ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

ESTUDIO DE ALGORITMOS PARA LA ESTIMACIÓN DE MOVIMIENTO
(STUDY OF ALGORITHMS FOR MOTION ESTIMATION)

Para acceder al Título de

Graduado en
Ingeniería de Tecnologías de Telecomunicación

Autor: Sergio Barba Eguren

Octubre - 2017

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Sergio Barba Eguren
Director del TFG: Pablo Pedro Sánchez Espeso

Título: "Estudio de algoritmos para la estimación de movimiento"
Title: "Study of algorithms for motion estimation"

Presentado a examen el día: 27 de Octubre de 2017

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Antonio Tazón Puente
Secretario (Apellidos, Nombre): Héctor Posadas Cobo
Vocal (Apellidos, Nombre): Pablo Pedro Sánchez Espeso

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Agradecimientos

A mis padres, por su apoyo y confianza, por darme el empujón necesario para sacar las cosas adelante, por creer en mí.

A Pablo, mi tutor, por darme la oportunidad de trabajar con él y enseñarme, por ser paciente y dedicado.

A mis amigos, por hacer que mi sonrisa sea permanente.

A la gente del departamento, por las charlas a la hora del café y su ayuda cuando la necesitaba.

Palabras clave

Estimación de movimiento, algoritmo, búsqueda, codificación, video, macrobloque, pixel.

Keywords

Motion estimation, algorithm, search, codification, video, macroblock, pixel.

Índice

Listado de figuras

1. INTRODUCCIÓN

En este trabajo se desarrolla un sistema software de captura de video, con el consiguiente procesado fotograma a fotograma de las imágenes obtenidas de forma que se calculen los vectores de movimiento dentro de cada una de ellas. Dichos vectores pueden ser utilizados para estimar el movimiento de los elementos de la imagen.

El documento presenta las bases teóricas sobre la codificación de video y la estimación de movimiento, incluyéndose la descripción de varios algoritmos para realizar dicha estimación, y el análisis de sus prestaciones con un ejemplo práctico de uso de dos de ellos.

1.1. Objetivos

El principal objetivo de este trabajo consiste en implementar un sistema de captura de video y procesado del mismo, obteniendo los vectores de estimación de movimiento que se emplea en los códec de video.

Para ello se capturan las imágenes con una cámara y utilizando la librería de procesado de video OpenCV. La información es procesada empleando dos algoritmos diferentes: una búsqueda secuencial y una búsqueda en tres pasos de los elementos de la imagen que se han movido.

Cada uno de los algoritmos tiene unas características propias en cuanto a tiempo de procesado de la información, lo que se refleja en el resultado final.

Finalmente se muestran los resultados obtenidos y las diferencias existentes, mostrando una clara correspondencia con los resultados teóricos calculados.

1.2. Motivación

En el siglo XXI nuestra sociedad ha seguido experimentado el enorme desarrollo tecnológico iniciado un siglo antes, desde la aparición de la televisión a color hasta internet, pasando por el *boom* de la telefonía móvil a finales de los 90. Con todo ello, los sistemas se han ido perfeccionando desde sus inicios, mejorando sus prestaciones técnicas y la calidad de los servicios ofrecidos.

En el caso del video, esta mejora se puede comprobar día a día. Los dispositivos capaces de capturar, reproducir y gestionar video han pasado por varias etapas en las

que se han empleado diferentes tecnologías, siendo cada vez más pequeños, eficientes y baratos.

Esta mejora tecnológica trajo consigo un aumento en la calidad de los videos, que se hizo muy evidente con la aparición del video digital. Estos videos contienen una enorme cantidad de información que es necesario almacenar y transmitir, por lo que las técnicas de codificación de video han seguido desarrollándose en paralelo al aumento de la calidad de las grabaciones.

Dentro del proceso de codificación de video, la etapa de mayor importancia es la estimación de movimiento, ya que la diferencia entre su inclusión en un codificador de video y su ausencia puede suponer en algunos casos un 50% del tiempo total de cómputo.

Es por lo cual que los algoritmos ideados para aprovechar la redundancia temporal característica de un video han mejorado notablemente sus prestaciones, suponiendo una reducción muy significativa en el tiempo de cómputo mientras se minimiza el error cometido en la estimación.

El empleo de diferentes codificadores de video con características propias ha dado lugar a estándares muy extendidos, tales como la familia de MPEG's, H.264/MPEG-4 o el software libre empleado en internet, Theodora. [1]

1.3. Especificaciones

A continuación se describen las especificaciones que debe incluir el diseño final:

1) El sistema debe ser capaz de capturar imágenes en video.

2) Ese video debe de ser convertido de un formato RGB a YUV.

3) Las imágenes o *frames* capturados han de ser procesados de dos en dos, para hacer uso de la redundancia temporal.

4) Se deben calcular los vectores de movimiento de cada imagen, haciendo uso de dos algoritmos diferentes.

5) Los resultados del cómputo deben ser mostrados, así como las características de la salida en función del algoritmo empleado.

2. ESTADO DEL ARTE

2.1. ¿Qué es la codificación de video?

La codificación de video es un proceso mediante el cual se convierten las señales de video analógico captadas por una cámara en una señal de video digital, reduciendo y eliminando datos redundantes para codificar la información de manera más óptima sin afectar significativamente a la calidad de las imágenes.

Sin embargo, este procedimiento es un compromiso entre compresión y calidad. Una mayor reducción del tamaño del video codificado supone una transmisión más liviana del mismo, sacrificando calidad de imagen en el proceso.

La codificación de video se realiza a través de un códec de video, encargado de transformar el video empleando técnicas de compresión tanto estándar como patentadas. Estas técnicas de compresión estándar son fundamentales, ya que la codificación de video y la decodificación del mismo no suelen realizarse en el mismo terminal, siendo imprescindible que ambos compartan un procedimiento para tratar los datos.

El fundamento del códec de video es el algoritmo que emplea para codificar el video, que debe ser conocido por el decódec de video. Los códec/decódec de video que funcionan con estándares diferentes (diferentes algoritmos) no son capaces, por norma general, de tratar videos codificados/decodificados con otros estándares. Esto se debe a que un algoritmo trata y presenta la información de una manera concreta que otro algoritmo puede no ser capaz de interpretar.

2.2. Codificación de imagen vs codificación de video

2.2.1. Redundancia espacial redundancia temporal

Una de las principales referencias a la hora de estudiar la codificación de video es la codificación de imagen.

La compresión de imagen aprovecha la **redundancia espacial** que presenta cada fotograma para reducir la cantidad de información a enviar. Esto se debe a que en las imágenes no todo son zonas de transiciones bruscas entre pixeles adyacentes, sino que existen regiones dentro de cada imagen bastante homogéneas, con una variación nula o muy poco significativa entre los pixeles de esa región en concreto.

Figura 1: Redundancia espacial [2]

Es esta redundancia espacial lo que permite comprimir las imágenes de forma eficiente, seleccionando un pixel representativo de esa región uniforme y enviando solo las pequeñas diferencias (si existiesen) con respecto a este pixel de referencia. Las variaciones entre cada uno de los pixeles de esa zona y el pixel de referencia pueden codificarse empleando menos bits que si se mandase la información de cada uno de los pixeles de forma completa. [3]

Evidentemente, al realizar este procedimiento se puede producir una pérdida de información en función del nivel de compresión del codificador de imagen: codificadores con una estrategia más conservadora, llamados *lossless* o sin pérdidas, y codificadores de imagen que llevan a cabo una estrategia de compresión más agresiva a riesgo de perder algo de información, resultando en un impacto razonable en la imagen comprimida, llamados *lossy* o con pérdidas.

En definitiva, se podría considerar un video como una secuencia de imágenes consecutivas, que se muestras a una velocidad lo suficientemente grande como para que el ojo humano lo perciba como un movimiento continuado. Siguiendo este razonamiento, un codificador de video no sería más que un codificador de imagen secuencial, que comprime cada *frame* del video como si se tratase de una imagen independiente de las demás.

En este punto se llega a la clave de la codificación de video, el aprovechamiento de la **redundancia temporal** que existe dentro del mismo.

Como se ha mencionado anteriormente, la velocidad de captura de imágenes es lo suficientemente grande como para ser percibida por el ojo como un movimiento continuado y fluido. Sin embargo, la velocidad no es el único factor que permite este fenómeno, a ello también se le incluye que la diferencia entre cada imagen (o *frame*) es muy pequeña; es decir, las variaciones entre imágenes consecutivas de un video son sutiles, y eso es aprovechado para comprimir aún más un video.

La técnica que emplean los codificadores de video para aprovechar la redundancia temporal consiste en clasificar los *frames* en dos tipos:

- Tipo I o intra-frames: Se codifican igual que una imagen normal, solo aprovechando la redundancia espacial y no la temporal. La forma de codificarlos es muy similar a la que llevaría a cabo un codificador de imagen habitual. Se tratan como imágenes independientes.

- Tipo P o inter-frames: No son independientes respecto a otros *frames*, sino que su codificación depende de las imágenes previas.

Los *frames* I aprovechan la redundancia espacial, enviándose como si de una imagen normal se tratase. Los *frames* P aprovechan la redundancia temporal, enviándose solamente la información que varía respecto a la imagen anterior. [4]

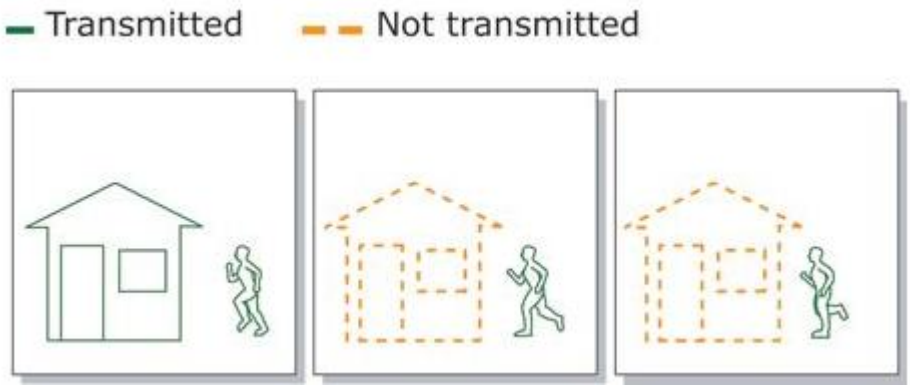


Figura 2: Redundancia temporal [2]

Como se aprecia en la figura 2, en esos tres *frames* consecutivos las variaciones dentro de la imagen son pequeñas, haciendo que solo sea necesario enviar los cambios entre imágenes, lo que reduce notablemente la información a enviar o almacenar.

Los *frames* tipo I se emplean como referencia que utilizan los *frames* P para reconstruir la imagen original en el decodificador. Conocida su referencia, solo deben aplicar los cambios que se han producido.

Es necesario que cada cierto tiempo se envíe un *frame* de tipo I para restablecer la referencia y no propagar un posible error en la cadena.

Este es el principio básico a la hora de aprovechar la redundancia temporal en video; sin embargo, existen técnicas más avanzadas que incluyen más tipos de *frames*, como los de tipo B o bipredictivos, cuyo objetivo es la compensación del movimiento tanto en *frames* anteriores como posteriores.

2.2.2. Redundancia estadística

Existe un tercer tipo de redundancia dentro de un video, la llamada **redundancia estadística**. Consiste en que dentro de una secuencia de imágenes, unos valores tienden a repetirse con más frecuencia que otros, lo que tiene un impacto positivo en la codificación.

