TV: TeleVisión – Plan 2010

Estimación y compensación de movimiento

Estimación y compensación de movimiento

1. Introducción
2. Estimación por bloques
3. Introducción al ajuste de bloques
    1. Criterios de selección
    2. Tamaño del bloque
4. Algoritmos de ajuste de bloques
    1. Búsqueda exhaustiva
    2. Búsqueda logarítmica
    3. Búsqueda jerárquica

Estimación y compensación de movimiento

5. Algoritmos piramidales
6. Estimación con precisión fraccionaria
    1. Algoritmo en 2 pasos
7. Métodos avanzados de compensación de movimiento
    1. Compensación por solapamiento de bloques
    2. Compensación por interpolación de la rejilla de control

1. Introducción

- **Objetivo:**
  - Usar la predicción temporal para reducir la redundancia temporal existente en una secuencia de imágenes.

1. **Estimación de movimiento:**
   - Objetivo → Establecer correspondencias entre píxeles de diferentes imágenes.
   - Da lugar a vectores de movimiento, que se transmiten.
   - Sólo se realiza en el codificador.
2. **Compensación de movimiento:**
   - Objetivo → Compensar, respecto a la estimación, el desplazamiento experimentado por los objetos presentes en la escena de una imagen a otra.
   - Da lugar a imágenes de error de predicción, que se transmiten.
   - Se realiza tanto en el codificador como en el decodificador.

Imágenes de error → Menor redundancia temporal → Mayor compresión

1. Introducción

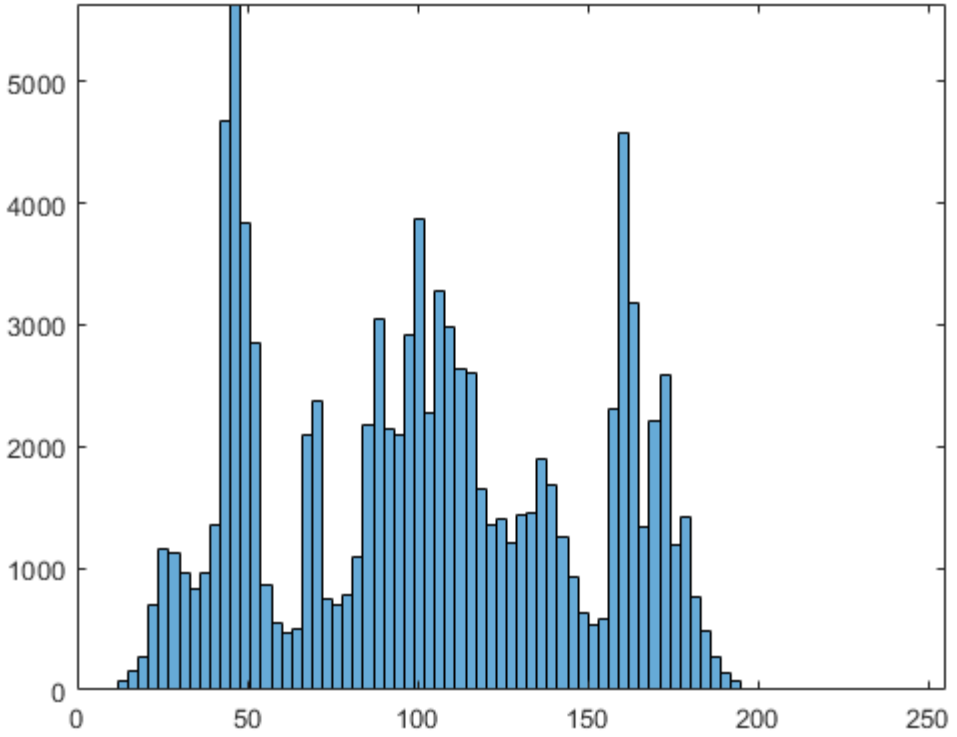
- **Ejemplo:**

Supongamos que queremos codificar esta secuencia.

1. Introducción

- **Ejemplo:**

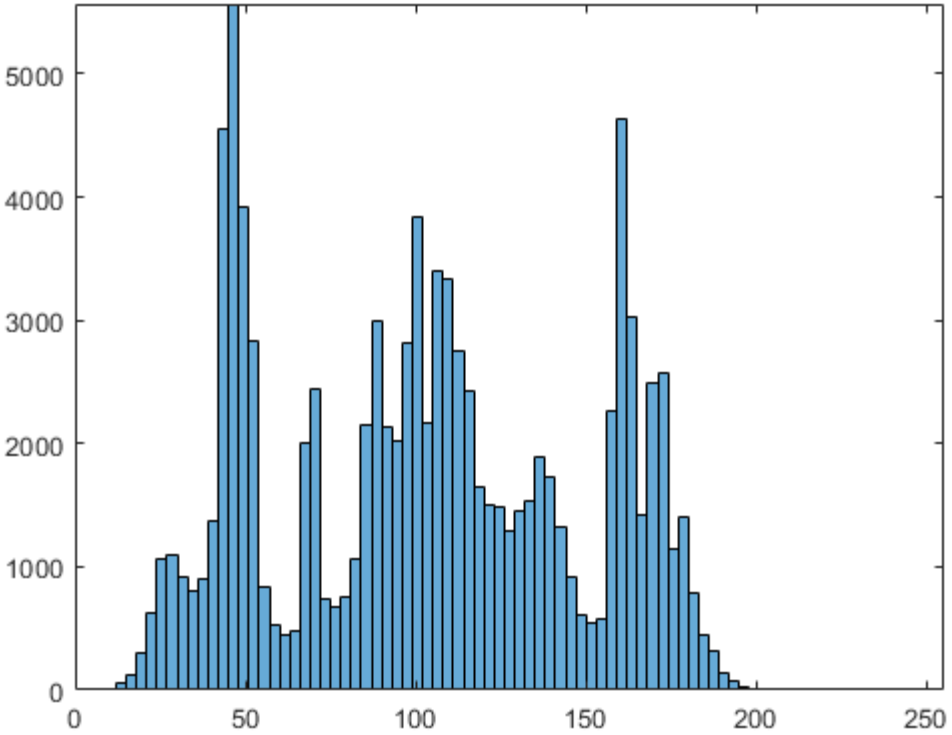
Primer frame



1. Introducción

- **Ejemplo:**

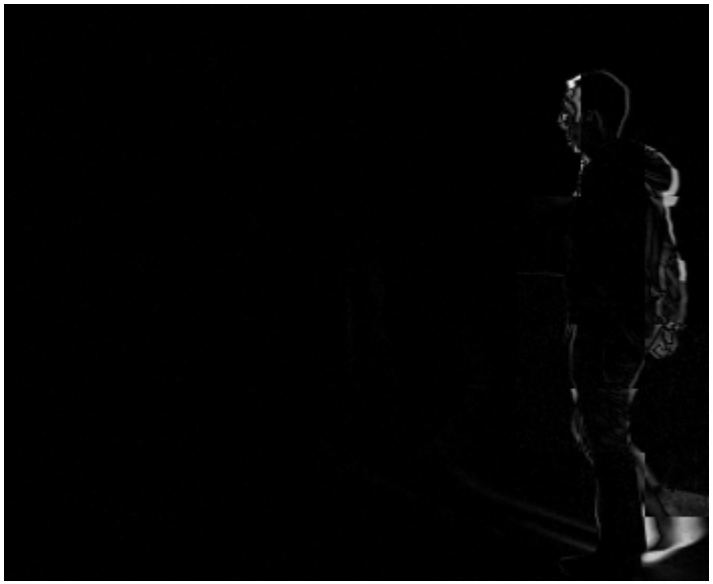
Segundo frame

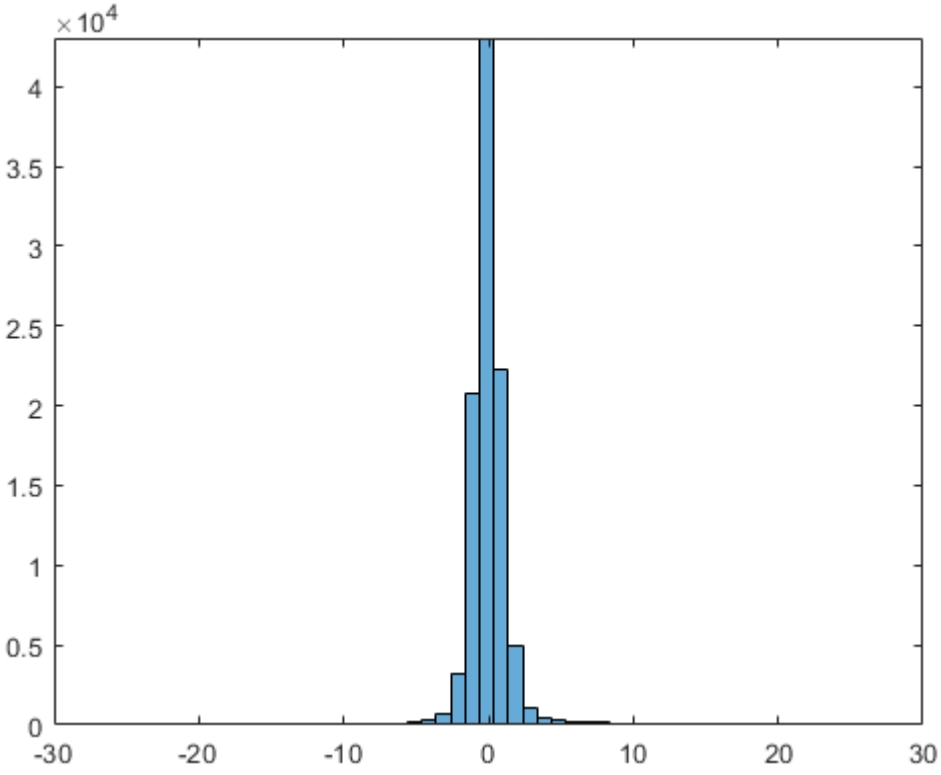


1. Introducción

- **Ejemplo:**

Imagen de error resultante de comparar las dos anteriores

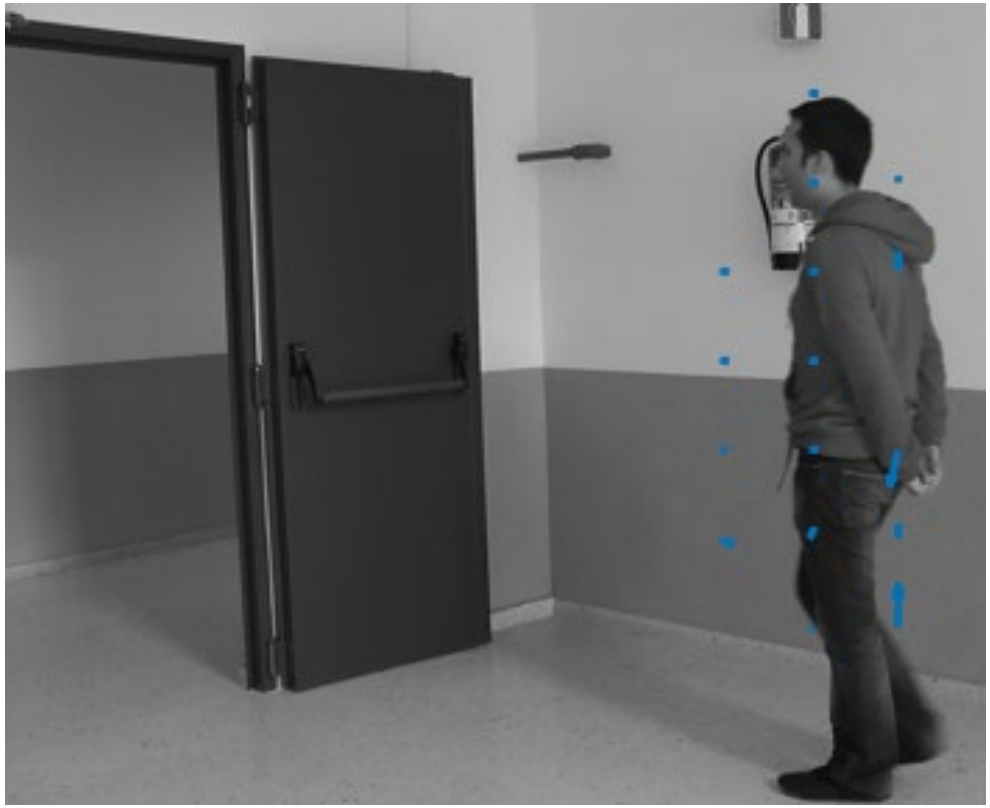




1. Introducción

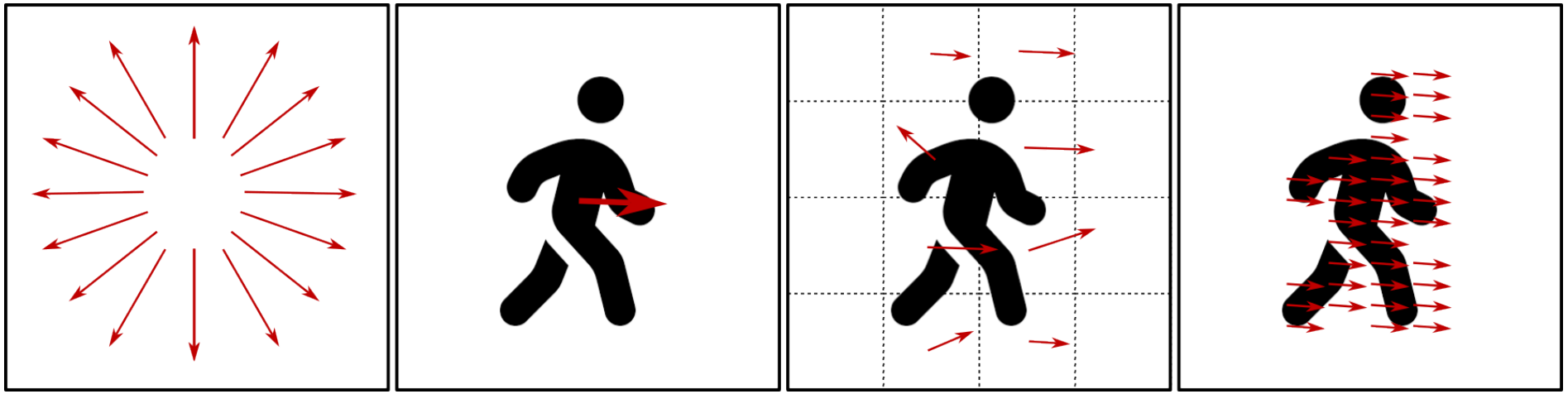
- **Ejemplo:**

Vectores de movimiento



1. Introducción

- La etapa de estimación y compensación de movimiento es clave en todos los codificadores de vídeo que vamos a estudiar:
  - H.261
  - MPEG-1
  - MPEG-2
  - AVC
  - HEVC
  - …
- Por simplicidad y eficiencia, la mayor parte de los esquemas se basan en la estimación por bloques, aunque también existen otras alternativas.



