UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES ARAGÓN

COMPRENSIÓN DE SECUENCIAS DE IMÁGENES POR ALGORITMOS DE ACOPLAMIENTO DE BLOQUES

TESIS QUE PARA OBTENER EL GRADO DE:

INGENIERO EN COMPUTACIÓN

PRESENTAN :

GONZÁLEZ FLORES, GRACIELA LETICIA

Y

RODRÍGUEZ PACHECO, JOSÉ ERWIN

1996

TESIS CON
FALLA DE ORIGEN

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

"Compresión de Secuencias de Imágenes por Algoritmos de Acoplamiento de Bloques"

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ARAGÓN

Graciela Leticia González Flores.

José Erwin Rodríguez Pacheco.

Director de Tesis: Dr. Francisco García Ugalde.

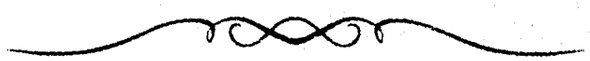
1996

Nunca consideres al estudio como un deber sino como una oportunidad para penetrar en el bello y maravillosos mundo del saber.

El talento gana juegos, pero el trabajo de equipo y la inteligencia gana campeonatos.

A Nuestra Familia:

Con gran admiración por su inmensa comprensión y apoyo para realizar una de las mas grande de nuestras metas, GRACIAS.....



Especialmente al Dr. Francisco García Ugalde:

El mas sincero agradecimiento por el apoyo y confianza que en nosotros deposito.

Y a todas las personas que directa o indirectamente nos ayudaron.....

ÍNDICE

INTRODUCCIÓN.

OBJETIVOS.

Se determinará la compresión de datos para la transmisión de imágenes tomando en cuenta las secuencias de imágenes de televisión mencionando algunas de las características de la compensación de movimiento.

La compresión de datos en imágenes consiste básicamente en minimizar el número de unidades o símbolos de información usados para la representación de una imagen; cada muestra de la imagen llamada pixel, se cuantiza a un número fijo de bits, y entonces la imagen se guarda en algún dispositivo de almacenamiento digital como disco magnético, o bien, se transmite digitalmente.

La utilidad de la compresión de datos consiste en la gran eficiencia para el almacenamiento y transmisión de imágenes, donde el objetivo principal es minimizar la memoria de almacenamiento o bien, minimizar el ancho de banda para la transmisión.

Casi siempre, una imagen comprimida al ser decodificada para la reconstrucción de su forma original, estará acompañada por alguna distorsión. La eficiencia del algoritmo de compresión se mide por medio de su calidad en la compresión de datos (tasa de compresión), la distorsión resultante, además de la complejidad en su implementación (si es rápido o lento). Este último punto, se refiere a la eficiencia teórica del algoritmo (número de cálculos por ciclo) así como la rapidez física del "hardware" sobre el que se implanta dicho algoritmo.

La compensación de movimiento es uno de los componentes esenciales del código de video; en la predicción del área de movimiento una escena puede ser desarrollada por ésta, las técnicas están aplicadas en casi todos los códigos de secuencias de imágenes. Algunas de estas aproximaciones calculan un método conocido como gradiente del espacio temporal, el cual consiste en aproximaciones diferenciales.

Si nos referimos a los esquemas de código de compensación de movimiento, estas ventajas dependen de lo siguiente:

1) La equivalencia del objeto en movimiento se encuentra en escenas reales de televisión.

2) La habilitación de un algoritmo de estimación de movimiento con una buena exactitud de predicción en la intensidad.

3) Los algoritmos de estimación de movimiento menos complicado en amplitud, espacio y resolución temporal de la imagen transmitida.

La compensación de movimiento llega a ser un poderoso recurso para reducir el equivalente a la definición de televisión, esta examina los objetos en movimiento de una secuencia de imágenes de prueba que obtienen los vectores.

La compensación de movimiento usa el conocimiento de objetos en movimiento obtenidos para lograr grandes datos de compensación, un número de técnicas tienen que ser desarrolladas para producir vectores de desplazamiento que pueden ser usadas para sistemas en códigos predictivos.

En una secuencia de imágenes de televisión un objeto en movimiento genera cambios luminosos de cuadro a cuadro. Estos cambios luminosos pueden ser usados para estimar los parámetros de un modelo matemático que describe el movimiento del

objeto, los modelos en movimiento pueden ser usados para perfeccionar la eficiencia de técnicas predictivas.

En las señales de televisión se generan escenas en un tiempo de 1 segundo, igualmente no hacen cambios en escenas de cuadro a cuadro. Estos resultados son considerables redundancias de señales, la existencia en la redundancia se tomo en cuenta y se tuvieron que realizar mejoras para la codificación de cuadro a cuadro resultando un prototipo o implementaciones reales: codificador y decodificador.

Los cuadros de código están basado en lo siguiente:

1) Existen segmentos de cuadros de televisión los cuales están divididos en dos partes, la primera parte es el cuadro anterior, y la otra parte conocida como el área de movimiento que cambio desde el cuadro anterior.

2)Transmisión de dos tipos de información en el área de movimiento:

a) Direccionar específicamente la localidad de los elementos de la imagen en el área de movimiento.

b) Información para que la intensidad de los elementos del área de movimiento puedan ser elevados.

3) Acoplar el código del rango de bits.

a) El rango de información es transmitido para almacenar en un buffer de transmisión.

b) Usa el buffer menos lleno del rango de bits codificado para variar la amplitud, espacio y resolución temporal de la señal de televisión.

La intensidad de los elementos de la imagen en el área de movimiento son transmitidos por códigos predictivos, que envían diferencias de cuadros, diferencia de elementos o diferencia de líneas (o estas combinadas).

CAPITULO 1.

COMPRESIÓN DE SECUENCIAS DE IMÁGENES.

OBJETIVOS.

Conocer algunas de las técnicas mas conocidas de los algoritmos de Pixeles Recursivos y los algoritmos de Acoplamiento de Bloques para poder conocer algunas de las características de cada uno.

Existen dos técnicas de compensación de movimiento: acoplamiento de bloques y pixeles recursivos, donde se usa un código predictivo de compensación de movimiento, estos algoritmos pueden ser clasificados como operaciones lentas y de secuencias posteriores.

La estimación lenta usa la información transmitida anteriormente y es usada por algoritmos de pixeles recursivos donde la información extra tiene que ser transmitida en secuencias posteriores y esta normalmente toma la forma de vectores únicos de bloques producidos por operaciones de acoplamientos de bloques. En códigos de imagen la compensación de movimiento tiene que ser usualmente restringida a movimientos traslacionales debido a la búsqueda compleja y operaciones requeridas en tiempo real.

Los algoritmos recursivos obtienen una estimación de desplazamiento iterativa local, basada en estimaciones previas, las iteraciones deben ser obtenidas para todos los pixeles o para cada bloque de pixeles.

En una escena de televisión que contiene objetos en movimiento y si una estimación de su translación es factible de obtener, entonces, se podrá realizar una predicción mucho más eficiente utilizando elementos en el cuadro previo que estén apropiadamente desplazados. En escenas reales de televisión, el movimiento de los objetos puede ser una complicada combinación de translaciones y rotaciones de estos. Por tanto resultaría muy difícil intentar estimar dicho movimiento, pues resultaría en una gran cantidad de procesamiento. Sin embargo el movimiento translacional es relativamente fácil de estimar, y de hecho, los algoritmos que se utilizan para estimación de movimiento parten de la premisa de que los objetos en la escena sólo se mueven translacionalmente, lo que implica una muy aproximada estimación de movimiento. El éxito de estas técnicas se basa, entonces en la gran cantidad de movimiento translacional que se da en la escena.

La mayoría de los algoritmos para estimación de movimiento en codificación entre cuadros basan su operación en las siguientes suposiciones:

1) Los objetos se mueven translacionalmente en un plano que es paralelo al plano de la cámara. No se consideran efectos de acercamiento ("zoom"), ni de rotación.

2) La iluminación es uniforme temporal y espacialmente.

3) La oclusión de un objeto por otro y el fondo descubierto no se toman en cuenta.

Existen varios métodos para la estimación de movimiento tales como el de Cafforio y Rocca, el de Netravali y Robbins o el Walker y Rao por citar algunos. Estos últimos métodos, se basan en algoritmos recursivos donde se estima el movimiento para cada pixel individualmente.

Un método, el cual no requiere codificar los bordes (fronteras) de los objetos móviles, es llamado estimación de movimiento por *acoplamiento de bloques.* Este método, consiste en dividir el cuadro de la imagen en bloques rectangulares pequeños de tamaño fijo. Para cada bloque, se asume un movimiento de translación lineal, y así, el vector de desplazamiento de cada bloque se codifica.

Para dominar algunos de los problemas de la compresión de imágenes, algunos algoritmos de estimación tienen que ser desarrollados, recientemente los algoritmos fueron clasificados en dos grandes grupos, que son denotados como algoritmos recursivos y algoritmos de acoplamiento de bloques.

A continuación se nombraran algunos algoritmos para estimación de movimientos, ellos intentan minimizar recursivamente una determinada cantidad (función del error de estimación de movimiento).

1.1 Técnicas en algoritmos de pixel recursivo.

Los algoritmos recursivos tienen la habilidad de resolver los problemas en los objetos de movimientos múltiples.

Se presentan algunas técnicas para la estimación de movimiento, estas técnicas intentan minimizar recursivamente la dimensión en la predicción del error de la compensación en movimiento, dando como resultado la estimación de desplazamiento, obteniendo una estimación semejante que genera el error de predicción en compensación de movimiento, de esta resulta una estimación mas baja que la anterior, la minimización recursiva es desarrollada por un gradiente.

Algunas técnicas en los algoritmos recursivos como el de Netravali y Robbins[I], desarrollan un primer algoritmo de estimación de desplazamiento recursivo, que perfeccionó la estimación de precisión. Los algoritmos de estimación recursivos son asumidos con una estimación inicial, que es usada para producir una nueva perfección de la estimación.

Las iteraciones pueden ser ejecutadas cada una por una señal de elementos de imágenes consecutivas alrededor de una línea examinada, ya sea de línea a línea o cuadro a cuadro, estas técnicas deben ser denotadas como algoritmos en estimaciones de pixeles recursivos, con recursión horizontal, vertical o temporal, conociendo una función de desplazamiento diferencial de cuadros, puede ser usada como un criterio para calcular la estimación. Estos proponen un algoritmo de estimación que intenta minimizar los valores cuadráticos en cuadros de desplazamiento diferenciales recursivamente usados en los métodos del gradiente.

Netravali y Robbins, proponen versiones simplificadas para la interpolación y algoritmos de estimación de desplazamiento, este método consiste en investigar el rango de convergencia y la precisión de los algoritmos de desplazamiento, si el número de iteraciones es limitado, para simplificar la descripción de los algoritmos solo los componentes en (x) del vector de desplazamiento son considerados.

Limb y Murphy[II], proponen un algoritmo que mide la velocidad de un objeto en movimiento junto a un fondo fijo, Cafforio y Rocca; dan una pequeña fundamentación teórica para pequeñas estimaciones equivalentes en movimiento e introducen un algoritmo de segmentación con pixeles sujetos a diferentes movimientos basados sobre lo anteriormente logrado. Netravali y Robbins desarrollaron un algoritmo en estimación de movimiento que usa recursivamente cambios relativos luminosos y el gradiente para encontrar un vector de

[I] Bibliografía VI
[II] Bibliografía VIII

desplazamiento para todos y cada uno de los pixeles, el vector de movimiento necesita ser transmitido.

Para un pixel con una cierta intensidad en el área de movimiento de un cuadro común, se trata de encontrar la intensidad correspondiente en el cuadro anterior en una cierta localización desplazada. Para perfeccionar la precisión de la estimación y ganancia sobre el rango medido de desplazamiento, Netravali y Robbins proponen el esquema de estimación recursivo, el de una estimación perfeccionada que es producida desde la última estimación.

Otros desarrollos de los algoritmos de pixeles recursivos (pel-recursivo) fueron desarrollados con la introducción del filtrado de Wiener por Biemond, éste muestra que los resultados en base Wiener, son mejores algoritmos que los de Netravali y Robbins, una iteración es hecha usualmente de pixel al pixel anterior, mientras que las iteraciones múltiples son necesarias en el mas reciente. Estos algoritmos son ligeramente mas complicadas. Moorhead, da un buen análisis de convergencia y ensayo para perfeccionar las técnicas de pixeles recursivos, desde otro ángulo, también llamado Proyección a lo Largo de la Trayectoria en Movimiento (PMAT). Basado sobre la idea que la velocidad de movimiento de cuadro a cuadro es constante.

1.2. Técnicas en algoritmos de acoplamiento de bloques.

La máxima correlación en acoplamiento de bloques, se encuentra entre el bloque del cuadro actual y el bloque del cuadro anterior.

Para reducir el tiempo de estimación del método realizado, la búsqueda del desplazamiento es reestructurada para un número total fijo de posiciones posibles. Una desventaja de los bloques basados en estimación de movimiento es el método de segmentación para objetos en movimiento.

Con los métodos de bloques, se asume que el desplazamiento del objeto es constante dentro de un pequeño bloque **B** bidimensional de pixeles, si el tamaño del bloque es pequeño, entonces, la suposición anterior se hace más válida; sin embargo, la cantidad de operaciones de búsqueda y la transmisión de información del desplazamiento se incrementan.

El desplazamiento, puede ser estimado por técnicas de correlación o acoplamiento. De esta forma, D puede ser escogido tal que minimice alguna medida del error[III] de predicción como:

$$PE(D) = \sum N\left\{u_K(x,y) - u_{K-1}(x-q, y-l)\right\} = \textit{Distorsion}$$

Donde N(.) es una distancia que puede ser la magnitud o la función cuadrada. Una vez que hemos encontrado la mejor D, todos los pixeles que pertenecen al bloque B son predichos.

[III] Bibliografía IV

Si por ejemplo, pensamos que la imagen ha sido dividida en bloques de MxN pixeles, entonces considerando uno de esos bloques de centro en el pixel de coordenadas (i,j) en el cuadro actual (k) y suponiendo que sólo puede haber desplazamientos máximos horizontales y verticales de d_{max} pixeles, entonces, la región en el cuadro anterior donde se debe encontrar el mínimo de la ecuación para PE, debe ser una región de $(M+2d_{max})x(N+2d_{max})$ pixeles centrada en (i,j). Esta sería la región de búsqueda.

Por ejemplo, si el tamaño del bloque es de 9x9 y el máximo desplazamiento posible es d_{max}= l0, la región de búsqueda en el cuadro anterior (k-l) sería un área de 29x29 pixeles. Un método de búsqueda llamado "búsqueda exhaustiva", consiste en evaluar PE (Error de Predicción o distorsión) para cada corrimiento de un solo pixel en las direcciones horizontal y vertical para todas las posiciones del bloque en la región de búsqueda, es decir, si la suma de la ecuación anterior se efectúa sobre todos los pixeles (x,y) del bloque del cuadro actual, y se efectúa dicha suma para un determinado valor de q y l, entonces, si se hace variar q y l, pixel por pixel sobre toda el área de búsqueda hasta encontrar el valor mínimo de esta suma, habremos encontrado el vector de desplazamiento D = (q,l), esto requeriría $(2d_{max} + 1)^2$ evaluaciones de la ecuación PE.

Se han propuesto algunos criterios para especificar la ecuación de PE. Los más comunes son, el del cuadrado (Mínimo Diferencia Cuadrática)[IV] y el del valor absoluto:

$$MSE(i,j) = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}\left[u_k(m,n) - u_{k-1}(m+i,n+j)\right]^2$$

[IV] Bibliografía VII, I

$$MAD(i,j) = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}\left|u_k(m,n) - u_{k-1}(m+i,n+j)\right|$$

donde $-d_{max} \leq i, j \leq +d_{max}$. El criterio MAD (Mínima Diferencia Absoluta) tiene la ventaja de que no se necesitan multiplicaciones ni divisiones.

Además de estos dos criterios de apareamiento, de los cuales ya se ha visto que el segundo (MAD) es el más adecuado, se han propuesto varios métodos para realizar la búsqueda sin tener que evaluar el criterio de apareamiento sobre toda el área de búsqueda.

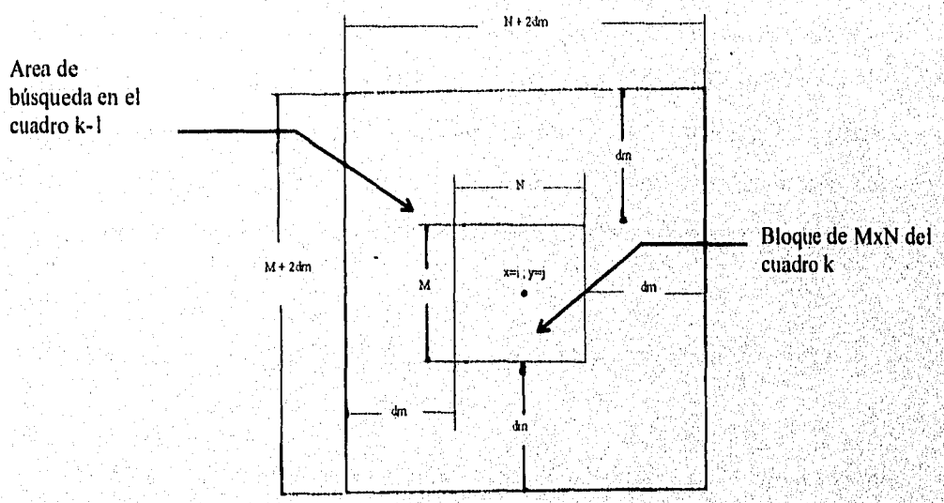


Figura 1. Area de búsqueda de un bloque de MxN en un cuadro k

J. R. Jain y A. K. Jain, sugirieron un método para reducir el número de corrimientos requeridos para encontrar el mejor acoplamiento. La importancia de tales procedimientos de búsqueda se pueden ver si consideramos la situación extrema del caso de un objeto que se mueve a través de la escena de televisión de 512 x 512 pixeles en 1 segundo. Esto corresponde a un desplazamiento de cuadro a cuadro de alrededor de d_{max} = 20. Para cada corrimiento, el criterio MSE (Error Cuadrático Medio) o el MAD (Mínima Diferencia Absoluta) tendría que ser evaluado.

Para poder procesar la imagen mediante el método de acoplamiento de bloques (block matching) una imagen es dividida en pequeños bloques de tamaños fijos.

Las técnicas de acoplamiento de bloques sobre bloques rectangulares producen un vector de desplazamiento para cada bloque.

Se ha encontrado que el desempeño de las técnicas de correlación de área resulta algo pobre para bloques pequeños, en áreas de baja actividad espacial, y para bloques que no experimentan translación pura. La Dirección de Mínima Distorsión (DMD)[V] se obtiene en una posición (i,j) tal que la distorsión entre el bloque del cuadro actual y el bloque del cuadro anterior sea minimizado, esto es:

$$Distorsion = D(i,j) \equiv \sum_{m=1}^{M}\sum_{n=1}^{N}\left\{u(m,n) - u_0(m+i,n+j)\right\}^2 \ debe\ ser\ minima$$

donde $u(m,n)$ y $u_0(m,n)$ son las luminancias de los pixeles correspondientes al cuadro actual y al cuadro anterior.

[V] Bibliografía I, XVII

Una de las aplicaciones donde se encuentra la utilidad en la estimación de movimiento es en una técnica de compresión denominada *salto de cuadro.* Para la estimación de movimiento entre cuadros, la búsqueda se limita usualmente a una ventana de 5x5 pixeles. Una vez que hemos estimado el movimiento, se logra la compresión salto de cuadros de la imagen hasta llegar al siguiente *cuadro anterior.*

Con el desconocimiento de la trayectoria de movimiento de los pixeles, un salto de cuadro, generalmente se reproduce, o bien repitiendo el cuadro anterior, o por interpolación entre los cuadros anterior y siguiente. Ambos métodos tienen serios efectos en la calidad de la reproducción del movimiento.

