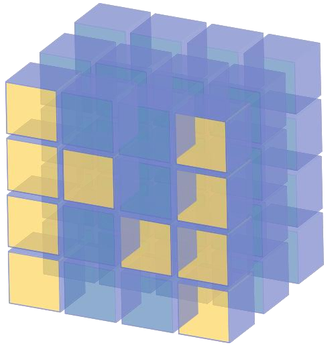
PROCESAMIENTO DIGITAL DE IMAGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Módulos principales



NumPy

matplotlib

OpenCV

pip install opencv-python

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

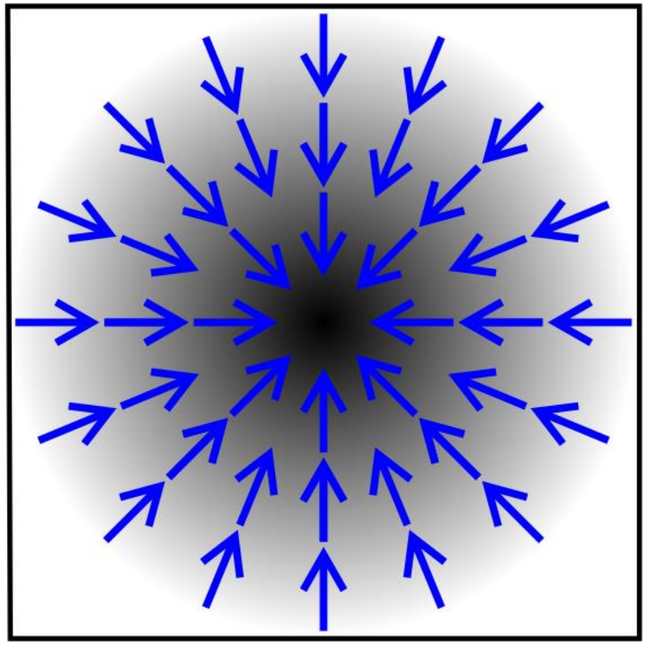
TEMAS QUE VEREMOS HOY

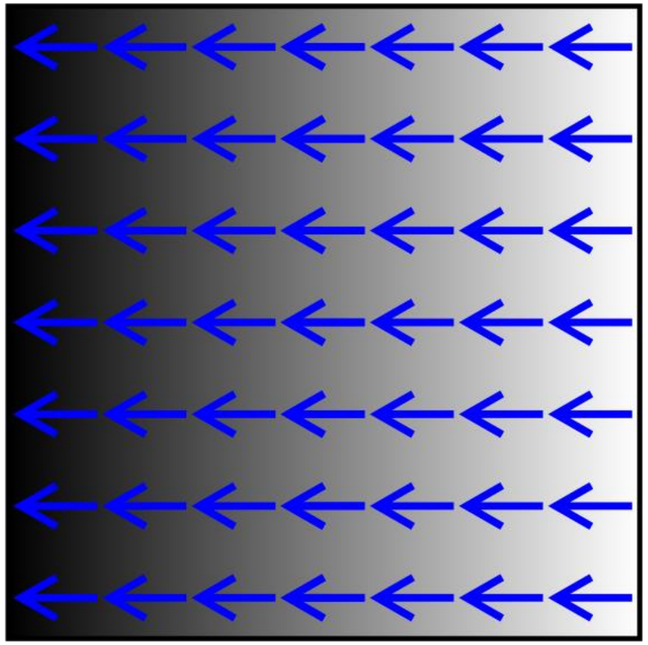
- Gradiente de imágenes
- Derivadas Sobel y Scharr
- Derivadas Laplacianas
- Algoritmo de Canny
- Reducción de ruido
- Encontrando el gradiente de intensidad de la imagen
- Supresión de falsos máximos
- Umbral de histéresis
- Algoritmo de Canny en OpenCV

PROCESAMIENTO DIGITAL DE IMÁGENES

GRADIENTE

- Indica la dirección en la cual varía más rápido un campo
- Representado vectorialmente
- Su módulo representa el ritmo de variación del campo en la dirección del vector gradiente

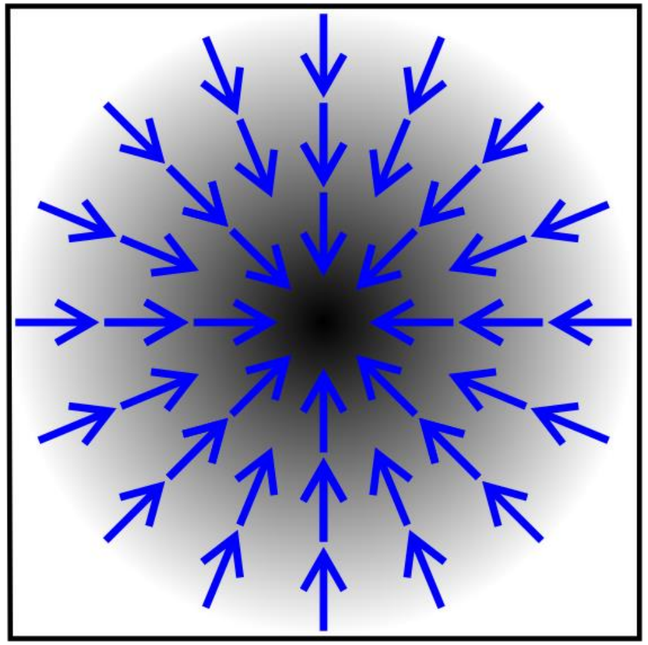


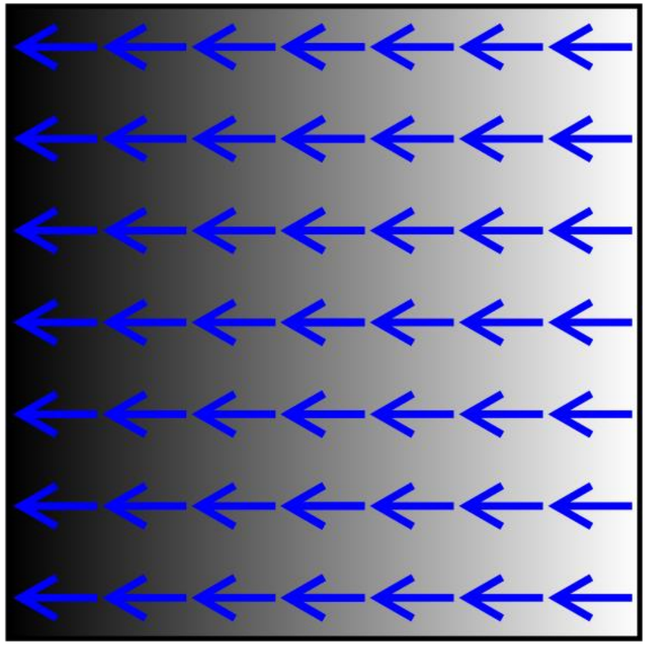


PROCESAMIENTO DIGITAL DE IMÁGENES

GRADIENTE

- Representa la pendiente de la línea tangente a la gráfica de una función
- Apunta a los puntos de la gráfica a los cuales la gráfica tiene un mayor incremento
- La magnitud del gradiente es la pendiente de la gráfica en esa dirección



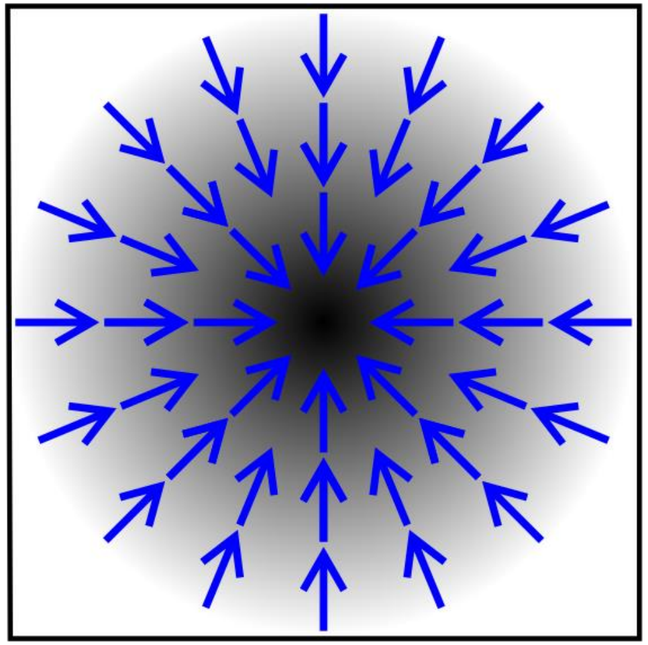


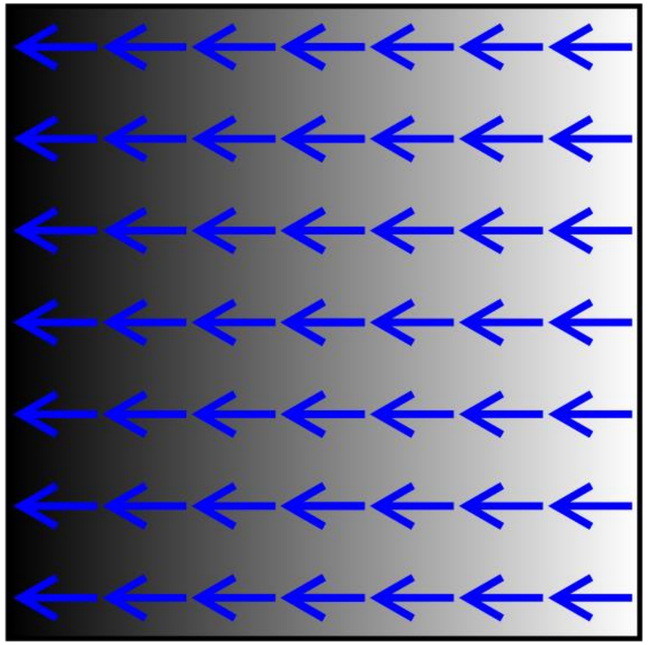
PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

GRADIENTE DE IMÁGENES

- Determina cómo una imagen cambia en términos de color o intensidad
- La magnitud del gradiente indica la rapidez con la que la imagen está cambiando
- La dirección del gradiente indica la dirección en la que la imagen está cambiando más rápidamente