Para su detección se emplean VLCs (*Variable Lenght Code*), que asignan códigos más largos en bits a los valores que aparecen menor número de veces, y códigos más cortos en los que aparecen con mayor frecuencia, optimizando la entropía del código. De esta forma, la cantidad de información a transmitir o almacenar es menor. [3]

2.3. Estimación de movimiento

Como se mencionaba anteriormente, la clave para alcanzar un buen ratio de compresión dentro de un video se basa en el aprovechamiento de la redundancia temporal, es decir, en la pequeña variación que se produce entre dos *frames* consecutivos. A pesar de la aparente sencillez de este procedimiento, es computacionalmente muy costoso, siendo el modulo incluido dentro de un codificador de video que más tiempo de ejecución necesita.

Supongamos una pelota que describe una trayectoria por el aire y que es, o bien observada por una persona o bien grabada con una cámara.

Si se mostrase el video fotograma a fotograma a baja velocidad, una persona sería capaz de identificar a la perfección las variaciones entre fotogramas. Es decir, nuestro cerebro puede identificar objetos, independientemente de su color, forma, grado de iluminación, rotación, etc. Y esto se debe a que el concepto de “pelota”, como de otros objetos, está en nuestro cerebro. Por otro lado, la cámara o el dispositivo que posteriormente realice la codificación de video no tiene ese tipo conocimiento, solo “ve” datos.

Con esto se concluye que lo que para nosotros es un proceso intuitivo y automático, su resolución a nivel tecnológico no es trivial.

La estimación de movimiento consiste en el cálculo de los **vectores de movimiento (MV)**, pudiendo entenderse estos como el número de pixeles de desplazamiento, tanto en una coordenada horizontal como vertical, que ha sufrido un objeto dentro de la imagen.

Considerando que una imagen es solamente información para un computador, surge la necesidad de dividir la misma en **macrobloques**, agrupaciones de pixeles con un tamaño de NxN pixeles, típicamente 8x8 o 16x16.

Estos macrobloques pertenecientes a una imagen concreta "i" se buscan en la imagen "i-1", calculándose el vector de movimiento asociado a ese macrobloque dentro de la imagen "i". Si este proceso se repite para todos los macrobloques, se obtendrían todos los vectores de movimiento de la imagen.

A fin de utilizar una nomenclatura más precisa, la imagen "i" se denomina *Target Frame* mientras que la imagen "i-1" se denomina *Reference Frame*.

2.4. Búsqueda con vectores de movimiento

La búsqueda de vectores de movimiento MV(u,v) se considera un problema de correspondencia entre imágenes. Como la búsqueda de MV es computacionalmente muy costosa, se limita a un pequeño rango en torno a las coordenadas del macrobloque que se pretende buscar.

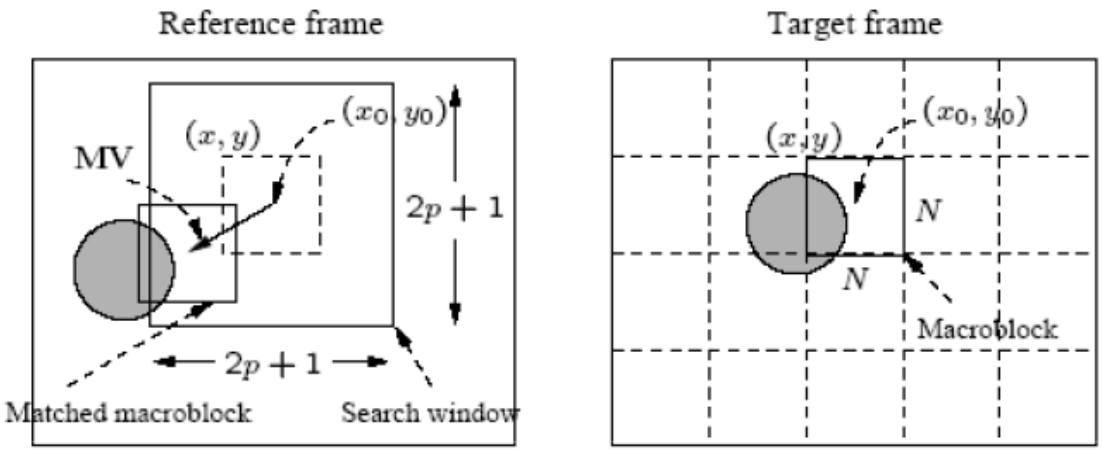


Figura 3: Ventana de búsqueda y vector de movimiento [5]

Como se ve en la figura 3, en el *Target Frame* se selecciona un macrobloque de NxN pixeles, ubicado en una posición (x0,y0). En el *Reference Frame* se expande una venta de búsqueda (*search window*) en torno a esa posición de macrobloque de tamaño 2p+1, en la que p es un numero entero de pequeño valor, y empleando un algoritmo concreto, se busca el conjunto de pixeles de NxN en el *Reference Frame* más parecido al macrobloque NxN en el *Target Frame*. La diferencia entre la posición final del macrobloque (x,y) y la posición original (x0,y0) es el vector de movimiento. [5]

Estas posiciones para referenciar los macrobloques no siempre son el centro mismo del macrobloque; de hecho, lo más habitual es hacer uso de la esquina superior izquierda de los macrobloques.

Una vez en esa ventana de búsqueda se procede a calcular numéricamente el macrobloque de mayor parecido al de referencia. Para este cálculo se emplean diferentes tipos de operaciones, como se mostrará en el siguiente apartado.

Por último es de vital importancia diferenciar entre el algoritmo de búsqueda y la operación a realizar.

- El algoritmo de búsqueda determina las posiciones dentro de la ventana de búsqueda en las que se pretende encontrar el macrobloque.
- La operación a realizar es la forma matemática de encontrar esa mayor similitud.

El algoritmo de búsqueda se posiciona dentro de la imagen de búsqueda, y una vez hecho esto, se comparan los bloques de NxN de ambas imágenes mediante la operación concreta de resolución de la discrepancia. Este procedimiento se repetirá tantas veces como iteraciones tenga el algoritmo.

Una descripción más detallada de los diferentes tipos de algoritmo de búsqueda dentro de la *search window* podrá encontrarse en capítulos posteriores.

2.1.1. Mean Absolute Difference

El MAD (*Mean Absolute Difference*) es un tipo de operación que se aplica para obtener la similitud entre el macrobloque que se busca y su posible correspondiente en otra imagen. [6]
Considerando que se emplea la esquina superior izquierda (x,y) como el origen del macrobloque en el *Target Frame*, macrobloques cuadrados de tamaño N, con

C(x+k,y+l) como los pixeles del macrobloque en el *Target Frame*, R(x+i+k,y+j+l) los pixeles del macrobloque en el *Reference Frame*, los iteradores *k* y *l* como índices para los pixeles y finalmente, *i* y *j* como los desplazamientos horizontal y vertical respectivamente. La diferencia entre dos macrobloques puede ser calculada usando el MAD:

$$\mathrm{MAD}(i,j) = \frac{1}{N^2} \cdot \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} |C(x+k, y+l) - R(x+i+k, y+j+l)|$$

El objetivo de la búsqueda es encontrar un vector (i,j) como el vector de movimiento MV=(u,v) que minimice el MAD(i,j) [5]

$$(u,v) = [(i,j)\ |\mathrm{MAD}(i,j) \quad \text{es minimo}, \quad i \in [-p,p], \quad j \in [-p,p]]$$

2.4.2. Sum Absolute Difference

El SAD (*Sum Absolute Difference*) es otro tipo de operación muy similar al MAD, con la diferencia que el resultado no está dividido por N^2. En cuanto a las consideraciones previas al planteamiento de la ecuación, son iguales al MAD. [6]

Considerando que se emplea la esquina superior izquierda (x,y) como el origen del macrobloque en el *Target Frame*, con C(x+k,y+l) como los pixeles del macrobloque en el *Target Frame*, R(x+i+k,y+j+l) los pixeles del macrobloque en el *Reference Frame*, los iteradores *k* y *l* como índices para los pixeles y finalmente, *i* y *j* como los desplazamientos horizontal y vertical respectivamente. La diferencia entre dos macrobloques puede ser calculada usando el SAD:

$$\mathrm{SAD}(i,j) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} |C(x+k, y+l) - R(x+i+k, y+j+l)|$$

El objetivo de la búsqueda es encontrar un vector (i,j) como el vector de movimiento MV=(u,v) que minimice el SAD(i,j) [5]

$$(u, v) = [(i, j) \,|\mathrm{SAD}(i, j) \quad \text{es minimo}, \quad i \in [-p, p], \quad j \in [-p, p]]$$

2.4.3. Mean Squared Error

El MSE (*Mean Squared Error)* es otro tipo de operación empleada en la búsqueda de los vectores de movimiento. De nuevo, las condiciones son iguales que en el caso del MAD, solo que el término dentro de los sumatorios está elevado al cuadrado. [6]

Considerando que se emplea la esquina superior izquierda (x,y) como el origen del macrobloque en el *Target Frame*, macrobloques cuadrados de tamaño N, con C(x+k,y+l) como los pixeles del macrobloque en el *Target Frame*, R(x+i+k,y+j+l) los pixeles del macrobloque en el *Reference Frame*, los iteradores *k* y *l* como índices para los pixeles y finalmente, *i* y *j* como los desplazamientos horizontal y vertical respectivamente. La diferencia entre dos macrobloques puede ser calculada usando el MSE:

$$\mathrm{MSE}(i, j) = \frac{1}{N^2} \cdot \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} C((x + k, y + l) - R(x + i + k, y + j + l))^2$$

El objetivo de la búsqueda es encontrar un vector (i,j) como el vector de movimiento MV=(u,v) que minimice el MSE(i,j) [5]

$$(u, v) = [(i, j) \,|\mathrm{MSE}(i, j) \quad \text{es minimo}, \quad i \in [-p, p], \quad j \in [-p, p]]$$

2.5. Algoritmos de búsqueda

2.5.1. Búsqueda secuencial

Es el algoritmo más sencillo de implementar y el que mejores resultados proporciona, sin embargo a nivel computacional es extremadamente costoso.

Se basa es una búsqueda exhaustiva pixel a pixel de una zona dentro de la ventana de búsqueda de (2p+1)·(2p+1) en el *Reference Frame*. No optimiza la búsqueda de ninguna manera, simplemente coteja todos los posibles macrobloques aplicando una operación concreta, generalmente el MAD, y el mejor resultado proporciona un MV (u,v).

Una descripción del algoritmo se muestra a continuación:

```
BEGIN
min_MAD = LARGE_NUMBER;          //Inicialización
for i = -p to p
   for i = -p to p
      {
         cur_MAD = MAD(i,j);
         if cur_MAD < min_MAD
        {
           min_MAD = cur_MAD;
           u = i; v = j;   //MV
        }
   }
END
```

Como se ha mencionado anteriormente, es muy costoso ya que cada comparación de pixel requiere tres operaciones (resta, valor absoluto y suma). Sin embargo, el cuello de botella de este algoritmo se encuentra en el elevadísimo número de iteraciones que realiza por cada macrobloque. [5]

$$(2p + 1) \cdot (2p + 1) \cdot N^2 \cdot 3$$

Aplicado a un caso concreto de un video de resolución 720x480, a 30 fps (*frames per second*), con un tamaño de macrobloque de N=16 y una ventana de búsqueda con p=15, se calcula el número de operaciones por segundo que necesitaría: [5]

$$\frac{OPS}{sec} = (2p + 1) \cdot (2p + 1) \cdot N^2 \cdot 3 \cdot \frac{720 \cdot 480}{N \cdot N} \cdot 30 \approx 3 \cdot 10^{10}$$

2.5.2. Búsqueda logarítmica 2D

Este algoritmo de búsqueda es una versión menos costosa que la búsqueda secuencial, pero que proporciona muy buenos resultados.

