Detección de bordes

Bordes

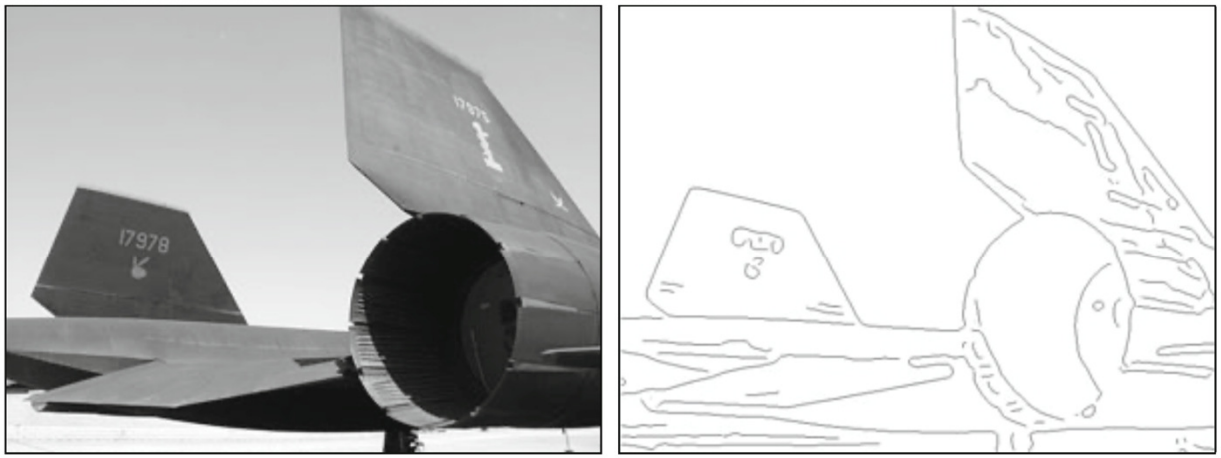


Fig. 6.1
Edges play an important role in human vision. Original image (a) and edge image (b).

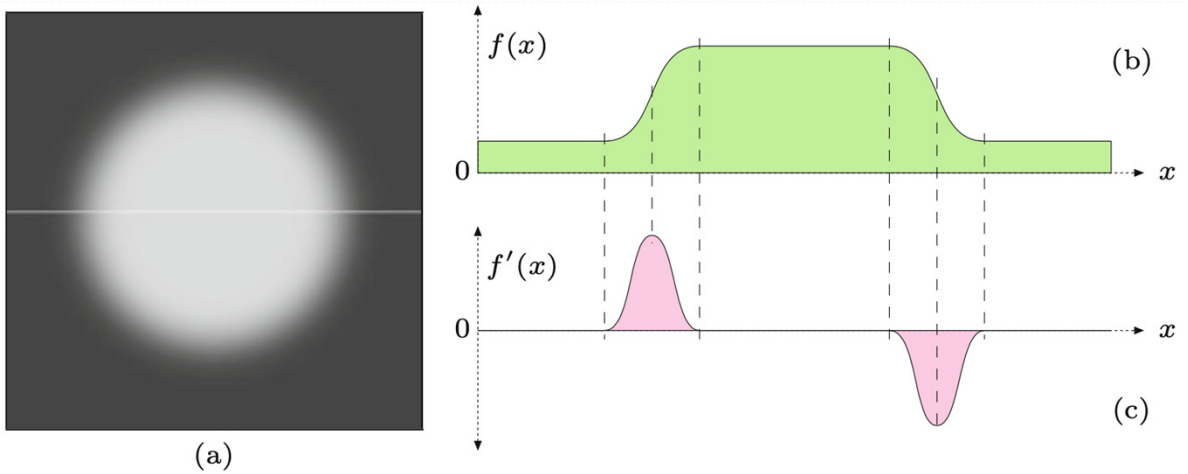
- Los bordes son lugares de la imagen en los que la intensidad local cambia a lo largo de una orientación particular.

Detección de bordes

- Operadores basados en la primera derivada (gradiente).
- Operadores de brújula.
- Operadores basados en la segunda derivada.
- Operador de Canny.