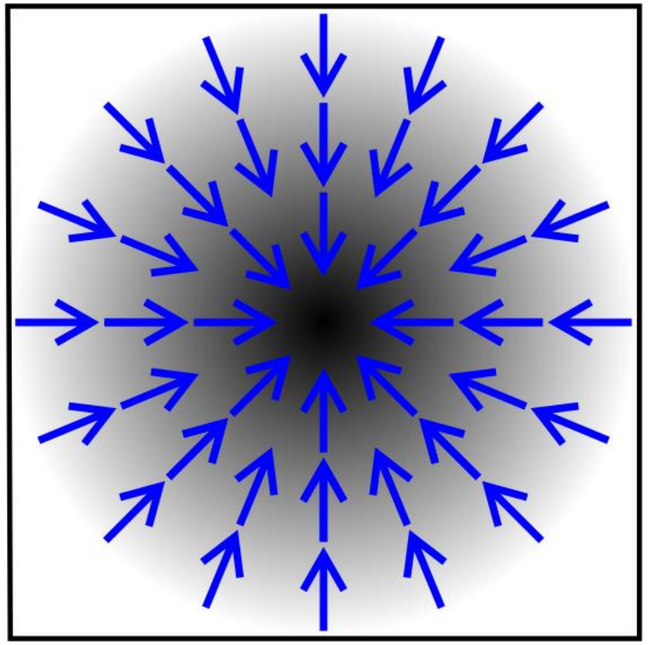


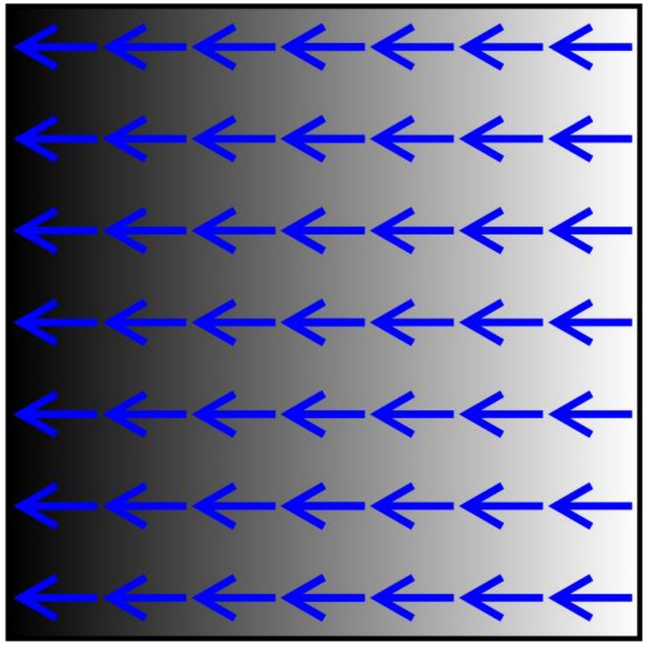


PROCESAMIENTO DIGITAL DE IMÁGENES

GRADIENTE DE IMÁGENES

- Las imágenes se pueden denotar mediante los ejes X e Y
- El gradiente es definido por las derivadas parciales de una función dada
- La función a usar en las imágenes es la intensidad
- El gradiente se aplica a lo largo de las direcciones X e Y



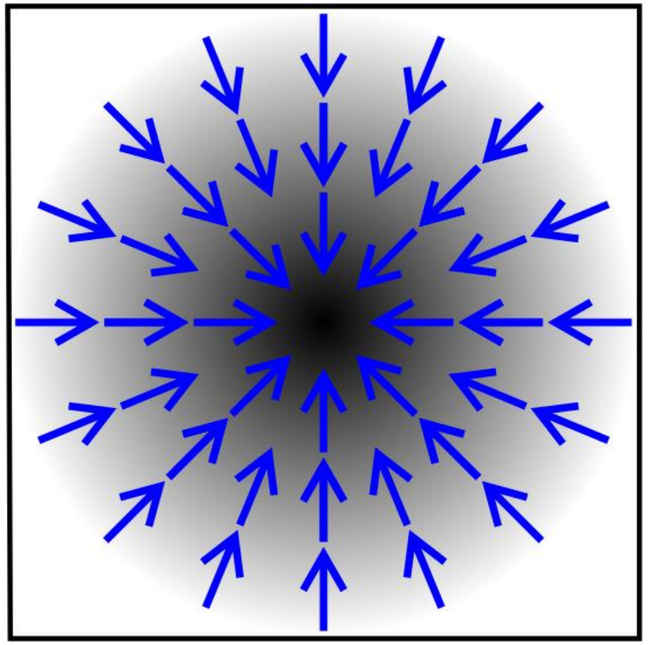


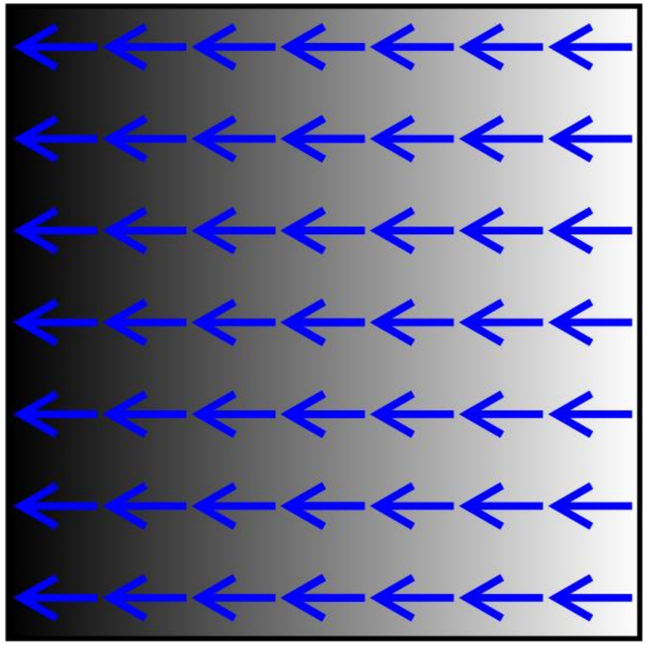
PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

GRADIENTE DE IMÁGENES

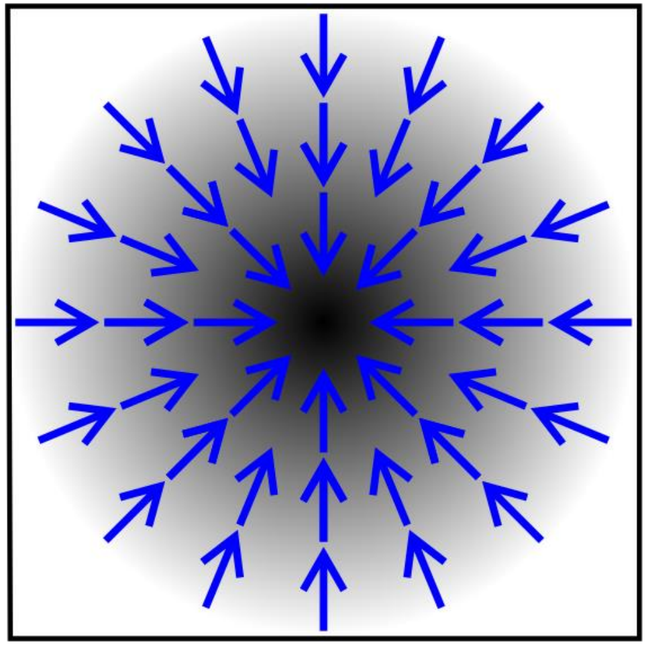
- Aquellos puntos donde la derivada es máxima, o mayor a cierto umbral, representan cambios de intensidad grandes
- Estos cambios de intensidad grandes normalmente están asociados a los bordes de los objetos en la imagen

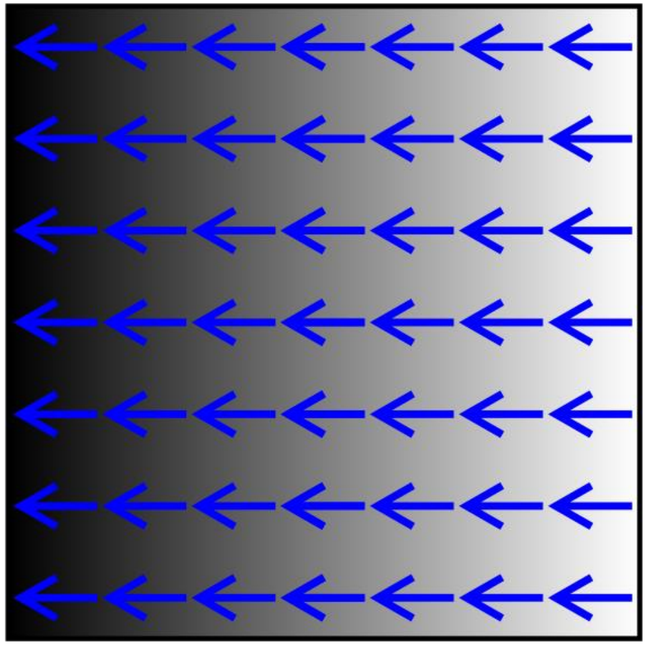




GRADIENTE DE IMÁGENES

- Debido que los cambios de intensidad grandes suelen estar asociados a los bordes, el gradiente de imágenes es útil para encontrar los bordes de los objetos en una imagen
- Los filtros pasa altos (filtros de gradientes) son: Sobel, Scharr y Laplaciano

