REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 8, número 1
Págs. 62 - 78

Usando Diferencias en la Estimación de Movimiento

Carlos Andrés Mera Banguero * María Patricia Trujillo *

Resumen

La estimación de movimiento es comúnmente utilizada para reducir la redundancia temporal, en compresión de video, la cual tiene un alto costo computacional. La reducción de la redundancia temporal se realiza mediante la estimación de los cambios de posición de los objetos entre dos frames consecutivos para transmitir únicamente la información que ha cambiado. Los algoritmos comúnmente usados en la estimación de movimiento están basados en la comparación de bloques, llamado Block Matching. En este artículo se propone el uso de las diferencias entre frames para reducir el costo computacional de los algoritmos basados en Block Matching. La propuesta se basa en las diferencias entre frames, lo cual permite detectar movimiento. Esta información es usada como información a priori para decidir si se usa o no una estrategia de Block Matching para la estimación de movimiento. Se comprobó experimentalmente que el rendimiento de los algoritmos mejora considerablemente usando la detección de movimiento como información a priori, en tanto que la calidad de la reconstrucción se mantiene.

Palabras clave: *Block Matching, Estimación de Movimiento, Compresión de Video, Diferencias.*

Abstract

Video compression technique removes temporal redundancy within frames and thus provides coding systems with high compression ratio. Temporal redundancy is detected by motion estimation. Motion estimation is usually the most computationally intensive part in a video encoder. Block Matching approach is commonly used for motion estimation due to its simplicity and good performance but is computationally expensive. This paper presents an approach which uses the difference between frames motion detection as a priori information in order to reduce the high computational complexity in an exhaustive search and fast search algorithms. A priori information is used for deciding whether or not to search for the closest block within a search window. The proposed approach maintains signal quality close to that of full search but with significant saving in computation.

Keywords: *Block Matching, Motion Estimation, Video Compresion, Differences.*

---

* Universidad del Valle, Escuela de Ingeniería de Sistemas y Computación, Ciudad Universitaria Melendez, Cali, Colombia, {carlosm, mtrujillo}@eisc.univalle.edu.co

1 Introducción

En la actualidad, diversas aplicaciones como videoconferencia, telemedicina y televisión digital requieren de altas tasas de compresión para transmitir y almacenar señales de video. La estimación de movimiento, una de las etapas en la compresión de video, juega un papel importante en la reducción de redundancia temporal. Estándares de compresión de video como MPEG-1, MPEG-2, MPEG-4, H.2.6.1 y H.2.6.3 [3][7][8] usan una estrategia de Block Matching para la estimación de movimiento.

Existen diferentes algoritmos basados en Block Matching para la estimación de movimiento, entre ellos, Full Search (FS) es considerado el más simple [9], aunque es computacionalmente costoso. FS busca dentro de todos los posibles bloques candidatos de una ventana de búsqueda, aquel bloque que sea más parecido al bloque de referencia, en términos de una función de costo (MAD, MAE). En la literatura se han propuesto diferentes algoritmos para reducir el costo computacional de FS, los cuales se pueden clasificar en: algoritmos que usan patrones de búsqueda, algoritmos que usan muestreo de píxeles y bloques, algoritmos que usan correlación espacial y/o temporal, y algoritmos que usan estrategias bioinspiradas.

Entre los algoritmos que utilizan patrones de búsqueda, los más reconocidos son Three-Step Search (TSS) [9], New Three-Step Search (NTSS) [11], Four-Step Search (FSS) [12] y Diamond Search [22]. Estos algoritmos se basan en la comparación entre bloques por pasos, evaluando heurísticamente algunos de bloques en la ventana de búsqueda. Entre las ventajas de los algoritmos que usan patrones de búsqueda están su simplicidad y eficiencia, sin embargo incurren en errores en la estimación de movimiento debido a que omiten cerca del 80 % de los bloques en la ventana de búsqueda.

Los algoritmos de muestreo de píxeles se caracterizan por reducir el número de píxeles que son usados para calcular la función de costo [18]. Otras propuestas, como las presentadas en [16] y [19] combinan el muestreo de píxeles con una estrategia de muestreo de bloques. Los algoritmos que usan correlación espacial y/o temporal estiman el vector de movimiento del bloque de referencia usando los vectores de movimiento de sus bloques vecinos [20], [21]. El vector de movimiento estimado es generalmente usado para cambiar el centro de búsqueda y

algunas veces el tamaño de la ventana de búsqueda. Así mismo, se han desarrollado algoritmos de Block Matching basados en estrategias bioinspiradas, como algoritmos genéticos en [6], [13] y [17], y optimización basada en enjambres en [5].

En este artículo, se propone el uso de las diferencias entre frames para detectar movimiento [14]. Esta información es usada como información a priori para disminuir el costo computacional de la estimación de movimiento usando Block Matching. Los resultados experimentales muestran que la estrategia propuesta mejora el desempeño global de los algoritmos y reduce el tiempo de ejecución, en aproximadamente entre 30 % y 50 %, dependiendo de la cantidad de movimiento en la secuencia de video.

