

Máster Universitario en  
Nuevas Tecnologías en Informática

Asignatura “Visión Artificial”

# Extracción de características II:

*Transformada de Hough (HT)*

*Random Sample Consensus (RANSAC)*

*Scale Invariant Feature Transform (SIFT)*

*Maximally Stable Extrema Regions (MSER)*

Facultad de Informática  
Universidad de Murcia  
Curso 2018/19

# Extracción de características (continuación)

- Veremos algoritmos algo más avanzados, y muy populares entre la comunidad de visión:
  - La **HT** (*Hough transform*): un detector de modelos más o menos complejos en un determinado espacio de entrada.
  - **RANSAC** (*Random Sample Consensus*): una técnica de estimación estadística robusta genérica.
    - RANSAC Y HT usados inicialmente para detectar *rectas en imágenes...*
    - ...pero en realidad son técnicas de estimación estadística robusta generales extrapolables a otros muchos problemas de visión.
  - El extractor de características (y descriptor asociado) invariante a escala más potente: el **SIFT** (*Scale Invariant Feature Transform*).
  - Un extractor de regiones (invariantes afines) eficiente: el **MSER** (*Maximally Stable Extrema Regions*).

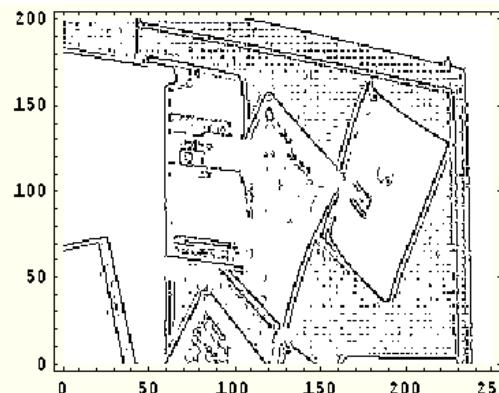
# HT / RANSAC:

## Búsqueda robusta en espacios paramétricos

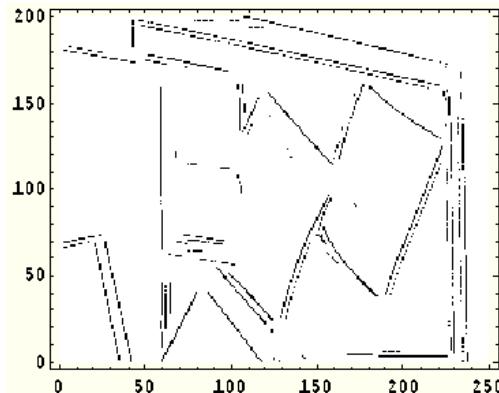
- Ajuste de *features* como búsqueda en espacio paramétrico:
  - Escogemos un modelo parametrizado para representar un *modelo*, y buscamos instancia(s) de ello(s) en unos datos de entrada
  - **Ejemplo:** búsqueda de segmentos en imágenes binarias de borde, con posibles *outliers*.



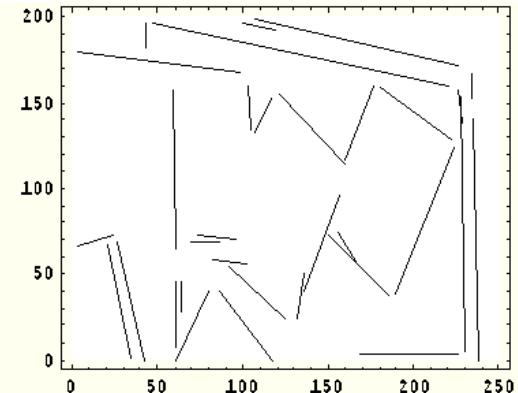
Original



Puntos de borde



Inliers



Segmentos

# HT / RANSAC:

## Búsqueda robusta en espacios paramétricos

- El criterio de búsqueda **NO** es local:
  - No se sabe si un punto pertenece o no a un modelo sin mirar más puntos.
- Cuestiones:
  - **Asociación:** ¿Cuántas instancias del modelo hay en los puntos de entrada, y qué instancias del modelo se corresponden con cada punto de entrada?
  - **Estimación:** ¿Qué modelo representa mejor a un conjunto de puntos de entrada?

# Transformada de Hough (I)

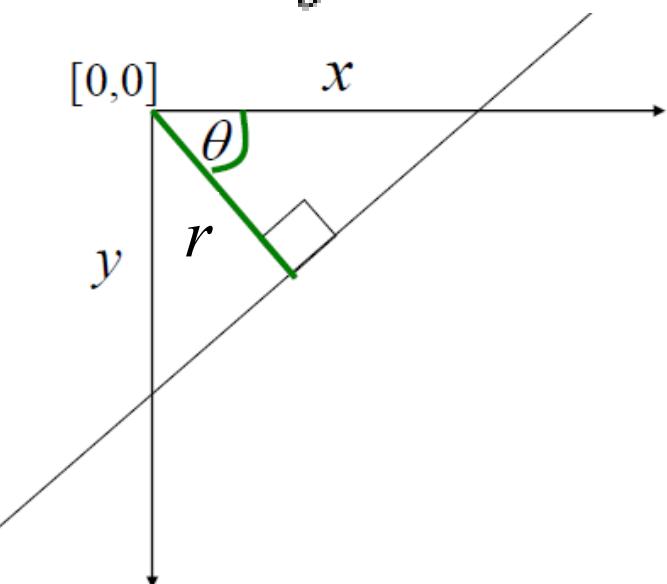
- Hough (1964), Duda & Hart (1972), Ballard (1981).
- El propósito de esta técnica es encontrar instancias imperfectas de objetos de una determinada forma parametrizable, mediante un **esquema de votación**.
- Idea principal:
  - Registrar todas las posibles rectas (modelos) en las que podría estar cada punto.
  - Buscar rectas (modelos) que obtienen muchos votos.
- La votación se lleva a cabo en el **espacio de parámetros** (=acumulador), y las formas candidatas serán **máximos de dicho espacio**, convenientemente discretizado.

# Detección de rectas con HT

- Caso más simple: detección de rectas  $y=mx+b$ :
- Para evitar problemas con rectas casi verticales ( $m$  muy grande), se suele cambiar de parametrización:

$$y = \left( -\frac{\cos \theta}{\sin \theta} \right) x + \left( \frac{r}{\sin \theta} \right) \longleftrightarrow r = x \cdot \cos \theta + y \cdot \sin \theta$$

- El parámetro  $r$  representa la distancia de la línea al origen, y  $\theta$  el ángulo que forma con el eje X la normal a la recta trazada desde el origen.
- Animación ilustrativa:



# Detección de rectas con HT

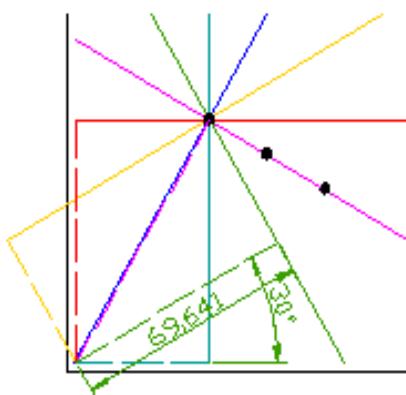
- Un número infinito de líneas pueden pasar por cada punto del plano (*pencil of lines*). Si tenemos un punto  $(x_0, y_0)$  de la imagen, todas las líneas que pasan por él satisfacen la ecuación:

$$r(\theta) = x_0 \cdot \cos \theta + y_0 \cdot \sin \theta$$

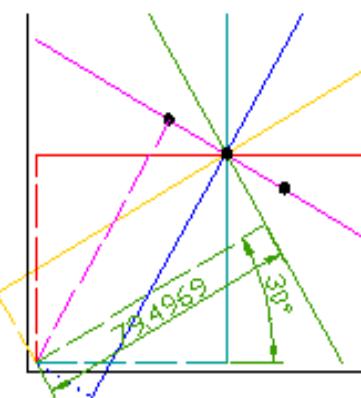
- Esto se corresponde con una curva sinusoidal en el plano  $(r, \theta)$ , única para cada punto.
- Este plano  $(r, \theta)$ , será nuestro espacio paramétrico (donde se realizará la “votación”).

# Detección de rectas con HT

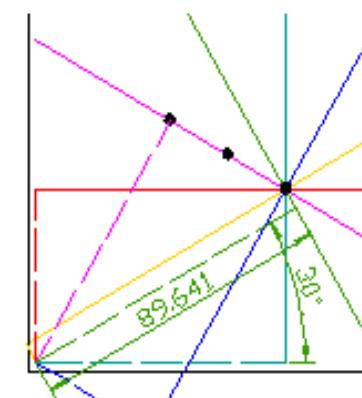
- Transformación *espacio de entrada* → *espacio paramétrico*:



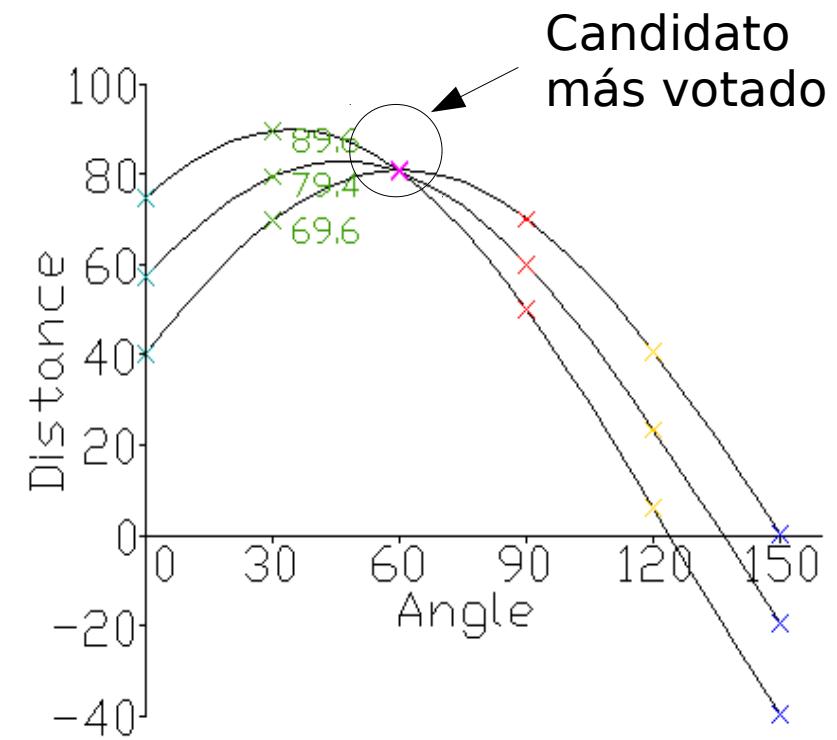
Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4



Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	-19.5



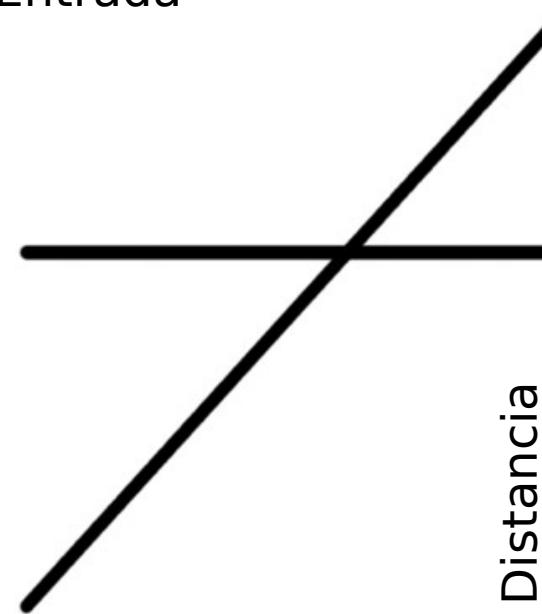
Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	-39.6



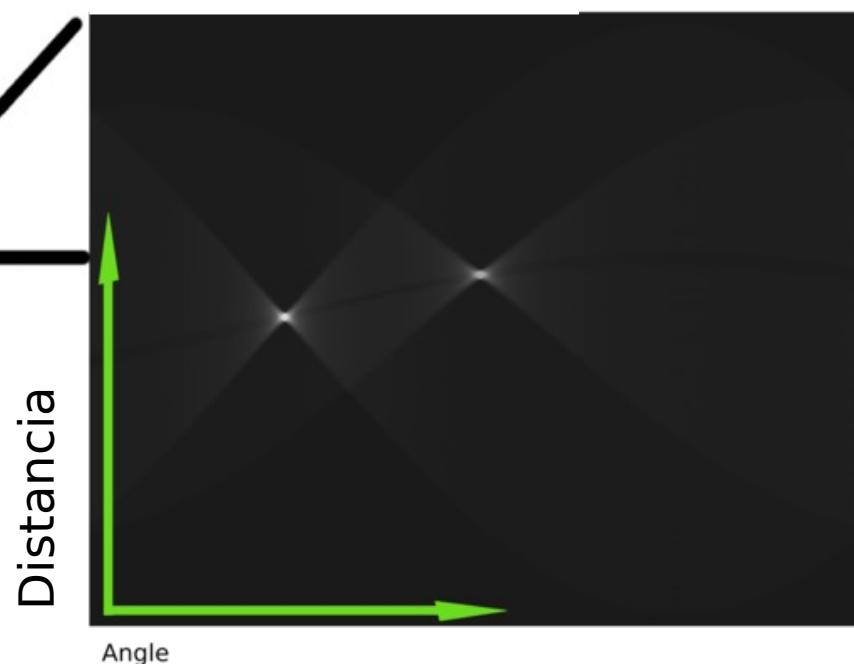
# Detección de rectas con HT

- Ejemplo continuo:

Entrada



Resultado de la HT



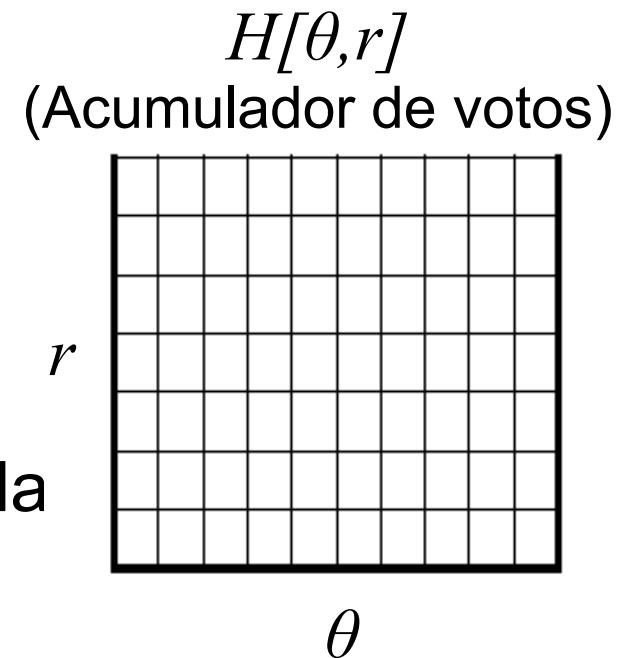
- Vídeo ilustrativo:

<https://www.youtube.com/watch?v=4zHbl-fFIII>

# Algoritmo básico HT

- Esquema:

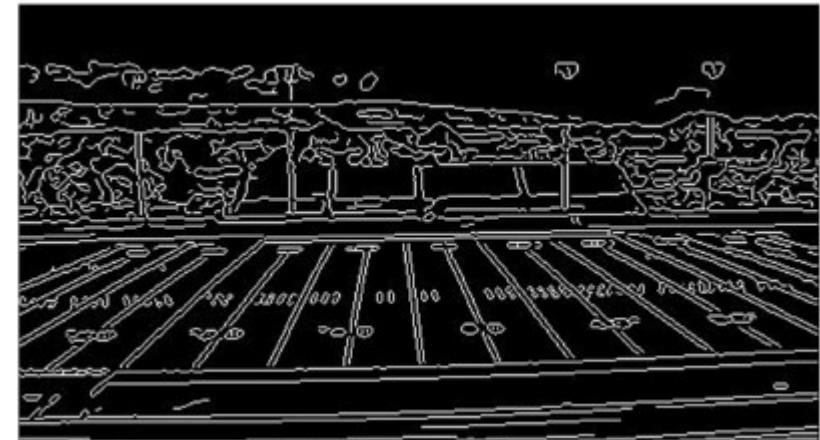
1. Inicializar  $H[\theta,r]=0$ .
2. Para cada punto  $(x,y)$  de la imagen
  - Para cada  $\theta$  (con la cuantización deseada)  
 $H[\theta,r] += 1$ , con  $r = x \cdot \cos\theta + y \cdot \sin\theta$
3. Encontrar máximo local de  $H[\theta,r]$ .
4. La línea  $(\theta_{max}, r_{max})$  se marca como encontrada, y (opcionalmente) se eliminan los votos asociados.
5. Repetir pasos 2-4 hasta que el máximo de votos de la celda  $(\theta_{max}, r_{max}) < umbral$ .



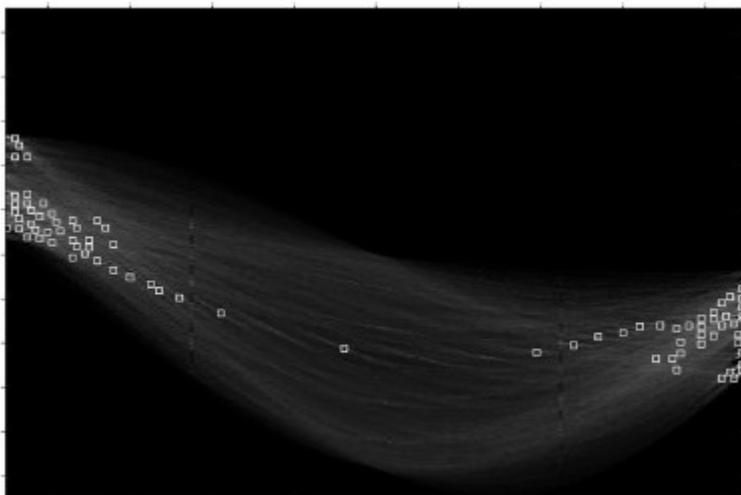
# Ejemplo de resultados HT (detección de segmentos)



Original



Puntos de borde (p.e. Canny)



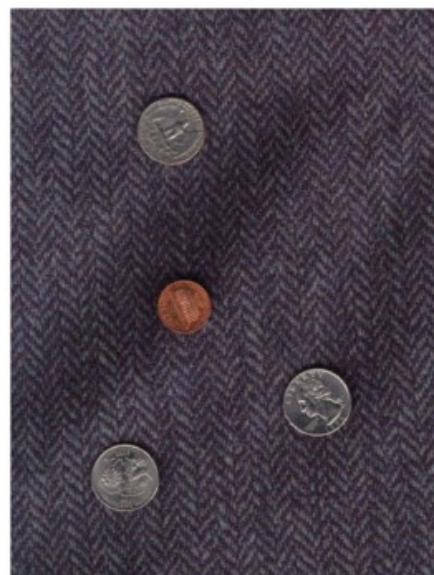
Espacio paramétrico  
de votación



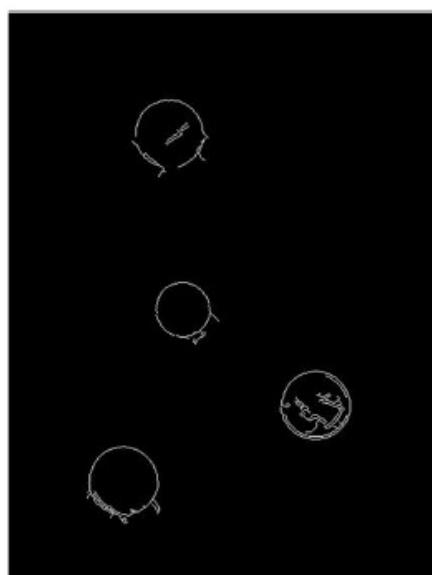
Segmentos (tras acotar proyección de agrupaciones de puntos sobre rectas encontradas)

# Generalización de la HT

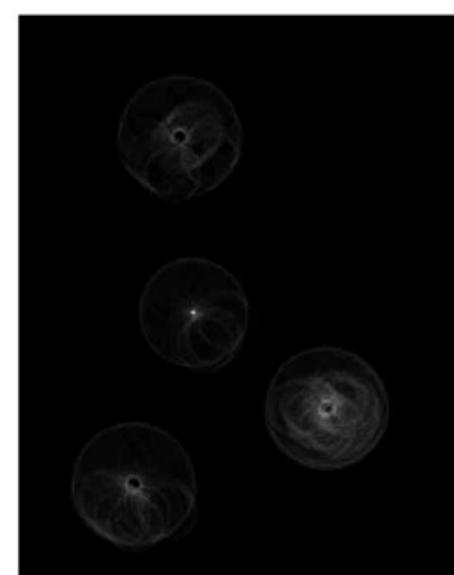
- Fácilmente trasladable a otro tipo de estructuras (círculos, elipses, ...), cambiando el espacio de parámetros.
  - Ojo con el posible crecimiento del espacio paramétrico.
  - Ejemplo detección de círculos (de radio conocido; si no lo es el espacio de parámetros es 3D → más costoso):



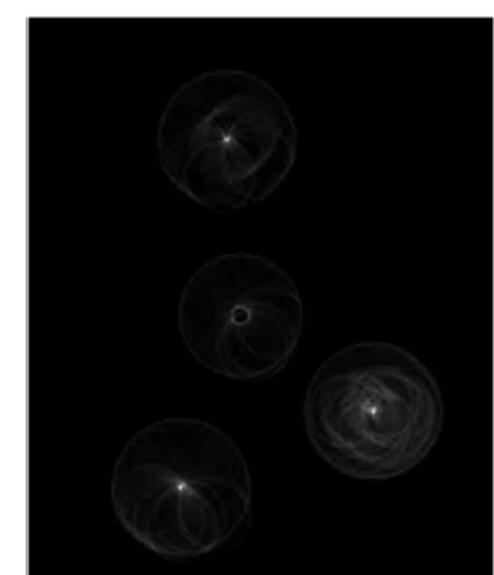
Original



Bordes



HT (radio penique)



HT (radio cuarto de dólar)

# Detalles implementación HT

- A tener en cuenta:
  - **Eficiencia:** un punto de entrada → ¿múltiples votos? → intentar sólo uno (ahora veremos posibilidades).
  - **Discretización** del espacio de parámetros (= espacio de votación):
    - Demasiado “gruesa” → poca precisión en modelos encontrados.
    - Demasiado “fina” → poca agrupación en los votos.
  - A veces, mejor tratar votos simplemente como listas de puntos → necesario **clustering** robusto en el espacio de parámetros.