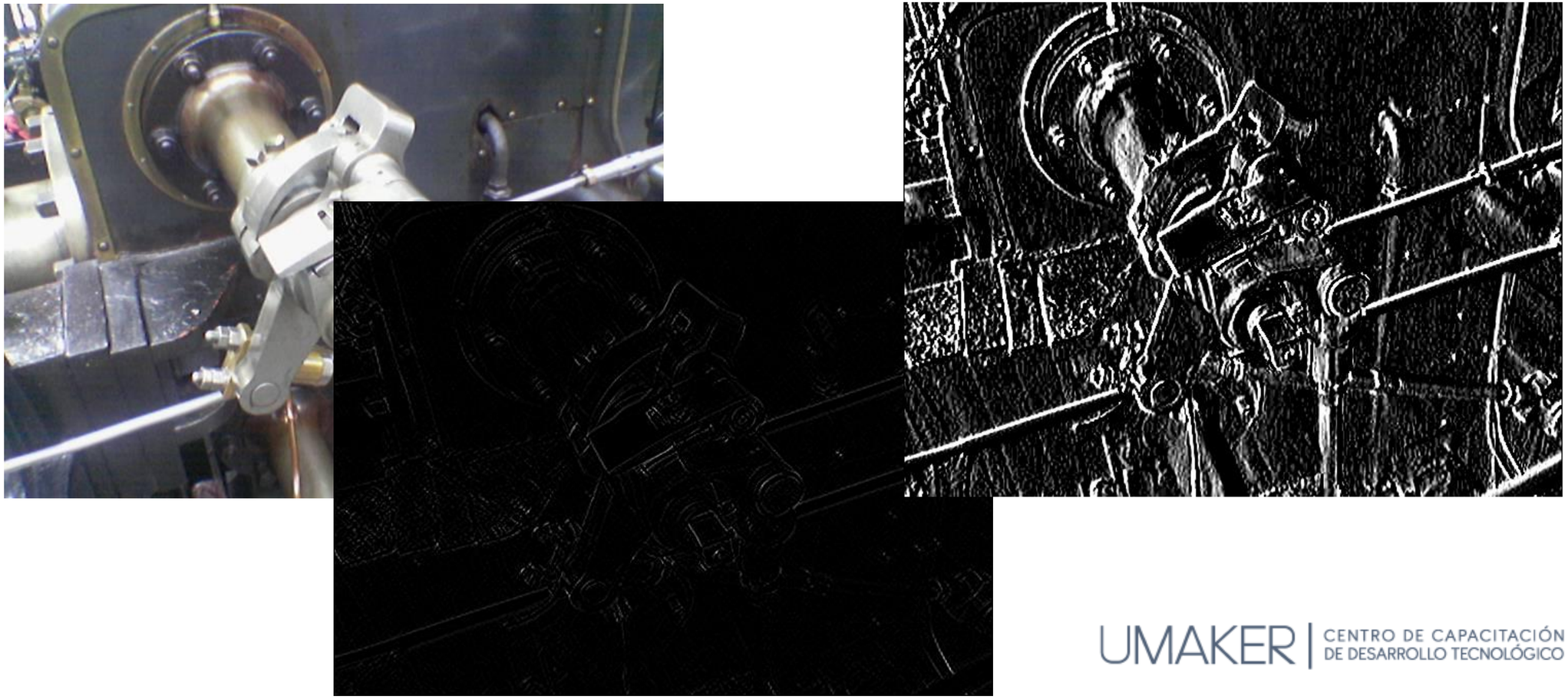




PROCESAMIENTO DIGITAL DE IMÁGENES

DERIVADAS SOBEL Y SCHARR

- Son aproximaciones que nos permiten calcular el gradiente de una imagen
- Al igual que en los filtros pasa bajos, se usa kernels cuadrados con el objetivo de definir estas derivadas
- Se aplica en cada punto (píxel) de la imagen

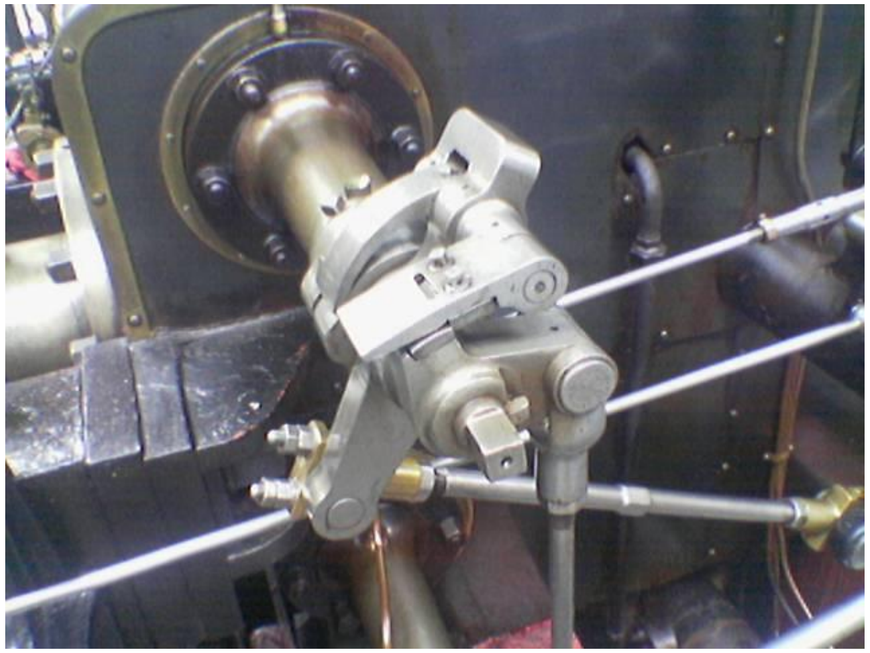


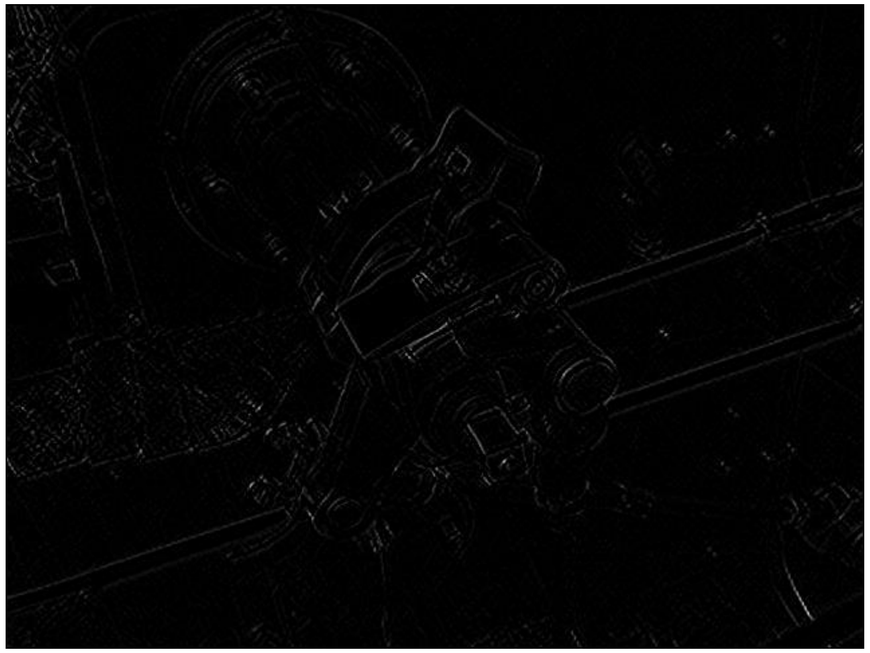
PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

DERIVADA SOBEL

- La derivada/operador Sobel estima las derivadas parciales a lo largo del eje X e Y
- Calcula el gradiente de la intensidad de una imagen en cada píxel
- Para cada píxel, devuelve la magnitud del mayor cambio posible junto a la dirección y sentido del cambio





PROCESAMIENTO DIGITAL DE IMÁGENES

DERIVADA SOBEL

- Es una operación conjunta de suavizado gaussiano más la diferenciación, lo que le permite ser más resistente al ruido
- Debido que el operador puede ser aplicado sobre el eje X e Y, se usa 2 kernels para aplicar una convolución a la imagen, con el fin de obtener las derivadas
- Un kernel se usa para los cambios horizontales y otro para los verticales

A: Imagen original

B: Imagen modificada

K: Kernel

$$B_x = K_x * A \qquad B_y = K_y * A$$

PROCESAMIENTO DIGITAL DE IMÁGENES

DERIVADA SOBEL

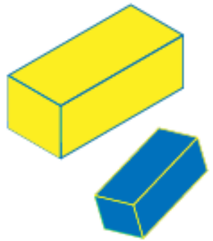
- Es una operación conjunta de suavizado gaussiano más la diferenciación, lo que le permite ser más resistente al ruido
- Debido que el operador puede ser aplicado sobre el eje X e Y, se usa 2 kernels para aplicar una convolución a la imagen, con el fin de obtener las derivadas
- Un kernel se usa para los cambios horizontales y otro para los verticales

A: Imagen original
B: Imagen modificada

$$B_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \qquad B_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A$$

$$B = \sqrt{B_x^2 + B_y^2} \quad \text{ó} \quad B = |B_x| + |B_y|$$

Aplicar operador Sobel a una imagen



Sobel : Aplica el operador Sobel a una imagen

Sintaxis: cv2.Sobel(imagen, profundidad, dx, dy, kernel)

Parametros:
imagen : Imagen a la cual aplicar la operación
profundidad: Es la profundidad deseada para la imagen resultante
dx : Orden de la derivada en el eje X
dy : Orden de la derivada en el eje Y
kernel: Tamaño del kernel a aplicar en la imagen

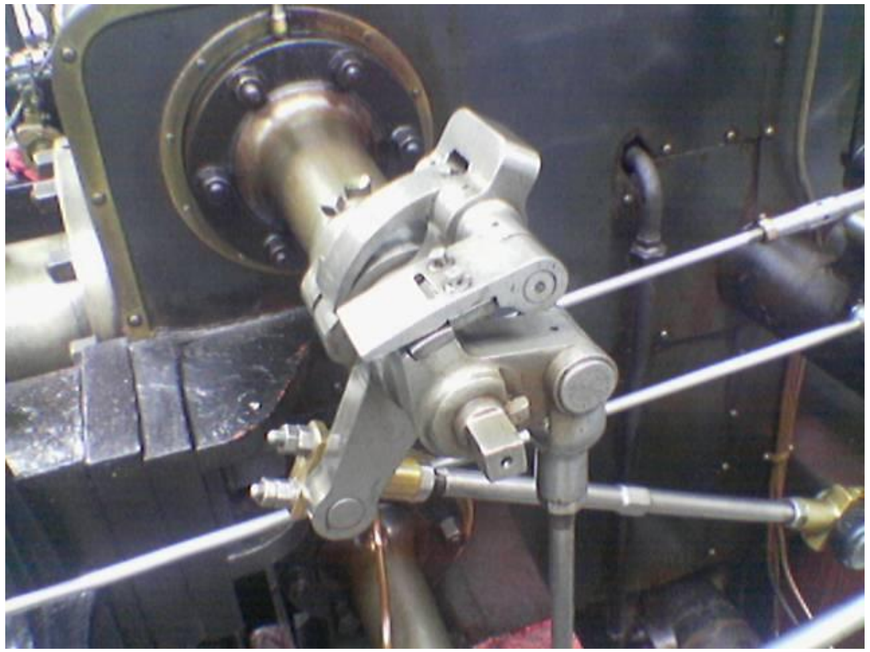
Nota Para imágenes de 8 bits, el resultado serán derivadas truncadas. El tamaño del kernel debe ser números enteros positivos e impares.