1.2.1. Algoritmo de búsqueda en tres pasos.

Koga et al.,[VI] desarrollaron un procedimiento de búsqueda de tres pasos. En este método, con la excepción del punto de partida (i,j), ocho puntos de búsqueda se prueban en el primer paso. Estos puntos se encuentran relativamente lejos alrededor del centro x=i, y=j. La siguiente figura 2, ejemplifica un esquema de este algoritmo, en el cual, el punto (i+3, j+3) se encuentra como una primera aproximación utilizando para ello el criterio MAD. En un segundo paso, ocho puntos de búsqueda que están espaciados más cercanamente alrededor de la primera aproximación, se prueban para la mínima distorsión, de forma tal que el punto (i+3,j+5) es encontrado para este ejemplo. Se repite el segundo paso hasta que se alcance la precisión requerida. Para el caso particular de un área de búsqueda con $d_{max} \leq 6$ el tercer paso da el vector de desplazamiento final que en este ejemplo es (i+2,j+6).

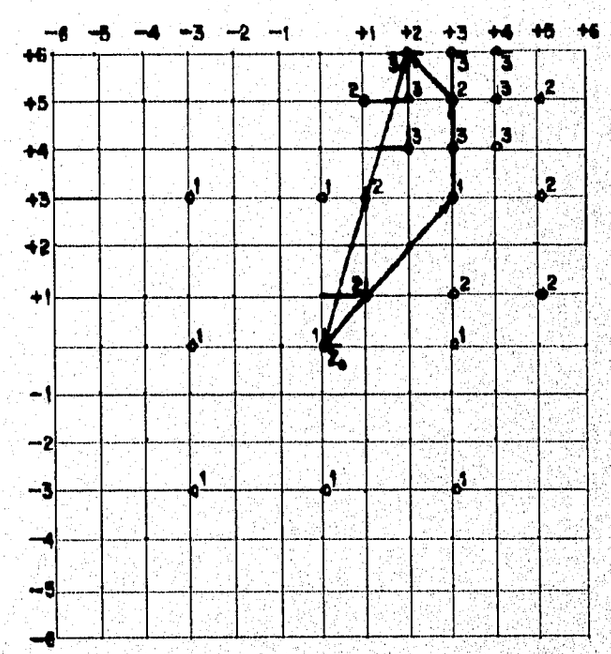


Figura 2. Algoritmo de los tres pasos

[VI] Bibliografía VIII

1.2.2. Algoritmo OTS One at the Time Search (Uno a la vez).

Otro algoritmo de búsqueda llamado búsqueda de la dirección conjugada[VII], el cual modificado después y con el nombre de búsqueda de uno a la vez ofrece mejores resultados que el primero.

Los métodos de búsqueda para optimización, intentan reducir el valor de alguna función objetiva mediante el uso de pruebas estimadas para dar la solución. Se ve una dirección de búsqueda en la cual se espera encontrar el mínimo. Puesto que la dirección obtenida puede no ser correcta, el proceso es iterativo. Después de cada paso de minimización, búsquedas subsecuentes son efectuadas hasta que se satisface un criterio que define el momento en que el mínimo deseado se ha encontrado.

Una técnica simple en una optimización de dos variables, es ajustar una variable a la vez para mínima distorsión. Esto se repite hasta que ya no se pueda alcanzar ninguna mejora.

Tal tipo de ajuste es denominando *búsqueda de uno a la vez* (OTS). Este es un método de búsqueda básico. En un caso de optimización de dos variables, cada variable se ajusta con la otra variable. La dirección de búsqueda en cada paso será paralela a uno de los ejes coordenados. La figura 3, ilustra la OTS para una función convexa en dos variables.

Así pues, este algoritmo busca en la dirección de mínima distorsión la cual se define para este caso como el criterio MAD. En la primera búsqueda, el mínimo en la dirección i es determinado al calcular Dst(i-1,j), Dst(i,j), Dst(i+1,j). Si Dst(i+1,j) resulta ser la menor, entonces, Dst(i+2,j) también es calculada y el valor más pequeño de Dst(i,j), Dst(i+1,j), Dst(i+2,j) es encontrado.

[VII] Bibliografía VII,VI

Si se continúa de esta forma, el mínimo en la dirección i se detecta cuando el valor más pequeño está situado entre dos valores más altos.

Posteriormente en una búsqueda subsiguiente, el mínimo en la dirección j se determina mediante el mismo procedimiento, comenzando en el mínimo de la primera búsqueda.

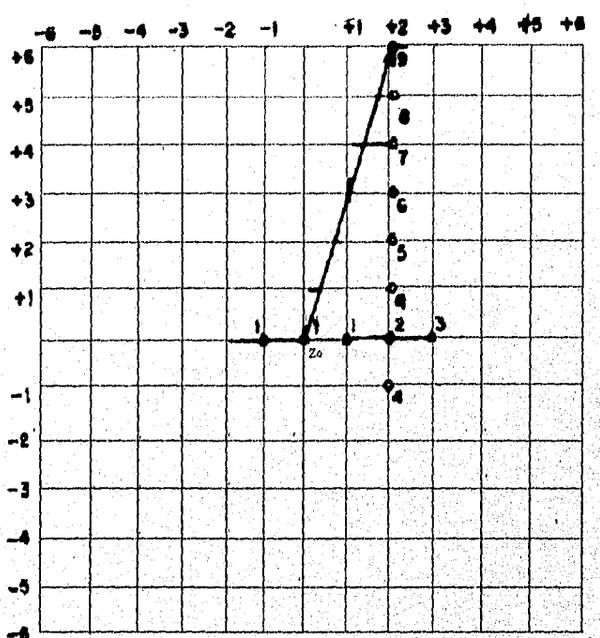


Figura 3. Algoritmo de uno a la vez (OTS One at the Time Search)

1.2.3. Salto de cuadro.

Es uno de los métodos mas simples de compresión de datos, para imágenes en movimiento entre cuadros. Suponiendo que solo los cuadros alternados son saltados, sin conocimiento de las trayectorias en el movimiento de los pixeles, un salto de cuadro es generalmente reproducido cada uno para interpolar entre el cuadro anterior y el siguiente, este método tiene serios efectos sobre la calidad de la reproducción de movimiento.

El salto de cuadros[VIII] es uno de los métodos más simples de compresión de datos para imágenes con movimiento entre cuadros.

Cuando la trayectoria de movimiento de los pixeles es desconocida, un cuadro saltado generalmente se reproduce, o repitiendo el cuadro anterior, o bien por la interpolación entre los cuadros anterior y siguiente (al cuadro presente). Estos dos métodos tiene serios efectos en la calidad de la reproducción del movimiento.

Para simplificar las cosas, podemos suponer que nos saltamos cuadros de forma alternada (uno si, uno no). Saltarse cuadros, es una manera muy popular de compresión de datos incluso cuando no se dispone de medidas de movimiento, tales como la estimación de movimiento a que nos hemos estado refiriendo.

Si nos imaginamos que U_{2k} es un bloque del 2 K-ésimo cuadro, el cual nos saltamos para k= 1, 2, 3....., entonces, en la ausencia de compensación de movimiento, el cuadro reproducido se puede obtener de la siguiente manera:

Repetición de cuadro: $u^{*}_{2k}(m,n) = u_{2k-1}(m,n)$

[VIII] Bibliografía VII,XVII, XIV.

Interpolación de cuadro: $u^{*}_{2k}(m,n) = \frac{1}{2}\left[u_{2k-1}(m,n) + u_{2k+1}(m,n)\right]$

La interpolación entre cuadros reduce las sacudidas o vibraciones que se pueden presentar en los métodos de repetición, pero requiere una memoria de cuadro adicional.

La información de la trayectoria de movimiento se puede utilizar en predicción así como en interpolación. De esta forma, con compensación de movimiento, tenemos que:

Repetición de cuadro: $u^{*}_{2k}(m,n) = u_{2k-1}(m+q, n-l)$

Interpolación de cuadro:

$$u^{*}_{2k}(m,n) = \frac{1}{2}\left[u_{2k-1}(m+q, n+l) + u_{2k+1}\left(m+q', n+l'\right)\right]$$

donde (q,l) y (q',l') son los vectores de desplazamiento de U_{2k} relativos al cuadro anterior y al siguiente, respectivamente.

CAPITULO 2.

ALGORITMOS DE CODIFICACIÓN DE SECUENCIAS DE IMÁGENES POR ACOPLAMIENTO DE BLOQUES.

OBJETIVOS.

Desarrollar los algoritmos mas importantes de acoplamiento de bloques para comprimir una imagen y poder hacer una comparación de estos métodos en cuanto a tiempo y distorsión.

2.1. Descripción general de los métodos conocidos de estimación de movimiento por acoplamiento de bloques.

La Estimación de Movimiento (ME)[IX], forma parte del proceso de medidas de bloques en movimiento de imágenes de datos y entre imágenes sucesivas. Algunas técnicas existen para hacer estas medidas, incluyendo los métodos de pixeles recusivos y acoplamiento de bloques.

Con objeto de reducir la complejidad de búsqueda para procesamiento en tiempo real, se utiliza la técnica de acoplamiento de bloques, en la cual una imagen es dividida en bloques de tamaño fijo con la suposición de que todos los pixeles en un bloque siguen el mismo desplazamiento de un cuadro al siguiente. Así pues, mostremos el fundamento teórico de esta técnica.

[IX] Bibliografía IX

Bajo las suposiciones planteadas con anterioridad, de que si un pixel se mueve de un cuadro a otro su intensidad permanece sin alteración, tenemos que las intensidades monocromáticas $u_k(m,n)$ y $u_{k-1}(m,n)$ de dos cuadros consecutivos se relacionan por: $u_k(x,y) = u_{k-1}(x-q,y-l)$

Donde k es el cuadro actual y k-1 el cuadro anterior, **D** = (q,l), este es el vector de translación bidimensional del objeto durante el intervalo de tiempo, (k-1,k) y (x,y) es el vector bidimensional de la posición espacial. El problema es entonces, estimar **D** de las intensidades del cuadro presente y el cuadro anterior. Para esto último se desarrollaron los métodos de acoplamiento de bloques.

Los métodos de acoplamiento de bloques en estimación de movimiento implican varios bloques de búsqueda, comparando el bloque de la imagen de datos del cuadro anterior con una serie de bloques del cuadro actual. La posición de los bloques dentro de la ventana de búsqueda varían con respecto al método de búsqueda que se este evaluando.

Los bloques de los cuadros anteriores deben ser seleccionados para comparar las diferentes tecnicas de búsquedas, ya sean exhaustivas o iterativas.

Con el objetivo de superar esta dificultad, se han investigado varios métodos para simplificar el procedimiento de búsqueda.

Se menciona que las técnicas más conocidas son 3:

1.- Búsqueda exhaustiva.

2.- Búsqueda logarítmica bidimensional (2-D).

3.- Búsqueda en dirección conjugada.

2.2. Algoritmo de Búsqueda Exhaustiva.

Una búsqueda exhaustiva de acoplamiento de bloques implica todas los bloques posibles del área de búsqueda con respecto al bloque actual, una de las desventajas de esta búsqueda con respecto a las demás se debe al tiempo de búsqueda, este método es demasiado lento comparado con cualquiera de los diferentes algoritmos existentes. Tomando en cuenta lo anterior, en este método exhaustivo, el área de búsqueda es en toda la imagen actual, y de allí el nombre del método.

1) El bloque caracterizado por su pixel (i,j) del centro del bloque de búsqueda se compara siempre con el principio de la imagen actual.

1a) Si M(0,0) < T, donde T es el umbral, concluye la búsqueda; si no ir al paso 2.

2) Se incrementa en una posición el bloque actual, es decir (i+1,j), se vuelve a comparar con este bloque, si es menor termina la búsqueda, sino ir al paso 3.

3) Se incrementa en la siguiente posición el bloque de búsqueda, (i+2,j), nuevamente se compara, si es menor concluye la búsqueda, si no ir al paso 4.

4) Es similar al paso anterior, es decir (i+3,j), siempre se va a ir incrementado en una posición el bloque de búsqueda, y lo mismo se hace con el eje j, hasta recorrer toda la extensión de la imagen actual y poder así obtener el vector de desplazamiento[X].

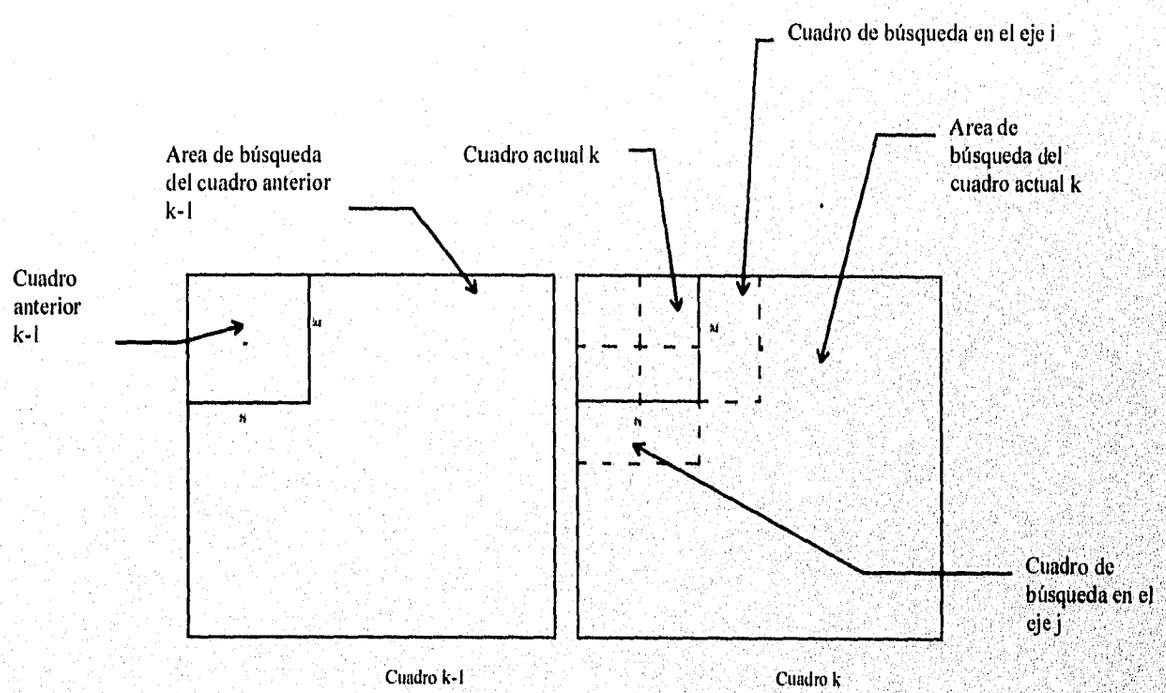


Figura 4. Búsqueda exhaustiva.

[X] Bibliografía VIII, XX.

2.3. Algoritmo de búsqueda logarítmica bidimensional (2-D).

El procedimiento de búsqueda logarítmica bidimensional, se basa en la suposición del criterio de apareamiento MSE[I] = Distorsión, esta incrementa monotonicamente a medida que la búsqueda se aleja de la dirección de mínima distorsión (mínimo PE). La dirección de mínima distorsión es definida por (i,j), tal que Dst(i,j)=MSE(i,j) sea mínima. Este método rastrea la dirección de mínima distorsión.

En cada paso, cinco puntos de búsqueda son evaluados, como muestra la figura 5. La distancia entre los puntos de búsqueda debe ser reducida si el mínimo está en el centro de los puntos de búsqueda o en la frontera del área de búsqueda. En el ejemplo de la figura, se requieren cinco pasos para encontrar el vector de desplazamiento en el punto (i+2,j+6).

La gráfica muestra sólo las posiciones posibles del pixel superior izquierdo del bloque dentro del área de movimiento de éste. Recordemos que el área de búsqueda está formada por $(2d_{max}+1)x(2d_{max}+1)$ puntos, pues equivale al número de corrimientos de un pixel que puede hacer el bloque dentro del perímetro rectangular definido por el desplazamiento máximo del bloque que es igual a d_{max} en las direcciones **x** *y* **y**. Este rectángulo, por tanto, define el área de movimiento del bloque, y su tamaño es de $(M+2d_{max})x(N+2d_{max})$[XI].

1) El bloque caracterizado por el pixel (i,j) de la esquina superior izquierda en el bloque de búsqueda se compara con el bloque correspondiente del cuadro anterior.

[I] Capitulo 1
[XI] Bibliografía VIII, XX

1a) Si M(0,0) < T, donde T es el umbral, entonces el bloque se denota como un bloque estacionario o sin cambio, y concluye la búsqueda; si no, se evalúan los cuatro siguientes puntos (i+2,j), (i,j+2), (i-2,j) y (i,j-2), se busca el mínimo de estos puntos, y se sigue al paso 2.

2) Suponiendo que el mínimo estuvo en el punto (i,j+2), se toma a este como origen y se obtienen las siguientes posiciones de búsqueda, que para el ejemplo son (i+2,j+2), (i,j+4), (i-2,j+2) y (i,j) este último ya evaluado en el paso anterior, volviendo a calcular el mínimo para estos puntos; en caso de que alguno de estos puntos sea menor que T, la búsqueda termina, de otra forma ir al paso 3.

3) Para el ejemplo el mínimo se encontró en el punto (i,j+4), las siguientes búsquedas se hacen en los puntos (i+2,j+4), (i,j+6), (i-2,j+4) y (i,j+2), evaluado en el punto anterior, se procede de la misma manera que el paso 2 , si no, ir al paso 4.

4) Tomando en cuenta que ahora el supuesto origen es (i+2,j+4) y se evalúan los siguientes puntos (i+4,j+4), (i+2,j+6), (i,j+4) y (i+2,j+2), estos dos últimos puntos ya evaluados en los pasos anteriores, se prosigue de la misma forma que el paso 2, de lo contrario ir al paso 5.

5) Para nuestro ejemplo, el mínimo se encontró en el punto (i+2,j+6), tomando en cuenta que en este momento estamos en el limite del área de búsqueda, se hace una última búsqueda, obteniendo cinco puntos, los mas cercanos al supuesto punto de

origen, que son los siguientes (i+3,j+6), (i+1,j+6), (i+1,j+5), (i+2,j+5) y (i+3,j+5), resultando de esta forma las coordenadas del vector de desplazamiento.

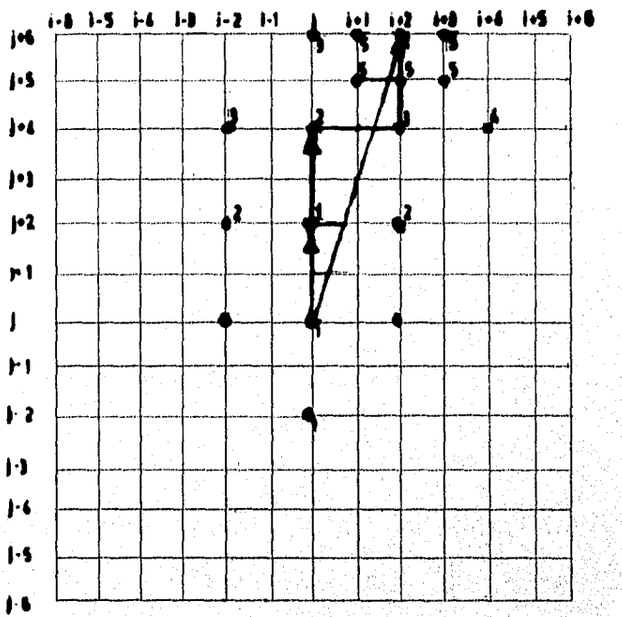


Figura 5. Algoritmo logarítmico 2-D.

2.4. Algoritmo de búsqueda en la dirección conjugada.

A continuación se describe una modificación al algoritmo de Jain y Jain de búsqueda logarítmica en 2-D, la cual fue propuesta por S. Kappagantula y K. Rao[XII], y utiliza la medida MAD[2] . Este algoritmo se puede describir de la manera siguiente:

l) El bloque, caracterizado por su píxel (i,j) de la esquina superior izquierda se compara con el bloque correspondiente en el cuadro anterior.

1a) Si M(0,0) < T, donde T es un umbral (256 niveles), entonces el bloque se clasifica como un bloque sin cambio o estacionario y la búsqueda finaliza allí. De otro modo se sigue al paso 2a. Aquí M(0,0) es el MAD entre los bloques correspondientes (sin mover la posición de un cuadro con respecto al otro) de cuadros adyacentes.

2a) Las siguientes posiciones de pixeles investigadas son (i-4,j), (i,j+4), (i+4,j), y (i,j-4). Si el mínimo de estas búsquedas es > M(0,0), ir al paso 3. Si esto no es verdad y un mínimo es encontrado para las búsquedas en el paso 2a, entonces, este valor es comparado contra T. Si el valor es < T, la búsqueda finaliza allí. De otro modo ir al paso 2b.

