Bài giảng môn học:

Thị giác máy tính (7080518)

CHƯƠNG 3:
CÁC PHƯƠNG PHÁP PHÁT HIỆN BIÊN TRONG ẢNH

Đặng Văn Nam

dangvannam@humg.edu.vn

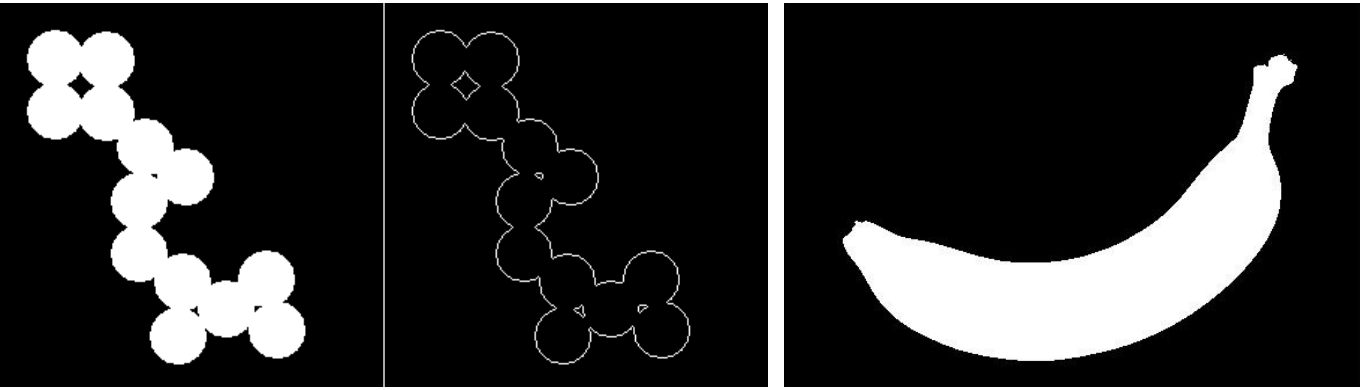
Nội dung chương 3

1. Bài toán phát hiện biên (Edge detection) và ứng dụng
2. Các phương pháp phát hiện biên
3. Gradient base (1st order)
    1. Sobel edge detection
    2. Prewitt edge detection
    3. Roberts edge detection
4. Laplacian based (2nd order)
    1. Laplace edge detection
    2. Canny edge detection

1. Bài toán phát hiện biên và ứng dụng

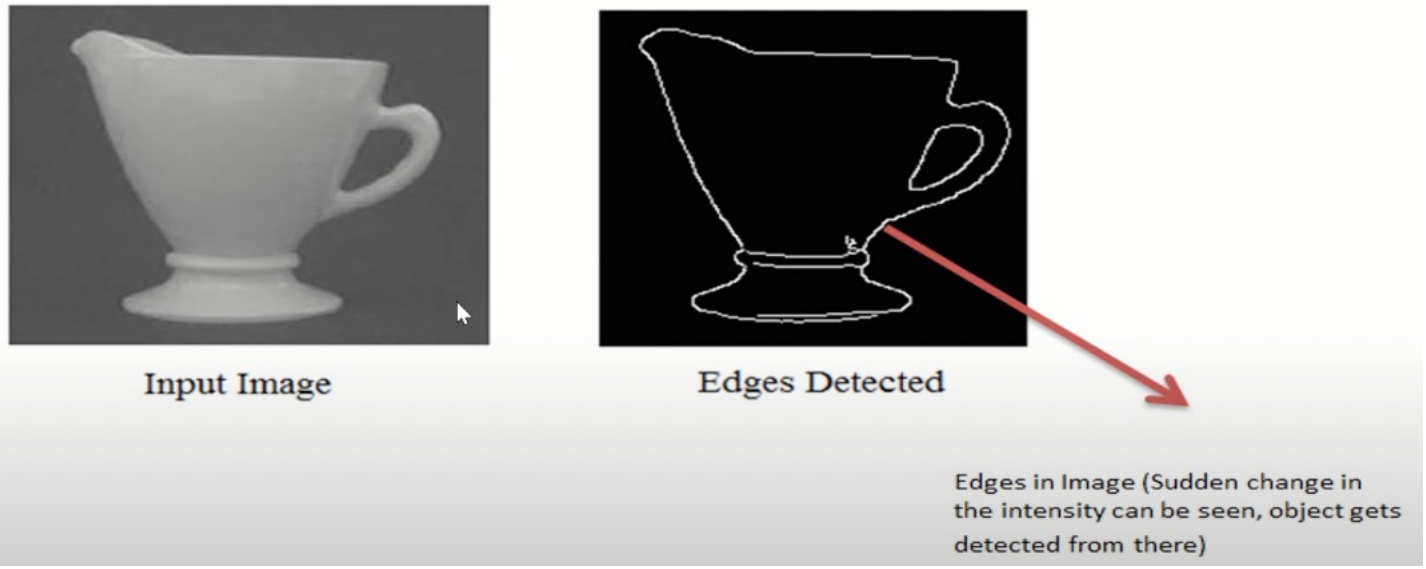
Giới thiệu

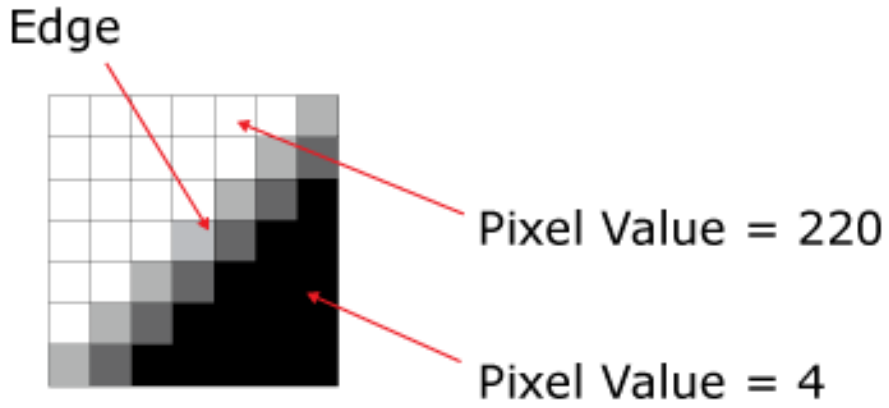
- Bài toán phát hiện cạnh, biên (Edge Detection) được ứng dụng rất nhiều trong các bài toán về thị giác máy tính. Như trong các bài toán về trích xuất thông tin, nhận dạng đối tượng….
- Biên là nơi mã hóa nhiều thông tin ngữ nghĩa (semantics information) và hình dạng (shape) trong một bức ảnh.



Giới thiệu

- **Điểm biên: Một điểm ảnh được coi là điểm biên nếu có sự thay đổi nhanh hoặc đột ngột về mức xám (hoặc màu)**
  - ❖ Ví dụ: trong ảnh nhị phân, điểm đen được coi là điểm biên nếu lân cận của nó có ít nhất một điểm trắng.
- **Đường biên còn được gọi là đường bao (boundary): là tập hợp các điểm biên liên tiếp.**





Giới thiệu

- **Ý nghĩa của đường biên:**
  - Đường biên là một loại đặc trưng cục bộ tiêu biểu trong phân tích, nhận dạng ảnh.
  - Người ta sử dụng biên làm phân cách các vùng xám (hoặc màu) cách biệt. Ngược lại, người ta cũng sử dụng các vùng ảnh để tìm phân cách.

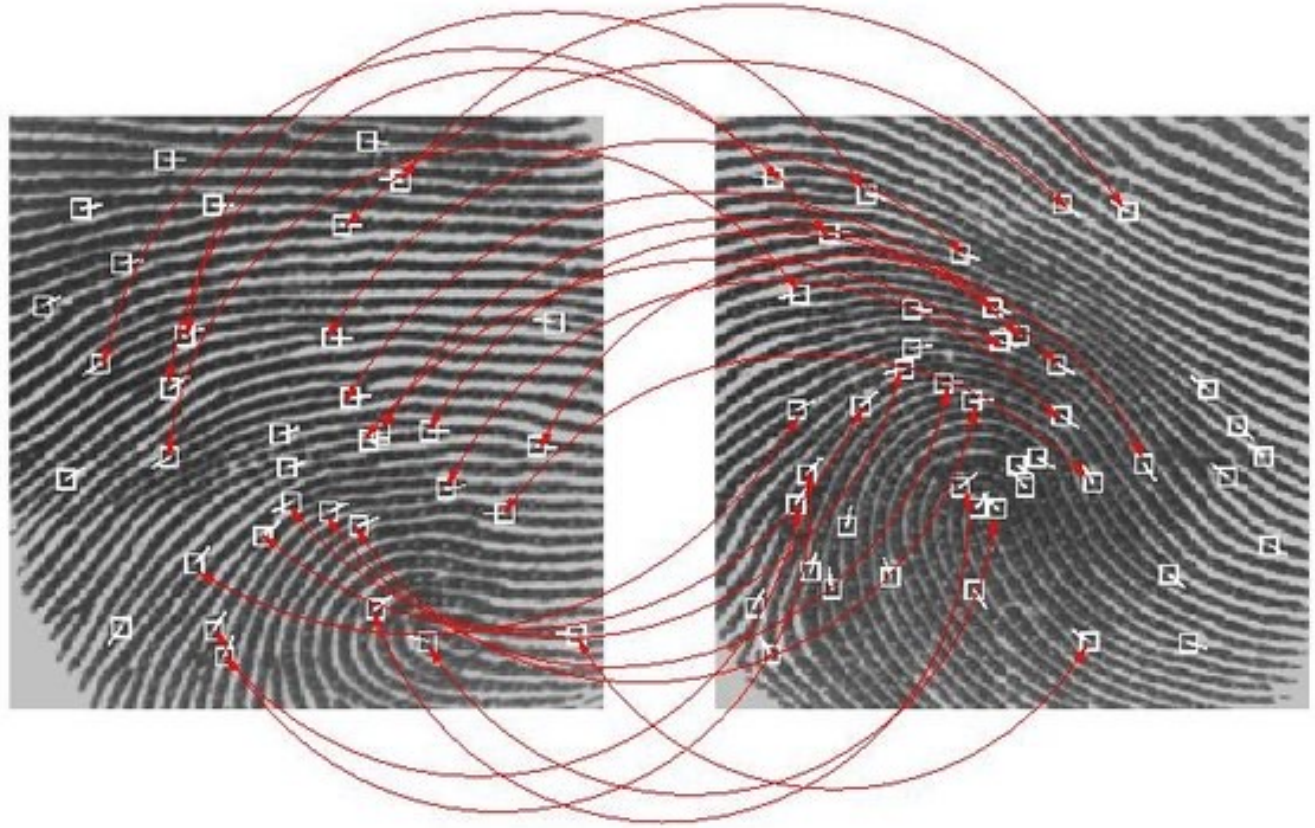
Giới thiệu

- **Ứng dụng của phát hiện biên (Edge detection):**
  - Giảm bớt thông tin không cần thiết trong ảnh mà vẫn giữ nguyên cấu trúc của ảnh.
  - Trích xuất các đặc tính quan trọng của ảnh như đường cong, góc và đường thẳng.
  - Nhận dạng các đối tượng, ranh giới và phân đoạn ảnh.
  - Đóng vai trò quan trọng trong nhận dạng và thị giác máy tính.

