1_TurtleNumber1_ND

January 18, 2024

#I. Các khái niệm cơ bản.

1. Điểm ảnh:

- Điểm ảnh *(Pixel)* là một phần tử của ảnh số tại tọa độ (x, y) với độ xám hoặc màu nhất định. Kích thước và khoảng cách giữa các điểm ảnh đó được chọn thích hợp sao cho mắt người cảm nhận được sự liên tục về không gian và mức xám (hoặc màu) của ảnh số gần như ảnh thật. Mỗi phần tử trong ma trận được gọi là một phần tử ảnh.
- Điểm ảnh *(pixel)* có vị trí (x, y) và có độ xám I(x, y).
- Với ảnh màu ảnh ta hay nhìn thì mỗi điểm ảnh sẽ có 3 giá trị tương ứng với độ sáng của các màu đỏ, xanh lục, xanh dương (RGB).

2. Ảnh số:

- Ảnh số: "một hình ảnh có thể được định nghĩa là hàm hai chiều, f(x, y), trong đó x và y là tọa độ không gian (mặt phẳng) và biên độ của f tại bất kỳ cặp tọa độ (x, y) nào được gọi là cường độ hoặc mức độ màu xám của hình ảnh tại điểm đó. Khi x, y và các giá trị cường độ của f đều là các đại lượng hữu hạn, rời rạc, chúng ta gọi hình ảnh là hình ảnh kỹ thuật số".
- Hay có thể hiểu 1 cách đơn giản rằng "Ảnh số là số hóa làm cho một hình ảnh kỹ thuật số trở thành một xấp xỉ của một cảnh thực".

3. Ảnh nhị phân:

- Ảnh nhị phân: Đúng như tên gọi, ảnh nhị phân chỉ chứa những pixel có giá trị 0 hoặc 1 trong đó 0 chỉ màu đen và 1 chỉ màu trắng. Hình ảnh này còn được gọi là Đơn sắc.

#II. Thresholding trong OpenCV

1. Khái niệm

Ngưỡng hoá (Thresholding) là quá trình chuyển đổi ảnh xám thành ảnh nhị phân, trong đó các giá trị pixel chỉ có thể là 0 hoặc 255.

Ví dụ: Chúng ta sẽ chọn một giá trị p, từ đó các giá trị pixel trong hình nhỏ hơn p sẽ bằng 0, còn lại sẽ bằng 255. Từ đó có thể biểu diễn nhị phân của ảnh.

Thông thường, ngưỡng hoá được sử dụng để tập trung vào các đối tượng hay một vùng trong hình ảnh cần quan tâm. Kết quả của hình ảnh nhị phân, ảnh ban đầu sẽ được chia thành 2 phần: vùng vật thể và vùng nền *(background)*.

Đầu vào của một thuật toán phân ngưỡng trong OpenCV thường có input là ảnh nguồn *(source image)* và giá trị ngưỡng. Đầu ra là ảnh đích đã được phân ngưỡng *(destination image)*. Một số

thuật toán phân ngưỡng sẽ kèm thêm vài giá trị khác, nhưng ở đây chúng ta sẽ không đề cập tới chúng.

Mã giả của thuật toán phân ngưỡng:

```
if src[i] >= T:
  dest[i] = MAXVAL
else:
  dest[i] = 0
```

Phương pháp Threshold được sử dụng để thực hiện các tác vụ như:

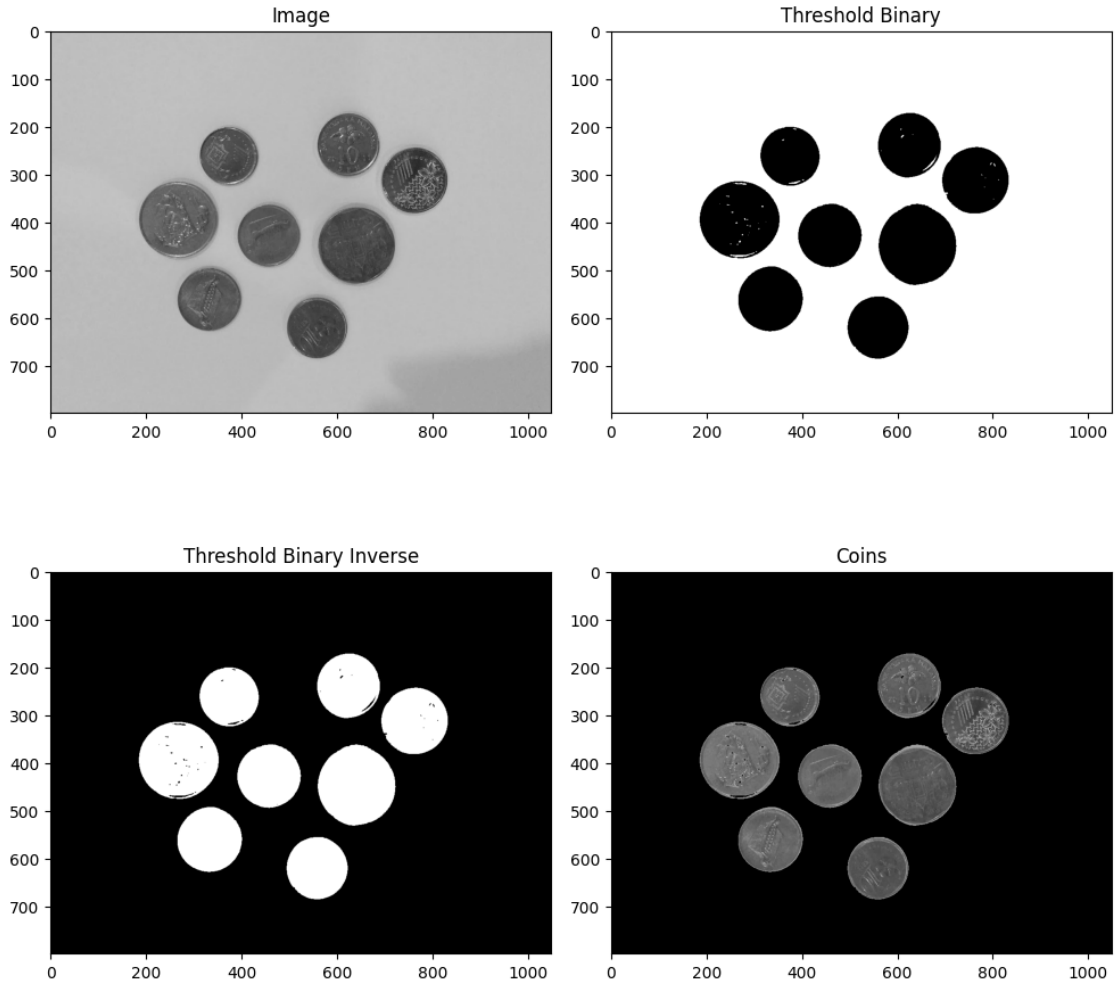
- Chuyển đổi ảnh sang ảnh nhị phân: Dùng để tạo ra ảnh chỉ có hai mức xám là trắng và đen, tách biệt các vùng hoặc đối tượng trong ảnh.
- Phân đoạn ảnh: Giúp phân chia ảnh thành các vùng có tính chất tương tự như màu sắc, độ sáng, hoặc đặc trưng khác. Phát hiện biên cạnh: Dùng để phát hiện và tách biệt các biên cạnh hoặc đường viền trong ảnh.
- Nhận dạng và phân loại đối tượng: Threshold có thể được sử dụng để phân loại đối tượng dựa trên đặc trưng của chúng.
- OpenCV là một thư viện phổ biến được sử dụng trong xử lý ảnh, cung cấp các hàm và công cụ để thực hiện phương pháp Threshold trên ảnh. Các hàm như *cv2.threshold()* và *cv2.adaptiveThreshold()* trong OpenCV cung cấp các phương pháp Threshold cơ bản và Threshold thích ứng để xử lý ảnh.