2b) Suponga que, del paso anterior, la posición que tenía la mínima disparidad estaba en (i,j+4). Las siguientes posiciones de búsqueda son (i+4,j+4) y (i+4,j-4), respectivamente. La prueba para un mínimo y el umbral se realizan de nuevo. Si la prueba es verdadera, la búsqueda se detiene. De otra forma ir al paso 3a.

3a) Este es similar al paso 2a, excepto que el espacio de las posiciones de búsqueda se reduce ahora a la mitad, es decir, a este punto se le considera como un

---

[XII] Bibliografía XVI
[2] Capitulo I

nuevo origen (que en realidad seria el punto (i+4,j+4)), tomando en cuenta esto, las siguientes posiciones de pixeles investigadas son (i-2,j), (i,j+2), (i+2,j), y (i,j-2). Si el mínimo de estas búsquedas es > M(0,0), ir al paso 4a. Si esto no es verdad y un mínimo es encontrado para las búsquedas en el paso 3a, entonces, este valor es comparado contra T. Si el valor es < T, la búsqueda finaliza allí. De otro modo ir al paso 3b.

3b) Suponiendo que, del paso anterior, la posición que tenía la mínima disparidad estaba en (i+2,j). Las siguientes posiciones de búsqueda son (i+2,j+2) y (i+2,j-2), respectivamente. La prueba para un mínimo y el umbral se realizan de nuevo. Si la prueba es verdadera, la búsqueda se detiene. De otra forma ir al paso 4a.

4a) El espacio de búsqueda se reduce nuevamente a la mitad, de otro modo ir al paso 4b.

4b) Este es el último paso posible de la búsqueda.

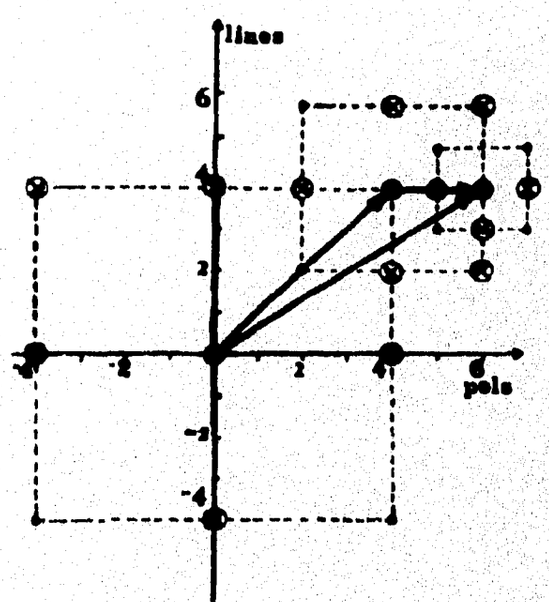


Figura 6. Algoritmo de búsqueda en la dirección

CAPITULO 3.

SIMULACIONES Y RESULTADOS.

OBJETIVOS.

Observar las diferencias de cada imagen dependiendo del tamaño del bloque y el algoritmo empleado, determinando que método seria el mas eficiente para la compresión de datos.

Se realizarán comparaciones de los tres algoritmos de búsqueda que son nombrados como: algoritmo de búsqueda exhaustiva, algoritmo de búsqueda logarítmica 2D y algoritmo de búsqueda en la dirección conjugada [3].

Como se ha mencionado, cada búsqueda en una imagen se compara con respecto a la trama anterior, en este caso se tienen 6 tramas sucesivas llamadas "*Interview*", en las cuales se debe determinar la búsqueda mas eficiente y con mayor exactitud con respecto a la imagen original, observando las diferencias de las imágenes y el error de predicción, que depende del área de búsqueda y del tamaño del bloque, que será variado en cada uno de los algoritmos y en cada una de las diferentes tramas.

Determinamos la imagen "*Interview*" de un tamaño original de 256x256, en esta solo se escoge una pequeña sección de la imagen, ya que si se realiza la búsqueda

[3] Ver Capitulo 2

en el tamaño original de la imagen, el tiempo de búsqueda seria demasiado alto a diferencia de las tramas que se presentan y los resultados serian los mismos.

La siguiente imagen es la primera trama o la trama de partida, de las 6 tramas sucesivas de *"Interview"*.

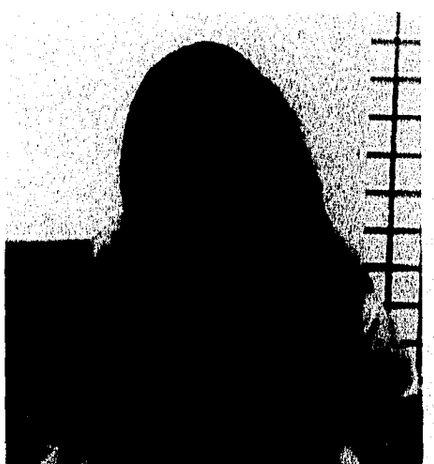


Imagen "Interview" trama de patida.

En las siguientes páginas observaremos las diferentes imágenes en diferentes tamaños de bloque de 16x16, 8x8 y 4x4 pixeles, para identificar, cual de las imágenes y los métodos es el mas adecuado.

Se determinarán los resultados y las gráficas con respecto a lo que observaremos en la diferentes imágenes en su correspondiente algorítmo.

3.1. Simulación de la trama 2 comparada con la trama 1.

3.1.1. Imágenes de bloque 4x4 pixeles.

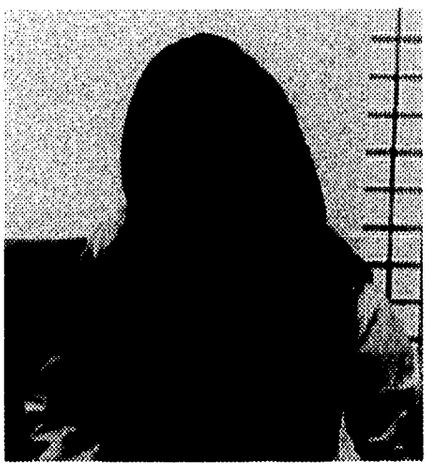


Imagen "Interview" trama 2.

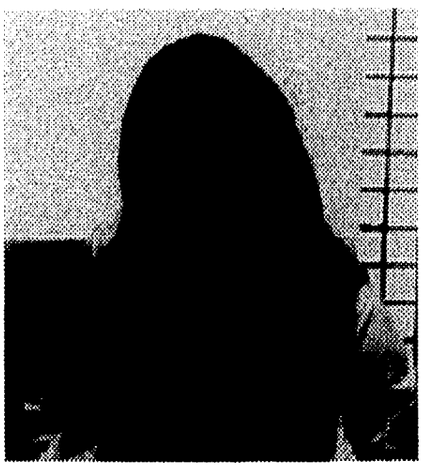


Imagen de búsqueda exhaustiva

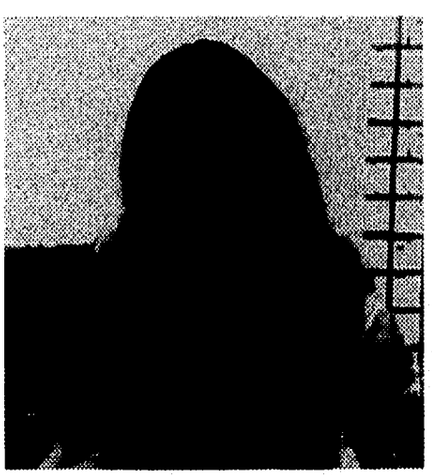


Imagen de búsqueda logarítmica 2D

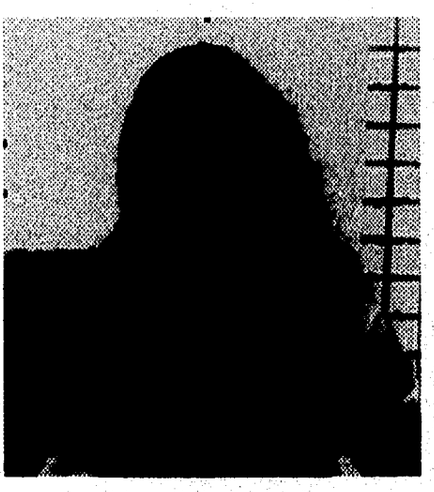


Imagen de búsqueda en la dirección conjugada

3.1. Simulación de la trama 2 comparada con la trama 1.

3.1.1. Imágenes de bloque 4x4 pixeles.

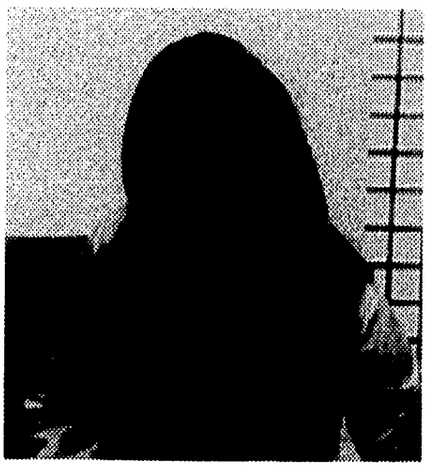


Imagen "Interview" trama 2.

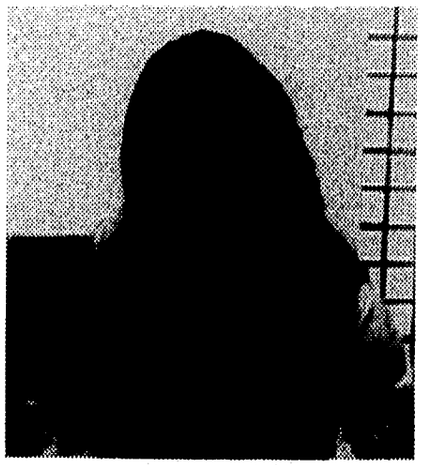


Imagen de búsqueda exhaustiva

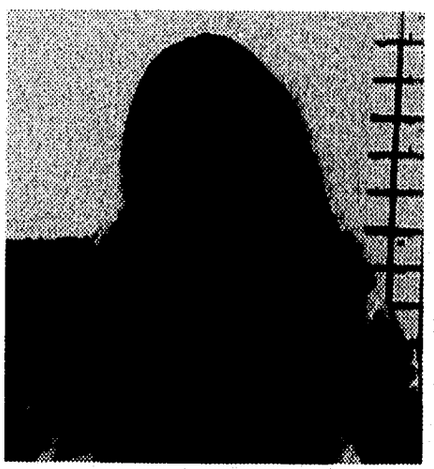


Imagen de búsqueda logaritmica 2D

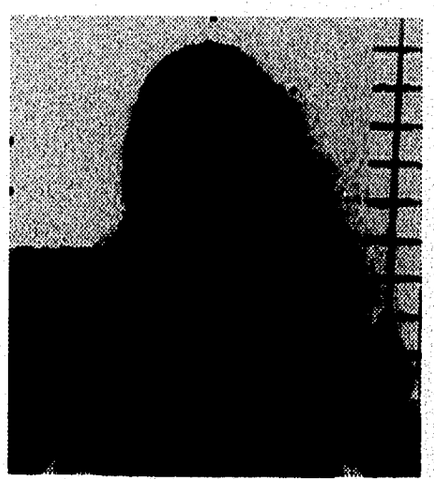


Imagen de búsqueda en la dirección conjugada

3.1.2. Imágenes de bloque 8x8 pixeles.

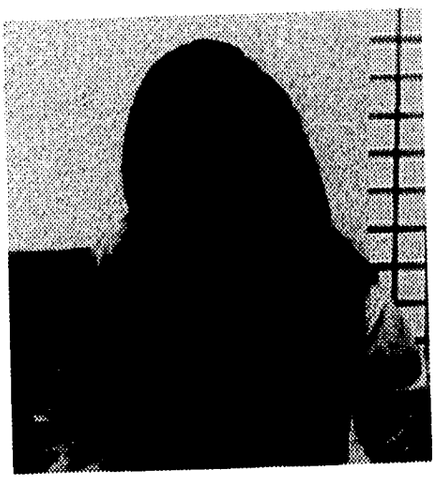


Imagen "Interview" trama 2

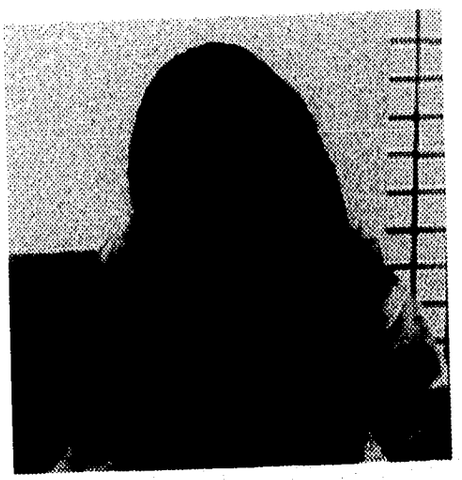


Imagen de búsqueda exhaustiva

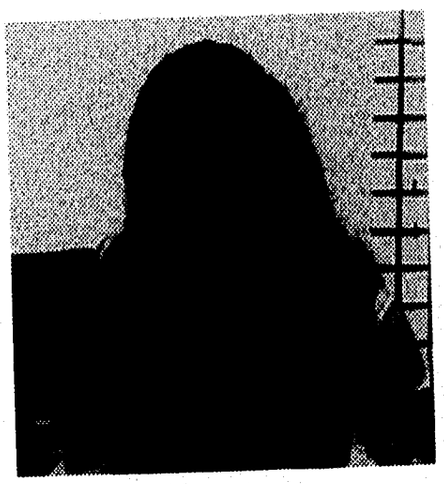


Imagen de búsqueda logarítmica 2D

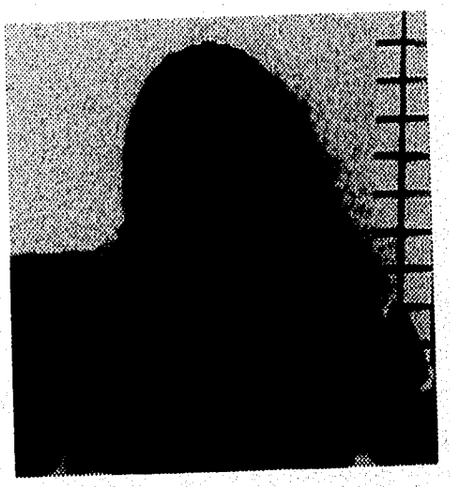


Imagen de búsqueda en la dirección conjugada

3.1.3. Imágenes de bloque 16x16 pixeles.

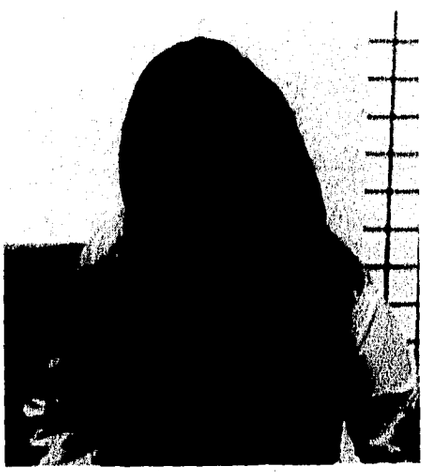


Imagen "Interview" trama 2.

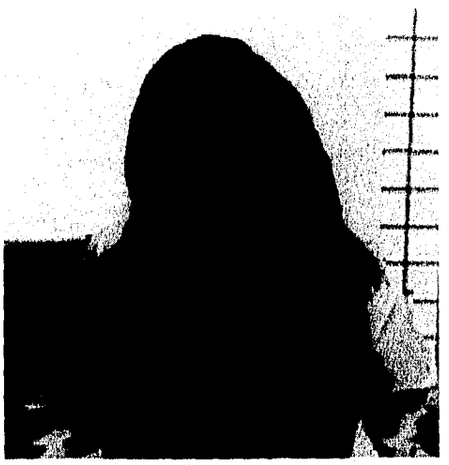


Imagen de búsqueda exhaustiva.

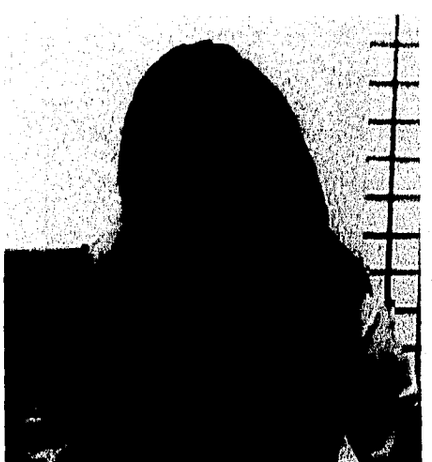


Imagen de búsqueda logarítmica 2D.

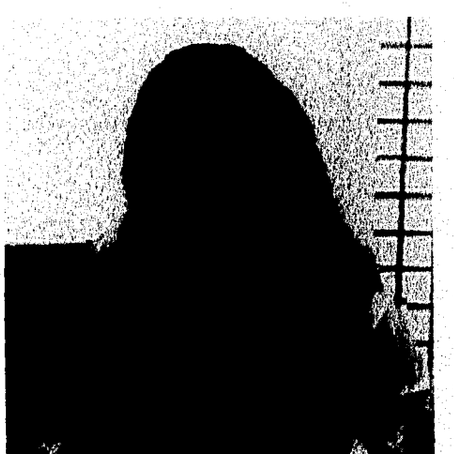
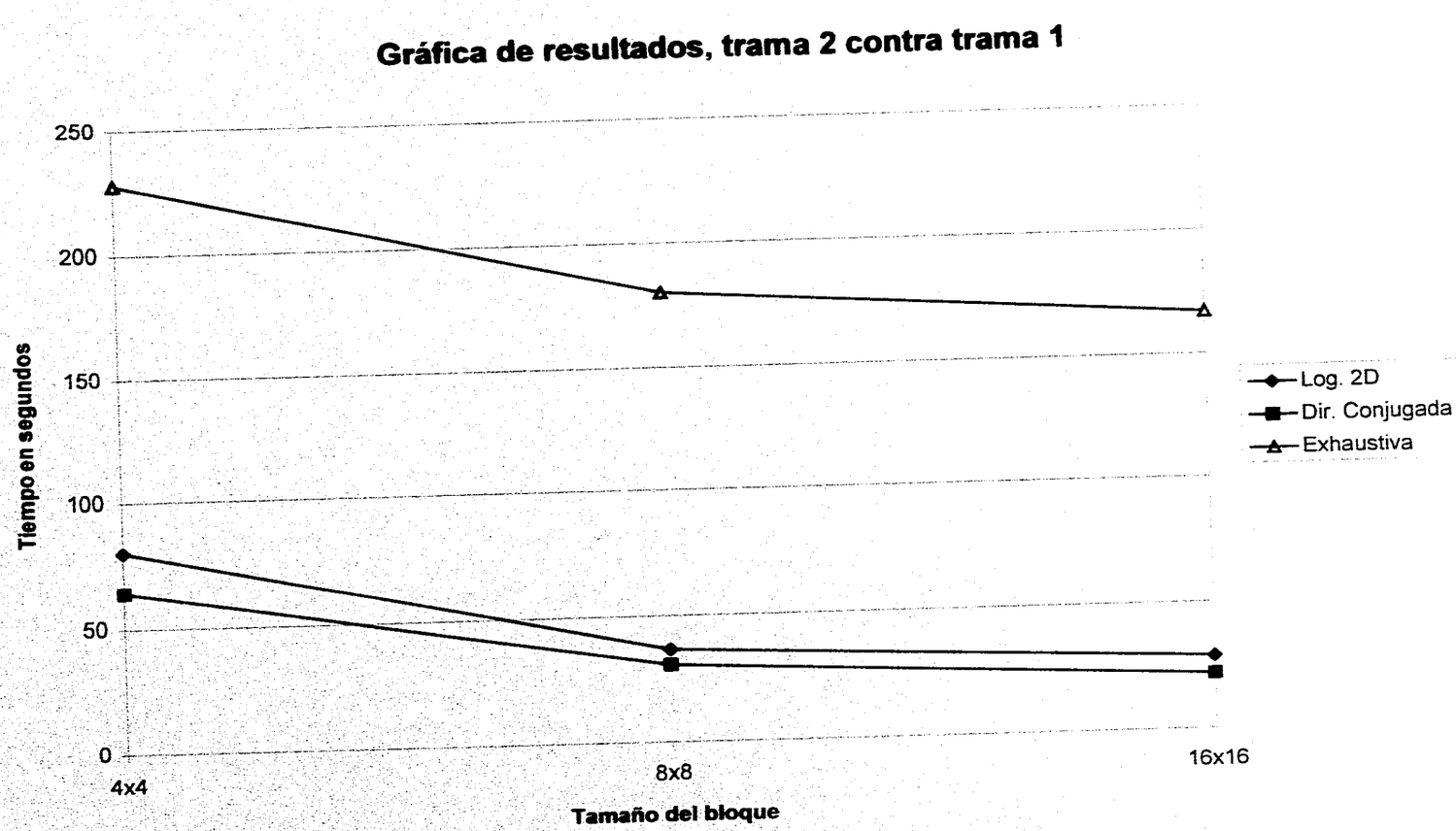


Imagen de búsqueda en la dirección conjugada.