# Variantes de la HT

- HT con gradiente: un punto + una orientación → un sólo voto.
  - Una variante es la HT conectiva, que también pretende un sólo voto, pero no usando el gradiente, sino agrupaciones locales de píxeles alineados.
- HT probabilística: trabajar sólo sobre un subconjunto aleatorio de los puntos de entrada
  - La solución no varía mucho, y se gana en eficiencia.
- HT aleatoria: Ir cogiendo los puntos de dos en dos, aleatoriamente → Cada par de puntos es un sólo voto.
  - Al iterar este proceso, llega un momento en que una celda excede el umbral → se detecta una recta, se borran los puntos correspondientes y se continúa.

# *Random Sample Consensus* (RANSAC)

- RANSAC: “consenso en muestreo aleatorio”
- Queremos evitar el impacto de los *outliers* → busquemos *inliers* para cada recta.
- Intuición:
  - Si cogemos algún *outlier* y estimamos una recta usándolo, la recta conseguida tendrá “poco soporte” en los datos.

# Algoritmo básico RANSAC

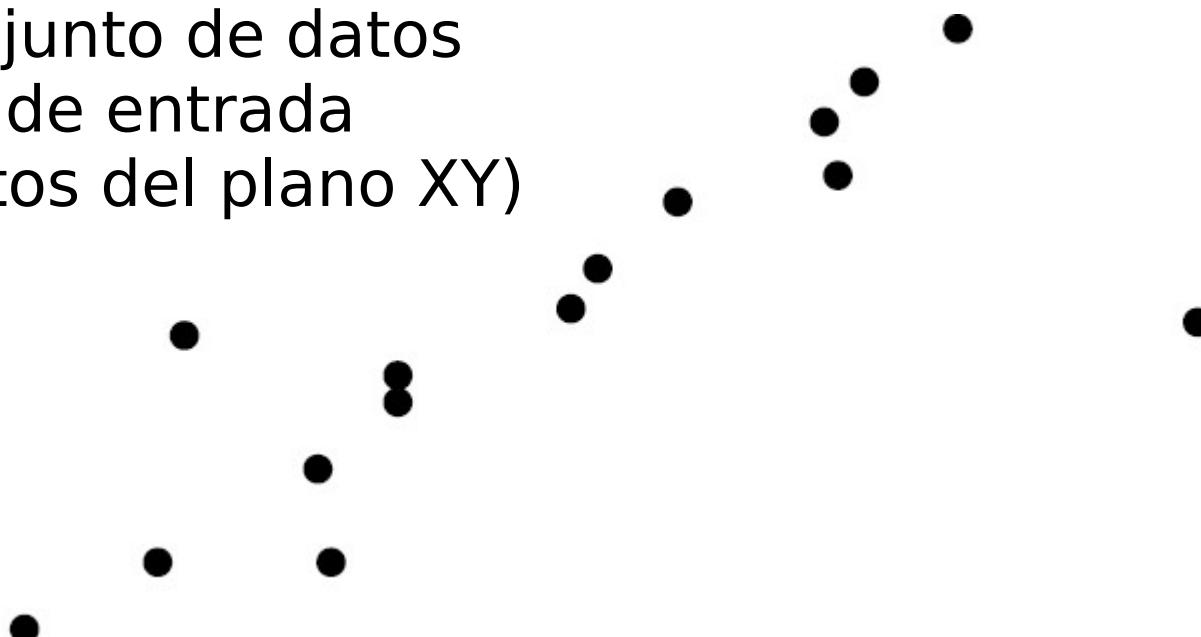
1. Seleccionar aleatoriamente un conjunto de puntos semilla, con el tamaño mínimo necesario para realizar una estimación del modelo (2 puntos  $(x,y)$  para el problema de estimar rectas del plano).
2. Estimar el modelo (recta 2D) a partir del conjunto anterior.
3. Buscar *inliers* del modelo estimado (conjunto de “*puntos de soporte*” bien explicados por la recta encontrada).
4. Si hay un número suficiente de *inliers*, recomputar la recta por mínimos cuadrados usando sólo los *inliers*. En otro caso, volver al paso 1.

(Este algoritmo sólo encuentra un segmento; para localizar más, se borrarían los *inliers* de la recta encontrada, y se volvería a empezar).

# Algoritmo básico RANSAC

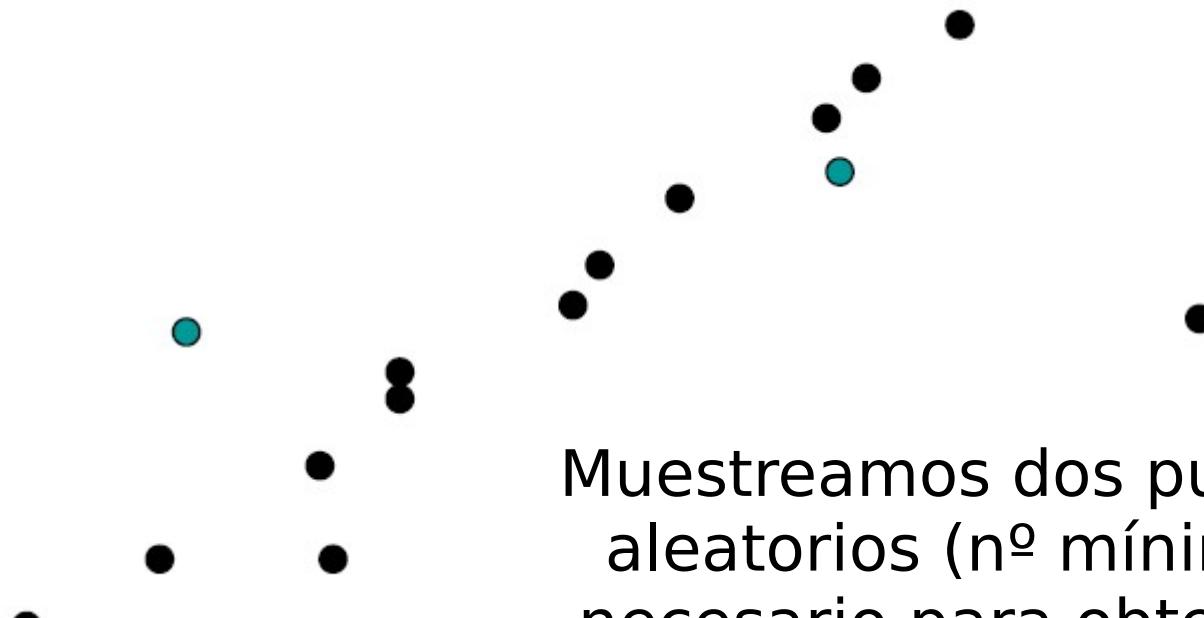
Ejemplo búsqueda recta para explicar conjunto de puntos 2D

Conjunto de datos  
de entrada  
(puntos del plano XY)



# Algoritmo básico RANSAC

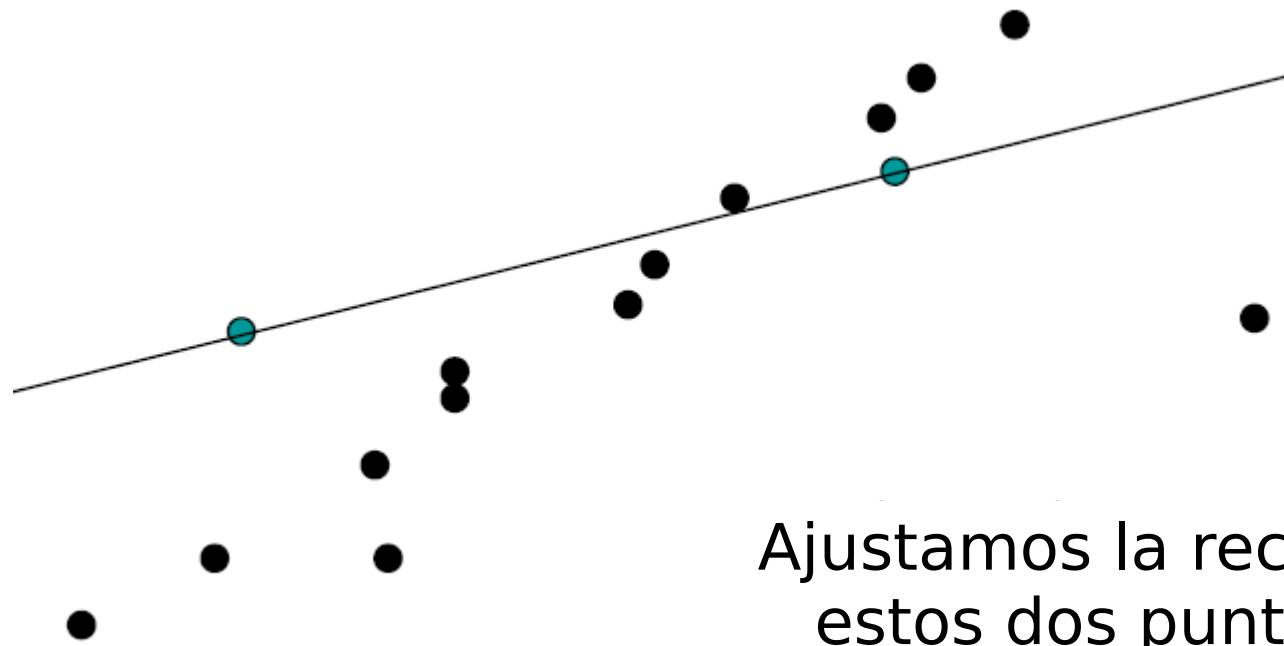
Ejemplo búsqueda recta para explicar conjunto de puntos 2D



Muestreamos dos puntos aleatorios (nº mínimo necesario para obtener una recta única)