Giới thiệu

- **Ứng dụng của phát hiện biên (Edge detection):**

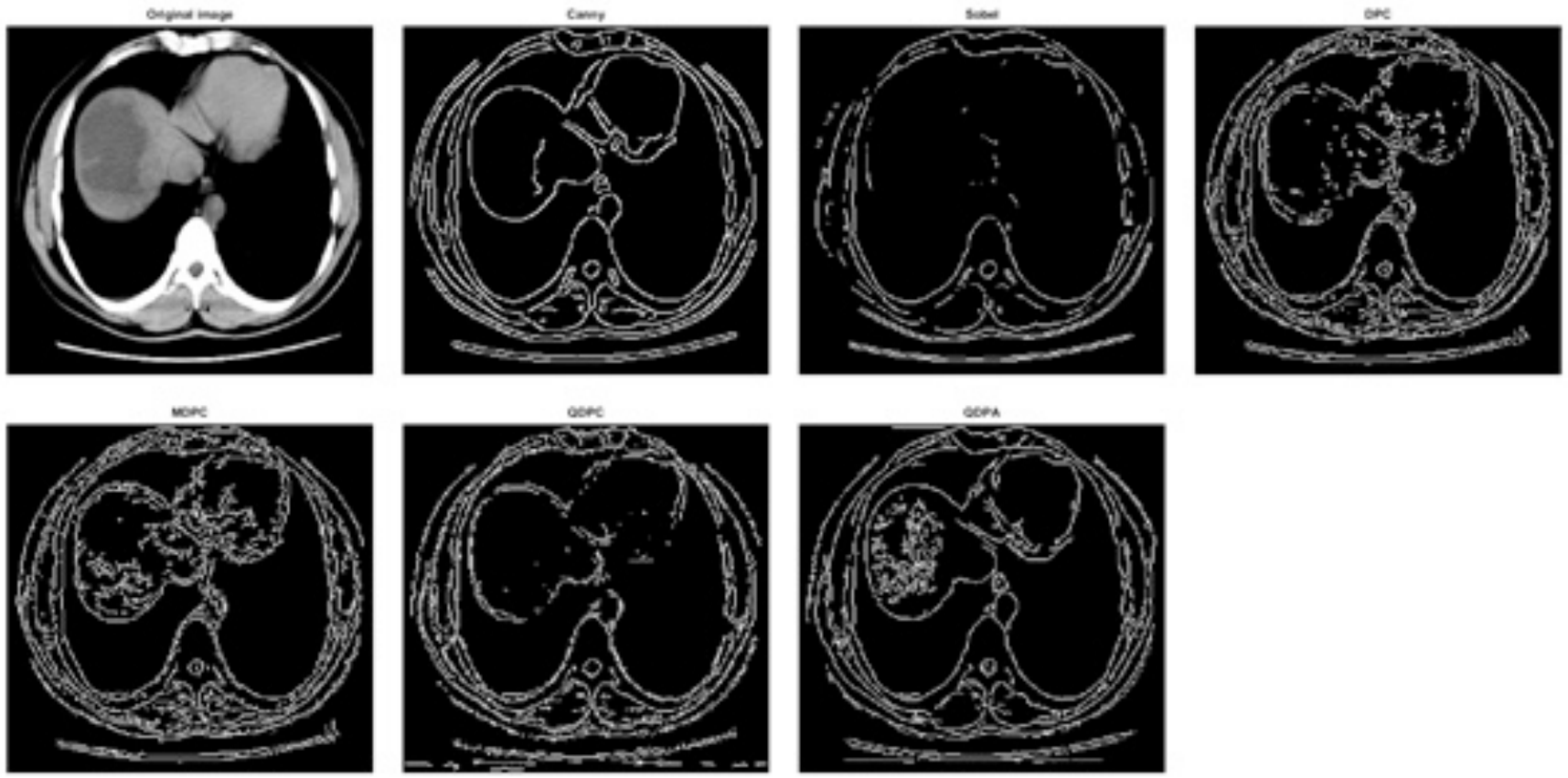
Đối sánh dấu vân tay



Giới thiệu

- **Ứng dụng của phát hiện biên (Edge detection):**

Chuẩn đoán y tế



Giới thiệu

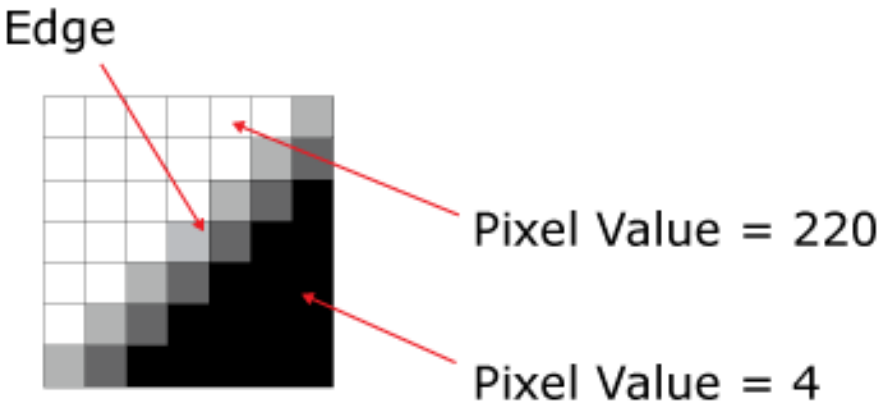
- **Ứng dụng của phát hiện biên (Edge detection):**

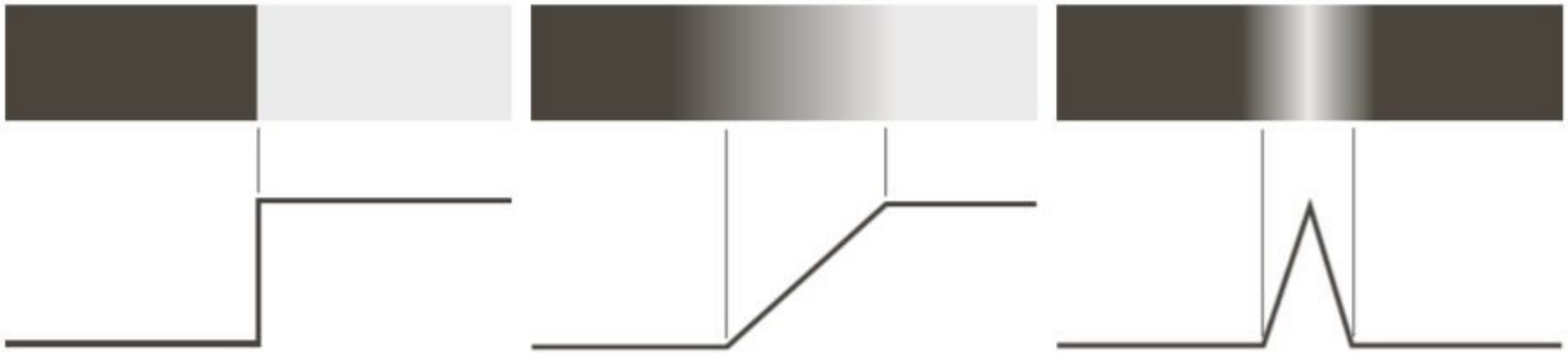
Nhận dạng biển số xe

a) Complemented Image b) Edged Image

Figure 3. Complemented Image and Edged Image

Một số loại biên trong ảnh (Types of Edges)



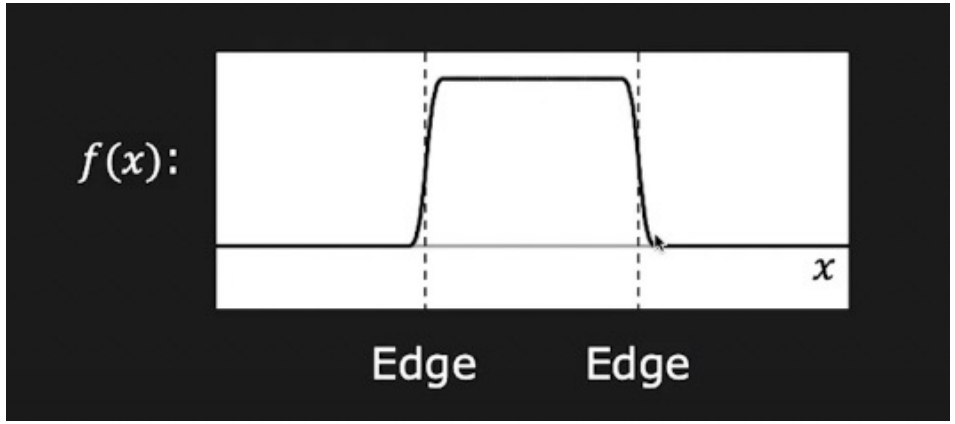


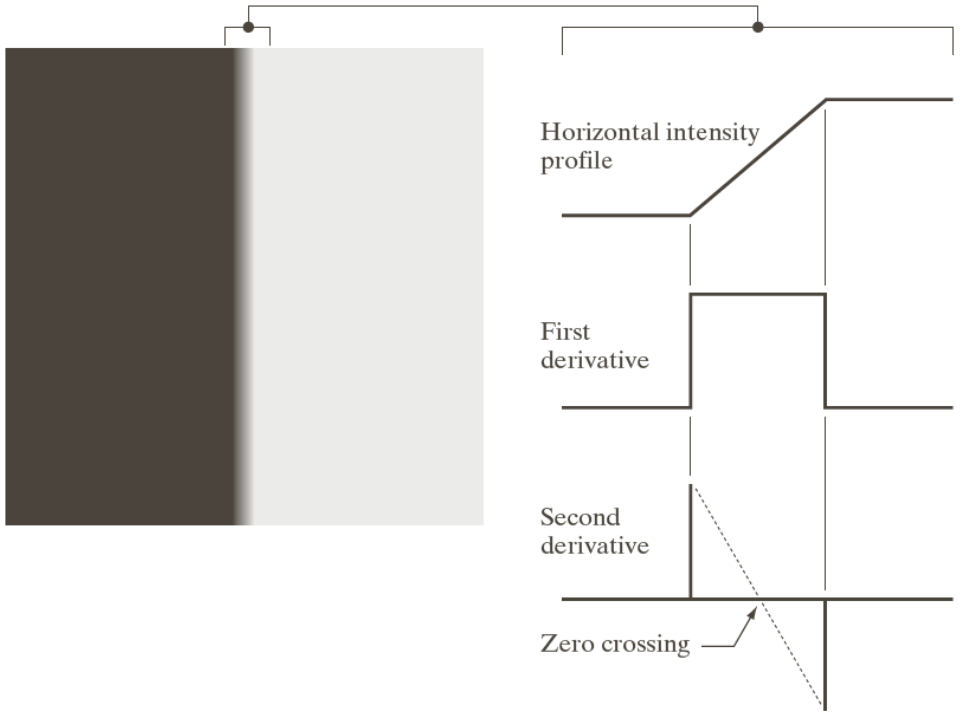
Step edge Ramp edge Roof edge

2.Phương pháp phát hiện biên

2. Phương pháp phát hiện biên

- Theo toán học, điểm ảnh có sự biến đổi mức xám u(x) một cách đột ngột





  - Nếu lấy đạo hàm bậc nhất của u(x) → Gradient
  - Nếu lấy đạo hàm bậc hai của u(x) → Laplace

Phát hiện biên gradient

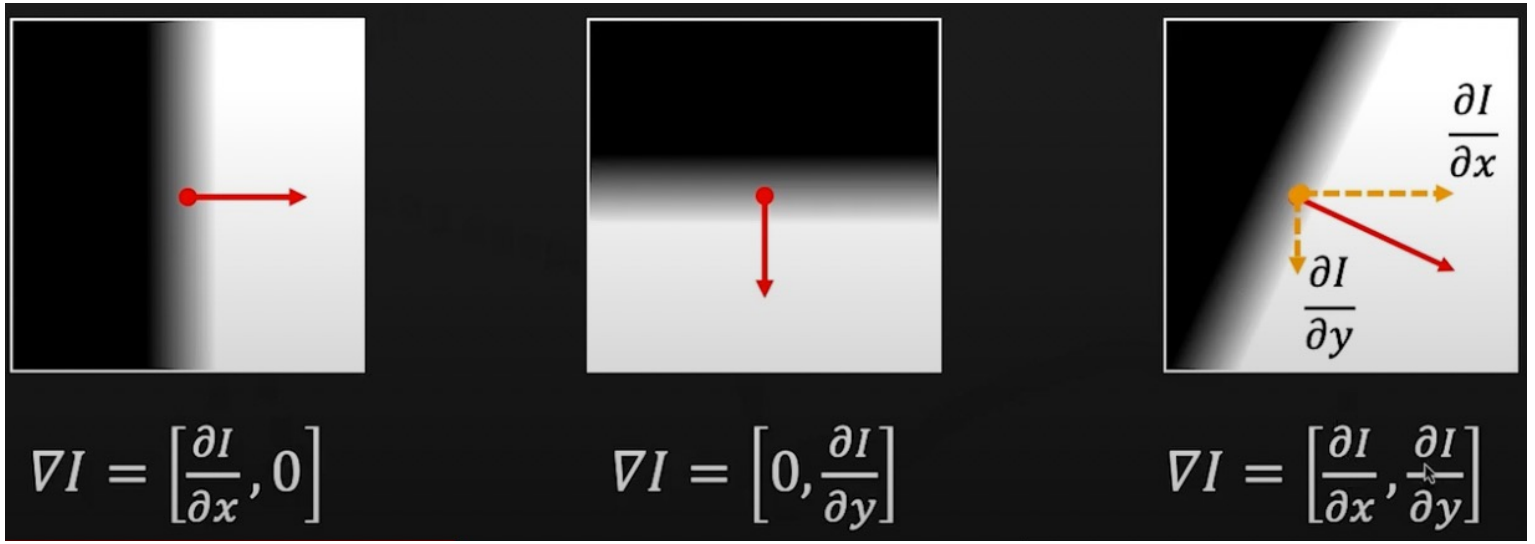
- Gradient là một vectơ có các thành phần biểu thị tốc độ thay đổi mức xám của điểm ảnh (theo hai hướng *x,y* đối với ảnh 2 chiều) :

$$\Delta f = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}$$

$$\frac{\partial f(x,y)}{\partial x} = f'_x \approx \frac{f(x+dx,y)-f(x,y)}{dx}$$

$$\frac{\partial f(x,y)}{\partial y} = f'_y \approx \frac{f(x,y+dy)-f(x,y)}{dy}$$