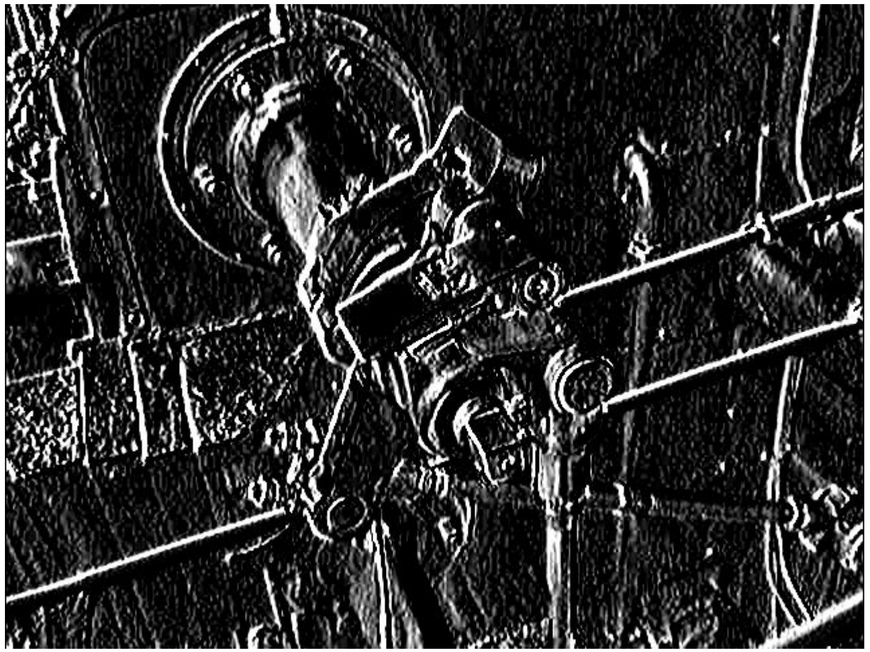
Ejemplo:

```python
import cv2
import matplotlib.pyplot as plt

image = cv2.imread("image.png")
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

dx = 1
dy = 0
tamaño_kernel = 3
image_with_sobel = cv2.Sobel(image, cv2.CV_64F, dx, dy, tamaño_kernel)

plt.subplot(1, 2, 1)
plt.title("Imagen")
plt.imshow(image, cmap="gray")
plt.xticks([])
plt.yticks([])

plt.subplot(1, 2, 2)
plt.title("Imagen con Sobel")
plt.imshow(image_with_sobel, cmap="gray")
plt.xticks([])
plt.yticks([])

plt.show()
```

DERIVADA SCHARR

- Este operador es el resultado de una optimización que minimiza el error angular cuadrático medio ponderado en el dominio de Fourier
- La condición para aplicar esta optimización es que los filtros resultantes sean numéricamente más consistentes





PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

DERIVADA SCHARR

- Al igual que el operador Sobel, se utilizan 2 kernels para aplicar una convolución a la imagen, con el fin de obtener las derivadas
- Los kernels usados son:

A: Imagen original

B: Imagen modificada

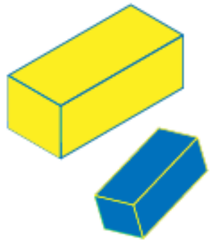
$$B_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix} * A \qquad B_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix} * A$$

$$B = \sqrt{B_x^2 + B_y^2} \quad \text{ó} \quad B = |B_x| + |B_y|$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Aplicar operador Scharr a una imagen



Scharr : Aplica el operador Scharr a una imagen

Sintaxis: cv2.Scharr(imagen, profundidad, dx, dy)

Parametros:
imagen : Imagen a la cual aplicar la operación
profundidad: Es la profundidad deseada para la imagen resultante
dx : Orden de la derivada en el eje X
dy : Orden de la derivada en el eje Y

Nota Si en el método «Sobel», el parámetro «kernel» es igual a -1, entonces se aplica el operador Scharr.

PROCESAMIENTO DIGITAL DE IMÁGENES

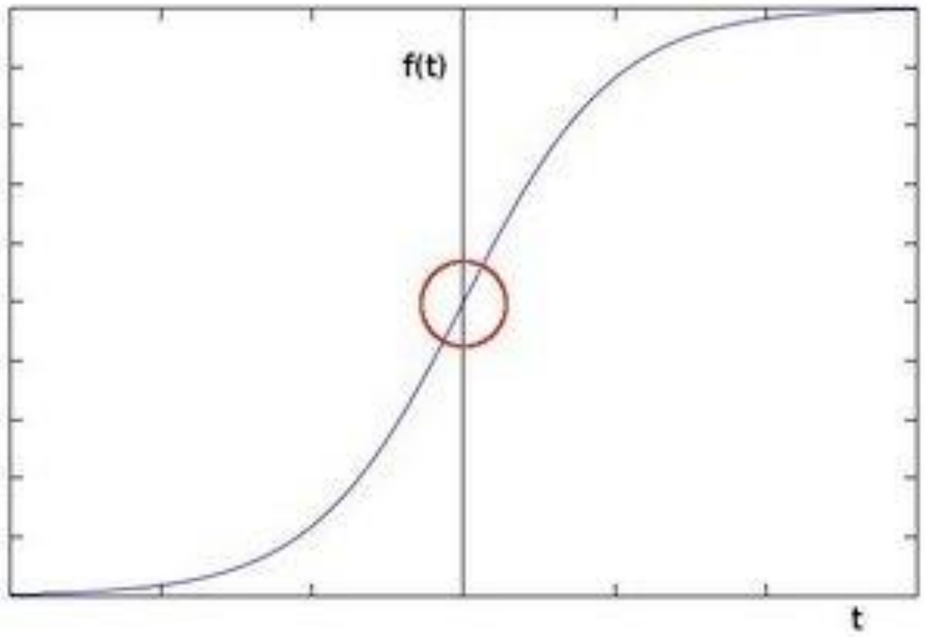
UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

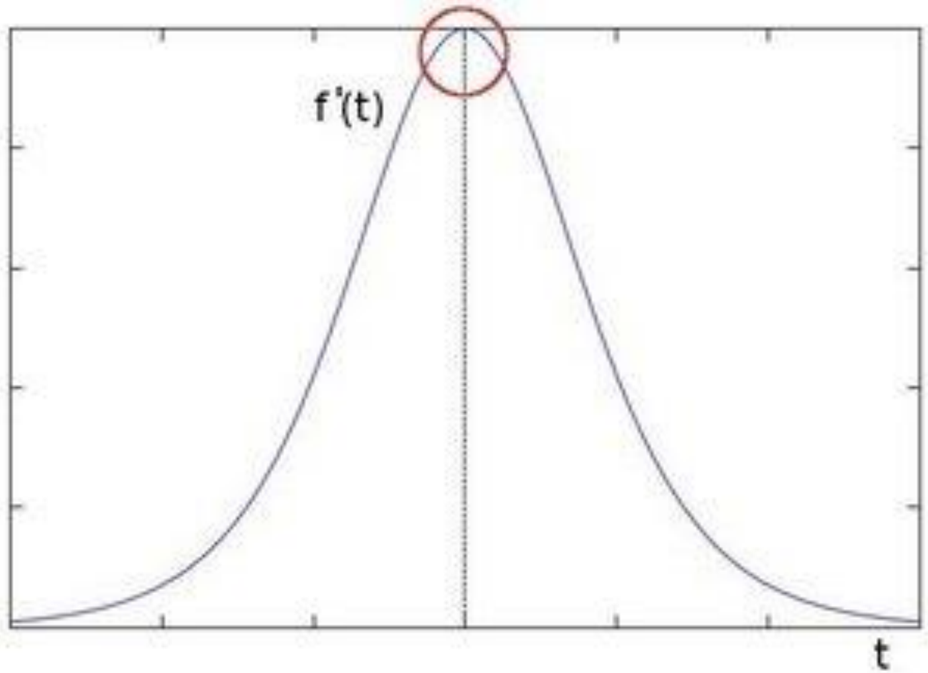
Ejemplo:

```python
import cv2
import matplotlib.pyplot as plt

image = cv2.imread("image.png")
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

dx = 1
dy = 0
image_with_scharr = cv2.Scharr(image, cv2.CV_64F, dx, dy)

plt.subplot(1, 2, 1)
plt.title("Imagen")
plt.imshow(image, cmap="gray")
plt.xticks([])
plt.yticks([])

plt.subplot(1, 2, 2)
plt.title("Imagen con Scharr")
plt.imshow(image_with_scharr, cmap="gray")
plt.xticks([])
plt.yticks([])

plt.show()
```

DERIVADA LAPLACIANA

- El operador Sobel se basa en el hecho que en el área del borde, la intensidad de los píxeles muestra una variación de intensidad (un salto)
- Para conseguir esto, se obtiene la primera derivada de la intensidad, tal que se puede apreciar que un borde se caracteriza por un máximo:

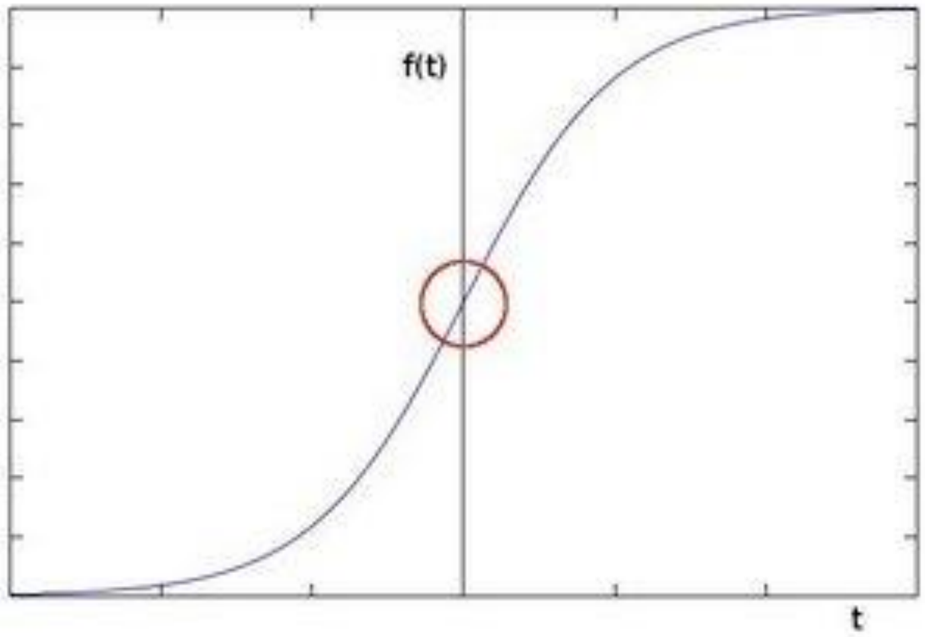


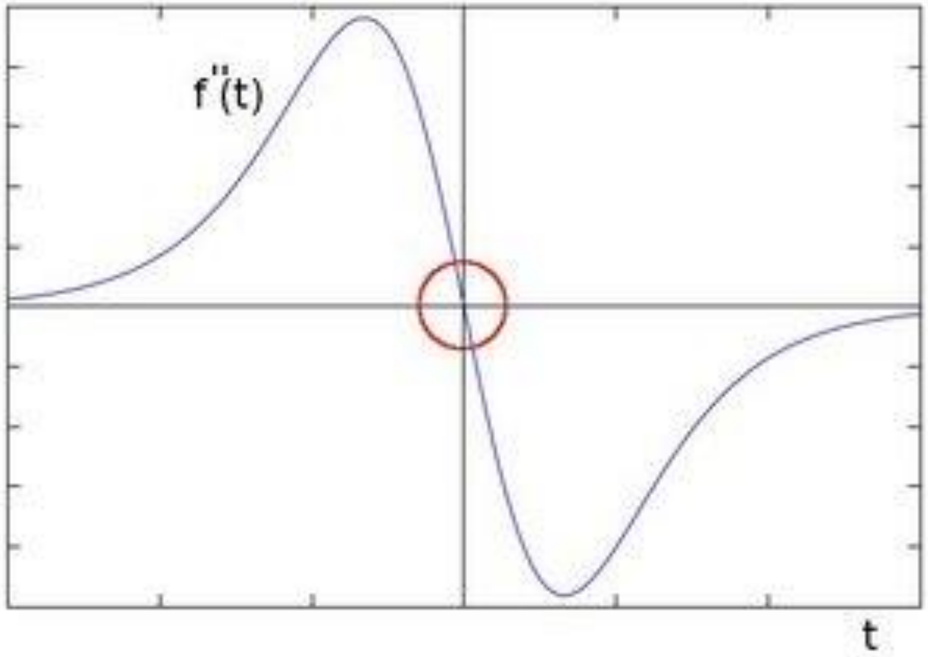


PROCESAMIENTO DIGITAL DE IMÁGENES

DERIVADA LAPLACIANA

- Por otro lado, si obtenemos la segunda derivada de la intensidad, podremos observar que el valor es cero
- Por ende, este criterio también puede ser usado para detectar los bordes. Sin embargo, los valores cero no sólo aparecen en los bordes puesto que pueden aparecer en otras ubicaciones, lo cual puede ser solucionado aplicando filtros donde se requiera

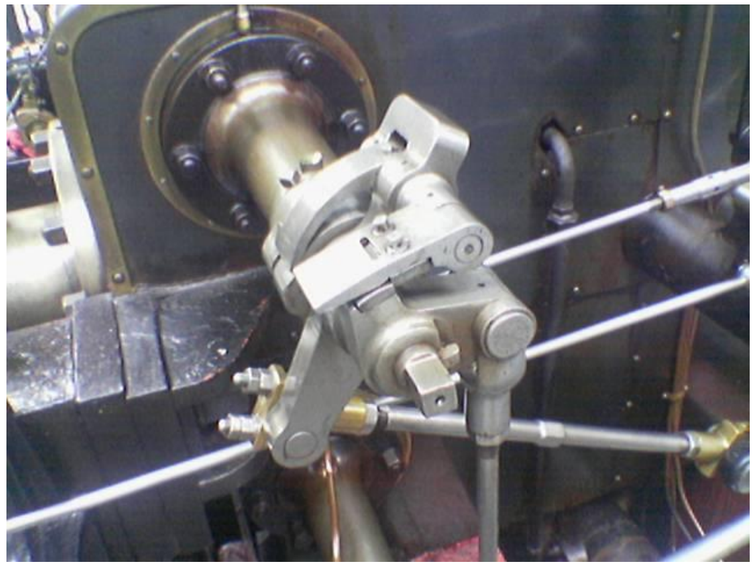


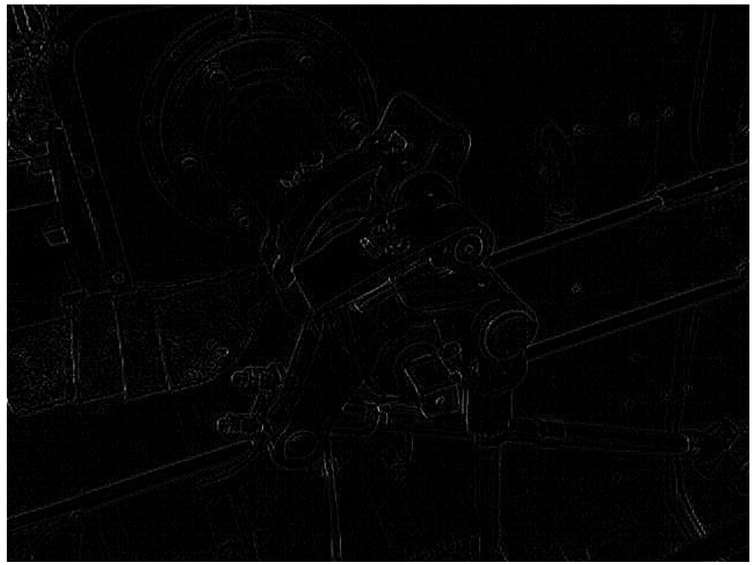


DERIVADA LAPLACIANA

- Debido que las imágenes están en 2 dimensiones, necesitamos tomar la derivada en ambas direcciones (eje X e Y)
- El operador Laplaciano está definido como:

$$Laplace(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$





Aplicar operador Laplaciano a una imagen

Laplacian : Aplica el operador Laplaciano a una imagen

Sintaxis: cv2.Laplacian(imagen, profundidad[, nueva_imagen[, kernel]])

Parametros:
imagen : Imagen a la cual aplicar la operación
profundidad: Es la profundidad deseada para la imagen resultante
nueva_imagen : Imagen resultante, con el mismo tamaño que la imagen recibida
kernel: Tamaño del kernel a aplicar en la imagen

Nota Cuando el parámetro «kernel» es mayor a 1 se aplica la suma de las segundas derivadas de la intensidad. Pero si el parámetro «kernel» es igual a 1 entonces se usa la siguiente matriz para filtrar la imagen:

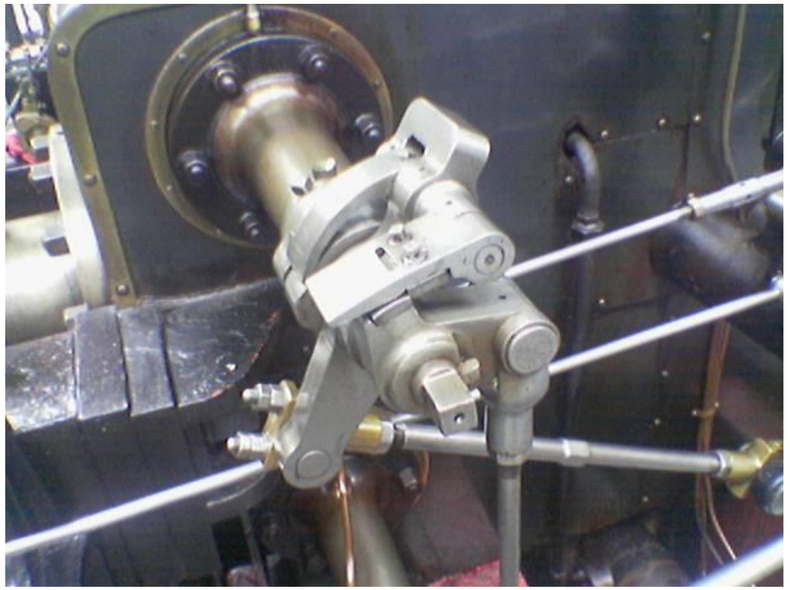
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

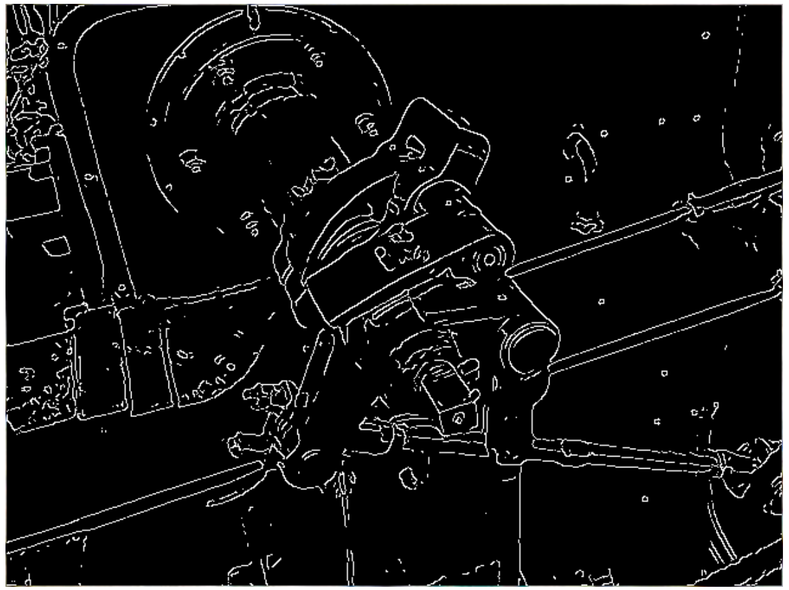
Ejemplo:

```python
import cv2
import matplotlib.pyplot as plt

image = cv2.imread("image.png")
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

tamaño_kernel = 1
image_with_laplace = cv2.Laplacian(image, cv2.CV_64F, ksize=tamaño_kernel)

plt.subplot(1, 2, 1)
plt.title("Imagen")
plt.imshow(image, cmap="gray")
plt.xticks([])
plt.yticks([])

plt.subplot(1, 2, 2)
plt.title("Imagen con Laplace")
plt.imshow(image_with_laplace, cmap="gray")
plt.xticks([])
plt.yticks([])

plt.show()
```

