14
24

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGÓN

CODIFICACIÓN DE SECUENCIAS DE IMÁGENES DE TELEVISIÓN UTILIZANDO UN ALGORITMO DE ESTIMACIÓN DE MOVIMIENTO

TESIS QUE PARA OBTENER EL GRADO DE:

INGENIERO EN COMPUTACIÓN

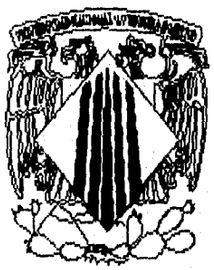
PRESENTA:

CORTES CARMONA, JULIETA

1996

TESIS CON
FALLA DE ORIGEN

14
2y



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
CAMPUS ARAGÓN - ING. EN COMPUTACIÓN

TESIS QUE PARA OBTENER EL TITULO DE **INGENIERO EN COMPUTACIÓN**

PRESENTA: JULIETA CORTÉS CARMONA

DIRECTOR DE TESIS
DR. FRANCISCO J. GARCÍA UGALDE

ESTADO DE MÉXICO **SEPTIEMBRE DE 1996**

CODIFICACIÓN DE SECUENCIAS DE IMÁGENES DE TELEVISIÓN UTILIZANDO UN ALGORITMO DE ESTIMACIÓN DE MOVIMIENTO

SINODALES:

DR. FRANCISCO J. GARCÍA UGALDE
ING. LILIA ENCISO GARCÍA
ING. DONACIANO JIMÉNEZ VÁZQUEZ
ING. DAVID MOISÉS TERAN PÉREZ
ING. SILVIA VEGA MUYTOY

El presente trabajo es dedicado a mis padres y hermanos.

Por su apoyo.

Gracias

Índice

Introducción

La ilusión de movimiento en una escena de televisión es generada por la superposición de imágenes, a una alta velocidad; que varían muy poco entre si. En el Procesamiento Digital de Imágenes cada uno de estos cuadros de movimiento, son conocidos como tramas, las cuales son rejillas cuyas celdas están constituidas por pixeles, que a su vez representan el nivel en la escala de grises por medio de un valor numérico.

Debido a que las tramas son muy similares, la transmisión de los datos de la secuencia de movimiento por medio de métodos directos conlleva a un envío de información redundante y por otra parte la necesidad de hacer un largo reconocimiento entre trama y trama.

Para resolver el problema de la redundancia entre tramas, se han realizado diversos estudios aplicando diferentes métodos entre los cuales se encuentra el de compensación de movimiento.

Este tipo de métodos se basa en lo siguiente:

a) Cada trama de televisión puede ser segmentada en dos partes; la primera donde no hubo movimiento, siendo esta área igual que la de la trama anterior y la otra parte, es donde hubo algún tipo de desplazamiento en la imagen, es decir en las áreas donde existió algún cambio.

b) Del área en movimiento se pueden transmitir dos tipos de información; una puede ser la dirección, en la cual se especifica la localización de los elementos del área en movimiento y la otra, la información para que la intensidad de los elementos de esta área puedan ser reconstruidos.

Con lo anterior podemos observar que la compensación de movimiento consta de dos partes: La primera se refiere a una técnica de **estimación de movimiento**, la misma que predice el desplazamiento de los pixeles entre tramas, por medio del vector de movimiento estimado, el cual indica donde un pixel u objeto estuvo colocado en la trama anterior tomando como referencia la trama actual.

En la segunda parte los vectores de desplazamiento, son utilizados para hacer la **compensación de movimiento**, con la finalidad de realizar la transmisión y recepción de la imagen.

El presente trabajo está enfocado a la codificación de secuencias de imágenes de televisión, en tonos de grises, utilizando algoritmos de estimación de movimiento por Pel-Recursivo para atacar el problema de la redundancia entre tramas.

El método de estimación de movimiento Pel-Recursivo, fue elegido entre los principales métodos de estimación existentes, porque a diferencia de los otros, no existe una transmisión del vector de desplazamiento y por ser uno de los más exactos, debido a que las estimaciones son realizadas pixel por pixel.

La tesis estará estructurada de la siguiente forma:

En el capítulo 1, se tratarán las técnicas básicas de compresión de imágenes, sustentadas en los esquemas de codificación. Las técnicas descritas son utilizadas tanto en imágenes fijas, como en secuencias de imágenes.

Una vez que ha sido ubicado el esquema de codificación a utilizar; el capítulo 2 se enfocará en la parte de estimación de movimiento, donde se

estudiarán dos técnicas principales, que son: Pel-Recursivo y Acoplamiento por Bloques.

En el capítulo 3, profundizaremos en los algoritmos de estimación de movimiento por Pel-Recursivo; mostrando técnicas, basadas en los métodos de Netravali-Robbins y Walker-Rao.

En el capítulo 4; utilizaremos los algoritmos definidos en el capítulo anterior, y se analizarán los algoritmos de transmisión y recepción, aplicados al método de compresión por compensación de movimiento.

La evaluación del desempeño de los diferentes algoritmos Pel-Recursivos, aplicados en la compensación de movimiento, es realizada en el capítulo 5.

Para culminar con nuestro trabajo en el capítulo 6, se proporcionarán las conclusiones de esta investigación.

Capítulo 1

Técnicas de Compresión

La compresión de imágenes consiste en minimizar el número de bits requeridos para representar una imagen, es decir minimizar los datos necesarios para su correcta representación. Tal vez la forma más simple y burda de compresión, es muestrear al límite de banda de una imagen, donde un número infinito de pixeles por unidad de área es reducido a una muestra, sin pérdida de información (suponiendo que se dispone de un filtro paso-bajas ideal). Por consiguiente, y suponiendo siempre lo anterior, el número de muestras por pixel por unidad de área, es reducido infinitamente.

Las aplicaciones de compresión de datos son primordiales en la transmisión y almacenamiento de la información. La transmisión de imágenes se aplica en emisiones de televisión, imágenes vía satélite, conferencias televisivas, comunicación por computadora, transmisión vía fax, y semejantes. Por otra parte el almacenamiento de imágenes es requerido para la educación, cultura y todo aquello que implique el uso de imágenes de alta calidad, así como documentos de trabajo, imágenes médicas que se presentan en tomografía computarizada (CT), imágenes de resonancia magnética (MRT) y radiología digital, imágenes en movimiento, imágenes satelitales, reconocimientos geológicos, etc. En las diferentes aplicaciones de compresión de datos es posible y muy importante el desarrollo de algoritmos rápidos, en los cuales el número de operaciones requeridas reducen el trabajo de compresión

Como las aplicaciones para la compresión de imágenes conllevan un campo muy amplio, varias técnicas de compresión han sido desarrolladas; las mismas que siempre deben de considerar las propiedades estadísticas de las imágenes y las propiedades psicofísicas de la vista humana.

Para realizar una división de las técnicas de compresión, nos basaremos en una clasificación de codificación de imágenes, mostrado en la tabla 1.1. Esta

clasificación se basa en las formas de onda de codificación, las cuales pueden ser distribuidas en 4 categorías principales: Codificación por Modulación de Pulsos (PCM), Codificación Predictiva, Codificación por Transformada y Codificación Interpolativa/Extrapolativa. Además de estas 4 clases existen otros esquemas que no entran en ninguna de las anteriores categorías, pero están diseñados para otro tipo de imágenes. Por ejemplo la Codificación Run Length es usada para imágenes de facsímil y en blanco y negro. Cada una de las clases enunciadas puede subdividirse basándose en si los parámetros de codificación son fijos o adaptativos. En la práctica cualquiera de estas técnicas pueden ser utilizadas para la compresión de imágenes, ya sean estáticas o dinámicas, es decir, secuencias de imágenes; aunque algunos métodos son especialmente elaborados para secuencias de imágenes como es el de estimación de movimiento, el cual pertenece al esquema de Codificación Predictiva.

En la Codificación por Modulación de Pulsos (PCM), que también es conocida como conversión analógico/digital o A/D, en tiempo y amplitud discreta, la representación de los elementos de la imagen (pixeles) son hechos sin eliminar en demasía la redundancia estadística o perceptual de la señal. El tiempo discreto es proporcionado muestreando la señal generalmente a la frecuencia de Nyquist[1], mientras que la amplitud discreta es proporcionada usando un número suficiente de niveles de cuantización de modo que la degradación debida a la cuantización sea tolerable.

En la Codificación Predictiva, se intenta predecir el pixel a ser codificado. La predicción es hecha usando el valor de codificación del pixel anterior transmitido (o los pixeles ya codificados), y solo es cuantizado el error de predicción para la transmisión (diferencia de la señal). Tal que una aproximación puede ser adaptada por los cambios de la predicción, basada en las estadísticas de la imagen local o por variaciones ordinales de la cuantización, y

clasificación se basa en las formas de onda de codificación, las cuales pueden ser distribuidas en 4 categorías principales: Codificación por Modulación de Pulsos (PCM), Codificación Predictiva, Codificación por Transformada y Codificación Interpolativa/Extrapolativa. Además de estas 4 clases existen otros esquemas que no entran en ninguna de las anteriores categorías, pero están diseñados para otro tipo de imágenes. Por ejemplo la Codificación Run Length es usada para imágenes de facsímil y en blanco y negro. Cada una de las clases enunciadas puede subdividirse basándose en si los parámetros de codificación son fijos o adaptativos. En la práctica cualquiera de estas técnicas pueden ser utilizadas para la compresión de imágenes, ya sean estáticas o dinámicas, es decir, secuencias de imágenes; aunque algunos métodos son especialmente elaborados para secuencias de imágenes como es el de estimación de movimiento, el cual pertenece al esquema de Codificación Predictiva.

En la Codificación por Modulación de Pulsos (PCM), que también es conocida como conversión analógico/digital o A/D, en tiempo y amplitud discreta, la representación de los elementos de la imagen (pixeles) son hechos sin eliminar en demasía la redundancia estadística o perceptual de la señal. El tiempo discreto es proporcionado muestreando la señal generalmente a la frecuencia de Nyquist[1], mientras que la amplitud discreta es proporcionada usando un número suficiente de niveles de cuantización de modo que la degradación debida a la cuantización sea tolerable.

En la Codificación Predictiva, se intenta predecir el pixel a ser codificado. La predicción es hecha usando el valor de codificación del pixel anterior transmitido (o los pixeles ya codificados), y solo es cuantizado el error de predicción para la transmisión (diferencia de la señal). Tal que una aproximación puede ser adaptada por los cambios de la predicción, basada en las estadísticas de la imagen local o por variaciones ordinales de la cuantización, y

en un criterio visual; o bien, no transmitiendo el error de predicción, cuando se encuentre por debajo de un cierto umbral, esto es una condición de reaprovisionamiento. Otra posibilidad es demorar la decodificación de un pixel hasta el siguiente curso de la señal, tomando el código una ventaja en relación con el anterior, esto es llamado Codificación por Retraso.

En la Codificación por Transformada, en vez de la utilización de una codificación de imágenes con valores de intensidad discretos, en una porción de puntos muestreados; se presenta una alternativa diferente, primero transformamos linealmente bloques de pixel a bloques de datos, que son llamados coeficientes y después cuantizamos estos coeficientes seleccionados para la transmisión. Llegan a usarse varias transformaciones para la transmisión (tales como el sencillo método de Walsh-Hadamard o el más complejo Karhunen-Loeve[2]). La transformada coseno es una de las más populares, porque es el mejor método para las señales estadísticas. Dependiendo de las consideraciones costo/funcionamiento una, dos o tres dimensiones de los bloques (es decir dos dimensiones espaciales y una dimensión temporal) son usadas para la transformación. Para cualquiera de esas tres dimensiones, son posibles adaptaciones de la Codificación por Transformada, cambiando el orden de la transformación igualándola con la imagen estadística o cambiando el criterio para la selección de la cuantización del orden de los coeficientes.

La Codificación Interpolativa/Extrapolativa es una técnica que trabaja sobre un principio diferente, intenta enviar un subgrupo de pixeles de los que han sido recibidos y después extrapolando o interpolando intensidades de pixel, pueden obtenerse los pixeles que no se transmitieron. Esta técnica es usada con frecuencia para sistemas entre tramas junto con la Codificación Predictiva. La adaptación de este sistema consiste en variar el criterio para la selección de la estrategia para interpolación y extrapolación de las muestras.

Tabla 1.1 Técnicas de Codificación de Imágenes.

En la práctica un sistema de codificación, combina algunos de los esquemas de la tabla 1.1. Una de las mas interesantes combinaciones es llamada Codificación de Transformada Híbrida, en la que la transformación lineal de bloques de pixeles es seguida por una codificación predictiva de los coeficientes resultantes, basados en el bloque anterior adyacente transmitido (temporal o espacial). Otro método incluye Codificación Run-Length y este es de extensión bidimensional para señales de 2 niveles. En la Codificación Run-Length, se hace un recorrido consecutivo por los pixeles del mismo valor (0 o 1) asignándoles una palabra de código. Esto puede ser extendido a señales multinivel por codificación en diferentes planos de bits. La Codificación del Contorno es un esquema en el que se separa,

1) alto contraste o límite (o contornos) y

2) todo el resto.

Ya que una imagen de contornos es de dos niveles, la técnica que deberá ser aplicada en este caso es Run-Length, y el resto de la imagen que contiene solo frecuencias bajas y la información de la textura, deberá ser codificada por la técnica Predictiva o de Transformada.

1.1 Codificación por Modulación de Pulsos (PCM)

PCM es una forma de representación de la información visual en tiempo discreto y amplitud discreta. Aunque conceptualmente es sencilla, esta primera aplicación, no es usada para la televisión si no hasta 1951 por Goodall. Esto se debe quizá al hecho que la tecnología electrónica no era capaz de manejar las altas velocidades requeridas para las imágenes de televisión. A partir de entonces PCM fue usado para la digitalización de esquemas de video, para su almacenamiento y transmisión; además se utiliza antes de la aplicación de técnicas de codificación más sofisticadas.

La figura 1.1 muestra un esquema básico de PCM, que consiste de muestras en una dimensión, explorando las formas de onda (usualmente en la frecuencia de Nyquist[1]) y cuantizando cada muestra usando 2^K niveles. Aunque no es explícito en esta figura; un filtro apropiado es usado antes del muestreo, de modo que la estimación de las muestras son aproximaciones de Nyquist[1] y la distorsión es evitada. Cada nivel es representado para cada palabra binaria contenida en k bits. Usualmente las palabras en código binario son relacionadas monotónicamente a los niveles, de tal forma que pueden ser relacionados aritméticamente con facilidad. En el decodificador estas palabras binarias son convertidas a una secuencia de tiempo, de niveles de amplitud discretas con un filtro paso bajas. Básicamente el PCM muestra una simplicidad conceptual, muy poco común en relación a la mayoría de los otros esquemas de codificación, pero tiene la desventaja de ser ineficiente en el aspecto de que no explota completamente la redundancia presente en la señal de la imagen.

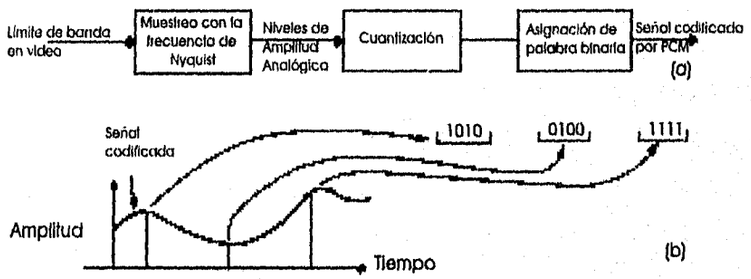


Figura 1.1. Codificación PCM. (a) Componentes de la codificación PCM. (b) 4 bits, representación de los niveles de amplitud 0 a 15.

Los criterios para realizar la selección de los niveles de cuantización de la codificación PCM pueden ser basados en gran medida en el criterio psicovisual, para imágenes de niveles de grises los efectos que se usan principalmente son la regla de Weber[3], que son estados del umbral de la visibilidad de una perturbación en relación con la luminancia DL, creciendo casi linealmente con

respecto a la luminancia de fondo. EL sistema PCM para video monocromático, en general, requiere de 128 a 256 niveles (7 a 8 bits) por pixel para una buena calidad de la imagen bajo las condiciones visuales. Para televisión monocromática con una frecuencia de muestreo de 8 MHz, se generan de 56 a 64 Mbits/segundo. Para sistemas usando tramas fijas, el ruido de cuantización no tiene variación en el tiempo como en la televisión, por lo tanto es congelada en la pantalla. Esto reduce la visibilidad y por lo consiguiente son suficientes 6 o 7 bits por pixel para la cuantización.

El ruido de cuantización que puede ser visible debido a lo amplio de la cuantización, puede ser reducido con varias técnicas. Una de ellas es aplicarle filtros antes y después del cuantizador. Simulaciones computarizadas muestran que usando estos filtros, pueden obtenerse imágenes esencialmente libres de contornos artificiales con mas o menos 5 a 6 bits por pixel. Sin embargo la desventaja de los filtros es que cada filtro reduce la resolución de la imagen reproducida.

Como el ojo humano es más sensible al ruido y a la distorsión que se encuentra en un solo lugar, que al ruido esparcido por toda la imagen. Para lograr una alta calidad de la imagen original debe ser muestreada a una frecuencia lo suficientemente alta, sin embargo, si reducimos los niveles de cuantización, el ruido es más perceptible en los contornos. Este ruido de cuantización puede ser reducido realizando la adición de algunas frecuencias altas (llamado Dither) a la señal original antes de la cuantización. Este ruido agregado artificialmente, causa que la codificación de la señal oscile entre los niveles de cuantización, logrando así que se incremente la frecuencia contenida en el ruido de cuantización. Los falsos contornos que son altamente visibles, son eliminados y el ruido de altas frecuencias es menos visible. Dithering es entonces una técnica que consiste en sumar un ruido aleatorio a la señal de la imagen

antes de la cuantización y en el proceso de recepción sustraer el mismo ruido al realizar la cuantización de la imagen.