- Trong đó *dx, dy* là khoảng cách giữa 2 điểm kế cận theo hướng x,y tương ứng (thực tế chọn *dx*=*dy*=1)



Phát hiện biên gradient

- Để đơn giản tính toán, tính toán bằng h_x,h_y
  - Một số cặp mặt nạ tiêu biểu như Prewitt, Sobel, Robert đẳng hướng (Isometric)
  - g_x, g_y: Gradient theo hai hướng x,y

$$\text{Edge normal: } \nabla f \equiv grad(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \qquad G(m,n) = \sqrt{G_x^2(m,n) + G_y^2(m,n)}$$

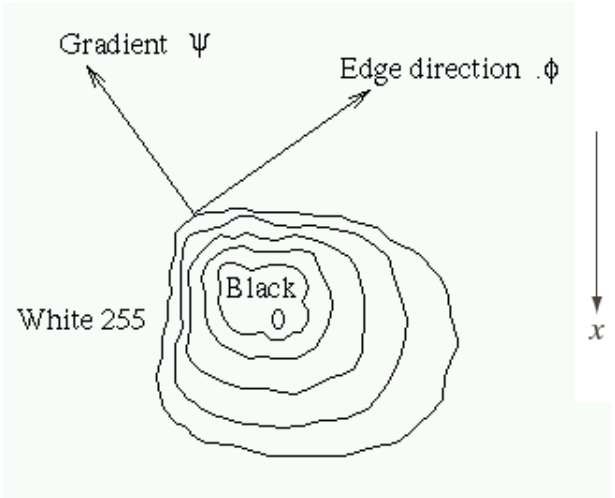
$$\text{Edge unit normal: } \quad \nabla f / \text{mag}(\nabla f)$$

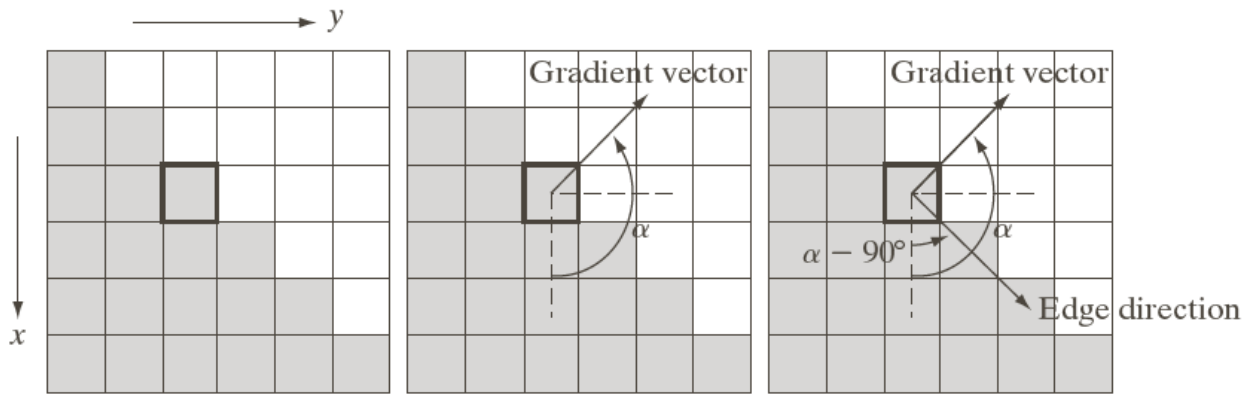
In practice,sometimes the magnitude is approximated by

$$\text{mag}(\nabla f) = \left|\frac{\partial f}{\partial x}\right| + \left|\frac{\partial f}{\partial y}\right| \text{ or } \text{mag}(\nabla f) = \max\left(\left|\frac{\partial f}{\partial x}\right|, \left|\frac{\partial f}{\partial y}\right|\right)$$

Phát hiện biên gradient

- Các bước thực hiện

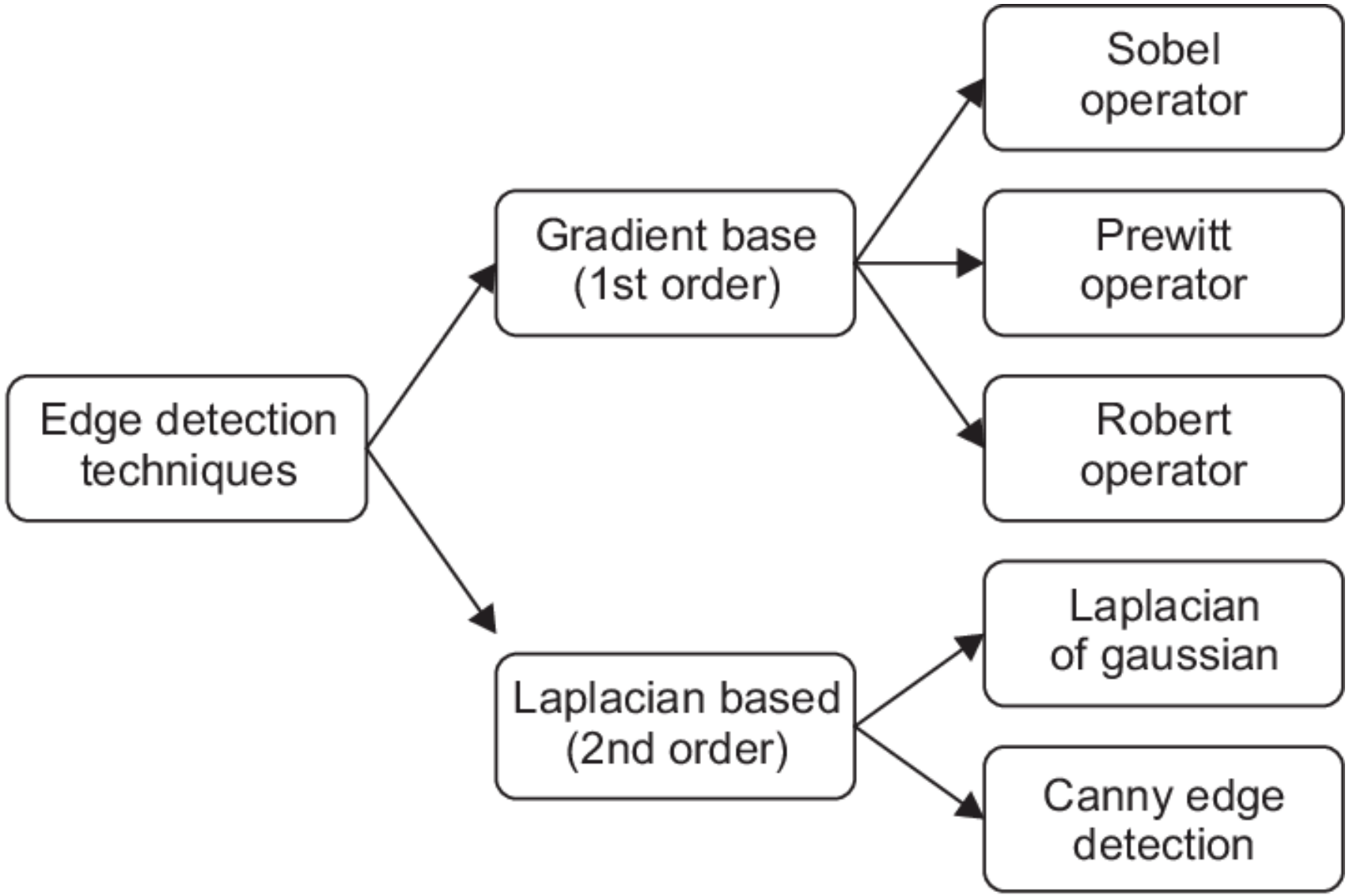




a b c

FIGURE 10.12 Using the gradient to determine edge strength and direction at a point. Note that the edge is perpendicular to the direction of the gradient vector at the point where the gradient is computed. Each square in the figure represents one pixel.

Phát hiện biên gradient



3. Gradient base (1st order)

Gradient base (1st order)

1	0
0	-1

0	1
-1	0

Robert Mask

-1	0	1
-1	0	1
-1	0	1

1	1	1
0	0	0
-1	-1	-1

Prewitt Mask

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

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

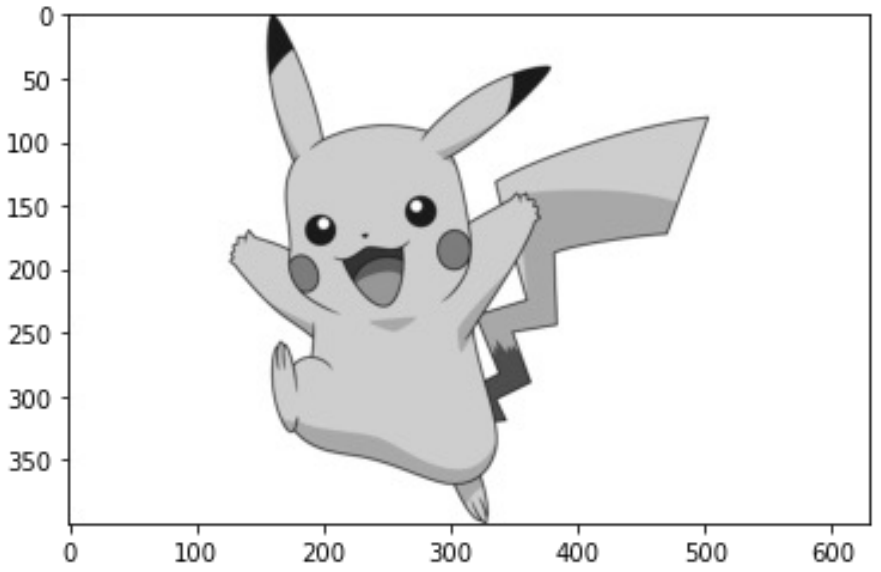
Sobel Mask

3.1 Sobel edge detection

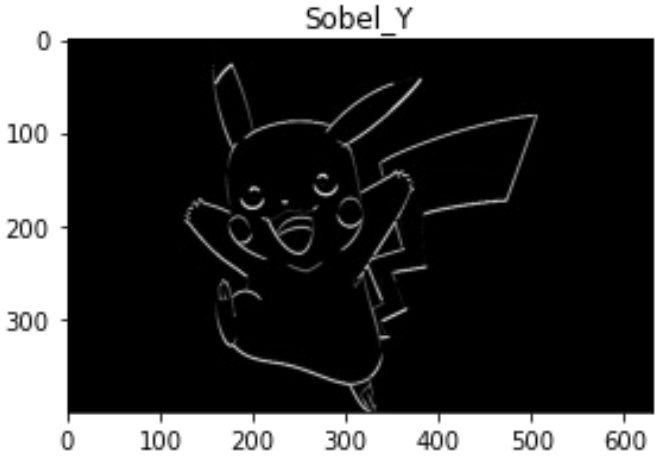
Sobel edge detection

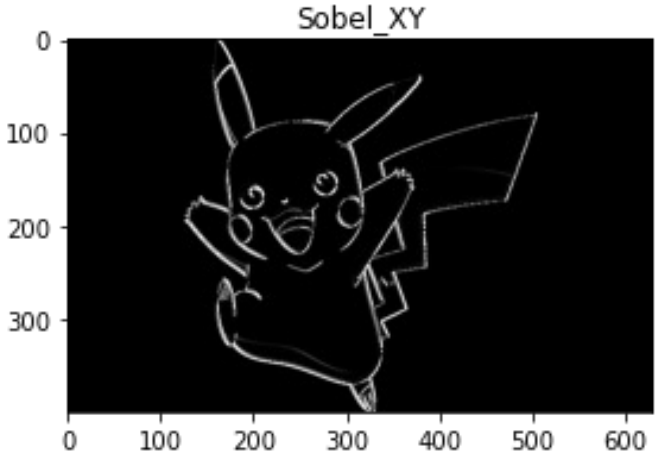
- Bộ lọc sobel theo hướng x, y

$$H_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad H_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

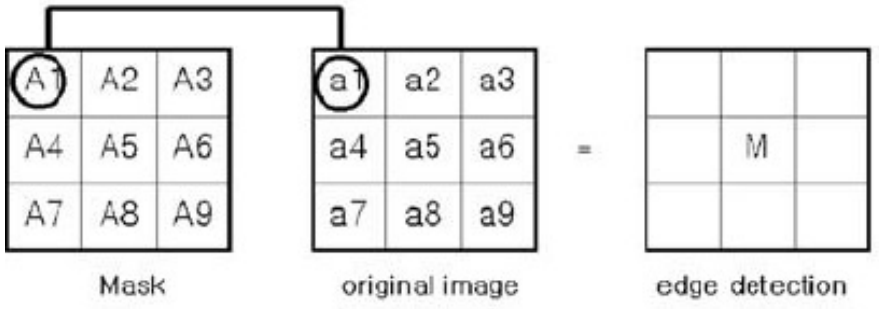


- **Bước 1:** Tính $I \otimes H_x$ và $I \otimes H_y$
- **Bước 2:** Tính $I \otimes H_x + I \otimes H_y$