ALGORITMO DE CANNY

- Es un método de detección de bordes
- Es un algoritmo que utiliza múltiples etapas para detectar los bordes de los objetos en una imagen
- La detección inteligente de bordes es una técnica para extraer información estructural útil de diferentes objetos y reducir drásticamente la cantidad de datos a procesar

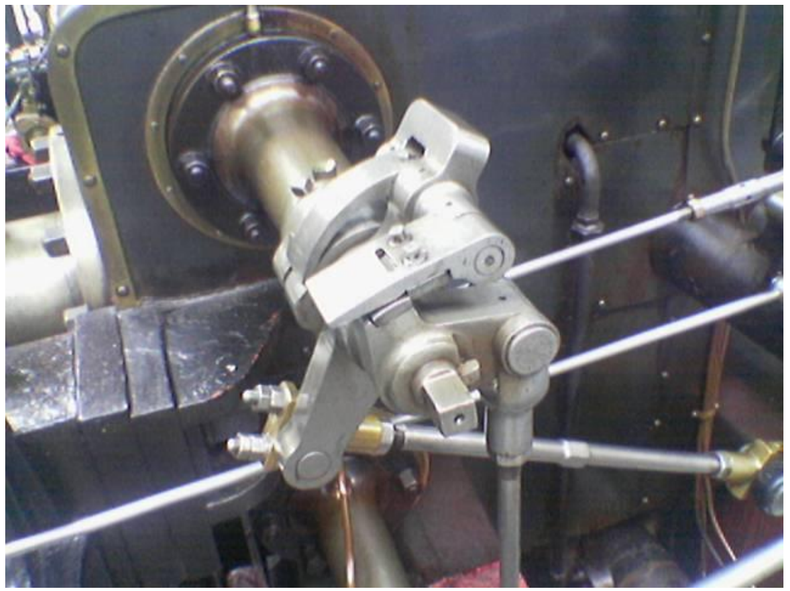


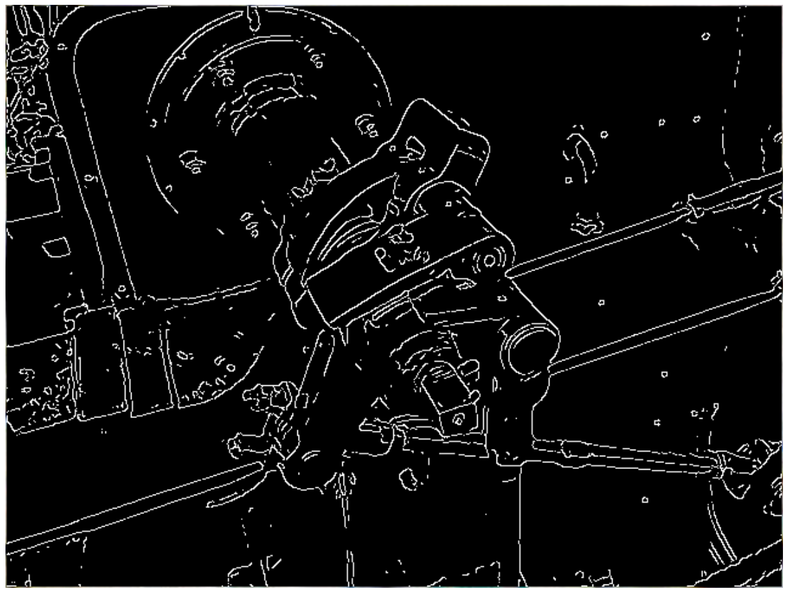


PROCESAMIENTO DIGITAL DE IMÁGENES

ALGORITMO DE CANNY

- Este algoritmo fue desarrollado para hacer una detección óptima de bordes
- Por ello, la detección debe capturar con precisión tantos bordes como sea posible en la imagen
- Además, el punto del borde detectado por el operador debe ubicarse con precisión en el centro del borde
- Finalmente, un borde determinado en la imagen, sólo debe marcarse una vez. Cuando sea posible, el ruido de la imagen no debe crear bordes falsos

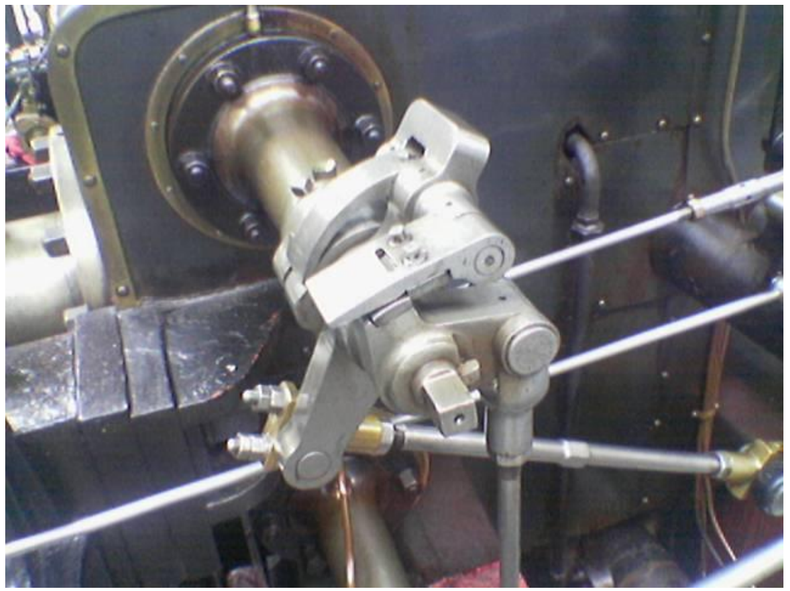


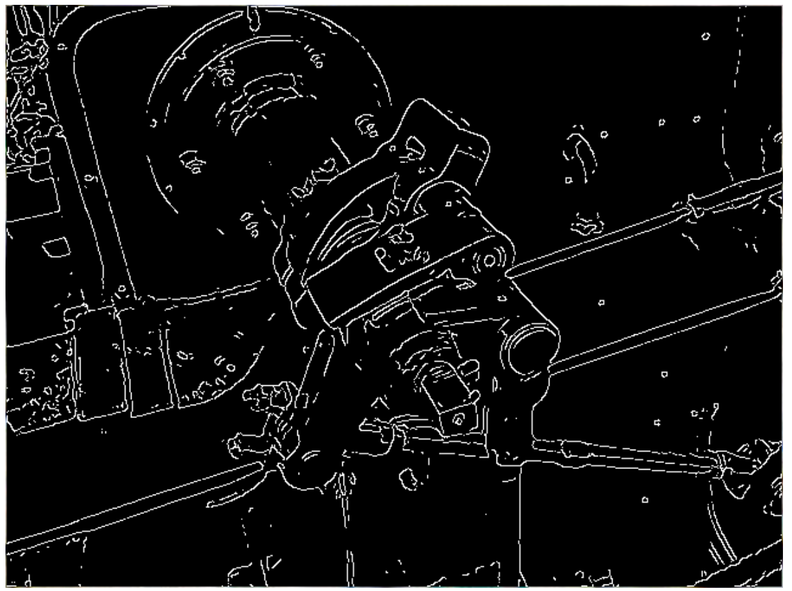


PROCESAMIENTO DIGITAL DE IMÁGENES

ALGORITMO DE CANNY

- Los criterios mencionados pueden ser simplificados a:
- Se requiere una buena detección, esto significa que el algoritmo debe marcar el mayor número real en los bordes de la imagen como sea posible
- Se necesita una buena localización, o sea que los bordes de marca deben estar lo más cerca posible del borde de la imagen real
- Es necesario una respuesta mínima, lo cual implica que el borde de la imagen sólo debe ser marcado una vez, y de ser posible, el ruido de la imagen no debe crear bordes falsos





ALGORITMO DE CANNY

- Este algoritmo consta de las siguientes etapas:

Reducción de ruido

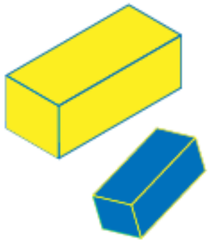
Encontrar el gradiente de la intensidad en la imagen

Supresión de falsos máximos

Umbral de histéresis

PROCESAMIENTO DIGITAL DE IMÁGENES

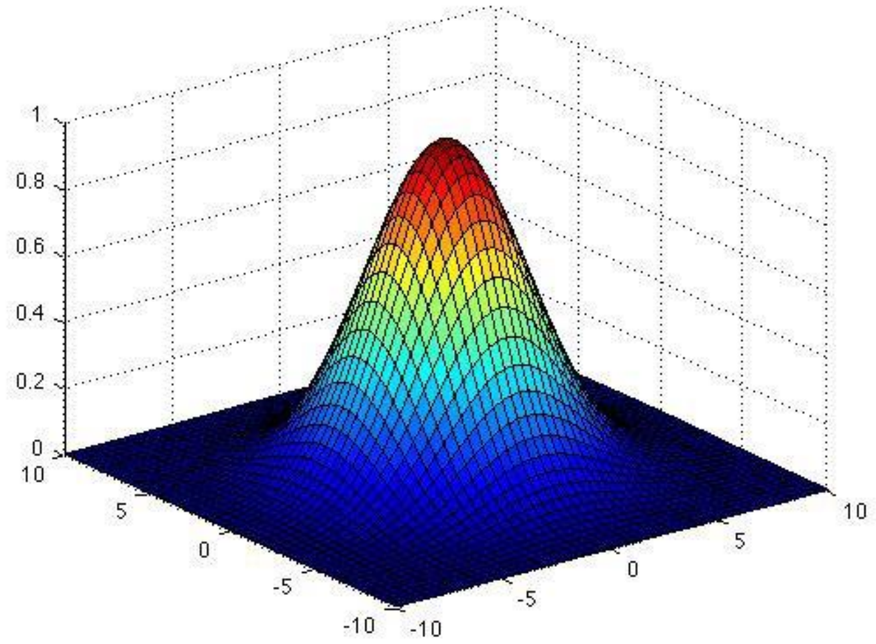
REDUCCIÓN DE RUIDO



- La detección de bordes es susceptible al ruido en la imagen
- Por ello, se debe eliminar o reducir el ruido lo más que se pueda
- Para este fin, se utiliza un filtro Gaussiano
- Normalmente, se aplica un kernel de tamaño 5x5
- Esto es con el fin de eliminar considerablemente el ruido
- Sin embargo, no se desea alisar demasiado la imagen
- Puesto que de hacerlo se perdería muchas características, al punto que complicaría obtener los bordes de la imagen