1.2 Codificación Predictiva

El sistema PCM transmite las amplitudes cuantizadas de cada pixel; pero como ya se había comentado, no explota la fuerte correlación que existe entre los pixeles adyacentes espaciales y los pixeles adyacentes temporales. La ventaja de la codificación predictiva es que explota esta correlación. En un sistema básico predictivo una aproximación de la predicción de la muestra a ser codificada es realizada (ver figura 1.2). El error resultante (o diferencia de señal) de la resta con la predicción del valor del pixel actual, es cuantizado dentro de un bloque L de niveles de amplitud discretos. Estos niveles son representados en una palabra binaria (largo de palabra) de tamaño fijo o variable y enviada al canal del codificador para su transmisión.

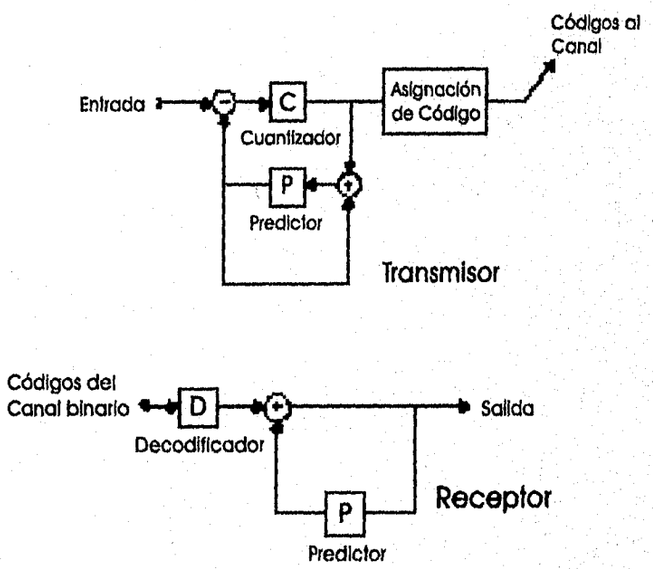


Figura 1.2. Diagrama de un Codificador y Decodificador Predictivo

La codificación predictiva tiene 3 componentes básicos:

1) Predictor, 2) Cuantizador y 3) Asignación de Código.

Dependiendo del número de niveles de cuantización a menudo se hace una distinción entre la Modulación Delta (DM), con L=2 y la Modulación por Codificación de Pulsos Diferencial (DPCM) con L mayor que 2. Como en la Modulación Delta solo se usan dos niveles para obtener una calidad adecuada de la imagen, las muestras son estimadas en varios tiempos con la frecuencia de Nyquist[1]. Aunque DM también se usa en otras formas de onda; no es un buen método para ser usado en codificación de imágenes, debido quizá a la alta estimación de muestras requeridas.

1.2.1 Modulación por Codificación de Pulsos Diferencial (DPCM)

En la codificación DPCM, la señal analógica es muestreada primero por la frecuencia de Nyquist[1]. en una forma simple, DPCM usa los valores de codificación del pixel horizontal previo a la predicción. Esto es equivalente a la cuantización en una aproximación del gradiente horizontal de la señal. Otros predictores más sofisticados hacen un mejor uso de la correlación usando más elementos en el campo presente (incluyendo la línea presente y la anterior) así como campos o tramas de información. Esto es llamado Predicción Intra-campo (dominio espacial), Inter-trama (dominio temporal), respectivamente. Los sistemas de codificación basados en tales predictores son llamados tradicionalmente Codificadores Intra-campo (dominio espacial) o Inter-trama(dominio temporal). Los Codificadores Inter-trama requieren un campo o trama de memoria y son generalmente más complejos que los codificadores Intra-campo. Sin embargo, como la memoria en cualquiera de los dos codificadores es baja, la distinción entre un Codificador Intra o Inter trama llega a ser poco importante.

1.2.2 Predictores

Los predictores para DPCM pueden ser clasificados en lineales y no lineales dependiendo si la predicción se realiza sobre una función lineal o no lineal de los valores transmitidos previamente. Una división más puede ser hecha dependiendo de la localización del pixel usado previamente: Un Predictor Unidimensional usa un pixel previo en la misma línea del pixel a ser pronosticado; los Predictores Bidimensionales usan pixeles de las línea(s) previas a ser predichas; mientras que los Predictores Inter-trama usan pixeles de los campos o tramas previas.

Los predictores pueden ser fijos o adaptativos. Los Predictores Adaptativos, cambian sus características en función de los datos, mientras que los Predictores Fijos, mantienen las mismas características independientemente de los datos.

1.2.2.1 Predictores Lineales

Los Predictores Lineales para Imágenes monocromáticas son estudiados usando la teoría general de la predicción lineal.

Si $\{b_1,...b_N\}$ es un bloque de pixeles de media 0 para cada $b_1,...b_{N-1}$ ya transmitidos, (y por lo tanto puede ser usado el predictor b_N), un predictor lineal para b_N puede ser escrito como:

$$\hat{b}_N = \sum_{i=1}^{N-1} \alpha_i \, b_{N-i} \qquad (1.1)$$

Esto es mostrado en el diagrama de la figura 1.3. Los coeficientes $\{\alpha_i\}$ pueden ser obtenidos por la minimización del error cuadrático medio de

predicción (MSPE), $E(b_N-\hat{b}_N)^2$ [4], usando un coeficiente óptimo, el MSPE es dado por:

$$\text{(MSPE) óptimo} = \sigma^2 - \sum_{i=1}^{N-1} \alpha_i d_i \qquad (1.2)$$

donde $d_i=E(b_N \cdot b_{N-i})$, y además se asume que los pixeles tienen una distribución promedio y varianza σ^2 . De este modo MSPE, en la entrada del cuantizador DPCM reduce $\sum_{i=1}^{N-1} \alpha_i d_i$ de σ^2, el cuadro promedio de entrada a un cuantizador PCM. Esto es encontrado en más casos, donde la suma de los coeficientes $\{\alpha_i\}$ son aproximados a uno y por lo tanto la ecuación 1.1, es típicamente usada con el valor del pixel original, p.e., sin la primera sustracción de los valores promedio de los pixeles.

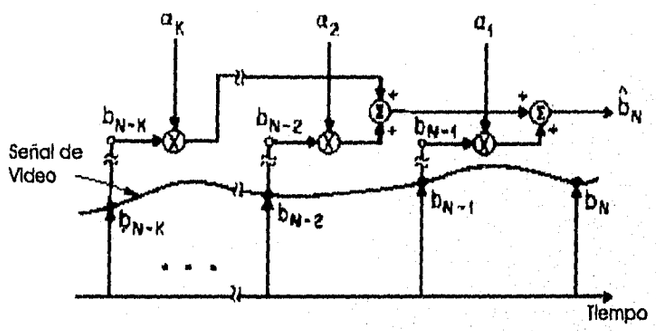


Figura 1.3 Predictor Lineal usado para la Codificación Predictiva.

En el análisis se asumen estadísticas fijas e idénticas de la secuencia de pixeles $\{b_N\}$. Aunque la teoría de predicción lineal puede tratar señales teniendo una gran variedad de estadísticas fijas, al aplicarla para codificación de imágenes no se obtienen muy buenos resultados. Las principales razones son porque, primero, no satisface suficientemente los modelos estadísticos que

describen con precisión la señal de la imagen y, segundo es que durante la minimización, el error cuadrático medio de predicción representa un factor importante, esto no es equivalente a la minimización de estimación de bit o la optimización de la calidad de la imagen codificada. Por otra parte analizando las fallas de los efectos de cuantización en un codificador DPCM; en un codificador real DPCM, la predicción de pixeles b_N puede ser realizada solamente usando representaciones codificadas previas de las muestras pasadas $\tilde{b}_{N-1}$, $\tilde{b}_{N-2}$,..., y no usando los valores originales de los pixeles. Esto es necesario para permitir la recepción para un posterior cálculo de la predicción. Aunque para un codificador que produce una alta calidad de la imagen, los efectos de cuantización son usualmente pequeños y con frecuencia existen errores depreciables.

1.2.2.2 Predictores intra-campo

En la práctica con frecuencia son usados los Predictores Bidimensionales. Aunque la mejora en la entropía de la predicción del error usando el Predictor Bidimensional no sea significativa, sin embargo puede existir una considerable disminución en la predicción del error pico. Por otra parte evaluaciones subjetivas indican que al usar el Predictor Bidimensional, en imágenes cuantizadas que tienen muchos bordes con diferentes orientaciones, existe una mejora significativa.

También para la propiedad de elección de los coeficientes, es posible mejorar las predicciones, además de un rápido descenso de los efectos de transmisión de error en la reconstrucción de la imagen. En general desde que las correlaciones son usualmente altas y el contenido de la señal de la imagen no tiene cambios drásticos dentro de algunas muestras, la suma de los coeficientes

del predictor óptimo MSPE, se cierra a uno. Sin embargo aseguran que los efectos de algunas perturbaciones (cuantizador) en el lazo DPCM desaparezcan en función del tiempo (es decir que el lazo sea estable), en el peor de los casos, $\sum_{i=1}^{N} |\alpha_i|$ es una posibilidad remota. Por lo tanto esto indica que es mejor usar un número grande de elementos de la imagen, con cada coeficiente positivo en el proceso de predicción.

La posición relativa de los pixeles en la predicción intra-campo, se muestra en la figura 1.4.

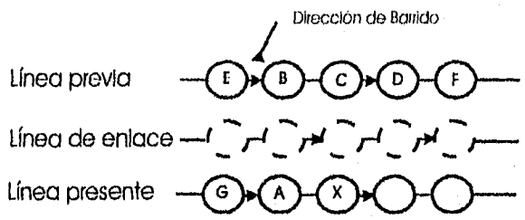


Figura 1.4 Muestra la posición y nombres de los elementos usados para la predicción de un pixel X. Las líneas previa y presente son del mismo campo.

En pruebas realizadas para imágenes con una predicción del error de luminancia usando Predictores Unidimensionales y Bidimensionales cuantizados a 4 bits en un lazo DPCM; el Predictor Bidimensional reduce el error en los bordes verticales, pero el error sobre los bordes horizontales se incrementa, en relación con el resultado arrojado por el Predictor Unidimensional.

1.2.2.3 Predictores Inter-trama

Los Predictores Inter-trama, en general usan combinaciones de pixeles del campo actual y combinaciones de pixeles del campo anterior. Para escenas

con bajo detalle y pequeño movimiento, la predicción por diferencia de tramas, parece dar los mejores resultados. En escenas con mucho detalle y gran cantidad de movimiento, la predicción por diferencia intra-campo es mejor que la predicción por diferencia de tramas. A medida que el movimiento de la escena aumenta, los Predictores Intra-campo son mejores en cuanto a desempeño.

Esto es debido a dos razones:

1) Para mucho movimiento, hay menos correlaciones entre el pixel presente y los pixeles del campo o trama anterior.

2) Debido al tiempo de integración de la señal en la cámara de video, la correlación espacial de la señales de televisión en la dirección de movimiento, es incrementada.

En la figura 1.5 se muestra la configuración de la posición de los pixeles en la predicción Inter-trama. En función de diferentes predictores y movimiento de los objetos en la escena.

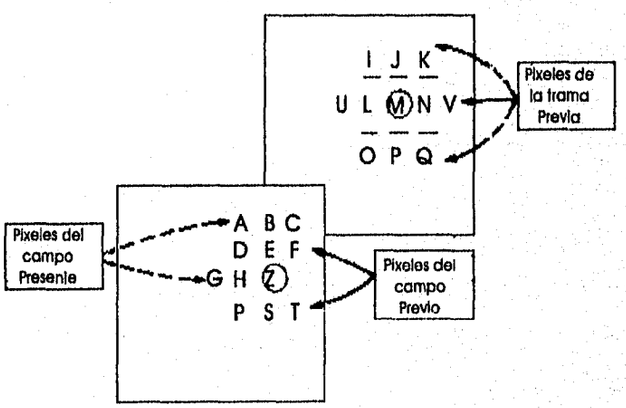


Figura 1.5 Configuración de los Predictores Inter-trama.

La predicción adaptativa trama a trama es basada en una filosofía similar a la predicción intra-campo. Un ejemplo de esto es la extensión de la predicción de Graham[5] tridimensional que puede seleccionar los predictores intra campo o trama dependiendo de la información circundante. De este modo en la figura 1.5 el pixel Z puede ser predicho por el pixel H (elemento previo), pixel B (línea previa en el mismo campo) o pixel M (trama anterior), dependiendo de cual sea menor de las siguientes diferencias: La de |H-G| (diferencia de elemento), |H-A| (diferencia en línea), o |H-L| (diferencia entre trama). Cada cambio implica que el valor de código de muestra, G, H y B, este disponible. Sin embargo, en algunos codificadores inter-trama, durante la codificación de sobrecarga, estos pixeles tal vez no se encuentran disponibles debido a su submuestreo. En cada caso, el predictor debe tener modificaciones convenientes.

Con respecto a lo anterior un funcionamiento mejorado en el desempeño de los predictores adaptativos trama a trama, tomando en cuenta la velocidad y dirección del movimiento de objetos, son tratados en el siguiente punto.

1.2.3 Estimación de Movimiento

Si una escena de televisión cuenta con objetos en movimiento y si se dispone de una estimación de la traslación, entonces una predicción más eficiente puede ser realizada usando elementos de la trama anterior, que son desplazados espacialmente. Cada predicción es llamada compensación de movimiento. El proceso de compensación de movimiento es recursivo y sobre el barrido en televisión (de izquierda a derecha y de arriba hacia abajo). En la figura 1.6 se muestra un diagrama de bloque de compensación, el cual es basado en el esquema DPCM, con la diferencia del bloque de estimación.

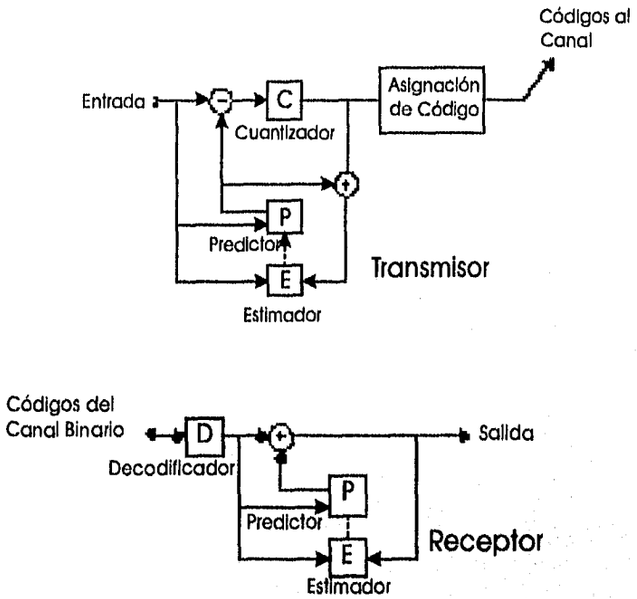


Figura 1.6 Diagrama de un Codificador y Decodificador para Compensación de Movimiento.

En escenas reales el movimiento puede ser una compleja combinación de traslación y rotación. Cada movimiento es difícil de estimar y requiere una larga cantidad de procesos. Sin embargo el movimiento de traslación es fácil de estimar y puede llegar a usarse satisfactoriamente para la codificación y compensación del movimiento. El tener éxito depende de la cantidad de movimiento de traslación en la escena y la habilidad de un algoritmo de estimación de traslación, para tener la exactitud necesaria para una buena predicción. El punto crucial por consiguiente es el algoritmo usado para la estimación de movimiento.

La mayoría de los algoritmos para estimación de movimiento en codificación inter-trama toman en cuenta las siguientes suposiciones:

i. Los objetos se mueven en traslación en un plano que es paralelo al plano de la cámara, es decir no considerando los efectos de aumento y acercamiento en una cámara, y los objetos en rotación.

ii. La iluminación es uniforme espacial y temporalmente.

iii. La obstrucción de un objeto a otro y parte del fondo es constante.

De acuerdo con la suposición anterior la intensidad monocromática $I(z,t)$ y $I(z, t-\tau)$ de dos tramas consecutivas son relacionados por:

$$I(z,t)=I(z-D,t-\tau) \qquad (1.3)$$

donde t es el tiempo entre dos tramas y D es el vector de traslación del objeto durante el intervalo de tiempo $[t-\tau, t]$ y z es el vector bidimensional $[x,y]'$ de la posición espacial. De este modo, en una escena real una buena predicción de $I(z,t)$ es $I'=(z-D, t-\tau)$. El problema a solucionar es la estimación D para la intensidad de la trama presente y pasada.

Dos métodos son usados ampliamente:

1) Métodos de Acoplamiento por Bloques.

2) Métodos Pel-Recursivo.

Aunque también es posible una combinación de estos dos métodos en una variación de trayectorias. Dichos métodos son profundizados en el siguiente capítulo.

[1] Proakis, John G., **Digital Communications**, Ed. Mc. Graw, Hill, segunda edición, United States of America, 1989, pp. 53.

[2] Netravali, Arun N. y Haskell, Barry G., **Digital pictures: Representation and Compression**, Ed. Plenum Press, primera edición, New York, N.Y., 1988, pp. 170.

[3] Jain, Anil K., **Fundamentals of Digital Image Processing**, Ed. Prentice-Hall, primera edición, Englewood Cliffs, New Jersey, 1989, pp. 51.

[4] Netravali, Arun N. y Haskell, Barry G., **Digital pictures: Representation and Compression**, Ed. Plenum Press, primera edición, New York, N.Y., 1988, pp. 152.

[5] Netravali, Arun N. y Haskell, Barry G., **Digital pictures: Representation and Compression**, Ed. Plenum Press, primera edición, New York, N.Y., 1988, pp. 325.

Capítulo 2

Estimación de Movimiento

Como se expresó anteriormente la compensación de movimiento es un poderoso medio para reducir la cantidad de información que resulta redundante entre tramas de una secuencia de imágenes. La aplicación de esta técnica se divide en dos partes:

1) La técnica de estimación de movimiento, la cual examina el movimiento de los objetos en una secuencia de imágenes, para así poder predecir los vectores que representan la estimación de desplazamiento existente en la imagen.

2) La compensación de movimiento, donde los vectores obtenidos en la estimación deben ser utilizados de manera óptima, para la transmisión y recepción de la imagen.

Se han desarrollado numerosas técnicas para realizar la predicción de los vectores de movimiento. De las cuales nos enfocaremos al estudio de la estimación de movimiento predictiva, el mismo que tiene dos técnicas principales:

1) Acoplamiento por Bloques.