El resto del artículo está organizado como sigue. En la Sección 2 se presenta el problema de la estimación de movimiento. En la Sección 3 se muestra de manera general el proceso de Block Matching. En la Sección 4 se presenta la complejidad computacional de los algoritmos FS, TSS, NTSS y FSS. En la Sección 5 se presenta la propuesta de detección de movimiento para la estimación de movimiento. En la Sección 6 se presenta la complejidad computacional de la propuesta. En la Sección 7 se muestran algunos resultados experimentales. Finalmente, las conclusiones son presentadas en la Sección 8.

2 Estimación de Movimiento

El movimiento relativo entre los objetos de una escena y la cámara da lugar al movimiento aparente de los objetos en una secuencia de vídeo [1]. El objetivo de la estimación de movimiento es determinar este movimiento, el cual puede formularse en términos de la velocidad instantánea o del desplazamiento en los píxeles [4].

Debido a la naturaleza discreta de la información de luminancia en los frames de un video, la velocidad instantánea y el desplazamiento están relacionados por un factor constante Δt que corresponde al intervalo de captura entre frames. Generalmente, la velocidad instantánea y el desplazamiento son equivalentes, sin embargo en la estimación de movimiento no se estima el moviendo en cada píxel, sino que se estima el movimiento en un grupo de píxeles [4].

En general, en la estimación de movimiento se usa un modelo para representar el movimiento de los objetos. La ecuación (1) corresponde a

un modelo de movimiento que tiene en cuenta la traslación, rotación y escalamiento de los objetos, donde d_x y d_y son los componentes del vector de traslación, s es la razón de escalamiento y θ es el ángulo de rotación.

$$\begin{pmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \end{pmatrix} = s \begin{pmatrix} \cos\theta & -sen\theta \\ sen\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_t \\ y_t \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \end{pmatrix} \quad (1)$$

Debido a la complejidad y al costo computacional del modelo en (1) comúnmente se usa un modelo más simple que, bajo ciertos supuestos, aproxima movimientos complejos como la suma de traslaciones infinitesimales [4]. Este modelo es presentado en la ecuación (2).

$$\begin{pmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \end{pmatrix} \quad (2)$$

De acuerdo al modelo en (2), el movimiento de los objetos puede expresarse como el desplazamiento relativo de la información de luminancia en los píxeles de un *frame* con respecto a los *frames* subsiguientes de la secuencia de video. Este desplazamiento se calcula usando las variaciones espaciotemporales de luminancia en los píxeles. A pesar de la simplicidad del modelo en (2), establecer el desplazamiento a través de los cambios espaciotemporales de luminancia en cada píxel tiene un alto costo computacional.

El problema de estimación de movimiento puede definirse como la estimación de los componentes d_x y d_y del vector de traslación, donde el vector $(x_t, y_t)^T$ es la coordenada en el *frame* capturado en el tiempo t y; el vector $(x_{t+\Delta t}, y_{t+\Delta t})^T$ es la coordenada en el *frame* capturado en el tiempo $t+\Delta t$.

3 Block Matching

Debido al alto costo computacional de la estimación de movimiento, los componentes del vector de traslación son estimados para grupos de píxeles, llamados bloques. Los algoritmos de *Block Matching* calculan una función de costo para establecer la similitud entre los bloques de dos *frames* sucesivos de una secuencia de video [2]. En la figura 1 se ilustra este proceso, el cual se describe a continuación.

Sean I_t e I_{t+1}, el *frame* de referencia y el *frame* siguiente, respectivamente. I_t se divide en bloques de tamaño NxM y para cada bloque se sigue el siguiente procedimiento:

I. Se establece una ventana de búsqueda en I_{t+1} cuya ubicación se fija de acuerdo a la localización del bloque de referencia. Esta ventana de búsqueda tiene un tamaño de (2P+1)x(2P+1) píxeles, donde P es un parámetro que especifica el máximo desplazamiento del bloque.

II. La información de luminancia en el bloque de referencia se compara con la información de luminancia en todos los bloques dentro de la ventana de búsqueda y se elige aquel bloque que minimice la función de costo.

III. Se establece el vector de movimiento, el cual corresponde al desplazamiento relativo de la información de luminancia en el bloque de referencia con relación al bloque más parecido en la ventana de búsqueda (aquel que minimice la función de costo). Este vector de movimiento hace referencia al vector de traslación en la ecuación (2).