Tabla 1. Tiempos de procesamiento en la trama 2 comparada con la trama 1.

ALGORITMO	TAMAÑO DE BLOQUE	AREA DE BUSQUEDA	TIEMPO Segundos
Logarítmico 2D	4x4	8	80
Dirección conjugada	4x4	8	64.4
Exhaustivo	4x4	--	208.8
Logarítmico 2D	8x8	8	37
Dirección conjugada	8x8	8	31
Exhaustivo	8x8	--	180
Logarítmico 2D	16x16	8	29
Dirección conjugada	16x16	8	22
Exhaustivo	16x16	--	167.4


Gráfica de resultados, trama 2 contra trama 1
250
200
150
100
50
0
Tiempo en segundos
4x4
8x8
16x16
Tamaño del bloque
Log. 2D
Dir. Conjugada
Exhaustiva

3.2. Simulación de la trama 3 comparada con la trama 2.

3.2.1. Imágenes de bloque 4x4 pixeles.

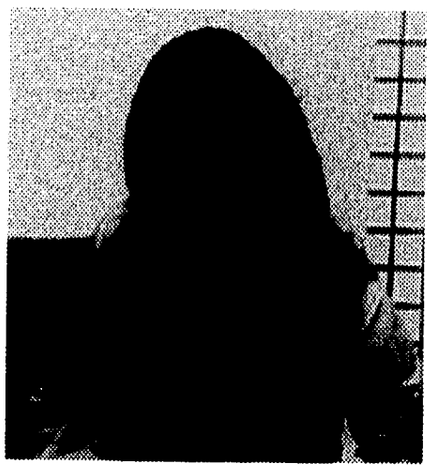


Imagen "Interview" trama 3.

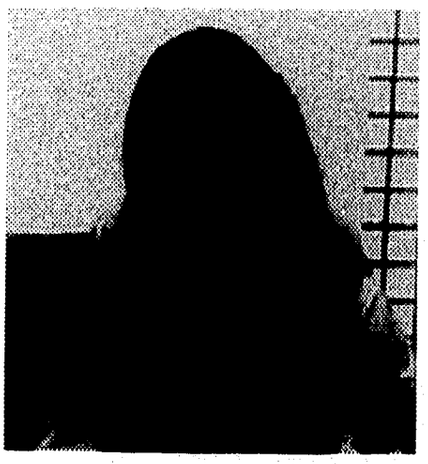


Imagen de búsqueda exhaustiva

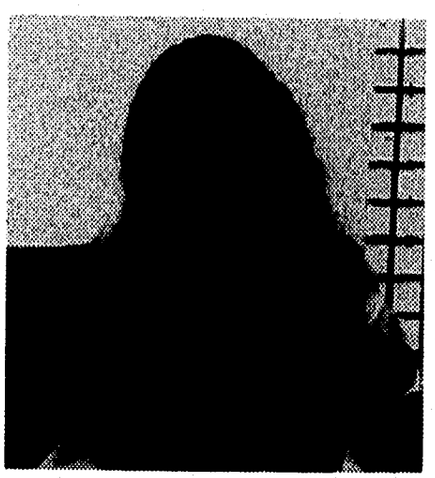


Imagen de búsqueda logarítmica 2D

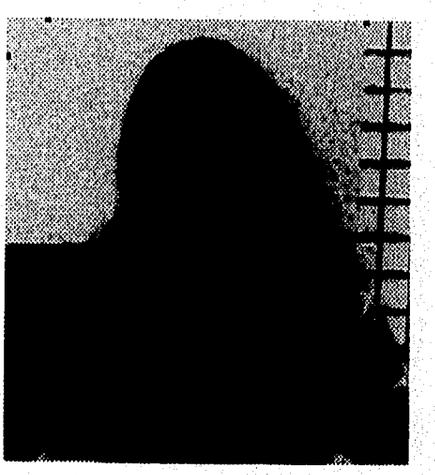


Imagen de búsqueda en la dirección conjugada

3.2.2. Imágenes de bloque 8x8 pixeles.

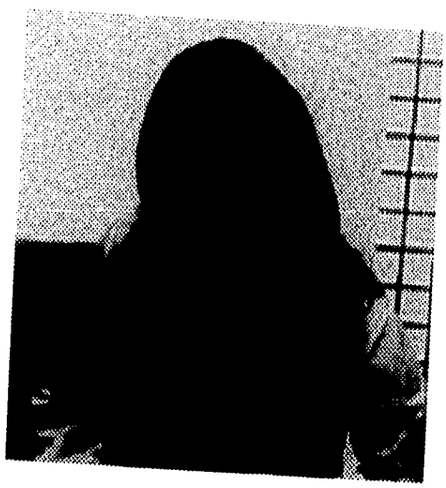


Imagen "Interview" trama 3

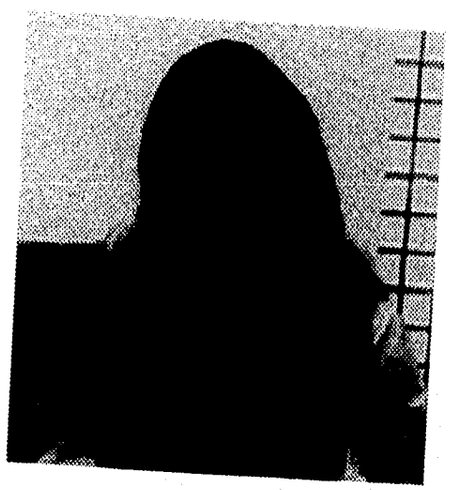


Imagen de búsqueda exhaustiva

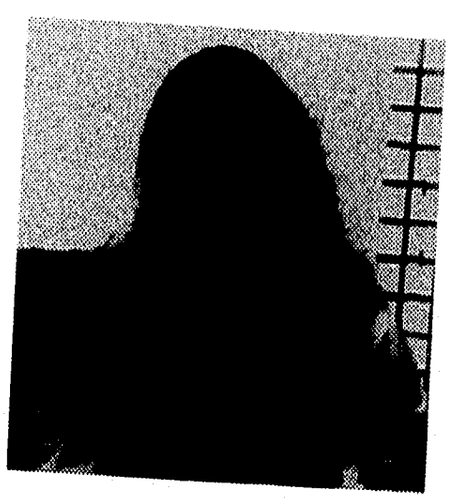


Imagen de búsqueda logaritmica 2D

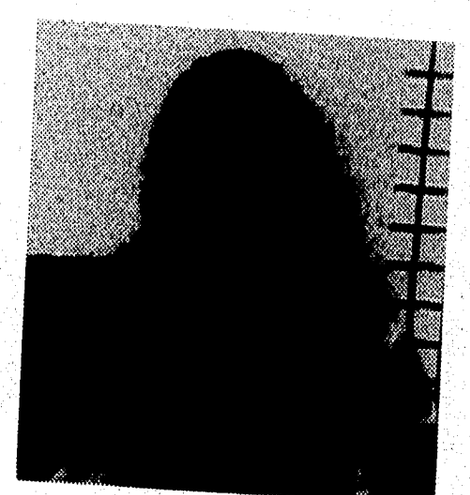


Imagen de búsqueda en la dirección conjugada

3.2.3. Imágenes de bloque 16x16 pixeles.

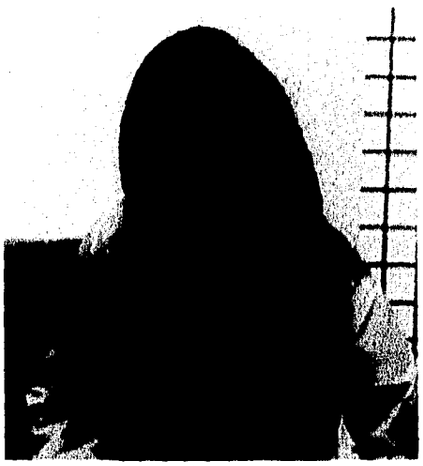


Imagen "Interview" trama 3

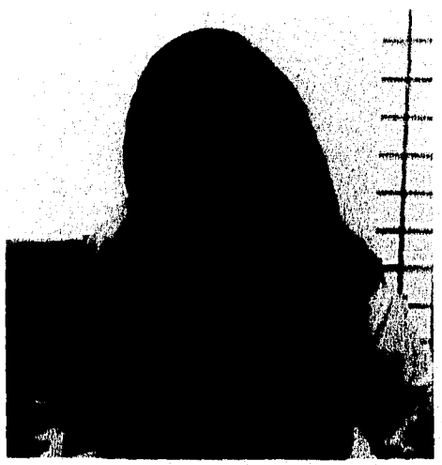


Imagen de búsqueda exhaustiva

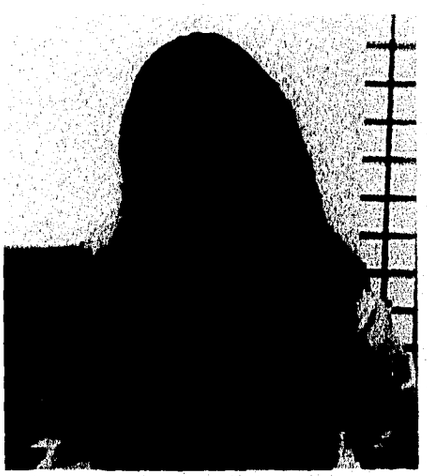


Imagen de búsqueda logarítmica 2D

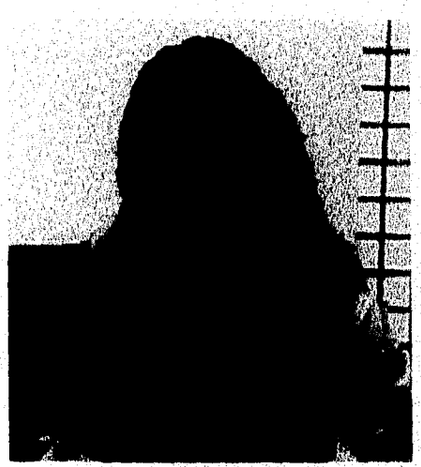
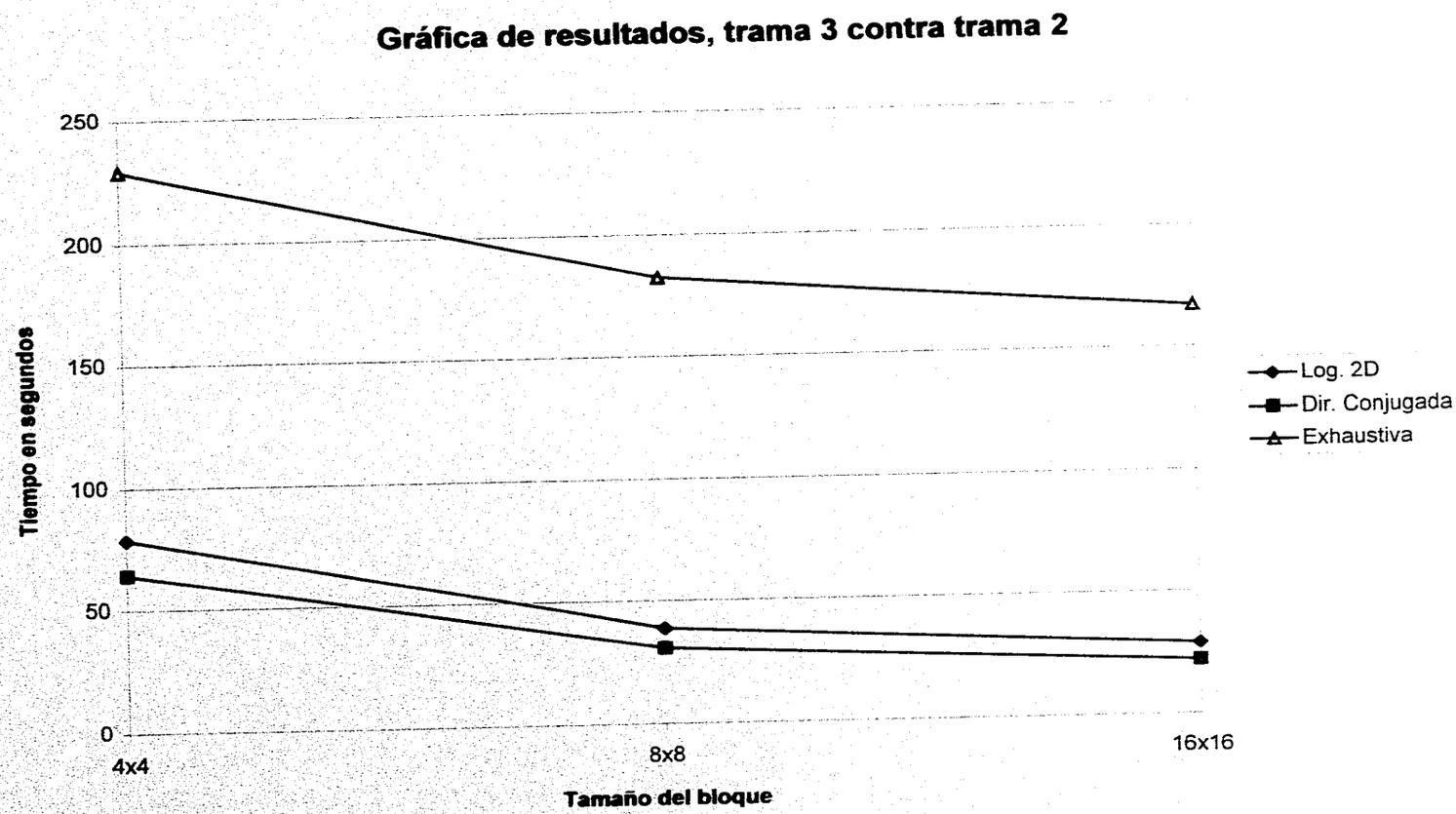


Imagen de búsqueda en la dirección conjugada

Tabla 2. Tiempos de procesamiento en la trama 3 comparada con la trama 2.

ALGORITMO	TAMAÑO DE BLOQUE	AREA DE BUSQUEDA	TIEMPO Segundos
Logarítmico 2D	4x4	8	78.8
Dirección conjugada	4x4	8	64.4
Exhaustivo	4x4	--	228.9
Logarítmico 2D	8x8	8	39
Dirección conjugada	8x8	8	31
Exhaustivo	8x8	--	182.2
Logarítmico 2D	16x16	8	29
Dirección conjugada	16x16	8	22
Exhaustivo	16x16	--	167.4


Gráfica de resultados, trama 3 contra trama 2
250
200
150
100
50
0
Tiempo en segundos
4x4
8x8
16x16
Tamaño del bloque
Log. 2D
Dir. Conjugada
Exhaustiva

3.3. Simulación de la trama 4 comparada con la trama 3.

3.3.1. Imágenes de bloque 4x4 pixeles.

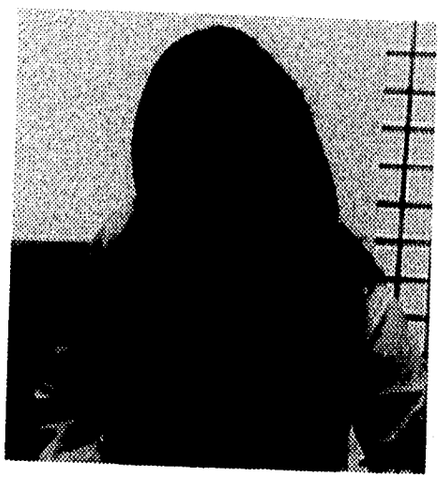


Imagen "Interview" trama 4.