La ventana de búsqueda se caracteriza en 9 puntos dentro de la misma (numerados con 1). En esos puntos se calcula la operación para comparar macrobloques, generalmente el MAD, y se comparan los 9 resultados obtenidos. El punto que minimice el MAD será tomado como referencia para la siguiente iteración, pasando a ser el centro de un nuevo conjunto de 9 puntos, esta vez distanciados la mitad que en la iteración anterior, obteniéndose los puntos denominados como 2. [7][8]

Este proceso se realiza hasta encontrar el punto en el que el MAD global sea mínimo, denominado P(x,y). La diferencia entre P(x,y) y O(x0,y0) es el vector de movimiento.

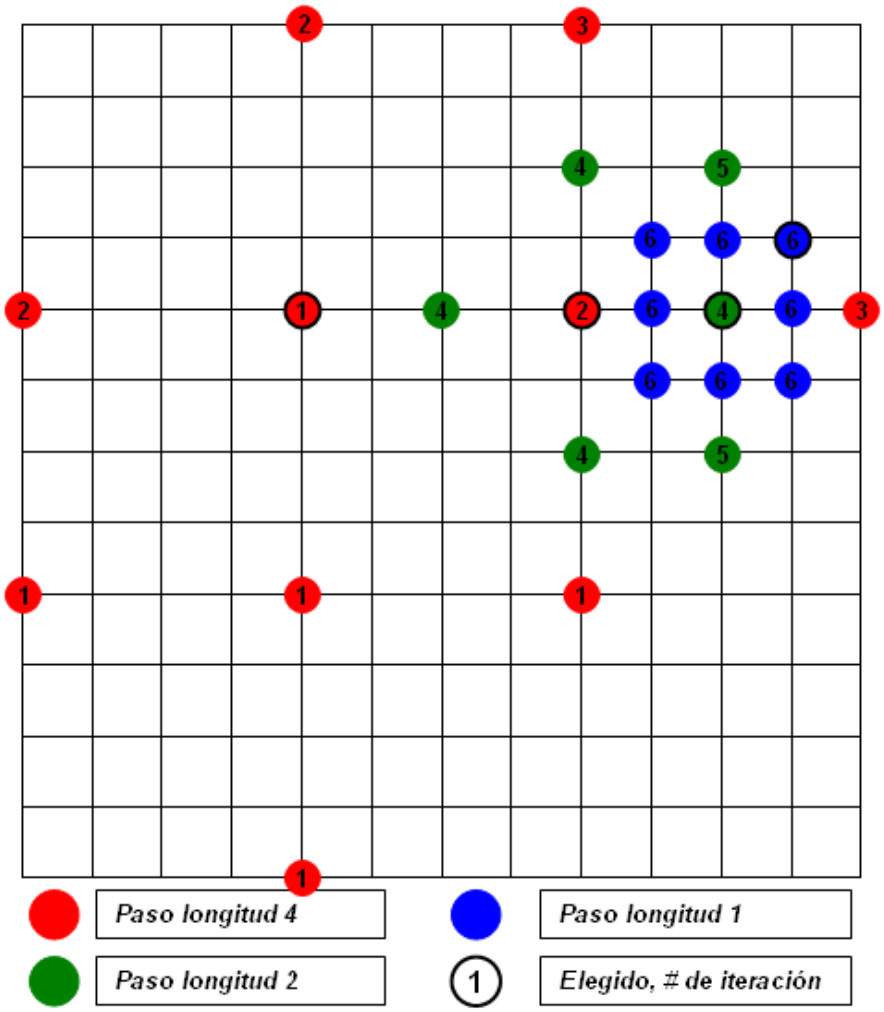


Figura 4: Pasos de búsqueda logarítmica 2D

Un pseudocódigo para la implementación del algoritmo es el siguiente:

BEGIN

offset = [p/2];

Se buscan 9 puntos dentro de la *search window*, centrados en el (x0,y0) y se parados por el offset tanto vertical como horizontalmente

WHILE last =! TRUE

{

Buscar uno de los 9 macrobloques que minimice el MAD;

If offset = 1 then las t= TRUE

offset = offset/2;

Conformar una nueva región con el nuevo offset y el nuevo centro

}

END

A diferencia de la búsqueda secuencial, en la que se comparaban (2p+1)·(2p+1) macrobloques respecto al original, en una búsqueda logarítmica solo se comparan $9 \cdot ([\log_2 p] + 1)$ macrobloques. Realmente el número de macrobloques es de $8 \cdot ([\log_2 p] + 1) + 1$, ya que el MAD del punto de referencia ya fue calculado en la iteración anterior, por lo que no sería necesario repetirlo.

Considerando un video de 720x480 a 30 fps con un tamaño de macrobloque de N=16 y una ventana de búsqueda con p=15, se calcula el número de operaciones por segundo necesarias: [5]

$$\frac{\text{OPS}}{\text{sec}} = (8 \cdot ([\log_2 p] + 1) + 1) \cdot N^2 \cdot 3 \cdot \frac{720 \cdot 480}{N \cdot N} \cdot 30 \approx 1.25 \cdot 10^9$$

La relación entre el número de operaciones en ambos algoritmos es de 1:24, siendo el logarítmico 24 veces más rápido.

2.5.3. Búsqueda jerárquica

La búsqueda de vectores de movimiento puede resultar beneficiada gracias a una aproximación jerárquica, en la cual la obtención de los primeros vectores de movimiento puede realizarse a partir de una reducción en la resolución de las imágenes.

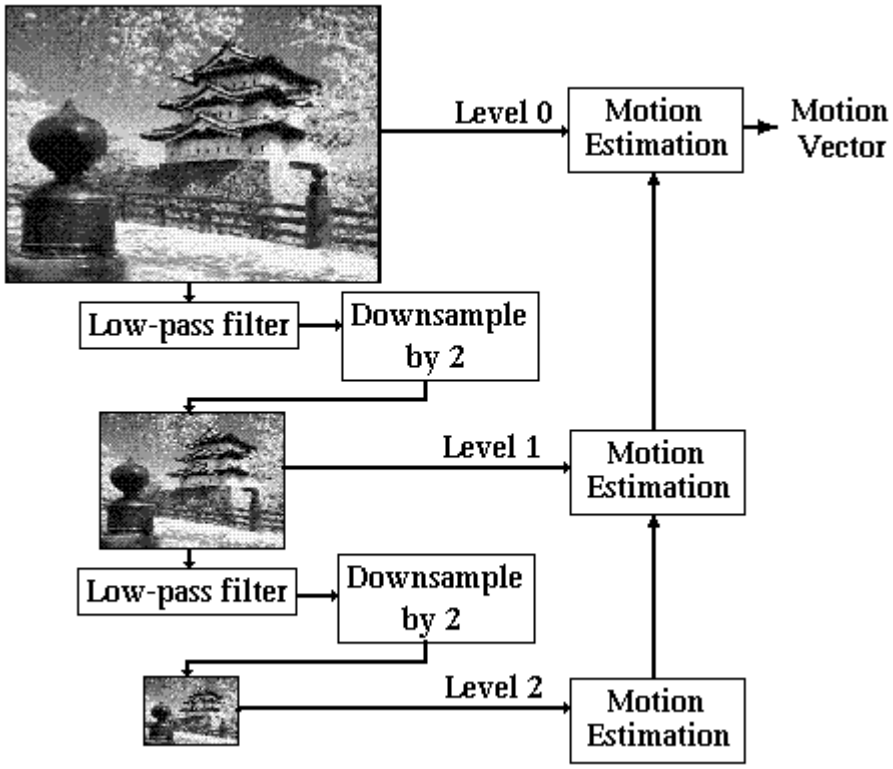


Figura 5: Esquema de implementación de una búsqueda jerárquica

Como se muestra en la figura [X], a partir de la imagen original se realiza un *downsampling* con factor 2, descendiendo en la jerarquía para calcular la estimación de movimiento en el nivel 2. A partir de ese punto, en el que se ha realizado un cálculo menos costoso que directamente con la imagen original (nivel 0), se sube en la jerarquía para seguir obteniendo la estimación de movimiento de forma más simple, hasta llegar al nivel 0, del que se extrae el vector de movimiento.

Debido a que el tamaño de los macrobloques es menor, la ventana de búsqueda p también se reduce, lo que supone una disminución del número de operaciones.

El primer cálculo de la estimación de movimiento (nivel 2) es demasiado tosco, debido a la falta de resolución y detalle de la imagen. Sin embargo, se irá refinando a medida que se aumenta de nivel en la jerarquía, y esta primera aproximación constituye una referencia válida para la búsqueda en niveles superiores. [7]

Dado un vector de movimiento (u^k, v^k) en el nivel jerárquico k, se busca en un marco de 3x3 centrado en $(2 \cdot u^k, 2 \cdot v^k)$ en el nivel k-1, para obtener un MV más preciso. [5]

Visto de otra forma, ese refinamiento en el nivel k-1 hace que los vectores de movimiento (u^{k-1}, v^{k-1}) cumplan:

$$\left(2 \cdot \mathrm{u}^{\mathrm{k}} - 1 \leq \mathrm{u}^{\mathrm{k}-1} \leq 2 \cdot \mathrm{u}^{\mathrm{k}} + 1, \qquad 2 \cdot \mathrm{v}^{\mathrm{k}} - 1 \leq \mathrm{v}^{\mathrm{k}-1} \leq 2 \cdot \mathrm{v}^{\mathrm{k}} + 1\right)$$

Con (x_0^k, y_0^k) siendo el centro del macrobloque en el nivel k del *Target Frame*, el procedimiento para obtener el vector de movimiento centrado en el macrobloque (x_0^0, y_0^0) puede ser descrito a partir del siguiente pseudocódigo:

BEGIN

Obtener el punto central del macrobloque en el nivel k de menor resolución

$x_0^k = x_0^0/2^k;\ y_0^k = y_0^0/2^k$

WHILE last =! TRUE

{

Encontrar el minimo MAD en los 9 macrobloques en el nivel k-1 en el rango

$\left(2 \cdot (\mathrm{u}^{\mathrm{k}} + x_0^k) - 1 \leq \mathrm{x} \leq 2 \cdot (\mathrm{u}^{\mathrm{k}} + x_0^k) + 1\right)$

$\left(2 \cdot (\mathrm{v}^{\mathrm{k}} + y_0^k) - 1 \leq \mathrm{y} \leq 2 \cdot (\mathrm{v}^{\mathrm{k}} + y_0^k) + 1\right)$

if k = 1 then last = TRUE

k = k-1;

Asignar (x_0^k, y_0^k) y $(\mathrm{u}^{\mathrm{k}}, \mathrm{v}^{\mathrm{k}})$ al nuevo centro de búsqueda y a su MV

}

END

Asumiendo la misma información que en el cálculo del número de operaciones por segundo en algoritmos descritos anteriormente (N=16, p=15, resolución 720x480, 30 fps) y con 3 niveles de jerarquía, el número de operaciones necesario es el siguiente: [5]

$$\frac{\mathrm{OPS}}{\mathrm{sec}} = [\left(2 \cdot \left[\frac{p}{4}\right] + 1\right)^2 \cdot \left(\frac{N}{4}\right)^2 + 9 \cdot \left(\frac{N}{2}\right)^2 + 9 \cdot N^2] \cdot 3 \cdot \frac{720 \cdot 480}{\mathrm{N} \cdot \mathrm{N}} \cdot 30 \approx 0.51 \cdot 10^9$$

2.5.4. Búsqueda en tres pasos (TSS/3SS)

La *Three Step Search* o TSS es un algoritmo de búsqueda ideado en 1981, muy empleado en estimación de movimiento debido a su gran simplicidad y alto rendimiento. En cada paso del algoritmo este busca en cada una de las 8 direcciones, como se muestra en la figura 6. [7]

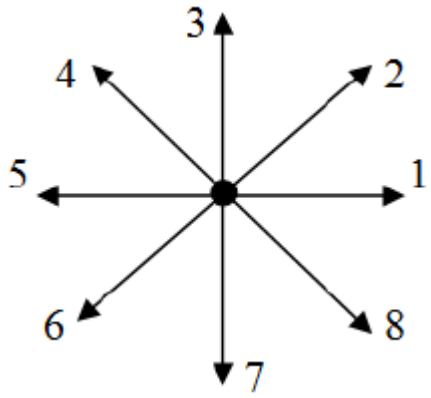


Figura 6: Puntos de búsqueda del TSS

El algoritmo puede ser descrito como:

1) Se escoge un *step* inicial. Se eligen 8 macrobloques alrededor del origen para compararlos con el macrobloque de referencia.

