delta_{m,n} \leq s_i \text{ , se transmite el código } C_i \qquad (V.21)$$

El decodificador de transmisión (Fig. V.7(b)), efectúa la operación complementaria:

si, para el punto (m,n), se recibe el código C_i , entonces:

$$\hat{\delta}_{m,n} = d_i \qquad (V.22)$$

En el caso de un cuantificador adaptable, se denotarán los niveles de decisión y los niveles de reconstrucción, como: s_i^E y d_i^E respectivamente.

Debe remarcarse que el estado local, E , que determina los parámetros de adaptación AP y AQ , se estima a partir del conjunto (o una parte de él) de los valores previamente decodificados; por lo tanto, no se transmite ninguna información, respecto a los parámetros de adaptabilidad.

V.4.2 Principio de funcionamiento del sistema DPCM.

Si bien la codificación predictiva se mencionó con anterioridad, es conveniente hacer una descripción breve de la forma en que opera este sistema cuando se introduce el cuantificador.

Un diagrama de bloques simplificado de la figura V.7 , se muestra en la figura V.9 ; en este los parámetros permanecen fijos. Constituye el sistema

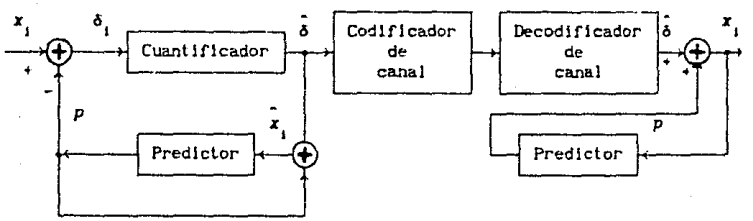


Fig. V.9 Sistema de codificación DPCM simplificado.

DPCM más simple y se utilizará para describir su principio de funcionamiento. El predictor en el bloque de codificación y en el bloque de decodificación son idénticos. Ambos, simplemente retrasan la entrada un tiempo igual al inverso de la tasa de muestreo y escalan esta salida retrasada por una constante α . Este coeficiente se denomina *coeficiente de predicción*.

Como ya se ha señalado, los valores de los elementos de las imágenes están altamente correlacionados, tanto espacial como temporalmente. En la figura V.10(a) se muestra la función de autocorrelación para la imagen "*bateau et phare*". Si el punto x_{i-1} tiene un cierto nivel de gris, entonces el punto adyacente x_i a lo largo de la línea de barrido tiene una gran probabilidad de tener un valor similar. Esto se muestra en el histograma de diferencias de puntos $x_i - x_{i-1}$ de la figura V.10 (b). Mientras que los valores de los pixeles tienen un rango dinámico de 256 niveles de gris diferentes, muchas diferencias de pixeles adyacentes tienen un rango de alrededor de 20 niveles de gris. La técnica de codificación DPCM, aprovecha esta propiedad de las imágenes de la forma siguiente: Se observa el punto x_{i-1} y con base en él, se predice el valor del siguiente punto x_i . Siendo p_i la predicción del valor x_i , se resta este valor al pixel actual x_i .

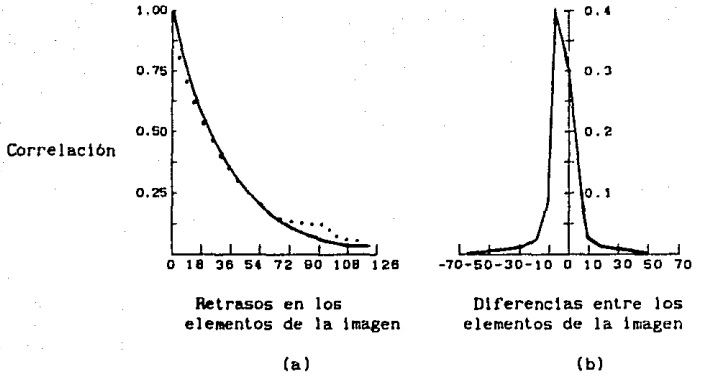


Fig. V.10 Función de autocorrelación (a) e histograma de diferencias (b) para una porción de la imagen "*bateau et phare*".

para obtener la diferencia:

$$\delta_i = x_i - p_i \qquad (V.23)$$

Si se asume que la predicción es buena, entonces la diferencia δ, será en promedio, significativamente más pequeña que la magnitud del punto x_i. Consecuentemente, se requiere un número menor de niveles de cuantificación para codificar la secuencia de diferencias, con respecto a la que se requeriría para codificar la secuencia de elementos de la imagen original.

El problema es entonces, estimar x_i conociendo x_{i-1}. El estimador lineal que resulta de una estimación media cuadrática que minimiza el error, $E\{(x_i - p_i)\}$; está dado por:

$$p_i = \rho \cdot x_{i-1} + (1 - \rho) \cdot m \qquad (V.24)$$

donde m es el nivel de gris promedio y ρ es la correlación normalizada entre los puntos adyacentes, esto es:

$$\rho = \frac{E\{(x_i x_{i-1})\}}{E\{x_i^2\}} \qquad (V.25)$$

La predicción p_i en la ecuación V.24 puede interpretarse como el promedio ponderado del punto precedente x_{i-1} y la media de x_i, es el valor óptimo para el coeficiente de predicción. Los pesos dependen del coeficiente de correlación ρ. Cuando los valores de los puntos están muy correlacionados, ρ se aproxima a 1 y $1 - \rho$, se aproxima a 0, con lo que la predicción es x_{i-1}. Cuando los pixeles no están muy correlacionados, sucede la situación contraria y la estimación se toma con respecto a la media. Para imágenes apropiadamente muestreadas, ρ toma valores típicos entre 0.85 y 0.95.

Se puede mostrar fácilmente que la varianza de la diferencia δ_i está dada por:

$$\sigma_{\delta i}^2 = (1 - \rho^2)\,\sigma_{xi}^2 \qquad (V.26)$$

donde σ_{xi}^2 es la varianza de x_i. También es posible mostrar que las diferencias no están correlacionadas, esto es, la transformación de x_i a $\hat{\delta}_i$ produce coeficientes descorrelacionados. Nótese que, si $\rho = 1$, esta transformación es idéntica a la transformación de diferencia definida en la ecuación V.8. Dicho de otro modo, la transformación de diferencia es equivalente a decir que p_i es igual a x_{i-1}.

Las etapas restantes consisten en cuantificar y codificar las diferencias δ_i. Sin el cuantificador, las predicciones generadas por el bloque de codificación utilizan las diferencias exactas δ_i. Con el cuantificador, las predicciones generadas por el bloque de codificación, utilizan diferencias cuantificadas $\hat{\delta}$. Esto provoca degradaciones irreversibles.

V.4.3 Los predictores de un sistema DPCM.

Los predictores para una codificación DPCM pueden clasificarse como funciones *lineales* o *no lineales* de los puntos previamente transmitidos. Se

puede hacer una división posterior dependiendo de la posición de los puntos utilizados para hacer la predicción: los predictores *unidimensionales*, los cuales utilizan puntos de la línea a la que pertenece el punto a predecir. Los predictores *bidimensionales*, que utilizan puntos en la(s) línea(s) previas. Los predictores intermarco utilizan puntos en los marcos o campos previamente transmitidos, explotando de esta manera la redundancia temporal.

V.4.3.1 Predictores lineales.

Si $\{b_1, b_2, \ldots, b_N\}$ es un bloque de puntos cuya media es cero, de los cuales $b_1, \ldots, b_{N-1}$ han sido previamente decodificados (y por lo tanto pueden ser utilizados para predecir b_N), entonces un predictor lineal para b_N puede escribirse como:

$$\hat{b} = \sum_{i=1}^{N-1} \alpha_i b_{N-1} \qquad (V.27)$$

Los coeficientes $\{\alpha_i\}$ pueden ser obtenidos minimizando el error cuadrático medio de predicción (*MSPE*), $E\,(b_N - \hat{b}_N)^2$, tal y como se señaló en la sección III.3 . De este modo los coeficientes óptimos están dados por:

$$(MSPE)\ \text{óptimo} = \sigma^2 - \sum_{i=1}^{N-1} \alpha_i \cdot d_i \qquad (V.28)$$

donde $d_i = E\,(b_N \cdot b_{N-1})$ y los puntos se consideran igualmente distribuidos con media cero y varianza σ^2 . En las consideraciones anteriores, se asume que la estadística es estacionaria e idéntica para todos los puntos de la imagen. Si bien la teoría de la predicción lineal puede tratar señales que tienen una amplia variedad de estadísticas estacionarias, su aplicación a la codificación de imágenes no ha sido óptima. Las principales razones son que no existen modelos lo suficientemente aproximados para describir a la señal de las imágenes. Y segundo, si bien se minimiza el error cuadrático medio de predicción, no se minimiza respecto a la calidad subjetiva de la imagen. Se desprecia además la presencia del cuantificador en el sistema.

El error cuadrático medio de predicción, puede ser determinado experimentalmente para predictores lineales que utilizan diferentes elementos en la vecindad local.

En la práctica, los predictores bidimensionales son frecuentemente utilizados. Si bien las mejoras en la entropía del error de predicción no son sustanciales, la utilización de predictores bidimensionales disminuye considerablemente el error pico de predicción. Más aún, las evaluaciones subjetivas de las imágenes cuantificadas indican que la eficacia de los predictores en los contornos se incrementa considerablemente. Mediante la elección adecuada de los coeficientes, es posible optimizar los predictores para que la degradación provocada por un error de predicción decaiga rápidamente (§ V.4.4). En general, debido a que la correlación es alta y la media de la imagen no varía dramáticamente en pocas muestras la suma de los coeficientes $\{\alpha_i\}$ es usualmente cercana a uno.

V.4.4 Propiedades de la codificación DPCM con parámetros fijos.

La codificación DPCM con parámetros fijos, presenta tres degradaciones clásicas (ver Fig. V.11) , introducidas por la transformación no lineal del bloque de cuantificación: el *ruido granular*, la *flotación de contorno* y la *sobrecarga de pendiente*. Existe otra degradación debida a los errores introducidos por el canal en la transmisión.

El ruido granular aparece cuando la variación de la señal de entrada es pequeña con respecto al espaciamiento entre los niveles de decisión, s_i , del cuantificador. Se presenta principalmente en áreas de la imagen donde la intensidad es prácticamente constante. La sobrecarga de pendiente ocurre siempre que el espaciamiento entre los niveles de decisión del cuantificador es demasiado pequeño para valores locales de δ : truncamiento del error de predicción. Este fenómeno aparece en zonas donde existen contornos muy contrastados, interviniendo como un error fijo $+\Delta$ o $-\Delta$, al inicio o al final de la transición respectivamente. La flotación de contorno aparece en todo punto donde el contraste local es muy elevado, o el error de predicción está cerca de uno de los niveles de decisión del cuantificador: es un ruido

esencialmente bimodal ±Δ localizado en la transición de subida o de bajada del contorno respectivamente.

La degradación debida a los errores de transmisión se traduce en un código recibido C' diferente del código emitido C (Fig. V.7). La señal $\hat{\delta}$, es reemplazada entonces por la señal $\hat{\delta}'$ en el decodificador de transmisión. El efecto de un error de transmisión en el punto actual es un error de amplitud $\Delta = \hat{\delta}' - \hat{\delta}$; la predicción del punto actual no se ve afectada. Sin embargo, el error se propagará a los puntos siguientes.

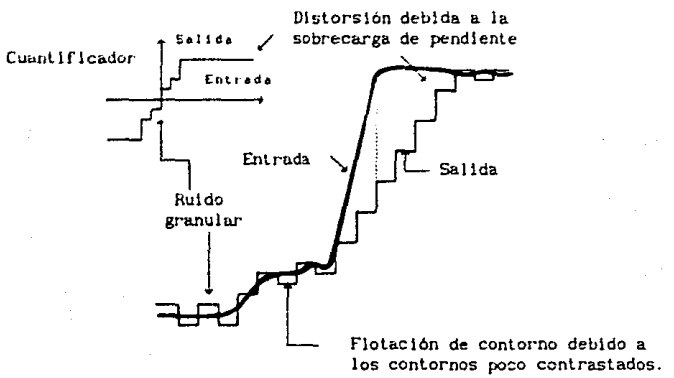


Fig. V.11 Degradaciones clásicas en un sistema de codificación DPCM con parametros fijos.

V.4.4.1 Efectos de los errores de transmisión.

En un sistema de codificación de fuente DPCM, un error en la transmisión en el punto actual puede afectar uno o más de los puntos siguientes de la imagen. La forma en que se propaga depende de la definición funcional del predictor utilizado. En el caso de predictores lineales unidimensionales con

media cero, es sabido que para los diversos tipos de correlación, propios de las imágenes, los coeficientes del predictor provocan que el decaimiento de la propagación sea tan rápido como se desee.

Para los predictores bidimensionales, la situación es más compleja debido a que se presenta inestabilidad y aún no existe un criterio de estabilidad adecuado.

En el caso de los predictores adaptables, la dinámica del error de predicción es muy variable y es analizada en la mayoría de los casos por la vía experimental.

V.4.5 Propiedades de la fuente (imagen) concernientes a la adaptabilidad del predictor.

Las propiedades de las imágenes que nos conciernen en este trabajo, son aquellas que permiten mejorar la eficacia de una codificación de tipo DPCM. Estas se enumeran a continuación:

- los contornos son rectilíneos,
- los contornos horizontales y verticales son los mas probables y
- el histograma del error de predicción (para la predicción A : δ es el gradiente horizontal), indica que las zonas correspondientes a los puntos no contorno son más angostas que el histograma acumulado sobre toda la figura (Fig. V.12)

Estas propiedades implican que si los contornos pueden ser predichos correctamente, la dinamica útil del error de prediccion puede ser reducida sensiblemente. Por lo tanto, el estudio de la prediccion adaptable se relaciona directamente con la prediccion de contornos (es decir, su orientación). De la misma forma, una predicción adaptable con pruebas locales en las zonas de no contorno, podría mejorar la prediccion en las zonas de textura.

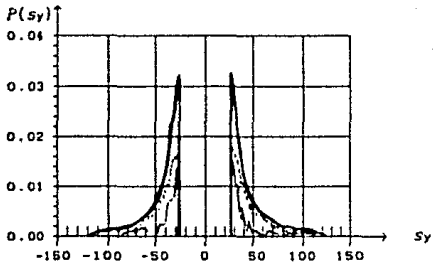


Fig. V.12 Histograma del gradiente horizontal acumulado (error de predicción A).

V.4.5.1 Los contornos en el dominio de la frecuencia.

Existe una relación muy estrecha entre la presencia de un contorno espacial en una imagen y su contribución al espectro de la misma. La interpretación intuitiva puede encontrarse en los textos como sigue: un contorno introduce en el espectro, frecuencias espaciales altas a lo largo de la dirección que es ortogonal a la dirección del mismo. En general, las altas frecuencias corresponden a contornos, mientras que las bajas frecuencias, corresponden a regiones uniformes. Y aunque esta propiedad no se aprovecha en el algoritmo que aquí se presenta, se ha utilizado ya para determinar la dirección de los contornos en un esquema de codificación DPCM : el contenido espectral puede indicar en forma económica y precisa si un contorno se encuentra o no presente en una cierta dirección [5].

C A P I T U L O V I

ANALISIS COMPARATIVO DE DIVERSOS ALGORITMOS DE CODIFICACION PCM DIFERENCIAL CON PARAMETROS FIJOS Y ADAPTABLES

CAPITULO VI. ANALISIS COMPARATIVO DE DIVERSOS ALGORITMOS DE CODIFICACION PCM DIFERENCIAL CON PARAMETROS FIJOS Y ADAPTABLES.

Este capitulo está dedicado al análisis descriptivo y comparativo de cinco algoritmos de codificación PCM diferencial, con predicción adaptable, utilizando el algoritmo de cuantificación adaptable propuesto por Schäfer [18].

El primer algoritmo de predicción adaptable fue propuesto por Graham [4]. Los cuatro algoritmos siguientes son modificaciones del algoritmo original de Zschunke [19,20]. De estos cuatro, el primero es debido a Kretz [10], el siguiente a Dewitte [2] y los dos restantes se proponen en este trabajo.

El concepto de adaptabilidad para el cálculo de la predicción en los primeros tres algoritmos, consiste en utilizar en la linea actual, el predictor que haya resultado ser el mejor para la linea precedente. Lo cual se traduce en una estimación de la orientación local, mediante la comparacion de los errores de predicción de distintos predictores seleccionados, de tal manera, que éstos sean óptimos para las diferentes orientaciones.

En los algoritmos que aqui se proponen, no se utilizan predictores óptimos para ciertas orientaciones, sino que se prefieren predictores que se comporten isotrópicamente para todas las orientaciones.

Cada algoritmo de predicción adaptable, fue probado con un cuantificador fijo de 11 niveles de reconstrucción. Y el cuantificador adaptable fue probado con tres predictores lineales fijos. Todo ello, con el objeto de contar con más elementos para contrastar cada par predictor-cuantificador.

El cuantificador adaptable [18] se diseñó con base en un criterio subjetivo, tomando en cuenta las propiedades del sistema visual humano.

VI.1 Análisis de los principales algoritmos de predicción adaptable.

VI.1.1 El algoritmo de Graham.

La noción de predicción adaptable o *no lineal* fue propuesta por R.E. Graham [4], que la define como una selección para cada punto, de una predicción a partir de un conjunto de predicciones lineales. El algoritmo de selección consiste entonces en una serie de pruebas lógicas sobre los puntos previamente decodificados. El ejemplo de Graham es una predicción de dos tipos: $P = A$ o $P = C$, dependiendo de que B esté más cercano a C o a A. Se puede considerar que el estado de adaptabilidad admite dos valores e_1 y e_2:

$$\begin{aligned} &\text{Si } |B - A| \geq |B - C| \text{, el estado es } e_1 \text{ y } P = A \\ &\text{Si } |B - C| > |B - A| \text{, el estado es } e_2 \text{ y } P = C \end{aligned} \qquad \text{(VI.1)}$$

Este sencillo algoritmo se puede interpretar como la detección de un contorno horizontal (estado e_1) y la detección de un contorno vertical (estado e_2). El autor remarca que los errores de predicción más grandes se producen en los codos, en las bifurcaciones, en los contornos a 45° y en pequeños detalles horizontales y verticales. Debe notarse además, que las pruebas lógicas de este algoritmo (ecuación VI.1), son particularmente sensibles al ruido de fuente, debido a la acumulación de operaciones de diferencia (las dos pruebas equivalen a la comparación de $|B - A| - |B - C|$ con 0).

El autor reconoce la gran sensibilidad del algoritmo a los errores de transmisión y sugiere introducir un coeficiente de fuga α, que conmuta a las predicciones αA o αC en vez de A o C, respectivamente.

VI.1.2 El algoritmo de Zschunke.

Este algoritmo incluye dos etapas que consisten en: el cálculo de la orientación local para cada punto decodificado y el cálculo de la predicción a partir del estado de la vecindad local del punto a codificar [20].

VI.1.2.1 Cálculo de la orientación local en $\hat{X}$.

Sea $\hat{X}$ un punto ya decodificado y sean A, B, C, ... los puntos vecinos de $\hat{X}$, también ya decodificados (Fig. V.7). Deben de utilizarse ocho predictores P_θ, asociados a ocho orientaciones, en donde θ está en el rango de -3 a 4 :

$$P_{-3} = E\ ,\quad P_{-2} = D\ ,\quad P_{-1} = (C + D)/2\ ,\quad P_0 = C\ ,$$
$$P_1 = (B + C)/2,\quad P_2 = B\ ,\quad P_3 = AB \quad \text{y} \quad P_4 = A\ . \tag{VI.2}$$

Si $\hat{X}$ no es un punto perteneciente a un contorno (prueba: $|\hat{X} - A| \leq 25$), entonces es afectado por un ángulo $\theta = 4$. En los otros casos, se comparan las diferentes predicciones, P_θ , con $\hat{X}$ bajo la restricción de sólo considerar aquellas que cumplan con las siguientes condiciones:

$$|\hat{X} - P_\theta| \leq 64\ ,\quad \theta = -3 \text{ a } 4 \tag{VI.3}$$

$$\text{signo}\ (\hat{X} - A) = \text{signo}\ (P_\theta - P_{\theta+1})\ ,\quad \theta = -3 \text{ a } 2 \tag{VI.4}$$

Si ninguna P_θ cumple con estas restricciones, entonces $\theta = 4$; de otra manera se tomará el valor de θ que minimice $|\hat{X} - P_\theta|$ (cuando haya ambigüedad, -varios ángulos θ equivalentes- , se tomará el más grande algebraicamente). Por lo tanto, todas las $\hat{X}$ son afectadas por un ángulo θ ($\theta = 4$ para las zonas uniformes, los contornos horizontales y los puntos impredecibles).

VI.1.2.2 Cálculo de la predicción a partir del estado de la vecindad local en X .

Sea ahora el punto actual X y sus seis puntos vecinos ya decodificados A, AB, B, C, D y E . El algoritmo debe seleccionar uno de ocho predictores, $P_{\theta(X)}$, o bien un ángulo $\theta(X)$. Ambos determinados por las ecuaciones VI.2 . El procedimiento es el siguiente: cuando los seis puntos vecinos han sido afectados por un ángulo $\theta = +4$, entonces $\theta(X) = 4$ y $P_{\theta(X)} = A$. Si uno

de los seis puntos vecinos ha sido afectado por un ángulo diferente de +4 , sea θ_0 , entonces, $\theta(X) = \theta_0$, de donde $P = P_{\theta 0}$. Si varios puntos vecinos son afectados por un ángulo $\theta \neq +4$, se define para cada uno un ángulo φ con respecto a X ($\varphi = -3$ para E , -2 para D , 0 para C , $+2$ para B , $+3$ para AB y $+4$ para A) y se selecciona para $\theta(X)$, el ángulo θ que afecta el punto vecino de X tal que $[\theta - \varphi]$ sea mínimo (si hay varios, se toma el ángulo θ más grande algebraicamente). Se hace notar que $(B + C)/2$ y $(C + D)/2$ no intervienen en esta etapa de decisión. El interés de este procedimiento es predecir $\theta(X)$ a partir de la orientación más probable de un punto vecino a X , situado sobre el mismo contorno (en la misma dirección con respecto a X). También cabe mencionar que $\theta(X)$ puede tener los valores $+1$ y -1 contrariamente a φ . La predicción es entonces determinada mediante la ecuación VI.2 .

El algoritmo de Zschunke tiene la ventaja de separar los puntos no contorno, para los que $\theta = 4$ ($P = A$), de los puntos contorno, para los que solamente se determina una orientación local (contrariamente al algoritmo de Graham, donde se determina una sola orientación en cada punto). En contraparte, las pruebas lógicas de este algoritmo son únicamente locales y existe el riesgo de sensibilidad al ruido de fuente, ya que no hay un seguimiento, ni un filtrado de la predicción de la orientación local, a lo largo de los contornos.

VI.1.3 El algoritmo de Dewitte.

En la referencia [2], Dewitte propone un esquema para optimizar los predictores frecuentemente utilizados en una codificación PCM diferencial adaptable. La base del algoritmo de optimización, es la comparación entre las curvas de eficiencia de los predictores y la de selectividad de los estimadores de orientación. Estas últimas propuestas por el autor, se muestran aquí en la figura VI.1 .

En la tabla (VI.1) se muestran los resultados de la optimización llevada a cabo, para los algoritmos de Graham, Zschunke y un predictor lineal fijo.

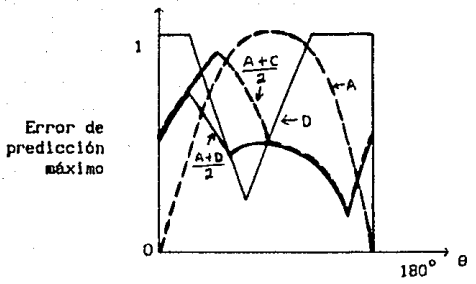


Fig. VI.1 Eficiencia de un conjunto de predictores de las orientaciones de los contornos.

Las ecuaciones VI.5 que se muestran en la tabla (VI.1) reemplazan a las ecuaciones VI.2 en el algoritmo de Dewitte.

TABLA VI.1 Predictores originales y predictores optimizados respecto a su selectividad a los contornos.

Predictor	Original	Optimizado
Fijo	A	$(2A + C + D)/4$
Graham	A	$(2A + D)/3$
	C	$(A + 2A)/3$
Zschunke	A	$(2A + C)/3$
	AB	$(2A + C)/3$
	B	$(A + B + C)/3$
	$(B + C)/2$	$(A + 2C)/3$
	C	$(A + 2C)/3$
	$(C + D)/2$	$(A + C + D)/3$
	D	$(A + D + E)/3$
	E	$(2A + C + E)/4$

(VI.5)

VI.1.4 Los algoritmos D^* y D2 .

Los algoritmos D^* y D2, modifican la forma en que se calcula la predicción de los puntos, a partir del estado de la vecindad local usada en el algoritmo de Zschunke y emplean las pruebas lógicas siguientes:

$$\text{Si } \theta < 0 \quad P = (A + D)/2$$
$$\text{Si no , } \quad P = (A + C)/2 \qquad \text{(VI.6)}$$

En estos dos algoritmos, existe una estimación de la orientación local para cada punto decodificado y se consideran las ecuaciones VI.2 para el algoritmo D^* y las ecuaciones VI.5 para el algoritmo D2 . El estado local está determinado por las pruebas lógicas anteriores. El predictor lineal $(A + C)/2$ se introduce debido a su comportamiento isotrópico frente a las orientaciones -1, 0, 1, 2, 3 y 4 y sólo se sustituye por el predictor $(A + D)/2$ para las orientaciones -2 y -3 , en donde éste último resulta ser más eficaz. Se ha podido demostrar experimentalmente, que los dos predictores presentan gran robustez frente a los errores de transmisión [18].

VI.2 Descripción del algoritmo de cuantificación adaptable.

En esta sección, se describe el algoritmo de cuantificación adaptable de Schäfer [18]. La característica principal de este algoritmo, es que aprovecha las propiedades del sistema visual humano. Con él es posible lograr una razón de compresión de 8 a 3.5 . En particular, el sistema de cuantificación utiliza el fenómeno de enmascaramiento espacial, comentado en el capítulo I , el cual permite ocultar los errores inherentes a la presencia del cuantificador en el sistema.

Con respecto al efecto de enmascaramiento espacial, en el diseño del cuantificador se tomaron en cuenta las siguientes consideraciones:

a) El umbral de visibilidad en un contorno se incrementa con la pendiente del mismo y con la diferencia de la señal (luminancia) en ambos

lados del contorno.

b) El efecto de enmascaramiento decrece rápidamente con la distancia (medida a partir del contorno).

c) El incremento del nivel de percepción para diferencias de luminancia (debido al efecto de enmascaramiento), es proporcional a la diferencia de intensidad entre el estímulo de prueba y el estímulo de enmascaramiento. Para este caso, el incremento es casi independiente del valor de luminancia del fondo.

d) El signo de la diferencia de luminancia (error de predicción) es de suma importancia, pues cuando el estímulo de enmascaramiento es más brillante que el estímulo de prueba, el incremento en el nivel de visibilidad es de alrededor de 100% mayor que aquel correspondiente al caso en el que el estímulo de enmascaramiento es menos brillante que el estímulo de prueba. Esto significa que existe una característica de asimetría entre los dos casos.

Para que el fenómeno de enmascaramiento espacial en un sistema DPCM pueda ser utilizado adecuadamente [15], es indispensable encontrar una función que describa la relación entre el umbral de visiblidad del error que debe ser ocultado y el patrón que se observa en la imagen. Para ello, se define una *función de enmascaramiento* y una *función de la actividad local de luminancia*, las cuales permiten obtener una medida cuantitativa del enmascaramiento que se presentará en la vecindad del punto actual. De este modo, aquellos patrones con el mismo valor de actividad local, tendrán un nivel de visibilidad de error igual, puesto que el efecto de enmascaramiento presente, afecta de la misma forma la observación de los patrones.

Estas dos funciones están relacionadas de la manera siguiente: la función de enmascaramiento, es la relación que existe entre el nivel de visibilidad de las distorsiones de una imagen y la función de actividad local. Debido a que esta distorsión es provocada en el cuantificador del sistema DPCM, por los errores de cuantificación, diremos que la función de enmascaramiento relaciona la visibilidad de los errores de cuantificación y la función de actividad local:

$$V = M(A) \qquad \text{(VI.7)}$$

donde V, es el nivel de visibilidad, M, la función de enmascaramiento y A, la función de actividad local (de luminancia).

Para cada función de actividad local que se define, la función de enmascaramiento debe de encontrarse a partir de pruebas subjetivas.

VI.2.1 Funciones de actividad local de luminancia.

Las funciones de actividad local utilizadas en el cuantificador que se describe son las siguientes:

$$A_{SD} = \max_{i,j=1\ldots4} |s_i - s_j| \qquad \text{(VI.8)}$$

donde: $s_1 = A$, $s_2 = B$, $s_3 = C$ y $s_4 = D$.

$$A_{WD} = \max \{ |d_i| + 0.25\,(d_i + |d_i|)\,(1 - \text{signo}(\delta)) \} \qquad \text{(VI.9)}$$

donde δ es el error de predicción y $d_i = s_i - P$.

La función A_{SD} fue elegida después de haber sido comparadas cuatro funciones diferentes [18]. La función A_{WD}, se definió tomando en cuenta las características de asimetría del efecto de enmascaramiento: en la ecuación (VI.9) el valor d_i, se multiplica por dos, si el punto s_i es definitivamente más brillante que el punto actual X.

La figura (VI.2) aclara el procedimiento de definición de la ecuación VI.9 : se define como actividad brillante cuando la diferencia de luminancia, d_k, de mayor valor, es positiva y el error de predicción, δ, para el punto actual, X, es negativo. La actividad obscura se presenta cuando el punto, s_k (cuya diferencia de luminancia d_k, es mayor), es definitivamente más obscuro que el punto actual, X. La actividad brillante proporciona un nivel de percepción de errores más alto que la actividad obscura.

Si las caractertísticas del cuantificador adaptable tienen el mismo número de niveles positivos y negativos, otro aspecto relevante de la función A_{WD}, es que el signo del error de predicción puede utilizarse en la codificación de canal. En este caso, el decodificador es capaz de reconocer

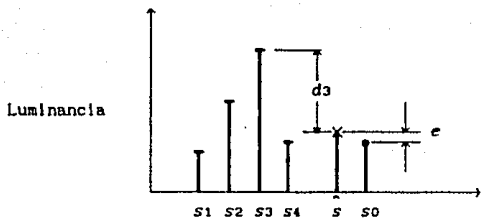


Fig. VI.2 Ejemplo de la actividad brillante. El valor más grande de luminancia d_3 es positivo y el error de predicción, δ , es negativo. El elemento de la imagen s_3 es definitivamente más brillante que X .

el signo del código recibido, antes de decidir a que cuantificador conmutar. Sin embargo, esta función no puede utilizarse, pues si el valor de la actividad local es pequeño y el valor del error de predicción es grande, la decisión en la conmutación es errónea. Por esta razón, la función de actividad local óptima es:

$$A_{SWD} = \max (A_{SD} , A_{WD}) \qquad (VI.10)$$

VI.2.2. Función de enmascaramiento, actividad local y diseño del cuantificador adaptable.

Como ya se ha señalado, una vez que se ha elegido la función de actividad local, se procede a obtener la función de enmascaramiento por la via experimental. Este procedimiento es seleccionado por el investigador, utilizando como referencia los diferentes tipos de estimulos que pueden presentarse a los observadores, para producir un efecto de enmascaramiento.

La función de enmascaramiento se obtiene como una característica E vs. δ, donde E, representa los diferentes valores para el error de cuantificación y δ el error de predicción. El método de diseño consiste en aplicar la ecuación VI.11 en cada intervalo del cuantificador, Q:

$$Q(\delta) = | \delta - r_k | \leq M(\delta) \qquad \text{(VI.11)}$$

donde r_k son los niveles representativos de Q, es decir, los puntos donde el error de cuantificación es cero. Procediendo de este modo se obtiene una característica de cuantificación no uniforme, óptima respecto a la visibilidad (casi nula) de los errores de cuantificación.

El siguiente paso, es obtener una característica de la función de actividad de luminancia vs. el error de cuantificación (nivel de percepción). Esta característica junto con la función de enmascaramiento, generan la función buscada, (ecuación VI.7).

En general, el número de niveles del cuantificador no es lo suficientemente pequeño, para poder lograr una razón de compresión adecuada. Por ello, el cuantificador fijo original que se obtiene a partir del procedimiento descrito, es generalmente particionado convenientemente en varios cuantificadores fijos, con lo que es posible lograr una compresión al menos aproximada a la que se desea. En resumen, la función de actividad de luminancia es calculada en la vecindad de cada elemento de la imagen. El valor obtenido se compara con ciertos intervalos, definidos experimentalmente, que permiten seleccionar la característica de cuantificación correcta. En la tabla VI.2 se muestran: el cuantificador fijo (obtenido mediante el método descrito previamente) y los cuantificadores derivados del mismo. Puede notarse que para la característica *BQC*, existen 17 niveles, que constituyen un cuantificador no uniforme de tipo *midtread*. La compresión que se puede lograr con él es baja, aunque la calidad de la imagen codificada es muy adecuada. Sin embargo, el cuantificador adaptable (integrado por un conjunto de cuatro cuantificadores no uniformes, de característica midtread e intervalos de conmutación) permite alcanzar una razon de compresion de 8 a 3.5 bits por punto. Los niveles de conmutacion son tres para estas características, dando lugar a cuatro intervalos, que corresponden a cada uno de los cuantificadores que se pueden utilizar.

VI.3 Propiedades de la imagen concernientes a la adaptabilidad del cuantificador.

Otra alternativa para el diseño de un cuantificador, es adaptar su dinámica a las propiedades de la fuente para reducir el número de códigos útiles. Un cuantificador que atiende a estas características es el de *dinámica deslizable*. En este trabajo fue probado un cuantificador de este tipo y los resultados obtenidos mostraron la imposibilidad de alcanzar la razón de compresión que se buscaba (8 a 3.5) con calidad aceptable. Sin embargo, se describe a continuación su principio de operación, debido a que se hará referencia a él en el capitulo VII .

VI.3.1 El cuantificador deslizable.

Cuando el nivel de luminancia de una imagen es cercano al negro, los valores negativos de la característica de cuantificación pueden evitarse, debido a que el rango dinámico de la imagen está limitado. Las palabras de código que no se utilizan pueden ser usadas sin ambigüedad, adicionando niveles de cuantificación al final de la característica, mejorándose así la asignación de códigos para la transmisión. Un razonamiento similar se aplica cuando el nivel de gris es cercano al blanco. Un cuantificador de dinámica deslizable, basado en este principio, tiene la ventaja de extender el rango de la característica de cuantificación para un cierto número de bits por punto. Una ventaja adicional es que reduce la sensibilidad a los errores de transmisión: si los valores cuantificados son enumerados en cada zona (del más negativo al más positivo), los efectos debidos a errores se atenúan y desaparecen rápidamente [17].

La figura VI.3 muestra en forma esquemática un cuantificador con esta dinámica. Se puede observar como el rango dinámico de los niveles de predicción ha sido dividido en siete zonas, cada una de las cuales determina un cuantificador específico para esa zona. Cada cuantificador tiene un rango de variación limitado, de acuerdo a los valores que puede tomar la predicción. De este modo es posible asignar ocho niveles a cada

Tabla VI.2 Niveles representativos de cuantificadores con características de simetría.

Característica de cuantificación.	No. de niveles.	Niveles representativos positivos.
(f i j o)		
B Q C	17	0,3,8,15,24,35,48,65,86
(a d a p t a b l e)		
P Q C 111	11	0,3,8,15,24,35
P Q C 112	11	0,7,14,23,34,47
P Q C 113	11	0,11,23,35,48,65
P Q C 114	11	0,15,30,45,64,85
Niveles de conmutación para A_{S*D}		
Intervalos definidos por:		15,35,100

cuantificador, con lo que solo se utilizan tres bits por punto en la región de variación correspondiente a cada zona.

VI.4 Estudio comparativo de los diversos algoritmos.

El estudio de los cinco algoritmos que presentamos previamente, esta restringido a la imagen "*bateau et phare*" y ello limita en parte la generalidad del estudio. Sin embargo, la intención fundamental es desglosar el problema para tener elementos de juicio que permitan decidir que algoritmo es el mejor. La imagen de prueba es una arreglo cuadrado de 256 x 256 elementos y corresponde a una parte de la imagen original. Esta contiene 625 líneas y fue muestreada a una frecuencia de 10 MHz.

El análisis en el estudio es básicamente cualitativo: se trata de

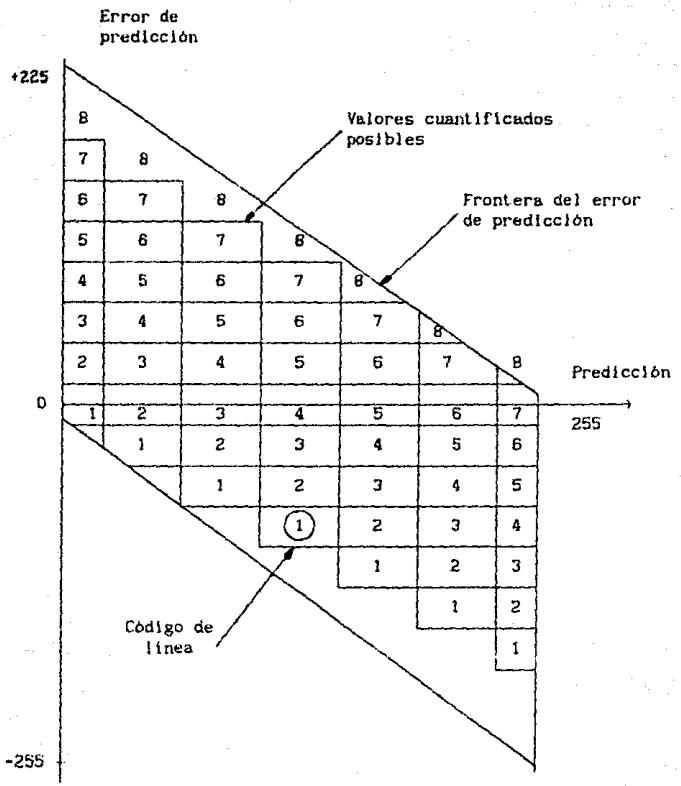


Fig VI.3 Diagrama de un cuantificador de dinámica deslizable.

determinar que par predictor-cuantificador resulta más eficaz atendiendo a diversos criterios, como por ejemplo: el comportamiento del algoritmo al ruido de canal.

Un estudio comparativo sobre varias imágenes, a partir del algoritmo seleccionado, debe de realizarse en situaciones realistas donde el ruido de fuente y los errores de transmisión cambian de imagen a imagen. Sin embargo, es indispensable, antes de llevar a cabo dicho estudio, realizar una serie de pruebas para evaluar la factibilidad de implantación del algoritmo en alguna arquitectura. Si los resultados indican que dicha implantación es posible en tiempo real, entonces se puede proceder a depurar el algoritmo y probarlo en condiciones más próximas a las reales.

La comparación que se describe a continuación toma en cuenta cuatro aspectos: la observación del error de predicción $\hat{X} - P$ para los diferentes predictores, la medición de las probabilidades de que el error de predicción sobrepase un umbral U , $P_U = P_r\{|X - P| > U\}$, la observación del efecto de los errores de transmisión en la decodificación y la calidad subjetiva de las imágenes decodificadas.

VI.4.1 Observación del error de predicción.

Si P es la predicción del punto actual X , para alguno de los algoritmos estudiados, usualmente se observa la imagen del error de predicción $(X - P)$ obtenida, adicionándole un gris uniforme $(G = 80)$. La imagen visualizada es entonces $Y = 80 + X - P$. Es conveniente tener conciencia de los límites de esta aproximación: la comparación de tales imágenes permite al menos decir en qué zona de la imagen el algoritmo de predicción parece mejor que en otra, porque el error de predicción en esa zona es más pequeño. Pero nada podemos concluir acerca del nivel de calidad que alcanzarán los algoritmos (predicción y cuantificación) en condiciones reales. Las pruebas fueron realizadas inicialmente con un cuantificador fijo de 11 niveles.

Las figuras VI.4 y VI.5 muestran las imágenes del error de predicción para los predictores fijos: A, $(A + C)/2$ y $(A + D)/2$ y para los cinco

algoritmos adaptables estudiados. La imagen original que corresponde a estas imágenes, se muestra en la figura VI.12(a) . El predictor A (Fig. VI.4 (a)) produce errores grandes sobre los contornos cercanos a la vertical (gradiente horizontal elevado) y errores muy pequeños sobre los contornos cercanos a la horizontal. El predictor $(A + C)/2$ (Fig. VI.4(b)) produce errores de amplitud máxima más pequeños que C , en contornos verticales ($P = C$) y en contornos oblicuos con pendiente negativa ($P = AB$ o E), pero su comportamiento es similar al predictor A en los contornos oblicuos con pendiente positiva ($P = D$ o E), presentando errores en contornos cercanos a la horizontal ($P = A$) . El predictor $(A + D)/2$ (Fig VI.4(c)), produce errores de amplitud máxima más pequeños que A , pero en forma casi isotrópica (todas las orientaciones son afectadas) y también en las regiones del fondo cercanas a los contornos.

El algoritmo de Graham da buenos resultados en los contornos horizontales ($P = A$) y verticales ($P = C$), pero es mala su adaptabilidad para el caso de contornos oblicuos (Fig. VI.4 (d)).

El algoritmo de Zschunke-Kretz, parece relativamente isotrópico, pero contrariamente al predictor $(A + D)/2$ que produce errores de predicción correlacionados a lo largo de los contornos, este algoritmo parece descorrelacionar el error de predicción a lo largo de ellos: esto aparece, en efecto, como un ruido a nivel de los contornos (Fig. VI.5(a)). Esta es, seguramente, una prueba de que la adaptación se produce, pero el tipo de propagación del error parece relativamente elevado y por otra parte, los errores de predicción tienen tendencia a propagarse alrededor de las transiciones.

El algoritmo de Zschunke-Dewitte, (Fig. VI.5(b)) mejora sensiblemente la descorrelación que existe en el algoritmo original de Zschunke-Kretz, lo que prueba que la optimización de los predictores llevada a cabo por Dewitte, usando como criterio la eficacia y la selectividad del predictor para ciertas orientaciones, es adecuada. Sin embargo como se verá después, los esquemas adaptables que utilizan predictores que se comportan isotrópicamente para todas las orientaciones, dan mejores resultados.

Los algoritmos D* y D2 (Fig. VI.5(c) y (d)), también presentan indicios de que la adaptación se efectúa adecuadamente; sin embargo, la descorrelación

y propagación del error de predicción en y alrededor de los contornos, es prácticamente inexistente. Su comportamiento es casi isotrópico para todas las orientaciones.