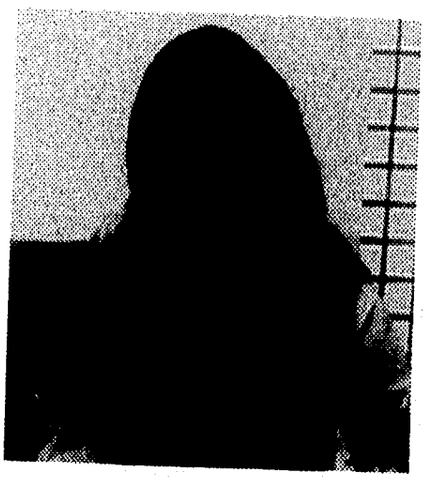


Imagen de búsqueda exhaustiva

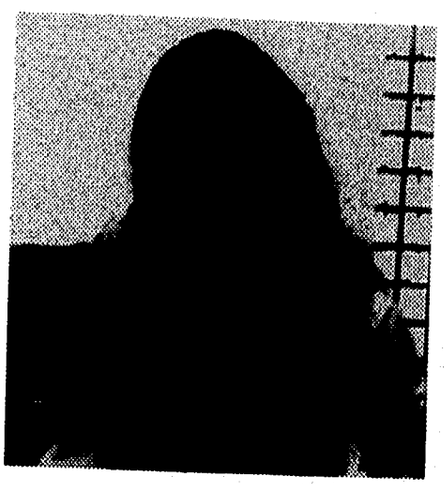


Imagen de búsqueda logarítmica 2D

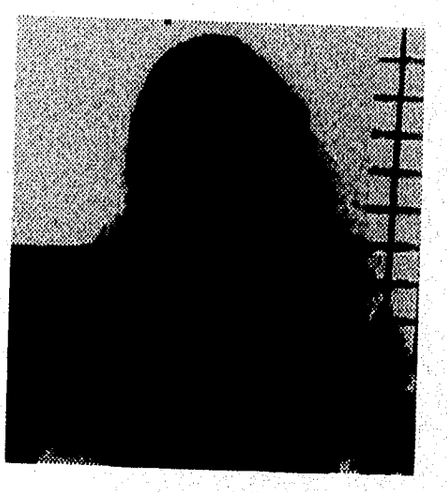


Imagen de búsqueda en la dirección conjugada

3.3.2. Imágenes de bloque 8x8 pixeles.

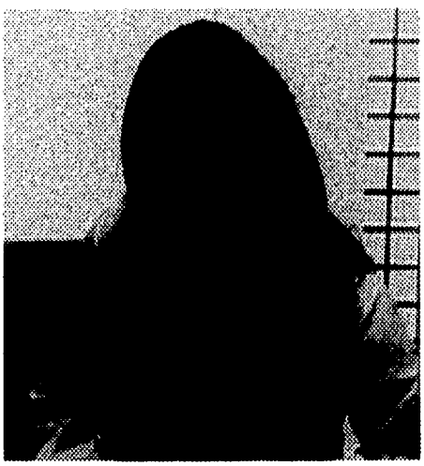


Imagen "Interview" trama 4

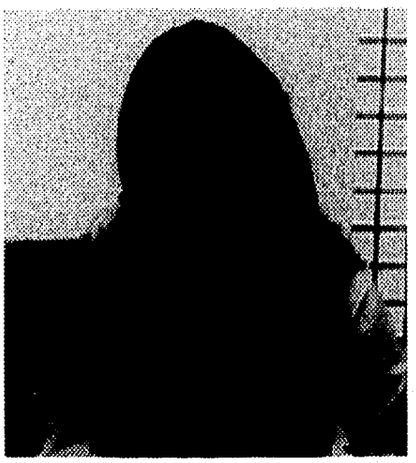


Imagen de búsqueda exhaustiva

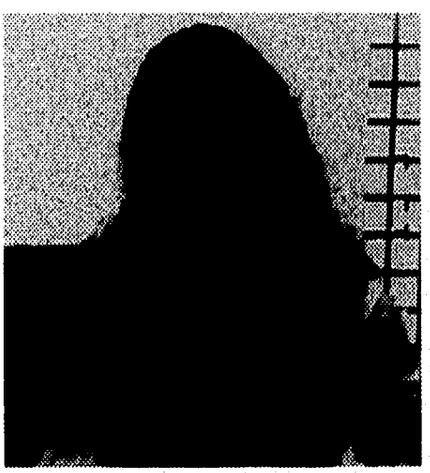


Imagen de búsqueda logarítmica 2D

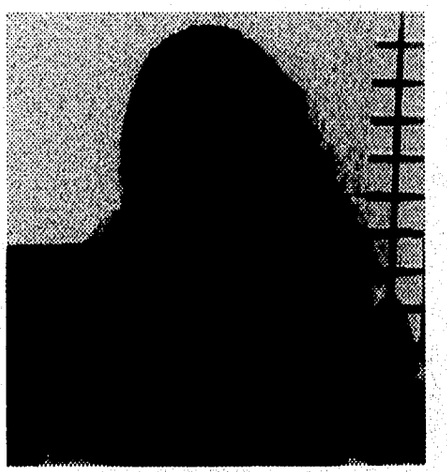


Imagen de búsqueda en la dirección conjugada

3.3.3. Imágenes de bloque 16x16 pixeles.

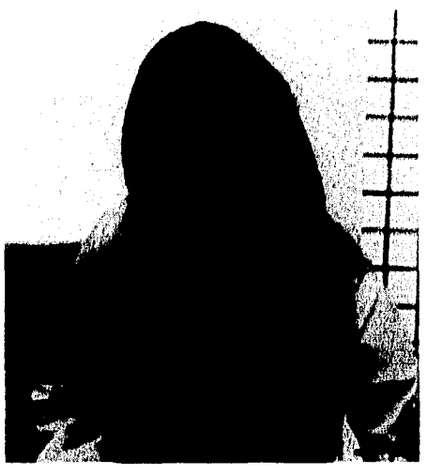


Imagen "Interview" trama 4

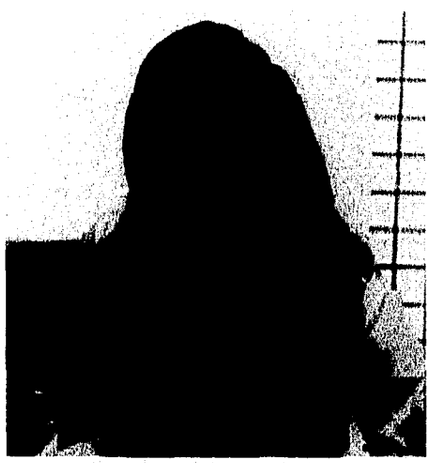


Imagen de búsqueda exhaustiva

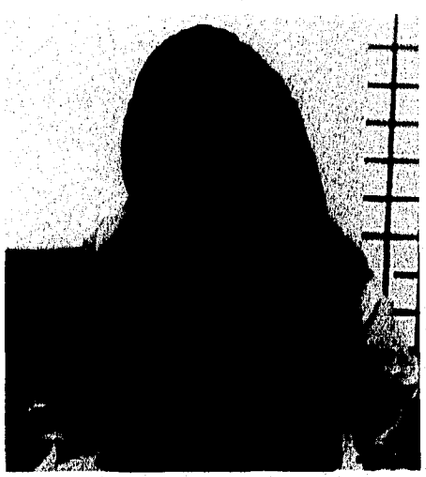


Imagen de búsqueda logarítmica 2D

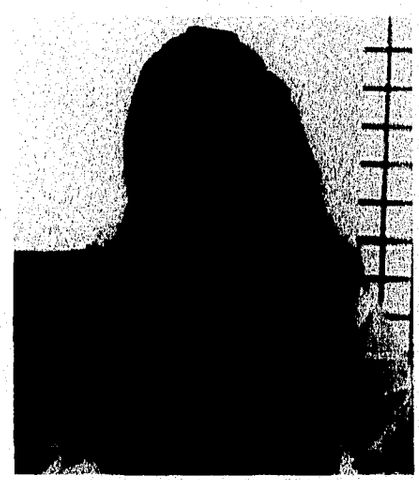
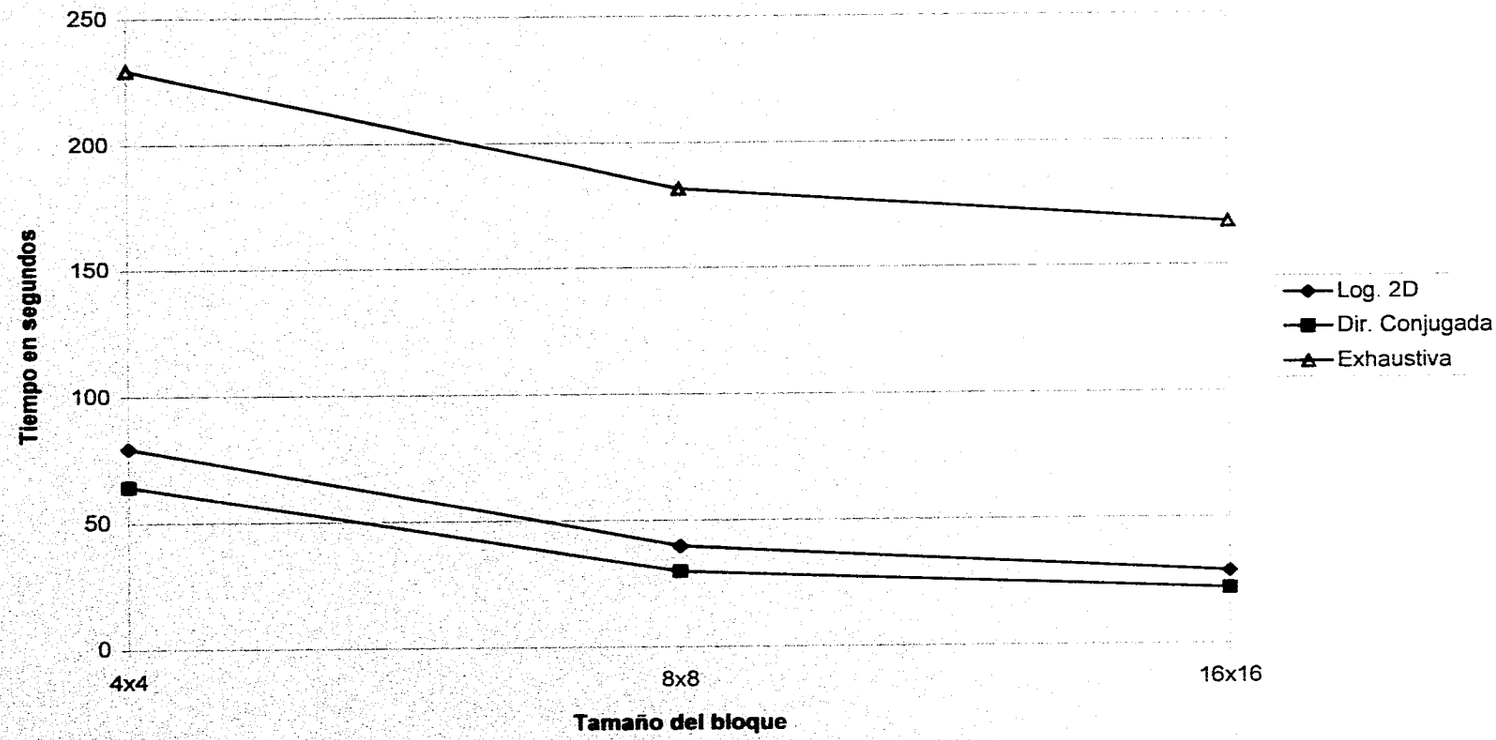


Imagen de búsqueda en la dirección conjugada

Tabla 3. Tiempos de procesamiento en la trama 4 comparada con la trama 3.

ALGORITMO	TAMAÑO DE BLOQUE	AREA DE BUSQUEDA	TIEMPO Segundos
Logarítmico 2D	4x4	8	79.4
Dirección conjugada	4x4	8	64.4
Exhaustivo	4x4	--	209.4
Logarítmico 2D	8x8	8	40
Dirección conjugada	8x8	8	30
Exhaustivo	8x8	--	181.6
Logarítmico 2D	16x16	8	29
Dirección conjugada	16x16	8	22
Exhaustivo	16x16	--	167.2

Gráfica de resultados, trama 4 contra trama 3



3.4. Simulación de trama 5 comparada con la trama 4.

3.4.1. Imágenes de bloque 4x4 pixeles.

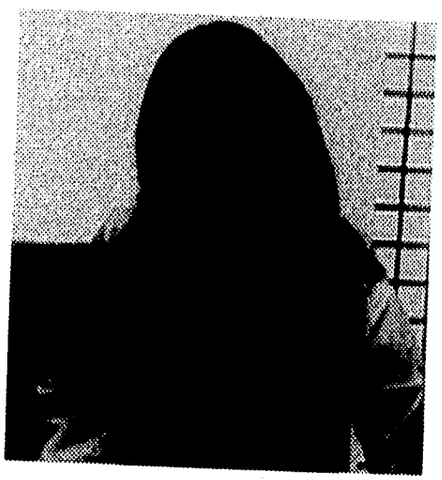


Imagen "Interview" trama 5.

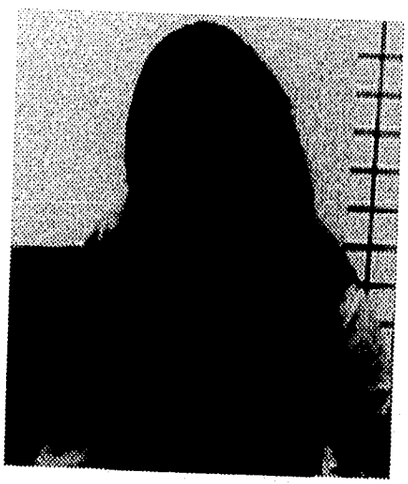


Imagen de búsqueda exhaustiva

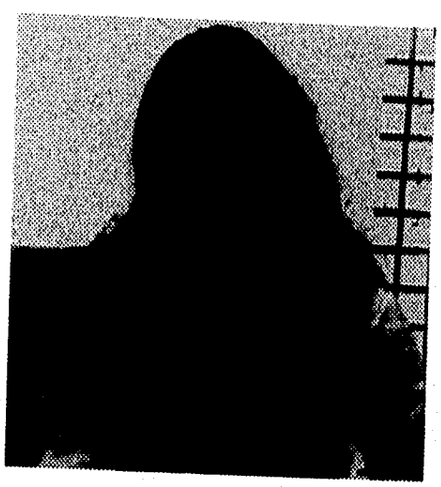


Imagen de búsqueda logarítmica 2D

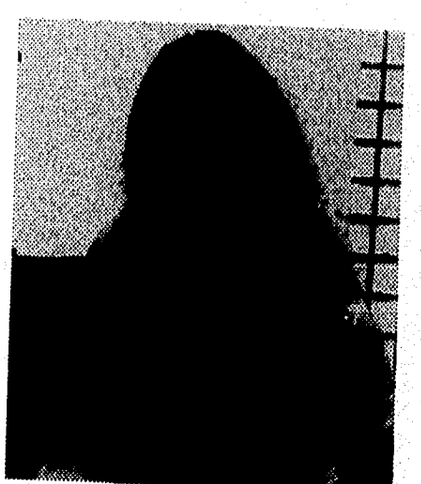


Imagen de búsqueda en la dirección conjugada

3.4.2. Imágenes de bloque 8x8 pixeles.

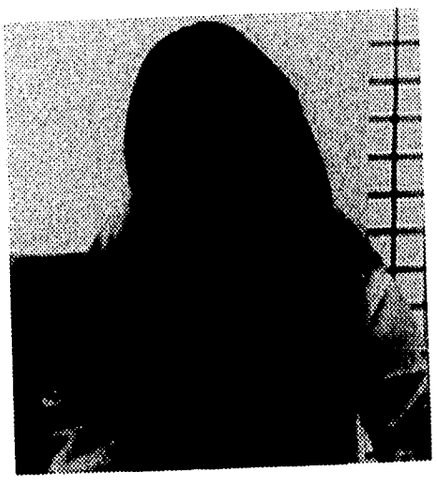


Imagen "Interview" trama 5

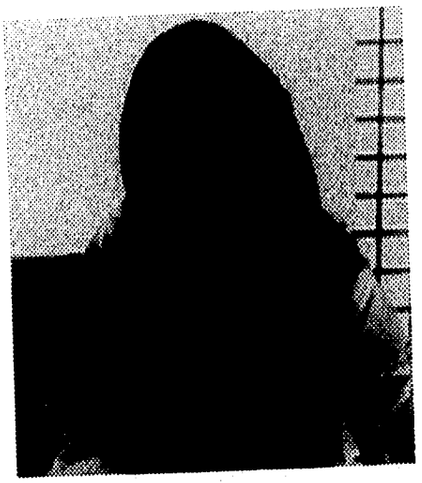


Imagen de búsqueda exhaustiva

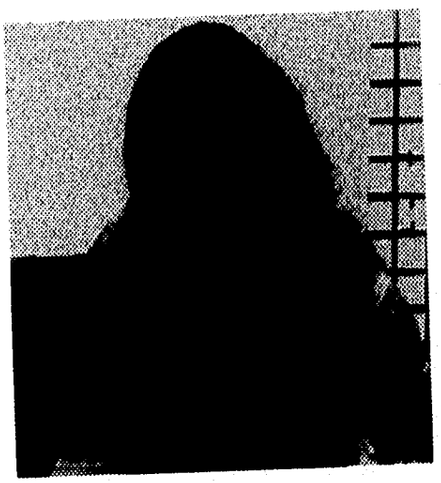


Imagen de búsqueda logarítmica 2D

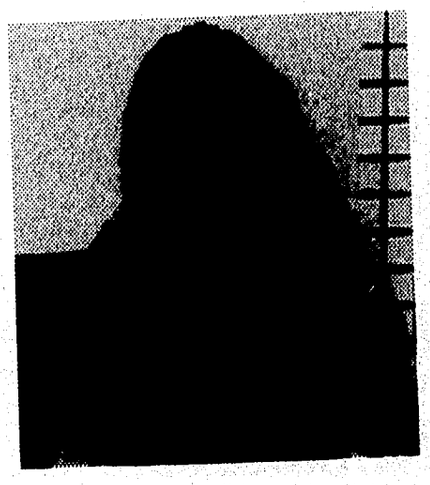


Imagen de búsqueda en la dirección conjugada

3.4.3. Imágenes de bloque 16x16 pixeles.

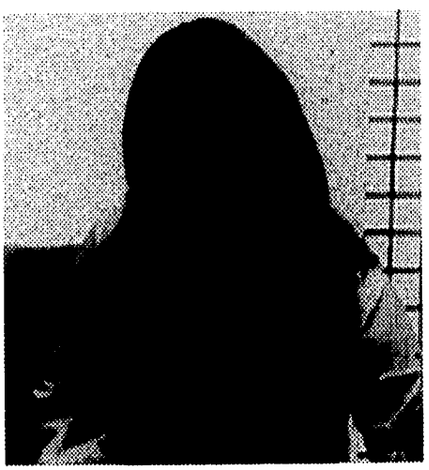


Imagen "Interview" trama 5

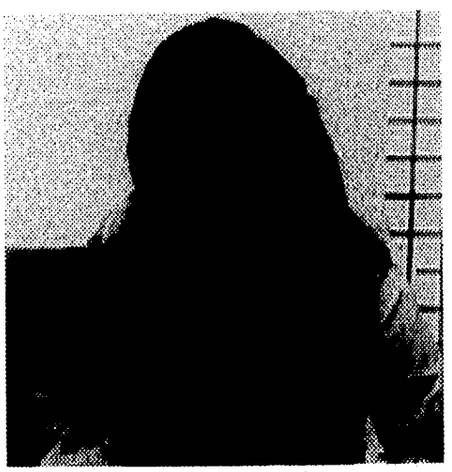


Imagen de búsqueda exhaustiva

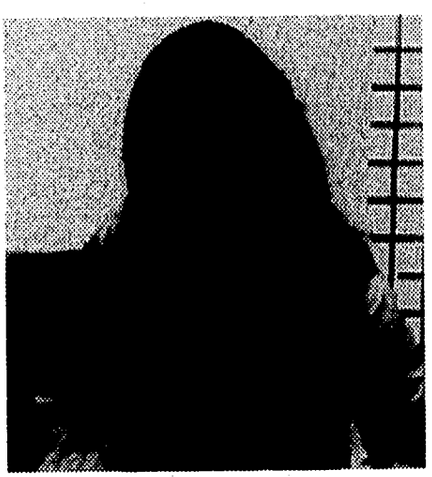


Imagen de búsqueda logarítmica 2D

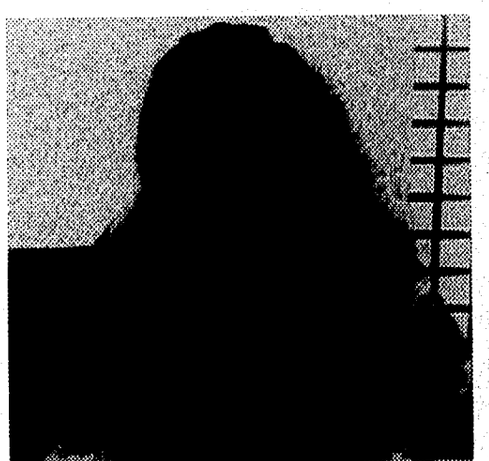
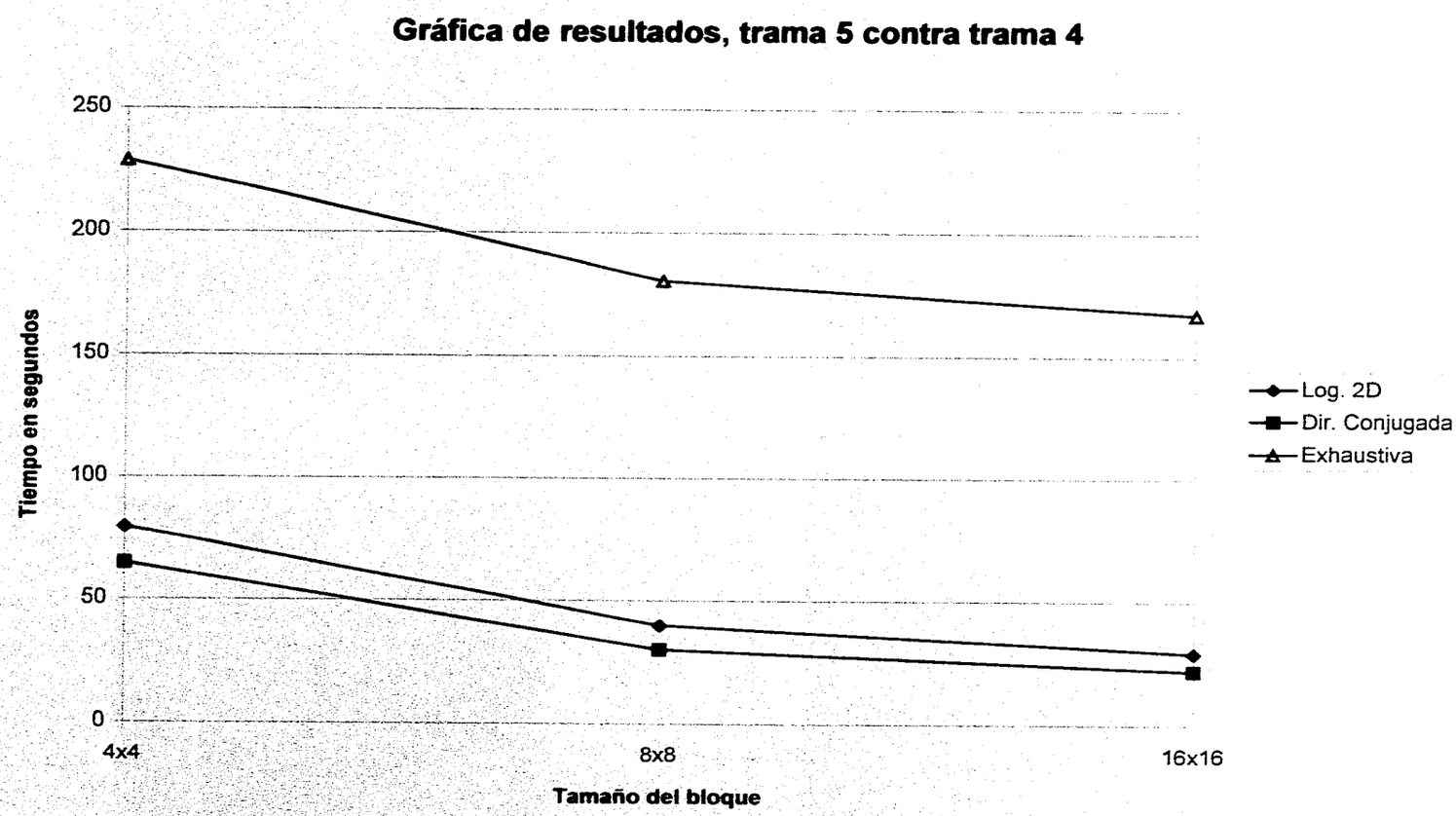


Imagen de búsqueda en la dirección conjugada

Tabla 4. Tiempos de procesamiento de la trama 5 comparada con la trama 4.

ALGORITMO	TAMAÑO DE BLOQUE	AREA DE BUSQUEDA	TIEMPO Segundos
Logarítmico 2D	4x4	8	79.4
Dirección conjugada	4x4	8	65
Exhaustivo	4x4	--	228.8
Logarítmico 2D	8x8	8	41
Dirección conjugada	8x8	8	31
Exhaustivo	8x8	--	181
Logarítmico 2D	16x16	8	29
Dirección conjugada	16x16	8	22
Exhaustivo	16x16	--	167.2


Gráfica de resultados, trama 5 contra trama 4
250
200
150
100
50
0
Tiempo en segundos
4x4
8x8
16x16
Tamaño del bloque
Log. 2D
Dir. Conjugada
Exhaustiva

3.5. Simulación de la trama 6 comparada con la trama 5.

3.5.1. Imágenes de bloque 4x4 pixeles.

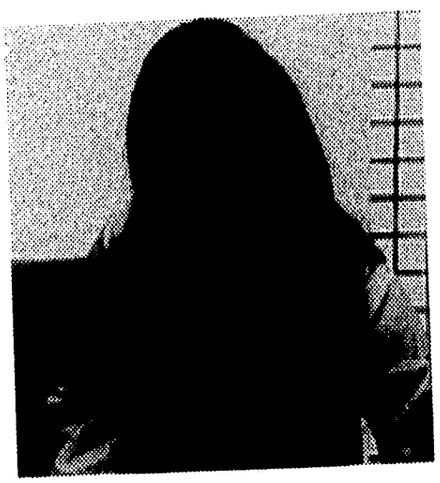


Imagen "Interview" trama 6.

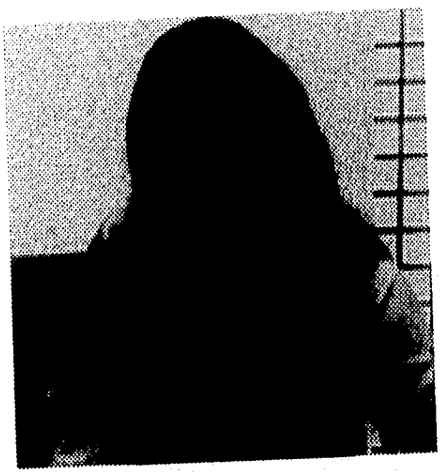


Imagen de búsqueda exhaustiva

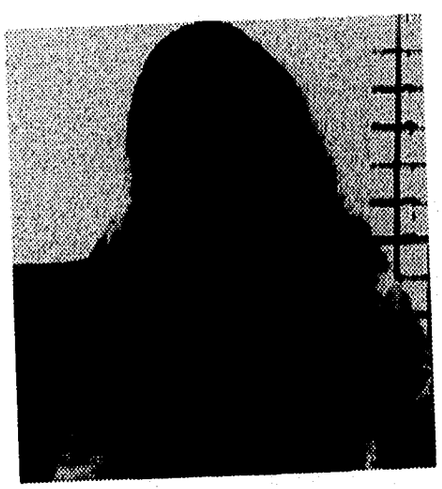


Imagen de búsqueda logarítmica 2D

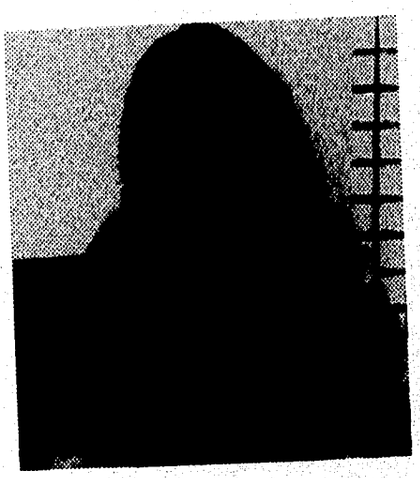


Imagen de búsqueda en la dirección conjugada

3.5.2. Imágenes de bloque 8x8 pixeles.

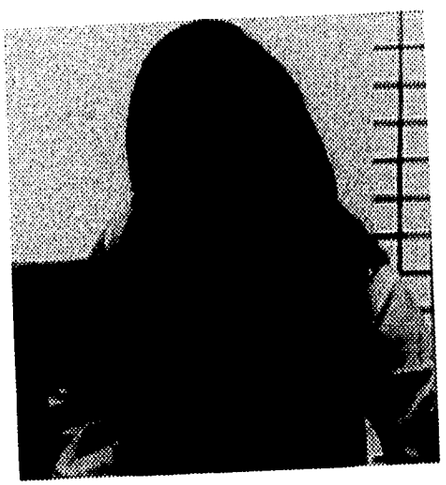


Imagen "Interview" trama 6

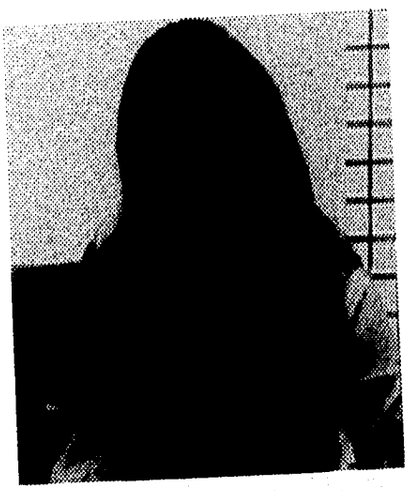


Imagen de búsqueda exhaustiva

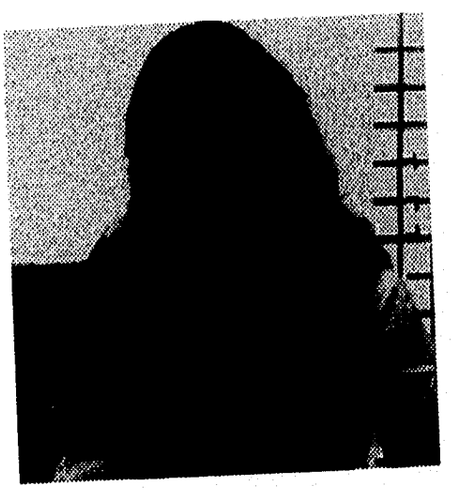


Imagen de búsqueda logarítmica 2D

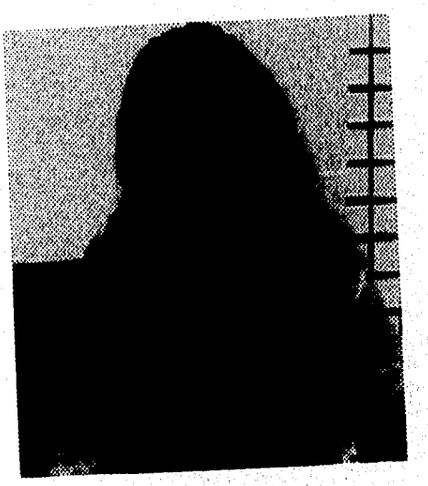


Imagen de búsqueda en la dirección conjugada

3.5.3. Imágenes de bloque 16x16 pixeles.

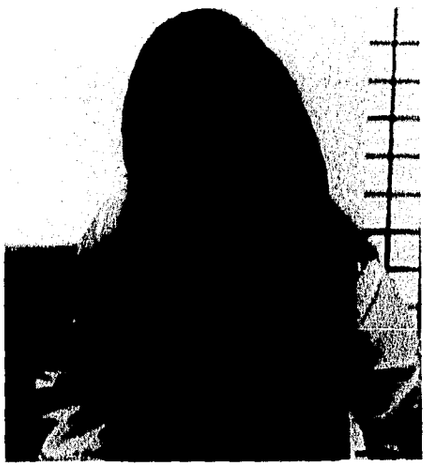


Imagen "Interview" trama 6

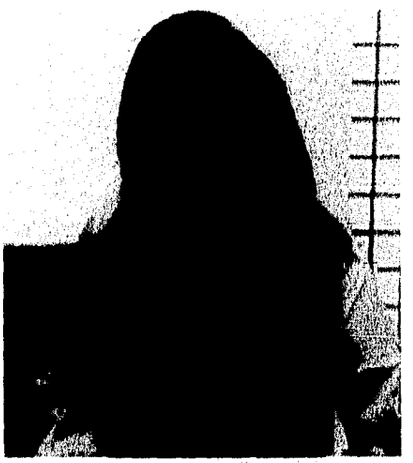


Imagen de búsqueda exhaustiva

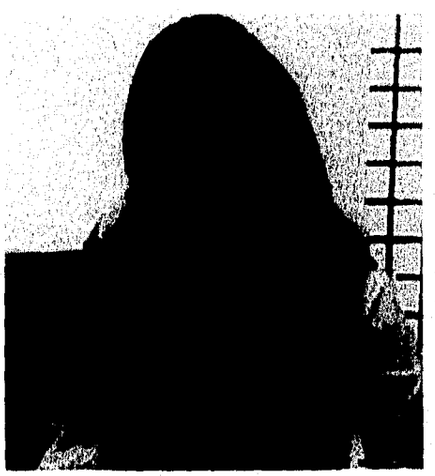


Imagen de búsqueda logarítmica 2D

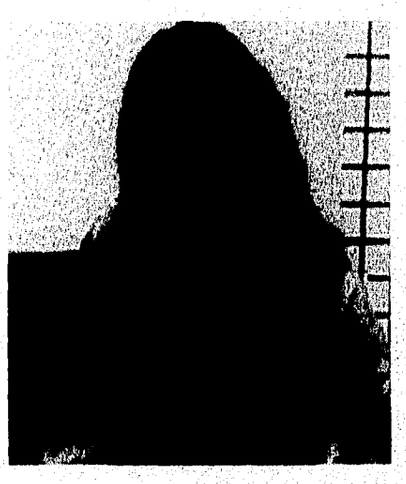
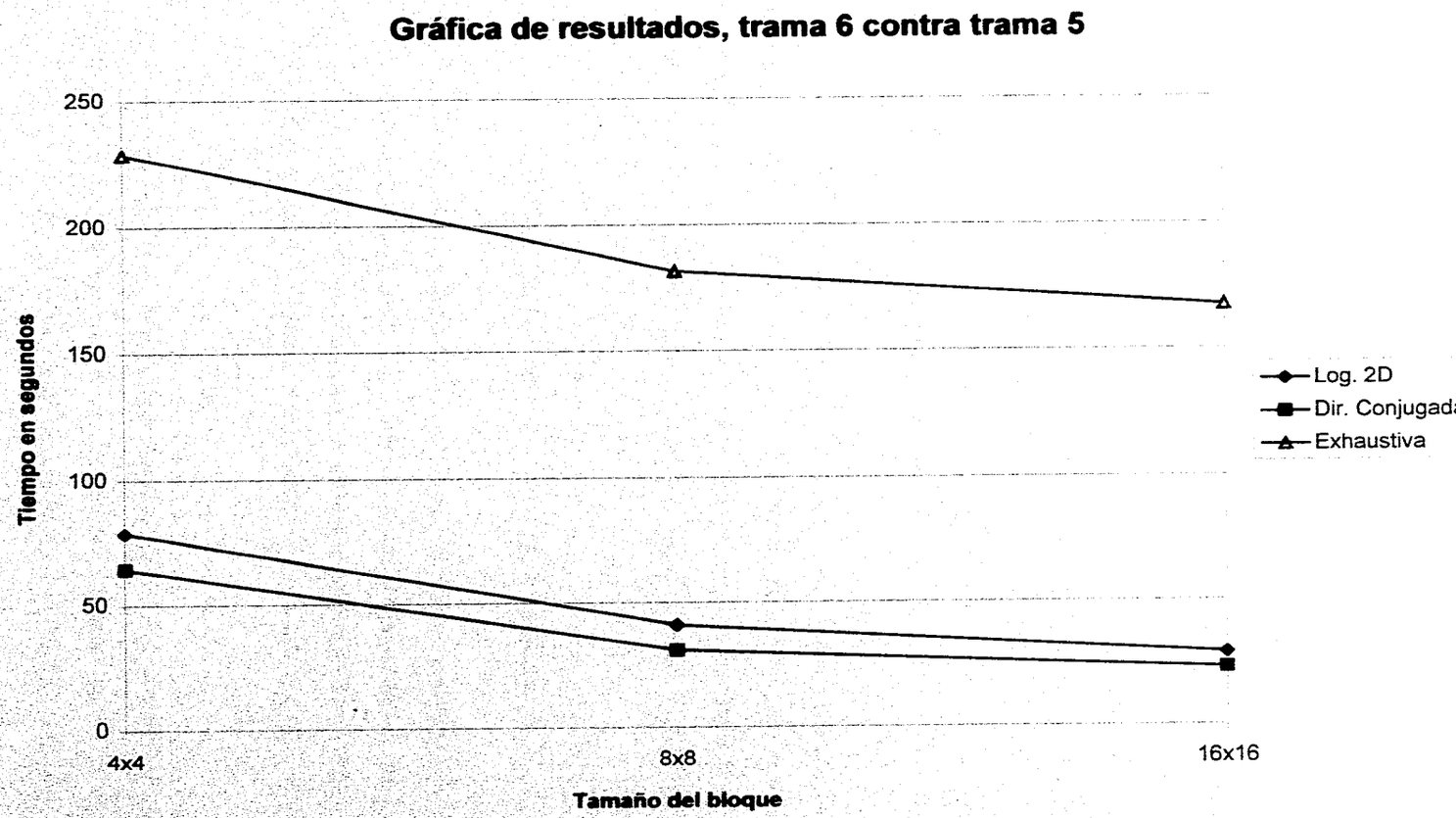


Imagen de búsqueda en la dirección conjugada

Tabla 5. Tiempos de procesamiento en la trama 6 comparada con la trama 5.

ALGORITMO	TAMAÑO DE BLOQUE	AREA DE BUSQUEDA	TIEMPO Segundos
Logarítmico 2D	4x4	8	78.8
Dirección conjugada	4x4	8	64.3
Exhaustivo	4x4	--	228.7
Logarítmico 2D	8x8	8	41
Dirección conjugada	8x8	8	31
Exhaustivo	8x8	--	181.4
Logarítmico 2D	16x16	8	29
Dirección conjugada	16x16	8	23
Exhaustivo	16x16	--	167.3


Gráfica de resultados, trama 6 contra trama 5
250
200
150
100
50
0
Tiempo en segundos
4x4
8x8
16x16
Tamaño del bloque
Log. 2D
Dir. Conjugada
Exhaustiva

CAPITULO 4.

CONCLUSIONES.

Como se ha visto, las mejoras que proporciona la estimación de movimiento a los sistemas de compresión utilizando el acoplamiento de bloques es evidente, tanto en su implementación de búsqueda, como en el número menor de operaciones y complejidad matemática comparada con otras técnicas de estimación de movimiento, tales como las de pixeles recursivas. Es así, que la utilización de la estimación de movimiento usando las técnicas de acoplamiento de bloques, constituye una herramienta muy adecuada y logra bajas tasas de bits para su transmisión o almacenamiento de secuencias de imágenes con movimiento.

Los resultados en los algoritmos de acoplamiento de bloques son usados en las secuencias *"Interview"*. Se compara la capacidad básica de acoplamiento y se observa la imagen en cada uno de los diferentes algoritmos, como son el algoritmo de búsqueda exhaustiva, el algoritmo de búsqueda logarítmica 2D y el algoritmo de búsqueda de dirección conjugada.

El tamaño original de la imagen *"Interview"* es de 256x256.

Al obtener los algoritmos de acoplamiento de bloques, podemos observar las diferencias que existen con respecto al tiempo de compresión de datos evaluando los algoritmos que se han desarrollado, en las imágenes podemos observar que existen algunas irregularidades o errores representados por pixeles que distorsionan la imagen.

Podemos comparar, después de las pruebas realizadas, que uno de los mejores métodos en cuanto a la exactitud de la predicción, es el algoritmo de la búsqueda exhaustiva aunque este, tiene una gran desventaja si lo comparamos con los otros dos

algoritmos, esto se debe a que el área de búsqueda es representada en toda la imagen, es decir no tiene un área de búsqueda específica y por consiguiente el tiempo es muy alto para obtener el código de la imagen.

El algoritmo de búsqueda logarítmica 2D, lo podemos definir como el punto medio entre los tres algoritmos evaluados, ya que en tiempo se encuentra entre los otros dos y la eficiencia de la predicción puede catalogarse como buena.

El algoritmo de búsqueda en la dirección conjugada, es el mas rápido de los tres algoritmos, pero la gran desventaja que tiene con respecto a los dos anteriores es que la predicción no es muy buena, es decir la imagen obtenida es la mas distorsionada de los tres métodos evaluados.

En las pruebas realizadas hemos hecho el análisis de los tres algoritmos variando el tamaño de bloque del algoritmo, también logramos ver que el mejor con respecto a la calidad de la predicción de la imagen se logró con el algoritmo de búsqueda logarítmica 2D con un tamaño de bloque de 16x16, ya que con este tamaño de bloque fue el mas rápido que con los otros dos tamaños de bloques y algoritmos, esto es posible compararlo al observar las tablas ya que fue evaluado para las seis tramas sucesivas de imágenes y en todas se logró ver que este algoritmo fue el óptimo y el mas adecuado. Tomando como referencia las gráficas es posible confirmar que lo anterior es cierto.

Los códigos de vídeo pueden ser usados en una variedad de aplicaciones incluyendo: vídeo teléfonos y vídeo conferencias. La estimación de movimiento es de alta compresión, donde se requiere una buena calidad de la imagen procesada.

BIBLIOGRAFÍA

I.- A. K. Jain, "Image data compression: A Review," Proceedings of the IEEE, vol. 69,1991.

II.- J. R. Jain y A. K. Jain, "Displacement Measurement and Its Application in Interframe Image Coding", IEEE transactions on communications, vol. COM-29, Diciembre 1993.

III.- H. G. Musmann y P. Pirsch, "Advances in Picture Coding". Proceedings of the IEEE, vol. 73, Abril 1985.

VI.- R. Srinivasan y K. R. Rao, "Predictive coding based on efficient motion estimation", IEEE Transactions on communications, vol. COM - 33, Agosto 1985.

V.- R. Srinivasan y K. R. Rao, "Motion Compensated Coder for Videoconferencing", IEEE Transactions on communications, vol. Com -35, Marzo 1987.

IV.- A. N. Netravali y B. G. Haskell, "Digital Pictures", Plenum Press, 1988.

VII.- Q. Wang y R. J. Clarke, "Motion estimation and compensation for image sequence coding", Signal processing: Image communication 4, Elsevier Science Publishers B. V., 1992.

VIII.- R. C. Gonzalez and R. E. Woods, "Digital Image Processing", Reading, MA: Addison - Wesley, 1992.

IX.- M. Rabbani and P. W. Jones, "Digital Image Compression Techniques", Bellingham, WA: SPIE Press, 1991.

X.- W. K. Pratt, "Image Transmission Techiques". New York : Academic Press, 1979.

XI.- A. N. Netravali and J. O. Limb. "Picture coding: A review". Proc. IEEE, vol. 68. 1980

XII.- R. Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation", in ICC 1989.

XIII.- W. K. Pratt, "Digital Image Processing". New York: Wiley, 1995.

XIV.- M. I. Sezan y R. L. Lagendijk, "Motion Analysis and Image Sequence Processing", Norwell , MA, Kluwer, 1993.

XV.- Berli, Springer - Verlag, "Image Sequence Analysis", T. S. Huang. 1991.

XVI.- W. K. Pratt, "Digital Image Processing", New York, Wiley, 1992.

XVII.- L.D.Davisson and R.M. Gray, Eds "Data Compression", (BenchMark Papers in Electrical Engineering and computers Science),Stroudsberg,P.A.:Dowthen H. 1989.

XVIII.- D.J. Connor And J.O. Limb "Properties of Frame Diference Signals Generated by Moving Images", IEEE Trance Commun., Vol Com-22.

XIX.- J.R. Jain And A.K. Jain, "Interframe Adaptive data Compression Techniques for Images", Sig. & Image Proc. Lab., Dep. Elec. & Compute. Eng.,Univ.CA Davis.,1989.

XX.- A.K. Jain, "Some NewTechniques in image Processing", In Proc. Symp. on Current Mathematical Problems in Image Science, 1988.

XXI.- C.M. Lind and S.C. Kwatra, "Motion Compensation interframe Image Coding", in ICC 1992.

XXII.- H.G. Musmann, "Predictive Image Coding", in Image Transmission Techniques, W. K. Pratt. Ed. New York. 1993.

XXIII.- D.N. Hein, "Video Data Compression Using Motion Compensation", in MIDCON , 1993.

XXIV.- N.S. Jayant, "Signal Compession: Tecnology Targets and Research Direcctions ", IEEE J. Selected Areas Commun, 1992.

XXV.- I.H. Witten, R.M. Neal, and J.G.Cleary "Aritmethic Coding for Data Compression", Commun. ACM 1989.

XXVI.- J.Max, "Quantizing for Minimum Distortion" IRE Trans.Informat. 1993.

XXVII.- A.N.Netravalli and J.A. Stuller, "Motion Compensation Transform Codig", Bell Syst Tech. 1994.

XXVIII.- A. Puri and R.Aravind, "Motion Compensated Video Coding", IEEE Trans. Circuits, Sys.Video Technol, Vol 1, 1991.