2. Estimación por bloques

¿Por qué utilizar bloques?

- El objetivo no es buscar el movimiento real de los objetos, sino reducir el error de predicción que posteriormente se va a codificar.

- Al utilizar grupos de píxeles adyacentes, la estimación es más robusta al ruido.

- Cuanto menor sea el nº de bloques, menor nº de vectores de movimiento hay que transmitir → mayor compresión.

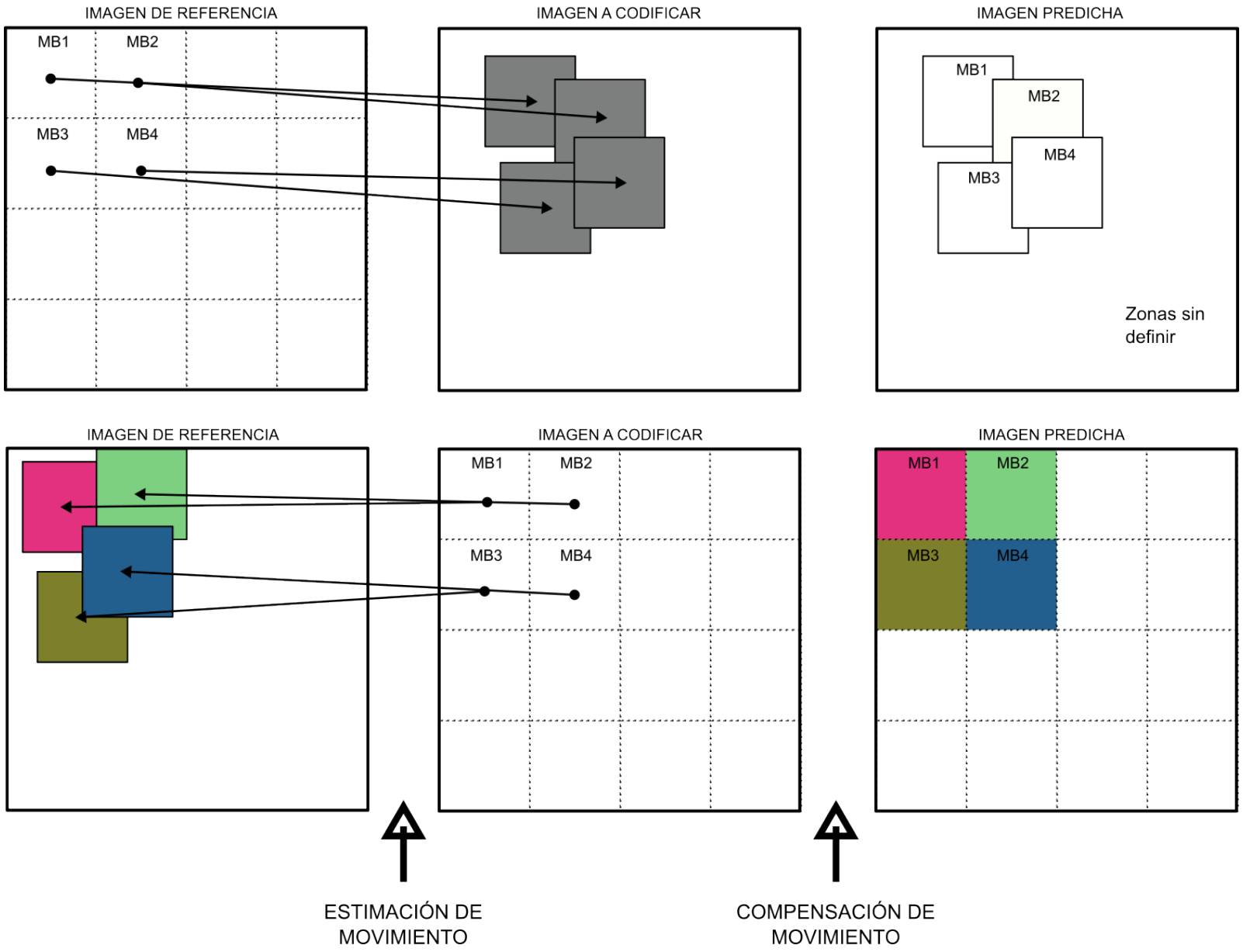
- Se realizan otras operaciones también a nivel de bloque (e.g., DCT 8×8) → Realizando la estimación sobre bloques que sean múltiplos enteros de 8×8 se reducen efectos no deseados (discontinuidades entre bloques).

- Facilitan el procesamiento en paralelo (muy tenido en cuenta en los codificadores más modernos).

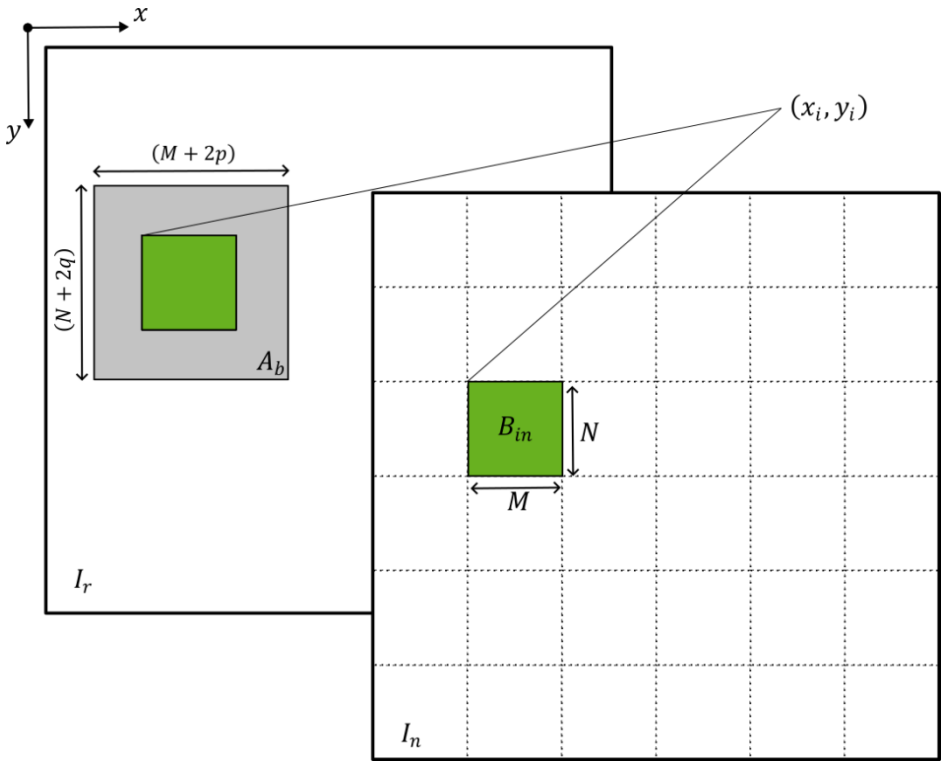
2. Estimación por bloques

- **Objetivo:**
  - Sea una imagen I_n dividida en bloques de $(M \times N)$ píxeles. Para cada bloque se tratará de localizar otro bloque en una imagen de referencia I_r, que minimice una determinada función de coste.
  - Menor error → Codificable con menos bits.
- **Restricciones:**
  - Sólo se consideran movimientos traslacionales (en el caso general también podría haber cambios de forma, de tamaño, etc.).
  - Inicialmente la predicción se obtenía de imágenes anteriores en el tiempo, pero actualmente también se obtiene de imágenes posteriores.
  - La imagen utilizada como referencia, I_r, **es en realidad una imagen reconstruida** → así tanto el codificador como el decodificador utilizan la misma información en la compensación.

2. Estimación por bloques



3. Introducción al ajuste de bloques



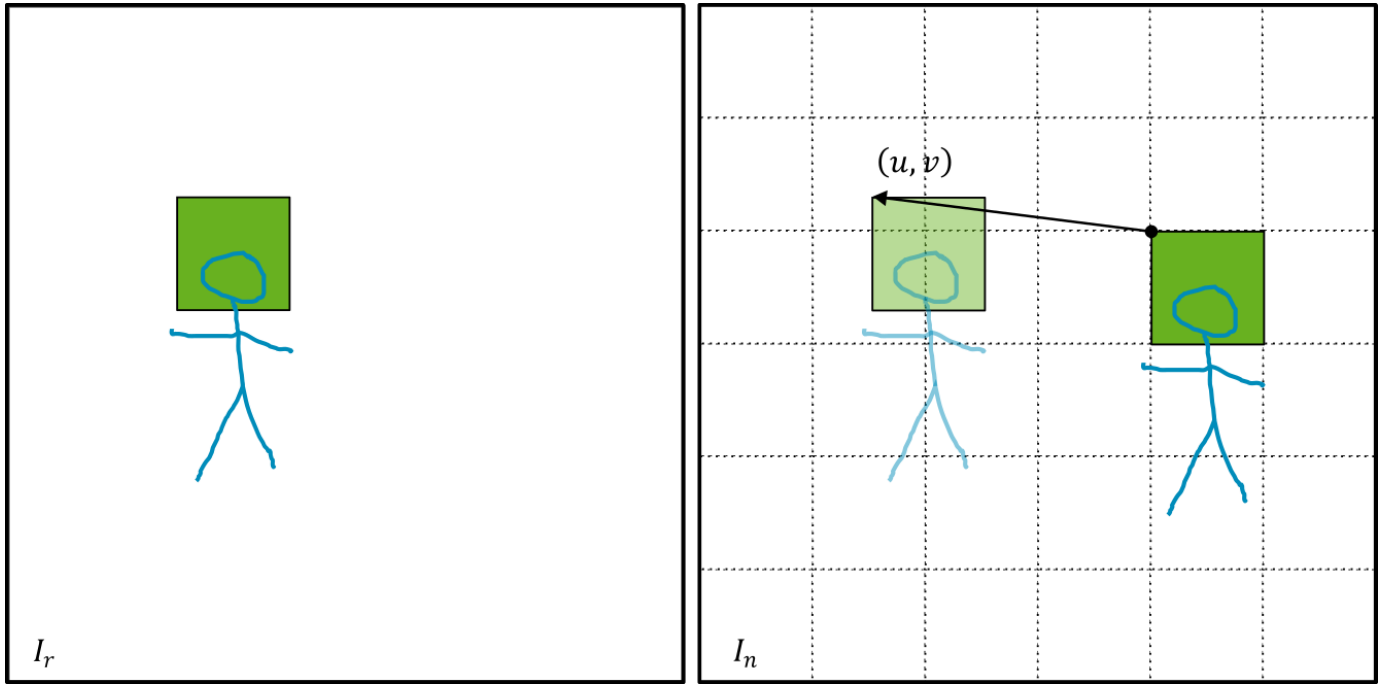
- Sea un bloque B_{in} de $(M \times N)$ píxeles en la imagen I_n.
  - Sus coordenadas serán las de su esquina superior izquierda (x_i, y_i).

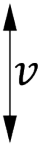
- Este bloque se comparará con bloques del mismo tamaño en la imagen de referencia I_r, dentro de un área de búsqueda, A_b, alrededor de la posición del bloque B_{in}.

Área de búsqueda:

- Se define con dos parámetros, (p, q), fijando su tamaño en $(M + 2p) \times (N + 2q)$ píxeles.

3. Introducción al ajuste de bloques





- Se busca el bloque B_{jr} que minimice (o maximice) cierta función de coste.
- Si las coordenadas de este bloque son (x_j, y_j) → El vector de movimiento asociado al bloque B_{in} será:

$$(u, v) = (x_j - x_i, y_j - y_i)$$

- El rango de posibles valores de los vectores de movimiento depende del área de búsqueda → $[\pm p, \pm q]$

3.1. Criterios de selección

- Para decidir qué bloque seleccionar en la imagen de referencia se utiliza una **función de coste** que hay que minimizar (o maximizar).

- **Error cuadrático medio:**

$$MSE(u,v) = \frac{1}{M \cdot N} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left(B_{in}(x,y) - B_{jr}(x+u, y+v) \right)^2$$

- **Error absoluto medio:**

$$MAE(u,v) = \frac{1}{M \cdot N} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left| B_{in}(x,y) - B_{jr}(x+u, y+v) \right|$$

- **Función de correlación cruzada normalizada:**

$$NCCF(u,v) = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left(B_{in}(x,y) B_{jr}(x+u, y+v) \right)}{\left(\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left(B_{in}(x,y) \right)^2 \right)^{\frac{1}{2}} \cdot \left(\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left(B_{jr}(x+u, y+v) \right)^2 \right)^{\frac{1}{2}}}$$

3.2. Tamaño del bloque

- El tamaño de los bloques se selecciona teniendo en cuenta lo siguiente:

1. Error de predicción:
   - Los bloques pequeños permiten aproximar mejor el movimiento en la escena y, por lo tanto, dan lugar a menores errores de predicción.