El algoritmo D^* se asemeja al predictor $(A + C)/2$ pero en general son menores los errores en todas las orientaciones y sobretodo en los contornos oblicuos con pendiente positiva, que en el caso del predictor $(A + C)/2$, se presenta en forma más marcada.

El algoritmo D2 se parece al predictor $(A + D)/2$, aunque provoca una mejora en varias zonas de la imagen, ya que no se observa una propagación del error en el fondo de la misma, ni en las zonas cercanas a los contornos. Existe mayor uniformidad en toda la imagen.

Para las pruebas con los diferentes predictores utilizando ahora un cuantificador adaptable (por función de enmascaramiento), el cual emplea cuantificadores fijos cada uno de 11 niveles, los resultados observados se indican a continuación: Para la predicción A (Fig VI.6(a)), se observa un comportamiento igual que con el cuantificador fijo: hay errores grandes en los contornos cercanos a la vertical y errores pequeños en los contornos horizontales. En las zonas cercanas a los contornos se presenta una disminución de la propagación del error de predicción. Para el predictor $(A + C)/2$ (Fig VI.6(b)), se observa que el error de predicción se atenúa y se localiza a ambos lados de los contornos. En la predición $(A + D)/2$ (Fig. VI.6(c)), se observan resultados similares.

En el algoritmo de Graham, hay una notable mejoría al utilizar un cuantificador adaptable, ya que no se presentan errores en el fondo como en el caso del cuantificador fijo. El error en los contornos es muy parecido al del caso con cuantificador fijo, con errores pequeños en los contornos horizontales y verticales y errores muy marcados en los contornos oblicuos (Fig VI.6(d)).

En el algoritmo de Zschunke-Kretz, disminuye la descorrelación. En el algoritmo de Zschunke-Dewitte, se observa poca diferencia en comparación con el cuantificador fijo. Son ligeramente menores los errores en los contornos verticales y decrece la propagación del error de predicción hacia el fondo de la imagen (Fig. VI.7(a) y (b)).

En el algoritmo D^* se observa una disminución del error en los contornos

oblicuos y se observa mayor uniformidad sobre toda la imagen. Para el algoritmo D2 los resultados muestran una ligera disminución del error, pero su comportamiento es semejante que con el cuantificador fijo. Se observa una imagen con cambios menos bruscos en el fondo y en las fronteras de los objetos (Fig. VI.7(c) y (d)).

Las observaciones indican que los predictores adaptables que utilizan predicciones isotrópicas para las diferentes orientaciones, tienen gran similitud con los predictores lineales fijos. Sin embargo, el error de predicción pico es mucho menor [6]. Es decir, las características deseables de los predictores $(A + C)/2$ y $(A + D)/2$ están incluidas en estos esquemas adaptables. Y además, se preservan las ventajas del cálculo de una estimación de la orientación local. Con todo ello, las imágenes presentan en promedio un nivel de error menos elevado y más uniforme sobre toda la imagen.

El algoritmo de Graham no proporciona ninguna ventaja en comparación con los otros algoritmos de predicción adaptable. Como se anotará más adelante, otra gran desventaja es su susceptibilidad a los errores de transmisión.

Las observaciones también indican que el efecto del cuantificador adaptable se traduce en una disminución del error de predicción: se nota que las transiciones de luminancia se suavizan, con lo que se aprecia un efecto subjetivo de filtrado pasabajas sobre la señal δ , ya que los contornos en la imagen contribuyen con altas frecuencias en el espectro.

El efecto de filtrado se atenúa si la transición es demasiado brusca. Sin embargo, son precisamente estas transiciones las que aprovecha el cuantificador adaptable para enmascarar los errores de cuantificación. Se puede concluir así, que sólo se permite el paso de frecuencias para las que se presentará un nivel alto de visibilidad del error de predicción (conforme se incrementa el nivel de visibilidad, la percepción del error disminuye).

Si la transición es tenue, se minimiza el error de predicción. Si la transicion es de intensidad media, el error de predicción se atenúa y se hace simétrico alrededor de los contornos. (*v. gr.* Fig. VI.4(b) y Fig. VI.6(b)).

Se puede decir entonces que el cuantificador adaptable tiende a correlacionar el error de predicción al inicio y al final de una transición de luminancia, cuando dicha transición no es demasiado brusca.

Probablemente, el efecto subjetivo de filtrado, es responsable de la

eliminación de la descorrelación en los algoritmos adaptables: la descorrelación del error de predicción indica que la adaptación se está llevando a cabo. Pero en presencia del cuantificador adaptable, los valores de la función de actividad pueden discriminar eficientemente la actividad local de luminancia y conmutar al cuantificador fijo adecuado. De este modo, la pareja predictor-cuantificador converge rápidamente. Entre más robusto sea el esquema de predicción, la pareja funcionará más eficazmente. Nótese que cuando se habla de una transición tenue, media o brusca, se hace referencia a la imagen del error de predicción.

El cuantificador adaptable, en el caso de predictores fijos, tiende a corregir el error de predicción para aquellas orientaciones en las que estos últimos son poco eficientes. En el caso de los predictores adaptables se presenta un efecto similar, además de que el error de predicción tiende a estar más correlacionado.

VI.4.2 Medición de las probabilidades condicionales de que $|X - P|$ sobrepase un umbral U .

El objetivo de una predicción adaptable es reducir la dinámica del error de predicción. Para ello se han evaluado las probabilidades de que $|X - P| > U$, para $U = 40$, 50 y 65 , condicionalmente respecto a diversas clases de puntos. Estas clases están determinadas para cada punto decodificado a partir de las orientaciones locales que produce el algoritmo de Zschunke. Si $\theta = 4$ el punto es considerado clase 2 . Para $\theta \neq 4$ el punto se considera clase 1 . La clase 2 contiene aquellos puntos que no son contorno o son contornos horizontales. La clase 1 contiene a todos aquellos puntos contorno que no son horizontales.

Debe destacarse que para esta prueba, las probabilidades de las clases 1 y 2 son diferentes para cada pareja predictor-cuantificador. Por ello las probabilidades condicionales están normalizadas respecto a las probabilidades de cada clase.

Se pudo observar durante las pruebas que los pares predictor - cuantificador que tenían una mayor probabilidad de clase 1 fueron: P5C1 y

P5C5, sin embargo en la tabla VI.3 estos tienen una menor probabilidad de que el error de predicción sobrepase el umbral U . Respecto a las demás parejas predictor cuantificador y en este sentido, su comportamiento es el mejor. Le sigue en eficacia el P6C1 † .

VI.4.3 Efectos de los errores de transmisión.

Se abordará aquí uno de los aspectos esenciales de la eficacia de un algoritmo de codificación de fuente: el efecto de los errores de transmisión. Se admite generalmente que para la transmisión numérica de televisión, un nivel de calidad adecuado debe mantenerse el 99% del tiempo. En el 1% restante, se acepta un nivel menor de calidad. Aún más, si el sistema cuenta con dispositivos de protección contra los errores, no es necesario que el algoritmo sea poco sensible a los errores de transmisión.

No existe una relación directa entre la complejidad de un algoritmo y su robustez contra el ruido de canal: la relación de eficiencia y susceptibilidad a los errores de transmisón implica relaciones poco sencillas de analizar.

Para poder comparar los diversos algoritmos, no se simuló, en este estudio, una vía de transmisión real, en donde se debe generar ruido a diferentes tasas. Se evaluaron condiciones críticas que permiten una comparación directa: se introdujeron a la imagen 16 errores de amplitud fija

---

† Considérese:
P3 Algoritmo de Zschunke-Kretz.
P4 Algoritmo de Zschunke-Dewitte.
P5 Algoritmo D*.
P6 Algoritmo D2.
C1 Cuantificador fijo de 11 niveles.
C5 Cuantificador adaptable que utiliza una función de actividad de luminancia.

Tabla VI.3 Probabilidades condicionales de que el error de predicción sobrepase un cierto umbral dadas las clases 1 y 2 .

Algoritmo	P[|X-P|>40|1 o 2]		P[|X-P|>50|1 o 2]		P[|X-P|>65|1 o 2]	
	P[|X-P|>40|1 o 2]		P[|X-P|>50|1 o 2]		P[|X-P|>65|1 o 2]	
	P[1 o 2]		P[1 o 2]		P[1 o 2]	
	clase 1	clase 2	clase 1	clase 2	clase 1	clase 2
P3C1	0.0334 0.1511	0.0007 0.0009	0.0254 0.1149	0.0004 0.0005	0.0172 0.0549	0.0001 0.0001
P3C5	0.0726 0.4296	0.0013 0.0016	0.0445 0.2633	0.0005 0.0006	0.0275 0.1627	0.0001 0.0001
P4C1	0.0382 0.2198	0.001 0.0012	0.0265 0.1525	0.0003 0.0004	0.0235 0.1352	0.0002 0.0002
P4C5	0.079 0.6706	0.001 0.001	0.0447 0.3795	0.0003 0.0003	0.0344 0.292	0.0001 0.0001
P5C1	0.0213 0.1062	0.0009 0.0011	0.0197 0.0983	0.0002 0.0003	0.019 0.0948	0.0001 0.00013
P5C5	0.0244 0.1311	0.0001 0.0002	0.0211 0.1133	0.00008 0.0001	0.0204 0.1096	0.00006 0.0001
P6C1	0.0293 0.1947	0.0008 0.0009	0.0264 0.1754	0.0002 0.00024	0.0253 0.1681	0.00005 0.00006
P6C5	0.0325 0.2270	0.0001 0.0001	0.0309 0.2158	0.00004 0.00005	0.0265 0.1851	0.00002 0.000002

(+75 , -75). Los puntos afectados por el error de amplitud fueron idénticos para todos los algoritmos.

El tratamiento consistió en reemplazar en la malla del bloque de decodificación DPCM, $\hat{e}$ por $\hat{e} + 75$ o $\hat{e} - 75$, según el caso, en un punto en donde se deseaba introducir error.

En la secuencia de fotografías de las figuras VI.8 y VI.9 se observan los resultados obtenidos para los diferentes algoritmos con cuantificación fija. En la figura VI.8(a) se muestra el predictor A. El error de predicción se propaga solamente en la línea en que se presenta y prácticamente no existe atenuación del error. Los predictores $(A + C)/2$ y $(A + D)/2$ presentan una gran robustez frente al ruido de canal. Generalmente los predictores bidimensionales producen un patrón de error que se atenúa rápidamente en el área cercana al error (Fig. VI.8(b) y (c)). En la figura VI.8(d) se observa el algoritmo de Graham, en donde como puede observarse, se presentan degradaciones muy importantes.

La susceptibilidad de los algoritmos de Zschunke-Kretz y Zschunke - Lewitte es patente en las figuras V.8(a) y (b). En ellas está presente el efecto poco deseable de la predicción A en las zonas uniformes, el cual propaga los errores a las líneas subsecuentes. Sin embargo, en los algoritmos D^* y $D2$ la robustez de las predicciones $(A + D)/2$ y $(A + C)/2$ frente al ruido de canal, se preserva.

Las figuras VI.10 y VI.11 muestran el efecto de la introducción del cuantificador adaptable en el sistema. De las fotografías se desprende que si el esquema de predicción no es robusto frente al ruido de canal, el cuantificador adaptable propagará aún más este efecto. Esto puede observarse para el predictor A en la (Fig VI.10(a)), en donde se acentúa severamente el efecto de los errores de transmisión. Sin embargo, puede observarse que si el esquema de predicción es lo suficientemente robusto, las operaciones del cuantificador convergen a valores deseables. Obsérvense los predictores de las figuras VI.10(b), (c) y VI.11(c), en comparación con las figuras VI.8(b), (c) y VI.9(c), respectivamente.

VI.4.4 Observación de las imágenes decodificadas.

La observación de las imágenes decodificadas se llevó a cabo en un sistema PS/2 modelo 60. La resolución del sistema es de 320 x 256 con 64 niveles de gris. Subjetivamente se concluyó que las imagenes no diferían sustancialmente las unas de las otras. Se utilizó la opción de falso color (en tonos de azul, rojo y verde). Se observó que el algoritmo PSC1 presentaba mayor semejanza con la imagen original (Fig VI.12). Las observaciones finales se llevaron a cabo en un sistema de alta resolución. Subjetivamente existen muy pocas diferencias entre los diferentes algoritmos como puede observarse en las figuras VI.12 y VI.13 .

VI.5 Discusión.

El análisis comparativo que hasta aquí se ha llevado a cabo, permite por el momento seleccionar tres pares predictor-cuantificador: en primer lugar la pareja predictor D^* y cuantificador fijo de 11 niveles (PSC1). En segundo lugar el predictor fijo $(A + C)/2$ y el cuantificador adaptable. Finalmente el predictor $(A + C)/2$ y el cuantificador fijo de 11 niveles. La primer pareja se selecciona debido a que presenta gran robustez frente al ruido de canal y por otro lado, la estimación de la orientación local le permite obtener un error de predicción aceptable (nivel del error de predicción pico relativamente pequeño). Las probabilidades condicionales para este esquema resultan ser las más bajas.

La segunda pareja se elige debido a que tiene mucha similitud con la pareja PSC1 . Su comportamiento frente al ruido de canal es adecuado y la imagen del error de predicción es aceptable. Sin embargo, en este esquema no hay una estimación de la orientación. Este factor, puede ser decisivo para descartar a este algoritmo en pruebas con diferentes imágenes y en condiciones más reales de transmisión.

La última pareja se elige, ya que si bien su comportamiento no es tan adecuado como el de las parejas anteriores, puede servir como parametro de comparación para evaluar la factibilidad de implantación en una arquitectura

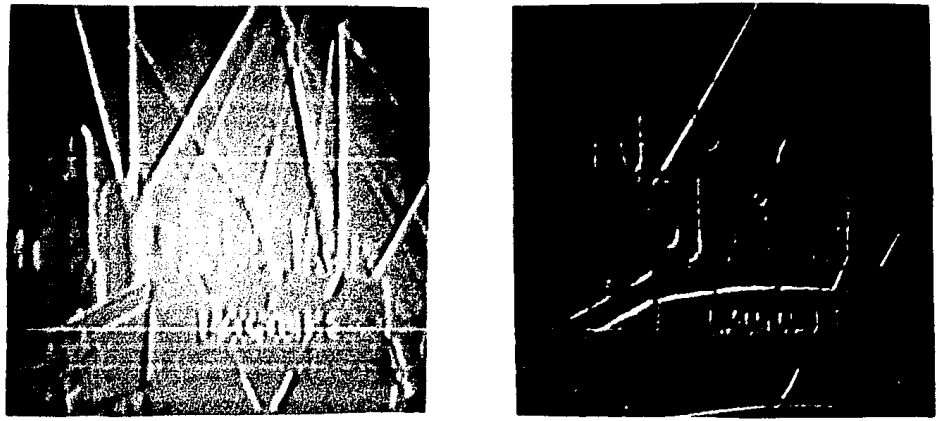


Fig. VI.4 Error de predicción (cuantificación fija). A la izquierda a) A, a la derecha b) (A+C)/2 .

Fig. VI.4 (Continuación) Error de predicción (cuantificación fija). A la izquierda c) (A+D)/2 y a la derecha d) Graham.

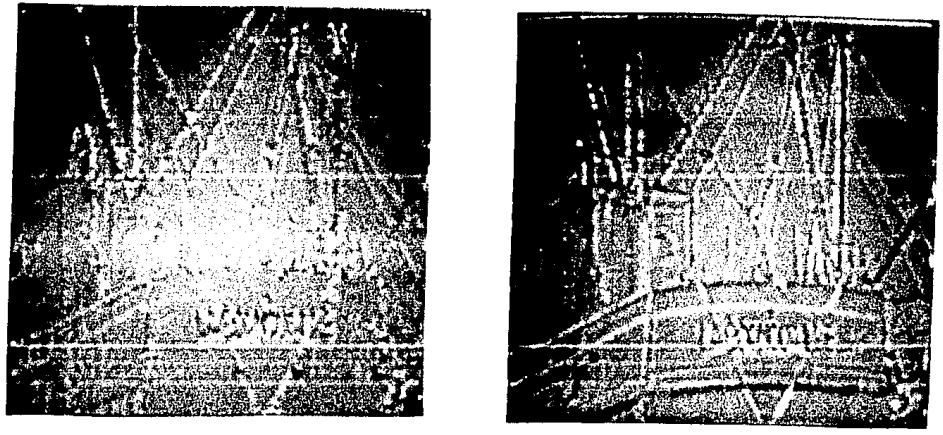


Fig. VI.5 Error de predicción (cuantificación fija). A la izquierda a) Zschunke-Kretz, a la derecha b) Zschunke-Dewitte.

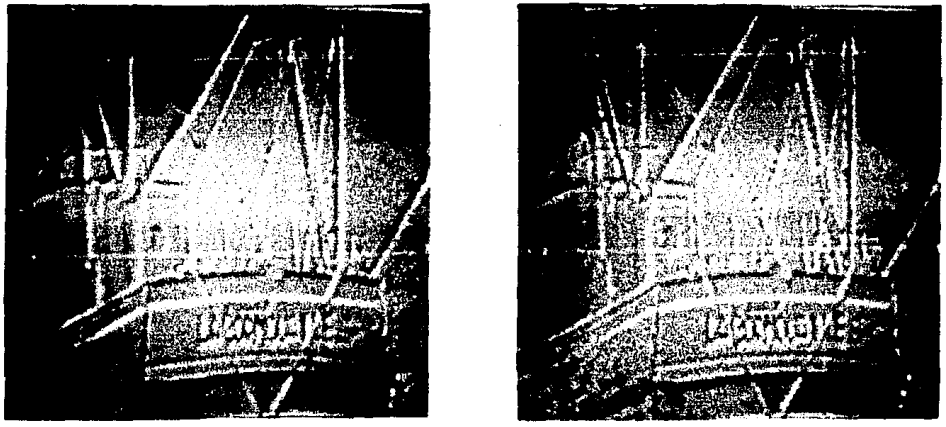


Fig. VI.5 (Continuación) Error de predicción (cuantificación fija). A la izquierda c) D* y a la derecha d) D2.

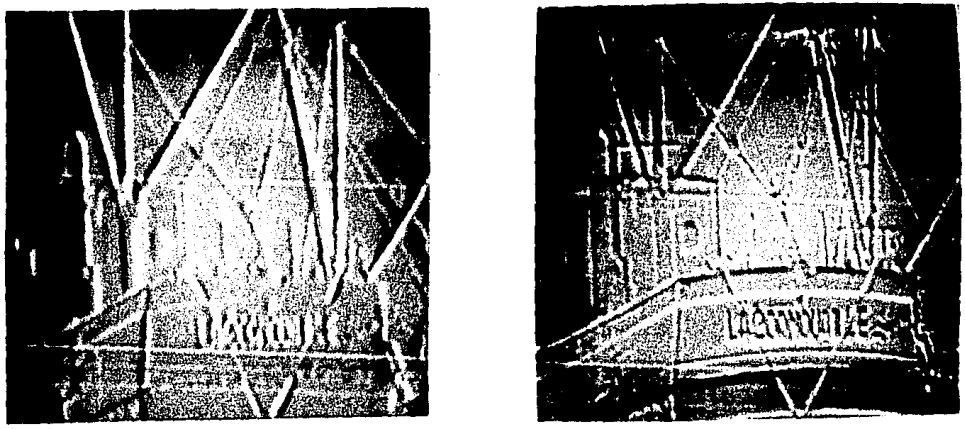


Fig. VI.6 Error de predicción (cuantificación adaptable).
A la izquierda a) A, a la derecha b) (A+C)/2 .

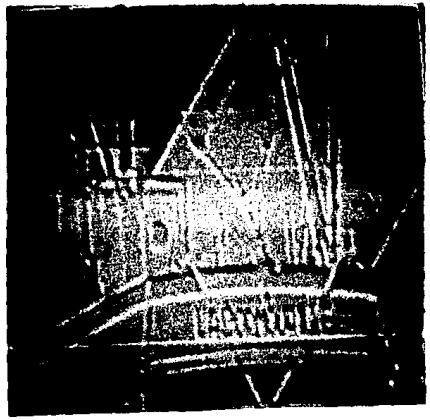


Fig. VI.6 (Continuación) Error de predicción (cuantificación adaptable). A la izquierda c) (A+D)/2 y a la derecha d) Graham.

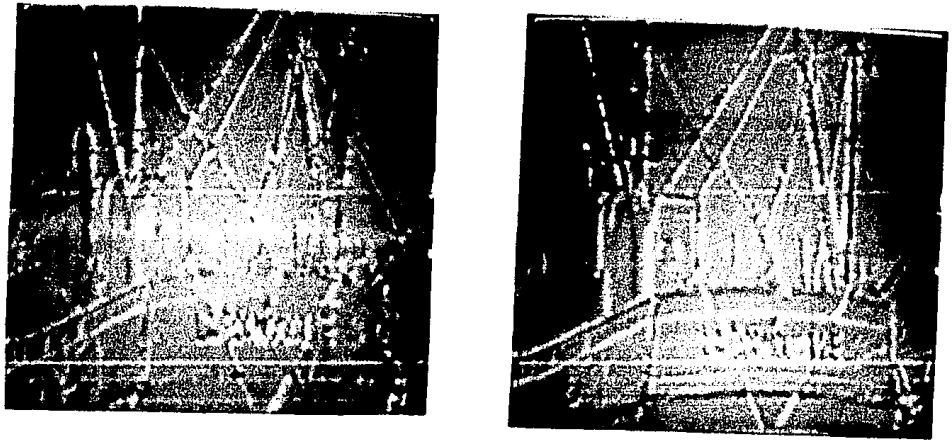


Fig. VI.7 Error de predicción (cuantificación adaptable). A la izquierda a) Zschunke-Kretz, a la derecha b) Zschunke-Dewitte.

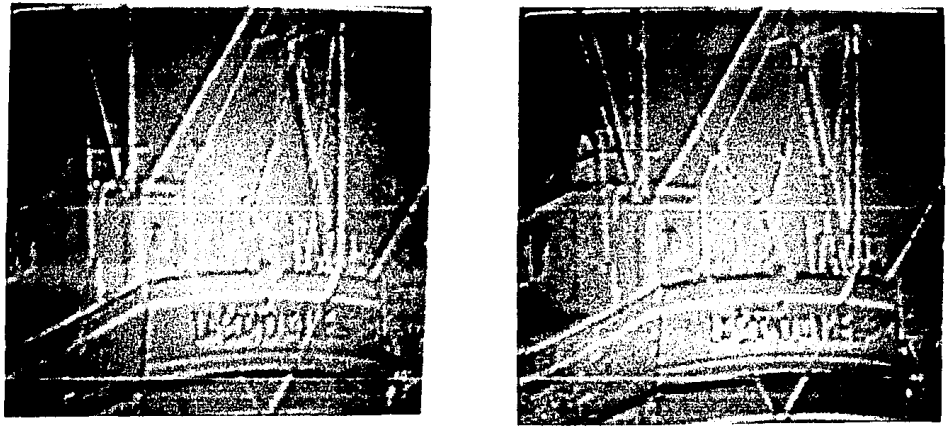


Fig. VI.7 (Continuación) Error de predicción (cuantificación adaptable). A la izquierda c) D* y a la derecha d) D2 .

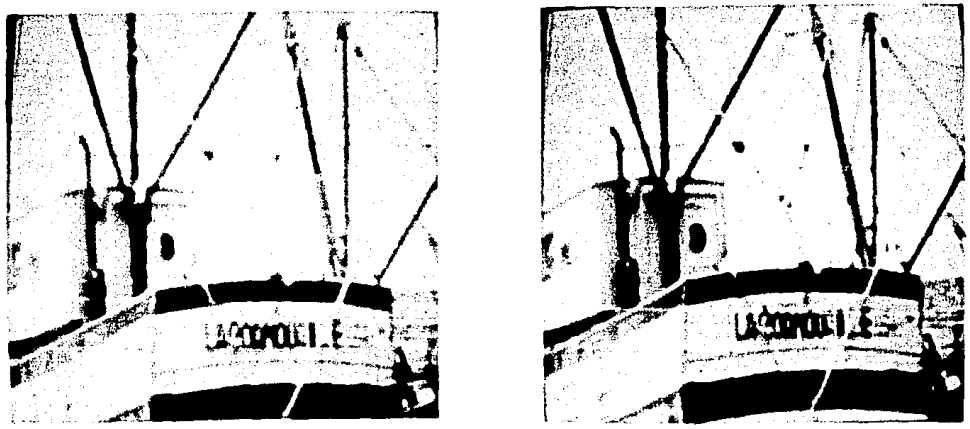


Fig. VI.8 Errores de transmisión (cuantificación fija).
A la izquierda a) A, a la derecha b) (A+C)/2 .

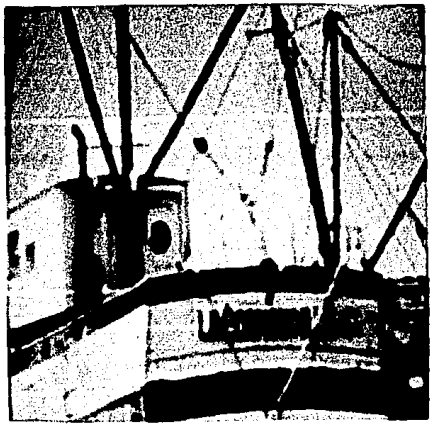


Fig. VI.8 (Continuación) Errores de transmisión (cuantificación fija). A la izquierda c) (A+D)/2 y a la derecha d) Graham.

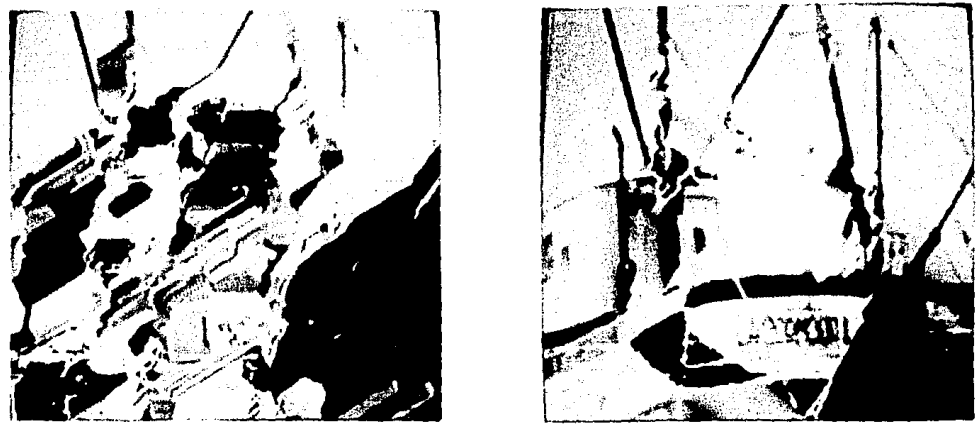


Fig. VI.9 Errores de transmisión (cuantificación fija). A la izquierda a) Zschunke-Kretz, a la derecha b) Zschunke-Dewitte.

Fig. VI.9 (Continuación) Errores de transmisión (cuantificación fija). A la izquierda c) D* y a la derecha d) D2 .

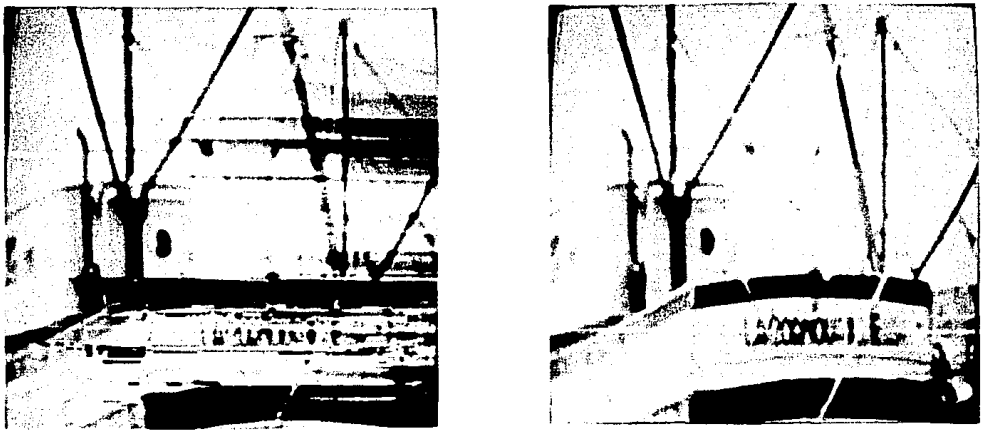


Fig. VI.10 Errores de transmisión (cuantificación adaptable). A la izquierda a) A, a la derecha b) (A+C)/2 .

Fig. VI.10 (Continuación) Errores de transmisión (cuantificación adaptable). A la izquierda c) (A+D)/2 y a la derecha d) Graham.

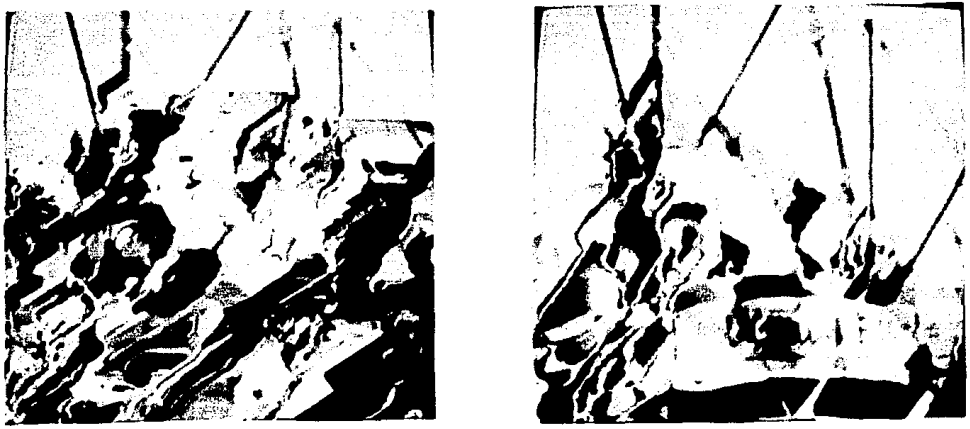


Fig. VI.11 Errores de transmisión (cuantificación adaptable). A la izquierda a) Zschunke-Kretz, a la derecha b) Zschunke-Dewitte.

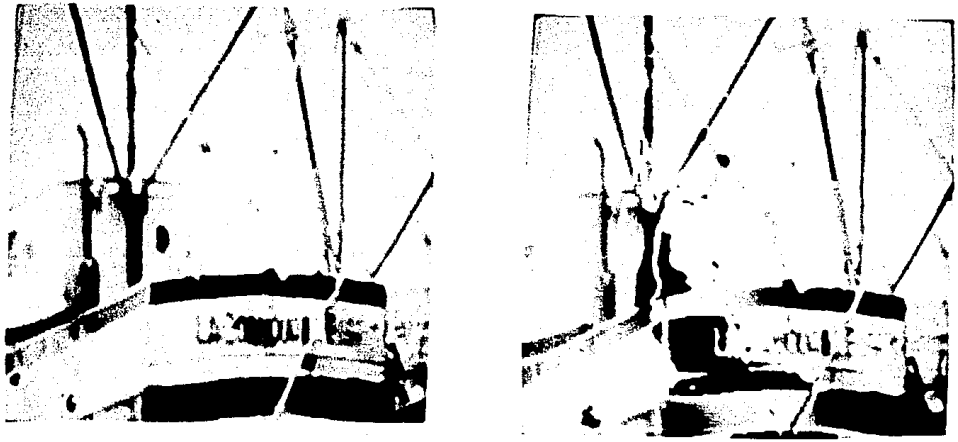


Fig. VI.11 (Continuación) Errores de transmisión (cuantificación adaptable). A la izquierda c) D* y a la derecha d) D2 .

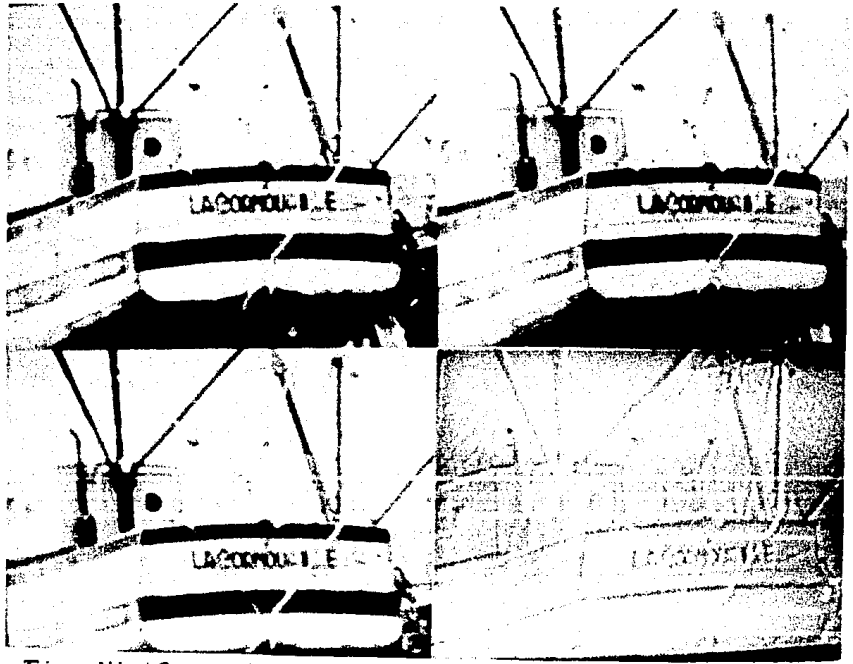


Fig. VI.12 Arriba a la izquierda a) Imagen original. Imágenes procesadas con el algoritmo PSC1: arriba a la derecha b) punto flotante, abajo a la izquierda c) punto fijo y abajo a la derecha d) error de predicción.

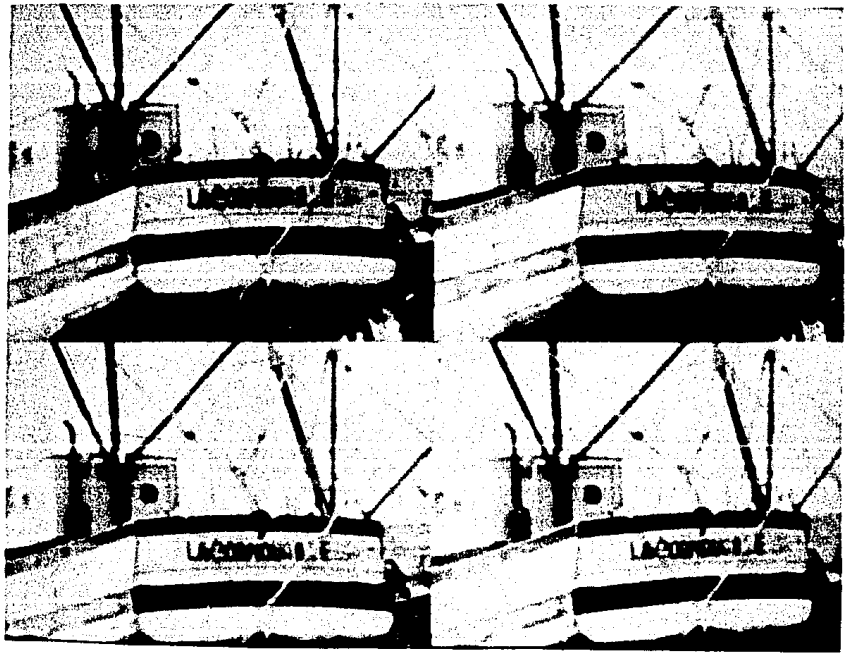


Fig. VI.13 Imágenes procesadas con el algoritmo: arriba a la izquierda a) P5C1, arriba a la derecha b) P5C5, abajo a la izquierda c) P6C1 y abajo a la derecha d) P6C5 .

especializada para llevar a cabo la codificación de fuente en tiempo real.

VI.5.1 Sensibilidad a la orientación de los predictores.

El algoritmo D° sólo utiliza dos predicciones *isotrópicas a las diferentes orientaciones*: $(A + C)/2$ y $(A + D)/2$. El papel de éste último es principalmente mejorar el comportamiento del sistema para las orientaciones en las que $(A + C)/2$ es poco eficaz. La conmutación se realiza tomando como base la determinación de la orientación a partir del estado de la vecindad local al punto que se está codificando. Se observa pues, que la selección de la predicción a partir de un conjunto de predictores muy selectivos en la orientación, llega a perjudicar el comportamiento del error de predicción.

VI.5.2 El problema de la determinación de los puntos contorno.

En el algoritmo de Graham la adaptabilidad del predictor consiste en dos tratamientos distintos: el primero, es la estimación de la orientación con base en el estado de los puntos previamente decodificados; el segundo (que puede ser llamado proyección), consiste en predecir si el punto actual es punto contorno a partir del estado previamente determinado; si lo es, es afectado (proyectado) por una orientación, la cual determina que predictor se adapta mejor.

La estimación del estado es local: si en la vecindad del punto actual, un punto previamente decodificado es afectado por una orientación y está situado en un contorno, entonces el punto actual es declarado contorno y la orientación local estimada se utiliza para seleccionar la predicción. Si las predicciones se seleccionan tomando como base su isotropia a las diferentes orientaciones, el procedimiento descrito incrementa su eficacia sensiblemente.

VI.5.3 Efectos de la cuantificación adaptable.

Los efectos de una cuantificación adaptable, que utiliza una determinación del estado local, a través de una función de actividad de luminancia, permite que se disminuya sustancialmente la dinámica del error de predicción, además, el gradiente de luminancia se hace uniforme sobre toda la imagen del error. Y lo más importante de todo, es que el sistema de cuantificación adaptable puede insertarse en cualquier sistema de predicción adaptable ya que es muy robusto frente al ruido de cuantificación. Así lo demuestra el efecto subjetivo de filtrado pasabajas que se presenta. La única restricción que se impone al esquema de predicción es que sea robusto al ruido de canal.

VI.6 Conclusión.

La optimización completa de un algoritmo con predicción y cuantificación adaptable es difícil de realizar. Existen una gran cantidad de parámetros que manejar y diversos tratamientos que efectuar, para lograr resultados óptimos. Por ejemplo, si se desea obtener una señal ô con un rango dinámico limitado y con una uniformidad adecuada, es probable que se presenten problemas de robustez del sistema frente al ruido de canal. Probablemente, una *estimación recursiva* [10] del estado local en el predictor y en el cuantificador, resulten en un sistema bastante eficaz. El problema relativo de tal sistema es su implantación en tiempo real.

CAPITULO VII

DISEÑO DEL PAQUETE MICD PARA LA SIMULACION DE UN SISTEMA DE MODULACION POR PULSOS CODIFICADOS DIFERENCIAL (DPCM)

CAPITULO VII. DISEÑO DEL PAQUETE MICD PARA LA SIMULACION DE UN SISTEMA DE MODULACION POR PULSOS CODIFICADOS DIFERENCIAL (DPCM).

En el presente capitulo se describe la documentación de un paquete de simulación de codificación de fuente DPCM, denominado MICD. Se incluye una breve descripción de los objetivos, así como el análisis y los requerimientos del mismo. Además, se muestran algunos diagramas de flujo y esquemas de la estructura del paquete y una descripción breve de cada uno de los módulos que lo componen. En el apéndice C se anexan los listados de los programas que están debidamente estructurados y documentados para su fácil comprensión.

Para poder planear en forma adecuada un sistema, es necesario emplear el enfoque sistémico, que es una metodologia para desarrollar programación. Este identifica varias fases que siguen una secuencia temporal, las cuales son: a) definición de requerimientos, b) diseño, c) desarrollo y d) operación y mantenimiento. Cada una de las secciones siguientes, constituye el documento de las etapas correspondientes a las fases del proceso de programación, llevadas a cabo para la realización del sistema MICD .

VII.1 Definición de los requerimientos.

VII.1.1 Planteamiento de los objetivos.

El objetivo general del desarrollo de este sistema será formalizar los programas de simulación de algunas técnicas de codificación DPCM intercampo, utilizadas en el procesamiento de imágenes digitales. La estructura del paquete deberá permitir una interacción muy versátil con el usuario.

El sistema contemplará un conjunto de algoritmos de codificación DPCM intercampo tanto con parámetros fijos como adaptables, para lograr las diversas configuraciones del sistema.

El sistema podrá ejecutarse en cualquier equipo digital de cómputo (PC, XT, AT, PS/XX o compatible), con coprocesador matemático.

El sistema estará estructurado en forma modular para que sea fácil de

entender y para futuras modificaciones o mejoras a éste.

El tiempo de procesamiento en la simulación de cada algoritmo deberá ser el menor posible.

Los datos de salida del sistema deberán permitir al usuario el manejo de ellos, para llevar a cabo posteriormente un análisis objetivo y subjetivo, que permita evaluar el desempeño del algoritmo simulado.

VII.1.2 Análisis.

Las diferentes configuraciones del sistema MICD se formarán por las combinaciones del conjunto de predictores y cuantificadores implantados que se enlistan a continuación.

Conjunto de predictores.

Fijos	Adaptables
a) Cualquier combinacion lineal de una vecindad A, AB, B, C, D, E . (§ V.4.1)	b) Algoritmo de Graham c) Algoritmo de Zschunke-Kretz d) Algoritmo de Zschunke-Dewitte e) Algoritmo D* f) Algoritmo D2 .

Conjunto de cuantificadores.

Fijos	Adaptables
a) de 11 niveles C = 2.2857 b) de 16 niveles C = 2 c) de 17 niveles C = 1.9512	d) por función de enmascaramiento (criterios PQC_111, PQC_112, PQC_113, PQC_114) C = 2.2857 e) de dinámica deslizable

$$C = \frac{8}{\text{No. bits del cuantificador (1 a 8 bits)}}$$

donde:

$$C : \text{razón de compresión} = \frac{\text{No. bits de la imagen original}}{\text{No. bits del cuantificador utilizado}}$$

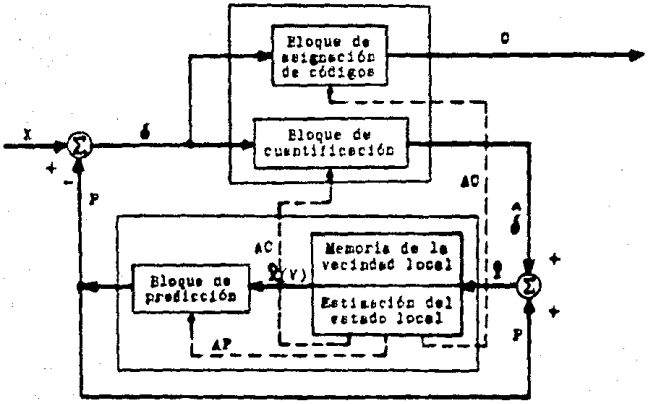
Sólo podrán codificarse imágenes monocromáticas digitalizadas con 8 bits y de un tamaño de 256 x 256 puntos.