2. Simple Thresholding (Ngưỡng đơn giản).

Áp dụng phương pháp ngưỡng hoá đơn giản cần phải có sự can thiệp từ con người, tức là chúng ta cần chọn một ngưỡng giá trị T. Từ đó, tất cả giá trị pixel trong ảnh bé hơn T sẽ thành 0, và lớn hơn T sẽ thành 255. Chúng ta có thể áp dụng ngược lại (giá trị bé hơn T sẽ thành 255, nhỏ hơn T sẽ thành 0)

[ ]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

#Đọc ảnh
image = cv2.imread("coins.jpg")

#Chuyển đổi thành ảnh xám
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Áp dụng bộ lọc để làm mờ ảnh
blurred = cv2.GaussianBlur(image_gray, (5, 5), 0)

#Ngưỡng hóa ảnh
(T, thresh) = cv2.threshold(blurred, 155, 255, cv2.THRESH_BINARY)

#Ngưỡng đảo
```

```
(T, threshInv) = cv2.threshold(blurred, 155, 255, cv2.THRESH_BINARY_INV)

#Sử dụng bitwise_and để xuất ra ảnh tập trung vào các đồng xu
coins = cv2.bitwise_and(image_gray, image_gray, mask=threshInv)

# Tạo subplot 2x2 để hiển thị 4 hình ảnh
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Hiển thị hình ảnh gốc
axs[0, 0].imshow(image_gray, cmap = 'gray')
axs[0, 0].set_title('Image')

# Hiển thị hình ảnh Threshold Binary
axs[0, 1].imshow(thresh, cmap = 'gray')
axs[0, 1].set_title('Threshold Binary')

# Hiển thị hình ảnh Threshold Binary Inverse
axs[1, 0].imshow(threshInv, cmap = 'gray')
axs[1, 0].set_title('Threshold Binary Inverse')

# Hiển thị hình ảnh Coins
axs[1, 1].imshow(coins, cmap = 'gray')
axs[1, 1].set_title('Coins')

plt.tight_layout()
plt.show()
```



3. Adaptive Thresholding (Ngưỡng thích nghi/thích ứng).

Thuật toán Simple Thresholding hoạt động khá tốt. Tuy nhiên, nó có 1 nhược điểm là giá trị ngưỡng được gán toàn cục. Đôi khi, hình ảnh chúng ta nhận được thường bị ảnh hưởng của nhiễu, ví dụ như là bị phơi sáng, bị đèn flash, v.v...

Một trong những cách được sử dụng để giải quyết vấn đề trên là chia nhỏ bức ảnh thành những vùng nhỏ (region), và đặt giá trị ngưỡng trên những vùng nhỏ đó -> Adaptive Thresholding ra đời. OpenCV cung cấp cho chúng ta hai cách xác định những vùng nhỏ, đó chính là:

i. Mean (Trung bình).

ii. Gaussian.

```
cv2.adaptiveThreshold(img, max_value, adaptive_method, threshold_type, block_size, constant_val
```

- adaptive_method: nhận vào một trong hai giá trị là cv.ADAPTIVE_THRESH_MEAN_C và cv.ADAPTIVE_THRESH_GAUSSIAN_C, đó là các phương pháp tính ngưỡng.
- thresholdType: Tương tự như Simple Thresholding đã trình bày ở trên.

- block_size: kích thước vùng.
- constant_value: giá trị hằng, gán từ -255 tới 255 tùy ý.

1. **Mean**.

```
cv2.adaptiveThreshold(img, max_value, cv2.ADAPTIVE_THRESH_MEAN_C, threshold_type, block_size,
```

- Ở phương pháp tính Mean, ta tính trung bình các láng giềng xung quanh điểm cần xét trong vùng block_size * block_size trừ đi giá trị hằng số C (constant_value).

2. **Gaussian**.

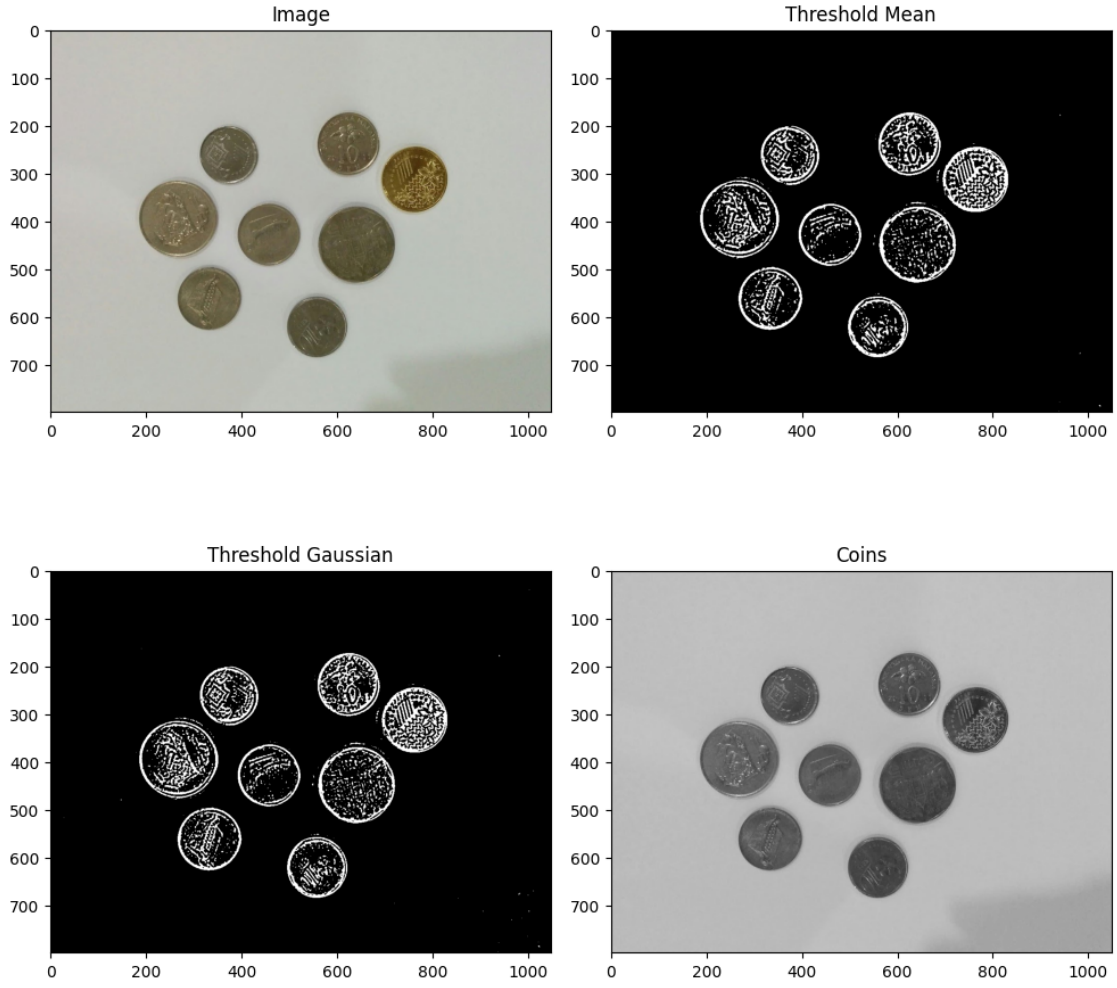
```
cv2.adaptiveThreshold(img, max_value, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, threshold_type, block_si
```

- Ở phương pháp tính Gaussian, ta nhân giá trị xung quanh điểm cần xét với trọng số Gauss rồi tính trung bình của nó, sau đó trừ đi giá trị hằng số C (constant_value).

Để giải quyết vấn đề này, chúng ta có thể dùng ngưỡng thích ứng, trong đó xét các vùng lân cận nhỏ của các pixel, sau đó tìm ra T tối ưu cho mỗi vùng lân cận đó. Phương pháp này cho phép chúng ta xử lý các trường hợp có thể có sự biến đổi lớn về độ sáng của pixel. Gía trị T tối ưu có thể thay đổi cho từng phần khác nhau của hình ảnh.

[ ]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

#Đọc ảnh
image = cv2.imread("coins.jpg")

#Chuyển đổi ảnh màu BGR sang RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

#Chuyển đổi ảnh thành ảnh xám
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Áp dụng bộ lọc để làm mờ ảnh
blurred = cv2.GaussianBlur(image_gray, (5, 5), 0)

#Áp dụng cách xác định bằng phương pháp Mean
thresh_mean = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_MEAN_C,␣
 ↪cv2.THRESH_BINARY_INV, 11, 4)

#Áp dụng cách xác định bằng phương pháp Gaussian
thresh_gaussian = cv2.adaptiveThreshold(blurred, 255, cv2.
 ↪ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 15, 3)

# Tạo subplot 2x2 để hiển thị 4 hình ảnh
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
```

```
# Hiển thị hình ảnh gốc
axs[0, 0].imshow(image_rgb, cmap = 'gray')
axs[0, 0].set_title('Image')

# Hiển thị hình ảnh Threshold Mean
axs[0, 1].imshow(thresh_mean, cmap = 'gray')
axs[0, 1].set_title('Threshold Mean')

# Hiển thị hình ảnh Threshold Gaussian
axs[1, 0].imshow(thresh_gaussian, cmap = 'gray')
axs[1, 0].set_title('Threshold Gaussian')

# Hiển thị hình ảnh Gray
axs[1, 1].imshow(image_gray, cmap = 'gray')
axs[1, 1].set_title('Coins')

plt.tight_layout()
plt.show()
```