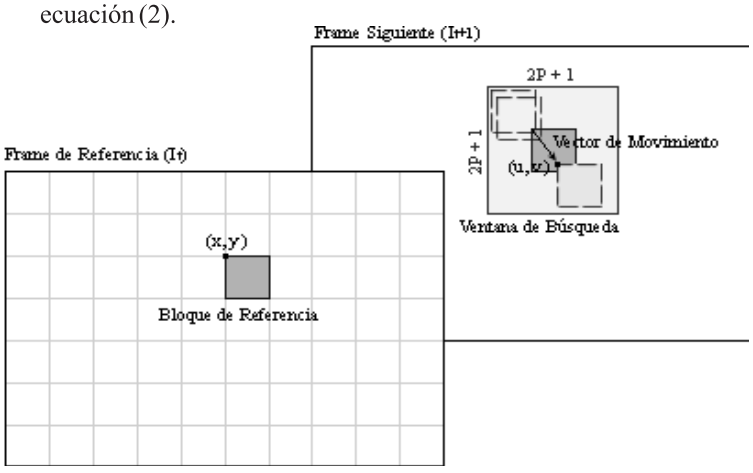


Fig 1. Ilustración del proceso de *Block Matching*

Existen varias funciones de costo [10], siendo las más usadas en la literatura: *Mean Absolute Difference* (MAD) en (3) y *Mean Squared Error* (MSE) en (4). En las ecuaciones (3) y (4), $I_t(x,y)$ e $I_{t+1}(u,v)$ corresponden a los valores de luminancia en las posiciones (x,y) y (u,v) en el bloque de referencia y el bloque en la ventana de búsqueda, respectivamente. $I_t(x+i,y+j)$ e $I_{t+1}(u+i,v+j)$ son los valores de luminancia con desplazamiento relativo (i, j).

$$MAD = \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \left| I_t(x+i, y+j) - I_{t+1}(u+i, v+j) \right| \quad (3)$$

$$MSE = \frac{1}{NM}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}\left(I_t(x+i,y+j) - I_{t+1}(u+i,v+j)\right)^2 \quad (4)$$

A pesar de su uso común, las técnicas basadas en *Block Matching* presenta dos grandes desventajas: la estimación de movimiento depende directamente del tamaño del bloque y del tamaño de la ventana de búsqueda. Adicionalmente, estas técnicas intentan estimar los vectores de movimiento en áreas que pertenecen al background.

4 Complejidad de los Algoritmos

La complejidad de los algoritmos más usados en estimación de movimiento se presenta a continuación, usando la siguiente notación:

- Tamaño del bloque: M x N píxeles.
- Tamaño del *frame*: W x H píxeles.
- Función de costo: *Mean Absolte Difference* (MAD).
- Costo computacional al restar el valor de intensidad de dos píxeles: K
- Máximo desplazamiento permitido: P píxeles
- Área de la ventana de búsqueda: $(2P+1)^2$ píxeles.

En general, el costo computacional de los algoritmos de estimación de movimiento usando *Block Matching* está determinado por el número de comparaciones que se realizan en la ventana de búsqueda, el número de bloques por *frame* y el costo de cada comparación. Si el tamaño del bloque aumenta entonces el número de bloques disminuye y viceversa. Por esta razón, el costo computacional no depende directamente del número de bloques en el *frame*, depende directamente del tamaño del *frame* y del tamaño de la ventana de búsqueda.

En la figura 2 se presentan las estrategias de búsqueda utilizadas por los algoritmos FS, TSS, NTSS y FSS usando una ventana de búsqueda de 15x15 píxeles. Con diferentes colores y figuras geométricas, se ilustra la posición de la coordenada en la esquina superior izquierda del bloque a comparar con el bloque de referencia. En la figura 2a, los cuadros corresponden a los bloques evaluados en el algoritmo FS. En la figura 2b, se presenta la estrategia de búsqueda del algoritmo TSS, donde los cuadros representan los bloques evaluados en el primer paso, los triángulos corresponden a los bloques evaluados en el segundo paso y los círculos corresponden a los bloques evaluados en el tercer paso. En la figura 2c, los cuadros corresponden a los bloques evaluados en el primer paso del algoritmo NTSS, los triángulos y los círculos muestran, respectivamente, un patrón de búsqueda con 3 o 5 bloques para el segundo y tercer paso. La figura 2d ilustra los bloques evaluados por el

algoritmo FSS, donde los cuadros, triángulos, círculos y estrellas son, respectivamente, los bloques evaluados en el primer, segundo, tercer y cuarto paso.