2) Pel-Recursivo.

2.1 Método de Acoplamiento por Bloques

Con el método de acoplamiento por bloques, se asume que el desplazamiento de los objetos es el mismo dentro de un pequeño bloque B, de

pixeles bidimensionales. Esta suposición representa dificultades en una escena con múltiples objetos en movimiento o en escenas en que diversas partes del mismo objeto se mueven con desplazamientos diferentes. Si el tamaño del bloque decrece entonces esta suposición llega ha tomar mayor validez; sin embargo la sobrecarga computacional y la transmisión de los vectores de desplazamiento D, aumentan.

El desplazamiento D puede ser estimado por correlación o técnicas de acoplamiento; de esta forma, D puede ser seleccionada de cada minimización del error de predicción, tal que:

$$\mathrm{PE}(D)=\sum N(b(z,t)-b(z-D,t-\tau)) \qquad (2.1)$$

donde $N(\bullet)$ es una distancia métrica, o una función cuadrática. Para encontrar el mejor D, todos los pixeles $z \in B$ son predichos, codificados y transmitidos usando el valor de D.

Por ejemplo considérese un bloque de MxN pixeles centrados alrededor del pixel z_0 en la trama actual con tiempo t, asumiendo un desplazamiento máximo horizontal y vertical de d_{max} pixeles, la región en la trama anterior donde la minimización de la ecuación 2.1, esta dada por* :

$$\begin{bmatrix} \frac{1}{2}(M-1)+d_{\max} \\ \frac{1}{2}(N-1)+d_{\max} \end{bmatrix} \qquad (2.2)$$

el número de pixeles en esta región esta dado por:

* En la ecuación se asume que M, N son impares.

$$(M+2d_{max})(N+2d_{max}) \tag{2.3}$$

De este modo si el bloque es de 9X9 y tiene un desplazamiento máximo $d_{max}=10$, la región de búsqueda a utilizar en la trama anterior puede ser un área contando 29X29 pixeles. Un método de búsqueda puede ser evaluado por medio de la ecuación 2.1. Para cada cambio el pixel es seleccionado en la dirección horizontal y vertical. Esto puede requerir $(2d_{max}+1)^2$ evaluaciones de la ecuación 2.1. Por otra parte a menos que los cambios de distancia sean funcionales son incluidos, la exactitud con que D puede ser obtenida, es limitada a un pixel. Un tipo de simplificación, es usando un simple criterio para la ecuación 2.1.

Un ejemplo es:

$$\mathrm{PE}(z_0,i,j)=\frac{1}{MN}\sum_{|m|\le\frac{M}{2}}\sum_{|n|\le\frac{N}{2}}[b(z_{mn},t)-b(z_{m+i,n+j},t-\tau)]^2 \tag{2.4}$$

o

$$\mathrm{PE}(z_0,i,j)=\frac{1}{MN}\sum_{|m|\le\frac{M}{2}}\sum_{|n|\le\frac{N}{2}}|b(z_{mn},t)-b(z_{m+i,n+j},t-\tau)| \tag{2.5}$$

donde

$$-d_{max} \le i,j \le +d_{max}$$

$$z_{mn}=z_0+[m,n]' \tag{2.6}$$

La segunda definición de $\mathrm{PE}(\bullet)$ en la ecuación 2.5 tiene la ventaja que no requiere de multiplicaciones. Además se puede observar experimentalmente que la definición de precisión de PE no tiene un efecto significativo sobre la cantidad de cambios o la precisión de la estimación de D y, por lo tanto la simplicidad y el criterio de la ecuación 2.5 es generalmente preferible.

Además de la simplificación de criterios de acoplamiento, se han desarrollado algunos métodos para simplificar el procedimiento de búsqueda, de los cuales, mostraremos 3 métodos principales:

1) Búsqueda Logarítmica 2D

2) Búsqueda por los 3 pasos

3) Búsqueda por Dirección Conjugada

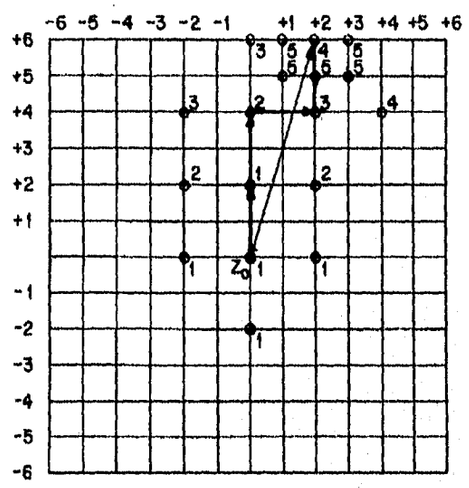
Estos métodos se basan en que los criterios matemáticos antes mencionados, decrecen monotónicamente hacia la posición de la mejor igualdad, y su búsqueda termina al encontrarla. Para llegar a su meta, es necesario realizar varios cambios, y en cada uno se requiere del cálculo de estos criterios para determinar la dirección del vector.

2.1.1 Búsqueda Logarítmica 2D

En la búsqueda Logarítmica 2D, los algoritmos siguen la dirección de la distorsión mínima. A cada paso, 5 movimientos son realizados y mostrados en la figura 2.1. La distancia entre los puntos de búsqueda decrece si el mínimo se localiza en el centro o en el límite del área de búsqueda.

Por citar un ejemplo, (ver figura 2.1), donde 5 pasos se requieren para llegar al vector de desplazamiento en el punto $(i,j)=(2,6)$. Los movimientos en el área de búsqueda de la trama anterior son mostrados con respecto al pixel (z_0) de la trama actual. Aquí las aproximaciones de los desplazamientos de los vectores son (0,2)', (0,4)', (2,4)', (2,5)' y (2,6)' que son encontrados en 5 pasos,

denotando los puntos que se usan en cada paso con los números 1, 2, 3, 4, y 5 respectivamente. d_{max} es 6 pixeles.



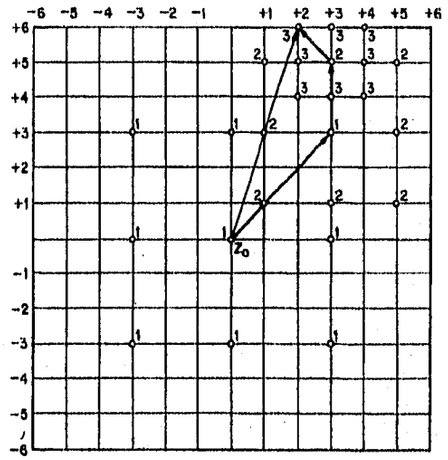
o^n denota el punto de búsqueda en el paso n

Figura 2.1 Procedimiento de búsqueda Logarítmica 2D.

2.1.2 Búsqueda por los 3 pasos

En el procedimiento de búsqueda por los 3 pasos, 8 pixeles alrededor del pixel central z_0 son examinados en el primer paso. En el segundo paso, de nuevo se toman 8 pixeles, alrededor del mínimo PE encontrado en el primer paso, pero en este, el espacio es más reducido que en el primero. En la figura 2.2 se muestran los puntos z_0+(3,3) y z_0+(3,5) que son la 1ª y la 2ª aproximación respectivamente. Este proceso continua para otro paso, el resultado obtenido en el desplazamiento final es el pixel z_0+(2,6).

Las aproximaciones de los vectores de desplazamiento (3,3)', (3,5)' y (2,6) son encontradas en 3 pasos, denotando los puntos que se usan en cada paso con los números 1, 2 y 3 respectivamente.



o^n denota el punto de búsqueda en el paso n

Figura 2.2. Procedimiento de la búsqueda por los 3 pasos.

2.1.3 Búsqueda en la Dirección Conjugada

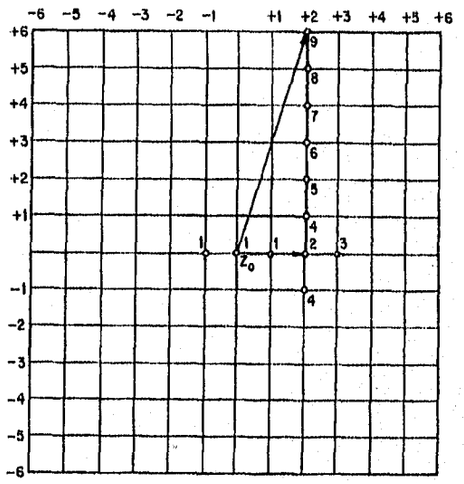
Mostraremos la búsqueda en la Dirección Conjugada en su forma simple. Este algoritmo busca la dirección del mínimo $\mathrm{PE}(z_0,i,j)$. Este proceso cuenta con dos pasos:

1) Como primer paso, la búsqueda es realizada a lo largo de la dirección horizontal. Guiándonos en la figura 2.3, $\mathrm{PE}\ (z_0,-1,0)$, $\mathrm{PE}(z_0,0,0)$ y $\mathrm{PE}(z_0,1,0,)$ que son z_0 y sus puntos vecinos de izquierda y

derecha, son calculados. Si $\mathrm{PE}(z_0,1,0)$ es el menor, entonces $\mathrm{PE}(z_0,2,0)$ es también comparado y calculado, por ser un punto vecino del menor $\mathrm{PE}(z_0,1,0)$. El proceso en esta dirección, determina el mínimo en la dirección horizontal, cuando este es localizado entre dos valores altos para los pixeles vecinos.

2) El segundo paso del procedimiento es idéntico en la realización de todos los cálculos pero ahora en la dirección vertical.

En la figura 2.3, se muestra un ejemplo de la técnica en la dirección conjugada, en la cual, el resultado de dos pasos es el pixel z_0+(2,0)' en la búsqueda horizontal y z_0+(2,6)' en la búsqueda vertical.



o^n denota el punto de búsqueda en el paso n

Figura 2.3. Procedimiento de la búsqueda en la Dirección Conjugada en su forma simple.

Cabe mencionar que la complejidad de los procedimientos computacionales, están relacionados con el número de movimientos o cambios que PE realiza en el algoritmo. La tabla 2.1, muestra una comparación del número de búsquedas y pasos secuenciales, requeridos para cada uno de los procedimientos. Esto puede indicar que la exactitud de los desplazamientos en la estimación de los métodos de bloque son limitados generalmente a 0.5 pixeles.

Procedimiento de búsqueda	No. requerido de puntos de búsqueda		No. requerido de pasos secuenciales	
	a	b	a	b
Logarítmica 2D	18	21	5	7
3 pasos	25	25	3	3
Dirección Conjugada	12	15	9	12

a) Para el desplazamiento espacial de los vectores $(2,6)'$.
b) Para un caso extremo.

Tabla 2.1 Muestra el número de puntos de búsqueda requeridos y pasos secuenciales para los procedimientos de búsqueda.

Aunque los métodos de acoplamiento por bloques tienen una mayor tasa de compresión, por utilizar bloques de pixeles para obtener el vector de desplazamiento, también tiene algunos problemas mencionados anteriormente, como es el generar ruido, en áreas de mucho movimiento, en el proceso de reconstrucción.

2.2 Método Pel-Recursivo

Limb y Murphy [1] propusieron un algoritmo que mide la velocidad del movimiento de los objetos, con un fondo fijo. Así mismo Cafforio y Rocca [2] dieron fundamentos teóricos para la estimación de pequeños movimientos de

traslación e introducen un algoritmo de segmentación que reporta pixeles con diferentes movimientos.

Basado en lo anterior Netravali y Robbins desarrollaron un algoritmo de estimación que usa recursivamente los cambios relativos de luminancia y gradientes para encontrar un vector de movimiento para cada pixel. En este algoritmo no se necesita transmitir la información del vector de movimiento.

Walker y Rao, retomando la técnica de Netravali y Robbins, observaron que en términos de velocidad de convergencia era pobre. Pues muchas iteraciones eran necesarias para producir un vector de movimiento satisfactorio y considerando que el término de actualización varia dependiendo de la diferencia de trama desplazada y el gradiente, proponen una modificación a la técnica de Netravali y Robbins, variando el factor de convergencia.

Fundamentos:

La idea básica de las técnicas Pel-Recursivo es la siguiente:

Asumiendo la siguiente notación, $I(z,t-\tau)$ e $I(z, t)$ son valores de intensidad de dos tramas sucesivas en función de una localización espacial z (vector bidimencional) y el tiempo t. El tiempo entre dos tramas es $t-\tau$.

Si un objeto se mueve en traslación, en el área de movimiento (no considerando los bordes) :

$$I(z,t)=I(z-D,t-\tau) \qquad (2.7)$$

Donde D es el vector de traslación del objeto durante el intervalo de tiempo $[t,t-\tau]$.

Por lo anterior una diferencia de trama FD es definida como:

$$FD=I(z,t)-I(z,t-\tau) \tag{2.8}$$

Si un objeto esta en movimiento, entonces esta estimación de desplazamiento entre trama puede ser representado por $\hat{D}^i$. Para que haya mejor precisión de la estimación de desplazamiento, esta puede ser obtenida linealizando la función de intensidad para $(i-1)$ campos. Este proceso resulta en la siguiente recursión :

$$\hat{D}^i=\hat{D}^{i-1}+U^i \tag{2.9}$$

Donde $\hat{D}^{i-1}$ es un una estimación inicial de $\hat{D}^i$ y U^i es el la actualización de $\hat{D}^{i-1}$ para hacer ésta más exacta.

Tomando en cuenta lo anterior, la diferencia de trama desplazada (DFD) esta definida como:

$$DFD(z,\hat{D}^i)=I(z,t)-I(z-\hat{D}^{i-1},t-\tau) \tag{2.10}$$

DFD es definido en términos de dos cantidades, la localización espacial de z y el desplazamiento de $\hat{D}^i$. Si no existe error en $\hat{D}^i$, $D^i=D$, la DFD puede ser cero, puesto que idealmente el valor del pixel es idéntico. Un desplazamiento $\hat{D}^i$ puede ser buscado con DFD para un valor cercano o igual a cero. Ya que DFD es una función de valores de intensidad.

Existen varias técnicas para encontrar la estimación de $\hat{D}^i$, pero trataremos solamente las desarrolladas por :

1) Netravali y Robbins.

2) Walker y Rao.

2.2.1 Técnica de estimación de movimiento por Netravali y Robbins

Este método implica calcular el desplazamiento del movimiento, donde los valores de estimación de movimiento son renovados en cada pixel, a medida que se van dando las iteraciones, el algoritmo continua de pixel en pixel, la estimación de movimiento converge hacia el desplazamiento verdadero. Por lo consiguiente al adquirir un desplazamiento verdadero de un objeto, este vector puede ser usado para predecir los valores sucesivos de los pixeles. Para determinar si se a llegado al desplazamiento verdadero la estimación de movimiento se apoya en la predicción de error. Así si un pixel localizado en z_a es predicho con un vector de desplazamiento $\hat{D}^{i-1}$ e intensidad $I(z_a-\hat{D}^{i-1}, t-\tau)$, resultando en la predicción de error $DFD\ (z_a, \hat{D}^{i-1})$, la estimación debe tratar de producir una nueva estimación $\hat{D}^i$, tal que $|DFD(z_a, \hat{D}^i)| \leq |DFD(z_a, \hat{D}^{i-1})|$. A este término se le aplica una minimización recursiva $[DFD(z, \hat{D}^i)]^2$ a cada elemento del área en movimiento, usando un gradiente, por ejemplo:

$$\hat{D}^i = \hat{D}^{i-1} - (\varepsilon/2)\nabla_D[DFD(z_a - \hat{D}^{i-1})]^2$$

$$= D^{i-1} - \varepsilon DFD(z_a, \hat{D}^{i-1})\nabla_D DFD(z_a, \hat{D}^{i-1})$$

donde ∇_D es el gradiente con respecto al desplazamiento D y ε es una constante escalar. El gradiente ∇_D puede ser evaluado usando la definición DFD y

$$\nabla_D(DFD(z_a, \hat{D}^{i-1})) = +\nabla I(z_a - \hat{D}^{i-1}, t-\tau) \tag{2.11}$$

Esto nos da

$$\hat{D}^i = \hat{D}^{i-1} - \varepsilon DFD(z_a, \hat{D}^{i-1})\nabla I(z_a - \hat{D}^{i-1}, t-\tau) \tag{2.12}$$

donde DFD y ∇I pueden ser calculados por interpolación para $\hat{D}^{i-1}$ no-entera. Una reducción significativa en los cálculos de ∇I es lograda por la cuantización de $\hat{D}^{i-1}$ a un valor entero. Así, si $[\hat{D}^{i-1}]$ representa un valor redondeado trunco de cada una de las componentes de $\hat{D}^{i-1}$, entonces el estimador de ecuación 2.12 puede ser simplificado por

$$\hat{D}^i = \hat{D}^{i-1} - \varepsilon DFD(z_a, \hat{D}^{i-1})\nabla I(z_a - [\hat{D}^{i-1}], t-\tau) \tag{2.13}$$

Es importante observar que a muchas iteraciones se añadirá a nuestra estimación pasada un vector paralelo a la dirección del gradiente espacial de la intensidad de la imagen, cuya magnitud es proporcional al movimiento compensado de predicción del error. Las cantidades involucradas en la técnica anterior son mostradas en la figura 2.4.

Una estimación inicial del desplazamiento del pixel $z_a - \hat{D}^{i-1}$, es actualizada usando la ecuación 2.13, produciendo $\hat{D}^i$.

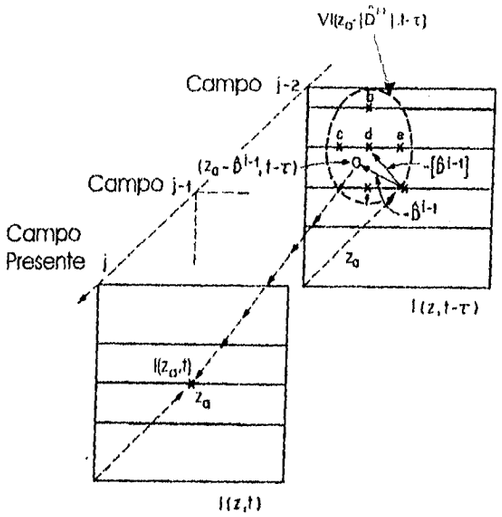


Figura 2.4 Estimación de movimiento recursiva.

