



RECONOCIMIENTO DE PATRONES EN IMÁGENES TICS 585

FACULTAD DE INGENIERÍA Y CIENCIAS
UNIVERSIDAD ADOLFO IBÁÑEZ

SEGUNDO SEMESTRE 2021

PROFESOR: MIGUEL CARRASCO

INTRODUCCIÓN IMÁGENES

■ Geométricas

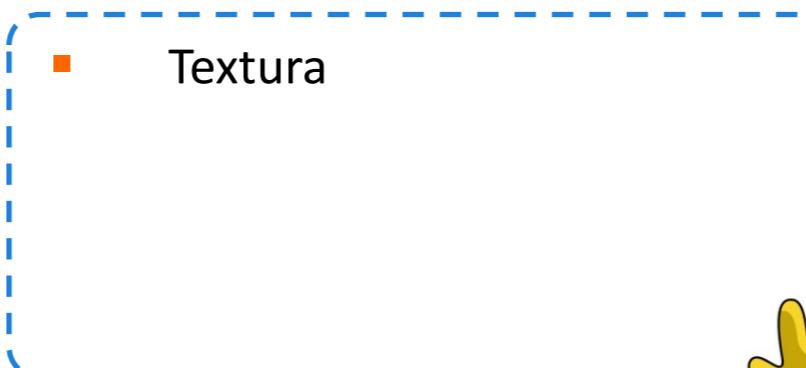
- Centro de masa
- Tamaño
- Perímetro
- Redondez
- Momentos binarios
- Descriptores de Fourier
- Elipses
- Distancia al borde

■ Cromáticas

- Color promedio
- Gradiente promedio
- Promedio segunda derivada
- Contraste
- Momentos de color

■ Otros descriptores

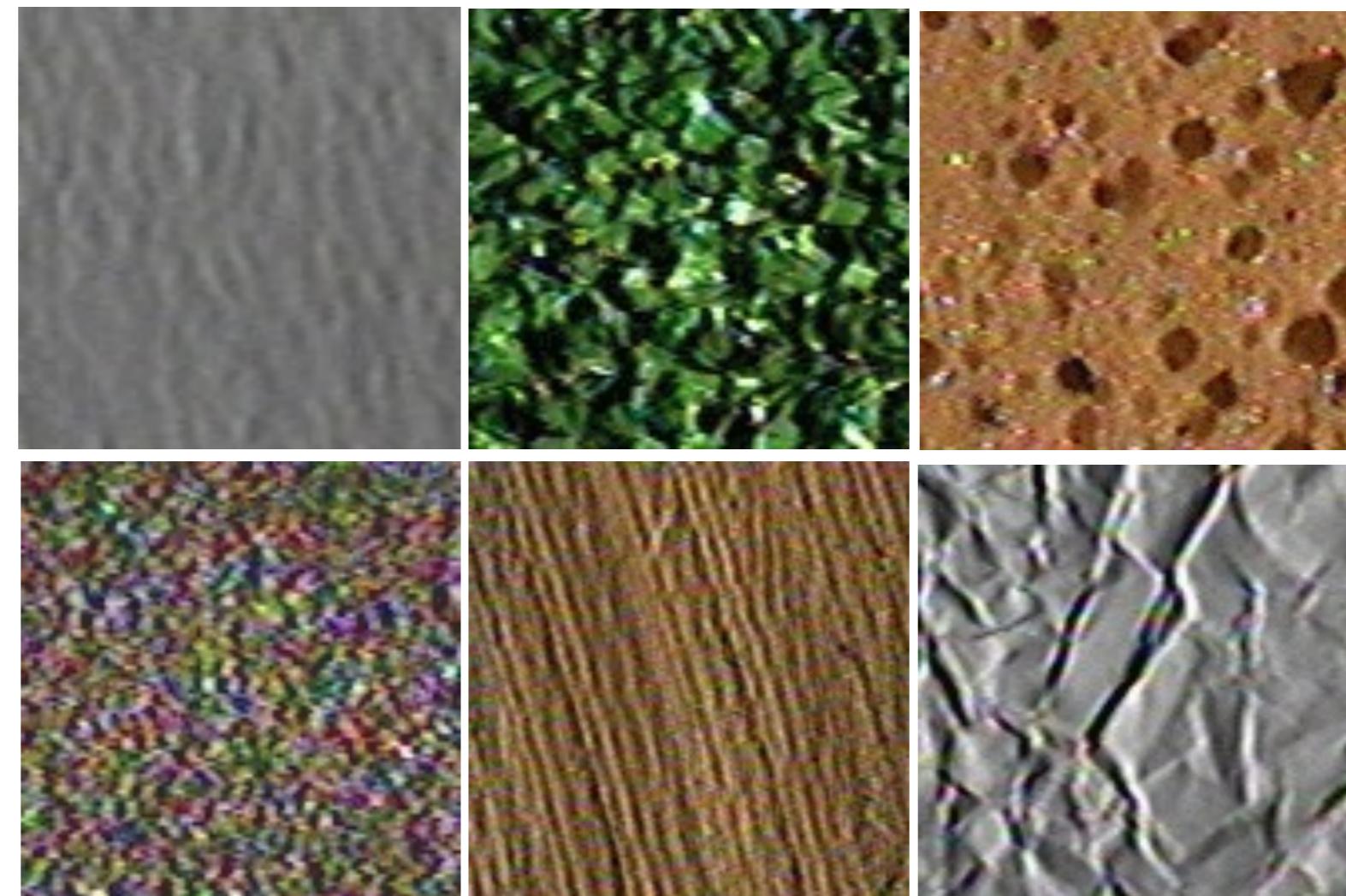
- PHOG
- SURF
- SIFT
- GPD



■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura

- Las características de textura indican información sobre la distribución espacial del color en la imagen.

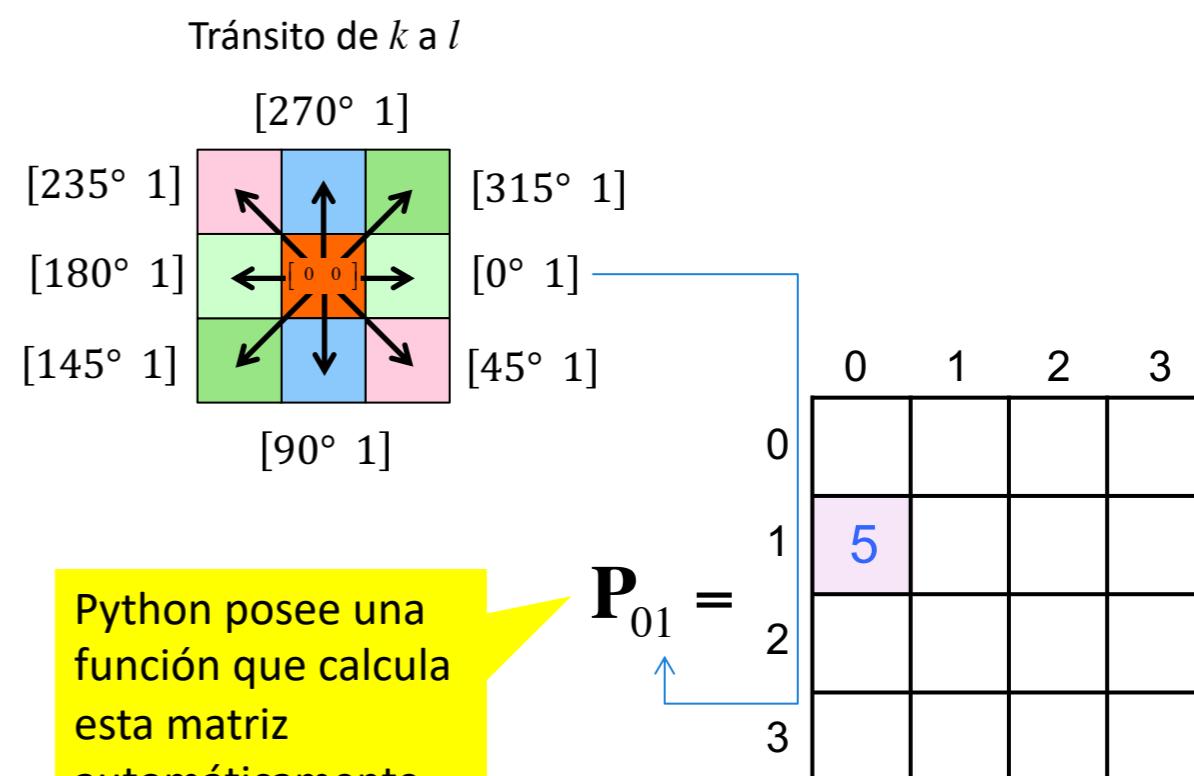


■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura**

- El procedimiento clásico para calcular la textura proviene del cálculo de la matriz de co-ocurrencia. El proceso es el siguiente.
- Sea $\mathbf{P}_{kl}(i, j)$ la matriz de co-ocurrencia donde el elemento $p_{kl}(i, j)$ es la frecuencia o el número de veces que aparece el valor de intensidad i y j al transitar de k a l

		j	1	2	3	4	5	6
		i	1	2	3	4	5	6
0	0	1	0	1	2			
1	0	0	1	0	3			
3	1	0	0	1	1			
2	2	2	1	3	0			
1	0	3	0	2	2			
3	0	1	2	3	1			



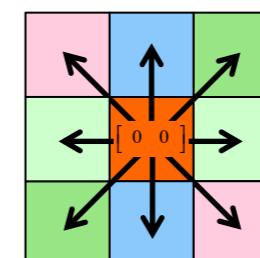
■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura**

- El procedimiento clásico para calcular la textura proviene del cálculo de la matriz de co-ocurrencia. El proceso es el siguiente.
- Sea $\mathbf{P}_{kl}(i, j)$ la matriz de co-ocurrencia donde el elemento $p_{kl}(i, j)$ es la frecuencia o el número de veces que aparece el valor de intensidad i y j al transitar de k a l

		j	1	2	3	4	5	6
		i	1	2	3	4	5	6
0	0	1	0	1	2			
1	0	0	1	0	3			
2	1	0	0	1	1			
3	2	2	2	1	3	0		
4	1	0	3	0	2	2		
5	3	0	1	2	3	1		

Tránsito de k a l



Python posee una función que calcula esta matriz automáticamente

$$\mathbf{P}_{01} =$$

	0	1	2	3
0	3	5	1	2
1	5	1	2	1
2	0	1	3	1
3	3	2	0	0

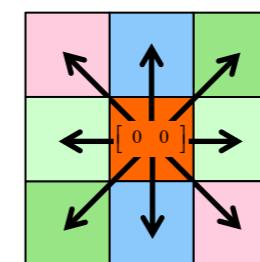
■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura**

- El procedimiento clásico para calcular la textura proviene del cálculo de la matriz de co-ocurrencia. El proceso es el siguiente.
- Sea $\mathbf{P}_{kl}(i, j)$ la matriz de co-ocurrencia donde el elemento $p_{kl}(i, j)$ es la frecuencia o el número de veces que aparece el valor de intensidad $i - j$ al transitar de k a l

		j	1	2	3	4	5	6
		i	1	2	3	4	5	6
0	0	1	0	0	1	0	1	2
1	0	0	1	0	1	0	3	
3	1	0	0	1	0	1	1	
2	2	2	1	3	0			
1	0	3	0	2	2			
3	0	1	2	3	1			

Tránsito de k a l



[90° 1]

$$\mathbf{P}_{10} =$$

0	1	2	3
3	5	3	0
5	0	1	3
1	2	0	3
0	2	2	0

■ Características cromáticas



- Python posee una función que permite modificar el número de niveles en la imagen, modificar la dirección de tránsito (de k a l), seleccionar el rango de intensidad a analizar, entre otros.

	j					
i	1	2	3	4	5	6
1	0	0	1	0	1	2
2	1	0	0	1	0	3
3	3	1	0	0	1	1
4	2	2	2	1	3	0
5	1	0	3	0	2	2
6	3	0	1	2	3	1

```
img= [[0, 0, 1, 0, 1, 2],
      [1, 0, 0, 1, 0, 3],
      [3, 1, 0, 0, 1, 1],
      [2, 2, 2, 1, 3, 0],
      [1, 0, 3, 0, 2, 2],
      [3, 0, 1, 2, 3, 1]]

img = np.array(img, 'uint8')
l = 4 #numero de niveles de la imagen
P_0_1 = greycomatrix(img,
                      distances=[1],
                      angles=[0],
                      levels=l,
                      symmetric=False,
                      normed=False)

g = P_0_1.reshape(l,l)
```

greycomatrix

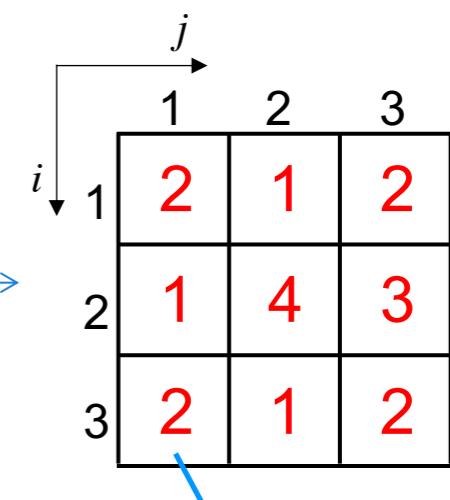
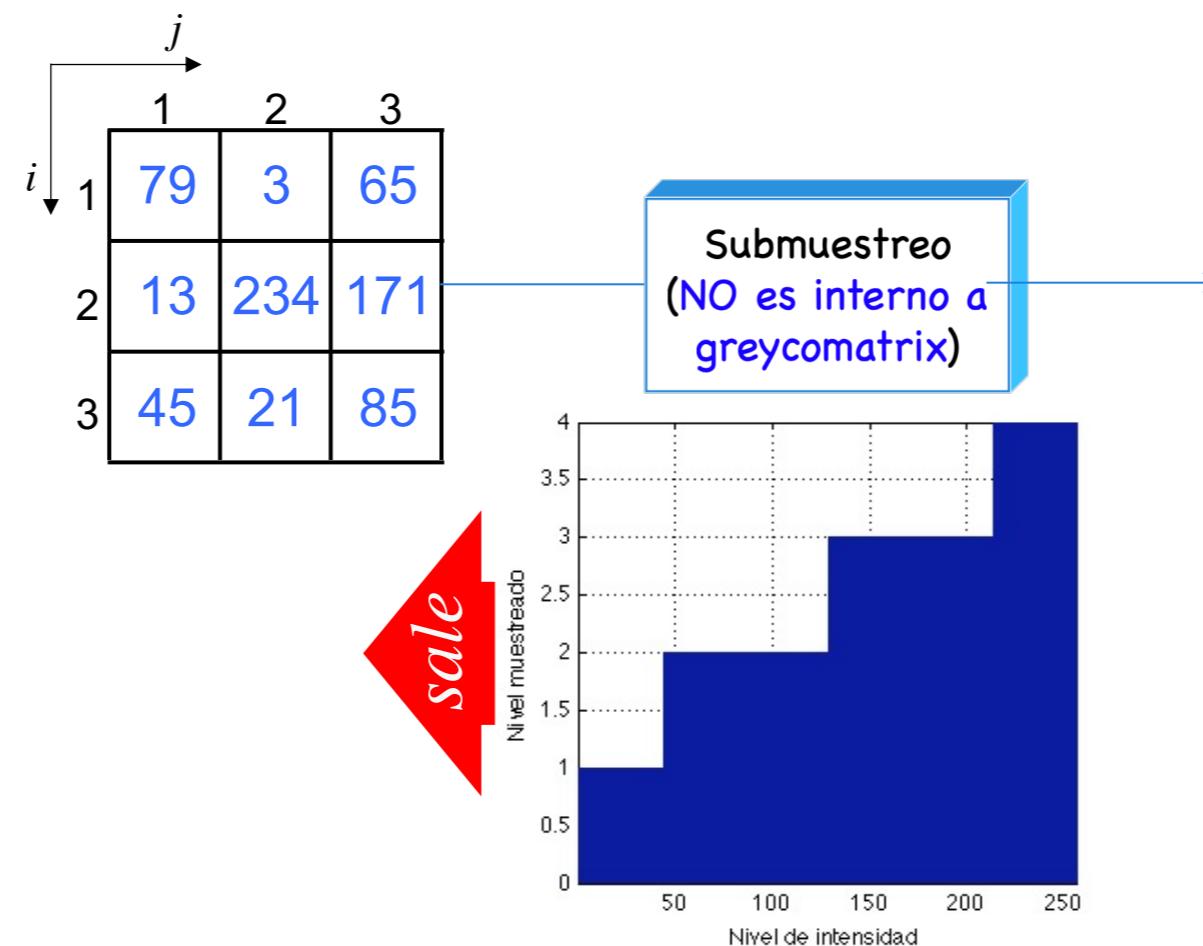
	0	1	2	3
0	3	5	1	2
1	5	1	2	1
2	0	1	3	1
3	3	2	0	0

$$\mathbf{P}_{01} =$$

■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura

- Como usted sabe, las imágenes tienen 256 niveles de intensidad. Esto implicaría crear una matriz de 256×256 , lo cual es costoso computacionalmente. Por ello es recomendable sub muestrear.



Python determina la matriz de Co-ocurrencia sobre una imagen de 256 niveles. Para disminuir el cómputo debemos aplicar un submuestreo a la imagen original



■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura**

- Como usted sabe, las imágenes tienen 256 niveles de intensidad. Esto implicaría crear una matriz de 256 x 256, **lo cual es costoso computacionalmente**. Por ello es recomendable sub muestrear.

		j	1	2	3
		i	1	2	3
Imagen	Submuestreo	1	79	3	65
		2	13	234	171
Imagen	Submuestreo	3	45	21	85

		j	1	2	3
		i	1	2	3
Imagen	Submuestreo	1	2	1	2
		2	1	4	3
Imagen	Submuestreo	3	2	1	2

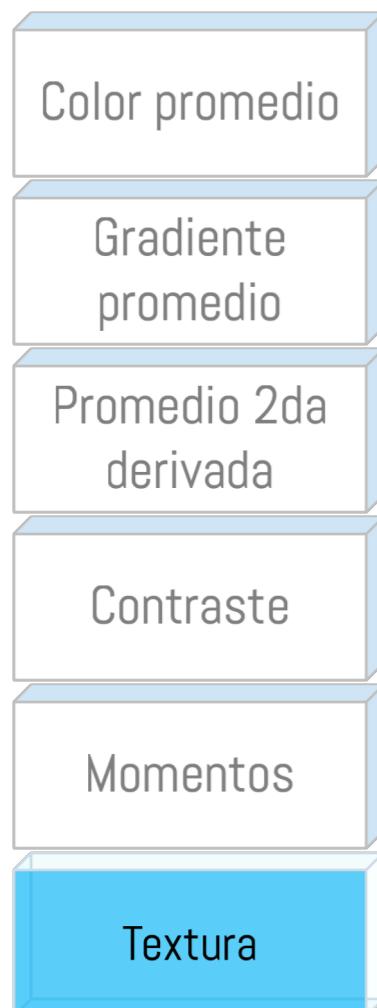
$$\mathbf{P}_{01} = \begin{matrix} & & & & \\ & 0 & 1 & 2 & 3 \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{matrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{matrix} \end{matrix}$$

Imagen

Submuestreo

Resultado

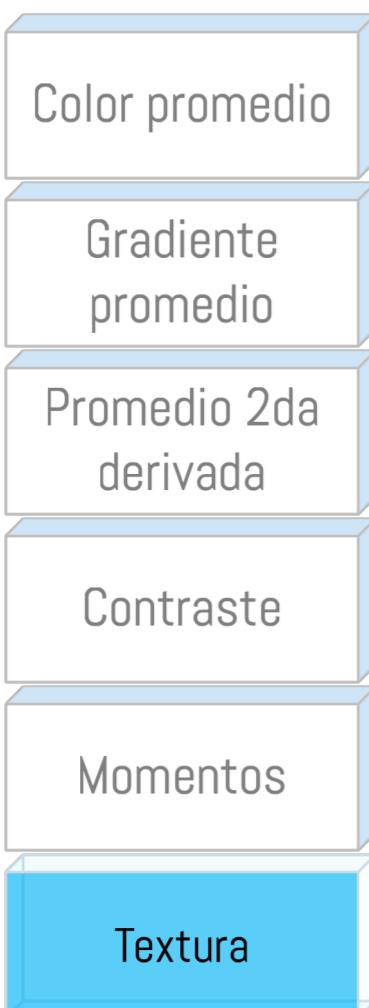
■ Características cromáticas



- Sólo a partir de la matriz de co-ocurrencia podemos extraer un conjunto de características definidas por Haralick.

descriptor	fórmula	descripción	rango
Contraste	$\sum_{i,j} i - j ^2 \mathbf{P}(i, j)$	Mide el contraste de intensidad entre un píxel y su vecino en toda la imagen.	$[0 \rightarrow (size(P,1)-1)^2]$
Entropía	$-\sum_{i,j} \mathbf{P}(i, j) \cdot \log(\mathbf{P}(i, j))$	Mide la cantidad de información en los datos. Es máxima cuando los elementos son aleatorios.	$[bits]$
Energía	$\sum_{i,j} \mathbf{P}(i, j)^2$	Es una medida de la uniformidad. Es máximo cuando la imagen es constante.	$[0 \rightarrow 1]$
Homogeneidad	$\sum_{i,j} \frac{\mathbf{P}(i, j)}{1 + (i - j)^2}$	Mide la relación espacial de distribución. Es cero cuando la matriz es distribuida y uno cuando está en la diagonal.	$[0 \rightarrow 1]$
Correlación	$\sum_{i,j} \frac{(i - \mu_i) \cdot (j - \mu_j) \cdot \mathbf{P}(i, j)}{\sigma_i \sigma_j}$	Mide la correlación de un píxel con su vecino a lo largo de toda la imagen.	$[-1 \rightarrow 1]$

■ Características cromáticas



- Analicemos por ejemplo el descriptor de correlación

$$\sum_{i,j} \frac{(i - \mu_i)(j - \mu_j) \cdot P(i, j)}{\sigma_i \sigma_j}$$

Primer paso: Normalizar

$$P(i, j) = \frac{P(i, j)}{\sum_{i,j} P(i, j)}$$

Segundo paso: Calcular las medias:

$$\mu_i = \sum_{i,j} i \cdot P(i, j) \quad \mu_j = \sum_{i,j} j \cdot P(i, j)$$

Tercer paso: Calcular la desviación estándar :

$$\sigma_i = \sqrt{\sum_{i,j} (i - \mu_i)^2 P(i, j)} \quad \sigma_j = \sqrt{\sum_{i,j} (j - \mu_j)^2 P(i, j)}$$

Cuarto paso: Calcular la correlación

$$\sum_{i,j} \frac{(i - \mu_i)(j - \mu_j) \cdot P(i, j)}{\sigma_i \sigma_j}$$

```
def texture_correlation(P):
    fil, col = P.shape
    vfil = np.arange(0, fil)
    vcol = np.arange(0, col)
    c, r = np.meshgrid(vfil, vcol)
```

```
# Normalizamos
P = P / np.sum(P)
```

```
# Media en dirección del pixel
mr = np.sum(r*P)
mc = np.sum(c*P)
```

```
# Desviacion estandar
Sr = np.sqrt(np.sum((r-mr)**2*P))
Sc = np.sqrt(np.sum((c-mc)**2*P))
```

```
c = np.sum((r-mr)*(c-mc)*P)/(Sr*Sc)
return c
```

■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura

- Python posee la función `greycoprops` que extrae algunas de las características antes mencionadas. Veamos un ejemplo.

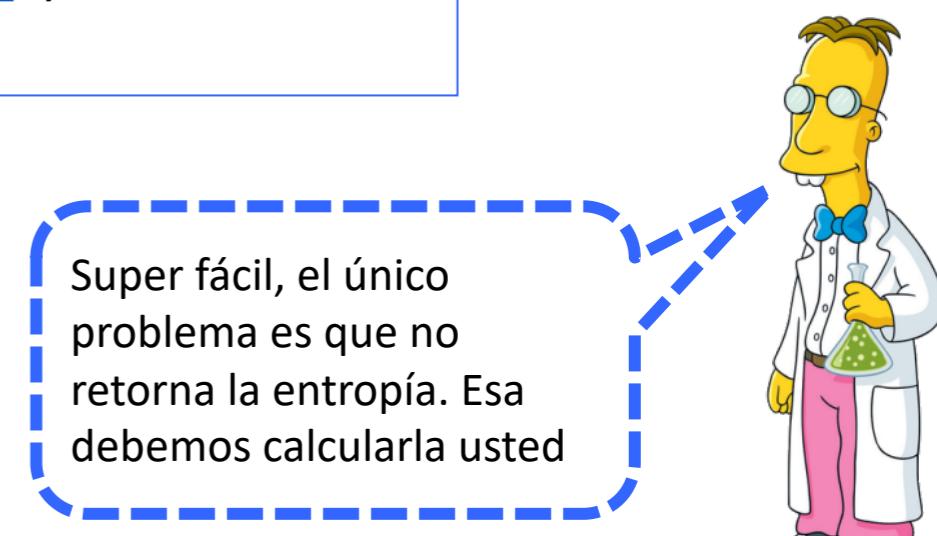
```
l= 12 #numero de niveles de la imagen  
  
P_1_0 = greycomatrix(new_gray,  
                      distances=[1],  
                      angles=[0], levels=l,  
                      symmetric=False,  
                      normed=False)  
  
features = ['contrast','correlation',  
           'dissimilarity','homogeneity','ASM','energy']  
  
for ft in features:  
    sts = float(greycoprops(P_1_0, ft))  
    print(f'{ft}: {sts:2.4f}')
```

				↓
0	1	2	3	
0	3	5	3	0
1	5	0	1	3
2	1	2	0	3
3	0	2	2	0

Resultado

```
contrast: 5.0698  
correlation: 0.5915  
dissimilarity: 1.7414  
homogeneity: 0.4167  
ASM: 0.0154  
energy: 0.1242
```

Super fácil, el único problema es que no retorna la entropía. Esa debemos calcularla usted



■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura

Sobre esta matriz podemos calcular los descriptores

	1	2	3	4
1	3	5	1	2
2	5	1	2	1
3	0	1	3	1
4	3	2	0	0

- Para emplear los descriptores, siempre debemos normalizar haciendo que la **suma de la matriz sea uno**. Este proceso consiste en sumar todos los elementos de la matriz y luego dividir cada elemento por dicha suma (**greycoprops** lo hace internamente).

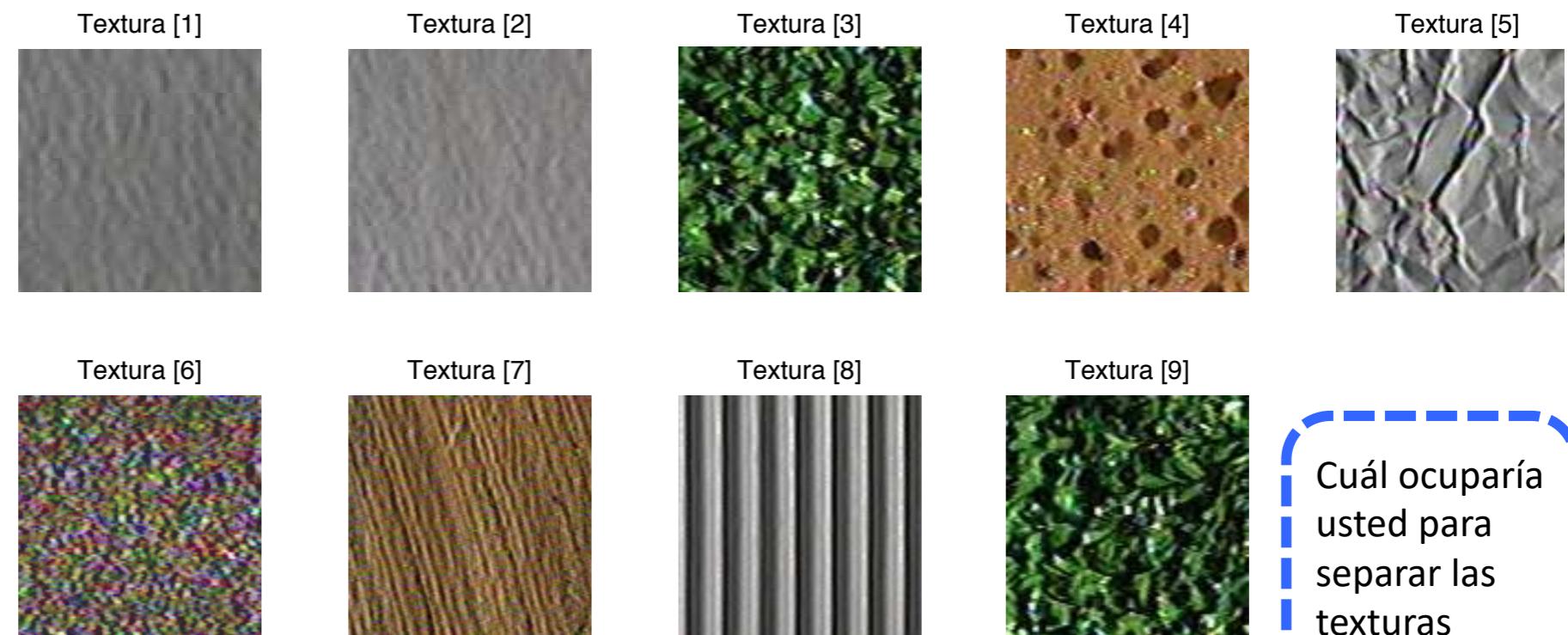
```
img= [ [0, 0, 1, 0, 1, 2],  
       [1, 0, 0, 1, 0, 3],  
       [3, 1, 0, 0, 1, 1],  
       [2, 2, 2, 1, 3, 0],  
       [1, 0, 3, 0, 2, 2],  
       [3, 0, 1, 2, 3, 1]] }
```

1	2	3	4	5	6	
1	0	0	1	0	1	2
2	1	0	0	1	0	3
3	3	1	0	0	1	1
4	2	2	2	1	3	0
5	1	0	3	0	2	2
6	3	0	1	2	3	1

```
img = np.array(img, 'uint8')  
  
l = 4 #numero de niveles de la imagen  
P_0_1 = greycomatrix(img,  
distances=[1],  
angles=[0],  
levels=l,  
symmetric=False,  
normed=False)  
  
g = P_1_0.reshape(l,l)
```

■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura



- Analicemos los siguientes resultados para nueve texturas diferentes

	Energía	Contraste	Correlación	Homogeneidad	Entropía	Momento
Textura [1]:	0.0873	12.8823	0.6008	0.5359	3.9902	0.4572
Textura [2]:	0.0809	11.8414	0.6258	0.5332	4.0887	0.4522
Textura [3]:	0.0390	18.0805	0.7537	0.5229	6.2451	0.4648
Textura [4]:	0.0354	7.2391	0.7763	0.5456	5.9993	0.4775
Textura [5]:	0.0085	17.0069	0.7562	0.4551	7.8791	0.3857
Textura [6]:	0.0108	31.1488	0.5887	0.3762	7.3759	0.2940
Textura [7]:	0.0118	17.1816	0.5238	0.3839	7.0999	0.2929
Textura [8]:	0.0074	12.7922	0.8820	0.3808	7.7091	0.2930
Textura [9]:	0.0478	19.1941	0.7418	0.5278	6.1327	0.4718