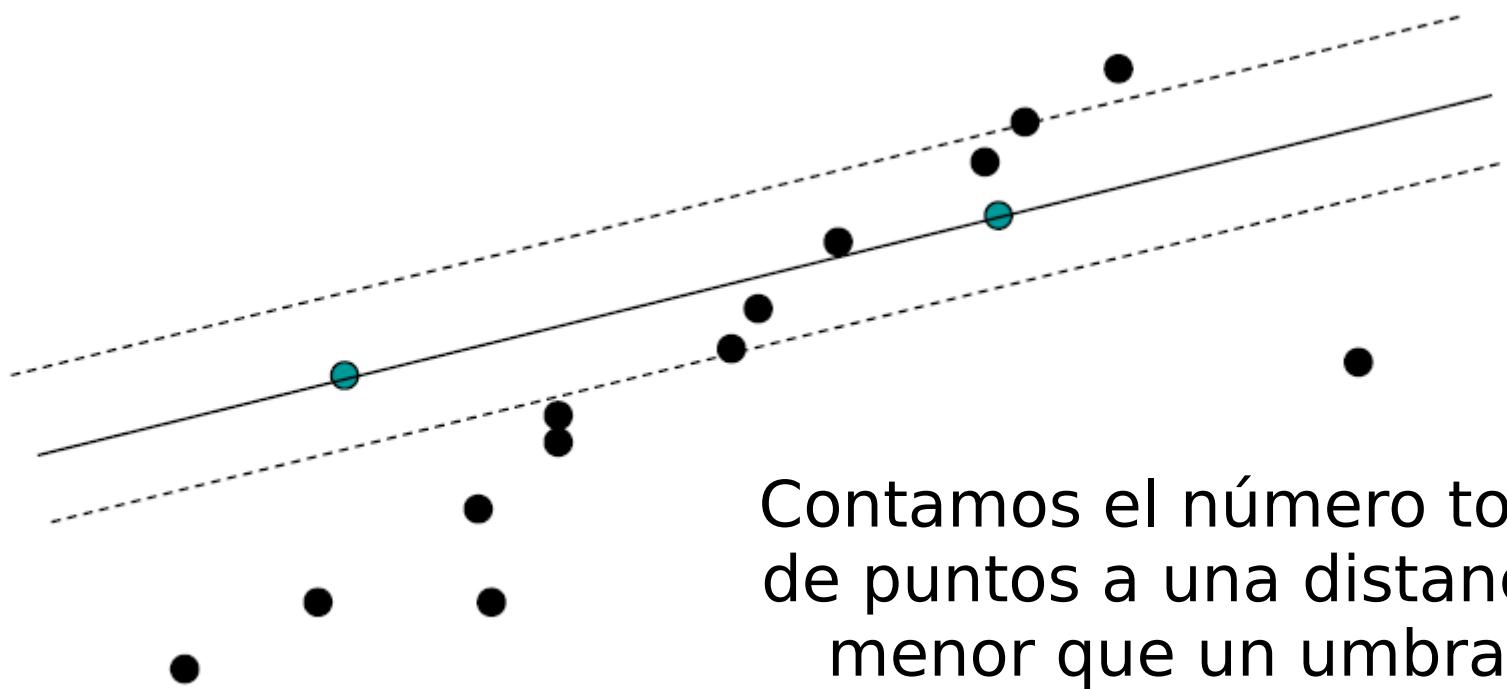
# Algoritmo básico RANSAC

Ejemplo búsqueda recta para explicar conjunto de puntos 2D



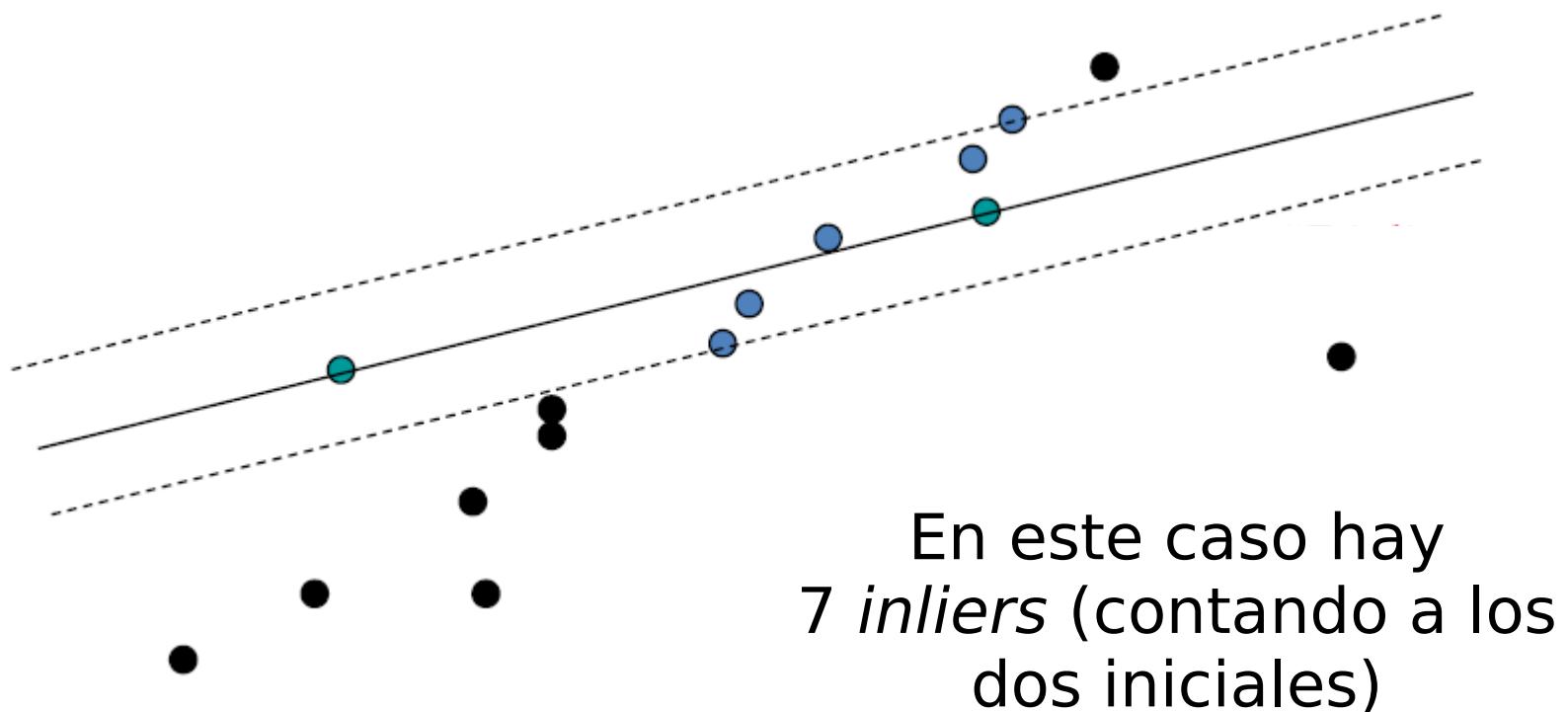
# Algoritmo básico RANSAC

Ejemplo búsqueda recta para explicar conjunto de puntos 2D



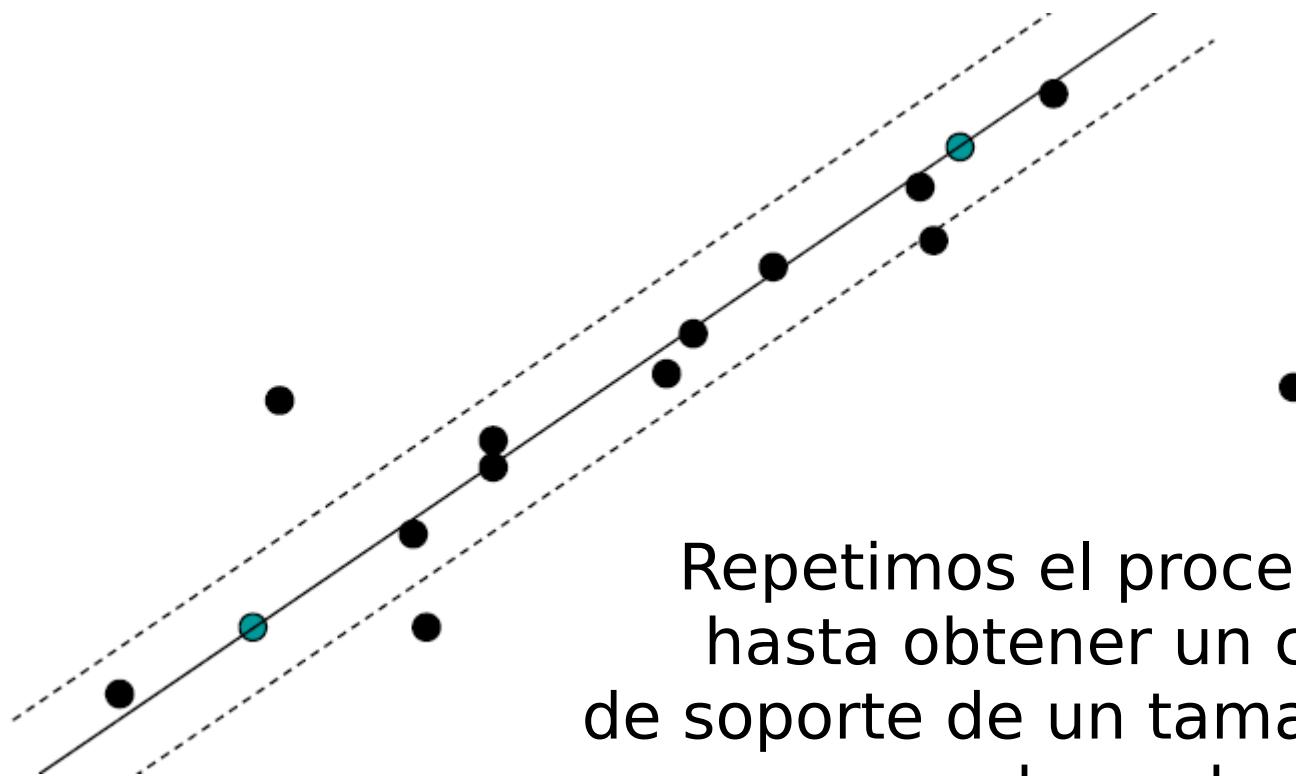
# Algoritmo básico RANSAC

Ejemplo búsqueda recta para explicar conjunto de puntos 2D



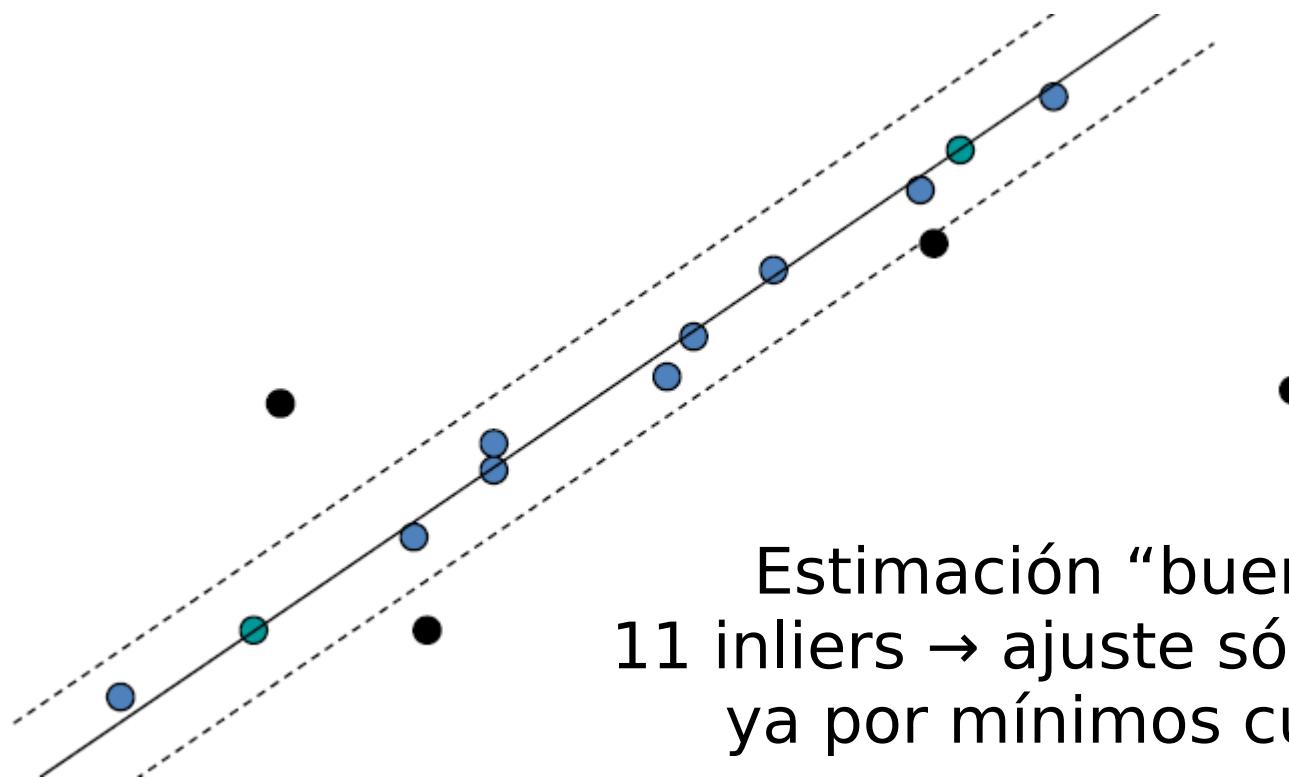
# Algoritmo básico RANSAC

Ejemplo búsqueda recta para explicar conjunto de puntos 2D



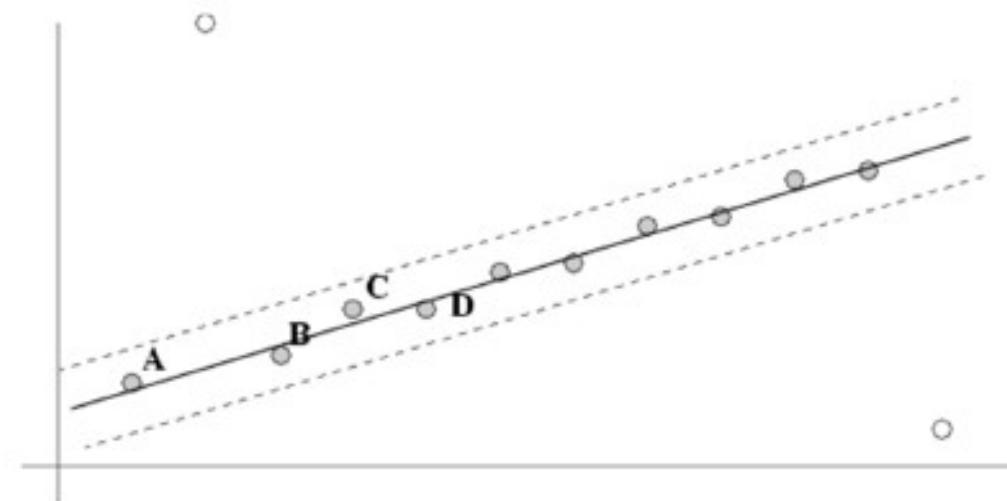
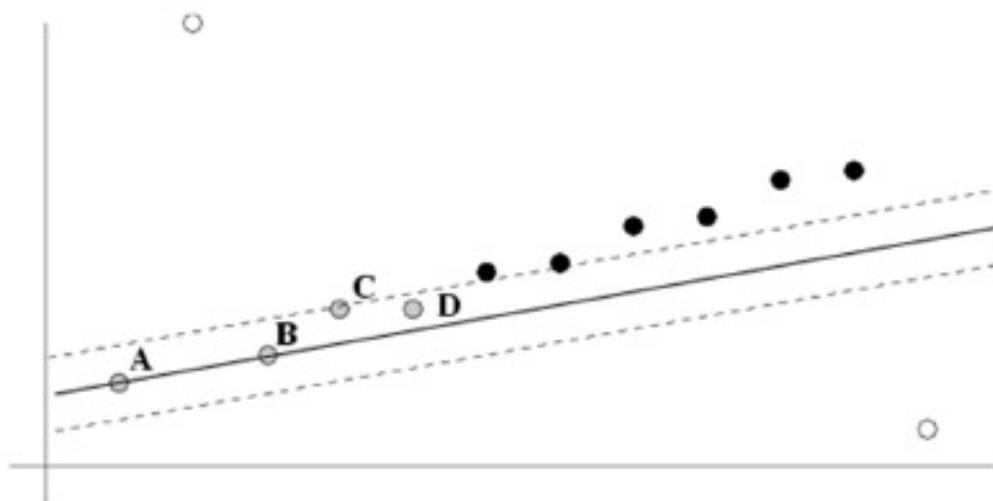
# Algoritmo básico RANSAC

Ejemplo búsqueda recta para explicar conjunto de puntos 2D



# Iteración hasta convergencia

- Incluso después de haber localizado un conjunto de *inliers*, la cantidad de éstos puede variar tras la reestimación por mínimos cuadrados → iterar hasta convergencia:



# Genericidad de RANSAC

- En realidad, el algoritmo anterior estima un modelo plausible para explicar unos datos con posibles *outliers*, independientemente del modelo y los datos concretos.
- P.e., para el caso de la búsqueda de un conjunto de segmentos en una imagen que nos ocupa, cada modelo es un segmento, y los datos son puntos 2D...
- ... pero, p.e., los datos podrían ser *matchings* entre puntos de dos imágenes distintas, e.d., 4-tuplas de la forma  $(x_1, y_1, x_2, y_2)$ , y el modelo a buscar una transformación afín que lleva de una a otra imagen.
  - En ese caso ¿cuál sería el mínimo número de matchings necesarios? → lo veremos más adelante, al ver reconocimiento de objetos con SIFT.
- Por tanto, RANSAC es un potente método genérico de estimación robusta (en el sentido de que puede trabajar con *outliers*).

# Número de muestreos necesarios

- ¿Cuántos muestreos independientes se necesitan?
  - Sea  $w$  [0.0-1.0] la proporción de *inliers*.
  - Sea  $n$  el mínimo número de datos para calcular un modelo (2 en el caso de las rectas a partir de puntos).
  - Sea  $k$  el número de muestreos realizados.
- Entonces:
  - La probabilidad de que un muestreo completo sea correcto es  $w^n$ .
  - La probabilidad de que fallen todos los muestreos es  $(1-w^n)^k \rightarrow$  hay que coger  $k$  suficientemente grande para que dicho valor esté por debajo de un umbral deseado (probabilidad de fallo).

# Número de muestreos necesarios

- Ejemplo muestreos mínimos ( $k$ ) para probabilidad de fallo  $< 0.01$ , para distintos valores de  $n$  y  $w$ :

Tamaño muestra ( $n$ )	Proporción outliers ( $1-w$ )							
	5%	10%	20%	25%	30%	40%	50%	
2	2	3	5	6	7	11	17	Rectas ( $n=2$ )
3	3	4	7	9	11	19	35	
4	3	5	9	13	17	34	72	Homografías proyectivas ( $n=4$ )
5	4	6	12	17	26	57	146	
6	4	7	16	24	37	97	293	
7	4	8	20	33	54	163	588	
8	5	9	26	44	78	272	1177	Stereo ( $n=8$ )

# HT / RANSAC: Ventajas e inconvenientes

- Ambos son algoritmos genéricos de estimación robusta de modelos en presencia de *outliers* en los datos.
- HT tiene los problemas de elección de parámetros, discretización del espacio y crecimiento de éste para modelos con muchas dimensiones...
- ... pero puede soportar % de outliers muy elevados.
- RANSAC no necesita discretización del espacio de parámetros...
- ... pero a mayor % de *outliers* se puede disparar su costo.

# Scale Invariant Feature Transform (SIFT)

- David Lowe, 2004 (primeras versiones en 1999).
- Método de **extracción de *features robusto, invariante a escala***, ...
- ... **más un descriptor asociado** invariante a escala, rotación, cambio de iluminación y ligeras deformaciones.
- En el *paper* original se describe también una aplicación al reconocimiento de objetos en condiciones relativamente incontroladas (escala y posición del objeto completamente desconocidas, *cluttered background*, etc.)

# Motivación

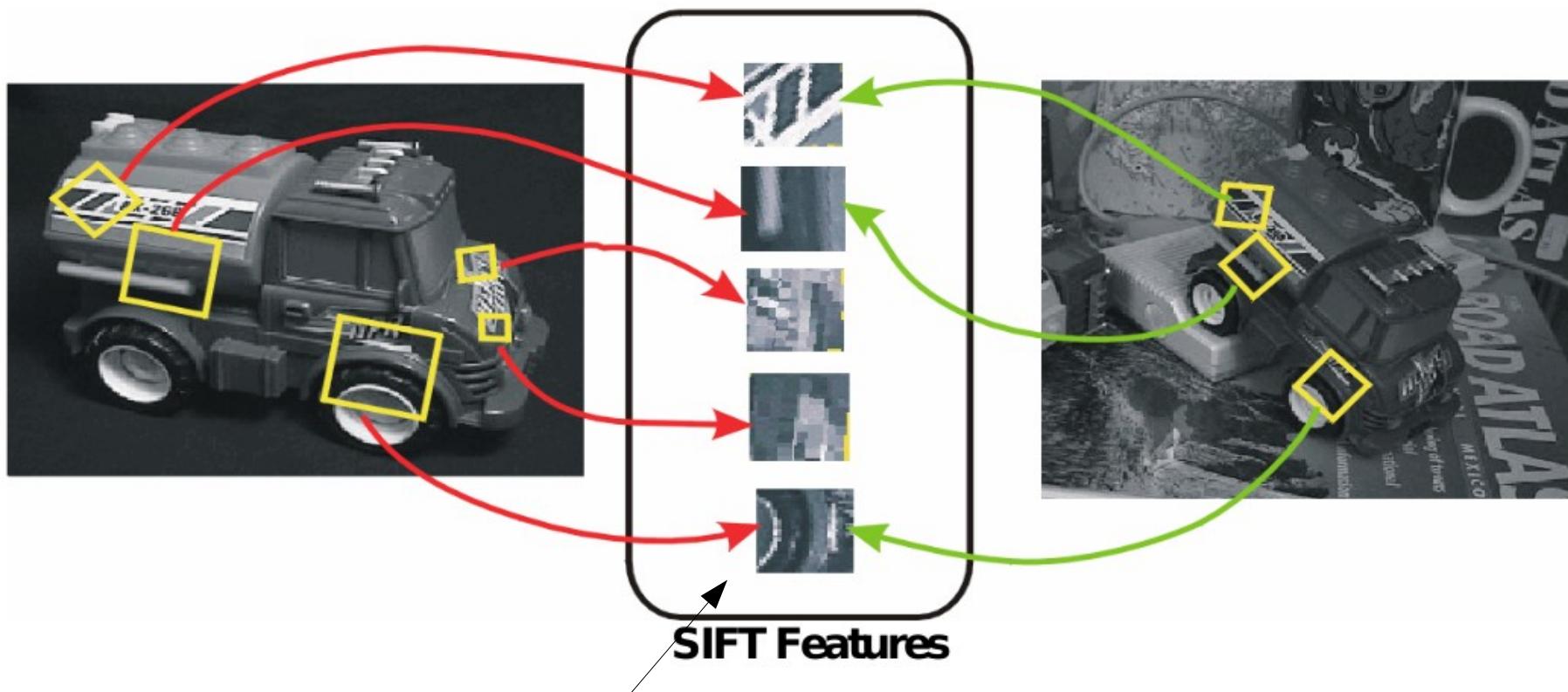
- Ejemplos de localización y reconocimiento de objetos:



- Observar las ocultaciones, los cambios de escala, el *background* desconocido...

# Idea para el reconocimiento de objetos

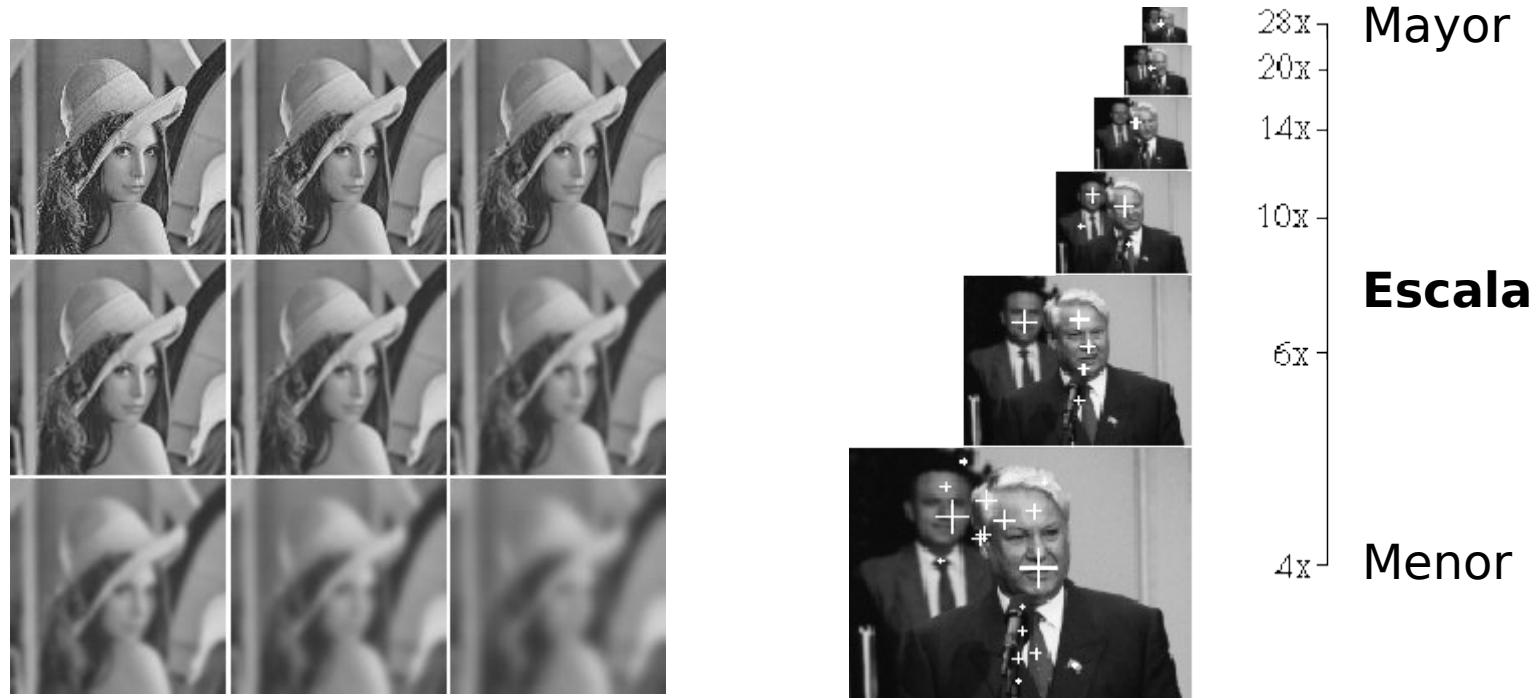
- Matching entre descriptores SIFT “imagen de entrada” ↔ “modelo”:



En realidad, los descriptores serán algo más complejos que el simple parche de imagen, pero aquí se recoge la idea de las invarianzas a escala/rotación

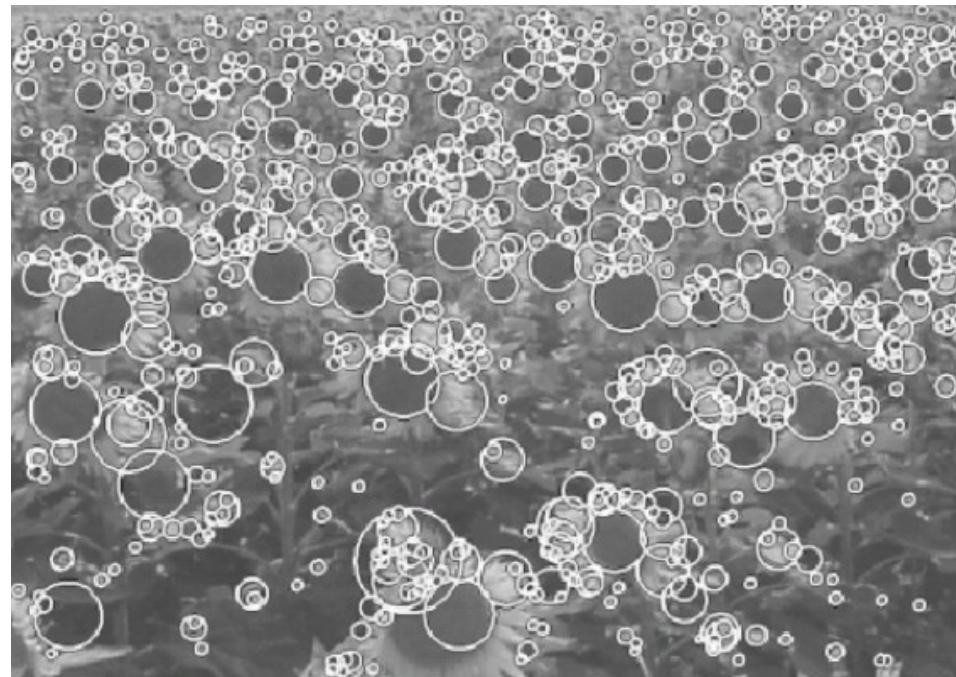
# Detección de extremos en el espacio de escala