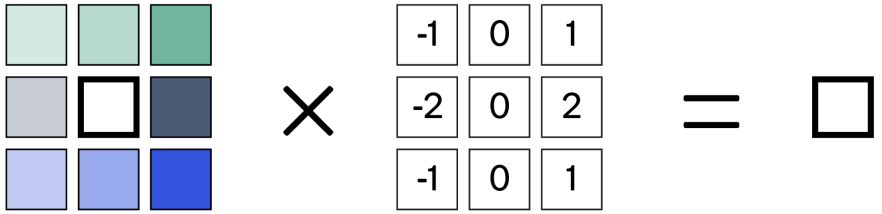
Sobel edge detection



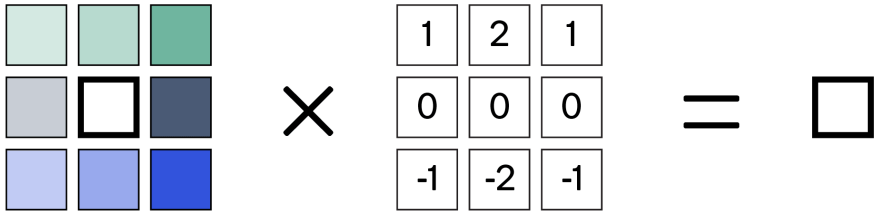
(a)

(b)

x direction



y direction



3x3 convolutional Sobel filters:

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

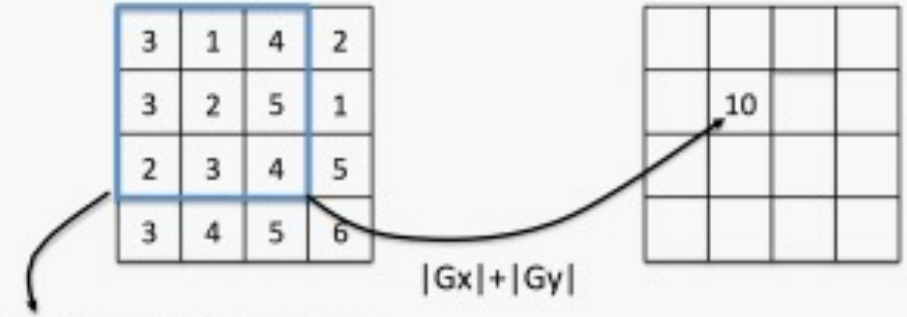
Gx

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

Gy

Gx = (-1*3)+(1*4)+(-2*3)+(2*5)+(-1*2)+(1*4) = 7

Gy = (1*3)+(2*1)+(1*4)+(-1*2)+(-2*3)+(-1*4) = -3



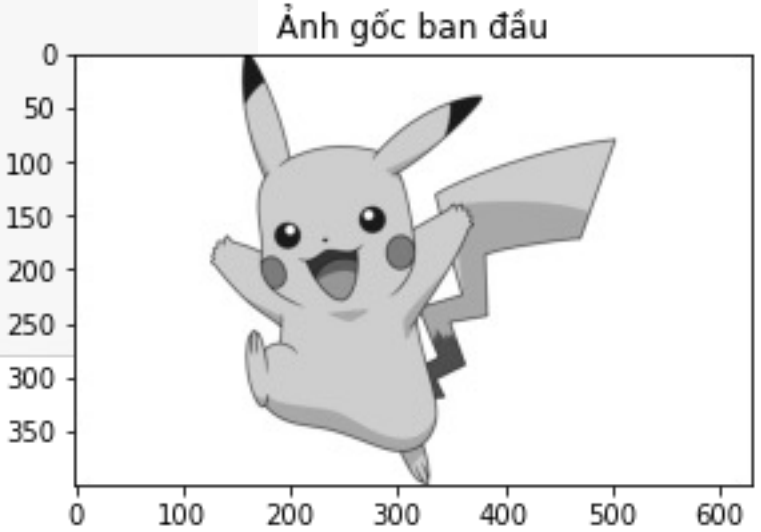
Sobel edge detection

- Bộ lọc sobel theo hướng x:

```python
#Khởi tạo kernel Sobel theo hướng X:
sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])
sobel_x
```

```
array([[-1,  0,  1],
       [-2,  0,  2],
       [-1,  0,  1]])
```

```python
#Thực hiện lọc ảnh với kernel ở trên:
img_x = cv2.filter2D(img_orignal, -1, sobel_x)

#Hiển thị ảnh đã xử lý qua bộ lọc:
plt.figure(figsize=(10,8))
plt.subplot(1,2,1)
plt.imshow(img_orignal,cmap='gray')
plt.title('Ảnh gốc ban đầu')

plt.subplot(1,2,2)
plt.imshow(img_x,cmap='gray')
plt.title('Sobel_X')
plt.show()
```



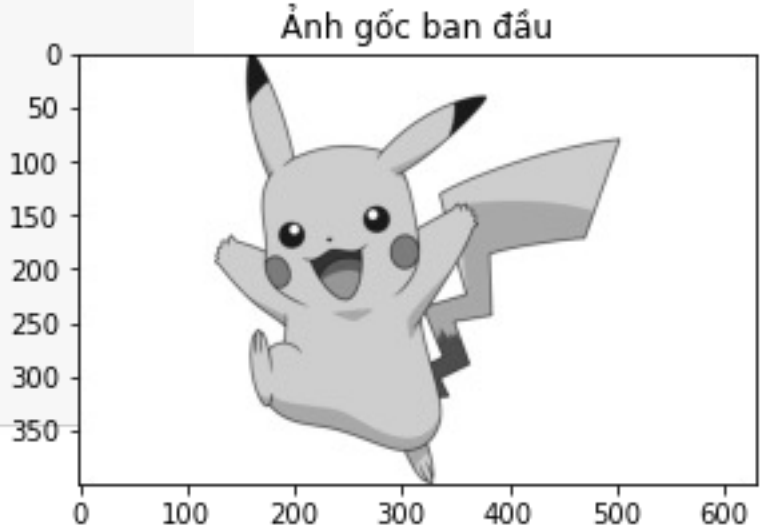
Sobel edge detection

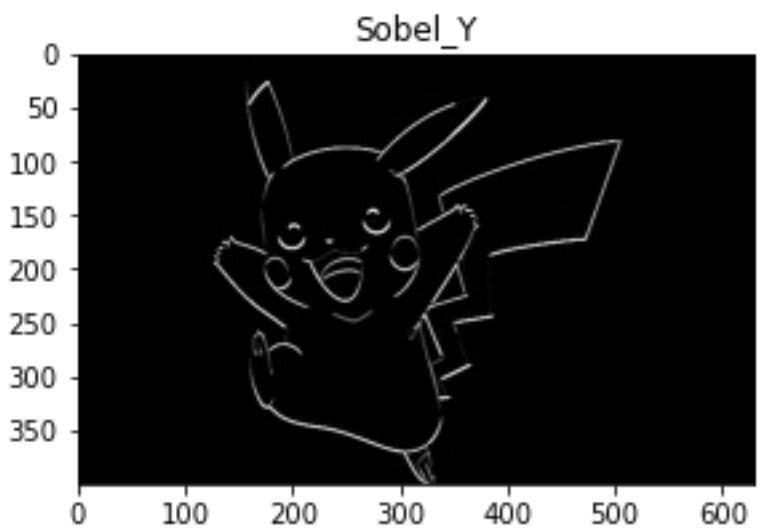
- Bộ lọc sobel theo hướng y:

```python
#Khởi tạo kernel sobel theo hướng Y:
sobel_y = np.array([[-1,-2,-1],
                    [ 0, 0, 0],
                    [ 1, 2, 1]])
sobel_y
```

```
array([[-1, -2, -1],
       [ 0,  0,  0],
       [ 1,  2,  1]])
```

```python
#Thực hiện lọc ảnh với kernel ở trên:
img_y = cv2.filter2D(img_orignal, -1, sobel_y)

#Hiển thị ảnh đã xử lý qua bộ lọc:
plt.figure(figsize=(10,8))
plt.subplot(1,2,1)
plt.imshow(img_orignal,cmap='gray')
plt.title('Ảnh gốc ban đầu')


plt.subplot(1,2,2)
plt.imshow(img_y,cmap='gray')
plt.title('Sobel_Y')
plt.show()
```





Sobel edge detection

- Sử dụng phương thức Sobel của OpenCV:

  cv2.Sobel(original_image,ddepth,xorder,yorder,ksize)

Trong đó:

1. original_image: Ảnh gốc (ảnh xám)
2. ddepth: Độ sâu (-1 | cv2.CV_64F)
3. xorder: Cạnh theo trục x (=1)
4. yorder: Cạnh theo trục y (=1)
5. ksize: kích thước kernel sobel (3, 5, 7..)

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

Gx

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

Gy

+2	+2	+4	+2	+2
+1	+1	+2	+1	+1
0	0	0	0	0
-1	-1	-2	-1	-1
-2	-2	-4	-2	-2

Gx

+2	+1	0	-1	-2
+2	+1	0	-1	-2
+4	+2	0	-2	-4
+2	+1	0	-1	-2
+2	+1	0	-1	-2

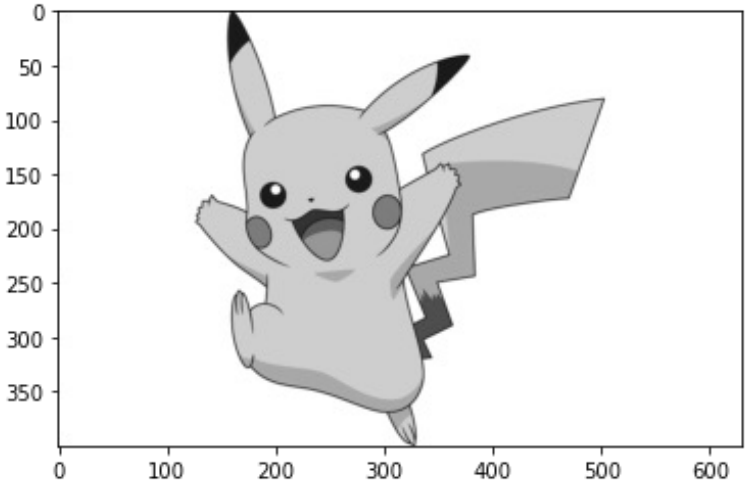
Gy

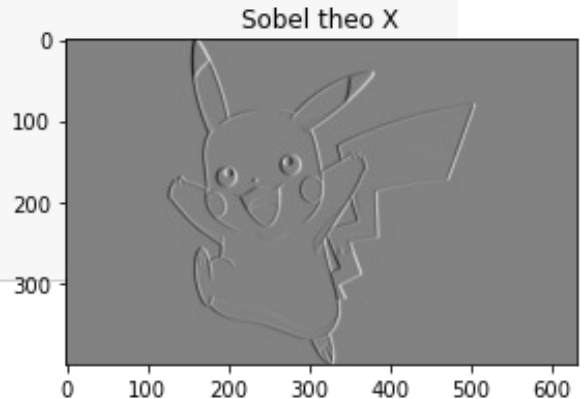
Sobel edge detection

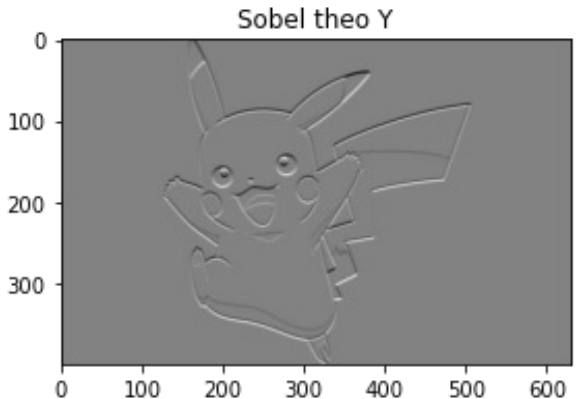
- Sử dụng phương thức Sobel của OpenCV:

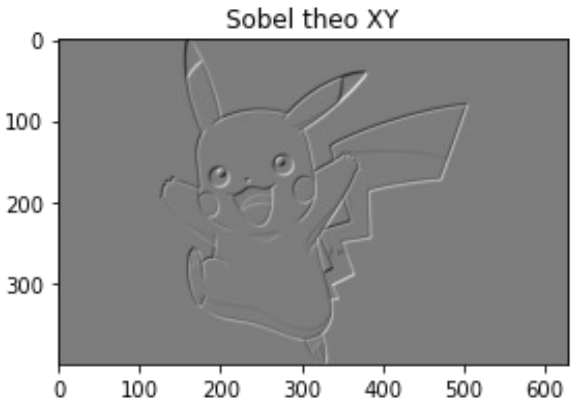
cv2.Sobel(original_image,ddepth,xorder,yorder,ksize)

```
#Sobel edges:
sobel_x = cv2.Sobel(img_orignal,cv2.CV_64F,1,0,ksize=5)
sobel_y = cv2.Sobel(img_orignal,cv2.CV_64F,0,1,ksize=5)

#Sobel theo cả x, y
img_sobel = sobel_x + sobel_y

#Hiển thị ảnh gốc và ảnh thay đổi kích thước
plt.figure(figsize=(15,5))
plt.subplot(1, 3, 1)
plt.imshow(sobel_x,cmap='gray')
plt.title('Sobel theo X')

plt.subplot(1, 3, 2)
plt.imshow(sobel_y,cmap='gray')
plt.title('Sobel theo Y')

plt.subplot(1, 3, 3)
plt.imshow(img_sobel,cmap='gray')
plt.title('Sobel theo XY')
plt.show()
```









Thực hành 3.1

Thực hành 3.1

1. Đọc và hiển thị ảnh Thuchanh3_1.jpg ở dạng ảnh xám
2. Sử dụng bộ lọc Sobel theo hướng x, y và XY. hiển thị kết quả.
3. Sử dụng phương thức Sobel với ksize = 7 theo X, Y và XY. Hiển thị kết quả.