FILTRO GAUSSIANO

- No se utiliza un kernel con los coeficientes iguales (como en el caso del Promedio)
- Se utiliza un kernel gaussiano
- Cuanto más cerca se encuentre un píxel del píxel central, más afecta al promedio ponderado usado para establecer el nuevo valor del píxel central
- Se hace la asunción que los píxeles más cercanos al píxel central están más cerca del valor real de dicho píxel, razón por la cual influyen más

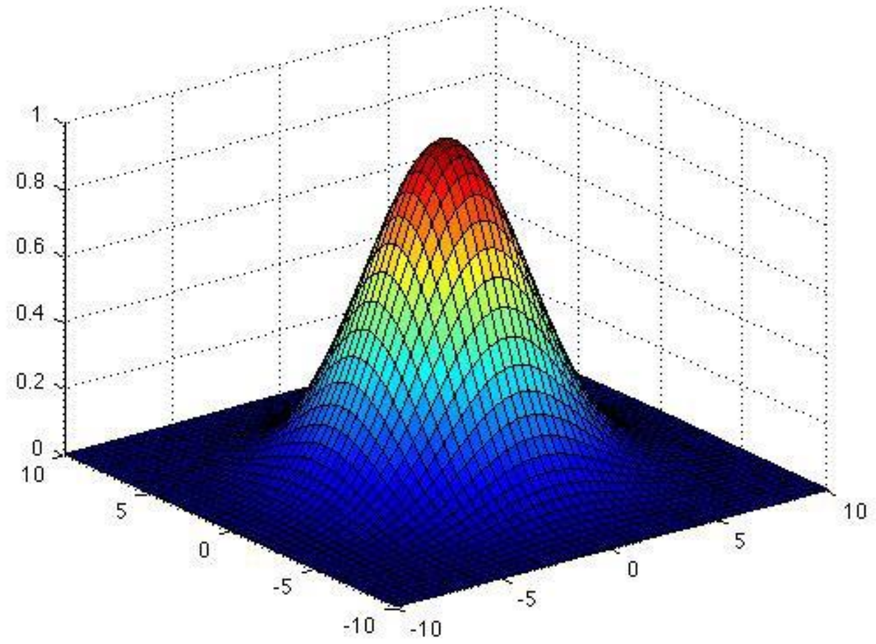


PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

FILTRO GAUSSIANO

- Representamos la desviación estándar mediante sigma
- Dado que se trabaja en 3 dimensiones, se tiene un valor sigma para el eje X e Y
- El valor de sigma para el eje X puede ser distinto que el del eje Y
- Sigma indica que tan ancha debe ser la curva dentro del kernel, similar a como sigma define el ancho de una curva de distribución normal



PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

REDUCCIÓN DE RUIDO

- Un ejemplo de un kernel Gaussiano de tamaño 5 es:

$$kernel = \frac{1}{159} * \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

ENCONTRANDO EL GRADIENTE DE INTENSIDAD DE LA IMAGEN

- Este paso se realiza luego que la imagen ha sido alisada mediante el filtro gaussiano
- Se calcula el gradiente de la imagen, para lo cual se aplica un filtro sobre la imagen
- El gradiente se obtiene mediante un operador Sobel, aplicado tanto en el eje X como en el eje Y, con el fin de obtener las respectivas imágenes resultado de los 2 kernels

A: Imagen original

B: Imagen modificada

$$B_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix} * A$$

$$B_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix} * A$$

ENCONTRANDO EL GRADIENTE DE INTENSIDAD DE LA IMAGEN

- A partir de las imágenes obtenidas con el operador Sobel, se puede obtener los bordes de la imagen
- Además, con las imágenes obtenidas también se puede obtener la dirección del gradiente en cada píxel de la imagen
- Bordes_de_la_imagen es el valor del gradiente en los bordes de los objetos de la imagen
- Ángulo indica la dirección perpendicular a los bordes

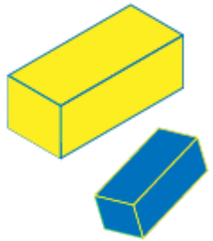
$$Bordes\_de\_la\_imagen = \sqrt{B_x^2 + B_y^2}$$

$$Ángulo = arctan(\frac{B_x}{B_y})$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

ENCONTRANDO EL GRADIENTE DE INTENSIDAD DE LA IMAGEN



- El valor de «Ángulo» es redondeado a 1 de 4 ángulos
- Un ángulo representa la dirección horizontal (0°)
- Otro ángulo representa la dirección vertical (90°)
- Otro ángulo representa una diagonal (45°)
- Otro ángulo representa otra diagonal (135°)

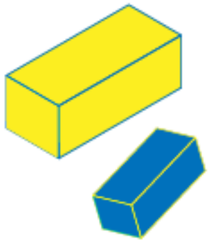
$$Bordes\_de\_la\_imagen = \sqrt{B_x^2 + B_y^2}$$

$$Ángulo = arctan(\frac{B_x}{B_y})$$

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

SUPRESIÓN DE FALSOS MÁXIMOS



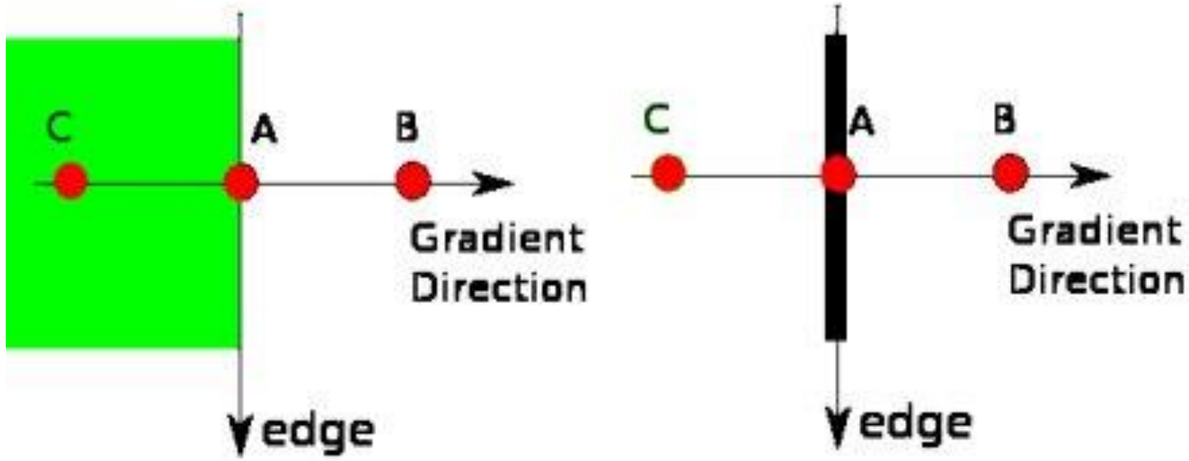
- Este paso se realiza luego de encontrar el gradiente de intensidad en la imagen
- Esta técnica se usa para afinar los bordes encontrados al aplicar la gradiente en el paso anterior
- Consiste en escanear la imagen para eliminar los píxeles que no formen parte de los bordes
- Con ese fin, se compara el valor de cada píxel con sus vecinos cercanos en la dirección del gradiente, la cual es perpendicular al borde
- En el caso que el valor del píxel sea mayor a los de su vecindad, entonces es considerado como un máximo local y por ende es aceptado por el algoritmo
- Caso contrario, si el valor del píxel no es un máximo local, entonces es suprimido

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

SUPRESIÓN DE FALSOS MÁXIMOS

- El píxel A está en el borde, en la dirección vertical
- La dirección del gradiente es normal al borde (edge)
- Los píxeles B y C están en las direcciones del gradiente
- El punto A se verifica con los puntos B y C para determinar si forma o no un máximo local. De ser un máximo local es considerado para la siguiente etapa; caso contrario, es suprimido
- Con esto, se obtiene una imagen binaria bordes muy finos



UMBRAL DE HISTÉRESIS

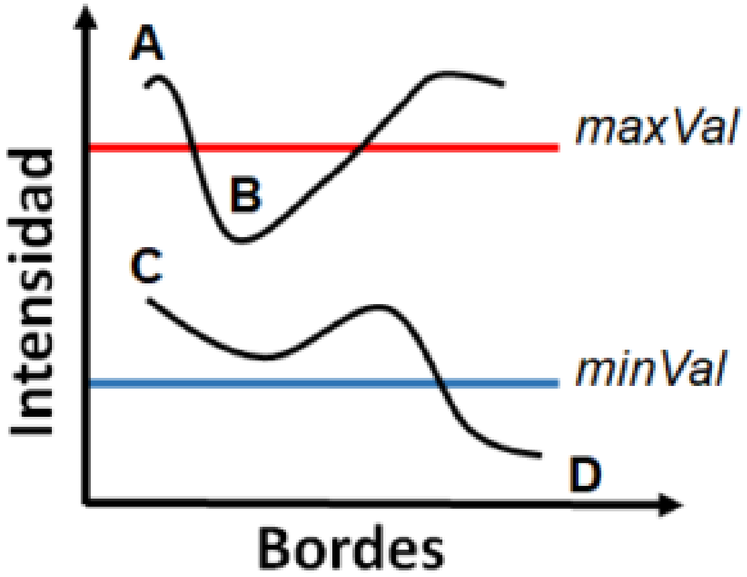
- Este paso se realiza luego de suprimir los falsos máximos
- Mediante la supresión de falsos máximos se consigue determinar, con bastante precisión, los píxeles que conforman los bordes. Sin embargo, aún pueden existir píxeles provenientes del ruido o de variaciones producto de los colores en la imagen
- En esta etapa se determina los píxeles que realmente pertenecen a los bordes y los que no
- Por ello, se debe establecer 2 valores umbrales, un mínimo y un máximo
- Aquellos píxeles con gradientes de intensidad mayores al valor umbral máximo son considerados como pertenecientes a los bordes
- Aquellos píxeles con gradientes de intensidad menores al valor umbral mínimo son descartados