Gradiente en 1D

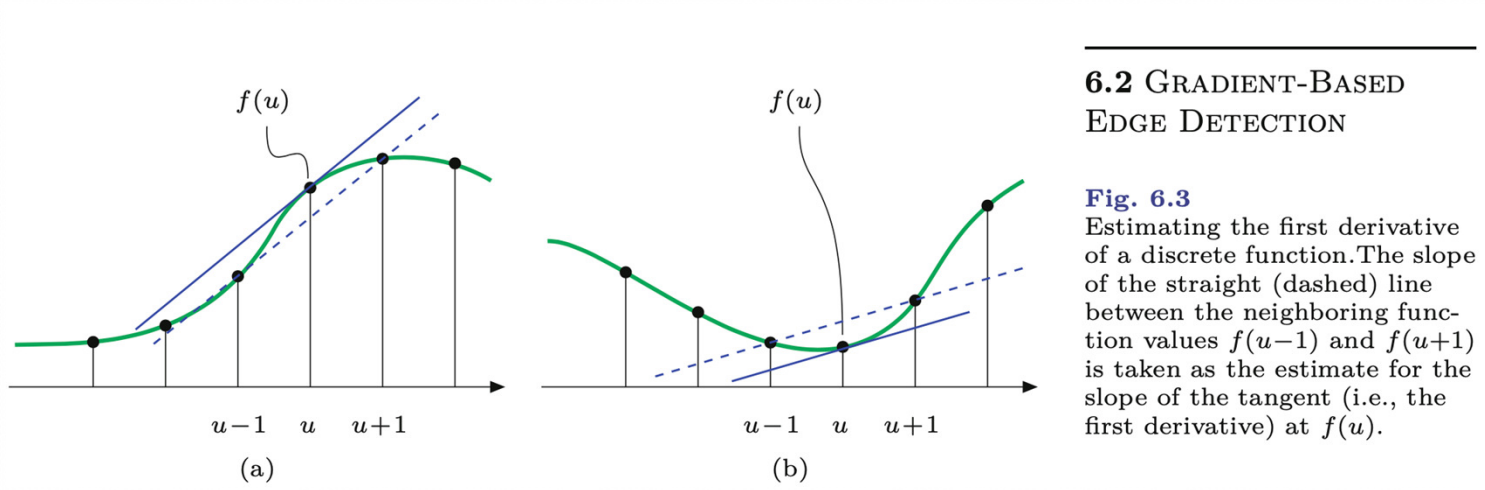
Fig. 6.2
Sample image and first derivative in one dimension: original image (a), horizontal intensity profile $f(x)$ along the center image line (b), and first derivative $f'(x)$ (c).



- La primera derivada $f'(x) = \frac{df}{dx}(x)$ se muestra como una oscilación positiva las posiciones donde aumenta la intensidad y una oscilación negativa donde la intensidad disminuye.

Gradiente en 1D

- La primera derivada no está definida para una función discreta $f(u)$.
- Una forma de aproximar la tangente de $f(u)$ en la posición u es trazar una recta entre sus vecinos $f(u-1)$ y $f(u+1)$.



6.2 GRADIENT-BASED EDGE DETECTION

Fig. 6.3
Estimating the first derivative of a discrete function.The slope of the straight (dashed) line between the neighboring function values $f(u-1)$ and $f(u+1)$ is taken as the estimate for the slope of the tangent (i.e., the first derivative) at $f(u)$.

Gradiente en 2D

- Para el eje u:
- $\frac{df}{dx}(u) \approx \frac{f(u+1)-f(u-1)}{(u+1)-(u-1)} = \frac{f(u+1)-f(u-1)}{2}$
- Para el eje v:
- $\frac{df}{dy}(v) \approx \frac{f(v+1)-f(v-1)}{(v+1)-(v-1)} = \frac{f(v+1)-f(v-1)}{2}$

Operadores de gradiente

- Operador de Prewitt.
- Operador de Sobel.
- Operador de Scharr (a.k.a. operador mejorado de Sobel).
- Operador de Roberts.

Operadores de gradiente

- Calculan el gradiente de la imagen mediante filtros.
- Utilizan un filtro para el eje u y otro para el eje v.
- Se puede combinar para obtener una sola imagen.

Operador de Prewitt

- Calcula el gradiente promedio sobre tres columnas y renglones adyacentes.
- Los kerneles son:
- $H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ y $H_y^P = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

Operador de Sobel

- Es similar al operador Prewitt, solo varían los pesos en los filtros.
- Los kerneles son:
- $H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ y $H_y^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
- Por lo general da mejores resultados que el operador de Prewitt.

Operador de Scharr

- También conocido como operador mejorado de Sobel.
- Los kerneles son:
- $H_x^{S'} = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$ y $H_y^{S'} = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$
- Puede dar mejores resultados que el operador de Sobel.

Operador de Roberts

- Es uno de los primeros filtros de borde.
- Se estudia por razones históricas.
- Los kerneles son.
- $H_1^R = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ y $H_2^R = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$
- No es muy selectivo a la orientación.

Operadores de gradiente

- Los operadores de Sobel y Scharr están implementados en OpenCV: cv.Sobel, cv.Scharr.
- Los operadores de Prewitt y Roberts se pueden implementar usando la función filter2d.

Ejemplo

```
# File: sobel.py
# directional gradients
grad_x = cv.Sobel(image, ddepth=cv.CV_16S, dx=1, dy=0, ksize=3)
grad_y = cv.Sobel(image, ddepth=cv.CV_16S, dx=0, dy=1, ksize=3)
# scales, calculates absolute values, and converts the result to 8-bit
abs_grad_x = cv.convertScaleAbs(grad_x)
abs_grad_y = cv.convertScaleAbs(grad_y)
# gradient
grad = cv.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
```

Imagen original

Resultado

Operadores de brújula (compass)

- Los operadores anteriores usan dos filtros, uno para el eje u y otro para el eje v.
- Una alternativa es utilizar varios filtros con distintas orientaciones.

Operadores de brújula (compass)

- Operador de Kirsch.
- Operador extendido de Sobel.