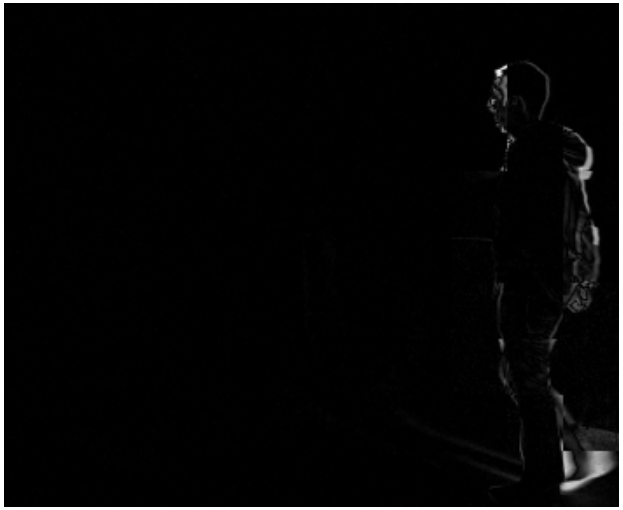
2. Volumen de información:
   - Cuanto más pequeños son los bloques, la cantidad de información de movimiento a transmitir es mayor (más vectores por imagen).

3. Eficiencia:
   - Los algoritmos rápidos de búsqueda son más eficientes si los bloques son grandes.

3.2. Tamaño del bloque

Referencia

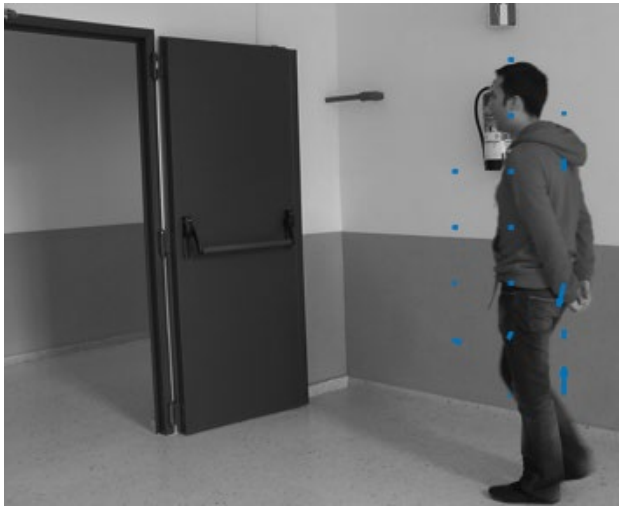
Error



$(M, N) = (32,32)$

Actual

Vectores



$RMS = 6{,}99$

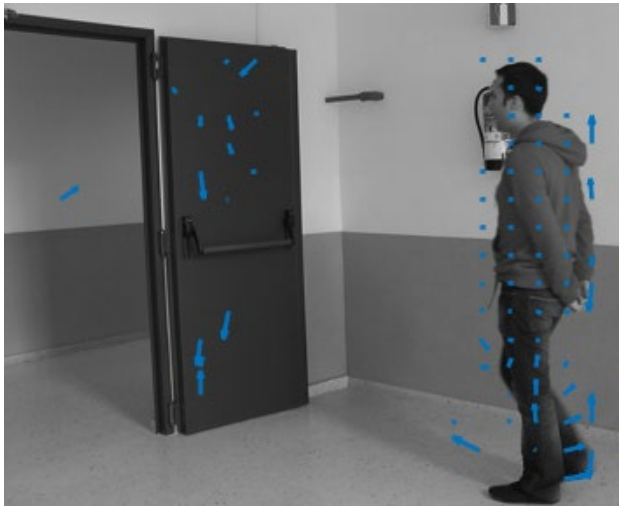
3.2. Tamaño del bloque

Referencia

Error

Actual

Vectores



$(M, N) = (16,16)$

$RMS = 3{,}31$

3.2. Tamaño del bloque

Referencia

Error

Actual

Vectores

$$(M, N) = (8,8)$$

$$RMS = 2{,}02$$

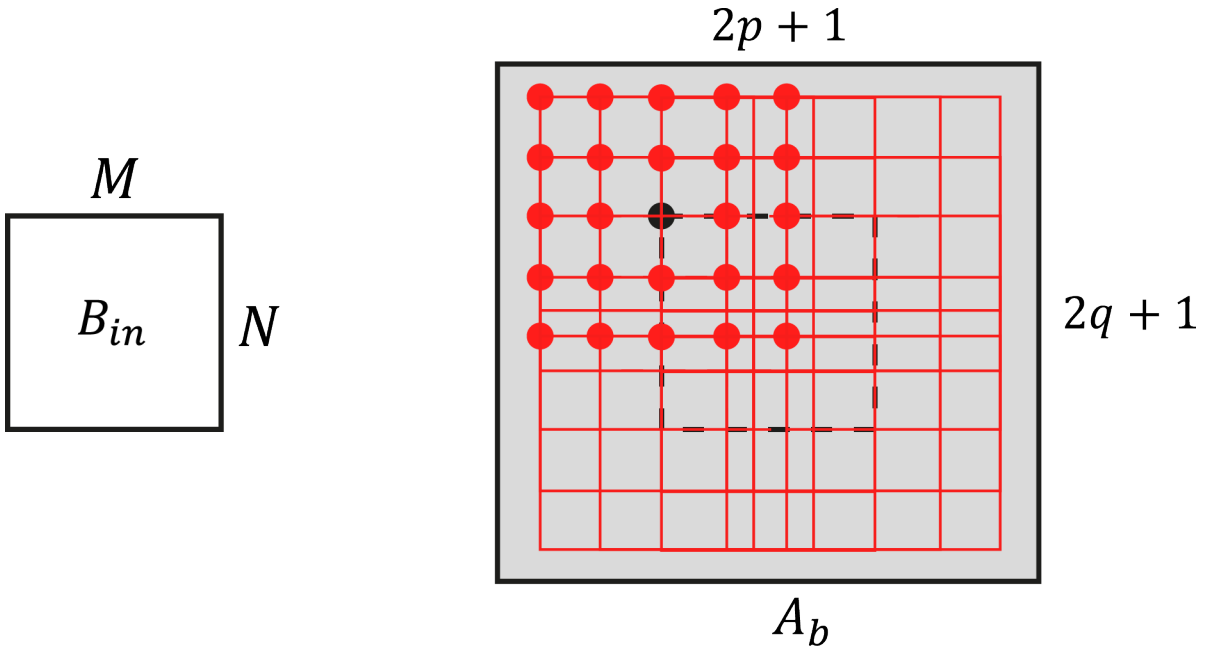
4. Algoritmos de ajuste de bloques

- Vamos a estudiar algunos de los algoritmos más comunes.
  - El criterio de selección (funciones de coste) y el tamaño de los bloques será configurable.
  - Analizaremos en detalle el coste computacional.

- **Búsqueda exhaustiva:**
  - Se analizan todas las posiciones posibles.
- **Búsqueda logarítmica:**
  - Búsqueda en posiciones concretas dentro de áreas de distinto tamaño (escaladas logarítmicamente).
- **Búsqueda jerárquica:**
  - Búsqueda en varios niveles en los que se analizan bloques en posiciones concretas.
- **Algoritmos piramidales y algoritmos con precisión fraccionaria:**
  - Complementan a cualquiera de las anteriores para mejorar la calidad de los resultados (precisión fraccionaria) o la eficiencia (algoritmos piramidales).

4.1. Búsqueda exhaustiva

- Es el método más simple, pero es muy costoso.
- Consiste en determinar el valor de la función de coste para cada posible localización dentro del área de búsqueda.
- El vector de desplazamiento será el correspondiente a la comparación que dé lugar al menor coste.



4.1. Búsqueda exhaustiva

Imagen de referencia

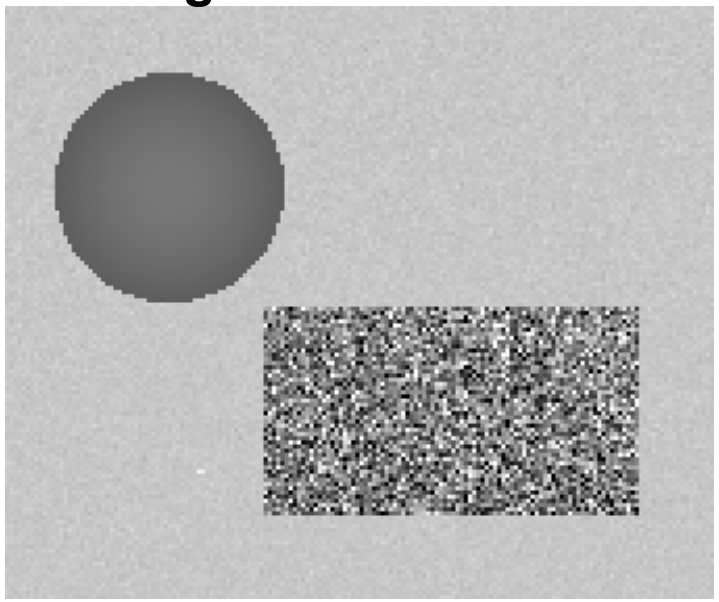
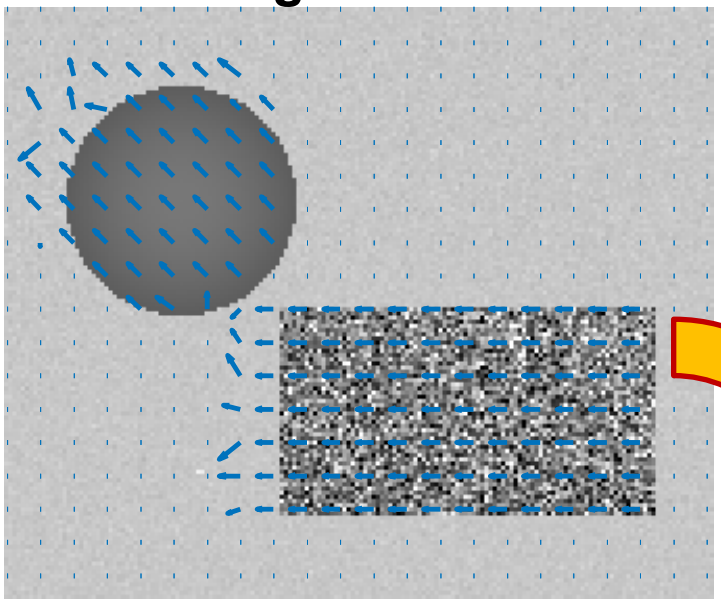
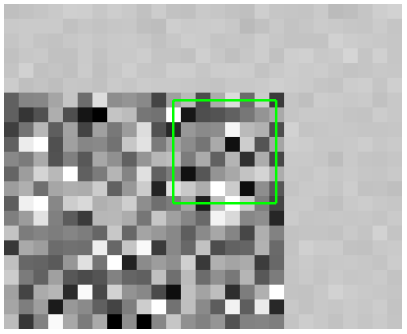


Imagen actual



Demo con Matlab



4.1. Búsqueda exhaustiva

- **Coste:**
  - Función de coste → *MAE*
  - Área de búsqueda → (p, q)
  - Tamaño de los bloques → $(M \times N)$
  - Tamaño de las imágenes → $(F \times C)$
  - Nº de imágenes por segundo → *T*
  - Nº de bloques con los que compararse → $(2p + 1,2q + 1)$
  - Nº de píxeles sobre los que calcular el *MAE* en cada bloque → $(M \times N)$
  - Nº de operaciones en el cálculo del *MAE* (por píxel) → 3 (diferencia, valor absoluto y suma)
  - Coste total:

$$3 \cdot T \cdot F \cdot C \cdot (2p + 1)(2q + 1)$$

  - Si $T = 25$, $(F, C) = (576,720)$ y $p = q = 16$ → 33,872 Gop/s
  - Si $T = 25$, $(F, C) = (576,720)$ y $p = q = 8$ → 8,989 Gop/s

4.2. Búsqueda logarítmica

- Es más complejo que la búsqueda exhaustiva.

- Restringe el número de bloques con los que compararse dentro del área de búsqueda:
  - **Reduce el coste computacional de la búsqueda**.
  - No asegura el mejor resultado (mínimo error).

- Es adecuada para secuencias sencillas, con pocas texturas.