Para la descripción de los algoritmos refiérase al capitulo VI .

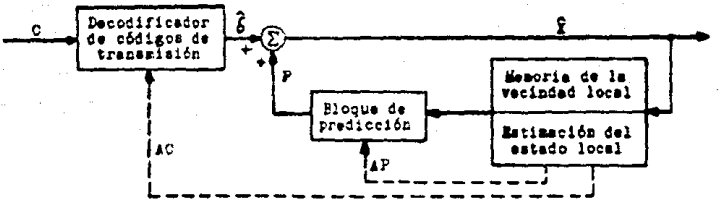
El sistema MICD utilizará argumentos de comandos de linea, para interactuar con otros programas. En particular podrá interactuar con los programas SPI e Y64_2 (apéndice D).

En la figura VII.1 se muestra el diagrama de bloques del sistema MICD (codificador y decodificador), desde el punto de vista funcional. Destacan en el esquema dos bloques susceptibles de ser configurados por el usuario: el bloque de predicción y el bloque de cuantificación. En la figura VII.2 se pueden observar los dos diferentes esquemas de predicción : fija y adaptable, tanto en el codificador como en el decodificador. En la figura VII.3 se pueden identificar los diferentes procesamientos que se llevan a cabo, cuando se calcula el estado local del punto actual (a codificar), en el marco de una predicción o cuantificación adaptable. Se pueden observar también los elementos que se almacenan en la memoria del sistema y que forman la memoria de la vecindad del punto actual a codificar (vector vecino). En la figura VII.4 se pueden observar los diferentes tipos de cuantificadores con que cuenta el sistema: fijos (de memoria cero) y adaptables, uno de dinámica deslizable y otro que utiliza el valor de una función de actividad local de luminancia, como parámetro de adaptación.

SISTEMA DE CODIFICACION MICD

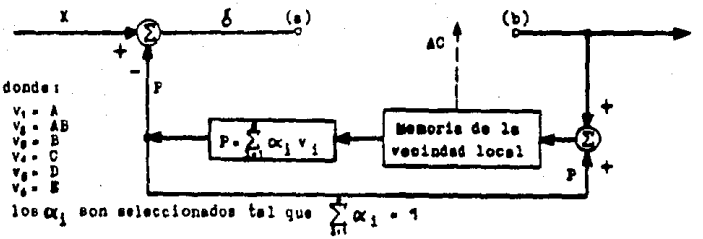


CODIFICADOR

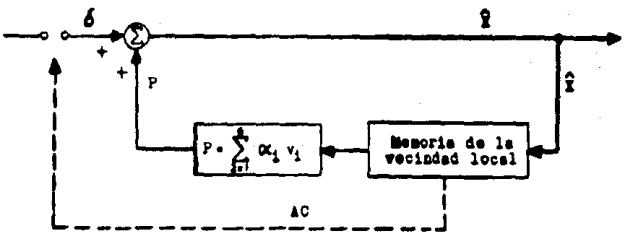


DECODIFICADOR

Fig. VII.1 Diagrama de bloques funcional del sistema MICD .

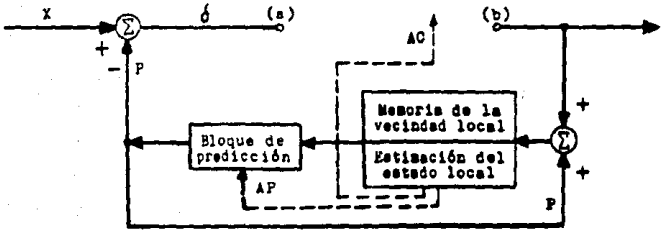


CODIFICADOR CON PREDICTOR FIJO

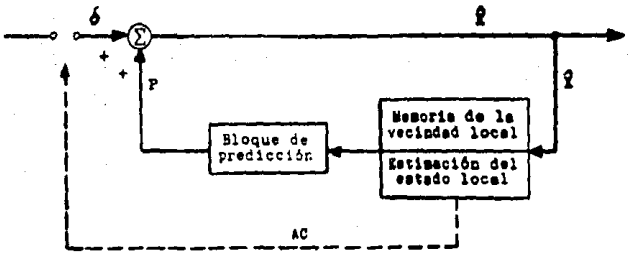


DECODIFICADOR CON PREDICTOR FIJO

Fig. VII.2 Esquemas de predicción. a) Predicción fija en la que sólo se utilizan los elementos de la memoria de la vecindad. Nótese que el bloque que determina el estado local proporciona los parámetros de adaptación para la predicción (AP) y la cuantificación (AC). Cada esquema acepta también, diferentes esquemas de cuantificación (puntos a y b en las figuras).



CODIFICADOR CON PREDICTOR ADAPTABLE



DECODIFICADOR CON PREDICTOR ADAPTABLE

Fig. VII.2 Continuación. Esquemas de predicción. b) Predicción adaptable, en la que además se determina el estado local de la vecindad local. Nótese que el bloque que determina el estado local, proporciona los parámetros de adaptación para la predicción (AP) y la cuantificación (AC). Cada esquema acepta también, diferentes esquemas de cuantificación (puntos a y b en las figuras).

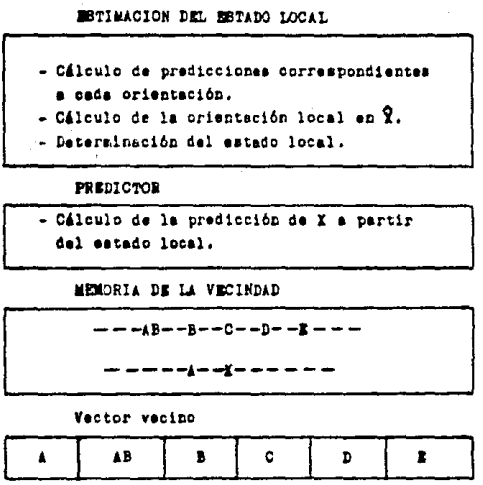


Fig. VII.3 En el primero y segundo cuadros, se muestran las operaciones que se llevan a cabo en la determinación del estado local y el cálculo de la predicción. En el tercero y cuarto, los los elementos que forman la memoria de la vecindad.

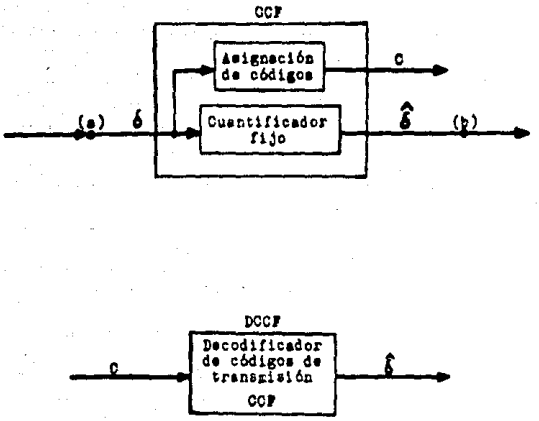


Fig. VII.4 Esquemas de cuantificación. a) Cuantificación fija (de memoria cero). Nótese que los bloques de las figuras incluyen la asignación de códigos a los valores cuantificados. Se muestra la parte en el codificador y su correspondiente en el decodificador.

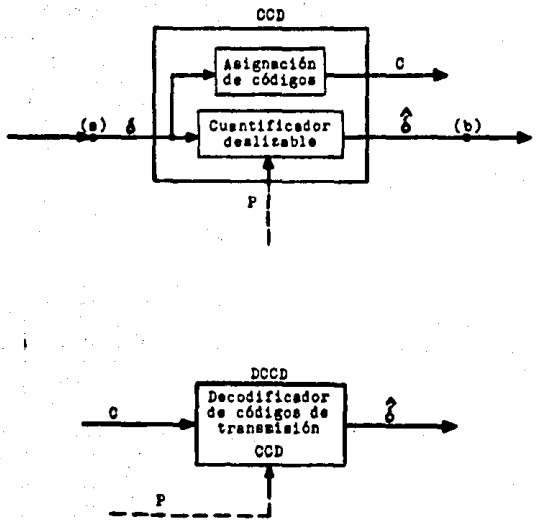


Fig. VII.4 Continuación. Esquemas de cuantificación. b) Cuantificación de dinámica deslizable. El parámetro de adaptación es el valor de la predicción. Nótese que los bloques de las figuras incluyen la asignación de códigos a los valores cuantificados. Se muestra la parte en el codificador y su corrrespondiente en el decodificador.

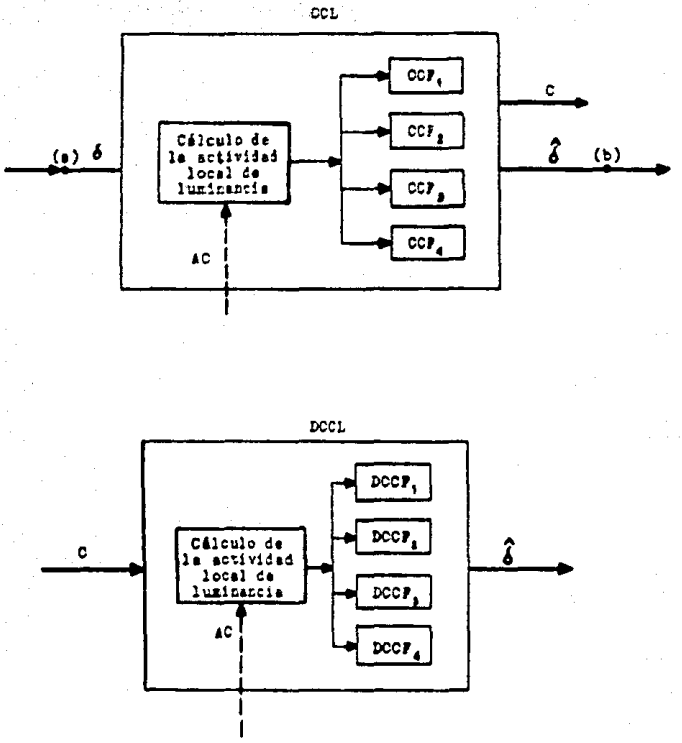


Fig. VII.4 Continuación. Esquemas de cuantificación. c) Cuantificación que utiliza el valor de una función de actividad local de luminancia como parámetro de adaptación. Nótese que los bloques de las figuras incluyen la asignación de códigos a los valores cuantificados. Se muestra la parte en el codificador y su correspondiente en el decodificador.

VII.1.3 Requerimientos.

VII.1.3.1 Activación.

El paquete se podrá ejecutar invocándolo, o por medio del programa SPI .

El programa MICD deberá ser ejecutado desde el sistema operativo DOS (versión 3.0 en adelante). Podrá además aceptar tres comandos de línea. El primer comando de línea es el nombre del programa ejecutable (MICD). Cuando desde el sistema operativo se ejecuta el programa tecleando MICD, el sistema pedirá el nombre de la imagen a procesar y el tipo de procesamiento ("c" para codificar, o "d" para decodificar dicha imagen). El segundo comando de línea es opcional y será el archivo de la imagen a procesar. De este modo al teclear

[MICD *nombre.extensión*]

se ejecutará el programa y solamente se requerirá proporcionar el tipo de procesamiento (codificar o decodificar). Tanto para el primero como para el segundo caso, deberá existir el archivo de configuración del MICD (MICD.CFG), el cual contiene la configuración del MICD . Se podrá ver cual es la configuración, ejecutando el comando *type* del archivo MICD.CDF . El tercer comando de línea (segundo para el MICD) es un '-', con el cual se logrará que se ejecute el proceso y al terminar se invoque a otro programa: el SPI . Así, si se teclea:

[MICD *nombre.extensión* -]

se pedirá la opción a seguir (codificar o decodificar) y al terminar el procesamiento, ejecutará el programa SPI .

VII.1.3.2 Datos de entrada y validación de datos.

El programa MICD pide el nombre del archivo de la imagen a procesar. Este debe proporcionarse con su extensión, ya que el programa verifica la existencia de archivos de lectura. Si existe el archivo continua la ejecución.

Un nombre no válido es:

nombre	*null*

El formato correcto es:

nombre	.	*[ext]*

Al ejecutar el programa MICD utilizando un solo comando de linea, se piden los siguientes datos:

Nombre de la imagen de entrada

Si no existe el archivo, o no se dió el formato correcto, se indicará el siguiente mensaje de error:

"Error al abrir archivo de entrada."

en cuyo caso, se aborta el programa. Luego se pide la opción a ser ejecutada:

Codificar	[c]
Decodificar	[d]

si se teclea alguna letra diferente a 'c', 'C', 'd' o 'D', el programa mandará el mensaje:

"Selección no implantada"

y termina su ejecución. Cuando se da una de las opciones válidas, el programa abre archivos para lectura o para escritura según sea el caso (Fig. VII.5).

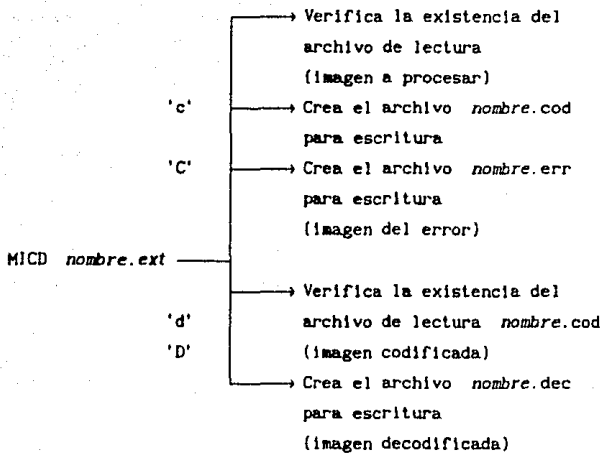


Fig. VII.5 Opciones de ejecución del sistema MICD.

El programa verifica si se lograron abrir los archivos correctamente, si no, mandará cualquiera de los siguientes mensajes:

"Error al abrir archivo de codificacion"
"Error al abrir archivo de error"
"Error al abrir archivo de decodificacion"

A su vez, el programa reserva espacio en memoria RAM para almacenar las imágenes procesadas, que posteriormente serán transferidas a memoria ROM (archivos de escritura *nombre*.cod, *nombre*.err, *nombre*.dec). En el caso de que no haya suficiente memoria RAM para almacenamiento, el programa indicará el siguiente mensaje:

"Error: no se pudo reservar memoria para la imagen"

Cuando termina la ejecución y se dió el tercer comando de línea ('-'),

pero no existe SPI.EXE , aparecerá el mensaje:

"Imposible de ejecutar SPI.EXE"
"Error"

VII.1.3.3 Resolución.

La resolución de los datos de entrada es de 8 bits. En la ejecución del programa se utilizara la representación en punto fijo para valores enteros y en punto flotante para valores reales, de acuerdo a los formatos indicados en la figura VII.6 .

VII.1.3.4 Diagnóstico del problema.

Si los datos de entrada fueron correctos, fue posible reservar memoria para las imágenes y no hubo error durante el proceso, el programa escribirá en los archivos creados (imágenes procesadas), cuyos nombres son:

nombre.cod Imagen codificada
nombre.err Imagen del error de predicción

en el caso de haber seleccionado la opción codificar ('c' o 'C'). Y

nombre.dec Imagen decodificada

si se seleccionó la opción decodificar ('d' o 'D')

Durante el proceso pueden llegar a ocurrir tres problemas y el sistema abortará el proceso. Uno de estos errores sucederá, si el error de predicción llegara a ser mayor que 255, en cuyo caso se indicara el mensaje:

"Error MICD : abs(delta) > 255"

Otro ocurrirá, si se detectara un código que no existe, es decir, si el

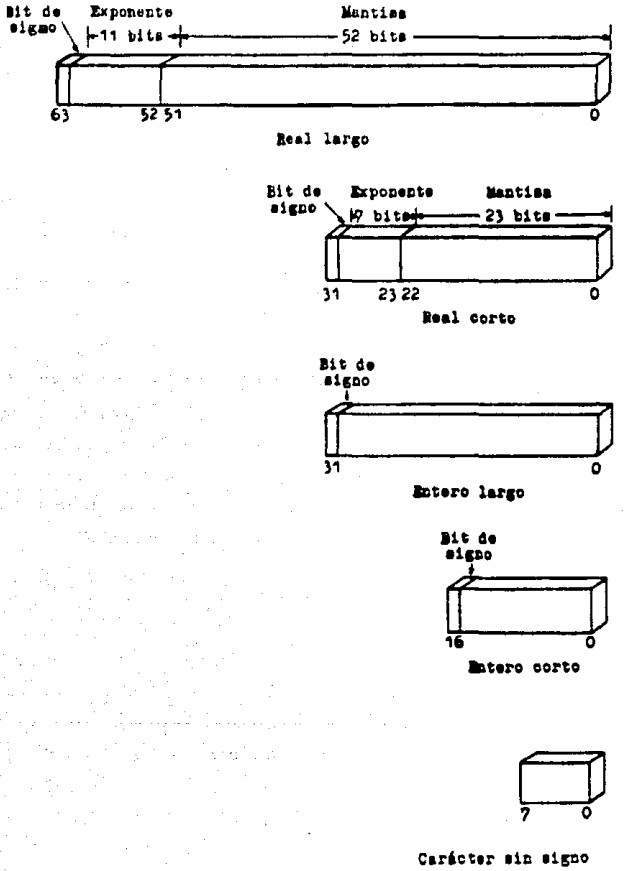


Fig. VII.6 Formato para la representación de datos y variables.

archivo *nombre*.cod , después de ser creado, es modificado accidentalmente, algunos bytes no podrán ser interpretados si su valor excede un cierto numero y y se producirá el error. Esto se debe a que los archivos (imagenes) en el sistema, son leidos en modo binario y los códigos de transmisión son enteros sin signo que estan en el rango de 0 a y . El número y corresponde a los niveles de reconstrucción del cuantificador. El programa abortará mandando antes el siguiente mensaje:

"Error MICD: código de transmision inexistente"

Finalmente el programa será abortado, si se presentara algún sobreflujo. En este caso aparecerá el mensaje:

"Floating point error: Domain"

La utilización del programa MICD se sintetiza en el diagrama de la figura VII.7 . El programa puede ser invocado de tres formas utilizando los formatos mostrados. Una vez que el sistema MICD ha sido invocado, se llevará a cabo una interacción entre el sistema y el usuario. En el caso de que se presente algún inconveniente, la ejecucion será abortada y el error correspondiente aparecerá en el monitor del sistema digital.

VII.2 Diseño.

VII.2.1 Arquitectura.

Dentro del diseño del sistema (la arquitectura), se definen primero los conjuntos de informacion (Fig. VII.8).

a) Información de entrada. El programa MICD, captura los datos de la pantalla (nombre de la imagen de entrada y opción seleccionada) y toma de un archivo (MICD.CFG) la configuracion del mismo. En el caso en que se interactúe con el programa SPI, se conformará la configuración por medio de la selección de las opciones en él.

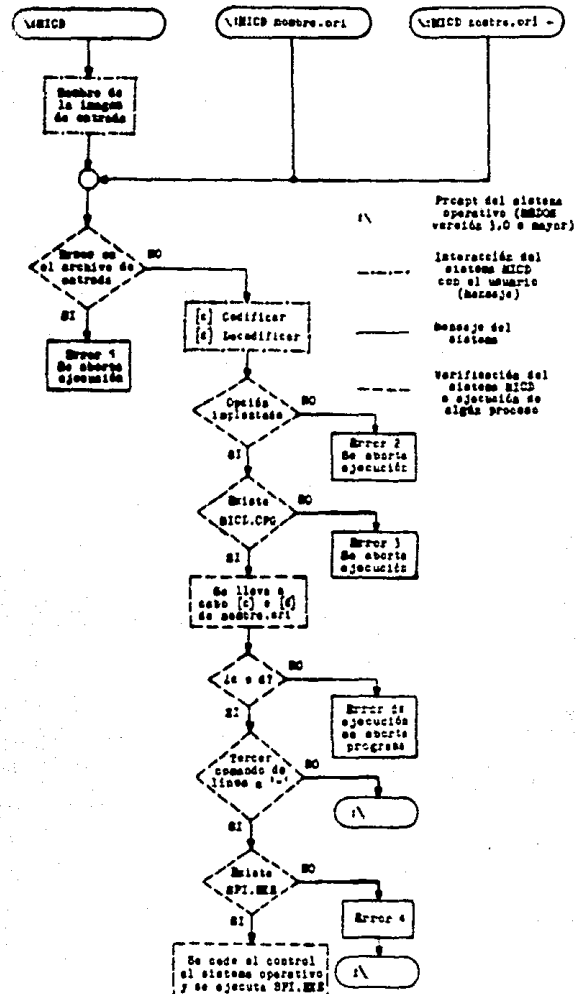


Fig. VII.7 Diagrama que sintetiza la utilización del sistema MICD .

b) Información de salida. El programa MICD presenta en la pantalla los mensajes de error. Como salidas del proceso, se tienen los archivos *imagen*.cod, *imagen*.dec e *imagen*.err en memoria ROM (disco), como datos de salida.

c) Base de datos. El programa MICD toma del archivo MICD.CFG la configuración del sistema.

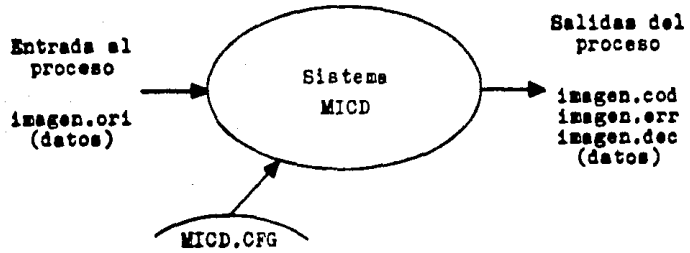


Fig. VII.8 Conjuntos de información del sistema MICD .

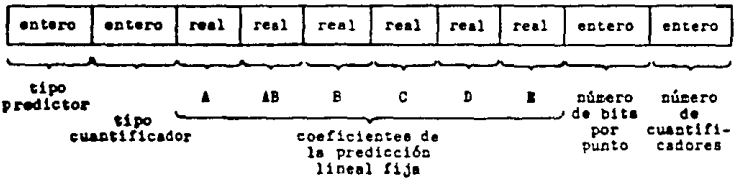


Fig. VII.9 Formato del archivo MICD.CFG .

El archivo MICD.CFG es leído en modo texto. El formato es el mostrado en la figura VII.9 .

Campo 1 Tipo de predictor.

Código	Predicción
48	Fija
49	Algoritmo de Graham
50	Algoritmo de Zschunke-Kretz
51	Algoritmo de Zschunke-Dewitte
52	Algoritmo D*
53	Algoritmo D2

Campo 2 Tipo de cuantificador.

Código	Cuantificación
48	Fija de 11 niveles
49	Fija de 16 niveles
50	Fija de 17 niveles
51	Deslizable
52	Por luminancia

Campos 3-8 Coeficientes del predictor fijo (la suma de éstos no debe exceder el valor 1.0).

Campo 9 Número de bits por punto para el cuantificador deslizable (de 1 a 8).

Campo 10 Número de cuantificadores a utilizar en el cuantificador deslizable (se recomienda usar entre 4 y 10).

Ejemplos:

48	51	0.5	0.0	0.0	0.5	0.0	0.0	4	9

Esta configuración indica que se utilizará una predicción fija (A+C)/2 y el cuantificador deslizable, utilizando 4 bits por punto y 9 cuantificadores fijos.

52	53	*	*	*	*	*	*	*	*

Esta configuración indica que se utilizará el algoritmo D* en la predicción y el cuantificador que utiliza una función de luminancia.

Nota: * no importa.

Para mostrar el flujo de la información del sistema MICD, se ilustra por medio de la figura VII.10 .

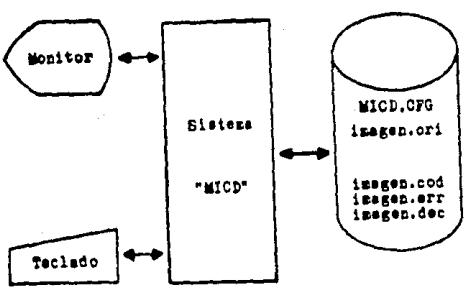


Fig. VII.10 Diagrama de flujo de información del del sistema MICD.

Conforme a los requerimientos del sistema MICD, establecidos en la sección VII.1.3 , el sistema se divide en varios módulos que se encargan de tareas especificas como se muestra en el figura VII.11 .

Las condiciones que afectan al sistema son: el sobreflujo, datos erróneos (alteraciones de la datos en el archivo *imagen*.cod que es un archivo intermedio en el proceso) y que el error de predicción sea mayor que 255

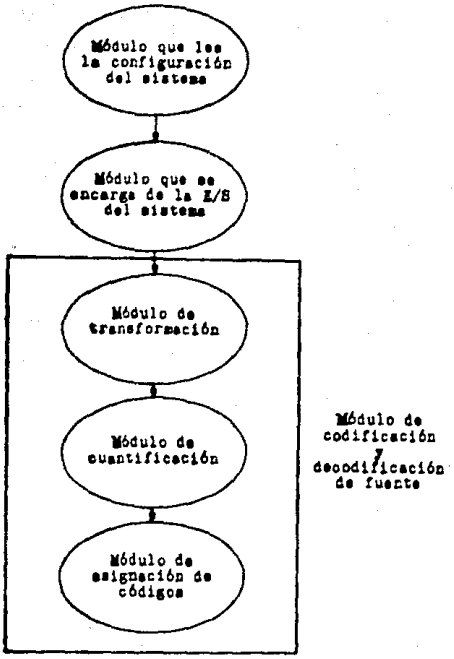


Fig. VII.11 División en módulos del sistema MICD.

VII.2.2 Diagrama de estructura.

En la figura VII.12 se muestra la descomposición del sistema MICD en módulos (nombres de los archivos), de las funciones principales del mismo.

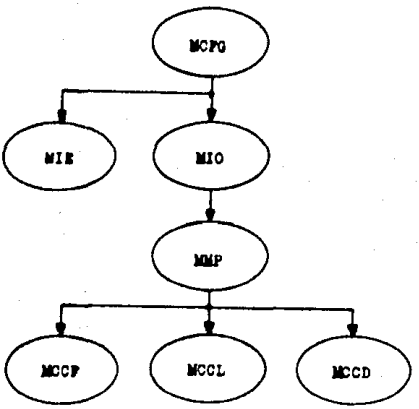


Fig. VII.12 Diagrama de estructura del sistema MICD por módulos (archivos).

El diagrama de estructura del sistema MICD por funciones, se muestra en la figura VI.13, además del módulo o archivo al que pertenece cada función, para conocer como está integrado cada módulo. Las funciones realizan tareas específicas y son idependientes por lo que pueden ser reutilizadas en otros programas.

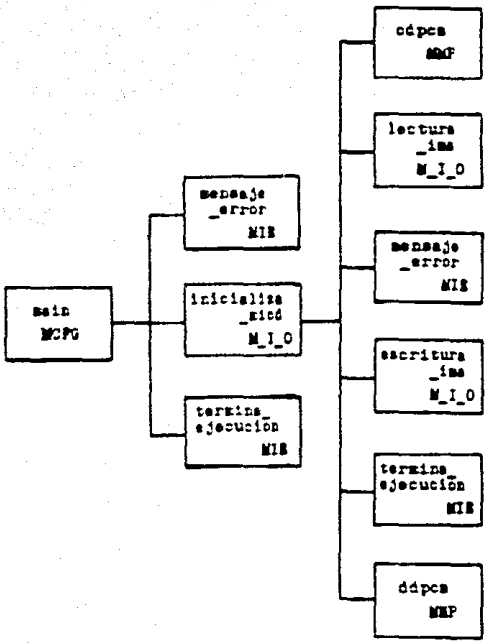


Fig. VII.13 Diagrama de estructura del sistema MICD por funciones (subrutinas). Parte inicial.

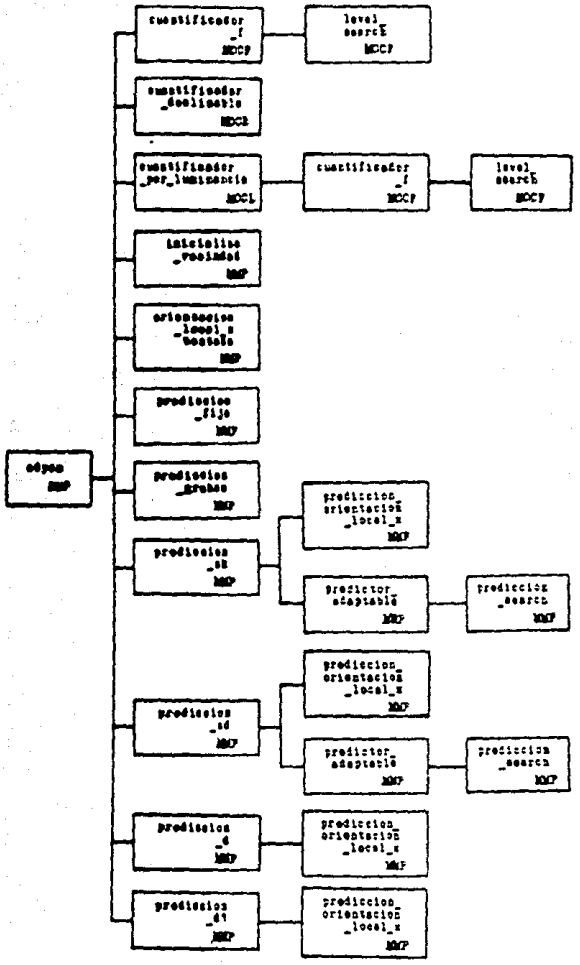


Fig. VII.13 Continuación. Diagrama de estructura del sistema MICD por funciones (subrutinas). Parte del codificador.

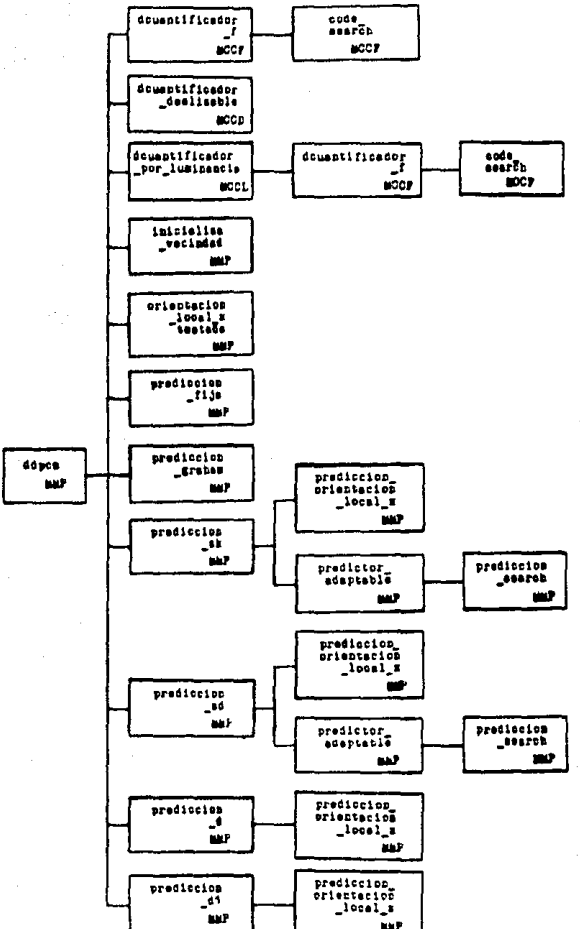


Fig. VII.13 Continuación. Diagrama de estructura del sistema MICD por funciones (subrutinas). Parte del decodificador.

VII.2.3 Detalle de módulos.

Para la descripción de los módulos, se dará una reseña del proceso que realiza cada una de las funciones que integran al sistema MICD, según el diagrama de estructura de la figura VII.13 .

. *main*

Inicio del programa, activa las rutinas con las que comienza la codificación.

. *mensaje_error*

Despliega en la pantalla del monitor, la cadena de caracteres que se le envía como parámetro.

. *termina_ejecucion*

Cede el control al sistema operativo, indicándole si ha de ejecutar o no el programa SPI.

. *inicializa_micd*

Función que se encarga de invocar a las rutinas de simulación de la codificación de fuente.

. *cdpcm*

Simula la codificación de fuente.

. *ddpcm*

Simula la decodificación de fuente.

. *lectura_ima*

Se encarga de leer la imagen a procesar del disco duro y carga los datos en RAM.

. *escritura_ima*

Escribe la imagen procesada, de la RAM a un archivo en disco duro.

. *cuantificador_f*

Permite implantar la cuantificación en un sistema de codificación predictiva de fuente, utilizando parámetros fijos.

. *cuantificador_deslizable*

Permite implantar la cuantificación en un sistema de codificación predictiva de fuente, utilizando como parámetro de adaptación la predicción.

. *cuantificador_por_luminancia*

Permite implantar la cuantificación en un sistema de codificación predictiva de fuente, utilizando como parámetro de adaptación el valor de una función de actividad de luminancia.

. *prediccion_fija*

Permite implantar el bloque de predición en un sistema de codificación predictiva de fuente, utilizando parámetros fijos en la ejecución.

. *prediccion_graham*

Permite implantar el bloque de predición en un sistema de codificación predictiva de fuente, utilizando parámetros adaptables atendiendo a las características del algoritmo de Graham.

. *prediccion_zk*

Permite implantar el bloque de predición en un sistema de codificación predictiva de fuente, utilizando parámetros adaptables atendiendo a las características del algoritmo de Zschunke-Kretz.

. *prediccion_zd*

Permite implantar el bloque de predición en un sistema de codificación predictiva de fuente, utilizando parámetros adaptables atendiendo a las características del algoritmo de Zschunke-Dewitte.

. *prediccion_d*

Permite implantar el bloque de predición en un sistema de codificación predictiva de fuente, utilizando parámetros adaptables atendiendo a las características del algoritmo d° .

. *prediccion_d1*

Permite implantar el bloque de predición en un sistema de codificación predictiva de fuente, utilizando parámetros adaptables atendiendo a las características del algoritmo d2 .

. *inicializa_vecindad*

Permite a las funciones *cdpcm* y *ddpcm*, inicializar el vector que contiene los puntos vecinos al punto actual, a decodificar y a decodificar respectivamente.

. *orientacion_local_x_testada*

Permite calcular la orientación que corresponde al punto actual decodificado.

. *level_search*

Permite determinar el intervalo al que corresponde un valor que se va a cuantificar.

. *prediccion_orientacion_local_x*

Permite obtener, a partir del estado local de la vecindad del punto actual a decodificar, la orientación de la predicción más conveniente.

. *predictor_adaptable*

Determina a partir del valor que entrega la función denominada *prediccion_orientacion_local_x* , la predicción del punto actual a decodificar.

. *prediccion_search*

Búsqueda binaria que utiliza la función *predictor_adaptable*, para encontrar que predicción corresponde al valor del ángulo que entrega la función *prediccion_orientacion_local_x* .

. *dcuantificador_f*

Permite la decodificación a partir del código generado por la función *cuantificador_f* .

. *dcuantificador_deslizable*

Permite la decodificación a partir del código generado por la función *cuantificador_deslizable* .

. *dcuantificador_por_luminancia*

Permite la decodificación a partir del código generado por la función *cuantificador_por_luminancia* .

. *code_search*

Búsqueda binaria que permite determinar el nivel de reconstrucción, correspondiente al código que se le pasa como argumento.

A continuación se presentan los detalles de algunas de las funciones que se pueden modificar con el objeto de optimizar y acrecentar la capacidad del sistema MICD, para simular la codificación predictiva.

PROPOSITO Utilizar la función *cdpcm*, para simular uno de 30 diferentes esquemas de codificación DPCM intercampo.

SINTAXIS

int *cdpcm*(unsigned char far **buffer_e_s*,
unsigned char far **buffer_s*)

buffer_e_s Apuntador al arreglo en el que se almacena la imagen a codificar. Cada punto de la imagen se se representa con ocho bits como máximo.

buffer_s Arreglo en el que se almacena el error de predicción de la imagen.

EJEMPLO DE USO

cdpcm(&*imagen_entrada_salida*,
&*imagen_de_error*)

INCLUIR MGD.H , MGL.H , MCFG.H , MMP.H , MCCF.H , MCCL.H , MCCD.H .

DESCRIPCION La funcion *cdpcm* recibe un arreglo de dimensión MAX_PIXELES, que corresponde a la imagen a codificar. El resultado de la codificación (codigos) se almacena en el parámetro *buffer_e_s* . La función se encuentra definida en el archivo MMP.C . Es capaz de simular 30 diferentes esquemas de codificación. Para elegir la pareja predictor-cuantificador, se utilizan las variables *dpcm_cfg.tipo_pred* y *dpcm_cfg.tipo_cuan* , definidas en MCFG.C . Se encuentran implantados seis predictores y cinco cuantificadores diferentes.

REGRESA La función regresa un valor diferente de cero si hubo algún error en la ejecución.

COMENTARIOS La función utiliza las variables globales definidas en el archivo MMP.C .

MANTENIMIENTO En la subrutina se encuentran tres estructuras "switch", dos de ellas permiten seleccionar el tipo de cuantificador a usar y la otra el tipo de predictor. Las variables que se utilizan en las sentencias son: *dpcm_cfg.tipo_pred* y *dpcm_cfg.tipo_cuan* . De este modo para agregar un predictor o un cuantificador, sólo se deben alterar estas sentencias. Consúltese el archivo MMP.H , para verificar las constantes de prueba en las sentencias "switch".

FUNCIONES RELACIONADAS

ddpcm	Función complementaria.
cuantificador_f	Bloque de cuantificación y codificación.
cuantificador_por_luminancia	Bloque de cuantificación y codificación.
cuantificador_deslizable	Bloque de cuantificación y codificación.
prediccion_d	Para bloque de predicción.
prediccion_dl	Para bloque de predicción.
prediccion_fija	Para bloque de predicción.
prediccion_graham	Para bloque de predicción.
prediccion_zd	Para bloque de predicción.
prediccion_zk	Para bloque de predicción.

PROPOSITO Utilizar la función *ddpcm* para simular la decodificación de fuente DPCM intercampo, de los códigos generados por la función *cdpcm* .

SINTAXIS

int *ddpcm* (unsigned char far **buffer_e_s*)

buffer_e_s Arreglo en el que se almacenan los códigos de transmisión correspondientes a la imagen codificada mediante la función *cdpcm* .

EJEMPLO DE USO

ddpcm(*&imagen_entrada_salida*)

INCLUIR MGD.H , MGL.H , MCFG.H , MMP.H , MCCF.H , MCCL.H , MCCD.H .

DESCRIPCION La función *ddpcm* es capaz de simular la decodificación de diversos algoritmos DPCM intercampo. Una vez que finaliza la ejecución de la función, la imagen decodificada se encuentra en el parámetro de la función. Complementa el uso de la función *cdpcm* .

REGRESA La función regresa un valor diferente de cero si se presentó algún error en la ejecución.

COMENTARIOS La función utiliza las variables globales definidas en el archivo MMP.C .

MANTENIMIENTO En la subrutina se encuentran tres estructuras "switch", dos de ellas permiten seleccionar el tipo de cuantificador a usar y la otra el tipo de predictor. Las variables que se

utilizan en las sentencias son: *dpcm_cfg.tipo_pred* y *dpcm_cfg.tipo_ cuan* . De este modo para agregar un predictor o un cuantificador, sólo se deben alterar estas sentencias. Consúltese el archivo MMP.H , para verificar las constantes de prueba en las sentencias "switch".

FUNCIONES RELACIONADAS

cdpcm	Función complementaria.
cuantificador_f	Bloque de cuantificación y codificación.
cuantificador_por_luminancia	Bloque de cuantificación y codificación.
cuantificador_deslizable	Bloque de cuantificación y codificación.
prediccion_d	Para bloque de predicción.
prediccion_d1	Para bloque de predicción.
prediccion_fija	Para bloque de predicción.
prediccion_graham	Para bloque de predicción.
prediccion_zd	Para bloque de predicción.
prediccion_zk	Para bloque de predicción.

PROPOSITO Utilizar la función *cuantificador_deslizable* , para obtener un valor cuantificado de acuerdo a un algoritmo adaptable, que utiliza las propiedades estadisticas de la fuente de información de donde se obtiene el valor a cuantificar. Adicionalmente se obtiene la palabra de código correspondiente al valor cuantificado.

SINTAXIS

```
void cuantificador_deslizable (float          p,
                               float          delta,
                               float          *delta_testada,
                               unsigned char  *codigo)
```

p	Predicción del punto actual a codificar.
delta	Variable que almacena el valor a cuantificar.
**delta_testada*	Apuntador a la localidad en donde se almacena el valor cuantificado.
**codigo*	Apuntador a la localidad en donde se almacena el código correspondiente al valor cuantificado.

EJEMPLO DE USO

```
cuantificador_deslizable (prediccion,
                          valor_a_cuantificar,
                          &valor_cuantificado,
                          &codigo)
```

INCLUIR MGL.H, MGD.H, MCFG.H Y MCCD.H .

DESCRIPCION La función cuantificador deslizable, utiliza el valor del argumento *p* y los valores de las variables *numbits* y *numcuan* , declaradas externas en MCCD.H y definidas en MCFG.C , para calcular el nivel de reconstrucción correspondiente al valor del argumento *delta* . El resultado se almacena en la variable **delta_testada* .

USO COMUN La función se utiliza para formar el bloque de cuantificación y asignación de códigos en un sistema de codificación DPCM.

COMENTARIOS La función utiliza las variables globales del archivo MCCD.C .

FUNCIONES RELACIONADAS

dcuantificador_deslizable	Función complementaria.
cuantificador_f	Para cuantificación fija.
cuantificador_por_luminancia	Cuantificación adaptable que utiliza una función de actividad de luminancia.

PROPOSITO Utilizar la función *dcuantificador_deslizable* , para obtener un valor cuantificado a partir del código generado por la función *cuantificador_deslizable* .

SINTAXIS

void *dcuantificador_deslizable* (float *p*,
float **delta_testada*,
unsigned char **codigo*)

p	Predicción del punto actual a codificar.
**delta_testada*	Apuntador a la localidad en donde se almacena el valor cuantificado.
**codigo*	Apuntador a la localidad en donde se almacena el código correspondiente al valor cuantificado.

EJEMPLO DE USO

dcuantificador_deslizable (*prediccion*,
&*valor_cuantificado*,
codigo)

INCLUIR MGL.H, MGD.H, MCFG.H y MCCD.H .

DESCRIPCION La función *dcuantificador_deslizable* , utiliza el valor del argumento *p* y los valores de las variables *numbits* y *numcuan*, declaradas externas en MCCD.H y definidas en MCFG.C , para calcular el nivel de reconstrucción correspondiente al valor de la variable código.

USO COMUN La función *dcuantificador_deslizable* , complementa el uso de la función *cuantificador_deslizable* .

COMENTARIOS La función utiliza las variables globales del archivo MCCD.C .

FUNCIONES RELACIONADAS

cuantificador_deslizable	Función complementaria.
cuantificador_fijo	Para cuantificación fija.
cuantificador_por_luminancia	Cuantificación adaptable, que utiliza una función de actividad de luminancia.