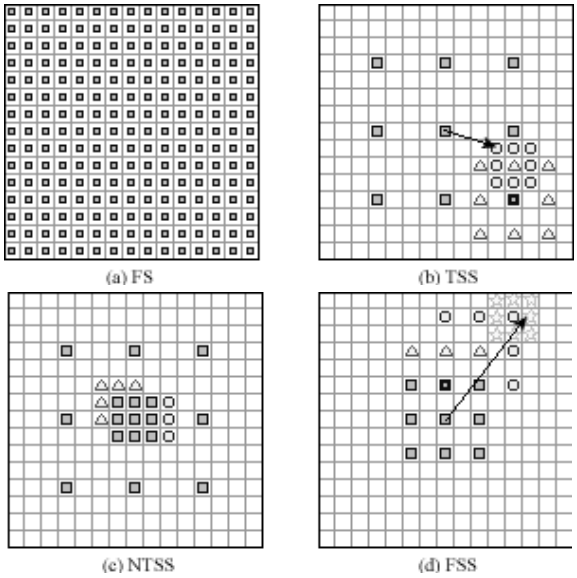


Fig. 2. Ilustración de los algoritmos de búsqueda FS, TSS, NTSS y FSS

El algoritmo FS realiza una búsqueda exhaustiva sobre toda la ventana de búsqueda. Para un desplazamiento máximo de 7 píxeles (P = 7) se obtiene una ventana de búsqueda de 15x15 píxeles, la cual contiene un total 225 bloques sobre los cuales se realizara la búsqueda. Por tanto, el algoritmo calcula la función de costo 225 veces. El costo computacional del algoritmo FS se presenta en (5). En esta ecuación, el factor *(M*N*K)* es el número de operaciones que se requieren para calcular la función de costo.

$$C_{FS} = 225 * \frac{W * H}{M * N} * (M * N * K) \qquad (5)$$

En 1981, T. Koga *et al* proponen el algoritmo *Three-Step Search* (TSS) [9] para reducir el costo computacional de FS. Koga *et al* sugieren que las comparaciones entre los bloques pueden hacerse por pasos evaluando heurísticamente ciertos bloques en la ventana de búsqueda. La estrategia planteada, en este algoritmo, requiere que en el primer paso se evalúen 9 bloques candidatos de los cuales se selecciona aquel que minimice la función de costo, de manera que en los pasos siguientes solo se evalúan los 8 bloques vecinos al bloque seleccionado en el paso previo, la figura 2a ilustra este proceso. Usando una ventana

de búsqueda de 15x15 píxeles el algoritmo calcula la función de costo 25 veces. La ecuación (6) ilustra la complejidad del algoritmo TSS.

$$C_{TSS} = 25 * \left(\frac{W * H}{M * N} \right) * (M * N * K) \qquad (6)$$

En 1994, R. Li *et al* presentan una modificación del TSS, llamada *New Three-Step Search* (NTSS) [11] la cual se basada en un patrón de búsqueda que inicia en el centro de la ventana de búsqueda para estimar pequeños movimientos de los bloques. En esta propuesta se utilizan los 9 bloques del primer paso del TSS y se agregan 8 comparaciones con los bloques ubicados en el vecindario del centro de la ventana de búsqueda, como lo muestra la figura 2b. NTSS usa una estrategia de parada en los dos primeros pasos (llamada halfway-stop) para mejorar el desempeño cuando el desplazamiento de los bloques ha sido nulo o casi nulo. De acuerdo a los resultados presentados en [11] NTSS es más robusto y produce errores de compensación más pequeños que TSS. Este algoritmo requiere, en el peor caso, calcular la función de costo en 33 bloques, mientras en el mejor caso solo requiere el cálculo en 17 bloques. La complejidad computacional de este algoritmo en el peor caso se presenta en la ecuación (7).

$$C_{NTSS} = 33 * \left(\frac{W * H}{M * N} \right) * (M * N * K) \qquad (7)$$

Otro algoritmo comúnmente usado es el *Four-Step Search* (FSS) [12], propuesto por Lai-Man Po y Wing-Chung Ma en 1996. Este algoritmo, al igual que NTSS, se basa en un patrón de búsqueda que usa el centro de la ventana de búsqueda para estimar pequeños movimientos de los bloques, sin embargo difiere en el número y la posición de los bloques utilizados para estimar los vectores de movimiento. La figura 2c muestra los patrones de búsqueda planteados por FSS, en los cuales solo se utilizan 3 o 5 bloques, en el segundo y tercer paso, en lugar de 8 como ocurre en TSS o NTSS. De acuerdo a los resultados presentados en [12] el algoritmo FSS produce mejores resultados que TSS y resultados comparables con los de NTSS. En el peor caso, FSS requiere calcular la función de costo en 27 bloques y en el mejor caso se calcula en 17 bloques. La complejidad computacional de este algoritmo para el peor caso se presenta en la ecuación (8).

$$C_{FSS} = 27 * \left(\frac{W * H}{M * N} \right) * (M * N * K) \qquad (8)$$

A manera de ilustración, en la tabla 1 se presenta el número de operaciones requeridas por los algoritmos mencionados. El tamaño de los *frames* con los que se calculó el número de operaciones corresponden a los estándares SIF (352x288), NTSC (720x480) y PAL (720x576).

Tabla 1. Número de operaciones requeridas por los algoritmos de estimación de movimiento que usan *Block Matching*

WxH	FS	TSS	NTSS	FSS
352x288	25.850.880*K*	2.534.400*K*	3.345.408*K*	2.737.152*K*
720x480	88.128.000*K*	8.640.000*K*	11.404.800*K*	9.331.200*K*
720X576	105.753.600*K*	10.368.000*K*	13.685.760*K*	11.197.440*K*

Los valores en la tabla 1 evidencian que de los algoritmos presentados FS es el aquel que más operaciones realiza, en tanto que TSS es el que menos operaciones lleva a cabo.