3.2 Prewitt edge detection

Prewitt edge detection

- Thực hiện tương tự như với Sobel, Bộ lọc Prewitt theo hướng x, y

Image Neighbourhood

a_1	a_2	a_3
a_4		a_6
a_7	a_8	a_9

Sobel, G_x

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

Sobel, G_y

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

Prewitt, G_x

-1	-1	-1
0	0	0
1	1	1

Prewitt, G_y

-1	0	1
-1	0	-1
-1	0	-1

$$H_x = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

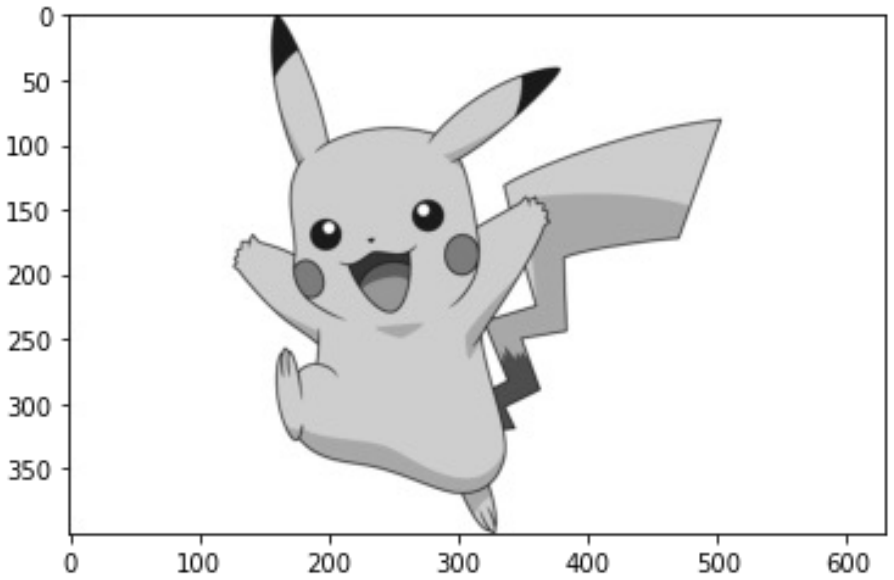
$$H_y = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

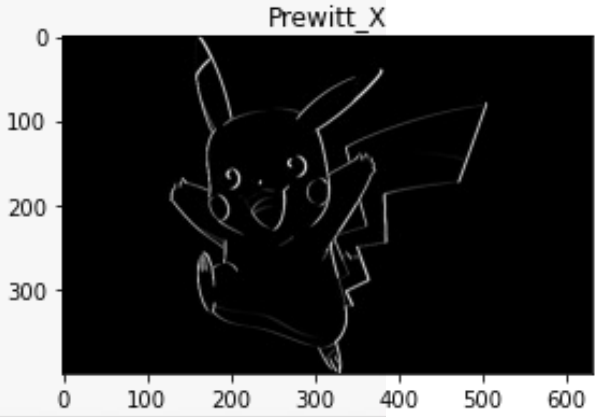
- **Bước 1:** Tính $I \otimes H_x$ và $I \otimes H_y$
- **Bước 2:** Tính $I \otimes H_x + I \otimes H_y$

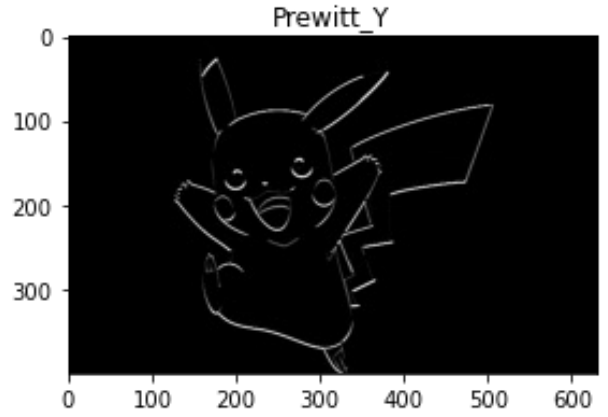
Prewitt edge detection

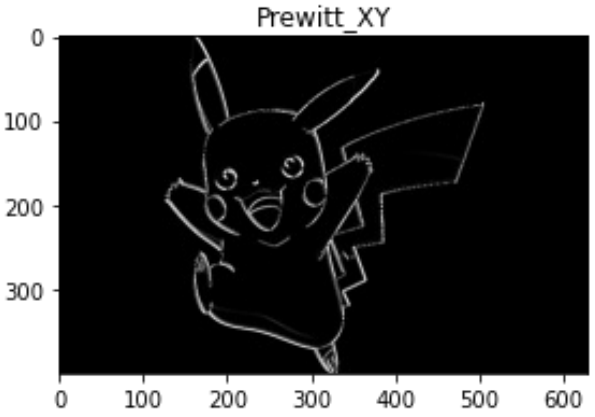
```python
#Khởi tạo kernel prewitt theo hướng x:
prewitt_x = np.array([[-1, 0, 1],
                      [ -1, 0, 1],
                      [ -1, 0, 1]])

#Khởi tạo kernel prewitt theo hướng y:
prewitt_y = np.array([[-1, -1,-1],
                      [  0,  0, 0],
                      [  1,  1, 1]])
```

```python
#Thực hiện lọc ảnh với kernel ở trên:
img_px = cv2.filter2D(img_orignal, -1, prewitt_x)

#Thực hiện lọc ảnh với kernel ở trên:
img_py = cv2.filter2D(img_orignal, -1, prewitt_y)

#kết hợp prewitt theo X, Y
img_pxy = img_px + img_py

#Hiển thị ảnh đã xử lý qua bộ lọc:
plt.figure(figsize=(15,8))
plt.subplot(1,3,1)
plt.imshow(img_px,cmap='gray')
plt.title('Prewitt_X')

plt.subplot(1,3,2)
plt.imshow(img_py,cmap='gray')
plt.title('Prewitt_Y')

plt.subplot(1,3,3)
plt.imshow(img_pxy,cmap='gray')
plt.title('Prewitt_XY')
plt.show()
```









3.3 Roberts edge detection

Roberts edge detection

- Roberts sử dung bộ lọc có kích thước 2x2, do Lawrence Roberts đề xuất vào năm 1963. Đây là một trong những bộ phát hiện cạnh đầu tiên.

1	0
0	-1

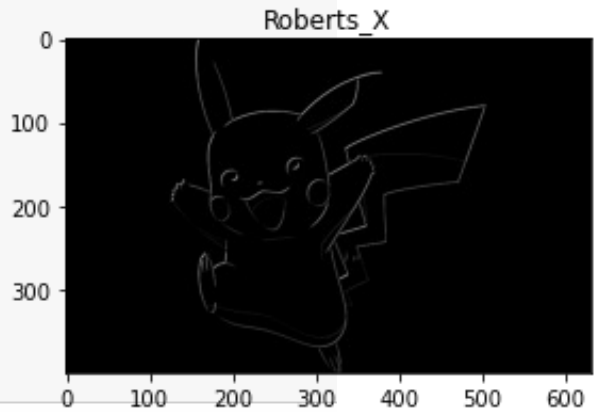
G_x

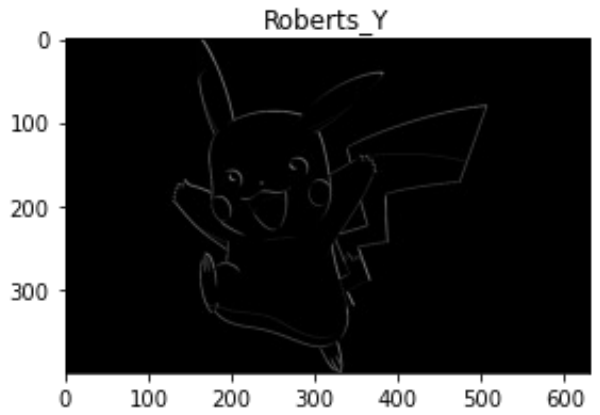
0	1
-1	0

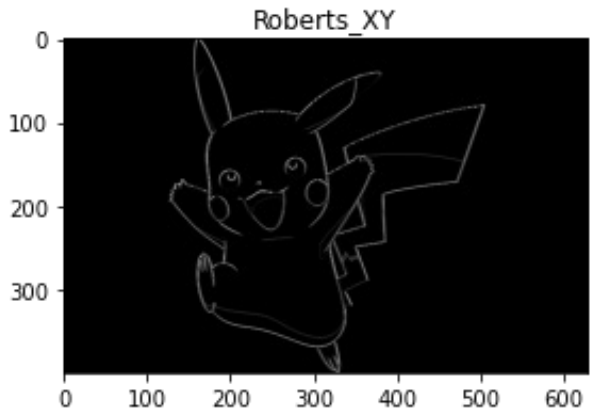
G_y

```python
#Khởi tạo kernel Roberts theo hướng X,Y:
roberts_x = np.array( [[1, 0 ],
                       [0,-1 ]] )

roberts_y = np.array( [[ 0, 1 ],
                       [ -1, 0 ]])
```

```python
#Thực hiện lọc ảnh với kernel ở trên:
img_rx = cv2.filter2D(img_orignal, -1, roberts_x)

#Thực hiện lọc ảnh với kernel ở trên:
img_ry = cv2.filter2D(img_orignal, -1, roberts_y)

#kết hợp prewitt theo X, Y
img_rxy = img_rx + img_ry

#Hiển thị ảnh đã xử lý qua bộ lọc:
plt.figure(figsize=(15,8))
plt.subplot(1,3,1)
plt.imshow(img_rx,cmap='gray')
plt.title('Roberts_X')

plt.subplot(1,3,2)
plt.imshow(img_ry,cmap='gray')
plt.title('Roberts_Y')

plt.subplot(1,3,3)
plt.imshow(img_rxy,cmap='gray')
plt.title('Roberts_XY')
plt.show()
```