PROPOSITO Utilizar la función *cuantificador_f* , para obtener un valor cuantificado de acuerdo a los niveles de decisión y niveles de reconstrucción especificados en una estructura tipo *CUANTIFICADOR_F*, definida en MCCF.C . Adicionalmente se obtiene la palabra de código, correspondiente al valor cuantificado.

SINTAXIS

```
void cuantificador_f (CUANTIFICADOR_F   *ptr_cuantificador,
                      float                        *delta,
                      float                  *delta_testada,
                      float                        *codigo)
```

**ptr_cuantificador*	Apuntador a la estructura tipo *CUANTIFICADOR_F* , en la que se almacenan las características del cuantificador fijo a utilizar.
**delta*	Apuntador a la localidad en donde se almacena el valor a cuantificar.
**delta_testada*	Apuntador a la localidad en donde se almacena el valor cuantificado.
**codigo*	Apuntador a la localidad en donde se almacena el código correspondiente al valor cuantificado.

EJEMPLO DE USO

```
cuantificador_f (&cuanfijo,
                 &valor_a_cuantificar,
                 &valor_cuantificado,
                 &codigo_de_transmision)
```

INCLUIR MGD.H, MCCF.H Y MGL.H .

DESCRIPCION La función *cuantificador_f* , recibe la dirección del valor correspondiente a delta, que será cuantificado de acuerdo a los parámetros especificados en la estructura *CUANTIFICADOR_F*. Utiliza una búsqueda binaria, para encontrar el intervalo que corresponde al valor a cuantificar y así obtener el valor de reconstrucción y su código correspondiente.

USO COMUN Implantar cuantificadores uniformes o no uniformes, midtread o midriser, de memoria cero. Se utiliza como bloque de cuantificación y bloque de asignación de códigos.

COMENTARIOS La función cuenta con siete cuantificadores, los cuales pueden ser utilizados para cualquier aplicación; éstos se encuentran declarados como estructuras externas en MCCF.H . La definición de esta función está localizada en MCCF.C .

MANTENIMIENTO Si se desea agregar un cuantificador fijo, defínase en MCCF.C y declárese como una estructura externa en MCCF.H, atendiendo a las características de la estructura *CUANTIFICADOR_F* . El número máximo de niveles de reconstrucción es *MAX_NIVELES*, constante que se encuentra definida en MCCF.H .

FUNCIONES RELACIONADAS

dcuantificador_f	Función complementaria.
cuantificador_deslizable	Para cuantificación que se adapta a la fuente de información, utilizando una dinámica deslizable.
cuantificador_por_luminancia	Cuantificación adaptable, que utiliza una función de actividad de luminancia.

MENSAJES DE ERROR

Si el valor a cuantificar excede el valor *MAX_PIXELES* - 1, la ejecución del programa será abortada.

PROPOSITO Utilizar la función *dcuantificador_f* , para obtener un valor cuantificado a partir del código generado por la función *cuantificador_f* .

SINTAXIS

void *dcuantificador_f* (*CUANTIFICADOR_F* **ptr_cuantificador*,
float **delta_testada*,
unsigned char *codigo*)

**ptr_cuantificador*	Apuntador a la estructura tipo *CUANTIFICADOR_F* , que almacena las características del cuantificador fijo.
**delta_testada*	Apuntador a la localidad en donde se almacena el valor cuantificado, correspondiente al argumento *codigo* .
codigo	Localidad que almacena el código generado por la función *cuantificador_f* .

EJEMPLO DE USO

```
d_cuantificador_f(&cuanfijo,
                  &valor_cuantificado,
                  &codigo_de_transmision);
```

INCLUIR MGD.H , MGL.H , MCCF.H .

DESCRIPCION La función *d_cuantificador_f* , recibe un valor que corresponde al código asignado a un valor cuantificado por la función *cuantificador_f* , de acuerdo a los parámetros especificados en la estructura *CUANTIFICADOR_F* . Utiliza una búsqueda binaria, para encontrar el valor cuantificado a partir del valor del argumento código.

USO COMUN Esta función complementa la utilización de la función *cuantificador_f* , para implantar cuantificadores fijos.

COMENTARIOS En el archivo MCCF.C se encuentra la definición de esta función.

MANTENIMIENTO Ver función *cuantificador_f* .

FUNCIONES RELACIONADAS

cuantificador_f Función complementaria.

MENSAJES DE ERROR

Si el valor de la variable código no es encontrado en la búsqueda binaria, será abortada la ejecución del programa.

PROPOSITO Utilizar la función *cuantificador_por_luminancia* , para obtener un valor cuantificado de acuerdo a un algoritmo de cuantificación adaptable, que utiliza el valor de una función de actividad local de luminancia como parámetro de adaptación. Adicionalmente se obtiene la palabra de código correspondiente al valor cuantificado.

SINTAXIS

```
int cuantificador_por_luminancia(float          p,
                                 float          *delta,
                                 float          *delta_testada,
                                 float          *val_actual,
                                 float          *val_precedente,
                                 int            posicion,
                                 unsigned char  *codigo)
```

p	Predicción del punto actual a codificar.
**delta*	Apuntador a la localidad en donde se almacena el valor a cuantificar.
**delta_testada*	Apuntador a la localidad en donde se almacena el valor cuantificado.
**val_actual*	Apuntador a un arreglo que almacena la linea actual de recorrido.
**val_precedente*	Apuntador a un arreglo que almacena la linea precedente de recorrido.
posicion	Variable en la que se almacena la posición actual en el arreglo *val_actual* .
**codigo*	Apuntador a la localidad en donde se almacena el código correspondiente al valor cuantificado.

EJEMPLO DE USO

```
cuantificador_por_luminancia(prediccion,
                             &valor_a_cuantificar,
                             &valor_cuantificado,
                             &linea_actual,
                             &linea_precedente,
                             posicion,
                             &codigo);
```

INCLUIR MGD.H , MCCL.H , MGL.H .

DESCRIPCION La función *cuantificador_por_luminancia* , recibe la dirección de un valor real que será cuantificado con base en un criterio de adaptación, que consiste en calcular la actividad local de luminancia en la vecindad del punto a codificar. (Fig. VII.14)

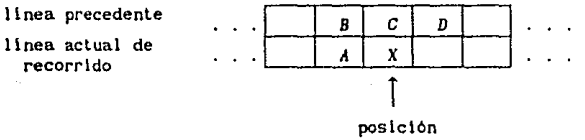


Fig. VII.14 Vecindad del punto actual.

REGRESA La función regresa un valor diferente de cero, si hubo algún error de ejecución.

USO COMUN El uso de las funciones está restringido a la codificación de imágenes monocromáticas en un esquema predictivo. Se utiliza como bloque de cuantificación y bloque de asignación de códigos.

COMENTARIOS La función utiliza un algoritmo de adaptación, diseñado específicamente para ocultar a la vista los errores inherentes a la presencia de un cuantificador, en un sistema predictivo de codificación. La función debe contar con la vecindad local del punto actual a codificar. La definición de la función *cuantificador_por_luminancia* se encuentra en el archivo MCCL.C .

FUNCIONES RELACIONADAS

dcuantificador_por_luminancia	Función complementaria.
cuantificador_f	Para cuantificación fija.

PROPOSITO Utilizar la función *dcuantificador_por_luminancia* para obtener un valor cuantificado a partir del código generado por la función *cuantificador_por_luminancia*.

SINTAXIS

```
int dcuantificador_por_luminancia(float          p,
                                  float      *delta_testada,
                                  float        *val_actual,
                                  float     *val_precedente,
                                  int            posicion,
                                  unsigned char    *codigo)
```

p	Predicción del valor a decodificar.
**delta*	Apuntador a la localidad en donde se almacena el valor cuantificado.
**delta_testada*	Apuntador a la localidad en donde se almacena el valor cuantificado.
**val_actual*	Apuntador a un arreglo que almacena la línea actual de recorrido.
**val_precedente*	Apuntador a un arreglo que almacena la línea precedente de recorrido.
posicion	Variable en la que se almacena la posición actual en el arreglo *val_actual* .
**codigo*	Variable que almacena el código que corresponde al valor cuantificado deseado.

EJEMPLO DE USO

```
dcuantificador_por_luminancia(prediccion,
                              &valor_cuantificado,
                              &linea_actual,
                              &linea_precedente,
                              posicion,
                              &codigo);
```

INCLUIR MGD.H , MGL.H , MCCL.H .

DESCRIPCION La función *dcuantificador_por_luminancia* , obtiene a partir del argumento *codigo*, el valor generado a partir de la función *cuantificador_por_luminancia* . En la obtención del valor cuantificado a partir del código, se utiliza un criterio de adaptación que consiste en calcular la actividad local de luminancia, en la vecindad del punto a decodificar (Fig. VII.9) .

REGRESA La función regresa un valor diferente de cero si hubo algún error de ejecución.

USO COMUN El uso de esta función está restringido a la codificación de imágenes monocromáticas, en un esquema predictivo. Constituye la función complementaria a la función *cuantificador_por_luminancia*. La definición de la función se encuentra en el archivo MCCL.C .

FUNCIONES RELACIONADAS

cuantificador_por_luminancia Función complementaria.

prediccion_graham
prediccion_zk
prediccion_zd
prediccion_d
prediccion_dl

PROPOSITO El conjunto de funciones *prediccion* , permiten implantar esquemas adaptables en una codificación DPCM de imágenes monocromáticas. Las rutinas determinan la predicción más conveniente para el punto actual a codificar, a partir del estado de la vecindad local de dicho punto (Fig. VII.15). En el argumento **p* se almacena el valor de predicción.

AB	*B*	*C*	*D*	*E*
	A	*X*		

Fig. VII.15 Vecindad local del punto X .

SINTAXIS

void *prediccion_graham*(float **p*);
void *prediccion_zk*(float **p*);
void *prediccion_zd*(float **p*);
void *prediccion_d*(float **p*);
void *prediccion_dl*(float **p*);

**p* Apuntador a la localidad en donde se almacena la predicción.

EJEMPLO DE USO

prediccion_graham(&*prediccion*)
prediccion_zk(&*prediccion*)
prediccion_zd(&*prediccion*)
prediccion_d(&*prediccion*)
prediccion_dl(&*prediccion*)

INCLUIR MMP.H .

DESCRIPCION Las funciones *prediccion* , calculan la predicción del punto actual a codificar, a partir del estado de la vecindad local en ese punto. Para ello utilizan la función *prediccion_orientacion_local_x* . La variable *orientacion* de la función, indica qué predicción es la mejor. La definición de las funciones se puede encontrar en el archivo MMP.C .

COMENTARIOS Cuando la función *prediccion* utiliza más de dos valores diferentes, correspondientes a la variable *orientacion* , la selección de la predicción se lleva a cabo mediante búsqueda binaria. Para la utilización de estas funciones se requiere utilizar el conjunto de variables globales del archivo MMP.C .

FUNCIONES RELACIONADAS

prediccion_fija Para la predicción fija.

PROPOSITO Utilizar la función *prediccion_fija* para implantar un predictor lineal fijo, en un esquema de codificación DPCM intercampo. En el parámetro **p* se almacena la predicción.

SINTAXIS void *prediccion_fija*(float **p*)

**p* Apuntador a la localidad en donde se almacena el valor de la predicción.

EJEMPLO DE USO

prediccion_fija(&*prediccion*)

INCLUIR MMP.H .

DESCRIPCION La función *prediccion_fija* , permite obtener una combinanación lineal de los puntos, en la vecindad local del punto actual a codificar. La combinación lineal constituye la predicción de este punto (ecuación VII.1) .

$$*p = \sum_{i=1}^{6} \alpha_i \beta_i \qquad \text{(VII.1)}$$

donde los coeficientes α_i pueden ser modificados por el usuario. Los β_i , corresponden a los elementos de la vecindad local del punto a codificar.

USO COMUN La función se puede usar para implantar el bloque de predicción en una codificación DPCM intercampo.

COMENTARIO Los coeficientes α_i corresponden al conjunto de variables { *A, AB, B, C, D, E* }. Se encuentran declaradas externas en MCFG.H y definidas en MCFG.C .

FUNCIONES RELACIONADAS

prediccion_graham	Para predicción adaptable.
prediccion_zk	Para predicción adaptable.
prediccion_zd	Para predicción adaptable.
prediccion_d	Para predicción adaptable.
prediccion_d1	Para predicción adaptable.

VII.3 Desarrollo.

VII.3.1 Codificación de módulos.

Para la codificación de los módulos se eligió el lenguaje de programación C (Turbo C versión 2.0), por las razones siguientes:

a) Velocidad.
b) Versatilidad en el manejo de memoria.
c) Variedad en las funciones implantadas.

El lenguaje de programación C, permite utilizar eficientemente una gran variedad de operaciones con apuntadores a localidades de memoria. Este hecho hace que un programa codificado en lenguaje C, sea más eficiente en el aprovechamiento del hardware. La eficiencia se traduce, por un lado, en el incremento de la velocidad en la ejecución de un programa y por otro lado en una transferencia expedita de información en la memoria.

Por otra parte, el compilador de Turbo C del lenguaje C tiene dos importantes características adicionales:

a) Un ambiente de programación muy cómodo para el usuario (que incluye depurador).
b) Una gran cantidad de funciones matemáticas, de graficación de control de procesos, de manejo de memoria y manejo de periféricos.

Todas las características señaladas, lo hacen adecuado para los fines que se persiguen en este trabajo.

En el apéndice C se anexa la codificación, en este lenguaje, de los módulos del sistema MICD . Se incluyen los encabezados de los módulos .C (archivos con extensión .H) .

VII.3.2 Pruebas de alto nivel.

En los requerimientos presentados en la sección VII.1.3 , se establecen las condiciones del sistema MICD , tanto para la activación, la validación de los datos de entrada y salida, así como los errores posibles en el proceso.

Para encontrar errores de análisis, de especificación y diseño en el ambiente real (equipo, sistema operativo, interfaz, etc.), se realizarán pruebas de alto nivel.

La dinámica del error de predicción, para una gran variedad de algoritmos de codificación DPCM intercampo, ha sido ampliamente estudiada. En particular en el estudio llevado a cabo por Kretz, se muestran una serie de imágenes que corresponden al error de predicción de diversos algoritmos de codificación DPCM .

Las pruebas de alto nivel consistirán entonces, en comparar los resultados (imágenes de error) obtenidos con el sistema MICD y los resultados obtenidos por Kretz [10]. Imágenes similares, indicarán el adecuado comportamiento del sistema.

Adicionalmente, la dinámica de los errores de transmisión en las imágenes procesadas mediante diferentes algoritmos de codificación DPCM, servirán como parámetro para comprobar el funcionamiento del sistema.

CAPITULO VIII

IMPLANTACION DE UN ALGORITMO DE CODIFICACION DE FUENTE DPCM EN UNA ARQUITECTURA BASADA EN EL MICROPROCESADOR TMS320C25

CAPITULO VIII. IMPLANTACION DE UN ALGORITMO DE CODIFICACION DE FUENTE DPCM EN UNA ARQUITECTURA BASADA EN EL MICROPROCESADOR TMS320C25.

VIII.1 Preliminar.

Una vez llevada a cabo la simulación de los distintos algoritmos y de haber realizado diferentes pruebas para la elección de uno de ellos en base a su mejor desempeño, se procedió a realizar su traducción del lenguaje de alto nivel, al lenguaje ensamblador del microprocesador TMS320C25. La meta fundamental que se buscó con dicha tarea, fué la de estudiar la velocidad de ejecución del algoritmo, al implantarlo en una arquitectura basada en un microprocesador especialmente diseñado para el procesamiento digital de señales, tal y como lo es el TMS320C25.

El presente capitulo, desarrolla en primera instancia una presentación en forma resumida de las principales caracteristicas del microprocesador TMS320C25, haciendo énfasis en su arquitectura, modos de direccionamiento, configuración de mapa de memoria y en general de todo aquello que se consideró importante para la fácil comprensión del algoritmo una vez implantado en el lenguaje ensamblador del microprocesador.

Posteriormente, se presenta la versión del algoritmo en el lenguaje ensamblador del TMS320C25, haciéndose una descripción general por bloques del sistema, así como del tipo de configuración y formato empleados para el manejo de los datos. Finalmente, se hace un resumen de los resultados obtenidos al procesar una imagen con la versión del algoritmo en lenguaje ensamblador. Tanto desde el punto de vista cualitativo, tomando como referencia la calidad de la imagen procesada, como desde el punto de vista de la velocidad de procesamiento, éste último de especial interés si se piensa en una posible implantación en tiempo real.

VIII.1.1 Descripción general del microprocesador TMS320C25.

El procesador digital de señales TMS320C25, es miembro de la familia de muy alta escala de integración TMS320 de procesadores digitales de señales y periféricos. La familia TMS320 soporta en tiempo real procesamiento digital de señales (PDS) y aplicaciones de intensa computación en áreas tales como : telecomunicaciones, modems, procesamiento de voz, procesamiento de gráficas, análisis espectral, procesamiento de audio, filtrado digital, controladores de alta velocidad, instrumentación y procesamiento numérico.

La familia TMS320 está formada por cinco distintos procesadores (TMS32010, TMS320C10, TMS32011,TMS32020,TMS320C25). El primero de ellos, es el TMS32010, siendo éste una microcomputadora con una arquitectura tipo Harvard de 32 bits y una interfaz externa de 16 bits, capaz de ejecutar cinco millones de instrucciones por segundo. El siguiente procesador de la familia es el TMS32020, el cual tiene una arquitectura basada en la del TMS32010, sin embargo, algunos cambios empleados para mejorar dicha arquitectura permiten generar un menor costo del sistema; además de aumentar en razón de dos a tres veces la capacidad de posibles aplicaciones de procesamiento digital de señales con respecto al TMS32010.

El TMS320C25 es una versión CMOS compatible de el TMS32020, con un ciclo de tiempo por instrucción más rápido y la inclusión adicional de ventajas tanto en hardware como en software. El código objeto del TMS320C25 es totalmente compatible con el del TMS32020.

Algunas de las características más importantes del TMS320C25 son las siguientes :

-Ciclo de tiempo por instrucción : 100ns.
-Tecnología CMOS de bajo consumo de potencia.
-4K palabras de 16 bits de memoria ROM "on-chip" enmascarable.
-8 registros auxiliares con una unidad aritmética dedicada.
-Un puerto serie doblemente "buffereado".
-Accesso directo a memoria (DMA) concurrente, usando una operación entrelazada.
-Modo de direccionamiento de "bit-reverse" para radix-2FFT's.

-Aritmética de precisión extendida y soporte para filtros adaptables.

-Operaciones de alta velocidad para intercambio de datos en memoria externa.

-Bit de acarreo en el acumulador e instrucciones relacionadas.

Existen dos versiones disponibles del TMS320C25 para el desarrollo de los requerimientos en las diferentes aplicaciones, siendo éstas:

a) versión con ciclo de tiempo por instrucción de 100ns.

b) versión con ciclo de tiempo por instrucción de 125ns.

La versión indicada en el inciso a), permite realizar un mayor número de instrucciones por ciclo, de tal forma que el procesador ejecuta 10 millones de instrucciones por segundo (10 MIPS). Con esto, las posibles aplicaciones del TMS320C25 en PDS se ven incrementadas al contar con instrucciones de multiplicación/acumulación, realizadas en un simple ciclo y con opción al manejo o movimiento de datos. Además cuenta con ocho registros auxiliares, con una unidad aritmética dedicada, un conjunto de instrucciones que permite el desarrollo de filtros adaptables y aritmética de precisión extendida. Cuenta también con un direccionamiento del tipo "bit-reverse" y un veloz sistema de comunicación de entrada/salida, el cual es muy útil para el procesamiento intenso de datos.

En la tabla VIII.1 se muestran algunas aplicaciones típicas.

Tabla VIII.1 Aplicaciones típicas dadas a la familia TMS320.

PROPOSITO GENERAL DSP	GRAFICAS/IMAGENES	INSTRUMENTACION
FILTRADO DIGITAL	ROTACION 3-D	ANALISIS ESPECTRAL
CONVOLUCION	VISION ROBOTICA	GENERACION DE FUNCIONES
CORRELACION	COMPRESION / TRANSMISION DE IMAGENES	IGUALACION DE PATRONES
TRANSFORMADA DE HILBERT	RECONOCIMIENTO DE PATRONES	ANALISIS TRANSITORIO
TRANSFORMADA RAPIDA DE FOURIER	RESTAURACION DE IMAGENES	FILTRADO DIGITAL
FILTRADO ADAPTIVO	PROCESAMIENTO HOMOMORFICO	PLL
VENTANEO	ESTACIONES DE TRABAJO	
GENERACION DE SEÑALES	ANIMACION/MAPEO DIGITAL	

VOZ	CONTROL	MILITAR
CORREO DE VOZ	CONTROL DE DISCOS	SEGURIDAD EN COMUNICACIONES
RECONOCIMIENTO DE PALABRAS	CONTROL DE SERVOS	PROCESAMIENTO DE RADAR
VERIFICACION DE VOZ	CONTROL DE ROBOTS	PROCESAMIENTO DE SONAR
RESTAURACION DE PALABRAS	CONTROL DE IMPRESORAS LASER	PROCESAMIENTO DE IMAGEN
SINTESIS DE PALABRAS	CONTROL DE MAQUINAS	GUIA PARA MISILES
TEXTO HABLADO	CONTROL DE MOTORES	MODEMS DE RADIO FRECUENCIA

TELECOMUNICACIONES		AUTOMOTORES
CANCELACION DEL ECO	FAX	CONTROL DE MAQUINAS
TRANSCODERS ADPCM	TELEFONIA CELULAR	ANALISIS DE VIBRACIONES
PBXs DIGITALES	TELEFONIA PARLANTE	CONTROL DE MANEJO ADAPTIVO
REPETIDORES DE LINEA	LENGUAJE DIGITAL	POSICIONAMIENTO GLOBAL
MULTIPLEXADO DE CANAL	INTERPOLACION (DSI)	NAVEGACION
MODEMS DE 1200 A 19200 bps	PAQUETE X 25 DE CONMUTACION	COMANDO POR VOZ
ECUALIZADORES ADAPTIVOS	VIDEOCONFERENCIA	RADIO DIGITAL
DTMF CODIFICACION/DECODIFICACION	ENCRIPTACION DE DATOS	TELEFONIA CELULAR

DE CONSUMO	INDUSTRIA	MEDICINA
DETECTORES DE RADAR	ROBOTS	AYUDA AUDITIVA
HERRAMIENTAS DE POTENCIA	CONTROL NUMERICO	MONITOREO DE PACIENTES
AUDIO DIGITAL / TV	ACCESOS DE SEGURIDAD	EQUIPO DE ULTRASONIDO
SINTETIZADORES MUSICALES	MONITOREO DE LINEAS DE POTENCIA	UTILERIAS PARA DIAGNOSTICO
JUGUETES EDUCATIVOS		MONITOREO FETAL
		PROTESIS

VIII.1.2 Arquitectura del TMS320C25.

El TMS320C25 es un procesador digital de señales de alto desempeño, que cuenta con un acumulador único . Su arquitectura es del tipo Harvard, en la cual la memoria de programa y la memoria de datos residen en espacios de direcciones separados. Esto permite una completa superposición de las etapas de reconocimiento y de ejecución de las instruccines. También cuenta con instrucciones que permiten la transferencia de datos entre los dos espacios de memoria. Externamente, los espacios de memoria de programa y de datos, están multiplexados a través del mismo canal, del tal forma que se maximiza el rango de direcciones para ambos espacios, en tanto que se minimiza el número de terminales del dispositivo. Internamente, la arquitectura del procesador maximiza la potencia de procesamiento al mantener dos estructuras de canal independientes , una para datos y otra para programa; de tal suerte que el resultado es una ejecución a gran velocidad. La flexibilidad del sistema se ve incrementada al contar con dos grandes bloques de memoria RAM del tipo "on-chip", uno de los cuales puede ser configurado como memoria de datos o como memoria de programa de acuerdo a las necesidades de la aplicación.

VIII.1.3 Diagrama funcional por bloques.

El diagrama funcional del TMS320C25 que se muestra en la fig. VIII.1; permite observar sus principales bloques constitutivos, así como las diferentes vías de acceso para la transferencia de datos dentro del procesador. Este diagrama, también muestra las terminales con que cuenta el procesador para establecer una interfaz en forma completa.

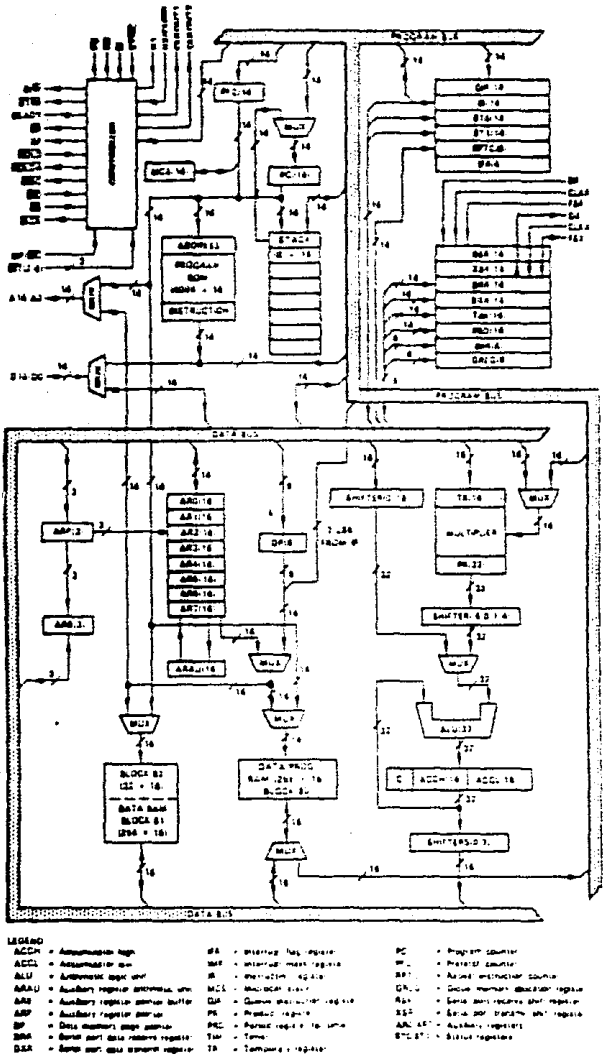


Figura VIII.1 Diagrama de bloques del microprocesador TMS320C25.

Puede observarse, que la arquitectura del TMS320C25 está construida alrededor de dos canales principales : el canal de programa y el canal de datos. El canal de programa conduce tanto a los códigos de instrucción, como a los operandos inmediatos desde la memoria de programa. El canal de datos interconecta a varios elementos, tales como la unidad aritmética lógica central y la fila de registros auxiliares; la RAM de datos y la memoria interna o externa de programa con el multiplicador, esto se realiza en un ciclo simple para operaciones de multiplicación/acumulación.

El TMS320C25 tiene un alto grado de paralelismo, por ejemplo, mientras los datos están siendo operados por la CALU (Unidad Aritmetica Logica Central), las operaciones aritméticas pueden ser desarrolladas en la Unidad Aritmética Auxiliar de Registros (ARAU). Por otro lado, el paralelismo se ve incrementado como resultado de un potente conjunto de operaciones aritméticas, lógicas y de manipulación de bits, todas ellas para ser ejecutadas en un ciclo simple de máquina.

El TMS320C25 está provisto de una fila de ocho registros auxiliares (AR0-AR7), los cuales pueden ser usados para el direccionamiento indirecto de la memoria de datos, o para el almacenamiento de datos temporales. Estos registros, pueden ser direccionados a su vez en forma directa por alguna instrucción, o indirectamente por el Apuntador de Registro Auxiliar (ARP). Tanto los registros auxiliares como el ARP pueden ser cargados ya sea por algún dato de memoria o por algún operando inmediato, definido por una instrucción determinada. El contenido de estos registros puede ser almacenado dentro de la memoria de datos.

La fila de registros auxiliares está conectada a la Unidad Aritmética de Registros Auxiliares (ARAU). Esta unidad, puede autoincrementar al registro auxiliar instantaneo, mientras la localidad de memoria de datos está siendo direccionada. Este autoincremento puede ser realizado por uno de dos caminos: ya sea sumandole o restándole un uno a la ARAU, o sumándole o restándole el contenido del registro AR0.

La ARAU es muy útil para la manipulación de direcciones en paralelo con otras operaciones, de tal forma que puede ser usada como una unidad aritmética general adicional, permitiendo que la fila de registros auxiliares pueda comunicarse directamente con la memoria de datos. La ARAU, desarrolla

una aritmética sin signo de 16 bits, en tanto que, la CALU (Unidad Aritmética Lógica Central), permite una aritmética de 32 bits en complemento a dos. Distintas operaciones de saltos pueden llevarse a cabo con ayuda de los registros auxiliares, dependiendo del resultado que se tenga entre la comparación del registro ARO, con el registro auxiliar intantaneo apuntado por el apuntador ARP.

El microprocesador contiene un contador de programa (PC) de 16 bits, un contador de pre-reconocimiento de instrucciones (PFC) de 16 bits, un registro de "micro-llamado" a la pila (MCS) y ocho niveles de la pila en hardware, para el almacenamiento del PC. El contador de programa, contiene la dirección de la instrucción que se está ejecutando y el contador de pre-reconocimiento de instruciones es usado para el reconocimiento o lectura de las instrucciones. Los ocho niveles de la pila son usados durante la ejecución de interrupciones o subrutinas y el micro-llamado a la pila se usa para almacenar el contenido del contador PFC durante el desarrollo de instrucciones tales como BLKD/BLKP, MAC/MACD y TBLR/TBLW.

Se cuenta también con operaciones de "PUSH" y "POP" para necesidades de subrutinas o interrupciones, permitiendo construir dentro de la memoria de datos una pila de más de ocho niveles. Estas instrucciones, permiten almacenar datos en el tope de la pila dentro de la memoria de datos, o cargar a la pila en el acumulador.

La interfaz local de memoria del TMS320C25, consiste en un canal de datos de 16 bits en paralelo (D15-D0), un canal de direcciones de programa de 16 bits (A15-A0), tres terminales ($\overline{DS}$, $\overline{PS}$ e $\overline{IS}$) para seleccionar los distintos espacios de memoria (datos, programa e I/O respectivamente) y varias señales de control.

La señal R/$\overline{W}$ controla la dirección del flujo de datos y $\overline{STRB}$ proporciona una señal de tiempo para controlar la transferencia.

El uso de la señal READY permite la generación de ciclos de espera para la comunicación con memorias de acceso lento.

El TMS320C25 permite el direccionamiento directo de memoria (DMA) hacia su memoria externa de programa y de datos, mediante el uso de las señales $\overline{HOLD}$ y $\overline{HOLDA}$. De esta forma, otro procesador puede tomar el control completo de la memoria externa del TMS320C25 y desarrollar de esta manera un

procesamiento en paralelo.

La unidad aritmética lógica central (CALU) consta de: un registro escalador por corrimientos de 16 bits, un multiplicador paralelo de 16 X 16 bits, una unidad aritmética lógica (ALU) de 32 bits, un acumulador de 32 bits y algunas operaciones adicionales de escalamiento disponibles tanto a la salida del acumulador como del multiplicador.

Una de las entradas a la ALU se proporciona siempre desde el acumulador y la otra puede ser transferida desde el registro de producto (PR) del multiplicador, o bien por el escalador por corrimientos. Este último, puede a su vez ser cargado desde la memoria de datos.

El escalador por corrimientos tiene una entrada de 16 bits conectada al canal de datos y una salida de 32 bits conectada a la ALU. El escalador produce corrimientos a la izquierda de 0 a 16 bits, de acuerdo a lo programado por la instrucción. Los bits menos significativos a su salida son llenados con ceros y los bits más significativos pueden ser llenados ya sea con ceros o con el signo extendido.

La ALU en combinación con el acumulador desarrolla un amplio rango de instrucciones lógicas y aritméticas, la mayoría de las cuales pueden ejecutarse en un solo ciclo de reloj. A través de las instrucciones de SOVM y ROVM, puede programarse el modo de operación de saturación "overflow" si se desea; de tal forma que cuando en el acumulador se presente un estado de saturación, una bandera se enciende en nivel alto para indicar dicho estado.

El acumulador está dividido internamente en 2 segmentos de 16 bits, para almacenar datos en memoria (ACCH acumulador parte alta y ACCL acumulador parte baja). Por otro lado, se encuentran disponibles corrimientos a la izquierda de 0 a 7 bits, a la salida del acumulador para realizar operaciones de escalamiento y más aún, existen operaciones de corrimiento de un solo bit, ya sea a la izquierda o a la derecha y operaciones de rotación.

El multiplicador construido en hardware de (16 X 16 bits), es capaz de calcular un producto de 32 bits durante cada ciclo de máquina. Dos registros se encuentran asociados a el :

-El registro temporal de 16 bits (TR), el cual contiene uno de los operandos para el multiplicador.

-El registro de producto (PR), el cual contiene el resultado del

producto.

La salida del registro de producto puede ser recorrida a la izquierda de uno a cuatro bits, lo cual resulta útil para la construcción de una aritmética fraccionaria. La salida del PR puede también ser recorrida a la derecha en 6 bits, para habilitar así la ejecución de 128 multiplicaciones acumuladas, sin tener condición de "overflow".

El diseño del TMS320C25 incluye una función de repetición, la cual permite desarrollar una misma instrucción hasta 256 veces en forma consecutiva. Esta función, puede ser usada con instrucciones tales como multiplicaciones/acumulativas, movimientos en bloque, transferencia de datos entre puertos de entrada salida (I/O), lectura y escritura de tablas, etc. De esta manera, este tipo de instrucciones que normalmente son ejecutadas en varios ciclos de máquina, al hacer uso de la función de repetición pueden ser ejecutadas en un solo ciclo.

También cuenta con tres terminales de interrupción enmascarables para el usuario ($\overline{INT2}$-$\overline{INT0}$). Dichas terminales pueden ser usadas para interrumpir al procesador desde dispositivos externos. Las interrupciones internas son generadas en una de 3 formas : por el puerto serie, por el temporizador, o por la instrucción de interrupción. Las interrupciones son atendidas de acuerdo a su prioridad, teniendo el "reset" la mayor prioridad y el puerto serie la menor prioridad al transmitir.

Las condiciones y modos de operación son almacenados en dos registros de estado ST0 y ST1. Distintas instrucciones permiten almacenar y cargar los registros de estado en y desde la memoria de datos.

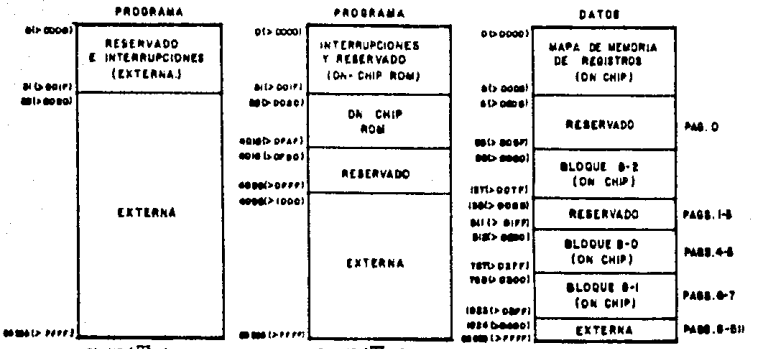
El espacio de direcciones designado por el procesador para operaciones de entrada/salida (I/O), consta de 16 puertos de salida y 16 puertos de entrada. Estos puertos proveén una interfaz muy completa de 16 bits en paralelo vía el canal de datos. Por otra parte, un puerto serie del tipo "on-chip" permite una comunicación directa con dispositivos tipo serie como lo son : codificadores-decodificadores (codecs), convertidores A/D y otros sistemas más.

VIII.1.4 Mapa de memoria.

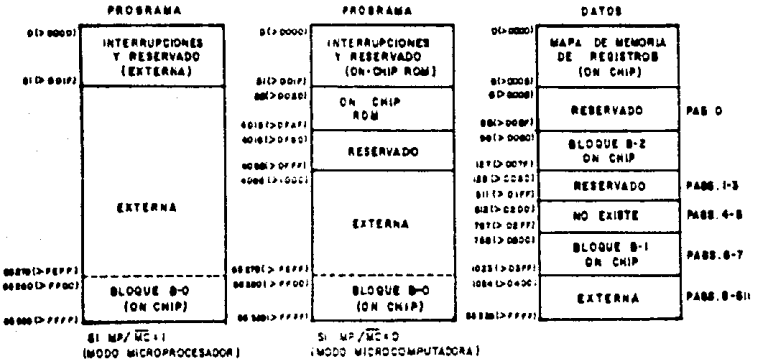
El TMS320C25 proveé un total de 544 palabras de 16 bits de memoria RAM tipo "on-chip", la cual se encuentra dividida en tres bloques (B0, B1, B2). De las 544 palabras , 256 (bloque B0) pueden ser configuradas ya sea como memoria de datos o como memoria de programa a través de las instrucciones CNFD o CNFP ; las restantes 288 palabras (bloques B1 y B2) son siempre memoria de datos. El microprocesador proveé además, un espacio de 64K palabras de memoria de datos del tipo "off-chip", para ser direccionado directamente .

Un segundo bloque de 64K palabras de memoria para programa también puede ser direccionado por el microprocesador, en el cual los programas pueden ser ejecutados a velocidad completa si se cuenta con memorias de suficiente rapidez o con ciclos de espera insertados para memorias de acceso lento. Como se mencionó antes, el bloque B0 puede ser usado como memoria de programa . De esta forma, pequeños bloques de la memoria de programa de tiempo de procesamiento crítico, se pueden almacenar y ejecutar a máxima velocidad.

Se cuenta con tres espacios separados de direcciones : para memoria de programa, para memoria de datos y para operaciones I/O. En adición a los bloques B0, B1 y B2, el mapa de memoria mostrado en la fig. VIII.2, incluye las localidades destinadas a los registros auxiliares, así como también las localidades reservadas. Seis registros periféricos incluyendo los registros de puerto serie , registros de tiempo, el registro de período, el registro de máscara de interrupción y el registro de memoria global, han sido igualmente localizados dentro del espacio de memoria de datos para su fácil modificación. Finalmente, cabe mencionar que las localidades reservadas no pueden ser usadas para almacenar datos y sus contenidos son indefinidos para operaciones de lectura.



(a) MAPA DE MEMORIA DESPUES DE UNA INSTRUCCION CNFD



(b) MAPA DE MEMORIA DESPUES DE UNA INSTRUCCION CNFP

Figura VIII.2 Mapa de memoria del microprocesador TMS320C25.

VIII.1.5 Modos de direccionamiento e instrucciones.

El conjunto de instrucciones del microprocesador cuenta con tres modos de direccionamiento disponibles: directo, indirecto e inmediato. Los modos de direccionamiento directo e indirecto pueden ser usados para accesar la memoria de datos. Cuando se usa el direccionamiento directo, siete bits de la palabra de instrucción son concatenados con los nueve bits del apuntador de página de la memoria de datos (DP), para formar la palabra de 16 bits de la dirección deseada. Cada página tiene una longitud de 128 palabras y existe un total de 512 páginas disponibles, lo cual completa un espacio total de 64K palabras, de memoria de datos directamente direccionables. El direccionamiento directo puede ser usado con todas las instrucciones excepto: CALL, instrucciones de salto, instrucciones con operandos inmediatos, e instrucciones sin operandos.

Un direccionemiento indirecto flexible y potente es proporcionado por los ocho registros auxiliares (AR0-AR7). La dirección del dato que será usado en una instrucción, debe ser puesta dentro de uno de los ocho registros auxiliares. Para seleccionar un registro auxiliar, se carga el apuntador de registro auxiliar (ARP) con un valor entre 0 y 7, el cual designará a un registro de AR0 a AR7 respectivamente.

Exiten disponibles siete tipos de direccionamiento indirecto : indirecto con incremento o decremento, indirecto con adición o substracción del contenido de AR0, indirecto con adición o substracción del contenido de AR0 y con propagación de acarreo revertida (para FFTs) y finalmente indirecto sin modificación, todo esto se muestra en la tabla VIII.2 . Todas las operaciones de autodireccionamiento son desarrolladas en el registro auxiliar instantaneo en el mismo ciclo de máquina de la instrucción original, con una opcion a un nuevo valor de ARP.

Tabla VIII.2 Modos de direccionamiento del microprocesador TMS320C25.

MODO DE DIRECCIONAMIENTO	OPERACION
OP A	Direccionamiento directo.
OP *(,NARP)	Indirecto, AR no cambia.
OP *+(,NARP)	Indirecto, el valor instantaneo de AR se incrementa.
OP *-(,NARP)	Indirecto, el valor instantaneo de AR decrementa.
OP *0+(,NARP)	Indirecto, AR0 es adicionado al AR instantaneo.
OP *0-(,NARP)	Indirecto, AR0 es substraido al AR instantaneo.
OP *BR0+(,NARP)	Indirecto, AR0 es adicionado al AR instantaneo (con propagación de acarreo revertida).
OP *BR0-(,NARP)	Indirecto, AR0 es substraido al AR instantaneo (con propagación de acarreo revertida).

NOTA : NARP especifica un nuevo valor para ARP

En el modo de direccionamiento inmediato, el código de la instrucción contiene el valor del operando inmediato. El microprocesador tiene dos tipos de instrucciones con direccionamiento inmediato : para palabras de valores constantes simples de 8 y 13 bits. También cuenta con dos tipos de instrucciones para constantes largas de 16 bits. En el caso de instrucciones inmediatas para constantes largas, la palabra siguiente al código de instrucción es usada como el operando inmediato. Cabe mencionar que dentro del conjunto de instrucciones existe un total de 17 instrucciones con operandos inmediatos.

Instrucciones.

A continuación se presenta el grupo de tablas que resume al conjunto de instrucciones. En primer lugar, la tabla VIII.3 resume las abreviaciones y símbolos usados en las instrucciones , las cuales se encuentran contenidas en la tabla VIII.4, de acuerdo a la función que realizan y en orden alfabético.

Para finalizar con esta breve explicación acerca del microprocesador TMS320C25. Debe mencionarse que ésta, de ninguna manera ha pretendido ser exhaustiva, el principal objetivo que se ha buscado con ella es el de proporcionar una introducción tanto sobre su arquitectura como de su forma de operación; para con ello, facilitar la comprensión del algoritmo DPCM (elegido con anterioridad) una vez implantado en el microprocesador y cuya versión se procederá a describir a continuación.

Tabla VIII.3 Simbolos para las instrucciones.