2) Una vez encontrado el que minimice el MAD, se reposiciona el centro en ese macrobloque y se reduce el *step* a la mitad.

3) Repetir los pasos 1 y 2 hasta que el *step* sea menor o igual a 1.

Un ejemplo del funcionamiento del algoritmo se muestra a continuación:

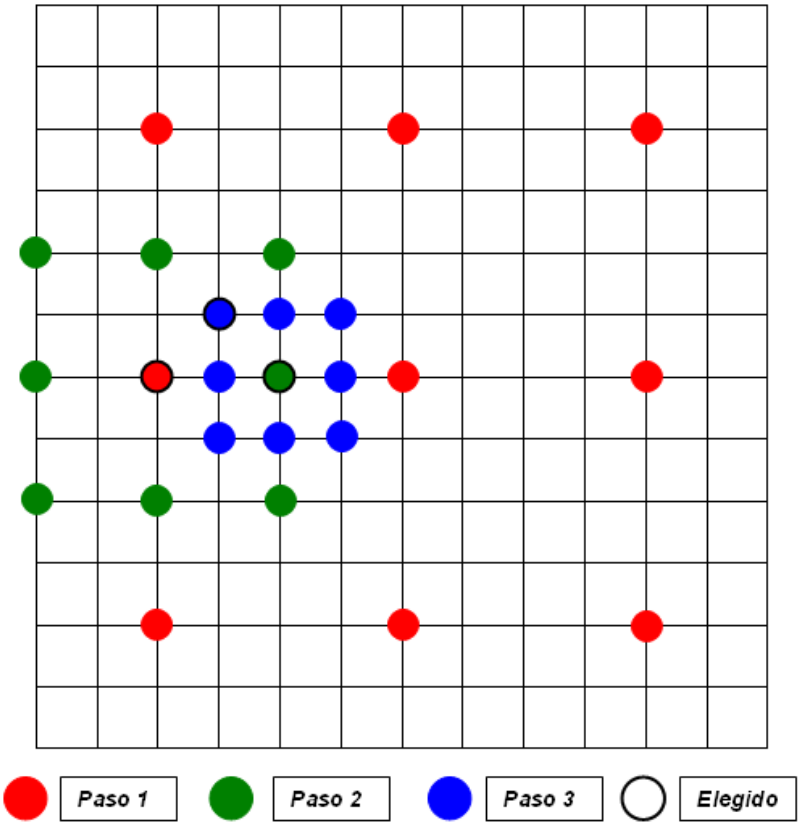


Figura 7: Pasos de búsqueda TSS

A nivel computacional, el TSS se comporta como una búsqueda logarítmica, realizándose 9 operaciones en el primer paso y 8 en los siguientes, ya que el MAD del punto central de referencia en el paso k fue calculado previamente en el paso k-1. [9][10]

Generalmente, para un algoritmo de este tipo en el que el número de pasos es k=3, supone:

$$\frac{\text{op}}{\text{mblq}} = \left(9\,(\text{k}=1) + (\text{k}-1)\cdot 8\,(\text{k}>1)\right)\cdot \text{n}_{\text{op}_{\text{MAD}}} = (9+2\cdot 8)\cdot 9 = 255$$

2.5.5. Nueva búsqueda en tres pasos (NTSS/N3SS)

El algoritmo NTSS (*New Three Step Search*) es una versión mejorada del TSS. El TSS emplea un patrón de búsqueda uniforme, lo que provoca que su eficiencia se reduzca en estimaciones de movimiento pequeñas.

La descripción del algoritmo es la siguiente:

1) Se inicia la búsqueda en 8 puntos a una distancia +/- S=4 (*step*) y en otros 8 puntos a +/- Sf=1, todos centrados en el origen.

2) Si el punto que minimiza el MAD se corresponde con el origen (0,0), se detiene el algoritmo, pues MV(0,0).

Si el punto que minimiza el MAD se corresponde con alguno de los puntos de S=4, se ejecuta el algoritmo como si se tratase de TSS, pasándose a la siguiente iteración del algoritmo.

Si el punto que minimiza el MAD se corresponde con alguno de los puntos de S=1, significa que se ha encontrado el MV(i,j), y se hace una búsqueda de Sf=1 para afinar su posición, y se termina el algoritmo.

3) Repetir los pasos 1 y 2 hasta que el *step* sea menor o igual a 1.

La gran diferencia con en TSS radica en la búsqueda fina de Sf=1. En ese caso pueden darse dos casos:

1) La búsqueda se inicia en unos de los 4 puntos situados en las esquinas del macrobloque, por lo que se deben comparar 5 puntos más.

2) Si la búsqueda se inicia en uno de los otros 4 puntos, el número de búsquedas adicional es de 3 puntos más.

Por cada macrobloque que se desea buscar, se deben realizar entre 17 y 33 operaciones de MAD sobre los posibles macrobloques de la *search window* para encontrar el vector de movimiento. [11]

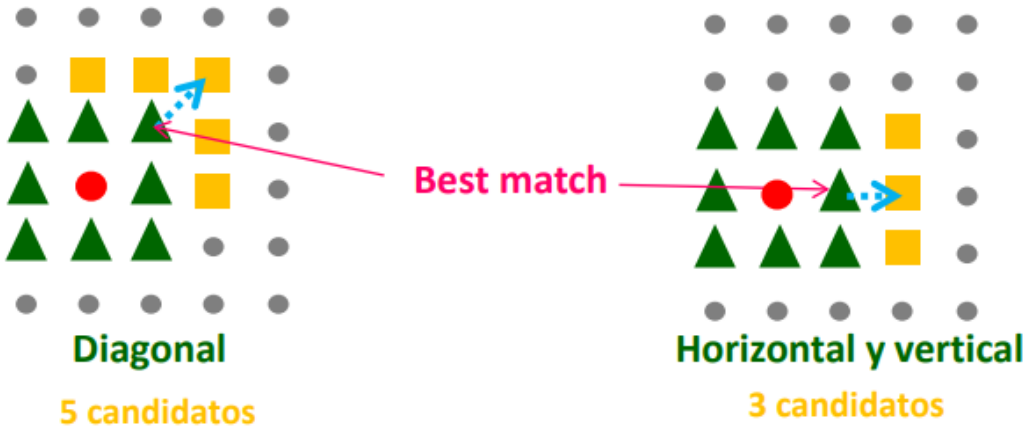


Figura 8: Último paso del la búsqueda NTSS [11]

Sabiendo que el cálculo del MAD supone 9 operaciones, el número de operaciones para cada macrobloque es el siguiente:

$$n_{op_{MAD}} \cdot 17 \leq \frac{op}{mblq} \leq n_{op_{MAD}} \cdot 33 \rightarrow \ 153 \leq \frac{op}{mblq} \leq 297$$

2.5.6. Búsqueda en cuatro pasos (FSS/4SS)

El algoritmo *Four Step Search* (FSS) es una mejora del TSS en términos de menor coste computacional y similar al NTSS. Se caracteriza por el uso de dos patrones de búsqueda sobre 9 bloques (puntos) iniciales con un *step* de 2.

La descripción del algoritmo es la siguiente:

1) Primeramente se fija un paso de valor 2, y se toman nueve bloques en torno al centro de la ventana de búsqueda. Se calcula el error correspondiente a cada uno (MAD) para hallar el que supone una relación mejor con el macrobloque de referencia. Si dicho bloque resulta ser el central, se siguen las instrucciones descritas en el punto 4. Si no lo es, se continúa con el punto 2.

2) La nueva referencia pasa a ser el punto de mínimo MAD de la iteración anterior, manteniéndose el paso igual a 2. El patrón de búsqueda depende del punto de mínimo MSE de la etapa anterior.

3) Ejecutando estos pasos se llegará a un momento en el que el punto más optimo no se encuentre entre los 8 puntos del marco de búsqueda, sino que se corresponda con el central.

4) Se reduce el *step* a 1 y se afina la búsqueda sobre los 8 puntos alrededor del obtenido en el paso 3. La distancia entre ese punto y el origen es el MV.

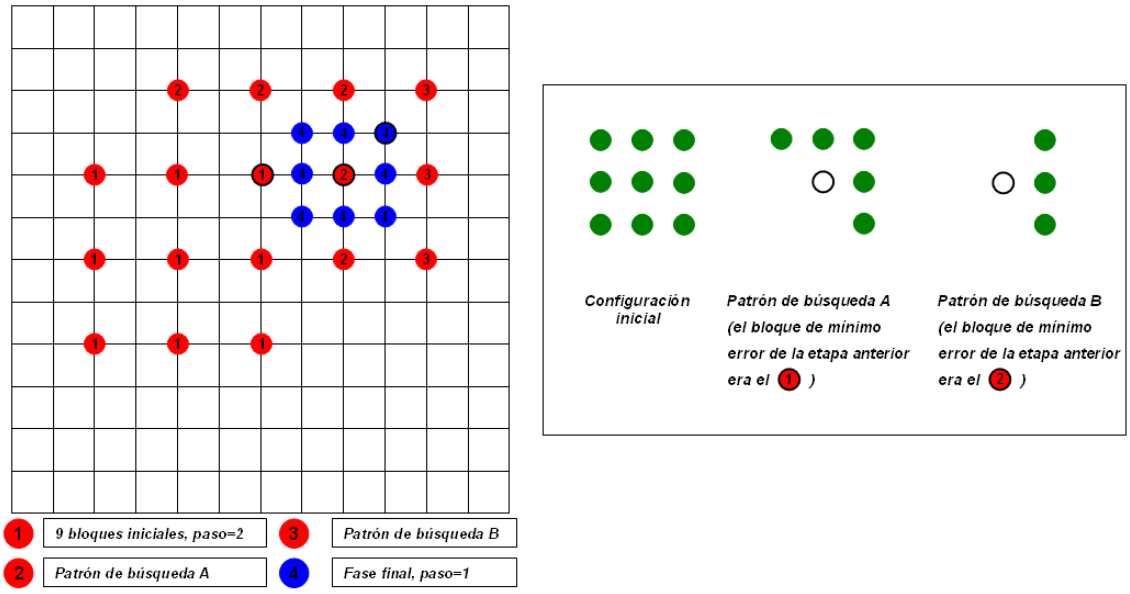


Figura 9: Pasos de búsqueda FSS y último paso

Es importante notar que este algoritmo evalúa la función de coste 17 veces en el mejor de los casos y 27 veces en el peor de los casos. Esto supone una pequeña mejora con respecto al NTSS, que se obtenían 17 comparaciones en el mejor caso y 33 en el peor. [11]

De nuevo, realizando el cálculo del número de operaciones necesario por cada macrobloque, se obtiene:

$$n_{op_{MAD}} \cdot 17 \leq \frac{op}{mblq} \leq n_{op_{MAD}} \cdot 27 \rightarrow \ 153 \leq \frac{op}{mblq} \leq 243$$

2.5.7. Búsqueda en diamante (DS)

Este algoritmo es muy similar al algoritmo FSS, solo que en lugar de utilizar un patrón de búsqueda cuadrado emplea un rombo y además no limita el número de pasos que puede dar el algoritmo.