- Intentar localizar lugares de la imagen de forma repetible frente a cambios de escala, orientación y luminosidad.
- Aproximación: buscar repetidamente a distintas escalas:



# Detección de extremos en el espacio de escala

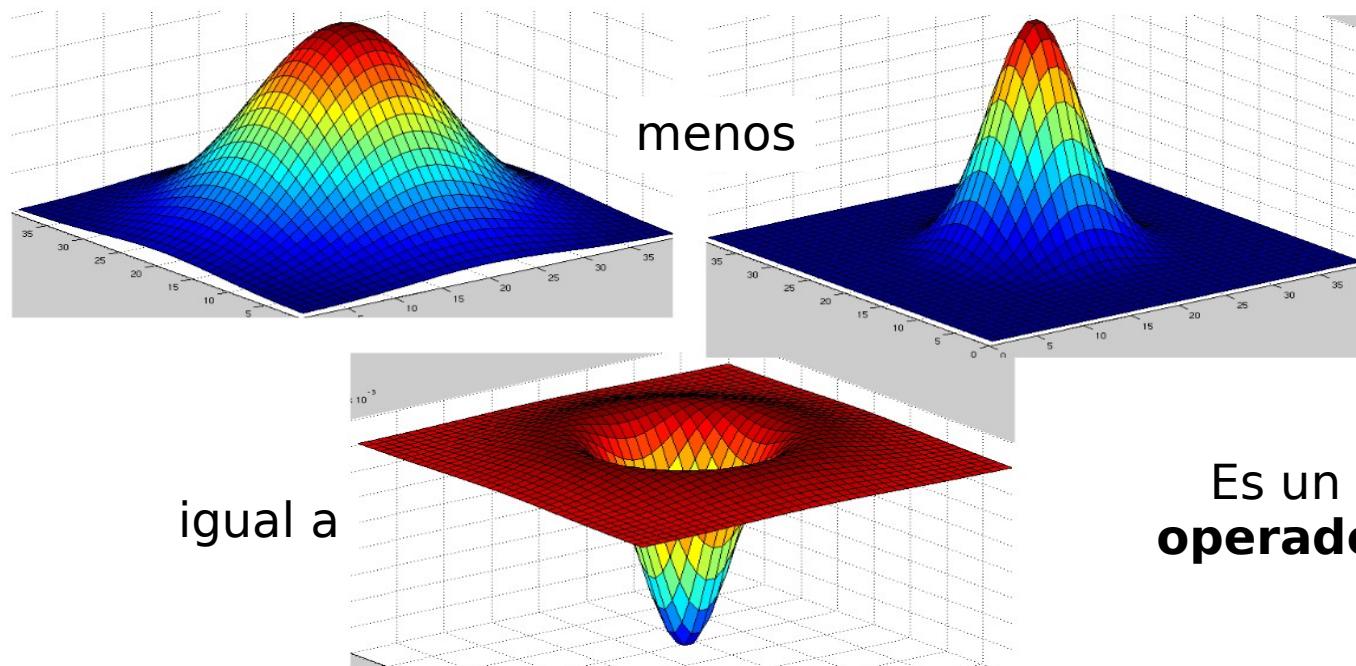
- Nos interesa buscar *blobs* de imagen que sean repetibles a distintas escalas:



- Ojo, espacio de salida 3D ( $X \times Y \times scale$ ).
- También se detectará la orientación característica.

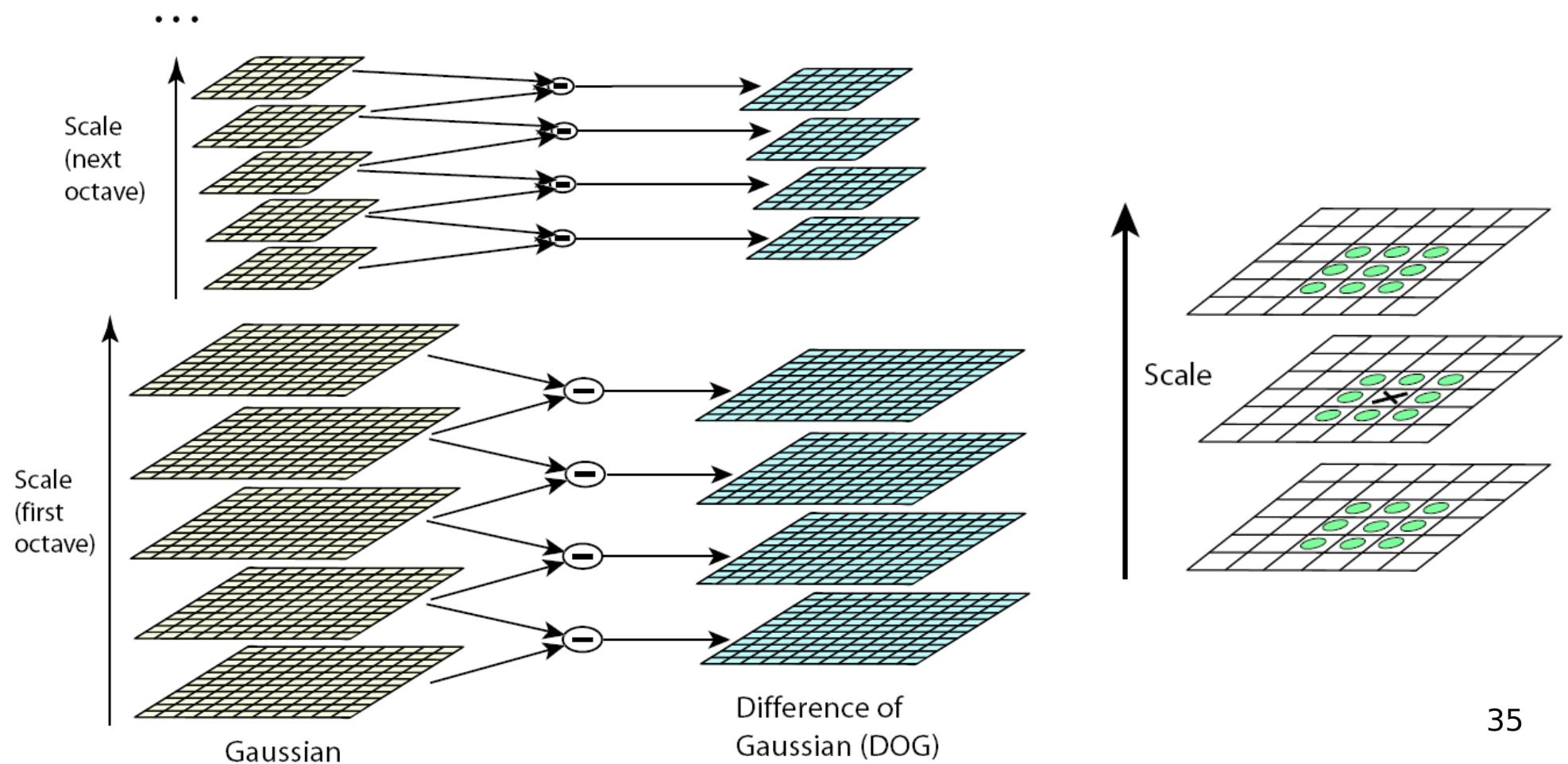
# Paso 1: Detección de puntos clave $(x, y, \sigma)$

- Para forzar la invarianza a escala, realizaremos de un filtro DoG (*Difference of Gaussians*) a distintas escalas, y nos quedaremos con mínimos locales del espacio de salida.
- DoG aproxima LoG (*Laplacian of Gaussian*); (el laplaciano es la suma de las segundas derivadas parciales,  $D_{xx} + D_{yy}$ ):



# Paso 1: Detección de puntos clave (x,y, $\sigma$ )

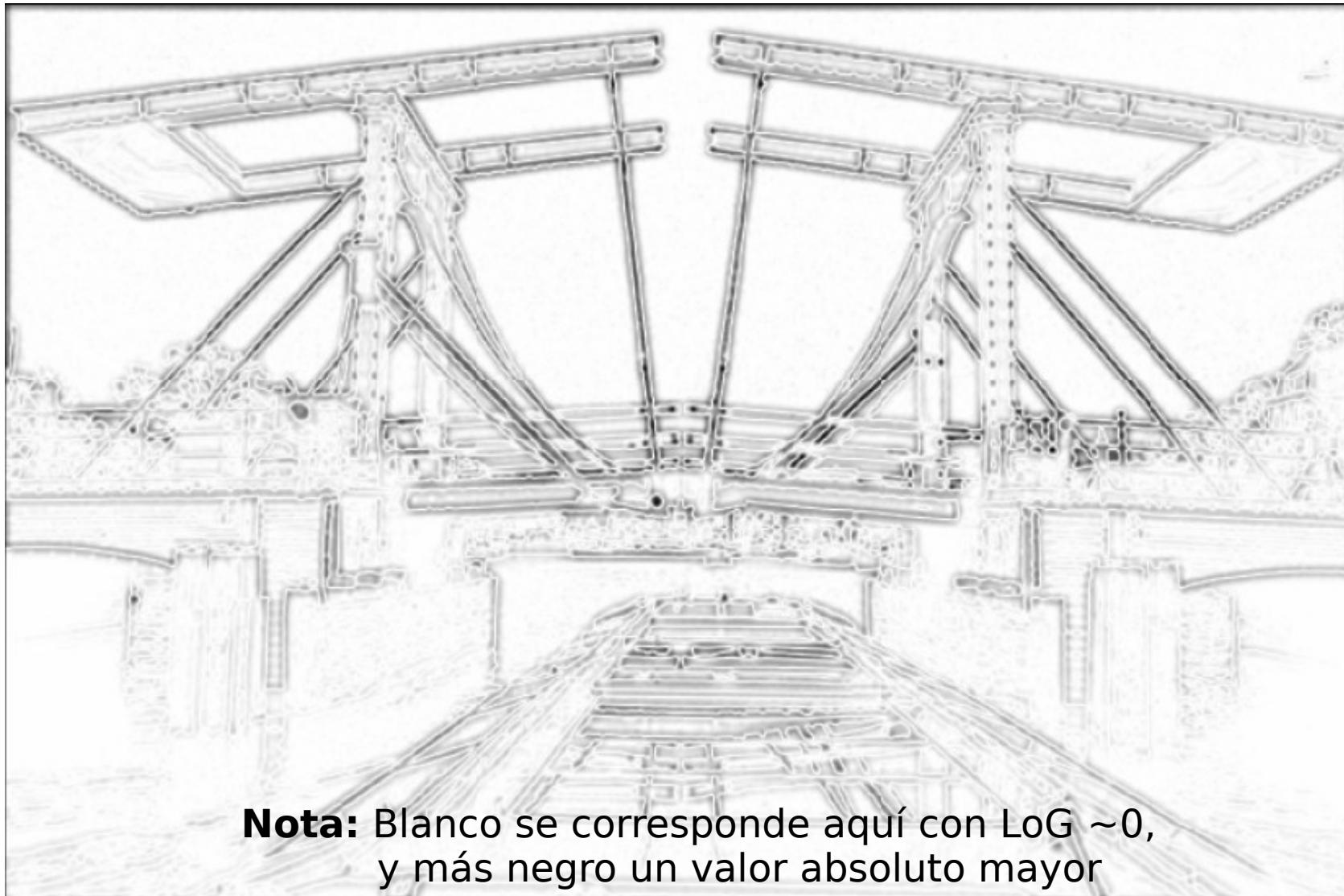
- Localización de máximos en espacio de escala:



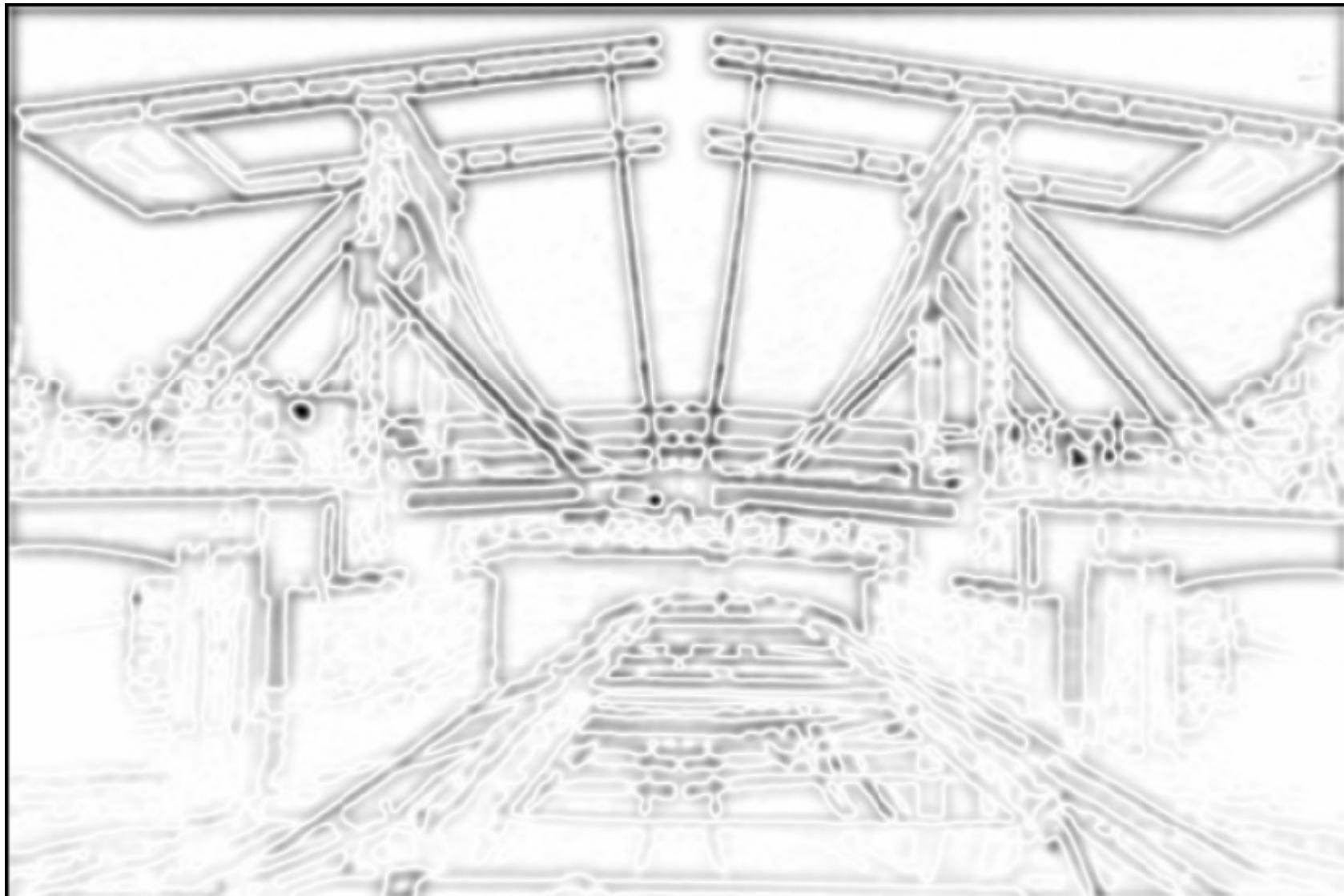
# Espacio de escala de la LoG (imagen original)



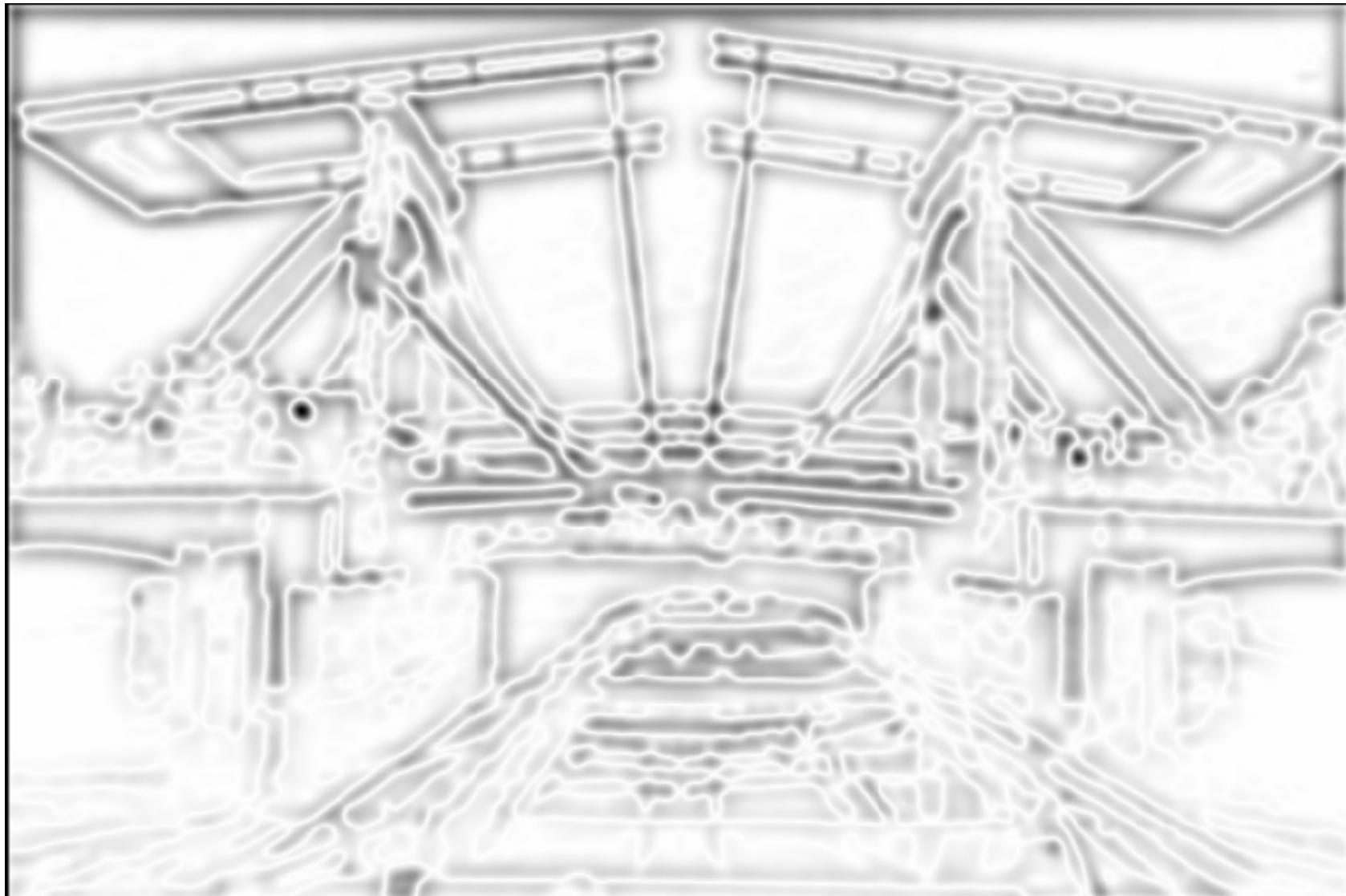
# Espacio de escala de la LoG (suavizado progresivo)