SYMBOL	MEANING
ACC	Accumulator
ARB	Auxiliary register pointer buffer
ARn	Auxiliary Register n (AR0 through AR7 are predefined assembler symbols equal to 0 through 7, respectively.)
ARP	Auxiliary register pointer
BIO	Branch control input
C	Carry bit
CM	2-bit field specifying compare mode
CNF	On-chip RAM configuration control bit
dma	Data memory address
DP	Data page pointer
FO	Format status bit
FSM	Frame synchronization mode bit
HM	Hold mode bit
INTM	Interrupt mode flag bit
>nn	Indicates nn is a hexadecimal number. (All others are assumed to be decimal values.)
OV	Overflow flag bit
OVM	Overflow mode bit
P	Product register
PA	Port address. (PA0 through PA15 are predefined assembler symbols equal to 0 through 15, respectively.)
PC	Program counter
PM	2-bit field specifying P register output shift code
pma	Program memory address
Preg	Product register
RPTC	Repeat counter
STn	Status Register n (ST0 or ST1)
SXM	Sign-extension mode bit
T	Temporary register
TC	Test control bit
TOS	Top of stack
Treg	Temporary register
TXM	Transmit mode bit
Usgn	Unsigned value
XF	XF pin status bit
←	Is assigned to
| |	An absolute value
[ ]	Optional items
( )	Contents of

Tabla VIII.4 Instrucciones del microprocesador TMS320C25.

ACCUMULATOR MEMORY REFERENCE INSTRUCTIONS			
MNEMONIC	DESCRIPTION	NO WORDS	OPERATION
ABS	Absolute value of accumulator	1	|(ACC)| → ACC
ADD	Add to accumulator with shift	1	(ACC) + (dma) × $2^{(shift)}$ → ACC
ADDC†	Add to accumulator with carry	1	(ACC) + (dma) + (C) → ACC
ADDH	Add to high accumulator	1	(ACC) + (dma) × 2^{16} → ACC
ADDK†	Add to accumulator short immediate	1	(ACC) + 8-bit constant → ACC
ADDS	Add to low accumulator with sign extension suppressed	1	(ACC) + (dma) → ACC
ADDT†	Add to accumulator with shift specified by T register	1	(ACC) + (dma) × $2^{(T reg)}$ → ACC
ADLK†	Add to accumulator long immediate with shift	2	(ACC) + (16 bit constant × $2^{(shift)}$ → ACC
AND	AND with accumulator	1	(ACC(15-0)).AND.(dma) → ACC(15-0), 0 → ACC(31-16)
ANDK†	AND immediate with accumulator with shift	2	(ACC(30-0)).AND.(16-bit constant × $2^{(shift)}$ → ACC(30-0), 0 → ACC(30-0)
CMPL†	Complement accumulator	1	$\overline{(ACC)}$ → ACC
LAC	Load accumulator with shift	1	(dma) × $2^{(shift)}$ → ACC
LACK	Load accumulator immediate short	1	8 bit constant → ACC
LACT†	Load accumulator with shift specified by T register	1	(dma) × $2^{(T reg)}$ → ACC
LALK†	Load accumulator long immediate with shift	2	(16 bit constant) × 2^{16} → ACC
NEG†	Negate accumulator	1	-(ACC) → ACC
NORM†	Normalize contents of accumulator	1	
OR	OR with accumulator	1	(ACC(15-0)).OR. (dma) → ACC(15-0)
ORK†	OR immediate with accumulator with shift	2	(ACC(30-0)).OR.(16-bit constant × $2^{(shift)}$ → ACC(30-0)
ROL†	Rotate accumulator left	1	(ACC(30-0)) → ACC(31-1), (C) → ACC(0), (ACC(31)) → C
ROR†	Rotate accumulator right	1	(ACC(31-1)) → ACC(30-0), (C) → ACC(31), (ACC(0)) → C
SACH	Store high accumulator with shift	1	((ACC) × $2^{(shift)}$ → dma
SACL	Store low accumulator with shift	1	((ACCL) × $2^{(shift)}$ → dma
SBLK†	Subtract from accumulator long immediate with shift	2	(ACC) - (16 bit constant × $2^{(shift)}$ → ACC
SFL†	Shift accumulator left	1	(ACC(30-0)) → ACC(31-1), 0 → ACC(0)
SFR†	Shift accumulator right	1	(ACC(31-1)) → ACC(30-0), (ACC(31)) → ACC(31)
SUB	Subtract from accumulator with shift	1	(ACC) - (dma) × $2^{(shift)}$ → ACC
SUBB†	Subtract from accumulator with borrow	1	(ACC) - (dma) - $(\overline{C})$ → ACC
SUBC	Conditional subtract	1	
SUBH	Subtract from high accumulator	1	(ACC) - (dma) × 2^{16} → ACC
SUBK†	Subtract from accumulator short immediate	1	(ACC) - 8 bit constant → ACC
SUBS	Subtract from low accumulator with sign extension suppressed	1	(ACC) - (dma) → ACC
SUBT†	Subtract from accumulator with shift specified by T register	1	(ACC) - (dma) × $2^{(T reg)}$ → ACC
XOR	Exclusive-OR with accumulator	1	(ACC(15-0)).XOR.(dma) → ACC(15-0)
XORK†	Exclusive-OR immediate with accumulator with shift	2	(ACC(30-0)).XOR.(16-bit constant × $2^{(shift)}$ → ACC(30-0)
ZAC	Zero accumulator	1	0 → ACC
ZALH	Zero low accumulator and load high accumulator	1	(dma) × 2^{16} → ACC
ZALR†	Zero low accumulator and load high accumulator with rounding	1	(dma) × 2^{16} + >8000→ ACC
ZALS	Zero accumulator and load low accumulator with sign extension suppressed	1	(dma) → ACCL, 0 → ACCH

Tabla VIII.4 Instrucciones del microprocesador TMS320C25 (continuación).

AUXILIARY REGISTERS AND DATA PAGE POINTER INSTRUCTIONS			
MNEMONIC	DESCRIPTION	NO WORDS	OPERATION
ADRK[1]	Add to auxiliary register short immediate	1	(ARn) + 8-bit constant → ARn
CMPR[1]	Compare auxiliary register with auxiliary register AR0	1	If ARn (CM) AR0, then 1 → TC, else 0 → TC
LAR	Load auxiliary register	1	(dma) → (ARn)
LARK	Load auxiliary register short immediate	1	8-bit constant → ARn
LARP	Load auxiliary register pointer	1	8-bit constant → ARP, (ARP) → AR8
LDP	Load data memory page pointer	1	(dma) → DP
LDPK	Load data memory page pointer immediate	1	9-bit constant → DP
LRLK[1]	Load auxiliary register long immediate	2	16-bit constant → ARn
MAR	Modify auxiliary register	1	
SAR	Store auxiliary register	1	(ARn) → dma
SBRK[1]	Subtract from auxiliary register short immediate	1	(ARn) - 8-bit constant → ARn

T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS			
MNEMONIC	DESCRIPTION	NO WORDS	OPERATION
APAC	Add P register to accumulator	1	(ACC) + (shifted Preg) → ACC
LPH[1]	Load high P register	1	(dma) → Preg (31:16)
LT	Load T register	1	(dma) → Treg
LTA	Load T register and accumulate previous product	1	(dma) → Treg, (ACC) + (shifted Preg) → ACC
LTD	Load T register, accumulate previous product and move data	1	(dma) → Treg, (dma) → dma + 1, (ACC) + (shifted Preg) → ACC
LTP[1]	Load T register and store P register in accumulator	1	(dma) → Treg, (shifted Preg) → ACC
LTS[1]	Load T register and subtract previous product	1	(dma) → Treg, (ACC) - (shifted Preg) → ACC
MAC[1]	Multiply and accumulate	2	(ACC) + (shifted Preg) → ACC, (pma) = (dma) → Preg
MACD[1]	Multiply and accumulate with data move	2	(ACC) + (shifted Preg) → ACC, (pma) = (dma) → Preg, (dma) = dma + 1
MPY	Multiply (with T register, store product in P register)	1	(Treg) * (dma) → Preg
MPYA[1]	Multiply and accumulate previous product	1	(ACC) + (shifted Preg) → ACC (Treg) * (dma) → Preg
MPYK	Multiply immediate	1	(Treg) * 13-bit constant → Preg
MPYS[1]	Multiply and subtract previous product	1	(ACC) - (shifted Preg) → ACC, (Treg) * (dma) → Preg
MPYU[1]	Multiply unsigned	1	Usgn (Treg) * Usgn (dma) → Preg
PAC	Load accumulator with P register	1	(shifted Preg) → ACC
SPAC	Subtract P register from accumulator	1	(ACC) - (shifted Preg) → ACC
SPH[1]	Store high P register	1	(shifted Preg (31:16)) → dma
SPL[1]	Store low P register	1	(shifted Preg (15:0)) → dma
SPM[1]	Set P register output shift mode	1	2-bit constant → PM
SQRA[1]	Square and accumulate	1	(ACC) + (shifted Preg) → ACC, (dma) = (dma) → Preg
SQRS[1]	Square and subtract previous product	1	(ACC) - (shifted Preg) → ACC (dma) = (dma) → Preg

[1]These instructions are not included in the TMS32010 instruction set
[2]These instructions are not included in the TMS32020 instruction set

Tabla VIII.4 Instrucciones del microprocesador TMS320C25 (continuación).

BRANCH/CALL INSTRUCTIONS			
MNEMONIC	DESCRIPTION	NO WORDS	OPERATION
B	Branch unconditionally	2	pma → PC
BACC[1]	Branch to address specified by accumulator	1	(ACC(15-0)) → PC
BANZ	Branch on auxiliary register not zero	2	If (ARAR(P)) ≠ 0, then pma → PC; else (PC) + 2 → PC
BBNZ[1]	Branch if TC bit ≠ 0	2	If (TC) = 1, then pma → PC; else (PC) + 2 → PC
BBZ[1]	Branch if TC bit = 0	2	If (TC) = 0, then pma → PC; else (PC) + 2 → PC
BC[1]	Branch on carry	2	If (C) = 1, then pma → PC; else (PC) + 2 → PC
BGEZ	Branch if accumulator ≥ 0	2	If (ACC) ≥ 0, then pma → PC; else (PC) + 2 → PC
BGZ	Branch if accumulator > 0	2	If (ACC) > 0, then pma → PC; else (PC) + 2 → PC
BIOZ	Branch on I/O status = 0	2	If $(\overline{BIO})$ = 0, then pma → PC; else (PC) + 2 → PC
BLEZ	Branch if accumulator ≤ 0	2	If (ACC) ≤ 0, then pma → PC; else (PC) + 2 → PC
BLZ	Branch if accumulator < 0	2	If (ACC) < 0, then pma → PC; else (PC) + 2 → PC
BNC[1]	Branch on no carry	2	If (C) = 0, then pma → PC; else (PC) + 2 → PC
BNV[1]	Branch if no overflow	2	If (OV) = 0, then pma → PC; else (PC) + 2 → PC
BNZ	Branch if accumulator ≠ 0	2	If (ACC) ≠ 0, then pma → PC; else (PC) + 2 → PC
BV	Branch on overflow	2	If (OV) = 0, then pma → PC; else (PC) + 2 → PC
BZ	Branch if accumulator = 0	2	If (ACC) = 0, then pma → PC; else (PC) + 2 → PC
CALA	Call subroutine indirect	1	(ACC(15-0)) → PC, (PC) + 1 → TOS
CALL	Call subroutine	2	(PC) + 2 → TOS, pma → PC
RET	Return from subroutine	1	(TOS) → PC

I/O AND DATA MEMORY OPERATIONS			
MNEMONIC	DESCRIPTION	NO WORDS	OPERATION
BLKD[1]	Block move from data memory to data memory	2	(pma), addressed by PC) → dma2
BLKP[1]	Block move from program memory to data memory	2	(pma), addressed by PC) → dma
DMOV	Data move in data memory	1	(dma) → dma + 1
FORT[1]	Format serial port registers	1	1-bit constant → FO
IN	Input data from port	1	(data bus addressed by PA) → dma
OUT	Output data to port	1	(dma) → data bus addressed by PA
RFSM[1]	Reset serial port frame synchronization mode	1	0 → FSM
RTXM[1]	Reset serial port transmit mode	1	0 → TXM
RXF[1]	Reset external flag	1	0 → XF
SFSM[1]	Set serial port frame synchronization mode	1	1 → FSM
STXM[1]	Set serial port transmit mode	1	1 → TXM
SXF[1]	Set external flag	1	1 → XF
TBLR	Table read	1	(pma addressed by ACC (15-0)) → dma
TBLW	Table write	1	(dma) → pma addressed by ACC (15-0

[1]These instructions are not included in the TMS32010 instruction set.
[2]These instructions are not included in the TMS32020 instruction set.

Tabla VIII.4 Instrucciones del microprocesador TMS320C25 (continuación).

CONTROL INSTRUCTIONS			
MNEMONIC	DESCRIPTION	NO. WORDS	OPERATIONS
BIT[1]	Test bit	1	(dma bit at (16-bit code)) → TC
BITT[1]	Test bit specified by T register	1	(dma bit at (15-Treg)) → TC
CNFD[1]	Configure block as data memory	1	0 → CNF
CNFP[1]	Configure block as program memory	1	1 → CNF
DINT	Disable interrupt	1	1 → INTM
EINT	Enable interrupt	1	0 → INTM
IDLE[1]	Idle until interrupt	1	(PC) + 1 → PC, power-down
LST	Load status register ST0	1	(dma) → ST0
LST1[1]	Load status register ST1	1	(dma) → ST1
NOP	No operation	1	(PC) + 1 → PC
POP	Pop top of stack to low accumulator	1	(TOS) → ACC
POPD[1]	Pop top of stack to data memory	1	(TOS) → dma
PSHD[1]	Push data memory value onto stack	1	(dma) → TOS
PUSH	Push low accumulator onto stack	1	(ACCL) → TOS
RC[2]	Reset carry bit	1	0 → C
RHM[1]	Reset hold mode	1	0 → HM
ROVM	Reset overflow mode	1	0 → OVM
RPT[1]	Repeat instruction as specified by data memory value	1	(dma) → RPTC
RPTK[1]	Repeat instruction as specified by immediate value	1	8-bit constant → RPTC
RSXM[1]	Reset sign-extension mode	1	0 → SXM
RTC[2]	Reset test/control flag	1	0 → TC
SC[2]	Set carry bit	1	1 → C
SHM[2]	Set hold mode	1	1 → HM
SOVM	Set overflow mode	1	1 → OVM
SST	Store status register ST0	1	ST0 → dma
SST1[1]	Store status register ST1	1	ST1 → dma
SSXM[1]	Set sign-extension mode	1	1 → SXM
STC[2]	Set test/control flag	1	1 → TC
TRAP[1]	Software interrupt	1	(PC) + 1 → TOS, 30 → PC

[1]These instructions are not included in the TMS32010 instruction set
[2]These instructions are not included in the TMS32020 instruction set

VIII.2 Versión del algoritmo DPCM PSC1 en lenguaje ensamblador del microprocesador TMS320C25.

Como se ha mencionado en el capitulo VII, el algoritmo DPCM que reunio las mejores cualidades, en base a las distintas pruebas realizadas, resultó ser el denominado PSC1. Por tal motivo, dicho algoritmo fué seleccionado para ser implantado en lenguaje ensamblador del microprocesador TMS320C25. El objetivo buscado, fué el de acelerar la velocidad de procesamiento, y con ello estimar su posible ejecución en tiempo real.

El algoritmo PSC1 ha sido descrito ampliamente en el capitulo VI, de tal forma que en la presente sección se procederá a describir únicamente las caracteristicas de la versión en lenguaje ensamblador. Cabe mencionar, que esta versión sigue en la medida de lo posible, a la versión previemente descrita en lenguaje de alto nivel.

VIII.2.1 Estructura general del programa.

Como en el caso de la versión en lenguaje de alto nivel, la versión en lenguaje ensamblador (versión TMS) se encuentra conformada por dos partes: una que realiza la función de codificación y otra que se encarga de realizar la correspondiente función de decodificación. Ambas partes tienen el mismo tipo de estructura, de tal suerte que puede realizarse una explicación general por bloques de ambas, para posteriormente señalar en detalle sus diferencias.

La estructura que siguen estas partes es la siguiente: en primer lugar se hace una inicialización del sistema, dentro de la cual se define el mapa de memoria a utilizar por parte del procesador. Se define el valor de los apuntadores que direccionan las localidades de los vectores valor y ángulo actual y valor y ángulo precedentes; se realiza el llenado de las tablas de datos que maneja el algoritmo (tablas de valores de decisión y reconstrucción del cuantificador, además de la tabla de códigos a transmitir en el caso del codificador y tablas de codigos recibidos y valores de reconstrucción del

cuantificador, en el caso del decodificador). Posteriormente, se realiza una inicialización de las variables que controlarán el flujo del programa en cuanto a ciclos de repetición, continuando en seguida con la correspondiente función de codificación o decodificación según se trate. En ésta última función se hace una llamado a la subrutina denominada "orient", cuyo objetivo es el de llevar a cabo el cálculo de la orientación del valor actual X. El programa termina cuando las variables de control de repetición adquieren su valor máximo permitido.

VIII.2.2 Caracteristicas del codificador.

En esta sección, se describen las caracteristicas del codificador, en cuanto a las localidades que utiliza dentro del mapa de memoria, para almacenar las distintas variables y constantes, al realizar su función.

A continuación en la tabla VIII.5, se muestra en forma resumida el conjunto de localidades usadas por el programa codificador, indicándose la variable o constante que almacena.

Algunas observaciones pertinentes sobre la tabla VIII.5 son las siguientes:

El algoritmo PSC1, como se especificó previamente, considera un cuantificador fijo de 11 niveles y un predictor adaptable, que de acuerdo a la orientación que tenga el punto que se esté procesando, hace una elección entre la predicción $(A + C) / 2$ ó $(A + D) /2$. Para tal efecto, cinco vectores importantes son manejados por el algoritmo, siendo estos :

- Valor actual. El cual almacena los valores de la linea que actualmente se está procesando.
- Valor precedente. El cual almacena los valores de la linea previamente procesada.
- Angulo actual, el cual almacena los valores de las orientaciones correspondientes a los puntos de la linea actualmente procesada.

Tabla VIII.5 Mapa de memoria del programa codificador.

PROGRAMA PRINCIPAL	
LOCALIDADES	VALOR I/O CONSTANTE
• De 3FE H a 502 H	valor actual
• De 5FE H a 702 H	valor precedente
• De 7FE H a 902 H	ángulo actual
• De 9FE H a 1102 H	ángulo precedente
• 2000 H	contador principal *j*
• 2001 H	TOTPIX = 65535
• 2002 H	posición
• 2003 H	MAXPIX + 1 = 257
• 2004 H	predicción
- 2005 H	x
+ 2008 H	delta
- 2009 H	sgn (delta)
- De 200A H a 200F H	niveles de decisión del cuantificador.
- De 2010 H a 2015 H	niveles de reconstruccion del cuantificador.
- 2017 H	delta testada o x testada
- De 2018 H a 2023 H	códigos a transmitir
+ 2030 H	código a transmitir seleccionado.
PUERTOS	
- PA0	recibe los datos a codificar
- PA1	imagen reconstruida
- PA2	envia códigos de transmisión

Tabla VIII.5 Mapa de memoria del programa codificador (continuación).

SUBRUTINA	
LOCALIDADES	VARIABLE I/O CONSTANTE
• De 3000 H a 3007 H	vecino
• 3020 H	abs(x testada - A)
• 3021 H	sgn(x testada - A)
• 3022 H	diferencia temporal (x testada - $vecino_1$)
• 3023 H	sgn (diferencia temporal)
• 3040 H	diferencia elegida
• 3041 H	orientación
• De 5000 H a 5007 H	pila de registros

- Angulo precedente. El cual almacena los valores de las orientacioneas correspondientes a los puntos de la linea previamente procesada.
- Vecino. Almacena los valores de las posibles predicciones a utilizar para calcular la orientación local.

El contador principal del programa (j), es el encargado de llevar la cuenta del número de veces que deberá ejecutarse el mismo. En este caso, cada vez que un punto ha sido procesado su valor se incrementa en uno y su cuenta está permitida únicamente entre 0 y TOTPIX. De lo contrario el programa se detiene automáticamente (se inicializa con cero).

La constante denominada TOTPIX igual a 65535, es el valor con el cual se compara el contador principal j, para determinar si el programa debe seguir ejecutándose.

La variable posición, sirve para indicar cual es el punto en la linea actual que se está procesando y en su momento (cuando el último elemento de la linea ha sido procesado) activa la actualización de los apuntadores de valor actual, valor precedente, angulo actual y angulo precedente.

La constante MAXPIX + 1 igual a 257 es el valor con el cual se compara la posición para determinar si los valores y ángulos deben ser actualizados o no.

El formato utilizado para el manejo de los datos dentro del programa, considera una representación en punto fijo con cuatro bits para la parte decimal, once bits para la parte entera y un bit de signo. Este formato, satisface en forma adecuada las aproximaciones necesarias al ejecutarse el programa. Cabe mencionar, que la operación "más difícil" que se realiza en el programa y con la cual se tiene el "mayor error de aproximación", es una división entre dos; de ahí que únicamente se consideren cuatro bits para la parte fraccionaria y los resultados obtenidos sean satisfactorios. De esto se hablara con más detalle al finalizar el capítulo.

Todo signo almacenado en una localidad de memoria, se interpreta como positivo cuando se almacena un cero y se considera negativo cuando se almacena un uno.

De lo anterior se desprende la forma en que fueron llenadas las tablas de datos y de las cuales se presenta a continuación una breve explicación.

Tanto la tabla de niveles de decisión como la de niveles de reconstrucción aprovechan la simetría de las tablas originales, de tal forma que únicamente se almacenan los valores positivos de dichas tablas y por programación, se obtienen los códigos correspondientes a los valores negativos cuando ésto es necesario. La tabla VIII.6, muestra la forma exacta en la cual se encuentran almacenados estos valores.

Puede apreciarse que si se está apuntando a un nivel de decisión determinado, con un simple corrimiento en seis localidades de memoria, se apunta al correspondiente nivel de reconstrucción. Esta característica es aprovechada por el programa para la obtención de los códigos.

Tabla VIII.6 Niveles de decisión y reconstrucción en el mapa de memoria del programa codificador.

NIVELES DE DECISION		
Localidad	Valor almacenado (con formato)	Valor original
200A H	18 H	1.5
200B H	48 H	4.5
200C H	138 H	19.5
200D H	248 H	36.5
200E H	358 H	53.5
200F H	1000 H	256.0
NIVELES DE RECONSTRUCCION		
2010 H	0 H	0
2011 H	30 H	3
2012 H	C0 H	12
2013 H	1C0 H	28
2014 H	2D0 H	45
2015 H	410 H	65

La tabla VIII.7, muestra la disposición que se tiene para los códigos a transmitir dentro del mapa de de memoria.

Tabla VIII.7 Localización de los códigos a transmitir dentro del mapa de memoria

Localidad	Código a transmitir	Valor correspondiente (con formato)	Valor original
2018 H	5	0 H	0
2019 H	6	30 H	3.0
201A H	7	C0 H	12.0
201B H	8	1C0 H	28.0
201C H	9	2D0 H	45.0
201D H	A	410 H	65.0
201E H	5	0 H	0
201F H	4	FFD0 H	- 3.0
2020 H	3	FF40 H	-12.0
2021 H	2	FE40 H	-28.0
2022 H	1	FD30 H	-45.0
2023 H	0	FBF0 H	-65.0

Para finalizar con la descripción del programa codificador, debe mencionarse la forma en que se realiza el flujo de datos hacia y desde el programa mediante los diferentes puertos.

A través del puerto cero (PA0), se realiza la introducción de los datos de la imagen a codificar. Estos datos deben estar organizados en un archivo el cual contenga un dato por renglón con formato ASCII - hexadecimal, ya que éste, es el formato manejado por el microprocesador TMS320C25 para su comunicacion a puertos. Por tanto, PA0 debe ser programado como puerto de entrada.

A través de los puertos uno y dos (PA1 y PA2), se realizan las salidas de los diferentes datos; por PA1 se obtienen los códigos de la imágen decodificada o reconstruida, en tanto que por el puerto PA2 se obtienen los códigos a transmitir. De esta forma, PA1 y PA2 deben ser programados como puertos de salida.

La versión del programa codificador en lenguaje ensamblador del microprocesador TMS320C25, se muestra al final del capitulo bajo la identificación de "CODIF1".

VIII.2.3 Caracteristicas del decodificador.

En esta sección, se describen las caracteristicas correspondientes a la parte de programación del decodificador. Sin embargo, debido a la gran similitud que tiene ésta con la parte del codificador, muchas de las notas aclaratorias de ésta última son aprovechadas, para no caer en una explicación redundante. De esta forma, en la tabla VIII.5, cada una de las localidades usadas por el programa codificador se han marcado de la siguiente manera :

a) Con un "*" para indicar que dicha localidad ha sido usada sin ninguna modificación en el programa decodificador y su contenido y significado es manejado de igual forma que en el programa codificador.

b) Con un "+" para indicar que en dicha localidad de memoria, el dato manejado por el progama decodifiador tiene un significado distinto, al manejado por el programa codificador.

c) Con un "-" para indicar que dichas localidades son usadas en forma totalmente distinta por el programa decodificador, o simplemente para indicar que estas localidades no son usadas.

De esta forma, la tabla VIII.8 muestra la parte complementaria del uso que se da a las localidades de memoria para el programa decodificador, que difieren con respecto al programa codificador. Debe aclararse que el formato manejado para los datos se conserva.

Tabla VIII.8 Mapa de memoria complementario para el programa decodificador.

PROGRAMA PRINCIPAL	
Localidad	Variable y/o constante
2008 H	Código recibido
De 2010 H a 201A H	Tabla de códigos recibidos
De 201B H a 2025 H	Tabla de niveles de reconstrucción del cuantificador
2030 H	x testada
PUERTOS	
PA3	recibe códigos transmitidos
PA4	imagen decodificada
PA5	error de predición

Puede observarse en la tabla anterior que realmente existen pocas modificaciones. Dos de ellas pueden entenderse fácilmente, en la localidad 2008 H, usada en el codificador para almacenar delta , es usada ahora para almacenar el código de delta testada y en la localidad 2030 H, en lugar de almacenar el código a transmitir, se almacena el valor de x testada, que es finalmente el código de salida que nos interesa.

Existe una disposición diferente en cuanto a las tablas de datos en el decodificador con respecto al codificador. En este caso, únicamente se requieren dos tablas: la tabla de códigos recibidos y la tabla de niveles de reconstrucción. La disposición de ambas dentro del mapa de memoria se muestran en las tablas VIII.9 y VIII.10 . Como puede observarse, estas tablas

corresponden exactamente a las usadas en el codificador y la única diferencia estriba en la forma en que han sido almacenadas en memoria.

Tabla VIII.9 Distribución de los códigos recibidos en el mapa de memoria del programa decodificador.

Localidad	Código recibido
2010 H	0
2011 H	1
2012 H	2
2013 H	3
2014 H	4
2015 H	5
2016 H	6
2017 H	7
2018 H	8
2019 H	9
201A H	A

Finalmente, el programa decodificador considera al puerto tres (PA3), como puerto de entrada y a través de éste son introducidos los códigos transmitidos por el programa codificador. Los puertos cuatro y cinco (PA5 y PA4), son considerados como puertos de salida y a través de ellos se obtienen los códigos de la imagen decodificada y códigos de error respectivamente.

La versión del programa decodificador en lenguaje ensamblador del microprocesador TMS320C25, se muestra al final del capitulo bajo el identificador "DECOD1".

Tabla VIII.10 Disposición de los niveles de reconstrucción del cuantificador, en el mapa de memoria del programa decodificador.

Localidad	Valor almacenado (con formato)
201B H	FBF0 H
201C H	FD30 H
201D H	FE40 H
201E H	FF40 H
201F H	FFD0 H
2020 H	0000 H
2021 H	0030 H
2022 H	00C0 H
2023 H	01C0 H
2024 H	02D0 H
2025 H	0410 H

VIII.3 Un algoritmo alternativo.

En las secciones anteriores se ha descrito la traducción del algoritmo DPCM P5C1, al lenguaje ensamblador del microprocesador TMS320C25; debido a que reunió las mejores cualidades de entre los distintos analizados. Sin embargo, por ser un algoritmo con características de adaptabilidad en el predictor, el tiempo de procesamiento que requiere para llevarse a cabo es mayor al que se tuviese si se implantara con características fijas, tanto en el cuantificador como en el predictor. Por ello, se ha realizado una versión alternativa en lenguaje ensamblador, que se desprende fácilmente a partir del algoritmo P5C1. Este algoritmo considera al mismo cuantificador fijo de once niveles y un predictor fijo $(A + C)/2$, de tal suerte que para pasar del algoritmo P5C1 a este último, únicamente hay que eliminar la parte de la

subrutina *orient* y todo aquello relacionado con el cálculo y almacenamiento de orientaciones, así como las instrucciones encaminadas a realizar la predicción (A + D) / 2. De esta forma, en las páginas finales de este capítulo se muestra el listado tanto del programa codificador como del decodificador de esta nueva versión (identificados con los nombres CODIF2 y DECOD2 respectivamente). Cabe señalar, que el objetivo de esta nueva alternativa, fue el de observar simplemente qué resultados se obtenían en cuanto a tiempo de procesamiento entre uno y otro; sin perder de vista que el algoritmo P5C1 mantiene las mejores cualidades, aún en la versión en lenguaje ensamblador. Cabe mencionar también, que el algoritmo alternativo presentado, es el que mejores resultados ofreció dentro del conjunto de algoritmos con características fijas estudiado.

VIII.4 Resultados obtenidos en el procesamiento de los algoritmos implantados en lenguaje ensamblador del microprocesador TMS320C25.

A continuación, se presenta un análisis de los resultados obtenidos al procesar una imagen con las versiones en lenguaje ensamblador, de los algoritmos DPCM implantados. Dos enfoques principales son tomados en cuenta para realizar este análisis: el primero de ellos, se avoca a dar una interpretación desde el punto de vista cualitativo referente a la calidad de la imagen procesada y el segundo describe los resultados obtenidos en cuanto a tiempo de procesamiento; éste último punto es de particular interés, si se piensa en la posible realización de estos algoritmos en tiempo real.

VIII.4.1 Características cualitativas de la imagen procesada en las versiones de lenguaje ensamblador del microprocesador TMS320C25.

Para poder establecer las características cualitativas de la imagen procesada en las versiones TMS320C25, es necesario indicar que éstas se fijan, en base a la comparación directa con las versiones en lenguaje de alto nivel.

En primer lugar, desde un punto de vista subjetivo la calidad de la imagen procesada, a simple vista es excelente como puede observarse en las imágenes mostradas en la fig. VI.12, donde en la parte superior izquierda, se muestra la imagen original; en la parte superior derecha, la imagen procesada con el algoritmo P5C1 en su versión en lenguaje de alto nivel, en la parte inferior izquierda, la imagen correspondiente a este mismo algoritmo pero en su versión en lenguaje ensamblador y finalmente en la parte inferior derecha la imagen correspondiente al error de predicción generado por este algoritmo. De lo anterior se desprende claramente, que la versión del algoritmo P5C1 en lenguaje ensamblador obtiene en forma prácticamente igual, los resultados subjetivos de calidad de imagen obtenidos por la versión en lenguaje de alto nivel. Y por lo tanto, el formato dado a los datos para ser manejados dentro del programa ensamblador es adecuado, ya que el error de aproximación involucrado es prácticamente imperceptible.

Los resultados obtenidos bajo este aspecto, en la versión en lenguaje ensamblador del algoritmo con características fijas, siguen el mismo camino que el indicado anteriormente. Es decir, las características visuales obtenidas por la versión en lenguaje ensamblador son prácticamente idénticas a las obtenidas por la versión en lenguaje de alto nivel , las cuales han sido presentadas en el capitulo VII.

VIII.4.2 Tiempo de procesamiento de los algoritmos implantados en lenguaje ensamblador del microprocesador TMS320C25.

Mediante la simulación llevada a cabo de los algoritmos implantantados en lenguaje ensamblador TMS320C25, pudieron obtenerse los tiempos de procesamiento, requeridos para la codificación de fuente. En el caso del algoritmo P5C1, el tiempo requerido para tal efecto fué de 4.2044 segundos. En tanto que para el algoritmo con caracteristicas fijas fue de 1.029 segundos. Como puede observarse, ninguno de estos algoritmos cumple con los requerimientos del procesamiento en tiempo real. Pero sin embargo, el tiempo de procesamiento obtenido para ambos algoritmos no es tan grande, de tal manera que se puede pensar en una posible aplicación práctica, como podria ser un sistema de videotelefonia.

Por otra parte, si se piensa en la posibilidad de implantar cualquiera de los dos algoritmos, en un sistema de transmisión monocromático de imágenes digitales en tiempo real (debido a la caracteristica de causalidad de los algoritmos), no se considera conveniente la implantación de éstos en una arquitectura basada en un arreglo de procesadores, sino más bien el desarrollo de una arquitectura en un circuito VLSI, adecuada a las caracteristicas funcionales de los algoritmos.

VIII.5 Discusión y conclusión.

En el presente capitulo, se ha desarrollado una presentación de los algoritmos DPCM, seleccionados para ser implantados en lenguaje ensamblador del microprocesador TMS320C25 debido a sus mejores cualidades. Para ello, se ha desarrollado en forma resumida una introducción sobre las caracteristicas fundamentales, tanto de arquitectura como de operación del procesador TMS320C25. Posteriormente, han sido presentadas las versiones en lenguaje ensamblador de los algoritmos seleccionados. Y finalmente, se ha llevado a cabo una revisión de los resultados obtenidos con estas versiones, los cuales arrojan desde el punto de vista cualitativo la misma calidad de imagen que la obtenida por las versiones en lenguaje de alto nivel, en tanto que el tiempo

de procesamiento desarrollado para estos algoritmos no satisface al requerido para su implantación en tiempo real en un sistema monocromático de imágenes digitales por lo que se propone como una posible solución, el desarrollar una arquitectura en circuitería VLSI que se adecúe a los requirimientos de los algoritmos.

VIII.6 Codificación de los algoritmos en lenguaje ensamblador del microprocesador TMS320C25.

En las páginas siguientes se presenta la codificación de los algoritmos implantados en el lenguaje ensamblador del microprocesador TMS320C25.