Se emplean dos tipos diferentes de patrones:

1) *Large Diamond Search Pattern* (LDSP)

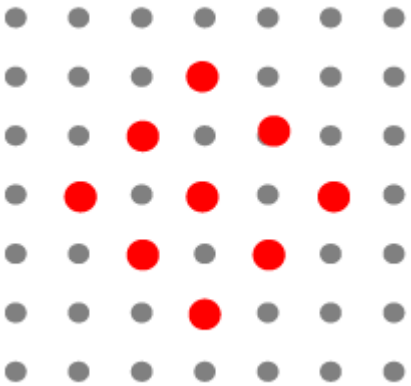


Figura 10: LDSP [11]

2) *Small Diamond Search Pattern* (SDSP)

Figura 11: SDSP [11]

Este algoritmo se aprovecha del hecho de que el movimiento general de un *frame* es normalmente coherente, esto es, que si los macro bloques que hay alrededor del macro bloque actual (del que queremos estimar su vector de movimiento) se han movido en una dirección en particular, entonces habrá una alta probabilidad de que el macro bloque actual también se haya movido en esa dirección. [13]

Los pasos de ejecución del algoritmo se detallan a continuación:

- LDSP

  1) Con un *step* S=2 y centrado en el origen se emplea el patrón de búsqueda en rombo. Uno de los 9 puntos cumple el mínimo MAD.

  Si ese punto es el central, se salta al SDSP.

  2) Si es uno de los otros 8 restantes, se considera como el nuevo origen y se repite el LDSP.

- SDSP

  1) El punto central obtenido en el LDSP es el nuevo origen. Se reduce el *step* a la mitad y se repite el procedimiento de búsqueda con el patrón de rombo pequeño. La diferencia entre el punto de menor MAD y el origen al inicio del algoritmo (0,0) es el *motion vector*.

Una consideración importante es la diferencia del desplazamiento de la venta de búsqueda con respecto a cómo se hacía en el NTSS y en el FSS. En ambos algoritmos, el número de comparaciones en el paso final dependía de la nueva posición del origen: 3 comparaciones en el caso de los puntos laterales y superiores, 5 en el caso de las esquinas. Como se muestra en la figura 12, en la búsqueda en diamante sucede lo contrario, ya que el patrón de búsqueda esta rotado 90 grados. [14]

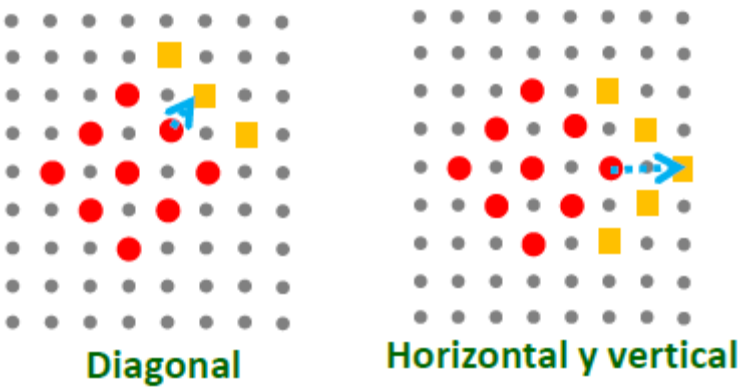


Figura 12: Último paso del la búsqueda DS [11]

3. DESARROLLO DE LA SOLUCIÓN

3.1. Arquitectura del sistema

En la implementación de la solución se sigue el flujo de diseño que muestra la figura 14:

1) Captura de las imágenes: Haciendo uso de la cámara se captura el video en tiempo real, o bien se trabaja sobre un video previamente grabado, sacándolo de la memoria.

2) División en macrobloques: Se subdivide cada *frame* en macrobloques de 16x16 para su análisis posterior haciendo uso de un algoritmo.

3) Implementación del algoritmo: Se devuelve como respuesta el MV(i,j) del macrobloque analizado en la imagen anterior.

4) Solución: Presentación del resultado.

En el apartado 3) se van a implementar dos algoritmos: una búsqueda secuencial y una búsqueda en tres pasos, con el objetivo de comparar y analizar los diferentes resultados obtenidos en cuanto al tiempo de ejecución y la precisión del sistema.

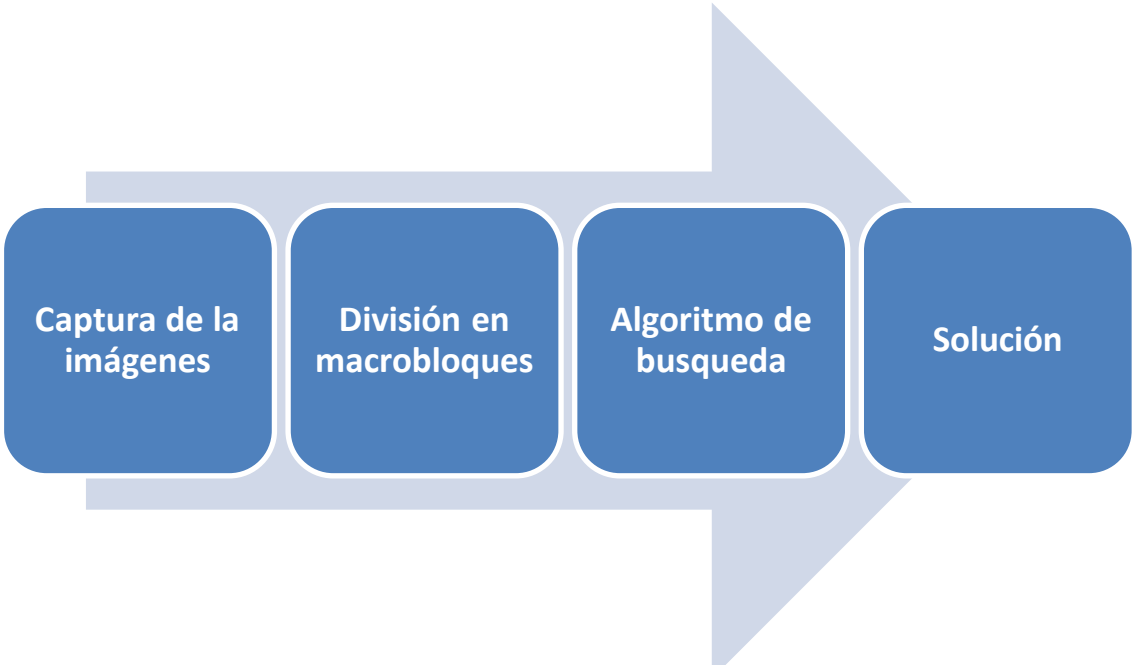


Figura 14: Diagrama del funcionamiento del sistema

3.1.1. Captura de las imágenes

La obtención de las imágenes puede realizarse de dos maneras diferentes:

1) Empleando una cámara y procesando la información en tiempo real.

2) Extrayendo la información de un archivo de video almacenado en el sistema.

En ambos casos, la imagen obtenida se redimensiona hasta un tamaño CIF: 352 pixeles de largo por 288 pixeles de ancho, aunque podrían emplearse tamaños de imagen superiores.

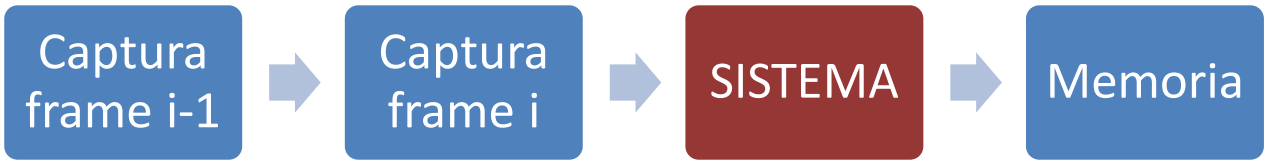


Figura 15: Diagrama de la captura de imágenes

En primer lugar se realiza la captura de un *frame* al que llamaremos *frame* i-1. Tras su captura, se procede a la obtención de la siguiente imagen, llamada *frame* i. Una vez que hay disponibles dos fotogramas del video se lleva la información, que ha sido almacenada, a la siguiente etapa del sistema, la división en macrobloques.

Cuando se ha realizado el cómputo, se hace un volcado de la memoria de un *frame* en la posición de memoria de otro. Esto se hace ya que no se deben capturar dos frames nuevos cada vez que se calcula la estimación de movimiento, sino que cada imagen es utilizada dos veces dentro del sistema. Sin este procedimiento, la estimación de movimiento solo se realizaría entre parejas de fotogramas consecutivas, y no entre todos los fotogramas.

En este volcado de la memoria se sustituye la imagen i-1 por la imagen i, de forma que la imagen (anteriormente denominada como i) se convierte en la más antigua dentro del sistema, capturándose una imagen nueva cada vez y comprando ambas.

Debido a la forma de capturar el video, este se presenta en un formato YUV. YUV es un espacio de color empleado como parte de un sistema de procesamiento de imagen o video a color. La codificación de los datos en este espacio de color se realiza teniendo en cuenta la percepción humana, siendo más efectivo que un modelo RGB.

Existen espacios de color muy similares a YUV, como YCbCr, que es el empleado en este caso. [16]

Mientras que en RGB las imágenes son la composición de tres variables de color, en YCbCr solo existen dos cromas.

1) Y representa la luminancia o luma de la imagen, es decir, el nivel de intensidad dentro de cada pixel. Una imagen que solo muestre la Y está en blanco y negro.

2) Cb y Cr representa las crominancias o cromas azules y rojas, respectivamente. Estas cromas son una diferencia de valores con respecto al color verde, de forma que no es necesario codificar la mezcla de los tres colores primarios con tres parámetros, solo con dos. El verde se emplea como referencia ya que el ojo humano es más sensible a el que al rojo o al azul, y por lo tanto, la información que representa el verde ocuparía más espacio. [17]

Para la estimación de movimiento, los valores de color propiamente dichos de cada pixel no son importantes, ya que esta estimación se puede realizar sin ellos. Su inclusión en los cálculos aumenta el gasto computacional y no aporta una gran diferencia con respecto al uso de la intensidad de los pixeles.

3.1.2. División en macrobloques

Este paso intermedio busca fraccionar las imágenes en secciones más pequeñas para poder procesarlas individualmente. Cada una de estas secciones recibe el nombre de macrobloque, y tanto su tamaño como su forma pueden ser variables.

Habitualmente, son agrupaciones de cuadradas de pixeles de tamaño NxN, donde a N típicamente se le asigna o bien el valor 8 o bien 16. Sin embargo, dentro de un mismo estimador de movimiento el tamaño de los macrobloques no tiene por qué ser constante, sino que puede variar en función de las necesidades del codificador.

Esta división es fundamental a la hora de realizar la estimación de movimiento, ya que cada uno de estos macrobloques ocupa una posición dentro de la imagen, y se debe encontrar su desplazamiento en otras imágenes, por lo que divisiones pequeñas otorgan mayor tiempo de computo, pero una mayor precisión y variedad de información en los resultados.