Usando la estimación inicial $\hat{D}^{i-1}$ y z_a, la muestra en el pixel anterior en las posiciones vecinas de la posición espacial z_a-$\hat{D}^{i-1}$ son localizadas (por ejemplo b, c, d, e, g, y f), Las muestras de estas vecindades son usadas después para la actualización de los términos en conjunción con $I(z_a,t)$. Así en la figura 2.4, los componentes de la intensidad del gradiente pueden ser aproximados por:

$\mathrm{EDIF}=(I_e-I_c)/2$ (en la posición horizontal)

$\mathrm{LDIF}=(I_b-I_f)/2$ (en la posición vertical)

De la misma forma, $I(z_a-\hat{D}^{i-1},t-\tau)$ puede ser aproximada por una interpolación lineal bidimensional usando la intensidad de las vecindades.

Una modificación que realizó Netravali y Robbins a su algoritmo original fue la siguiente simplificación del término de actualización:

$$\hat{D}^{i} = \hat{D}^{i-1} - \varepsilon sign\{DFD(z_a, \hat{D}^{i-1})\} sign\{\nabla I(z_a - \hat{D}^{i-1}, t-\tau)\} \qquad (2.12)$$

donde *"sign"* es el signo del vector de estas componentes. Tal que $sign(v)$=0, 1, -1, este resultado en vectores rectificados que pueden tomar solo ángulos de múltiplos de 45°, lo cual evita las multiplicaciones necesarias para los cálculos del término de actualización. De este modo el término de actualización de cada componente de desplazamiento para un elemento al siguiente de la imagen, consiste de solo 3 posibilidades; 0 o $\pm\varepsilon$.

2.2.2 Técnica de estimación de movimiento por Walker y Rao

Otra de las técnicas analizadas es la introducida por Walker y Rao, en la cual tomando como base la técnica desarrollada por Netravali y Robbins, observa que el algoritmo original tiene como desventaja la lentitud en la convergencia, es decir, muchas iteraciones son requeridas para obtener una estimación de movimiento certera. Por tal razón propone una modificación en la cual considerando la función de intensidad y los bordes de un objeto, se tiene que cuando el error es grande y el gradiente pequeño, el vector de corrección es grande y viceversa, por lo cual en este algoritmo el factor de convergencia varía acorde con el gradiente obtenido, lo cual nos arroja como resultado una mejora en la velocidad de convergencia.

Dándonos como resultado la siguiente ecuación:

$$\hat{D}^{i} = \hat{D}^{i-1} - \varepsilon' DFD(z_a, \hat{D}^{i-1}) I(z_a - \hat{D}^{i-1}, t-\tau) \qquad (2.13)$$

donde

$$\varepsilon^{i}=\frac{1}{2}\;\frac{1}{\left(\left|\nabla I\left(z_a-\hat{D}^{i-1},t-\tau\right)\right|\right)^2}$$

y

$$\{|\nabla I(z_a-\hat{D}^{i-1},t-\tau)|\}^2=\{\nabla_x I(z_a-\hat{D}^{i-1},t-\tau)\}^2+\{\nabla_y I(z_a-\hat{D}^{i-1},t-\tau)\}^2$$

---

[1] Limb, John O. y Murphy, J. A., **Measuring the Speed of Moving Objects from Television Signals**, IEEE Transactions on Communications, Vol. COM-23, No. 4, abril 1975, pp. 474-478.

[2] Cafforio, Ciro y Rocca, Fabio, **Methods for Measuring Small Displacements of Television Images**, IEEE Transactions on Information Theory, Vol. IT-22, No. 5, septiembre 1976, pp. 573-579.

Capítulo 3

Métodos de Pel-Recursivo de estimación de movimiento

En este capítulo se describirán los algoritmos de estimación de movimiento Pel-Recursivo, mismos que, examinando el desplazamiento de los objetos en una secuencia de imágenes, pueden realizar la predicción de los vectores que representan dicha estimación. La estimación se realiza recursivamente (pixel por pixel) y en el sentido del barrido de la televisión, intentando minimizar el error de predicción, para hacer una convergencia hacia el desplazamiento correcto.

Dichos algoritmos están basados en los cambios que se realizan entre tramas de televisión, los resultados son obtenidos en base a la traslación de un objeto en movimiento, por lo que el algoritmo itera con un gradiente descendente, de tal manera que las estimaciones que se realizan consecutivamente converjan al desplazamiento de traslación deseado.

El primer algoritmo que se describirá, es el desarrollado por Netravali y Robbins, que implica calcular un desplazamiento del movimiento, realizando iteraciones pixel a pixel para a continuación separar los datos calculados en pixeles predictivos y pixeles no predictivos.

Del algoritmo original se deriva una simplificación, que también es mostrada en esta sección.

Otro de los algoritmos que serán mostrados es el desarrollado por Walker y Rao, en donde al considerar la función de intensidad y los bordes de un objeto , observaron que cuando el error de predicción es grande y el gradiente pequeño, el vector de corrección es grande y viceversa, para lo cual el factor de convergencia en este algoritmo, varia acorde con el tamaño del gradiente obtenido, lo que nos da como resultado una mejora en la velocidad de convergencia.

3.1 Algoritmo de Netravali-Robbins

Tomando en cuenta los fundamentos descritos para las técnicas de estimación de movimiento por Pel-Recursivo discutidas en el capítulo anterior y basándonos en que el objetivo del método es encontrar los vectores de desplazamiento, de los objetos en movimiento, se tiene lo siguiente:

Netravali y Robbins en su algoritmo original, toma dos tramas sucesivas llamadas $I(z,t-\tau)$ (trama pasada) e $I(z,t)$ (trama presente).

Con las cuales se va a tratar de estimar los vectores de desplazamiento que localizaran los pixeles de la trama presente en la trama pasada, ayudándose del error de predicción.

El algoritmo comienza tomando un pixel z_a de la trama presente, y restando su intensidad con el pixel estimado en la trama anterior (considerando siempre que al inicio de nuestro algoritmo tenemos un vector 0, en la estimación).

Dado que $I(z_a,t-\tau)$ es la intensidad en un pixel de la trama pasada e $I(z_a,t)$ es un pixel de la trama presente, lo expresado anteriormente puede ser definido con la siguiente ecuación:

$$DFD(z_a,\hat{D}^i)=I(z_a,t)-I(z_a-\hat{D}^{i-1},t-\tau) \qquad (3.1)$$

Donde DFD es la diferencia de trama desplazada (error de predicción), $\hat{D}^{i-1}$, es el vector de desplazamiento del pixel anterior y $\hat{D}^i$ es la actualización de $\hat{D}^{i-1}$ para hacer esta más exacta, por lo que al transcurrir las iteraciones en el algoritmo, $\hat{D}^i$ pasa a ser $\hat{D}^{i-1}$.

La diferencia de tramas nos va a ayudar en la convergencia del vector de desplazamiento, pues al minimizar recursivamente el error de predicción el algoritmo estará más cerca de encontrar el vector óptimo.

Para la estimación del vector de actualización $\hat{D}^t$, el algoritmo de Netravali-Robbins usa una fórmula donde la estimación del vector correspondiente al pixel anterior $\hat{D}^{t-1}$, es restado al término de actualización. El término de actualización es calculado por medio de la multiplicación de ε, DFD y ∇I, definidos por:

$$\text{Término de actualización} = \varepsilon DFD(z_a, \hat{D}^{t-1}) \nabla I(z_a - \hat{D}^{t-1}, t-\tau) \qquad (3.2)$$

donde

ε, que es una constante pequeña y positiva, la cual nos sirve para controlar tanto la velocidad de convergencia como la precisión en el cálculo del vector.

DFD, que es el error de predicción, misma que nos va a indicar que tan cerca estamos de encontrar el vector de desplazamiento óptimo.

y ∇I, corresponde a la intensidad del gradiente, que es calculada en la trama anterior, y nos indica la posible dirección hacia donde apunta el vector estimado.

Dicha fórmula es mostrada a continuación:

$$\hat{D}^t = \hat{D}^{t-1} - \varepsilon\, DFD(z_a, \hat{D}^{t-1}) \nabla I(z_a - \hat{D}^{t-1}, t-\tau) \qquad (3.3)$$

Con los cálculos obtenidos de DFD, nuestro algoritmo toma dos caminos, uno si el valor absoluto del error de predicción es menor o igual que el umbral ($|DFD|$≤umbral), que es cuando se ha encontrado el vector óptimo; o bien, cuando no hubo movimiento en ese pixel. El otro caso es cuando el valor absoluto del error de predicción es mayor que el umbral ($|DFD|$>umbral), lo que nos indica que aun no tenemos el vector adecuado y se necesitan realizar n iteraciones, para encontrarlo.

Para realizar la nueva estimación $\hat{D}^i$ se toma la estimación del vector anterior $\hat{D}^{i-1}$, y se realiza el cálculo del error de predicción DFD con el valor del vector actualizado, solo si se requiere realizar n iteraciones; puesto que el número de iteraciones depende la exactitud de la estimación del vector. Después de realizar n iteraciones en ese pixel, se pasa al siguiente, tomando como $\hat{D}^{i-1}$ la última estimación del vector en el pixel anterior y se realiza un procedimiento recursivo del método para todos los pixeles restantes.

Tomando como punto de partida las ecuaciones (3.1) y (3.3), la lógica del algoritmo se resume en:

1) Realizar la DFD, con el vector de desplazamiento del pixel anterior, $\hat{D}^{i-1}$.

2) Si $|DFD|$≤umbral, $\hat{D}^{i-1}$ es considerado aceptable.

3) Si $|DFD|$>umbral, necesitamos realizar varias iteraciones. Con cada iteración una nueva estimación $\hat{D}^i$ es obtenida y evaluada, hasta que $|DFD|$≤umbral, o el límite de las iteraciones sea alcanzado.

4) Saltamos al siguiente pixel y se repite desde el paso 1.

Si $\hat{D}^i$ es exacta, entonces DFD llega a ser cero. En la práctica, por diversas razones, un resultado exacto es encontrado en muy raras ocasiones. Por lo tanto se toma un valor de umbral aceptable y se ajustara a él (umbral=3).

El programa correspondiente a este algoritmo, se muestra en el apéndice A (programa 3).

3.1.1 Algoritmo sign-sign de Netravali-Robbins

Después de varias investigaciones, Netravali y Robbins realizaron una simplificación de su algoritmo original en el término de actualización, dando con resultado la siguiente ecuación:

$$\hat{D}^i = \hat{D}^{i-1} - \varepsilon sign\{DFD(z_a, \hat{D}^{i-1})\} sign\{\nabla I(z_a - \hat{D}^{i-1}, t-\tau)\} \qquad (3.4)$$

donde

$sign(v)$=0,1,-1

"*sign*" es el signo del error de predicción DFD y del gradiente ∇I. Tal que la función *sign* toma solo valores de 0,1,-1, esto en vectores rectificados que pueden tomar solo ángulos de múltiplos de 45°. Lo cual evita las multiplicaciones necesarias para los cálculos del término de actualización. De este modo el término de actualización de cada componente de desplazamiento solo puede tomar valores de 0, o $\pm\varepsilon$.

Por lo cual $\hat{D}^i$ solo puede ser actualizado con el valor del factor ε para cada iteración. Este método funciona porque una vez obtenido el vector de

desplazamiento correcto, el movimiento entre pixel y pixel regularmente no cambia drásticamente, excepto en los bordes de los objetos.

La lógica del programa es igual a la del algoritmo original, tan solo con la modificación de la función sign, tanto para el error de predicción (DFD), como para el gradiente(∇I).

El programa correspondiente a este algoritmo, se muestra en el apéndice A (programa 4).

3.2 Algoritmo de Walker-Rao

El algoritmo que a continuación se describe fue introducido por Walker y Rao; este se basa tanto en el algoritmo original de Netravali-Robbins, como en su modificación (algoritmo sign-sign).

Para el algoritmo Walker-Rao se consideró la función de la intensidad en un objeto en movimiento, observando que cuando el valor absoluto del error de predicción ($|DFD|$) es grande y el valor absoluto del gradiente ($|\nabla I|$) es pequeño, el término de actualización es grande, y viceversa; por lo que se decide que el factor de convergencia (ε), varíe acorde al valor del gradiente; con lo cual, la convergencia hacia el vector de estimación óptimo es más rápida, resolviendo con esto el problema que los algoritmos anteriores no habían realizado.

El algoritmo de Walker-Rao se realiza de forma similar a la del algoritmo de Netravali-Robbins, salvo que el factor de convergencia ε no va a ser un valor constante, si no que, va ha tener variaciones dependiendo del gradiente.

Tomando en cuenta lo anterior la fórmula para la predicción del vector de estimación, queda de la siguiente forma:

$$\hat{D}^{i}=\hat{D}^{i-1}-\varepsilon^{i}DFD(z_a,\hat{D}^{i-1})\nabla I(z_a-\hat{D}^{i-1},t-\tau) \tag{3.5}$$

donde

$$\varepsilon^{i}=\frac{1}{2}\ \frac{1}{\left(\left|\nabla I\left(z_a-\hat{D}^{i-1},t-\tau\right)\right|\right)^2} \tag{3.6}$$

y

$$\{|\nabla I(z_a-\hat{D}^{i-1},t-\tau)|\}^2=\{\nabla_x I(z_a-\hat{D}^{i-1},t-\tau)\}^2+\{\nabla_y I(z_a-\hat{D}^{i-1},t-\tau)\}^2$$

como puede verse ε, es ½ multiplicado por uno sobre la suma de las componentes del gradiente al cuadrado.

Observando detenidamente la ecuación pasada nos damos cuenta que al tener un gradiente muy pequeño o cero, surge una indeterminación, por lo cual se debe realizar una modificación de la siguiente forma:

$$\varepsilon^{i}=\frac{1}{\sigma^2+\left(\left|\nabla I\left(z_a-\hat{D}^{i-1},t-\tau\right)\right|\right)^2} \tag{3.7}$$

donde

$$\sigma^2=100$$

σ^2 que es una constante incluida simplemente para resolver el problema de la división entre cero.

Ya definida la fórmula con la cual se realizan los cálculos del vector, el algoritmo sigue el mismo curso que el de Netravali-Robbins, con la diferencia que al calcular el error de predicción (DFD) el algoritmo de Walker-Rao implementa varias opciones a seguir para la optimización, tanto en el tiempo necesario para efectuar los cálculos, como en la velocidad de convergencia del vector de estimación.

La primera opción es cuando el valor absoluto del error de predicción es igual o menor que el umbral ($|DFD| \leq$umbral). En este caso el término de actualización nos va a dar como resultado cero , por lo tanto no es necesario el cálculo del vector de actualización $\hat{D}^t$, puesto que el vector de actualización va a ser igual al vector anterior ($\hat{D}^t = \hat{D}^{t-1}$).

La segunda opción es cuando el valor absoluto del error de predicción es mayor que el umbral ($|DFD| >$umbral) y el valor absoluto del gradiente es diferente de cero ($|\nabla I| \neq 0$). En este caso, se realiza el cálculo del término de actualización, dando como resultado dos posibilidades, la primera de estas es, si el valor absoluto del término de actualización es menor que $^1/_{16}$, (es decir, es un valor muy pequeño, |término de actualización| $< {}^1/_{16}$), para el cual se necesitarían muchas iteraciones para llegar a una estimación ideal, por lo que se ajusta a $\pm {}^1/_{16}$; y a continuación se realiza la actualización del vector $\hat{D}^t$, observando que esto sería igual a la aplicación de la fórmula del algoritmo sign-sign, de Netravali-Robbins. La segunda posibilidad es cuando el valor absoluto del término de actualización sea mayor que 2 (el cual es un valor muy grande para poder realizar una estimación exacta, |término de

actualización|>2), por lo que es ajustado a ±2, y se realiza la estimación del vector $\hat{D}^i$.

Como tercera y última opción tenemos que cuando el valor absoluto del error de predicción es mayor que el umbral ($|DFD|$>umbral), y el valor absoluto del gradiente es cero ($|\nabla I|=0$), el término de actualización es cero, por lo que no es necesario realizar la actualización del vector, puesto que $\hat{D}^i = \hat{D}^{i-1}$.

Tomando como punto de partida las ecuaciones (3.1) y (3.5), la lógica del algoritmo se resume en:

1) Realizar la DFD, con el vector de desplazamiento del pixel anterior, $\hat{D}^{i-1}$.

2) Si $|DFD|$≤umbral, el término de actualización es 0.

3) Si $|DFD|$>umbral, y si $|\nabla I|\neq 0$, el término de actualización es calculado con la ecuación 3.5.

   - Cuando |término de actualización|$<{}^1/_{16}$, al término de actualización se le asigna el valor $\pm{}^1/_{16}$, que es simplemente la ecuación 3.4.

   - Si el |término de actualización|>2, se le asigna a un valor de ± 2.

4) Si $|DFD|$>umbral, y si $|\nabla I|=0$, entonces el término de actualización es 0.

5) Saltamos al siguiente pixel y se repite desde el paso 1.

Recordando que el resultado de la diferencia de trama desplazada es encontrado exacto en raras ocasiones, un umbral es ajustado (umbral=3).

El programa correspondiente a este algoritmo, fué aplicado a los programas 1 y 2 del apéndice A.

3.3 Factores que intervienen en la implementación de los algoritmos de estimación de movimiento

El desempeño de los algoritmos citados anteriormente, depende de varios factores, como son la variación del parámetro ε, el cual controla la velocidad de convergencia; otro de los factores es el cálculo del gradiente, que determina la dirección del vector de desplazamiento. Y como último factor se tomará en cuenta la interpolación que puede ser aplicada en la estimación resultante del vector de actualización para hacer más exactos los cálculos posteriores.

3.3.1 Parámetro de estimación ε

ε, el cual controla la velocidad de convergencia y la precisión del vector de estimación.