UMBRAL DE HISTÉRESIS

- Los píxeles con gradientes de intensidad que se encuentren entre los 2 valores umbrales son clasificados como píxeles débiles

- Los píxeles débiles son o no aceptados, dependiendo de su conectividad

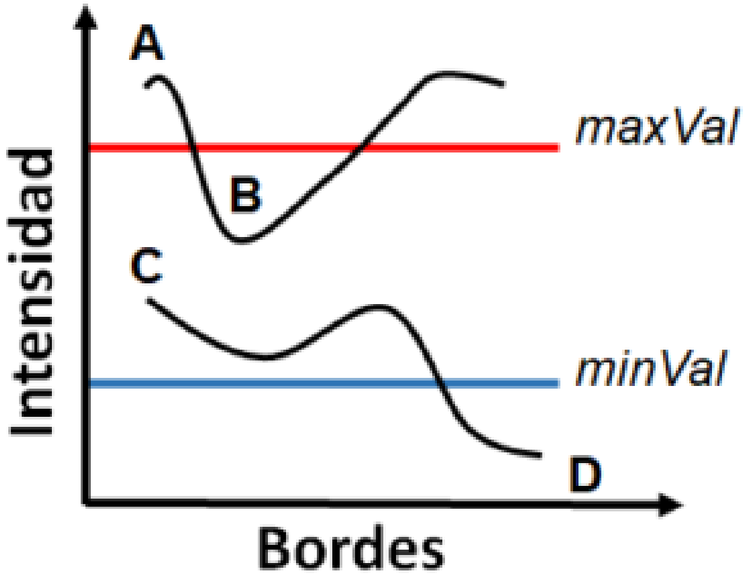
- Si están conectados a los píxeles de borde, se consideran como parte del borde. Caso contrario, se descartan



PROCESAMIENTO DIGITAL DE IMÁGENES

UMBRAL DE HISTÉRESIS

- El píxel A está por encima del valor umbral máximo, por lo que es aceptado como un borde seguro
- Los píxeles B y C son considerados como píxeles débiles por estar entre los dos valores umbrales
- El píxel D está por debajo del valor umbral mínimo, por lo que es descartado
- Dado que el píxel B está conectado con A, es considerado como un borde válido
- Por otro lado, como C no está conectado a un borde seguro, es descartado



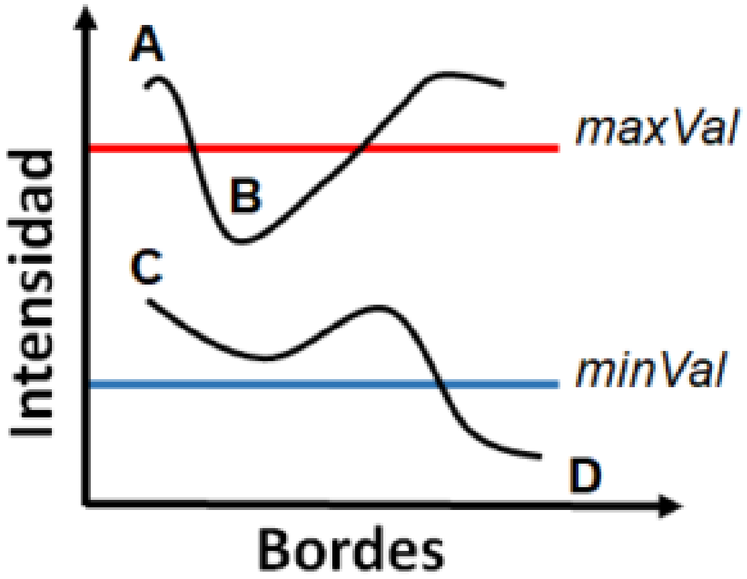
UMBRAL DE HISTÉRESIS

- Este paso también elimina los pequeños ruidos de píxeles en el supuesto de que los bordes son líneas largas

- Por ello, lo que obtenemos son bordes fuertes en la imagen

- Sin embargo, es importante recordar que esto dependerá de los 2 valores de umbral seleccionados previamente

- Se recomienda un ratio entre el valor umbral máximo y mínimo de 2 a 1 o de 3 a 1



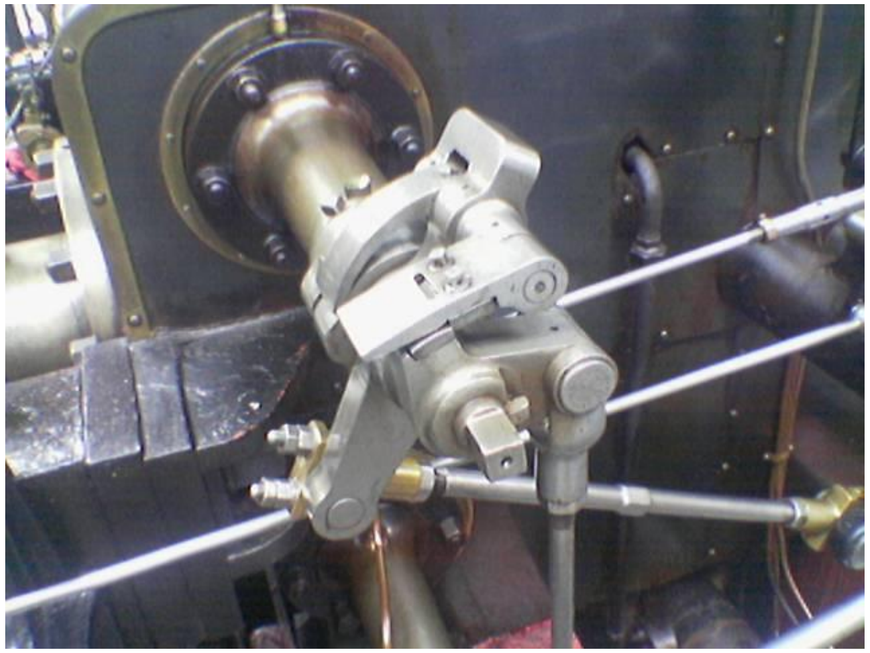
PROCESAMIENTO DIGITAL DE IMÁGENES

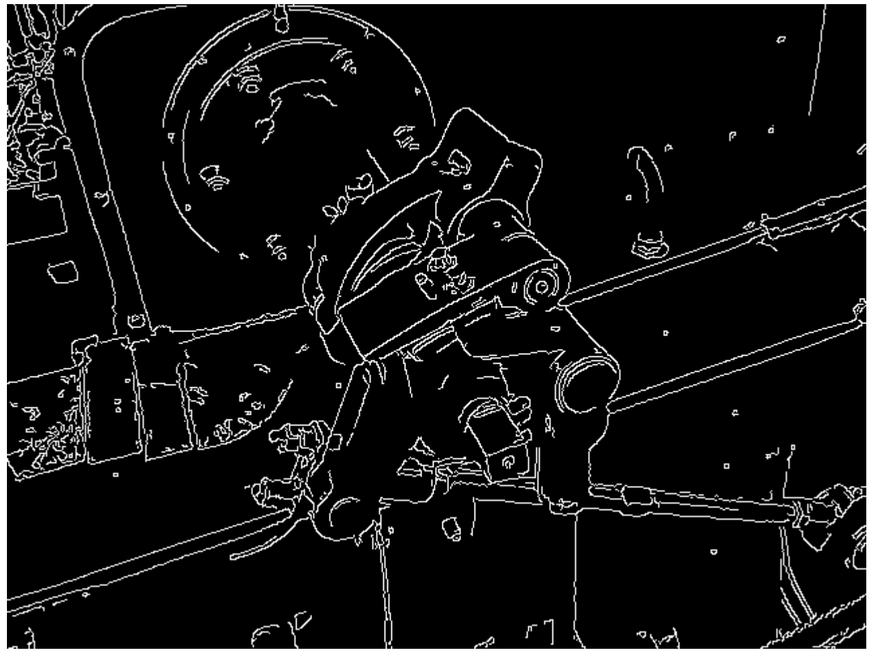
UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

ALGORITMO DE CANNY EN OPENCV

- En OpenCV, todos los pasos descritos están contenidos en un método

- Este método recibe una imagen, el tamaño del kernel para el operador Sobel, los 2 valores de umbral y un parámetro extra que permite elegir el modo de combinar las imágenes obtenidas por el operador Sobel





PROCESAMIENTO DIGITAL DE IMÁGENES

Aplicar algoritmo Canny a una imagen

Canny : Aplica el algoritmo Canny a una imagen

Sintaxis: cv2.Canny(imagen, valor_1, valor_2[, nueva_imagen[, kernel[, flag]]])

Parametros:
imagen : Imagen a la cual aplicar la operación
valor_1 : Valor umbral mínimo
valor_2 : Valor umbral máximo
nueva_imagen : Imagen resultante, con el mismo tamaño que la imagen recibida
kernel : Tamaño del kernel, para el operador Sobel, a aplicar en la imagen
flag : Determina el modo de combinar las imágenes obtenidas por el operador Sobel

Nota Cuando el parámetro «flag» es «True» se usa la notación que implica la raíz cuadrada de la suma de cuadrados. Si es «False» se usa la notación de la suma de valores absolutos

Ejemplo:

```python
import cv2

image = cv2.imread("image.png")
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

valor_umbral_min = 100
valor_umbral_max = 200
tamaño_kernel = 3
image_with_canny = cv2.Canny(image, valor_umbral_min, valor_umbral_max, apertureSize=tamaño_kernel)

cv2.imshow("Imagen", image)
cv2.imshow("Imagen con Canny", image_with_canny)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO