Universidad Nacional de San Antonio Abad del Cusco
Departamento Académico de Ing. Informática
VISION COMPUTACIONAL
Práctica Nº 07

DETECCIÓN DE BORDES

Iván C. Medrano Valencia

1. OBJETIVO.

- Detectar bordes en una imagen por diferentes métodos
- Utilizar las funciones de OpenCV para detectar bordes.

2. INTRODUCCIÓN.

La segmentación de una imagen es el proceso de dividir la imagen en partes de manera que se separen diferentes objetos o partes de objetos. Este es un paso de procesamiento esencial en la comprensión de imágenes (donde el objetivo es comprender el contenido de las imágenes). Hay dos formas principales de abordar la segmentación de imágenes.

1. Procesamiento de bordes, donde identificamos las discontinuidades (los bordes) en las imágenes.
2. Procesamiento de regiones, donde buscamos regiones (o secciones) homogéneas de las imágenes.

La visión binaria es un ejemplo muy simple de procesamiento de regiones y existen enfoques mucho más complejos para este propósito.

Existen muchas técnicas tanto en la visión basada en bordes como en la visión basada en regiones, todas las cuales tienen el objetivo de determinar la mejor representación de la escena en términos de bordes o regiones. Para la mayoría de las imágenes, no hay una respuesta correcta absoluta, ya que la respuesta es típicamente subjetiva (dependerá del observador y dependerá del propósito de la segmentación).

En esta práctica veremos algunas técnicas básicas de detección de bordes para determinar qué píxeles deben considerarse píxeles de borde, cómo se pueden procesar estos puntos de borde.

3. ACTIVIDAD I. OPERADOR DE ROBERTS

El operador de Roberts, es uno de los operadores más simples, que utiliza operadores de diferencia local para encontrar bordes. Utiliza la diferencia entre dos píxeles adyacentes en la dirección diagonal para aproximar la amplitud del gradiente para detectar bordes. El efecto de detectar bordes verticales es mejor que el de los bordes oblicuos, la precisión de posicionamiento es alta, es sensible al ruido y no puede suprimir la influencia del ruido.

En 1963, Roberts propuso este operador de búsqueda de bordes. Es una plantilla de 2x2 que usa la diferencia entre dos píxeles adyacentes en la dirección diagonal.

El kernel del operador de Roberts se divide en dirección horizontal y dirección vertical.

$$g_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$g_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Para realizar el operador de Roberts, pasamos principalmente el filtro de OpenCV *filter2D ()*. La función principal de esta función es realizar la operación de convolución de la imagen a través del kernel de convolución:

```
def filter2D(src, ddepth, kernel, dst=None, anchor=None,
delta=None, borderType=None)
```

donde:

src	: imagen de entrada
ddepth	: Es la profundidad deseable de la imagen de destino. El valor -1 representa que la imagen resultante tendrá la misma profundidad que la imagen de origen.
kernel	: kernel de convolución

- **Ejercicio 7.1.**

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
#--leer la imagen
img = cv.imread('images/mary.jpg', cv.COLOR_BGR2GRAY)
rgb_img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

# convertirlo a escala de grises
grayImage = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
#-- operador de Roberts
kernelx = np.array([[-1, 0], [0, 1]], dtype=int)
kernely = np.array([[0, -1], [1, 0]], dtype=int)

x = cv.filter2D(grayImage, cv.CV_16S, kernelx)
y = cv.filter2D(grayImage, cv.CV_16S, kernely)
# convertir a uint8, fusionar la imagen
absX = cv.convertScaleAbs(x)
absY = cv.convertScaleAbs(y)
Roberts = cv.addWeighted(absX, 0.5, absY, 0.5, 0)
#--mostrar los gráficos
```

```
titles = ['Imagen original', 'Operador de Roberts']
images = [rgb_img, Roberts]

for i in range(2):
    plt.subplot(1, 2, i + 1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```

4. ACTIVIDAD II. OPERADOR DE PREWITT.

El operador Prewitt es una especie de detección de bordes del operador diferencial de primer orden. Utiliza la diferencia de gris entre los puntos adyacentes superior e inferior, izquierdo y derecho del píxel para alcanzar el valor extremo en el borde y detectar el borde, eliminar algunos bordes falsos y suavizar el ruido.

Dado que el operador Prewitt utiliza una plantilla de 3 x 3 para calcular el valor de píxel en el área, y la plantilla del operador de Robert es 2 x 2, los resultados de detección de bordes del operador de Prewitt son más obvios en las direcciones horizontal y vertical que el operador de Robert. El operador Prewitt es adecuado para identificar imágenes con más ruido y escala de grises gradual.

La plantilla del operador Prewitt es la siguiente:

$$\mathrm{Pr}\,ewitt_x = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \mathrm{Pr}\,ewitt_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

- **Ejercicio 7.2**

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

#--Leer la imagen
img = cv.imread('images/mary.jpg', cv.COLOR_BGR2GRAY)
rgb_img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

#--convertir a escala de grises
grayImage = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

#--Operador de Prewitt
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]],dtype=int)
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]],dtype=int)
```

```
x = cv.filter2D(grayImage, cv.CV_16S, kernelx)
y = cv.filter2D(grayImage, cv.CV_16S, kernely)

#--Convertir a uint8, y fusionar la imagen
absX = cv.convertScaleAbs(x)
absY = cv.convertScaleAbs(y)
Prewitt = cv.addWeighted(absX, 0.5, absY, 0.5, 0)

#--Mostrar las imagenes
titles = ['Imagen original', 'Operador de Prewitt']
images = [rgb_img, Prewitt]

for i in range(2):
    plt.subplot(1, 2, i + 1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```

5. ACTIVIDAD III. OPERADOR SOBEL

Es un operador diferencial discreto utilizado para la detección de bordes, que combina el suavizado gaussiano y la derivación diferencial.

El operador Sobel agrega el concepto de peso sobre la base del operador Prewitt. Se cree que la distancia entre puntos adyacentes tiene un impacto diferente en el píxel actual. Cuanto más cerca se corresponda el punto de píxel con el píxel actual, mayor será el impacto, para realizar la imagen, enfocar y resaltar los contornos de los bordes.

El kernel es el siguiente:

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

- **Ejercicio 7.3**

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

#--Leer la imagen
img = cv.imread('images/mary.jpg', cv.COLOR_BGR2GRAY)
rgb_img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
```

```
#--Convertir a escala de grises
grayImage = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

#--Operador de Sobel
x = cv.Sobel(grayImage, cv.CV_16S, 1, 0)
y = cv.Sobel(grayImage, cv.CV_16S, 0, 1)

#--Convertir a uint8, fusionar la imagen
absX = cv.convertScaleAbs(x)
absY = cv.convertScaleAbs(y)
Sobel = cv.addWeighted(absX, 0.5, absY, 0.5, 0)

#--Mostrar las imagenes
titles = ['Imagen original', 'Operador de Sobel']
images = [rgb_img, Sobel]

for i in range(2):
    plt.subplot(1, 2, i + 1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```

6. ACTIVIDAD IV. OPERADOR LAPLACIANO

El operador laplaciano es un operador diferencial de segundo orden en el espacio euclidiano n-dimensional, que se utiliza a menudo en el campo de la mejora de imágenes y la extracción de bordes.

La idea central del operador laplaciano es determinar el valor de gris del píxel central de la imagen y el valor de gris de otros píxeles a su alrededor. Si el gris del píxel central es más alto, el gris del píxel central aumentará; de lo contrario, se reducirá.

En el proceso de implementación, el operador laplaciano calcula los gradientes en las cuatro u ocho direcciones del píxel central en el vecindario, y luego agrega los gradientes para determinar la relación entre el nivel de gris del píxel central y los niveles de gris de otros píxeles en la vecindad, y finalmente el resultado de la operación de degradado.

El operador laplaciano se divide en cuatro barrios y ocho barrios. Cuatro vecindarios es obtener gradientes en cuatro direcciones del píxel central del vecindario y ocho vecindarios es obtener gradientes en ocho direcciones.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- **Ejercicio 7.4**

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

#--Leer la iamgen
img = cv.imread('images/mary.jpg', cv.COLOR_BGR2GRAY)
rgb_img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

#--Convertir a escala de grises
grayImage = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

#--Operador Laplaciano
dst = cv.Laplacian(grayImage, cv.CV_16S, ksize = 3)
Laplacian = cv.convertScaleAbs(dst)

#--Mostrar imagenes
titles = ['Imagen original', 'Operador Laplaciano']
images = [rgb_img, Laplacian]

for i in range(2):
    plt.subplot(1, 2, i + 1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```

7. ACTIVIDAD V. OPERADOR CANNY

El detector de bordes Canny, 1986 combina la detección de bordes de la primera derivada y la segunda derivada para calcular el gradiente y la orientación de los bordes.

El algoritmo de detección de bordes de Canny se compone de 5 pasos:

1. Reducción de ruido;
2. Cálculo de gradientes;
3. Supresión no máxima;
4. Doble umbral;
5. Seguimiento de bordes por histéresis.

- **Ejercicio 7.5**

```
import numpy as np
import argparse
import cv2
#--Leer la imagen
image = cv2.imread('images/mary.jpg', cv2.COLOR_BGR2GRAY)
```

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image = cv2.GaussianBlur(image, (5, 5), 0)
cv2.imshow("Blurred", image)

canny = cv2.Canny(image, 30, 150)
cv2.imshow("Canny", canny)
cv2.waitKey(0)
```

8. TAREA

Desarrollar los programas para detección de bordes mediante los operadores de Kirsch y de Frei-Chen.

9. REFERENCIAS BIBLIOGRÁFICAS

- Dawson-Howe, K. (2014). A Practical Introduction to Computer Vision With OpenCV, Wiley & Sons Ltd.
- Hafsa Asad, W. R., Nikhil Singh (2020). The Computer Vision Workshop. Birmingham U.K., Pack Publishing.
- Szeliski, R. (2011). Computer Vision Algorithms and Applications. London, Springer.