En los algoritmos de Netravali-Robbins donde ε es fija, su valor debe estar de acuerdo con los rangos de variación de movimiento de la imagen. Por

ejemplo, con una valor grande de ε, nuestro algoritmo tiene la habilidad de ajustarse rápidamente a cambios de movimiento considerables, pero en movimientos pequeños debido a su valor, el vector puede aumentar en demasía y nunca converger al pixel deseado, dando como resultado una estimación errónea y ruido en la reconstrucción de la imagen. Por otro lado cuando ε toma un valor pequeño, permite realizar un desplazamiento más fino en la estimación del vector, obteniendo con esto la convergencia al pixel óptimo; sin embargo el número de iteraciones aumentaría y con ello los cálculos además del tiempo requerido para la estimación.

Por lo anterior en los algoritmos de Netravali-Robbins, es necesario establecer un equilibrio entre la exactitud del método y el tiempo necesario para los cálculos de estimación.

En el algoritmo de Walker-Rao este problema es resuelto con la variación de ε, acorde con la magnitud del gradiente.

3.3.2 Determinación del gradiente

El gradiente es un método diferencial, el cual en un proceso de digitalización nos sirve para encontrar los bordes de una imagen, es decir, las partes donde existen cambios bruscos de intensidad en los pixeles.

En nuestro algoritmo el gradiente juega un papel muy importante, puesto que nos indica donde hubo un cambio de intensidad, ya sea de área lisa a borde o viceversa; dándonos con su valor la posible dirección del vector estimado para una convergencia satisfactoria.

Como la meta de los algoritmos es encontrar la ubicación de los pixeles de la trama presente en la trama pasada, dicho gradiente es aplicado a los pixeles de la trama anterior que están siendo estimados.

El gradiente puede ser calculado con diversos métodos, de los cuales nos enfocaremos al que utilizaron Netravali y Robbins en su método original.

Este método consiste, en tomar 4 pixeles vecinos al pixel de la trama anterior que esta siendo estimado. Refiriéndonos a la figura 3.1 tenemos que el pixel a estimar es I_5 y los pixeles vecinos son I_2, I_8, I_4 y I_6, que representan la intensidad de los pixeles en dichos puntos (asumiendo que existe un espacio uniforme de los elementos de la imagen).

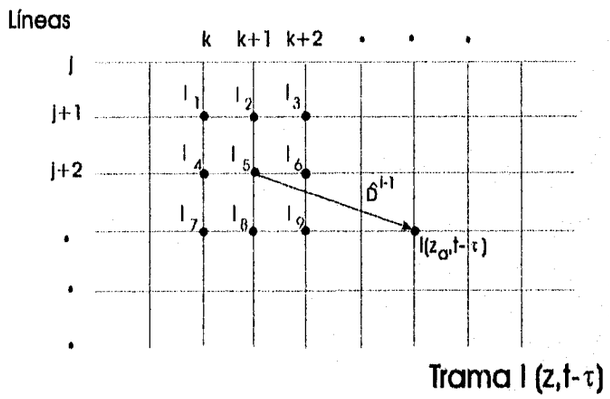


Figura 3.1 Cálculo del gradiente.

Ya que tenemos ubicados los pixeles vecinos, deben calcularse los componentes de la intensidad del gradiente tanto en el eje x, como en el eje y, esto se hace de la siguiente forma:

$$\text{Componente en } x = \frac{I_6 - I_4}{2}$$

$$\text{Componente en } y = \frac{I_8 - I_2}{2} \tag{3.8}$$

El programa correspondiente a este algoritmo, es mostrado en la función 1 del apéndice B.

3.3.3 Interpolación de la estimación del vector

Uno de los problemas que existen en los algoritmo es que al realizar la estimación del vector, si su magnitud es un número real, el valor debe ser truncado, provocando con ello un error en las estimaciones siguientes.

Este problema puede ser solucionado con una interpolación. Existen muchas técnicas posibles de interpolación, pero en este estudio nos limitaremos a la interpolación lineal.

De este modo tomando 4 pixeles vecinos (I_A, I_B, I_C y I_D) que representan la intensidad de los pixeles del lugar donde apunta nuestro vector de estimación con magnitud real (ver figura 3.2) nuestro desplazamiento D, va a estar formado con la suma de la parte entera, D^f, y la parte fraccionaria, D^f, con sus componentes de magnitud D^f_1 y D^f_2, la intensidad que puede ser interpolada por un estándar lineal bidimensional dado por:

$$I=(1-D^f_2)[(1-D^f_1)I_D+D^f_1 I_C]+D^f_2[(1-D^f_1)I_B+D^f_1 I_A] \tag{3.9}$$

Donde I es la intensidad obtenida de la interpolación.

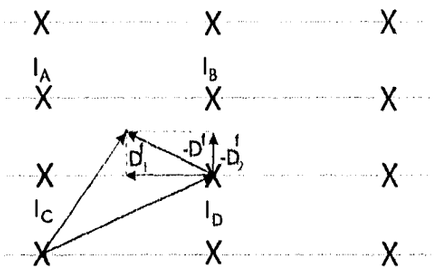


Figura 3.2 Cálculo de la interpolación.

Esta interpolación es usada para todos los algoritmos anteriores.

El programa correspondiente a este algoritmo, es mostrado en la función 2 del apéndice B.

Capítulo 4

Métodos de compresión por compensación de movimiento utilizando un algoritmo Pel-Recursivo

La compensación de movimiento, se realiza conjuntamente con los métodos de estimación de movimiento descritos en el capítulo anterior, y su finalidad es realizar por medio de ella, la transmisión y recepción de la imagen, utilizando de forma óptima tanto el error de predicción, como la estimación de los vectores para realizar la compresión y transmitir un mínimo de datos.

La ventaja de utilizar los métodos de estimación de movimiento Pel-Recursivo en la compensación, es que no es necesario hacer la transmisión de las direcciones de los vectores de estimación para una posterior reconstrucción de la imagen. Esto es posible, porque solo se transmite el error de predicción y la estimación de los vectores se realiza tanto en la transmisión como en la recepción.

4.1 Algoritmo de Transmisión.

Antes de pasar a la descripción detallada de nuestro algoritmo, se debe tomar en cuenta que el proceso de compensación de movimiento en la transmisión se realiza con 3 operaciones fundamentales (ver figura 1.6).

1. Predicción.- A un pixel $z_{n,}$ se le aplica el vector de desplazamiento del pixel anterior (siguiendo el sentido del barrido de la televisión); partiendo de la teoría que el desplazamiento de los pixeles adyacentes se da en un sentido similar.

2. Cuantización.- Es la parte donde el error de predicción es cuantizado y transmitido. Es necesario hacer mención que esta

parte del proceso de compensación de movimiento será simulada para poder realizar el proceso.

3. Estimación.- Si la predicción realizada con el vector de desplazamiento del pixel anterior, no fue satisfactoria, es ajustada con el algoritmo de estimación de movimiento Pel-Recursivo.

Con los fundamentos descritos en el capítulo 2, los métodos de estimación de movimiento para calcular el desplazamiento descritos en el capítulo 3 y apoyándonos en el diagrama de codificación predictiva (ver figura1.6); describiremos nuestro algoritmo de compensación de movimiento en el proceso de transmisión.

Nuestro algoritmo, toma dos tramas sucesivas $I(z,t-\tau)$ (trama pasada) e $I(z,t)$ (trama presente), con las cuales se realizará la compresión de la trama presente, para su transmisión, ayudado por los métodos de estimación de movimiento.

El paso siguiente es realizar la diferencia de trama, que es la resta del pixel de la trama actual con el pixel con la misma ubicación en la trama pasada.

$$FD=I(z_a,t)-I(z_a,t-\tau) \qquad (4.1)$$

A continuación se debe realizar la diferencia de trama desplazada, utilizando el vector de desplazamiento obtenido, en el pixel anterior (recordar que al inicio del algoritmo el vector de estimación inicial tiene un valor 0), realizando con ello el proceso de predicción.

$$DFD(z_a,\hat{D}^i)=I(z_a,t)-I(z_a-\hat{D}^{i-1},t-\tau) \qquad (4.2)$$

En base a los resultados obtenidos de estas dos diferencias, nuestro algoritmo de compensación de movimiento, puede tomar diferentes opciones.

La primera de ellas es cuando, usando el vector de desplazamiento obtenido del pixel anterior; el valor absoluto de la diferencia de trama desplazada es menor o igual que el umbral ($|DFD| \leq$umbral|), lo cual indica que converge al pixel deseado. Realizando entonces la transmisión de un cero.

La segunda opción es cuando, usando el vector de estimación obtenido del pixel anterior, el valor absoluto de la diferencia de trama desplazada es mayor que el umbral, indicando que la aplicación de dicho vector en el pixel actual, es una estimación errónea ($|DFD| >$umbral); y la diferencia de trama es menor o igual a el umbral ($|FD| \leq$umbral), indicando con estas dos condiciones, que el pixel actual coincide con el pixel ubicado en la misma posición en la trama pasada (es decir no hubo movimiento). Cuando se cumplen estas dos condiciones ($|DFD| >$umbral y $|FD| \leq$umbral), un reset es transmitido (que en nuestro algoritmo se le da un valor de 256*), y el vector de estimación se ajusta a cero ($\hat{D}^i=0$), para realizar las posteriores estimaciones.

La tercera y última opción, es cuando el valor absoluto de la diferencia de trama desplazada y la diferencia de trama, son mayores que el umbral ($|DFD| >$umbral y $|FD| >$umbral), indicando que no se ha encontrado la convergencia del pixel. Cuando estas dos condiciones se cumplen debe ser transmitido el error de predicción (DFD) y a continuación, se utiliza $I(z_n,t)$ para hacer la actualización del vector de desplazamiento, utilizando cualquiera de los tres algoritmos de estimación de movimiento vistos en el capítulo anterior, hasta que el valor absoluto de la diferencia de trama desplazada sea menor que el

* En el manejo de nuestras imágenes, se utilizan tonos de grises que tienen un valor de 0-255; dado que cuando se realiza el proceso se hace una diferencia de las intensidades de dos pixeles, obviamente nunca se llegará a un valor arriba de 255.

umbral ($|DFD|$≤umbral) o el número de iteraciones determinadas culmine, el proceso anterior se realiza para obtener un vector de desplazamiento óptimo y poderlo aplicar al siguiente pixel.

Los pasos anteriores son realizados de forma recursiva pixel a pixel y en el sentido del barrido de la televisión.

De lo anterior se desprende la siguiente lógica :

1) Usar $\hat{D}^i$, desplazamiento obtenido del pixel anterior (el valor inicial de $\hat{D}^i$ es igual a 0).

2) Encontrar FD y DFD; utilizando las ecuaciones 4.1 y 4.2.

3) Si $|DFD|$≤umbral, se transmite 0. Si $|DFD|$>umbral y $|FD|$≤umbral, se transmite un reset (256). Y si $|DFD|$>umbral y $|FD|$>umbral, se transmite DFD.

4)

- Si $|DFD|$≤umbral, $\hat{D}^i=\hat{D}^{i-1}$

- Si $|DFD|$>umbral y $|FD|$≤umbral, $\hat{D}^i=0$.

- Si $|DFD|$>umbral y $|FD|$>umbral, se utiliza $I(z_m, t)$, para encontrar un nuevo vector de desplazamiento $\hat{D}^i$, aplicando los algoritmos de estimación de movimiento, del capítulo 3, hasta que $|DFD|$≤umbral o hasta que el límite de las iteraciones sea alcanzado.

5) Saltamos al siguiente pixel (paso 1).

Recordando que, como el resultado obtenido de la diferencia de trama desplazada (DFD) y la diferencia de trama (FD), es encontrado exacto en raras ocasiones, se toma un valor de umbral aceptable y se ajusta a él (umbral=3).

El programa correspondiente a este algoritmo se muestra en el apéndice A (programa 1).

4.2 Algoritmo de Recepción

En la compensación de movimiento el algoritmo de recepción se realiza de forma simétrica, con respecto al proceso de transmisión. Por lo que guiándonos en el diagrama de codificación predictiva (ver figura 1.6), el proceso de compensación de movimiento en la recepción realiza 3 operaciones fundamentales:

1. Predicción.- A un pixel z_a, se le aplica el vector de desplazamiento del pixel anterior (siguiendo el sentido del barrido de la televisión), partiendo de la teoría que el desplazamiento de los pixeles adyacentes se desplazan en sentido similar.

2. Reconstrucción: El error de predicción (DFD, eventualmente codificada para la transmisión, y decodificada en la recepción) es sumado a la intensidad predicha con el vector de estimación del pixel anterior ($\hat{D}^i$), para reconstruir el valor de la intensidad del pixel z_a (intensidad del pixel de la trama presente).

3. Estimación.- Si la predicción realizada con el vector de desplazamiento del pixel anterior, no fue satisfactoria, es ajustada con el algoritmo de estimación de movimiento Pel-Recursivo.

El algoritmo de recepción usa la trama pasada $I'(z,t-\tau)$ y el error de predicción (DFD), obtenido en el algoritmo de transmisión, con los cuales se realizará el proceso de reconstrucción de la trama presente $I'(z,t)$ (donde ' nos indica la intensidad reconstruida), ayudados por los métodos de estimación de movimiento.

Al realizar la recepción, nuestro algoritmo toma diferentes trayectorias, dependiendo de si se recibió un cero (0), un reset (256) o bien el error de predicción (DFD).

Al recibir un cero (0), aplicamos el vector de desplazamiento del pixel anterior, obteniendo con ello el valor de la intensidad del pixel de la trama presente $I'(z_a,t)$.

Al recibir un reset (256), no aplicamos el vector de desplazamiento del pixel anterior, porque el reset nos indica que en el proceso de transmisión el vector del pixel anterior dió una estimación errónea, por lo cual, el vector de desplazamiento es inicializado a cero ($\hat{D}^i=0$), y la intensidad del pixel de la trama presente que esta siendo reconstruida se le asigna la intensidad del pixel de la trama pasada en la misma ubicación ($I'(z_a,t)=I'(z_a,t-\tau)$).

Por último al recibir el error de predicción (DFD), se aplica el vector de desplazamiento del pixel anterior ($\hat{D}^i$), para obtener la predicción del pixel de la trama presente y sumarle el error de predicción, realizando con esto la parte de la reconstrucción. Como este valor del pixel reconstruido no nos da aun el pixel

deseado, se realiza la estimación del vector de desplazamiento de forma simétrica al proceso de transmisión utilizando cualquiera de los tres algoritmos de estimación de movimiento vistos en el capítulo 3 (se utiliza el mismo algoritmo que se utilizo en la transmisión), hasta que el valor absoluto de la diferencia de trama desplazada sea menor que el umbral ($|DFD|\leq$umbral) o el número de iteraciones determinadas culmine, el proceso anterior se realiza para obtener un vector de desplazamiento óptimo y poder realizar la recontrucción del pixel presente ($I'(z_a,t)$).

Los pasos anteriores son realizados de forma recursiva pixel a pixel y en el sentido del barrido de la televisión.

De lo anterior se desprende la siguiente lógica :

1) Usar $\hat{D}^i$, desplazamiento obtenido del pixel anterior (el valor inicial de $\hat{D}^i$ es igual a 0)

2) Si se recibe un 0, $I'(z_a,t)=I'(z_a-\hat{D}^i,t-\tau)$; si se recibe un reset $I'(z_a,t)=I'(z_a,\ t-\tau)$ y si se recibe DFD, se le suma la predicción de $I'(z_a,t)$.

3)

- Si se recibe un 0, $\hat{D}^i=\hat{D}^{i-1}$.
- Si se recibe un reset (256), $\hat{D}^i=0$.
- Si se recibe DFD, se usa $I'(z_a,t)$, para encontrar un nuevo vector de desplazamiento $\hat{D}^i$, aplicando los algoritmos de estimación

de movimiento, del capítulo 3, hasta que $|DFD| \leq$umbral o hasta que el límite de las iteraciones sea alcanzado.

4) Saltamos al siguiente pixel (paso 1).

Recordando que, como el resultado obtenido de la diferencia de trama desplazada y la diferencia de trama, es encontrado exacto en raras ocasiones, se toma un valor de umbral aceptable y se ajusta a él (umbral=3).

El programa correspondiente a este algoritmo se muestra en el apéndice A (programa 2).

Capítulo 5

Evaluación del desempeño de los diferentes algoritmos Pel-Recursivo

Las pruebas realizadas para la evaluación del desempeño de los algoritmos se llevaron acabo con tramas de dos fragmentos de la secuencia "interview" (imagen monocromática).

"Interview" es una secuencia donde dos mujeres se encuentran sentadas en un sillón, la cámara sigue el movimiento que realiza una de ellas al levantarse, de ahí tomaremos el primer fragmento (mujer I), la segunda porción será tomada del rostro de la mujer que permanece sentada (mujer II), teniendo esta última menos movimiento que la anterior. Ambos fragmentos tienen un tamaño de 150 x 200 pixeles.

La evaluación de los algoritmos se dividirá en dos partes:

En la primera se realiza la prueba de convergencia con los algoritmos de estimación de movimiento:

1) Netravali-Robbins original
2) Netravali-Robbins sign-sign
3) Walker-Rao

En la segunda parte se realiza la compensación de movimiento utilizando el algoritmo 3 (Waker-Rao.), simulando la transmisión y recepción.

5.1 Prueba de convergencia con los algoritmos de estimación de movimiento

En esta prueba serán comparados los 3 algoritmos de estimación de movimiento, utilizando la trama 1 y 2 de las secuencias mujer I y mujer II, para estos algoritmos se variará el número de iteraciones por pixel entre 1 y 40.

Aunque los algoritmos se programaron de tal forma que tuvieran características lo mas similares posibles, no existe un exacta comparación puesto que las medidas de funcionamiento no son las mismas.

Los 3 algoritmos fueron probados con el mismo método de cálculo del gradiente e interpolación (ver apéndice B), además con la condición de que si el gradiente es igual a 0 no se realice la estimación (SI $|\nabla I|=0$, $\hat{D}^i=\hat{D}^{i-1}$), por otra parte cuando el vector de estimación sea mayor de 10 se inicializa a 0 (SI $\hat{D}^i>10$, $\hat{D}^i=0$).

Es necesario tomar en cuenta que el parámetro ε, varía para los diferentes métodos, tomando un valor de $^1/_{2048}$ para el algoritmo 1, $^1/_{16}$ para el algoritmo 2 y en el algoritmo 3, ε es variable, además de tener diferentes opciones a seguir cuando se cumple que $|DFD|>3$.

