Département MI Centre Universitaire de Mila

Traitement d’images Dr. B Maamar

Solution travaux pratiques N°3

1- Réponse 1

```python
import numpy as np
import math
import numpy
def convolution2D(X, H, moitie):
    s = X.shape
    py = int((H.shape[0] - 1) / 2)
    px = int((H.shape[1] - 1) / 2)
    Y = X.copy()
    if moitie:
        imax = int(s[1] / 2)
    else:
        imax = int(s[1] - px)
    for i in range(px, imax):
        for j in range(py, s[0] - py):
            somme = 0.0
            for k in range(-px, px + 1):
                for l in range(-py, py + 1):
                    somme += X[j + l][i + k] * H[l + py][k + px]
            Y[j][i] = somme
    return Y

image_test = np.array([[10,9,9,4,0],
                       [0,6,6,2,2],
                       [5,9,8,4,3],
                       [7,5,5,4,3],
                       [8,10,8,5,0]])
kernel_contour = np.array([[0, 1,0],
                           [1,-4,1],
                           [0, 1,0]])
res = convolution2D(image_test, kernel_contour, False)
print(res)
```

2- Réponse 2

$$I = \begin{array}{|c|c|c|}\hline 1 & 9 & 4 \\ \hline 6 & 3 & 2 \\ \hline 7 & 8 & 6 \\ \hline \end{array}$$

Gradient_x = Changement horizontal = $G_x = \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \otimes I$

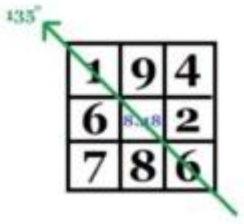
Gradient_y = Changement Verticall = $G_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \otimes I$

$$I = \begin{array}{|c|c|c|}\hline 1 & 9 & 4 \\ \hline 6 & 3 & 2 \\ \hline 7 & 8 & 6 \\ \hline \end{array}, \quad G_x = \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \otimes I, \quad G_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \otimes I$$

$G_x = 6, G_y = -6$

$G = \mathrm{sqrt}(36+36) = 8.48...$

$\Theta = \arctan(-6/6) = 135^{\circ}$



3- Réponse 3

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

def sobel_contour_detection(image_path , seuil):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    Amplitude = np.sqrt(sobel_x**2 + sobel_y**2)
    contour = Amplitude > seuil
    return contour

image_path = 'D:\Enseignements\Maamar\TP-02\img\color-img.png'
seuil = 60
edge_image = sobel_contour_detection(image_path,seuil)

original_image = cv2.imread(image_path)
original_image_rgb = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(original_image_rgb)
plt.title('Image Originale')
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(edge_image, cmap='gray')
plt.title('Contour detecté par Sobel avec un seuil = ' + str(seuil))
plt.axis('off')
plt.show()
```

4- Réponse 4

```python
import numpy as np
from PIL import Image
import math
# Conversion RGB image to grayscale
def gray(input_img):
    grayscale_img = input_img.convert("L")  # Convert to grayscale

    grayscale_img.save("grayscale_img.png")
    return np.array(grayscale_img)

# Apply 3x3 Gaussian filter
def filter(input_img, n):
    Gaussian3 = np.array([[1, 2, 1],
                          [2, 4, 2],
                          [1, 2, 1]])

    # Convolve the image with Gaussian filter
    height, width = input_img.shape
    filtered_img = np.copy(input_img)

    for y in range(1, height - 1):
        for x in range(1, width - 1):
            region = input_img[y - 1:y + 2, x - 1:x + 2]
            filtered_img[y, x] = np.sum(region * Gaussian3) // 16
    #print('filtered_img  ',filtered_img )
    return filtered_img

# Sobel edge detection
def sobel(input_img):
    sobel_dx = np.array([[-1, 0, 1],
                         [-2, 0, 2],
                         [-1, 0, 1]])

    sobel_dy = np.array([[-1, -2, -1],
                         [0, 0, 0],
                         [1, 2, 1]])

    height, width = input_img.shape
    gradient_magnitude = np.zeros_like(input_img, dtype=np.float32)
    gradient_direction = np.zeros_like(input_img, dtype=np.float32)

    for y in range(1, height - 1):
        for x in range(1, width - 1):
            region = input_img[y - 1:y + 2, x - 1:x + 2]
            gx = np.sum(region * sobel_dx)
            gy = np.sum(region * sobel_dy)

            magnitude = np.sqrt(gx ** 2 + gy ** 2)
            direction = math.degrees(math.atan2(gy, gx)) % 180

            gradient_magnitude[y, x] = magnitude
            gradient_direction[y, x] = direction

    #print('Amplitude , Direction : ' , gradient_magnitude, gradient_direction)
    return gradient_magnitude, gradient_direction
```

```python
# Non-maximum suppression
def suppression(gradient_magnitude, gradient_direction):
    height, width = gradient_magnitude.shape
    suppressed_img = np.zeros_like(gradient_magnitude)

    for y in range(1, height - 1):
        for x in range(1, width - 1):
            direction = gradient_direction[y, x]
            magnitude = gradient_magnitude[y, x]

            # Check direction and perform non-maximum suppression
            if (0 <= direction < 22.5) or (157.5 <= direction < 180):
                if magnitude >= gradient_magnitude[y, x - 1] and magnitude >= gradient_magnitude[y, x + 1]:
                    suppressed_img[y, x] = magnitude
            elif 22.5 <= direction < 67.5:
                if magnitude >= gradient_magnitude[y - 1, x + 1] and magnitude >= gradient_magnitude[y + 1, x - 1]:
                    suppressed_img[y, x] = magnitude
            elif 67.5 <= direction < 112.5:
                if magnitude >= gradient_magnitude[y - 1, x] and magnitude >= gradient_magnitude[y + 1, x]:
                    suppressed_img[y, x] = magnitude
            elif 112.5 <= direction < 157.5:
                if magnitude >= gradient_magnitude[y - 1, x - 1] and magnitude >= gradient_magnitude[y + 1, x + 1]:
                    suppressed_img[y, x] = magnitude

    #print('suppressed img' , suppressed_img)
    return suppressed_img
```

```python
# Double threshold
def db_threshold(img, high, low):
    height, width = img.shape
    strong = 255
    weak = 50

    result_img = np.zeros_like(img)

    strong_pixels = np.where(img >= high)
    weak_pixels = np.where((img >= low) & (img < high))

    result_img[strong_pixels] = strong
    result_img[weak_pixels] = weak
    #result_img.save("threshold.png")
    return result_img
```

```python
# Main function to apply the steps
def canny_edge_detection(image_path, high, low):
    input_img = Image.open(image_path)

    # Step 1: Convert to grayscale
    grayscale_img = gray(input_img)

    # Step 2: Apply Gaussian filter
    filtered_img = filter(grayscale_img, 3)

    # Step 3: Apply Sobel filter
    gradient_magnitude, gradient_direction = sobel(filtered_img)

    # Step 4: Non-maximum suppression
    suppressed_img = suppression(gradient_magnitude, gradient_direction)

    # Step 5: Double threshold
    edge_img = db_threshold(suppressed_img, high, low)

    return Image.fromarray(edge_img.astype(np.uint8))
```

```
if __name__ == "__main__":
    image_path = input("Input an image: ")
    high = int(input("High threshold: "))
    low = int(input("Low threshold: "))

    edge_image = canny_edge_detection(image_path, high, low)
    edge_image.save("edge_output.png")
    edge_image.show()
```