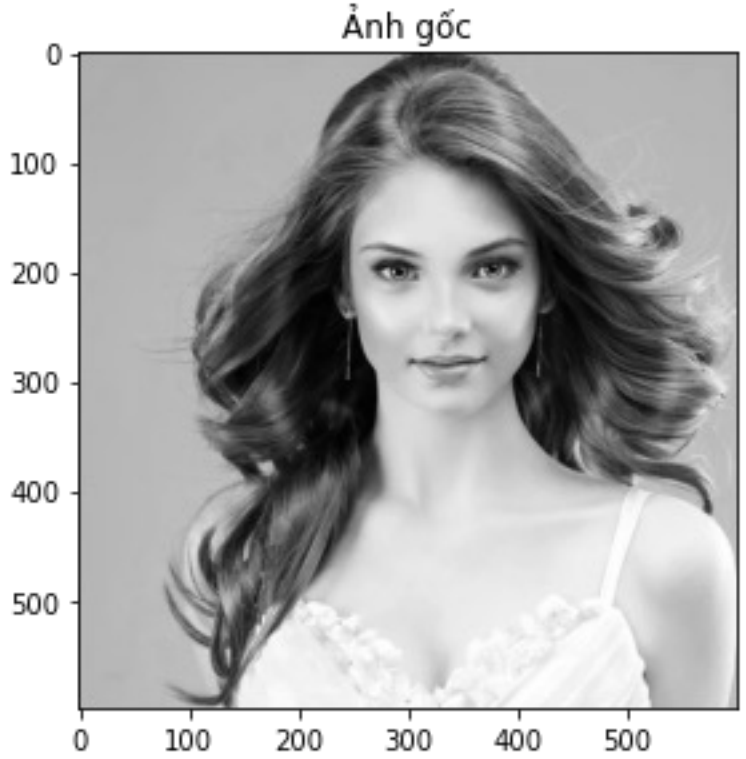


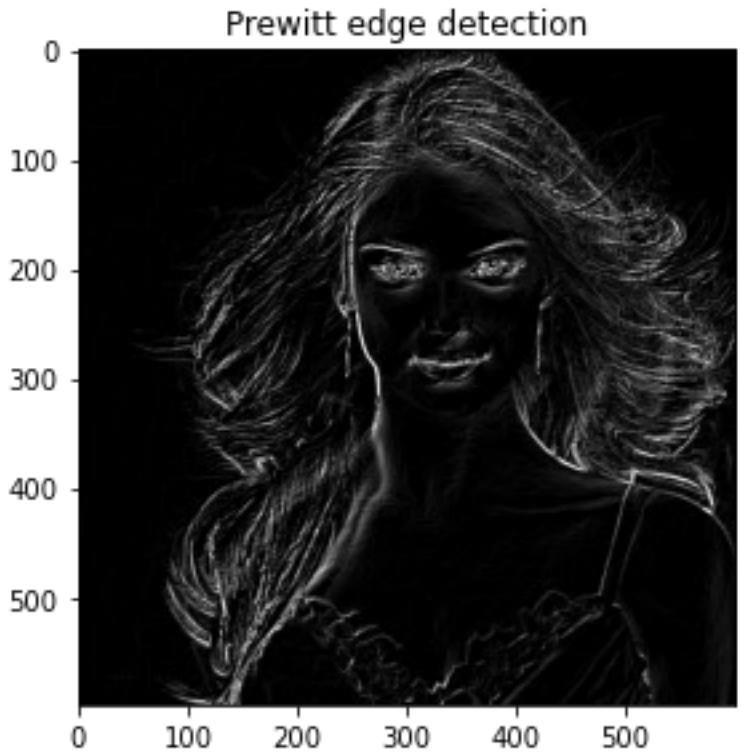


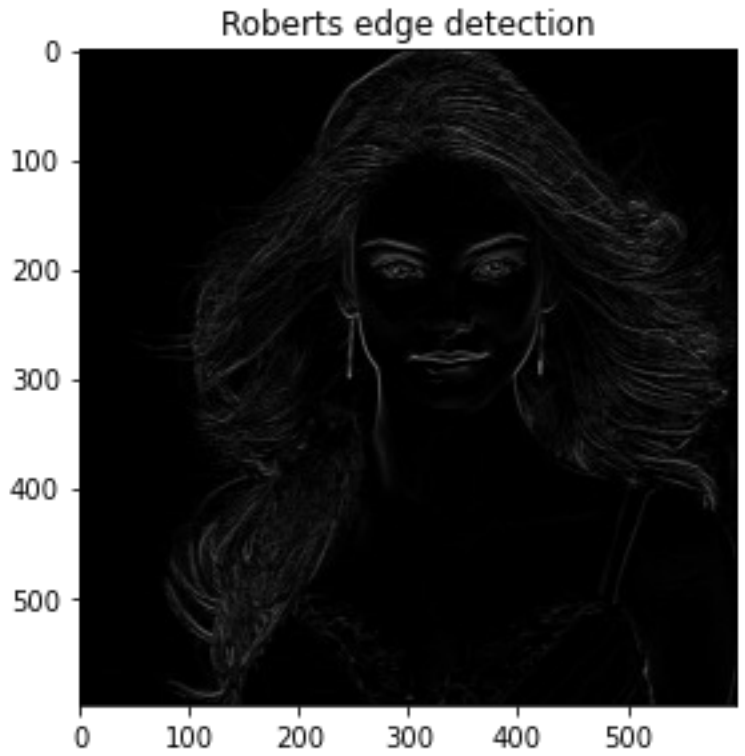
Thực hành 3.2

Thực hành 3.2

1. Đọc và hiển thị ảnh Thuchanh3_2.jpg ở dạng ảnh xám
2. Sử dụng bộ lọc Prewitt và Roberts để phát hiện cạnh, Hiển thị kết quả.

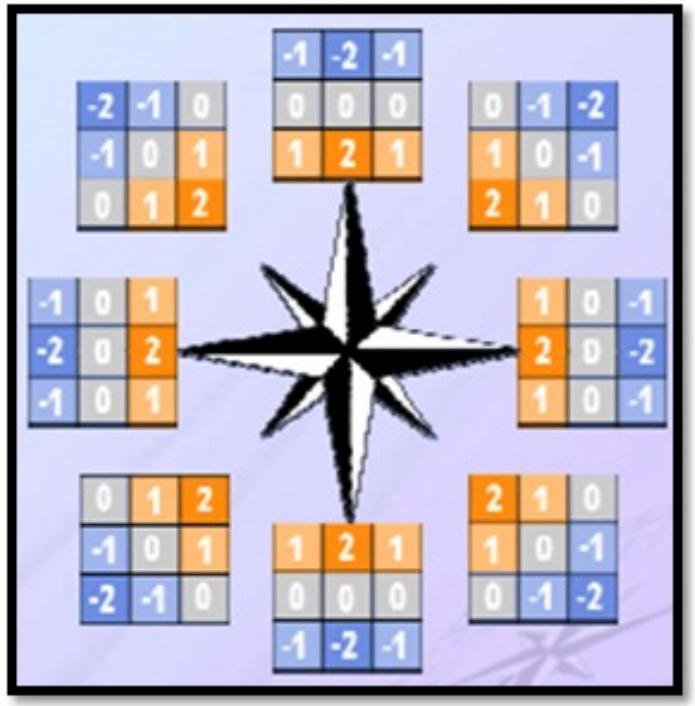






Thực hành 3.2

3. Sử dụng bộ lọc Robinson dưới đây để phát hiện cạnh và hiển thị kết quả với từng bộ lọc



4. Laplacian based (2nd order)

4.1 Laplace edge detection

Laplace edge detection

- Để khắc phục hạn chế và nhược điểm của phương pháp Gradient, trong đó sử dụng đạo hàm riêng bậc nhất người ta nghĩ đến việc sử dụng đạo hàm riêng bậc 2 hay toán tử Laplace.
- Phương pháp dò biên theo toán tử Laplace hiệu quả hơn phương pháp toán tử Gradient trong trường hợp mực xám biến đổi chậm, miền chuyển đổi mức xám có độ trải rộng.
- Toán tử Laplace dùng nhiều kiểu mặt nạ khác nhau để xấp xỉ rời rạc đạo hàm bậc hai. Ba kiểu mặt nạ hay dùng với toán tử Laplace.

0	1	0
1	−4	1
0	1	0

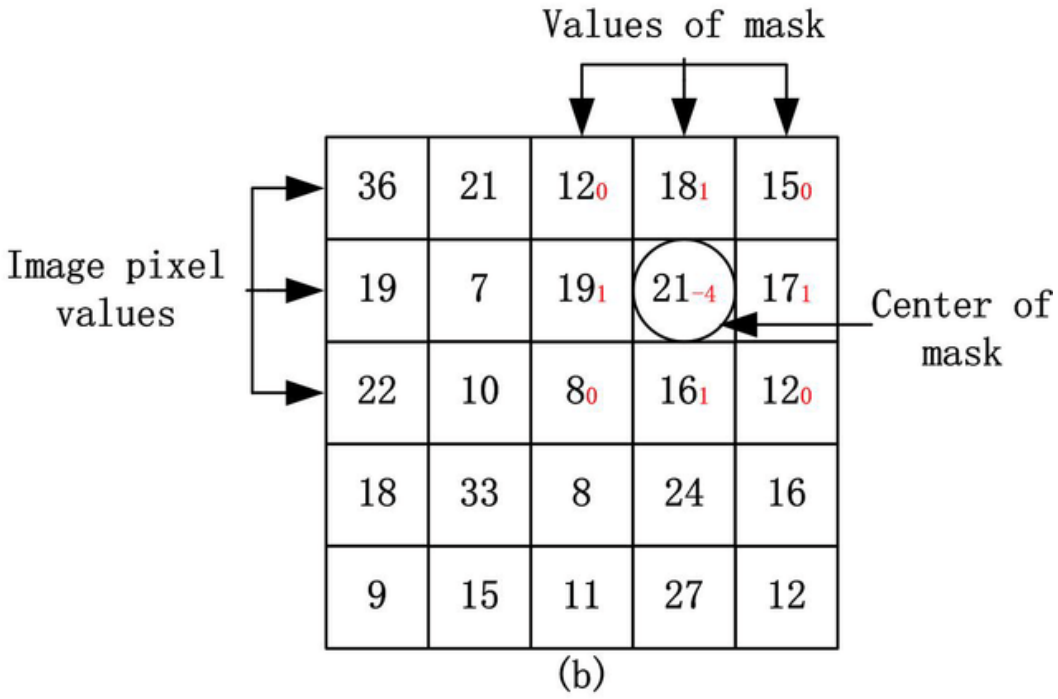
1	1	1
1	−8	1
1	1	1

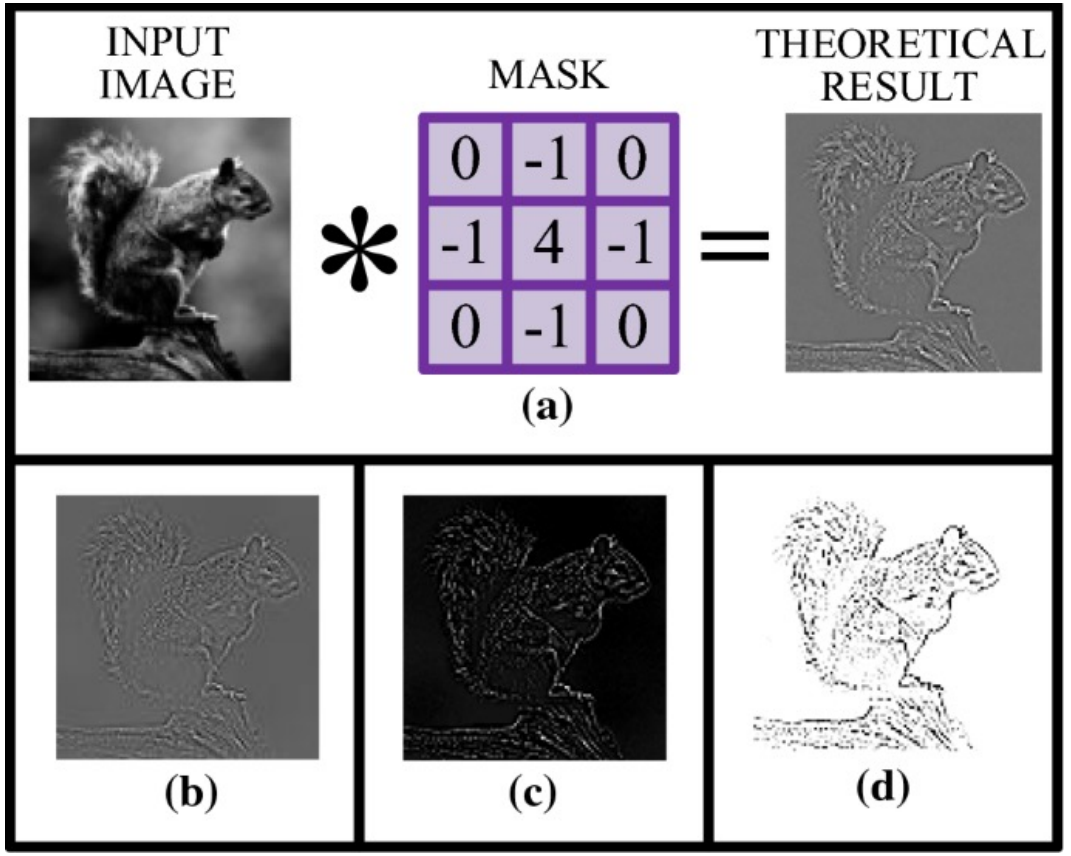
−1	2	−1
2	−4	2
−1	2	−1

Laplace edge detection

0	1	0
1	−4	1
0	1	0

(a)



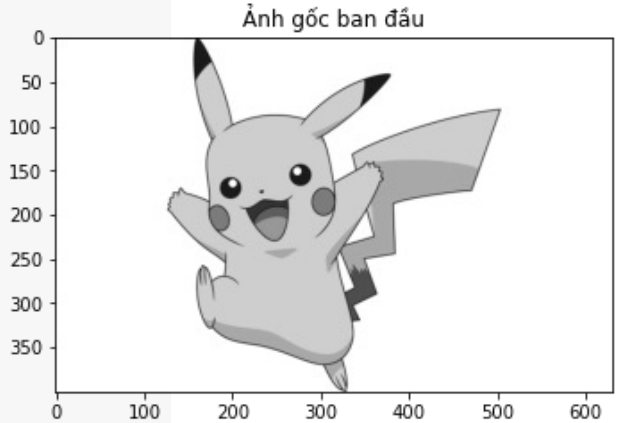


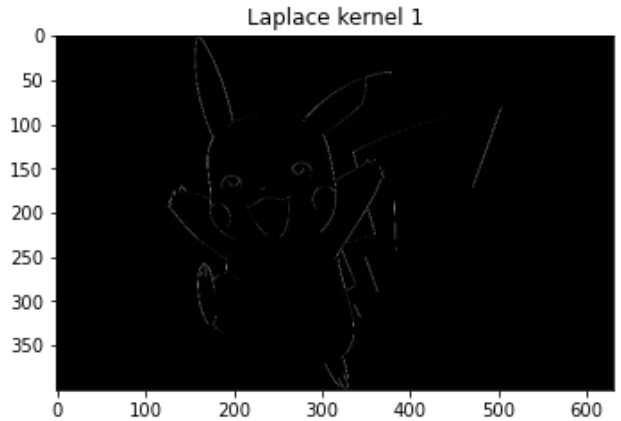
- Giả sử có ảnh I, khi đó tìm biên ảnh bằng cách lấy đạo hàm bậc 2 của ảnh I, nghĩa là nhân tích chập ảnh I với một trong 3 mặt nạ.