Los programas identificados con los nombres CODIF1 y DECOD2, corresponden a las partes de codificación y de decodificación respectivamente del algoritmo P5C1, en tanto que los programas CODIF2 y DECOD2 corresponden a dichas partes en el caso del algoritmo con características fijas.

```
        TITL    'M I C D'
        IDT     'CODIF1'
        DEF     RESET,INT0,INT1,INT2
        REF     ISR0,ISR1,ISR2
        DEF     EMP,FIN,
        REF     TIME,RCV,XMT,PROC
        AORG    >0000
RESET   B       INIT
INT0    B       ISR0
INT1    B       ISR1        ;BRINCOS A INTERRUPCIONES
INT2    B       ISR2

INIT    AORG    >0020       ;INICIO DEL PROGRAMA

        CNFD                ;PARTICION DE MEMORIA
        ROVM                ;DESHABILITA OVERFLOW
        LRLK    AR1,>400    ;INICIALIZA APUNTADOR DE VALOR ACTUAL
        LRLK    AR2,>600    ;INICIALIZA APUNTADOR DE VALOR PRECE.
        LRLK    AR3,>800    ;INICIALIZA APUNTADOR DE ANGULO ACTUAL
        LRLK    AR4,>1000   ;INICIALIZA APUNTADOR DE ANGULO PRECE.

TOTPIX  EQU 65535
MAXPIX  EQU 256             ;TABLA DE EQUIVALENCIAS
POSI    EQU  2

        LARP    7
        LDPK    0           ;HABILITA INTERRUPCIONES
        LACK    >3F
        SACL    4

        LRLK    AR7,>200A
        LALK    24
        SACL    *+
        LALK    72
        SACL    *+
        LALK    312
        SACL    *+
        LALK    584
        SACL    *+
        LALK    856
        SACL    *+
        LALK    4096
        SACL    *+
        LALK    0           ;LLENADO DE TABLA DEL CUANTI-
        SACL    *+          ;FICADOR A PARTIR DE LA LOCALI-
        LALK    48          ;DAD 2000H
        SACL    *+
        LALK    192
        SACL    *+
        LALK    448
        SACL    *+
        LALK    720
        SACL    *+
        LALK    1040
        SACL    *+
```

```
        LRLK    AR7,>2018
        LALK    5
        SACL    *+
        LALK    6
        SACL    *+
        LALK    7
        SACL    *+
        LALK    8
        SACL    *+
        LALK    9           ;LLENADO DE LA TABLA DE CODIGOS A
        SACL    *+          ;TRANSMITIR A PARTIR DE LA LOCALI-
        LALK    10          ;DAD 2018H
        SACL    *+
        LALK    5
        SACL    *+
        LALK    4
        SACL    *+
        LALK    3
        SACL    *+
        LALK    2
        SACL    *+
        LALK    1
        SACL    *+
        LALK    0
        SACL    *+

        ZAC                 ;ACC = 0
        LARP    5           ;HABILITA AR5
        LALK    MAXPIX+1    ;ACC = 257          ;INICIALIZACION DE
        LRLK    AR5,>2003   ;AR5 = 2003H        ;APUNTADORES PARA
        SACL    *           ;L 2003H CON 257    ;CICLOS
        LALK    TOTPIX      ;ACC = FFFFH
        LRLK    AR5,>2001   ;AR5 = 2001H
        SACL    *+          ;L 2001H CON FFFFH, AR5 = 2002H
        LALK    POSI        ;ACC = 2
        SACL    *           ;L 2002H CON 2


EMP     LARP    5           ;CICLO FOR PRINCIPAL
        LRLK    AR5,>2000   ;AR5 CON 2000H
        LAC     *+          ;ACC = CONTADOR PRIN. J
        SUB     *           ;SUSTRAE TOTPIX DE J

        BZ      FIN         ;SALTO A FIN SI FUE CERO

        ZAC                 ;ACC = 0
        LRLK    AR5,>2002   ;AR5 = 2002H
        LAC     *+          ;CARGA ACC [2002] POSICION AR5++
        SUB     *-          ;POSI - 257

        BLEZ    ET1         ;IF POSI
```

```
        LRLK   AR5,>600
        RPTK   >FF
        BLKD   >400,*+    ;ACTUALIZACION DE VALOR ACTUAL Y
        LRLK   AR5,>1000  ;ORIENTACIONES
        RPTK   >FF
        BLKD   >800,*+

        LALK   POSI
        LRLK   AR5,>2002  ;POSICION = 2

        SACL   *
        LRLK   AR5,>2004  ;INICIALIZA PREDICCION
        BLKD   >600,*

        LRLK   AR1,>400   ;REINICIA APUNTADOR DE VAL ACTUAL
        LRLK   AR2,>600   ;REINICIA APUNTADOR DE VAL PRECEDENTE
        LRLK   AR3,>800   ;REINICIA APUNTADOR DE ANG ACTUAL
        LRLK   AR4,>1000  ;REINICIA APUNTADOR DE ANG PRECEDENTE

        B      EMP        ;SALTO INCONDICIONAL A EMP

ET1     LARP   5
        LRLK AR5,>2005
        IN    *,PA0
        LAC   *-,4
        SUB    *          ;CALCULA DELTA

        LRLK    AR5,>2008 ;GUARDA DELTA EN >2008

        BLZ    SIG        ;BRINCO SI sgn( DELTA) ES NEG  A SIG
        SACL   *+

        ZAC               ;ALMACENA SIGNO POSITIVO DE DELTA
        SACL   *

ET2     LARP   6          ;CUANTIFICADOR  ,@ AR6
        LRLK   AR6,>2008  ;AR6 = 2008H
        LAC    *          ;ACC = DELTA
        LARP   5          ;@ AR5
        LRLK   AR5,>200A  ;AR5 APUNTA 1ER VALOR DEL CUANT.

ET4     LARP   5          ;@ AR5
        SUB    *+         ;ACC=DELTA-VALOR N DEL CUANTIZADOR
        BLZ    ET3        ;IDENTIFICACION DEL VALOR A ELEGIR

        LARP   6          ;SE ALMACENA NUEVAMENTE DELTA EN ACC
        LAC *

        B      ET4        ;SALTO A ET4 PARA PROBAR EL SIG. VALOR

ET3     ZAC               ;ACC=0
        ADRK   13         ;AR5=AR5+13 (LOCALIZA CODIGO A TRANS)
```

```
        LARP  7
        LRLK  AR7,>2030
        SAR   AR5,*       ;SE ALMACENA LA DIRECCION DEL CODIGO
        LAR   AR7,*       ;A TRANSMITIR EN LA DIRECCION 2030H

        LARP  AR5
        SBRK  8
        LAC   *
        LRLK  AR5,>2017
        SACL  *           ;SE ALMACENA DELTA TESTADA EN 2017H

        LARP  6
        ADRK  1
        ZAC
        SUB   *           ;IDENTIFICA SGN DE DELTA

        BNZ   ET5         ;SALTO A ET5 SI NEGATIVO

        B     ET6         ;SALTO A ET6 SI POSITIVO

ET5     LARP  AR7
        LRLK  AR7,>2030
        LAR   AR7,*       ;SE ALMACENA EN AR7 LA DIRECCION
        ADRK  6           ;DEL CODIGO A TRANSMITIR

        LARP  5
        LRLK  AR5,>2017
        LAC   *
        NEG
        SACL  *           ;SE OBTIENE EL VALOR NEG DE DELTA

ET6     LARP  5
        LRLK  AR5,>2017
        LAC   *
        LRLK  AR5,>2004
        ADD   *           ;CALCULA X_TESTADA

        LARP  1
        SACL  *           ;ALMACENA X_TESTADA -> VAL_ACTUAL


        RPTK  3
        SFR
        LARP  5
        LRLK ARS,>2017
        SACL  *           ;ENVIA X_TESTADA SIN
        OUT   *,PA1       ;CORRIMIENTOS A PA1

        LARP  7
        OUT   *,PA2       ;SACA CODIGO A CANAL DE T. POR PA2

        CALL  ORIENT      ;CALCULA LA ORIENTACION A X_TESTADA
```

```
        LARP 5
        LRLK AR5,>3041
        LAC  *
        LARP 3
        SACL *              ;SE ALMACENA ORIENT A ANG_ACTUAL

        LARP  3
        LAC   *+
        LARP  4
        SBRK  2
        ADD   *+            ;SE DETERMINA QUE TIPO DE PREDICTOR
        RPTK  3             ;SE VA A USAR
        ADD   *+
        SBRK  2
        SUB   3

        BGZ   ET10          ;SALTO SI P = (A + D)/2


        LARP  2
        ADRK  1
        ZAC
        LAC   *
        LARP  1
        ADD   *+
        SFR   1
        LARP  5
        LRLK  AR5,>2004     ;CALCULA P = (A + C)/2 Y LA
        SACL  *             ;ALMACENA EN 2004H

        B     ET7           ;SALTO INCONDICIONAL A ET7

ET10    LARP  2
        ADRK  2
        ZAC
        LAC   *-
        LARP  1
        ADD   *+
        SFR   1
        LARP  5
        LRLK  AR5,>2004     ;CALCULA P = (A + D)/2 Y LA
        SACL  *             ;ALMACENA EN 2004H

        B     ET7           ;SALTO INCONDICIONAL A ET7

SIG     ABS
        LARP  5
        LRLK  AR5,>2008
        SACL  *+
        ZAC
        LACK  1             ;ALMACENA ABS(DELTA) A [2008H]
        SACL  *+            ;Y SGN NEGATIVO DE DELTA=1 A [2009]

        B     ET2           ;SALTO INCONDICIONAL A ET2
```

```
ET7  * LARP   5
       LRLK   AR5,>2000
       LAC    *
       ADDK   1
       SACL   *+          ;ACTUALIZA CONTADOR PRINCIPAL J

       LRLK   AR5,>2002
       LAC    *
       ADDK   1
       SACL   *           ;ACTUALIZA POSICION

       B      EMP         ;NEXT FOR PRINCIPAL

FIN    B      FIN         ;FIN DEL PROGRAMA


ORIENT LARP   0
       LRLK   AR0,>5000
       SAR    AR1,*+
       SAR    AR2,*+
       SAR    AR3,*+
       SAR    AR4,*+
       SAR    AR5,*+
       SAR    AR6,*+
       SAR    AR7,*+      ;SALVA APUNTADORES DE PROG. PRIN.

       LARP   1
       SBRK   1
       LAC    *+
       LARP   7
       LRLK   AR7,>3000
       SACL   *+,0,AR2
       SBRK   2
       LAC    *+,0,AR7
       SACL   *+,0,AR2
       LAC    *+,0,AR7
       SACL   *+,0,AR2
       ADD    *,0,AR7
       SFR
       SACL   *+,0,AR2
       LAC    *+,0,AR7
       SACL   *+,0,AR2
       ADD    *,0,AR7
       SFR
       SACL   *+,0,AR2
       LAC    *+,0,AR7
       SACL   *+,0,AR2
       LAC    *,0,AR7
       SACL   *+          ;LLENA LA TABLA DE VECINO

       LARP   1
       LAC    *
       LARP   7
       LRLK   AR7,>3000
       SUB    *+          ;CALCULA X-A
```

```
        LARP  AR6          ;PREPARA APUNTADOR PARA ALMACENAR
        LRLK  AR6,>3020    ;X-A EN LA LACALIDAD 3020H

        BGZ   ALFA1        ;IDENTIFICA SIGNO POSITIVO DE X-A

        ABS
        SACL  *+
        ZAC
        ADDK  1            ;ALMACENA IX-AI EN 3020H Y SIGNO
        SACL  *+           ;NEGATIVO EN 3021H

        B     ALFA2        ; SALTO INCONDICIONAL A ALFA2

ALFA1   SACL  *+
        ZAC                ;ALMACENA X-A EN 3020H Y SIGNO
        SACL  *+           ;POSITIVO EN 3021H

ALFA2   LRLK  AR6,>3040
        LARP  6
        LALK  1600
        SACL  *+
        ZAC                ;INICIALIZA DIFERENCIA Y ORIENTACION
        SACL  *-           ;CON VALORES DE DEFAUL

        LRLK  AR6,>3007
        LRLK  AR0,4
        LARP  5            ;INICIALIZA APUNTADORES CONTROLADORES
        LRLK  AR5,7        ;DE CICLO [AR5]=7 , [AR0]=4

ALFA3   LARP  1
        LAC   *
        LARP  6
        SUB   *
        ABS
        LARP  7
        LRLK  AR7,>3022    ;CALCULA DIF TEMPORAL = IX-VECINOI
        SACL  *+           ;Y LA ALMACENA EN 3022H

        LARP  6
        LAC   *-
        SUB   *            ;CALCULA VECINO - (VECINO-1)

        BGZ   BETA1        ;IDENTIFICA SIGNO POSITIVO

        ZAC
        ADDK  1
        LARP  7            ;ALMACENA SIGNO NEGATIVO DE
        SACL  *-           ;VECINO - (VECINI -1) EN 3023H

        B     BETA2        ;SALTO ENCONDICIONAL A BETA2

BETA1   ZAC
        LARP  7            ;ALMACENA SIGNO POSITIVO DE
        SACL  *-           ;VECINO- (VECINO -1) EN 3023H
```

```
BETA2  LAC  *
       SBLK  >400        ;IDENTIFICA SI DIF TEMPORAL ES MENOR
       BGEZ  BETA4       ;DE 64 SI NO SALTA AL SIGUIENTE CICLO

       LARP 2
       LRLK AR2,>3042
       SAR  AR5,*
       LAC  *
       SUBK 1            ;IDENTIFICA SI HAY QUE HACER UNA O
       BLEZ BETA5        ;DOS PREGUNTAS

       LARP  3
       LRLK  AR3,>3021
       LAC  *
       LRLK  AR3,>3023   ;IDENTIFICA SI LOS SIGNOS DE X-A Y
       SUB  *            ;VECINO - (VECINO-1) SON IGUALES SI
       BNZ  BETA4        ;NO LO SON SALTA AL SIGUIENTE CICLO

BETA5  LRLK AR3,>3022
       LAC *
       LRLK AR3,>3040
       SUB *             ;COMPARA DIFERENCIA CON DIF TEMPORAL

       BGZ  BETA4        ;A BETA4 SI DIF TEMPORAL > DIFERENCIA

       BLKD >3022,*+     ;CAMBIA DIFERENCIA POR DIF TEMPORAL

       LARP 5
       CMPR 2
       BBNZ BETA3        ;ELIGE LA ORIENTACION QUE CORRESPONDE

       LARP 3
       ZAC
       SACL *
       LRLK AR5,>0       ;ALMACENA ORENTACION = 0 EN 3041H

       B    BETA4        ;BRINCO INCONDICIONAL A BETA4

BETA3  LARP 3
       ZAC
       ADDK 1
       SACL *            ;ALMACENA ORIENTACION = 1 EN 3041H

BETA4  LARP 5
       BANZ  ALFA3       ;DETERMINA FIN DE CICLO
       LARP  0
       LRLK  AR0,>5000
       LAR   AR1,*+
       LAR   AR2,*+
       LAR   AR3,*+
       LAR   AR4,*+
       LAR   AR5,*+
       LAR   AR6,*+
       LAR   AR7,*+      ;RECUP. APUNTADORES DE PROG. PRIN.
       RET               ;RETORNO A PROGRAMA PRINCIPAL
```

```
        TITL    'M I C D'
        IDT     'DECOD1'
        DEF     RESET,INT0,INT1,INT2
        REF     ISR0,ISR1,ISR2
        DEF     EMP,FIN
        REF     TIME,RCV,XMT,PROC

        AORG    >0000

RESET   B       INIT        ,IDENTIFICA RESET

INT0    B       ISR0
INT1    B       ISR1        ;BRINCOS A INTERUPCIONES
INT2    B       ISR2

INIT    AORG    >0020       ;INICIO DEL PROGRAMA

        CNFD                ;PARTICION DE MEMORIA
        ROVM                ;DESHABILITA OVERFLOW

        LRLK    AR1,>400    ;INICIALIZA APUNTADOR DE VALOR ACTUAL
        LRLK    AR2,>600    ;INICIALIZA APUNTADOR DE VALOR PRECE.
        LRLK    AR3,>800    ;INICIALIZA APUNTADOR DE ANG. ACTUAL
        LRLK    AR4,>1000   ;INICIALIZA APUNTADOR DE ANG. PRECE.

TOTPIX  EQU 65535
MAXPIX  EQU 256             ;TABLA DE EQUIVALENCIAS
POSI    EQU  2

        LARP    7
        LDPK    0           ;HABILITA INTERRUPCIONES
        LACK    >3F
        SACL    4

        LRLK    AR7,>2010
        LALK    0
        SACL    *+
        LALK    1
        SACL    *+
        LALK    2
        SACL    *+
        LALK    3
        SACL    *+
        LALK    4
        SACL    *+
        LALK    5
        SACL    *+
        LALK    6           ;LLENADO DE TABLA DE DE CODIGOS
        SACL    *+          ;TRANSMITIDOS
        LALK    7
        SACL    *+
        LALK    8
        SACL    *+
        LALK    9
        SACL    *+
```

```
        LALK    >A
        SACL    *+

        LALK    >FBF0
        SACL    *+
        LALK    >FD30
        SACL    *+
        LALK    >FE40
        SACL    *+
        LALK    >FF40
        SACL    *+
        LALK    >FFD0
        SACL    *+
        LALK    0
        SACL    *+
        LALK    >30
        SACL    *+
        LALK    >C0         ;LLENADO DE TABLA DE CUANTIFICADOR
        SACL    *+
        LALK    >1C0
        SACL    *+
        LALK    >2D0
        SACL    *+
        LALK    >410
        SACL    *+

        ZAC                 ;ACC = 0
        LARP    5           ;HABILITA AR5
        LALK    MAXPIX+1    ;ACC = 257        ;INICIALIZACION DE
        LRLK    AR5,>2003   ;AR5 = 2003H      ;APUNTADORES PARA
        SACL    *           ;L 2003H CON 257  ;CICLO
        LALK    TOTPIX      ;ACC = FFFFH
        LRLK    AR5,>2001   ;AR5 = 2001H
        SACL    *+          ;L 2001H CON FFFFH, AR5 = 2002H
        LALK    POSI        ;ACC = 2
        SACL    *           ;L 2002H CON 2
EMP     LARP    5           ;CICLO FOR PRINCIPAL
        LRLK    AR5,>2000   ;AR5 CON 2000H
        LAC     *+          ;ACC = CONTADOR PRINCIPAL J
        SUB     *           ;SUSTRAE TOTPIX DE J

        BZ      FIN         ;SALTO A FIN SI FUE CERO

        ZAC                 ;ACC = 0
        LRLK    AR5,>2002   ;AR5 = 2002H
        LAC     *+          ;CARGA ACC [2002] POSICION AR5++
        SUB     *-          ;POSI - 257
        BLEZ    ET1         ;IF POSI

        LRLK    AR5,>600
        RPTK    >FF
        BLKD    >400,*+     ;ACTUALIZACION DE VALOR ACTUAL Y
        LRLK    AR5,>1000   ;ORIENTACIONES
        RPTK    >FF
        BLKD    >800,*+
```

```
        LALK   POSI
        LRLK   AR5,>2002 ;POSI = 2

        SACL   *
        LRLK   AR5,>2004 ;INICIALIZA PREDICCION
        BLKD   >600,*

        LRLK   AR1,>400  ;REINICIA APUNTADOR DE VALOR ACTUAL
        LRLK   AR2,>600  ;REINICIA APUNTADOR DE VALOR PRECE.
        LRLK   AR3,>800  ;REINICIA APUNTADOR DE ANG. ACTUAL
        LRLK   AR4,>1000 ;REINICIA APUNTADOR DE ANG. PRECE.

        B      EMP       ;SALTO INCONDICIONAL A EMP

ET1     LARP   5
        LRLK   AR5,>2008
        IN     *,PA3     ;OBTIENE CODIGO DE TRANS.
        LAC    *

        LARP   AR6
        LRLK   AR6,>2010
ET4     LARP   6
        SUB    *+
        BZ     ET3
        LARP   5
        LAC    *
        B      ET4       ;IDENTIFICA CODIGO DE TRANS.

ET3     ADRK   10         ;IDENTIFICA DELTA TESTADA Y
        LAC    *

        RPTK   3
        SFR
        ADDK   80
        LARP   AR0
        LRLK   AR0,>1070
        SACL   *
        OUT    *,PA5      ;ESCRIBE CODIGO DE ERROR POR PA5

        LARP   6
        LAC    *
        LRLK   AR6,>2004
        ADD    *
        LARP   1          ;CALCULA X_TESTADA Y LA ALMACENA
        SACL   *          ;EN VALOR ACTUAL

        RPTK   3
        SFR
        LARP   AR6
        LRLK   AR6,>2030
        SACL   *          ;RECONSTRUYE Y ENVIA LA IMAGEN
        OUT    *,PA4      ;DECODIFICADA POR PA4

        CALL   ORIENT     ;CALCULA LA ORIENTACION A X_TESTADA
```

```
        LARP 5
        LRLK AR5,>3041
        LAC  *
        LARP 3
        SACL *              ;SE ALMACENA ORIENT A ANG_ACTUAL

        LARP  3
        LAC   *+
        LARP  4
        SBRK  2
        ADD   *+            ;SE DETERMINA QUE TIPO DE PREDICTOR
        RPTK  3             ;SE VA A USAR
        ADD   *+
        SBRK  2
        SUB   3

        BGZ   ET10          ;SALTO SI P = (A + D)/2

        LARP  2
        ADRK  1
        ZAC
        LAC   *
        LARP  1
        ADD   *+
        SFR   1
        LARP  5
        LRLK  AR5,>2004  ;CALCULA P = (A + C)/2 Y LA
        SACL  *             ;ALMACENA EN 2004H
        B     ET7           ;SALTO INCONDICIONAL A ET7

ET10    LARP  2
        ADRK  2
        ZAC
        LAC   *-
        LARP  1
        ADD   *+
        SFR   1
        LARP  5
        LRLK  AR5,>2004  ;CALCULA P = (A + D)/2 Y LA
        SACL  *             ;ALMACENA EN 2004H

ET7     LARP   5
        LRLK   AR5,>2000
        LAC    *
        ADDK   1
        SACL   *+           ;ACTUALIZA CONTADOR PRINCIPAL J

        LRLK   AR5,>2002
        LAC    *
        ADDK   1
        SACL   *            ;ACTUALIZA POSICION

        B      EMP          ;NEXT FOR PRINCIPAL

FIN     B      FIN          ;FIN DEL PROGRAMA
```

```
ORIENT LARP  0

       LRLK  AR0,>5000
       SAR   AR1,*+
       SAR   AR2,*+
       SAR   AR3,*+
       SAR   AR4,*+
       SAR   AR5,*+
       SAR   AR6,*+
       SAR   AR7,*+      ;SALVA APUNTADORES DE PROG. PRIN.

       LARP  1
       SBRK  1
       LAC   *+
       LARP  7
       LRLK  AR7,>3000
       SACL  *+,0,AR2
       SBRK  2
       LAC   *+,0,AR7
       SACL  *+,0,AR2
       LAC   *+,0,AR7
       SACL  *+,0,AR2
       ADD   *,0,AR7
       SFR
       SACL  *+,0,AR2
       LAC   *+,0,AR7
       SACL  *+,0,AR2
       ADD   *,0,AR7
       SFR
       SACL  *+,0,AR2
       LAC   *+,0,AR7
       SACL  *+,0,AR2
       LAC   *,0,AR7
       SACL  *+          ;LLENA LA TABLA DE VECINO

       LARP  1
       LAC   *
       LARP  7
       LRLK  AR7,>3000
       SUB   *+          ;CALCULA X-A

       LARP  AR6         ;PREPARA APUNTADOR PARA ALMACENAR
       LRLK  AR6,>3020   ;X-A EN LA LACALIDAD 3020H

       BGZ   ALFA1       ;IDENTIFICA SIGNO POSITIVO DE X-A

       ABS
       SACL  *+
       ZAC
       ADDK  1           ;ALMACENA |X-A| EN 3020H Y SIGNO
       SACL  *+          ;NEGATIVO EN 3021H

       B     ALFA2       ; SALTO INCONDICIONAL A ALFA2
```

```
ALFA1   SACL *+
        ZAC                 ;ALMACENA X-A EN 3020H Y SIGNO
        SACL *+             ;POSITIVO EN 3021H

ALFA2   LRLK AR6,>3040
        LARP 6
        LALK 1600
        SACL *+
        ZAC                 ;INICIALIZA DIFERENCIA Y ORIENTACION
        SACL *-             ;CON VALORES DE DEFAUL

        LRLK AR6,>3007
        LRLK AR0,4
        LARP 5              ;INICIALIZA APUNTADORES CONTROLADORES
        LRLK AR5,7          ;CICLO [AR5]=7 , [AR0]=4

ALFA3   LARP 1
        LAC  *
        LARP 6
        SUB  *
        ABS
        LARP 7
        LRLK AR7,>3022      ;CALCULA DIF TEMPORAL = IX-VECINOI
        SACL *+             ;Y LA ALMACENA EN 3022H

        LARP 6
        LAC  *-
        SUB  *              ;CALCULA VECINO - (VECINO-1)

        BGZ  BETA1          ;IDENTIFICA SIGNO POSITIVO

        ZAC
        ADDK 1
        LARP 7              ;ALMACENA SIGNO NEGATIVO DE
        SACL *-             ;VECINO - (VECINI -1) EN 3023H

        B    BETA2          ;SALTO INCONDICIONAL A BETA2

BETA1   ZAC
        LARP 7              ;ALMACENA SIGNO POSITIVO DE
        SACL *-             ;VECINO - (VECINO -1) EN 3023H

BETA2   LAC  *
        SBLK  >400          ;IDENTIFICA SI DIF TEMPORAL ES MENOR
        BGEZ  BETA4         ;DE 64 SI NO SALTA AL SIGUIENTE CICLO

        LARP 2
        LRLK AR2,>3042
        SAR  AR5,*
        LAC  *
        SUBK 1              ;IDENTIFICA SI HAY QUE HACER UNA O
        BLEZ BETA5          ;DOS PREGUNTAS
```

```
        LARP  3
        LRLK  AR3,>3021
        LAC   *
        LRLK  AR3,>3023  ;IDENTIFICA SI LOS SIGNOS DE X-A Y
        SUB   *          ;VECINO - (VECINO-1) SON IGUALES SI
        BNZ   BETA4      ;NO LO SON SALTA AL SIGUIENTE CICLO

BETA5   LRLK AR3,>3022
        LAC *
        LRLK AR3,>3040
        SUB *            ;COMPARA DIFERENCIA CON DIF TEMPORAL

        BGZ  BETA4       ;A BETA4 SI DIF TEMPORAL > DIFERENCIA

        BLKD >3022,*+    ;CAMBIA DIFERENCIA POR DIF TEMPORAL

        LARP 5
        CMPR 2
        BBNZ BETA3       ;ELIGE LA ORIENTACION QUE CORRESPONDE

        LARP 3
        ZAC
        SACL *
        LRLK AR5,>0      ;ALMACENA ORENTACION = 0 EN 3041H

        B    BETA4       ;BRINCO INCONDICIONAL A BATA4

BETA3   LARP 3
        ZAC
        ADDK 1
        SACL *           ;ALMACENA ORIENTACION = 1 EN 3041H

BETA4   LARP 5
        BANZ  ALFA3      ;DETERMINA FIN DE CICLO

        LARP  0
        LRLK  AR0,>5000
        LAR   AR1,*+
        LAR   AR2,*+
        LAR   AR3,*+
        LAR   AR4,*+
        LAR   AR5,*+
        LAR   AR6,*+
        LAR   AR7,*+     ;RECUP. APUNTADORES DE PROG. PRIN.

        RET              ;RETORNO A PROGRAMA PRINCIPAL
```

```
        TITL    'M I C D'
        IDT     'CODIF2'
        DEF     RESET,INT0,INT1,INT2
        REF     ISR0,ISR1,ISR2
        DEF     EMP,FIN
        REF     TIME,RCV,XMT,PROC
        AORG    >0000
RESET   B       INIT        ;IDENTIFICA RESET
INT0    B       ISR0
INT1    B       ISR1        ;BRINCOS A INTERRUPCIONES
INT2    B       ISR2

INIT    AORG    >0020       ;INICIO DEL PROGRAMA

        CNFD                ;PARTICION DE MEMORIA
        ROVM                ;DESHABILITA OVERFLOW
        LRLK    AR1,>400    ;INICIALIZA APUNTADOR DE VALOR ACTUAL
        LRLK    AR2,>600    ;INICIALIZA APUNTADOR DE VALOR PRECE.

TOTPIX  EQU 65535
MAXPIX  EQU 256             ;TABLA DE EQUIVALENCIAS
POSI    EQU  2

        LARP    7
        LDPK    0           ;HABILITA INTERRUPCIONES
        LACK    >3F
        SACL    4

        LRLK    AR7,>200A
        LALK    24
        SACL    *+
        LALK    72
        SACL    *+
        LALK    312
        SACL    *+
        LALK    584
        SACL    *+
        LALK    856
        SACL    *+
        LALK    4096
        SACL    *+
        LALK    0           ;LLENADO DE LA TABLA DEL
        SACL    *+          ;CUANTIFICADOR
        LALK    48
        SACL    *+
        LALK    192
        SACL    *+
        LALK    448
        SACL    *+
        LALK    720
        SACL    *+
        LALK    1040
        SACL    *+
```

```
        LRLK   AR7,>2018
        LALK   5
        SACL   *+
        LALK   6
        SACL   *+
        LALK   7
        SACL   *+
        LALK   8
        SACL   *+
        LALK   9          ;TABLA DE CODIGOS A TRANSMITIR
        SACL   *+
        LALK   10
        SACL   *+
        LALK   5
        SACL   *+
        LALK   4
        SACL   *+
        LALK   3
        SACL   *+
        LALK   2
        SACL   *+
        LALK   1
        SACL   *+
        LALK   0
        SACL   *+

        ZAC               ;ACC = 0
        LARP   5          ;HABILITA AR5
        LALK   MAXPIX+1   ;ACC = 257        ;INICIALIZACION DE
        LRLK   AR5,>2003  ;AR5 = 2003H      ;APUNTADORES PARA-
        SACL   *          ;L 2003H CON 257 ;CICLO
        LALK   TOTPIX     ;ACC = FFFFH
        LRLK   AR5,>2001  ;AR5 = 2001H
        SACL   *+         ;L 2001H CON FFFFH, AR5 = 2002H
        LALK   POSI       ;ACC = 2
        SACL   *          ;L 2002H CON 2

EMP     LARP   5          ;CICLO FOR PRINCIPAL

        LRLK   AR5,>2000  ;AR5 CON 2000H
        LAC    *+         ;ACC = CONTADOR PRINCIPAL J
        SUB    *          ;SUSTRAE TOTPIX DE J
        BZ     FIN        ;SALTO A FIN SI Z
        ZAC               ;ACC = 0
        LRLK   AR5,>2002  ;AR5 = 2002H
        LAC    *+         ;CARGA ACC [2002] POSICION AR5++
        SUB    *-         ;POSI - 257
        BLEZ   ET1        ;IF POSI

        LRLK   AR5,>600
        RPTK   >FF
        BLKD   >400,*+    ;ACTUALIZACION DE VALOR ACTUAL

        LALK   POSI
        LRLK   AR5,>2002  ;POSICION = 2
```

```
        SACL   *
        LRLK   AR5,>2004 ;INICIALIZA PREDICCION
        BLKD   >600,*

        LRLK   AR1,>400  ;REINICIA APUNTADOR DE VALOR ACTUAL
        LRLK   AR2,>600  ;REINICIA APUNTADOR DE VALOR PRECE.

        B      EMP       ;SALTO INCONDICIONAL A EMP

ET1     LARP   5
        LRLK AR5,>2005
        IN   *,PA0
        LAC  *-,4
        SUB    *         ;CALCULA DELTA
        LRLK    AR5,>2008 ;GUARDA DELTA EN >2008

        BLZ    SIG       ;BRINCO SI sgn( DELTA) = -  A SIG
                          SACL  *+

        ZAC              ;ALMACENA SIGNO POSITIVO DE DELTA
        SACL   *

ET2     LARP   6         ;CUANTIFICADOR  ,@ AR6

        LRLK   AR6,>2008 ;AR6 = 2008H
        LAC    *         ;ACC = DELTA
        LARP   5         ;@ AR5
        LRLK   AR5,>200A ;AR5  APUNTA 1ER VALOR DEL CUANT.
ET4     LARP   5         ;@ AR5
        SUB    *+        ;ACC=DELTA-VALOR N DEL CUANTIZADOR

        BLZ    ET3       ;IDENTIFICACION DEL VALOR A ELEGIR

        LARP   6         ;ALMACENA NUEVAMENTE DELTA EN ACC
        LAC *

        B      ET4       ;SALTO PARA PROBAR EL SIG. VALOR

ET3     ZAC              ;ACC=0
        ADRK   13        ;AR5=AR5+13 (LOCALIZA CODIGO A TRANS)

        LARP   7
        LRLK   AR7,>2030
        SAR    AR5,*     ;SE ALMACENA LA DIRECCION DEL CODIGO
        LAR    AR7,*     ;A TRANSMITIR EN LA DIRECCION 2030H

        LARP   AR5
        SBRK   8
        LAC    *
        LRLK   AR5,>2017
        SACL   *         ;SE ALMACENA DELTA TESTADA EN 2017H
```

```
        LARP  6
        ADRK  1
        ZAC
        SUB   *             ;IDENTIFICA SGN DE DELTA

        BNZ   ET5           ;SALTO A ET5 SI ES NEGATIVO

        B     ET6           ;SALTO A ET6 SI ES POSITIVO

ET5     LARP  AR7
        LRLK  AR7,>2030
        LAR   AR7,*         ;SE ALMACENA EN AR7 LA DIRECCION
        ADRK  6             ;DEL CODIGO A TRANSMITIR

        LARP  5
        LRLK  AR5,>2017
        LAC   *
        NEG
        SACL  *             ;SE OBTIENE EL VALOR NEG DE DELTA

ET6     LARP  5
        LRLK  AR5,>2017
        LAC   *
        LRLK  AR5,>2004
        ADD   *             ;CALCULA X_TESTADA
        LARP  1
        SACL  *             ;ALMACENA X_TESTADA -> VAL_ACTUAL


        RPTK  3
        SFR
        LARP  5
        LRLK AR5,>2017
        SACL  *             ;ENVIA X_TESTADA SIN
        OUT   *,PA1         ;CORRIMIENTOS A PA1

        LARP  7
        OUT   *,PA2         ;SACA CODIGO A CANAL DE T. POR PA2

        LARP  2
        ADRK  1
        ZAC
        LAC   *
        LARP  1
        ADD   *+
        SFR   1
        LARP  5
        LRLK  AR5,>2004  ;CALCULA P =(A + C)/2 Y LA
        SACL  *             ;ALMACENA EN 2004H

        B     ET7           ;SALTO INCONDICIONAL A ET7
```

```
SIG     ABS
        LARP  5
        LRLK  AR5,>2008
        SACL  *+
        ZAC
        LACK  1             ;ALMACENA ABS(DELTA) A [2008H]
        SACL  *+            ;Y SGN NEGATIVO DE DELTA=1 A [2009]

        B     ET2           ;SALTO INCONDICIONAL A ET2


ET7     LARP  5
        LRLK  AR5,>2000
        LAC   *
        ADDK  1
        SACL  *+            ;ACTUALIZA CONTADOR PRINCIPAL J

        LRLK  AR5,>2002
        LAC   *
        ADDK  1
        SACL  *             ;ACTUALIZA POSI

        B     EMP           ;NEXT FOR PRINCIPAL

FIN     B     FIN           ;FIN DEL PROGRAMA
```

```
        TITL    'M I C D'
        IDT     'DECOD2'
        DEF     RESET,INT0,INT1,INT2
        REF     ISR0,ISR1,ISR2
        DEF     EMP,FIN
        REF     TIME,RCV,XMT,PROC
        AORG    >0000
RESET   B       INIT        ;IDENTIFICA RESET
INT0    B       ISR0
INT1    B       ISR1        ;BRINCOS A INTERRUPCIONES
INT2    B       ISR2

INIT    AORG    >0020       ;INICIO DEL PROGRAMA
        CNFD                ;PARTICION DE MEMORIA
        ROVM                ;DESHABILITA OVERFLOW
        LRLK  AR1,>400      ;INICIALIZA APUNTADOR DE VALOR ACTUAL
        LRLK  AR2,>600      ;INICIALIZA APUNTADOR DE VALOR PRECE.

TOTPIX EQU 65535
MAXPIX EQU 256              ;TABLA DE EQUIVALENCIAS
POSI    EQU  2

        LARP   7
        LDPK   0            ;HABILITA INTERRUPCIONES
        LACK   >3F
        SACL   4

        LRLK   AR7,>2010
        LALK   0
        SACL   *+
        LALK   1
        SACL   *+
        LALK   2
        SACL   *+
        LALK   3
        SACL   *+
        LALK   4
        SACL   *+
        LALK   5
        SACL   *+
        LALK   6            ;ALMACENA LA TABLA DE VALORES TRANS-
        SACL   *+           ;TIDOS A PARTIR DE LA LOC 2010H
        LALK   7
        SACL   *+
        LALK   8
        SACL   *+
        LALK   9
        SACL   *+
        LALK   >A
        SACL   *+
```

```
        LALK    >FBF0
        SACL    *+
        LALK    >FD30
        SACL    *+
        LALK    >FE40
        SACL    *+
        LALK    >FF40
        SACL    *+
        LALK    >FFD0
        SACL    *+
        LALK    0
        SACL    *+
        LALK    >30
        SACL    *+
        LALK    >C0             ;LLENADO DE TABLA DEL CUANTIFICADOR
        SACL    *+
        LALK    >1C0
        SACL    *+
        LALK    >2D0
        SACL    *+
        LALK    >410
        SACL    *+

        ZAC                     ;ACC = 0
        LARP    5               ;HABILITA AR5
        LALK    MAXPIX+1        ;ACC = 257      ;INICIALIZACION DE
        LRLK    AR5,>2003       ;AR5 = 2003H    ;APUNTADORES PARA-
        SACL    *               ;L 2003H CON 257 ;CICLOS
        LALK    TOTPIX          ;ACC = FFFFH
        LRLK    AR5,>2001       ;AR5 = 2001H
        SACL    *+              ;L 2001H CON FFFFH, AR5 = 2002H
        LALK    POSI            ;ACC = 2
        SACL    *               ;L 2002H CON 2

EMP     LARP    5               ;CICLO FOR PRINCIPAL

        LRLK    AR5,>2000       ;AR5 CON 2000H
        LAC     *+              ;ACC = CONTADOR PRIN. J


        SUB     *               ;SUSTRAE TOTPIX DE J

        BZ      FIN             ;SALTO A FIN SI Z

        ZAC                     ;ACC = 0
        LRLK    AR5,>2002       ;AR5 = 2002H
        LAC     *+              ;CARGA ACC [2002] POSICION AR5++
        SUB     *-              ;POSI - 257

        BLEZ    ET1             ;IF POSI

        LRLK    AR5,>600
        RPTK    >FF
        BLKD    >400,*+         ;ACTUALIZACION DE VALOR ACTUAL
```

```
        LALK    POSI
        LRLK    AR5,>2002 ;POSI = 2

        SACL    *
        LRLK    AR5,>2004 ;INICIALIZA PREDICCION
        BLKD    >600,*

        LRLK    AR1,>400  ;REINICIA APUNTADOR DE VALOR ACTUAL
        LRLK    AR2,>600  ;REINICIA APUNTADOR DE VALOR PRECE.

        B       EMP       ;SALTO INCONDICIONAL A EMP

ET1     LARP    5
        LRLK    AR5,>2008
        IN      *,PA3
        LAC     *         ;OBTIENE CODIGO DE TRANSMISION POR PA3

        LARP    AR6
        LRLK    AR6,>2010
ET4     LARP    6
        SUB     *+
        BZ      ET3
        LARP    5
        LAC     *
        B       ET4       ;IDENTIFICA CODIGO DE TRANSMISION

ET3     ADRK    10
        LAC     *         ;IDENTIFICA DELTA TESTADA
        RPTK    3
        SFR
        ADDK    80
        LARP    AR0
        LRLK    AR0,>1070
        SACL    *
        OUT     *,PA5     ;ESCRIBE CODIGO DE ERROR POR PA5

        LARP    6
        LAC     *
        LRLK    AR6,>2004
        ADD     *
        LARP    1         ;CALCULA X_TESTADA Y LA ALMACENA
        SACL    *         ;EN VALOR ACTUAL

        RPTK    3
        SFR
        LARP    AR6
        LRLK    AR6,>2030
        SACL    *          ;RECONSTRUYE Y ENVIA IMAGEN
        OUT     *,PA4      ;DECODIFICADA POR PA4
```

```
        LARP  2
        ADRK  1
        LAC   *
        LARP  1
        ADD   *+
        SFR
        LARP  AR6
        LRLK  AR6,>2004   ;CALCULA P=(A + C)/2 Y LA
        SACL  *           ;ALMACENA EN 2004H

        LARP  5
        LRLK  AR5,>2000
        LAC   *
        ADDK  1
        SACL  *+          ;ACTUALIZA CONTADOR PRINCIPAL

        LRLK  AR5,>2002
        LAC   *
        ADDK  1
        SACL  *           ;ACTUALIZA POSICION

        B     EMP         ;NEXT FOR PRINCIPAL

FIN     B     FIN         ;FIN DEL PROGRAMA
```

C A P I T U L O I X

CONCLUSIONES

CAPITULO IX. CONCLUSIONES.

En el presente trabajo destacan cuatro aspectos fundamentales que se constituyen como la esencia del mismo. En primer lugar se llevó a cabo la revisión documental de una serie de algoritmos de codificación de fuente que utilizan un esquema predictivo: el DPCM (Modulación por pulsos codificados diferencial). A partir de esta investigación fue posible determinar las ventajas y las limitaciones de los mismos considerando diferentes criterios de comparación. Con todo ello se pudo concluir qué algoritmos prometían mejores resultados.

Una vez concluida la investigación documental se procedió a simular los algoritmos seleccionados en un sistema digital de cómputo. Con la simulación se pudieron contrastar, inicialmente, treinta esquemas diferentes que incluían parámetros fijos y adaptables en los bloques de predicción y cuantificación (seis esquemas de predicción y cinco de cuantificación). Los criterios de comparación fueron tres: a) la observación subjetiva de las imágenes en un sistema PS/60, b) el comportamiento del algoritmo en cuanto a la dinámica de los errores debidos al canal de transmisión y c) la razón de compresión máxima alcanzable.

De los treinta algoritmos se seleccionaron los doce que mejores resultados arrojaron. Sin embargo se observó que ninguno de ellos lograba la robustez requerida en cuanto a los tres aspectos. Por ello, surgió la necesidad de intentar conjuntar las características deseables de cada esquema; el resultado final fueron los algoritmos de predicción adaptable D* y D2 en los que se logra un equilibrio en la calidad subjetiva de las imágenes, la razón de compresión y la robustez frente al ruido de canal. La característica fundamental de estos algoritmos es el comportamiento isotrópico de la predicción para las diferentes orientaciones de los contornos y la estabilidad frente a los errores en la transmisión. En este contexto se inició una segunda etapa de simulación en donde se probaron dieciseis esquemas de codificación (parejas predictor-cuantificador) que incluían a los algoritmos de predicción D* y D2 . Se agregaron, además, dos

criterios adicionales de comparacion, a saber: a) minimizacion de las probabilidades condicionales de que el error de prediccion sobrepase un cierto umbral, dado que el punto actual a codificar ha sido declarado contorno y b) la observacion de las imagenes correspondientes al error de prediccion. La segunda etapa de la simulacion permitio corroborar el buen comportamiento de los algoritmos D* y D2, utilizando un cuantificador fijo no uniforme de once niveles y un cuantificador adaptable que utiliza una funcion de enmascaramiento para ocultar el error de cuantificacion. Entonces fue posible, seleccionar los tres esquemas (pareja prediccion-cuantificador), que lograban los resultados deseados: una razon de compresion de 2 a 3.5 bits por punto, robustez frente al ruido de canal y una calidad subjetiva excelente.

En sintesis, se encontro que los algoritmos de prediccion que utilizan el estado local de la vecindad del punto actual a codificar, para determinar la prediccion mas adecuada, alcanzan mayor eficacia si los predictores a seleccionar observan un comportamiento isotropico para las diferentes orientaciones de los contornos. Se pudo concluir tambien que el comportamiento de un cuantificador que utiliza una funcion de actividad local de luminancia, tiene un efecto subjetivo de filtrado pasabajas para transiciones de luminancia por debajo de un cierto umbral. Ello se traduce en una señal de error de prediccion sin cambios bruscos. Sin embargo, si el umbral es excedido, la funcion de actividad local permite detectar que en esa transicion, es posible ocultar el error de cuantificacion. Por estas razones el cuantificador enmascarado efectivamente las distorsiones de las imagenes debidas al efecto de la cuantificacion. El unico requerimiento de este cuantificador adaptable es, que el algoritmo de prediccion sea muy robusto frente al ruido de canal (requerimiento que satisfacen los algoritmos D* y D2).

La estabilidad del sistema predictor-cuantificador (que utiliza una funcion de luminancia), depende basicamente de la robustez del predictor frente al ruido del canal de transmision. Los predictores selectivos para las diferentes orientaciones de los contornos en la imagen, producen una descorrelacion en el error de prediccion a lo largo de los contornos y son muy inestables cuando se presenta un error en la recepcion.

Una tercera aportación de este trabajo es que todos los algoritmos de simulación y despliegue se integraron en un paquete llamado MICD, con el objeto de servir como material didactico de apoyo al laboratorio de procesamiento de imágenes digitales de la D.E.P.F.I. El paquete MICD se complementa con el programa SPI, que es una interfaz del paquete MICD con el usuario. Las caracteristicas principales del software desarrollado son las siguientes:

a) La codificación de las subrutinas se llevó a cabo en lenguaje C (Turbo C versión 2.0), con lo que la velocidad de ejecucion en un sistema digital AT, PS, etc. se incrementa considerablemente. Además de que existe una gran versatilidad para el manejo del hardware presente. Se desarrollaron además algunas subrutinas, en FORTRAN para interactuar con la imágenes almacenadas en el sistema VAX780 del laboratorio.

b) La programación es estructurada. Se hace gran énfasis en la modularidad de las subrutinas con lo que es posible utilizarlas como bloques independientes para ejecutar diferentes tareas.

c) La documentación de todas las subrutinas permite al usuario dar mantenimiento a cada una de ellas, teniéndose la opción de modificar e incrementar fácilmente las opciones de procesamiento para la simulación. Este es el caso del programa SPI, con el cual es posible facilitar al usuario la tarea de configurar las opciones en un sin número de programas de despliegue y simulación.

d) Los programas ofrecen al usuario una forma sencilla y veloz para simular hasta 40 diferentes esquemas de codificación DPCM intercampo. Adicionalmente se cuenta con un banco de 120 fotografias en las que se muestran algunos de los resultados obtenibles con el programa, así como diversos programas para desplegar imágenes utilizando el sistema PS/60 o la tarjeta PIB (professional image board) del laboratorio.

Finalmente fue necesario evaluar la factibilidad de implantacion de los algoritmos seleccionados en tiempo real. Para ello se desarrollaron, con base en las subrutinas codificadas en alto nivel, dos algoritmos en el lenguaje ensamblador del microprocesador TMS320C25. El primer problema a resolver lo constituyó el elegir la aritmetica de punto fijo a utilizar. Se concluyó que

el formato de números signados en complemento a dos con once bits para la parte entera, cuatro para la parte fraccionaria y uno para el signo, eran suficientes para lograr los mismos resultados alcanzados al utilizar un lenguaje de alto nivel.

Las pruebas de los algoritmos en lenguaje ensamblador se hicieron en un simulador del procesador TMS320C25 . Con él se encontraron los tiempos de ejecución de los algoritmos, con características adaptables y características fijas (4.2 s y 1.029 s, respectivamente). Estos permitien concluir que se requiere de un diseño en circuitería VLSI, en vez de un arreglo de procesadores si se desea alcanzar una implantación en tiempo real. Sin embargo para los fines didácticos y de la formación de recursos humanos en el laboratorio de procesamiento de imágenes digitales, resulta muy cómodo, en comparación con los tiempos alcanzados en un sistema PS/60 (120 s en promedio).

Perspectivas.

Este trabajo ha permitido dar continuidad y forma a uno de los tópicos más importantes desde el punto de vista didáctico: la codificación DPCM intercampo. Todo el software desarrollado permitirá a los nuevos alumnos del laboratorio aprender, observando resultados prácticos. Además de que ya existe una base sobre la que se desarrollará seguramente mayor infraestructura en beneficio de los estudiantes.

APENDICE A

EFECTOS DE LA CUANTIFICACION EN EL PROCESAMIENTO DE SEÑALES DIGITALES

APENDICE A. EFECTOS DE LA CUANTIFICACION EN EL PROCESAMIENTO DE SEÑALES DIGITALES.

A.1 Representación de números binarios en punto fijo y en punto flotante.

En el procesamiento digital de señales analógicas las muestras de las señales analógicas son representadas en forma digital (A/D). Básicamente, el proceso de conversión analógico digital implica el muestreo de la señal analógica y la transformación de las muestras en una secuencia de dígitos binarios que representan la amplitud cuantificada de la señal. Dentro del sistema de números binarios hay varios métodos mediante los cuales una muestra de una señal analógica puede ser representada en forma binaria. La clasificación general de las representaciones binarias puede ser dividida en las representaciones de punto fijo y de punto flotante. Dentro de la subclasificación de representaciones en punto fijo hay varias representaciones, por ejemplo, la representación binaria natural, magnitud signada, compensación binaria, complemento a uno y complemento a dos. Estas representaciones binarias son descritas brevemente a continuación.

A.1.1 Representación binaria en punto fijo.

La representación binaria o código binario mejor conocido es la representación binaria natural. En el código binario natural, una palabra de código de b bits, tal como 10011, corresponde al número decimal

$$A = 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 0 \times 2^3 + 1 \times 2^4$$
$$= 1 + 2 + 16 = 19$$

Por otro lado, la palabra de código binario 0.10011 representa una fracción correspondiente al número decimal

$$B = 1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-5}$$
$$= 1/2 + 1/16 + 1/32 = 19/32$$

Obsérvese que un corrimiento del punto binario a la izquierda en n posiciones corresponde a dividir el número por 2^n . Un corrimiento del punto binario a la derecha en n posiciones corresponde a multiplicar el numero por 2^n .

Ya que la palabra de código binario representa una fracción, un entero o un número mixto, el bit más a la izquierda es llamado el bit más significativo (MSB) y el bit más a la derecha es llamado el bit menos significativo (LSB) en la palabra de código. En la representación de una fracción, el MSB tiene un peso de $2^{-1} = 1/2$ y el LSB tiene un peso de $2^{-b} = 1/2^b$, donde b es el numero de bits en la representacion binaria fraccional. Obsérvese que el peso 2^{-b} asignado al LSB es la resolucion inherente en la representacion de una fraccion por b bits. Aun mas la palabra de código binario 0.11 ... 1 corresponde al número decimal $1 - 2^{-b}$.

En la conversión de señales analógicas bipolares, un bit adicional es requerido para indicar la información del signo. El resultado es un código bipolar.

Hay cuatro métodos usados comúnmente para representar los números bipolares : la representación magnitud signada, compensación binaria, complemento a uno y complemento a dos. La tabla A.1 ilustra las cuatro representaciones utilizando cuatro bits.