# Espacio de escala de la LoG (suavizado progresivo)



# Espacio de escala de la LoG (suavizado progresivo)



# Espacio de escala de la LoG (suavizado progresivo)



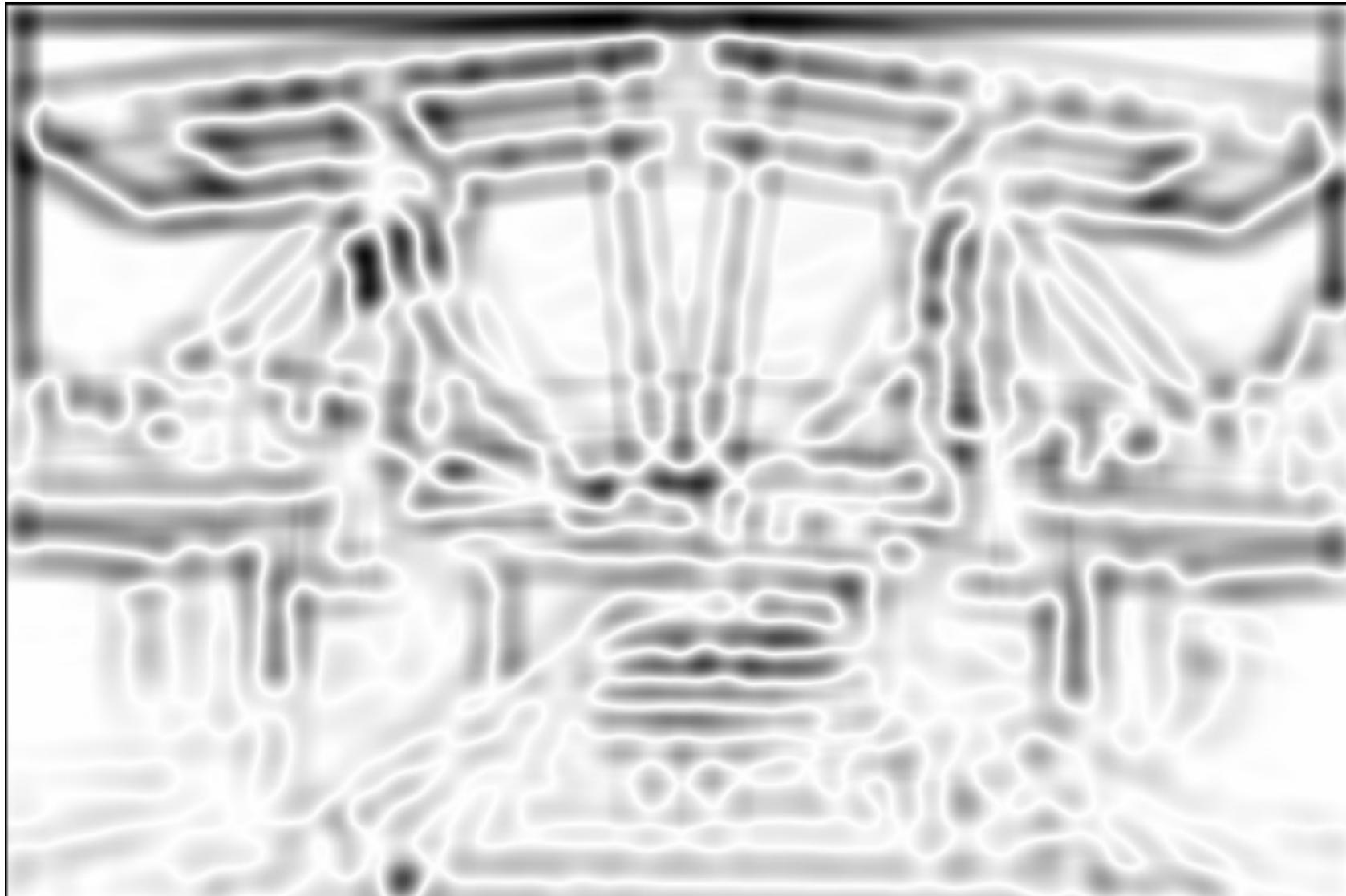
# Espacio de escala de la LoG (suavizado progresivo)



# Espacio de escala de la LoG (suavizado progresivo)



# Espacio de escala de la LoG (suavizado progresivo)



# Espacio de escala de la LoG (suavizado progresivo)



# Espacio de escala de la LoG (suavizado progresivo)

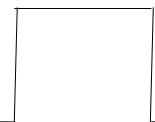


# Espacio de escala de la LoG (suavizado progresivo)

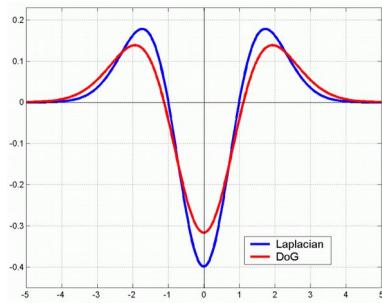


# LoG normalizada a escala

Respuesta LoG  
a un pulso  
cuadrado (1D)

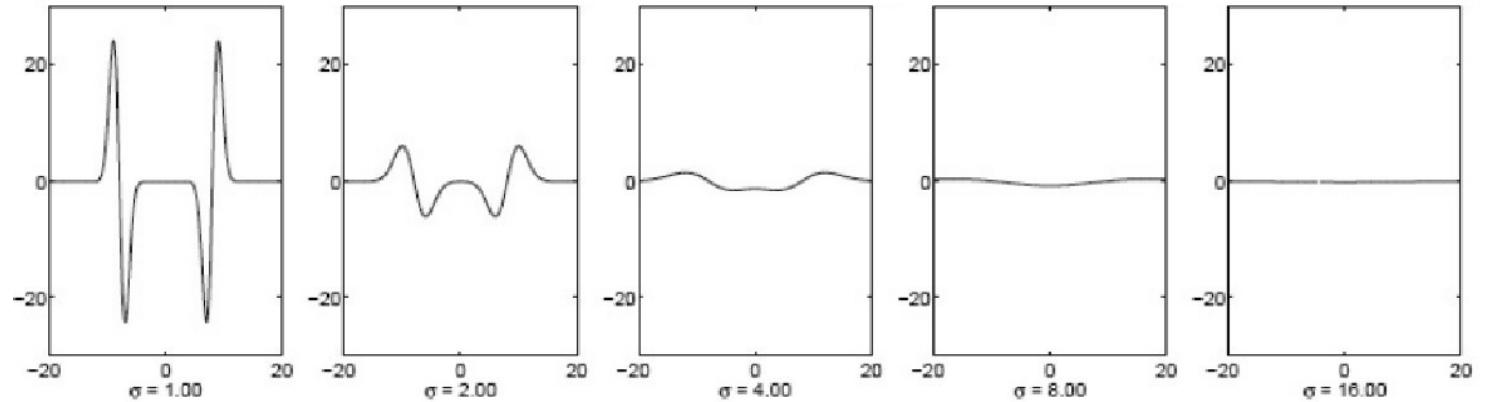


con diferentes  
escalas de  
suavizado

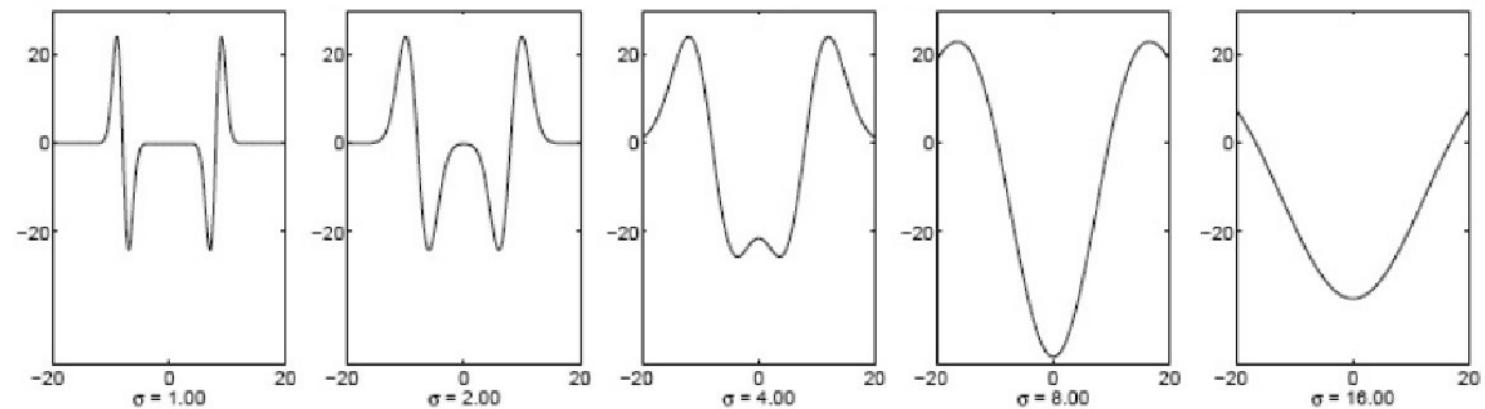


Laplacian/DoG 1D

Respuesta Laplaciano sin normalizar



Respuesta Laplaciano normalizada según escala



# Aproximación LoG mediante DoG

- La función LoG normalizada a escala se aproxima como la DoG de escalas  $\sigma$  y  $k\sigma$  (para  $k$  cerca de 1):

$$G_{k\sigma} - G_\sigma \approx (k - 1)\sigma^2 \nabla^2 G$$

- Escalas sucesivas irán variando en un factor constante  $k$ .
- Cuando la escala  $\sigma$  aumenta mucho, no merece la pena trabajar con la imagen grande → reducimos la imagen a la mitad en X e Y, manteniendo  $\sigma$  acotado (octavas).
- Si hay que estimar  $s$  niveles de máximos de DoG, hay que producir  $s+3$  imágenes suavizadas por octava.
- Un buen valor de  $k$  es, entonces,  $k=2^{1/s}$ , lo que nos llevará a que la imagen suavizada  $s+1$  (antepenúltima) se reduce (submuestrea) para comenzar la siguiente octava.

# Localización de máximos en espacio de escala LoG (resumen)

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

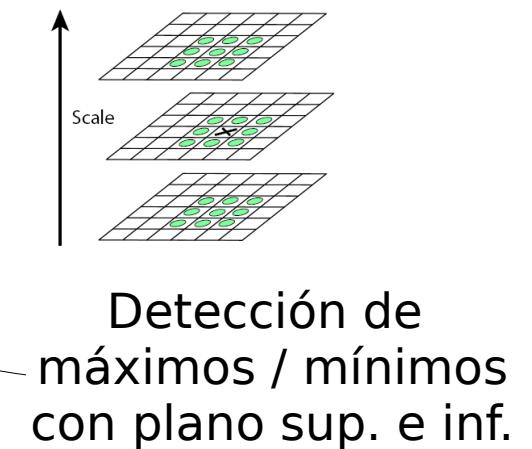
$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned}$$

Filtros Gaussianos (increm.  
 $k\sigma$  en cada paso)

Submuestreo  
( $W/2, H/2$ )

Dif. de Gaussianas  
(DoG)



En este ejemplo:  $s=2 \rightarrow 2+3=5$  suavizados con  $\sigma_0, \sigma_1=2^{1/2}\sigma_0, \sigma_2=2\sigma_0, \sigma_3=2^{3/2}\sigma_0$  y  $\sigma_4=4\sigma_0$ . Aquí se muestran dos octavas, y la imagen (antepenúltima) de cada octava es la que se submuestrearía (pixel sí, píxel no) a la mitad para empezar la siguiente.

# Paso 2: Estimación precisa de la localización

- Precisión subpíxel y continuo de escalas:
  - Para cada máximo/mínimo, ajustar una función cuadrática 3D (Taylor):

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}} \mathbf{x} + \frac{1}{2} \mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2} \mathbf{x}$$

D y sus derivadas se estiman  
directamente en el punto inicial  
 $\mathbf{x} = (x, y, \sigma)$

- Computar la posición exacta del máximo/mínimo con precisión subpíxel derivando e igualando a cero:

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}$$

- El valor de D en dicho extremo se calcula sustituyendo la solución en la primera ecuación:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} \frac{\partial D^T}{\partial \mathbf{x}} \hat{\mathbf{x}}$$

# Paso 3: Eliminar puntos inestables

- Eliminar puntos con un valor absoluto del máximo/mínimo por debajo de un umbral:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} \frac{\partial D^T}{\partial \mathbf{x}} \hat{\mathbf{x}} \quad \begin{array}{l} \text{Si no supera un umbral } |D(\hat{\mathbf{x}})| > 3\% \text{ del valor} \\ \text{máximo de la imagen, se descarta por inestable.} \end{array}$$

- Computar ratio de valores propios del Hessiano (recordar, al estilo del detector de Harris, pero con segunda derivada en lugar de orientación del gradiente), y eliminar puntos con ratio por debajo de un umbral:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Para calcular el ratio  $r$  entre valores propios  $\alpha$  y  $\beta$  de  $\mathbf{H}$  no hace falta la descomposición explícita, bastan la traza y el determinante (bastante más rápido):

$$\text{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\text{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

Despejamos  $r$  (las dos soluciones son una la inversa de la otra)

# Ejemplos de detección

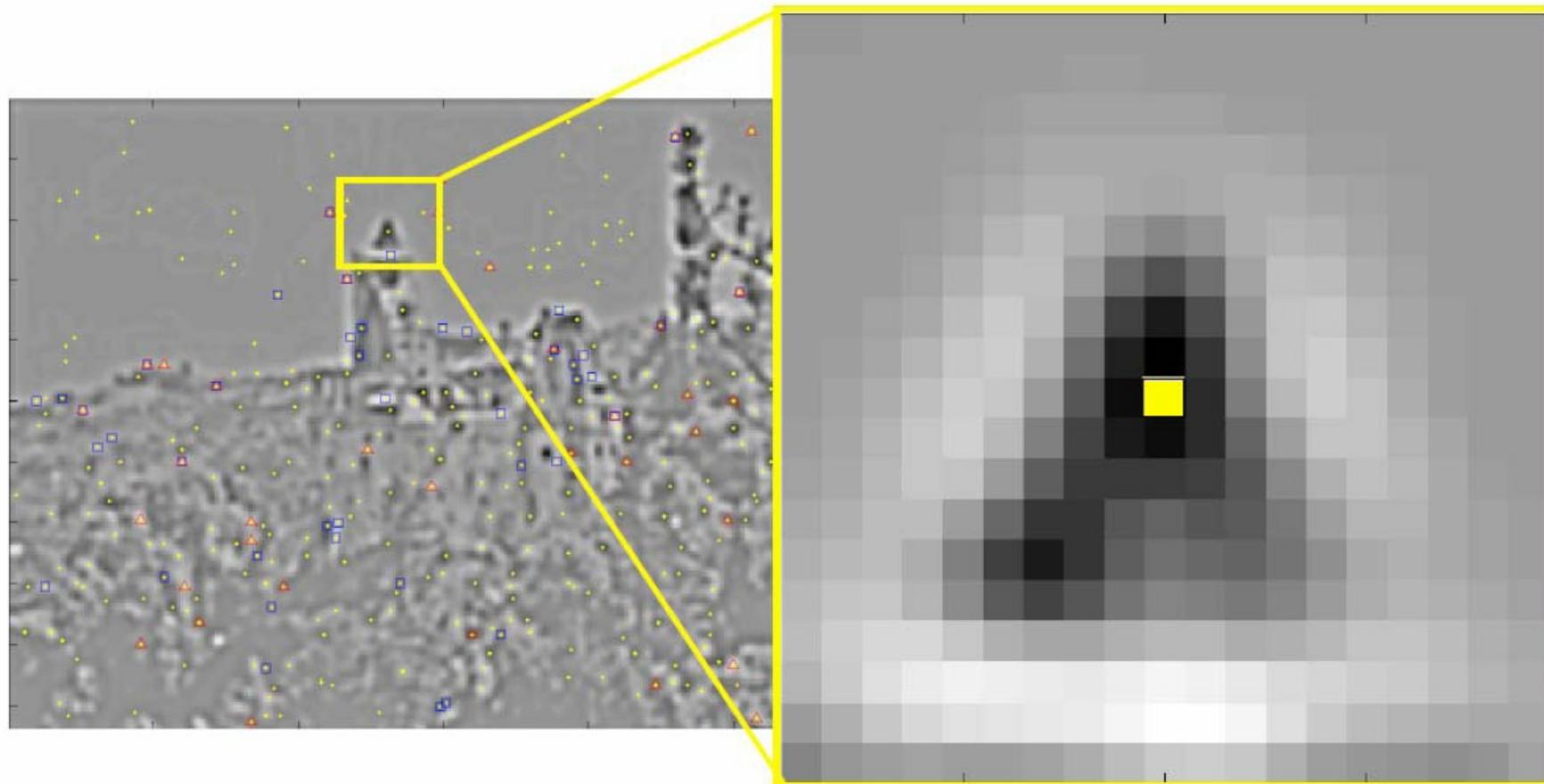
- Salida del detector (incluye posición, escala y orientaciones dominantes; más sobre esto último a continuación).



# Asignación de orientación a cada *feature* (I)

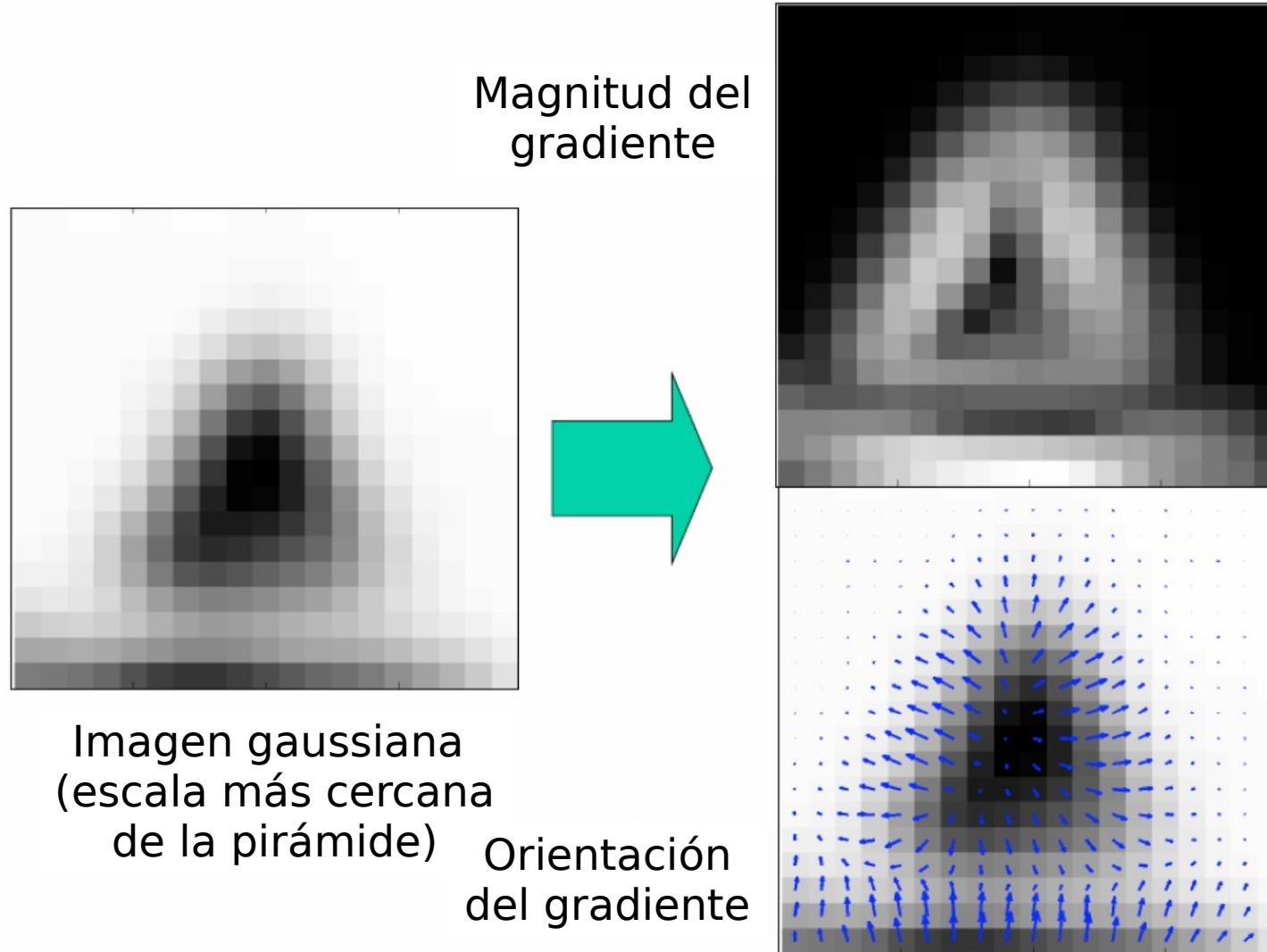
- Forzar invarianza a la orientación:
  - Computar orientación mediante dirección dominante del gradiente en la correspondiente imagen suavizada (histograma 36 posiciones, una por cada 10°).
  - Magnitud  $m$  y orientación  $\theta$  del gradiente en  $(x,y)$ :
$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$
$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1)) / (L(x+1,y) - L(x-1,y)))$$
- Rotar el parche de imagen para que esa orientación apunte hacia arriba (en realidad, de nuevo interpolación cuadrática, para detectar el pico del histograma algo mejor).