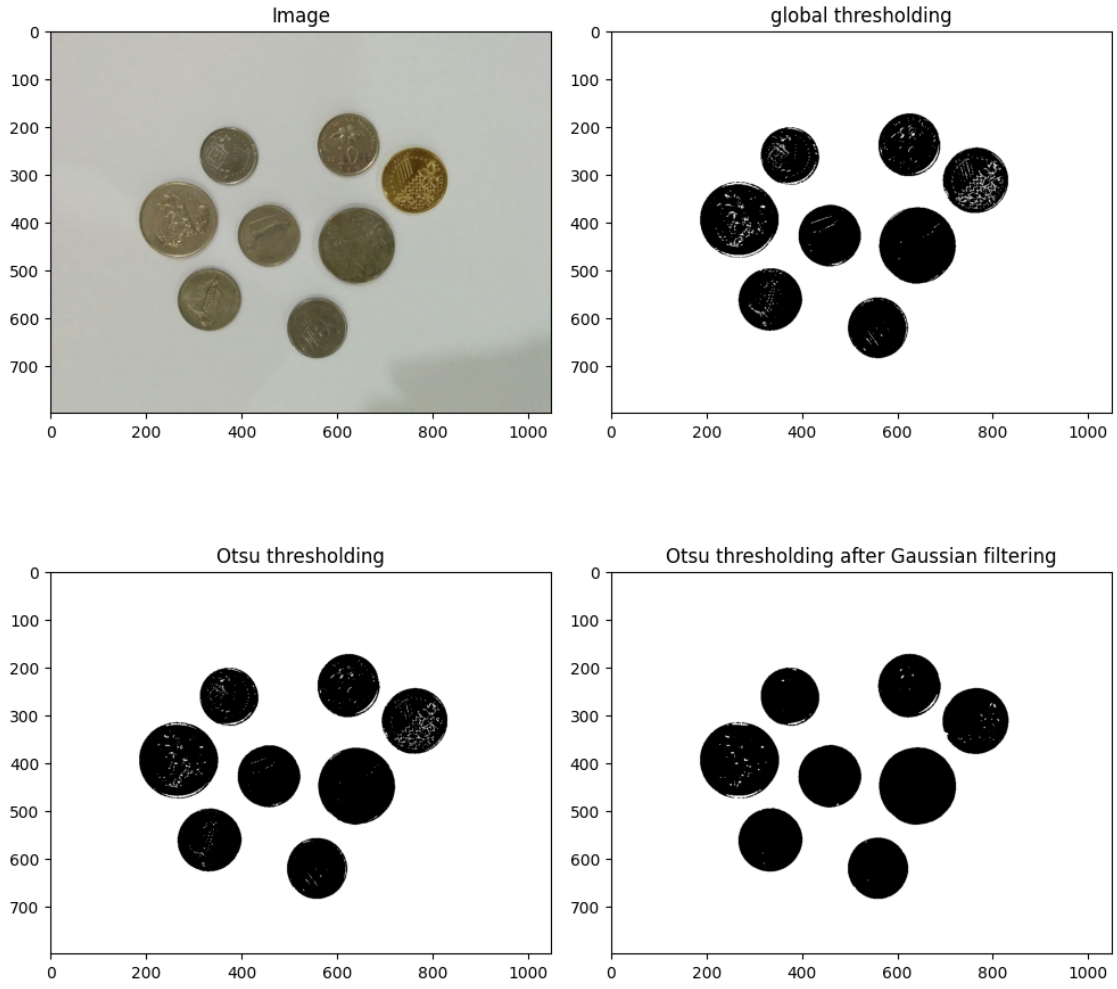


4. Otsu và Riddler-Calvard

- Otsu được lấy tên theo tên tác giả của phương pháp này Nobuyuki Otsu đã giới thiệu về kĩ thuật này trong tập báo của IEEE Transactions on Systems, Man, and Cybernetics - Hiệp hội hệ thống, con người và điều khiển mạng trang 62-66 của IEEE (Institute of Electrical and Electronics Engineers - Viện kỹ sư Điện và Điện Tử) - một tổ chức phi lợi nhuận.
- Một cách khác để tự động tính toán ngưỡng T là sử dụng phương pháp của Otsu. Phương pháp Otsu giả định rằng có hai đỉnh trong biểu đồ histogram của ảnh. Sau đó, nó cố gắng tìm giá trị tối ưu để phân tách hai đỉnh này - giá trị này là T.
- Phương pháp Otsu tập trung vào việc khai thác và tính toán từ thông tin Histogram của bức ảnh. Bằng việc tính toán trên tất cả các mức Threshold, ta có thể chọn mức thỏa mãn việc phân chia giữa Foreground và Background tốt nhất.

