

## Computer Vision

# THỊ GIÁC MÁY TÍNH

**ThS. Huỳnh Minh Vũ**

**Khoa Kỹ thuật cơ khí**

**Trường Đại học Kỹ thuật – Công nghệ Cần Thơ**

**Email: [hmvu@ctuet.edu.vn](mailto:hmvu@ctuet.edu.vn)**



# Chương 5: Các phương pháp phát hiện biên

---

## 5.1 Khái niệm vùng và biên

## 5.2 Quy tắc và quy trình phát hiện biên

## 5.3 Các phương pháp phát hiện biên

### 5.3.1 Phương pháp lọc đường biên cục bộ

Gradient

Laplace

Canny

### 5.3.2 Phương pháp lọc đường biên toàn cục

Hough

Contours

## 5.1 Khái niệm vùng và biên

### *Vùng*

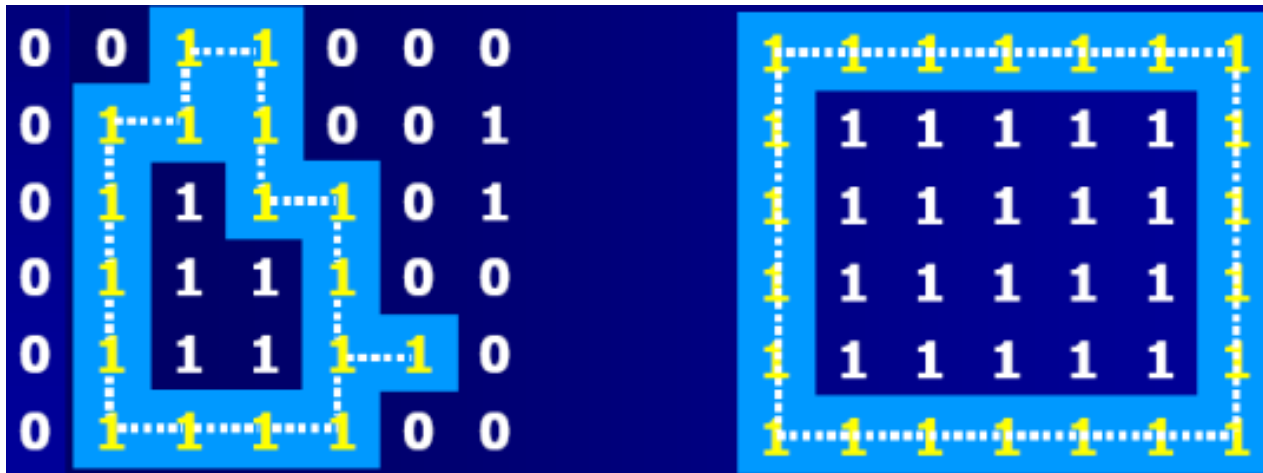
Cho  $R$  là một tập con của ảnh.  $R$  được gọi là vùng nếu  $R$  là một tập liên thông.



## 5.1 Khái niệm vùng và biên

### *Biên của vùng*

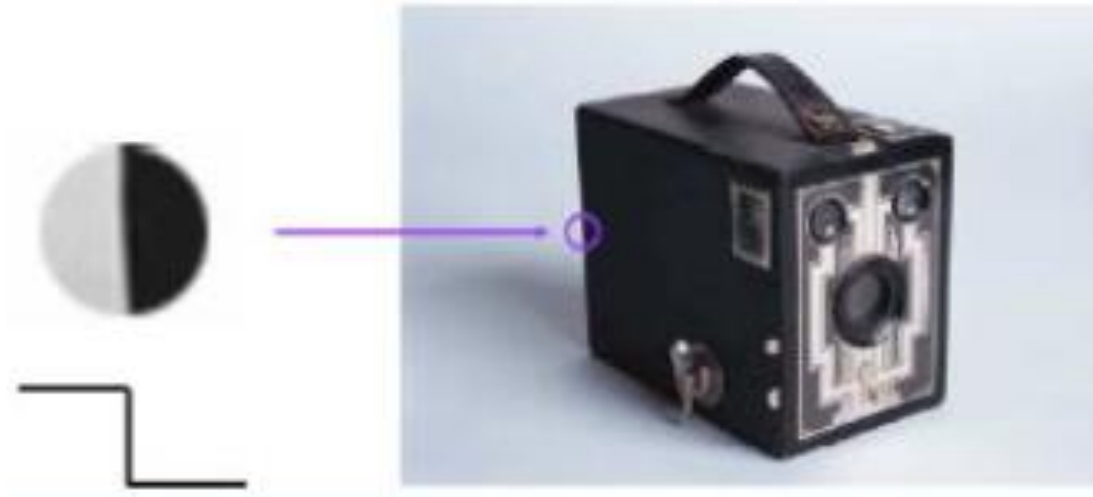
- Biên của vùng R là tập hợp các điểm trong vùng R mà có một hoặc nhiều lân cận không thuộc R.
- Nếu R phủ toàn ảnh thì biên của nó là dòng đầu tiên, cột đầu tiên, dòng cuối cùng, cột cuối cùng của ảnh.



## 5.1 Khái niệm vùng và biên

### *Khái niệm Biên*

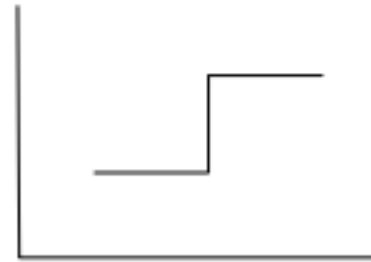
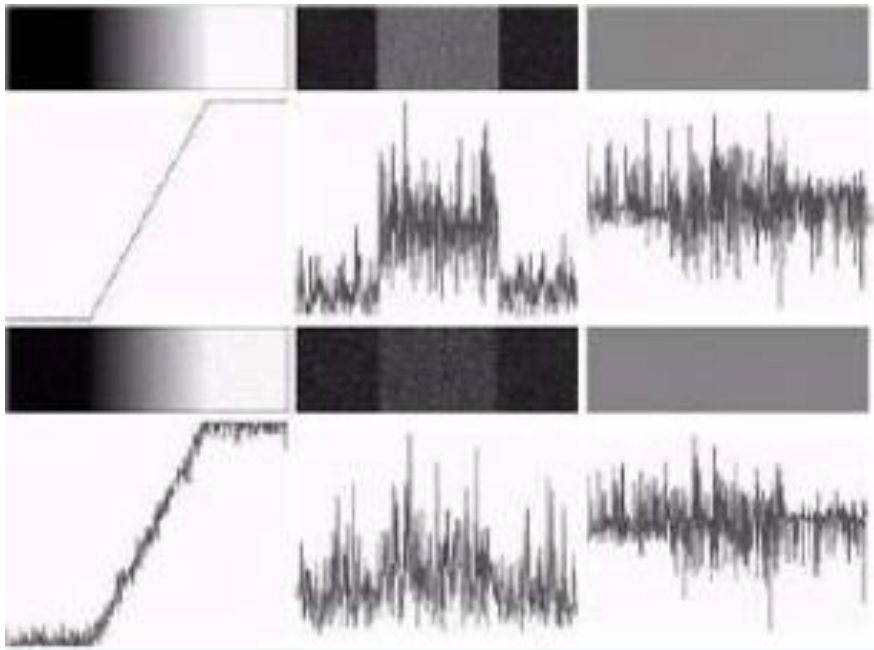
- Một điểm ảnh được gọi là biên nếu ở đó có sự thay đổi đột ngột về cấp xám.
- Tập hợp các điểm biên tạo thành một đường biên (đường bao) của ảnh.



## 5.1 Khái niệm vùng và biên

### *Các dạng đường Biên*

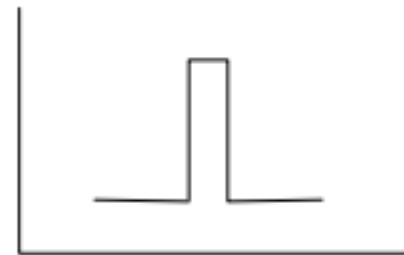
- Biên dạng nhảy bậc (a)
- Biên dạng dốc (b)
- Biên xung vuông (c)
- Biên dạng hình nón (d)



(a)



(b)



(c)



(d)

## 5.2 Quy tắc và quy trình phát hiện biên

### *Quy tắc phát hiện biên*

- Các phương pháp phát hiện biên truyền thống thường dựa trên kết quả của phép tích chập (convolution) giữa bức ảnh cần nghiên cứu  $f(x, y)$  và một bộ lọc 2D (filter)  $h(x, y)$  thường được gọi là mặt nạ.

$$h(x, y) * f(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} h(k_1, k_2) f(x - k_1, y - k_2) dk_1 dk_2$$

- Nếu  $h(x, y)$  và  $f(x, y)$  có dạng rời rạc thì sẽ được viết lại thành:

$$h(n_1, n_2) * f(n_1, n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} h(k_1, k_2) f(n_1 - k_1, n_2 - k_2)$$

- Trên thực tế người ta hay dùng  $h(n_1, n_2)$  là ma trận  $[3 \times 3]$  như sau:

$$h = \begin{pmatrix} h(-1, 1) & h(0, 1) & h(1, 1) \\ h(-1, 0) & h(0, 0) & h(1, 0) \\ h(-1, -1) & h(0, -1) & h(1, -1) \end{pmatrix}$$

## 5.2 Quy tắc và quy trình phát hiện biên

### *Quy trình phát hiện biên*

- Bước 1: Do ảnh ghi được thường có nhiễu, bước một là phải lọc nhiễu theo các phương pháp đã tìm hiểu ở các phần trước.
- Bước 2: Làm nổi biên sử dụng các toán tử phát hiện biên.
- Bước 3: Định vị biên. Chú ý rằng kỹ thuật nổi biên gây tác dụng phụ là gây nhiễu làm một số biên giả xuất hiện do vậy cần loại bỏ biên giả.
- Bước 4: Liên kết và trích chọn biên.



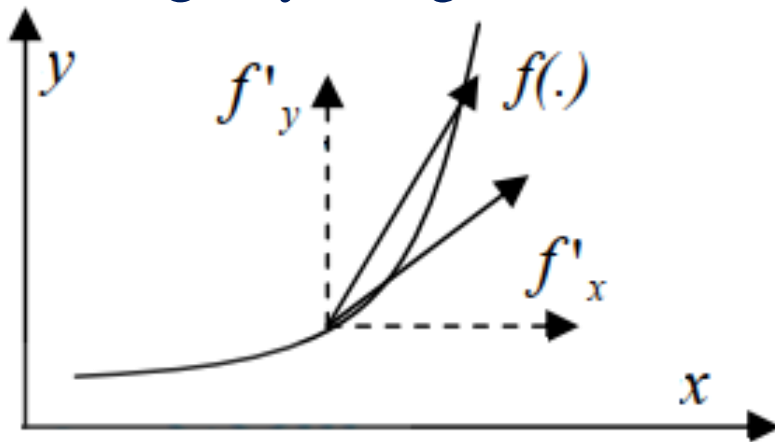
## 5.3 Các phương pháp phát hiện biên

- *Phương pháp lọc đường biên cục bộ*
  - ♦ Lọc biên theo phương pháp Gradient
    - Bộ lọc Roberts
    - Bộ lọc Sobel
    - Bộ lọc Prewitt
    - ...
  - ♦ Lọc biên theo phương pháp Laplace
  - ♦ Lọc biên theo phương pháp Canny
- *Phương pháp lọc đường biên toàn cục*
  - ♦ Phương pháp biến đổi Hough
  - ♦ Phương pháp xây dựng đường viền (Contours)

## 5.3 Các phương pháp phát hiện biên

### Lọc biên theo phương pháp Gradient

- *Gradient của ảnh (độ dốc của ảnh)*: là độ dốc về mức sáng của ảnh hay sự thay đổi các giá trị pixel trong ảnh.
- *Định nghĩa gradient*: Gradient là một vector  $f(x, y)$  có các thành phần biểu thị tốc độ thay đổi mức xám của điểm ảnh (theo hai hướng  $x, y$  trong bối cảnh xử lý ảnh hai chiều) tức là:



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

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

$$\nabla f = |\nabla f| = \sqrt{f'^2_x + f'^2_y} \quad \text{và} \quad \theta_r = \arctg\left(\frac{f'_x}{f'_y}\right)$$

Trong đó:  $dx, dy$  là khoảng cách giữa hai điểm kế cận theo hướng  $x, y$  tương ứng (thực tế chọn  $dx = dy = 1$ ). Đây là phương pháp dựa theo đạo hàm riêng bậc nhất theo hướng  $x, y$ .

## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Gradient*

#### ▪ *Kỹ thuật Gradient:*

- ♦ Theo định nghĩa về Gradient, nếu áp dụng nó vào xử lý ảnh, việc tính toán sẽ rất phức tạp. Để đơn giản mà không mất tính chất của phương pháp Gradient, người ta sử dụng kỹ thuật Gradient dùng cặp mặt nạ  $H_1, H_2$  theo 2 hướng vuông góc.
- ♦ Nếu định nghĩa  $g_1, g_2$  là Gradient theo hai hướng  $x, y$  tương ứng thì biên độ  $g(m,n)$  tại điểm  $(m,n)$  được tính theo công thức:

$$g(m,n) = \sqrt{g_1^2(m,n) + g_2^2(m,n)} = A_0$$

- ♦ Để giảm độ phức tạp khi tính toán có thể áp dụng công thức:

$$A_0 = |g_1(m,n)| + |g_2(m,n)|$$

- ♦ Hướng của Gradient được xác định theo công thức:

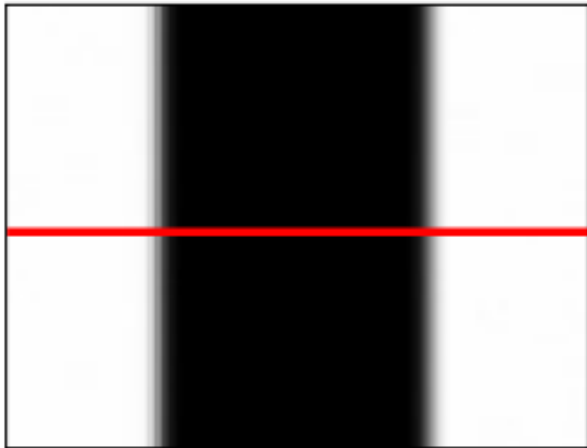
$$\theta_r(m,n) = \text{artg}(g_2(m,n))$$

## 5.3 Các phương pháp phát hiện biên

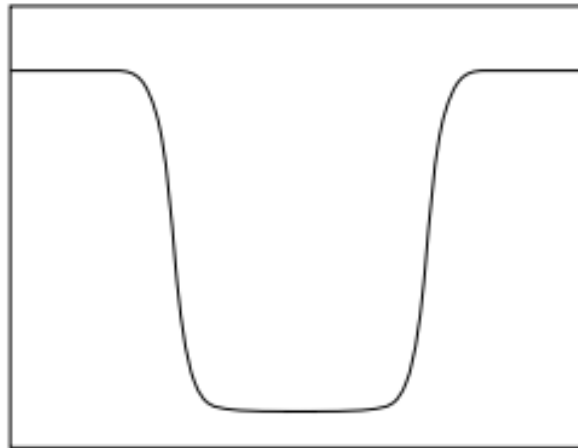
### *Lọc biên theo phương pháp Gradient*

- *Liên hệ giữa đạo hàm và biên ảnh*

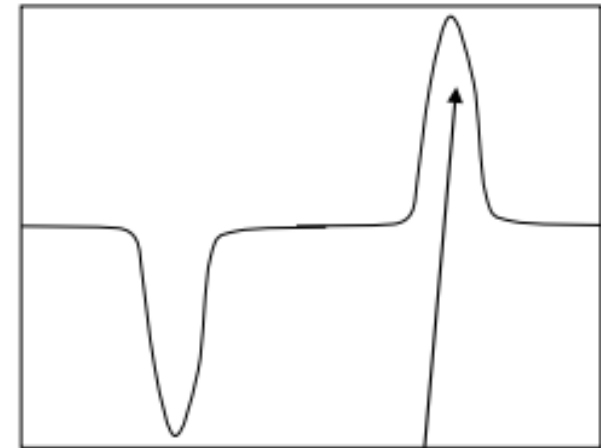
Hình ảnh



Dãy mức xám tại  
đường màu đỏ



Đạo hàm bậc nhất



Các cạnh trong ảnh ứng với  
các cực trị của đồ thị

## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Gradient*

#### ■ *Bộ lọc Roberts:*

- Năm 1965 Roberts đưa ra xấp xỉ đầu tiên theo đạo hàm bậc nhất cho một ảnh rời rạc.
- Việc tính toán được thực hiện trên 2 mặt nạ tích chập cho 2 hướng lấy vi phân:

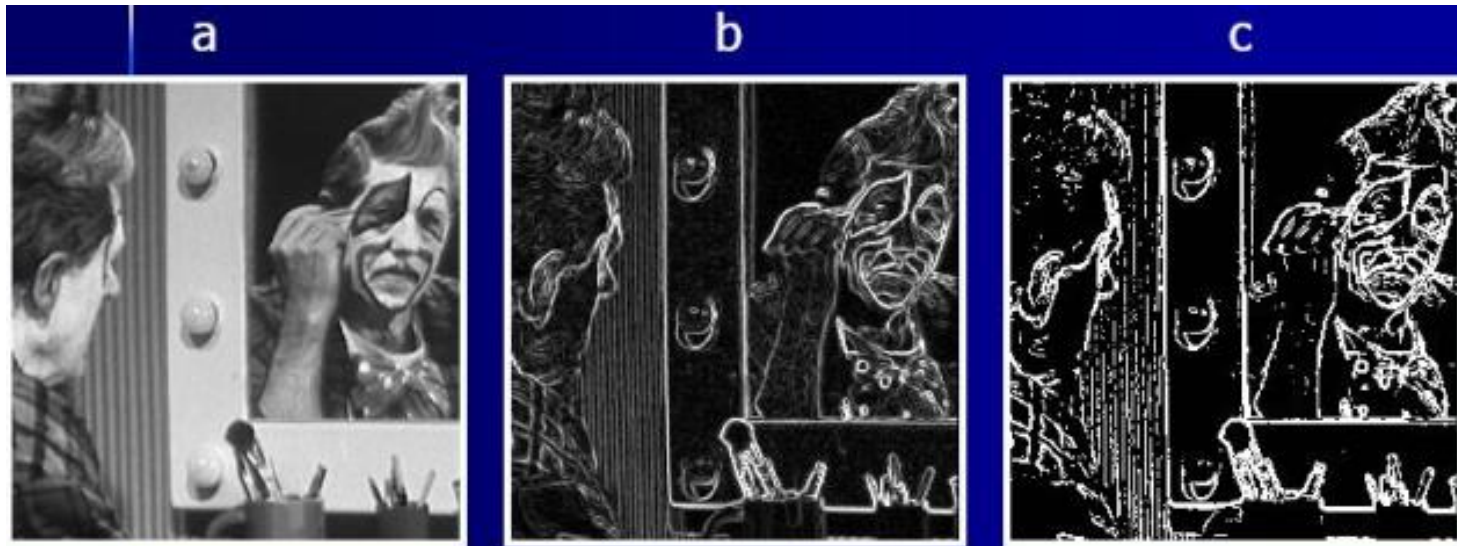
$$H_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad H_y = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}.$$

- Toán tử Roberts có tốc độ tính toán nhanh.
- Chỉ cần sử dụng 4 điểm ảnh để tính giá trị cấp xám của ảnh đầu ra.
- Chỉ phép toán cộng và trừ được thực hiện trong ảnh.

## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Gradient*

- *Bộ lọc Roberts:*



a) Ảnh gốc

b) Ảnh sau khi áp dụng toán tử Roberts

c) Ảnh sau khi phân ngưỡng ảnh b)

## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Gradient*

- *Bộ lọc Sobel:*

- Toán tử Sobel được Duda và Hart đặt ra năm 1973 với các mặt nạ tương tự như của Robert nhưng cấu trúc khác, như sau:

$$H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Công thức tính cụ thể như sau:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{I} \quad \text{và} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{I}$$

$$\mathbf{G}_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * ([+1 \quad 0 \quad -1] * \mathbf{I}) \quad \text{và} \quad \mathbf{G}_y = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix} * ([1 \quad 2 \quad 1] * \mathbf{I})$$

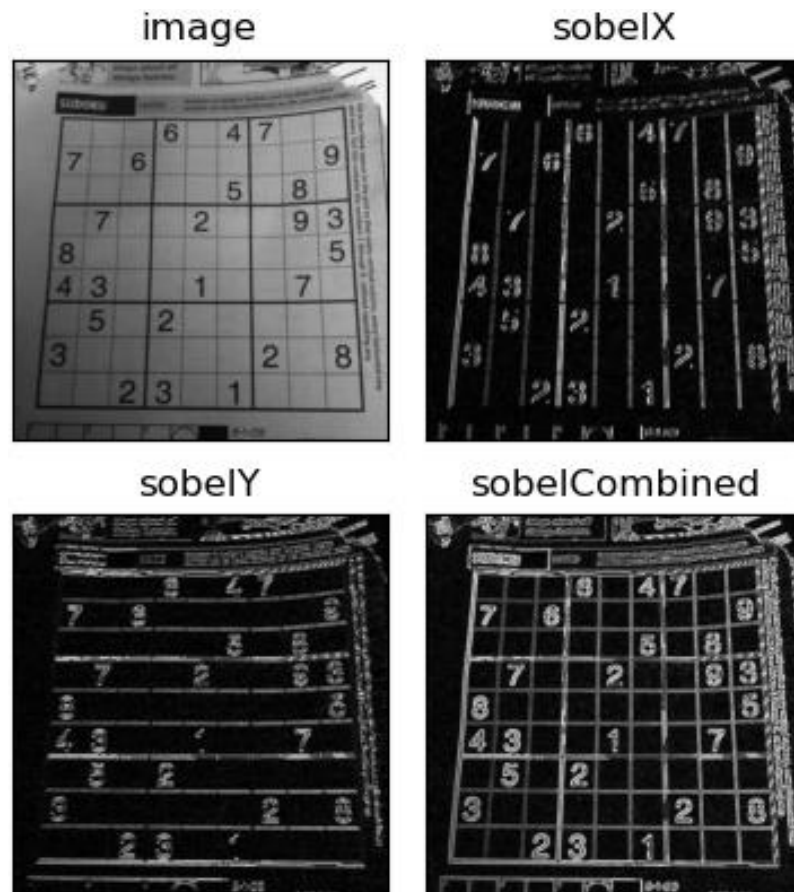
$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2} \quad \text{và} \quad \Theta = \text{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

## 5.3 Các phương pháp phát hiện biên

### Lọc biên theo phương pháp Gradient

#### ▪ Bộ lọc Sobel:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread("Sudoku.jpg", 0)
sobelX = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
sobelY = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
sobelX = np.uint8(np.absolute(sobelX))
sobelY = np.uint8(np.absolute(sobelY))
sobelCombined = cv2.bitwise_or(sobelX, sobelY)
titles = ['image', 'sobelX', 'sobelY', 'sobelCombined']
images = [img, sobelX, sobelY, sobelCombined]
for i in range(4):
    plt.subplot(2, 2, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```





## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Gradient*

- *Bộ lọc Prewitt:*

- Toán tử được Prewitt đưa ra vào năm 1970 có dạng:

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

Image\_Gray



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

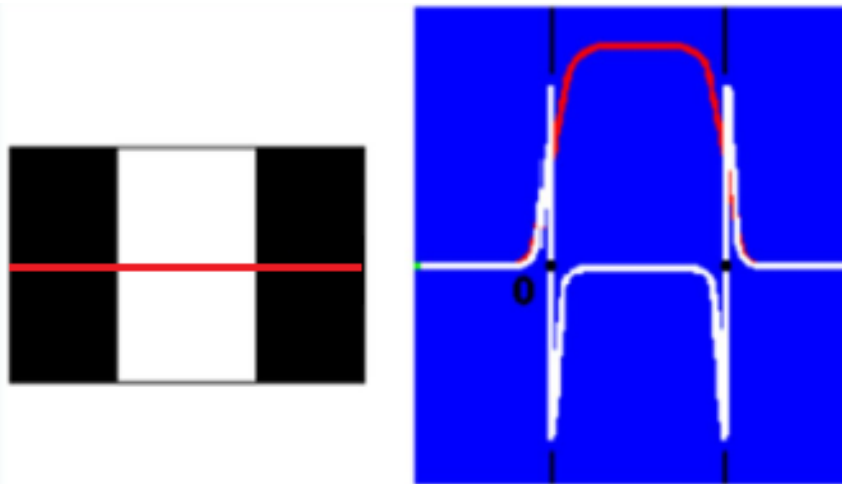
Prewitt



## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Laplace*

- Các phương pháp đánh giá gradient ở trên làm việc khá tốt khi mà độ sáng thay đổi rõ nét. Khi mức xám thay đổi chậm, miền chuyển tiếp trải rộng, phương pháp cho hiệu quả hơn đó là phương pháp sử dụng đạo hàm bậc hai Laplace.
- Phép tính Laplace là tính xấp xỉ đạo hàm bậc hai trong ảnh, nó có ý nghĩa quan trọng trong việc tìm biên ảnh, phân tích và ước lượng chuyển động của vật thể.



## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Laplace*

- Laplace của điểm ảnh có giá trị cường độ pixel  $f(x, y)$  được định nghĩa như sau:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- Khi đó mặt nạ (ksize = 1) là:

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

- Trong thực tế, người ta thường dùng nhiều kiểu mặt nạ khác nhau để xấp xỉ rời rạc đạo hàm bậc hai Laplace. Dưới đây là ba kiểu mặt nạ thường dùng:

$$H_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix};$$

$$H_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix};$$

$$H_3 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -1 \\ 1 & -2 & 1 \end{bmatrix}$$

## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Laplace*

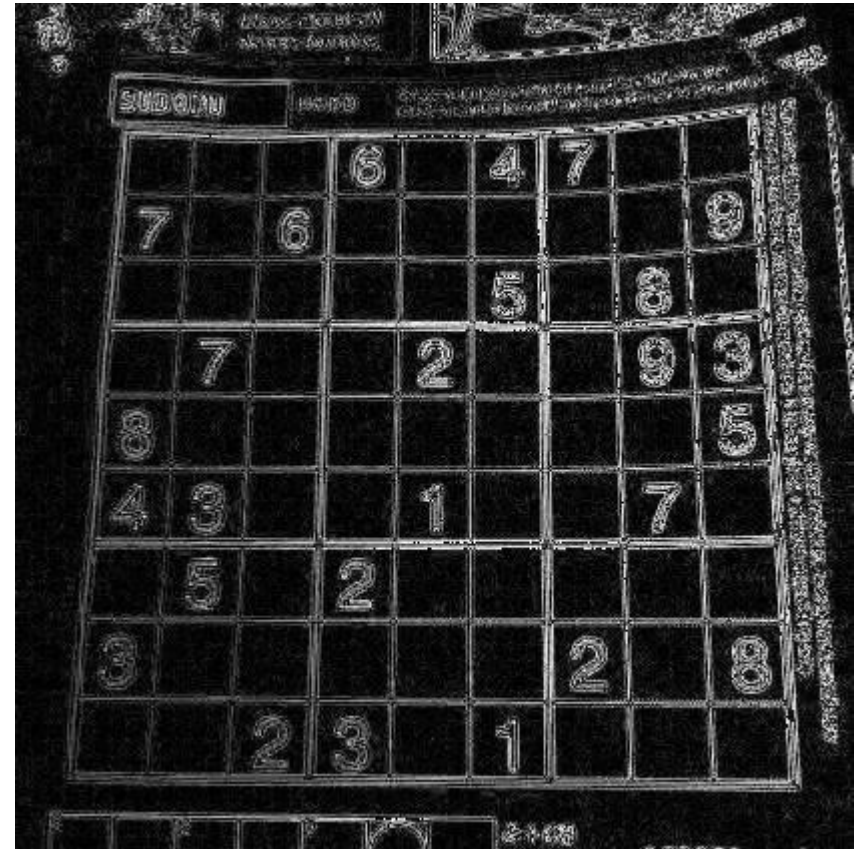
```
import cv2
import numpy as np

img = cv2.imread("Sudoku.jpg", 0)

lap = cv2.Laplacian(img, cv2.CV_64F, ksize=3)
lap1 = np.uint8(np.absolute(lap))

cv2.imshow('Image', img)
cv2.imshow('Image_Lap', lap)
cv2.imshow('Image_Lap1', lap1)

cv2.waitKey()
cv2.destroyAllWindows()
```



## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Canny*

- Thuật toán Canny (Canny Edge Detection) là một giải thuật phát hiện cạnh nổi tiếng được phát triển năm 1986 bởi John. F Canny.
- Đây là một thuật toán được phát triển khá sớm nhưng cho đến nay vẫn là một trong những kỹ thuật được sử dụng rộng rãi, cho các kết quả tương đối tốt, có khả năng đưa ra đường biên mảnh, phân biệt được điểm biên với điểm nhiễu.
- Phương pháp này sử dụng hai mức ngưỡng cao và thấp. Ban đầu ta dùng mức ngưỡng cao để tìm điểm bắt đầu của biên, sau đó chúng ta xác định hướng phát triển của biên dựa vào các điểm ảnh liên tiếp có giá trị lớn hơn mức ngưỡng thấp. Ta chỉ loại bỏ các điểm có giá trị nhỏ hơn mức ngưỡng thấp. Các đường biên yếu sẽ được chọn nếu chúng được liên kết với các đường biên khỏe.

## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Canny*

#### ▪ **Bước 1: Khử nhiễu (làm trơn ảnh):**

- Việc phát hiện ảnh bị ảnh hưởng bởi nhiễu, nên việc đầu tiên là loại bỏ nhiễu.
- Sử dụng bộ lọc Gauss 5x5 để khử nhiễu.

#### ▪ **Bước 2: Tìm Gradient của ảnh:**

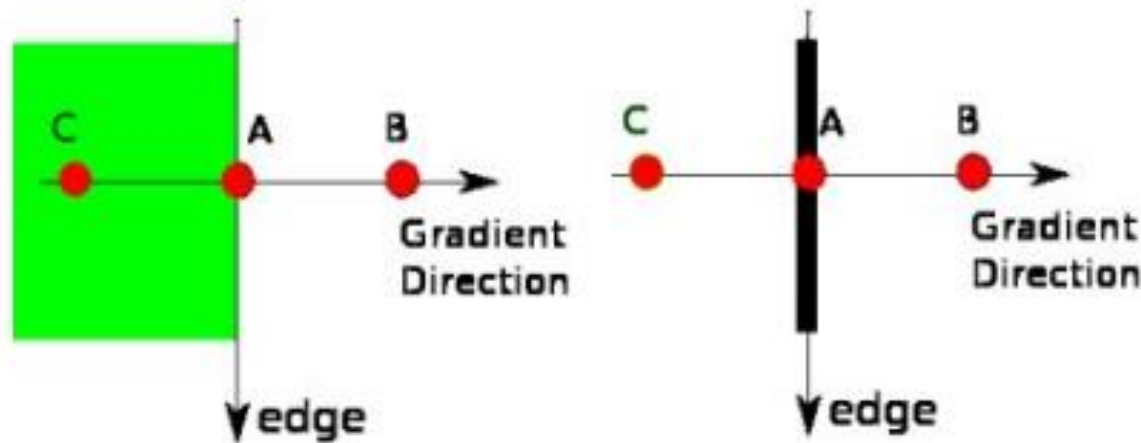
- Ảnh sau khi được làm mịn sẽ được nhân chập với Sobel Kernel để tìm đạo hàm bậc nhất theo cả chiều dọc ( $G_x$ ) và chiều ngang ( $G_y$ ).
- Từ hai thành phần đó, chúng ta có thể tìm giá trị edge gradient và hướng gradient (luôn vuông góc với cạnh) cho mỗi điểm ảnh như sau:

$$\text{Gradient}(G) = \sqrt{G_x^2 + G_y^2}$$
$$\text{Angle}(\theta) = \tan^{-1} \left( \frac{G_y}{G_x} \right)$$

## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Canny*

- **Bước 3: Triệt tiêu phi cực đại (Non-maximum Suppression):**
  - Bước này loại bỏ những điểm không phải là cực đại địa phương để xóa bỏ những điểm không thực sự là biên, bước này sẽ giúp biên mỏng hơn.
  - Nếu điểm ảnh là cực bộ địa phương thì tiến hành bước tiếp theo.

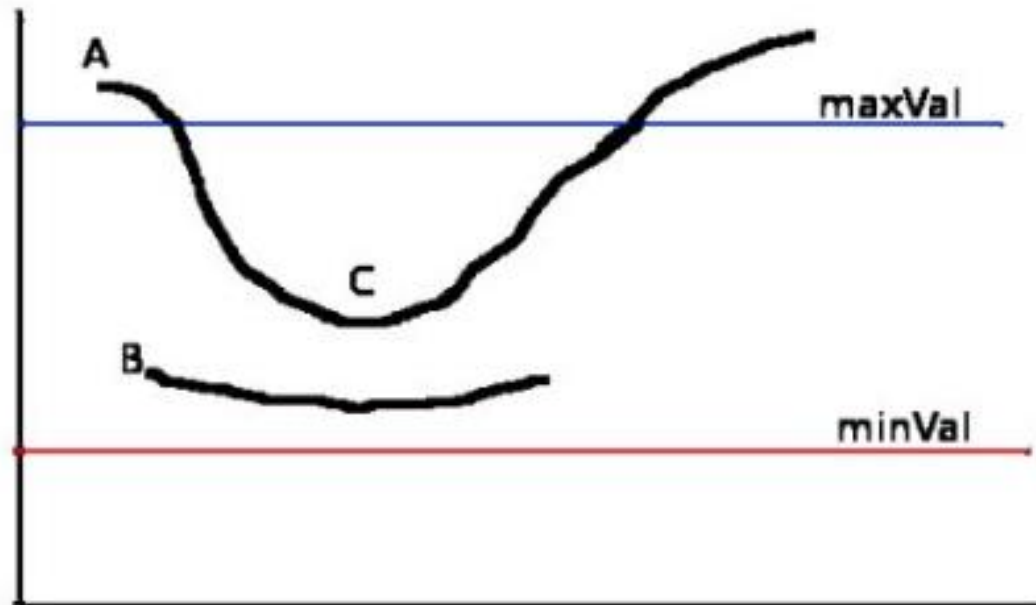


## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Canny*

#### ▪ **Bước 4: Ngưỡng độ trễ (Hysteresis Thresholding):**

- Bước này xét xem trong các cạnh tìm được ở bước trước cái nào thực sự là cạnh, cái nào không phải.
- Canny làm điều này bằng cách sử dụng giá trị ngưỡng dưới (minVal) và ngưỡng trên (maxVal).





## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Canny*

#### ▪ **Bước 4: Ngưỡng độ trễ (Hysteresis Thresholding):**

- Bất kỳ điểm ảnh nào có độ lớn của gradient lớn hơn  $\maxVal$  thì được xem là chắc chắn thuộc cạnh, nhỏ hơn  $\minVal$  thì được xem là chắc chắn không thuộc cạnh.
- Những điểm ảnh còn lại sẽ được xét dựa trên kết nối của nó với các điểm ảnh đã được phân loại. Nếu nó có kết nối với một điểm ảnh chắc chắn thuộc cạnh nó sẽ được xem là một phần của cạnh. Ngược lại, nó không thuộc cạnh.
- Như hình minh họa, điểm A là lớn hơn  $\maxVal$  nên nó được xem chắc chắn là cạnh. Mặc dù điểm C là ở giữa  $\maxVal$  và  $\minVal$  nhưng nó có kết nối với điểm A nên nó cũng được xem là thuộc cạnh. Điểm B cũng nằm giữa  $\maxVal$  và  $\minVal$  như C nhưng nó không có kết nối với bất kỳ điểm nào chắc chắn thuộc cạnh nên nó bị loại bỏ.

## 5.3 Các phương pháp phát hiện biên

### *Lọc biên theo phương pháp Canny*

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('Lena.png', 0)
edges = cv2.Canny(img, 100, 200)

plt.subplot(211), plt.imshow(img, cmap='gray')
plt.title('Gray Image'), plt.xticks([]), plt.yticks([])

plt.subplot(212), plt.imshow(edges, cmap='gray')
plt.title('Canny Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

Gray Image



Canny Image



## 5.3 Các phương pháp phát hiện biên

### *Đánh giá các phương pháp lọc đường biên cục bộ*

Để đánh giá các phương pháp phát hiện đường biên cục bộ chúng ta sử dụng phương pháp Sobel đại diện cho phương pháp Gradient, phương pháp Laplace đại diện cho phương pháp đạo hàm bậc 2 và phương pháp Canny với các giá trị  $\text{minVal}=100$ ,  $\text{maxVal}=200$ .

#### ▪ **Đối với ảnh không nhiễu:**

- Cả 3 phương pháp đều cho kết quả tốt.
- Sobel cho ảnh đường biên rõ nét nhưng lớn.
- Laplace cho ảnh đường biên rõ nét, đường biên mảnh.
- Canny do quá trình làm trơn ảnh nên ảnh không nhiễu thì biên ảnh bị mờ và to ra => Biên ảnh bị thiếu và đường biên lớn =>

**Không phù hợp.**

## 5.3 Các phương pháp phát hiện biên

### *Đánh giá các phương pháp lọc đường biên cục bộ*

#### ▪ **Đối với ảnh bị nhiễu:**

- Phương pháp đạo hàm bậc nhất cho biên ảnh với nhiễu điểm biên phụ.
- Phương pháp Laplace tạo biên kép, nên hoàn toàn không xác định được biên => **Không phù hợp**.
- Đối với phương pháp Canny do quá trình làm trơn ảnh để giảm nhiễu và quá trình triệt tiêu các phi cực đại nên sẽ giảm được các biên phụ và cho kết quả đường biên rất rõ nét.

## 5.3 Các phương pháp phát hiện biên

### *Đánh giá các phương pháp lọc đường biên cục bộ*

#### ▪ **Đối với ảnh có nhiều cạnh:**

- Phương pháp Sobel cho ảnh biên mờ, không rõ nét do trong ảnh có các vùng mức xám thấp, sự sai khác giữa các mức xám nhỏ => **Không phù hợp**.
- Phương pháp Laplace cho đường biên rõ nét hơn (do sử dụng phương pháp đạo hàm bậc 2, các điểm biên là điểm cắt không). Tuy nhiên, ảnh có rất nhiều điểm biên nhỏ => điểm biên rất nhiều và rối.
- Phương pháp Canny do quá trình triệt tiêu các phi cực đại và phân ngưỡng kép nên biên phụ bị loại bớt đi và biên chính được giữ lại và rõ nét hơn.
- Nếu sự biến thiên mức xám thấp => dùng Laplace.
- Ngược lại, nếu ảnh có quá nhiều biên => dùng Canny.