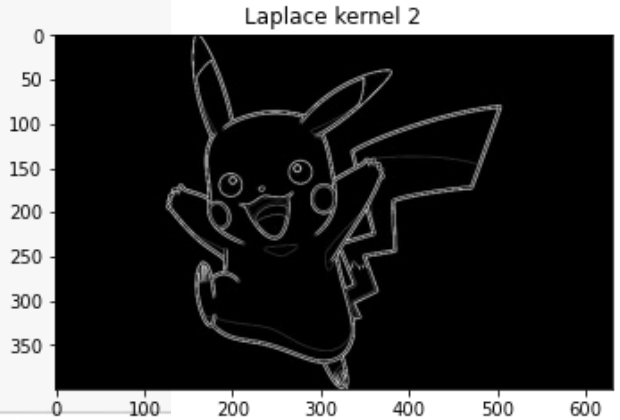
Laplace edge detection

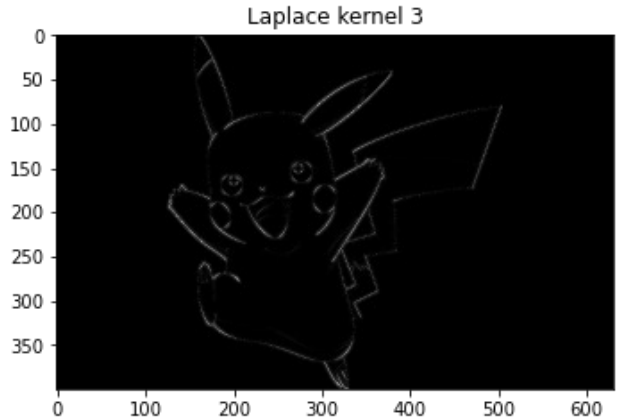
```python
#Khởi tạo kernel 1:
kernel_lap1 = np.array([[ 0, 1, 0],
                        [ 1,-4, 1],
                        [ 0,-1, 0]])

#Khởi tạo kernel 2:
kernel_lap2 = np.array([[-1,-1,-1],
                        [-1, 8,-1],
                        [-1,-1,-1]])

#Khởi tạo kernel 3:
kernel_lap3 = np.array([[-1, 2,-1],
                        [ 2,-4, 2],
                        [-1, 2,-1]])
```

```python
#Thực hiện lọc ảnh với kernel ở trên:
img_lap1 = cv2.filter2D(img_orignal, -1, kernel_lap1)
img_lap2 = cv2.filter2D(img_orignal, -1, kernel_lap2)
img_lap3 = cv2.filter2D(img_orignal, -1, kernel_lap3)

#Hiển thị kết quả:
plt.figure(figsize=(14,8))
plt.subplot(2,2,1)
plt.imshow(img_orignal,cmap='gray')
plt.title('Ảnh gốc ban đầu')

plt.subplot(2,2,2)
plt.imshow(img_lap1,cmap='gray')
plt.title('Laplace kernel 1')

plt.subplot(2,2,3)
plt.imshow(img_lap2,cmap='gray')
plt.title('Laplace kernel 2')

plt.subplot(2,2,4)
plt.imshow(img_lap3,cmap='gray')
plt.title('Laplace kernel 3')
plt.show()
```









Laplace edge detection

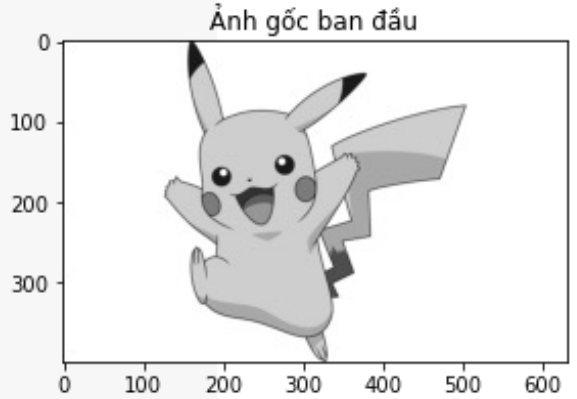
- Sử dụng phương thức Laplacian của OpenCV:

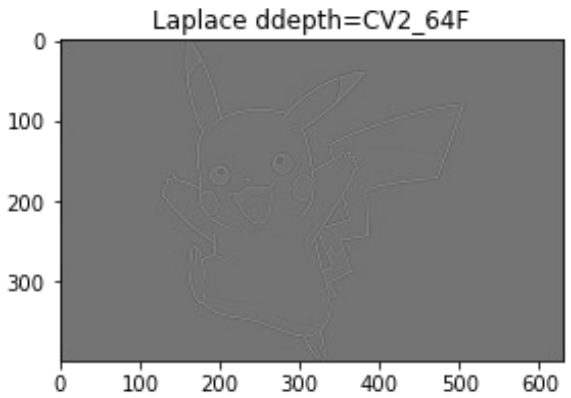
cv2.Laplacian(original_image,ddepth, ksize)

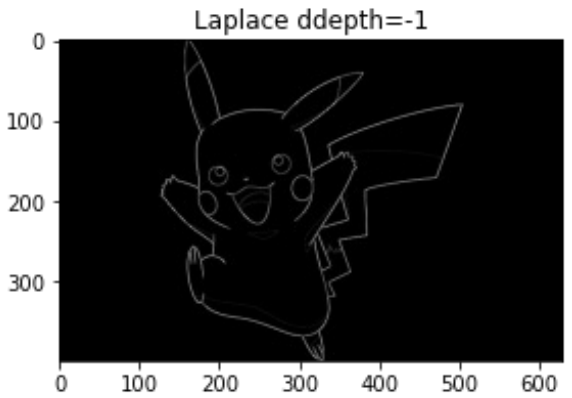
Trong đó:

1. original_image: Ảnh gốc (ảnh xám)
2. ddepth: Độ sâu (-1 | cv2.CV_64F)
3. ksize: kích thước kernel (3, 5, 7..)

```
#Sử dụng phương thức Laplacian của OpenCV:
img_l1 = cv2.Laplacian(img_orignal,cv2.CV_64F,3)
img_l2 = cv2.Laplacian(img_orignal,-1,7)

#Hiển thị kết quả:
plt.figure(figsize=(15,8))
plt.subplot(1,3,1)
plt.imshow(img_orignal,cmap='gray')
plt.title('Ảnh gốc ban đầu')

plt.subplot(1,3,2)
plt.imshow(img_l1,cmap='gray')
plt.title('Laplace ddepth=CV2_64F')

plt.subplot(1,3,3)
plt.imshow(img_l2,cmap='gray')
plt.title('Laplace ddepth=-1')
plt.show()
```







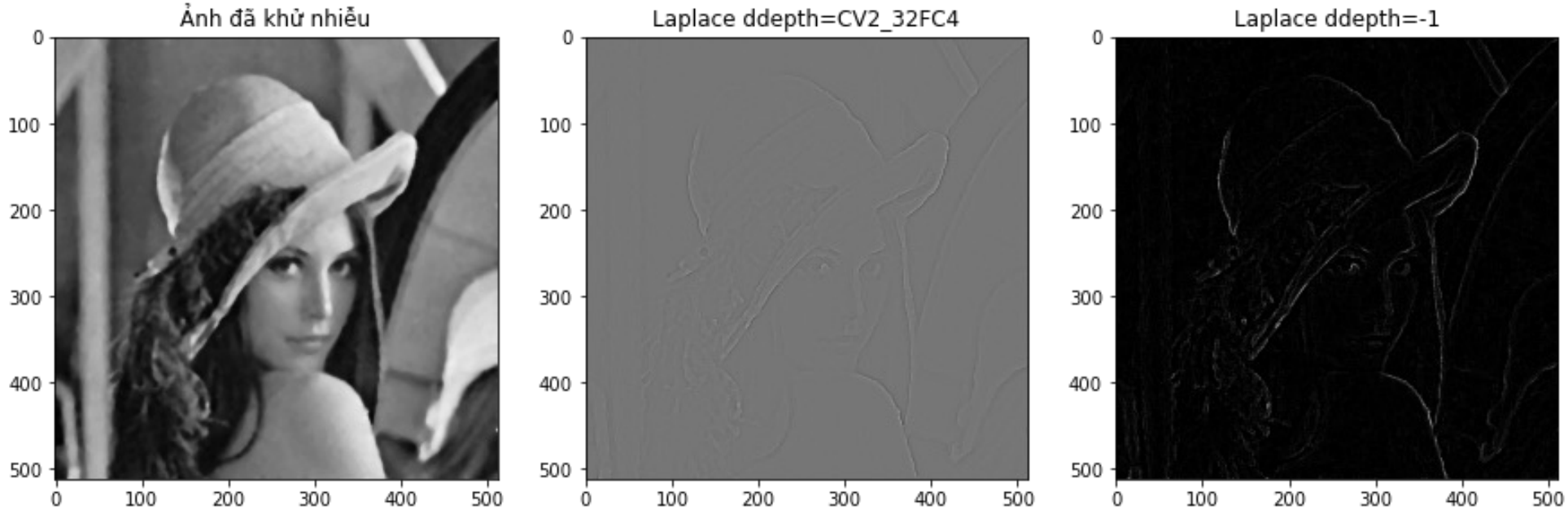
Thực hành 3.3

Thực hành 3.3

1. Đọc và hiển thị ảnh Thuchanh3_3.png.
2. Sử dụng bộ lọc Lapacian trên ảnh nhiễu muối tiêu → nhận xét kết quả khi lấy biên trên ảnh chưa khử nhiễu

Thực hành 3.3

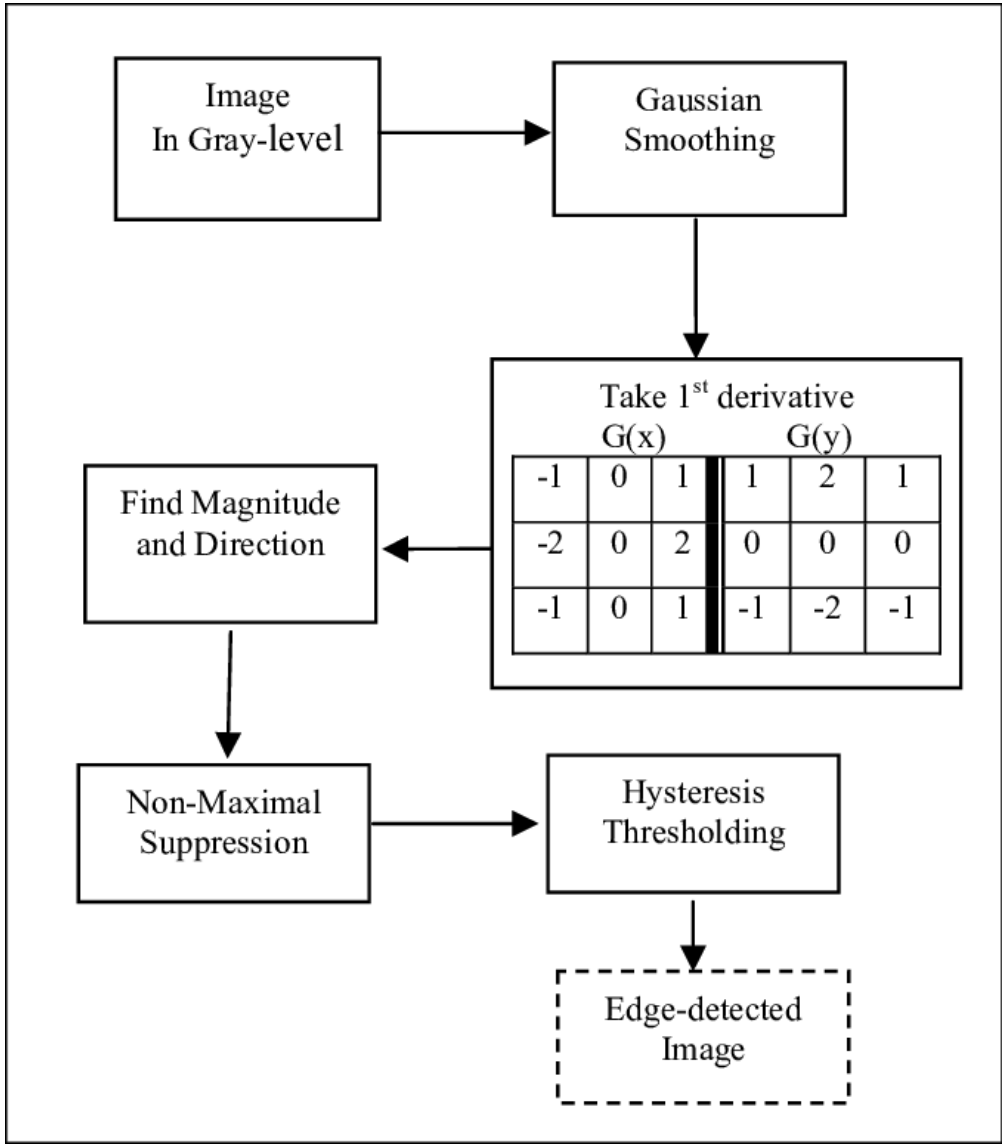
3. Sử dụng phương pháp đã học để khử nhiễu.
4. Áp dụng bộ lọc Lapacian trên ảnh đã làm trơn với ddepth = -1|CV_32FC4 và hiển thị kết quả. Xác định kSize phù hợp để cho ra kết quả tốt nhất!