La prueba de convergencia fue realizada debido a que los algoritmos permiten un largo número de iteraciones por pixel, y la estimación termina cuando el valor del desplazamiento es encontrado, (en el caso cuando $|DFD|\leq 3$). Por lo tanto el algoritmo que converge con un pequeño número de iteraciones es considerado mejor. Lo importante de esto es que el algoritmo con

una rápida convergencia tiene la habilidad de tolerar mejor los cambios de diversos movimientos dentro de una imagen.

Resultados del desempeño

Los resultados mostrados a continuación toman en cuenta las siguientes consideraciones:

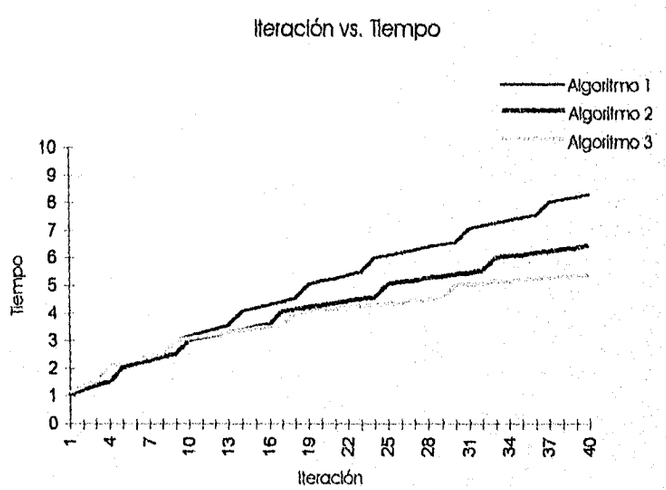
- Las imágenes (mujer I y mujer II), tienen 30,000 pixeles en total cada una.
- Los algoritmos a utilizar son los mostrados en el apéndice A.
- Aproximadamente un 58% de los pixeles se movieron de la trama 1 a la 2 (secuencia mujer I). Y un 50% de los pixeles se movieron de la trama 1 a la 2 (secuencia mujer II). Los pixeles que se consideran con movimiento son donde $|FD|>3$.
- El porcentaje de pixeles no predichos se extrae, de la parte del algoritmo donde $|DFD|>$umbral y $|FD|>$umbral.
- Si un pixel es predicho los cálculos no son requeridos para ese pixel. Por lo tanto los pixeles que se consideran para la búsqueda son estrictamente los no predichos, la búsqueda para un desplazamiento del vector, procede para cada pixel hasta que $|DFD|\leq 3$ o el número de iteraciones termine.Los programas se hicieron y compilaron en Turbo C versión. 2.0.

- Las pruebas se realizaron en una computadora personal 486SX a 33MHz.

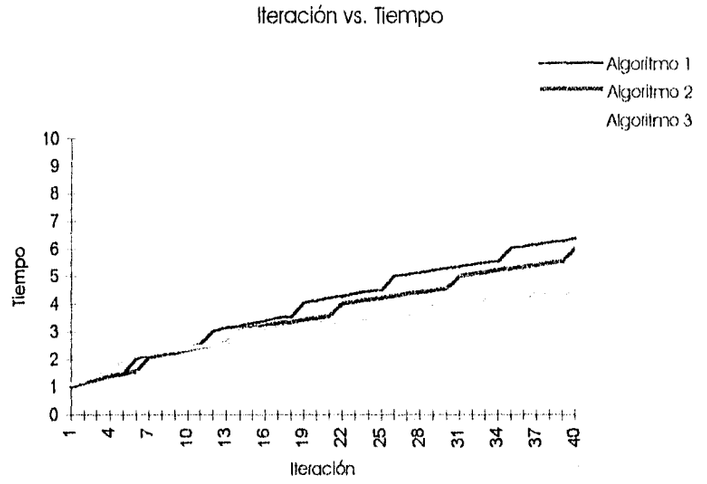
Los algoritmos se evaluaron en cuanto a tiempo y desempeño en la estimación por número de iteración, dando los siguientes resultados:

De la evaluación en cuanto a Iteraciones vs. Tiempo se obtuvieron las gráficas 5.1a para las tramas 1-2 de la secuencia I y 5.1b para las tramas 1-2 de la secuencia II.

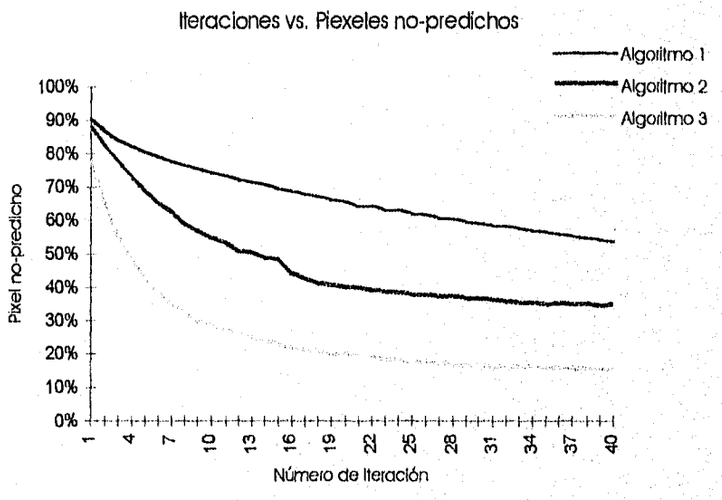
Los resultados obtenidos de Iteraciones vs. pixeles no-predichos, son mostrados en las gráficas 5.1c para las tramas 1-2 secuencia I y 5.1d para las tramas 1-2 de la secuencia II.



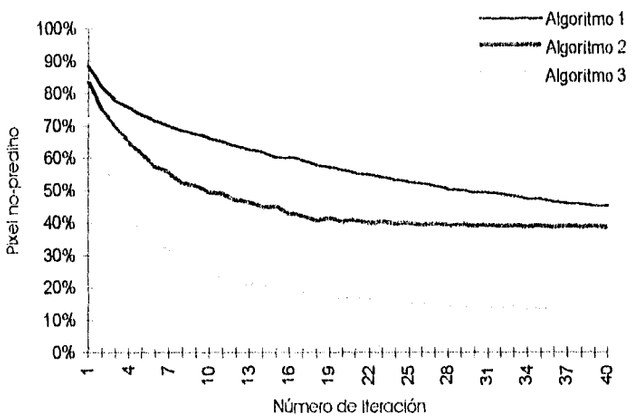
Gráfica 5.1a. Secuencia mujer I, trama 1-2.



Gráfica 5.1b. Secuencia mujer II, trama 1-2.



Gráfica 5.1c. Secuencia mujer I, trama 1-2.



Gráfica 5.1d. Secuencia mujer II, trama 1-2.

Analizando la gráfica 5.1a, tenemos que para 1 iteración los 3 algoritmos tardan aproximadamente lo mismo (1 minuto) para realizar las operaciones, no mostrando una diferencia muy marcada entre ellos, los resultados en cuanto a tiempo se mantienen similares hasta la iteración número 15, donde empieza a notarse que los algoritmos 2 y 3 tienen tiempos similares (3 minutos), pero el algoritmo 1 tiene una diferencia de casi un minuto con los algoritmos 2 y 3. En la iteración 26 el algoritmo 2 aumenta su tiempo comparado con el algoritmo 3 (aproximadamente 30 segundos). A partir de la iteración 31 el algoritmo 2 aumenta, 1 minuto comparado con el algoritmo 3, y el algoritmo 1 aumenta 2 minutos comparado con el algoritmo 3.

En la gráfica 5.1b, para 1 iteración en los 3 algoritmos no existe una diferencia significativa en tiempo, durando aproximadamente 1 minuto para la estimación; y se mantienen similares, hasta la iteración 20 donde existe una

diferencia de 30 segundos entre el algoritmo 2 y 3 con el 1. En la iteración 30 existe una diferencia aproximada de 1 minuto entre el algoritmo 2 y 3, y una diferencia de 1 minuto entre el algoritmo 1 y 3.

Examinando la gráfica 5.1c, podemos notar que a una iteración la convergencia se dio para un 12% de pixeles no predichos en el algoritmo 1, un 16% en el algoritmo 2 y un 27% en el algoritmo 3.

La diferencia entre estos 3 algoritmos se hace más notable a partir de la tercera iteración, donde el algoritmo 1 tiene un 16% de pixeles que convergen, un 22% para el algoritmo 2 y un 45% para el algoritmo 3.

El porcentaje de pixeles no predichos que convergen decrece en forma exponencial hasta la iteración número 19, donde los valores comienzan a ser más estables.

En la gráfica 5.1d, puede notarse que el desempeño es similar al de la gráfica 5.1c, solo que en éste caso el algoritmo 1 no tiene una diferencia tan marcada con respecto al algoritmo 2.

5.2 Compensación de movimiento utilizando el algoritmo de Walker-Rao

En esta parte de la evaluación, se realizara la compensación de movimiento, utilizando el algoritmo que tuvo mejor desempeño en las pruebas realizadas en el inciso anterior (Walker-Rao), realizando la simulación de la transmisón-recepción.

Para hacer la compensación serán utilizadas 6 tramas de la secuencia I y II.

Estas secuencias fueron seleccionadas por la marcada diferencia que existe en la cantidad de desplazamiento entre ellas, puesto que la secuencia I (la mujer que se levanta) tiene un mayor desplazamiento que la secuencia II (mujer que permanece sentada).

En la compensación de movimiento se realizarán las pruebas con 5, 10, 20 y 30 iteraciones por pixel.

La importancia de esta prueba es determinar la habilidad del algoritmo Walker-Rao, aplicándolo a dos tipos de secuencias con desplazamientos diferentes (uno menor que el otro).

Resultados del desempeño

Los resultados mostrados a continuación toman en cuenta las siguientes consideraciones:

- Las secuencias de imágenes (mujer I y mujer II), tienen 30,000 pixeles en total cada una.
- Los algoritmos a utilizar son los mostrados en el apéndice A.
- Como la compensación de movimiento abarca tanto las condiciones de compensación, como el algoritmo de estimación,

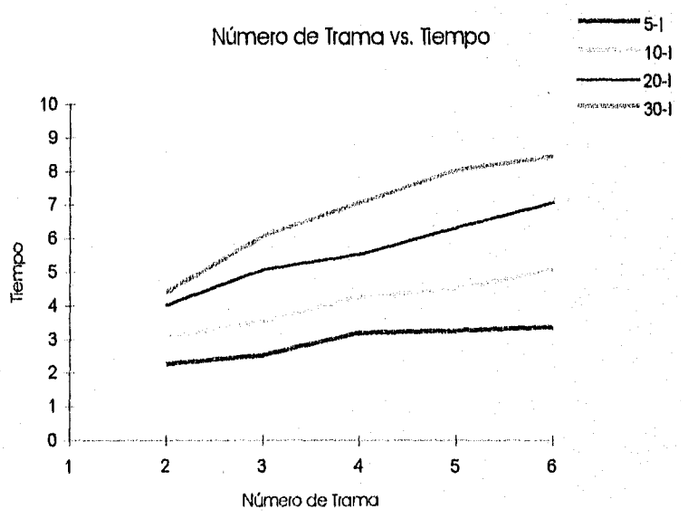
se tomarán el total de pixeles, para obtener el porcentaje de pixeles no compensados.

- Los programas se hicieron y compilaron en Turbo C versión. 2.0.

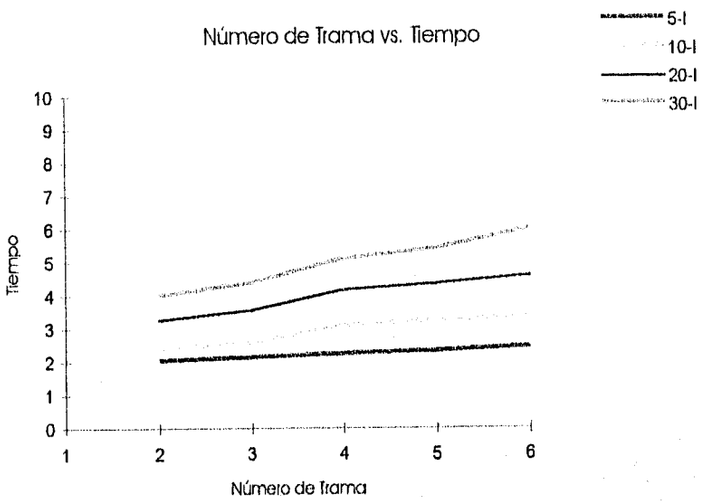
- Las pruebas se realizaron en una computadora personal 486SX a 33MHz.

Los algoritmos se evaluaron en cuanto a tiempo y desempeño en la compensación por número de iteración, dando los siguientes resultados:

Los resultados obtenidos de iteraciones vs. Tiempo, se muestran en las gráficas 5.2a para la secuencia I (trama 1-6) y 5.2b para la secuencia II (trama 1-6).

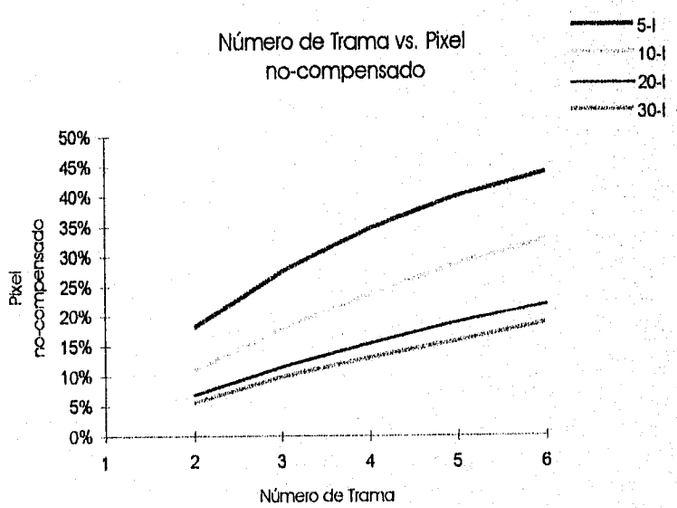


Gráfica 5.2a. Secuencia mujer I, trama 1-6.

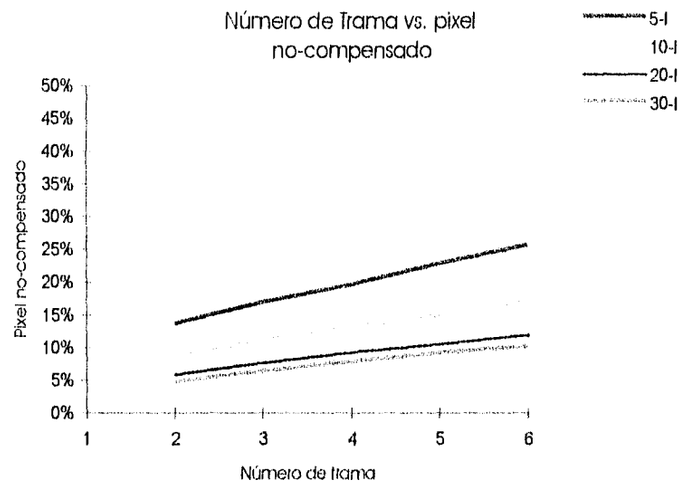


Gráfica 5.2b. Secuencia mujer II, trama 1-6.

Los resultados obtenidos de iteraciones vs. pixeles no-compensados, son mostrados en las gráficas 5.2c para la secuencia I (trama 1-6) y 5.2d para la secuencia II (trama1-6).



Gráfica 5.2c. Secuencia mujer I, trama 1-6.



Gráfica 5.2d. Secuencia mujer II, trama 1-6.

Realizando una comparación de las gráficas 5.2a y 5.2b, podemos observar que el tiempo que tarda en realizar 5 iteraciones, tanto para la secuencia de imágenes I como para la secuencia II es similar; pero conforme aumenta el número de iteraciones, el tiempo tiene un aumento considerable en la secuencia I (imagen de mayor movimiento) comparado con la secuencia II.

Realizando la comparación de las gráficas 5.2c y 5.2d, podemos observar que a 5 iteraciones, la secuencia II en la trama 6, obtiene un 74% de pixeles compensados, mientras que en la secuencia I, de esta misma trama, solo alcanza un 56% de pixeles compensados, esta diferencia se nota más al realizar 10 iteraciones (trama 6), puesto que la secuencia I alcanza un 67% de pixeles compensados mientras que la secuencia II alcanza un 83% de pixeles compensados.

Las imágenes resultantes de las pruebas de compensación de movimiento para 20 iteraciones son mostradas en la figura 5.2 para la secuencia mujer I y figura 5.3 para la secuencia mujer II.