[ ]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

image = cv2.imread("coins.jpg")
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(image_gray, (5,5), 0)

# global thresholding
ret1, th1 = cv2.threshold(image_gray, 127, 255, cv2.THRESH_BINARY)

# Otsu's thresholding
ret2, th2 = cv2.threshold(image_gray, 0, 255, cv2.THRESH_BINARY + cv2.
 ↪THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
ret3, th3 = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Tạo subplot 2x2 để hiển thị 4 hình ảnh
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Hiển thị hình ảnh gốc
axs[0, 0].imshow(image_rgb, cmap = 'gray')
axs[0, 0].set_title('Image')

# Hiển thị hình ảnh global thresholding
axs[0, 1].imshow(th1, cmap = 'gray')
axs[0, 1].set_title('global thresholding')
```

```
# Hiển thị hình ảnh Otsu's thresholding
axs[1, 0].imshow(th2, cmap = 'gray')
axs[1, 0].set_title('Otsu thresholding')

# Hiển thị hình ảnh Otsu's thresholding after Gaussian filtering
axs[1, 1].imshow(th3, cmap = 'gray')
axs[1, 1].set_title('Otsu thresholding after Gaussian filtering')

plt.tight_layout()
plt.show()
```



[ ]:
```
!pip install mahotas
```

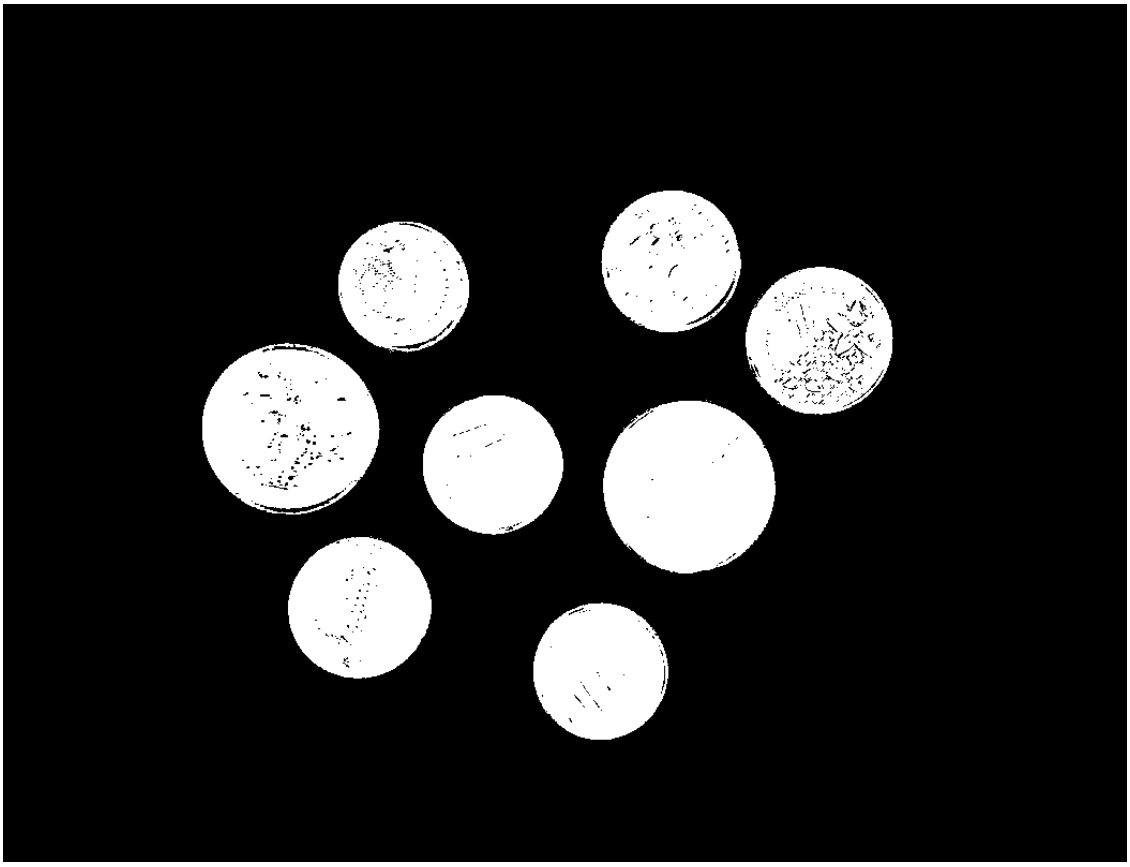
```
Collecting mahotas
  Downloading
mahotas-1.4.13-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.4
MB)
```

```
                                        5.4/5.4 MB
13.8 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-
packages (from mahotas) (1.23.5)
Installing collected packages: mahotas
Successfully installed mahotas-1.4.13
```

[ ]:
```
from __future__ import print_function
import mahotas
```

[ ]:
```
image = cv2.imread("coins.jpg")
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(image, (5,5), 0)

T = mahotas.thresholding.otsu(blurred)
print("Otsu's threshold: {}".format(T))

thresh = image.copy()
thresh[thresh > T] = 255
thresh[thresh < 255] = 0
thresh = cv2.bitwise_not(thresh)

cv2_imshow(thresh)
cv2.waitKey(0)
```

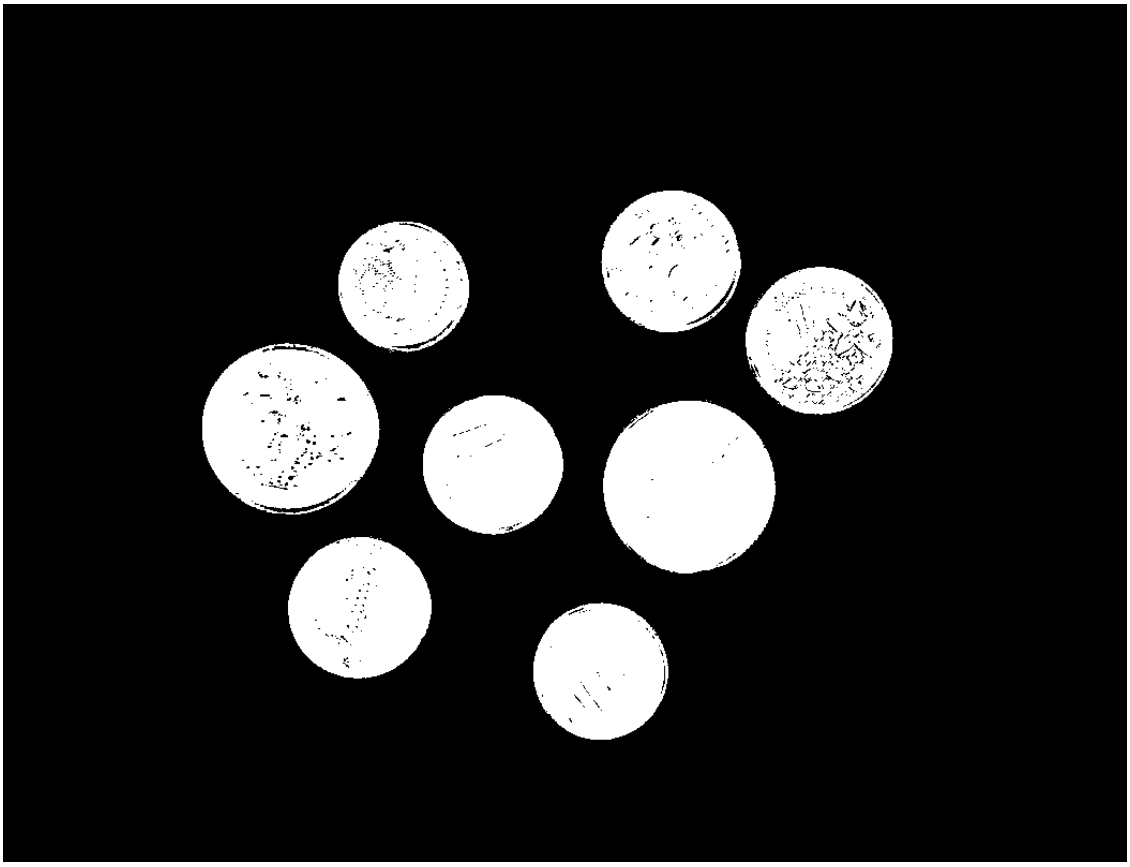
```
Otsu's threshold: 142
```



[ ]: -1

[ ]:
```
image = cv2.imread("coins.jpg")
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(image, (5,5), 0)