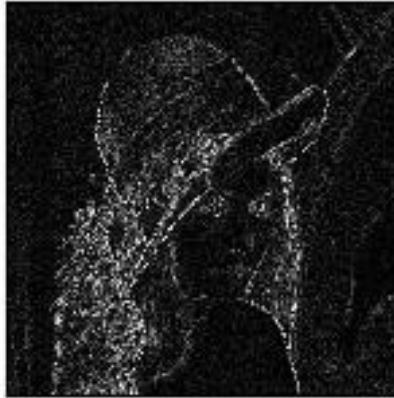
## 5.3 Các phương pháp phát hiện biên

*So sánh các phương pháp lọc đường biên cục bộ*

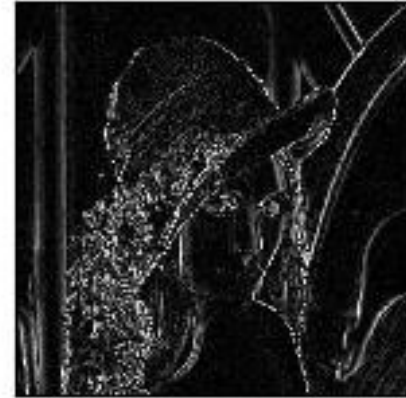
image



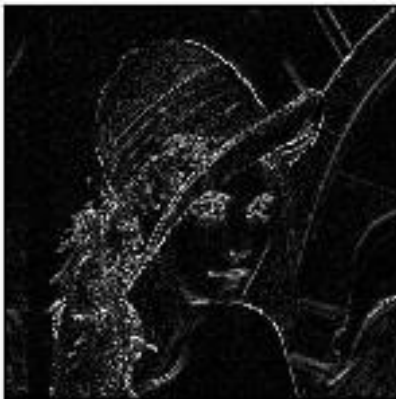
Laplacian



sobelX



sobelY



sobelCombined



Canny

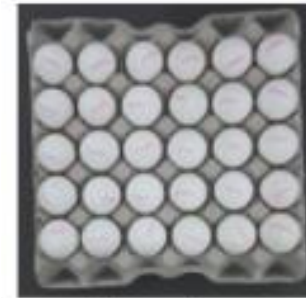
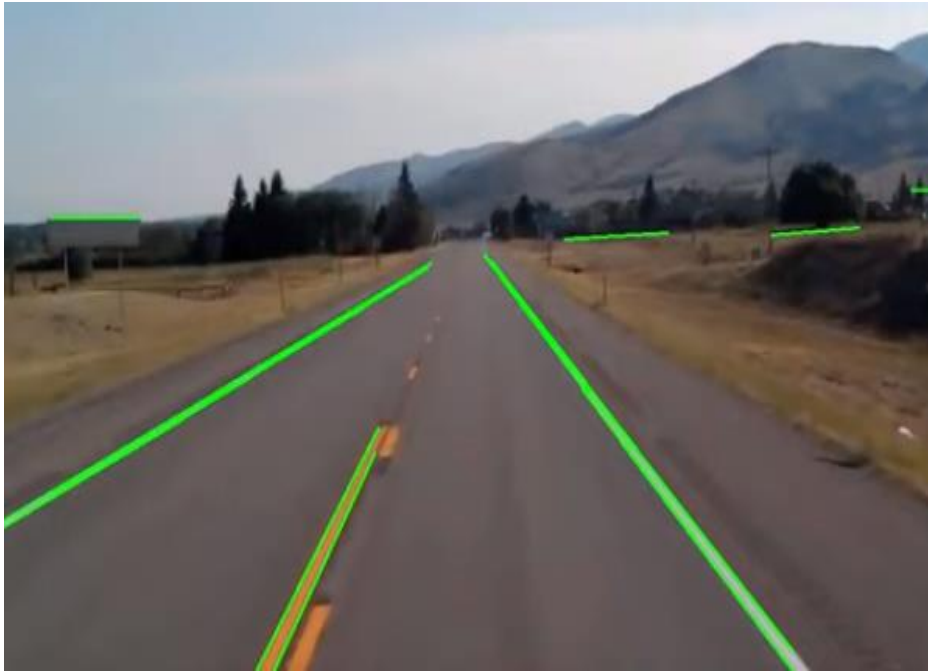




## 5.3 Các phương pháp phát hiện biên

### *Phương pháp biến đổi Hough (hough transform)*

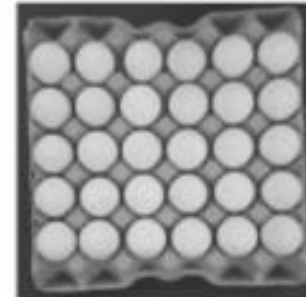
- Hough Line transform
- Hough Circle transform



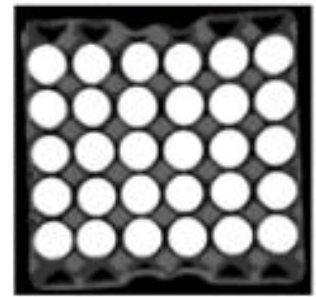
a) Source Image



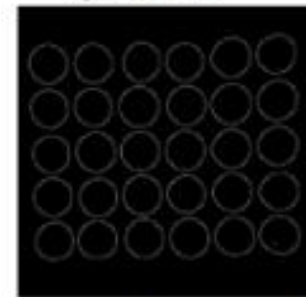
b) Gaussian Filter



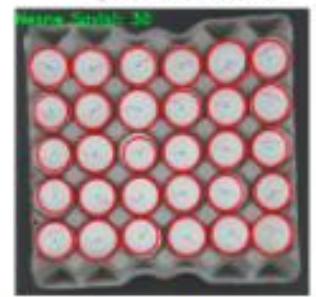
c) S Channel



d) Otsu Threshold



e) Sobel Edge Detection



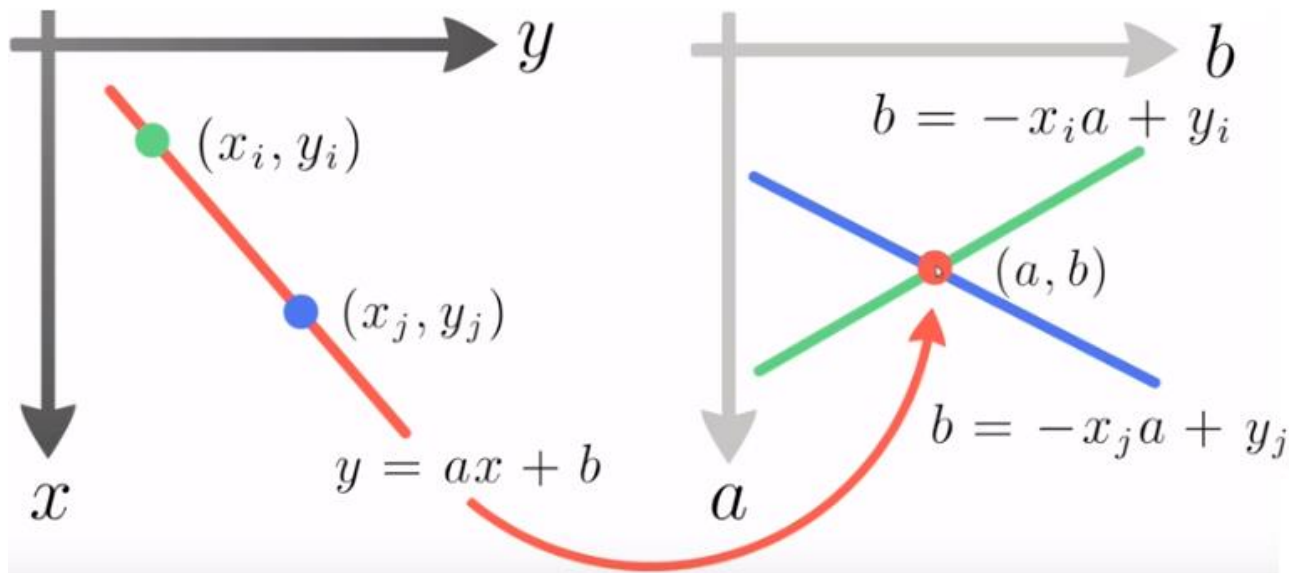
f) Hough Transform

## 5.3 Các phương pháp phát hiện biên

### *Hough Line transform*

*Nguyên tắc:* tìm các điểm (thuộc biên vừa tìm được trước đó) có nhiều đường thẳng đi qua.

- Trong hệ tọa độ Descartes, đường thẳng được mô tả bởi phương trình  $y = a.x + b$ , với  $x, y$  là biến số (trục tọa độ);  $a, b$  là tham số.
- Với biến đổi Hough, ta coi  $x, y$  là tham số và  $a, b$  là biến số. Khi đó ta có phương trình  $b = -x.a + y$ .

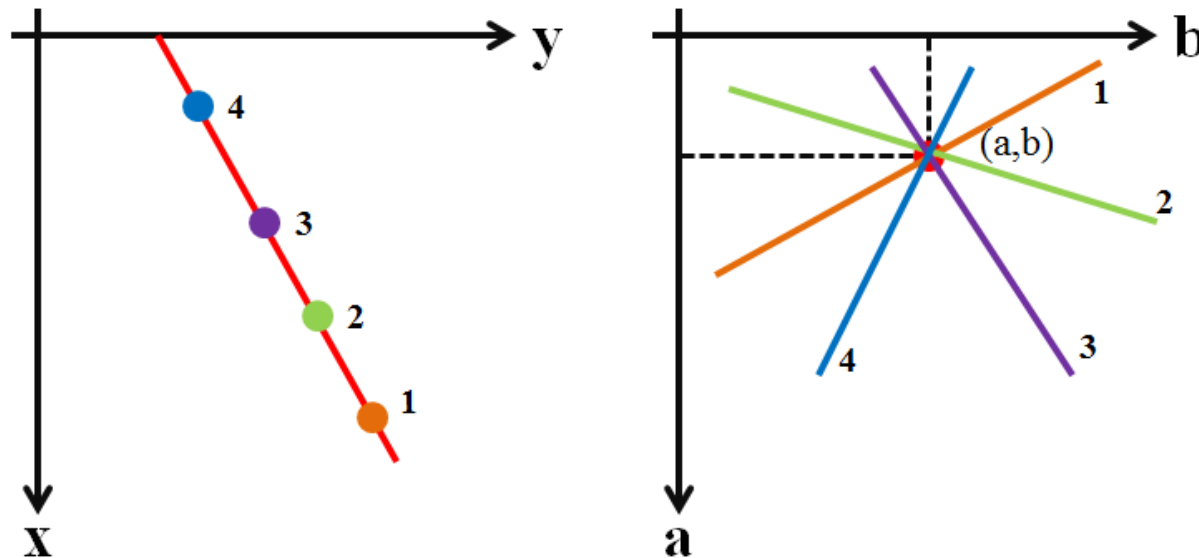




## 5.3 Các phương pháp phát hiện biên

### *Hough Line transform*

- Mỗi điểm trong không gian  $(x, y)$  ta vẽ được một đường thẳng trong không gian  $(a, b)$ .
- Nếu các đường này giao nhau cùng tại 1 điểm, chứng tỏ có 1 đường thẳng đi qua các điểm đó.
- Điểm nào có nhiều đường thẳng đi qua nhất sẽ được chọn, suy ngược lại ta sẽ có đường thẳng trong không gian  $(x, y)$

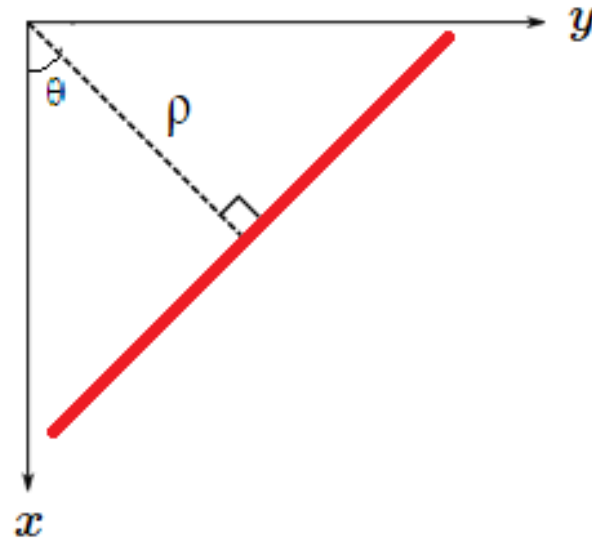


## 5.3 Các phương pháp phát hiện biên

### *Hough Line transform*

- Khi đường thẳng thẳng đứng thì giá trị  $b$  tiến về vô cực, do đó đường thẳng có thể biểu diễn dưới dạng tọa độ cực.

$$\rho = x.\cos \theta + y.\sin \theta$$



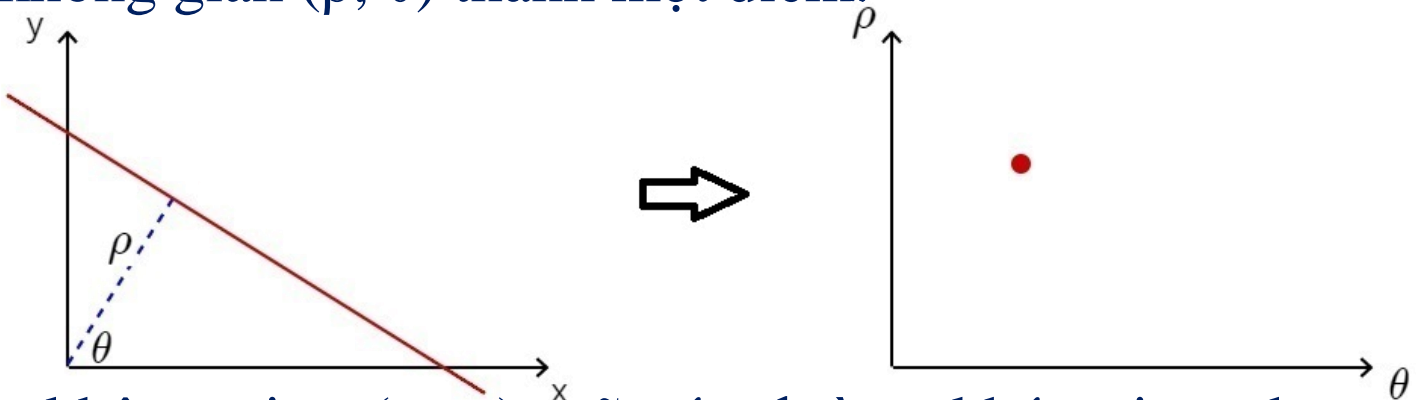
Trong đó:

- $\rho$  là khoảng cách từ đường thẳng tới gốc tọa độ.
- $\theta$  là góc giữa trục hoành và đoạn thẳng ngắn nhất nối tới gốc tọa độ (đơn vị là radian).

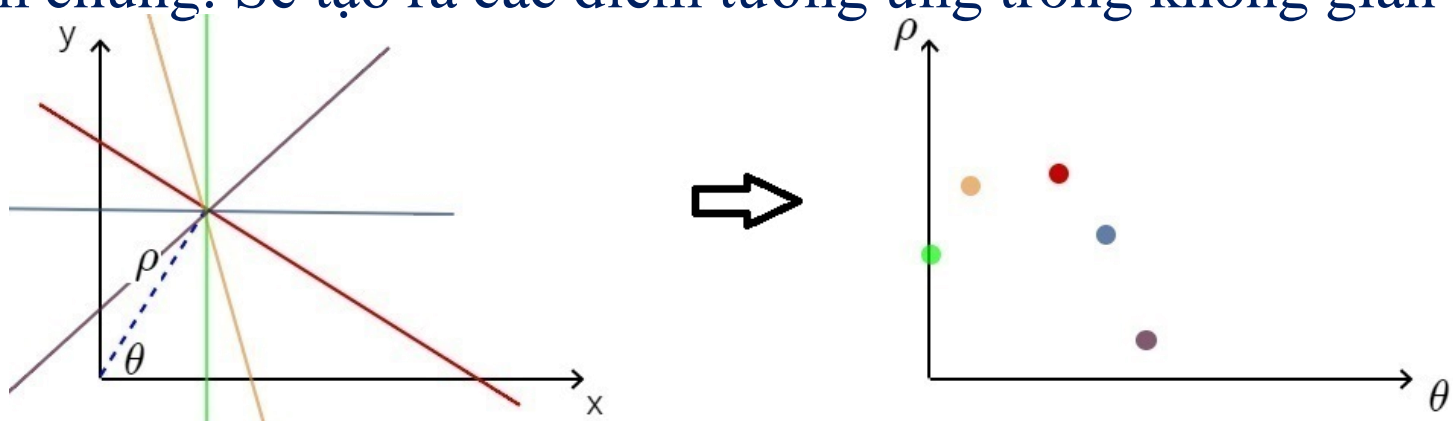
## 5.3 Các phương pháp phát hiện biên

### *Hough Line transform*

- Từ một đường thẳng trong không gian  $(x, y)$ , chúng ta sẽ chuyển sang không gian  $(\rho, \theta)$  thành một điểm.



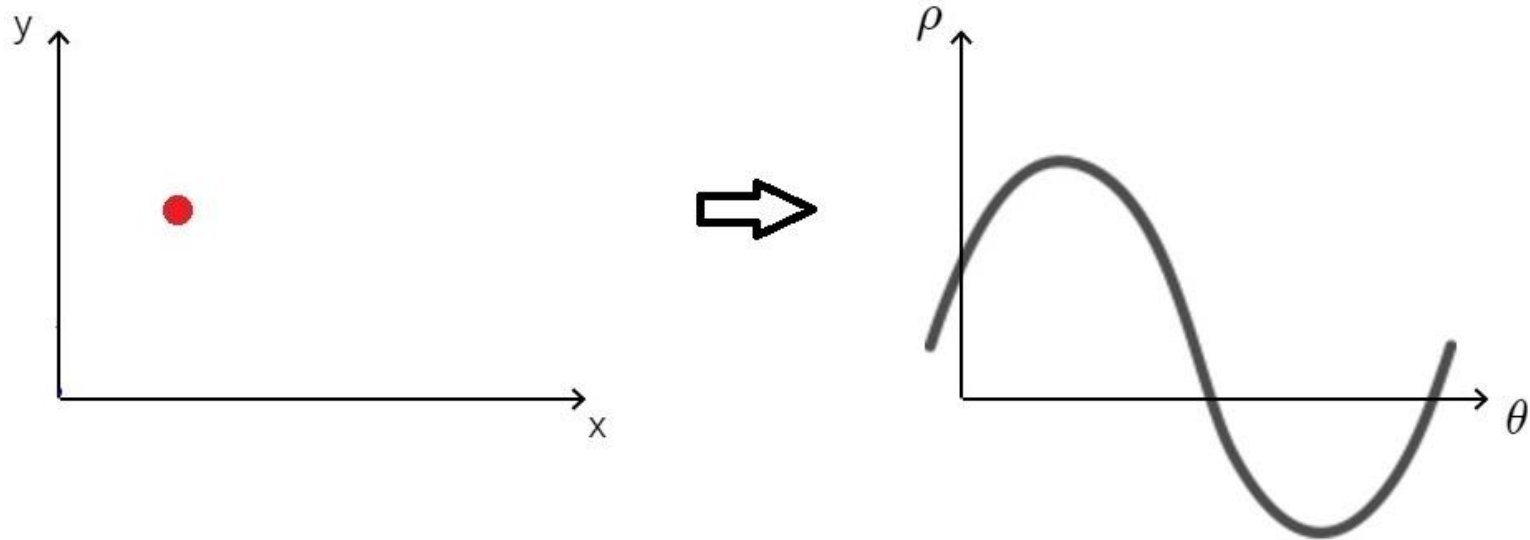
- Trong không gian  $(x, y)$ , vẽ các đường khác giao nhau tại một điểm chung. Sẽ tạo ra các điểm tương ứng trong không gian  $(\rho, \theta)$ .



## 5.3 Các phương pháp phát hiện biên

### *Hough Line transform*

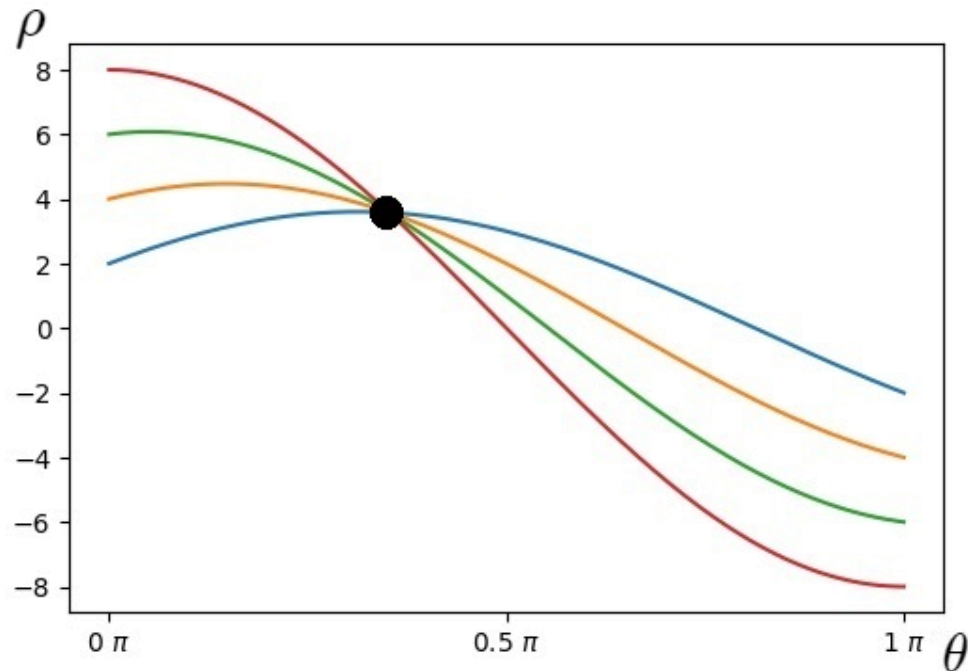
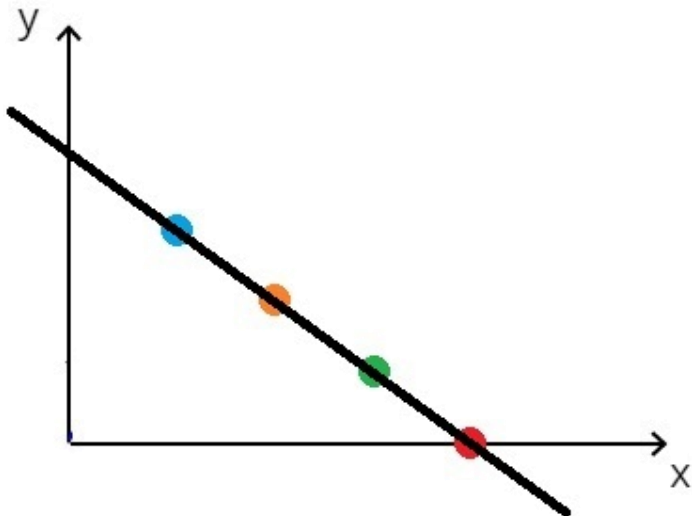
- Từ việc xoay liên tục đường thẳng quanh một điểm trong không gian  $(x, y)$ , sẽ tạo ra các điểm tương ứng trong không gian  $(\rho, \theta)$ , liên kết các điểm tương ứng trên ta sẽ được đường cong hình sin.



## 5.3 Các phương pháp phát hiện biên

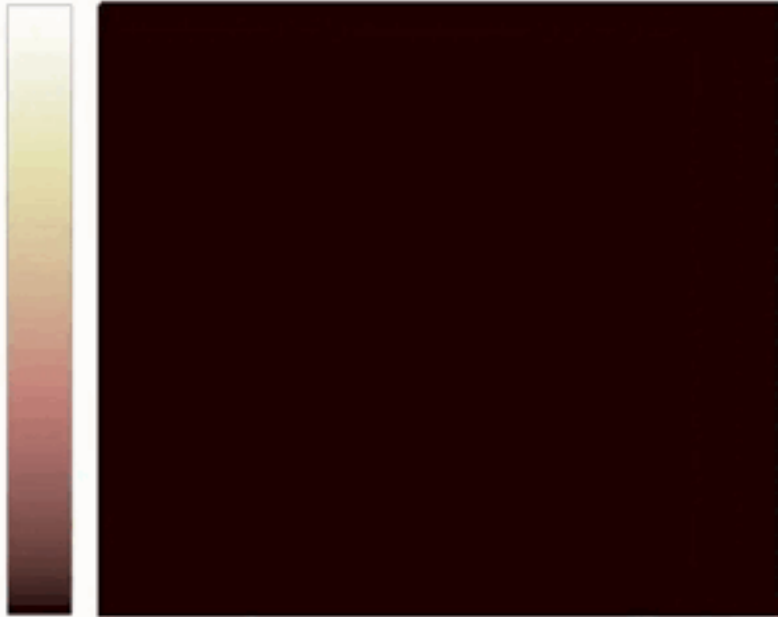
### *Hough Line transform*

- Vẽ các điểm tạo thành một đường trong không gian  $(x, y)$ , sẽ thu được một loạt các hình sin trong không gian  $(\rho, \theta)$ . Chúng sẽ giao nhau tại đúng một điểm.
- Nếu điểm có nhiều đường cong hình sin đi qua nhất sẽ được chọn. Suy ngược lại ta sẽ có đường thẳng trong không gian  $(x, y)$ .

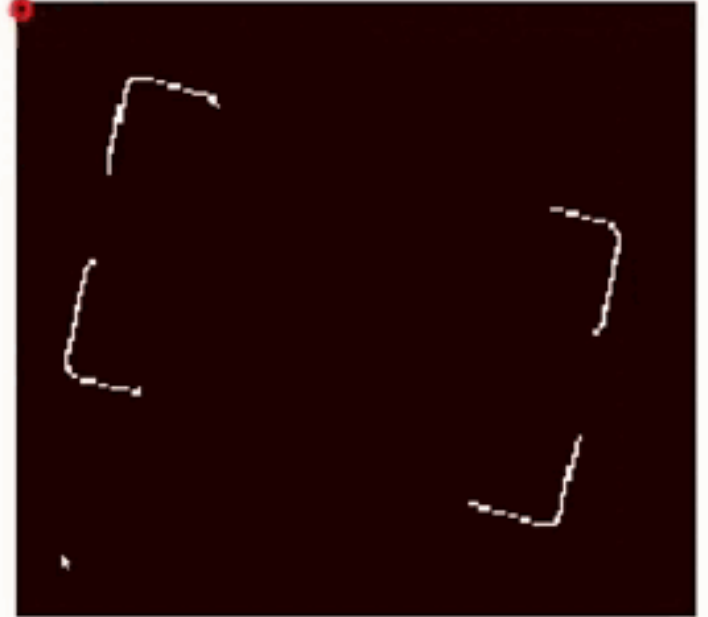


## 5.3 Các phương pháp phát hiện biên

### *Hough Line transform*



input image



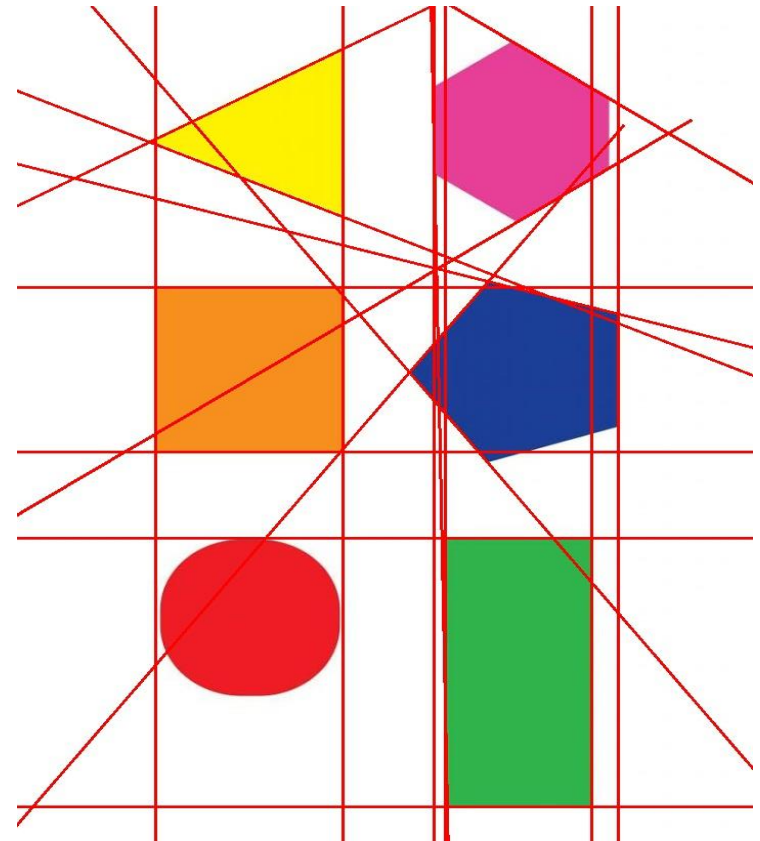
## 5.3 Các phương pháp phát hiện biên

### *Hough Line transform*

- **Ưu điểm:** Phát hiện được đường biên bị che khuất.
- **Nhược điểm:** chỉ cho kết quả là các đường thẳng, không phải là các đoạn thẳng đường biên.

```
import cv2
import numpy as np
img = cv2.imread('Sudoku.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
cv2.imshow('edges', edges)
lines = cv2.HoughLines(edges, 1, np.pi / 180, 200)
....
```