Operador de Kirsch

- El operador toma un kernel y lo gira en incrementos de 45 grados a través de las 8 direcciones de la brújula: N, NW, W, SW, S, SE, E y NE.
- Los kerneles son.

- $g^{(1)} = \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$ $g^{(2)} = \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$
- $g^{(3)} = \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix}$ $g^{(4)} = \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix}$

Operador de Kirsch

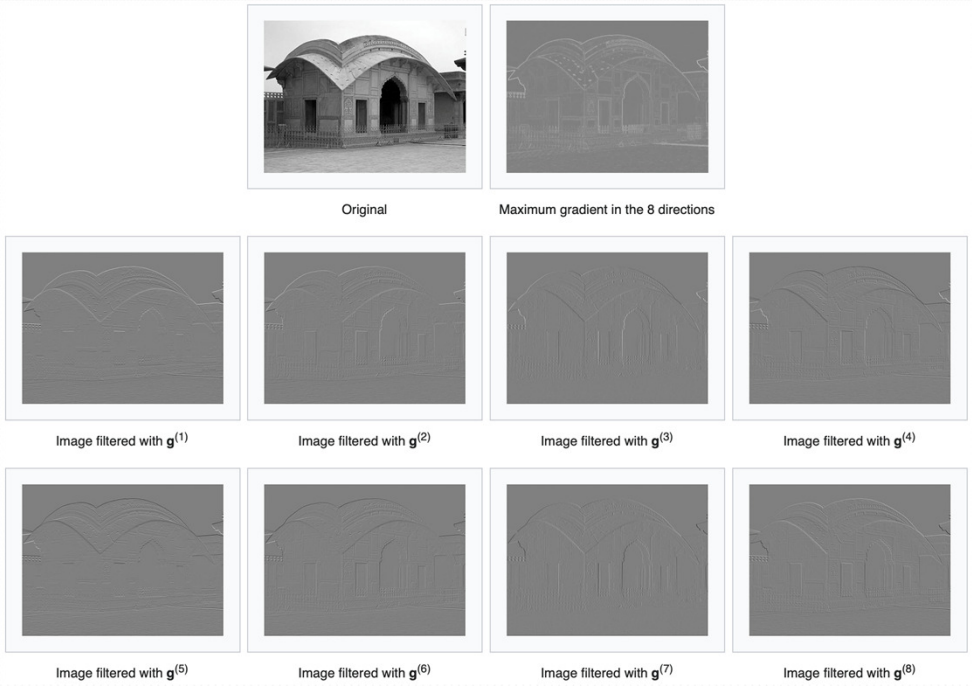
- $g^{(5)} = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix}$

$g^{(6)} = \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$

- $g^{(7)} = \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix}$

$g^{(8)} = \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix}$

Operador de Kirsch



Fuente: https://en.wikipedia.org/wiki/Kirsch_operator

Operador extendido de Sobel

- Emplea 8 filtros separados por 45^o.

$$H_0^{\mathrm{ES}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad H_1^{\mathrm{ES}} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \tag{6.17}$$

$$H_2^{\mathrm{ES}} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \qquad H_3^{\mathrm{ES}} = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}, \tag{6.18}$$

$$H_4^{\mathrm{ES}} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \qquad H_5^{\mathrm{ES}} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}, \tag{6.19}$$

$$H_6^{\mathrm{ES}} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \qquad H_7^{\mathrm{ES}} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}. \tag{6.20}$$

Conclusión

- En la práctica, los operadores de brújula muestran solo beneficios menores sobre los operadores de Sobel y Scharr.
- No están implementados en OpenCV.

Operadores basados en la segunda derivada

- Laplaciano.
- Laplaciano del gaussiano (LoG).

Filtro laplaciano

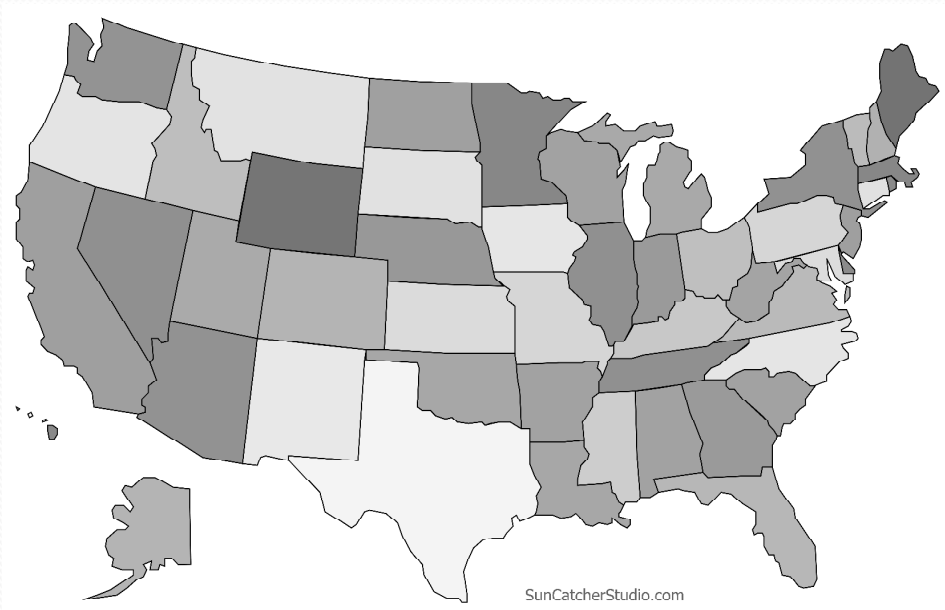
- El filtro laplaciano calcula la segunda derivada de una imagen.
- Se puede usar para detectar bordes y para mejorar la nitidez de la imagen (*image sharpening*).
- Dos kerneles utilizados para aproximar las segundas derivadas son:
- $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ y $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
- Se les conoce como kerneles laplacianos positivos.

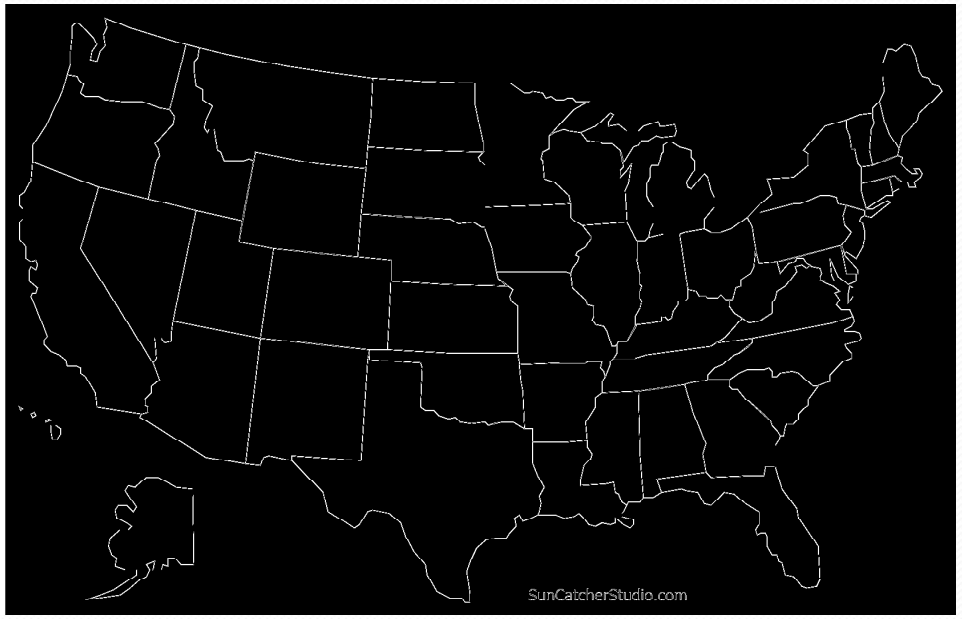
Filtro laplaciano

- De forma equivalente se definen los kerneles laplacianos negativos.
- $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ y $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
- OpenCV: cv.Laplacian.

Ejemplo

- Archivo: laplacian_filter.py.



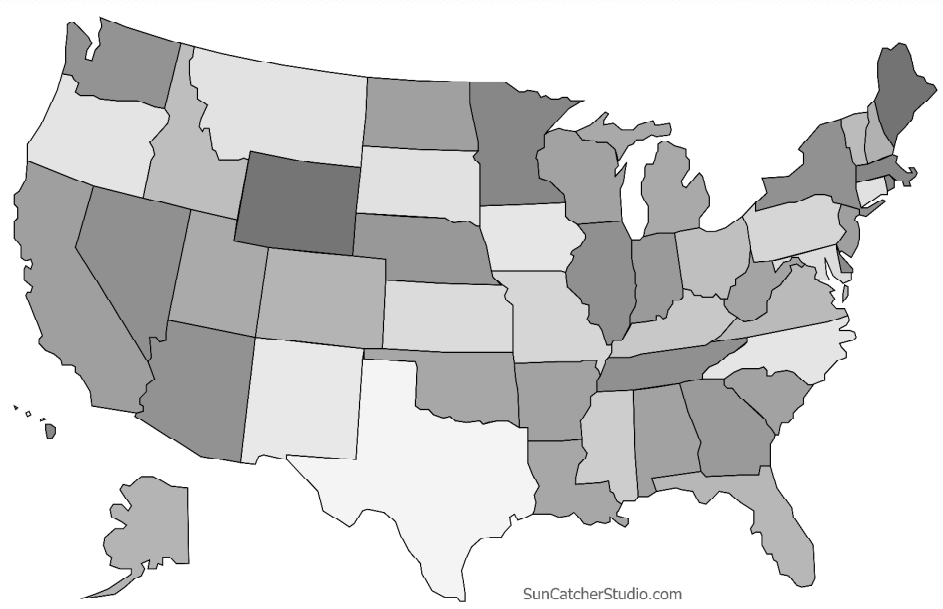


Laplaciano del gaussiano

- El laplaciano es muy sensible al ruido.
- Para contrarrestar la sensibilidad se puede aplicar primero un filtro gaussiano.
- A esto se le conoce como laplaciano del gaussiano (LoG).

Ejemplo

- Archivo: laplacian_of_Gaussian.py.



Laplaciano vs LoG

Image Sharpening

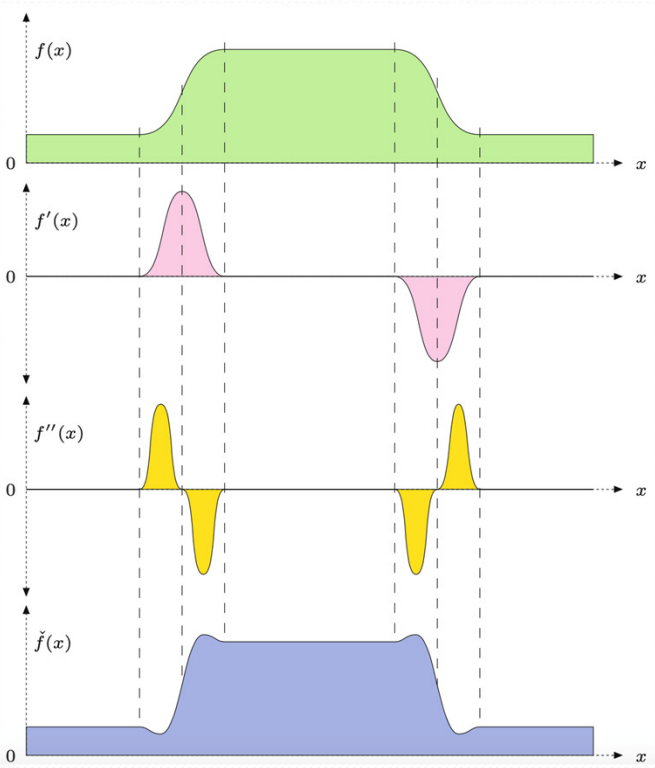
- El objetivo es aumentar la nitidez aparente de una imagen.
- Dos métodos para image sharpening:

1. Usar el laplaciano o el laplaciano del gaussiano.
2. Unsharp masking (USM).

Sharpening con el laplaciano

- Primero se aplica un filtro laplaciano H^L a la imagen I.
- Luego, a la imagen original se le resta una fracción del resultado.
- $I' \leftarrow I - w \cdot (H^L \star I)$
- El factor w, entre 0 y 1, especifica la fuerza del sharpening.
- Para eliminar algo de ruido, antes de aplicar el laplaciano, se puede aplicar un filtro gaussiano o de mediana.

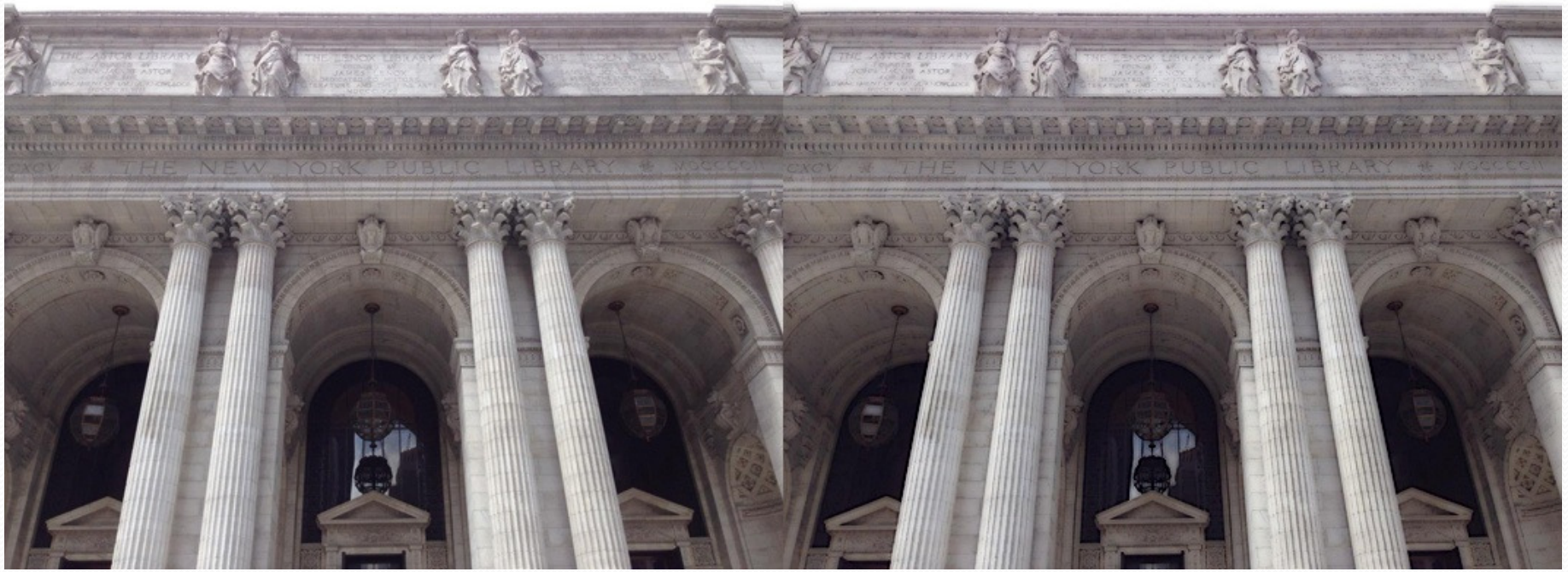
Sharpening con el laplaciano



Fuente: DIP-Java, p. 135

Ejemplo

- Archivo: laplacian_of_gaussian_sharpening.py.