T = mahotas.thresholding.rc(blurred)
print("Riddler-Calvard: {}".format(T))

thresh = image.copy()
thresh[thresh > T] = 255
thresh[thresh < 255] = 0
thresh = cv2.bitwise_not(thresh)

cv2_imshow(thresh)
cv2.waitKey(0)
```

```
Riddler-Calvard: 142.07525323283946
```



[ ]: -1

#III. Gradient và Edge Detection trong OpenCV Như tên chương đã đề cập, chương này ta sẽ tìm hiểu về "Gradient" và "Edge Detection" hay nói cách khác chính là "độ dốc" và "phát hiện cạnh" của một tấm ảnh.

1. Cạnh là gì? - Trong ảnh số, những điểm ảnh có cường độ ảnh sáng thay đổi mạnh so với các điểm xung quanh thường được gọi là các điểm cạnh *(edge point)*.

- Cạnh *(edge)* là tập hợp các điểm cạnh tạo nên một hình dạng có ý nghĩa nào đó liên quan đến thông tin hình dạng và cấu trúc của đối tượng trong ảnh.
- Trong ảnh, một cạnh (edge) là một ranh giới (boundary) hay một đường viền (contour) mà tại đấy xuất hiện một sự thay đổi lớn về một vài khía cạnh vật lý của ảnh (cường độ sáng, bề mặt phản chiếu, etc.)
- Trong ảnh số, một cạnh (edge) là một tập hợp các pixels mà tại pixel đấy xảy ra một sự thay đổi đột ngột về cường độ sáng.
- Về hình thức, phát hiện cạnh (edge detection) bao gồm các phương pháp toán học để tìm những điểm trong một bức ảnh mà tại đó cường độ sáng tại những điểm đó có sự thay đổi rõ rệt.

2. Gradient (Độ dốc).

- Ta đã học được học *Gradient* trong giải tích là *Gradient* của một trường vô hướng là một trường vectơ có chiều hướng về phía mức độ tăng lớn nhất của trường vô hướng, và có độ lớn là mức độ thay đổi lớn nhất.

Ví dụ như nhiệt độ trong phòng được cho bởi một trường vô hướng T sao cho tại mỗi điểm *(x, y, z)* có nhiệt độ là *T(x, y, z)*. Trong trường hợp này, tại mỗi điểm trong căn phòng, *Gradient* của T tại điểm đó cho biết hướng mà theo đó nhiệt độ tăng lên nhanh nhất. Độ lớn của *Gradient* cho biết nhiệt độ thay đổi nhanh đến mức nào nếuta đi theo hướng đó.

- Vậy, *Gradient* hay Độ dốc của một bức ảnh là gì?

Trong xử lý ảnh, độ dốc (tức gradient) đang nói đến ở đây chính là độ dốc về mức sáng. Hay nói cách khác chính là sự thay đổi các giá trị pixel trong ảnh.

Vùng ảnh trơn (smooth) thì các pixel trong vùng ảnh đó có giá trị xấp xỉ/gần bằng nhau, vì vậy khi tính toán đạo hàm sẽ gần bằng zero. Đạo hàm bằng 0 thể hiện không có biến thiên về giá trị (mức sáng).

Điều này có nghĩa là độ dốc của các pixel trong vùng ảnh trơn gần bằng zero. Đạo hàm dương tại một pixel thể hiện rằng biến thiên mức sáng đang ở chiều hướng đi lên, ngược lại đạo hàm âm tại một pixel cho biết biên thiên mức sáng tại đó đang giảm dần. Nói tóm gọn lại gradient của ảnh chính là đạo hàm ảnh.

Vậy tóm lại, để phát hiện cạnh (Edge Detection) ta sẽ đi tính độ dốc của bức ảnh (Gradient). Để có thể tính được Gradient của ảnh, ta có thể áp dụng 2 bộ lọc khá phổ biến trong OpenCV chính là Laplacian (cv2.Laplacian) và Sobel (cv2.Sobel)

---

3. Laplacian**

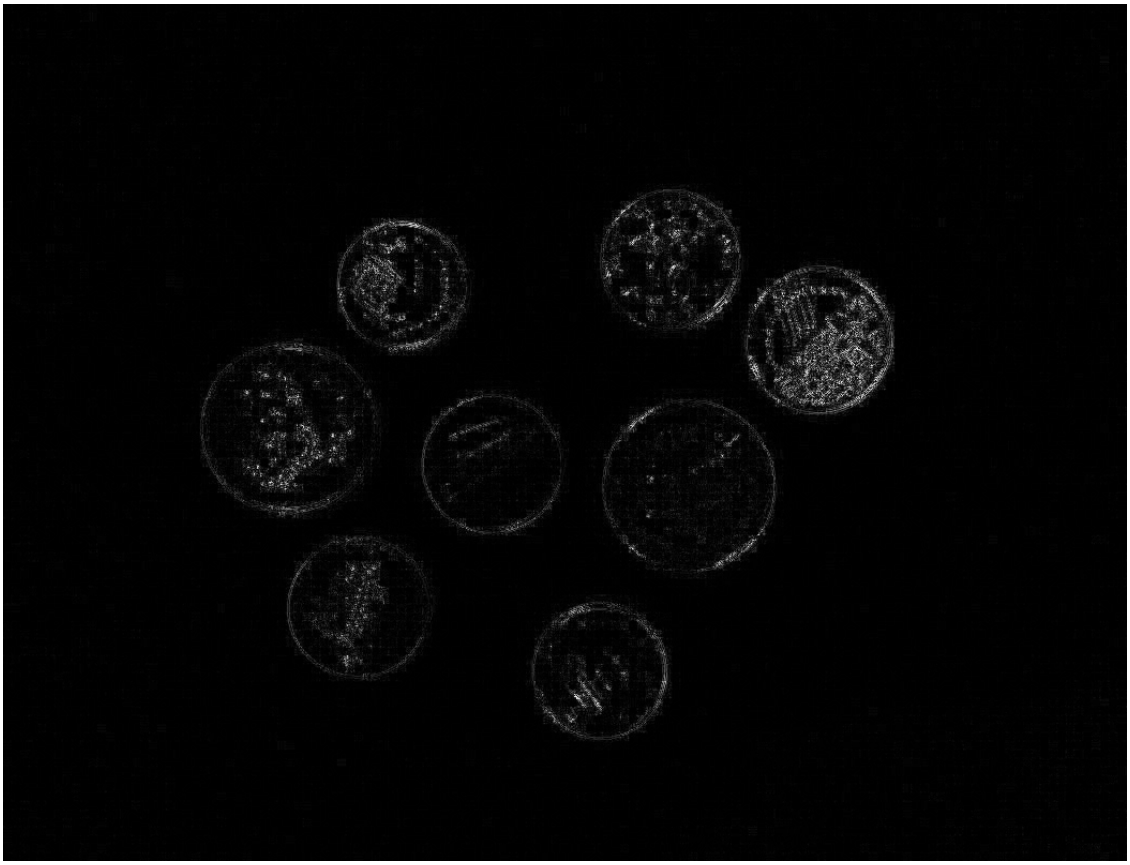
---

placian là một kỹ thuật xử lý ảnh giúp nổi bật các khu vực có biến độ chói mạnh, thường được sử dụng để phát
- Bộ lọc này được thiết kế để tăng cường cạnh và giảm vùng phẳng bằng cách sử dụng một hạt nhân có tổng bằ

- Sự chuyển đổi từ đen sang trắng được coi là một độ dốc dương, ngược lại sự chuyển đổi từu trắng sang đen được coi là một độ dốc âm. Theo đề cập ở chương 6 (Xử lý ảnh - Image Processing) thì ta biết rằng kiểu dữ liệu 8-bit unsigned integer không biểu diễn giá trị âm. Vậy để chắc chắn là ta sẽ tìm được tất cả các cạnh, ta sẽ sử dụng kiểu dữ liệu số chấm động (floating point), sau đó ta lấy giá trị tuyệt đối của độ dốc bức ảnh và chuyển thành kiểu 8-bit unsigned interger. Đây là một kỹ thuật quan trọng để không bị mất cạnh nào khi tính toán.

[ ]:
```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