4.2. Búsqueda logarítmica

1. Se determina el número de iteraciones de las que constará la búsqueda:

$$k_p = \lceil \log_2 p \rceil \qquad k_q = \lceil \log_2 q \rceil$$

2. Se calculan las distancias de los bloques con los que se comparará el actual en la primera iteración:

$$d_{1p} = 2^{k_p - 1} \qquad d_{1q} = 2^{k_q - 1}$$

- Ejemplos:
  - Si $p = q = 7$ → $k_p = k_q = \lceil 2{,}8 \rceil = 3$ → $d_{1p} = d_{1q} = 4$ (caso ilustrado a continuación)
  - Si $p = q = 15$ → $k_p = k_q = \lceil 3{,}9 \rceil = 4$ → $d_{1p} = d_{1q} = 8$

4.2. Búsqueda logarítmica

Vemos un ejemplo para el caso de:

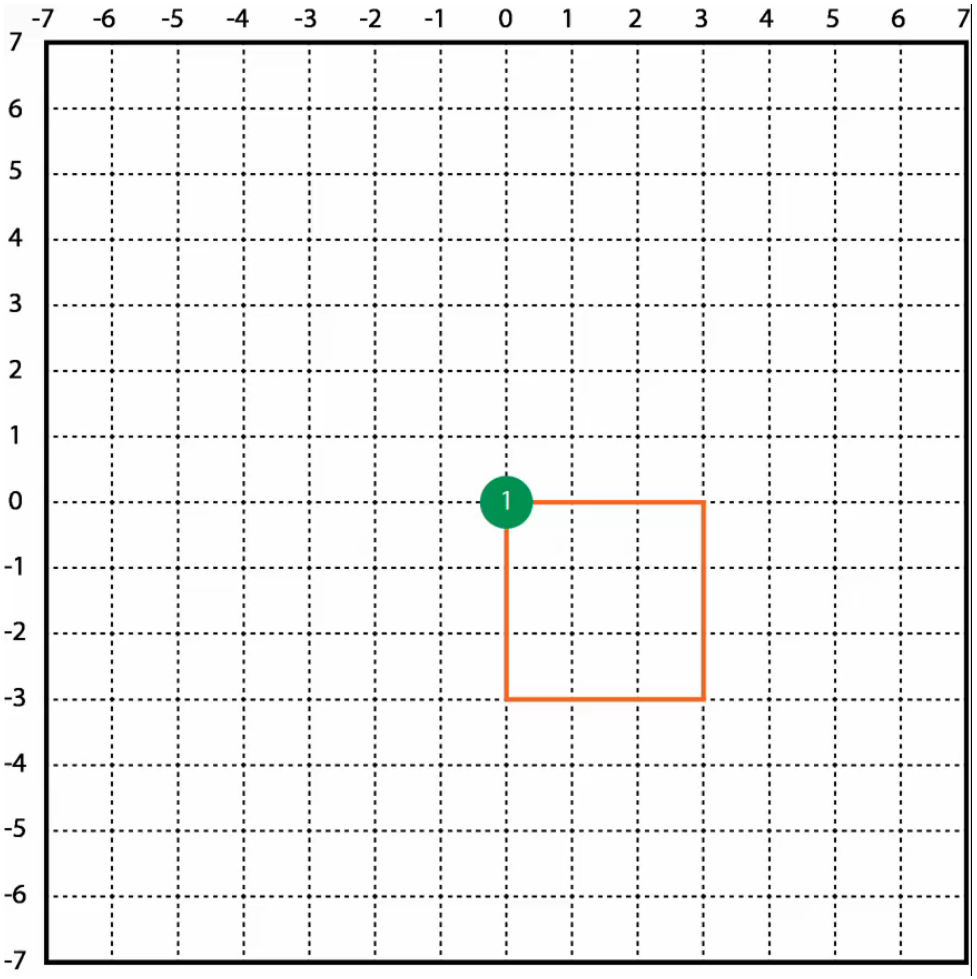
$$p = q = 7$$

$$M = N = 3$$

Por lo tanto:

- Iteraciones: $k_p = k_q = 3$
- Distancia inicial: $d_{1p} = d_{1q} = 4$

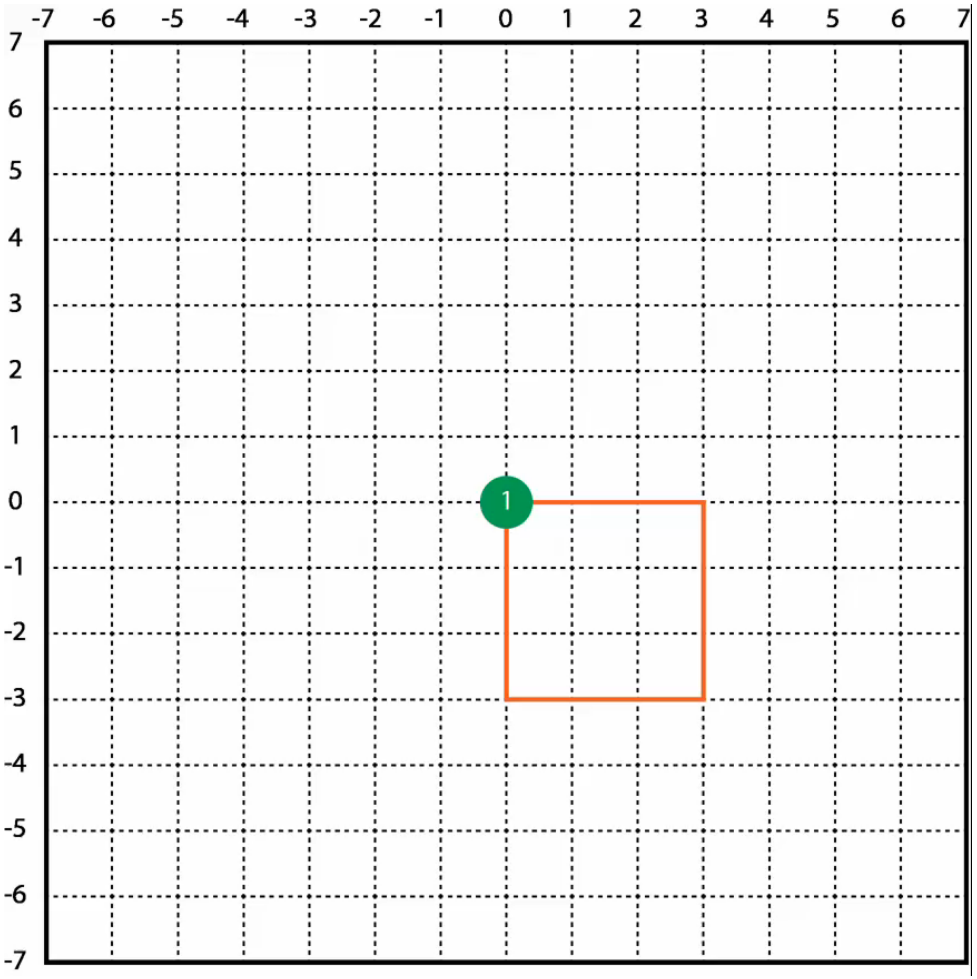
Este caso particular (3 saltos) se conoce como *Three-Step Search* (*TSS*).



4.2. Búsqueda logarítmica

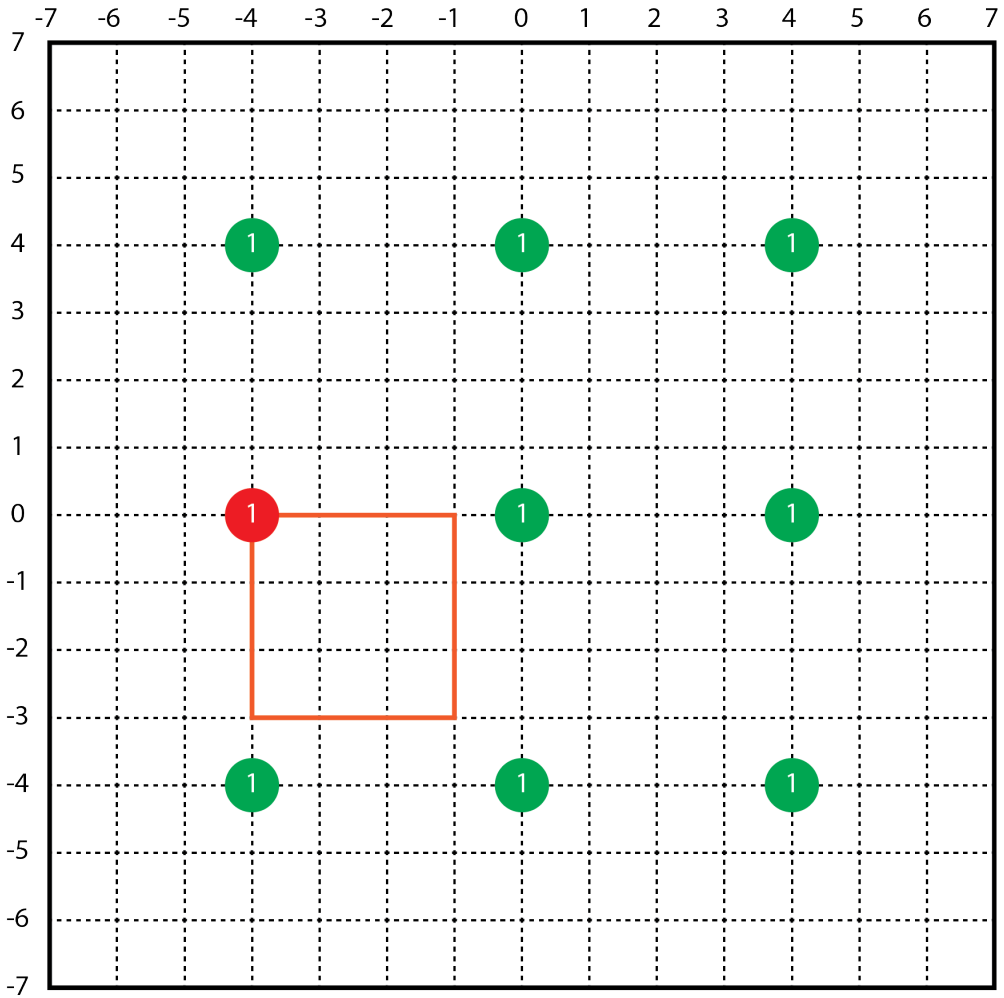
3. Se evalúa la función de coste para los bloques cuyo origen se encuentre en:

- El centro del área de búsqueda:
  - (0,0)
- Los vértices:
  - (d_{1p}, d_{1q})
  - $(-d_{1p}, d_{1q})$
  - $(d_{1p}, -d_{1q})$
  - $(-d_{1p}, -d_{1q})$
- Los puntos medios:
  - $(d_{1p}, 0)$
  - $(-d_{1p}, 0)$
  - $(0, d_{1q})$
  - $(0, -d_{1q})$



4.2. Búsqueda logarítmica

4. Se selecciona como nuevo origen el del bloque que haya dado lugar a un menor coste.



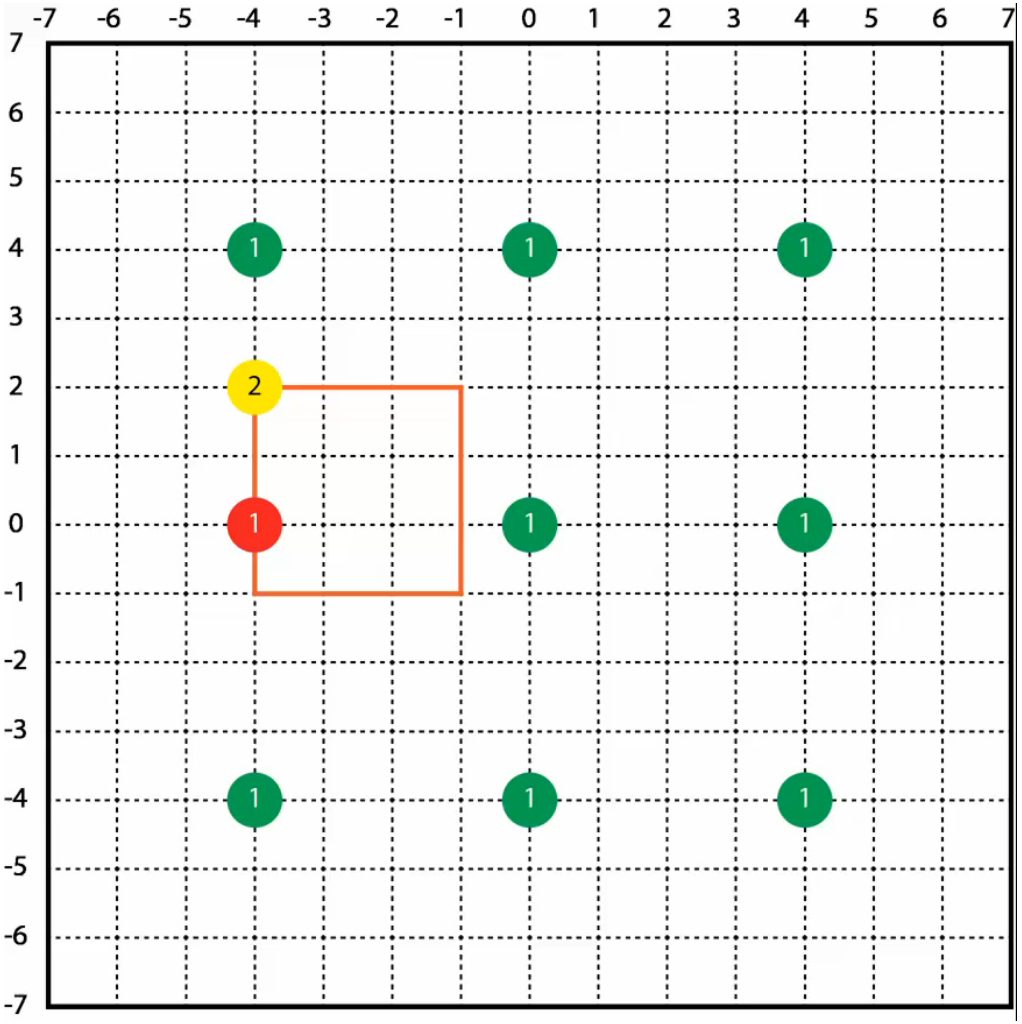
4.2. Búsqueda logarítmica

5. Se calcula una nueva distancia,

$$d_{2p} = \frac{d_{1p}}{2}$$

$$d_{2q} = \frac{d_{1q}}{2}$$

y se analiza el coste correspondiente a los bloques situados en los vértices y puntos medios del área determinada por dicha distancia.