# Asignación de orientación a cada *feature* (II)

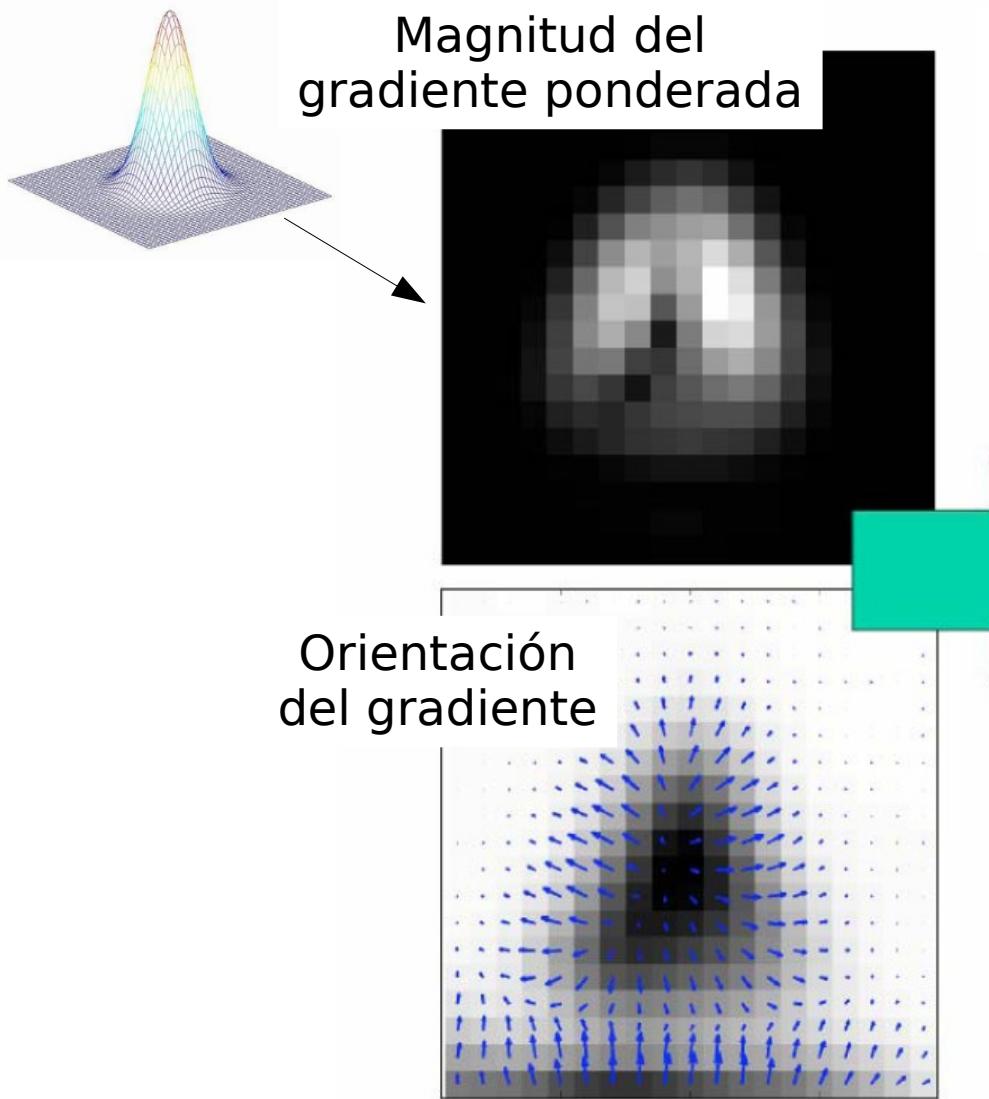


Localización del *keypoint* = posición extremo  
Escala del *keypoint* = escala de la imagen DoG

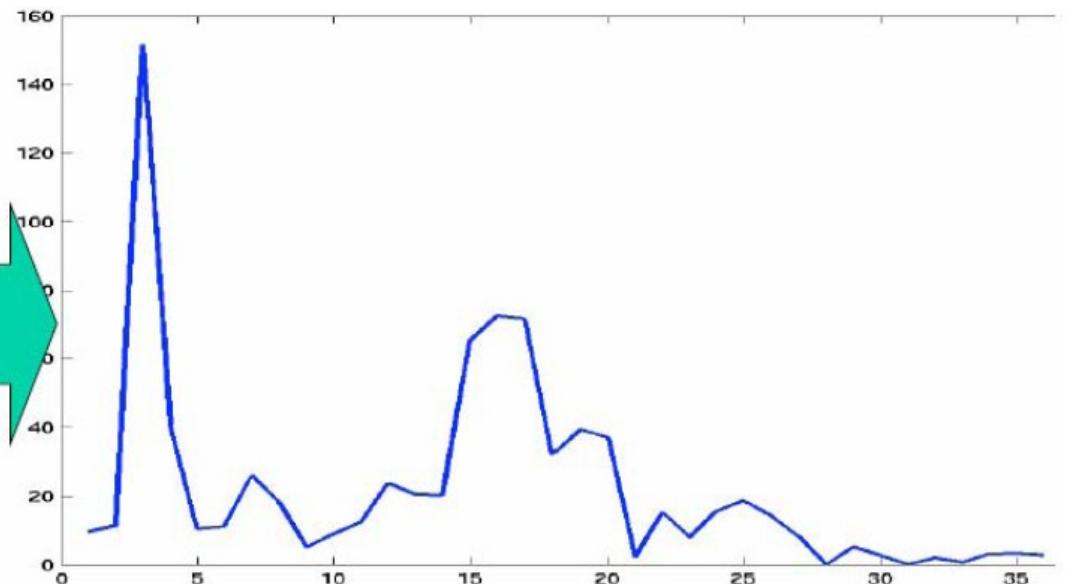
# Asignación de orientación a cada *feature* (III)



# Asignación de orientación a cada feature (IV)



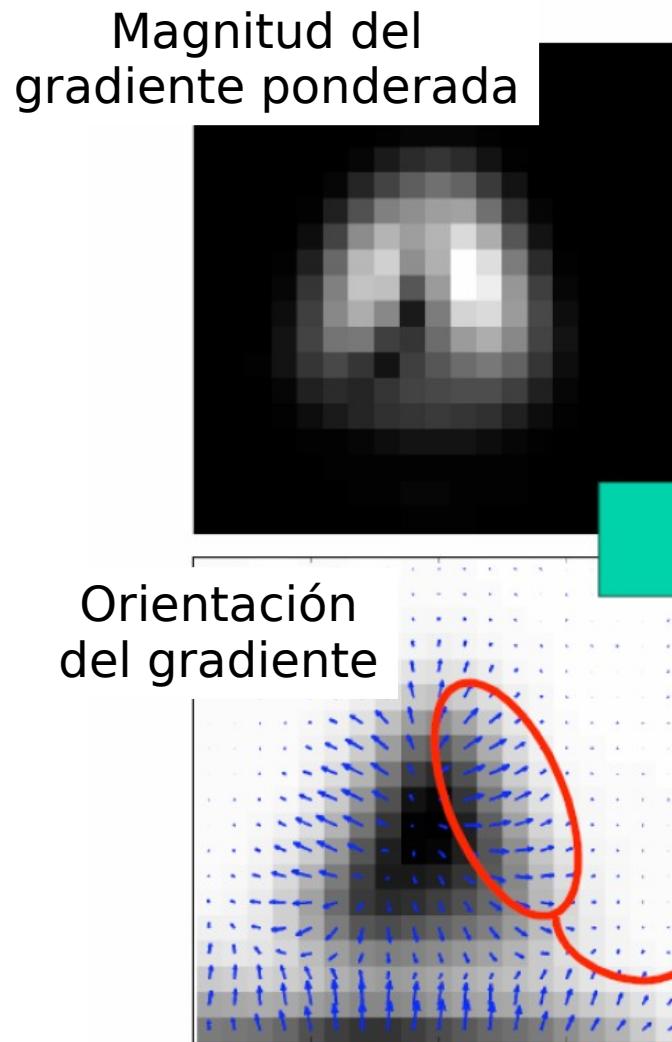
Histograma de orientaciones ponderado  
- Cada *bin* contiene suma de magnitudes del gradiente ponderadas para ángulos que caen en dicho *bin*.



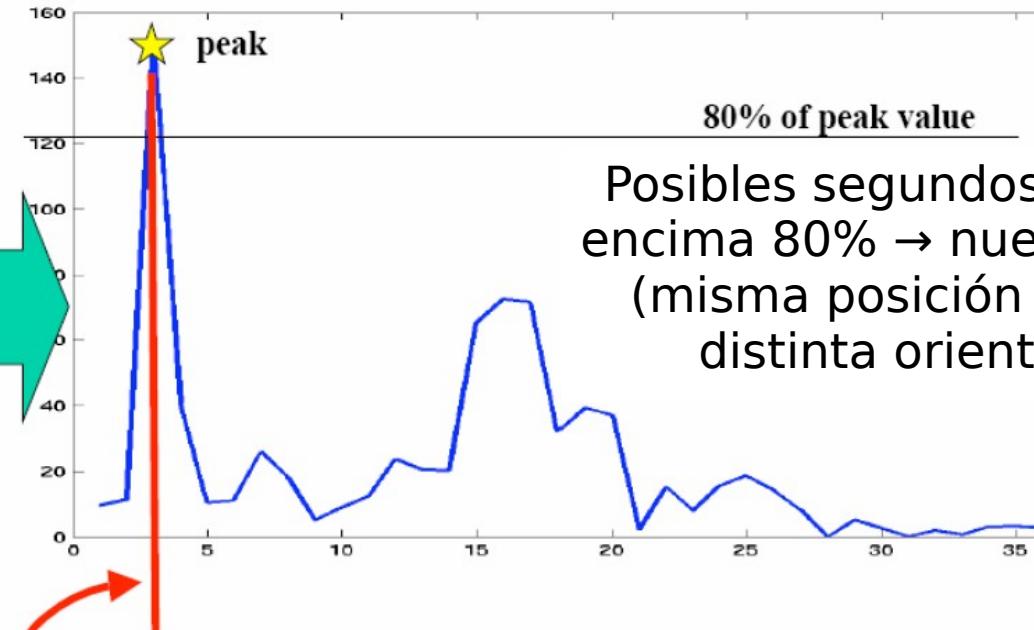
36 bins (10° cada uno)

- $0 \leq \text{ang} < 10$ : bin 1
- $10 \leq \text{ang} < 20$ : bin 2
- ...
- $350 \leq \text{ang} < 360$ : bin 36

# Asignación de orientación a cada feature (V)



Histograma de orientaciones ponderado

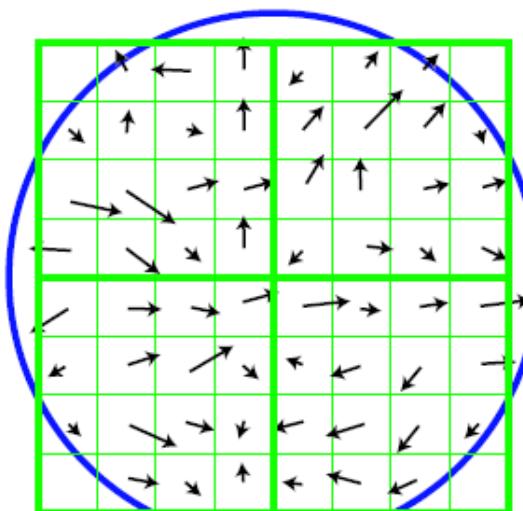


Posibles segundos picos por encima 80% → nueva feature  
(misma posición y escala,  
distinta orientación)

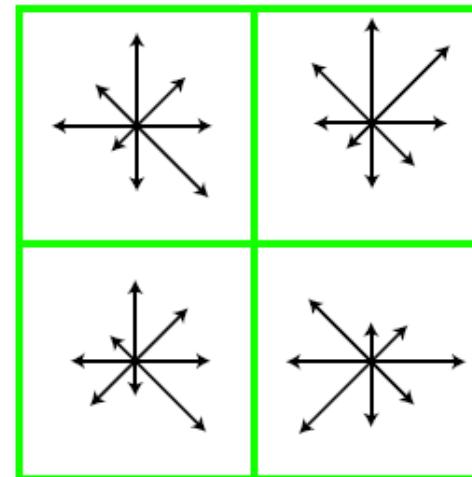
20-30 grados:  
Orientación de aproximadamente  
25 grados

# Caracterización de los puntos (*descriptors*) (I)

- Computar histograma del gradiente en la región (ya rotada) y dividida en  $4 \times 4$  regiones de  $4 \times 4$  píxeles cada una
- El resultado es un vector de características con 128 valores (16 campos, discretización de 8 orientaciones de gradiente en cada campo).



Gradientes de la imagen

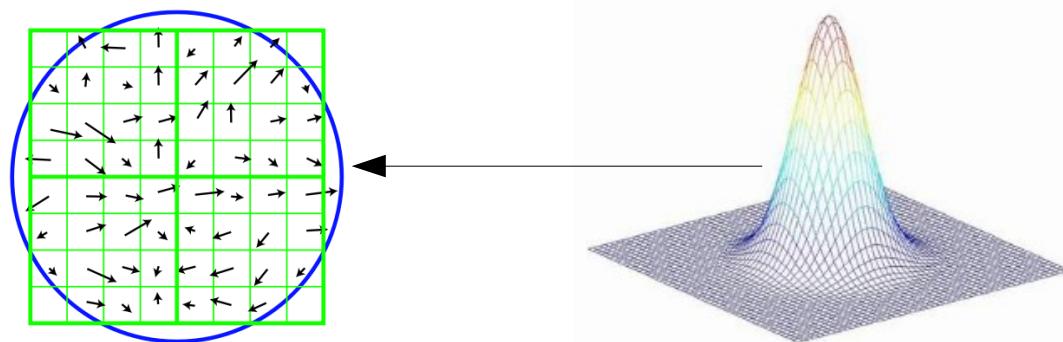


Descriptor del *keypoint*

Ejemplo simplificado:  
si fuesen  $2 \times 2$  regiones  
de  $4 \times 4$  píxeles cada una,  
saldría como descriptor  
un vector de dimensión 32

# Caracterización de los puntos (descriptors) (II)

- Para evitar *boundary effects*, interpolar trilinealmente para distribuir los valores de cada gradiente entre *bins* adyacentes de los histogramas.
- E.d., repartir entre los dos *bins* más cercanos en X, en Y, y en las 8 orientaciones discretas, usando pesos d y 1-d, siendo d la distancia normalizada en unidades de espaciado de *bins*.
- También ayuda el “pesado” previo de los gradientes, antes de realizar el resto de cuentas, con una ventana gaussiana:



# Caracterización de los puntos (descriptors) (III)

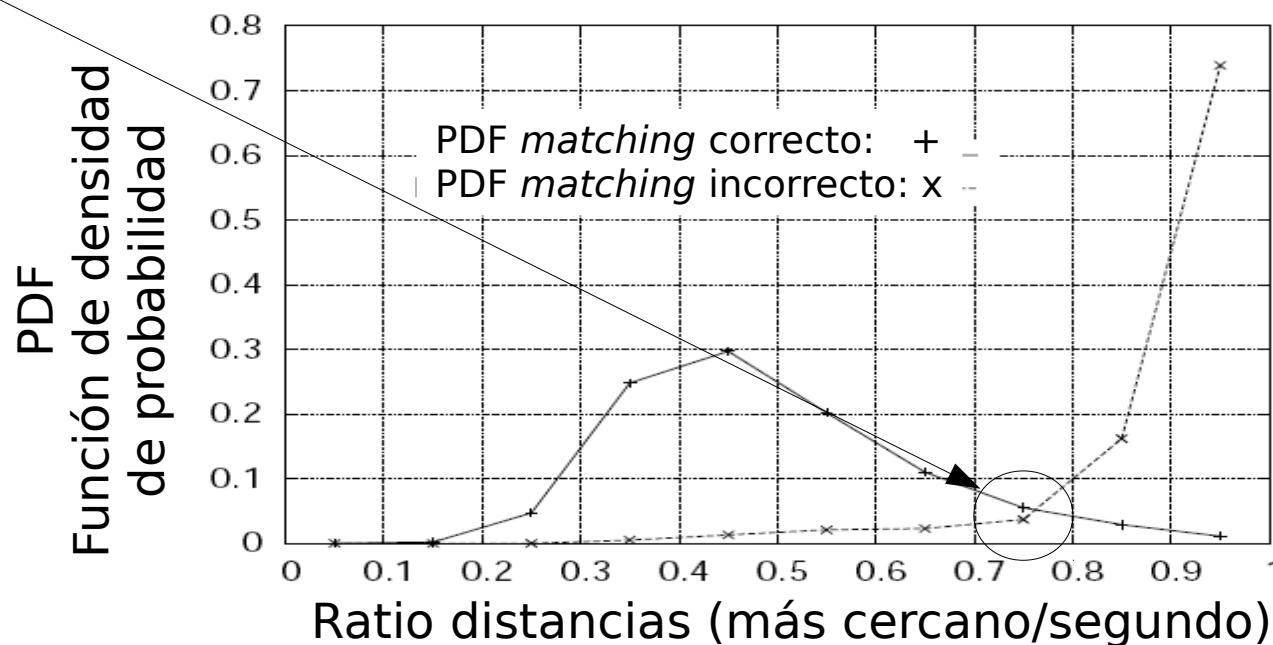
- Normalizar vector a longitud uno para obtener invarianza (afín, e.d, *bias* y *scale*) a cambios de iluminación.
- Finalmente, “recortar” valores muy positivos del vector, y volver a normalizar, para obtener cierta invarianza a cambios no lineales de iluminación (saturación, etc.)
- Propuesta Lowe → truncar componentes a  $<= 0.2$ , y volver a normalizar.
- Potenciamos así la influencia de la orientación del gradiente, más que de su magnitud.

# Reconocimiento de objetos (I)

- Se realizará partiendo de un **paso previo de *matching* individual** (*nearest neighbour*) de cada *keypoint* contra toda una base de datos de *keypoints* extraídas de las imágenes de entrenamiento.
- Pero muchos *keypoints* ni siquiera estarán en la base de datos (serán del fondo, otros objetos, etc.). Además, el *matching* puede fallar con relativa facilidad.
- Para lograr mayor robustez, al menos **3 *features* tendrán que coincidir aproximadamente en tamaño y ángulos relativos** entre un objeto de la BD y la imagen para presumir que pueda haber una detección de objeto.
- **Ya sobre este *cluster*, se ajusta con mayor precisión un modelo geométrico**, y según la calidad del *matching* se decide si había o no un objeto reconocido presente.

# Reconocimiento de objetos (II)

- Muchas veces el *nearest neighbour* (NN) no se corresponderá con el *keypoint* correcto de la base de datos.
- Así que habrá que rechazar algunos *matchings*. Pero un umbral absoluto no funciona bien...
- ...así que para aceptarlo, el 2º NN deberá estar como mucho a un **20%** de distancia mayor que el 1º:



# Reconocimiento de objetos (III)

- *Clustering con la transformada de Hough:*
- Cada ***matched feature*** vota por la(s) pose(s) aproximada(s) del objeto consistente(s) con ella:
  - Cada una especifica cuatro parámetros: localización 2D, escala y orientación.
  - Usando esos mismos parámetros en la *feature* de la BD (conocidos para el objeto en cuestión), la nueva *feature* vota por una posición del modelo.
  - La votación inicial es intencionadamente imprecisa (*coarse*): bins de 30° de orientación, factores de 2 para la escala, y 25% del tamaño de la imagen de la BD para la posición.

# Reconocimiento de objetos (IV)

- *Clustering* con la transformada de Hough (cont.):
  - Para evitar los *boundary effects*, de forma similar a como hicimos en los propios histogramas de gradientes, en cada dimensión, se vota en los dos bins adyacentes más cercanos.
  - Como hay 4D ( $x, y, \text{escala}, \text{rotación}$ ), ello implica  $2^4=16$  votos por cada característica.

# Reconocimiento de objetos (V)

- Finalmente, y si se tienen tres o más *matchings*, se hace un ajuste fino de la transformación afín modelo-imagen:

posición en el modelo

transformación afín y traslación a estimar

Cada *match* contribuye con dos ecuaciones...