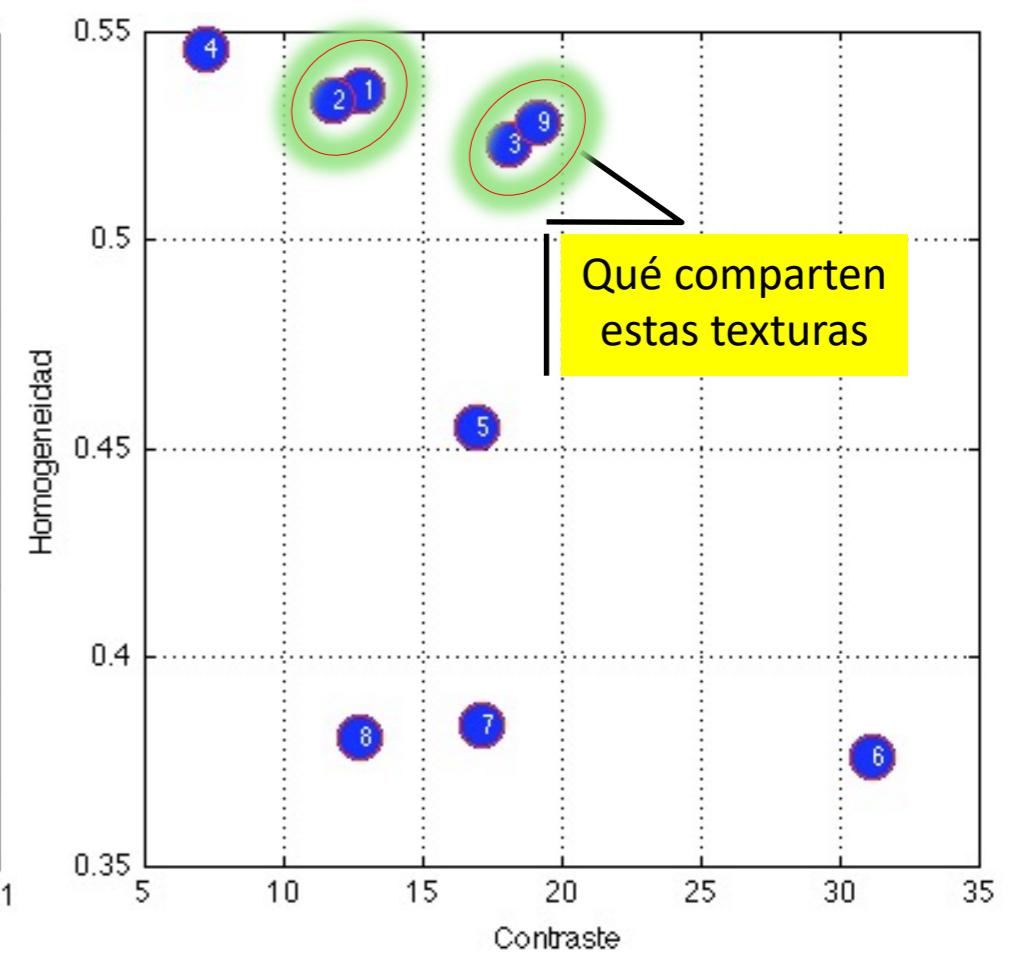
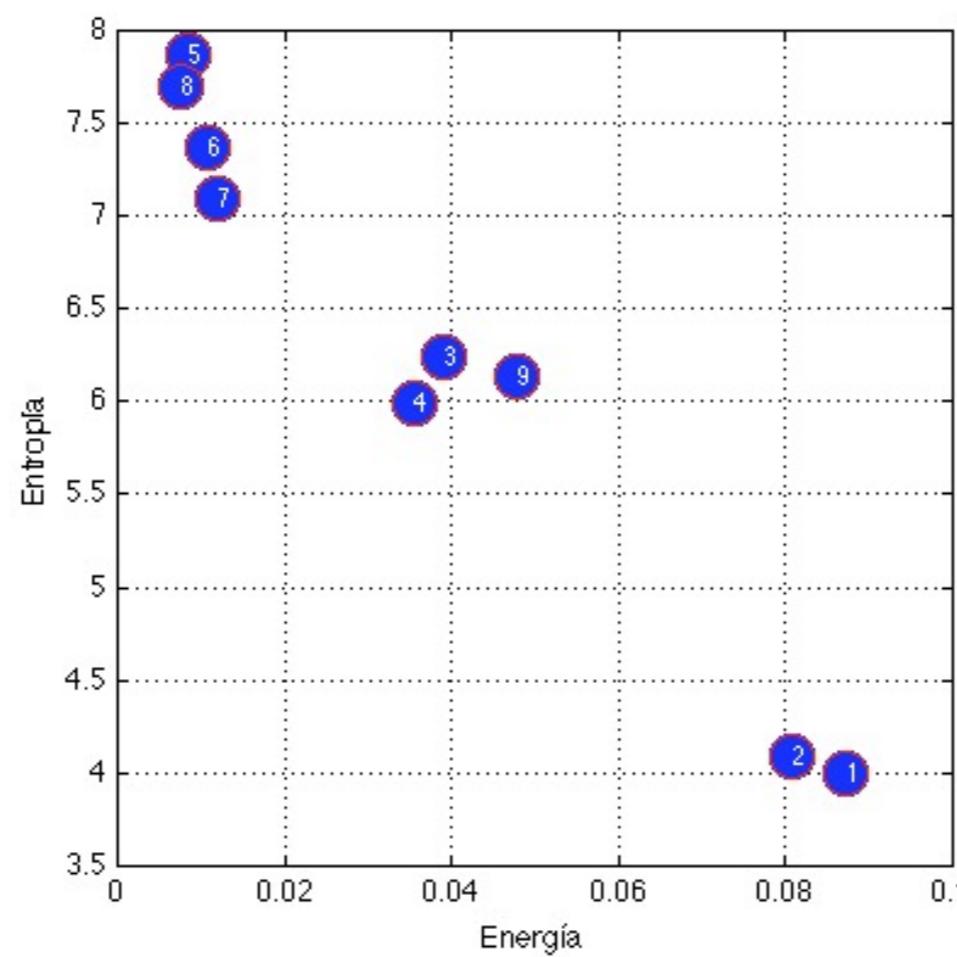
Cuál ocuparía usted para separar las texturas



■ Características cromáticas

- Color promedio
- Gradiente promedio
- Promedio 2da derivada
- Contraste
- Momentos
- Textura

- Analicemos los siguientes resultados para nueve texturas diferentes



■ Geométricas

- Centro de masa
- Tamaño
- Perímetro
- Redondez
- Momentos binarios
- Descriptores de Fourier
- Elipses
- Distancia al borde

■ Cromáticas

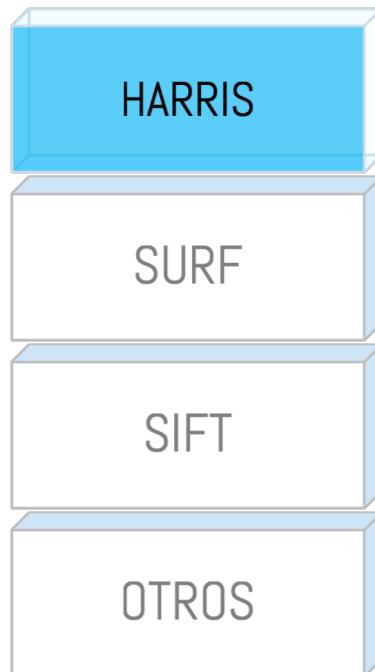
- Color promedio
- Gradiente promedio
- Promedio segunda derivada
- Contraste
- Momentos de color
- Textura

■ Otros descriptores

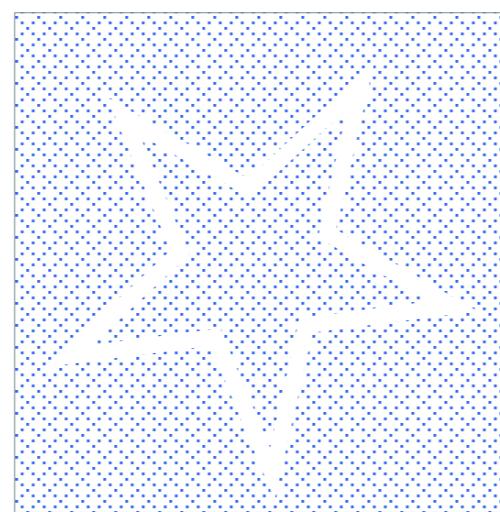
- HARRIS (Esquinas)
- SURF
- SIFT
- PHOG
- GPD



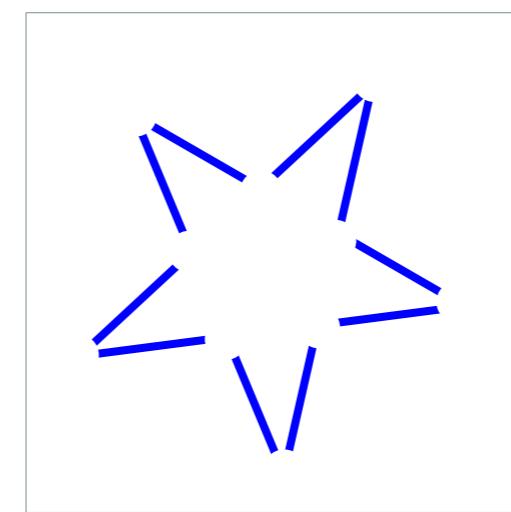
■ Otros descriptores



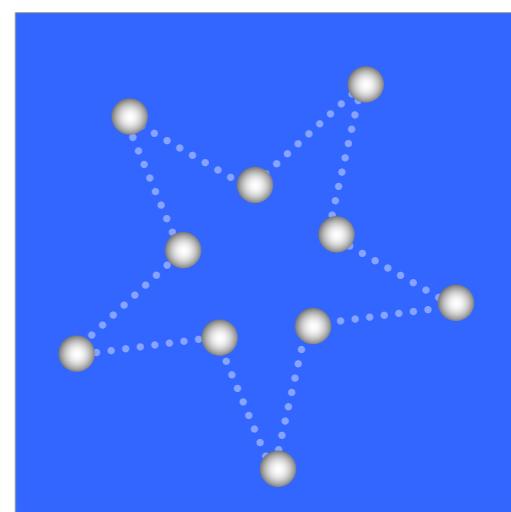
- En los descriptores previos sabemos de antemano la región donde aplicaremos el descriptor. Sin embargo, esto no siempre será así.
- Para detectar automáticamente **puntos de interés** debemos utilizar un algoritmo que detecte cambios significativos en la imagen. Uno de los más empleados es el detector de esquinas de Harris (1988).



Fondo



Bordes



Esquinas

C. Harris and M.J. Stephens. A combined corner and edge detector. In Alvey Vision Conference, pages 147–152, 1988

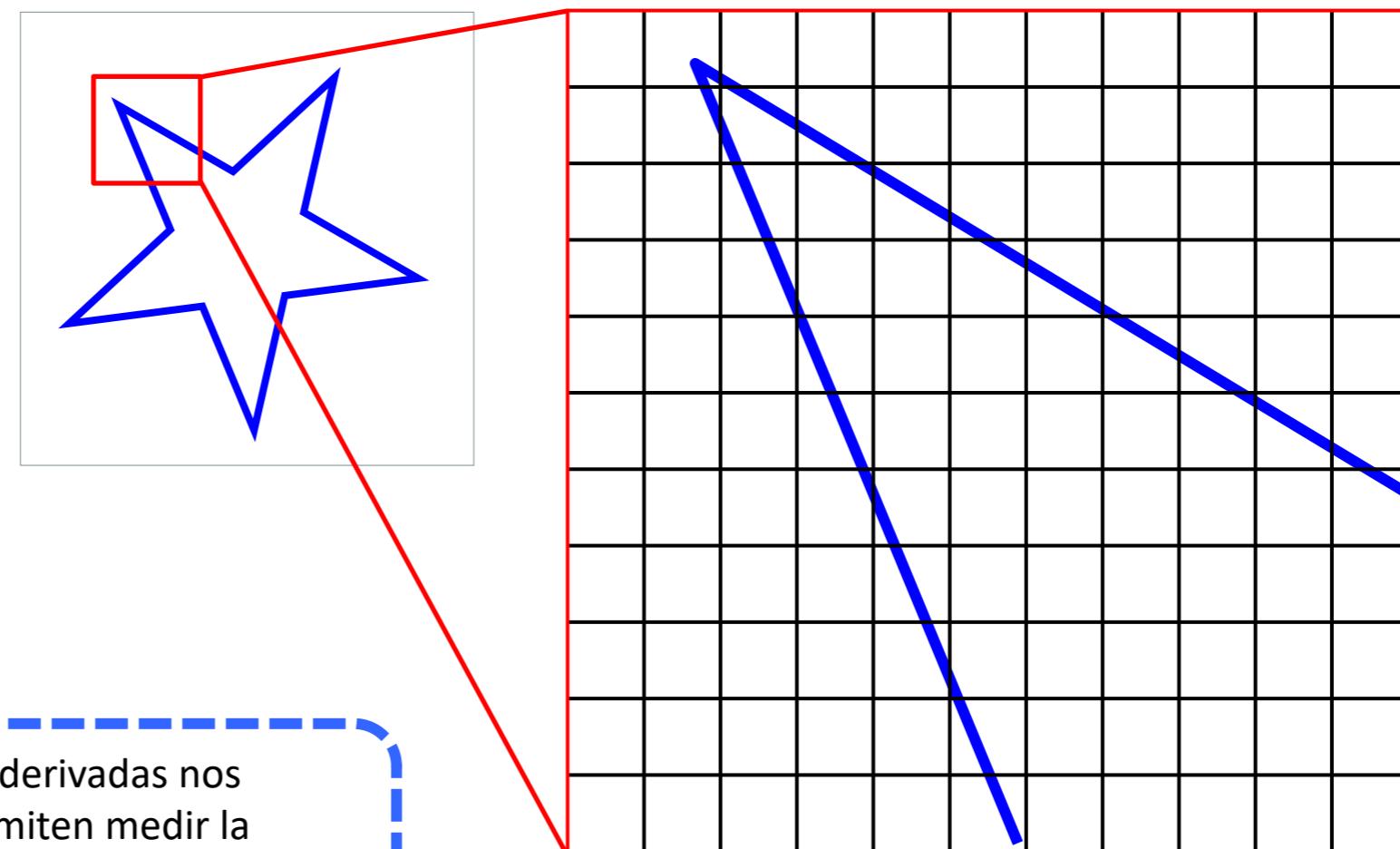
■ Otros descriptores



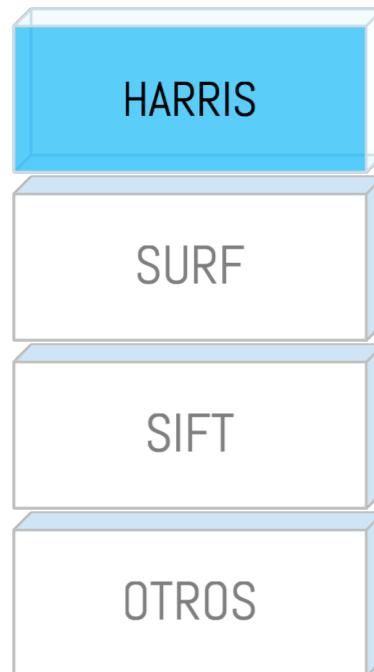
- El descriptor de Harris determina la intensidad de cambio en todas las direcciones de una ventana.



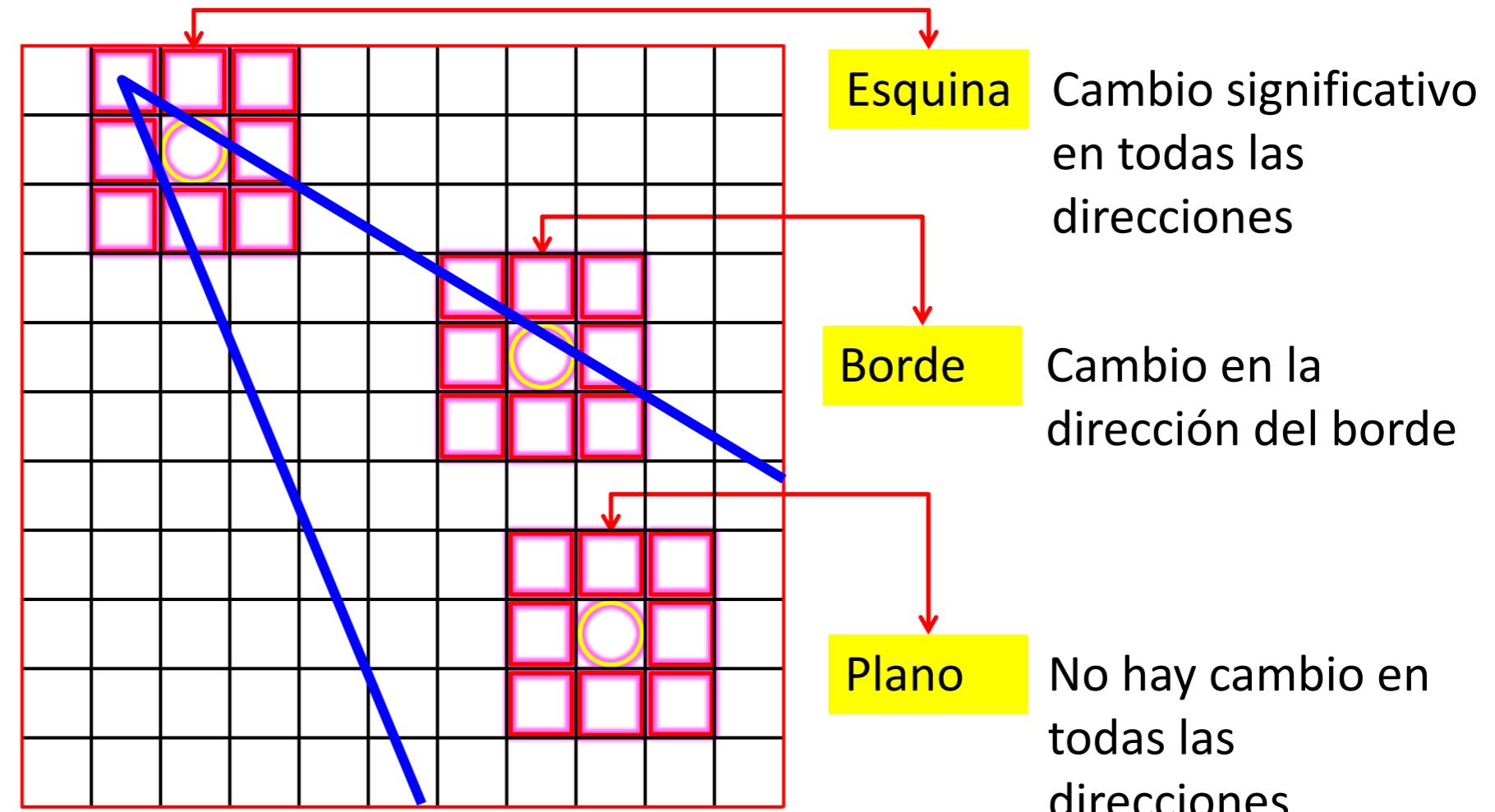
Las derivadas nos permiten medir la intensidad de cambio de un pixel a sus vecinos



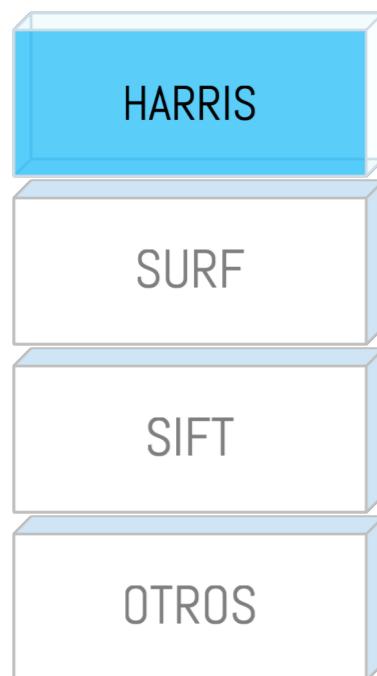
■ Otros descriptores



- El descriptor de Harris determina la intensidad de cambio en todas las direcciones de una ventana.

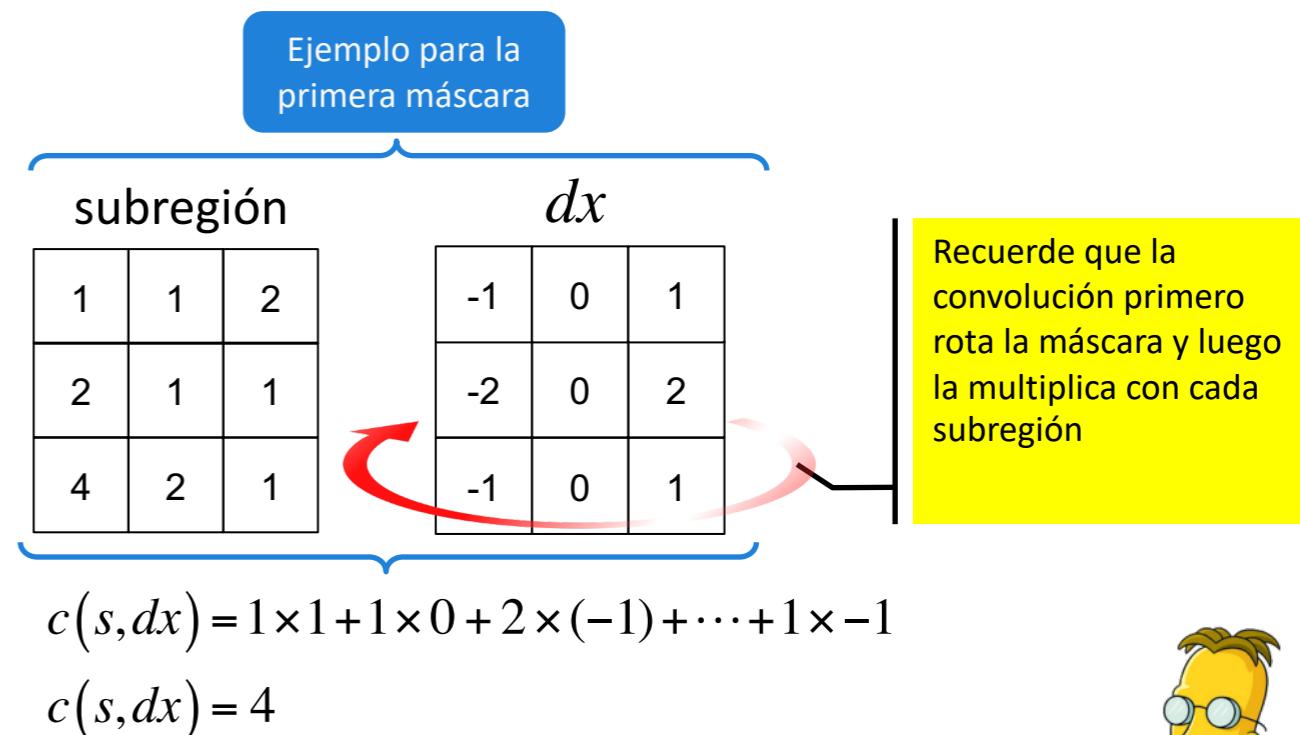


■ Otros descriptores



PRIMERO: Debemos calcular las derivadas direccionales en el eje-x y eje-y. Este proceso aplica una máscara que recorre la imagen píxel por píxel. Este proceso lo realizamos con una máscara que convoluciona la imagen.

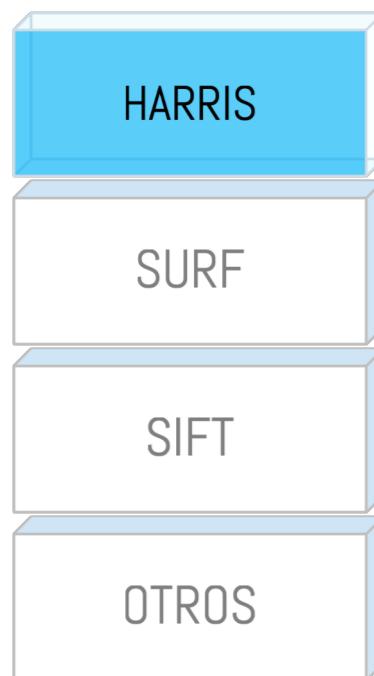
		j	1	2	3	4	5	6
		i	1	2	3	4	5	6
		1	1	1	2	1	2	3
		2	2	1	1	2	1	4
		3	4	2	1	1	2	2
		4	3	3	3	2	4	1
		5	2	1	4	1	3	3
		6	4	1	2	3	4	2



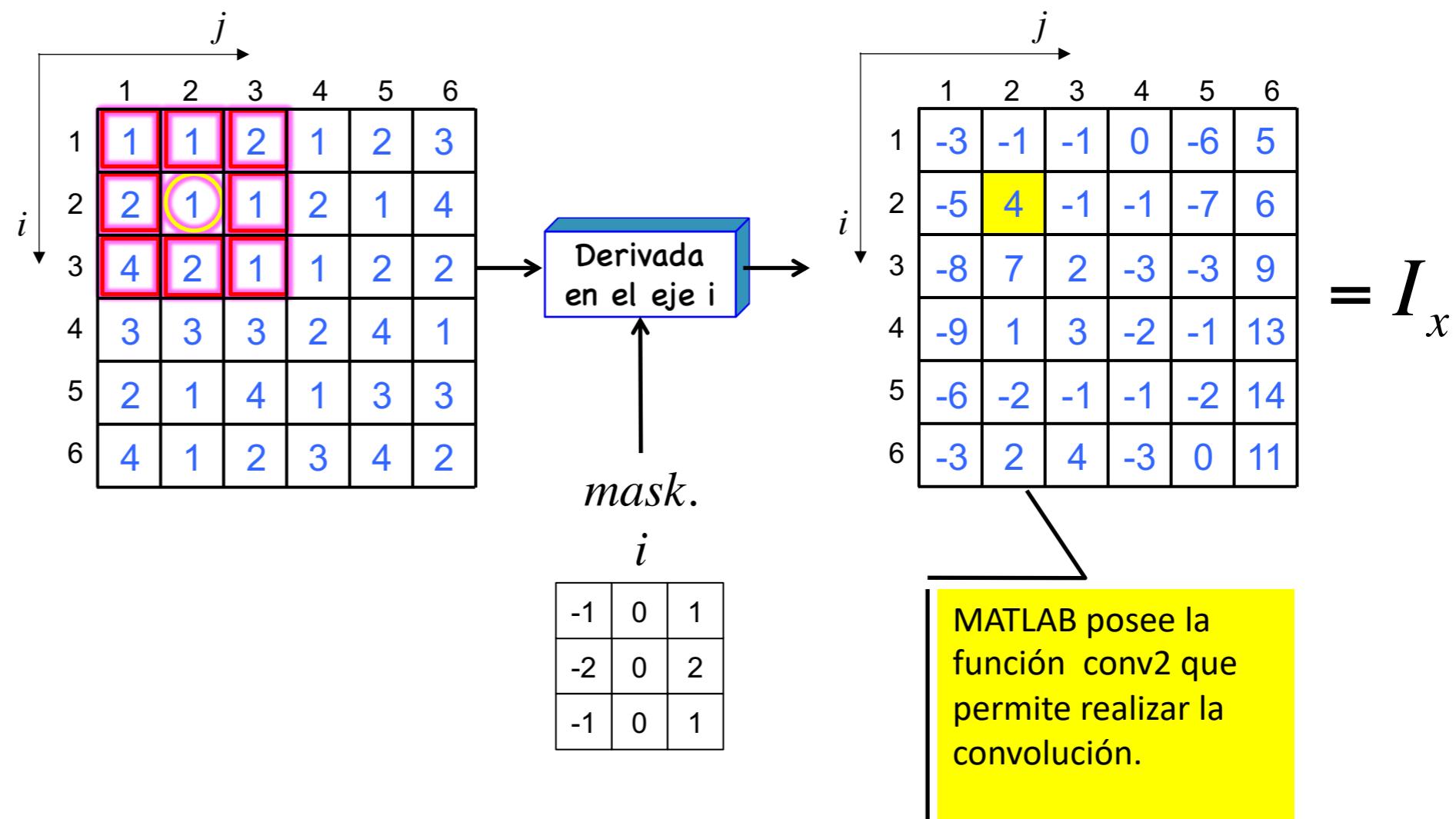
Cada resultado es almacenado en una nueva matriz sobre la cual calcularemos nuestro descriptor



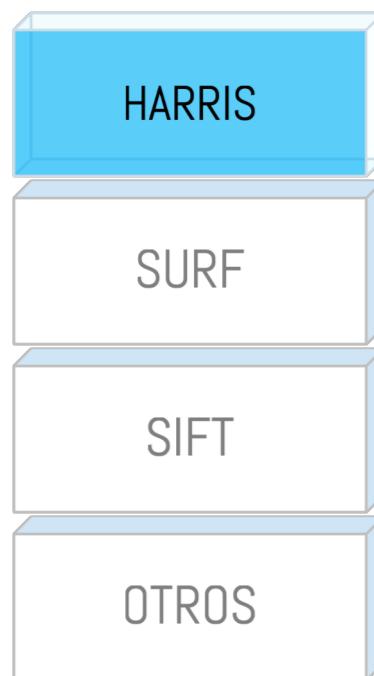
■ Otros descriptores



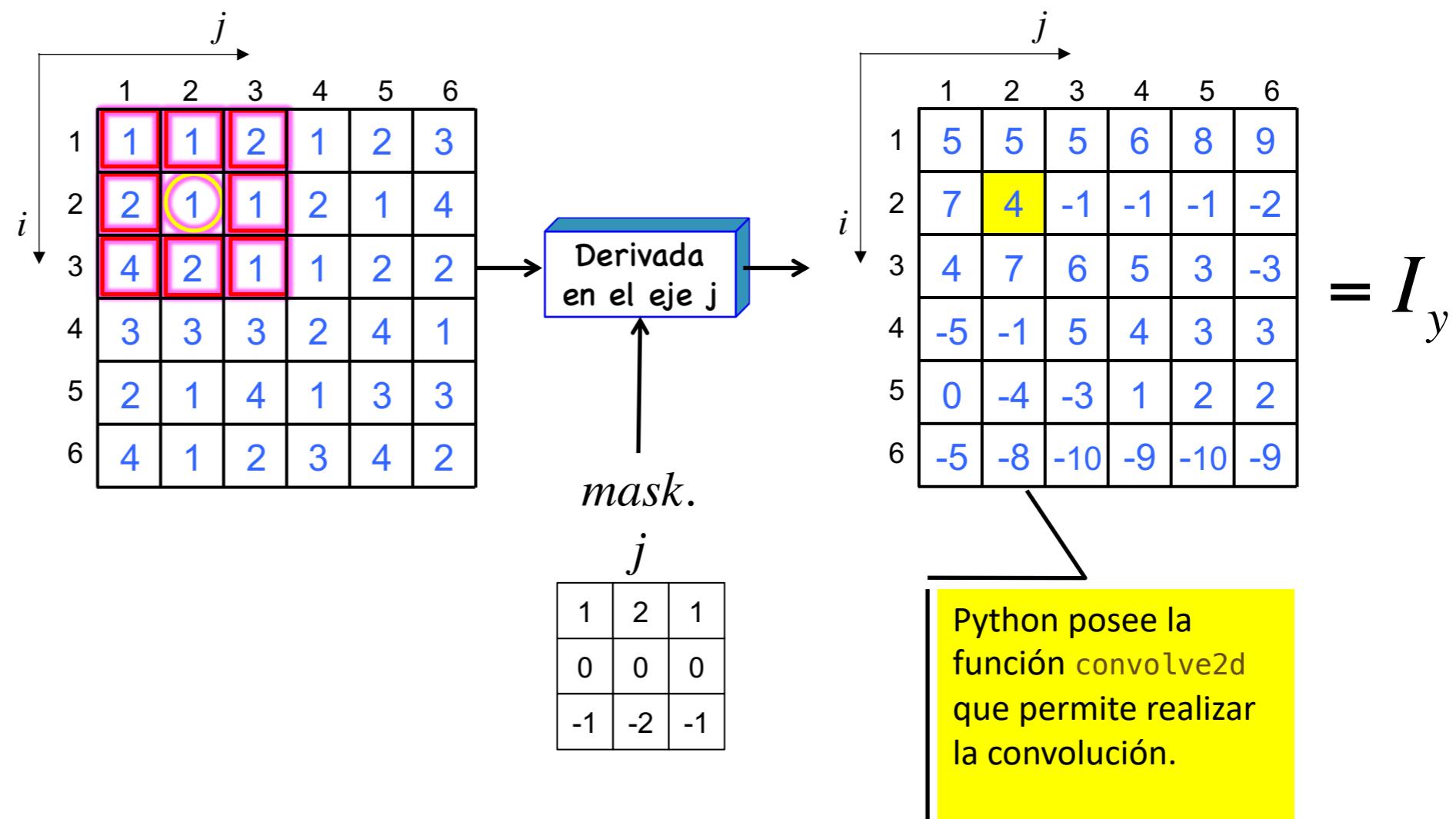
PRIMERO: Debemos calcular las derivadas direccionales en el eje-x y eje-y. Este proceso aplica una máscara que recorre la imagen píxel por píxel. Este proceso lo realizamos con una máscara que convoluciona la imagen.