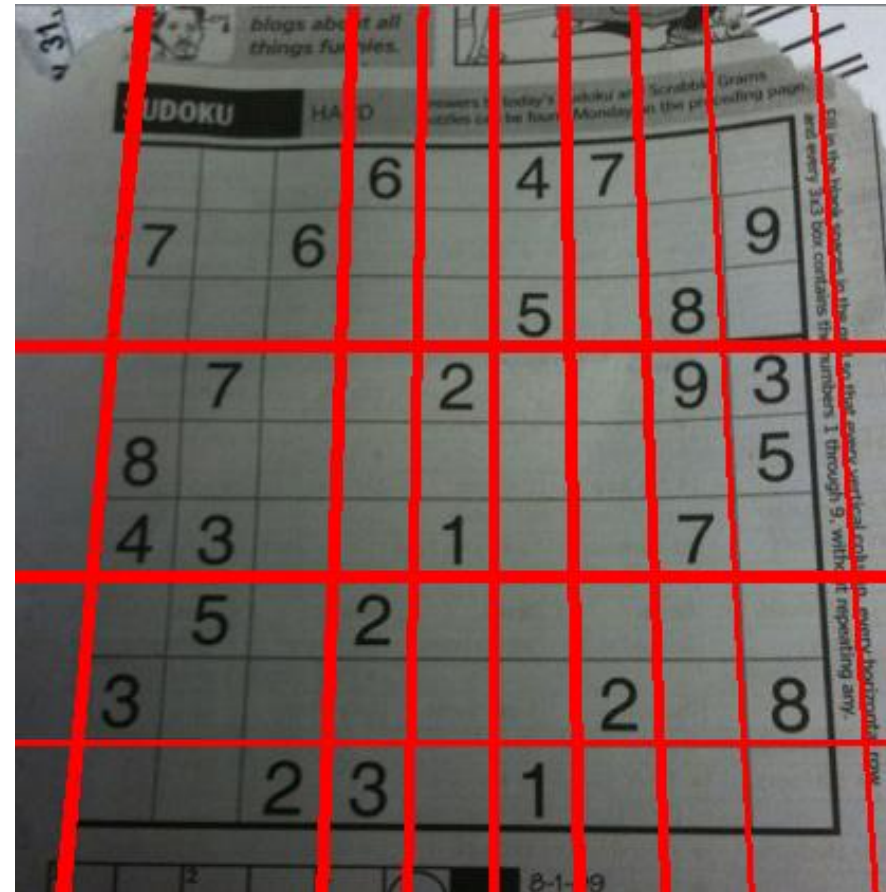
Để đạt độ chính xác cao trước khi sử dụng Hough Line, nên xử lý qua bằng thuật toán Canny.



## 5.3 Các phương pháp phát hiện biên

### *Hough Line transform*

```
....  
for line in lines:  
    rho, theta = line[0]  
    a = np.cos(theta)  
    b = np.sin(theta)  
    x0 = a * rho  
    y0 = b * rho  
    x1 = int(x0 + 1000 * (-b))  
    y1 = int(y0 + 1000 * (a))  
    x2 = int(x0 - 1000 * (-b))  
    y2 = int(y0 - 1000 * (a))  
    cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)  
cv2.imshow('image', img)  
k = cv2.waitKey(0)  
cv2.destroyAllWindows()
```

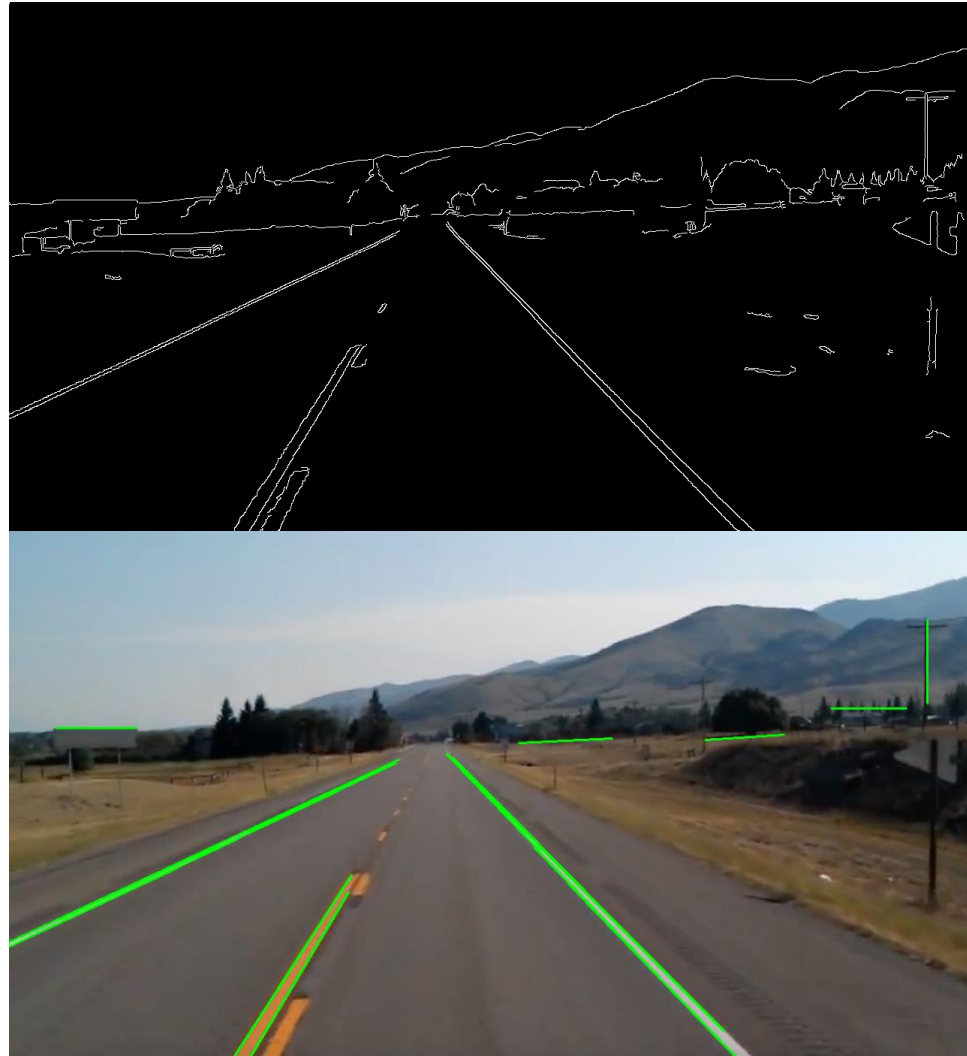




## 5.3 Các phương pháp phát hiện biên

### *Hough Line Probabilistic transform*

```
import cv2
import numpy as np
img = cv2.imread('Road.png')
gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,
\apertureSize = 3)
cv2.imshow('edges', edges)
lines = cv2.HoughLinesP(edges,1,
\np.pi/180, 100,
\minLineLength=100,maxLineGap=10)
for line in lines:
    x1,y1,x2,y2 = line[0]
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
cv2.imshow('image', img)
k = cv2.waitKey(0)
cv2.destroyAllWindows()
```



## 5.3 Các phương pháp phát hiện biên

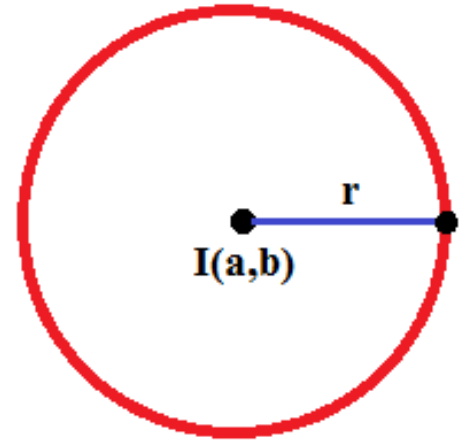
### *Hough Circle transform*

- Trong hệ tọa độ Descartes, phương trình đường tròn tâm  $I(a,b)$  và bán kính  $r$  có dạng:

$$(x-a)^2 + (y-b)^2 = r^2$$

Hoặc có thể viết dưới dạng tham số:

$$\begin{cases} x = a + r \cos \theta \\ y = b + r \sin \theta \end{cases}$$



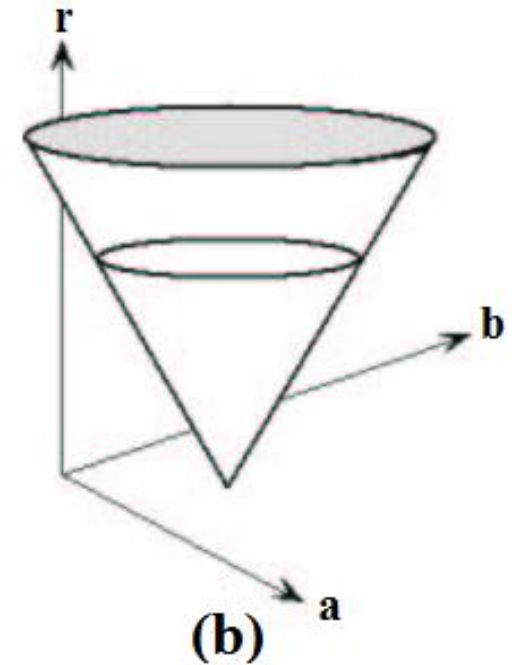
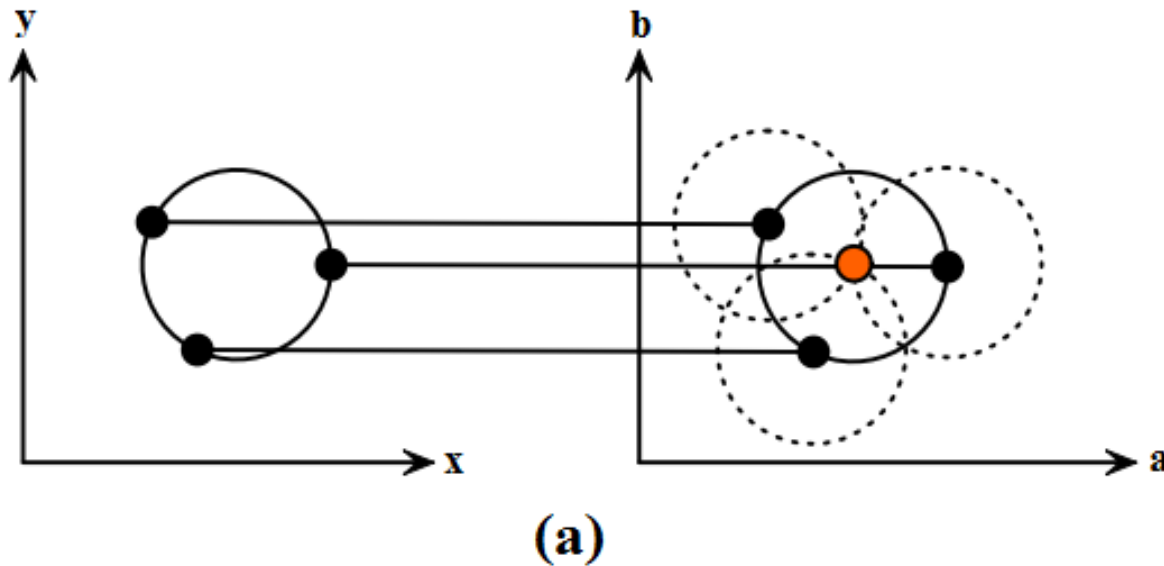
- Trong hệ tọa độ cực, phương trình đường có dạng:

$$\begin{cases} a = x - r \cos \theta \\ b = y - r \sin \theta \end{cases}$$

## 5.3 Các phương pháp phát hiện biên

### *Hough Circle transform*

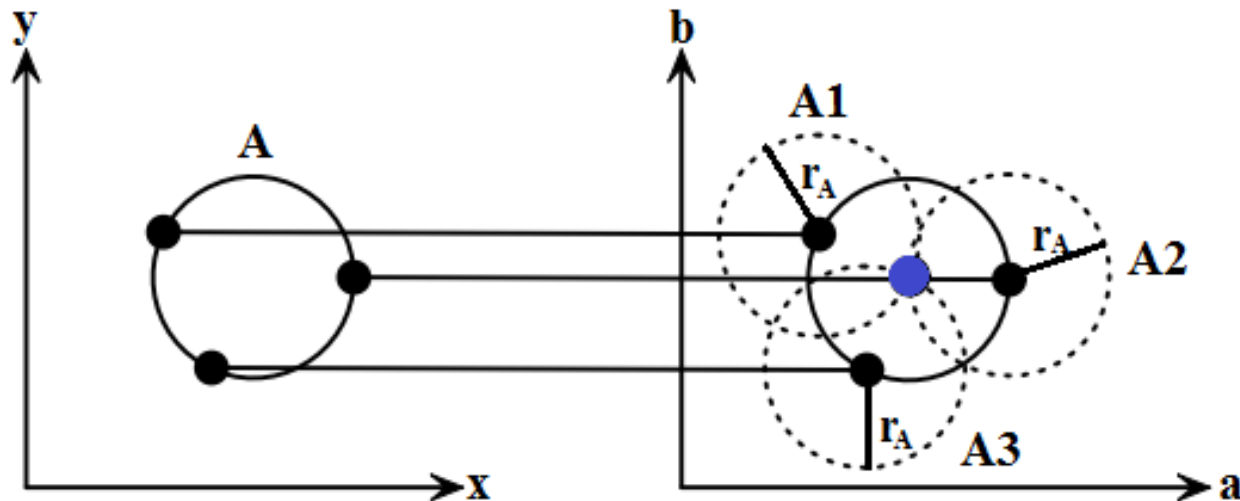
- Biến đổi Hough chuyển đường tròn từ không gian  $(x, y)$  sang không gian  $(a, b)$  cho trường hợp:
  - Bán kính  $r$  là hằng số ( $a$ )
  - Bán kính  $r$  là thay đổi ( $b$ )