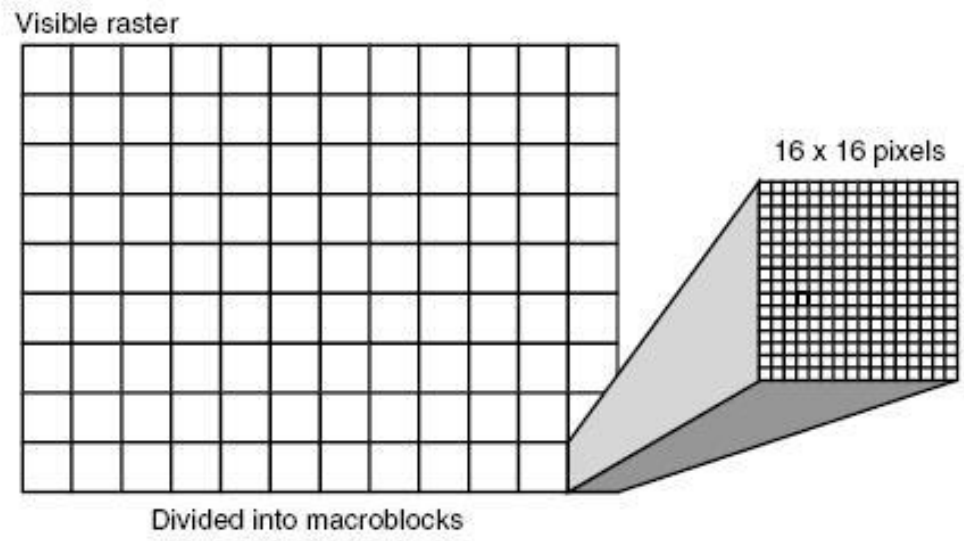


Figura 16: División de una imagen en macrobloques

Un detalle importante es que para una imagen de tamaño CIF, el número de macrobloques de 16x16 da un valor entero.

$$N_{macro(horizontal)} = \frac{352}{16} = 22 \qquad N_{macro(vertical)} = \frac{288}{16} = 18$$

En esta etapa también se lleva a cabo la creación de la ventana de búsqueda del macrobloque objetivo.

En el *frame* i-1 se va a utilizar cada uno de los macrobloques como referencia para hacer la búsqueda en el *frame* i. Dentro de este ultimo *frame* se debe expandir una ventana de búsqueda; es decir, empleando como centro de esa ventana el macrobloque que posee las mismas coordenadas dentro del fotograma i-1, se crea un marco de pixeles a su alrededor, conocido como *search window*, de tamaño variable. En este sistema el tamaño de esa ventana es de 3Nx3N.

Tanto el macrobloque de referencia como la ventana de búsqueda son parámetros que recibe el algoritmo.

3.1.3. Algoritmo de búsqueda

El algoritmo que se emplea para calcular los vectores de movimiento es la parte fundamental del estimador de movimiento, ya que supone la mayoría del gasto computacional del sistema.

Cada algoritmo implementado recibe como entrada el macrobloque sobre el que se debe calcular el vector de movimiento y una ventana de búsqueda.

Figura 17: Diagrama del funcionamiento genérico de un algoritmo de búsqueda

Como se puede ver en la figura 17, independientemente de cómo esté implementado el algoritmo y cual sea su forma de elegir que posiciones dentro de la ventana de búsqueda seleccionar, los pasos que realiza son siempre los mismos:

1) Primero selecciona un punto dentro de la ventana de búsqueda que sirve como referencia y a partir del cual se expande la ventana de búsqueda de 16x16 pixeles.

2) Opera con el MAD, comparando pixel a pixel el macrobloque de referencia con el seleccionado en la ventana de búsqueda. El MAD resta el valor de dos pixeles en posiciones correspondientes en ambos macrobloques, y su valor absoluto es añadido a las diferencias de los otros pixeles, y todo ello dividido entre el numero de pixeles (N^2). De esta forma se calcula el error medio por cada pixel.

3) El MAD calculado no tiene porque mejorar (reducir el error por pixel) con respecto a iteraciones anteriores del algoritmo, por lo que el valor del mejor MAD se mantiene. Si el nuevo MAD es mejor que el anterior, pasa a ser el mejor MAD hasta ese momento.

4) Si se han de realizar más iteraciones del algoritmo, se vuelve al punto 1. Si el algoritmo ha finalizado, devuelve el vector de movimiento asociado a la posición del mejor MAD.

3.1.3.1. Implementación de la búsqueda secuencial

Uno de los algoritmos que se ha implementado en el sistema es el de la búsqueda secuencial. Este tipo de búsqueda es muy efectiva, pero su tiempo de computación es demasiado elevado, con lo que su uso se descarta para aplicaciones de tiempo real.

Las características de esta implementación son las siguientes:

- ✓ Se emplea una forma de macrobloque constante
- ✓ El tamaño de los macrobloques también es constante e igual a 16
- ✓ El tamaño de la ventana de búsqueda es de 3Nx3N
- ✓ La operación a emplear es el MAD
- ✓ La referencia de los macrobloques es la esquina superior izquierda

Se implementa empleando dos parejas de ciclos:

1) La primera recorrerá la ventana de búsqueda de 0 a 2·N en horizontal y de 0 a 2·N.

2) La segunda recorrerá cada macrobloque de 0 a 15 en horizontal y de 0 a 15 en vertical.

Una representación visual de los pasos del algoritmo se muestra a continuación con las figuras 18, 19 y 20. El punto azul marca la posición de los iteradores que recorren los diferentes puntos de la ventana de búsqueda mientras que las zonas ensombrecidas en gris son el cálculo del MAD.

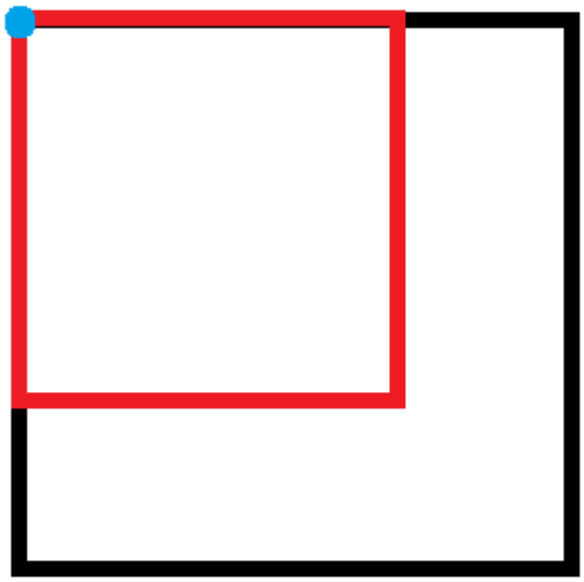


Figura 18: Primera iteración de una búsqueda exhaustiva

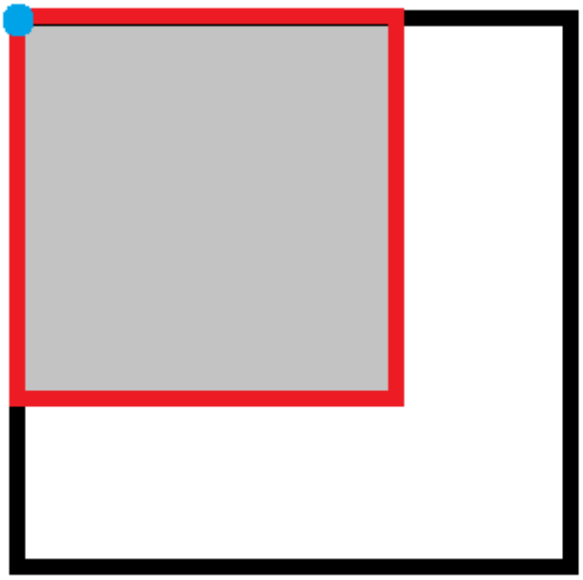


Figura 19: Calculo del MAD en la primera iteración de una búsqueda exhaustiva

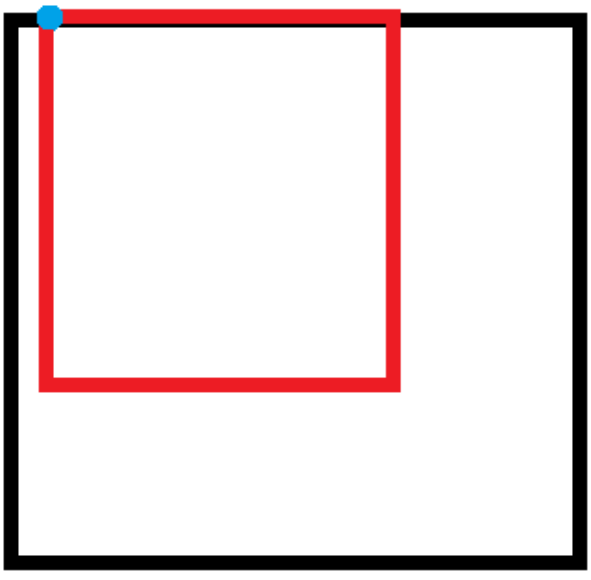


Figura 20: Segunda iteración de una búsqueda exhaustiva

3.1.3.2. Implementación de TSS

El *Three Step Search* es uno de los algoritmos de búsqueda más empleados debido a su fidelidad a la hora de obtener una solución y el bajo número de operaciones que debe realizar. Debido a estas condiciones, puede ser incluido en aplicaciones en tiempo real.

Las características de esta implementación son las siguientes:

- ✓ Se emplea una forma de macrobloque constante
- ✓ El tamaño de los macrobloques también es constante e igual a 16
- ✓ El tamaño de la ventana de búsqueda es de 3Nx3N
- ✓ La operación a emplear es el MAD
- ✓ La referencia de los macrobloques es la esquina superior izquierda

Se implementa empleando dos parejas de ciclos:

1) La primera recorrerá la ventana de búsqueda de ref_i-step a ref_i+step en horizontal y de de ref_j-step a ref_j+step.

2) La segunda recorrerá cada macrobloque de 0 a 15 en horizontal y de 0 a 15 en vertical.

Ambas referencias (ref_i y ref_j) son incializadas al valor del tamaño de macrobloque, debido a que la ventana es de 3Nx3N y se toma como referencia la esquina superior izquierda, el macrobloque central con las misma coordenadas que el macrobloque que se está buscando ocupa las coordenadas dentro de la ventana:

$$N \geq i \geq N - 1 \quad ; \quad N \geq j \geq N - 1$$

La primera pareja de ciclos selecciona 9 posiciones dentro de la ventana de búsqueda, separadas una cierta cantidad de pixeles, llamada *step* o paso. El valor inicial del *step* en la primera iteración se corresponde con el tamaño del macrobloque, N.

Cuando dentro de esas 9 posiciones se encuentra la que minimiza el MAD, esas coordenadas i_1, j_1 pasan a ser las nuevas referencias de i y de j, de forma que el centro de la nueva ventana de búsqueda se sitúa en i_1, j_1.

Esto implica que en una zona cercana al punto que minimiza el MAD se encuentra el macrobloque que se está buscando, y se reduce el *step* a la mitad para hacer una búsqueda mas precisa en esa zona. En el resto de puntos no se hacen más búsquedas ya que los valores del MAD son más elevados, y es improbable que el macrobloque solución se encuentre allí.

Este procedimiento se repite hasta que el paso sea menor o igual a 1.

Una representación visual de los pasos del algoritmo se muestra a continuación con las figuras 21, 22 y 23. Los puntos rojos/verdes/azules marcan las 9 posiciones seleccionadas por los dos ciclos, y el círculo rodeado representa el mejor de esos puntos (mínimo MAD).