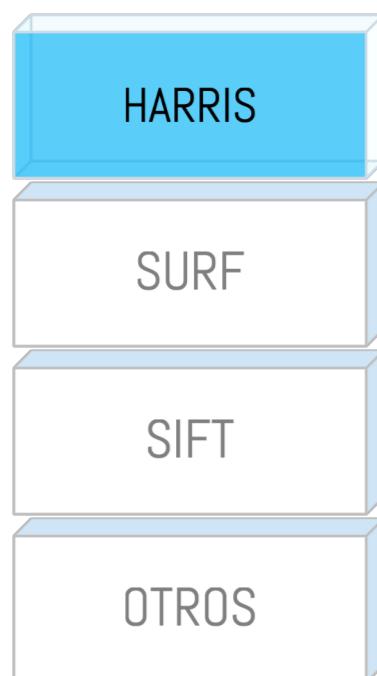
■ Otros descriptores



PRIMERO: Debemos calcular las derivadas direccionales en el eje-x y eje-y. Este proceso aplica una máscara que recorre la imagen píxel por píxel. Este proceso lo realizamos con una máscara que convoluciona la imagen.



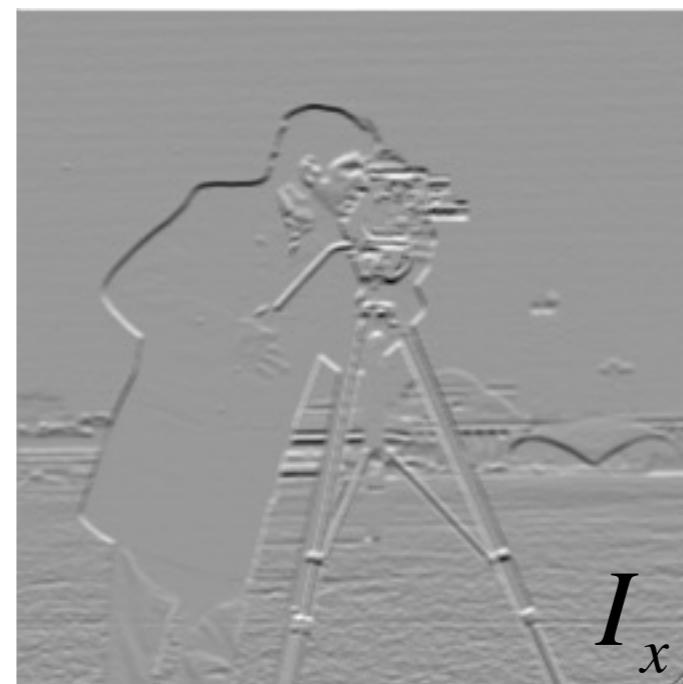
■ Otros descriptores



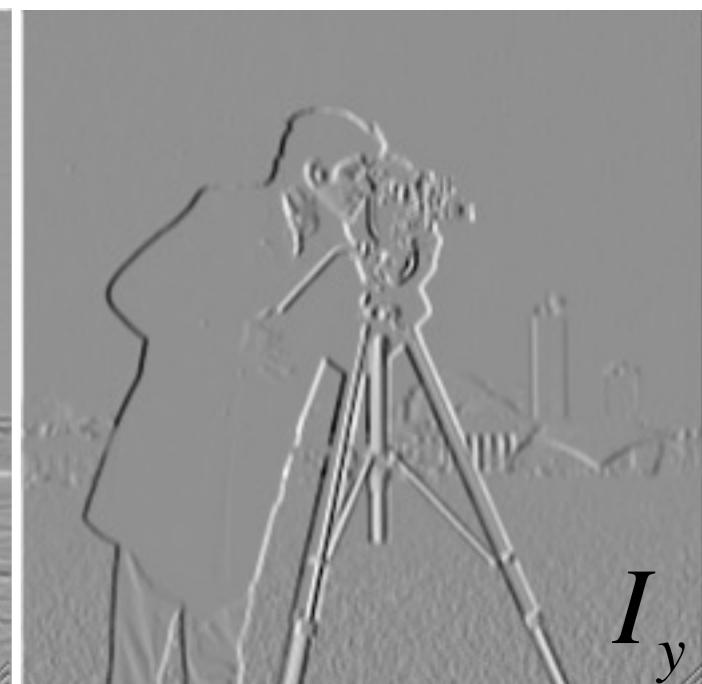
Aplicado a una
imagen, este es el
resultado

PRIMERO: Debemos calcular las derivadas direccionales en el eje-x y eje-y. Este proceso aplica una máscara que recorre la imagen píxel por píxel.

Derivada en Eje x



Derivada en Eje y

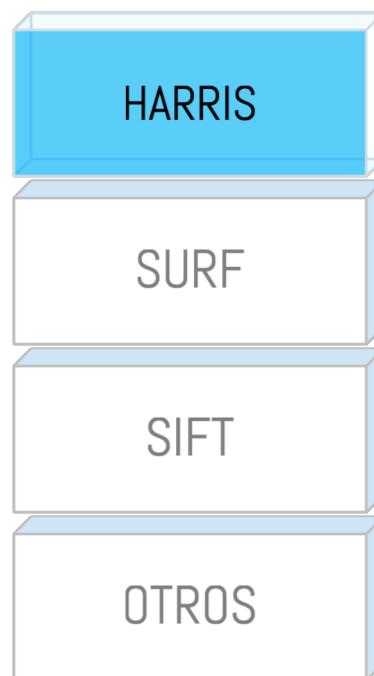


```
# Máscara X
dx = np.array([[-1,0,1], [-2,0,2], [-1, 0, 1]])

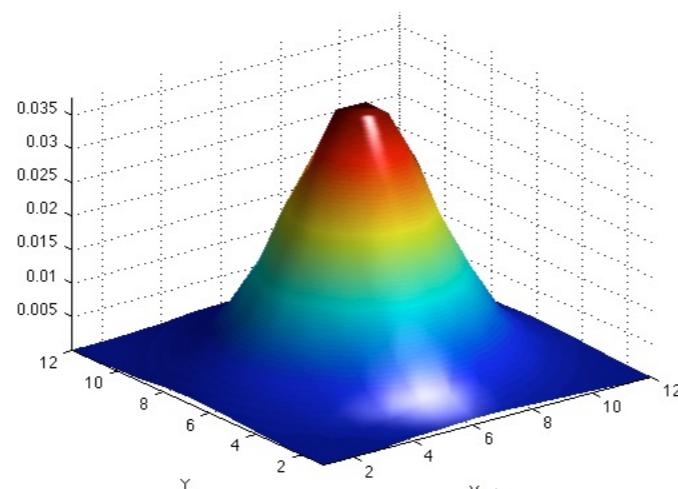
# Máscara Y
dy = np.array([[ 1,2,1], [ 0,0,0], [-1,-2,-1]])

# derivadas direccionales
Ix = convolve2d(im, dx, mode='same')
Iy = convolve2d(im, dy, mode='same')
```

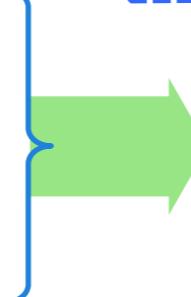
■ Otros descriptores



SEGUNDO: Sobre cada resultado previo calculamos las matrices A , B , C . Sin embargo, estas matrices contienen mucho ruido, por ello las suavizamos con un filtro gaussiano.



$$\begin{aligned} A &= I_x^2 \\ B &= I_x \cdot I_y \\ C &= I_y^2 \end{aligned}$$



```
# Mascara X
dx = np.array([[-1,0,1],[-2,0,2],[-1, 0, 1]])

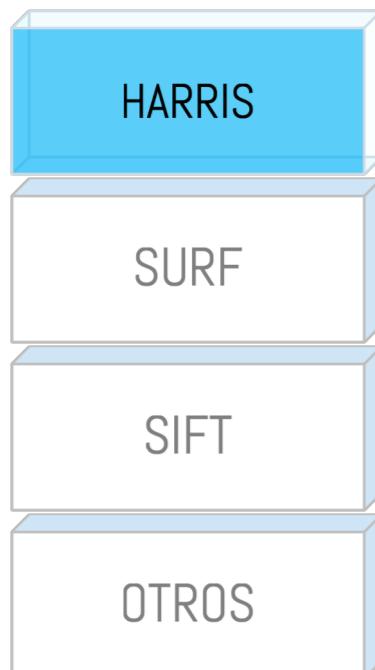
# Mascara Y
dy = np.array([[ 1,2,1],[ 0,0,0],[-1,-2,-1]])

# derivadas direccionales
Ix = convolve2d(im, dx, mode='same')
Iy = convolve2d(im, dy, mode='same')

# Convolucion de derivada con filtro gaussiano
g = fspecial_gauss(sze, sigma)

A = convolve2d(Ix**2, g, mode ='same')
B = convolve2d(Ix*Iy, g, mode= 'same')
C = convolve2d(Iy**2, g, mode= 'same')
```

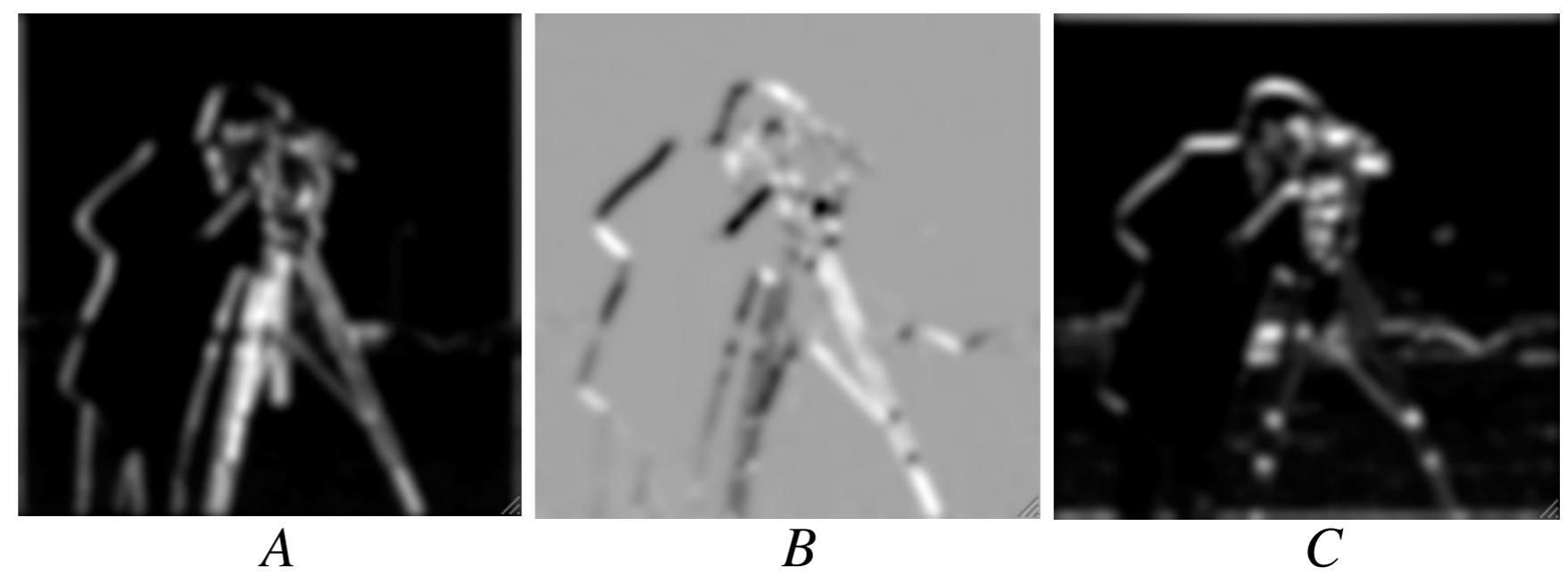
■ Otros descriptores



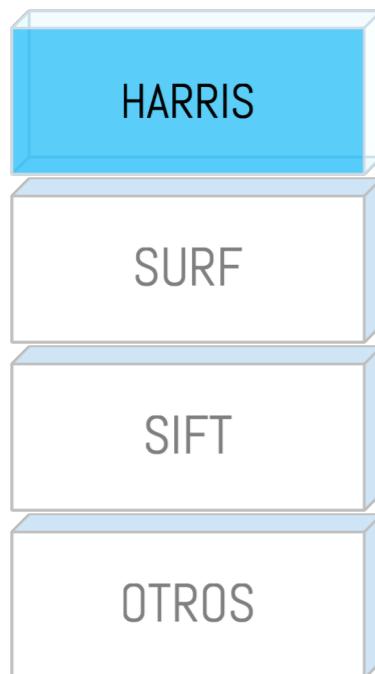
SEGUNDO: Sobre cada resultado previo calculamos las siguientes matrices. Sin embargo, estas matrices contienen mucho ruido, por ello las suavizamos con un filtro gaussiano

$$\left. \begin{array}{l} A = I_x^2 \\ B = I_x \cdot I_y \\ C = I_y^2 \end{array} \right\} \rightarrow Q(x, y) = \begin{bmatrix} I_x^2 & I_x \cdot I_y \\ I_x \cdot I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$$

Las matrices A , B , C conforman la matriz Q . Esta matriz contiene la información del cambio de intensidad en todas las direcciones.



■ Otros descriptores

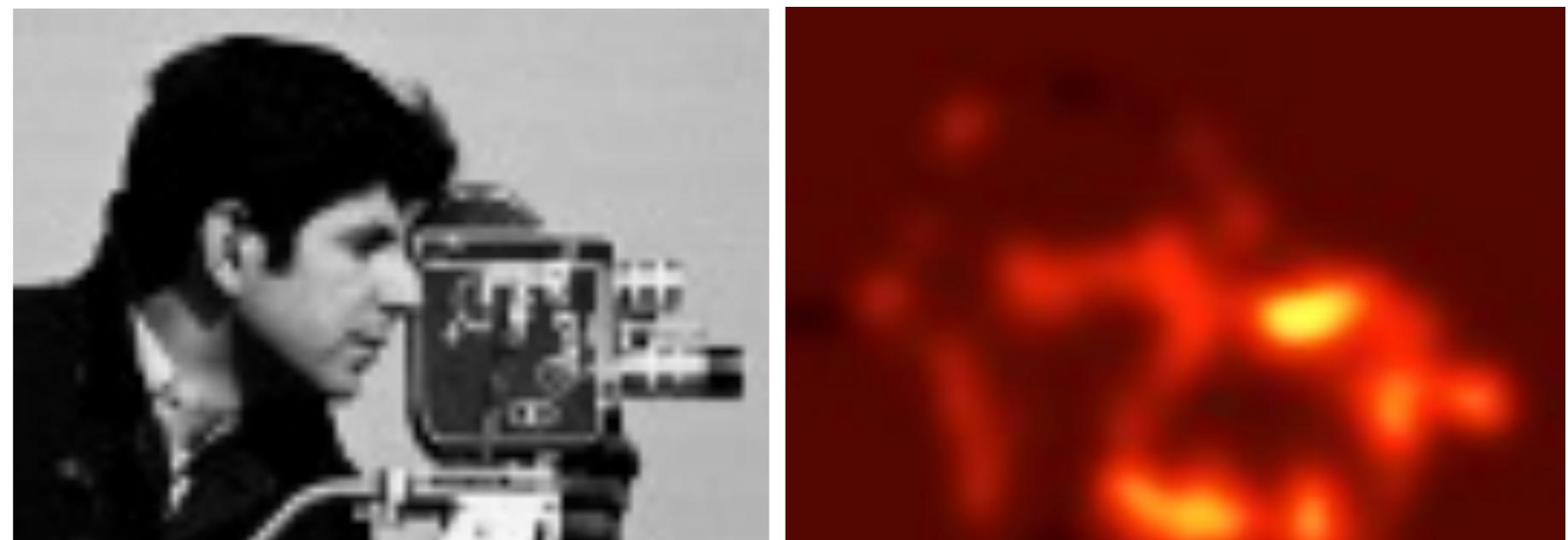


TERCERO: Aplicamos el criterio de Harris para determinar la respuesta a los bordes sobre la matriz Q .

$$Q(x,y) = \begin{bmatrix} I_x^2 & I_x \cdot I_y \\ I_x \cdot I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$$

Este matriz se construye para cada punto de la imagen

$$H = \det(Q) - k \cdot \text{traza}(Q)^2 \longrightarrow H = A \cdot C - B^2 - k \cdot (A + C)^2$$



La constante k tiene valor entre 0.04 y 0.06.

■ Otros descriptores

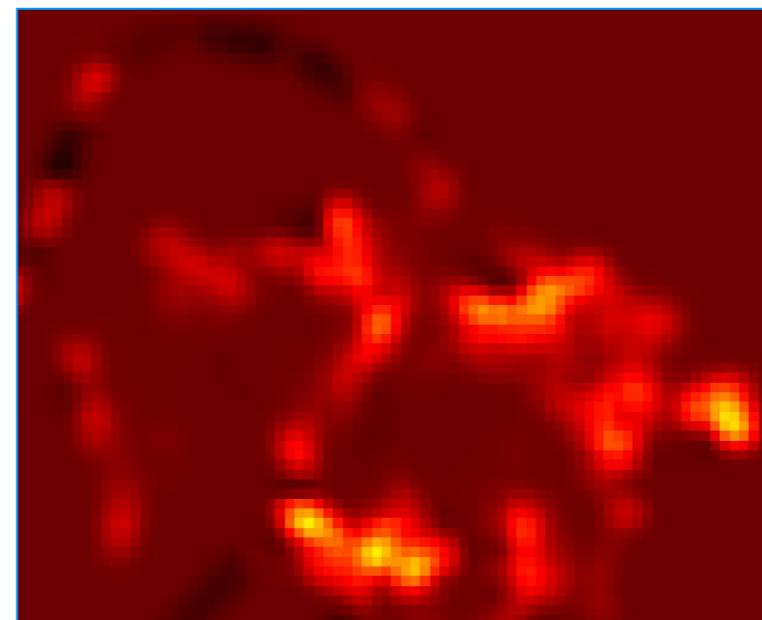
HARRIS

SURF

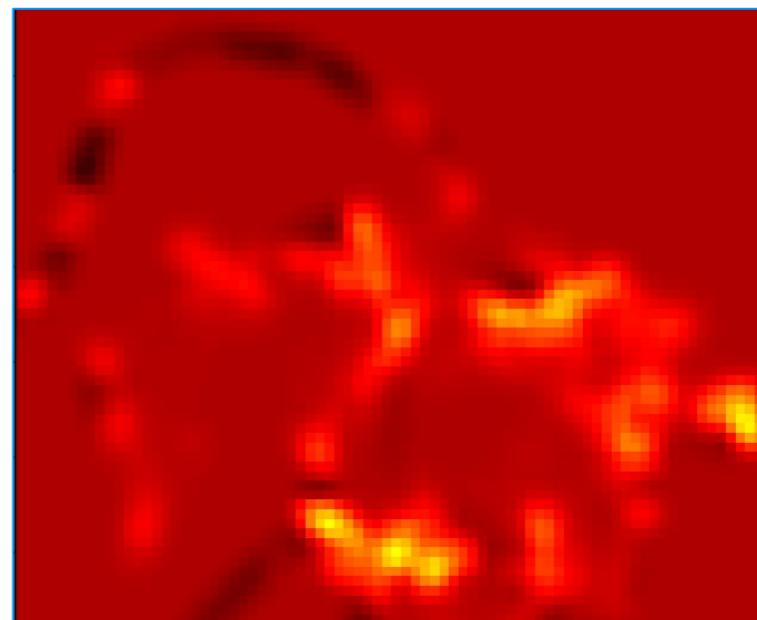
SIFT

OTROS

CUARTO: Buscamos los máximos locales que sean mayores a sus vecinos. Para ello podemos variar el parámetro k de la ecuación.



$k = 0.04$



$k = 0.06$

```
A = convolve2d(Ix**2, g, mode ='same')
B = convolve2d(Ix*Iy, g, mode= 'same')
C = convolve2d(Iy**2, g, mode= 'same')

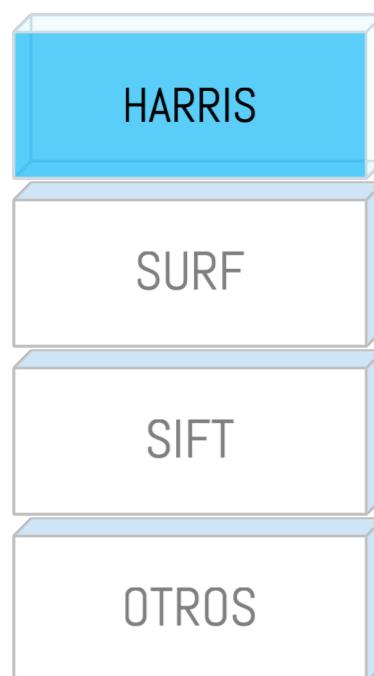
# Medición de respuesta de la esquina
k = 0.04

# Parametro de Harris
H = (A*C - B**2) - k*(A + C)**2
```

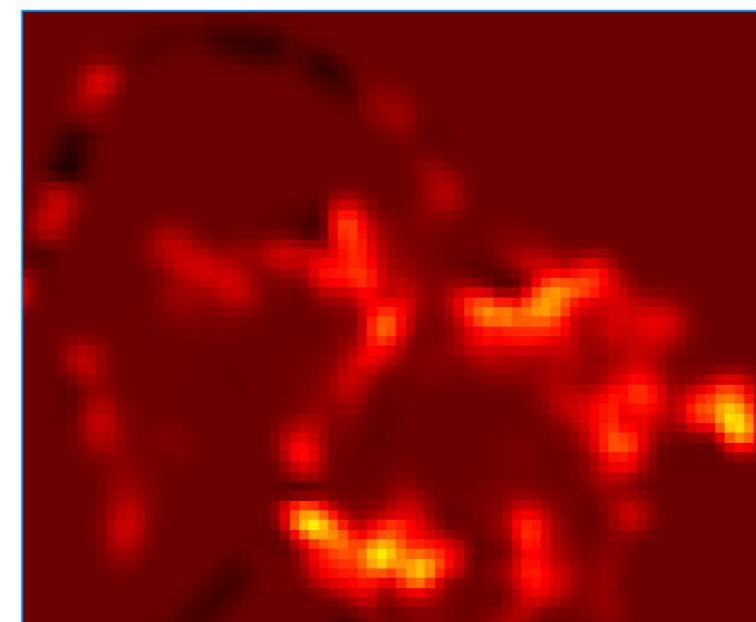
Observe como crece las regiones en la imagen HR



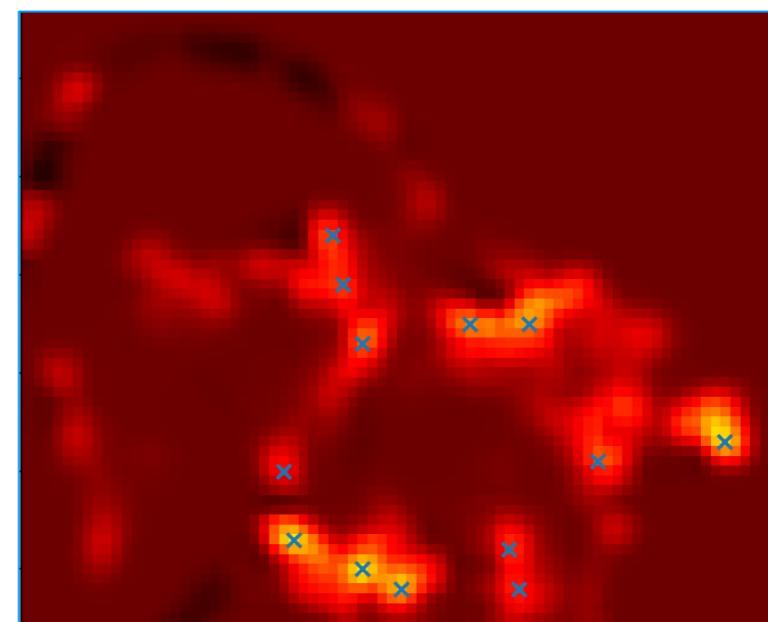
■ Otros descriptores



QUINTO: Buscamos la posición de los máximos locales, y mostramos los resultados



H



xy coordinates

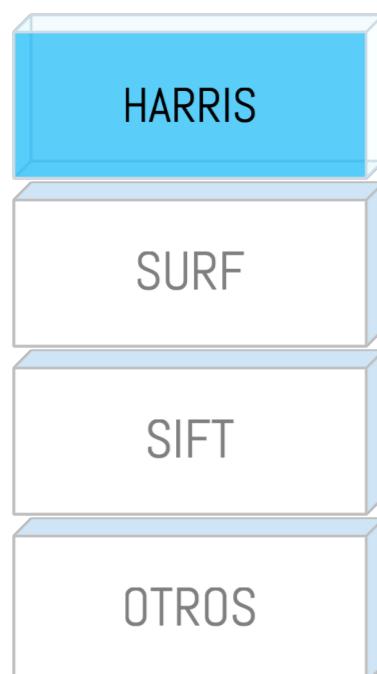
Sólo se cumple la condición cuando es un máximo local

```
# Medición de respuesta de la esquina
k = 0.04

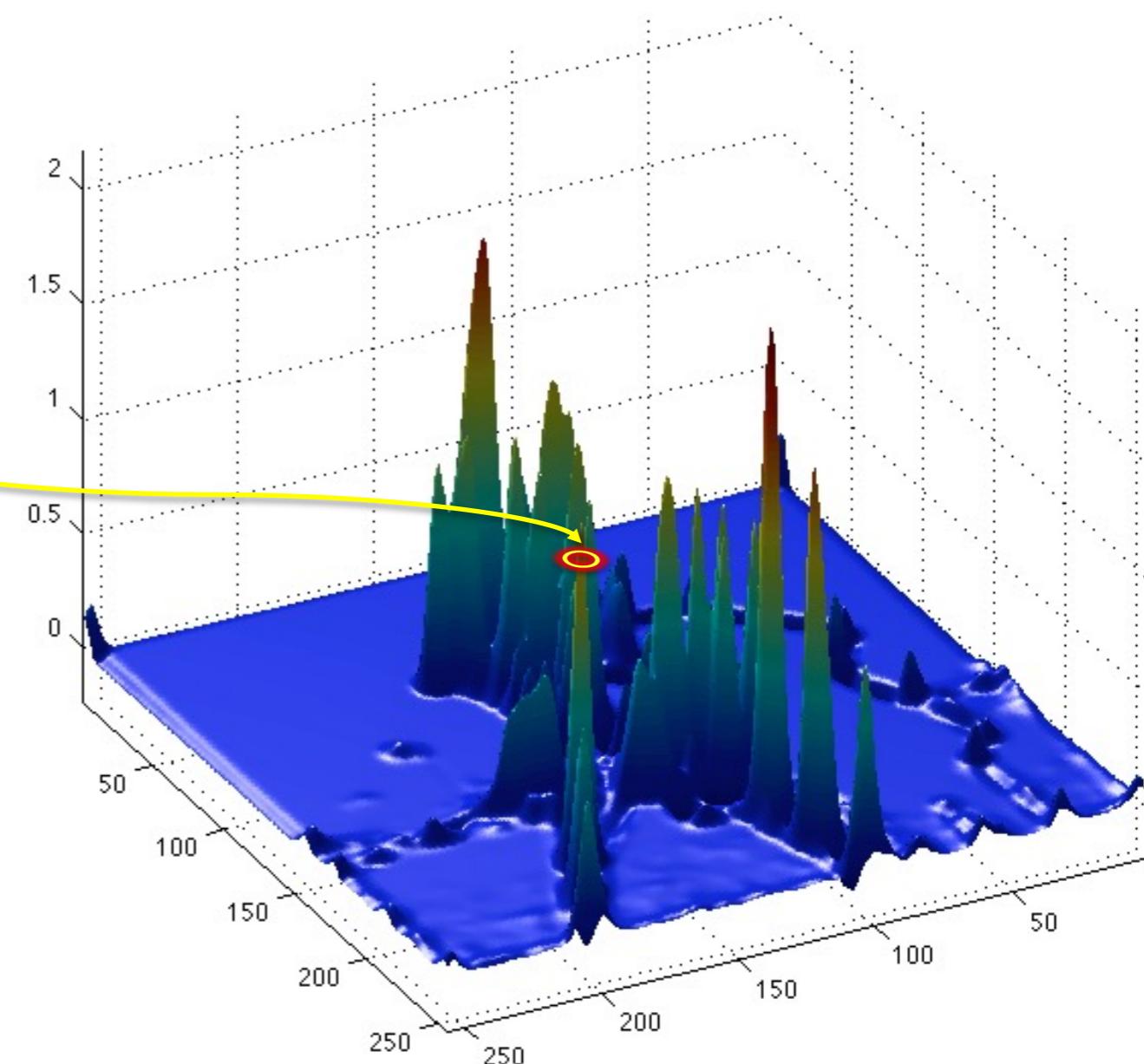
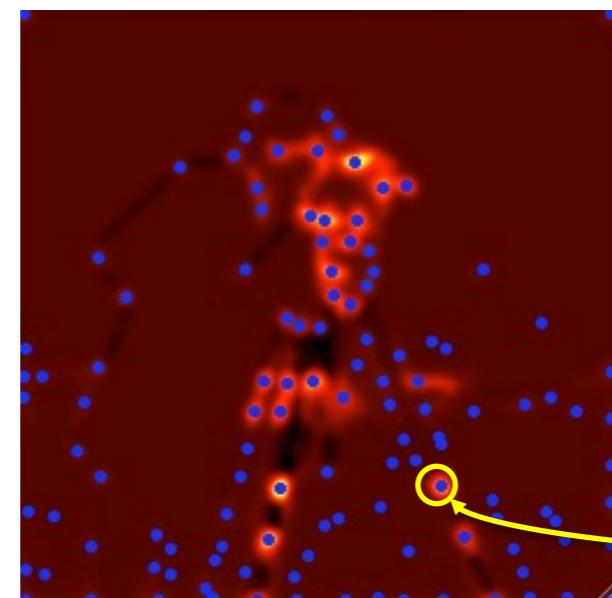
# Parametro de Harris
H = (A*C - B**2) - k*(A + C)**2
xy = peak_local_max(H,
                     min_distance=2,
                     threshold_abs=umbral)
```

En la variable xy quedan almacenadas las coordenadas de las esquinas

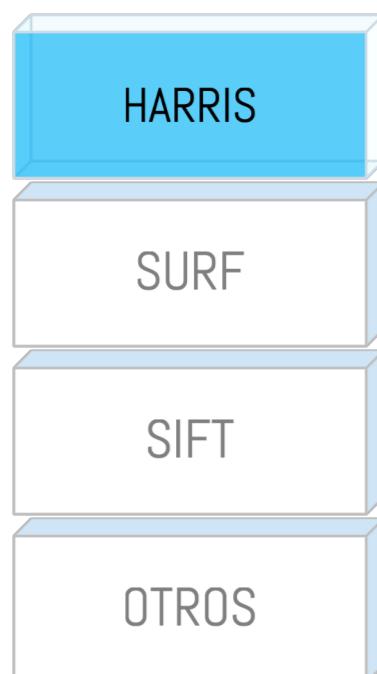
■ Otros descriptores



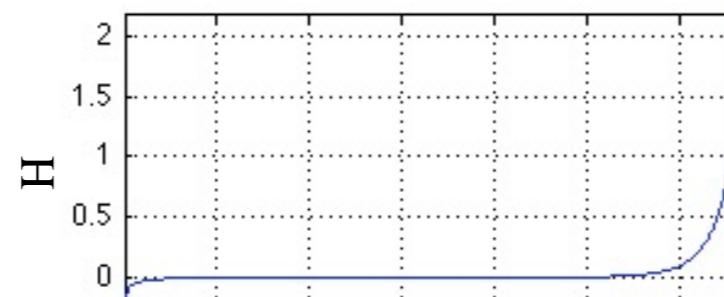
RESULTADO: El detector de esquinas de Harris puede ser afectado por el umbral que definamos.



■ Otros descriptores



RESULTADO: El detector de esquinas de Harris puede ser afectado por el umbral que definamos.



Clasificación de H

- Esquinas:* Umbral máximo positivo
Bordes: Umbral es negativo
Regiones: Umbral es bajo, cercano a cero

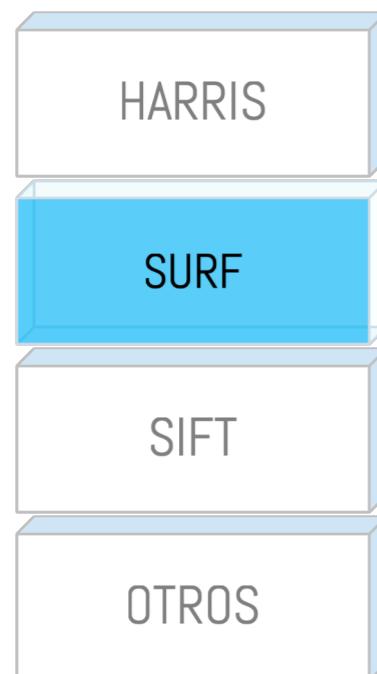


Umbral 0.5



Umbral 0.001

■ Otros descriptores

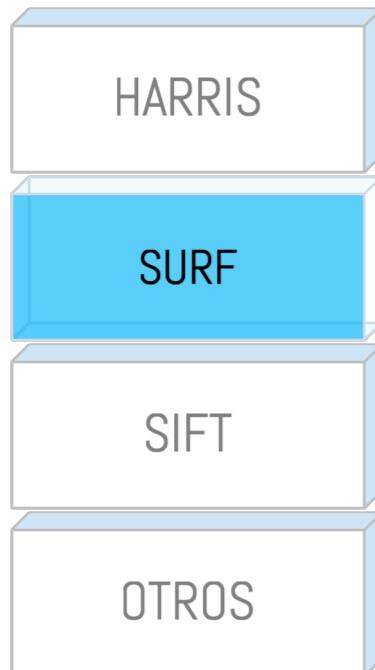


- SURF (Speeded Up Robust Feature) es un descriptor y detector de puntos de interés desarrollado por H. Bay en 2006.
- SURF posee como una de sus principales características la repetibilidad (repeatability), es decir, la capacidad que un punto de interés sea encontrado bajo diferentes transformaciones.



Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346--359, 2008

■ Otros descriptores



- El algoritmo SURF posee varias etapas. En forma general el algoritmo realiza las siguientes seis etapas que veremos en detalle a continuación:

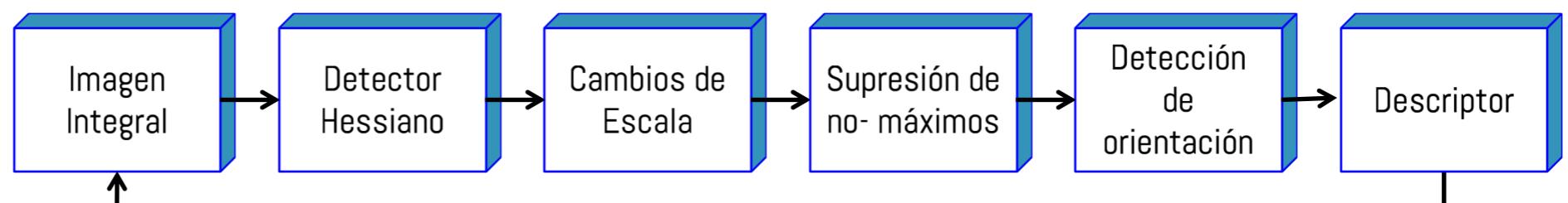
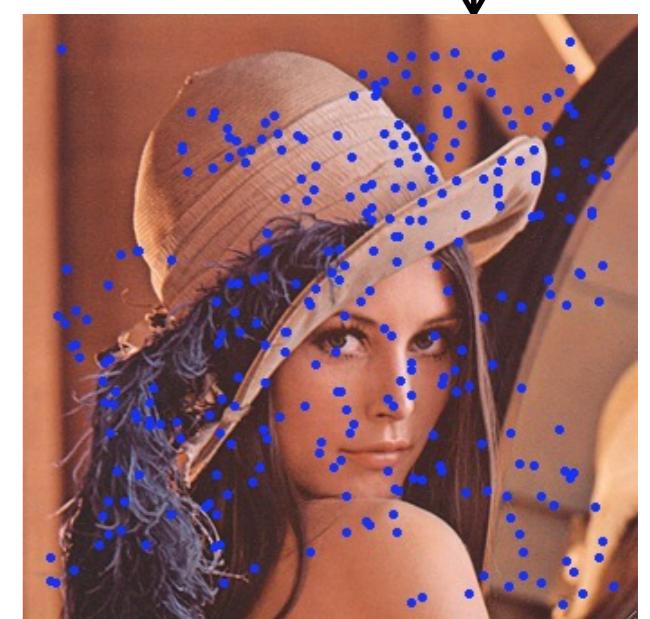
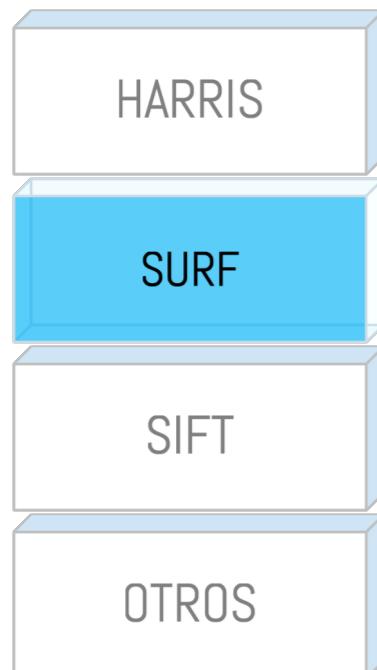


Imagen original

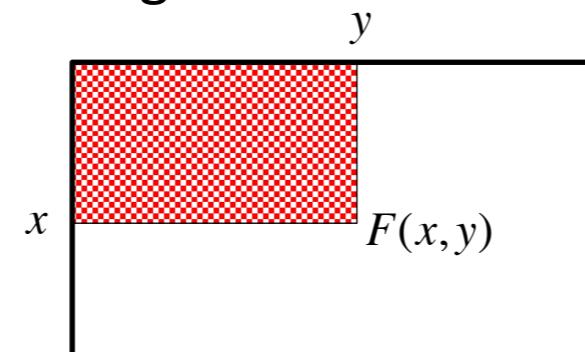


Puntos de interés

■ Otros descriptores



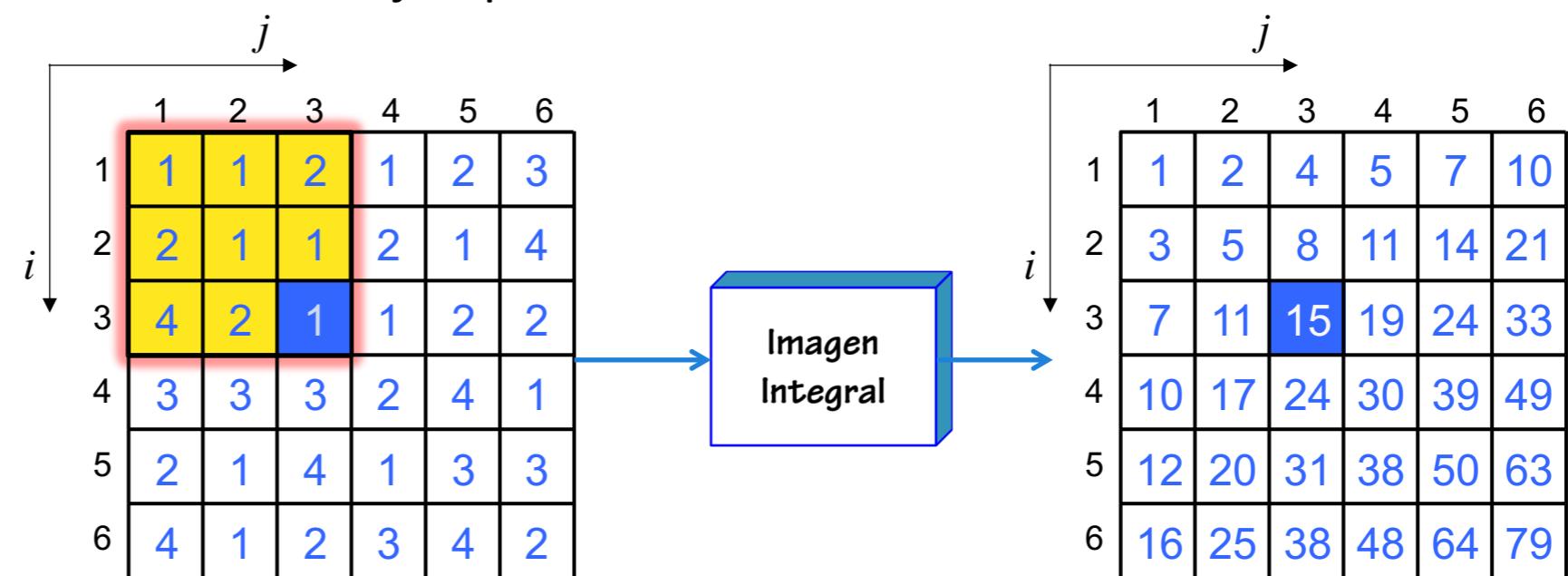
- *Imagen Integral*. Este proceso consiste crear una imagen que corresponde a la suma acumulada en el punto (x,y) de la imagen original



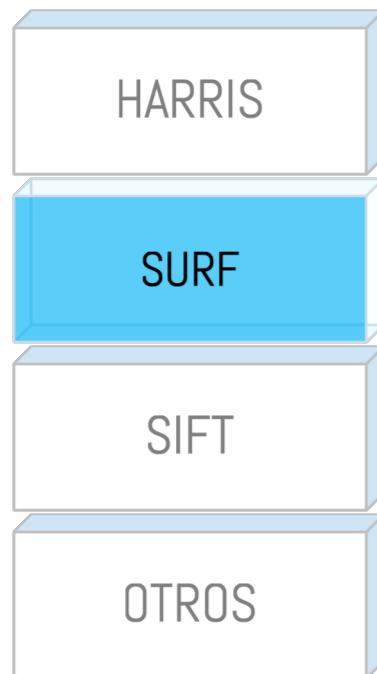
$$F(x,y) = \sum_{k=0}^x \sum_{l=0}^y f(k,l)$$

donde $f(k,l)$
es la imagen
original en la
posición k, l

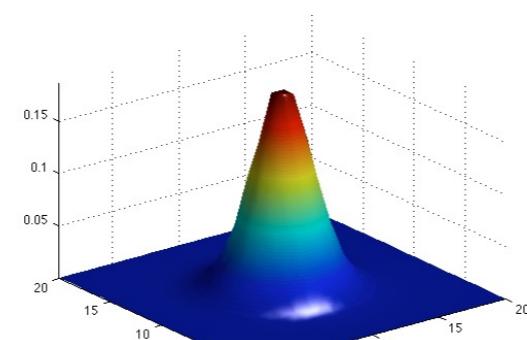
- Veamos un ejemplo



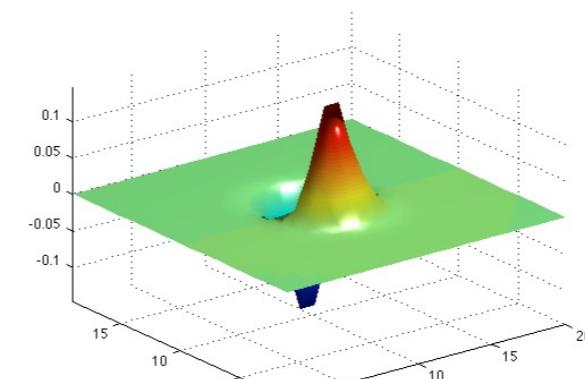
■ Otros descriptores



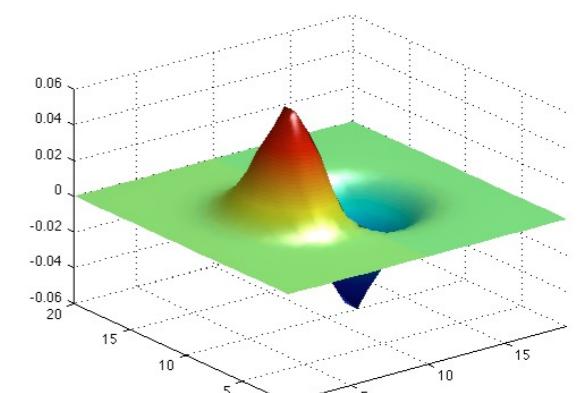
- **Detector Hessiano** : El detector Hessiano funciona de manera “similar” al detector de Harris. Consiste básicamente en convolucionar la imagen con la segunda derivada de una gaussiana.



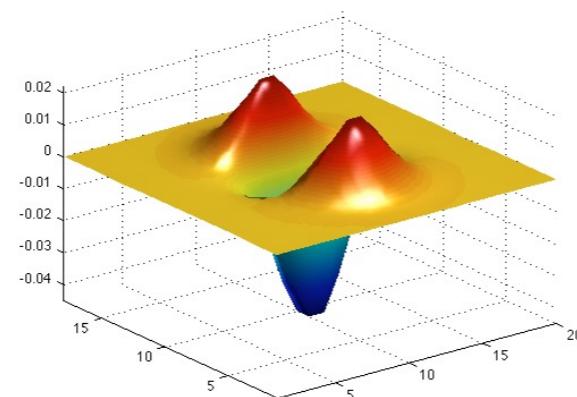
$$g = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot \exp\left(-\frac{x^2 + y^2}{2\pi\sigma^2}\right)$$



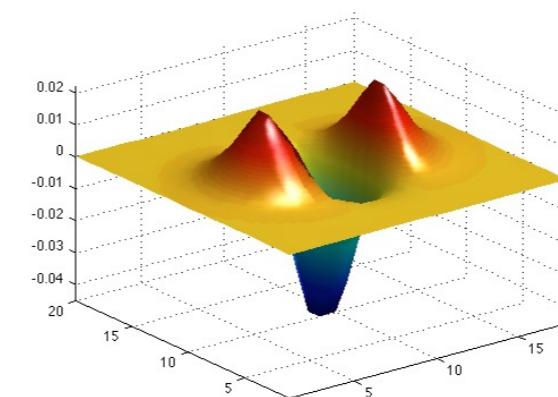
`Gx=diff(g,1,1)`



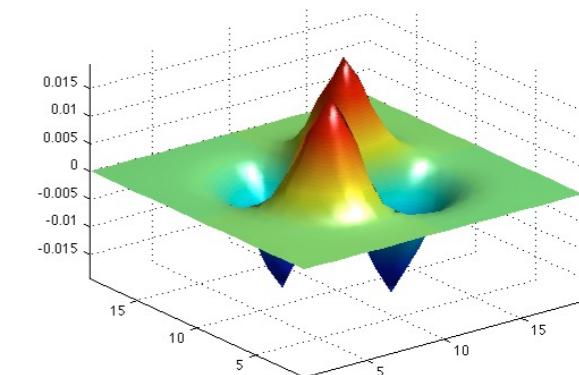
`Gy=diff(g,1,2)`



`Gxx=diff(Gx,1,1)`

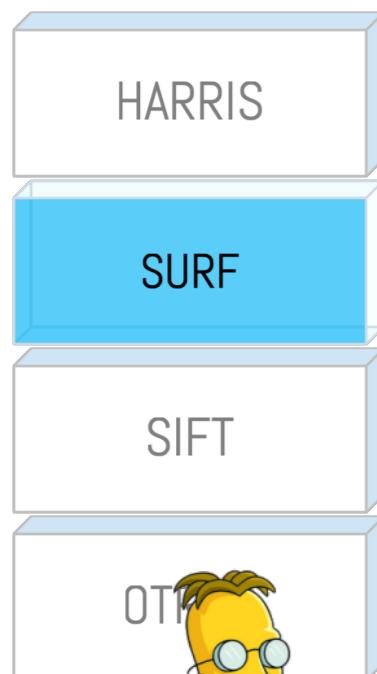


`Gyy=diff(Gy,1,2)`



`Gxy=diff(Gx,1,2)`

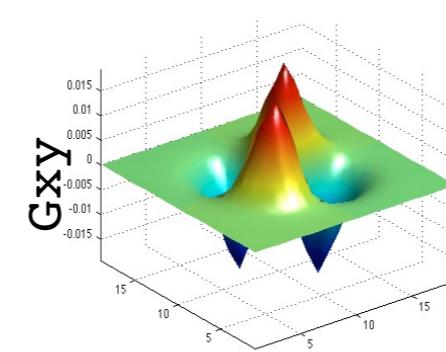
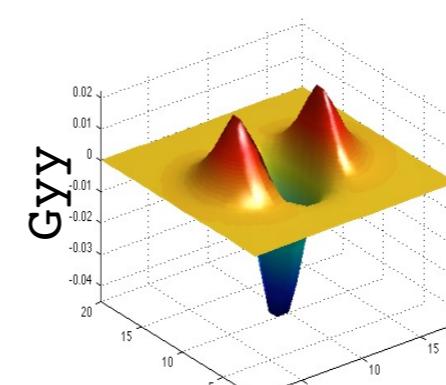
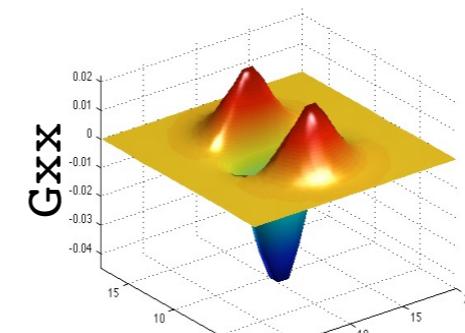
■ Otros descriptores



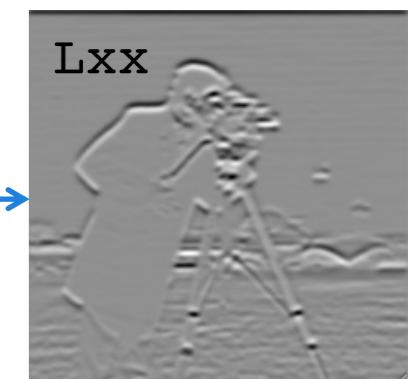
- *Detector Hessiano* : Una vez determinadas las segundas derivadas direccionales, debemos convolucionar con la imagen original



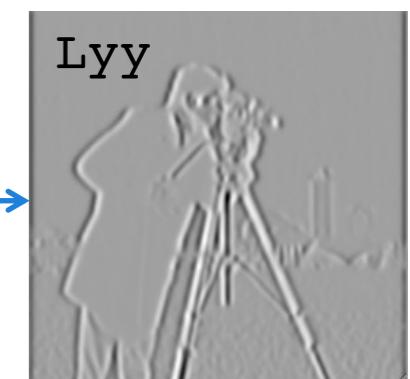
Imagen original



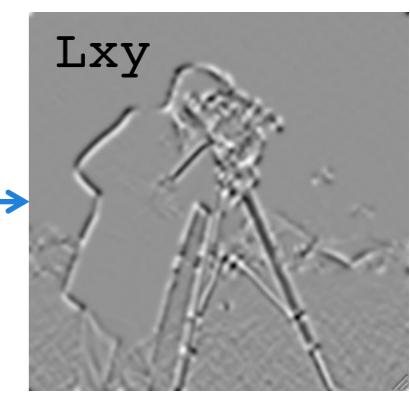
Convolución



Convolución



Convolución

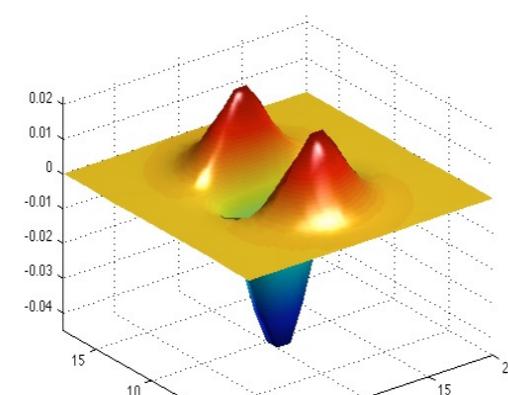


```
# derivadas direccionales
Lxx = convolve2d(im, Gxx, 'same')
Lyy = convolve2d(im, Gyy, 'same')
Lxy = convolve2d(im, Gxy, 'same')
```

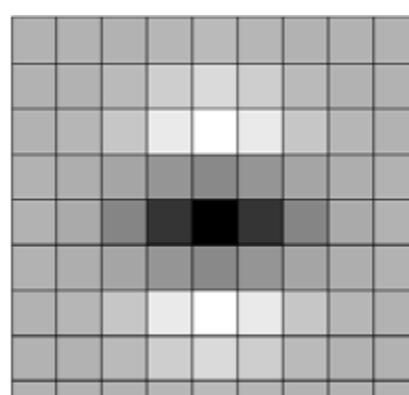
■ Otros descriptores



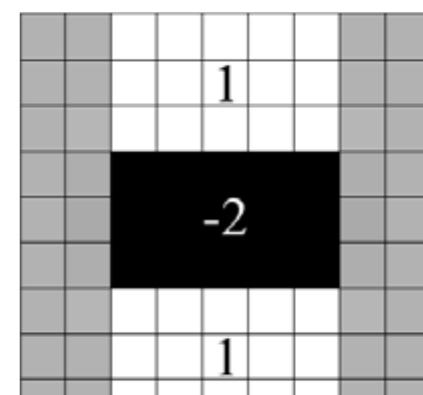
- *Detector Hessiano* : Debido a que es costoso calcular estas matrices, los desarrolladores de SURF hicieron la siguiente simplificación, denominada *box-filter*



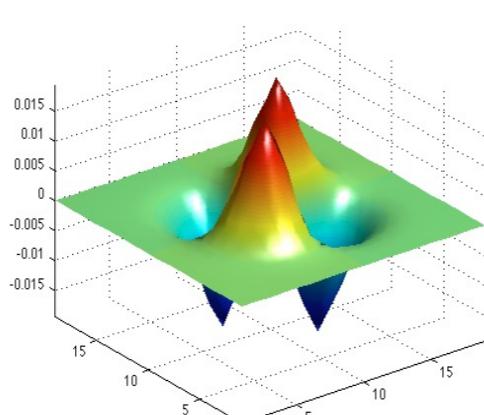
Vista 3D



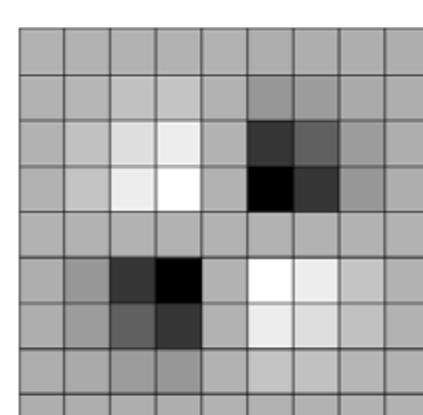
Vista 2D G_{xx}



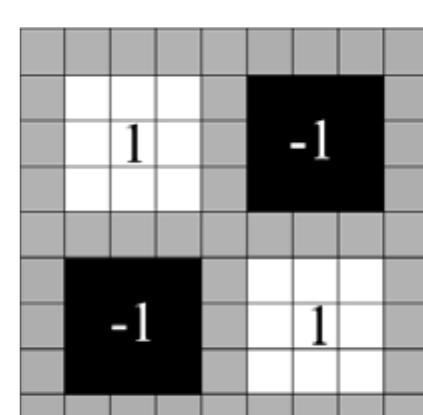
Simplificación G_{xx}



Vista 3D

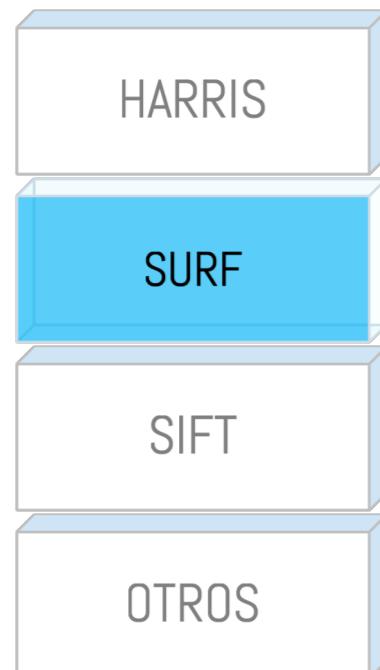


Vista 2D G_{xy}



Simplificación G_{xy}

■ Otros descriptores



- *Detector Hessiano* : Empleando los box-filter convolucionamos con la imagen original obtenemos los siguientes resultados parciales



$$D_{xx}$$

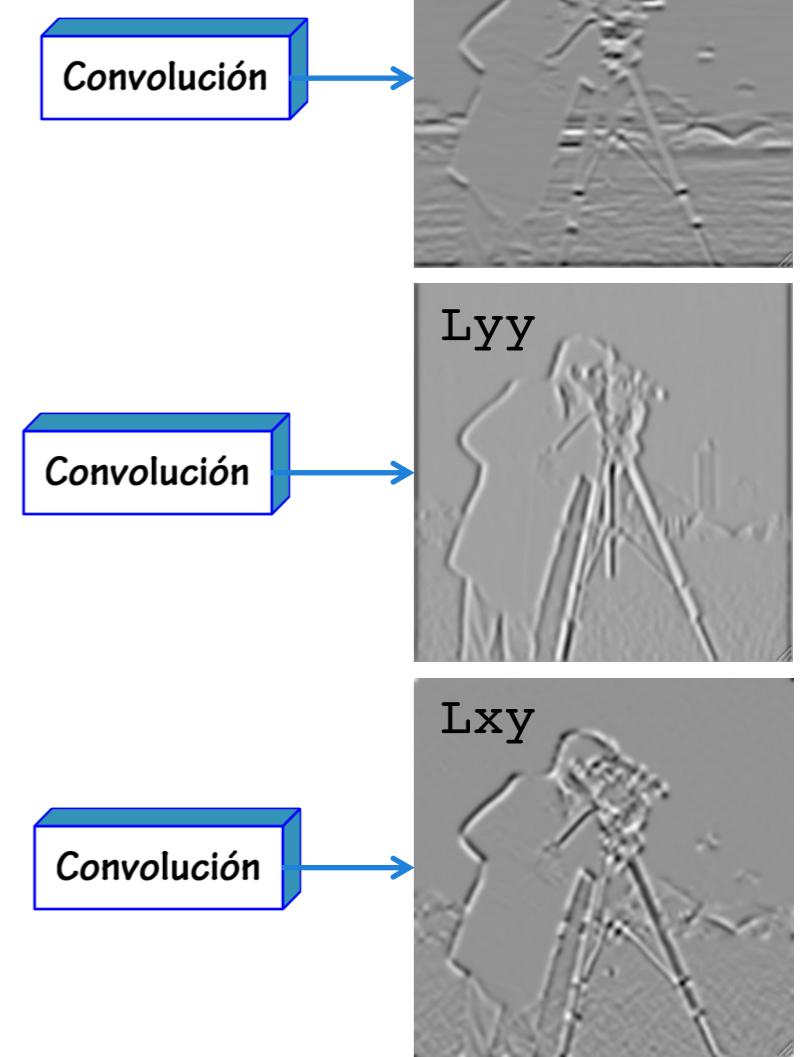
1		
	-2	
1		

$$D_{yy}$$

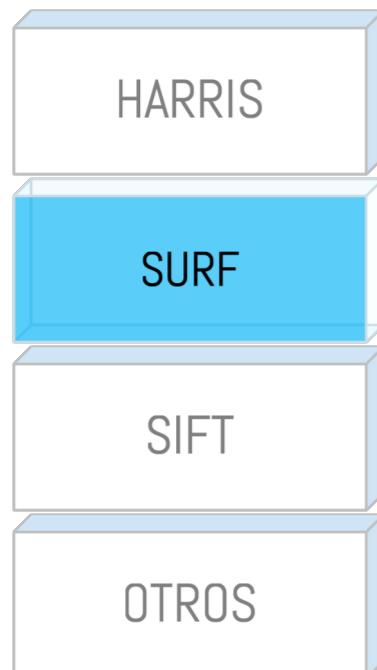
1		
	-2	
1		

$$D_{xy}$$

1		
	-1	
-1		1



■ Otros descriptores



- *Detector Hessiano* : Empleando los resultados previos, definimos la matriz Hessiana como:

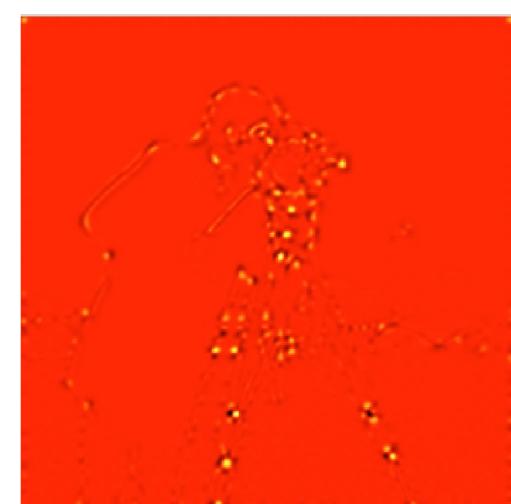
$$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}$$

IMPORTANTE:
Todos los valores anteriores
dependen de sigma

- Finalmente la respuesta del Hessiano aproximado es:

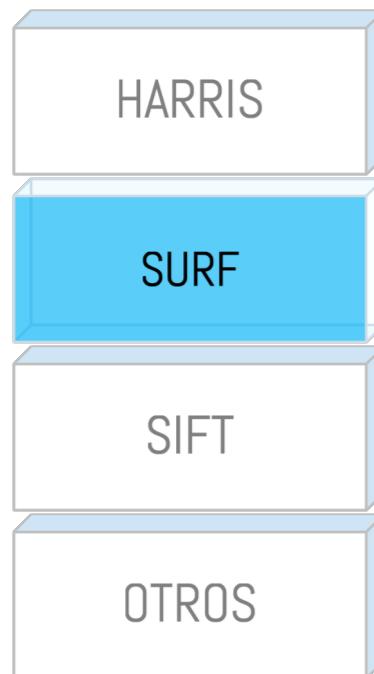
$$\det(H_{aprox}) = L_{xx} \cdot L_{yy} - (0.9 \cdot L_{xy})$$

$$Laplaciano = L_{xx} + L_{yy} \geq 0$$

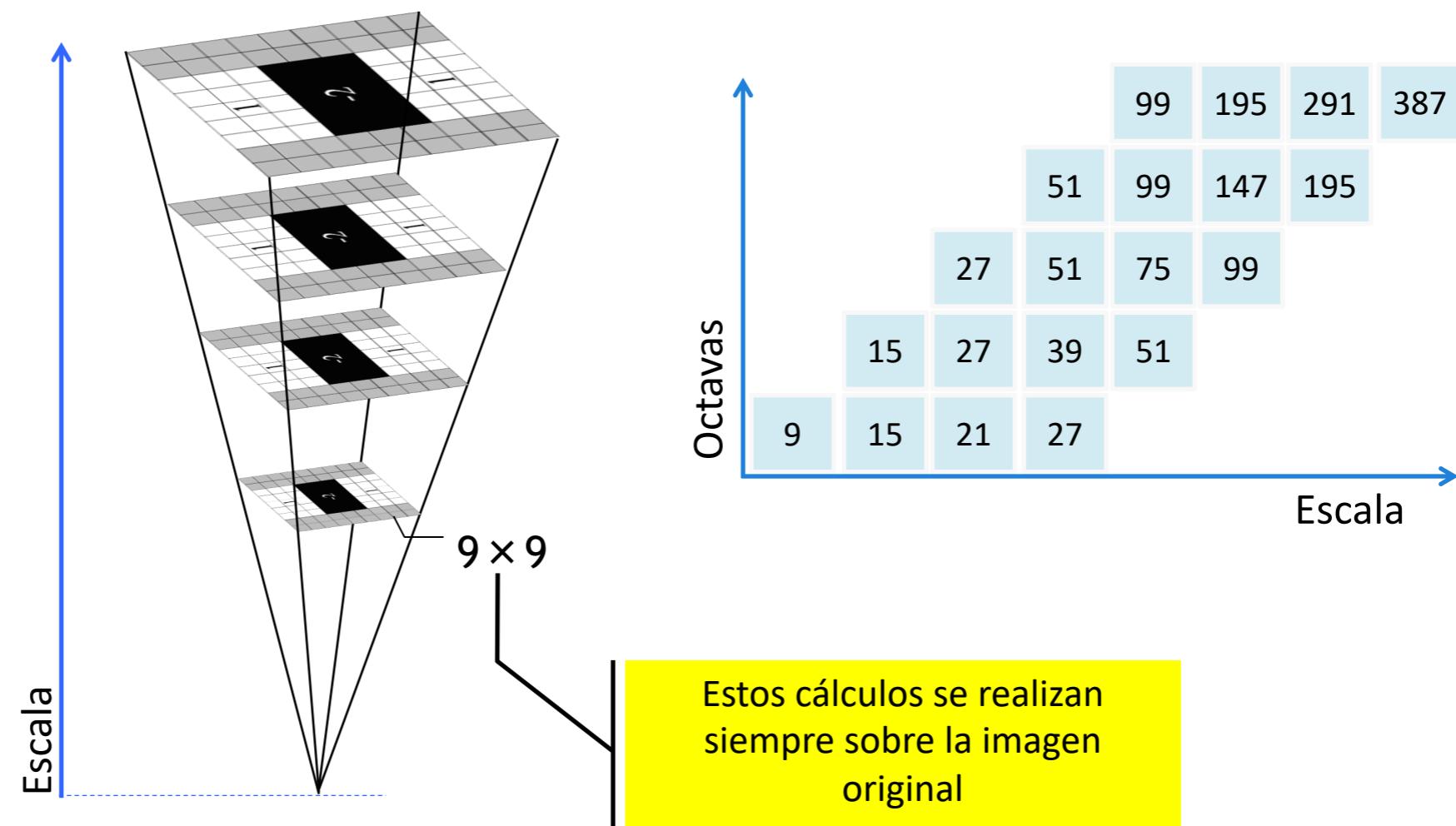


$$\det(H_{aprox})$$

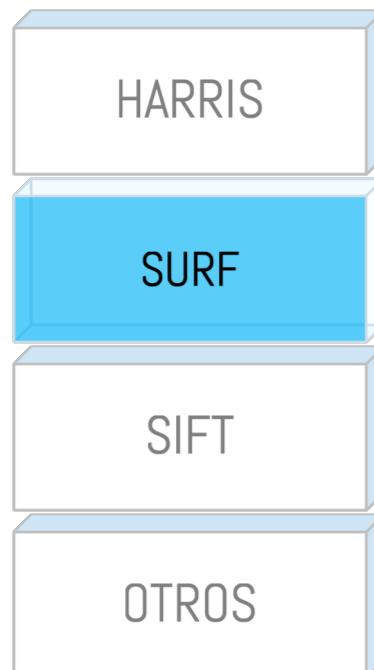
■ Otros descriptores



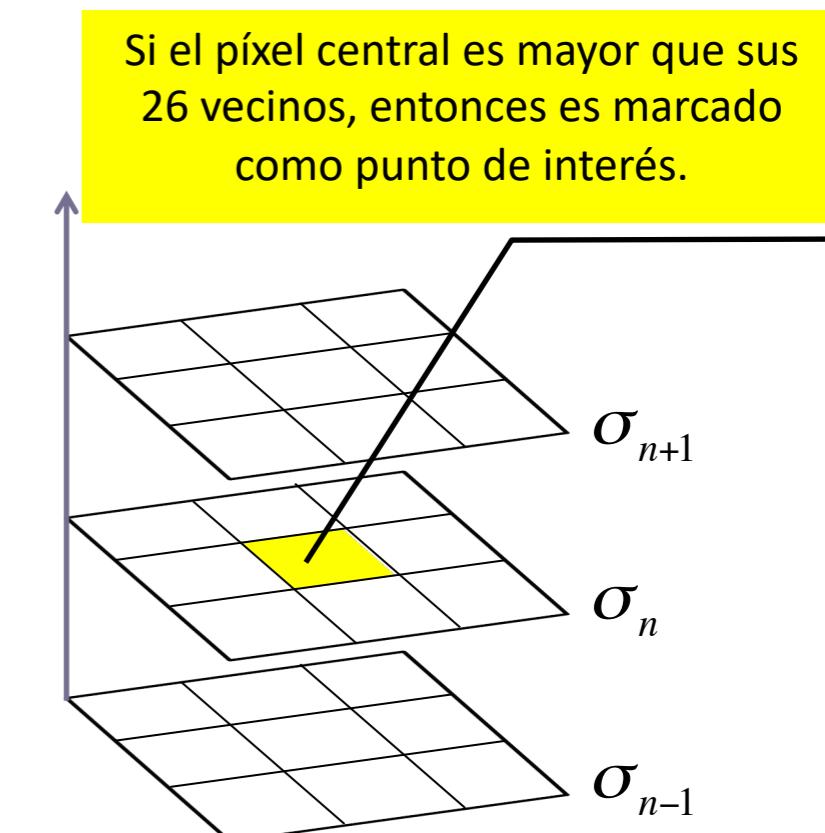
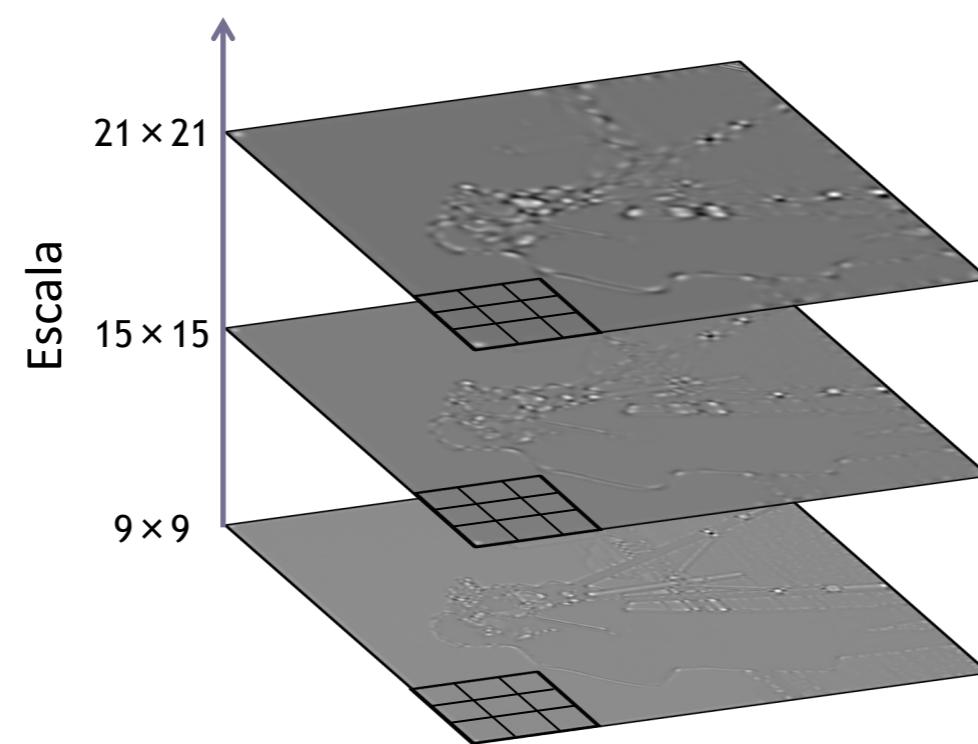
- **Cambios de escala:** El siguiente paso consiste en generar múltiples escalas del filtro (no de la imagen original). En la primera octava el crecimiento es de seis en seis, luego de 12 en 12, luego de 24 en 24 y así sucesivamente hasta completar el tamaño de la imagen.



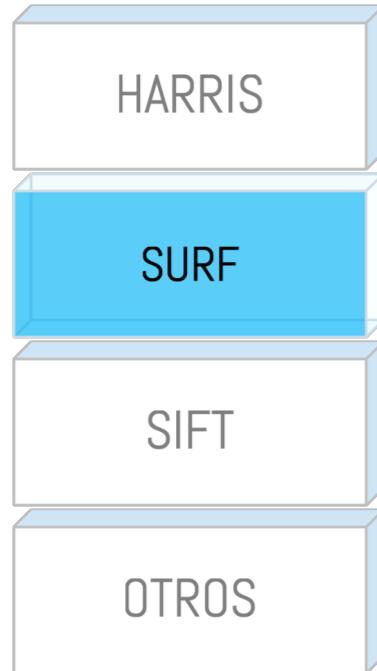
■ Otros descriptores



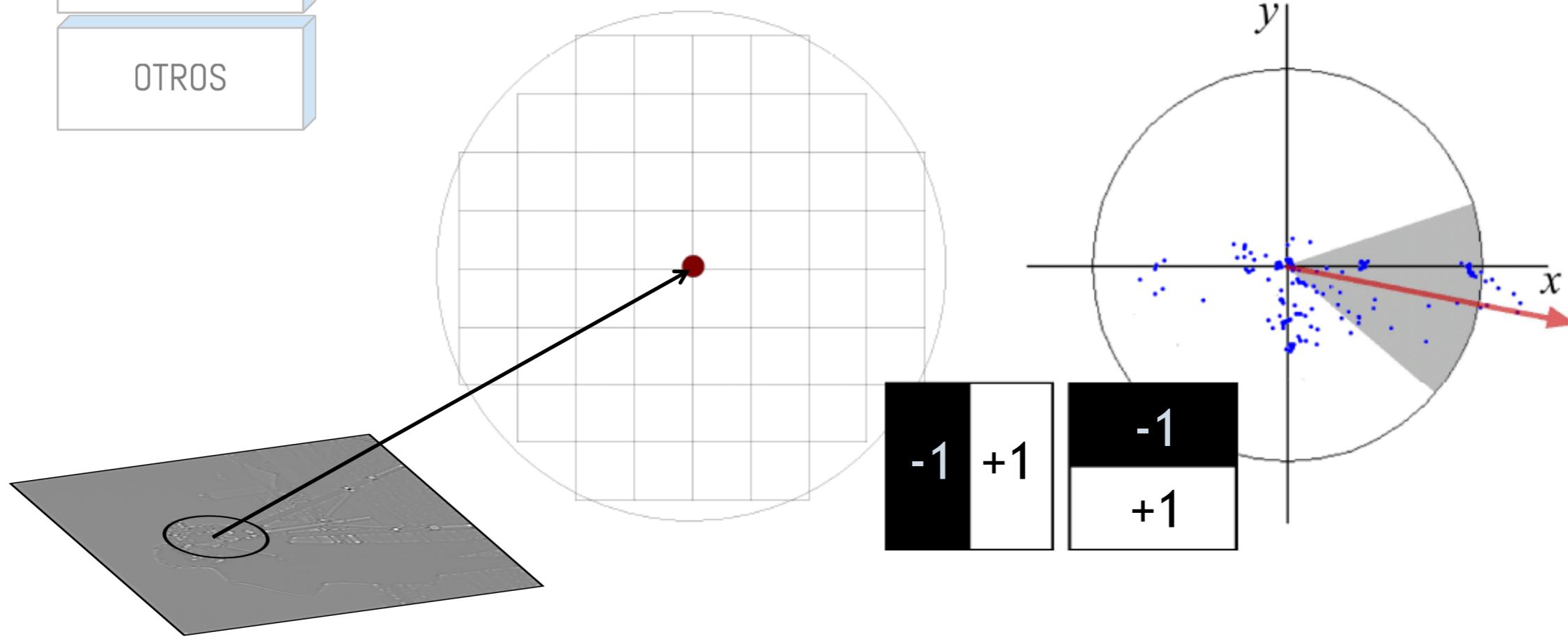
- *Supresión de no-maximos*: Este proceso busca conservar las respuestas del Hessiano que sean máximas en un bloque de $3 \times 3 \times 3$. Este proceso se realiza empleando tres escalas a la vez.
- Para mejorar el resultado del Hessiano, el máximo es interpolado en escala y posición



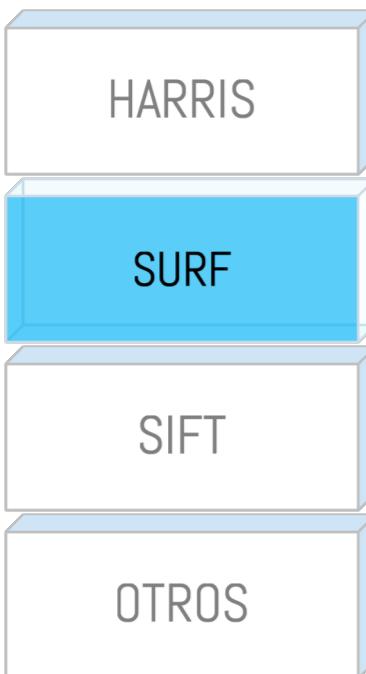
■ Otros descriptores



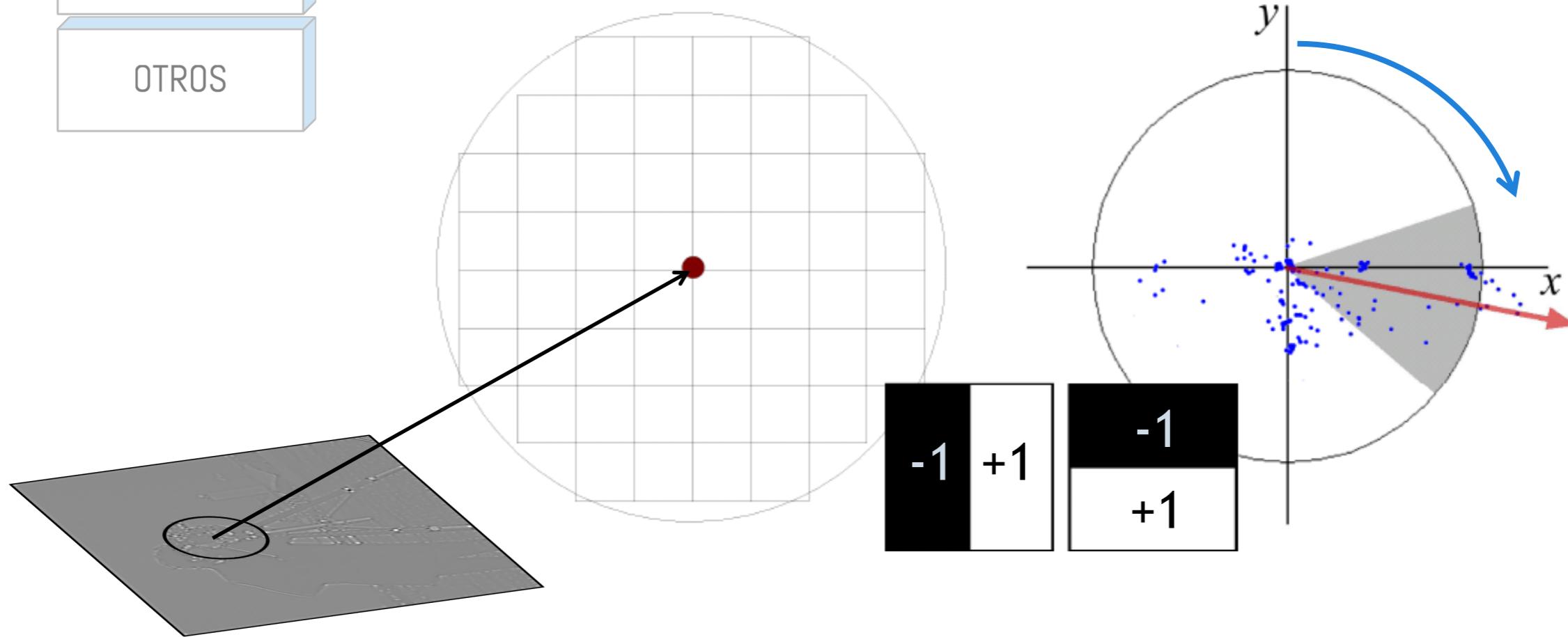
- *Detección de orientación*: Una vez encontrado los puntos de interés, tomamos un círculo de radio $6s$ alrededor del punto de interés (donde s es la escala a la cual fue detectado).



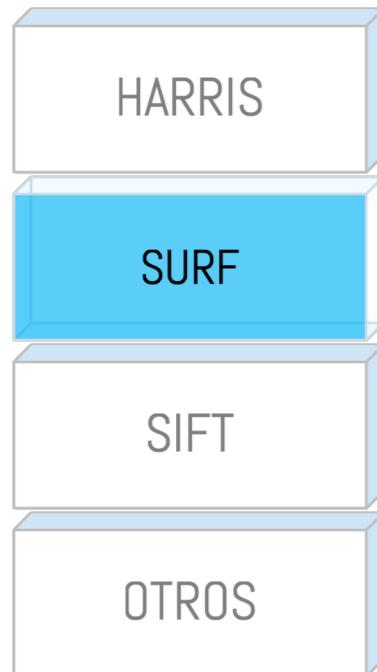
■ Otros descriptores



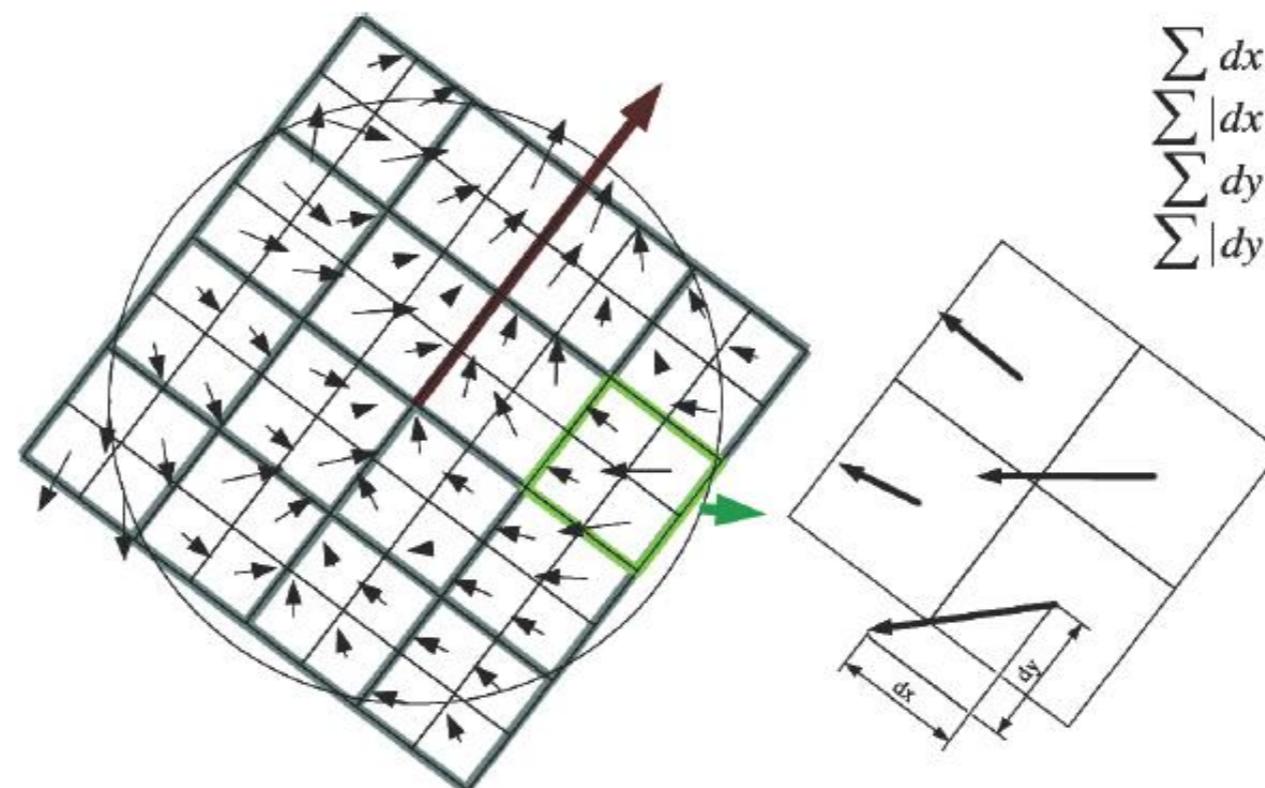
- *Detección de orientación*: Luego sumamos los valores Hessianos según un ángulo de orientación de 60 grados alrededor del punto. La suma de dos respuestas genera un nuevo vector. **El vector que sea más largo genera la orientación de dicho punto de interés**



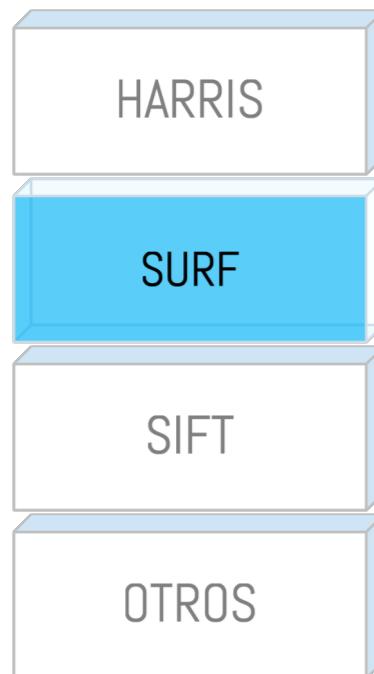
■ Otros descriptores



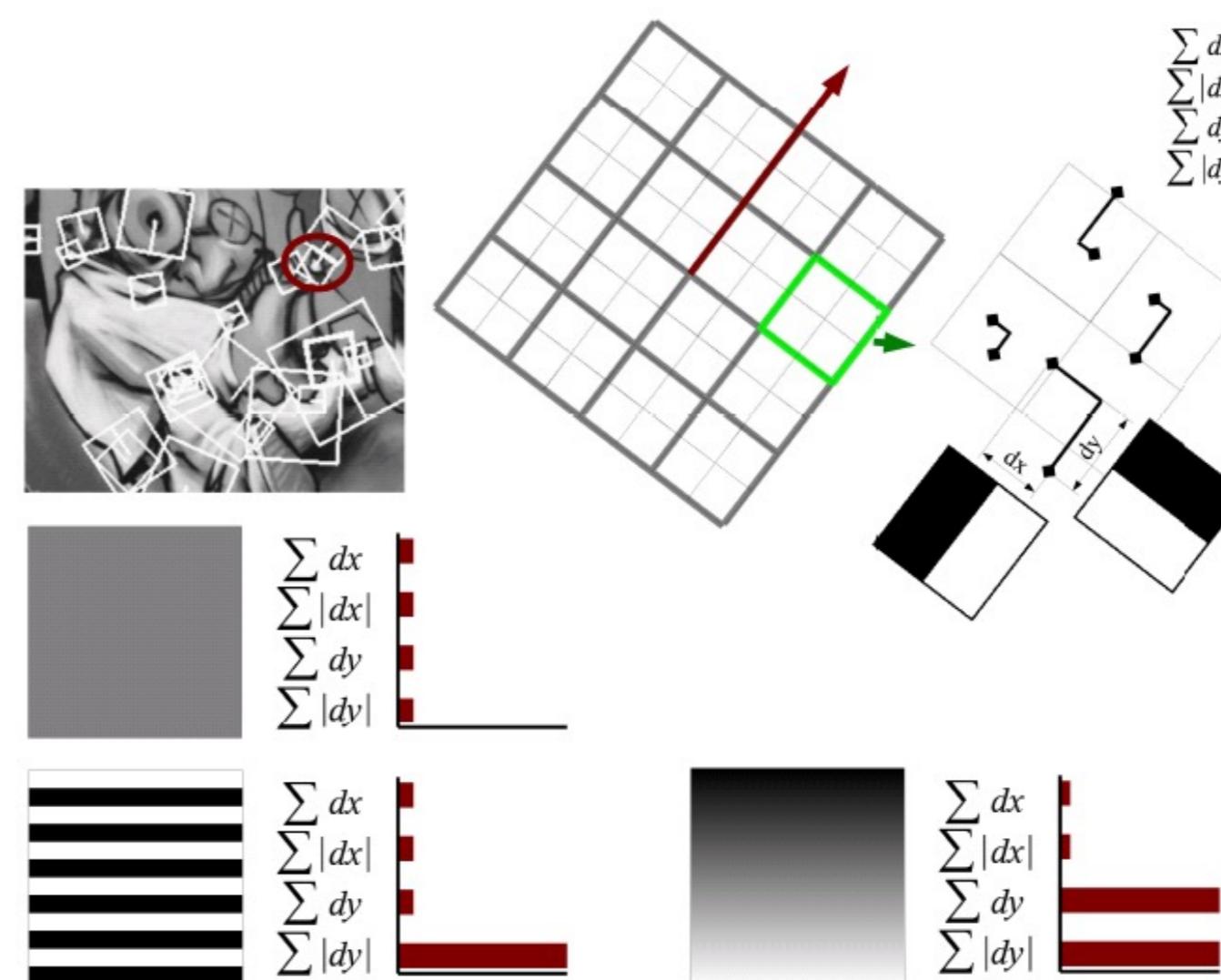
- **Descriptor:** Una vez conocida la orientación del descriptor, el último paso consiste en la extracción del descriptor. Esta consiste en dividir en subregiones de 4×4 (centrada en el punto de interés). Dentro de cada subregión debe haber 5×5 píxeles.



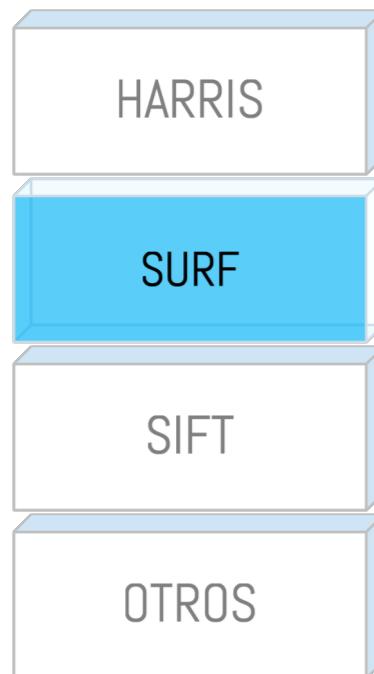
■ Otros descriptores



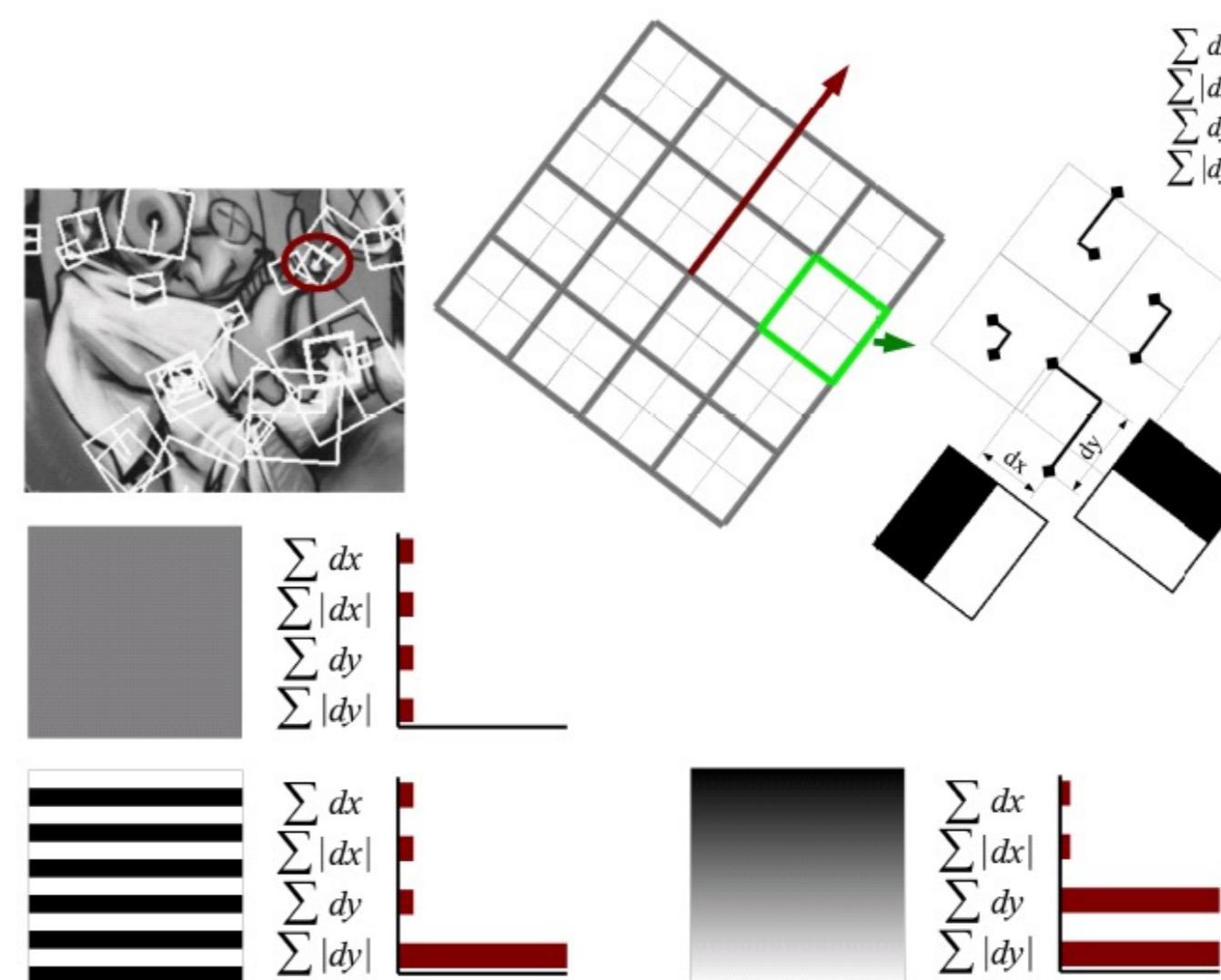
- **Descriptor:** Por cada subregión calculamos la suma horizontal y vertical. Para reducir el error, cada subregión es filtrada con un filtro Gaussiano.



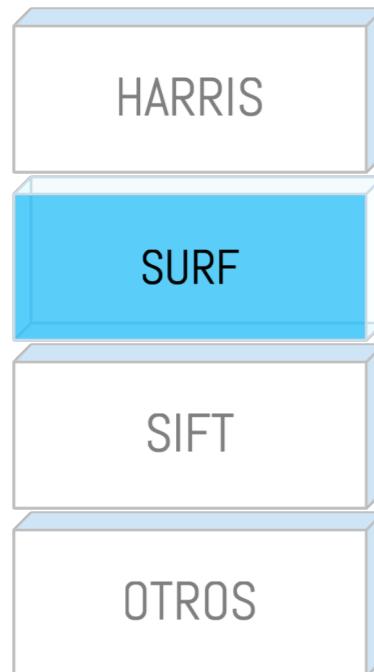
■ Otros descriptores



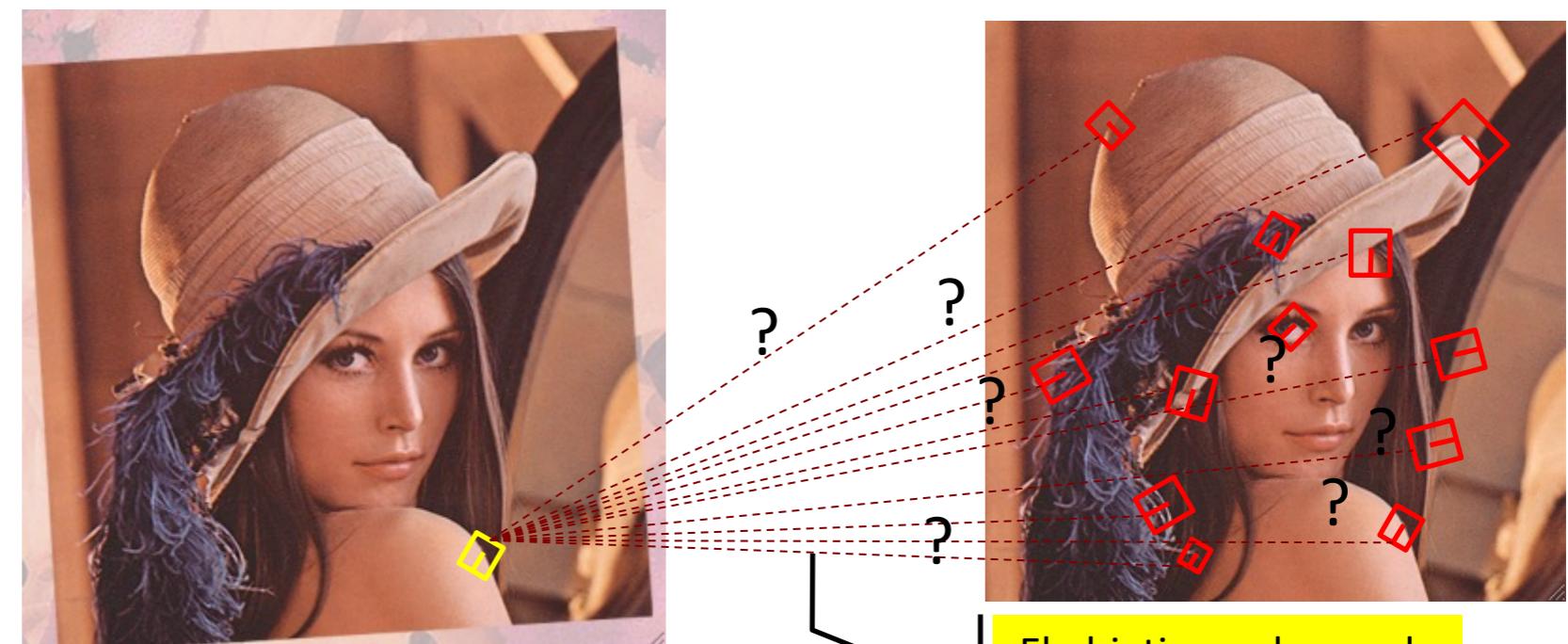
- **Descriptor:** Cada subregión genera 4 valores. Dado que tenemos 16 subregiones, nuestro descriptor finalmente queda de largo 64.



■ Otros descriptores

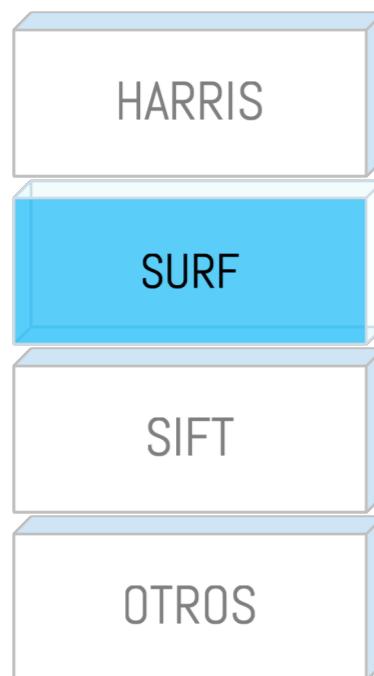


- Una vez generado el vector de características, es necesario buscar su correspondiente en otra imagen. Este proceso se denomina **Matching**.
- Un algoritmo muy empleado se conoce como NNDR (Nearest Neighbor Distance Ratio)



El objetivo es buscar la correspondencia en la otra imagen

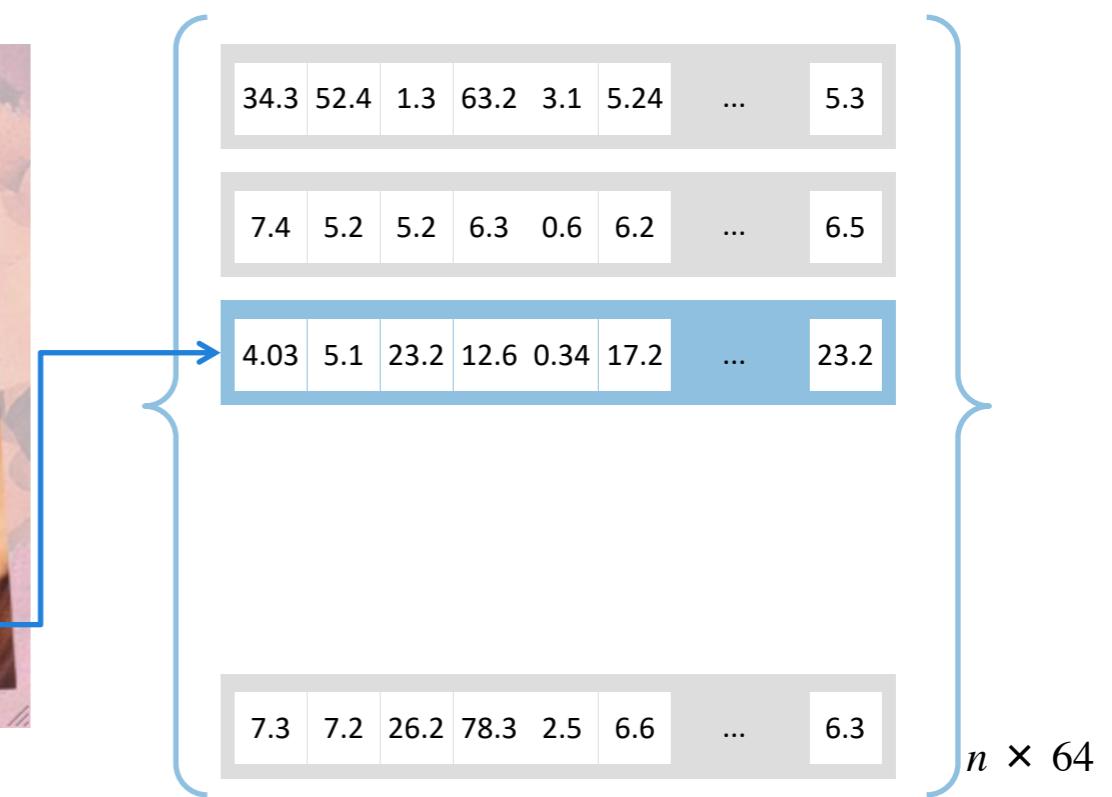
■ Otros descriptores



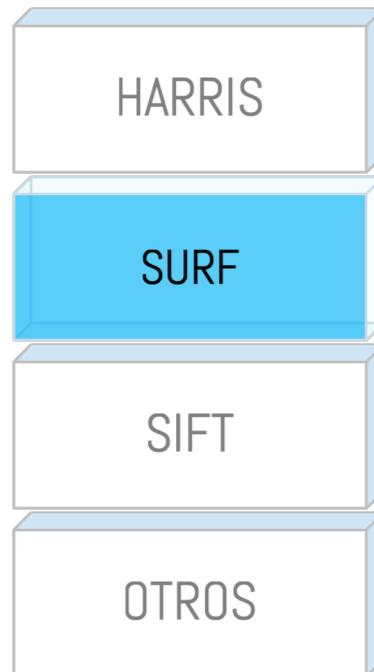
- El proceso de Matching NNDR es el siguiente:
 - Para cada descriptor de la imagen izquierda
 - a. Seleccione un vector de características de la imagen izquierda



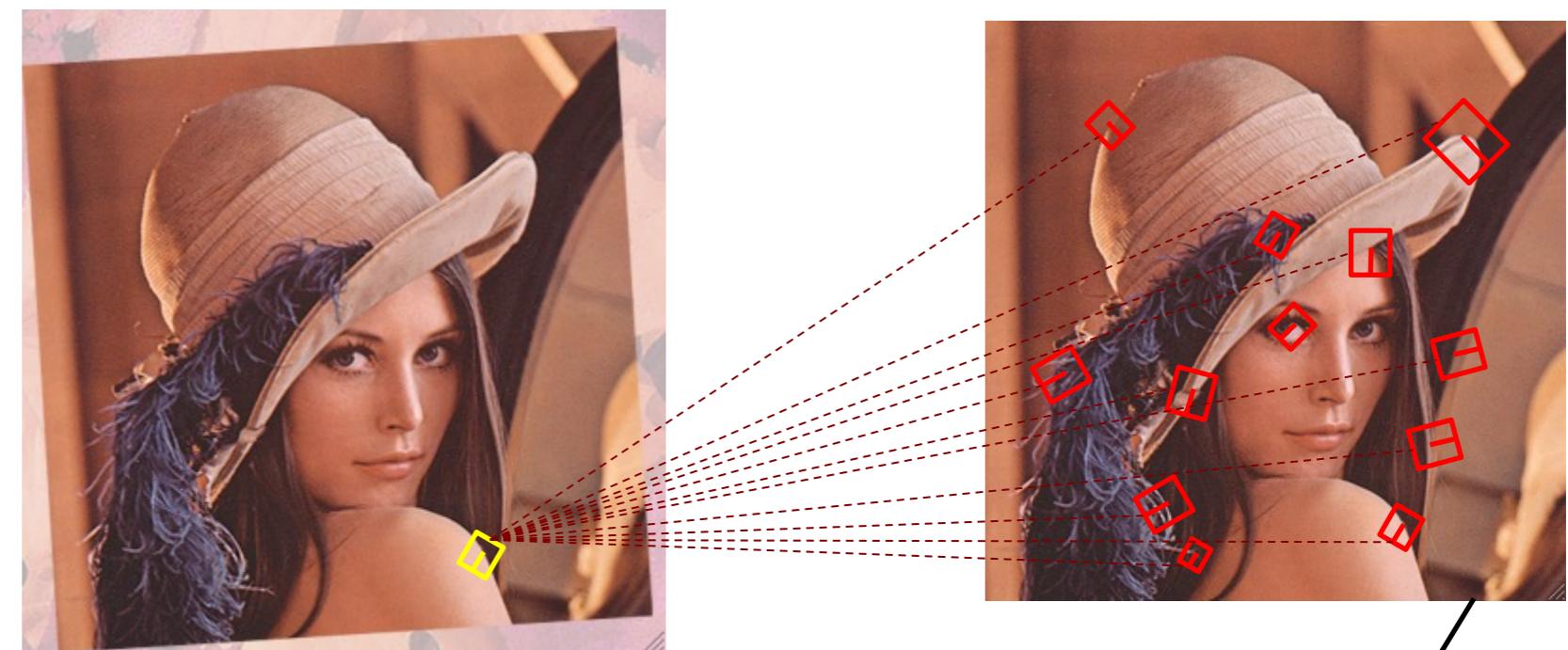
Suponga que este es un descriptor de la imagen izquierda y deseamos buscar su correspondiente



■ Otros descriptores

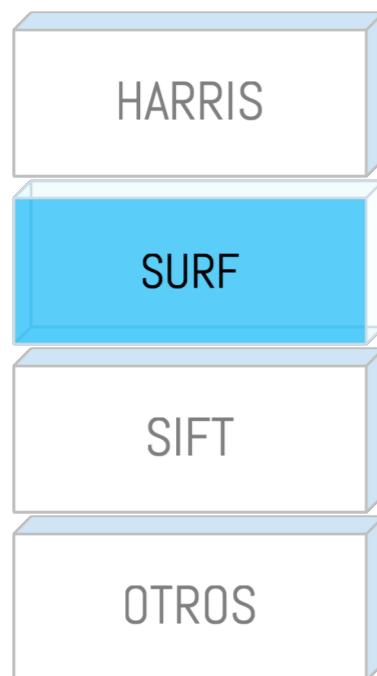


- El proceso de Matching NNDR es el siguiente:
 - Para cada descriptor de la imagen izquierda
 - a. Seleccione un vector de características de la imagen izquierda
 - b. Determine la distancia {Euclíadiana} respecto a todos los descriptores en la imagen derecha

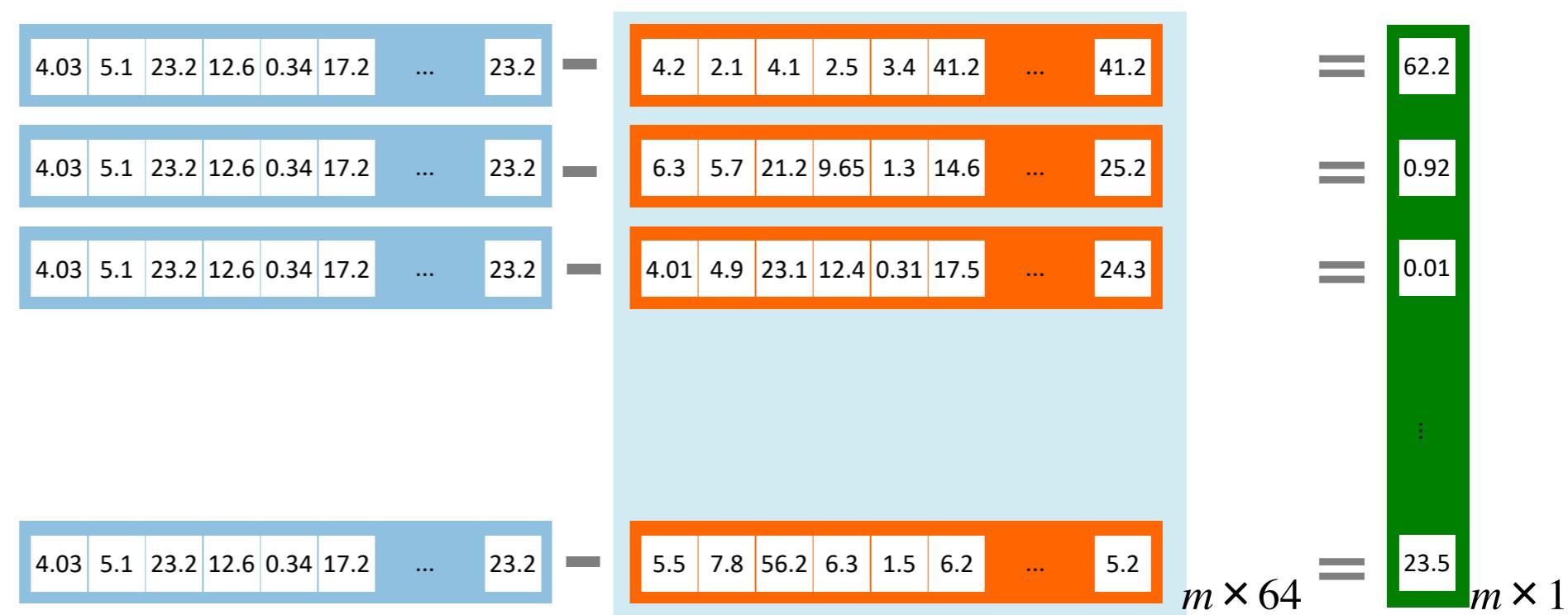


Suponga que en esta imagen se detectaron m descriptores

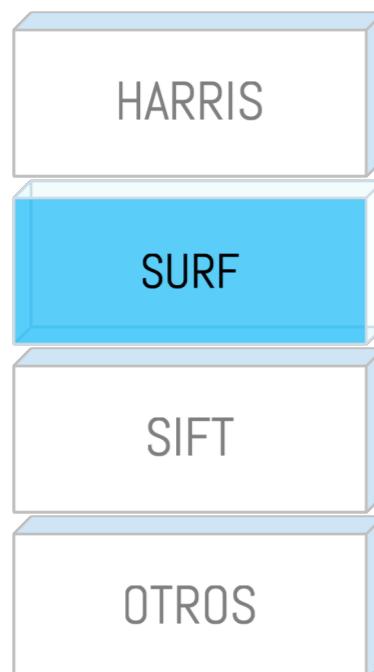
■ Otros descriptores



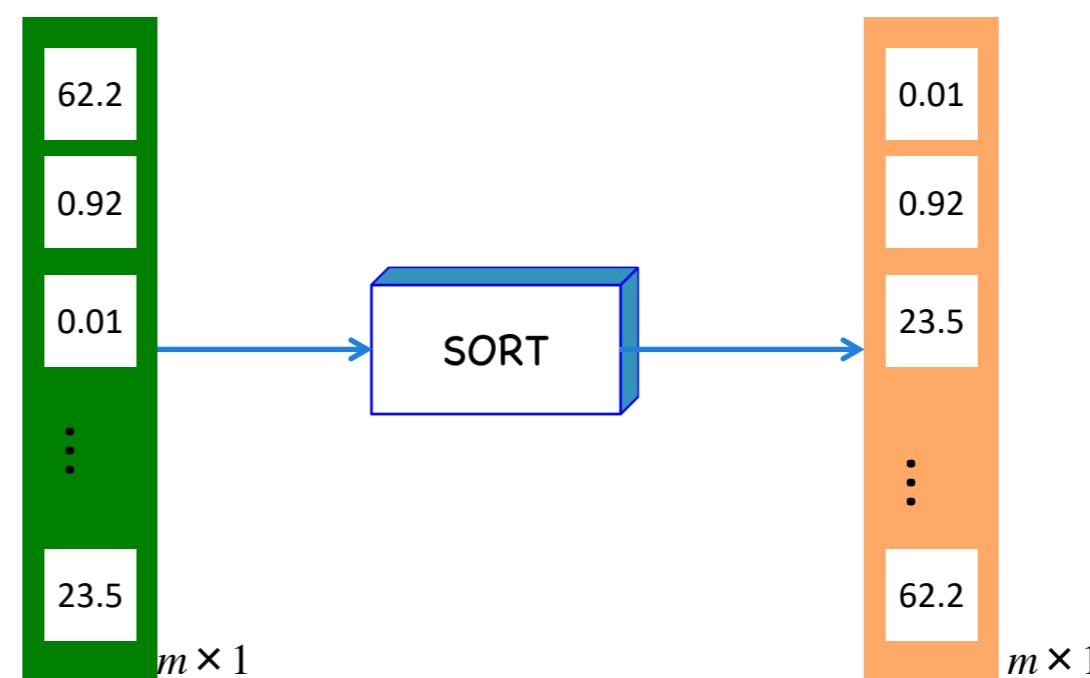
- El proceso de Matching NNDR es el siguiente:
 - Para cada descriptor de la imagen izquierda
 - a. Seleccione un vector de características de la imagen izquierda
 - b. Determine la distancia {Euclíadiana} respecto a todos los descriptores en la imagen derecha



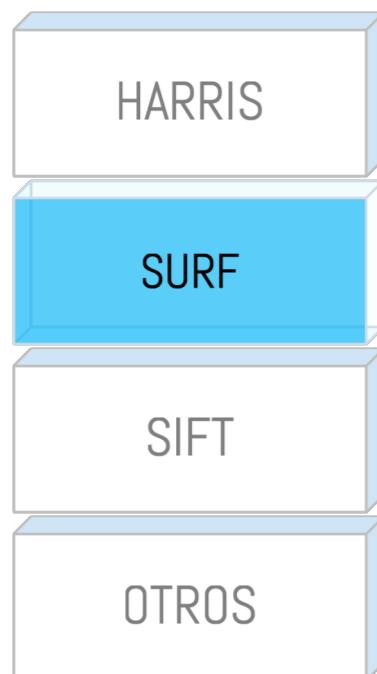
■ Otros descriptores



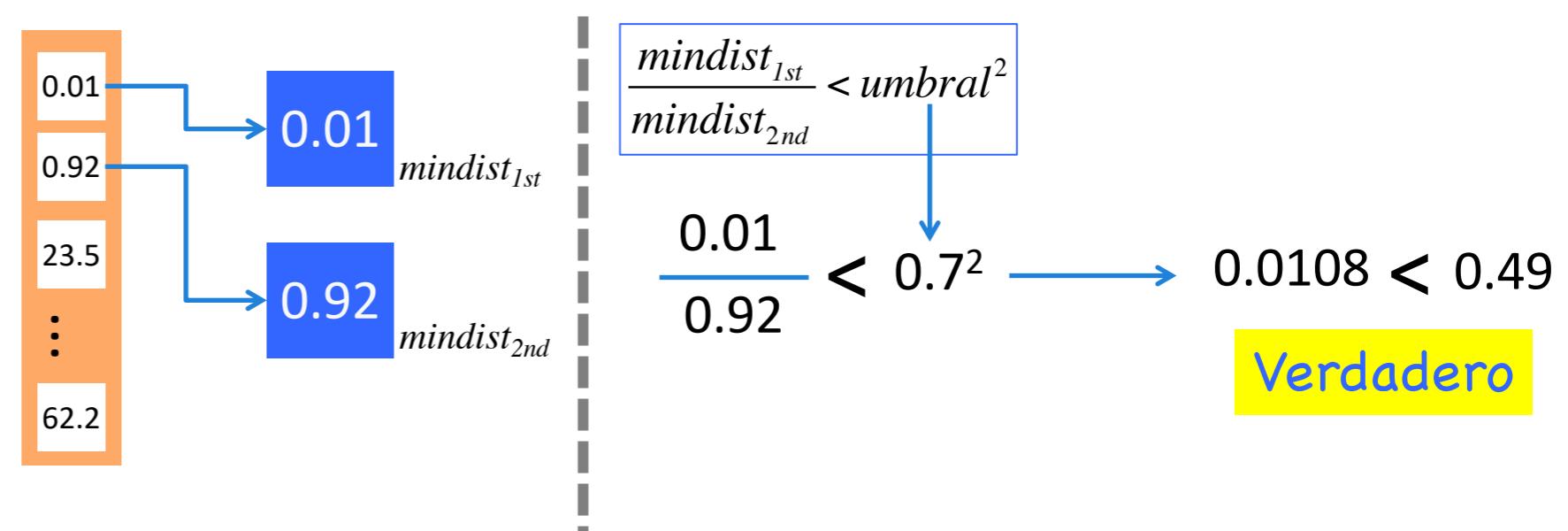
- El proceso de Matching NNDR es el siguiente:
 - Para cada descriptor de la imagen izquierda
 - a. Seleccione un vector de características de la imagen izquierda
 - b. Determine la distancia {Euclíadiana} respecto a todos los descriptores en la imagen derecha
 - c. Ordene la lista de distancias en orden ascendente. Recuerde almacenar el orden de los índices.



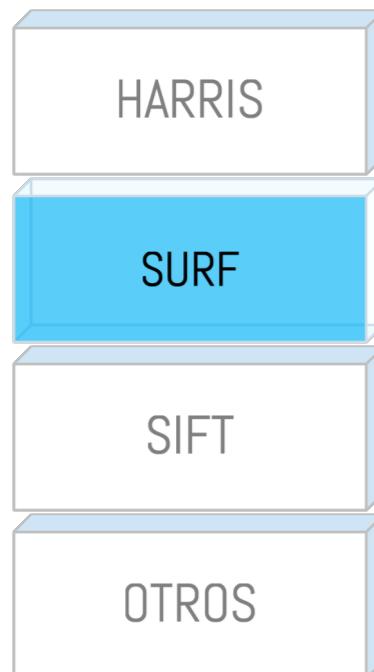
■ Otros descriptores



- El proceso de Matching NNDR es el siguiente:
 - Para cada descriptor de la imagen izquierda
 - a. Seleccione un vector de características de la imagen izquierda
 - b. Determine la distancia {Euclíadiana} respecto a todos los descriptores en la imagen derecha
 - c. Ordene la lista de distancias en orden ascendente
 - d. Seleccione las **dos menores distancias** y verifique si se cumple la siguiente condición según un umbral ($0.6 \sim 0.8$)

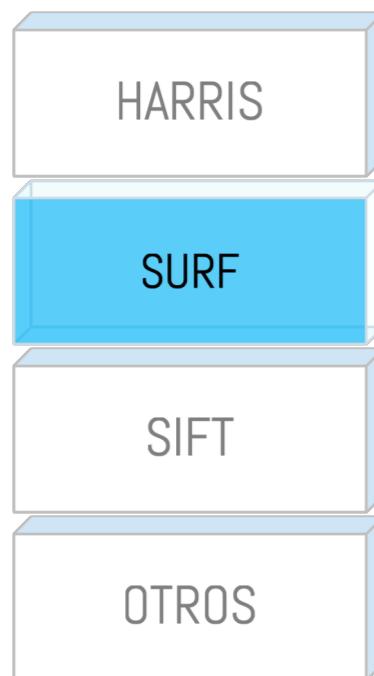


■ Otros descriptores



- El proceso de Matching NNDR es el siguiente:
 - Para cada descriptor de la imagen izquierda
 - a. Seleccione un vector de características de la imagen izquierda
 - b. Determine la distancia {Euclíadiana} respecto a todos los descriptores en la imagen derecha
 - c. Ordene la lista de distancias en orden ascendente
 - d. Seleccione las dos menores distancias y verifique si se cumple la siguiente condición según un umbral (0.6~0.8)
 - e. Si la condición es **verdadera**, entonces relacione ambos puntos.
Si la condición es **falsa**, no existe punto correspondiente en la otra imagen (o los descriptores no son buenos)

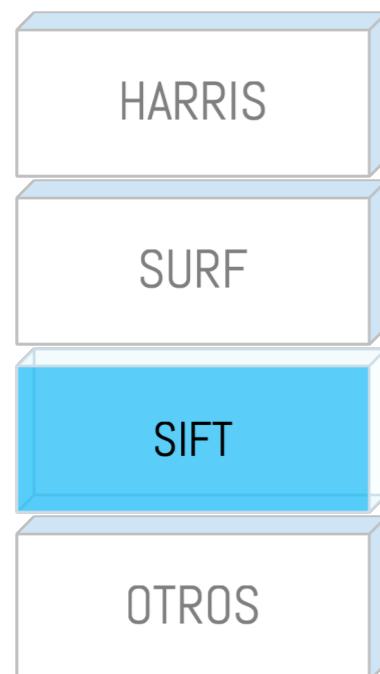
■ Otros descriptores



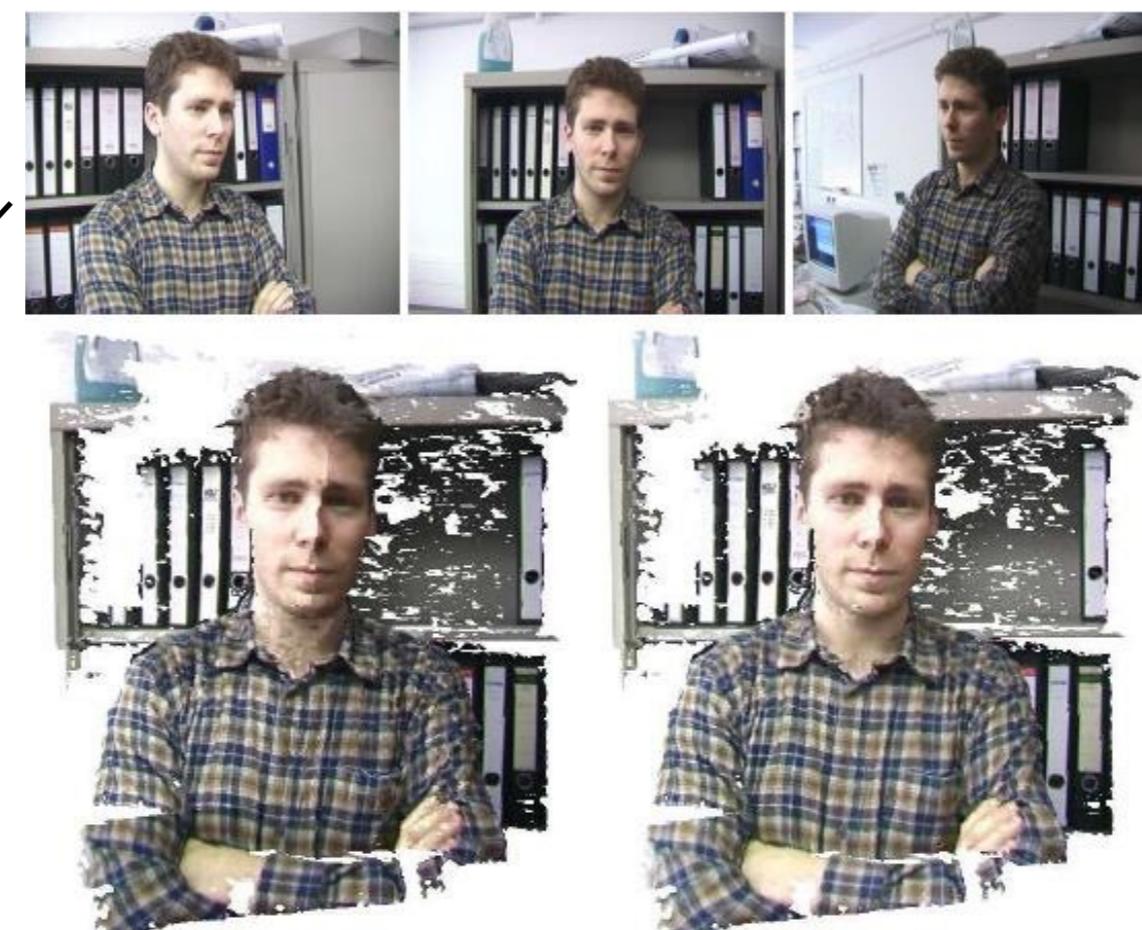
- El algoritmo NNDR tiene muy buenos resultados cuando los descriptores definen correctamente la región de interés.
- En el ejemplo anterior, la correspondencia fue correcta, y el algoritmo relacionó dos puntos de interés, a pesar de la rotación entre una imagen y otra.



■ Otros descriptores



- SIFT (Scale-invariant feature transform) es un algoritmo de detección y extracción de puntos de interés (D. Lowe, 1999)
- La mayor ventaja de SIFT es su invarianza a la escala, orientación, transformaciones afines, y parcialmente invariante ante cambios de iluminación.

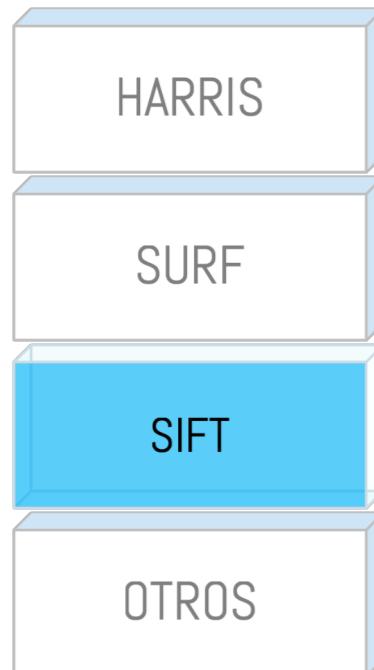


A partir de la correspondencia de tres imágenes podemos reconstruir una imagen tridimensional

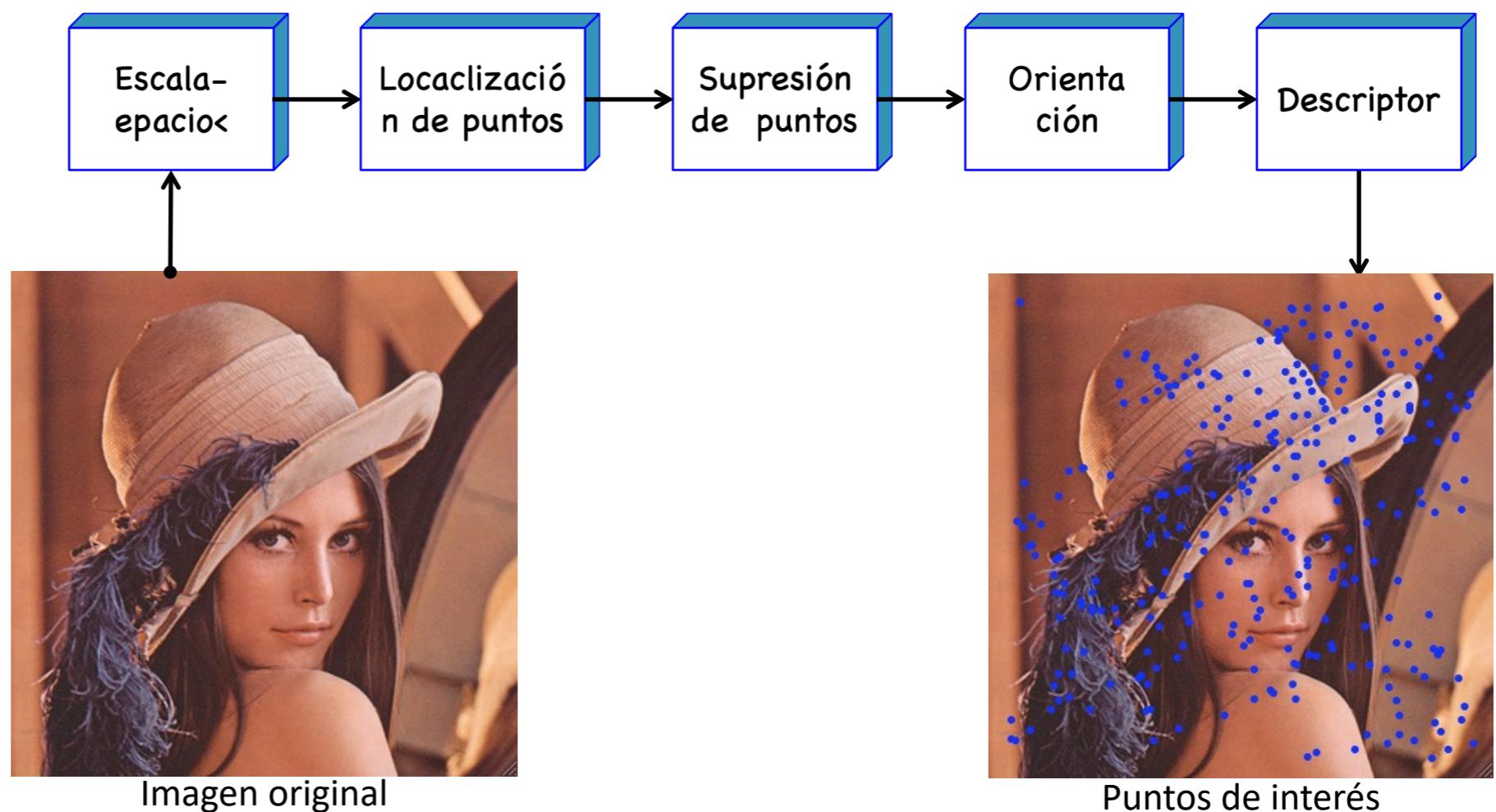
Para generar la reconstrucción 3D debemos buscar la correspondencia en tres vistas

Lowe, David G. (1999). "Object recognition from local scale-invariant features". Proceedings of the International Conference on Computer Vision. 2. pp. 1150–1157

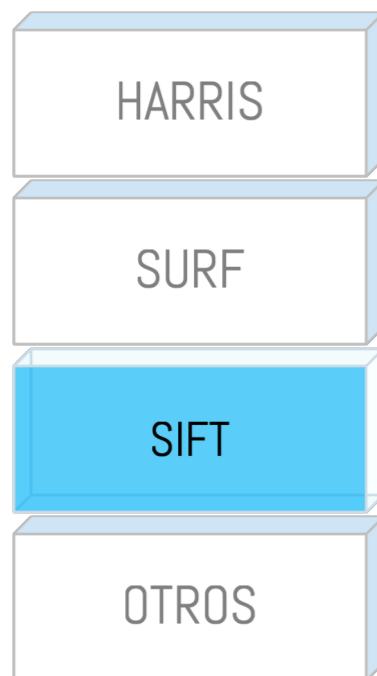
■ Otros descriptores



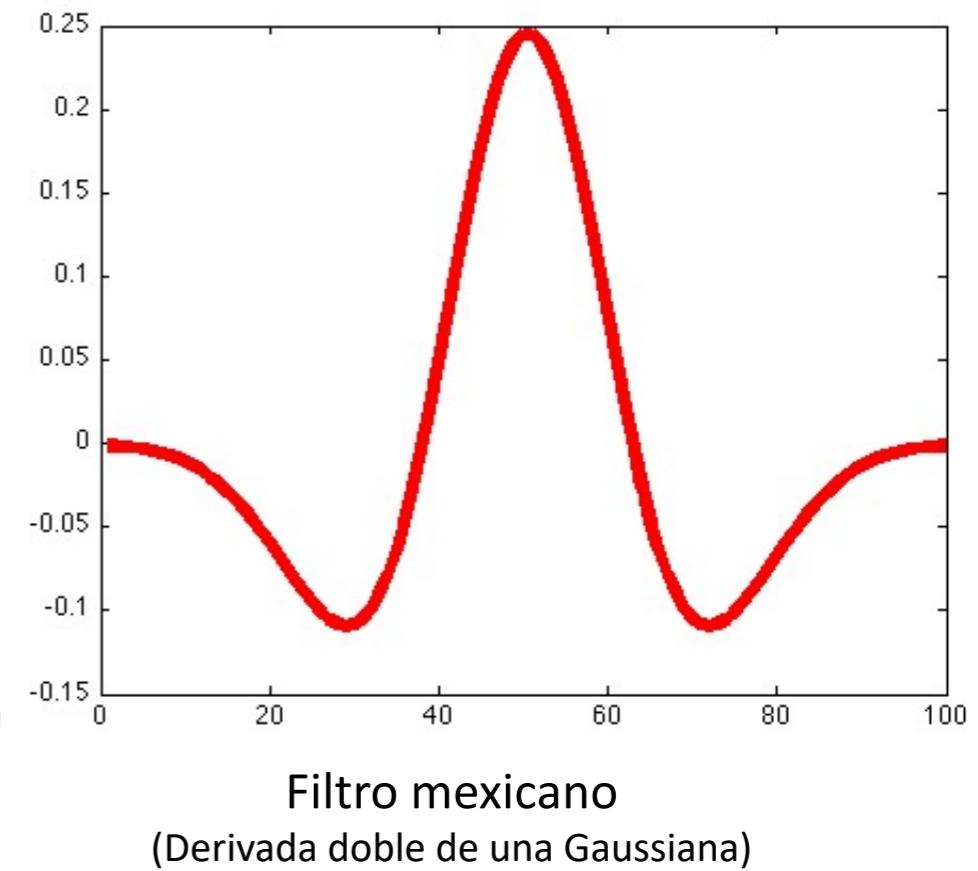
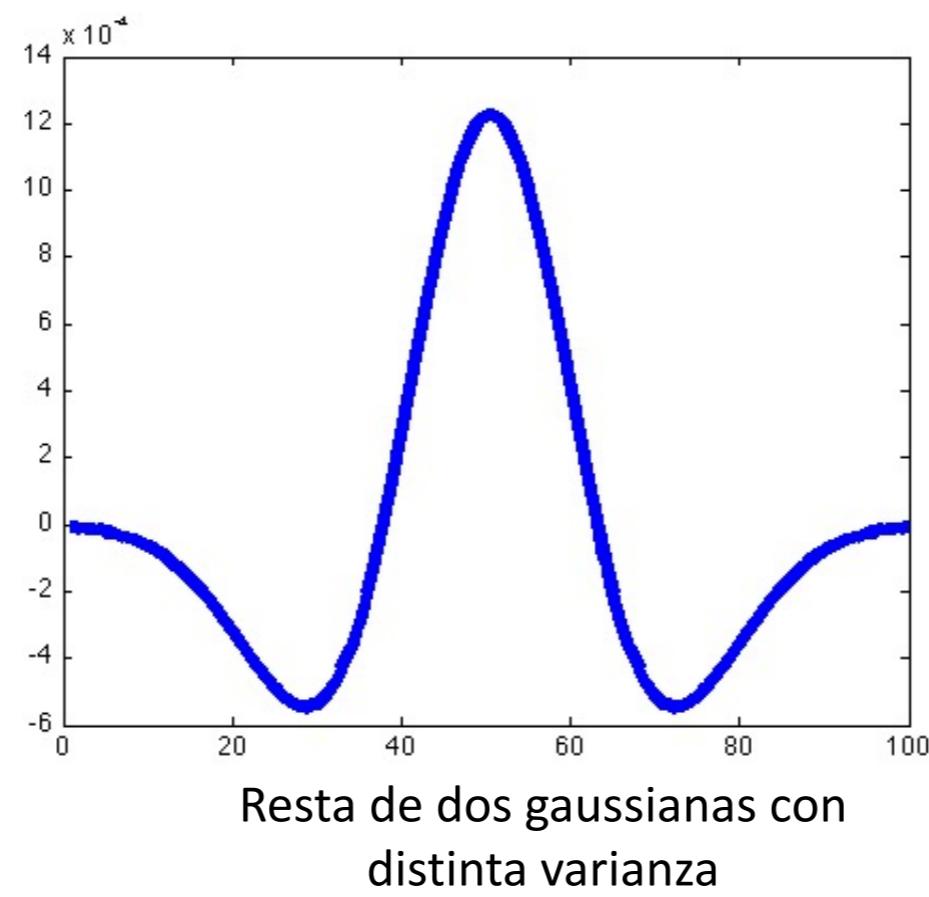
- El algoritmo SIFT está diseñado bajo un conjunto de etapas en cascada, las cuales emplean distintos principios matemáticos, como por ejemplo los filtros DoG. Estos principios los revisaremos a continuación.



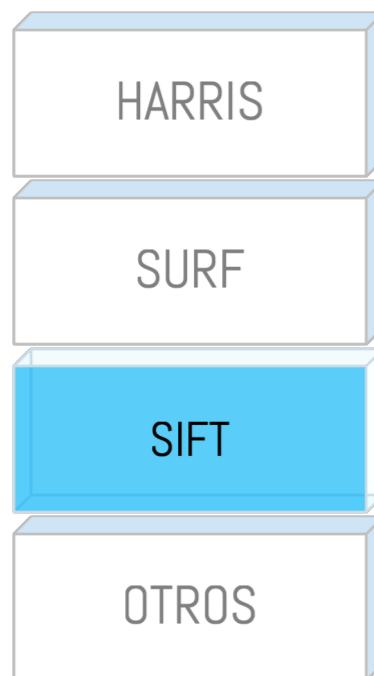
■ Otros descriptores



- *Detección de extremos*: Esta etapa detectan todos los puntos de interés candidatos en una escala-espacio, los cuales son posteriormente refinados en las siguientes etapas de SIFT.
- El principio básico consiste en calcular la diferencia entre Gaussianas (DoG) de distinto tamaño aplicados sobre la imagen.



■ Otros descriptores



- *Detección de extremos*: Lo anterior significa que **si restamos la convolución de dos gaussianas de distinto tamaño aplicadas a una imagen (DoG)** en realidad estamos derivando la imagen con el operador Laplaciano LoG (Laplacian of Gaussian). Lo anterior permite que el punto de interés sea invariante a los cambios de orientación y escala.

- Sea $L(x,y,\sigma)$ la convolución de una imagen $I(x,y)$ con una gaussiana, con desviación estándar sigma

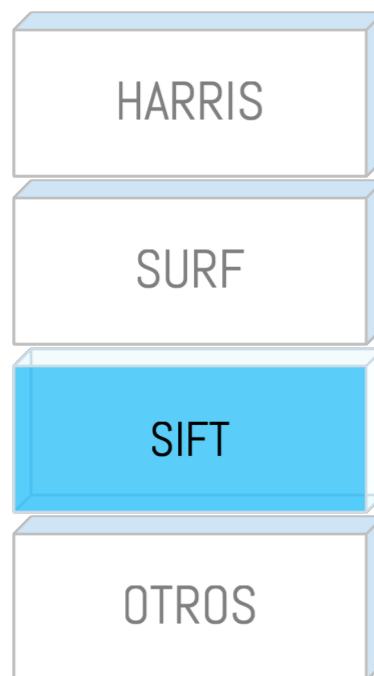
$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

- La función DoG aplicada sobre una imagen se define como:

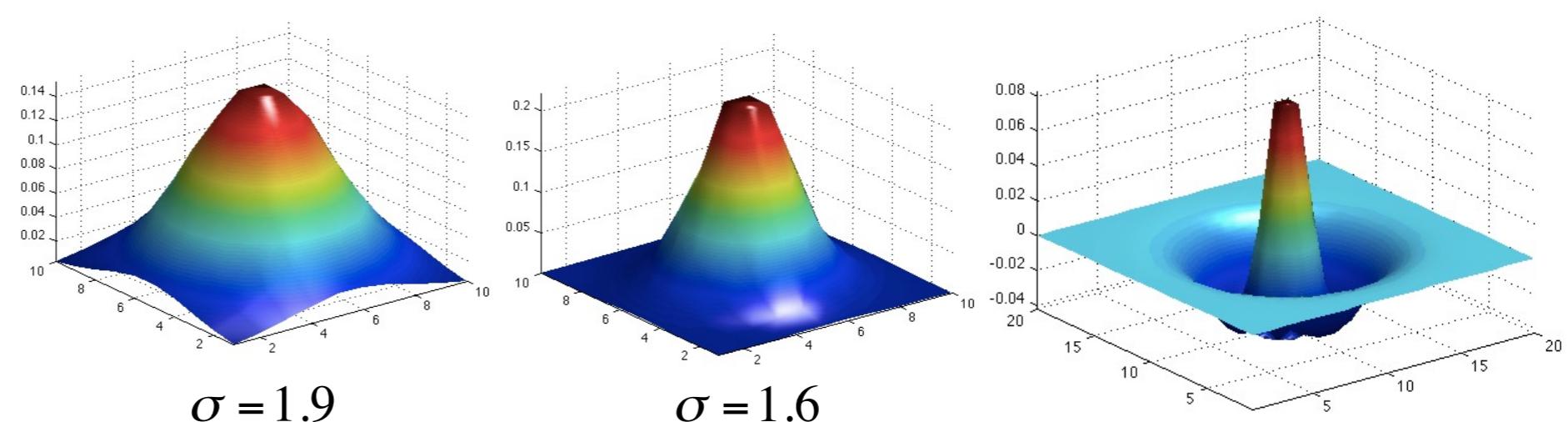
$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma)$$

donde k es un factor de escala

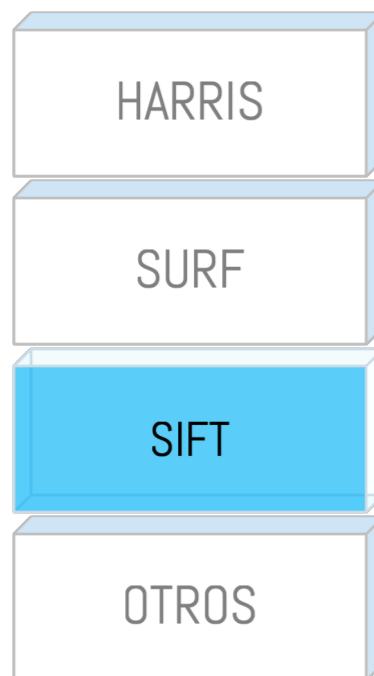
■ Otros descriptores



- *Detección de extremos*: Según los autores, esta simple forma de calcular las diferencias es más estable que el hessiano, o el detector de esquinas de Harris u otras.



■ Otros descriptores



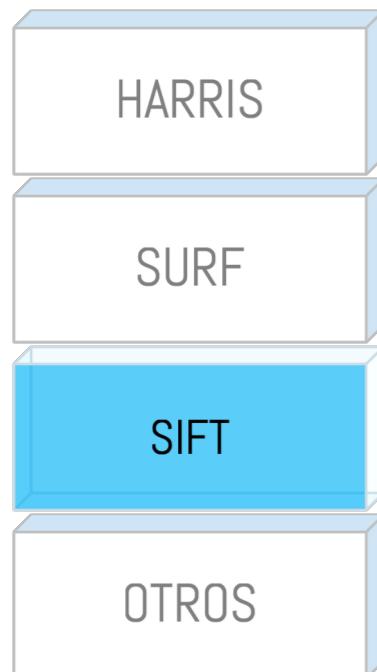
- *Detección de extremos*: El proceso de aplicar gaussianas sucesivas sobre la misma imagen es equivalente al proceso diferencial de una ecuación del calor.
- La fundamentación matemática sobre emplear una Gaussiana es porque el hecho que no se crean nuevas estructuras, además de cumplir los axiomas de escala-espacio.



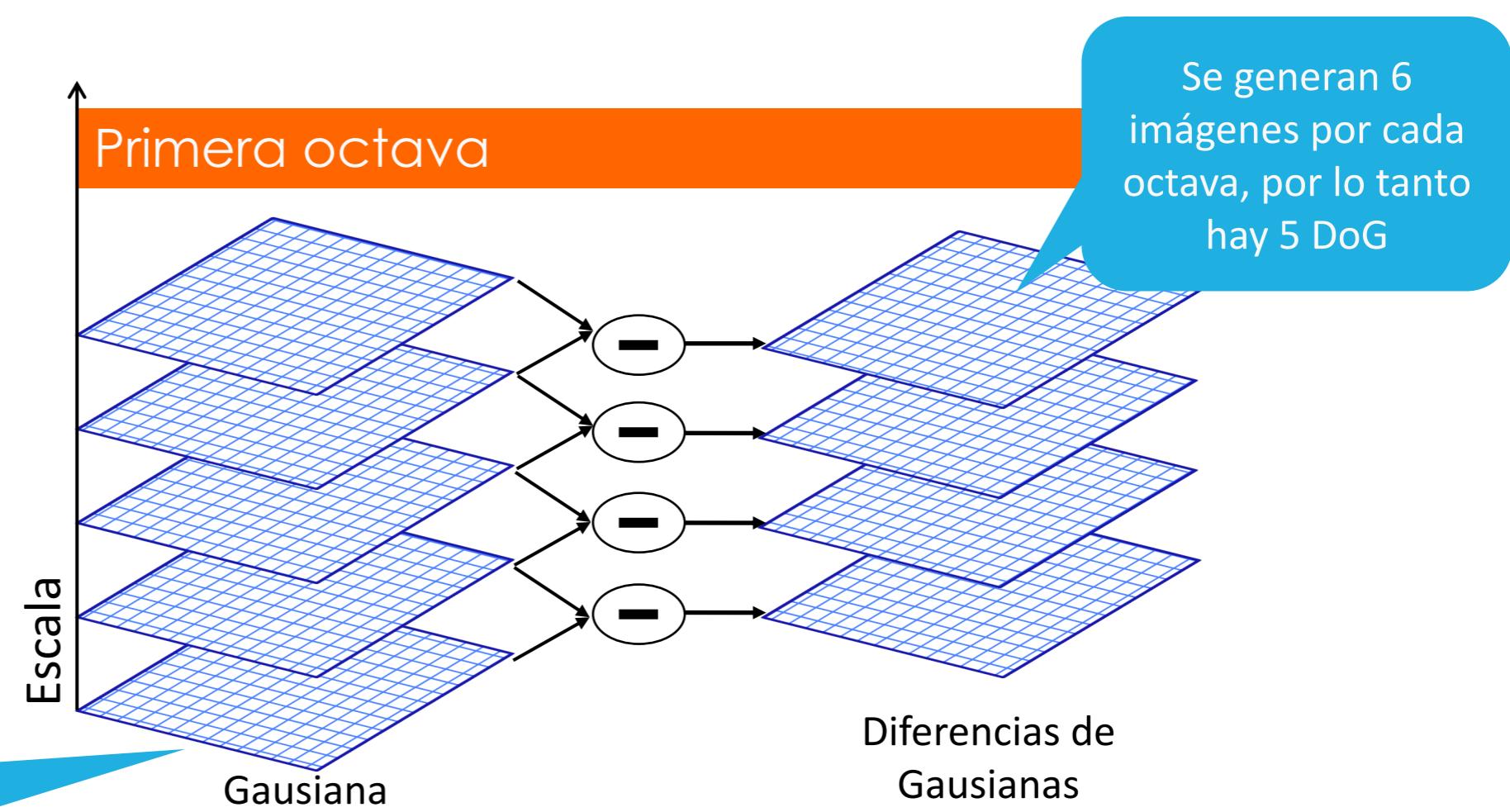
<http://en.wikipedia.org/wiki/Scale-space>

Lindeberg, Tony "Feature detection with automatic scale selection", International Journal of Computer Vision, 30, 2, pp 77–116, 1998.

■ Otros descriptores

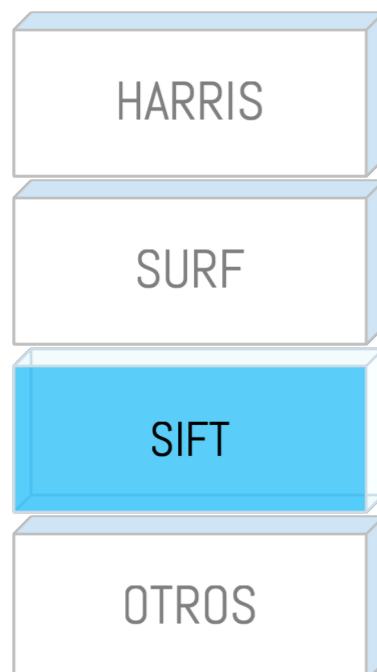


- **Detección de extremos:** El proceso de escala-espacio genera octavas. En cada octava la imagen es convolucionada repetidamente con distintas gaussianas (duplicando sigma por vez).

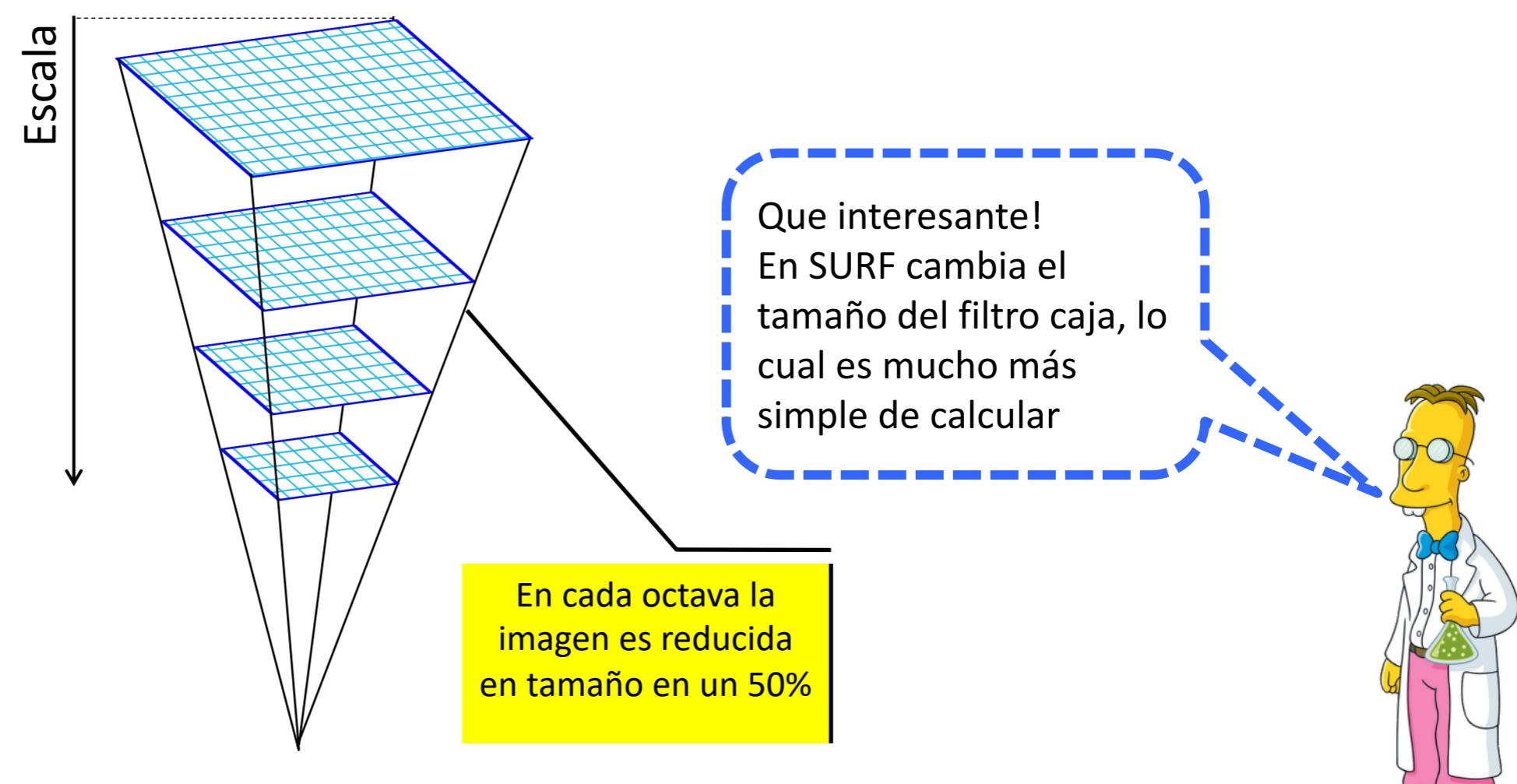


Para incrementar los puntos de interés, La imagen original de la primera octava es duplicada en tamaño

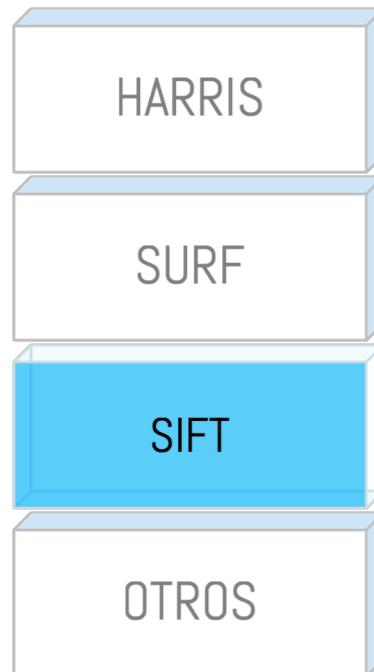
■ Otros descriptores



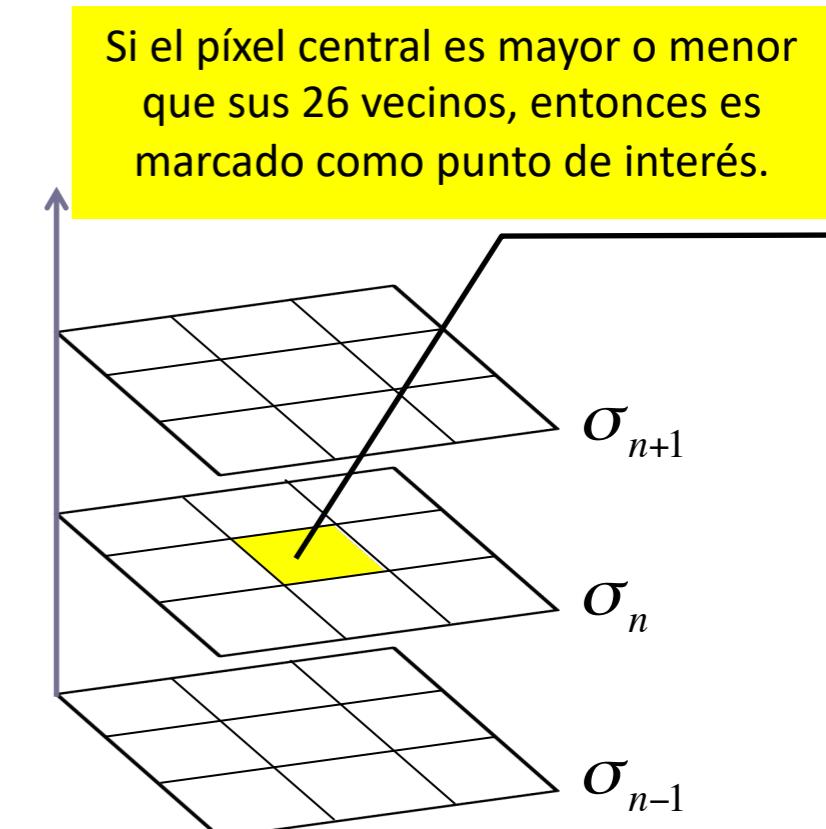
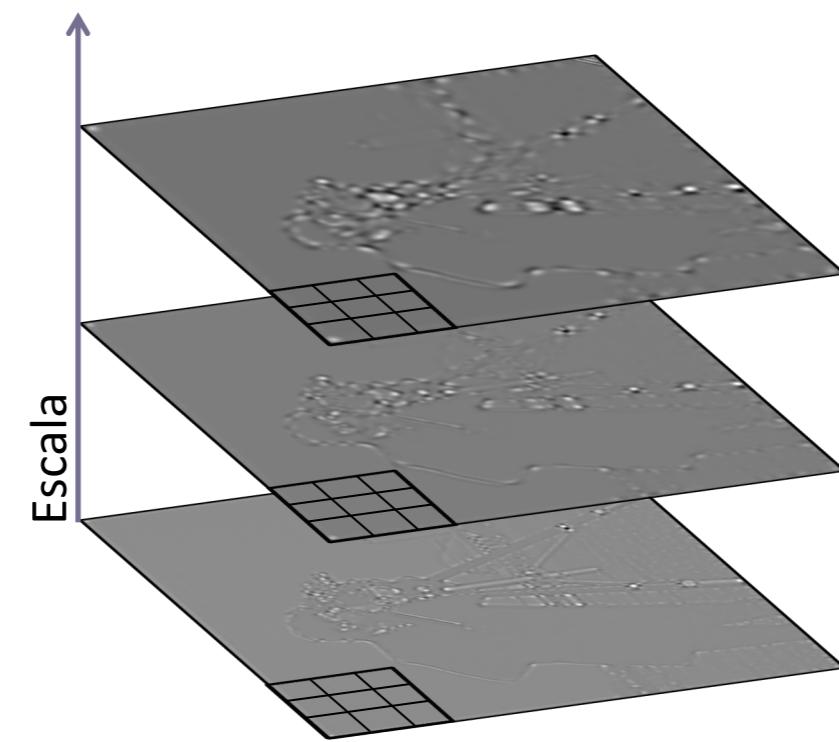
- **Detección de extremos:** Como muestra la figura, SIFT reduce la imagen original octava a octava en un factor de 2, siendo el inicio de cada octava la imagen de inicio de la octava anterior (con el doble de sigma)



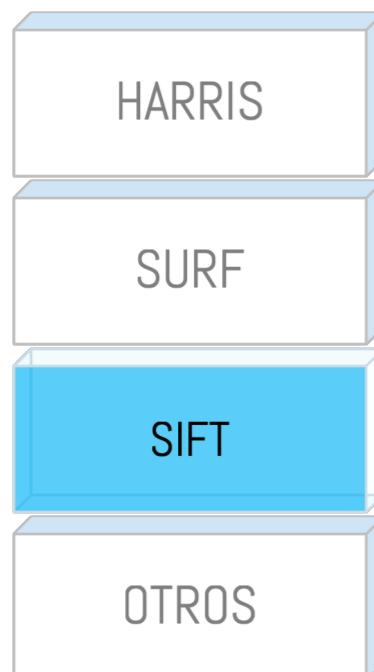
■ Otros descriptores



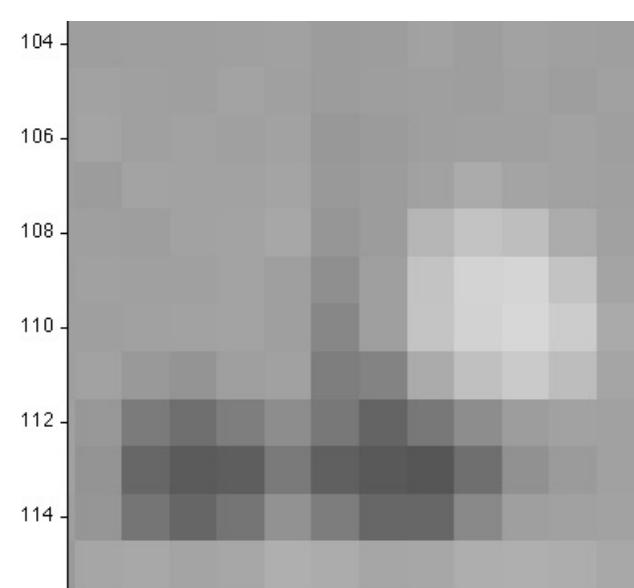
- *Localización de puntos de interés*: En este paso son determinados los puntos de interés. El proceso consiste en buscar los máximos y mínimos en un bloque de $3 \times 3 \times 3$.
- El píxel central del bloque es comparado con sus 26 vecinos. En caso que dicha píxel sea mayor o menor, éste es marcado como un punto candidato



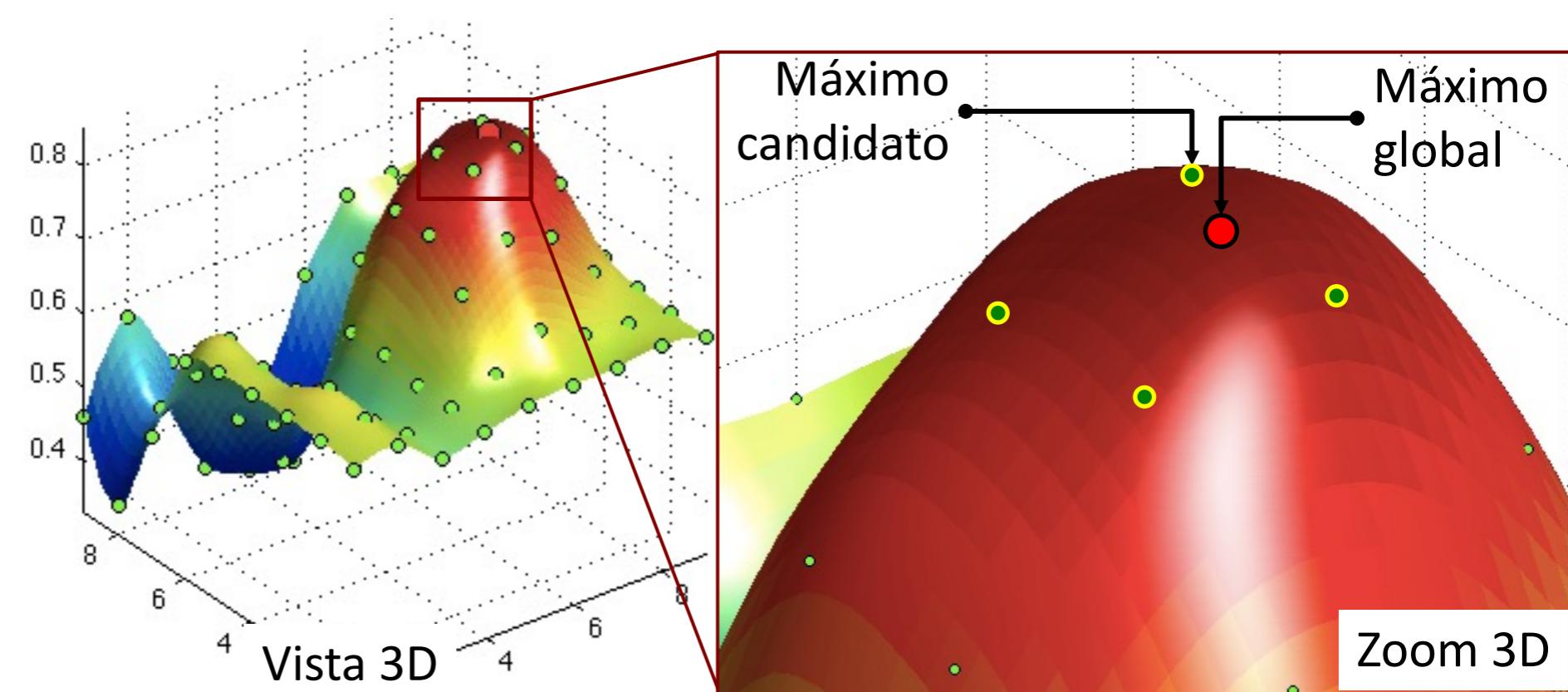
■ Otros descriptores



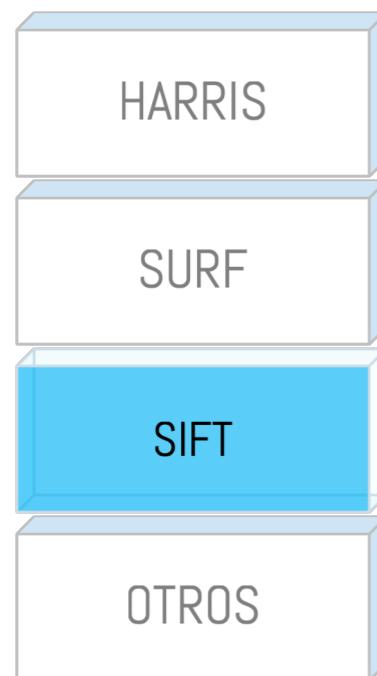
- *Supresión de puntos no estables*: Para mejorar el rendimiento, los autores proponen primero localizar el máximo o mínimo con una resolución sub píxel a través de una interpolación.
- Como observamos el **máximo candidato** puede no ser el **máximo global**. Si el punto candidato se encuentra cerca del punto global entonces mantenemos la posición del candidato.