4.2 Canny edge detection

Canny edge detection

- Phát hiện biên Canny do John Canny khởi xướng vào năm 1986. Là thuật toán được sử dụng phổ biến và hiệu quả trong việc xác định biên.
- Thuật toán bao gồm các bước:
  - Làm trơn ảnh với phương pháp Gaussian.
  - Lấy đạo hàm của ảnh theo chiều ngang và dọc.
  - Tính cường độ và hướng của Gradient
  - Non-Maximal Suppression để loại các pixel thừa.
  - Sử dụng ngưỡng để loại bỏ cạnh giả, xác định cạnh thực.



Canny edge detection

- Sử dụng phương thức Canny của OpenCV:

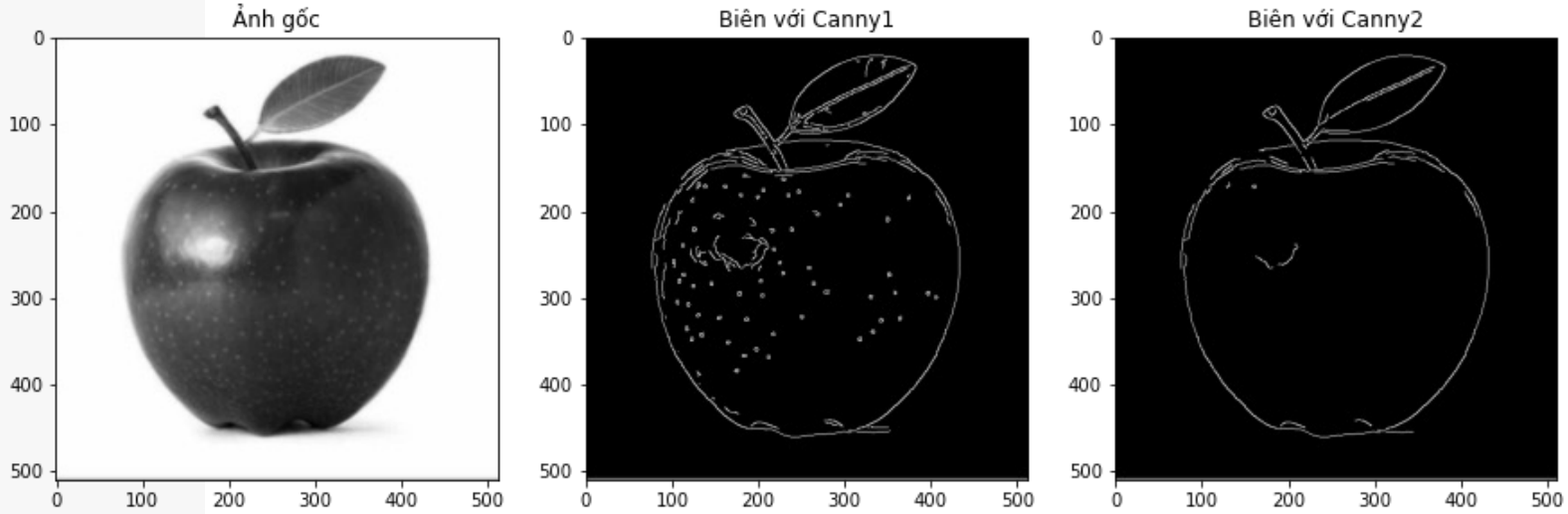
cv2.Canny(image, T_lower, T_upper, aperture_size, L2Gradient)

Trong đó:

1. Image: Input image to which Canny filter will be applied
2. T_lower: Lower threshold value in Hysteresis Thresholding
3. T_upper: Upper threshold value in Hysteresis Thresholding
4. aperture_size: Aperture size of the Sobel filter.
5. L2Gradient: Boolean parameter used for more precision in calculating Edge Gradient.

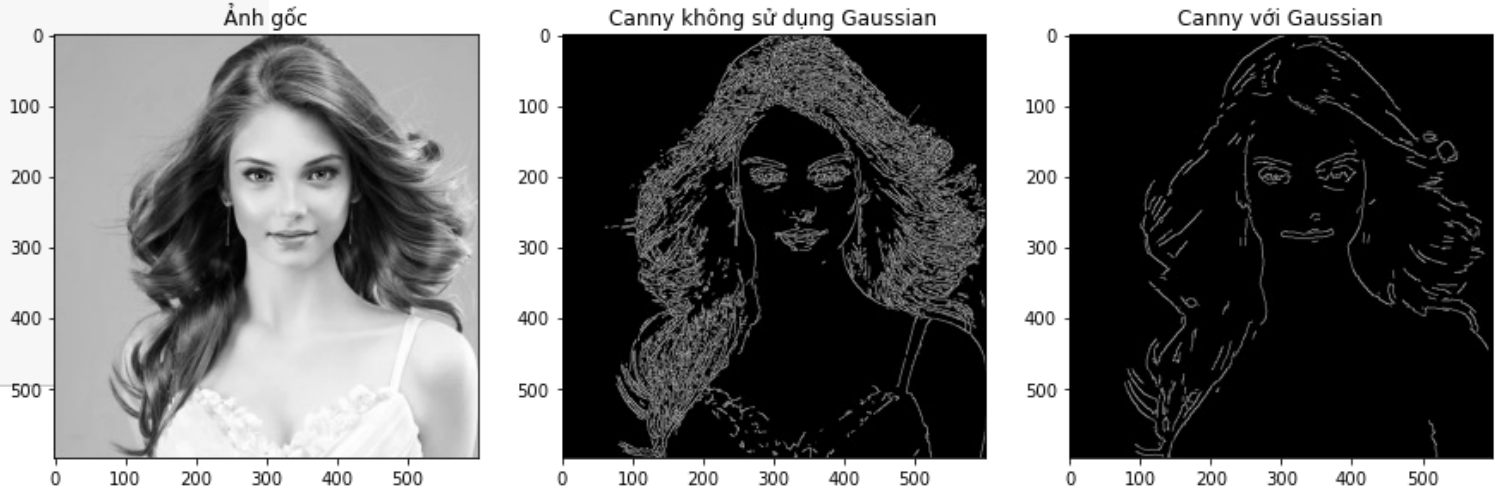
Canny edge detection

```python
#Làm trơn với Gaussian:
img_b = cv2.GaussianBlur(img_orignal,(5,5),0)

#Sử dụng phương thức canny:
img_canny1 = cv2.Canny(img_b, 50,100)
img_canny2 = cv2.Canny(img_b, 80,170)

#Hiển thị kết quả:
plt.figure(figsize=(15,5))
plt.subplot(1, 3, 1)

plt.imshow(img_b,cmap='gray')
plt.title('Ảnh gốc')

plt.subplot(1, 3, 2)
plt.imshow(img_canny1,cmap='gray')
plt.title('Biên với Canny1 ')

plt.subplot(1, 3, 3)
plt.imshow(img_canny2,cmap='gray')
plt.title('Biên với Canny2')

plt.show()
```



Canny edge detection

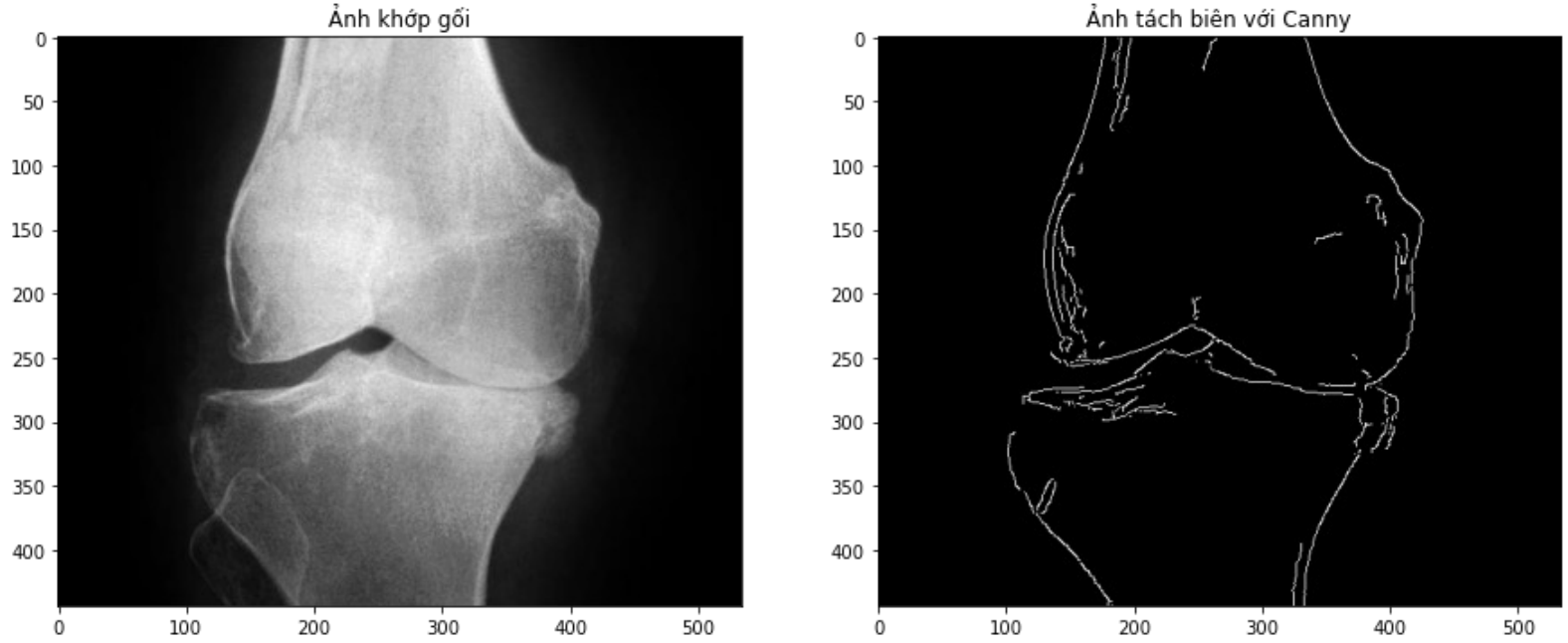
```python
#Tác dụng của làm mịn:
img_1 = cv2.imread('images/Thuchanh3_2.jpeg',0)
#Sử dụng phương thức canny
img_edge1 = cv2.Canny(img_1, 80,100)

#làm mịn với Gaussian:
img_2 = cv2.GaussianBlur(img_1,(7,7),10)
img_edge2 = cv2.Canny(img_2, 80,100)

#Hiển thị ảnh gốc và ảnh thay đổi kích thước
plt.figure(figsize=(15,5))
plt.subplot(1, 3, 1)
plt.imshow(img_1,cmap='gray')
plt.title('Ảnh gốc')

plt.subplot(1, 3, 2)
plt.imshow(img_edge1,cmap='gray')
plt.title('Canny không sử dụng Gaussian')

plt.subplot(1, 3, 3)
plt.imshow(img_edge2,cmap='gray')
plt.title('Canny với Gaussian')

plt.show()
```



Thực hành 3.4

Thực hành 3.4

1. Đọc và hiển thị ảnh Thuchanh3_4.jpeg.
2. Sử dụng phương pháp Canny với các tham số phù hợp để thu được kết quả biên như hình dưới đây.




QUESTIONS
Q&A
ANSWERS