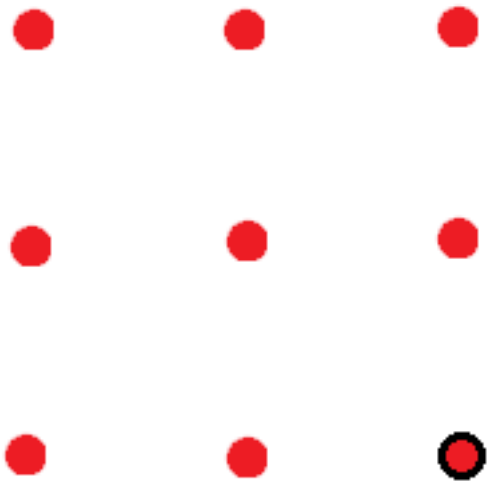


Figura 21: Primera iteración del TSS y mejor MAD

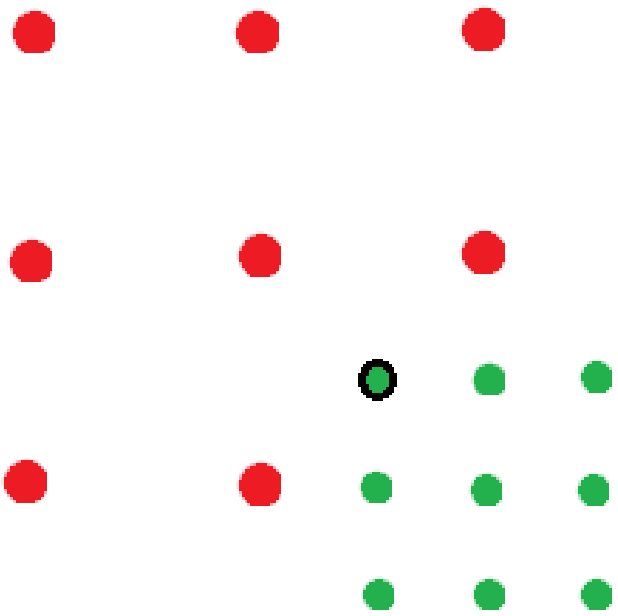


Figura 22: Segunda iteración del TSS y mejor MAD

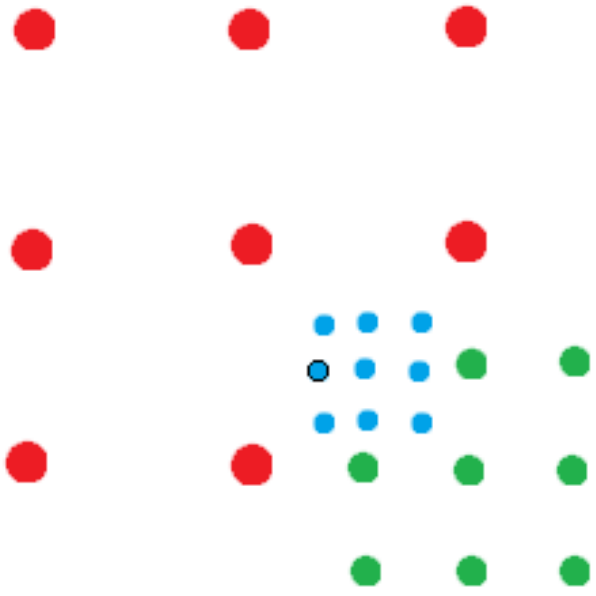


Figura 23: Tercera iteración del TSS y mejor MAD

3.1.4. Solución

Como imagen de salida para un video, en este caso tomado desde archivo, se muestran el *background*(parte de la imagen que no cambia entre fotogramas, fondo) y el *foreground* (parte de la imagen que cambia entre fotogramas).

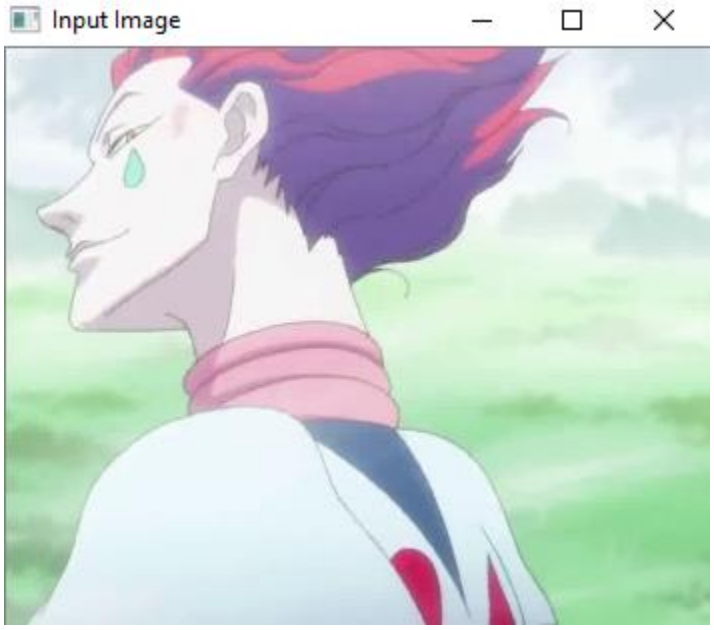


Figura 24: Fotograma i-1

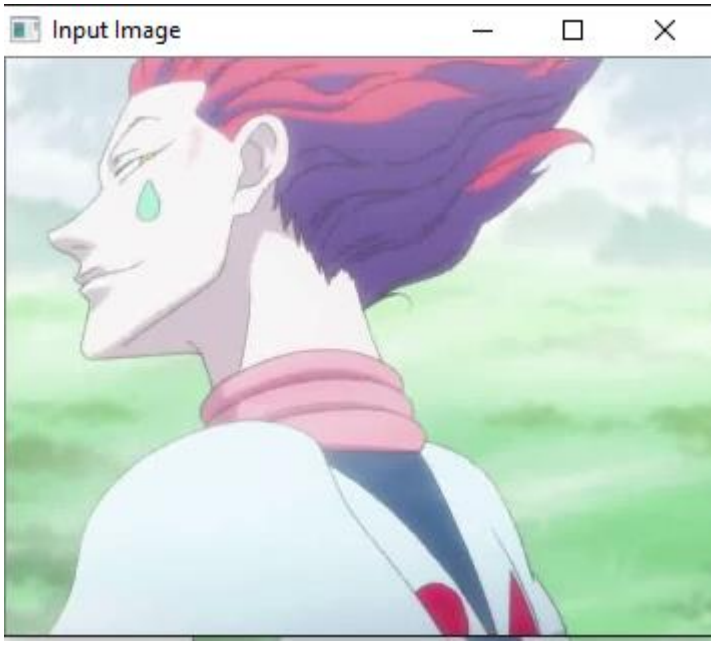


Figura 25: Fotograma i

Comparando ambos fotogramas se aprecia un movimiento en la cabeza del personaje. Realizadas estas capturas empleando el algoritmo TSS, se muestran a continuación el *background* y el *foreground* del video. En la figura 26 el fondo permanece inalterado, mientras que se representan con macrobloques blancos aquellos que se han desplazado, correspondiéndose con la silueta del personaje. Es en la figura 27 donde se muestra el cambio con respecto a la imagen anterior, pudiendo verse el rostro y cuerpo.

Figura 26: *Background*

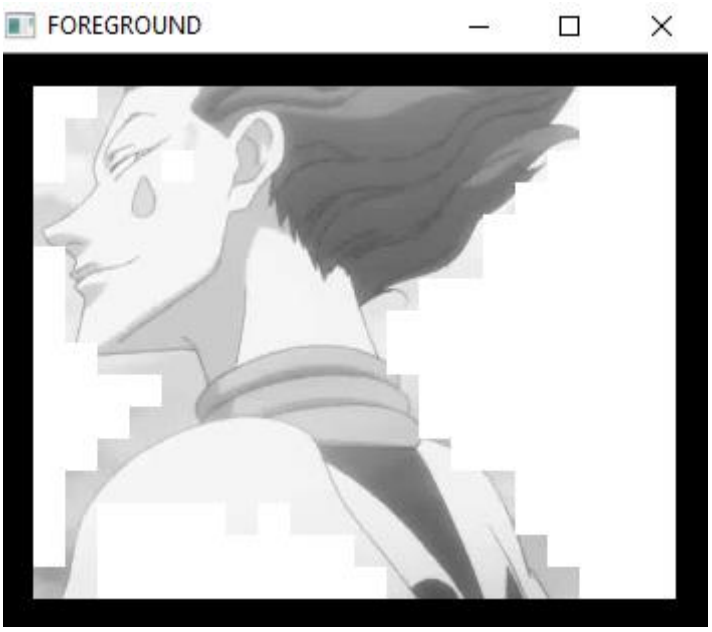


Figura 27: *Foreground*

3.2. Librerías de procesado de imagen

3.2.1. Entorno de programación: OpenCV

OpenCV (*Open Source Computer Vision*) es una librería de código abierto desarrollada por Intel en 1999. Su principal objetivo es el de proveer una infraestructura de visión por computador fácil de utilizar que ayuda a desarrollar aplicaciones de mayor complejidad rápidamente. Para ello cuenta con más de 500 funciones, pudiendo ser utilizadas en aéreas tan diversas como calibración de cámaras, robótica, control de procesos industriales, seguridad, etc. [15]

Una de sus principales ventajas es que cuenta con una licencia de código abierto BSD (*Berkeley Software Distribution*), lo que supone que el producto originado a partir del uso de OpenCV no está sujeto a pagos de licencias ni derechos de autor.

Otra de sus principales ventajas es que es altamente eficiente en cuanto al uso de recursos computacionales y claramente enfocado a las aplicaciones en tiempo real.

OpenCV está escrita en C y C++, teniendo soporte sobre sistemas operativos actuales muy extendidos, tales como Linux, Windows o MacOS, con interfaces para C, C++, Phyton, Java o MATLAB.

Como fuente de código abierto cuenta con una gran comunidad de usuarios detrás.

Figura 13: Logo de OpenCV

OpenCV tiene una estructura modular, con lo que incluye varias librerías, tanto estáticas como dinámicas. Algunos de esos módulos empleados en este proyecto se describen brevemente a continuación:

- core: Es el módulo básico de OpenCV. Incluye las estructuras de datos básicas, como el array multidimensional Mat, para el manejo de imágenes, y las funciones básicas de procesamiento de imágenes, empleadas por otros modulos como *highgui*.

- imgproc: Incluye algoritmos básicos de procesado de imágenes, incluyendo filtrado de imágenes, transformado de imágenes, etc.

- highgui: Este módulo provee interfaz de usuario, codecs de imagen y video, capacidad de capturar imagen y video, etc.

- video: Un módulo de análisis de video que incluye obtención de primeros planos, algoritmos de seguimiento de objetos o estimación de movimiento.

- videoio: Modulo que provee una interfaz sencilla para leer video de una cámara o fichero y escribir almacenar video en un fichero.

3.2.2. Librería propia: VideoUtils

A pesar de que OpenCV provee de todas las funciones necesarias para la captura y manejo de video, en este proyecto se ha incluido la librería *VideoUtils*. Esta librería hace uso de funciones incluidas dentro de OpenCV, siguiendo una notación y terminología similares, pero implementando las funcionalidades de manera ligeramente diferente.