Vista 2D



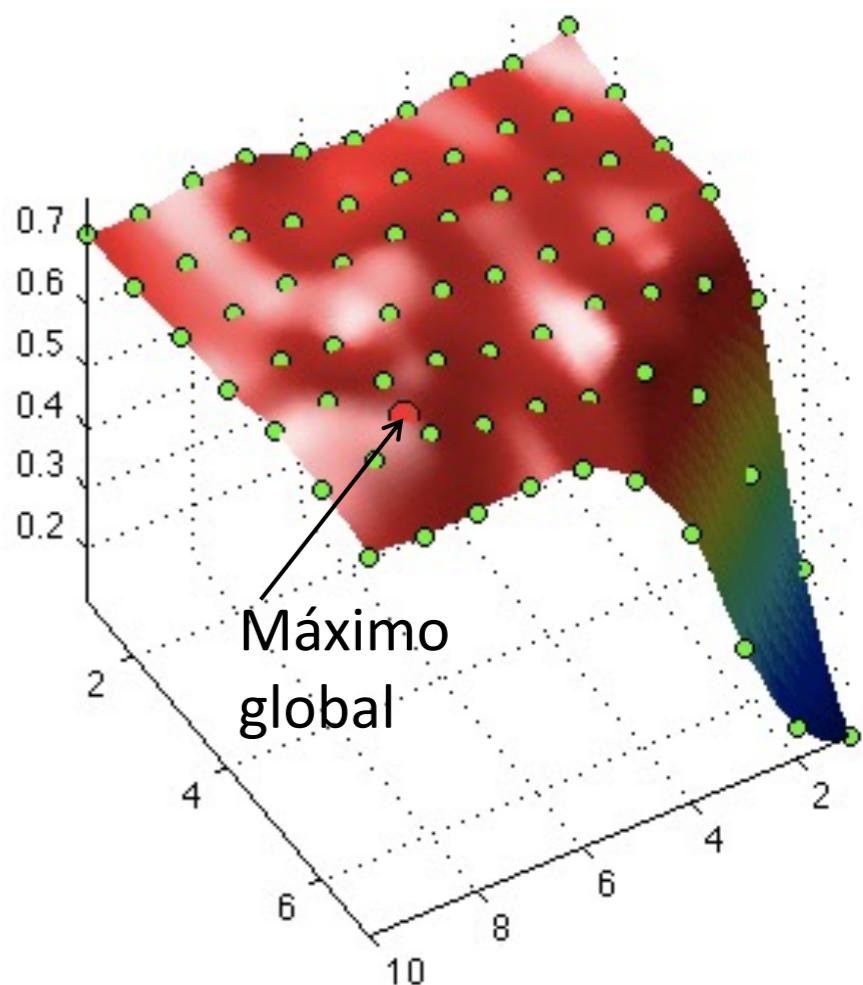
■ Otros descriptores



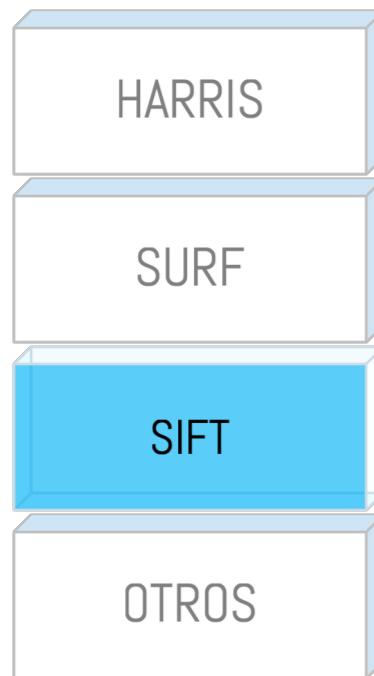
- *Supresión de puntos no estables*: Algunos de los puntos detectados anteriormente pueden ser inestables. Esto sucede cuando no están situados sobre bordes, o tienen bajo contraste, por lo cual son sensibles al ruido.
- Para excluir dichos puntos, los autores proponen dos criterios.

PRIMER CRITERIO:

- Para eliminar los puntos con bajo contraste, aplicamos una umbralización. Si un punto candidato se encuentra bajo un umbral D , éste será excluido por ser considerado inestable ($D=0.03$).



■ Otros descriptores



- *Supresión de puntos no estables*: Algunos de los puntos detectados anteriormente pueden ser inestables. Esto sucede cuando no están situados sobre bordes, o tienen bajo contraste, por lo cual son sensibles al ruido.
- Para excluir dichos puntos, los autores proponen dos criterios.

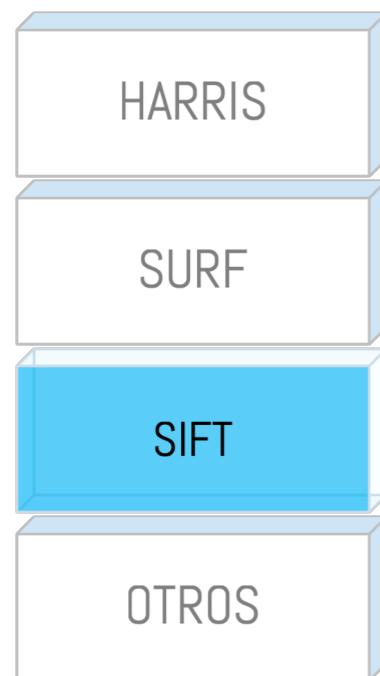
SEGUNDO CRITERIO:

- Para estudiar la respuesta al borde, calculamos el Hessiano en dicho punto
- Luego evaluamos la siguiente desigualdad

$$H = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix}$$

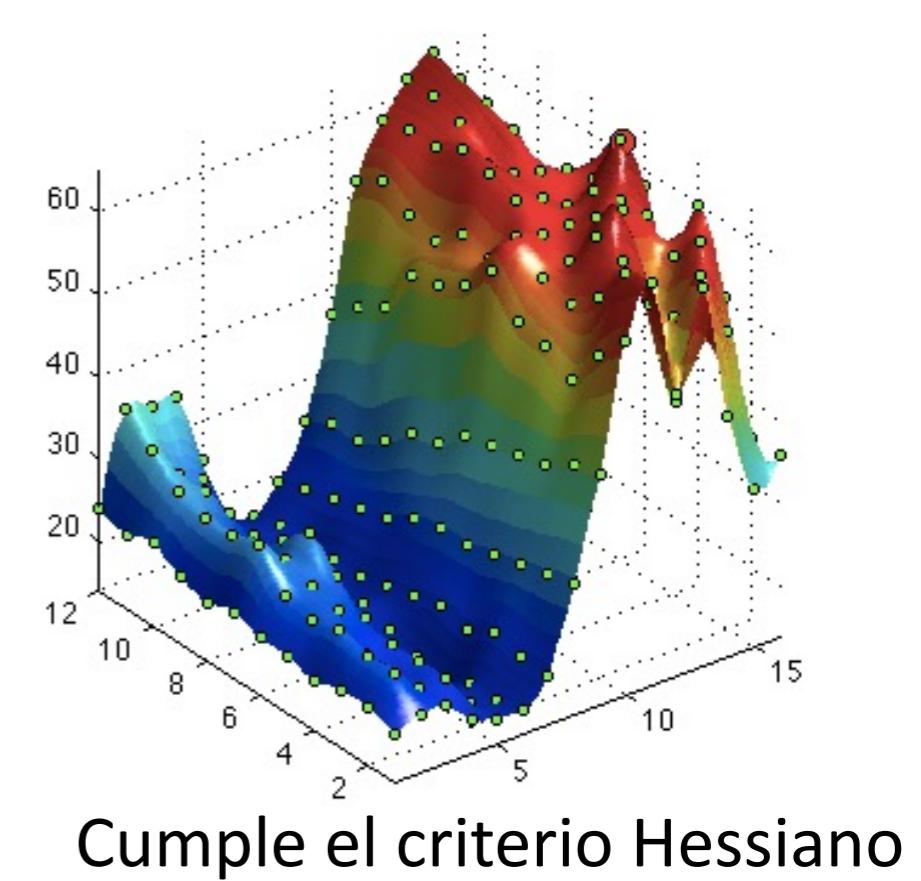
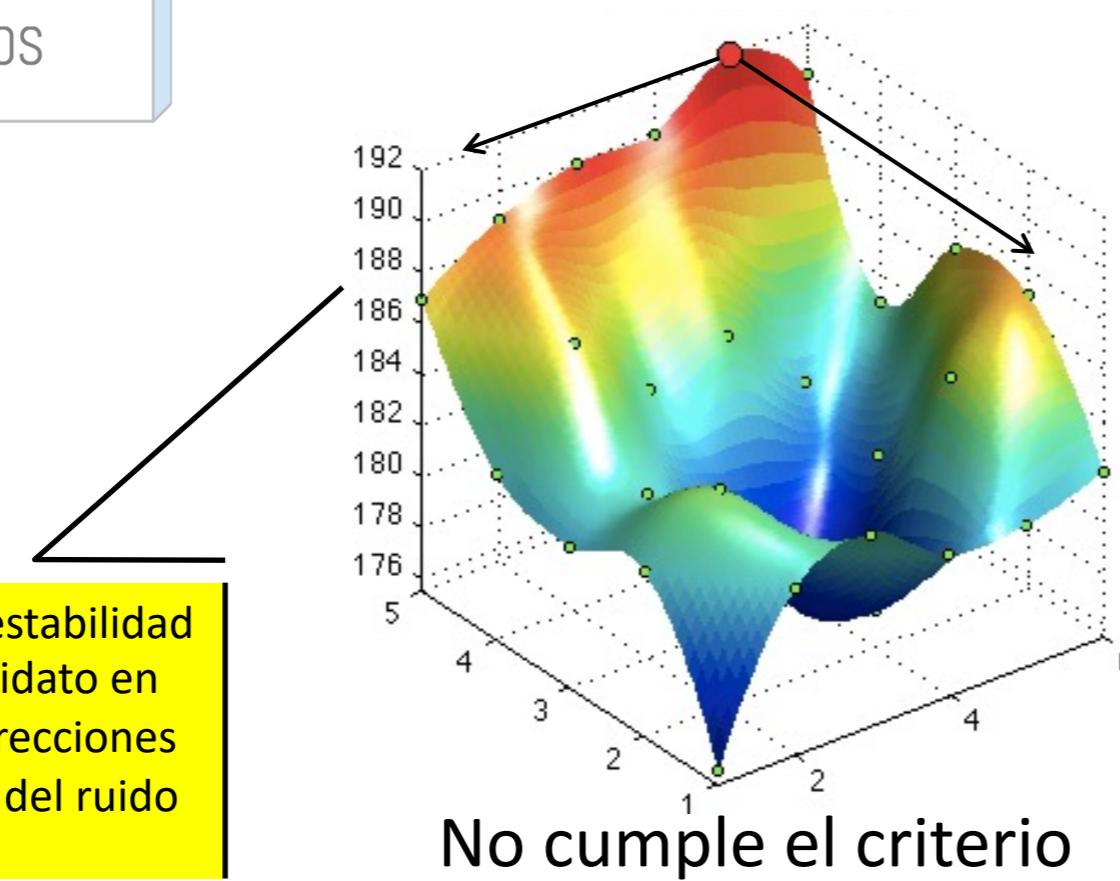
$$\frac{Tr(H)}{det(H)} = \frac{\left(\frac{\partial^2 D}{\partial x^2} + \frac{\partial^2 D}{\partial y^2} \right)^2}{\left(\frac{\partial^2 D}{\partial x^2} \times \frac{\partial^2 D}{\partial y^2} \right) - \left(\frac{\partial^2 D}{\partial x \partial y} \right)^2} < \frac{(r+1)^2}{r}$$

■ Otros descriptores

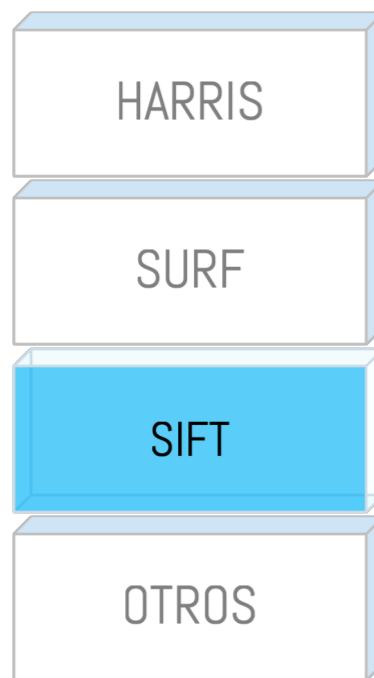


- *Supresión de puntos no estables*: Algunos de los puntos detectados anteriormente pueden ser inestables. Esto sucede cuando no están situados sobre bordes, o tienen bajo contraste, por lo cual son sensibles al ruido.
- Analicemos dos ejemplos del segundo criterio Hesiano.

Mucha inestabilidad
del candidato en
ambas direcciones
producto del ruido

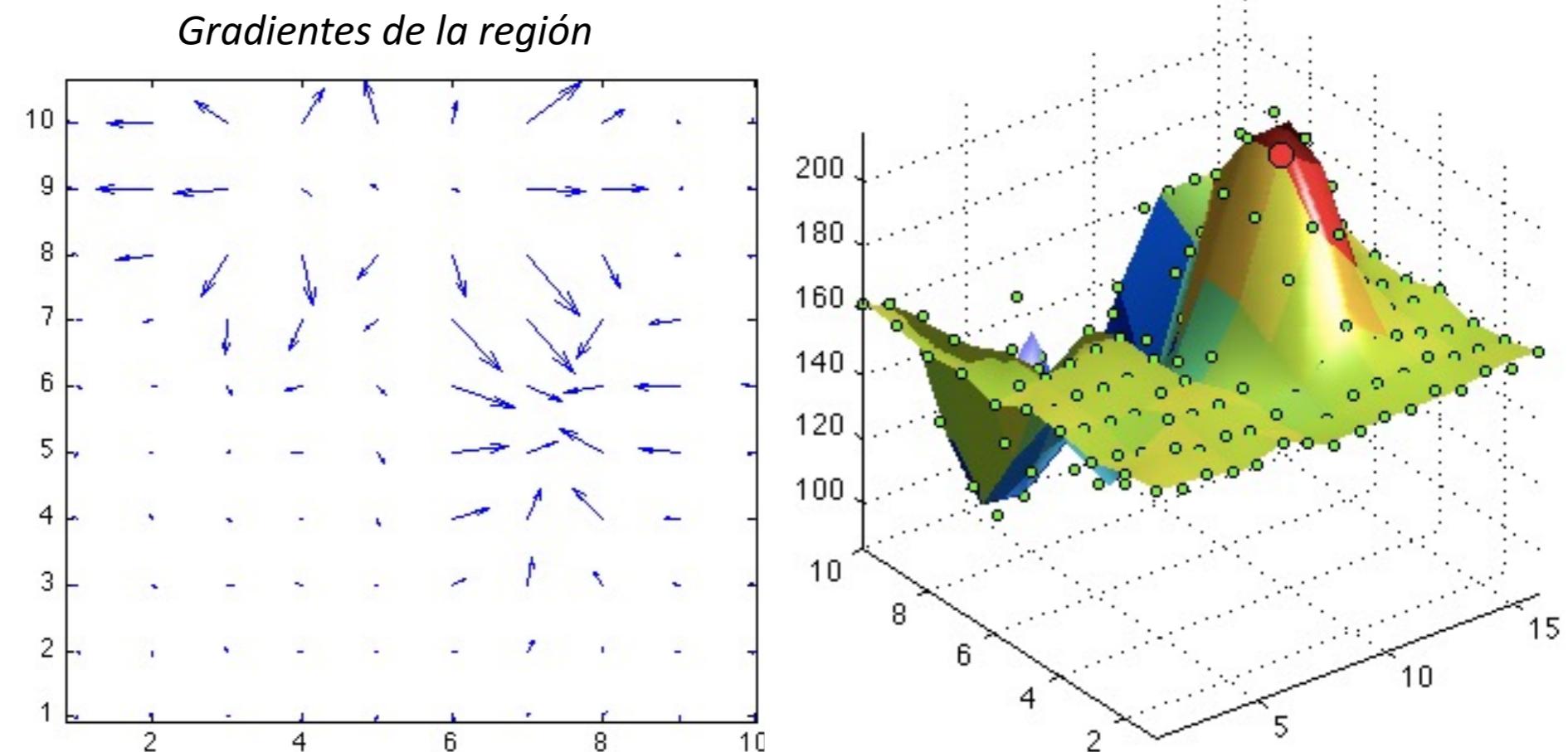


■ Otros descriptores

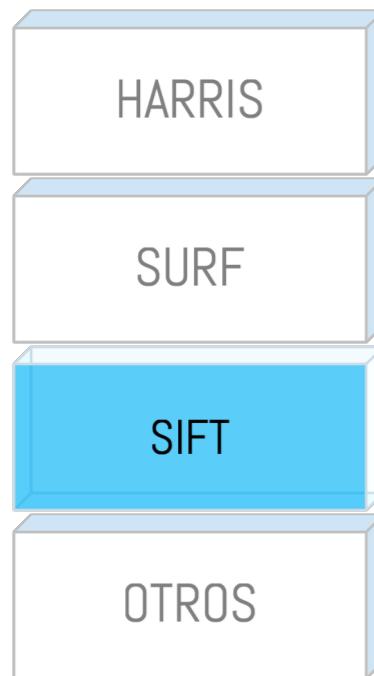


- **Orientación:** Los puntos candidatos que pasan los test anteriores ya son puntos de interés. A todos ellos es necesario calcular la orientación centrada en el punto de interés según la siguiente ecuación

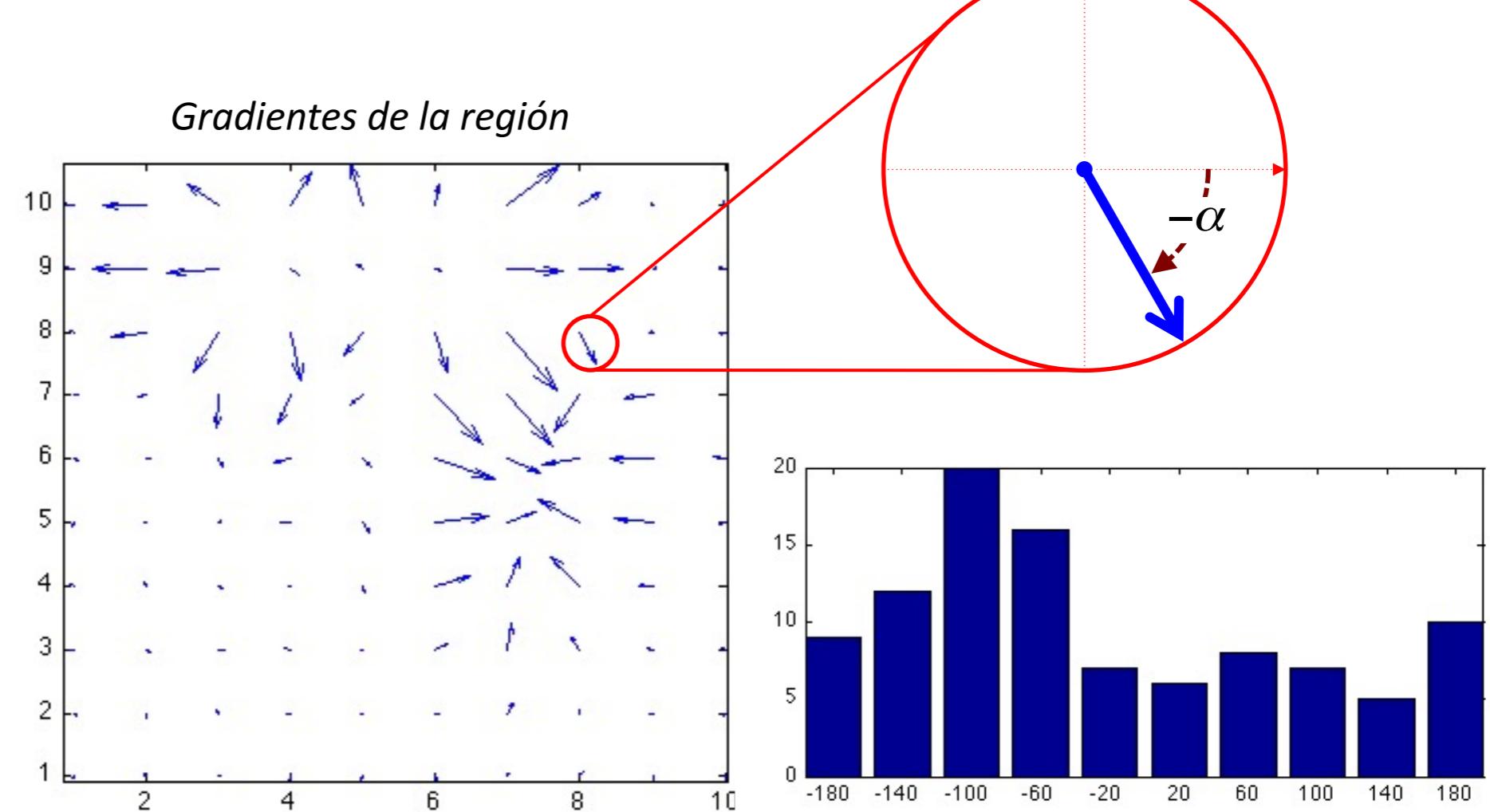
$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,-1y))^2}$$



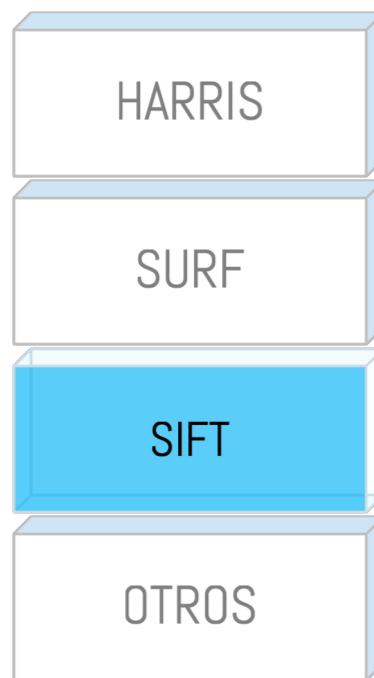
■ Otros descriptores



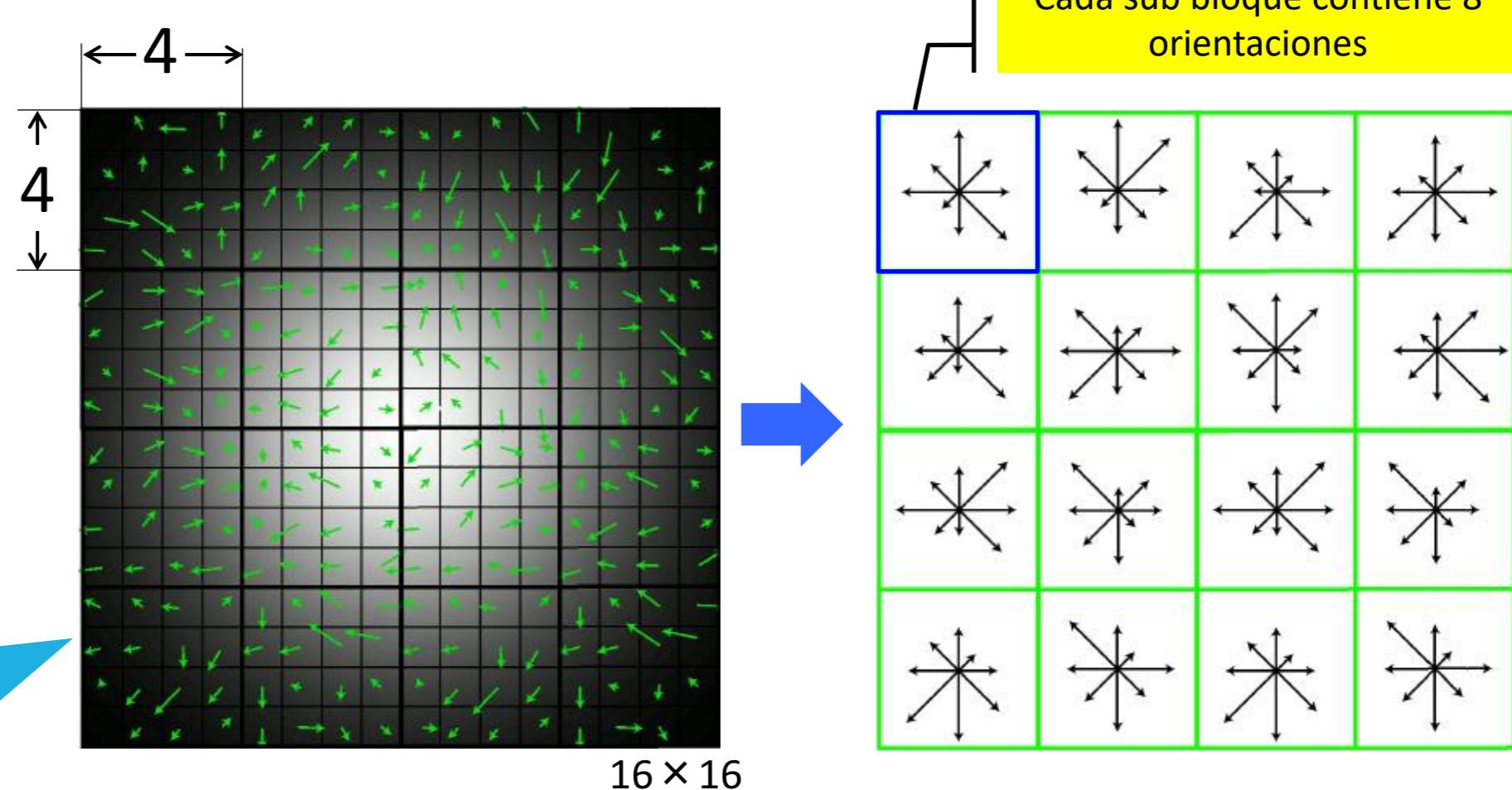
- **Orientación:** Por cada gradiente calculamos su ángulo y luego calculamos su histograma en bloques de 10 grados, hasta completar los 360 grados. Al mayor histograma se le asigna la orientación de la región.



■ Otros descriptores

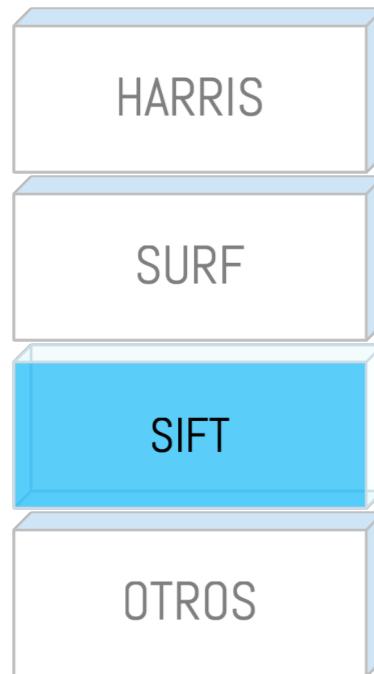


- **Descriptor:** Finalmente construimos el descriptor centrado en punto de interés en un bloque de 16×16 píxeles.
- Dentro de cada bloque de 4×4 px calculamos los gradientes en 8 direcciones (45 grados). De esta forma generamos un descriptor de $4 \times 4 \times 8 = 128$



Para evitar los efectos de la iluminación, los gradientes son ponderados por una Gaussiana.

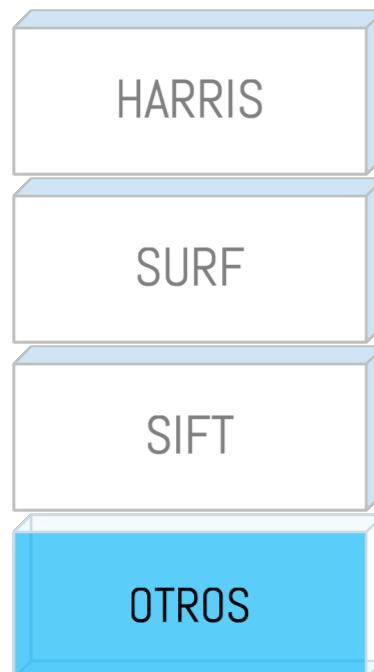
■ Otros descriptores



- Aplicaciones: Una de las primeras etapas de los algoritmos de visión por computador consiste en efectuar el matching entre pares de imágenes.

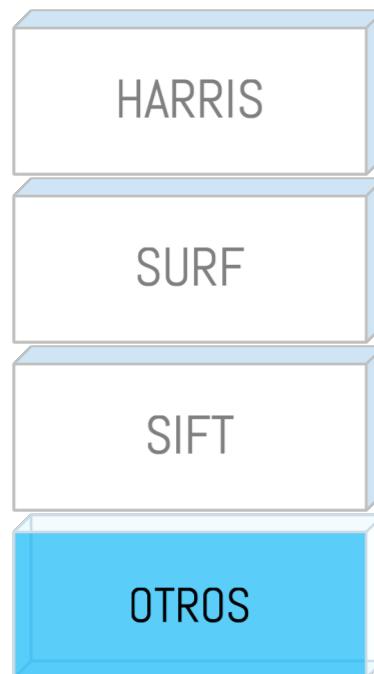


■ Otros descriptores



- Hemos visto dos importantes descriptores: SIFT y SURF. En ellos aprendido dos importantes etapas.
- **Primero**, la **detección de puntos de interés**, a través de:
 - Diferencia de Gaussianas (DoG)
 - Determinación del Hessiano (HoG)
 - Laplaciano del Gaussiano (LoG)
- Otros detectores corresponden a:
 - MSER, Matas 2002
 - EBR e IBR, Tuytelaars & Van Gool 2004
 - PCBR. Deng, et al. 2007
- Estos detectores tienen la capacidad para describir puntos de interés de la escena sin otro proceso previo.

■ Otros descriptores



- **Segundo:** por cada punto de interés, extraemos un conjunto de descriptores de una subregión centrada en cada punto de interés. En la literatura existen muchos más descriptores de los cuales destacamos:
 - PCA-SIFT: Principal Component Analysis-SIFT
 - GLOG : Gradient Location and Orientation Histogram
 - HOG : Histogram of Oriented Gradients
 - LESH : Local Energy based Shape Histogram, 2008
- Características que todo “buen” descriptor debe poseer
 - **Invariante** : No cambia a pesar de las transformaciones.
 - **Robusto** : Ruido, compresión, blur.
 - **Distintivo** : Capacidad de ser distingible frente a una BD
 - **Cantidad** : Detectar puntos en objetos pequeños
 - **Precisión** : Localización precisa.
 - **Eficiente** : Cercano a tiempo-real