## 5.3 Các phương pháp phát hiện biên

### *Hough Circle transform*

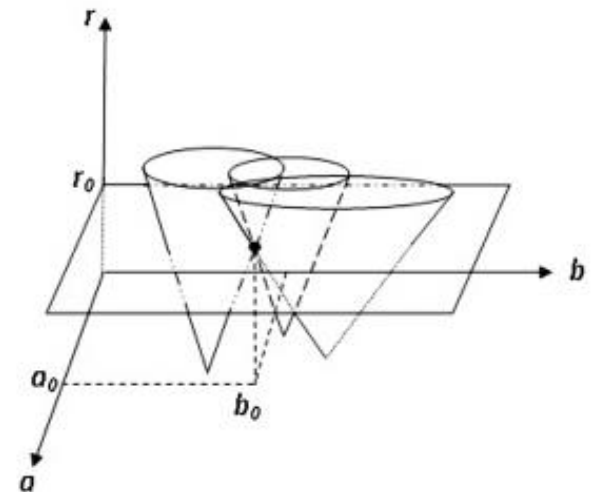
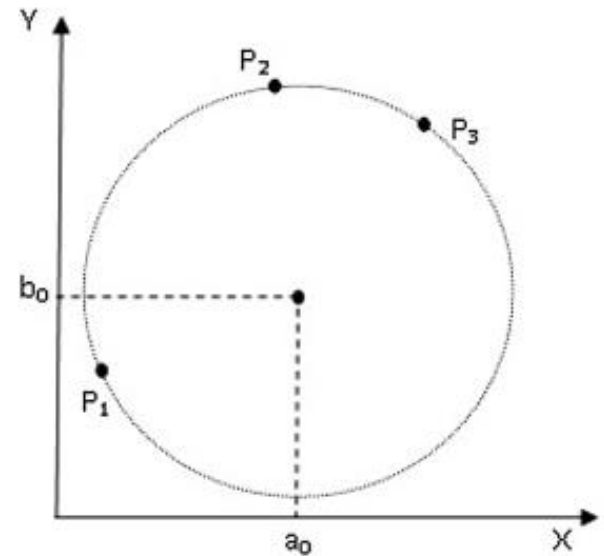
- *Trường hợp bán kính  $r$  cố định*
  - Vẽ lần lượt các đường tròn  $A1, A2, A3, \dots$  với tâm là những điểm  $(a_i, b_i)$  nằm trên đường tròn  $A$ , bán kính  $r_A$
  - Việc vẽ các đường tròn này chính là việc cho chạy các giá trị  $(x_i, y_i)$  và  $\theta$  cùng với giá trị  $r_A$  là cố định.
  - Điểm có số giao điểm nhiều nhất của các đường tròn này chính là tâm  $(a, b)$  của đường tròn cần xác định và bán kính là  $r_A$ .



## 5.3 Các phương pháp phát hiện biên

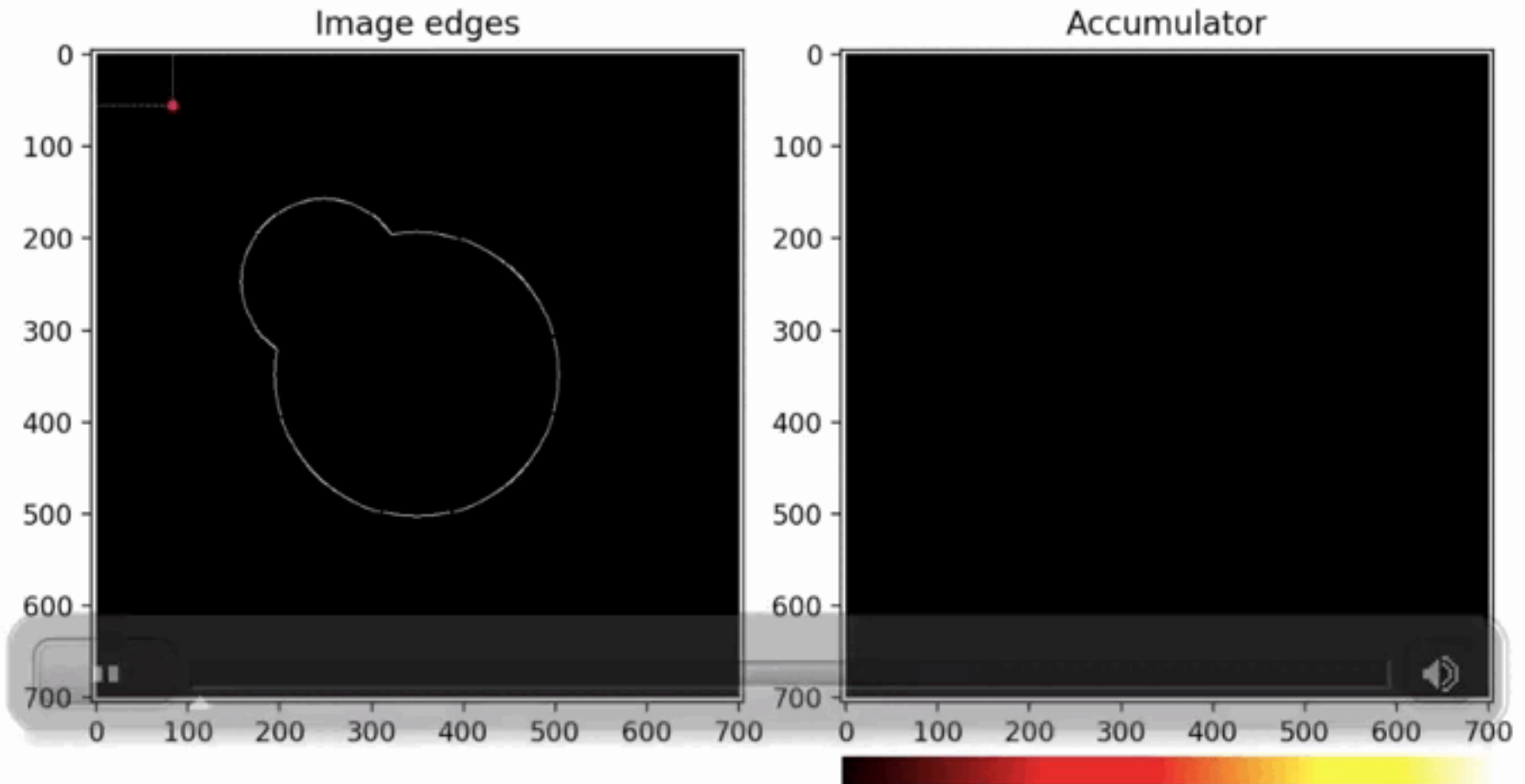
### *Hough Circle transform*

- *Trường hợp bán kính  $r$  thay đổi*
  - Thay vì  $r = r_A$  ta phải vẽ thử tất cả các đường tròn với  $r$  thay đổi từ  $r = r_{\min}$  đến  $r = r_{\max}$ .
  - Điểm có số giao điểm nhiều nhất của các đường tròn chính là tâm  $(a, b)$  của đường tròn cần xác định.
  - Bán kính của đường tròn chính là bán kính của trường hợp  $r_i$  (với  $i$  thuộc đoạn  $[r_{\min}, r_{\max}]$ ) đã tạo nên số giao điểm nhiều nhất của các đường tròn.



## 5.3 Các phương pháp phát hiện biên

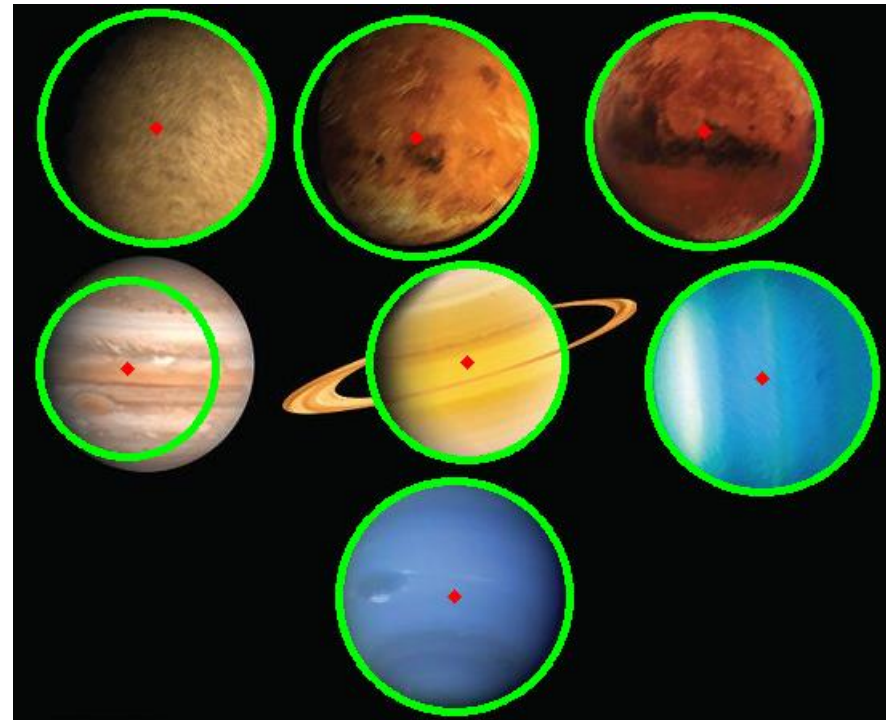
### *Hough Circle transform*



## 5.3 Các phương pháp phát hiện biên

### *Hough Circle transform*

```
import numpy as np
import cv2
img = cv2.imread('Planet.jpg')
output = img.copy()
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = cv2.medianBlur(gray, 5)
circles = cv2.HoughCircles(gray,
cv2.HOUGH_GRADIENT, 1, 120, param1=100,
\param2=30, minRadius=0, maxRadius=0)
detected_circles = np.uint16(np.around(circles))
for (x, y, r) in detected_circles[0, :]:
    cv2.circle(output, (x, y), r, (0, 255, 0), 3)
    cv2.circle(output, (x, y), 2, (0, 0, 255), 3)
cv2.imshow('output', output)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



## 5.3 Các phương pháp phát hiện biên

### *Phương pháp xây dựng đường viền (Contours)*

- Hiểu đơn giản contours là một đường bao khép kín nối các điểm liên tục (trên đường viền bao), các điểm trên contour có cùng giá trị màu và cường độ. Contour là một công cụ hữu dụng trong phân tích hình dáng, phát hiện đối tượng và nhận dạng.
- Một số lưu ý về tìm contours trong OpenCV:
  - Để đạt độ chính xác cao trước khi tìm contours, nên xử lý qua bằng phương pháp phân ngưỡng hoặc thuật toán Canny.
  - Hàm **findContours** có thể làm thay đổi ảnh gốc. Do đó, nếu muốn giữ lại ảnh gốc hãy tạo một bản sao.
  - Trong OpenCV, tìm contours giống như tìm vật thể trắng từ nền đen. Do đó, vật thể nên có màu trắng và nền nên có màu đen.




## 5.3 Các phương pháp phát hiện biên

### *Phương pháp xây dựng đường viền (Contours)*

- Sử dụng hàm **cv2.findContours()** để tìm contours.
- Sau khi trích xuất được các contour thì chúng ta sẽ vẽ các contour đó thông qua hàm **cv2.drawContours()**. Nó có thể vẽ bất kì hình dạng nào khi biết được các tọa độ điểm biên của nó.
- Đếm số Contours ta sử dụng hàm **cv2.len(contours)**.

## 5.3 Các phương pháp phát hiện biên

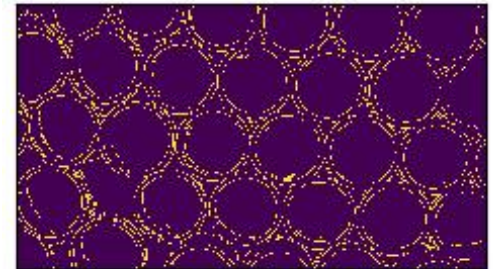
### *Phương pháp xây dựng đường viền (Contours)*