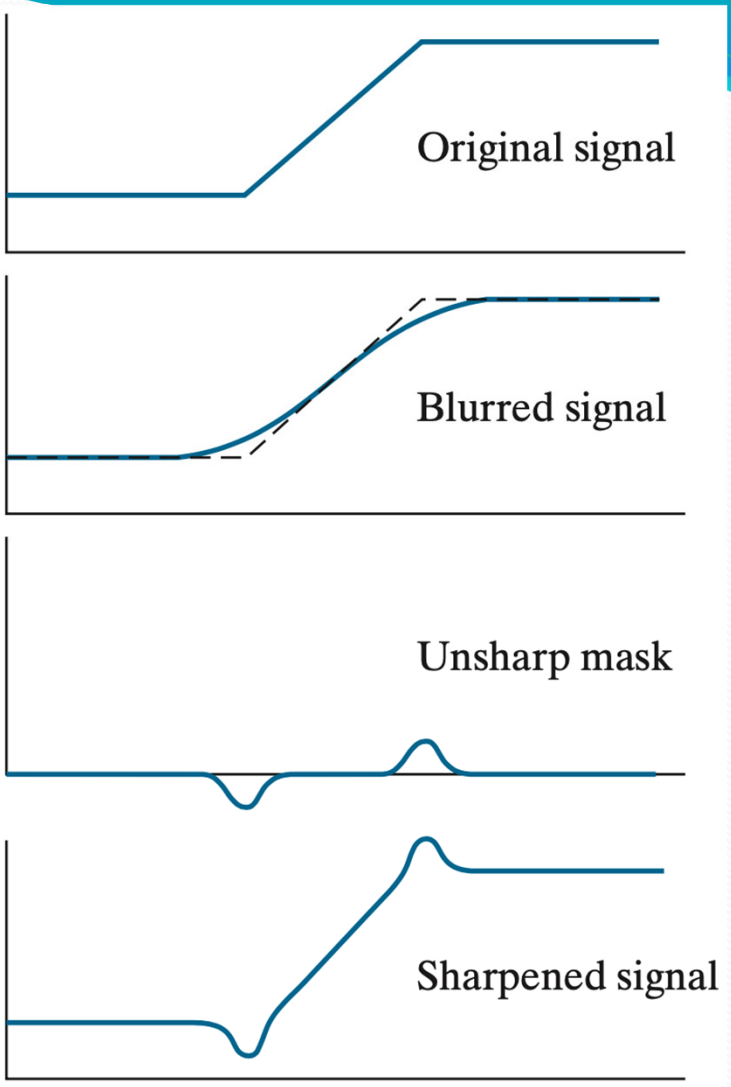
Unsharp masking

- Abreviado como USM.
- Paso 1. Restar una versión suavizada de la imagen del original.
- El resultado se llama máscara (mask).
- $M \leftarrow I - \left(I \star \widetilde{H}\right) = I - \tilde{I}$
- El kernel $\widetilde{H}$ del filtro de suavizado debe estar normalizado.
- Paso 2. Una parte de la máscara se agrega nuevamente al original.
- $\check{I} \leftarrow I + a \cdot M$
- Donde a es un factor.

Unsharp masking

- Otra forma de expresarlo:
- $\tilde{I} = \left(I \star \widetilde{H}\right)$
- $M = I - \tilde{I}$
- $\check{I} = I + a \cdot M$
- Cómo como $M = I - \tilde{I}$:
- $\check{I} = I + a \cdot (I - \tilde{I})$
- $\check{I} = (1 + a) \cdot I - a \cdot \tilde{I}$
- Valores típicos para la σ del filtro $\widetilde{H}$ son de 1 a 10 y para a de -2 a 2.