4.2. Búsqueda logarítmica

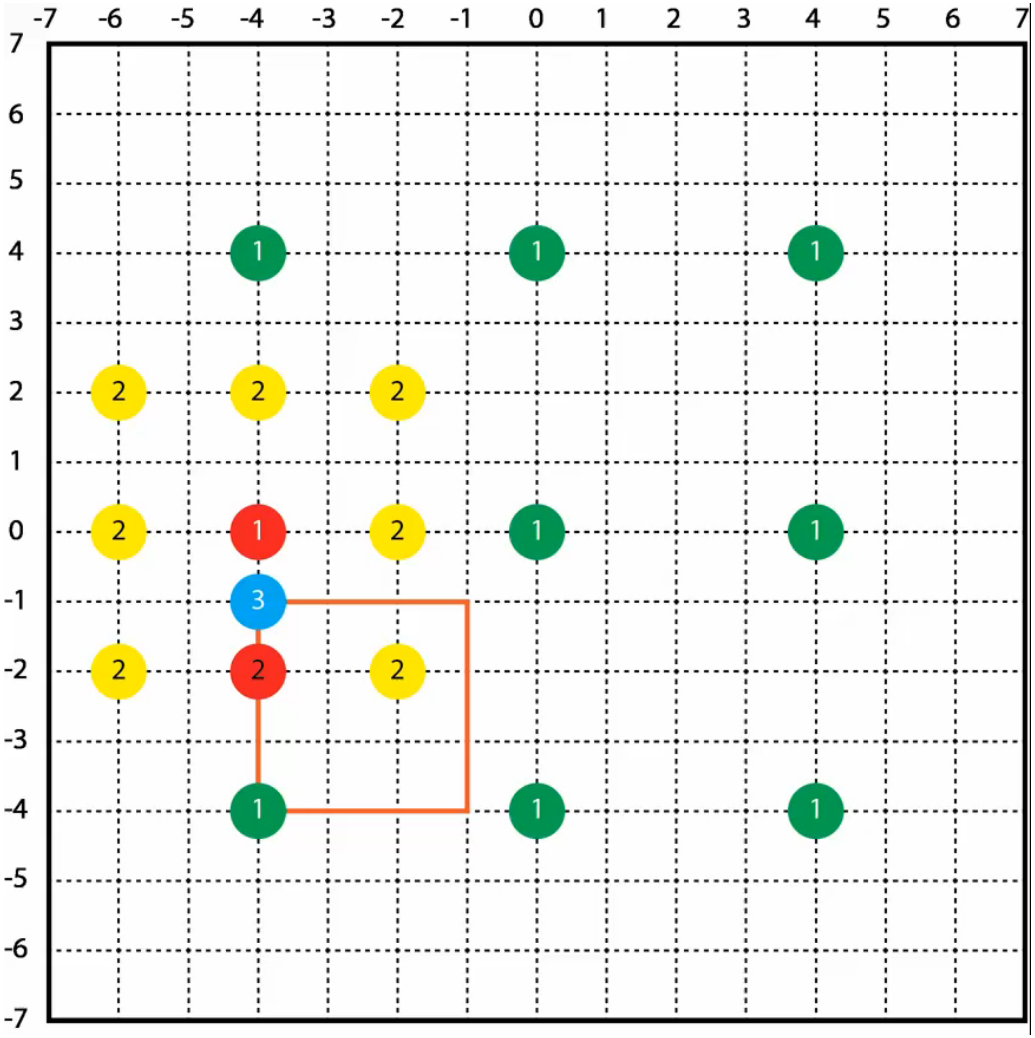
Si $k_p = k_q = k$, este proceso se repite k veces (i.e., hasta que se analizan los bloques a distancia 1 en ambas dimensiones).

En este ejemplo:

$$d_{1p} = d_{1q} = 4$$

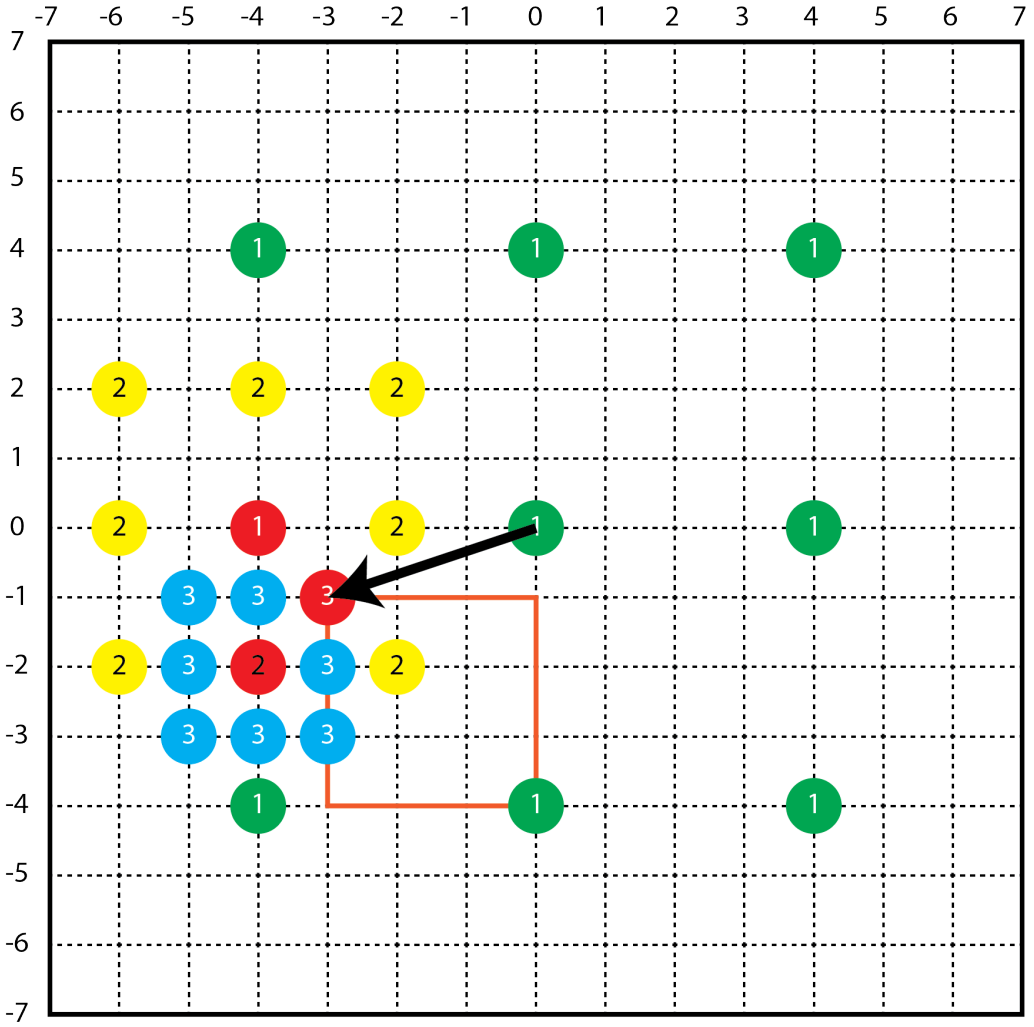
$$d_{2p} = d_{2q} = 2$$

$$d_{3p} = d_{3q} = 1$$



4.2. Búsqueda logarítmica

- El vector de movimiento finalmente obtenido sería el mostrado en la figura.



4.2. Búsqueda logarítmica

Si $k_p \neq k_q$, hay dos opciones para finalizar:

a) Cuando se alcanza la distancia unidad en una de las dos dimensiones.

- Ejemplo:
  - $(p, q) = (7,15) \rightarrow (k_p, k_q) = (3,4)$
  - $(d_{1p}, d_{1q}) = (4,8), (d_{2p}, d_{2q}) = (2,4), (d_{3p}, d_{3q}) = (1,2)$

b) Cuando se alcanza la distancia unidad en una de las dos dimensiones, a esa dimensión se pone a 0 y se continua en la restante.

- Ejemplo:
  - $(p, q) = (7,15) \rightarrow (k_p, k_q) = (3,4)$
  - $(d_{1p}, d_{1q}) = (4,8), (d_{2p}, d_{2q}) = (2,4), (d_{3p}, d_{3q}) = (1,2), (d_{4p}, d_{4q}) = (0,1)$

4.2. Búsqueda logarítmica

- **Coste:**
  - Función de coste → *MAE*
  - Área de búsqueda → (p, q)
  - Tamaño de los bloques → $(M \times N)$
  - Tamaño de las imágenes → $(F \times C)$
  - Nº de imágenes por segundo → *T*
  - Nº de iteraciones necesarias → *k* (suponemos que $p = q$)

1. Nº de bloques con los que comparar cada bloque de la imagen actual:
   - Primera iteración → 9 bloques
   - Resto de iteraciones (*k* – 1) → 8 bloques
   - Total → (8*k* + 1) bloques

4.2. Búsqueda logarítmica

- **Coste:**

2. Nº de pixeles por bloque sobre los que evaluar la función de coste → $(M \times N)$

3. Nº de operaciones a realizar (por píxel) → Una diferencia, un valor absoluto y una suma → 3

4. Nº de bloques en la imagen original para los que realizar la búsqueda logarítmica: $\frac{F \cdot C}{M \cdot N}$

- Coste total:

$$3 \cdot T \cdot F \cdot C \cdot (8 \cdot k + 1)$$

  - Si $T = 25$, $(F, C) = (576,720)$ y $p = q = 16$ $(k = 4)$ → 1,026 Gop/s
  - Si $T = 25$, $(F, C) = (576,720)$ y $p = q = 8$ $(k = 3)$ → 0,778 Gop/s

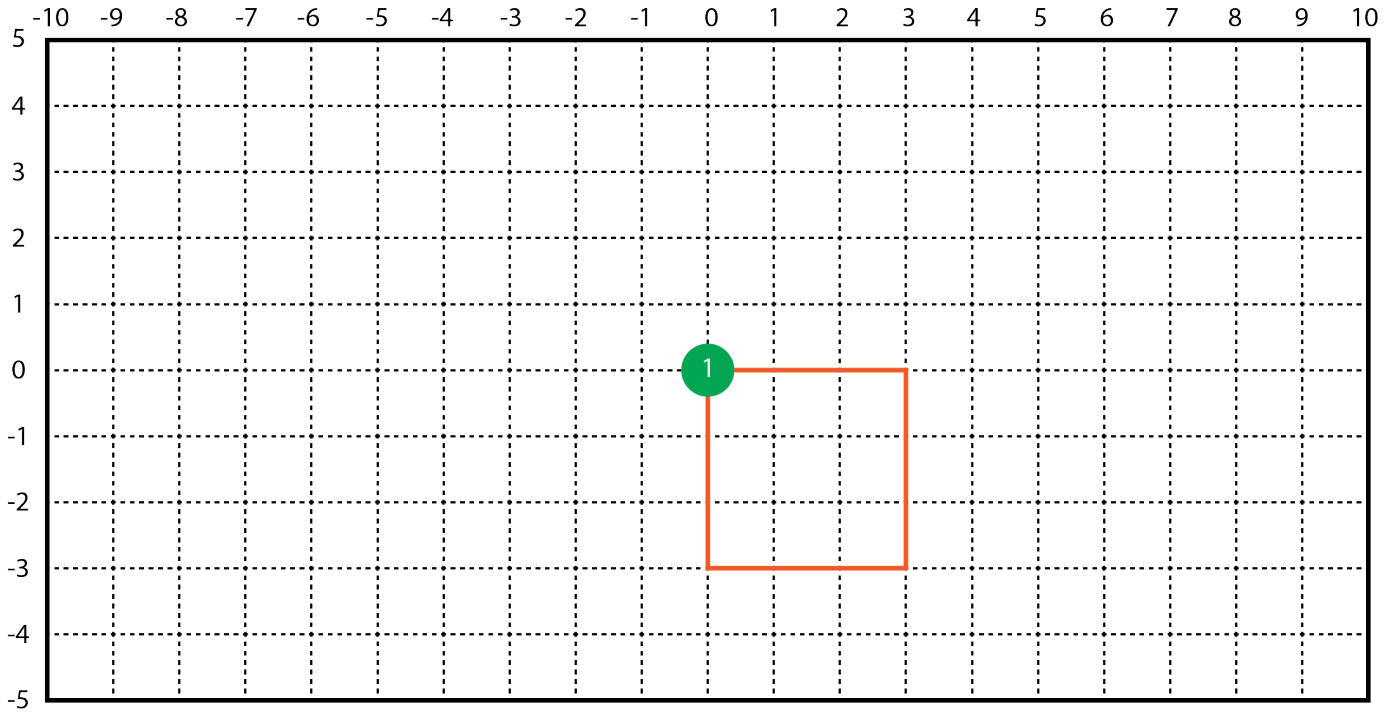
4.3. Búsqueda jerárquica

- Realiza el ajuste de bloques en varios pasos o niveles.
  - En cada nivel únicamente se ven involucrados algunos bloques (situados en posiciones concretas del área de búsqueda).
- Parámetros de los que depende la búsqueda (predeterminados por el usuario):
  - Tamaño del área de búsqueda (A_b) → ($p \times q$) píxeles.
  - Nº de niveles jerárquicos:
    - Vamos a particularizar el análisis del método al caso de 2 niveles.
  - Distancia inicial de análisis → (d_x,d_y)
    - Distancia, en columnas y filas, entre los bloques analizados en el primer nivel jerárquico.
    - En los siguientes niveles se va reduciendo.