La representación en magnitud signada es el método más simple y el mas directo para la representación de números signados en forma digital. Un cero en la posición del bit en la extrema izquierda representa un número positivo, un uno en esta posición representa un número negativo. Los b bits restantes representan la magnitud. Debe destacarse que el número cero tiene dos representaciones 00 ... 0 o 100 ...0 .

La representacion de compensacion binaria es similar a la representación binaria natural. En esta representación, el número mas negativo es representado por la palabra de código con mas ceros en $b + 1$ bits y el número positivo más grande es representado por la palabra de $b + 1$ unos

Tabla A.1 Códigos bipolares.

Número	Magnitud signada	Compensación binaria	Complemento a dos	Complemento a uno
7	0111	1111	0111	0111
6	0110	1110	0110	0110
5	0101	1101	0101	0101
4	0100	1100	0100	0100
3	0011	1011	0011	0011
2	0010	1010	0010	0010
1	0001	1001	0001	0001
0	0000	1000	0000	0000
0	1000	1000	0000	1111
-1	1001	0111	1111	1110
-2	1010	0110	1110	1101
-3	1011	0101	1101	1100
-4	1100	0100	1100	1011
-5	1101	0011	1011	1010
-6	1110	0010	1010	1001
-7	1111	0001	1001	1000

Por ello puede verse a la representación de compensación binaria idéntica a la representación binaria natural, la cual podría obtenerse sumando un número positivo a la señal bipolar tal que la señal resultante se haga unipolar con la más pequeña amplitud de valor cero. Nótese que hay solamente una palabra de código correspondiente al valor cero. Consecuentemente este código evita la ambigüedad inherente a la representación signo y magnitud. Por otra parte existe un inconveniente mayor con la representación mediante compensación binaria: un error en la lectura del bit de la extrema izquierda como un uno en vez de un cero o viceversa, resulta en un error de amplitud grande. Por ejemplo, de la tabla A.1 notamos que el 1001 y el 0001 representan las amplitudes de +1 y -7 respectivamente. Consecuentemente, un error en el bit de la extrema izquierda corresponde a un error de amplitud +8, lo cual corresponde a la mitad de la totalidad de la escala, es decir, un número muy grande.

La representación en complemento a dos es idéntica a la representación de magnitud signada para números positivos. De aquí que todos los números

positivos están representados con un cero en la posición del bit de signo (extrema izquierda). Un número negativo es representado tomando el complemento a dos del correspondiente número positivo, en otras palabras el número negativo es obtenido al restar el número positivo de la potencia de 2 mayor más cercana a dicho número. En una forma más sencilla, el complemento a dos de un número se obtiene complementando dicho número (cambiando todos los ceros por unos y los unos por ceros) y sumándole un uno al LSB. Por ejemplo, el número -3/8 es obtenido simplemente complementando 0.011 (3/8) para obtener 1.100 y posteriormente sumándole 0.001 . Esto da 1.101, el cual representa -3/8 en complemento a dos.

Si comparamos el complemento a dos con la compensación binaria, encontramos que difieren solamente en el bit más significativo (es decir, el bit de la extrema izquierda). De aquí que es un problema sencillo, cambiar de complemento a dos a compensación binaria y viceversa.

La representación de complemento a uno es idéntica a la de complemento a dos y a la de magnitud signada para números positivos, un número negativo es obtenido complementando su correspondiente representación del valor positivo. Por ejemplo, la representación de - 3/8 es 1.100 , el cual es el complemento a uno de 0.011 (3/8) . Obsérvese que un cero es ahora representado como 00 ... 0 o 11 ... 1, lo cual representa una ambigüedad poco deseada.

Es interesante notar en esto el efecto de un acarreo en el MSB con aritmética de complemento a dos y complemento a uno. Por ejemplo, 4/8 - 3/8 = 1/8 . En complemento a dos tenemos

$$0.100 \oplus 1.101 = 0.001$$

donde $\oplus$ indica adición módulo 2 . Nótese que si el bit de acarreo está presente en el MSB, éste es descartado. Por otro lado, en aritmética de complemento a uno, el acarreo en el MSB, si está presente, es acarreado en el LSB . Entonces el cálculo 4/8 - 3/8 = 1/8 es

$$0.100 \oplus 1.100 = 0000 \oplus 0001 = 0001$$

Si bien una variedad de representaciones adicionales de punto fijo son posibles, las descritas anteriormente son las más frecuentemente usadas en la práctica.

En sumas o substracciones de dos números de punto fijo, cada uno de

longitud b bits (con un bit adicional para el signo) el resultado es un número de b bits. Si el resultado de la suma excede al número más grande que pueda ser representado con b bits, ocurre un sobreflujo. La única forma para evitar este problema es incrementar el número de bits en el acumulador y entonces aumentar el rango dinámico que pueda ser acomodado.

La multiplicación de dos números de punto fijo cada uno de longitud de b bits resulta en un producto de $2b$ bit de longitud. En aritmética de punto fijo, el producto es truncado o redondeado a b bits. Como resultado, tenemos un error de truncamiento o de redondeo ("round-off") en los b bits menos significativos. La caracterización de tales errores es tratada más adelante.

A.1.2 Representación binaria en punto flotante.

Una representación de números en punto fijo nos permite cubrir un rango de números, es decir, $x_{max} - x_{min}$ con una resolución

$$\Delta = \frac{x_{max} - x_{min}}{m - 1}$$

donde $m = 2^b$ es el número de niveles y b es el número de bits. Una característica básica de la representación en punto fijo es que la resolución es fija. Además, Δ aumenta en proporción directa a un incremento en el rango dinámico.

Una representación en punto flotante puede ser empleada como medio para cubrir un rango dinámico más grande. La representación binaria en punto flotante comúnmente usada en la práctica consiste de una mantisa, M, la cual es la parte fraccionaria del número y cae en el rango $1/2 \leq M < 1$, multiplicada por el factor exponencial 2^p donde el exponente p es un entero positivo o negativo. Por tanto un número N es representado como

$$N = M \cdot 2^p$$

La mantisa requiere un bit de signo para representar números positivos y

negativos, y el exponente requiere un bit de signo adicional. Como la mantisa es una fracción con signo, podemos usar cualquiera de las cuatro representaciones de punto fijo descritas arriba.

Por ejemplo, el número $N_1 = 5$ es representado por la siguiente mantisa y exponente:

$$M_1 = 0.101000$$
$$p_1 = 011$$

mientras el número $N_2 = 3/8$ es representado por la mantisa y el exponente siguientes:

$$M_2 = 0.110000$$
$$p_2 = 101$$

donde el bit de la extrema izquierda en el exponente representa el signo del bit.

Si los dos números fueran multiplicados, las mantisas serían multiplicadas y los exponentes sumados. Entonces el producto de los dos números dados antes es

$$\begin{aligned} N_1 N_2 &= M_1 M_2 \; 2^{p_1 + p_2} \\ &= (0.011110) \cdot 2^{010} \\ &= (0.111100) \cdot 2^{001} \end{aligned}$$

Por otro lado, la suma de los dos números en punto flotante requiere que los exponentes sean iguales. Esto puede ser realizado haciendo un corrimiento de la mantisa del número más pequeño a la derecha y compensándolo incrementando el exponente correspondiente. Entonces el número N_2 puede ser expresado como

$$M_2 = 0.0000111$$
$$p_2 = 001$$

Con $p_2 = p_1$, podemos sumar los dos números N_1 y N_2. El resultado es

$$N_1 + N_2 = (0.101011) \cdot 2^{011}$$

Puede observarse que la operación de corrimiento requerida para igualar el exponente de N_2 con el de N_1 resulta en general en una pérdida de precisión, en general. En el ejemplo de arriba la mantisa de 6 bits fue lo suficientemente grande para acomodar un corrimiento de 4 bits a

la derecha para M_2 sin la pérdida de algunos de los unos. Sin embargo, un corrimiento de 5 bits habria causado la pérdida de un solo bit y un corrimiento de 6 bits a la derecha habria resultado en una mantisa de $M_2 = 0.000000$, a menos que se redondee al entero inmediato superior después del corrimiento, así que $M_2 = 0.000001$.

El sobreflujo ocurre en la multiplicación de dos números en punto flotante cuando la suma de los exponentes excede al rango dinámico de la representación en punto fijo del exponente.

Comparando una representación en punto fijo con una representación en punto flotante con el mismo número total de bits es aparente que la representación en punto flotante nos permite cubrir un rango dinámico más amplio variando la resolución a través del rango. La resolución decrece, con un incremento en el tamaño de números sucesivos. En otras palabras, la distancia entre dos números en punto flotante sucesivos se incrementa conforme los números crecen en tamaño. En ésta, la resolución de las variables resulta en un rango dinámico más grande. Alternativamente, si deseamos cubrir el mismo rango dinámico con ambas representaciones, la de punto fijo y la de punto flotante, la representación en punto flotante proporciona una resolución más fina, para números pequeños, pero para números grandes, la resolución es más burda. En contraste, la representación en punto fijo provee una resolución uniforme para todo el rango de números.

Por ejemplo, si tenemos una computadora con un tamaño de palabra de 32 bits, es posible representar 2^{32} números. Si deseamos representar los enteros positivos comenzando con cero, el entero más grande posible que podemos representar es

$$2^{32} - 1 = 4,294,967,295$$

La distancia entre números sucesivos (la resolución) es 1. alternativamente, podemos designar al bit de la extrema izquierda como el bit de signo y usar los restantes 31 bits para la magnitud. En tal caso una representación en punto fijo nos permite cubrir el rango

$$-(2^{31} - 1) = -2,147,483,647 \text{ a } (2^{31} - 1) = 2,147,483,647$$

otra vez con una resolución de 1 .

Por otro lado, suponga que incrementamos la resolución colocando 10

bits para una parte fraccional, 21 bits para la parte entera y 1 bit para el signo. Entonces esta representación nos permite cubrir el rango dinámico

$$-(2^{31} - 1)\cdot 2^{-10} = -(2^{21} - 2^{-10}) \quad \text{a} \quad (2^{31} - 1)\cdot 2^{-10} = 2^{21} - 2^{-10}$$

o, equivalentemente,

$$-2,097,151.999 \quad \text{a} \quad 2,097,151.999$$

en este caso, la resolución es 2^{-10}. Entonces el rango dinámico ha sido decrementado por un factor de aproximadamente 1000 (actualmente 2^{10}), mientras la resolución ha sido incrementada por el mismo factor.

Para comparar, suponga que la palabra de 32 bits es usada para representar números de punto flotante. En particular, si la mantisa es representada por 23 bits mas un bit de signo y si el exponente es representado por 7 bits mas un bit de signo. Ahora el número más pequeño en magnitud tendrá la representación:

signo	23 bits	signo	7 bits	
0	100...0	1	1111111	$= 1/2 \times 2^{-127} \approx 0.3\times10^{-38}$

En el otro extremo, el número más largo con esta representación en punto flotante es

signo	23 bits	signo	7 bits	
0	111...1	0	1111111	$= (1-2^{-23}) \times 2^{127} \approx 1.7\times10^{38}$

Entonces hemos alcanzado un rango dinámico de aproximadamente 10^{76} pero con una resolución variable. En particular tenemos una resolución fina para números pequeños y una resolución burda para los numeros grandes.

A.1.3 Errores resultantes del redondeo y truncamiento.

En la realización de cálculos tales como multiplicaciones con una aritmética de punto fijo o de punto flotante usualmente se tiene el problema de cuantificar un número via truncamiento o redondeo de un nivel de una precisión a un nivel de menor precisión. El efecto de redondeo y truncamiento es introducir un error cuyo valor dependerá del número de bits

obtenido después de la cuantificación con respecto al número de bits original. Las características de los errores introducidos por redondeo o truncamiento dependen de la forma particular de representación del número.

Si consideramos una representación de punto fijo en la cual un número x es cuantificado de b_u a b bits. Entonces el número

$$x = 0.1011\ldots01$$

consistente de b_u bits antes de la cuantificación es representado como

$$x = 0.101\ldots111$$

con b bits después de la cuantificación, donde $b < b_u$. Por ejemplo, si x representa la muestra de una señal analógica, entonces b_u puede ser tomado como infinito. En cualquier caso si el cuantificador trunca el valor de x, el error de truncamiento está definido como

$$E_t = Q_t(x) - x \qquad (A.1)$$

Primero consideramos el rango de valores del error para las representaciones de magnitud signada y de complemento a dos. En ambas representaciones los números positivos tienen representaciones idénticas. Para números positivos, el resultado del truncamiento es un número que es más pequeño que el número no cuantificado. Consecuentemente, el error de truncamiento resultante de una reducción del número de bits significativos de b_u a b es

$$-(2^{-b} - 2^{-b_u}) \leq E_t \leq 0 \qquad (A.2)$$

donde el error más grande se produce al descartar $b_u - b$ bits, los cuales tiene un valor de uno.

En el caso de números de punto fijo negativos basados en la representación en magnitud signada, el error de truncamiento es positivo, ya que el truncamiento básicamente reduce la magnitud de los números. Consecuentemente, para números negativos, tenemos

$$0 \leq E_t \leq (2^{-b} - 2^{-b_u}) \qquad (A.3)$$

En la representación en complemento a dos, el negativo de un número es

obtenido substrayendo el correspondiente número positivo a la potencia de 2 del número de bits en la representación. Como una consecuencia, el efecto del truncamiento en un número negativo es incrementar la magnitud del número negativo. Consecuentemente, $x > Q_t(x)$ y de aquí

$$-\left(2^{-b} - 2^{-bu}\right) \leq E_t \leq 0 \qquad (A.4)$$

De esto se concluye que el error de truncamiento para la representación en magnitud signada es simétrica respecto al cero y está en el rango

$$-\left(2^{-b} - 2^{-bu}\right) \leq E_t \leq \left(2^{-b} - 2^{-bu}\right) \qquad (A.5)$$

Por otro lado, para la representación en complemento a dos, el error de truncamiento es siempre negativo y está en el rango

$$-\left(2^{-b} - 2^{-bu}\right) \leq E_t \leq 0 \qquad (A.6)$$

Se considerarán ahora los errores de cuantificación debido al redondeo de un número. Un número x representado por b_u bits antes de la cuantificación y b bits después de la cuantificación incurre en un error de cuantificación

$$E_r = Q_r(x) - x \qquad (A.7)$$

Básicamente, el redondeo involucra sólamente a la magnitud del número y, consecuentemente, el error de redondeo es independiente del tipo de representación de punto fijo. El máximo error que puede ser introducido por medio del redondeo es $\left(2^{-b} - 2^{-bu}\right) / 2$ y este pude ser positivo o negativo, dependiendo del valor de x. Por lo tanto, el error de redondeo es simétrico alrededor del cero y está en el rango

$$-\frac{1}{2}\left(2^{-b}\, 2^{-bu}\right) \leq E_r \leq \frac{1}{2}\left(2^{-b} - 2^{-bu}\right) \qquad (A.8)$$

Estas relaciones son resumidas en la figura A.1 donde x es una señal continua con amplitud ($b_u = \infty$).

En una representación en punto flotante, la mantisa es redondeada o

truncada. Debido a la resolución no uniforme, el error correspondiente en una representación en punto flotante es proporcional al número siendo cuantificado. Una representación apropiada para el valor cuantificado es

$$Q(x) = x + e\,x \qquad (A.9)$$

donde e es llamado el error relativo. Ahora

$$Q(x) - x = e\,x \qquad (A.10)$$

En el caso del truncamiento basado en la representación en complemento a dos de la mantisa, tenemos

$$-2^{+p}\,2^{-b} < e_t\,x < 0 \qquad (A.11)$$

para números positivos. Como $2^{+p-1} \le x < 2^{+p}$, esto resulta en que

$$-2^{-b+1} < e_t \le 0 \qquad x > 0 \qquad (A.12)$$

Por otro lado, para un número negativo en representación en complemento a dos el error es

$$0 \le e_t\,x < 2^{p}\,2^{-b}$$

y por lo tanto

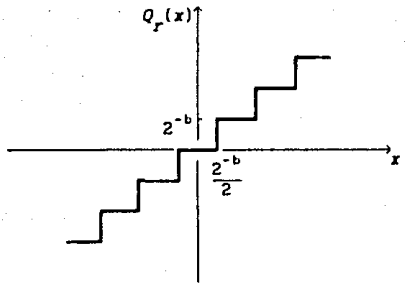
$$0 \le e_t < 2^{-b+1} \qquad x < 0 \qquad (A.13)$$

En el caso donde la mantisa es redondeada, el error resultante es simétrico relativo al cero y tiene un valor máximo de $\pm\,2^{-b}/2$. Consecuentemente, el error de redondeo queda

$$-2^{p}\cdot 2^{-b}/2 < e_r\,x \le 2^{p}\cdot 2^{-b}/2 \qquad (A.14)$$

Además, como x cae en el rango $2^{p-1} \le x < 2^{p}$, dividimos entre 2^{p-1} tal que

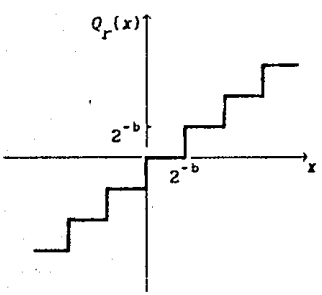
$$-2^{-b} < e_r \le 2^{-b} \qquad (A.15)$$



$$E_r = Q_r(x) - x$$

$$-\frac{1}{2}2^b \leq E_r \leq \frac{1}{2}2^{-b}$$

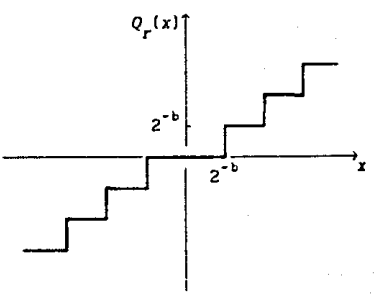
(a)



$$E_t = Q_t(x) - x$$

$$-2^{-b} \leq E_r \leq 0$$

(b)



$$E_t = Q_t(x) - x$$

$$-2^{-b} \leq E_t \leq 2^{-b}$$

(c)

Fig. A.1 Errores de cuantificación en el redondeo y en el truncamiento.

En cálculos aritméticos que involucran cuantificación vía truncamiento y redondeo, es conveniente adoptar una aproximación estadística para la caracterización de tales errores. El cuantificador puede ser modelado introduciendo un ruido aditivo al valor x no cuantificado. Entonces podemos escribir

$$Q(x) = x + \varepsilon \qquad (A.16)$$

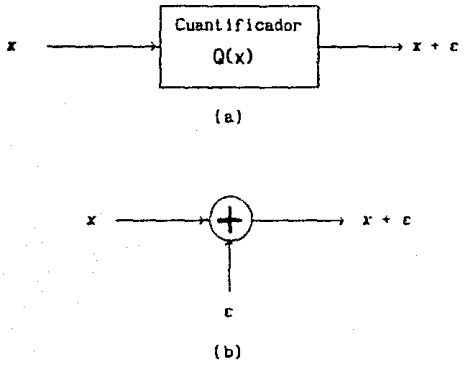


Fig. A.2 Modelo de ruido aditivo para el proceso de cuantificación no lineal: (A) sistema actual; (b) modelo para cuantificación.

donde $\varepsilon = Er$ por redondeo y $\varepsilon = Et$ por truncamiento. Este modelo es mostrado en la figura A.2 .Como x debe ser cualquier número que caiga dentro de cualquiera de los niveles del cuantificador, el error de cuantificación es usualmente modelado como una variable aleatoria que cae dentro de los límites especificados arriba. Esta variable aleatoria se asume es uniformemente distribuida dentro de los rangos especificados por las representaciones de punto fijo mencionadas anteriormente. Además, en la práctica, $bu >> b$, tal que podemos olvidar el factor de 2^{-bu} . Bajo

estas condiciones, las funciones de densidad de probabilidad para los errores de redondeo y truncamiento en las representaciones en punto fijo son mostradas en la figura A.3.

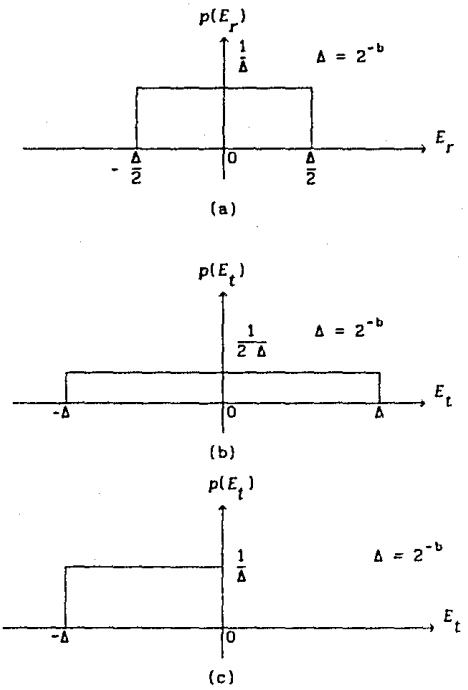


Fig. A.3 Caracterización estadística de los errores de cuantificación: (a) error de redondeo "round-off"; (b) error de truncamiento para magnitud signada; (c) error de truncamiento para complemento a dos.

Nótese que en el caso del truncamiento en la representación en complemento

a dos del número, el valor promedio del error tiene una propensión (bias) de $2^{-b}/2$, ya que en todos los otros casos mostrados con anterioridad, el error tiene un valor promedio de cero.

A P E N D I C E B

PROCEDIMIENTO EN LA TOMA DE FOTOGRAFIAS

APENDICE B . PROCEDIMIENTO EN LA TOMA DE FOTOGRAFIAS.

Una parte fundamental para llevar a cabo la evaluación subjetiva de la calidad de las imágenes procesadas con los diferentes algoritmos implantados, es la obtención de las fotografías correspondientes a la combinación predictor cuantificador seleccionada. A continuación se indica el procedimiento usado para la obtención de las fotografías, con el objeto de tener una referencia, ya que se requiere un procedimiento especial para tomar fotografías sobre un monitor de televisión o de un sistema digital de cómputo.

En primer lugar debe mencionarse que las imágenes mostradas en un monitor de televisión son más tenues que las obtenidas de objetos reales [21], por tal motivo, al tomar una fotografía de un aparato de TV, es necesario aislar al aparato de posibles reflejos producidoss por la luz proveniente de objetos cercanos. Lo más conveniente es situar dicho aparato en un cuarto obscuro. Por otra parte, es necesario cuidar la velocidad de obturación con que es tomada la fotografía (ya que las imágenes en los aparatos de TV comunes son formadas en 1/30 de segundo), de tal forma que la velocidad de obturación permita que la imagen se forme una vez al menos.

Usando una película de sensibilidad media (ASA 100) es conveniente ajustar velocidad de obturación a 1/15 de segundo con una abertura del diafragma (foco) 2, para la obtención de imágenes aceptables. Debe mencionarse, que es muy necesario utilizar un tripie para mantener estable la cámara al realizar el disparo y evitar con ello exposiciones que traigan como consecuencias fotografías de poca calidad (borrosas).

Finalmente, para ayudar a mejorar la calidad de las imágenes, es conveniente ajustar el contraste de la imagen a un nivel un poco más bajo al normal y subir el brillo a un nivel en donde se puedan observar detalles de la imagen; en algunas ocasiones, dependiendo de la imagen que se tenga, es necesario colocar el control del brillo a su nivel máximo.

A P E N D I C E C

CODIFICACION DE LOS MODULOS DE LOS PROGRAMAS EN LENGUAJE DE ALTO NIVEL

APENDICE C. CODIFICACION DE LOS MODULOS DE LOS PROGRAMAS EN LENGUAJE DE ALTO NIVEL.