#Load image and convert image to grayscale
image = cv2.imread('coins.jpg')
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Use Laplacian method
lap = cv2.Laplacian(image_gray, cv2.CV_64F)
lap = np.uint8(np.absolute(lap))
```

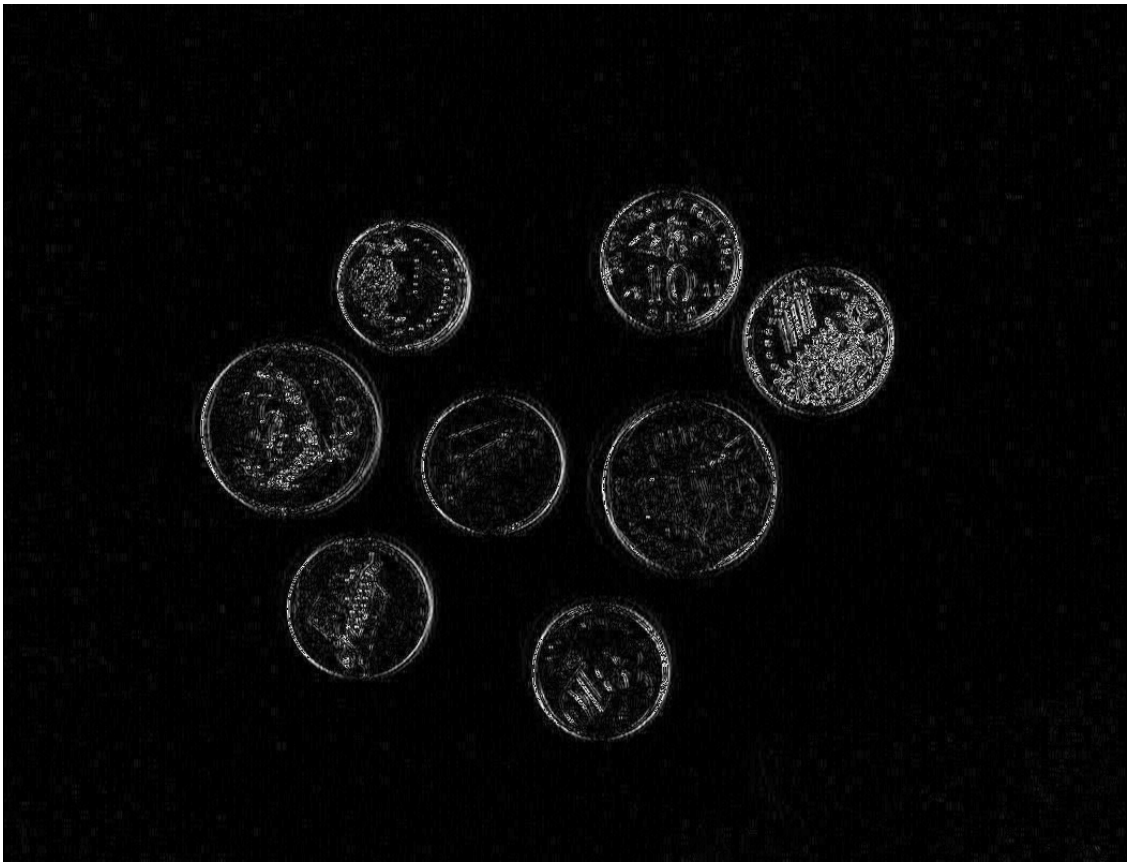
```
# Hiển thị hình ảnh Laplacian
cv2_imshow(lap)
```

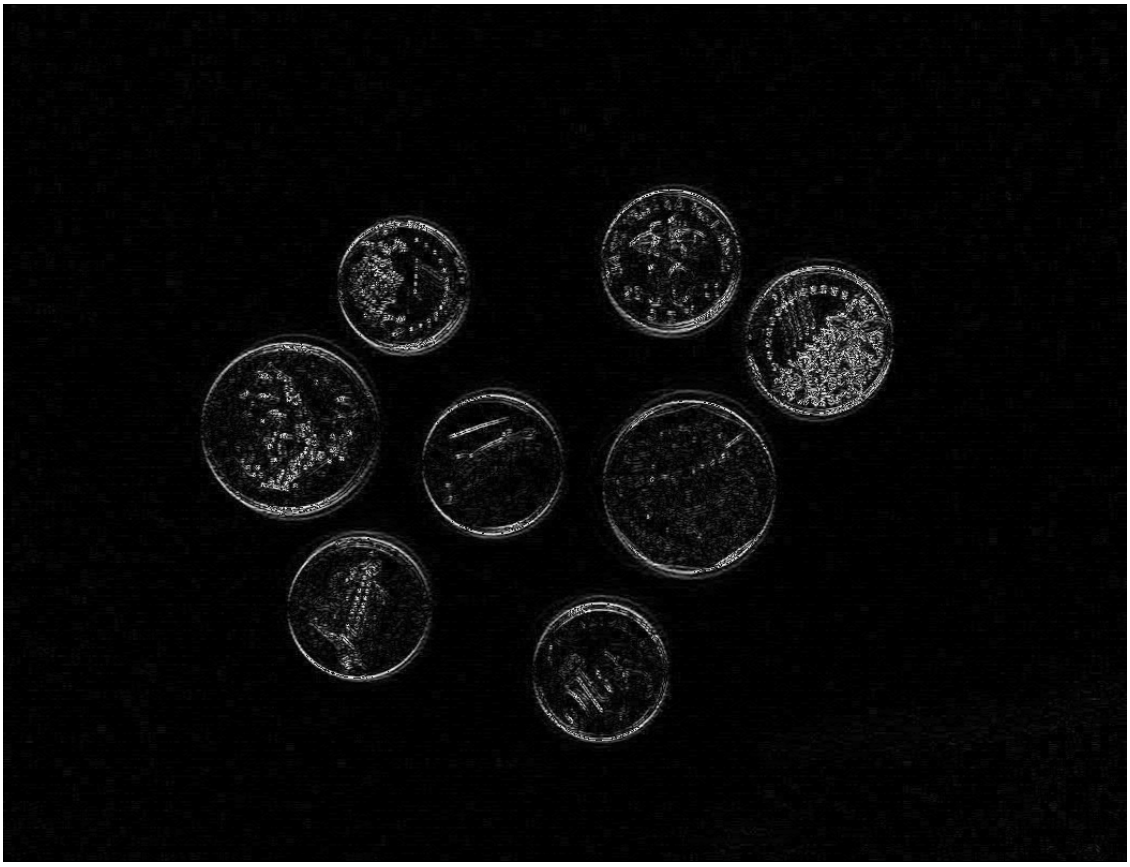


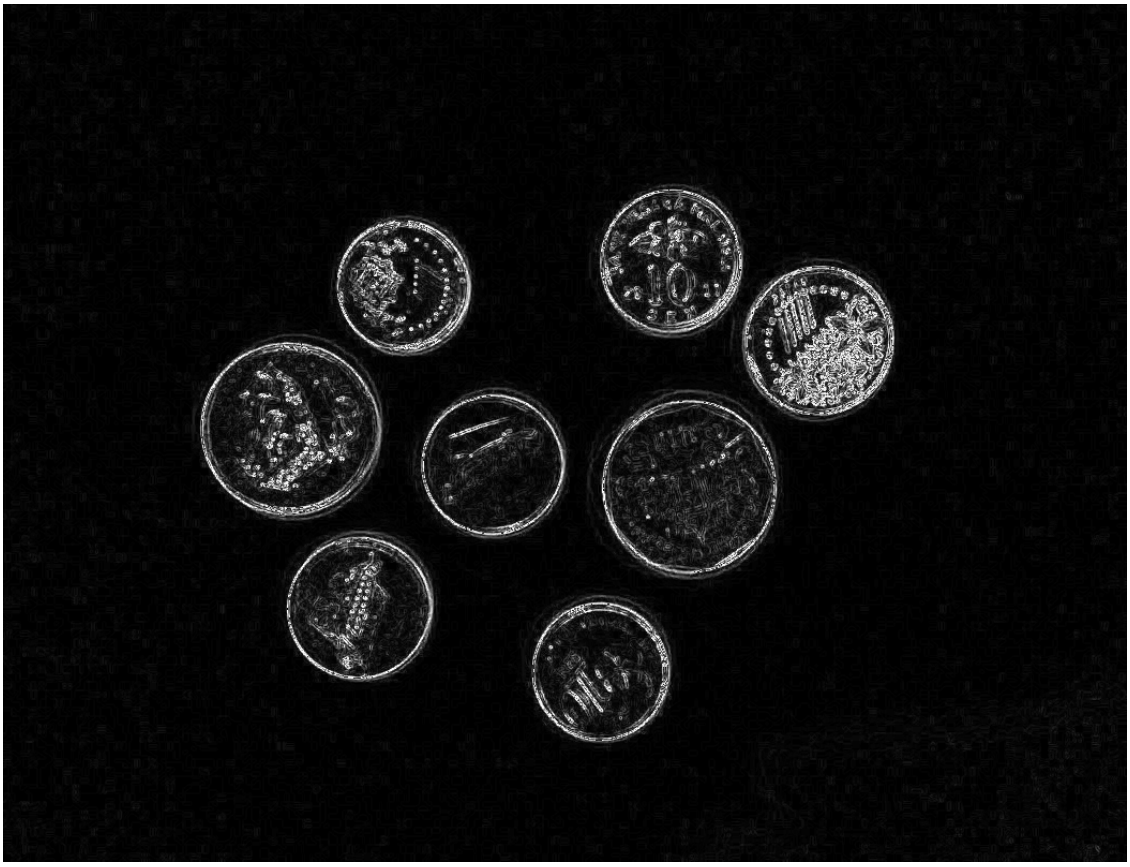
4. Sobel

- Bộ lọc Sobel là một trong những bộ lọc đơn giản và lâu đời nhất để phát hiện cạnh. Nó sử dụng hai hạt nhân (kernel) 3x3, một cho các cạnh ngang và một cho các cạnh dọc, được tích hợp với hình ảnh.
- Các hạt nhân được thiết kế gần đúng với đạo hàm bậc nhất của hình ảnh dọc theo trục x và y, để đo độ dốc hoặc tốc độ thay đổi cường độ điểm ảnh. Bộ lọc Sobel sau đó kết hợp các gradient theo chiều ngang và chiều dọc để thu được độ lớn và hướng của cạnh.
  - Để kết hợp hay sử dụng độ dốc ảnh trong cả 2 chiều x và y, chúng ta có thể sử dụng OR theo bit. Một phép toán OR đúng khi một trong hai *pixel* lớn hơn 0. Do đó, một pixel được đề cập sẽ trả về giá trị TRUE nếu một trong hai chiều ngang, chiều dọc được biểu diễn.
- Ưu điểm của bộ lọc Sobel là dễ thực hiện, tính toán nhanh và chống nhiễu tốt. Nhược điểm là nó nhạy cảm với các cạnh chéo, tạo ra các cạnh dày và không tính đến tính liên tục hoặc độ mịn của cạnh.

[ ]:
```
sobelX = cv2.Sobel(image_gray, cv2.CV_64F, 1, 0)
sobelY = cv2.Sobel(image_gray, cv2.CV_64F, 0 ,1)