Nomenclatura → ***Jerar*** **[d_x,d_y]**

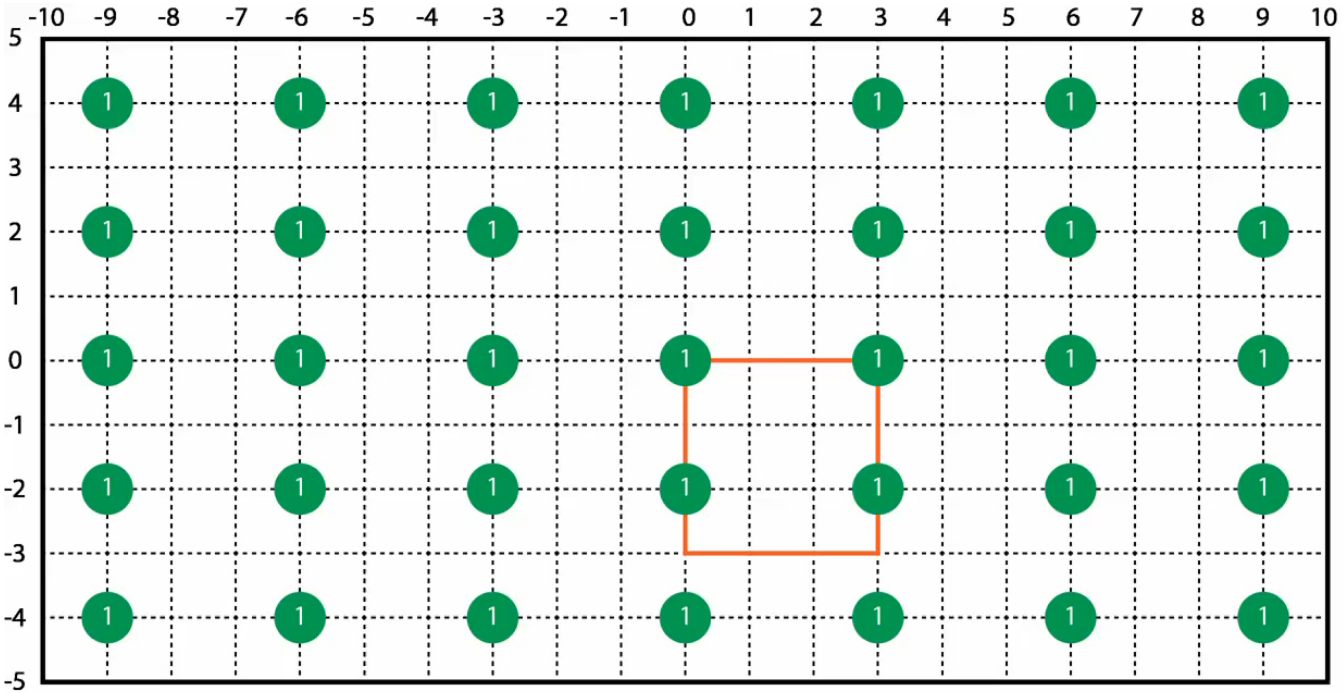
4.3. Búsqueda jerárquica

- Ejemplo:
  - Dos niveles de jerarquía.
  - Área de búsqueda definida por p = 10 y q = 5.
  - Distancia inicial entre bloques → (d_x, d_y) = (3, 2) → $Jerar[3,2]$.



4.3. Búsqueda jerárquica

- Primer nivel jerárquico:
  - Se compara con los bloques cuyo origen dista de la posición (0,0) múltiplos enteros de d_x en horizontal y múltiplos enteros de d_y en vertical.
  - El resultado es un vector de movimiento (u^1,v^1) que apunta al bloque con el que se haya obtenido la menor función de coste.

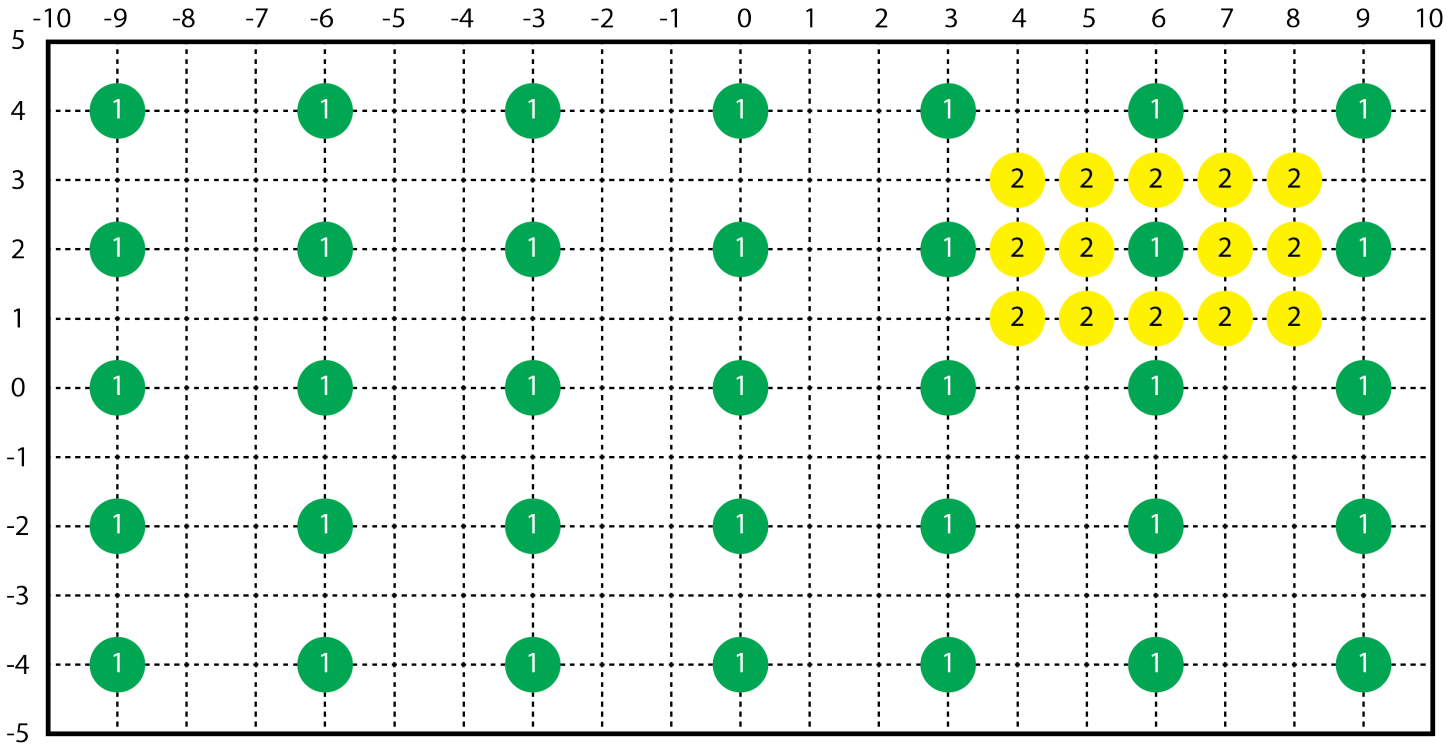


4.3. Búsqueda jerárquica

- Segundo nivel jerárquico:

  1. Se define una nueva área de búsqueda (A_b^2) formada por todos los bloques cuyo origen (x_i,y_i) pertenezca al área de búsqueda inicial y verifiquen que: $\left|x_i - u^1\right| \le d_x - 1 \quad \left|y_i - v^1\right| \le d_y - 1$

  - En el ejemplo → (d_x – 1, d_y – 1) = (2, 1)

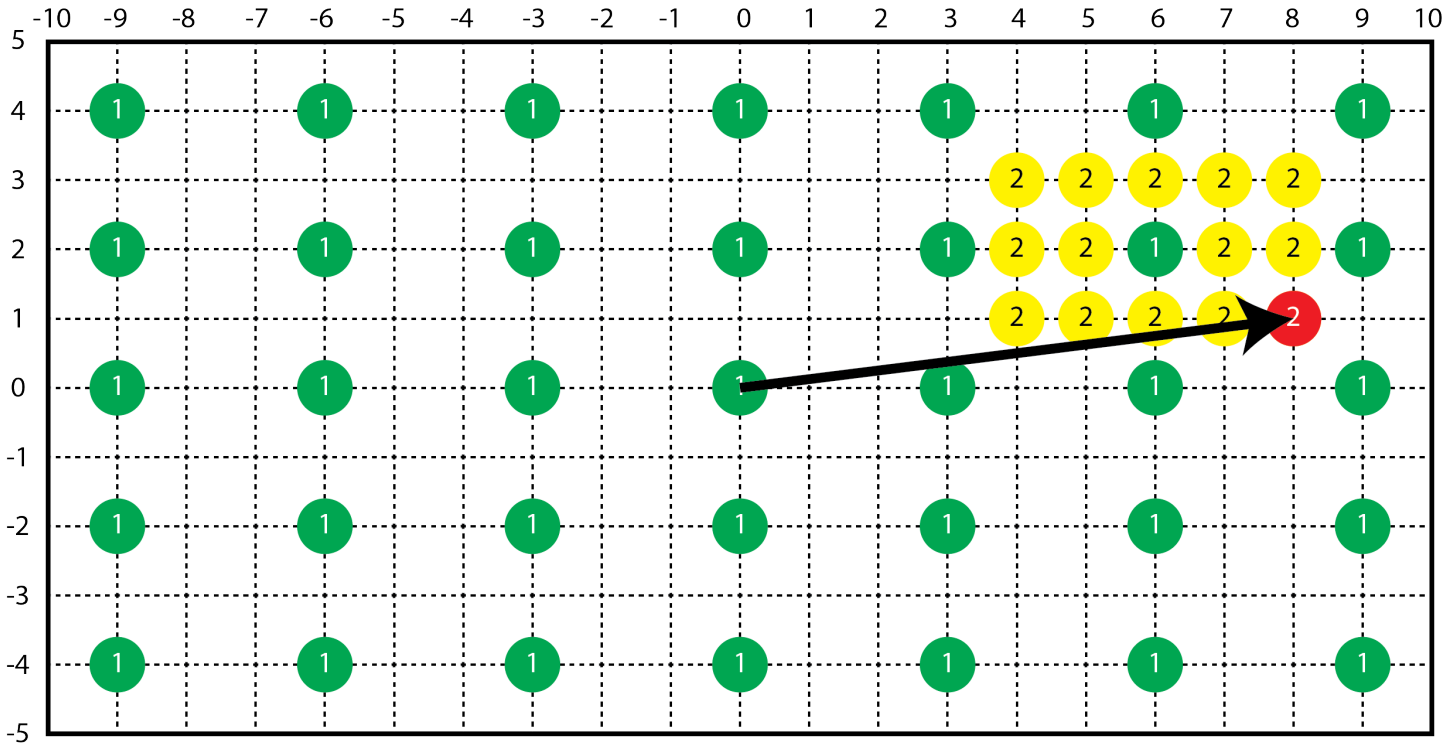


4.3. Búsqueda jerárquica

- Segundo nivel jerárquico:

  2. Dentro de la nueva área de búsqueda se lleva a cabo una búsqueda exhaustiva.

  3. El vector de movimiento final será el que minimice la función de coste en esta búsqueda.



4.3. Búsqueda jerárquica

- **Coste:**
  - Función de coste → *MAE* (*u*,v)
  - Área de búsqueda → (*p*,*q*)
  - Tamaño de los bloques → (*M* × *N*)
  - Tamaño de las imágenes → (*F* × *C*)
  - Nº de imágenes por segundo → *T*
  - Nº de niveles jerárquicos → 2
  - Característica del algoritmo → *Jerar* [d_x,d_y]

1. Nº de bloques en la imagen de referencia con los que comparar cada bloque de la imagen actual:

   - Primer nivel jerárquico → $\left(2\cdot\left\lfloor\frac{p}{d_x}\right\rfloor+1\right)\cdot\left(2\cdot\left\lfloor\frac{q}{d_y}\right\rfloor+1\right)$

   - Segundo nivel jerárquico → $\left((2d_x-1)\cdot(2d_y-1)-1\right)$

4.3. Búsqueda jerárquica

- **Coste:**

2. Nº de pixeles por bloque sobre los que evaluar la función de coste → ($M \times N$)

3. Nº de operaciones a realizar (por píxel) → Una diferencia, un valor absoluto y una suma → 3

4. Nº de bloques en la imagen original para los que realizar la búsqueda: $\frac{F \cdot C}{M \cdot N}$

- Coste total:

$$3 \cdot T \cdot F \cdot C \cdot \left(\left(2 \cdot \left\lfloor \frac{p}{d_x} \right\rfloor + 1 \right) \cdot \left(2 \cdot \left\lfloor \frac{q}{d_y} \right\rfloor + 1 \right) + \left((2d_x - 1) \cdot (2d_y - 1) - 1 \right) \right)$$

  - Si T=25, (F,C)=(576,720) y p=q=16, *Jerar*[3,2] → 6,252 GOPS
  - Si T=25, (F,C)=(576,720) y p=q=8, *Jerar*[3,2] → 1,835 GOPS

5. Algoritmos piramidales

- **Objetivos:**
  - Mejorar la eficiencia computacional (reducir la complejidad) de los algoritmos de búsqueda vistos hasta ahora.

- **¿Cómo?**
  - Reduciendo el número de bloques de la imagen de referencia con los que comparar cada bloque de la imagen actual.
    - Esto ya se conseguía con los algoritmos de búsqueda logarítmica o jerárquica.
  - Reduciendo el número de píxeles empleados para evaluar la función de coste en cada comparación.
    - Ahorro computacional adicional.
    - Se modifica tanto el área de búsqueda como el tamaño de los bloques a comparar.

5. Algoritmos piramidales

- **Pasos del algoritmo:**
  - Supongamos un análisis caracterizado por:
    - Un área de búsqueda A_b definida por (p,q) en una imagen de (F×C) píxeles.
    - Un bloque de la imagen actual, B_i, para el que se quiere determinar un vector de movimiento que estime su movimiento respecto de una imagen de referencia:
      - Dimensiones → (M×N)
      - Coordenadas → (x_i,y_i)
  - Se llevarán a cabo los siguientes pasos:

    1. Generación de L imágenes de menor resolución (para la imagen actual y para la de referencia).
    2. Obtención de un vector de movimiento entre las imágenes de menor resolución.
    3. Iterativamente, partiendo del último vector de movimiento estimado, se obtienen vectores de movimiento en el resto de las resoluciones.