5 Usando Diferencias en la Estimación de Movimiento

La detección de movimiento puede realizarse mediante las diferencias píxel a píxel entre dos *frames* consecutivos de una secuencia de video [14]. Esta información puede usarse como información *a priori* en la estimación de movimiento usando *Block Matching*, para reducir la complejidad computacional.

En esta aplicación, la diferencia entre dos *frames* está representada por:

$$I_{diff}(x,y) = \begin{cases} 255 & si \mid I_t(x,y) - I_{t+1}(x,y) \mid \leq \beta \\ 0 & en\ otro\ caso \end{cases} \quad (9)$$

Donde β indica la diferencia máxima permitida entre dos valores de intensidad en dos píxeles para que sean considerados iguales. El uso del parámetro β permite de alguna manera eliminar el ruido.

La diferencia entre dos *frames* detecta los píxeles donde hay un cambio sustancial de intensidad y por tanto puede ser considerada como una estimación inicial del movimiento. Esta aproximación es válida bajo el supuesto de que la cámara está fija y la iluminación en la escena es constante [14].

Un píxel con valor cero en I_{diff} puede ser originado por una de las siguientes razones:

a. $I_t(x,y)$ pertenece a un objeto en movimiento y $I_{t+1}(x,y)$ pertenece al background o viceversa.
b. $I_t(x,y)$ e $I_{t+1}(x,y)$ pertenecen a objetos distintos que están en movimiento.
c. $I_t(x,y)$ pertenece un objeto en movimiento e $I_{t+1}(x,y)$ pertenece a una parte distinta del mismo objeto en movimiento.
d. Ruido que no fue eliminado con el parámetro β.

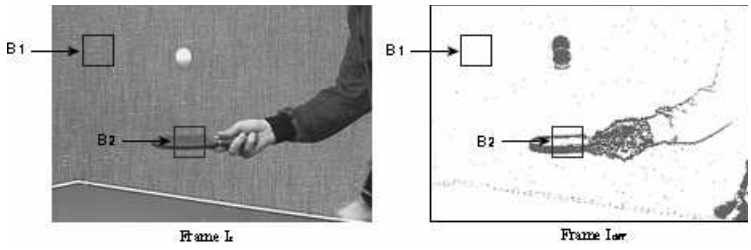
A continuación se presentan los pasos del algoritmo para usar las diferencias como información a priori en la estimación de movimiento.

I. Calcule I_{diff} de acuerdo a la ecuación (9).

II. Divida I_t e I_{diff} en bloques de tamaño NxM.

III. Para cada bloque en I_t, calcule el número de píxeles con valor cero presentes dentro de el bloque correspondiente en I_{diff} y llame a este valor NP. Si NP es mayor que un parámetro m, se ejecuta uno de los algoritmos de *Block Matching* para la estimación de movimiento de el bloque.

La figura 3 ilustra el uso de las diferencias. Para el bloque B_1 no se ejecuta un algoritmo de *Block Matching* debido a que el número de píxeles con valor cero en I_{diff} es menor que el umbral m. Sin embargo para el bloque B_2 un algoritmo de *Block Matching* es ejecutado dado que el número de píxeles con valor cero en I_{diff} es mayor que el umbral *u* .



priori en la estimación de movimiento

6 Complejidad Computacional de la Propuesta

La propuesta del uso de las diferencias adiciona a la complejidad de los algoritmos de estimación de movimiento basados en Block Matching, dos factores que corresponden al cálculo de las diferencias entre los frames y el costo de determinar el número de píxeles con valor diferente de cero de los bloques en I_{diff}. Las ecuaciones (10) y (11) ilustran, respectivamente, el costo computacional de estos dos factores, donde K' es una constante que determina el costo de incrementar el contador NP.

$$C_{Diff} = W * H * K \tag{10}$$

$$C_{sumasf} = W * H * K' \tag{11}$$

El peor caso del algoritmo propuesto se presenta cuando se detecta

movimiento en todos los bloques de I_{Idiff}. Cuando esto ocurre la complejidad de la estimación de movimiento está dada por la ecuación (12).

$$C_{\Pr opuesta} = C_{A\lg oritmo} + (W * H * K) + (W * H * K') \quad (12)$$

Donde, $C_{Propuesta}$ es el costo de la propuesta y $C_{Algoritmo}$ es el costo computacional presentado en la Sección 4 para cualquiera de los algoritmos FS, TSS, NTSS o FSS.