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$
$$\begin{bmatrix} x & y & 0 & 0 & 1 & 0 \\ 0 & 0 & x & y & 0 & 1 \\ \dots & & \dots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u \\ v \\ \vdots \end{bmatrix}$$

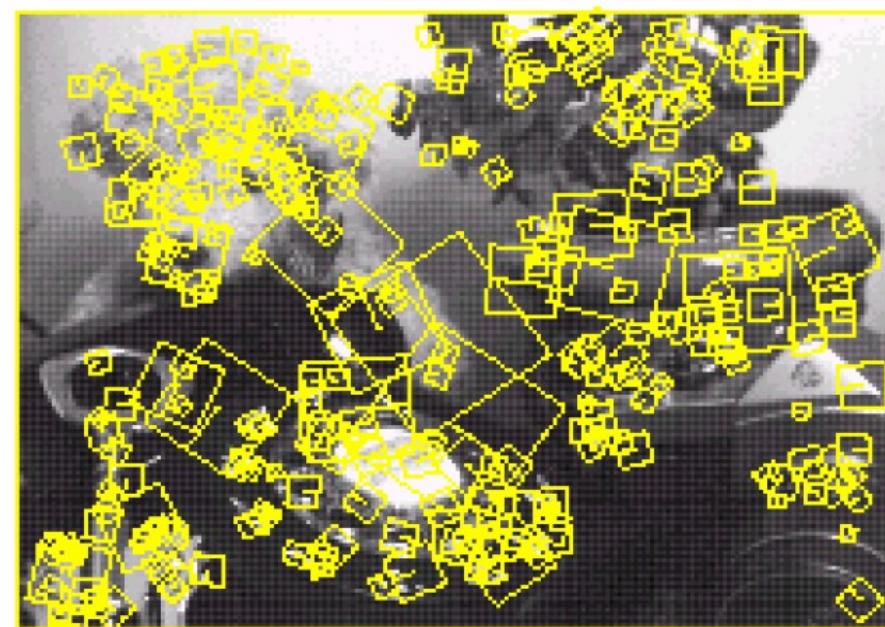
Nx6 matriz  $\rightarrow \mathbf{Ax} = \mathbf{b}$   $\leftarrow$  Nx1 vector

Resolvemos el sistema por mínimos cuadrados, usando al menos 3 *matchings*

$$\mathbf{x} = [\mathbf{A}^T \mathbf{A}]^{-1} \mathbf{A}^T \mathbf{b}$$

# Ejemplos (I)

- Ejemplo de estabilidad en diferentes condiciones de luz:



273 matchings verificados

# Ejemplos (II)

- Ejemplos de localización (incluso con ocultación):

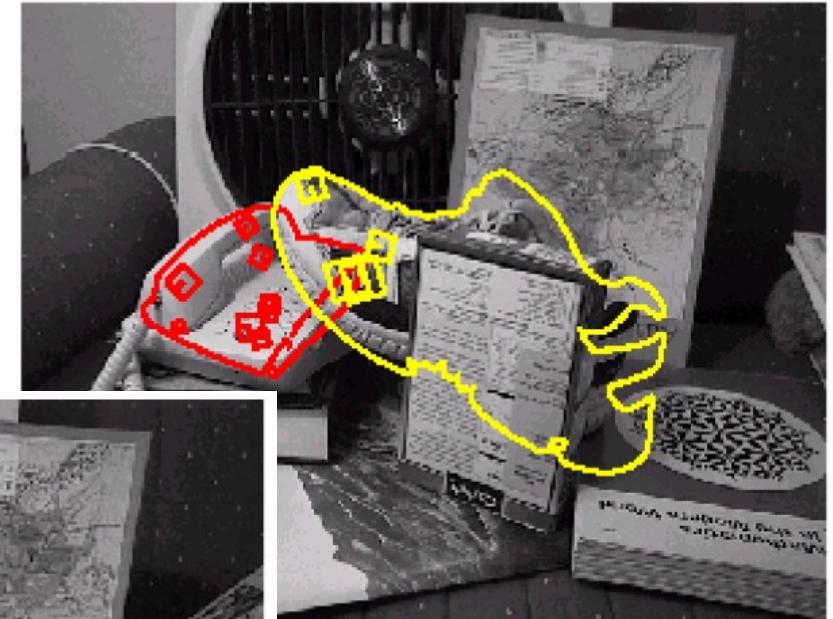
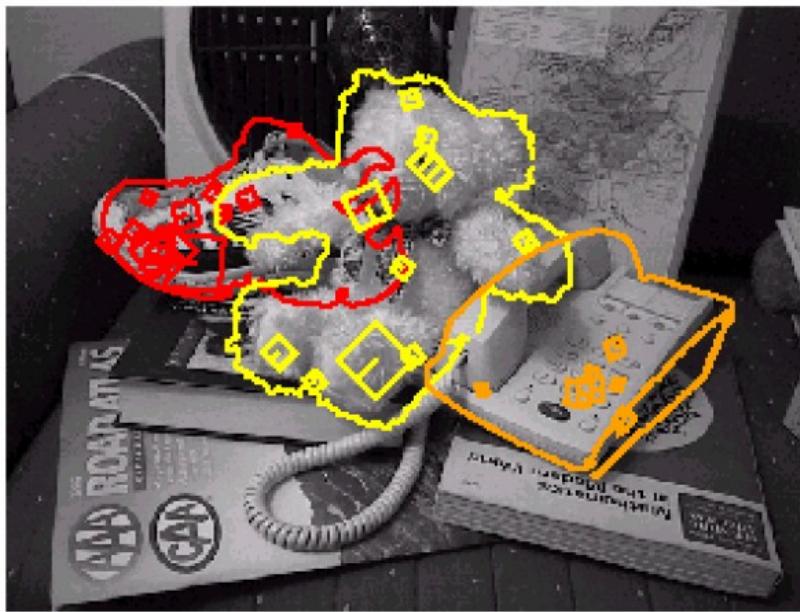


Localizar modelos como éstos...

...en imágenes con *clutter* como ésta.

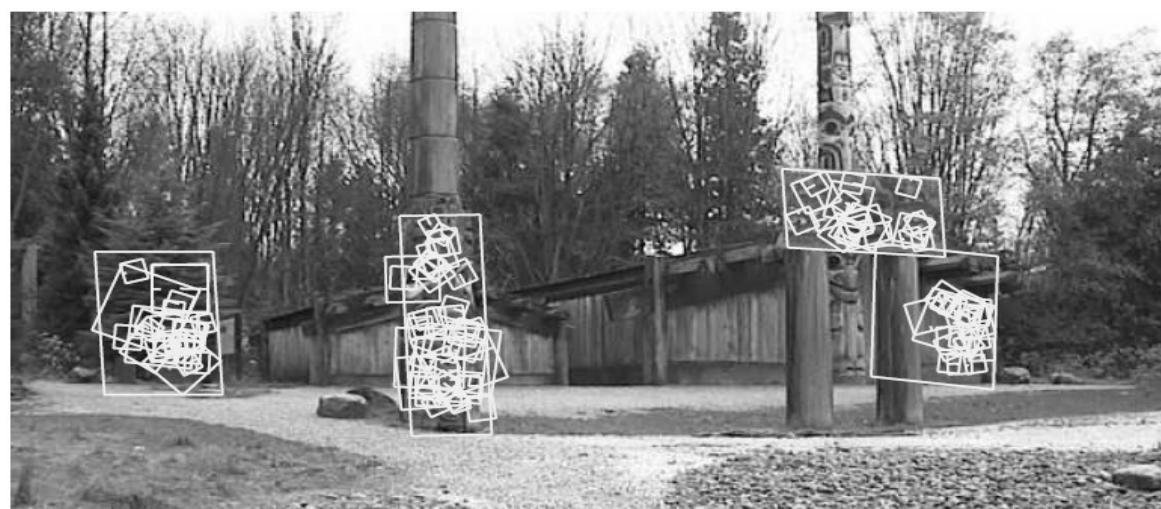
# Ejemplos (III)

- Ejemplos de localización (incluso con ocultación):



# Ejemplos (IV)

- Ejemplos de localización (incluso con ocultación):



# Ejemplos (V)

- Resultados práctica reconocimiento objetos planares:



# Ventajas SIFT

- **Localidad en la detección:**
  - No involucra más que cálculos en el entorno de la característica.
  - Ello implica cierta **eficiencia** (aunque, ojo, el cálculo a todas las escalas lo hace más lento).
  - La localidad también implica resistencia a la oclusión y al *ruido de fondo*.
- Capacidad **discriminante** de los descriptores:
  - *Matching* individual de features contra grandes BDs.
- **Cantidad** de detecciones:
  - ~2000 *features* estables, a distintas escalas, en imágenes típicas de 500x500.

# *Maximally Stable Extrema Regions* (MSER)

- Paper original: Matas et alt., BMVC 2002.
- Como SIFT, método de **extracción de *features* robusto, invariante a escala y rotación, ...**
- ... pero también invariante afín:
  - Sigue detectando la misma región aunque experimente escalado de tipo anisotrópico → útil para definir descriptores en marcos afines locales (LAF: *local affine frames*).
- En el *paper* original se motivan para hacer *wide baseline matching* entre imágenes (estéreo, reconstrucción 3D, etc), pero, dotados del descriptor adecuado, pueden usarse también en reconocimiento de objetos.

# Comparación MSER / SIFT

- MSER más eficiente en la detección:
  - No necesario trato explícito de múltiples escalas.
- MSER más invariante:
  - MSER invariante afín completo (6 gdl), SIFT sólo posición, escala y rotación (4 gdl).
  - En realidad, MSER invariante proyectivo (8 gdl) e incluso a otro tipo de deformaciones que simplemente “preserven adyacencia”.
- MSER devuelve contorno asociado
  - P.e., para trabajar con descriptores frecuenciales, *signaturas* respecto al centro de gravedad, etc.

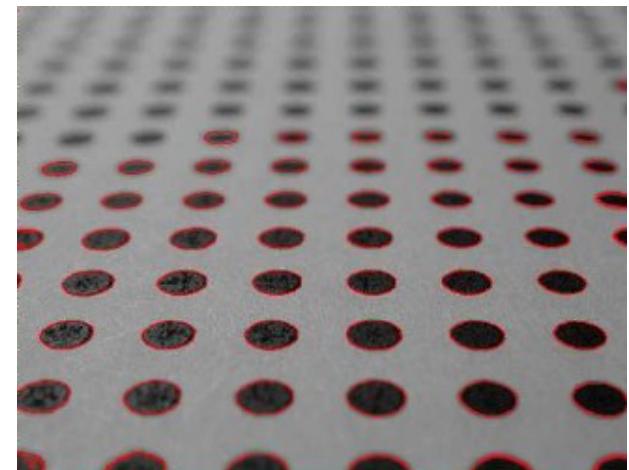
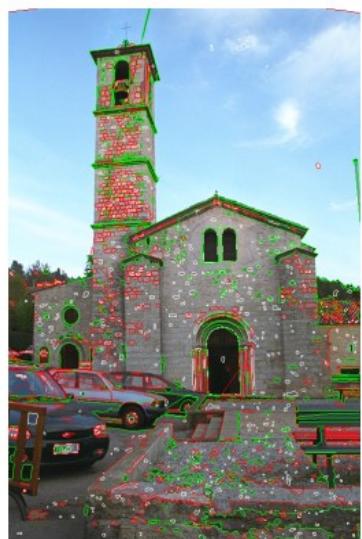
# Ejemplos (I)

- Regiones oscuras sobre fondo claro, o viceversa, con bordes bien definidos, pero necesariamente “cerrados”:



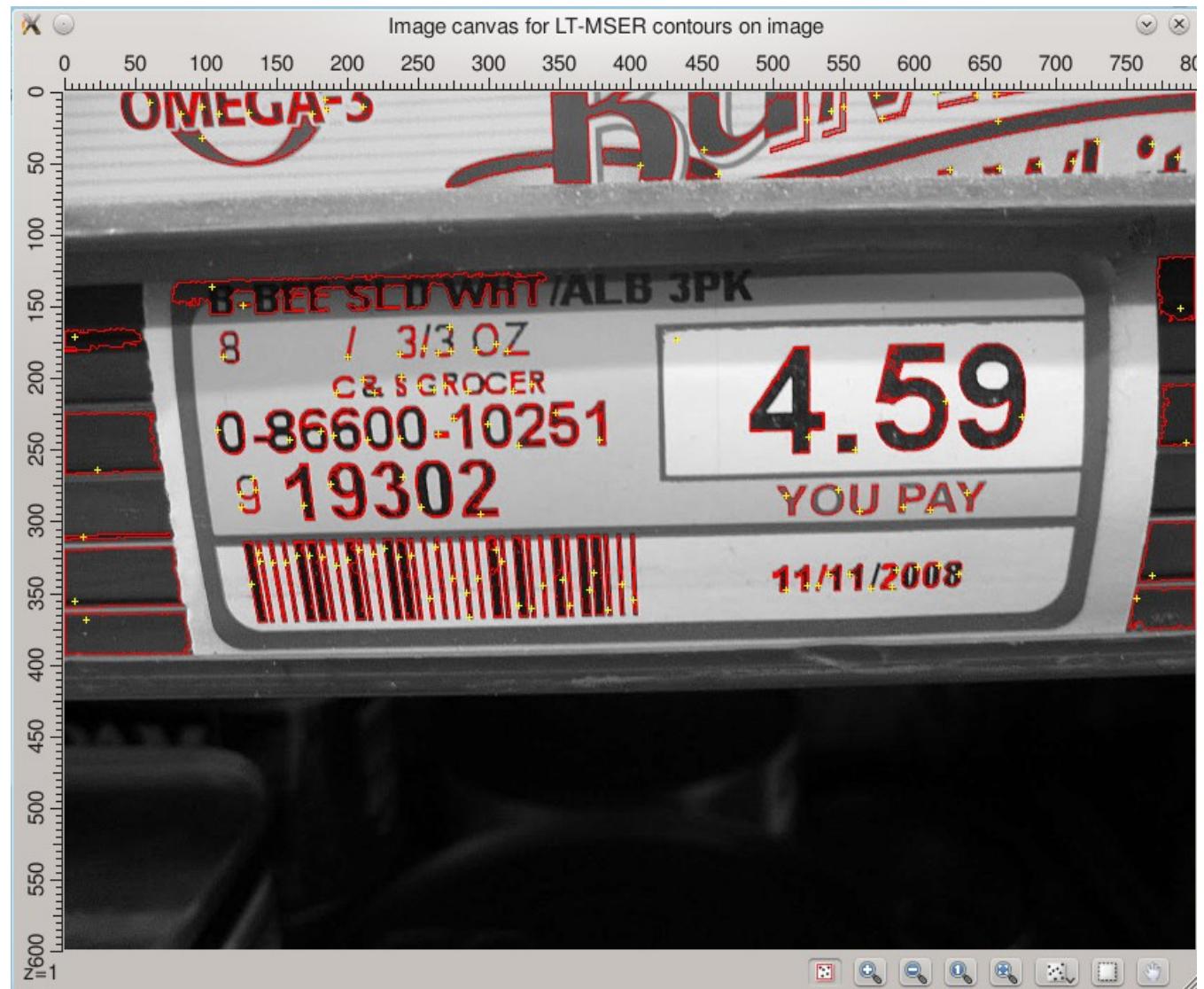
# Ejemplos (II)

- Más “fiables” en escenas “no naturales”:



# Ejemplos (III)

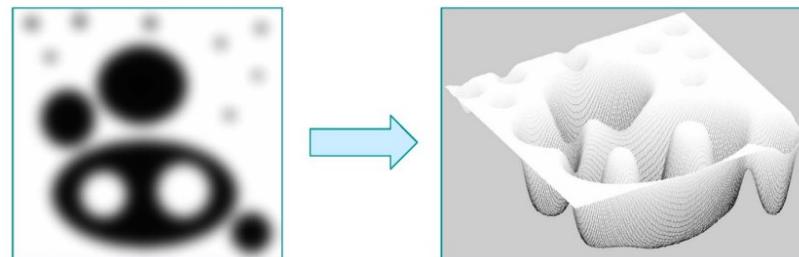
- Ideal para detección de letras, signos, etc., con posterior procesamiento del contorno:
- En cierto sentido, es como un **umbralizado “dinámico”**



# Idea principal MSER (I)

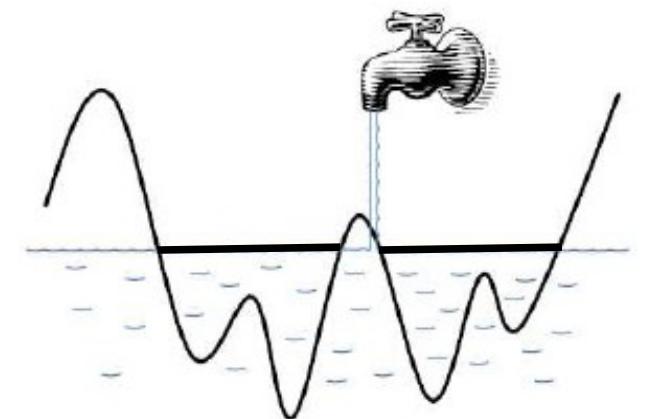
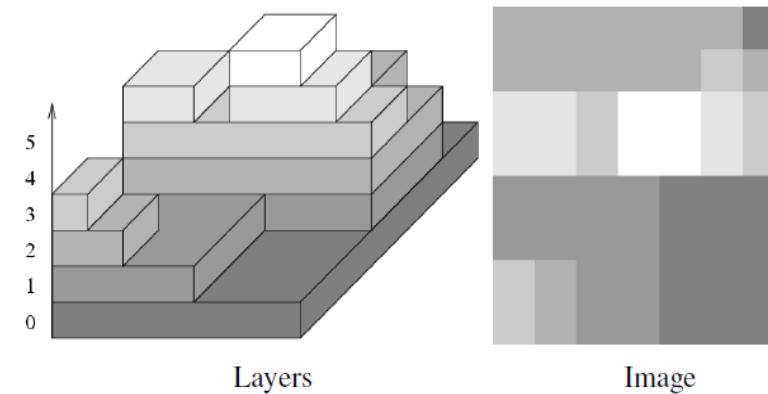
- Metáfora:

- Imagen como array 2D de alturas (“*landscape*”).



intensity image

shown as a surface function



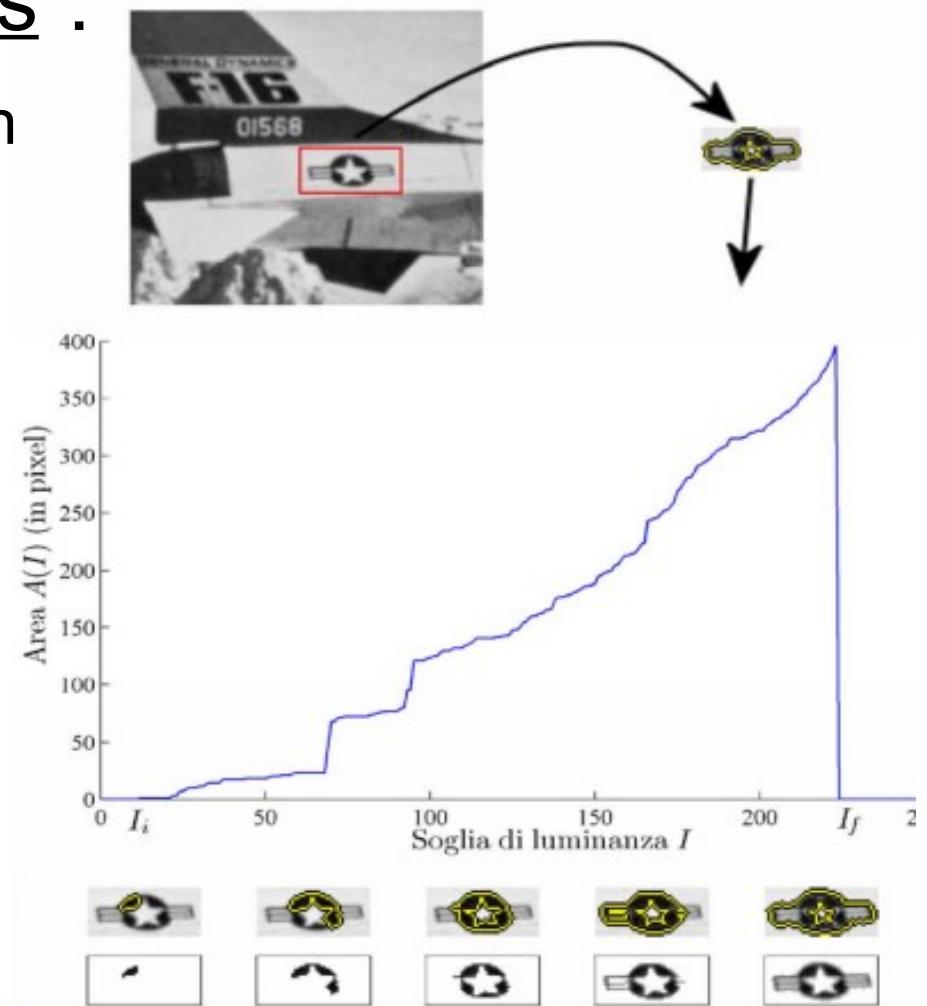
“Inmersión del paisaje”

- El algoritmo realiza una “inmersión gradual” en agua, y va examinando la velocidad a la que las distintas regiones sumergidas van creciendo de tamaño.