4. ANÁLISIS DE RESULTADOS

Con la ejecución del sistema empleando los dos algoritmos se pueden apreciar las diferencias plasmadas en la sección 3 con respecto a los algoritmos

- La búsqueda secuencial o exhaustiva es mucho más pesada, dando una imagen de salida de tan solo 2 fps. Esto se debe a que el tiempo entre llegadas de los frames (aproximadamente unos 20 fps) es demasiado pequeño para realizar todo el procesado.
- La búsqueda en tres pasos emplea menor tiempo de cómputo, y para una misma tasa de llegada de las imágenes, muestra una salida a 18 fps.
- Las diferencias de cálculo de los vectores de movimiento entre algoritmos son mínimas.
- A pesar de que la búsqueda exhaustiva no puede fallar, su tiempo de ejecución no justifica la precisión de los resultados.
- El espacio (en líneas de código) que ocupa cada algoritmo es similar, así como el uso de recursos que emplea (accesos a memoria, número de variables, etc).

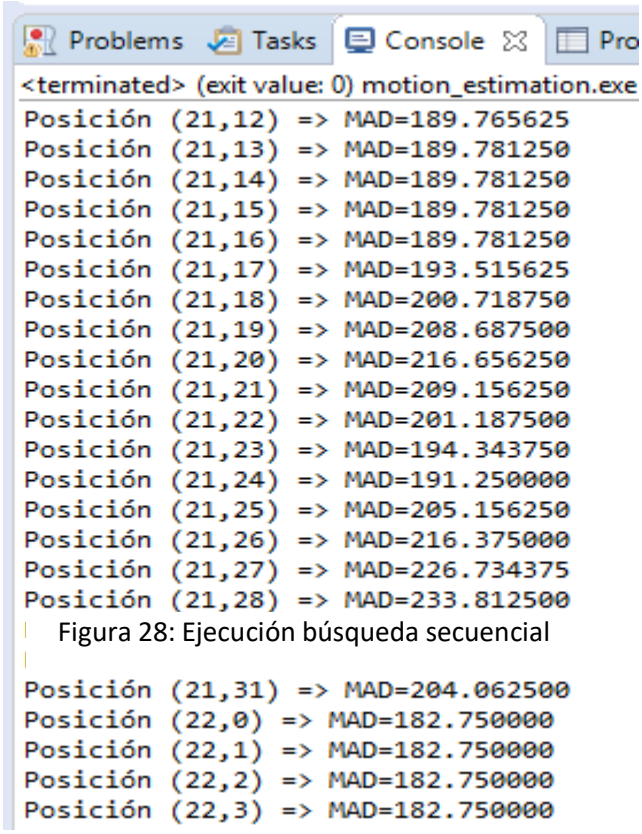


Figura 28: Ejecución búsqueda secuencial

Como puede verse en la figura 24, la implementación de la búsqueda secuencial realiza todas y cada una de las búsquedas posibles dentro de la *search window*, obteniendo en muchos casos valores muy similares o iguales en posiciones cercanas, muy lejanos al valor de referencia, y cuyo cálculo podría ser evitado.

Por otro lado, en la figura 25 se observa uno de los pasos de ejecución del algoritmo TSS: la selección de los puntos a analizar en cada iteración y la obtención del MAD en cada uno de ellos. Una vez finalizada cada iteración se muestra el mejor caso de MAD de los calculados, que pasa a ser la referencia para iteraciones siguientes. Así se evitan los cálculos innecesarios en otras zonas del macrobloque (p.e: zona 6,6).

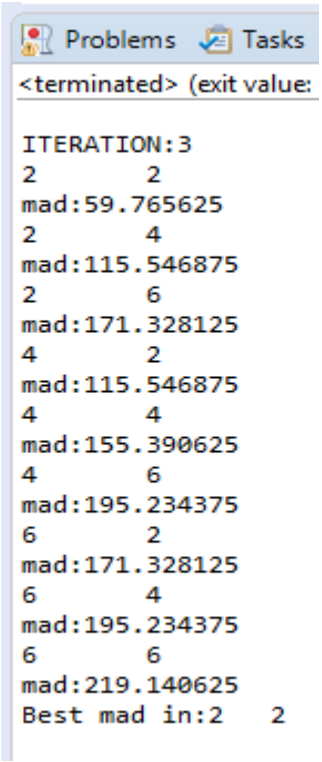
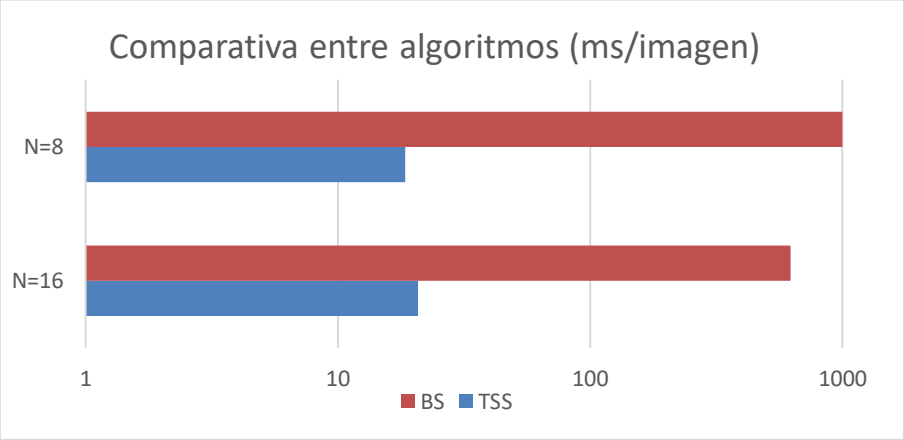


Figura 29: Ejecución TSS

Sabiendo que una búsqueda secuencial es la más precisa, ya que analiza todas las combinaciones posibles, su coste no justifica el resultado obtenido, debido a que las aproximaciones del algoritmo TSS generan unos "*motion vector*" con el mismo nivel de precisión. Esas diferencias se muestran en la siguiente gráfica:



Los resultados empíricos muestran que para un tamaño de macrobloque de N=8, el algoritmo de tres pasos emplea 65 µs por cada macrobloque (20.8 ms por cada imagen), mientras que si se reduce dicho tamaño a N=8, sus prestaciones incluso mejoran un poco, alcanzándose los 13 µs por macrobloque (18.564 ms en todo el *frame*).

Por el contrario, los valores temporales de la búsqueda secuencial se disparan hasta los 0.624 s para un tamaño N=16, empeorándose aún más si cabe con N=8, cuyo tiempo de procesado por imagen superaría el segundo de ejecución.

5. CONCLUSIONES

En este trabajo se ha llevado a cabo la implementación de un sistema software para el cálculo de la estimación de movimiento en tiempo real sobre un video.

En primer lugar se ha realizado la captura del video, a través de una cámara estándar, empleando un lenguaje de programación de alto nivel (C++), apoyándose en la librería de código abierto OpenCV para el procesado de imagen.

A continuación se almacena cada fotograma del video, haciendo uso únicamente de la información de las luminancias (Y), ya que la información sigue un esquema de color YUV.

Una vez extraídas las luminancias, se procede a fraccionar la imagen en macrobloques, consideradas como unidades básicas para la estimación de movimiento. Esta división se lleva un paso más allá expandiendo una *"search window"* o ventana de búsqueda sobre la que se busca el macrobloque objetivo.

Esa información preparada se envía al siguiente modulo, que implementa el algoritmo de estimación de movimiento y devuelve el vector de movimiento. Sobre este modulo se han diseñado dos versiones: una implementa una búsqueda secuencial y la otra una búsqueda en tres pasos. Los resultados obtenidos en ambos estudios son muy similares a nivel de precisión, pero el tiempo de computo de la búsqueda secuencial es mucho mayor.

Con una reducción del tamaño de macrobloque se eleva el número de cálculos a realizar, pero estos son mas ligeros, por lo que en una búsqueda logarítmica se mejora el resultado.

Por último, la comprobación visual del funcionamiento del sistema es bastante fiable, ya que se muestran en tiempo real la salida de video procesada en forma de *"background"* o fondo de cada imagen que permanece inalterado, y *"foreground"* o secciones del video que han experimentado un desplazamiento entre frames consecutivos.

Estas salidas han sido refinadas haciendo uso de un umbral que coteja el macrobloque de referencia y el macrobloque con el que comparte posición en la ventana de búsqueda con el macrobloque de referencia y el punto de mejor correlación, permitiendo eliminar el ruido que se introduce en la imagen y mostrando unas salidas más claras.

6. REFERENCIAS

[1] *Estimación de movimiento*

https://es.wikipedia.org/wiki/Estimaci%C3%B3n_de_movimiento

[2] AXIS Communications

Video Compression

https://www.axis.com/es/es/learning/web-articles/technical-guide-to-network- video/video-compression

[3] Escuela de Ingenieros Industriales de la Universidad de Castilla La-Mancha

Compresión de la información de video

http://edii.uclm.es/~jmlova/Archivos/VD/Archivos/VdCompresion.pdf

[4] PABLO PEDRO SÁNCHEZ ESPESO

Técnicas básicas de compresión de video

Sistemas Electrónicos Multimedia, UNICAN

[5] ZE-NIAN LI, MARK S. DREW

Fundamentals of Multimedia

School of Computing Science, Simon Fraser University, 2004

[6] *Block matching*

https://es.wikipedia.org/wiki/Block_matching

[7] IVÁN LOPEZ ESPEJO, JONATHAN PRADOS GARZÓN

Estimación de movimiento

Marzo 2012

[8] BÉATRICE PESQUET-POPESCU, MARCO CAGNAZZO, FRÉDÉRIC DUFAUX

Motion Estimation Techniques

TELECOM ParisTech

[9] DISHA D. BHAVSAR, RAHUL N. GONAWALA

Three Step Search method for block matching algorithm

Abril 2014

[10] RENXING LI, BING ZENG, MING L. LIOU

A New Three-Step Search Algorithm for Block Motion Estimation

IEEE Transactions on Circuits and Systems for Video Technology, vol 4, nº 4

Agosto 1994

[11] JUAN A. MICHELL MARTÍN, GUSTAVO A. RUIZ ROBREDO

Compresión de video. Estimación de movimiento y compensación: conceptos básicos y algoritmos

https://ocw.unican.es/pluginfile.php/171/course/section/75/Bloque_2._Tema_3._Esti macion_de_movimiento_y_compensacion.pdf

[12] LAI-MAN PO, WING-CHUNG MA

A Novel Four-Step Search Algorithm for Fast Block Motion Estimation

IEEE Transactions on Circuits and Systems for Video Technology, vol 6, nº 3

Junio 1996

[13] SHAN ZU, KAI-KUANG MA

A New Diamond Search Algorithm for Fast Block Matching Motion Estimation

School of Electrical and Electronic Engineering, Nanyang Technology University, Singapore

Septiembre 1997

[14] JO YEW THAM, SURENDRA RANGANATH, MAITREYA RANGANATH, ASHRAF ALI KASSIM

A Novel Unrestricted Center-Biased Diamond Search Algorithm for Block Motion Estimation

IEEE Transactions on Circuits and Systems for Video Technology, vol 8, nº 4

Agosto 1998

[15] ADRIÁN RODRIGUEZ BAZAGA

OpenCV: Librería de Visión por Computador

Oficina de Software Libre de la Universidad de La Laguna

https://osl.ull.es/software-libre/opencv-libreria-vision-computador/

18 Agosto 2015

[16] Grupo de Redes de Computadores, Universidad Politécnica de Valencia

Codificación y compresión de video

http://www.grc.upv.es/docencia/tdm/slides/T6_video.pdf

[17] EDUARDO ALARCÓN G., LUIS ENRIQUE ESPINOZA S.

Algoritmos de compresión de video

Departamento de Electrónica de la Universidad Técnica Federico Santa María, Chile

http://profesores.elo.utfsm.cl/~agv/elo330/2s06/projects/EspinozaAlarcon/ELO- 330%20Algoritmos%20de%20compresion%20de%20Video.html