Apéndices

APÉNDICES

```
/* ************************************************************ */
/*                  Algoritmo de la búsqueda exhaustiva                  */
/* ************************************************************ */


/*Declaracion de variables*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <alloc.h>
#include <dos.h>
#include <graphics.h>
#include <conio.h>
#include <io.h>
#include <dir.h>
unsigned char pixel,carac;
unsigned long  int   search_win[50][50];
unsigned long  int   bloc[33][33];
unsigned long int    renglones;
unsigned long int    columnas;


/* **********Funci¢n que reserva memoria para cargar la imagen******** */
/* ********************************************************** */

unsigned char huge *dar_mem(unsigned long int tam_imag)
{
 unsigned char  huge *p;
 p=farcalloc(tam_imag,sizeof(char));
 if (p==NULL) {
          printf("asignaci¢n fallida - abortar ejecuci¢n ");
          exit(0);
         }
 return p;
}
/* ********************** Fin de la funci¢n ************************** */

/* ************************************************************ */

void b_r(
          char nombre[20],
```

```
            char trama_out[20],
            unsigned char huge *ini_imag,
            unsigned long int renglones,
            unsigned long int columnas)
{
FILE *archivo;
FILE *arch_header;
FILE *arch_raw;
struct ffblk ffblk;
unsigned char huge  *pp;
unsigned char huge  *pointer1;
unsigned char huge  *pointer2;
unsigned char     carac1;
unsigned long int tamano;
unsigned long int tamano_total;
unsigned long int header;
unsigned long int n,k;
unsigned long int size;
long int  long_arch;
long int  da;


tamano = renglones*columnas;

archivo=fopen(nombre,"rb");

da = fileno(archivo);
long_arch = filelength(da);


printf("%lu\n",long_arch);

tamano_total = long_arch;

header = tamano_total - tamano;


                        /* Se lee el encabezado del archivo bmp */

arch_header=fopen("header.hdr","wb");      /* y se crea un archivo   */

for(n=0;n<header;n++)                  /*header.hrd que contendr*/
        {
        carac1 = fgetc(archivo);          /* el encabezado        */
        fprintf(arch_header,"%c",carac1);
```

```
        }
fclose(arch_header);

for (n=0;n<renglones;n++)
        {
        for (k=0;k<columnas;k++)
                {
                carac1=fgetc(archivo);
                pointer1=( ini_imag + k + columnas*(renglones-n-1) );
                pointer2=pointer1;
                *pointer2 = carac1;
                }
        }

fclose(archivo);

arch_raw = fopen( trama_out,"wb");

pp=ini_imag;                    /* Se coloca el apuntador al  */
                                        /* principio  del segmento   */

for (n=0;n<tamano;n++)
        {
        fprintf(arch_raw,"%c",*pp);
        pp = pp + 1;
        }
fclose (arch_raw);
                        /* ************************************ */


}
/* ******************** FIN DE LA FUNCION ***************** */

/* ********************************************************** */
/* *                FUNCION BMP-RAW                         */
/* ****************************************** */

void bmp_raw (void)
{
FILE  *arch_datos;
char  nombre1[20];
char  nombre2[20];
unsigned long int   renglones;
unsigned long int   columnas;
unsigned long int   M;
```

```
unsigned long int    vel;
unsigned long int    tamano;
unsigned char  huge  *pp;
unsigned char  huge  *ini_imag;
unsigned char  carac1;
unsigned long int   n;



fscanf(arch_datos,"%lu",&renglones);
fscanf(arch_datos,"%lu",&columnas);
fscanf(arch_datos,"%lu",&M);
fscanf(arch_datos,"%lu",&vel);
fscanf(arch_datos,"%s",&nombre1);
fscanf(arch_datos,"%s",&nombre2);
fclose(arch_datos);



tamano=renglones*columnas;


pp=dar_mem(tamano);
ini_imag=pp;

                              /* Transforma al archivo1 de bmp a raw */
b_r(nombre1,"trama1.raw",ini_imag,renglones,columnas);
                              /* ********************************** */


                              /* Transforma al archivo2 de bmp a raw */
b_r(nombre2,"trama2.raw",ini_imag,renglones,columnas);
                              /* ********************************** */

farfree(ini_imag);


}


/* ************************************************************** */
/* ************************************************************** */
double  BMA(          unsigned char huge *pp,
                      unsigned long int M,
```

```
                    unsigned long int vel,
                    unsigned long int search_win[50][50],
                    unsigned long int bloc[33][33])
{
long int  mm,nn,m,n,blo,sea;
long int  i,j,Suma,menor,error_centro;
unsigned char nchar,mchar;
double   menor_flt,M_flt,MAE,err_centro_flt,MAE_centro;
menor = 100000;
/* ************************************************************** */
for (m=1;m<=(2*vel +1);m++)
        {
        for (n=1;n<=(2*vel+1);n++)
                {
                Suma = 0;
                for (i=1;i<=M;i++)
                        {
                        for (j=1;j<=M;j++)
                                {
                                blo=bloc[i][j];
                                sea= search_win[m+i-1][n+j-1];
                                Suma=Suma+abs(blo - sea);
                                }
                        }

                if ((m==(vel+1))&&(n==(vel+1))) {error_centro = Suma;}
                if ( Suma<=menor ) {mm=m;nn=n;menor=Suma;}

                }

        }

err_centro_flt = error_centro;
menor_flt = menor;
M_flt    = M;
MAE_centro= err_centro_flt/(M_flt*M_flt);
MAE      = menor_flt/(M_flt*M_flt);

if (MAE>(0.9)*MAE_centro) {mm=vel+1; nn=vel+1;}

mchar = mm;
*pp = mchar;
pp = pp + 1;
nchar= nn;
*pp = nchar;
```

```
return MAE;

}


/* ************************************************** */
/* * Funci¢n para cargar el arreglo search_win con la * */
/* *      parte correspondiente del frame k-1       * */
/* ************************************************** */
void llenar_srchwin(  unsigned long int M,
                      unsigned long int vel,
                      unsigned long int pc,
                      unsigned long int search_win[50][50],
                      unsigned char huge *inip_framek_1)

{
unsigned char huge   *ppp1;
unsigned long  int   pos_x,pos_y,renglo,colu,num,algo,otracosa,i,j;



                pos_y=(pc-1)/columnas;    pos_x=(pc-1)%columnas;
                renglo = (vel+1 - pos_y); colu = (vel+1 - pos_x);
                num = 1;
                if ( (vel+1) <= pos_y ) {renglo = 1;
                                         num = (pos_y - vel)*columnas+1;}
                if ( (vel+1) <= pos_x ) {colu   = 1;
                                         num = pos_x - (vel-1);}
                if ( ((vel+1) <= pos_x ) && ( (vel+1) <= pos_y ) )
                        { num = pc - vel * (columnas + 1);}

                 /* La posici¢n inicial de rellenado es */
                 /*        ( renglo, colu )          */

                algo    = (M + 2*vel);

                if ( pos_y > renglones - (vel+M) )
                { algo = renglones-(pos_y - vel); }

                otracosa = (M + 2*vel);

                if ( pos_x > columnas - (vel+M) )
```

```
                { otracosa = columnas-(pos_x - vel); }

                for (i=0;i < algo-renglo+1; i++)
                        { for (j=0; j < otracosa-colu+1; j++)
                                {
                                ppp1 = (num-1) + inip_framek_1;
                                search_win[renglo+i][colu+j]= *ppp1;
                                num = num + 1;
                                }
                          num = num - (otracosa-colu+1) + columnas;
                        }


                /* ***************** Fin de la funcion ***************** */

}


/* ************************************************** */
/* *   Rutina que carga en el arreglo "bloc"      * */
/* *       un bloque de MxM del frame k           * */
/* ************************************************** */

void  llenar_block(  unsigned long int M,
                     unsigned long int pck,
                     unsigned long int bloc[33][33],
                     unsigned char huge *inip_frame_k)
{
unsigned char  huge  *ppp2;
unsigned long int    i,j;

                for (i=0;i<M;i++)
                        { for (j=0;j<M;j++)
                                {
                                ppp2 = (pck + j) - 1 + inip_frame_k;
                                bloc[i+1][j+1]= *ppp2;
                                }
                                pck = pck + columnas;
                        }

                /* ***************** Fin de la funcion ***************** */

 }
```

```
/*
*********************************************************************
*/
/*
*********************************************************************
*/

void main (void)
{
FILE  *arch_datos;
FILE  *archivo;
char  nombre1[20]="trama1.raw";
char  nombre2[20]="trama2.raw";

unsigned long int   tamano;
unsigned long int   M;
unsigned long int   trama_k;
unsigned long int   adicional;
unsigned char  huge  *pp;
unsigned char  huge  *inip_framek_1;
unsigned char  huge  *inip_frame_k;
unsigned char  huge  *inip_Dvectors;
unsigned char  huge  *pointer;
unsigned char  huge  *puntero;
unsigned long  int   m,n,veces1,veces2;
unsigned long  int   vel,i,j,p,pck,pc;
double               distorsion;

bmp_raw();


/* Archivo que contiene la trama [k-1] */

fscanf(arch_datos,"%lu",&renglones);
fscanf(arch_datos,"%lu",&columnas);
fscanf(arch_datos,"%lu",&M);
fscanf(arch_datos,"%lu",&vel);

fclose(arch_datos);


archivo=fopen(nombre1,"rb");


/* *** Rutina que carga en la memoria RAM la imagen de trama [k-1] *** */
```

```
/* ************************************************************ */

tamano=renglones*columnas;
trama_k=M*columnas;  /*  segmento de M renglones de la trama k  */
veces1=renglones/M;
veces2=columnas/M;
adicional=2*(veces1*veces2);  /* segmento donde se almacenarn los */
                              /*   vectores de desplazamiento    */

pp=dar_mem(tamano+trama_k+adicional+1);

inip_framek_1=pp;



for(n=0;n<tamano;n++)
        {
        carac = fgetc(archivo);
        *pp =carac;
        pp = pp + 1;
        }
fclose(archivo);
/* ********************** fin de la rutina ************************** */

inip_frame_k=pp;


pointer=inip_frame_k + M*columnas;


archivo=fopen(nombre2,"rb");


/* **** Principia la aplicación del BMA sobre toda la imagen ********* */
/* ************************************************************ */

for (m=0;m<veces1;m++)
        {
        p=M*m*columnas;
        pp=inip_frame_k;



        /* ******* Rutina que carga M renglones de la trama k ******** */
```

```
        /* ********************************************************** */
        for(n=0;n<(columnas*M);n++)
                {
                carac = fgetc(archivo);
                *pp =carac;
                pp = pp + 1;
                }
        inip_Dvectors=pp;

        /* ******************** fin de la rutina ********************** */


        for (n=0;n<veces2;n++)
                {

                /* defino pc */
                pck = n*M + 1;
                pc  = pck + p;


                llenar_srchwin(M, vel, pc, search_win, inip_framek_1);

                llenar_block(M, pck, bloc, inip_frame_k);

                distorsion = BMA(pointer, M, vel, search_win, bloc);
                pointer = pointer + 2;

                }
        }

fclose (archivo);

/* ******** Guardando los resultados en el disco ************ */
/* ********************************************************** */
pp=inip_Dvectors;

archivo=fopen("vectores.mov","wb");

for (n=0;n<adicional;n++)
        {
        fprintf(archivo,"%c",*pp);
        pp = pp + 1;
        }
fclose (archivo);
}
```

```
/*.************************************************************ */
/*  Algoritmo de búsqueda Logarítmico 2D                        */
/*.************************************************************ */


/*Declaracion de variables*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <alloc.h>
#include <dos.h>
#include <graphics.h>
#include <conio.h>
#include <io.h>
#include <dir.h>
unsigned char pixel,carac;


/* *****Funci¢n que reserva memoria para cargar la imagen******* */
/*.************************************************************/

unsigned char huge *dar_mem(unsigned long int tam_imag)
{
 unsigned char  huge *p;
 p=farcalloc(tam_imag,sizeof(char));
 if (p==NULL) {
          printf("asignaci¢n fallida - abortar ejecuci¢n ");
          exit(0);
         }
  return p;
}
/* ********************** Fin de la función ***************** */
/************************************************************** */

void b_r(
          char nombre[20],
          char trama_out[20],
          unsigned char huge *ini_imag,
          unsigned long int renglones,
          unsigned long int columnas)
{
FILE *archivo;
FILE *arch_header;
FILE *arch_raw;
```

```
struct ffblk ffblk;
unsigned char huge  *pp;
unsigned char huge  *pointer1;
unsigned char huge  *pointer2;
unsigned char     carac1;
unsigned long int tamano;
unsigned long int tamano_total;
unsigned long int header;
unsigned long int n,k;
unsigned long int size;
long int  long_arch;
long int  da;


 tamano = renglones*columnas;

 archivo=fopen(nombre,"rb");

 da = fileno(archivo);
 long_arch = filelength(da);


 printf("%lu\n",long_arch);

 tamano_total = long_arch;

 header = tamano_total - tamano;

                        /* Se lee el encabezado del archivo bmp */

 arch_header=fopen("header.hdr","wb");    /* y se crea un archivo*/

 for(n=0;n<header;n++)              /*header.hrd que contendr*/
        {
        carac1 = fgetc(archivo);        /* el encabezado        */
        fprintf(arch_header,"%c",carac1);
        }
 fclose(arch_header);

                {
                carac1=fgetc(archivo);
                pointer1=( ini_imag + k + columnas*(renglones-n-1) );
                pointer2=pointer1;
                *pointer2 = carac1;
```

```
                }
        }

fclose(archivo);
                        /* ************************************ */
                        /* Ahora se lee la memoria y se escribe  */
                        /* un nuevo archivo con extension .raw   */
arch_raw = fopen( trama_out,"wb");

pp=ini_imag;                    /* Se coloca el apuntador al  */
                                        /* principio  del segmento    */

for (n=0;n<tamano;n++)
        {
        fprintf(arch_raw,"%c",*pp);
        pp = pp + 1;
        }
fclose (arch_raw);
                        /* ************************************ */


}
/* *************** FIN DE LA FUNCION ************************** */


/* ************************************************************ */
/****************************************************************** */

void bmp_raw (void)
{
FILE   *arch_datos;
char   nombre1[20];
char   nombre2[20];
unsigned long int    renglones;
unsigned long int    columnas;
unsigned long int    M;
unsigned long int    vel;
unsigned long int    tamano;
unsigned char  huge  *pp;
unsigned char  huge  *ini_imag;
unsigned char  carac1;
unsigned long int   n;


fscanf(arch_datos,"%lu",&renglones);
```

```
fscanf(arch_datos,"%lu",&columnas);
fscanf(arch_datos,"%lu",&M);
fscanf(arch_datos,"%lu",&vel);
fscanf(arch_datos,"%s",&nombre1);
fscanf(arch_datos,"%s",&nombre2);
fclose(arch_datos);



tamano=renglones*columnas;


pp=dar_mem(tamano);
ini_imag=pp;

                                /* Transforma al archivo1 de bmp a raw */
b_r(nombre1,"trama1.raw",ini_imag,renglones,columnas);
                                /* ********************************** */

                                /* Transforma al archivo2 de bmp a raw */
b_r(nombre2,"trama2.raw",ini_imag,renglones,columnas);
                                /* ********************************** */

farfree(pp);


}

/* ************************************************************** */

/*****************************************************************/
unsigned char huge *BMA(unsigned char huge *pp,
                                unsigned long int M,
                unsigned long vel,
                                unsigned long int search_win[40][40],
                                unsigned long int bloc[20][20])
{
long int  Suma,i,j,k,ii,jj,Dmin,q,l,stop,mm,nn,blo,sea,nprima,n,D[20][20];
double p,nnprima,nreal;
unsigned char nchar,mchar;

for (i=0;i<(2*vel+1);i++)
        { for (j=0;j<(2*vel+1);j++) {D[i+1][j+1]=0;} }
q=(vel+1);l=(vel+1);stop=0;
p=vel;
```

```
nprima = ( (log (p))/log(2) );
nnprima = nprima;
nreal = max(2,pow(2,(nnprima-1)));
n = nreal;

while (stop==0)
 {
 Dmin=10000;

 for (k=1;k<=5;k++)
        {

        if (k==1) {mm=q; nn=l;}
        if (k==2) {mm=q+n; nn=l;}
        if (k==3) {mm=q; nn=l-n;}
        if (k==4) {mm=q; nn=l+n;}
        if (k==5) {mm=q-n; nn=l;}

        if (D[mm][nn]==0)
                {
                /* ********Procedimiento para calcular distorsión********** */
                Suma=0;
                for (i=1;i<=M;i++)
                        {
                        for (j=1;j<=M;j++)
                                {
                                blo=bloc[i][j];   sea=search_win[mm+i-1][nn+j-1];
                                Suma=Suma+abs( blo - sea );
                                }
                        }
                D[mm][nn]=Suma;
                /*
*********************************************************** */
                }

        if (D[mm][nn] < Dmin) { Dmin=D[mm][nn]; ii=mm-q; jj= nn-l; }

        }  /* ** Se cierra el lazo "for" ** */

 if ( ((ii==0)&&(jj==0))||(abs(mm-(vel+1))>=vel)||(abs(nn-(vel+1))>=vel) )
        {
        n=n/2;
        if (n==1)
                {
                Dmin=10000;
```

```
nprima = ( (log (p))/log(2) );
nnprima = nprima;
nreal = max(2,pow(2,(nnprima-1)));
n = nreal;

while (stop==0)
 {
 Dmin=10000;

 for (k=1;k<=5;k++)
        {

        if (k==1) {mm=q; nn=l;}
        if (k==2) {mm=q+n; nn=l;}
        if (k==3) {mm=q; nn=l-n;}
        if (k==4) {mm=q; nn=l+n;}
        if (k==5) {mm=q-n; nn=l;}

        if (D[mm][nn]==0)
                {
                /* ********Procedimiento para calcular distorsión********** */
                Suma=0;
                for (i=1;i<=M;i++)
                        {
                        for (j=1;j<=M;j++)
                                {
                                blo=bloc[i][j];   sea=search_win[mm+i-1][nn+j-1];
                                Suma=Suma+abs( blo - sea );
                                }
                        }
                D[mm][nn]=Suma;
                /*
**********************************************************  */
                }

        if (D[mm][nn] < Dmin) { Dmin=D[mm][nn]; ii=mm-q; jj= nn-l; }

        }  /* ** Se cierra el lazo "for" ** */

 if ( ((ii==0)&&(jj==0))||(abs(mm-(vel+1))>=vel)||(abs(nn-(vel+1))>=vel) )
        {
        n=n/2;
        if (n==1)
                {
                Dmin=10000;
```

```
for (k=1;k<=9;k++)
        {
        if (k==1) { mm=q; nn=l;}
        if (k==2) {mm=q+1; nn=l+1;}
        if (k==3) {mm=q-1; nn=l-1;}
        if (k==4) {mm=q; nn=l+1;}
        if (k==5) {mm=q+1; nn=l;}
        if (k==6) {mm=q-1; nn=l;}
        if (k==7) {mm=q; nn=l-1;}
        if (k==8) {mm=q+1;nn=l-1;}
        if (k==9) {mm=q-1;nn=l+1;}
        if (D[mm][nn]==0)
                {
                /* * Procedimiento para calcular distorsi¢n * */
                Suma=0;
                for (i=1;i<=M;i++)
                        {
                        for (j=1;j<=M;j++)
                                {
                                blo=bloc[i][j];
                                sea=search_win[mm+i-1][nn+j-1];
                                Suma=Suma+abs( blo - sea );
                                }
                        }
                D[mm][nn]=Suma;
                /* ************************************** */
                }
        if (D[mm][nn] < Dmin)
                {
                Dmin=D[mm][nn];
                ii = mm - q; jj = nn - l;
                }

        }
    q = ii + q; l = jj + l;
    stop=1;

    }

}

else { q = q + ii; l = l + jj; }
}

mchar = q;
```

```
for (k=1;k<=9;k++)
        {
        if (k==1) { mm=q; nn=l;}
        if (k==2) {mm=q+1; nn=l+1;}
        if (k==3) {mm=q-1; nn=l-1;}
        if (k==4) {mm=q; nn=l+1;}
        if (k==5) {mm=q+1; nn=l;}
        if (k==6) {mm=q-1; nn=l;}
        if (k==7) {mm=q; nn=l-1;}
        if (k==8) {mm=q+1;nn=l-1;}
        if (k==9) {mm=q-1;nn=l+1;}
        if (D[mm][nn]==0)
                {
                /* * Procedimiento para calcular distorsión * */
                Suma=0;
                for (i=1;i<=M;i++)
                        {
                        for (j=1;j<=M;j++)
                                {
                                blo=bloc[i][j];
                                sea=search_win[mm+i-1][nn+j-1];
                                Suma=Suma+abs( blo - sea );
                                }
                        }
                D[mm][nn]=Suma;
                /* ************************************** */
                }
        if (D[mm][nn] < Dmin)
                {
                Dmin=D[mm][nn];
                ii = mm - q; jj = nn - l;
                }

        }
    q = ii + q; l = jj + l;
    stop=1;

    }

}

else { q = q + ii; l = l + jj; }
}

mchar = q;
```

```
*pp = mchar;
pp = pp + 1;
nchar= l;
*pp = nchar;
pp = pp + 1;

return pp;

}


/* ************************************************************** */
/* ************************************************************** */

void main (void)
{
FILE  *archivo;
FILE  *arch_datos;
char  nombre1[20]="trama1.raw";
char  nombre2[20]="trama2.raw";

unsigned long int   renglones;
unsigned long int   columnas;
unsigned long int   tamano;
unsigned long int   M;
unsigned long int   trama_k;
unsigned long int   adicional;
unsigned char  huge  *pp;
unsigned char  huge  *inip_framek_1;
unsigned char  huge  *inip_frame_k;
unsigned char  huge  *inip_Dvectors;
unsigned char  huge  *pointer;
unsigned char  huge  *puntero;
unsigned char  huge  *ppp1;
unsigned char  huge  *ppp2;
unsigned long  int   m,n,veces1,veces2;
unsigned long  int   vel,pos_x,pos_y,renglo,colu,num,algo,otracosa,i,j,p,pck,pc;
unsigned long  int   search_win[40][40];
unsigned long  int   bloc[20][20];

bmp_raw();



arch_datos=fopen("datos.inp","rt");
```

```
/* Archivo que contiene la trama [k-1] */

fscanf(arch_datos,"%lu",&renglones);
fscanf(arch_datos,"%lu",&columnas);
fscanf(arch_datos,"%lu",&M);
fscanf(arch_datos,"%lu",&vel);

fclose(arch_datos);

archivo=fopen(nombre1,"rb");


/* *** Rutina que carga en la memoria RAM la imagen de trama [k-1] *** */
/* ********************************************************** */

tamano=renglones*columnas;
trama_k=M*columnas;  /*  segmento de M renglones de la trama k  */
veces1=renglones/M;
veces2=columnas/M;
adicional=2*(veces1*veces2);  /* segmento donde se almacenarn los */
                              /*   vectores de desplazamiento    */

pp=dar_mem(tamano+trama_k+adicional+1);

inip_framek_1=pp;

for(n=0;n<tamano;n++)
        {
        carac = fgetc(archivo);
        *pp =carac;
        pp = pp + 1;
        }
fclose(archivo);
/* ********************** fin de la rutina ************************** */

inip_frame_k=pp;


pointer=inip_frame_k + M*columnas;


archivo=fopen(nombre2,"rb");
```

```
/* **** Principia la aplicación del BMA sobre toda la imagen ********* */
/* ********************************************************** */

for (m=0;m<veces1;m++)
        {
        p=M*m*columnas;
        pp=inip_frame_k;

        /* ******* Rutina que carga M renglones de la trama k ******** */
        /* ************************************************** */
        for(n=0;n<(columnas*M);n++)
                {
                carac = fgetc(archivo);
                *pp =carac;
                pp = pp + 1;
                }
        inip_Dvectors=pp;

        /* ******************** fin de la rutina ********************** */


        for (n=0;n<veces2;n++)
                {
                /* ************************************************** */
                /* * Rutina para cargar el arreglo search_win con la * */
                /* *      parte correspondiente del frame k-1       * */
                /* ************************************************** */

                /* defino pc */
                pck = n*M + 1;
                pc  = pck + p;

                /* **** Poniendo en ceros el arreglo search_win **** */
                for (i=0;i<39;i++)
                        { for (j=0;j<39;j++) {search_win[i+1][j+1]=0;} }
                /* ************************************************** */

                pos_y=(pc-1)/columnas;    pos_x=(pc-1)%columnas;
                renglo = (vel+1 - pos_y); colu = (vel+1 - pos_x);
                num = 1;
                if ( (vel+1) <= pos_y ) {renglo = 1;
                                         num = (pos_y - vel)*columnas+1;}
                if ( (vel+1) <= pos_x ) {colu  = 1;
                                         num = pos_x - (vel-1);}
                if ( ((vel+1) <= pos_x ) && ( (vel+1) <= pos_y ) )
```

```
        { num = pc - vel * (columnas + 1);}

/* La posición inicial de rellenado es */
/*        ( renglo, colu )          */

algo    = (M + 2*vel);

if ( pos_y > renglones - (vel+M) )
{ algo = renglones-(pos_y - vel); }

otracosa = (M + 2*vel);

if ( pos_x > columnas - (vel+M) )
{ otracosa = columnas-(pos_x - vel); }

for (i=0;i < algo-renglo+1; i++)
        { for (j=0; j < otracosa-colu+1; j++)
                {
                ppp1 = (num-1) + inip_framek_1;
                search_win[renglo+i][colu+j]= *ppp1;
                num = num + 1;
                }
          num = num - (otracosa-colu+1) + columnas;
        }


/* ***************** Fin de la rutina ****************** */


/* ******************************************************** */
/* *   Rutina que carga en el arreglo "bloc"       * */
/* *       un bloque de MxM del frame k            * */
/* ******************************************************** */


for (i=0;i<M;i++)
        { for (j=0;j<M;j++)
                {
                ppp2 = (pck + j) - 1 + inip_frame_k;
                bloc[i+1][j+1]= *ppp2;
                }
                pck = pck + columnas;
        }

/* ***************** Fin de la rutina ****************** */
```

```
                puntero = BMA(pointer, M, vel, search_win, bloc);
                pointer = puntero;
                }

        }

fclose (archivo);


pp=inip_Dvectors;

for (n=0;n<adicional;n++)
        {
        fprintf(archivo,"%c",*pp);
        pp = pp + 1;
        }
fclose (archivo);



}
```

```
w* ************************************************** */
/*     Algoritmo de búsqueda de dirección conjugada                */
/* ************************************************** */


/*Declaracion de variables*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <alloc.h>
#include <dos.h>
#include <graphics.h>
#include <conio.h>
#include <io.h>
#include <dir.h>
unsigned char pixel,carac;

/* *****Funci¢n que reserva memoria para cargar la imagen*****/
/* ********************************************************/

unsigned char huge *dar_mem(unsigned long int tam_imag)
{
 unsigned char  huge *p;
 p=farcalloc(tam_imag,sizeof(char));
 if (p==NULL) {
          printf("asignaci¢n fallida - abortar ejecuci¢n ");
          exit(0);
         }
  return p;
}
/* ********************** Fin de la funci¢n ************** */
/* ********************************************************** */

void b_r(
          char nombre[20],
          char trama_out[20],
          unsigned char huge *ini_imag,
          unsigned long int renglones,
          unsigned long int columnas)
{
FILE *archivo;
FILE *arch_header;
FILE *arch_raw;
struct ffblk ffblk;
```

```
unsigned char huge  *pp;
unsigned char huge  *pointer1;
unsigned char huge  *pointer2;
unsigned char     carac1;
unsigned long int tamano;
unsigned long int tamano_total;
unsigned long int header;
unsigned long int n,k;
unsigned long int size;
long int  long_arch;
long int  da;


tamano = renglones*columnas;

archivo=fopen(nombre,"rb");

da = fileno(archivo);
long_arch = filelength(da);


printf("%lu\n",long_arch);

tamano_total = long_arch;

header = tamano_total - tamano;


                        /* Se lee el encabezado del archivo bmp */

arch_header=fopen("header.hdr","wb");  /* y se crea un archivo   */

for(n=0;n<header;n++)               /*header.hrd que contendr*/
        {
        carac1 = fgetc(archivo);        /* el encabezado         */
        fprintf(arch_header,"%c",carac1);
        }
fclose(arch_header);
                        /* ******************************** */

                {
                carac1=fgetc(archivo);
                pointer1=( ini_imag + k + columnas*(renglones-n-1) );
                pointer2=pointer1;
                *pointer2 = carac1;
```

```
                }
        }

fclose(archivo);
                        /************************************ */

                    /* Ahora se lee la memoria y se escribe  */
                /* un nuevo archivo con extension .raw   */
arch_raw = fopen( trama_out,"wb");

pp=ini_imag;                /* Se coloca el apuntador al  */
                                    /* principio  del segmento    */

for (n=0;n<tamano;n++)
        {
        fprintf(arch_raw,"%c",*pp);
        pp = pp + 1;
        }
fclose (arch_raw);
                        /* ********************************* */


}
/* ************* FIN DE LA FUNCION ********************** */

/* ******************************************************** */
/* ******************************************************** */

void bmp_raw (void)
{
FILE  *arch_datos;
char  nombre1[20];
char  nombre2[20];
unsigned long int   renglones;
unsigned long int   columnas;
unsigned long int   M;
unsigned long int   vel;
unsigned long int   tamano;
unsigned char  huge  *pp;
unsigned char  huge  *ini_imag;
unsigned char  carac1;
unsigned long int   n;

fscanf(arch_datos,"%lu",&renglones);
fscanf(arch_datos,"%lu",&columnas);
```

```
fscanf(arch_datos,"%lu",&M);
fscanf(arch_datos,"%lu",&vel);
fscanf(arch_datos,"%s",&nombre1);
fscanf(arch_datos,"%s",&nombre2);
fclose(arch_datos);



tamano=renglones*columnas;


pp=dar_mem(tamano);
ini_imag=pp;

                /* Transforma al archivo1 de bmp a raw */
b_r(nombre1,"trama1.raw",ini_imag,renglones,columnas);
              /* ********************************** */


                /* Transforma al archivo2 de bmp a raw */
b_r(nombre2,"trama2.raw",ini_imag,renglones,columnas);
              /* ********************************** */

farfree(pp);


}

/* ************************************************** */
/* ************************************************** */
unsigned char huge *BMA(unsigned char huge *pp,
                        unsigned long int M,
                  unsigned long int vel,
                        unsigned long int search_win[40][40],
                        unsigned long int bloc[20][20])
{
long int  m,n,bandera,stopx,stopy;
long int  inix,iniy,dirx,diry,i,j,cont,D[20][20],Suma;
long int     blo,sea;
unsigned char nn,mm;
m=vel+1;n=vel+1;
bandera=0;stopx=0;stopy=0;inix=1;iniy=1;dirx=1;diry=1;cont=0;

for (i=0;i<(2*vel+1);i++)
       { for (j=0;j<(2*vel+1);j++) {D[i+1][j+1]=0;} }
```

```
/* ********procedimiento para calcular distorsi¢n******* */
        Suma=0;
        for (i=1;i<=M;i++)
                {
                for (j=1;j<=M;j++)
                {
                blo=bloc[i][j];      sea = search_win[m+i-1][n+j-1];
                Suma=Suma + abs( blo - sea);
                }
                }
        D[m][n]=Suma;
/* ******************Fin del procedimiento************ */
n=n+dirx;

while (bandera < 1)
{
 if (stopx==0)
 {

 /* ******Procedimiento para calcular distorsi¢n******** */
        Suma=0;
        for (i=1;i<=M;i++)
                {
                for (j=1;j<=M;j++)
                {
                blo = bloc[i][j];        sea = search_win[m+i-1][n+j-1];
                Suma=Suma + abs( blo - sea);
                }
                }
        D[m][n]=Suma;
/* *********************Fin del procedimiento************** */
if (D[m][n] == D[m][n-dirx]) {stopx=1;n=n-dirx;}
if (D[m][n] < D[m][n-dirx]) {n=n+dirx;}
if ((D[m][n] > D[m][n-dirx])&&(inix==1)) {n=n-2*dirx; dirx = -dirx;}
else { if (D[m][n] > D[m][n-dirx]) {stopx=1; n=n-dirx;} }
inix=0;

} /* **** Termina b£squeda en x **** */

if (stopx==1)

{
if (iniy==1) {m=m+diry;}
```

```
/* **********Procedimiento para calcular distorsi¢n****** */
        Suma=0;
        for (i=1;i<=M;i++)
                {
                for (j=1;j<=M;j++)
                {
                blo = bloc[i][j];       sea = search_win[m+i-1][n+j-1];
                Suma=Suma + abs( blo - sea);
                }
                }
        D[m][n]=Suma;
/* *************Fin del procedimiento********************** */
if (D[m][n] == D[m-diry][n]) {stopy=1;m=m-diry;}
if (D[m][n] < D[m-diry][n]) {m=m+diry;}
if ((D[m][n] > D[m-diry][n])&&(iniy==1)) {m=m-2*diry; diry = -diry;}
else { if (D[m][n] > D[m-diry][n]) {stopy=1; m=m-diry;} }
iniy=0;

if (stopy==1) {bandera=1;}

} /* **** Termina b£squeda en y **** */

if ((m==(2*vel+1))||(n==(2*vel+1))) {bandera = 1;}
cont = cont + 1; if (cont==15) {bandera = 1;}
} /* **** Termina bucle "while" **** */


mm = m;
*pp = mm;
pp = pp + 1;
nn= n;
*pp = nn;
pp = pp + 1;

return pp;

}


/* ********************************************************** */
/* ********************************************************** */

void main (void)
{
FILE  *arch_datos;
```

```
FILE  *archivo;
char  nombre1[20]="trama1.raw";
char  nombre2[20]="trama2.raw";

unsigned long int   renglones;
unsigned long int   columnas;
unsigned long int   tamano;
unsigned long int   M;
unsigned long int   trama_k;
unsigned long int   adicional;
unsigned char  huge  *pp;
unsigned char  huge  *inip_framek_1;
unsigned char  huge  *inip_frame_k;
unsigned char  huge  *inip_Dvectors;
unsigned char  huge  *pointer;
unsigned char  huge  *puntero;
unsigned char  huge  *ppp1;
unsigned char  huge  *ppp2;
unsigned long  int   m,n,veces1,veces2;
unsigned long  int   vel,pos_x,pos_y,renglo,colu,num,algo,otracosa,i,j,p,pck,pc;
unsigned long  int   search_win[40][40];
unsigned long  int   bloc[20][20];

bmp_raw();



arch_datos=fopen("datos.inp","rt");

/* Archivo que contiene la trama [k-1] */

fscanf(arch_datos,"%lu",&renglones);
fscanf(arch_datos,"%lu",&columnas);
fscanf(arch_datos,"%lu",&M);
fscanf(arch_datos,"%lu",&vel);

fclose(arch_datos);



archivo=fopen(nombre1,"rb");

/* *** Rutina que carga en la memoria RAM la imagen de trama [k-1] *** */
```

```
/*
*********************************************************************
*/

tamano=renglones*columnas;
trama_k=M*columnas;  /*  segmento de M renglones de la trama k  */
veces1=renglones/M;
veces2=columnas/M;
adicional=2*(veces1*veces2);  /* segmento donde se almacenarn los */
                              /*   vectores de desplazamiento    */

pp=dar_mem(tamano+trama_k+adicional+1);

inip_framek_1=pp;

for(n=0;n<tamano;n++)
        {
        carac = fgetc(archivo);
        *pp =carac;
        pp = pp + 1;
        }
fclose(archivo);
/* ************* fin de la rutina ************************** */

inip_frame_k=pp;


pointer=inip_frame_k + M*columnas;


archivo=fopen(nombre2,"rb");


/* **** Principia la aplicación del BMA sobre toda la imagen * */
/* ********************************************************** */

for (m=0;m<veces1;m++)
        {
        p=M*m*columnas;
        pp=inip_frame_k;

        /* ******* Rutina que carga M renglones de la trama k ** */
        /* ********************************************************** */
        for(n=0;n<(columnas*M);n++)
                {
```

```
        carac = fgetc(archivo);
        *pp =carac;
        pp = pp + 1;
        }
inip_Dvectors=pp;

/* ******************** fin de la rutina ************ */


for (n=0;n<veces2;n++)
        {
/* ************************************************** */
/* * Rutina para cargar el arreglo search_win con la * */
/* *       parte correspondiente del frame k-1       * */
/* ************************************************** */

        /* defino pc */
        pck = n*M + 1;
        pc  = pck + p;


        pos_y=(pc-1)/columnas;    pos_x=(pc-1)%columnas;
        renglo = (vel+1 - pos_y); colu = (vel+1 - pos_x);
        num = 1;
        if ( (vel+1) <= pos_y ) {renglo = 1;
                                 num = (pos_y - vel)*columnas+1;}
        if ( (vel+1) <= pos_x ) {colu   = 1;
                                 num = pos_x - (vel-1);}
        if ( ((vel+1) <= pos_x ) && ( (vel+1) <= pos_y ) )
                { num = pc - vel * (columnas + 1);}

         /* La posición inicial de rellenado es */
         /*         ( renglo, colu )          */

        algo    = (M + 2*vel);

        if ( pos_y > renglones - (vel+M) )
        { algo = renglones-(pos_y - vel); }

        otracosa = (M + 2*vel);

        if ( pos_x > columnas - (vel+M) )
        { otracosa = columnas-(pos_x - vel); }

        for (i=0;i < algo-renglo+1; i++)
```

```
                { for (j=0; j < otracosa-colu+1; j++)
                        {
                        ppp1 = (num - 1) + inip_framek_1;
                        search_win[renglo+i][colu+j]= *ppp1;
                        num = num + 1;
                        }
                 num = num - (otracosa-colu+1) + columnas;
                }


        /* ***************** Fin de la rutina ***************** */


        /* ************************************************** */
        /* *    Rutina que carga en el arreglo "bloc"       * */
        /* *       un bloque de MxM del frame k             * */
        /* ************************************************** */


        for (i=0;i<M;i++)
                { for (j=0;j<M;j++)
                        {
                        ppp2 = (pck + j) - 1 + inip_frame_k;
                        bloc[i+1][j+1]= *ppp2;
                        }
                        pck = pck + columnas;
                }

        /* ***************** Fin de la rutina ***************** */


        puntero = BMA(pointer,M,vel,search_win, bloc);
        pointer = puntero;
        }

    }

fclose (archivo);
}
```