5. Algoritmos piramidales

- **Paso 1:**

  - Tanto para la imagen original como para la de referencia, se obtiene una jerarquía o pirámide de *L* imágenes de menor resolución.

    - Normalmente, cada nueva imagen se obtiene submuestreando por 2 cada una de las dimensiones de la anterior.

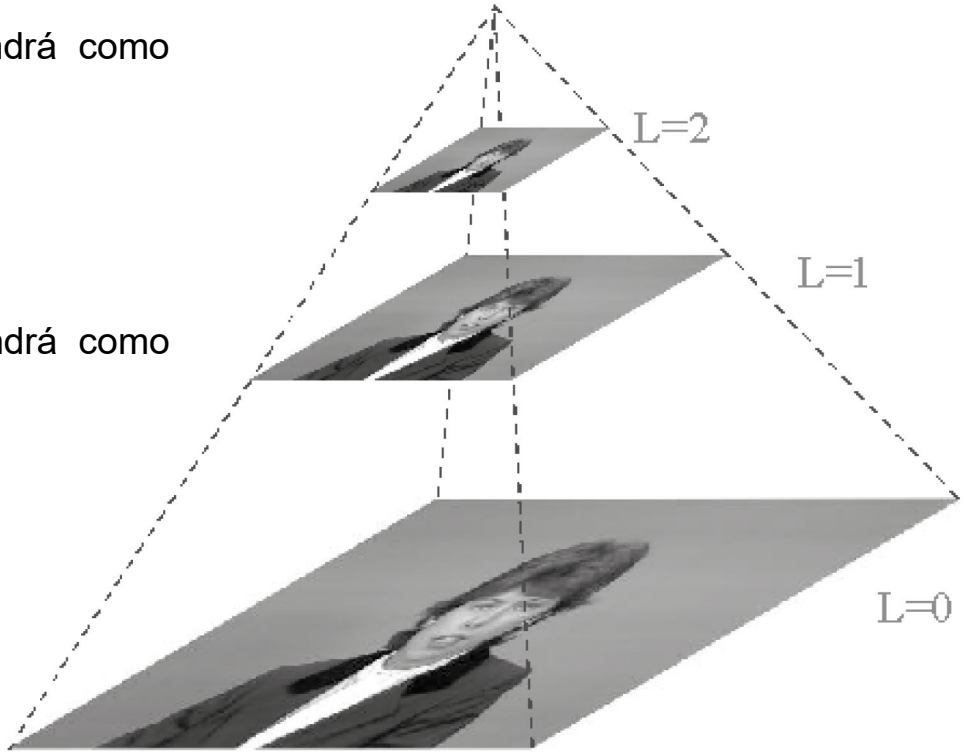
➢ La imagen correspondiente al nivel *L* tendrá como dimensiones:

$$F_L = F/2^L \qquad C_L = C/2^L$$

➢ El origen del bloque B_i en el nivel *L* tendrá como coordenadas:

$$(x_i^L, y_i^L) = \left(x_i/2^L, y_i/2^L \right)$$

➢ La imagen a resolución completa también se denomina imagen de nivel *L* = 0.



5. Algoritmos piramidales

- **Paso 2:**

  - En el nivel *L* se determina un vector de movimiento asociado al bloque B_i.
  - Se utiliza cualquiera de los algoritmos de búsqueda antes vistos.
  - Se ha de tener en cuenta que se ha reducido tanto el tamaño del bloque como los parámetros que definen el área de búsqueda:
    - Las dimensiones del bloque serán → $M_L = M/2^L \quad N_L = N/2^L$
    - El área de búsqueda vendrá definida por → $p_L = p/2^L \quad q_L = q/2^L$
  - El resultado de este paso será un vector de movimiento para el bloque en el nivel *L*: $\left(u_i^L, v_i^L\right)$

5. Algoritmos piramidales

- **Paso 3:**

  - Se va descendiendo de nivel de jerarquía, realizando iterativamente las mismas operaciones.

  a) Actualización de las coordenadas del vector:

  - En el nivel L, el vector (u_i^L, v_i^L) apunta a las coordenadas $(x_i^L + u_i^L, y_i^L + v_i^L)$.
  - En el nivel L-1, el vector (u_i^{L-1}, v_i^{L-1}) apunta a las coordenadas $(2x_i^L + 2u_i^L, 2y_i^L + 2v_i^L)$.

  b) Nueva búsqueda:

  - Área de búsqueda definida por $p = q = 1$.
  - Exhaustiva.
  - Nuevo tamaño de bloques → $M_{L-1} = M/2^{L-1} \quad N_{L-1} = N/2^{L-1}$

  c) Nuevo vector de movimiento (el que dé lugar al menor coste):

  - Coordenadas → $(u_i^{L-1} + x', v_i^{L-1} + y')$, donde x' e y' pueden valer -1, 0 o 1.

  d) Se repiten a), b) y c) hasta llegar al nivel $L = 0$.

5. Algoritmos piramidales

- **Coste:**
  - Función de coste → *MAE* (*u*,v).
  - Área de búsqueda → (*p*,*q*)
  - Tamaño de los bloques → (*M* × *N*)
  - Tamaño de las imágenes → (*F* × *C*)
  - Nº de imágenes por segundo → *T*
  - Tipo de búsqueda → Exhaustiva
  - Nº de niveles → *L*

1. Coste en el nivel *L*.
2. Coste en el resto de los niveles.

5. Algoritmos piramidales

- **Coste en el nivel L:**
  - Nº de bloques con los que comparar cada bloque de la imagen actual:

$$(2p_L+1)\cdot(2q_L+1)=\left(2\frac{p}{2^L}+1\right)\cdot\left(2\frac{q}{2^L}+1\right)$$

  - Dimensión de los bloques → $(M_L \times N_L) = (M \times N)(1/2^{2L})$
  - Dimensión de la imagen → $(F_L \times C_L) = (F \times C)(1/2^{2L})$
  - Nº de operaciones a realizar (por píxel) → Una diferencia, un valor absoluto y una suma → 3
  - Nº de bloques en la imagen original (nivel L) → $\left(F_L \cdot C_L / M_L \cdot N_L\right)=\left(F \cdot C / M \cdot N\right)$
  - Coste total:

$$3 \cdot F \cdot C \cdot T \cdot \frac{1}{2^{2L}}\left(2\frac{p}{2^L}+1\right)\cdot\left(2\frac{q}{2^L}+1\right)$$

5. Algoritmos piramidales

- **Coste en resto de niveles (L-1 niveles):**
  - Nº de bloques con los que comparar cada bloque de la imagen actual → 9
  - Dimensión de los bloques → $(M_{L-j} \times N_{L-j}) = (M \times N)(1/2^{2(L-j)})$
  - Dimensión de la imagen → $(F_{L-j} \times C_{L-j}) = (F \times C)(1/2^{2(L-j)})$
  - Nº de operaciones a realizar (por píxel) → Una diferencia, un valor absoluto y una suma → 3
  - Nº de bloques en la imagen original (nivel *L-j*) → $\left(F \cdot C / M \cdot N\right)$
  - Operaciones nivel *j* → $3 \cdot T \cdot F \cdot C \cdot 9 \cdot \frac{1}{2^{2(L-j)}}$
  - <u>Coste total:</u>

$$27 \cdot F \cdot C \cdot T \cdot \sum_{j=1}^{L} \frac{1}{2^{2(L-j)}}$$

5. Algoritmos piramidales

- **Coste total:**
  - Nivel L → $3 \cdot F \cdot C \cdot T \cdot \frac{1}{2^{2L}} \left(2\frac{p}{2^L}+1\right) \cdot \left(2\frac{q}{2^L}+1\right)$
  - Resto de niveles → $27 \cdot F \cdot C \cdot T \cdot \sum_{j=1}^{L} \frac{1}{2^{2(L-j)}}$
  - Total (suma de los 2 anteriores):

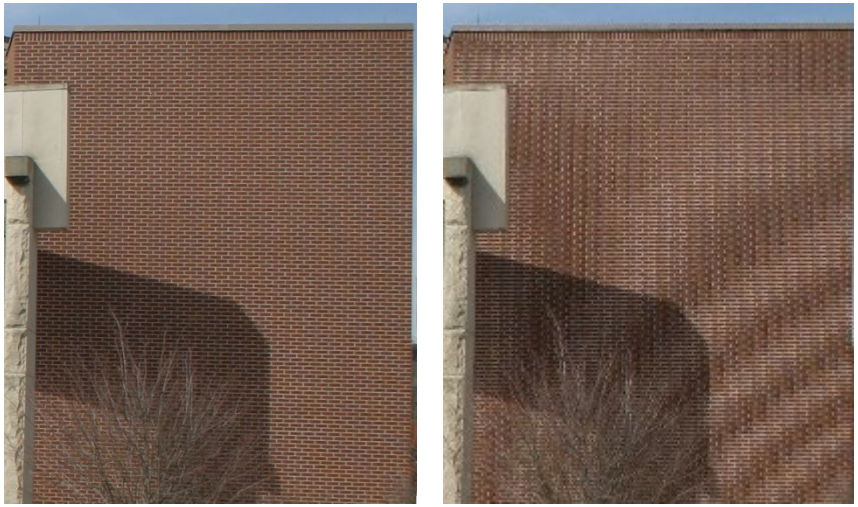
$$3 \cdot F \cdot C \cdot T \cdot \frac{1}{2^{2L}} \left(\left(2\frac{p}{2^L}+1\right) \cdot \left(2\frac{q}{2^L}+1\right) + 9\sum_{j=1}^{L} 2^{2j} \right)$$

- Búsqueda exhaustiva (ver p. 16):
  - Si T=25, (F,C)=(576,720) y p=q=16 → 33,872 GOPS
  - Si T=25, (F,C)=(576,720) y p=q=8 → 8,989 GOPS
- Búsqueda exhaustiva piramidal (con L=2):
  - Si T=25, (F,C)=(576,720) y p=q=16 → 0,507 GOPS
  - Si T=25, (F,C)=(576,720) y p=q=8 → 0,399 GOPS

La reducción es mayor cuanto más grandes son las imágenes

5. Algoritmos piramidales

- **Inconvenientes:**
  - Requieren de mucha memoria → Hay que almacenar varias versiones de las imágenes.
  - La generación de las imágenes originales a distintas resoluciones implica un coste computacional adicional.
    - Este coste es principalmente debido a la aplicación de un filtrado paso bajo para reducir el *aliasing*.
  - La pérdida de información de alta frecuencia (bordes) en las imágenes de menor resolución puede hacer que se ignoren detalles que harían más fiable la estimación.



6. Estimación con precisión fraccionaria

- Hasta ahora, las posibles posiciones de los bloques vienen dadas por la malla de muestreo (las posiciones de los píxeles).
  - En estos casos se habla de **estimación con precisión entera**.
- Sin embargo, los movimientos reales de los objetos no están limitados a esa resolución → Aumentando la resolución se podría mejorar la predicción.
  - En estos casos se habla de **estimación con precisión fraccionaria** o **sub-píxel**.

- **<u>Inconvenientes:</u>**
  - Se requieren muchas más operaciones por imagen → Mayor coste computacional.
  - Se necesitan más bits para representar los vectores de movimiento → Mayor coste de memoria.

➢ Hay estudios que demuestran que la estimación con precisión fraccionaria sólo compensa si se trabaja a nivel de ½ **píxel → obsoletos**

6. Estimación con precisión fraccionaria

- **Alternativas:**

1. Aumentar la resolución de la imagen actual y la de referencia → Interpolación.
   - Sobre las imágenes resultantes se aplicaría cualquiera de las técnicas que hemos visto.
   - Imágenes más grandes → Mayor coste de memoria.
   - Interpolación → Mayor coste computacional.

2. Métodos en 2 pasos:
   - Evitan la necesidad de aumentar la capacidad de almacenamiento.
   - Reducen el número de comparaciones necesarias.

3. Otros (codificadores modernos).

6.1. Algoritmo en dos pasos

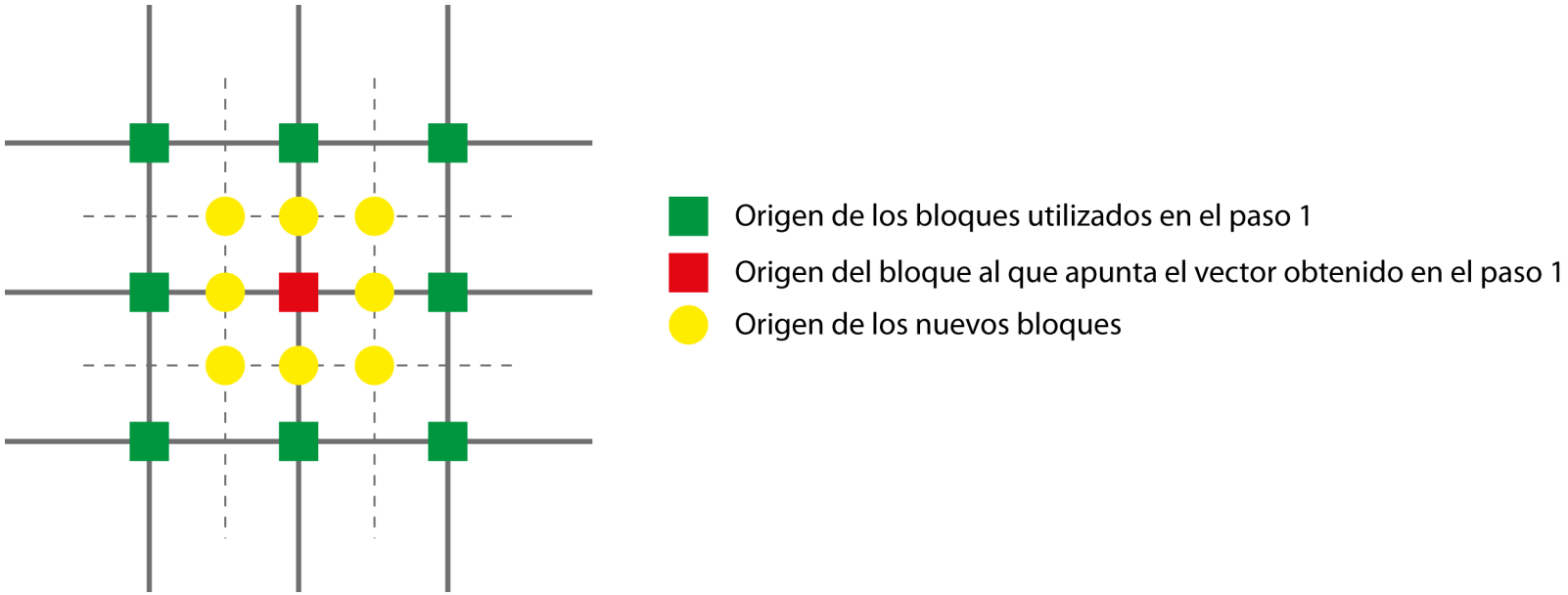
- Es muy común en los estándares de vídeo, ya que permite la precisión de medio píxel como una extensión de la precisión entera (no supone un aumento excesivo del coste computacional).
- Para cada bloque de la imagen original se realizan dos pasos (de ahí el nombre del algoritmo).