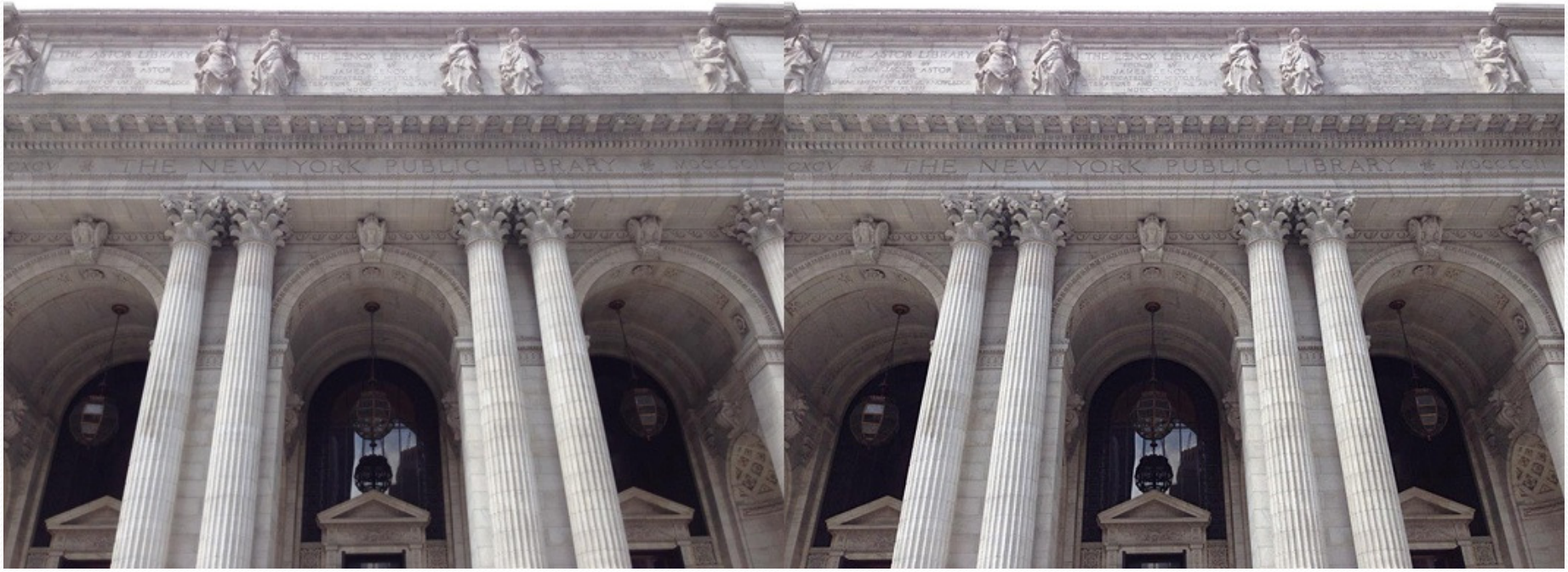
Unsharp masking



Fuente: DIP4, p. 183

Ejemplo

- Archivo: unsharp_mask.py.



Filtro high boost

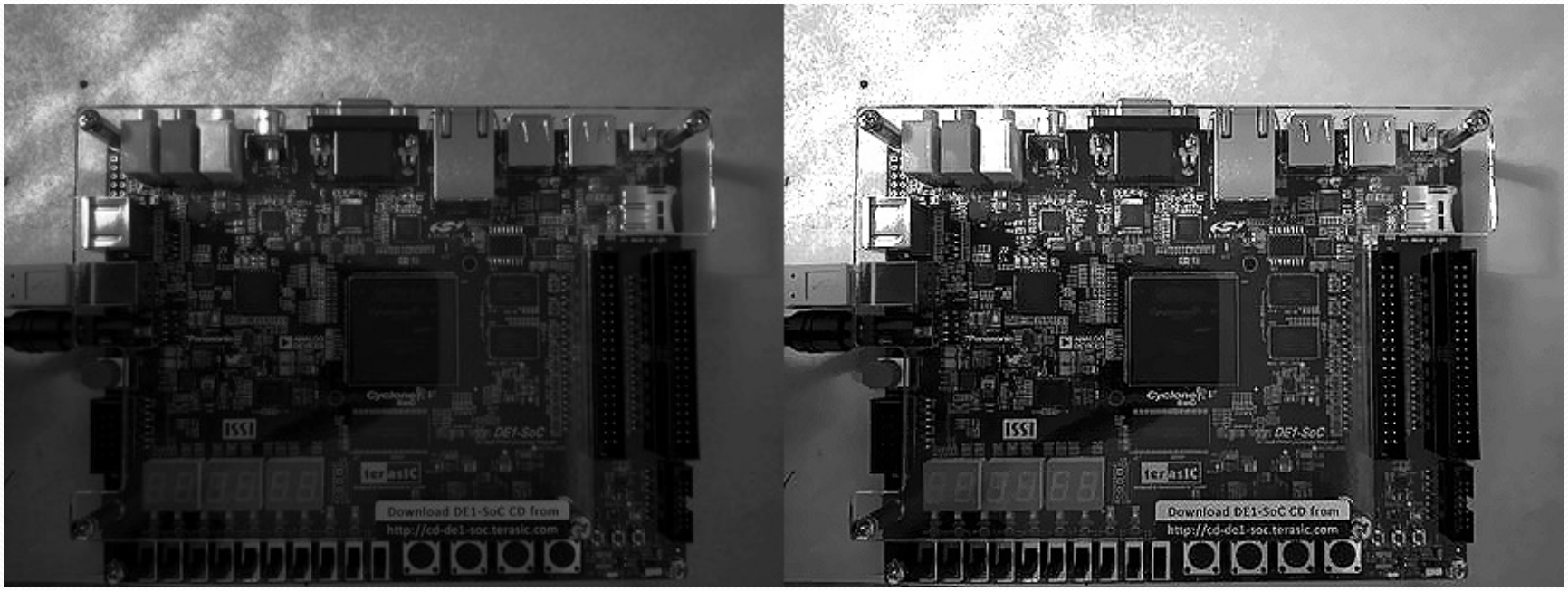
- Es una variante del filtro laplaciano.
- Se usa para ampliar (boost) los componentes de alta frecuencia de una imagen.
- Dos kerneles para high boost:
- $$\begin{bmatrix} 0 & -1 & 0 \\ -1 & \boldsymbol{w_1} & -1 \\ 0 & -1 & 0 \end{bmatrix}$$
- Donde $w_1 = A + 4$ y A es el factor de boost.