El mejor caso del algoritmo propuesto se presenta cuando no hay movimiento en los bloques de I_{Idiff}. La ecuación (13) ilustra la complejidad de la propuesta cuando esto ocurre.

$$C_{\Pr opuesta} = \left[(W * H)(K + K')\right] \quad (13)$$

Suponiendo que hay movimiento en la cuarta parte de los bloques de I_{Idiff} la complejidad computacional está determinada por la ecuación (14).

$$C_{\Pr opuesta} = \frac{1}{4} C_{A\lg oritmo} + \left[(W * H)(K + K')\right] \quad (14)$$

Suponiendo que hay movimiento en la mitad de los bloques de I_{Idiff}, la complejidad del algoritmo propuesto queda determinada por la ecuación (15).

$$C_{\Pr opuesta} = \frac{1}{2} C_{A\lg oritmo} + \left[(W * H)(K + K')\right] \quad (15)$$

Finalmente, si se supone que hay movimiento en las tres cuartas partes de los bloques de I_{Idiff} la complejidad del algoritmo propuesto queda determina por la ecuación (16).

$$C_{\Pr opuesta} = \frac{3}{4} C_{A\lg oritmo} + \left[(W * H)(K + K')\right] \quad (16)$$

7 Evaluación Experimental

En la evaluación experimental de la propuesta se utilizaron los algoritmos Full Search (FS) y Four Stept-Search (FSS). A los algoritmos usando información a priori se les nombró Full Search Modificado (FSM) y Four Stept-Search Modificado (FSSM). Estos algoritmos fueron implementados en C++ usando QT v4.1.4 sobre el sistema operativo Windows XP. Las pruebas se realizaron en un PC con 256 Mb de Memoria RAM y un procesador AMD Atlhon 2200 a 1.8 MHz de velocidad.

Para verificar la efectividad del método propuesto se emplearon los primeros 25 frames de las secuencias Claire (con poco movimiento), Salesman (con movimiento medio) y Table Tennis (con movimiento rápido). El tamaño del bloque se fijó en 8x8 píxeles y el tamaño de la ventana de búsqueda se fijó en 15x15 píxeles para las secuencias Claire y Salesman, y en 31x31 píxeles para la secuencia Table Tennis.

Las figuras 4, 6 y 8 muestran los tiempos de ejecución de los algoritmos para los 25 frames de cada secuencia usando FS, FSM, FSS y FSSM. Podemos observar que el desempeño de los algoritmos disminuye considerablemente cuando se usa la información a priori en la estimación de movimiento. El tiempo de ejecución varía entre frames cuando se usan FSM y FSSM debido a las variaciones de movimiento en los frames. Cuanto mayor es el movimiento entre frames mayor es el tiempo de estimación. En todos los casos, el uso de información a priori disminuye el tiempo de la estimación de movimiento, sin embargo para las secuencias poco movimiento esta disminución de tiempo es mayor que para las secuencias con movimiento medio y rápido.

Las figuras 5, 7 y 9 muestran los valores de PSNR para las secuencias de prueba usando los algoritmos FS, FSM, FSS y FSSM. Estas figuras evidencian que la calidad de la reconstrucción de los frames se mantiene e incluso en algunos casos mejora con respecto a los algoritmos que no usan información a priori. La figura 5 muestra que usando la estrategia planteada en secuencias con poco movimiento los valores de PSNR mejoran con respecto al algoritmo FS.