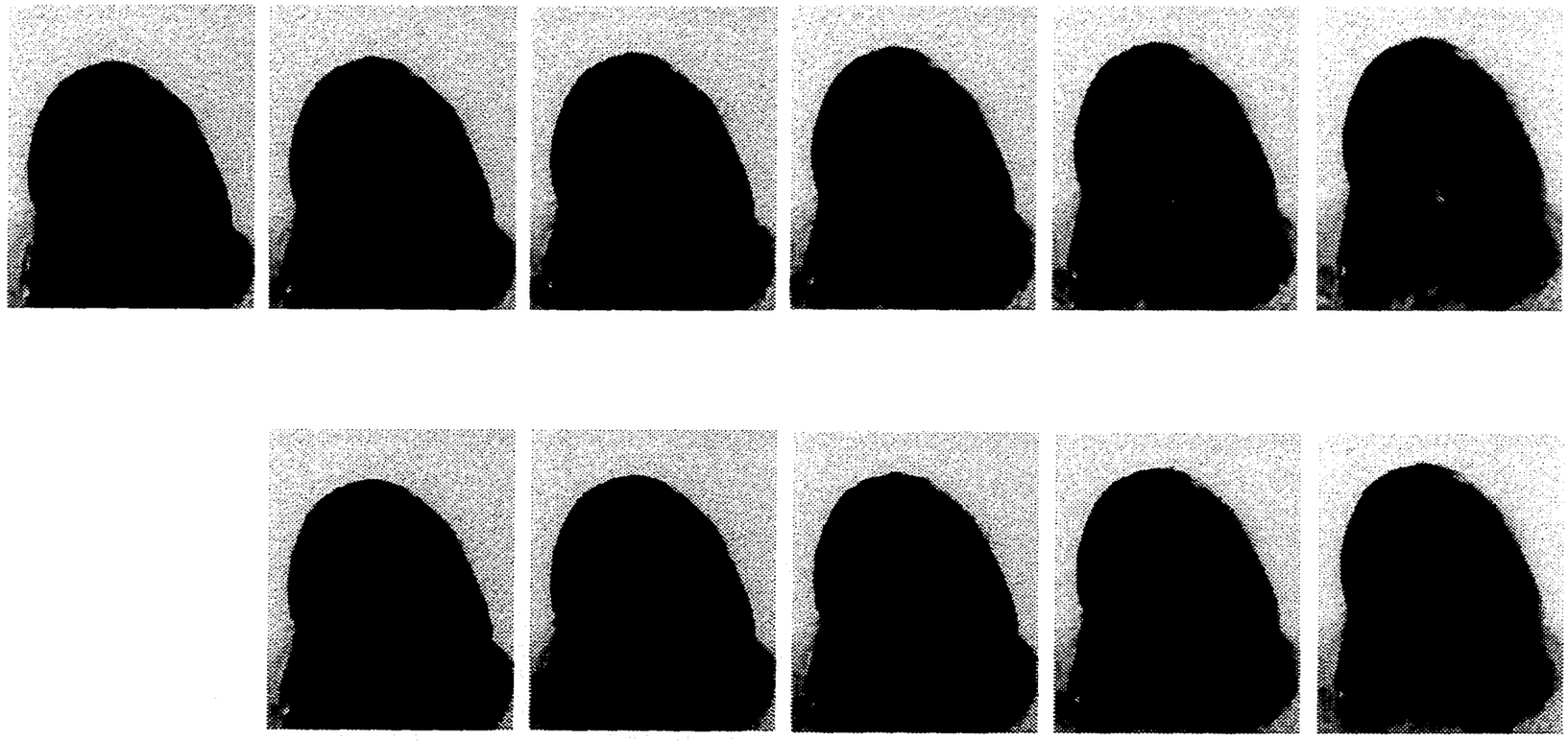


Figura 5.2. En la parte superior se muestra la secuencia original (tramas 1 a 6) y en la parte inferior, la secuencia reconstruida (tramas 2 a 6) de la imagen mujer I.

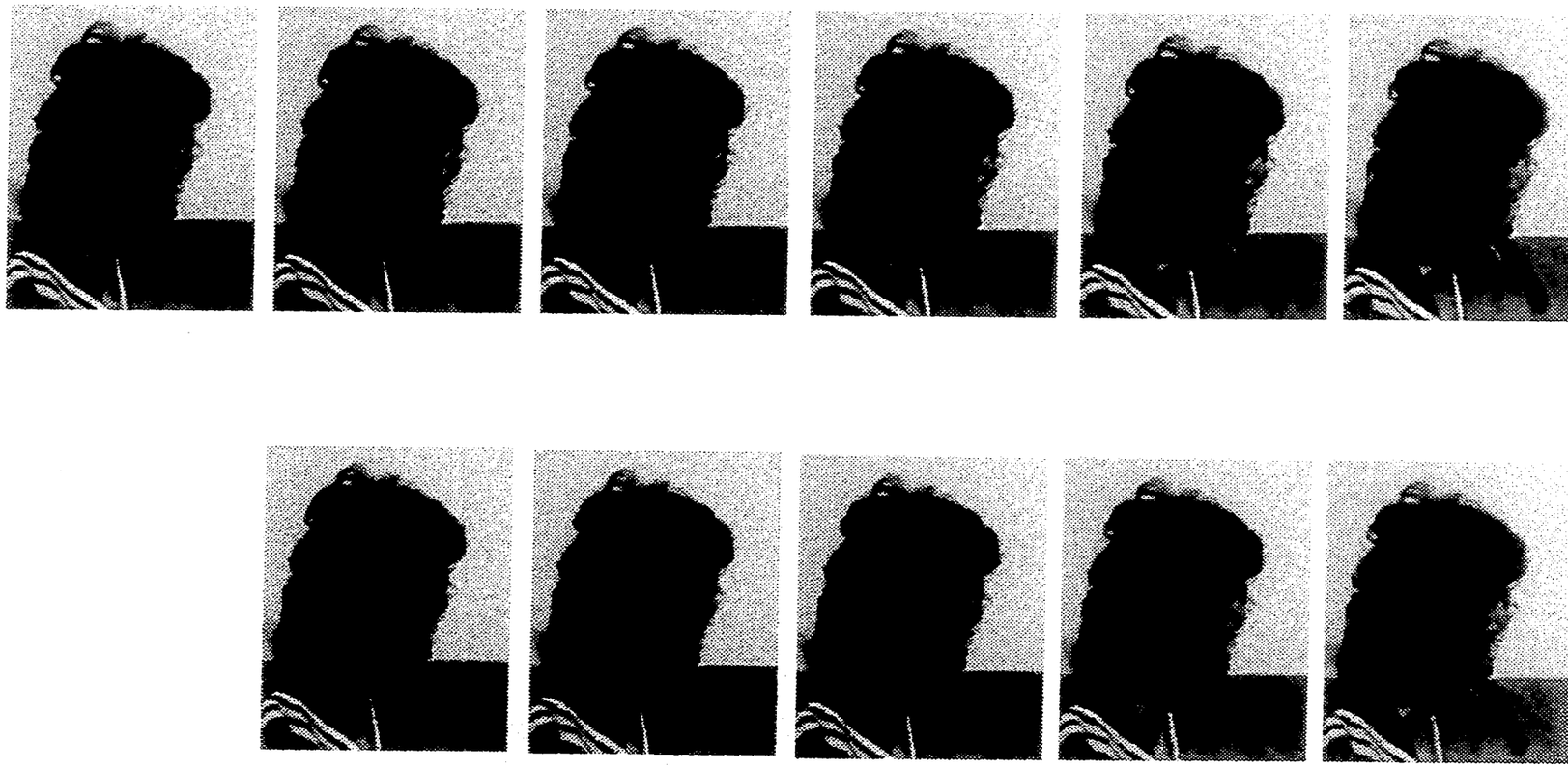


Figura 5.3. En la parte superior se muestra la secuencia original (tramas 1 a 6) y en la parte inferior, la secuencia reconstruida (tramas 2 a 6) de la imagen mujer II.

Capítulo 6

Conclusiones

Nuestro presente estudio fue enfocado a reducir la cantidad de información redundante que existe entre las tramas de televisión, basándonos en un esquema de codificación predictiva, utilizando algoritmos de estimación de movimiento.

Los algoritmos de estimación de movimiento que tomamos para realizar nuestro estudio fueron específicamente los Pel-Recursivo, siendo estos, los elaborados por Netravali-Robbins y Walker-Rao.

Tomando en cuenta los resultados obtenidos de las pruebas realizadas con los algoritmos mencionados, daremos paso a las conclusiones.

El desempeño de los algoritmos de estimación fue de la siguiente manera:

Observando la gráfica 5.1a y 5.1b podemos notar que comparativamente el algoritmo 1 (Netravali-Robbins original) es el menos eficiente en cuanto a tiempo por iteración, en secuencias con un desplazamiento considerable (mujer I); pero en secuencias de imágenes con movimientos pequeños (mujer II) su desempeño es similar a los otros 2 algoritmos, lo cual tiene mucha importancia en aplicaciones de tipo videoconferencia.

El algoritmo 2 (Netravali-Robbins sign-sign), ofrece una mejora aproximadamente de un 22%, con respecto a tiempo comparado con el algoritmo 1, esto se da en secuencias con desplazamientos grandes (mujer I); aunque en secuencias con movimientos poco significativos (mujer II), los tiempos se mantienen similares a los del algoritmo 1.

Los resultados obtenidos del algoritmo 3 (Walker-Rao), nos indican que es el mas eficiente en cuanto a tiempo, obteniendo un mejora de un 34% aproximadamente sobre el algoritmo 1 y un 16% aproximadamente sobre el

algoritmo 2, todo esto en secuencias de movimientos considerables; pero en secuencias con poco movimiento la diferencia en desempeño del tiempo por iteración, entre los 3 algoritmos no es muy significativa, sin embargo podemos hacer notar que a pesar de la mínima discrepancia entre los resultados, el algoritmo de Walker-Rao ofrece los mejores tiempos de estimación.

El segundo parámetro para determinar la eficiencia de este algoritmo, fue la evaluación con respecto a la habilidad de realizar la estimación de los pixeles, dependiendo del número de iteraciones que se realice.

Este parámetro fue determinado, por el porcentaje de pixeles no predichos contra el número de iteración.

Tomando en cuenta que el porcentaje de pixeles no predichos, se obtiene de la razón de los pixeles no predichos, contra los pixeles que no se estimaron al final de la iteración.

Se observa que el algoritmo que tiene el mejor desempeño, es el de Walker-Rao, teniendo una mejora de 70% y 54% aproximadamente, contra el algoritmo 1 y 2 respectivamente, esto para secuencias con mucho movimiento; y en secuencias con movimientos mínimos tiene un porcentaje de mejora de un 73% y 65% sobre los algoritmos 1 y 2 respectivamente.

Por otra parte, al inicio de las iteraciones los algoritmos 1 y 2, tienen aproximadamente el mismo porcentaje de pixeles no-predichos, sin embargo el algoritmo 3, comienza a mostrar mejoras significativas, conservando este mismo desempeño, a lo largo de las iteraciones.

En el desarrollo de las pruebas, se constato que los 3 algoritmos tienen un límite funcional, este límite nos indica que a cierta cantidad de iteraciones el porcentaje de pixeles no-predichos no sufre cambios significativos.

Basándonos en lo anterior es necesario hacer notar que los algoritmos pueden ser susceptibles a mejoras, esto puede ser un tema para futuras investigaciones realizando estudios con diferentes formas de calcular el gradiente y la interpolación.

Después de haber concluido que el algoritmo de Walker-Rao es el que tuvo un mejor desempeño en cuanto a calidad de estimación, pasaremos a la parte de la compensación de movimiento, donde se realizo la transmisión-recepción aplicando solamente este algoritmo.

Realizando la comparación, entre las gráficas 5.2a y 5.2b podemos determinar, que el tiempo sufre un incremento acumulado con respecto a las tramas que se van procesando, siendo esto más marcado en secuencias de desplazamientos considerables (mujer II).

Otra de las observaciones que debemos hacer notar, es que en secuencias con grandes movimientos, el porcentaje de pixeles no compensados aumentaron, por el contrario en secuencias de poco movimiento este porcentaje se mantiene relativamente estable.

Con esto se da por terminado el presente estudio de Codificación de secuencias de imágenes de televisión utilizando algoritmos de estimación de movimiento Pel-Recursivo.

Apéndice A

Programa 1

```
/* ************************************************/
/*          Compensación de movimiento por el      */
/*          método de Walker-Rao - Transmisión -   */
/* ********************************************** */
            /*Elaborado por: Julieta Cortés Carmona */
            /* Carrera: Ing. en Computación         */
            /*No. de cuenta: 8400212-9              */

main()

{           /* Abre menu principal*/
 float dix, diy, dix_n, diy_n, grad, eee, t_cx, t_cy;
 const float e = 1.0/16.0, ee=2.0;
 int indice, fd, dfd, pix, dix_e, diy_e, t, n_itera;
 FILE *archivo3, *resultados;

cargar_imagen();
cargar_imagen2();

 gotoxy(3,3);printf("Número de iteraciones:");
 gotoxy(3,4);scanf("%d",&n_itera);

archivo3 = fopen(nombre,"wb");

 if(!archivo3)
 {printf("Imposible abrir archivos.\n");
  exit(0); }

 regresa:
 while (i<r && j<c)
 {          /* Abre condición para que haga compensación de mov
                        hasta que termine la imagen t-1 */

pix=fgetc(archivo2);
    fd = pix-(imag[c*i+j]); /* resta la intensidad actual - el pixel de
                              la trama pasada en la misma ubicación */
    dix_n = (float) j-dix; /* localizamos el lugar del pixel en la trama pasada */
    diy_n = (float) i-diy;
    dix_e = (int) dix_n;
    diy_e = (int) diy_n;
    dix_f = dix_n - (float) dix_e;
    diy_f = diy_n - (float) diy_e;

    x = dix_e;
    y = diy_e;
     realiza_trunc();
    dfd = pix - realiza_interp();

    if(abs(dfd) <= 3)
```

```
{       /* Abre 1ª condición si dfd<=umbral */
    t=0;
    fprintf(archivo3,"%d,",t);
    j++;  /* incrementa las x's */
    if(j >= c)
    {j=0;  i++; }
}       /* Cierra 1ª condición si dfd<=umbral */

else if(abs(dfd) > 3 && abs(fd) <= 3)
{       /* Abre 2ª condición si dfd>umbral y fd<=umbral*/
    t=256;
    fprintf(archivo3,"%d,", t);
    dix=0.0;  diy=0.0;
    j++; if(j >= c)
    { j=0;  i++; }
}       /* Cierra 2ª condición si dfd>umbral y fd<=umbral*/

else if(abs(dfd) > 3 && abs(fd) > 3)
{       /* Abre 3ªcondición si dfd>umbral y fd>umbral */
    t=dfd;
    fprintf(archivo3,"%d,", t);
    indice=0;

    while (abs(dfd)>=3 && indice<n_itera)
    {   /* Abre condición para n iteraciones */
      regresa2:
      realiza_gradiente();
      grad=(ix*ix)+(iy*iy);
      if(grad!=0)
      {         /* Abre condición si gradiente es diferente de 0
                        realiza método de Walker-Rao */

          eee = (1.0/(100.0+grad));
          t_cx = eee * (float) dfd * ix;
          t_cy = eee * (float) dfd * iy;

          if ((fabs(t_cx)==0 || (fabs(t_cx)>=e && fabs(t_cx)<=ee))
                  && (fabs(t_cy)<e && fabs(t_cy)!=0))
            {   /*Condición cuando el valor absoluto del término
                  de corrección en x=0  e y<1/16  */

              realiza_sign();
              dfd = realiza_signd(dfd);
              t_cy = e * (float) dfd * iy;
             }

          else if ((fabs(t_cx)<e && fabs(t_cx)!=0)
          &&  (fabs(t_cy)==0 || (fabs(t_cy)>=e && fabs(t_cy)<=ee)))
            {   /*Condición cuando el valor absoluto del término
                        de corrección en x < 1/16  e  y=0 */
              realiza_sign();
              dfd = realiza_signd(dfd);
              t_cx = e * (float) dfd * ix;
```

```
 }

else if ((fabs(t_cx)<e && fabs(t_cx)!=0) &&
            (fabs(t_cy)<e && fabs(t_cy)!=0))
 {    /*Condición cuando el valor absoluto del término
                    de corrección en x < 1/16 e y<1/16 */
  realiza_sign();
  dfd = realiza_signd(dfd);
  t_cx = e * (float) dfd * ix;
  t_cy = e * (float) dfd * iy;
 }

else if ( (fabs(t_cx)==0 || (fabs(t_cx)<=ee && fabs(t_cx)>=e))
         && (fabs(t_cy)>ee && fabs(t_cy)!=0))
 {    /*Condición cuando el valor absoluto del término
                    de corrección en x=0 e y>2 */
 realiza_sign();
 dfd = realiza_signd(dfd);
 t_cy = ee * (float) dfd * iy;
 }

else if ((fabs(t_cx)>ee &&fabs(t_cx)!=0)
 &&  (fabs(t_cy)==0 ||(fabs(t_cy)<=ee && fabs(t_cy)>=e)))
 {    /*Condición cuando el valor absoluto del término
                    de corrección en x>2 e y=0 */
 realiza_sign();
 dfd = realiza_signd(dfd);
 t_cx = ee * (float) dfd * ix;
 }

else if ((fabs(t_cx)>ee && fabs(t_cx)!=0) &&
         (fabs(t_cy)>ee && fabs(t_cy)!=0))
 {    /*Condición cuando el valor absoluto del término
                    de corrección en x>2 e y>2 */
 realiza_sign();
 dfd=realiza_signd(dfd);
 t_cx = ee * (float) dfd * ix;
 t_cy = ee * (float) dfd * iy;
 }

dix = dix - t_cx;
diy = diy - t_cy;

 if (fabs(dix)>=10.0 || fabs(diy)>=10.0)
  { /* Abre condición para controlar que no
                    se dispare el valor del vector */
   dix=0.0;
   diy=0.0;
   goto regresa2;
  } /* Cierra condición para controlar que no
                    se dispare el valor del vector */
dix_n= (float) j - dix;
diy_n= (float) i - diy;
```

```
                dix_e= (int) dix_n;
                diy_e= (int) diy_n;
                dix_f=dix_n - (float) dix_e;
                diy_f=diy_n - (float) diy_e;
                x=dix_e;
                y=diy_e;

                realiza_trunc();
                dfd=pix-realiza_interp();

                 if (abs(dfd)<=3 )
                   {  /* Abre 1ª condición si dfd<=umbral
                                 dentro de condición 3ª */
                         j++; if(j>=c){
                          j=0; i++; }
                          goto regresa;
                   }  /* Cierra 1ª condición si dfd<=umbral
                                 dentro de condición 3ª */

                 indice++;
             }   /* Cierra condición si gradiente es diferente de 0
                           realiza método de Walker-Rao*/
              else break;

            } /* Cierra condición para iteraciones, hasta n */
            dix = 0;
            diy = 0;
             i++; if(j>=c){
              j=0; i++; }
    } /* Cierra 3ª condición si dfd>umbral y fd>umbral */

   }   /* Cierra condición para que haga compensación de mov
                       hasta que termine la imagen t-1 */

  fclose(archivo2);
  fclose(archivo3);
  farfree((void *) imag);
  clrscr();
  return;
} /* Cierra menu principal*/
```