sobelX = np.uint8(np.absolute(sobelX))
sobelY = np.uint8(np.absolute(sobelY))

sobelCombined = cv2.bitwise_or(sobelX, sobelY)

cv2_imshow(sobelX)
cv2_imshow(sobelY)

cv2_imshow(sobelCombined)

cv2.waitKey(0)
```







[ ]: -1

- Có một điều chúng ta cần chú ý đó là, tất cả các cạnh đều rất "nhiễu", không được sạch (clean) và giòn (crisp). Do đó, chúng ta cần sử dụng thuật toán cạnh Canny cho bước tiếp theo.

5. Canny Edge Detection

- Trong hình ảnh, thường tồn tại các thành phần như: vùng trơn, góc/cạnh và nhiễu. Cạnh trong ảnh mang đặc trưng quan trọng, thường là thuộc đối tượng trong ảnh (object). Do đó, để phát hiện cạnh trong ảnh, giải thuật Canny là một trong những giải thuật phổ biến/nổi tiếng nhất trong Xử lý ảnh.
- Thuật toán Canny là một thuật toán phổ biến để nhận diện cạnh trong xử lý ảnh. Nó được phát triển bởi John F. Canny vào năm 1986 và đã trở thành một trong những phương pháp phổ biến và hiệu quả nhất cho việc nhận diện cạnh.
- Hoạt động của phương pháp này có thể tóm tắt như sau:
    1. Giảm nhiễu: Ảnh thu nhận được từ thiết bị thu nhận thường có nhiều nhiễu, để nhiễu không ảnh hướng đến quá trình tách cạnh nó cần được giảm thiểu. Làm mờ ảnh, giảm nhiễu dùng bộ lọc Gaussian kích thước 5x5. Kích thước 5x5 thường hoạt động tốt cho giải thuật Canny.

2. Tính Gradient và hướng gradient: ta dùng bộ lọc SobelX và SobelY (3x3) để tính được ảnh đạo hàm Gx và Gy. Sau đó, ta tiếp tục tính ảnh Gradient và góc của Gradient theo công thức.

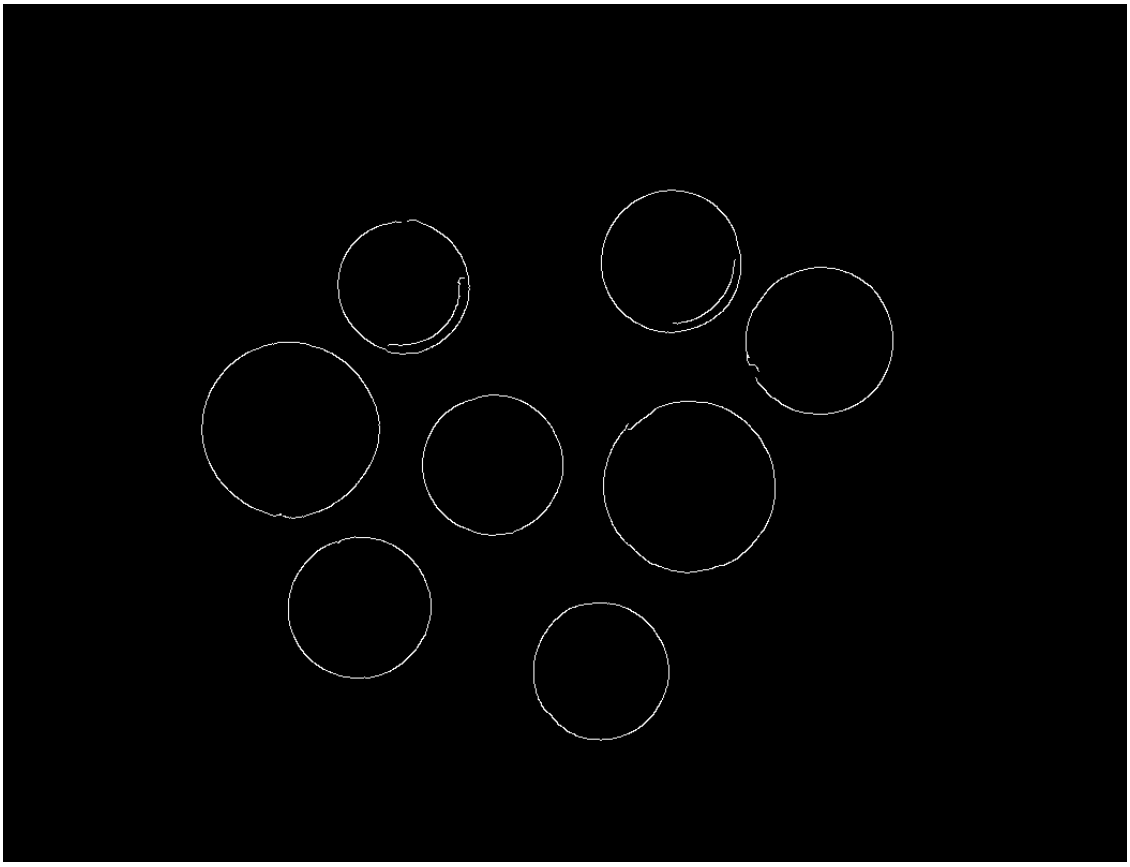
3. Non-maximum Suppression (viết tắt NMS): loại bỏ các pixel ở vị trí không phải cực đại toàn cục. Ở bước này, ta dùng một filter 3x3 lần lượt chạy qua các pixel trên ảnh gradient. Trong quá trình lọc, ta xem xét xem độ lớn gradient của pixel trung tâm có phải là cực đại (lớn nhất trong cục bộ - local maximum) so với các gradient ở các pixel xung quanh. Nếu là cực đại, ta sẽ ghi nhận sẽ giữ pixel đó lại. Còn nếu pixel tại đó không phải là cực đại lân cận, ta sẽ đặt độ lớn gradient của nó về 0. Ta chỉ so sánh pixel trung tâm với 2 pixel lân cận theo hướng gradient.

4. Lọc ngưỡng, ngưỡng độ trễ (Hysteresis Thresholding): Ở bước này ta sẽ quyết định xem toàn bộ các cạnh có những cạnh nào là thực sự và những cạnh nào không thông qua việc thiết lập ngưỡng gồm 2 giá trị, minVal và maxVal. Một cạnh bất kì có cường độ gradient lớn hơn maxVal thì chắc chắn là các cạnh và những cạnh có cường độ gradient nhỏ hơn sẽ không được coi là cạnh. Nếu một cạnh mà có những điểm ảnh nằm trong ngưỡng này thì có thể được xem xét thuộc cùng một cạnh hoặc không thuộc dựa trên sự kết nối của các điểm với nhau.

- Trong OpenCV, ta muốn sử dụng thuật toán Canny chỉ cần sử dụng những câu lệnh đơn giản sau:

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

[ ]:
```
blur_image = cv2.GaussianBlur(image_gray, (11, 11), 0)
cv2_imshow(blur_image)