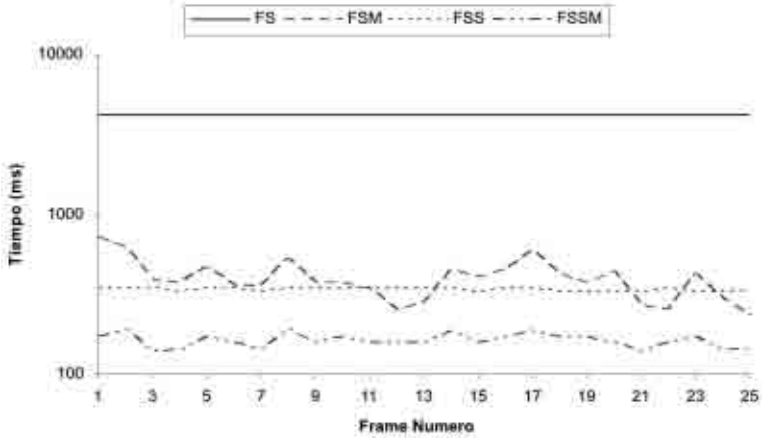


Fig. 4. Tiempos de ejecución con la secuencia *Claire*

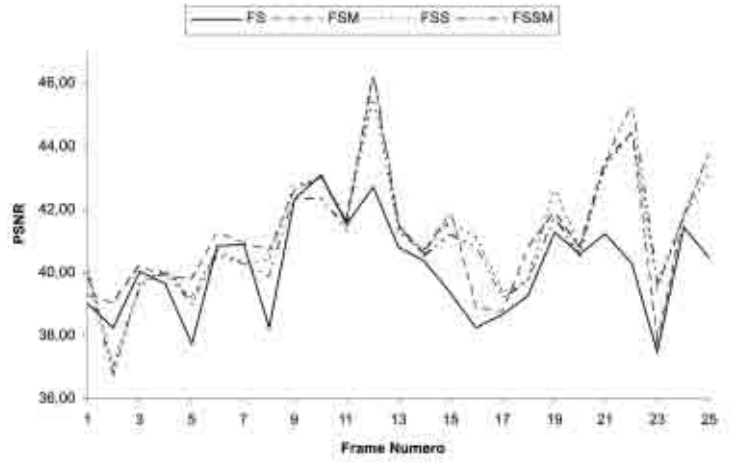


Fig. 5. PSNR obtenido con la secuencia *Claire*

La figura 7 muestra que para secuencias como *Salesman*, con movimiento medio, los valores de PSNR se mantienen muy cercanos para todos los algoritmos. Sin embargo, en la figura 9 se observa que para la secuencia *Table Tennis*, con movimientos rápidos, el algoritmo FSM obtuvo valores de PSNR más altos que los demás algoritmos, en algunos de los *frames*.

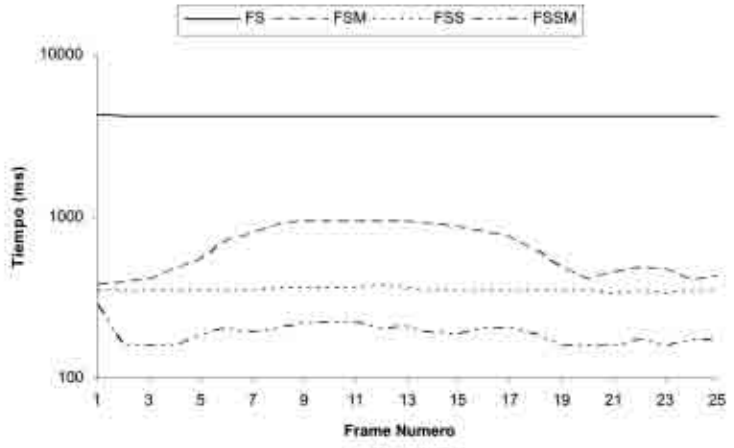


Fig. 6. Tiempos de ejecución con la secuencia *Salesman*

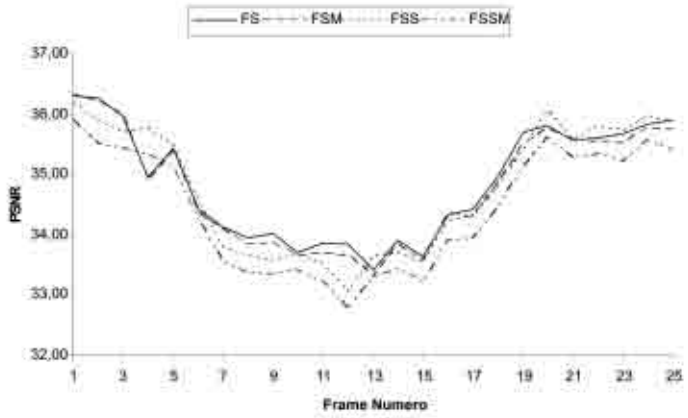


Fig. 7. PSNR obtenido con la Secuencia *Salesman*

8 Conclusiones

Se utilizó una estrategia basada en las diferencias entre *frames* para detectar movimiento. Esta detección de movimiento fue usada como información *a priori* en los algoritmos de estimación de movimiento usando *Block Matching* para reducir el costo computacional. El uso de información *a priori* redujo el número de bloques para los cuales se estiman los vectores de movimiento, en tanto que la calidad de la reconstrucción se mantiene. Los resultados experimentales mostraron que, usando información *a priori*, el rendimiento de los algoritmos de estimación de movimiento mejora considerablemente y en algunos casos también mejora la calidad de la reconstrucción en términos de PSNR.