Filtro high boost

- $\begin{bmatrix} -1 & -1 & -1 \\ -1 & \boldsymbol{w_2} & -1 \\ -1 & -1 & -1 \end{bmatrix}$
- Donde $w_2 = A + 8$ y A es el factor de boost.

Ejemplo

- Archivo: high_boost.py.



Operador de Canny

- Es considerado el estado del arte en detección de bordes.
- Reduce el ruido con un filtro gaussiano.
- Utiliza el operador de Sobel para calcular el gradiente.
- Elimina pixeles que no se consideran parte de un borde.
- Utiliza dos umbrales inferior y superior:

a) Si el gradiente del pixel es superior al umbral superior, se acepta como borde.
b) Si el gradiente del pixel es menor al umbral inferior, se rechaza.
c) En otro caso, el pixel se acepta como borde solo si está conectado a un pixel que es parte de un borde.

Operador de Canny

- Se recomienda una proporción entre los umbrales de entre 2:1 y 3:1.
- Implementado en OpenCV mediante la función cv.Canny.

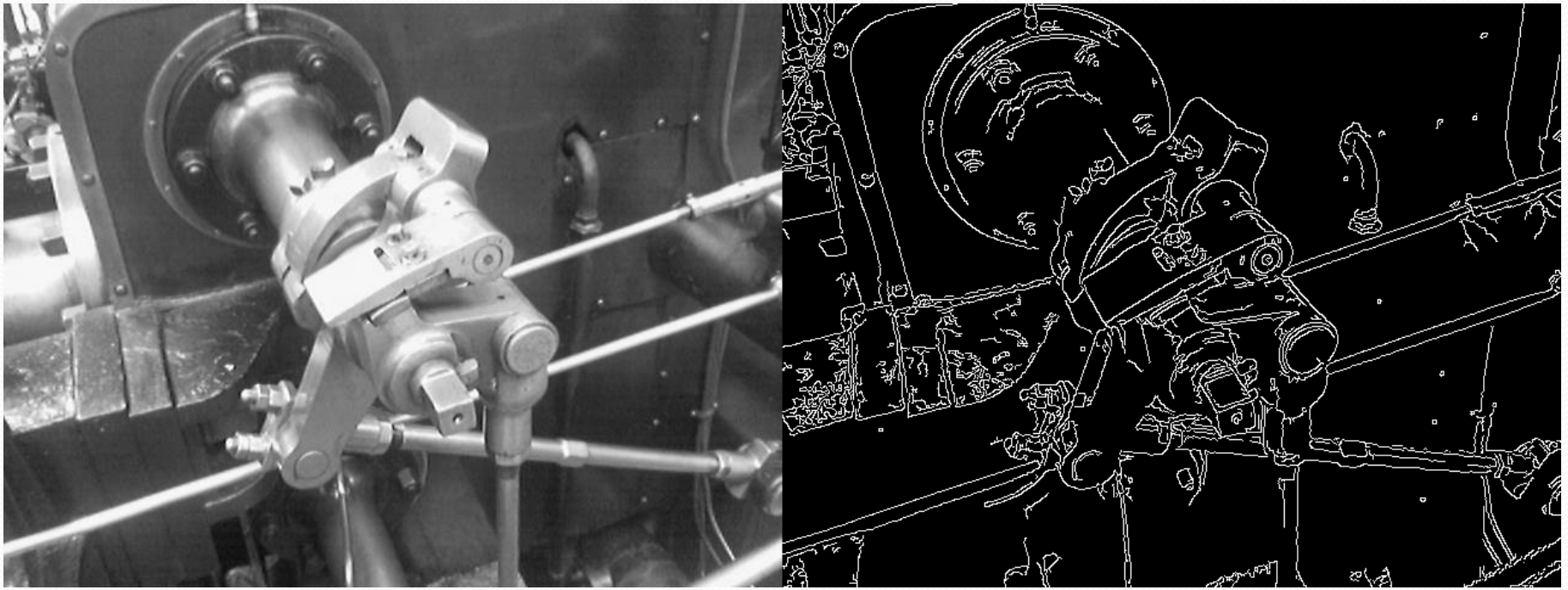
Ejemplo

```
import cv2 as cv
import numpy as np

image = cv.imread('valve_original.png', cv.IMREAD_GRAYSCALE)
edges = cv.Canny(image, 50, 170)
result = np.hstack((image, edges))
cv.imwrite('valve_original_canny.png', result)
```

Ejemplo

- Archivo: canny.py.



LoG vs Canny