# Idea principal MSER (II)

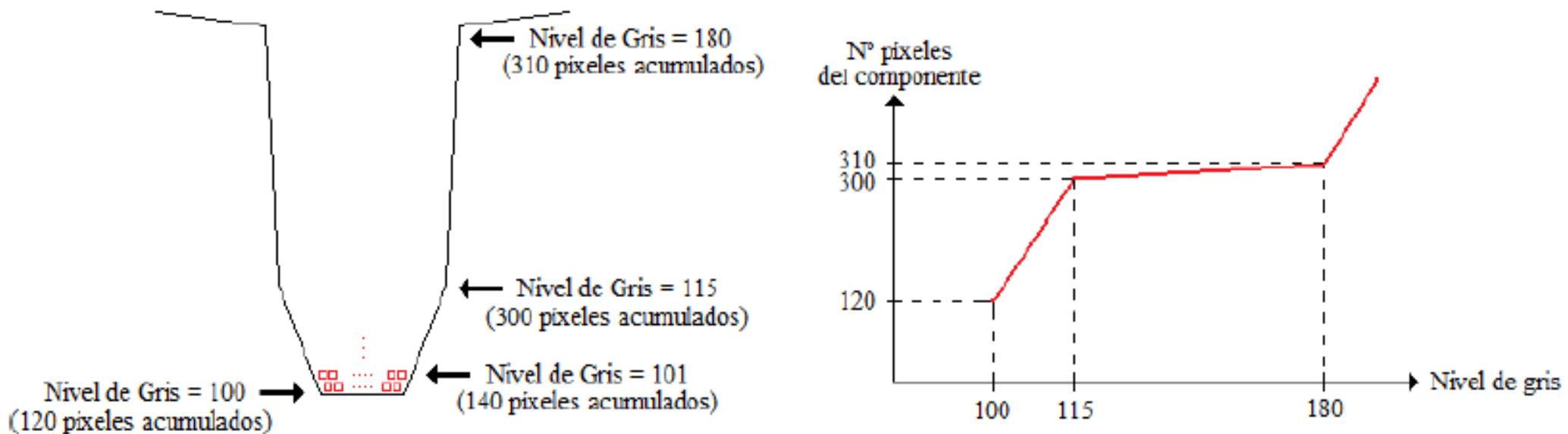
- Detectar “regiones estables”:
  - Detectar mínimos en la variación del área de una región al ir variando la intensidad:

Nota: en este ejemplo vamos de oscuro a claro; obviamente también nos interesa el otro sentido (claro a oscuro) → dos pasadas del algoritmo



# Importancia del historial

- El historial guarda la variación de área respecto a la variación en el nivel de gris:
  - Zonas de escasa variación para grandes intervalos se corresponderán con MSERs:



# Implementación (I)

- Conjuntos de nivel (level sets):
  - Componentes conexas con valor de gris mayor o igual a uno dado (en este ejemplo “emergen”, en lugar de “sumergirse”):

1	1	1	1	1	1	1	1
1	3	3	2	3	4	1	
1	3	3	2	3	4	1	
1	1	1	1	1	3	1	
1	3	3	2	1	1	1	
1	4	3	2	2	2	1	
1	1	1	1	1	1	1	

$F$

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

$F_1$

0	0	0	0	0	0	0	0
0	1	1	0	1	1	0	
0	1	1	0	1	1	0	
0	0	0	0	0	1	0	
0	1	1	0	0	0	0	
0	1	1	0	0	0	0	
0	0	0	0	0	0	0	0

$F_2$

0	0	0	0	0	0	0	0
0	1	1	0	1	1	0	
0	1	1	0	1	1	0	
0	0	0	0	0	1	0	
0	1	1	0	0	0	0	
0	1	1	0	0	0	0	
0	0	0	0	0	0	0	0

$F_3$

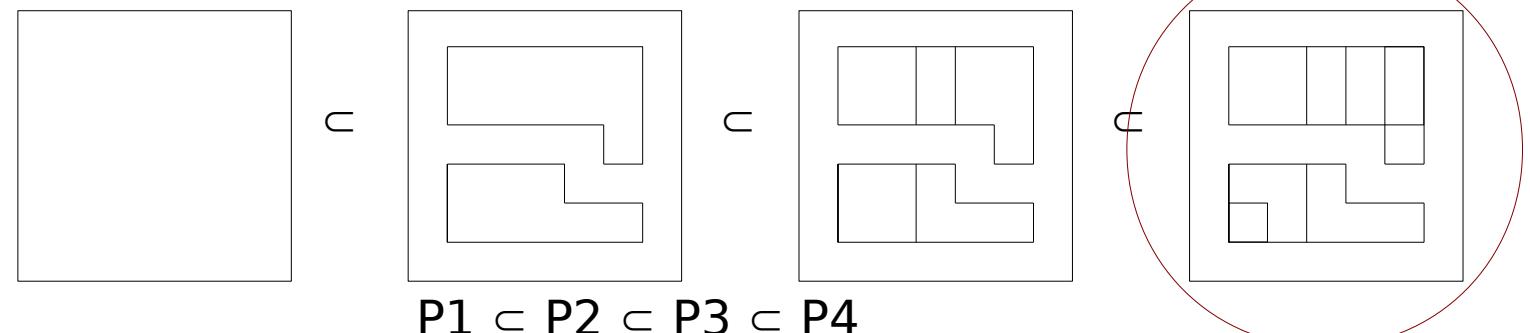
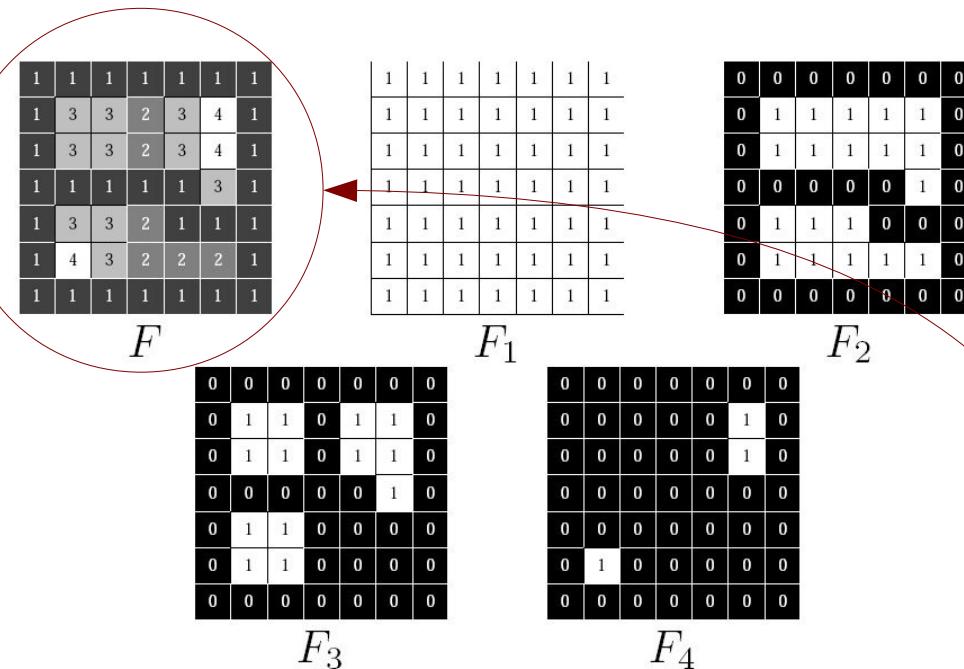
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$F_4$

Level sets

# Implementación (II)

Las componentes conexas van creciendo al variar el nivel de agua:



# Implementación (III)

- Ordenación inicial de los píxeles de la imagen (de menor a mayor en una pasada, al revés en la otra)
  - Algoritmo Counting Sort:  $O(n)$ , puesto que sabemos que todos los valores están en el conjunto finito  $[0, 255]$ .
- Ir tratando pixels de menor a mayor valor de gris, manteniendo subconjuntos de componentes conexas:
  - Algoritmo Union-Find (quasi-lineal,  $O(n \cdot \text{Ackerman}^{-1}(n))$ ):
    - *Find*: saber a qué subconjunto (= comp. conexa) pertenece un píxel.
    - *Union*: unir dos subconjuntos (comp. conexas) en uno.

# Procedimiento *Union-Find*

- Problema: mantener colección de subconjuntos bajo la operación de unión; 3 operaciones básicas:

---

**Procedure** MakeSet (*element x*)

---

    Par(*x*) := *x*; Rnk(*x*) := 0;

---

---

**Function** *element Find(element x)*

---

**if** (Par(*x*) ≠ *x*) **then** Par(*x*) := Find(Par(*x*));  
    **return** Par(*x*);

---

---

**Function** *element Link(element x, element y)*

---

**if** (Rnk(*x*) > Rnk(*y*)) **then** exchange(*x, y*);  
    **if** (Rnk(*x*) == Rnk(*y*)) **then** Rnk(*y*) := Rnk(*y*) + 1;  
    Par(*x*) := *y*;  
    **return** *y*;

---

- Intenta mantener los “árboles” lo más planos posible:
  - Enlazando directamente al padre en cada nodo visitado en *Find*.
  - Usando siempre como padre al subárbol más profundo

# Ejemplo uso union-find:

## Etiquetado de componentes conexas

---

### Algorithm 1: ConnectedComponents

---

**Data:**  $(V, E)$  - graph Píxeles activos

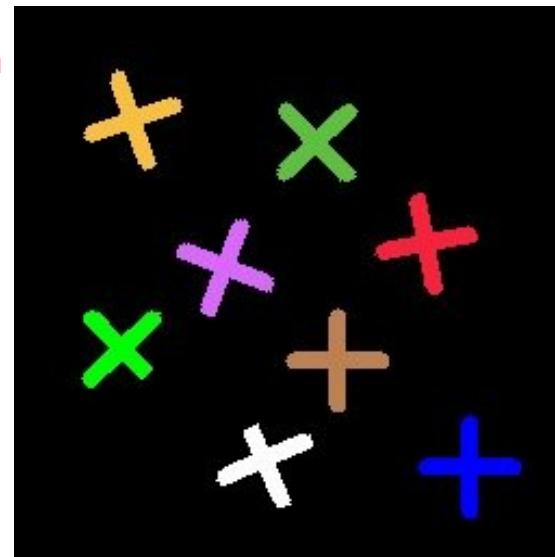
**Data:** A set  $X \subseteq V$  Dominio imagen

**Result:**  $M$  - map from  $X$  to  $V$

```

1 foreach  $p \in X$  do MakeSet( $p$ );
2 foreach  $p \in X$  do
3   compp := Find( $p$ );
4   foreach  $q \in \Gamma(p) \cap X$  do Píxeles vecinos
5     compq := Find( $q$ );
6     if ( $compp \neq compq$ ) then
7       compp := Link( $compq$ , compp);
8 foreach  $p \in X$  do  $M(p) :=$  Find( $p$ );

```



**Salida**

Salida: cada píxel se asocia a su representante  
(elemento canónico, raíz del árbol de su clase)

# Algoritmo básico MSER

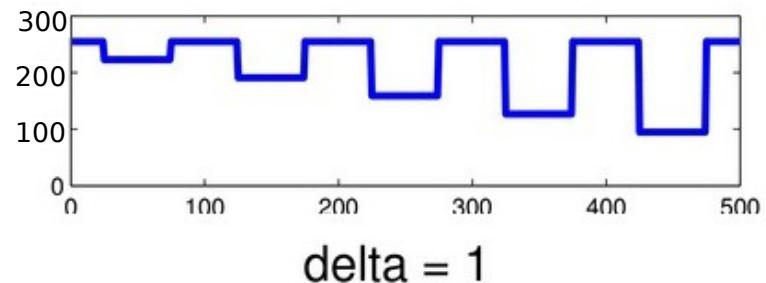
1. Ordenar píxeles de menor a mayor valor de gris (Counting Sort).
2. Inicializar partición de la imagen con las componentes conexas de píxeles conectados con igual valor de gris.
3. Comenzando con  $nivel\_agua=0$ , e incrementando en cada paso dicho valor en 1, usar Union-Find para, progresivamente, ir uniendo componentes conexas de píxeles con  $nivel\_gris \leq nivel\_agua$ .
4. En todo momento, para cada componente, guardar su historial de áreas (tabla de valores  $historial[nivel\_agua]=area\_componente$ , correspondientes a las áreas de las sucesivas regiones anidadas), y unir adecuadamente dichos historiales al fusionar componentes.
5. Antes de cada fusión, estudiar si la región es maximalmente estable:  $(\exists i \mid hist[i+\Delta]-hist[i-\Delta])/hist[i]$  tiene un mínimo local  $\rightarrow$  guardar MSER.  
( $\Delta$  parámetro principal del método; se aplica combinado con un umbral)

# Parámetros MSER (I)

- Parámetros típicos del algoritmo:
  - $\Delta$  el más importante: diferencia típica mínima permitida entre “fondo” y “forma”.
  - $\Delta_{th}$  evita detección de regiones demasiado llanas.
  - Áreas máxima y mínima de cada MSER.
  - Mínimo ratio de posibles áreas anidadas (no tan importante para detectar letras/símbolos).
  - Tamaño máscara filtrado mediano previo (opcional; puede aumentar rendimiento).
  - Si se desean sólo MSER+, MSER-, o ambos (en este caso, hay que invertir la imagen y repetir el proceso).

# Parámetros MSER (II)

- Influencia parámetro  $\Delta$ :



delta = 32



delta = 159



delta = 160

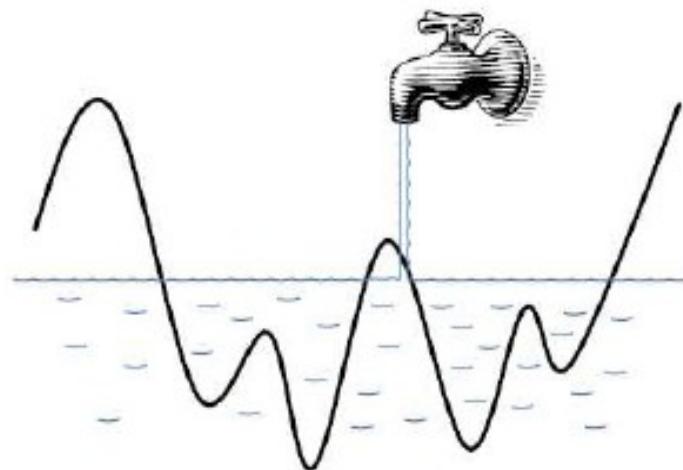


# Detalles de implementación

- ¿Hay que guardar explícitamente los píxeles de cada MSER detectada?:
  - Podría hacerse, pero es más eficiente guardar píxel *semilla* (más oscuro) y umbral hasta el que hay que llenar la imagen para obtener la MSER (valor *i* para el que se obtuvo).
  - Así, para cada componente, ir guardando sólo:
    - Tamaño en píxeles.
    - Historial de área vs. nivel de gris.
    - Semilla (X,Y)
  - Para cada par de componentes fusionados se puede liberar uno de los históricos (continuando con el otro).

# *Linear Time MSER (LTMSER)*

- Se puede acelerar el algoritmo modificando el “procedimiento de inmersión”:



**MSER original**



**MSER lineal**

# Ventajas *LTMSE*R

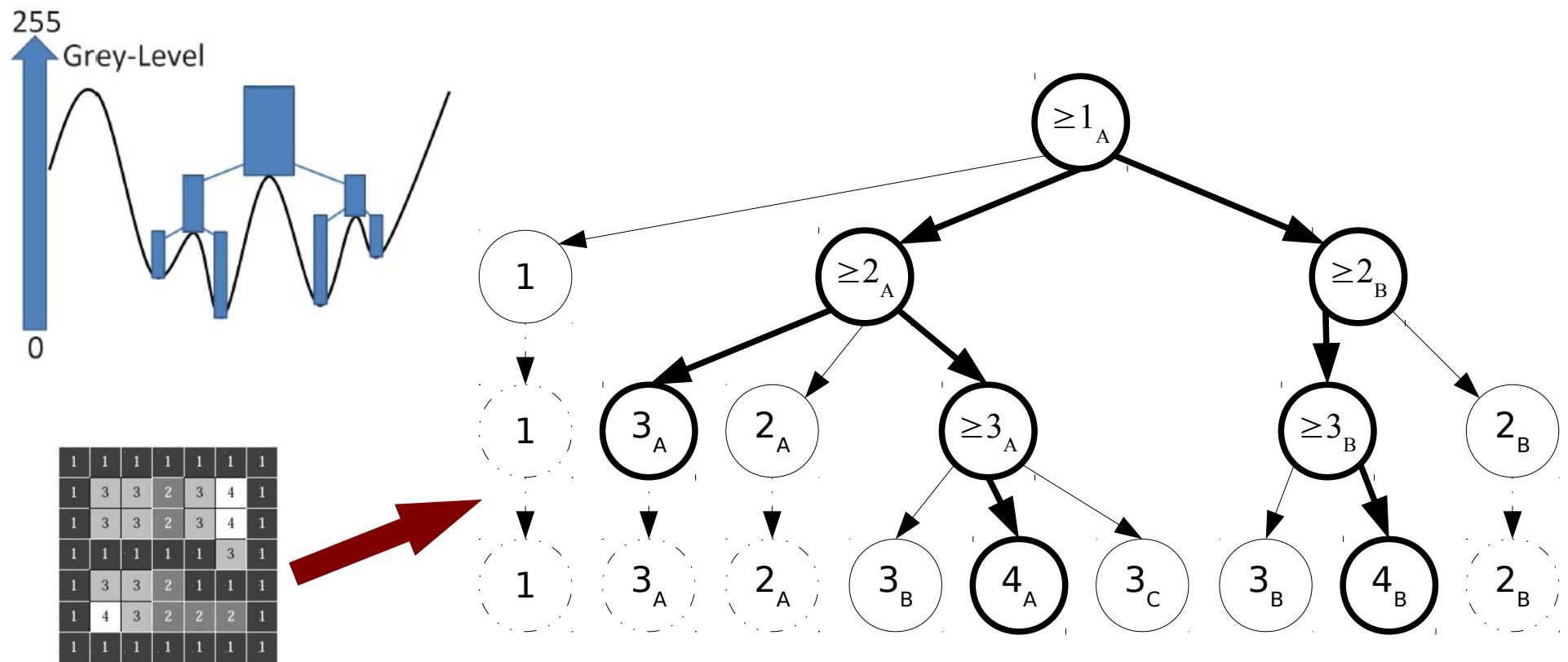
- No necesario guardar listas de píxeles explícitas
  - Basta con semilla y umbral (como antes)
- Fusión de historiales también igual
- Velocidad:
  - ~5-6 Mpixeles/s en CPU convencional (1 core 2.4 Ghz)
  - Depende bastante de la estructura de la imagen:
    - Muy rápido para imágenes “limpias” (texto, etc.).
    - Más lento para imágenes reales.
    - Por eso filtro de mediana → aceleración del proceso.

# Comparación detectores MSER-SIFT (cont.)

- SIFT más cantidad de detecciones:
  - MSER más sensible al tipo de imagen, suele detectar menos regiones
  - Blobs SIFT menos “complejos” → P.e., una letra = 1 MSER, pero quizá varios SIFT.
- Ambos pueden ser “positivos” o “negativos”.
  - Incrementa número, y facilita descartes tempranos en matching...
  - ...pero MSER necesita para ello dos “pasadas” del algoritmo (SIFT no).

# Component Trees

- Con muy poco trabajo más, árbol de componentes:
  - (Usados a veces como descriptores de imagen)



# Referencias adicionales (I)

- Wikipedia: “Hough transform”:
  - En la pagina hay algunos enlaces interesantes para mostrar su funcionamiento.
- Wikipedia: “RANSAC”.
- Paper SIFT:
  - “*Distinctive Image Features from Scale-Invariant Keypoints*”. D. Lowe.
- Explicación algo más sencilla y detallada algorítmicamente:
  - Extracción y Descripción de Características SIFT. E. Iniesta.

# Referencias adicionales (II)

- Papers MSER original y LTMSER:
  - “*Robust Wide Baseline Stereo from Maximally Stable Extremal Regions*”, Matas, Chum, Urban and Pajdla, BMVC 2002.
  - “*Linear Time Maximally Stable Extrema Regions*”, Nister and Stewenius, ECCV 2008.
- Implementaciones:
  - Vedaldi: <http://www.vlfeat.org>
  - Página “*feature detectors & descriptors evaluation*”: <http://www.robots.ox.ac.uk/~vgg/research/affine/index.html>
  - OpenCV (MSER, SIFT, SURF)