```
import cv2
from matplotlib import pyplot as plt
#img = cv2.imread('Contours.png')
img = cv2.imread('Pipes.jpg')
imgCanny = cv2.Canny(img, 100, 200)
contours, hierarchy = cv2.findContours(imgCanny,
\cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
imgOrigin = img.copy()
cv2.drawContours(img, contours, -1, (0, 255, 0), 3)
plt.figure(figsize = (3, 12))
plt.subplot(311),plt.imshow(imgOrigin),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(312),plt.imshow(imgCanny),plt.title('Canny Image')
plt.xticks([]), plt.yticks([])
plt.subplot(313),plt.imshow(img),plt.title('All Contours')
plt.xticks([]), plt.yticks([])
plt.show()
```

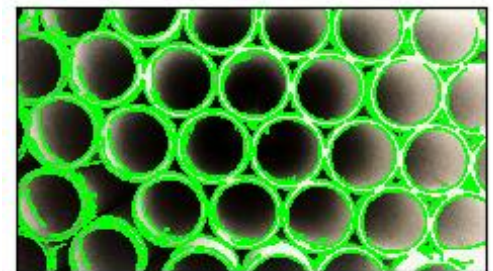
Original



Canny Image



All Contours

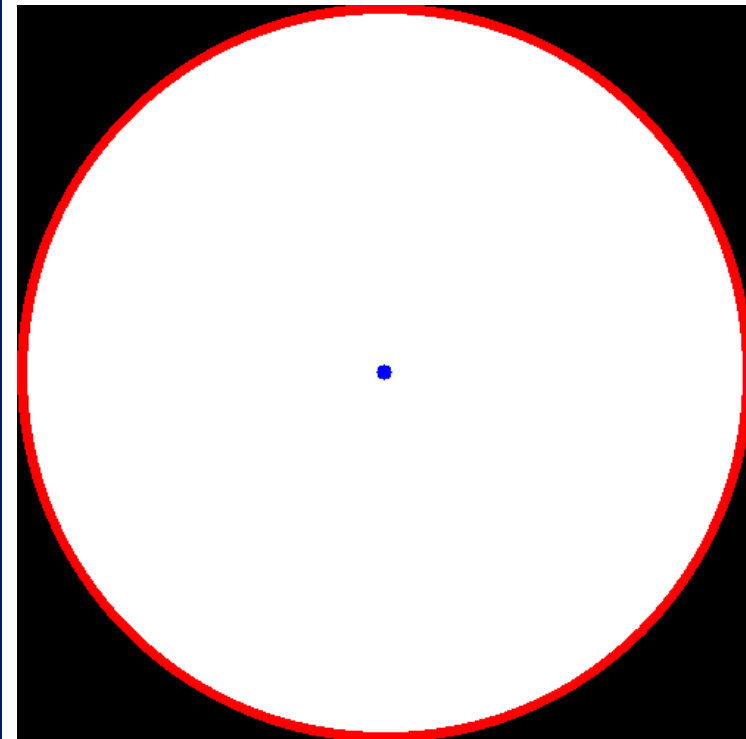


## 5.3 Các phương pháp phát hiện biên

### *Phương pháp xây dựng đường viền (Contours)*

#### ■ *Các đặc trưng contours*

```
import cv2
# Tim contours
img = cv2.imread('Circle.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 127, 255, 0)
contours, hierarchy = cv2.findContours(thresh, 1, 2)
img = cv2.drawContours(img, contours, -1, (0, 0, 255), 5)
# Tim moments
cnt = contours[0]
M = cv2.moments(cnt)
print("Moments: ", M)
# Tim trong tam
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
cv2.circle(img, (cx, cy), 5, (255, 0, 0), -1)
print("Trong tam: (%d, %d)" % (cx, cy))
```



## 5.3 Các phương pháp phát hiện biên

### *Phương pháp xây dựng đường viền (Contours)*

- *Các đặc trưng contours*

....

*# Tim dien tich*

```
area = cv2.contourArea(cnt)
```

```
print("Dien tich: ", area)
```

```
print("M['m00']: ", M['m00'])
```

*# Tim chu vi*

```
perimeter = cv2.arcLength(cnt, True)
```

```
print("Chu vi: ", perimeter)
```

```
cv2.imshow('img', img)
```

```
cv2.waitKey()
```

Trong tam: (249, 249)

Dien tich: 191729.0

M['m00']: 191729.0

Chu vi: 1637.415422797203

# TỔNG HỢP MỘT SỐ HÀM QUAN TRỌNG

---

**cv.Canny**(image, threshold1, threshold2)

## Parameters

- image** 8-bit input image.
- threshold1** first threshold for the hysteresis procedure.
- threshold2** second threshold for the hysteresis procedure.

# TỔNG HỢP MỘT SỐ HÀM QUAN TRỌNG

**cv.HoughLines**(image, rho, theta, threshold, min\_theta, max\_theta)

## Parameters

- image** 8-bit, single-channel binary source image. The image may be modified by the function.
- lines** Output vector of lines. Each line is represented by a 2 or 3 element vector  $(\rho, \theta)$ .
- rho** Distance resolution of the accumulator in pixels.
- theta** Angle resolution of the accumulator in radians.
- threshold** **Accumulator** threshold parameter.  
Only those lines are returned that get enough votes ( $> \text{threshold}$ ).
- min\_theta** For standard and multi-scale Hough transform, minimum angle to check for lines.
- max\_theta** For standard and multi-scale Hough transform, maximum angle to check for lines.

# TỔNG HỢP MỘT SỐ HÀM QUAN TRỌNG

**cv.HoughLinesP** (image, lines, rho, theta, threshold, minLineLength, maxLineGap)

## Parameters

<b>image</b>	8-bit, single-channel binary source image.
<b>lines</b>	output vector of lines(cv.32SC4 type). Each line is represented by a 4-element vector (x1,y1,x2,y2) ,where (x1,y1) and (x2,y2) are the ending points of each detected line segment.
<b>rho</b>	distance resolution of the accumulator in pixels.
<b>theta</b>	angle resolution of the accumulator in radians.
<b>threshold</b>	accumulator threshold parameter. Only those lines are returned that get enough votes
<b>minLineLength</b>	minimum line length. Line segments shorter than that are rejected.
<b>maxLineGap</b>	maximum allowed gap between points on the same line to link them.

# TỔNG HỢP MỘT SỐ HÀM QUAN TRỌNG

**cv.HoughCircles**(image, method, dp, minDist, circles, param1, param2, minRadius, maxRadius)

**image:** 8-bit, single channel image. If working with a color image, convert to grayscale first.

**method:** Defines the method to detect circles in images. Currently, the only implemented method is cv2.HOUGH\_GRADIENT.

**dp:** This parameter is the inverse ratio of the accumulator resolution to the image resolution.

**minDist:** Minimum distance between the center (x, y) coordinates of detected circles.

**param1:** Gradient value used to handle edge detection.

**param2:** Accumulator threshold value for the cv2.HOUGH\_GRADIENT method. The smaller the threshold is, the more circles will be detected (including false circles). The larger the threshold is, the more circles will potentially be returned.

**minRadius:** Minimum size of the radius (in pixels).

**maxRadius:** Maximum size of the radius (in pixels).



# TỔNG HỢP MỘT SỐ HÀM QUAN TRỌNG

**cv.findContours**(image, mode, method, contours, hierarchy, offset )

=> contours, hierarchy

## Parameters

**image** Source, an 8-bit single-channel image.

**mode** Contour retrieval mode, see **RetrievalModes**

**method** Contour approximation method, see **ContourApproximationModes**

**contours** Detected contours. Each contour is stored as a vector of points

**hierarchy** Optional output vector, containing information about the image topology.

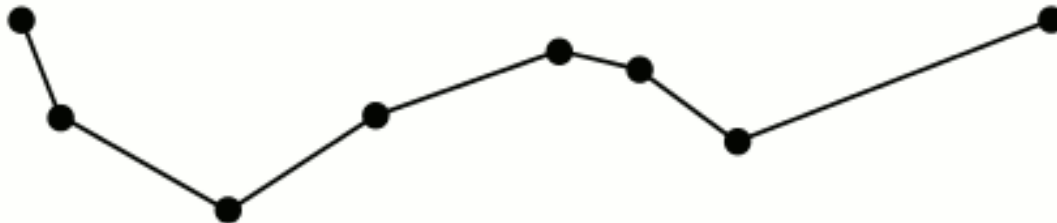
RETR_EXTERNAL Python: cv.RETR_EXTERNAL	retrieves only the extreme outer contours. It sets <code>hierarchy[i][2]=hierarchy[i][3]=-1</code> for all the contours.
RETR_LIST Python: cv.RETR_LIST	retrieves all of the contours without establishing any hierarchical relationships.
RETR_TREE Python: cv.RETR_TREE	retrieves all of the contours and reconstructs a full hierarchy of nested contours.
CHAIN_APPROX_NONE Python: cv.CHAIN_APPROX_NONE	stores absolutely all the contour points. That is, any 2 subsequent points (x1,y1) and (x2,y2) of the contour will be either horizontal, vertical or diagonal neighbors, that is, $\max(\text{abs}(x1-x2), \text{abs}(y2-y1))=1$ .
CHAIN_APPROX_SIMPLE Python: cv.CHAIN_APPROX_SIMPLE	compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.

# TỔNG HỢP MỘT SỐ HÀM QUAN TRỌNG

**cv.approxPolyDP**(curve, epsilon, closed)

## Parameters

<b>curve</b>	Input vector
<b>epsilon</b>	Parameter specifying the approximation accuracy.
<b>closed</b>	If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.



[https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker\\_algorithm](https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm)

# CASE STUDY 4.1: PHÁT HIỆN CẠNH VĨ THUỐC

**Mô tả:** Ứng dụng Hough Line để phát hiện bốn line trên vỉ thuốc.

**Yêu cầu:**

- Phát hiện bốn cạnh của vỉ thuốc;
- Vẽ bốn cạnh lên ảnh gốc.



## CASE STUDY 4.2: ĐẾM SỐ LƯỢNG ĐỒNG XU

**Mô tả:** Ứng dụng Hough Circle để đếm số lượng đồng xu.

**Yêu cầu:**

- Phát hiện đồng xu;
- Đếm số lượng đồng xu.

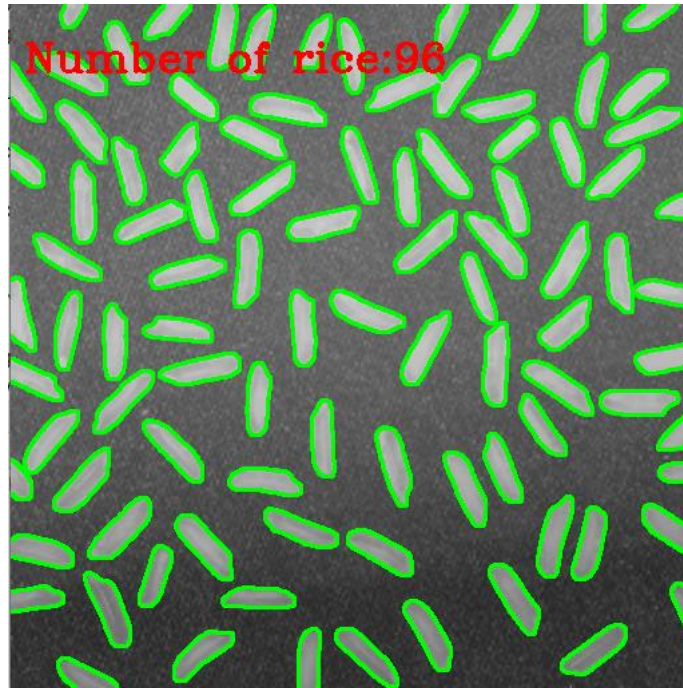


## CASE STUDY 4.3: ĐẾM SỐ HẠT GẠO

**Mô tả:** Ứng dụng Contour để đếm số hạt gạo.

**Yêu cầu:**

- Phát hiện số hạt gạo;
- Đếm số lượng hạt gạo và in ra màn hình.

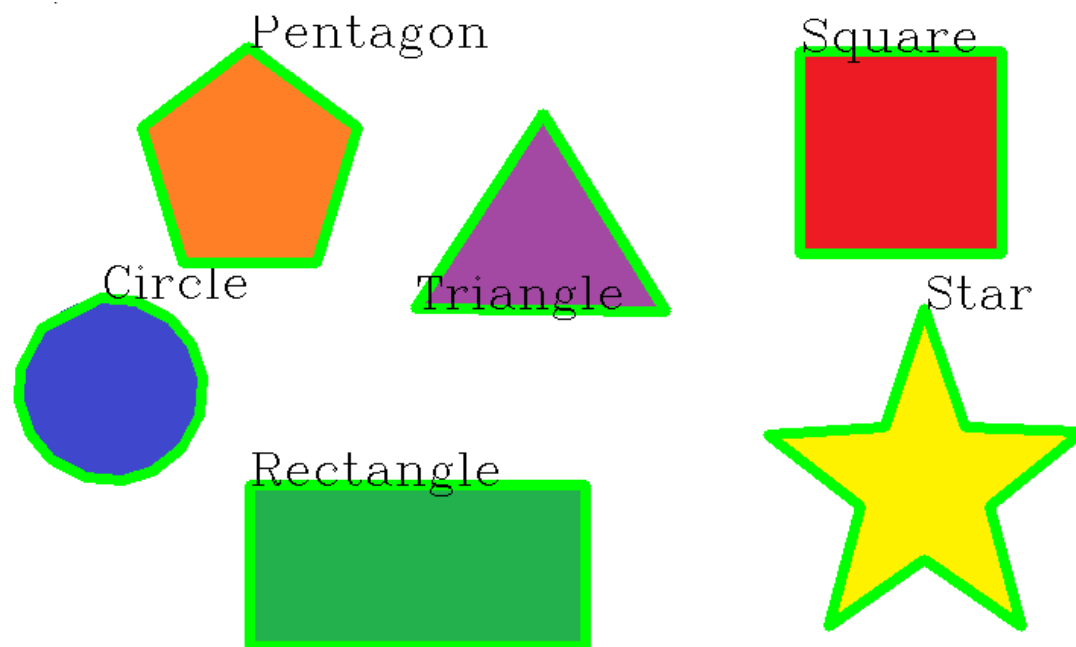


## CASE STUDY 4.4: HÌNH DÁNG VẬT THỂ

**Mô tả:** Ứng dụng Contour để biết được hình dáng vật thể.

**Yêu cầu:**

- Phát hiện vật thể;
- Vẽ biên dạng và in hình dáng vật thể ra màn hình.



# CASE STUDY 4.5: CHẤM BÀI THI TRẮC NGHIỆM

---

**Mô tả:** Ứng dụng Contour để chấm bài thi trắc nghiệm.

**Yêu cầu:**

- Cắt vùng chứa đáp án;
- Xoay vùng chứa đáp án;
- Phát hiện phương án được chọn;
- So sánh kết quả kết quả với đáp án;
- Hiển thị điểm.

**Thank you !!!**