Programa 2

```
/* ********************************************** */
/*          Compensación de movimiento por el      */
/*           método de Walker-Rao - Recepción -    */
/* ********************************************** */
            /* Elaborado por: Julieta Cortés Carmona */
            /* Carrera: Ing. en Computación          */
            /* No. de cuenta: 8400212-9              */

main()
{           /*Abre menu principal */

 float dix, diy, dix_n, diy_n, grad, t_cx, t_cy, eee;
 const float e = 1.0/16.0, ee = 2.0 ;
 int indice, fd, dfd, pix, pix_estima, pix_presente, pix_predic, dix_e, diy_e, recib,n_itera;
 FILE *archivo3;

cargar_imagen();
cargar_dfd();

  gotoxy(3,3);printf("Número de iteraciones:");
  gotoxy(3,4);scanf("%d",&n_itera);

 archivo3 = fopen(nombre," wb");

 if(!archivo3)
 { printf("Imposible abrir archivos.\n");
   exit(0); }

 regresa:
 while (i<r && j<c)
  {         /* Abre condición para que haga compensación de mov
                        hasta que termine la imagen t-1 */

   fscanf(archivo2,"%d,",&recib);

   dix_n = (float) j-dix;
   diy_n = (float) i-diy;
    dix_e = (int) dix_n;
   diy_e = (int) diy_n;
   dix_f = dix_n - (float) dix_e;
   diy_f = diy_n - (float) diy_e;
    x = dix_e;
    y = diy_e;

   realiza_trunc();
   pix_predic = realiza_interp();

   if (recib==0 ) /* Abre 1ª condición si recibe 0 */
   {
```

```
 pix = pix_predic;
 fprintf(archivo3,"%c", pix);
  j++; if(j>=c){
        j=0; i++; }
}        /* Cierra 1ª condición si recibe 0 */

else if (recib==256)
{        /* Abre 2ª condición si recibe reset */
 dix=0.0;
 diy=0.0;
 pix=imag[c*i+j];
 fprintf(archivo3,"%c", pix);
      j++; if(j>=c){
        j=0; i++; }
}        /* Cierra 2ª condición si recibe reset */
else if (recib!=0 && recib!=256)
{        /* Abre 3ª condición si recibe DFD */
 dfd=recib;
 pix_presente = recib + pix_predic;
 indice=0;

     while (indice < n_itera)
      {  /* Abre condición para n iteraciones */
        regresa2:
        realiza_gradiente();
        grad=(ix*ix)+(iy*iy);
        if(grad!=0)  /* Abre condición si gradiente es diferente de 0
                        realiza método de Walker-Rao*/
         {
           eee = (1.0/(100.0+grad));
           t_cx = eee * (float) dfd * ix;
           t_cy = eee * (float) dfd * iy;

           if ((fabs(t_cx)==0 || (fabs(t_cx)>=e && fabs(t_cx)<=ee))
                  && (fabs(t_cy)<e && fabs(t_cy)!=0))
             {    /*Condición cuando el valor absoluto del término
              de corrección en x=0  e y<1/16  */
              realiza_sign();
              dfd = realiza_signd(dfd);
              t_cy = e * (float) dfd * iy;
             }

           else if ((fabs(t_cx)<e && fabs(t_cx)!=0)
           &&  (fabs(t_cy)==0 || (fabs(t_cy)>=e && fabs(t_cy)<=ee)))
             {   /*Condición cuando el valor absoluto del término
                        de corrección en x < 1/16  e  y=0 */
              realiza_sign();
              dfd = realiza_signd(dfd);
              t_cx = e * (float) dfd * ix;
             }
           else if ((fabs(t_cx)<e && fabs(t_cx)!=0) &&
                    (fabs(t_cy)<e && fabs(t_cy)!=0))
             {    /*Condición cuando el valor absoluto del término
```

```
                        de corrección en x < 1/16  e  y<1/16 */
    realiza_sign();
    dfd = realiza_signd(dfd);
    t_cx = e * (float) dfd * ix;
    t_cy = e * (float) dfd * iy;
   }

  else if ( (fabs(t_cx)==0 || (fabs(t_cx)<=ee && fabs(t_cx)>=e))
          && (fabs(t_cy)>ee && fabs(t_cy)!=0))
   {    /*Condición cuando el valor absoluto del término
                        de corrección en x=0  e  y>2 */
    realiza_sign();
    dfd = realiza_signd(dfd);
    t_cy = ee * (float) dfd * iy;
    }

  else if ((fabs(t_cx)>ee &&fabs(t_cx)!=0)
   &&  (fabs(t_cy)==0 ||(fabs(t_cy)<=ee && fabs(t_cy)>=e)))
   {    /*Condición cuando el valor absoluto del término
                        de corrección en x>2  e  y=0 */
    realiza_sign();
    dfd = realiza_signd(dfd);
    t_cx = ee * (float) dfd * ix;
    }

  else if ((fabs(t_cx)>ee && fabs(t_cx)!=0) &&
          (fabs(t_cy)>ee && fabs(t_cy)!=0))
   {    /*Condición cuando el valor absoluto del término
                      de corrección en x>2  e y>2 */
    realiza_sign();
    dfd=realiza_signd(dfd);
    t_cx = ee * (float) dfd * ix;
    t_cy = ee * (float) dfd * iy;
    }

  dix = dix - t_cx;
  diy = diy - t_cy;

   if (fabs(dix)>=10.0 || fabs(diy)>=10.0)
   {    /* Abre condición para controlar que no se
                              dispare el valor del vector */
        dix = 0.0;
        diy = 0.0;
        goto regresa2;
   }  /* Cierra condición para controlar que no se
                              dispare el valor del vector */
dix_n = (float) i-dix;
diy_n = (float) i-diy;
dix_e = (int) dix_n;
diy_e = (int) diy_n;
dix_f = dix_n - (float) dix_e;
diy_f = diy_n - (float) diy_e;
```

```
            x = dix_e;
            y = diy_e;

            realiza_trunc();
            pix_estima=realiza_interp();

            dfd=pix_presente-pix_estima;
            if (abs(dfd)<=3)
            {       /* Abre condición si dfd<=umbral */
                    pix=pix_estima;

                    fprintf(archivo3,"%c", pix);
                      j++; if(j>=c){
                              j=0; i++; }
                     goto regresa;
                    }       /* Cierra condición si dfd <=umbral */
              indice++;

             }      /* Cierra condición si gradiente es diferente de 0
                                realiza método de Walker-Rao*/
              else break;
             }      /* Cierra condición para n iteraciones */

            pix=pix_estima;

            fprintf(archivo3,"%c", pix);
              dix = 0;
              diy = 0;
               j++; if(j>=c){
                      j=0; i++; }
        }  /* Cierra 3ª condición si recibe DFD */

    }  /* Cierra condición para que haga compensación de mov
                              hasta que termine la imagen t-1 */

    fclose(archivo2);
    fclose(archivo3);
    farfree((void *) imag);
    clrscr();
    return ;

    }   /* Cierra menu Principal */
```

Programa 3

```
/* ********************************************** */
/*          Compensación de movimiento por el      */
/*          método de Netravali-Robbins original   */
/*                  Transmisión/Recepción -        */
/* ********************************************** */

          /* Elaborado  por: Julieta Cortés Carmona */
          /* Carrera: Ing. en Computación           */
          /* No. de cuenta: 8400212-9               */

        while (abs(dfd)>=3 && indice<n_itera)
        {    /* Abre condición para n iteraciones */
          regresa2:
          realiza_gradiente();
          grad=(ix*ix)+(iy*iy);

          if(grad!=0)
          {          /* Abre condición si gradiente es diferente de 0
                realiza método de Netravali-Robbins original*/
                dix = dix - (e * (float) dfd * ix);
                diy = diy - (e * (float) dfd * iy);

                 if (fabs(dix)>=10.0 || fabs(diy)>=10.0)
                  { /* Abre condición para controlar que no
                                se dispare el valor del vector */
                    dix=0.0;
                    diy=0.0;
                    goto regresa2;
                   } /* Cierra condición para controlar que no
                                se dispare el valor del vector */
                dix_n= (float) i - dix;
                diy_n= (float) i - diy;
                dix_e= (int) dix_n;
                diy_e= (int) diy_n;
                dix_f=dix_n - (float) dix_e;
                diy_f=diy_n - (float) diy_e;
                x=dix_e;
                y=diy_e;
                realiza_trunc();
                dfd=pix-realiza_interp();
                 if (abs(dfd)<=3 )
                  {  /* Abre 1ª condición si dfd<=umbral
                                dentro de condición 3ª */
                        j++; if(j>=c){
                         j=0; i++; }
                          goto regresa;
                  }  /* Cierra 1ª condición si dfd<=umbral
                                dentro de condición 3ª */
                indice++;
```

```
    }    /* Cierra condición si gradiente es diferente de 0
            realiza método de Netravali-Robbins original */
    else break;
  } /* Cierra condición para iteraciones, hasta n */
```

Programa 4

```
/* ********************************************** */
/*          Compensación de movimiento por el      */
/*          método de Netravali-Robbins   sign-sign */
/*                  Transmisión/Recepción -         */
/* ********************************************** */

          /* Elaborado  por: Julieta Cortés Carmona */
          /* Carrera: Ing. en Computación           */
          /* No. de cuenta: 8400212-9               */

        while (abs(dfd)>=3 && indice<n_itera)
        {    /* Abre condición para n iteraciones */
          regresa2:
          realiza_gradiente();
          grad=(ix*ix)+(iy*iy);
          if(grad!=0)
          {          /* Abre condición si gradiente es diferente de 0
                realiza método de Netravali-Robbins sign-sign */

                realiza_sign();
                dfd=realiza_signd(dfd);
                dix = dix - (e * (float) dfd * ix);
                diy = diy - (e * (float) dfd * iy);

                 if (fabs(dix)>=10.0 || fabs(diy)>=10.0)
                  { /* Abre condición para controlar que no
                                    se dispare el valor del vector */
                    dix=0.0;
                    diy=0.0;
                    goto regresa2;
                    } /* Cierra condición para controlar que no
                                    se dispare el valor del vector */
               dix_n= (float) i - dix;
               diy_n= (float) i - diy;
               dix_e= (int) dix_n;
               diy_e= (int) diy_n;
               dix_f=dix_n - (float) dix_e;
               diy_f=diy_n - (float) diy_e;
               x=dix_e;
               y=diy_e;
               realiza_trunc();
               dfd=pix-realiza_interp();
```

```
            if (abs(dfd)<=3 )
              {  /* Abre 1ª condición si dfd<=umbral
                            dentro de condicón 3¦ */

                    j++; if(j>=c){
                     j=0;  i++; }
                     goto regresa;
              }  /* Cierra 1ª condición si dfd<=umbral
                            dentro de condición 3ª */

               indice++;
          }    /* Cierra condición si gradiente es diferente de 0
                        realiza método de Netravali-Robbins sign-sign */
           else break;
        } /* Cierra condición para iteraciones, hasta n */



/* ********************************************** */
/*              Función  signo del gradiente                */
/* ********************************************** */
            /*Elaborado por: Julieta Cortés Carmona */
            /* Carrera: Ing. en Computación          */
            /*No. de cuenta: 8400212-9                */

void realiza_sign(void)
{
  if (ix>0 && iy>0)
    { ix = 1.0;  iy = 1.0; }

  else if (ix<0 && iy<0)
    { ix = -1.0;  iy = -1.0; }

  else if (ix>0 && iy<0)
    { ix = 1.0;  iy = -1.0; }

  else if (ix<0 && iy>0)
    { ix = -1.0;  iy = 1.0; }

  else if (ix<0 && iy==0)
      ix = -1.0;

 else if (ix>0 && iy==0)
      ix = 1.0;

  else if (ix==0 && iy<0)
      iy = -1.0;

  else if (ix==0 && iy>0)
      iy = 1.0;
}
```

```
/* ********************************************** */
/*                  Función  signo de DFD                  */
/* ********************************************** */
            /*Elaborado por: Julieta Cortés Carmona */
            /* Carrera: Ing. en Computación              */
            /*No. de cuenta: 8400212-9                   */

int realiza_signd(int dfd_sign)
  {
  if (dfd_sign>0) return 1;
   else if (dfd_sign<0) return -1;
    return dfd_sign;
  }
```

* En los programas 3 y 4 solamente se lista la parte de estimación, dado que es la única porción del programa que cambia para el algoritmo, con respecto a los programas 1 y 2.

Apéndice B

Función 1

```
/* ********************************************** */
/*                  Función gradiente                 */
/* ********************************************** */
          /*Elaborado por: Julieta Cortés Carmona */
          /* Carrera: Ing. en Computación          */
          /*No. de cuenta: 8400212-9               */

void realiza_gradiente(void)
{
   if (y==0 && x==0)
     {  ix=( (float) imag[c*y+(x+1)])/2.0;
        iy=( (float) imag[c*(y+1)+x])/2.0; }

    else if  (y==0 && xl=0 && xl=(c-1))
       { ix= (( (float) imag[c*y+(x+1)])-( (float) imag[c*y+(x-1)]))/2.0;
        iy=( (float) imag[c*(y+1)+x])/2.0; }

    else if  ( y==0 && x==(c-1))
       { ix=( (float) imag[c*y+(x-1)])/2.0;
         iy=( (float) imag[c*(y+1)+x])/2.0; }

     else if  ( x==0 && yl=0 && yl=(r-1))
       {          lx=( (float) imag[c*y+(x+1)])/2.0;
        iy=(( (float) imag[c*(y+1)+x])-( (float) imag[c*(y-1)+x]))/2.0; }

     else if  ( x==0 && y==(r-1))
       { ix=( (float) imag[c*y+(x+1)])/2.0;
         iy=( (float) imag[c*(y-1)+x])/2.0; }

     else if  ( y==(r-1) && xl=0 && xl=(c-1))
       { ix=(( (float) imag[c*y+(x+1)])-( (float) imag[c*y+(x-1)]))/2.0;
         iy=( (float) imag[c*(y-1)+x])/2.0; }

    else if  ( y==(r-1)&& x==(c-1)) /* SIETE */
       { ix=( (float) imag[c*y+(x-1)])/2.0;
        iy=( (float) imag[c*(y-1)+x])/2.0; }

     else if  ( x==(c-1) && yl=0 && yl=(r-1))
       { ix=( (float) imag[c*y+(x-1)])/2.0;
         iy=(( (float) imag[c*(y+1)+x]- (float) imag[c*(y-1)+x]))/2.0;}

      else if ((xl=0) && (xl=(c-1)) && (yl=0) && (yl=(r-1)))
       { ix=(( (float) imag[c*y+(x+1)])-( (float) imag[c*y+(x-1)]))/2.0;
         iy=(( (float) imag[c*(y+1)+x])-( (float) imag[c*(y-1)+x]))/2.0; }
return;
}
```

Función 2

```
/* ************************************************/
/*                  Función interpolación                  */
/* ************************************************/
          /*Elaborado por: Julieta Cortés Carmona */
          /* Carrera: Ing. en Computación          */
          /*No. de cuenta: 8400212-9               */

int realiza_interp(void)
  {
   float intensidad, fraccion_y, fraccion_x, inten_punto, ia, ib, ic, id;

   fraccion_y = 1.0 - dif_y_f;
   fraccion_x = 1.0 - dif_x_f;
   id = (float) imag[c*y+x];
   ib = (float) imag[c*y+(x+1)];
   ic = (float) imag[c*(y+1)+x];
   ia = (float) imag[c*(y+1)+(x+1)];

   intensidad = (fraccion_y * ((fraccion_x * id) + (dif_x_f * ic))) +
                (dif_y_f * ((fraccion_x * ib) + (dif_x_f * ia)));

   return (int) intensidad;
  }
```

* Estas funciones son usadas para todos los programas del apéndice A.

Bibliografía

1) Jain, Anil K.
Fundamentals of digital image processing
Ed. Prentice-Hall, primera edición, Englewood Cliffs, New Jersey, 1989
pp. 569

2) Netravali, Arun N. y Haskell, Barry G.
Digital pictures: representation and compression
Ed. Plenum Press, primera edición, New York, N.Y., 1988
pp. 586

3) Proakis, John G.
Digital Communications
Ed. Mc Graw, Hill, segunda edición, United States of America, 1989
pp. 905

4) Gonzalez, Rafael y Woods, Richard E
Digital Image Processing
Ed. Adision Wesley, primera edición, United States of America, 1992
pp. 716

5) Gonzalez, Rafael y Wintz, Paul
Digital Image Processing
Ed. Adision Wesley, segunda edición, United States of America, 1987
pp. 503

6) Pratt, William K.
Digital Image Processing
Ed. Wiley-Interscience, segunda edición, United States of America, 1991
pp. 698

7) Limb, John O. y Murphy, J. A.
Measuring the Speed of Moving Objects from Television Signals
IEEE Transactions on Communications. Vol. COM-23, No. 4, abril 1975
pp. 474-478

8) Cafforio, Ciro y Rocca, Fabio
Methods for Measuring Small Displacements of Television Images
IEEE Transactions on Information Theory, Vol. IT-22, No. 5, septiembre 1976
pp. 573-579

9) Wang, Q. y Clarke, R. J.
Motion estimation and compensation for image sequence coding
Signal Processing: Image Communication, Vol 4, No. 2, abril de 1992
pp. 161-174.

10) Netravali, Arun N. y Limb, John O.
Picture Coding: A review
Proceedings of the IEEE. Vol. 68, No. 3, marzo 1980
pp. 366-406

11) Netravali, Arun N. y Robbins, J. D.
Motion-Compensated Television Coding: Part I
The Bell System Technical Journal. Vol. 58, No. 3, marzo de 1979
pp. 631-671

12) Netravali, Arun N. y Robbins, J. D.
Recursive Motion Compensation: A review
The Bell Laboratories Holmdel, NJ 07733. Vol. F2. 1983
pp. 75-103

13) Bergmann, Hans C.
Analysis of Diferent Displacement Estimation Algorithms for Digital Television Signals
Lehrstuhl für Theoretische Nachrichtentechnik und Informationsverarbeitung, Vol. F2. 1983
pp. 215-234

14) Walker, D. R. y Rao, K. R.
Improved Pel-Recursive Motion Compensation
IEEE Transactions on Communications. Vol. COM-32, No. 10, octubre de 1984
pp. 1128-1134.

15) Walker, D. R. y Rao, K. R.
Motion-Compensated Coder
IEEE Transactions on Communications. Vol. COM-35, No. 11, noviembre de 1987
pp. 1171-1178

16) Baaziz, Nydia
Approches d'estimation et de compensation de mouvement multiresolutions pour le codage de sequences d'images
Tesis Doctotal, Universidad de Rennes I., Francia, 1991