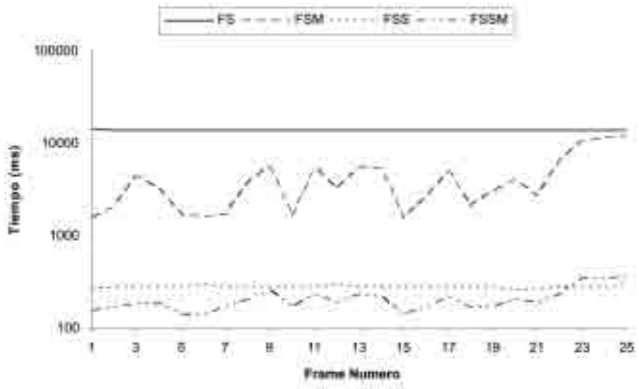


Fig. 8. Tiempos de Ejecución con la Secuencia *Table Tennis*

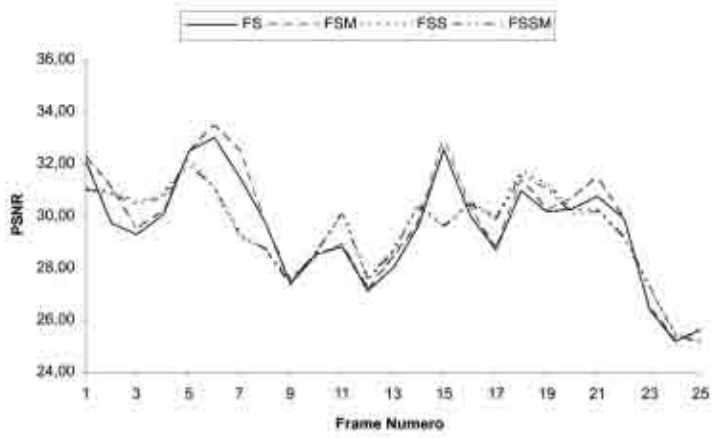


Fig. 9. PSNR Obtenido con la Secuencia *Table Tennis*

Referencias

[1] J. K. Aggarwal and N. Nandhakumar. On the Computation of Motion from Sequences of Images A Review; Proceedings of the IEEE, Vol. 76, 917 935, August 1988.

[2] E. Cahn and S. Panchanathan. Review of Block Matching Based Motion Estimation Algorithms for Video Compression; Canadian Conference on Electrical and Computer Engineering, Vol. 1, 151-154 , September 1993.

[3] CCITT Recommendation H.261. Video Codec for Audio Visual Services at p x 64 Kbits/s; COM XV-R 37-E, 1990.

[4] F. Defaux, and F. Moccheni. Motion Estimation Techniques for Digital TV: A Review and a New Contribution; Proceedings of the IEEE, Vol. 83, 858 876, June 1995.

[5] G. Y. Du; T. S. Huang; L. X. Song and B. J. Zhao. A Novel Fast Motion Estimation Method Based on Particle Swarm Optimization; Proceedings of International Conference on Machine Learning and Cybernetics, Vol. 8, 50385042, August 2005.

[6] K. Hung-Kei and M. L. Liou. Genetic Motion Search Algorithm for Video Compression; IEEE Transactions on Circuits and Systems for Video technology, Vol. 3, 440-445, December 1993.

[7] ISO/IEC 11172. Information Technology: Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to ~1.5 Mbit/s. 1993.

[8] ISO/IEC JTC1/SC29/WG11, CD 13818. Generic Coding of Moving Pictures and Associated Audio. November 1994.

[9] T. Koga *et al*. Motion Compensated Interframe Coding for Video Conferencing; Proceedings of the NTC, 9.6.1-9.6.5, New Orleans, November 1981.

[10] L. M. Lopes and A. P. Laves. Block Matching Algorithms in MPEG Video Coding; IEEE Proceedings of the 13th International Conference on Pattern Recognition, Vol. 3, 934-938, August 1996.

[11] R. Li, R; B. Zeng, and M. Liou. A New Three-Step Search Algorithm for Block Motion Estimation. IEEE Transactions Circuits System Video Technology, Vol. 4, 438-442, August 1994.

[12] L. M. Po, and W. C. Ma. A Novel Four-Step Search Algorithm for Fast Block Motion Estimation; IEEE Transactions Circuits System Video Technology, Vol. 6, 313-317, June 1996.

[13] M. F. So and A. Wu. Four-Step Genetic Search For Block Motion Estimation. IEEE International Conference on Acoustics, Speech, & Signal Processing, Vol. 3, 1393-1396, May 1998.

[14] M. Sonka; V. Hlavac and R. Boyle. Image Processing, Analysis and machine Vision, PWS Publishing Company, September 1998.

[15] J. Xuan and L. P. Chau. An Efficient Three-Step Search Algorithm for Block Motion Estimation; IEEE Transactions on Multimedia, Vol. 6, 435-438, June 2004.

[16] Y. Yu; J. Zhou and C. Wen. A Novel Fast Block Matching Motion Estimation Algorithm Based on Combined Subsamplings on Pixels and Search Candidates; Journal of Visual Communication and Image Representation, Vol. 12, 96-105, March 2001.

[17] X. Yuelei; B. Duyan and M. Baxin. A Genetic Search Algorithm for Motion Estimation; Proceedings of the 5th International Conference on Signal Processing, Vol. 2, 1058-1061, August 2000.

[18] A. Zaccarin and B. Liu. Fast Algorithms for Block Motion Estimation; IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol., 449-452, March 1992.

[19] Y. Yue *et al*. A Fast Effective Block Motion Estimation Algorithm; Proceedings Fourth International Conference on Signal Processing, 827-830, October 1998.

[20] R. K. Namuduri. Motion Estimation Using Spatio-Temporal Contextual Information. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 14, 1111-1115, August 2004.

[21] M. Rehan; P. Agathoklis and A. Antoniou. A New Motin-Estimation Technique for Efficient Video Compression. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing; Vol. 1, 326-329, August 1997.

[22] S. Zhu and K. K. Ma. A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation. IEEE Transactions on Image Processing, Vol. 9, 287-290, February 2000.