A continuación se presenta la codificación de las subrutinas que constituyen el programa MICD. La codificación se llevó a cabo en lenguaje C (Turbo C versión 2.0). La codificación de todos los programas que se realizaron en este trabajo serán proporcionados a quien los solicite en el la Sección de Eléctrica de la D.E.P.F.I.

```
/* FILE: MGL.H                                                          */
/************************************************************************/
/*                      MODULO GENERAL DE LIBRERIAS                     */
/************************************************************************/

/* LIBRERIA PARA OPERACIONES MATEMATICAS                                */

#include <math.h>


/*FILE: MGD.H                                                           */
/************************************************************************/
/*                  MODULO GENERAL DE DEFINICIONES                      */
/************************************************************************/


#define MAX_PIXELES        256       /* NUMERO DE COLUMNAS Y RENGLONES  */
#define TOTAL_PIXELES    65536L      /* TOTAL DE PIXELES DE UNA IMAGEN  */
#define BLOCK_SIZE           2       /* BLOQUES PARA PROCESAR LA IMAGEN */
#define DIF_MIN           -256       /* LIMITE INFERIOR (entero)        */
#define FDIF_MIN          -256.0     /* LIMITE INFERIOR (flotante)      */
#define ORIGEN_MENU        '-'       /* PARA EJECUCION DESDE EL SISTEMA
                                        OPERATIVO                       */

/* FILE: M_I_O.H                                                        */
/************************************************************************/
/*              DEFINICIONES DEL MODULO DE ENTRADA / SALIDA             */
/************************************************************************/


#define  FARCALLOC(X) X = farcalloc(TOTAL_PIXELES,sizeof(unsigned char))
#define  CALLOC(X) X    = (unsigned char *) calloc((int)
                          (TOTAL_PIXELES/BLOCK_SIZE), sizeof(unsigned char))


/* LECTURA DE LA IMAGEN                                                 */

int  lectura_ima(FILE               *fentrada,
                 unsigned char far  *imagen_buffer,
                 unsigned char      *buff_temp);


/* ESCRITURA DE LA IMAGEN                                               */

int  escritura_ima(FILE               *fsalida,
                   unsigned char far  *imagen_buffer,
                   unsigned char      *buff_temp);
```

```
/* FILE: MMP.H                                                          */
/************************************************************************/
/*              DEFINICIONES PARA LA RUTINA cdpcm y ddpcm               */
/************************************************************************/


/* NUMERO DE PUNTOS EN LA VECINDAD LOCAL                                */

#define  MAX_VECINOS          8
#define  INICIA_LINEA         2


/* OPCIONES PARA CONFIGURACION DEL SISTEMA                              */

#define  TIPO_0              48
#define  TIPO_1              49
#define  TIPO_2              50
#define  TIPO_3              51
#define  TIPO_4              52
#define  TIPO_5              53


/* DEFINICIONES PARA LA RUTINA orientacion_local_x_testada              */

#define  MAXK                 4
#define  CONDICION(a,b)       ((a<0) && (b<0) || (a>=0) && (b>=0))
#define  DIF_MAX            300.0


/* DEFINICIONES PARA LA RUTINA prediccion_orientacion_local_x           */

#define  UMBRAL_CONTORNO    25.0


/* DECLARACION DE FUNCIONES                                             */


/* RUTINA DEL BLOQUE DE CODIFICACION  MICD                              */

int  cdpcm(unsigned char far   *buffer_e_s,
           unsigned char far   *buffer_s);

/* RUTINA DEL BLOQUE DE DECODIFICACION  MICD                            */

int  ddpcm(unsigned char far   *buffer_e_s);

/* DETERMINACION DE LA ORIENTACION LOCAL DEL PIXEL: X_TESTADA           */

int  orientacion_local_x_testada(float   x_testada,
                                 float   *vecino);
```

```
/* BUSQUEDA BINARIA PARA ENCONTRAR LA ORIENTACION Y A PARTIR DE ELLA
   DETERMINAR LA PREDICCION                                          */

int  prediccion_search(int   *dir_base,
                       int   *minptr,
                       int   *maxptr,
                       int   orientacion);


/* RUTINA QUE DEVUELVE LA ORIENTACION PROBABLE DEL PUNTO A PREDECIR  */

int  prediccion_orientacion_local_x(int   *ang_a,
                                    int   *ang_p);

/* INICIALIZACION DE LA VECINDAD LOCAL                               */

void  inicializa_vecindad_local(float   *val_actual,
                                float   *val_precedente,
                                int     posicion);


/* PREDICCION FIJA                                                   */

void  prediccion_fija(float   *p);

/* PREDICCION ADAPTABLE DE Graham                                    */

void  prediccion_graham(float   *p);

/* PREDICCION ADAPTABLE  Zschunke  REVISADO POR Kretz                */

void  prediccion_zk(float   *p);

/* PREDICCION ADAPTABLE  Zschunke  REVISADO POR Dewitte              */

void  prediccion_zd(float   *p);

/* PREDICTOR ADAPTABLE D*                                            */

void  prediccion_d(float   *p);

/* PREDICCION ADAPTABLE D2                                           */

void  prediccion_d1(float   *p);

/* RUTINA QUE CALCULA LA PREDICION DEL PIXEL                         */

void  predictor_adaptable(int     orientacion,
                          float   *vecino,
                          float   *p);
```

```
/* FILE: MCCF.H                                                          */
/*************************************************************************/
/*    DECLARACIONES Y DEFINICIONES PARA EL MODULO DE CUANTIFICACION FIJA  */
/*            Y CODIFICACION CON LONGITUD DE PALABRA CONSTANTE            */
/*************************************************************************/


/* MAXIMO NUMERO DE NIVELES DE RECOONSTRUCCION Y DE CODIFICACION PARA LOS
   CUANTIFICADORES                                                       */

#define  MAX_NIVELES   17

/* DEFINICION DE LA ESTRUCTURA EN DONDE SE ALMACENAN LAS CARACTERISTICAS
   DE UN CUANTIFICADOR FIJO                                              */

typedef struct  CUANTIFICADOR_F
  {
    unsigned char    numero_niveles;
    unsigned char    codigo_levels[MAX_NIVELES];
    float            decision_levels[MAX_NIVELES+1];
    float            rec_levels[MAX_NIVELES];
  }CUANTIFICADOR_F;


/* DEFINICION EXTERNA DE LOS CUANTIFICADORES IMPLANTADOS                 */

extern struct  CUANTIFICADOR_F  cuan_fijo_11,
                                cuan_fijo_16,
                                cuan_fijo_17,
                                cuan_pqc_111,
                                cuan_pqc_112,
                                cuan_pqc_113,
                                cuan_pqc_114;


/* DECLARACION DE LAS RUTINAS DE CUANTIFICACION FIJA Y CODIFICACION      */


/* CUANTIFICACION FIJA                                                   */

void  cuantificador_f(CUANTIFICADOR_F   *ptr_cuantificador,
                      float             *delta,
                      float             *delta_testada,
                      unsigned char     *codigo);

/* CODIFICACION PARA EL CUANTIFICADOR FIJO                               */

void  d_cuantificador_f(CUANTIFICADOR_F   *ptr_cuantificador,
                        float             *delta_testada,
                        unsigned char     codigo);
```

```
/* BUSQUEDA BINARIA DE NIVELES DE RECONSTRUCCION A PARTIR DE LOS NIVELES
   DE DECISION                                                          */

int  level_search(float   *dir_base,
                  float   *minptr,
                  float   *maxptr,
                  float   value);


/* BUSQUEDA BINARIA DE NIVELES DE RECONSTRUCCION A PARTIR DE LOS CODIGOS
   DE TRANSMISION                                                       */

int  code_search(unsigned char   *dir_base,
                 unsigned char   *minptr,
                 unsigned char   *maxptr,
                 unsigned char   codigo);



/* FILE: MCCD.H
/*********************************************************************/
/*      DECLARACION DE LAS RUTINAS DE CODIFICACION Y DECODIFICACION DEL  */
/*                        CUANTIFICADOR DESLIZABLE                       */
/*********************************************************************/


/* RUTINA DE CODIFICACION                                               */

void  cuantificador_deslizable(float          p,
                               float          delta,
                               float          *delta_testada,
                               unsigned char  *codigo);


/* RUTINA DE DECODIFICACION                                             */

void  dcuantificador_deslizable(float          p,
                                float          *delta_testada,
                                unsigned char  codigo);
```

```
/* FILE: MCCL.H                                                          */
/************************************************************************/
/*       DECLARACION DE LAS RUTINAS PARA EL MODULO DE CUANTIFICACION Y  */
/*                    CODIFICACION POR LUMINANCIA                       */
/************************************************************************/


/* BLOQUE DE CODIFICACION                                                */

int  cuantificador_por_luminancia(float          p,
                                  float          *delta,
                                  float          *delta_testada,
                                  float          *val_actual,
                                  float          *val_precedente,
                                  int            posicion,
                                  unsigned char  *codigo);


/* BLOQUE DE DECODIFICACION                                              */

int  dcuantificador_por_luminancia(float         p,
                                   float         *delta_testada,
                                   float         *val_actual,
                                   float         *val_precedente,
                                   int           posicion,
                                   unsigned char codigo);


/* FILE: MCFG.H                                                          */
/************************************************************************/
/*     DEFINICIONES PARA LA CONFIGURACION DE LAS RUTINAS  cdpcm y ddpcm  */
/************************************************************************/


typedef struct  DPCM_CFG
  {
    /* TIPO DE PREDICTOR A USAR                                          */

    unsigned char    tipo_pred;

    /* TIPO DE CUANTIFICADOR A USAR                                      */

    unsigned char    tipo_cuan;

    /* NUMERO DE BITS A USAR EN EL CUANTIFICADOR DESLIZABLE              */

    unsigned char    n_bits;

    /* NUMERO DE CUANTIFICADORES FIJOS DEL CUANTIFICADOR DESLIZABLE      */

    unsigned char    n_cuan;
```

```
    /* BANDERA                                                    */
    unsigned char    bandera;
  }DPCM_CFG;

/* DECLARACION DE VARIABLES EXTERNAS                             */
extern struct  DPCM_CFG    dpcm_cfg;
void  inicializa_micd(char  *imagen_ori);



/* FILE: MIE.H                                                   */
/*****************************************************************/
/*          DECLARACIONES PARA EL MODULO DE INTERFAZ EXTERNA     */
/*****************************************************************/

/* RUTINA PARA TERMINAR LA EJECUCION                             */
void  termina_ejecucion(char   origen);
/* RUTINA PARA LOS MENSAJES DE ERROR                             */
void  mensaje_error(char  *mensaje_error);

/* RUTINA PARA INICIAR LA EJECUCION                              */
int  inicia_ejecucion(char  *argv0,
                      char  *argv1,
                      char  *argv2,
                      char  *argv3);
```

```
/* FILE: MIO.C                                                          */
/************************************************************************/
/*                      MODULO DE ENTRADA / SALIDA                      */
/************************************************************************/


#include <process.h>
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <stdlib.h>
#include <dos.h>
#include <mem.h>
#include <conio.h>

#include "m_i_o.h"    /* ENCABEZADO PARA EL ARCHIVO DE ENTRADA / SALIDA  */
#include "mgl.h"      /* PARA LIBRERIAS GENERALES                        */
#include "mgd.h"      /* PARA DEFINICIONES GENERALES                     */
#include "mmp.h"      /* PARA DEFINICION DE LAS RUTINAS cdpcm y ddpcm     */

/* VARIABLE EXTERNA                                                     */

exter  char    origen;

/* INICIALIZACION DE MICD, OPCION A EJECUTAR                            */

void  inicializa_micd(char   *imagen_ori)
  {
    unsigned char         *buff_temp;

    static char           imagen_ori[13],
                          imagen_cod[13],
                          imagen_err[13],
                          imagen_dec[13],
                          seleccion;

    unsigned char far     *imagen_buffer1;
                          *imagen_buffer2;

    int                   ocurre_punto;

    FILE                  *f_imagen_ori,
                          *f_imagen_cod,
                          *f_imagen_err,
                          *f_imagen_dec;


    printf("Codificar             [c]");
    printf("Decodificar           [d]");
    printf("Simular transmision   [t]");
```

```
seleccion = getch();
ocurre_punto = strcspn(imagen_ori, ".");
switch(seleccion)
  {
    case 'C' :
    case 'c' :  strncpy(imagen_cod, imagen_ori, ocurre_punto + 1);
                strncat(imagen_cod, "cod", 3);
                strncpy(imagen_err, imagen_ori, ocurre_punto + 1);
                strncat(imagen_err, "err", 3);
                if((f_imagen_ori = fopen(imagen_ori, "rb")) == NULL)
                  {
                    mensaje_error("Error al abrir archivo de entrada.");
                    termina_ejecucion(origen);
                  }
                if((f_imagen_cod = fopen(imagen_cod, "wb")) == NULL)
                  {
                    mensaje_error("Error al abrir archivo de
                                                        codificacion.");
                    termina_ejecucion(origen);
                  }
                if((f_imagen_err = fopen(imagen_err, "wb")) == NULL)
                  {
                    mensaje_error("Error al abrir archivo de error.");
                    termina_ejecucion(origen);
                  }
                if((FARCALLOC(imagen_buffer1)) == NULL)
                  {
                    mensaje_error("Error: no se pudo reservar memoria
                                                        para la imagen.");
                    termina_ejecucion(origen);
                  }
                if((FARCALLOC(imagen_buffer2)) == NULL)
                  {
                    mensaje_error("Error: no se pudo reservar memoria
                                                        para la imagen.");
                    termina_ejecucion(origen);
                  }
                if((CALLOC(buff_temp)) == NULL)
                  {
                    mensaje_error("Error: no se pudo reservar memoria
                                                        para la imagen");
                    termina_ejecucion(origen);
                  }
                lectura_ima(f_imagen_ori, imagen_buffer1, buff_temp);
                cdpcm(imagen_buffer1, imagen_buffer2);
                escritura_ima(f_imagen_cod, imagen_buffer1, buff_temp);
                escritura_ima(f_imagen_err, imagen_buffer2, buff_temp);
                fclose(f_imagen_ori);
                fclose(f_imagen_cod);
                fclose(f_imagen_err);
                break;
```

```
        case 'D' :
        case 'd' :  if((FARCALLOC(imagen_buffer1)) == NULL)
                      {
                        mensaje_error("Error: no se pudo reservar memoria
                                                        para la imagen.");
                        termina_ejecucion(origen);
                      }
                    if((CALLOC(buff_temp)) == NULL)
                      {
                        mensaje_error("Error: no se pudo reservar memoria
                                                        para la imagen");
                        termina_ejecucion(origen);
                      }
                    strncpy(imagen_cod, imagen_ori, ocurre_punto + 1);
                    strncat(imagen_cod, "cod", 3);
                    strncpy(imagen_dec, imagen_ori, ocurre_punto + 1);
                    strncat(imagen_dec, "dec", 3);
                    if((f_imagen_dec = fopen(imagen_dec, "wb")) == NULL)
                      {
                        mensaje_error("Error al abrir archivo de
                                                    decodificacion.");
                        termina_ejecucion(origen);
                      }
                    if((f_imagen_cod = fopen(imagen_cod, "rb")) == NULL)
                      {
                        mensaje_error("Error al abrir archivo de
                                                        codificacion.");
                        termina_ejecucion(origen);
                      }
                    lectura_ima(f_imagen_cod, imagen_buffer1, buff_temp);
                    ddpcm(imagen_buffer1);
                    escritura_ima(f_imagen_dec, imagen_buffer1, buff_temp);
                    fclose(f_imagen_cod);
                    fclose(f_imagen_dec);
                    break;
        default   : mensaje_error("Seleccion no implantada.");
                    termina_ejecucion(origen);
    }
}

/* RUTINA DE LECTURA DE LA IMAGEN                                          */

int  lectura_ima(FILE                 *fentrada,
                 unsigned char far    *imagen_buffer,
                 unsigned char        *buff_temp)
{
  unsigned          imag_buff_seg,
                    imag_buff_off,
                    buff_temp_seg,
                    buff_temp_off;
```

```
    unsigned int      bloque = (unsigned int) (TOTAL_PIXELES / BLOCK_SIZE);

    int               i = 0;

    unsigned long     bytes = 0;


    imag_buff_seg = FP_SEG(imagen_buffer);
    imag_buff_off = FP_OFF(imagen_buffer);
    buff_temp_seg = FP_SEG((void far *) buff_temp);
    buff_temp_off = FP_OFF((void far *) buff_temp);
    for(; i < BLOCK_SIZE; i++)
      {
        bytes += fread(buff_temp,
                       sizeof(unsigned char far),
                       bloque,
                       fentrada);
        movedata(buff_temp_seg,
                 buff_temp_off,
                 imag_buff_seg,
                 imag_buff_off + bloque * i,
                 bloque);
      }
    return(bytes != TOTAL_PIXELES)
           ? -1
           : 0;
  }

 * RUTINA DE ESCRITURA DE LA IMAGEN                                    */

int  escritura_ima(FILE                *fsalida,
                   unsigned char far   *imagen_buffer,
                   unsigned char       *buff_temp)
  {
    unsigned          imag_buff_seg,
                      imag_buff_off,
                      buff_temp_seg,
                      buff_temp_off;

    int               i = 0,
                      bloque = (int) (TOTAL_PIXELES / BLOCK_SIZE);

    unsigned long     bytes = 0;


    imag_buff_seg = FP_SEG(imagen_buffer);
    imag_buff_off = FP_OFF(imagen_buffer);
    buff_temp_seg = FP_SEG((void far *) buff_temp);
    buff_temp_off = FP_OFF((void far *) buff_temp);
```

```
    for(; i < BLOCK_SIZE; i++)
      {
        movedata(imag_buff_seg,
                 imag_buff_off + bloque * i,
                 buff_temp_seg,
                 buff_temp_off,
                 bloque);
        bytes += fwrite(buff_temp,
                        sizeof(unsigned char far),
                        bloque,
                        fsalida);
      }
    return(bytes != TOTAL_PIXELES )
          ? -1
          : 0;
  }


/* FILE: MMP.C                                                              */
/****************************************************************************/
/*                        MODULO DE MAPEO PREDICTIVO                        */
/****************************************************************************/

#include "mgd.h"    /* PARA DEFINICIONES GENERALES                          */
#include "mgl.h"    /* PARA LIBRERIAS GENERALES                             */
#include "mcfg.h"   /* PARA DEFINICION DE COEFICIENTES DEL PREDICTOR FIJO    */
#include "mmp.h"    /* ENCABEZADO DEL ARCHIVO DE  MAPEO  PREDICTIVO         */
#include "mccf.h"   /* PARA DEFINICION DE CUANTIFICADOR FIJO                */
#include "mccl.h"   /* PARA DEFINICION DE CUANTIFICADOR POR LUMINANCIA       */
#include "mccd.h"   /* PARA DEFINICION DE CUANTIFICADOR DESLIZABLE           */

float           a,
                ab,
                b,
                c,
                d,
                e;

static char     imagen[50];

static int      ang_actual[MAX_PIXELES + 5],
                ang_precedente[MAX_PIXELES + 5],
                posicion = INICIA_LINEA;

static float    vecino[MAX_VECINOS],
                val_actual[MAX_PIXELES + 5],
                val_precedente[MAX_PIXELES + 5];
```

```
/***************** RUTINA:  BLOQUE DE CODIFICACION  MICD ********************/

int  cdpcm(unsigned char far   *buffer_e_s,
           unsigned char far   *buffer_s)
  {
    unsigned char       codigo = 0;

    int                 n_pixeles = MAX_PIXELES,
                        i = 0;

    float               x = 0.0,
                        x_testada = 0.0,
                        delta = 0.0,
                        delta_testada = 0.0,
                        p = 0.0;

    unsigned long       j = 0L;

    CUANTIFICADOR_F     *ptr_cuantificador;     /* APUNTADOR A LOS DIFERENTES
                                                   CUANTIFICADORES FIJOS.    */

    void  (*prediccion_ptr) (float *);          /* APUNTADOR A LAS DIFERENTES
                                                   RUTINAS DE PREDICCION.    */


    /* SELECCION DEL CUANTIFICADOR FIJO, SI ESTA IMPLANTADO                   */

    switch(dpcm_cfg.tipo_cuan)
      {
        case TIPO_0 :  ptr_cuantificador = &cuan_fijo_11;
                       break;
        case TIPO_1 :  ptr_cuantificador = &cuan_fijo_16;
                       break;
        case TIPO_2 :  ptr_cuantificador = &cuan_fijo_17;
                       break;
        case TIPO_3 :
        case TIPO_4 :
                       break;
        default     :  return(-1);
      }

    /* SELECCION DEL PREDICTOR, SI ESTA IMPLANTADO                            */

    switch(dpcm_cfg.tipo_pred)
      {
        case TIPO_0 :  prediccion_ptr = prediccion_fija;
                       break;
        case TIPO_1 :  prediccion_ptr = prediccion_graham;
                       break;
```

```
    case TIPO_2 :  prediccion_ptr = prediccion_zk;
                   break;
    case TIPO_3 :  prediccion_ptr = prediccion_zd;
                   break;
    case TIPO_4 :  prediccion_ptr = prediccion_d;
                   break;
    case TIPO_5 :  prediccion_ptr = prediccion_dl;
                   break;
    default     :  return(-1);
  }

/* INICIO DEL BLOQUE DE MAPEO  MICD                                        */

for(; j < TOTAL_PIXELES ; j++)
  {
    /* LAS LINEAS DE VALOR Y ANGULO ACTUAL SE ASIGNAN A LAS LINEAS DE
       VALOR Y ANGULO PRECEDENTE RESPECTIVAMENTE                           */

    if(posicion > n_pixeles + 1)
      {
        posicion = INICIA_LINEA;
        for( i = INICIA_LINEA; i <= n_pixeles + 1; i++)
          {
            *(val_precedente + i) = *(val_actual + i);
            *(ang_precedente + i) = *(ang_actual + i);
          }
        p = *(val_precedente + posicion);
      }

    /* LECTURA DEL PIXEL                                                   */

    x =  *(buffer_e_s + (unsigned) j);

    /* OBTENCION DEL ERROR DE PREDICCION                                   */

    delta = x - p;

    /* SELECCION DEL CUANTIFICADOR ADAPTABLE, SI ESTA IMPLANTADO           */

    switch(dpcm_cfg.tipo_cuan)
      {
        case TIPO_3 :  cuantificador_deslizable(p,
                                                delta,
                                                &delta_testada,
                                                &codigo);
                       break;
```

```
        case TIPO_4 :  cuantificador_por_luminancia(p,
                                                    &delta,
                                                    &delta_testada,
                                                    val_actual,
                                                    val_precedente,
                                                    posicion,
                                                    &codigo);
                       break;
        default     :  cuantificador_f(ptr_cuantificador,
                                       &delta,
                                       &delta_testada,
                                       &codigo);
      }

    /* CALCULO DEL PIXEL RECONSTRUIDO                                    */

    x_testada = p + delta_testada;

    /* SALVAR EN ARREGLOS CODIGOS A ENVIAR Y LA  IMAGEN DE ERROR DE
       PREDICCION                                                        */

    *(buffer_e_s + (unsigned) j) = codigo;
    *(buffer_s + (unsigned) j) = (unsigned char) fabs(delta_testada + 80);

    /* ACTUALIZACION DEL VALOR                                           */

    *(val_actual + posicion) = x_testada;

    /* ACTUALIZACION DEL ANGULO                                          */

    if((dpcm_cfg.tipo_pred == TIPO_2) || (dpcm_cfg.tipo_pred == TIPO_3)
      ||(dpcm_cfg.tipo_pred == TIPO_4) || (dpcm_cfg.tipo_pred == TIPO_5))
      *(ang_actual + posicion) = orientacion_local_x_testada(x_testada,
                                                             vecino);

    /* INCREMENTO DE  posicion  PARA APUNTAR AL NUEVO PIXEL              */

    posicion++;
    inicializa_vecindad_local(val_actual, val_precedente, posicion);

    /* CALCULO DE LA PREDICCION                                          */

    (*prediccion_ptr) (&p);
  }
 return(0);
}
```

```
/************** RUTINA: BLOQUE DE DECODIFICACION  MICD ********************/

int ddpcm(unsigned char far   *buffer_e_s)
  {
    unsigned char       codigo = 0;

    int                 n_pixeles = MAX_PIXELES,
                        i = 0;

    float               x_testada = 0.0,
                        delta_testada = 0.0,
                        p = 0.0;

    unsigned long       j = 0L;

    CUANTIFICADOR_F     *ptr_cuantificador;     /* APUNTADOR A LOS DIFERENTES
                                                   CUANTIFICADORES FIJOS.     */

    void  (*prediccion_ptr) (float *);          /* APUNTADOR A LAS DIFERENTES
                                                   RUTINAS DE PREDICCION.     */


    /* SELECCION DEL CUANTIFICADOR FIJO, SI ESTA IMPLANTADO                   */

    switch(dpcm_cfg.tipo_cuan)
      {
        case TIPO_0 :   ptr_cuantificador = &cuan_fijo_11;
                        break;
        case TIPO_1 :   ptr_cuantificador = &cuan_fijo_16;
                        break;
        case TIPO_2 :   ptr_cuantificador = &cuan_fijo_17;
                        break;
        case TIPO_3 :
        case TIPO_4 :
                        break;
        default     :   return(-1);
      }

    /* SELECCION DEL PREDICTOR, SI ESTA IMPLANTADO                            */

    switch(dpcm_cfg.tipo_pred)
      {
        case TIPO_0 :   prediccion_ptr = prediccion_fija;
                        break;
        case TIPO_1 :   prediccion_ptr = prediccion_graham;
                        break;
        case TIPO_2 :   prediccion_ptr = prediccion_zk;
                        break;
        case TIPO_3 :   prediccion_ptr = prediccion_zd;
                        break;
```

```
    case TIPO_4 :   prediccion_ptr = prediccion_d;
                    break;
    case TIPO_5 :   prediccion_ptr = prediccion_d1;
                    break;
    default     :   return(-1);
  }

/* INICIO DEL BLOQUE DE MAPEO INVERSO  MICD                              */

for(; j < TOTAL_PIXELES ; j++)
  {
    /* LAS LINEAS DE VALOR Y ANGULO ACTUAL SE ASIGNAN A LAS LINEAS DE
       VALOR Y ANGULO PRECEDENTE RESPECTIVAMENTE                         */

    if(posicion > n_pixeles + 1)
      {
        posicion = INICIA_LINEA;
        for( i = INICIA_LINEA; i <= n_pixeles + 1; i++)
          {
            *(val_precedente + i) = *(val_actual + i);
            *(ang_precedente + i) = *(ang_actual + i);
          }
        p = *(val_precedente + posicion);
      }

    /* LECTURA DEL CODIGO DEL ERROR DE PREDICCION                        */

    codigo =  *(buffer_e_s + (unsigned) j);

    /* SELECCION DEL CUANTIFICADOR ADAPTABLE, SI ESTA IMPLANTADO         */

    switch(dpcm_cfg.tipo_cuan)
      {
        case TIPO_3 :  dcuantificador_deslizable(p,
                                                 &delta_testada,
                                                 codigo);
                       break;
        case TIPO_4 :  dcuantificador_por_luminancia(p,
                                                     &delta_testada,
                                                     val_actual,
                                                     val_precedente,
                                                     posicion,
                                                     codigo);
                       break;
        default     :  d_cuantificador_f(ptr_cuantificador,
                                         &delta_testada,
                                         codigo);
      }
```

```
        /* CALCULO DEL PIXEL RECONSTRUIDO                                    */

        x_testada = p + delta_testada;

        /* SALVAR EN UN ARREGLO LA IMAGEN DECODIFICADA                       */

        *(buffer_e_s + (unsigned) j) = (unsigned char) fabs(x_testada);

        /* ACTUALIZACION DEL VALOR                                           */

        *(val_actual + posicion) = x_testada;

        /* ACTUALIZACION DEL ANGULO                                          */

        if((dpcm_cfg.tipo_pred == TIPO_2) || (dpcm_cfg.tipo_pred == TIPO_3)
          ||(dpcm_cfg.tipo_pred == TIPO_4) || (dpcm_cfg.tipo_pred == tipo 5))
          *(ang_actual + posicion) = orientacion_local_x_testada(x_testada,
                                                                 vecino);

        /* INCREMENTO DE  posicion  PARA APUNTAR AL NUEVO PIXEL              */

        posicion++;
        inicializa_vecindad_local(val_actual,val_precedente,posicion);

        /* CALCULO DE LA PREDICCION                                          */

        (*prediccion_ptr) (&p);
      }
    return(0);
  }

/* SE INICIALIZA LA VECINDAD LOCAL                                           */

void  inicializa_vecindad_local(float   *val_actual,
                                float   *val_precedente,
                                int     posicion)
  {
    a  = *(val_actual + posicion - 1);
    ab = *(val_precedente + posicion - 2);
    b  = *(val_precedente + posicion - 1);
    c  = *(val_precedente + posicion);
    d  = *(val_precedente + posicion + 1);
    e  = *(val_precedente + posicion + 2);
  }
```

```
/* PREDICCION FIJA                                                    */

void  prediccion_fija(float   *p)
  {
    extern  float    A,AB,B,C,D,E;

    *p = a*A + ab*AB + b*B + c*C + d*D + e*E;
  }

/* PREDICCION ADAPTABLE DE  Graham                                    */

void  prediccion_graham(float   *p)
  {
    *p = (fabs(c - b) <= fabs(a - b))
         ? a
         : c;
  }

/* PREDICCION ADAPTABLE DE  Zschunke  REVISADO POR  Kretz             */

void  prediccion_zk(float   *p)
  {
    int    orientacion;

    *vecino       = a;
    *(vecino + 1) = ab;
    *(vecino + 2) = b;
    *(vecino + 4) = c;
    *(vecino + 3) = (b + c)/2;
    *(vecino + 6) = d;
    *(vecino + 5) = (c + d)/2;
    *(vecino + 7) = e;

    /* CALCULO DE LA PREDICCION                                       */

    orientacion = prediccion_orientacion_local_x(&(ang_actual[posicion - 1]),
                                                 &(ang_precedente[posicion - 2]));
    predictor_adaptable(orientacion, vecino, p);

  }

/* PREDICCION ADAPTABLE  Zschunke  REVISADO POR  DeWitte              */

void  prediccion_zd(float   *p)
  {
    int    orientacion;

    *vecino       = (2 * a + c ) / 3;
    *(vecino + 1) = (2 * a + c ) / 3;
    *(vecino + 2) = (a + b + c ) / 3;
```

```
    *(vecino + 3) = (a + 2 * c ) / 3;
    *(vecino + 4) = (a + 2 * c ) / 3;
    *(vecino + 5) = (a + c + d ) / 3;
    *(vecino + 6) = (a + d + e ) / 3;
    *(vecino + 7) = (2 * a + c + e) / 4;

    /* CALCULO DE LA PREDICCION                                          */

    orientacion = prediccion_orientacion_local_x(&(ang_actual[posicion - 1]),
                                                 &(ang_precedente[posicion - 2]));
    predictor_adaptable(orientacion, vecino, p);
  }

/* PREDICCIONES PARA EL ALGORITMO  D*                                    */

void  prediccion_d(float   *p)
  {
    int     orientacion;

    *vecino       = a;
    *(vecino + 1) = ab;
    *(vecino + 2) = b;
    *(vecino + 4) = c;
    *(vecino + 3) = (b + c) / 2;
    *(vecino + 6) = d;
    *(vecino + 5) = (c + d) / 2;
    *(vecino + 7) = e;
    orientacion = prediccion_orientacion_local_x(&(ang_actual[posicion - 1]),
                                                 &(ang_precedente[posicion - 2]));
    *p = (orientacion < 0)
         ? (a + d) / 2
         : (a + c) / 2;
  }

/* PREDICCIONES PARA EL ALGORITMO  D2                                    */

void  prediccion_d1(float   *p)
  {
    int     orientacion;

    *vecino       = (2 * a + c) / 3;
    *(vecino + 1) = (2 * a + c) / 3;
    *(vecino + 2) = (a + b + c) / 3;
    *(vecino + 3) = (a + 2 * c) / 3;
    *(vecino + 4) = (a + 2 * c) / 3;
    *(vecino + 5) = (a + c + d) / 3;
    *(vecino + 6) = (a + d + e) / 3;
    *(vecino + 7) = (2 * a + c + e) / 4;
    orientacion = prediccion_orientacion_local_x(&(ang_actual[posicion - 1]),
                                                 &(ang_precedente[posicion - 2]));
```

```
    *p = (orientacion < 0)
         ? (a + d) / 2
         : (a + c) / 2;
  }

/* CALCULO DE LA PREDICCION ADAPTABLE                                    */

void  predictor_adaptable(int     orientacion,
                          float   *vecino,
                          float   *p)
  {
    int     orientaciones[8] = {4,3,2,1,0,-1,-2,-3},
            indice_nivel;

    indice_nivel = prediccion_search(&orientaciones[0],
                                     &orientaciones[0],
                                     &orientaciones[7],
                                     orientacion);
    *p = *(vecino + indice_nivel);
  }

/* BUSQUEDA BINARIA PARA ENCONTRAR LA ORIENTACION Y A PARTIR DE ELLA
   DETERMINAR LA PREDICCION                                              */

int  prediccion_search(int   *dir_base,
                       int   *minptr,
                       int   *maxptr,
                       int   orientacion)
  {
    int     *midptr;

    midptr = minptr + (maxptr - minptr) / 2;
    if(maxptr < minptr)
      return(-1);
    if(*midptr == orientacion)
      return(midptr - dir_base);
    return(*midptr > orientacion)
          ? prediccion_search(dir_base, midptr + 1, maxptr, orientacion)
          : prediccion_search(dir_base, minptr, midptr - 1, orientacion);
  }

/* RUTINA DE PREDICCION MEDIANTE LA ORIENTACION LOCAL EN  X               */
```

```
int  prediccion_orientacion_local_x(int   *ang_a,
                                    int   *ang_p)
 {
   int            ang_phi[6]={4,3,2,0,-2,-3},
                  i = 0,
                  senal = 0,
                  ang_temp = DIF_MIN,
                  dif_ang_temp = (int) DIF_MAX,
                  dif_ang = 0,
                  indice=0;

   static int     ang_teta[6];

   if((*ang_teta = *ang_a) != 4)
     senal=1;
   for(; i < 5; i++)
     {
       if((*(ang_teta + i + 1) = *(ang_p + i)) != 4)
         senal=1;
     }
   if(senal == 0)
     return(4);
   senal = 0;
   for(i = 0; i < 6; i++)
     {
       dif_ang = *(ang_teta +i) - *(ang_phi + i);
       if(*(ang_teta + i) != 4)
         {
           senal++;
           indice = i;
         }
       if(dif_ang == dif_ang_temp)
         ang_temp = (ang_temp < *(ang_teta + i))
                    ? *(ang_teta +i)
                    : ang_temp;
       else
         if(dif_ang_temp > dif_ang)
           {
             ang_temp = *(ang_teta + i);
             dif_ang_temp = dif_ang;
           }
     }
   return (senal == 1)
          ? *(ang_teta + indice)
          : ang_temp;
 }
```

```
/* DETERMINACION DE LA ORIENTACION LOCAL DEL PIXEL: X_TESTADA                */

int  orientacion_local_x_testada(float   x_testada,
                                 float   *vecino)
  {
    int       orientacion = DIF_MIN,
              k = MAXK,
              i = 0;

    float     dif_temp = DIF_MAX,
              diferencia = 0.0,
              dif1 = x_testada - *vecino;

    if(fabs(dif1) <= UMBRAL_CONTORNO)
      return(4);
    for(; i < 8; i++)
      {
        diferencia = fabs(x_testada - *(vecino+i));
        if(i < 2)
          {
            if(diferencia <= 64.0)
              {
                if(dif_temp <= diferencia)
                   orientacion = (orientacion < k-i)
                                 ? (k-i)
                                 : orientacion;
                else
                  {
                    dif_temp= diferencia;
                    orientacion = (k-i);
                  }
              }
          }
        else
          {
            if((diferencia <= 64) &&
                CONDICION(dif1,*(vecino + i) - *(vecino + i - 1)))
              {
                if(dif_temp <= diferencia)
                  orientacion = (orientacion < k-i)
                                ? (k-i)
                                : orientacion;
                else
                  {
                    dif_temp= diferencia;
                    orientacion = (k-i);
                  }
              }
          }
      }
```

```
    if(orientacion == DIF_MIN)
      return(4);
    else
      return(orientacion);
  }



/* FILE: MCCF.C                                                              */
/*****************************************************************************/
/*            MODULO DE CUANTIFICACION Y CODIFICACION FIJA                   */
/*****************************************************************************/


#include "mccf.h"  /* ENCABEZADO DEL ARCHIVO DE CUANTIFICACION Y CODIFICACION
                      FIJA                                                    */
#include "mgl.h"   /* PARA LIBRERIAS GENERALES                                */


/* DEFINICION DE LOS CUANTIFICADORES IMPLANTADOS                             */

/* CUANTIFICADOR FIJO NO UNIFORME DE 11 NIVELES                              */

struct  CUANTIFICADOR_F  cuan_fijo_11=
    {
      11,
      0,1,2,3,4,5,6,7,8,9,10,0,0,0,0,0,0,0,
      -256,-53.5,-36.5,-19.5,-4.5,-1.5,1.5,4.5,19.5,36.5,53.5,256,0,0,0,0,0,0,
      -65,-45,-28,-12,-3,0,3,12,28,45,65,0,0,0,0,0,0
    };

/* CUANTIFICADOR FIJO NO UNIFORME DE 16 NIVELES                              */

struct  CUANTIFICADOR_F  cuan_fijo_16=
    {
      16,
      0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,
      -256,-112,-90,-64,-46,-28,-10,-4,0,6,10,30,46,66,90,114,256,0,
      -126,-101,-77,-55,-37,-19,-7,-2,3,8,20,38,56,78,102,127,0
    };
```

```
/* CUANTIFICADOR FIJO NO UNIFORME DE 17 NIVELES                          */

struct  CUANTIFICADOR_F  cuan_fijo_17=
    {
      17,
      0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,
      -256,-75.5,-56.5,-41.5,-29.5,-19.5,-11.5,-5.5,-1.5,1.5,5.5,11.5,19.5,
          29.5,41.5,56.5,75.5,256,
      -86,-65,-48,-35,-24,-15,-8,-3,0,3,8,15,24,35,48,65,86
    };

/* CUANTIFICADORES FIJOS DE 11 NIVELES NO UNIFORMES PARA EL CUANTIFICADOR
   ADAPTABLE POR FUNCION DE ENMASCARAMIENTO                              */

struct CUANTIFICADOR_F  cuan_pqc_111=
    {
      11,
      0,1,2,3,4,5,6,7,8,9,10,0,0,0,0,0,0,
      -256,-29.5,-19.5,-11.5,-5.5,-1.5,1.5,5.5,11.5,19.5,29.5,256,0,0,0,0,0,0,
      -35,-24,-15,-8,-3,0,3,8,15,24,35,0,0,0,0,0,0
    };

struct CUANTIFICADOR_F  cuan_pqc_112=
    {
      11,
      0,1,2,3,4,5,6,7,8,9,10,0,0,0,0,0,0,
      -256,-40.5,-28.5,-18.5,-10.5,-3.5,3.5,10.5,18.5,28.5,40.5,256,0,0,0,
          0,0,0,
      -47,-34,-23,-14,-7,0,7,14,23,34,47,0,0,0,0,0,0
    };

struct CUANTIFICADOR_F  cuan_pqc_113=
    {
      11,
      0,1,2,3,4,5,6,7,8,9,10,0,0,0,0,0,0,
      -256,-54.5,-41.5,-28.5,-16.5,-5.5,5.5,16.5,28.5,41.5,54.5,256,0,0,0,
          0,0,0,
      -65,-48,-35,-22,-11,0,11,22,35,48,65,0,0,0,0,0,0
    };

struct CUANTIFICADOR_F  cuan_pqc_114=
    {
      11,
      0,1,2,3,4,5,6,7,8,9,10,0,0,0,0,0,0,
      -256,-74.5,-54.5,-37.5,-22.5,-7.5,7.5,22.5,37.5,54.5,74.5,256,0,0,0,
          0,0,0,
         -85,-64,-45,-30,-15,0,15,30,45,64,85,0,0,0,0,0,0
    };
```

```
/* DEFINICION DE LAS RUTINAS DE CUANTIFICACION Y CODIFICACION FIJA          */

/* CUANTIFICACION FIJA                                                      */

void  cuantificador_f(CUANTIFICADOR_F   *ptr_cuantificador,
                      float             *delta,
                      float             *delta_testada,
                      unsigned char     *codigo)
  {
    int       indice_de_nivel;

    float     *intermedio = &(ptr_cuantificador -> decision_levels[0]);

    if((indice_de_nivel = level_search(intermedio,
                                       intermedio,
              &(ptr_cuantificador -> decision_levels[ptr_cuantificador ->
                                       numero_niveles]),
                                       *delta)) == -1)
      {
        printf("Error MICD : abs(delta) > 255 ");
        getch();
        exit(0);
      }
    *delta_testada = (ptr_cuantificador -> rec_levels[indice_de_nivel]);
    *codigo = indice_de_nivel;
  }

/* CODIFICACION PARA EL CUANTIFICADOR FIJO                                  */

void d_cuantificador_f(CUANTIFICADOR_F   *ptr_cuantificador,
                       float             *delta_testada,
                       unsigned char     codigo)
  {
    unsigned char    *intermedio = &(ptr_cuantificador -> codigo_levels[0]);

    int              indice_de_nivel;

    if((indice_de_nivel = code_search(intermedio,
                                      intermedio,
              &(ptr_cuantificador -> codigo_levels[ptr_cuantificador ->
                                      numero_niveles]),
                                      codigo)) == -1)
      {
        printf("Error MICD : codigo de transmision inexistente ");
        getch();
        exit(1);
      }
    *delta_testada = (ptr_cuantificador -> rec_levels[indice_de_nivel]);
  }
```

```
/* BUSQUEDA BINARIA DE NIVELES DE RECONSTRUCCION A PARTIR DE LOS NIVELES
   DE DECISION                                                          */

int level_search (float   *dir_base,
                  float   *minptr,
                  float   *maxptr,
                  float   value)
  {
    float    *midptr;

    if(fabs(value) > 255.0)
      return -1;
    midptr = minptr + (maxptr - minptr) / 2;
    if(*midptr == value)
      return midptr - dir_base - 1;
    if(midptr == minptr)
      return midptr - dir_base;
    return (value < *midptr)
           ? level_search(dir_base,minptr,midptr,value)
           : level_search(dir_base,midptr,maxptr,value);
  }

/* BUSQUEDA BINARIA DE NIVELES DE RECONSTRUCCION A PARTIR DE LOS CODIGOS
   DE TRANSMISION                                                       */

int code_search(unsigned char   *dir_base,
                unsigned char   *minptr,
                unsigned char   *maxptr,
                unsigned char   codigo)
  {
    unsigned char    *midptr;

    midptr = minptr + (maxptr - minptr) / 2;
    if(maxptr < minptr)
      return(-1);
    if(*midptr == codigo)
      return (midptr - dir_base);
    return(*midptr > codigo)
          ? code_search(dir_base, minptr, midptr - 1, codigo)
          : code_search(dir_base, midptr + 1, maxptr, codigo);
  }
```

```
/* FILE: MCCD.C                                                          */
/************************************************************************/
/*          MODULO DE CUANTIFICACION Y CODIFICACION  DESLIZABLE          */
/************************************************************************/


#include "mgl.h"   /* PARA LIBRERIAS GENERALES                          */
#include "mgd.h"   /* PARA DEFINICIONES GENERALES                       */
#include "mcfg.h"  /* PARA LA CONFIGURACION DEL SISTEMA MICD            */
#include "mccd.h"  /* ENCABEZADO DEL ARCHIVO DE CUANTIFICACION Y CODIFICACION
                      DESLIZABLE                                        */

int       nbitsi,
          nbitsc,
          x,
          incremento,
          nivmax,
          nivmin,
          niveles,
          tipo_cuantificador,
          paso;

float     delta_testada;


/* BLOQUE DE CODIFICACION DEL CUANTIFICADOR DESLIZABLE                   */

void  cuantificador_deslizable(float          p,
                               float          delta,
                               float          *delta_testada,
                               unsigned char  *codigo)
  {
    incremento          = MAX_PIXELES / dpcm_cfg.n_cuan;
    tipo_cuantificador  = (MAX_PIXELES - p) / incremento;
    nivmax              = MAX_PIXELES - (dpcm_cfg.n_cuan - 1 -
                            tipo_cuantificador) * incremento;
    nivmin              = tipo_cuantificador * incremento - MAX_PIXELES;
    niveles             = exp(dpcm_cfg.n_bits * log(2));
    paso                = ((nivmax - nivmin) + 2) / niveles;
    *codigo             = (nivmax - delta + 1) / paso;
    *delta_testada      = nivmax - paso * (*codigo) + 1 - (paso / 2);
  }
```

```
/* BLOQUE DE DECODIFICACION DEL CUANTIFICADOR DESLIZABLE                    */

void  dcuantificador_deslizable(float          p,
                                float          *delta_testada,
                                unsigned char  codigo)
  {
    incremento          = MAX_PIXELES / dpcm_cfg.n_cuan;
    tipo_cuantificador  = (MAX_PIXELES - p) / incremento;
    nivmax              = MAX_PIXELES - (dpcm_cfg.n_cuan - 1 -
                              tipo_cuantificador) * incremento;
    nivmin              = tipo_cuantificador * incremento - MAX_PIXELES;
    niveles             = exp(dpcm_cfg.n_bits * log(2));
    paso                = ((nivmax - nivmin) + 2) / niveles;
    *delta_testada      = nivmax - paso * codigo + 1 - (paso / 2);
  }



/* FILE: MCCL.C                                                            */
/*************************************************************************/
/*         MODULO DE CUANTIFICACION Y CODIFICACION POR LUMINANCIA          */
/*************************************************************************/


#include "mgl.h"   /* PARA LIBRERIAS GENERALES                             */
#include "mgd.h"   /* PARA DEFINICIONES GENERALES                          */
#include "mccl.h"  /* ENCABEZADO DEL ARCHIVO DE CUANTIFICACION Y CODIFICACION
                      POR LUMINANCIA                                       */
#include "mccf.h"  /* PARA DEFINICION DE CUANTIFICADORES FIJOS             */


/* BLOQUE DE CODIFICACION DE LA RUTINA DE CUANTIFICACION Y CODIFICACION
   POR LUMINANCIA                                                          */

int cuantificador_por_luminancia(float          p,
                                 float          *delta,
                                 float          *delta_testada,
                                 float          *val_actual,
                                 float          *val_precedente,
                                 int            posicion,
                                 unsigned char  *codigo)
  {
    int          i = 1,
                 j = 1,
                 sgn = 0;
```

```
float           var = 0.0,
                di = 0.0,
                d = 0.0,
                Asd = 0.0,
                Awd = FDIF_MIN,
                Aswd = 0.0;

static float    linea[5];

*(linea + 1) = *(val_actual + posicion - 1);
*(linea + 2) = *(val_precedente + posicion - 1);
*(linea + 3) = *(val_precedente + posicion);
*(linea + 4) = *(val_precedente + posicion + 1);


/* CALCULO DE LA FUNCION DE LUMINANCIA  Asd

for(; i < 5; i++)
  {
    for(j = 1; j < 5; j++)
      {
        if(i != j)
          {
            var = fabs(*(linea + i) - *(linea + j));
            Asd = (var > Asd)
                  ? var
                  : Asd;
          }
      }
  }


/* CALCULO DE LA FUNCION DE LUMINANCIA  Awd

for(i = 1; i < 5; i++)
  {
    di  = *(linea + i) - p;
    d   = (di < 0)
          ? di * -1
          : di;
    sgn = (*delta > 0)
          ? 1
          : -1;
    var = d + (0.25 * (di + d) * (1 - sgn));
    Awd = (var > Awd)
          ? var
          : Awd;
  }
```

```
    /* CALCULO DE LA FUNCION DE LUMINANCIA  Aswd                          */

    Aswd = (Asd > Awd)
           ? Asd
           : Awd;


    /* CRITERIO DE CONMUTACION A LOS CUANTIFICADORES NO UNIFORMES
       cuan_pqc_111, cuan_pqc_112, cuan_pqc_113 y cuan_pqc_114            */

    if(Aswd <= 15.0)
      cuantificador_f(&cuan_pqc_111,
                      delta,
                      delta_testada,
                      codigo);
    else
      if(Aswd > 15.0 && Aswd <= 35.0)
        cuantificador_f(&cuan_pqc_112,
                        delta,
                        delta_testada,
                        codigo);
      else
        if(Aswd > 35.0 && Aswd <= 100.0)
          cuantificador_f(&cuan_pqc_113,
                          delta,
                          delta_testada,
                          codigo);
        else
          if(Aswd > 100.0)
            cuantificador_f(&cuan_pqc_114,
                            delta,
                            delta_testada,
                            codigo);
    return(0);
  }


/* BLOQUE DE DECODIFICACION DE LA RUTINA DE CUANTIFICACION Y CODIFICACION
   POR LUMINANCIA                                                        */

int dcuantificador_por_luminancia(float          p,
                                  float          *delta_testada,
                                  float          *val_actual,
                                  float          *val_precedente,
                                  int            posicion,
                                  unsigned char  codigo)
  {
    int             i = 1,
                    j = 1,
                    sgn = 0;
```

```
float           var = 0.0,
                d1 = 0.0,
                d = 0.0,
                Asd = 0.0,
                Awd = FDIF_MIN,
                Aswd = 0.0;

static float    linea [5];

*(linea + 1) = *(val_actual + posicion - 1);
*(linea + 2) = *(val_precedente + posicion - 1);
*(linea + 3) = *(val_precedente + posicion);
*(linea + 4) = *(val_precedente + posicion + 1);
if(codigo == 5)
  {
    *delta_testada = 0.0;
    return(0);
  }
sgn = (codigo < 5)
       ? -1
       : 1;

/* CALCULO DE LA FUNCION DE LUMINANCIA  Asd                              */

for(; i < 5; i++)
  {
    for(j = 1; j < 5; j++)
      {
        if(i != j)
          {
            var = fabs(*(linea + i) - *(linea + j));
            Asd = (var > Asd)
                   ? var
                   : Asd;
          }
      }
  }

/* CALCULO DE LA FUNCION DE LUMINANCIA  Awd                              */

for(i = 1; i < 5; i++)
  {
    d1  = *(linea + i) - p;
    d   = (d1 < 0)
           ? d1 * -1
           : d1;
    var = d + (0.25 * (d1 + d) * (1 - sgn));
    Awd = (var > Awd)
           ? var
           : Awd;
  }
```

```
    /* CALCULO DE LA FUNCION DE LUMINANCIA  Aswd                          */

    Aswd = (Asd > Awd)
           ? Asd
           : Awd;

    /* CRITERIO DE CONMUTACION A LOS CUANTIFICADORES NO UNIFORMES
       cuan_pqc_111, cuan_pqc_112, cuan_pqc_113 y cuan_pqc_114            */

    if(Aswd <= 15.0)
      d_cuantificador_f(&cuan_pqc_111,
                        delta_testada,
                        codigo);
    else
      if(Aswd > 15.0 && Aswd <= 35.0)
        d_cuantificador_f(&cuan_pqc_112,
                          delta_testada,
                          codigo);
      else
        if(Aswd > 35.0 && Aswd <= 100.0)
          d_cuantificador_f(&cuan_pqc_113,
                            delta_testada,
                            codigo);
        else
          if(Aswd > 100.0)
            d_cuantificador_f(&cuan_pqc_114,
                              delta_testada,
                              codigo);
    return(0);
  }



/* FILE: MCFG.C                                                          */
/*************************************************************************/
/*                  MODULO DE LECTURA DE CONFIGURACION                   */
/*************************************************************************/

#include <process.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>

#include "mcfg.h"     /* ENCABEZADO DEL ARCHIVO DE LECTURA DE CONFIGURACION */
#include "mgd.h"      /* PARA DEFINICIONES GENERALES                        */
```

```
/* COEFICIENTES PARA EL PREDICTOR FIJO                                    */

float    A,AB,B,C,D,E;

/* DECLARACION DE VARIABLES EXTERNAS                                      */

char             origen;

struct  DPCM_CFG  dpcm_cfg =
                          {49,48,0,0,0};

/* PROGRAMA PRINCIPAL, CONFIGURACION DEL SISTEMA                          */

main(int    argc,
     char   *argv[])
  {
    char    **argptr = &argv[1],
            imagen_ori[13];

    FILE    *fcfg;

    origen = (*(argptr + 1))[0];
    strcpy(imagen_ori, *argptr);
    clrscr();
    printf("          Modulacion por impulsos y codificacion
                                                  diferencial ");
    printf("                              Codificacion de imagenes.");
    if(*imagen_ori == NULL)
      {
        printf("Nombre de la imagen de entrada");
        gets(imagen_ori);
      }
    if((strlen(imagen_ori) > 13) || (strchr(imagen_ori, '.') == NULL))
      {
        mensaje_error("Nombre no valido.");
        termina_ejecucion(origen);
      }
    if((fcfg = fopen("micd.cfg", "r")) == NULL)
      {
        mensaje_error("Archivo de configuracion  MICD  no encontrado");
        termina_ejecucion(origen);
      }
    fscanf(fcfg,"%d%d", &dpcm_cfg.tipo_pred, &dpcm_cfg.tipo_cuan);
```

```
    /* EN CASO DE SER EL PREDICTOR FIJO, LECTURA DE LOS VALORES DE LOS
       COEFICIENTES  A, AB, B, C, D  Y  E
       O SI ES EL CASO DEL CUANTIFICADOR DESLIZABLE, LA LECTURA DEL NUMERO
       DE BITS Y EL NUMERO DE CUANTIFICADORES                          */


    if((dpcm_cfg.tipo_pred == 48) || (dpcm_cfg.tipo_cuan == 51))
      fscanf(fcfg, "%f%f%f%f%f%f%d%d", &A, &AB, &B, &C, &D, &E,
             &dpcm_cfg.n_bits, &dpcm_cfg.n_cuan);
    inicializa_micd(imagen_ori);
    fclose(fcfg);
    termina_ejecucion(origen);
  }




/* FILE: MIE.C                                                          */
/************************************************************************/
/*                        MODULO DE INTERFAZ EXTERNA                    */
/************************************************************************/


#include <process.h>
#include <stdarg.h>
#include <stddef.h>
#include <string.h>

#include "mie.h"            /* ENCABEZADO DEL ARCHIVO DE INTERFAZ EXTERNA  */
#include "mmg.h"            /* PARA DEFINICION DE  ESC                     */
#include "mgd.h"            /* PARA DEFINICION DE ORIGEN_MENU              */

/* DEFINICION DE VARIABLE EXTERNA                                          */

extern char origen;

/* MANDA LOS MENSAJES DE ERROR                                             */

void mensaje_error(char   *mensaje_error)
  {
    printf("%s",mensaje_error);
    getch();
  }
```

```
/* RUTINA PARA TERMINAR LA EJECUCION                                    */

void termina_ejecucion(char   origen)
  {
    if(origen != ORIGEN_MENU)
      exit(0);
    else
      if(spawnlp(P_OVERLAY,"spi.exe",NULL) == -1)
        {
          perror("Imposible de ejecutar SPI.EXE Error");
          exit(1);
        }
  }


/* RUTINA PARA INICIAR LA EJECUCION                                     */

int inicia_ejecucion(char   *argv0,
                     char   *argv1,
                     char   *argv2,
                     char   *argv3)
  {
    char    argumento0[13],
            argumento1[13],
            argumento2[13],
            argumento3[13],
            ejecuta[13];

    strcpy(argumento0, argv0);
    strcpy(argumento1, argv1);
    strcpy(argumento2, argv2);
    strcpy(argumento3, argv3);
    if((spawnlp(P_OVERLAY, argumento0, argumento1, argumento1,
                argumento2,argumento3,NULL)) == -1)
      return(-1);
  }
```

APENDICE D

PROGRAMAS SUPLEMENTARIOS : SPI E Y64_2

APENDICE D. PROGRAMAS SUPLEMENTARIOS: SPI E Y64_2.

El programa SPI se desarrolló con el objeto de integrar el conjunto de programas que se han desarrollado en el Laboratorio de Procesamiento de Imágenes de la División de Estudios de Posgrado de la Facultad de Ingeniería. Se constituye como una interfaz entre el usuario y el programa a ejecutar. La transferencia del control entre el programa SPI y el programa a ejecutar se muestra en la figura D.1 .

El programa SPI facilita la utilización del sistema MICD . Con él, es posible seleccionar una de 40 diferentes combinaciones predictor cuantificador para simular una codificación de fuente DPCM intercampo. El programa SPI se encarga de crear el archivo de configuración MICD.CFG utilizado por el programa MICD.EXE . Se encarga además de transferir el control al sistema operativo indicándole si el programa (MICD) ha de ser o no ejecutado.

El programa Y64_2 permite desplegar imágenes codificadas con 8 bits de tamaño 256 x 200 , utilizando el hardware de una tarjeta VGA . Con él es posible observar las imágenes :

a) A 64 niveles de gris.
b) A 64 niveles de rojo, verde o azul.

El programa utiliza un cuantificador uniforme que emplea una búsqueda binaria para encontrar el nivel de reconstrucción. Está compilado con el compilador C de Microsoft, versión 5.1 . Cada color o nivel de gris está especificado como un byte de 8 bits, en donde los dos bits más significativos son cero.

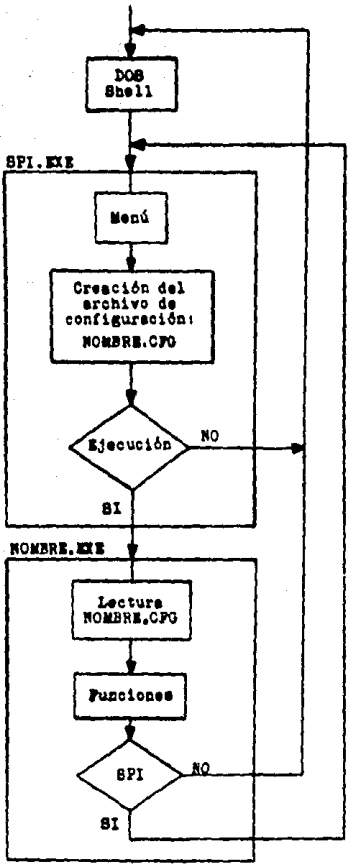


Fig. D.1 Transferencia del control del SPI .

REFERENCIAS

REFERENCIAS.

1. BENSON, K. B. *Television Engieneering Handbook*. USA, Mc Graw Hill, 1986. 37 pp.
2. DEWITTE, J. HENOT, P., J. " *Adaptive prediction DPCM schemes: comparaisons and optimizations*". CCETT. Francia.
3. GONZALEZ, C. Rafael. WINTZ, Paul. *Digital image processing* 2ª ed. U.S.A., Addison-Wesley, 1987, 503 pp.
4. GRAHAM R. E. " *Predictive quantizing of television signals* ". IRE Wescon Convention Record, U.S.A, 1958, 2, Pt 4, pp. 147-157.
5. IKONOMOPOULOUS, A. KUNT, M. "*High compression image coding via directional filtering*". Signal Processing, Vol. 8, Num. 3, mayo de 1985.
6. JAYANT, N. S. NOLL, Peter. *Digital coding of waveforms*. U.S.A., Prentice Hall, 1984, 688 pp.
7. KESKES, N., KRETZ, F., *et.al.* "*Statistical study of edge in TV pictures*". IEEE Trans. Com.,agosto de 1979, Vol. 27, Num. 8 pp. 1239-1247.
8. KNUTSSON, Hans, E. WILSON, Roland. GRANLUND, Goesta, H. "*Anisotropic nonstationary image estimation and its applications: part I - restoration of noisy images*" IEEE Trans. , Com. , Vol. 31, Num. 3, marzo de 1983, pp. 388-397.
9. KNUTSSON, H. WILSON, R. *et.al.* "*Anisotropic nonstationary image estimation and its applications: part II predictive image coding*", IEEE, Trans., Com., Vol. 31, Num. 3, marzo de 1983., pp. 398-406.
10. KRETZ, F. "*Codage MIC différentiel adaptif en télévision: qualité visualle,reduction de débit et susceptibilité aux erreurs de transmission*", Annales des télécommunications, tomo 37, Num. 7-8, julio-agosto de 1982.
11. KUNT, Murat. IKONOMOPOULOUS, Athanassios, *et. al.* "*Second- generation image coding techniques*". Proc. IEEE, Vol 73, Num. 4, abril de 1985. pp. 549-574.

12. MAXEMCHUK (N.F.), STULLER (J.A). *An adaptive intraframe DPCM codec based uppon non stationary image model.* Bell. Syst. Tech. J., U.S.A., (julio-agosto 1979), 58, no. 6, pp. 1395-1412.
13. NETRAVALI, N. A., HASKELL, B. G. *Digital Pictures. Representacion and compression.* U.S.A, AT&T Bell Laboratories, 1988. 586 pp.
14. NETRAVALI, N. Arun. PRASADA B. "*Adaptive quantization of picture signals using spatial masking*". Proc., IEEE, Vol. 65, Num. 4, abril de 1977, pp. 536-548.
15. PRISCH, P. "*Design of DPCM quantizers for video signals using subjective tests*". IEEE, Trans., Com., Vol. 29, Num. 7, julio de 1981.
16. PROAKIS, g. J. DIMITRIS, G. M. *Introduction to digital signal proccessing.* U.S.A, Macmillan Publishing Company. 944 pp.
17. SABATIE, J. *et. al.* "*Coding system for 34 Mbits/s, digital transmission of colour television*". CCETT .
18. SCHÄFER, Ralf. "*Design of adaptive and non adaptive quantizers using subjective criteria*". Signal Processing, Vol.51, Núm. 4, julio de 1983.
19. ZSCHUNKE W. "*Agencement pour la transmission d'images sous forme modulée par des impulsions différentielles codées. Demande de brevet d'invention*", Fr. (julio 1975).
20. ZSCHUNKE W. "*DPCM picture coding with adaptive prediction*" IEEE. Trans. COM, U.S.A. (nov. 1977), 25, n°11, pp 1259-1302.
21. Enciclopedia práctica de fotografía. Tomo 28. Kodak, SALVAT, 1988.