- **Paso 1:**
  - Se obtiene un vector de movimiento utilizando precisión entera y con cualquiera de los algoritmos que hemos visto previamente.
  - El resultado será un vector (u,v) con valores enteros dentro del rango permitido por el área de búsqueda considerada.

6.1. Algoritmo en dos pasos

- **Paso 2: Refinamiento del vector obtenido en el paso anterior.**
  - Se analizan 8 nuevos bloques de la imagen de referencia, cuyos orígenes se encuentran situados en las coordenadas:

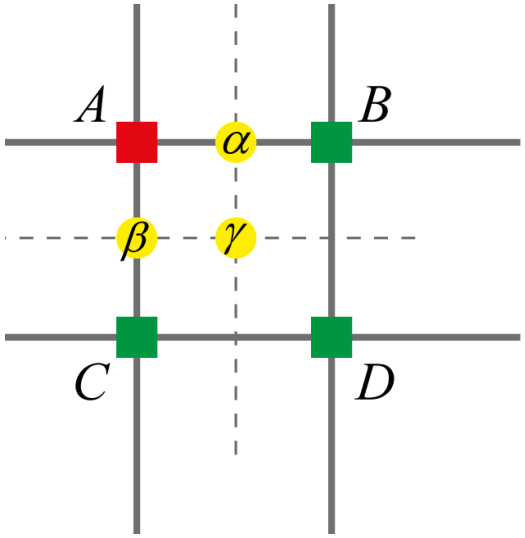
$$\left(u+\frac{m}{2}, v+\frac{n}{2}\right), \quad \text{donde } m,n \in [-1,0,1] \text{ y } (m,n) \neq (0,0)$$



6.1. Algoritmo en dos pasos

- **Paso 2: Refinamiento del vector obtenido en el paso anterior.**
  - Se evalúa la función de coste para los nuevos bloques, para lo que hará falta calcular los valores de los píxeles en estos bloques. Normalmente se usa la interpolación bilineal.
    - En el caso de la precisión de medio píxel, estos valores se calculan como:

$$\alpha = \frac{A+B}{2} \qquad \beta = \frac{A+C}{2} \qquad \gamma = \frac{A+B+C+D}{4}$$



Origen de los bloques utilizados en el paso 1

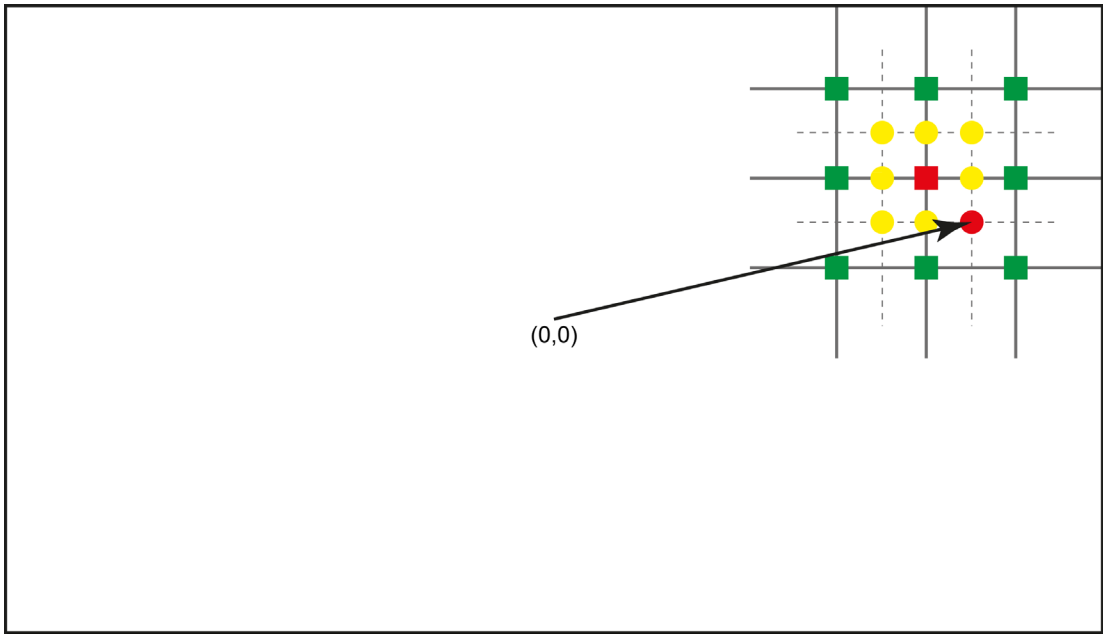
Origen del bloque al que apunta el vector obtenido en el paso 1

Origen de los nuevos bloques

6.1. Algoritmo en dos pasos

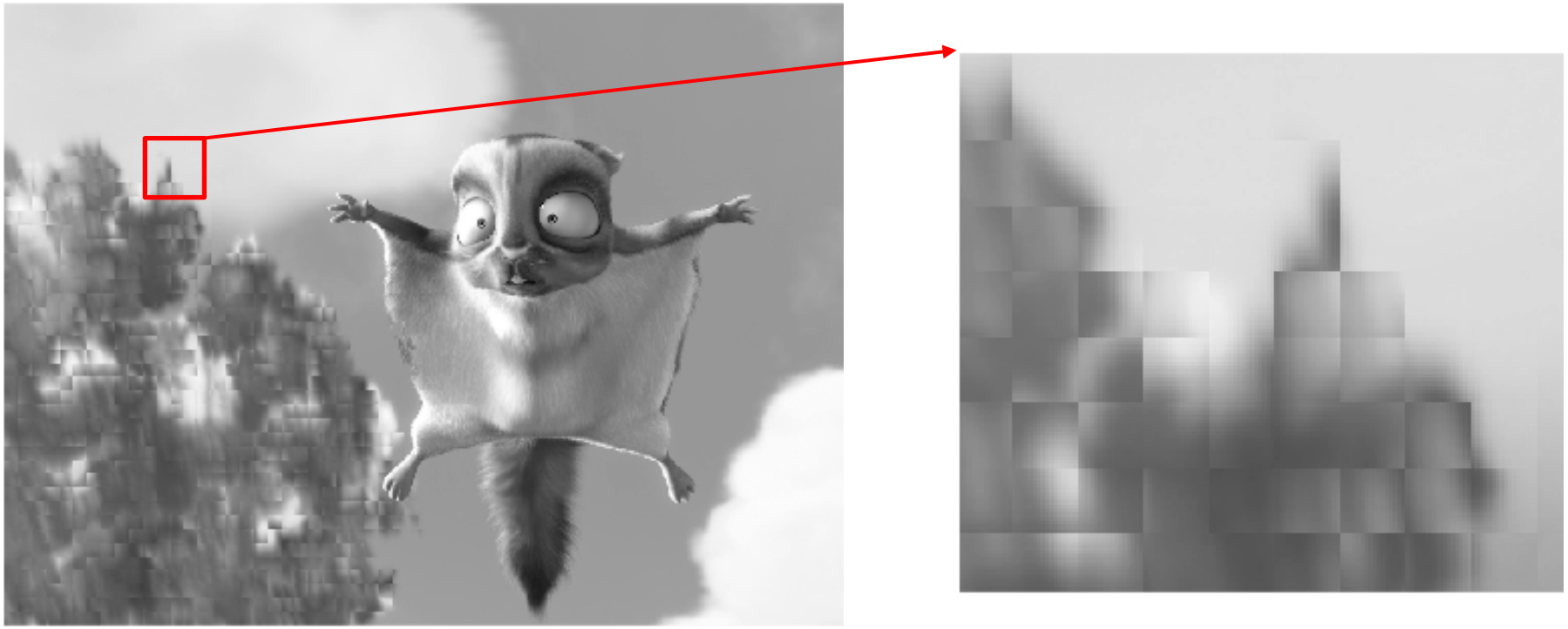
- **Paso 2: Refinamiento del vector obtenido en el paso anterior.**
  - El vector de movimiento final será el que minimice la función de coste (volviendo a incluir el resultado correspondiente al bloque de coordenadas (u,v)).
  - Si el menor error se da para el par (m,n) = (k,l), el vector de movimiento final será:

$$\left(u+\frac{k}{2},v+\frac{l}{2}\right)$$



7. Métodos avanzados de compensación de movimiento

- Existen alternativas que tratan de mejorar los resultados de los métodos convencionales:
  - No se limitan al análisis de movimientos traslacionales entre bloques.
  - Reducen el efecto de bloques.



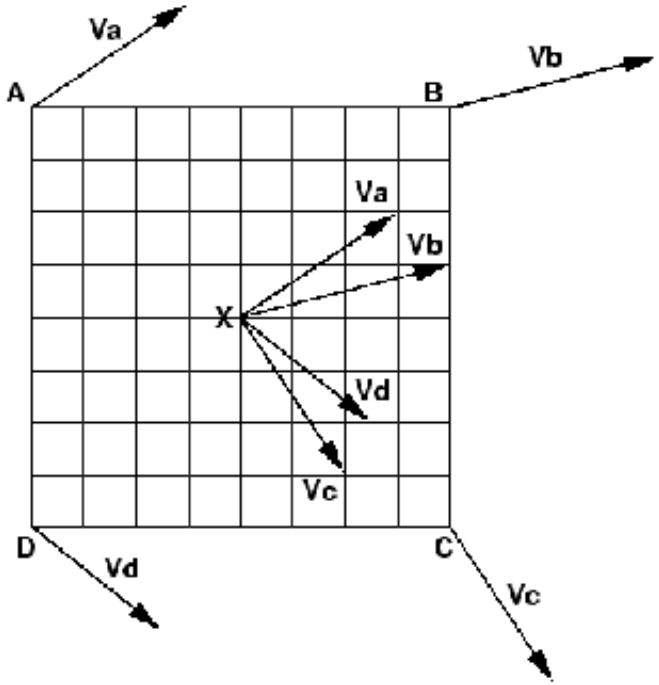
7. Métodos avanzados de compensación de movimiento

- **Algunos de los métodos más populares:**
  - Compensación de movimiento por solapamiento de bloques.
  - Compensación por interpolación de la rejilla de control.

- **Características comunes:**
  - Descripción muestreada del campo de vectores de movimiento estimados:
    - Se transmitirá un conjunto de vectores capaz de representar el movimiento en la imagen.
  - Se asume que el movimiento es continuo:
    - No son capaces de hacer frente a discontinuidades.
  - No se tiene en cuenta ningún tipo adicional de información de la imagen.

7.1. Compensación por solapamiento de bloques

- **Funcionamiento:**
  - Para cada punto de la imagen se obtienen diferentes predicciones a partir de los vectores de movimiento asociados a los puntos más próximos (determinados por una rejilla regular).
  - La estimación final se obtiene promediando las predicciones.



7.1. Compensación por solapamiento de bloques

- **Error de predicción:**

$$I_n^e(x,y) = I_n(x,y) - \sum_{i \in C(x,y)} \alpha_i(x,y)\,\tilde{I}_{n-r}(x-u_i, y-v_i)$$

$I_n(x,y)$ → Intensidad del píxel de coordenadas (x,y) en la imagen n

$\tilde{I}_{n-r}$ → Imagen de referencia reconstruida

$C(x,y)$ → Píxeles de la rejilla considerados

(u_i, v_i) → Vectores de movimiento asociados a dichos píxeles

$\alpha_i(x,y)$ → Coeficientes de ponderación

7.1. Compensación por solapamiento de bloques

- **Consideraciones:**

  - Al depender el error de un promedio de bloques:
    - Se reduce el efecto de bloques.
    - Aumenta el coste computacional.

  - A operar sobre sobre bloques, resulta compatible con la posterior etapa de codificación del error de predicción (DCT) .

7.2. Compensación por interpolación de la rejilla de control

- **Funcionamiento:**
  - Se parte de la hipótesis de que la predicción de la imagen que se desea codificar se puede obtener como una deformación de la imagen de referencia.
  - Únicamente se establecen correspondencias entre las dos imágenes (con vectores de movimiento) para algunos puntos en concreto de la imagen actual.
    - Estos puntos se denominan puntos de control y se establecen mediante una rejilla predefinida (rectángulos o triángulos).
  - Las correspondencias asociadas al resto de puntos de la imagen se obtienen interpolando los vectores obtenidos del análisis de los puntos de control.
  - Los vectores de movimiento asociados a los puntos de control se obtienen tratando de minimizar la función de coste sobre toda la imagen (cada vector depende de los vectores adyacentes):
    - Son algoritmos muy complejos y muy costosos.