canny_image = cv2.Canny(blur_image, 30, 180)
cv2_imshow(canny_image)
```



#IV. Contours

- Trước đó, chúng ta đã tìm ra cách phát hiện các cạnh của các đồng xu trong một bức ảnh. Bây giờ chúng ta sẽ sử dụng những cạnh này để tìm ra/nhận dạng và đếm có bao nhiêu đồng xu trong ảnh.
- OpenCV cung cấp các phương pháp để tìm *đường cong* (curves) trong ảnh, được gọi là contours (đường nét / đường viền). Có thể hiểu contours là "tập các điểm-liên-tục tạo thành một đường cong (curve) (boundary), và không có khoảng hở trong đường cong đó, đặc điểm chung trong một contour là các các điểm có cùng /gần xấu xỉ một giá trị màu, hoặc cùng mật độ. Contour là một công cụ hữu ích được dùng để phân tích hình dạng đối tượng, phát hiện đối tượng và nhận dạng đối tượng".
- Để tìm đường viền trong một hình ảnh, chúng ta cần một bản nhị phân của ảnh bằng cách sử dụng các phương pháp phát hiện cạnh (chương 10) hoặc ngưỡng (chương 9).
- Trong các ví dụ dưới, chúng ta sẽ sử dụng công cụ tìm cạnh Canny (dò cạnh Canny) để tìm đường viền của đồng xu, sau đó tìm đường viền thực thực sự của đồng xu

0.1 Khái niệm vè Contours, những điều liên quan

Contours trong xử lý ảnh là những đường nét, đường viền mà kết nối tất cả các điểm liên tục theo biên có cùng màu hoặc độ sáng. Các đường contour được sử dụng rộng rãi trong xử lý ảnh và thị

giác máy tính để phát hiện, nhận dạng và phân tích các đối tượng.

0.1.1 Đặc điểm của Contours:

Đường Nét Kết Nối Điểm Liên Tục: Contours trong xử lý ảnh được hiểu như những đường nét kết nối tất cả các điểm liên tục theo biên có cùng màu hoặc độ sáng. Điều này có nghĩa là, khi ta di chuyển từ một điểm trên contours đến một điểm khác, chúng ta sẽ chỉ gặp những điểm có giá trị pixel tương tự hoặc gần giống nhau. Contours tạo ra một đường biên liên tục và không bị gián đoạn, nối các điểm có liên quan đến nhau trên biên của đối tượng.

0.1.2 Biểu Diễn Đối Tượng:

Một trong những mục đích quan trọng của contours là biểu diễn ranh giới của các đối tượng trong ảnh. Các đối tượng có thể là các vật thể, kí tự, hoặc các thành phần khác của hình ảnh. Contours giúp chúng ta xác định vị trí và hình dạng của các đối tượng này, làm nổi bật các biên và đường viền của chúng.

Khi chúng ta xác định được contours, ta có thể thu được thông tin về diện tích, chu vi, và hình dạng của các đối tượng. Điều này là quan trọng trong nhiều ứng dụng như nhận dạng đối tượng, phân đoạn ảnh, và xử lý hình ảnh trong thị giác máy tính.

Contours giúp tạo nên một biểu diễn trực quan cho các đối tượng trong ảnh, giúp chúng ta hiểu và tận dụng thông tin biên của hình ảnh để thực hiện các nhiệm vụ xử lý và phân tích hình ảnh một cách hiệu quả.

0.2 Import các thư viện

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

0.3 enhance_coins()

[ ]:
```
def enhance_coins(image):
    if image is None:
        raise ValueError("Image not loaded successfully ")

    # Chuyển ảnh sang ảnh grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    plt.subplot(141)
    plt.imshow(gray_image, cmap='gray')
    plt.title('grayscale image')
    plt.axis('off')

    # GaussianBlur
    blurred_image = cv2.GaussianBlur(gray_image, (15, 15), 0)
```

```
    plt.subplot(142)
    plt.imshow(blurred_image, cmap='gray')
    plt.title('blurred image')
    plt.axis('off')

    # Canny
    canny_image = cv2.Canny(blurred_image, 30, 150)

    plt.subplot(143)
    plt.imshow(canny_image, cmap='gray')
    plt.title('canny image')
    plt.axis('off')

    # Dilate
    dilated_image = cv2.dilate(canny_image, None, iterations=2)

    plt.subplot(144)
    plt.imshow(dilated_image, cmap='gray')
    plt.title('dilated image')
    plt.axis('off')

    plt.show()

    # De-noise bằng Median Blur
    denoised_image = cv2.medianBlur(dilated_image, 5)

    return denoised_image
```

Hàm này nhận vào một ảnh màu và thực hiện các bước tiền xử lý để làm nổi bật các cạnh của đồng xu trong ảnh. Quá trình này bao gồm chuyển ảnh sang ảnh grayscale, áp dụng GaussianBlur để làm mịn ảnh và giảm nhiễu, sau đó sử dụng thuật toán Canny để phát hiện cạnh. Kết quả là ảnh nhị phân chứa các cạnh của đồng xu.

0.4 find_and_number_coins()

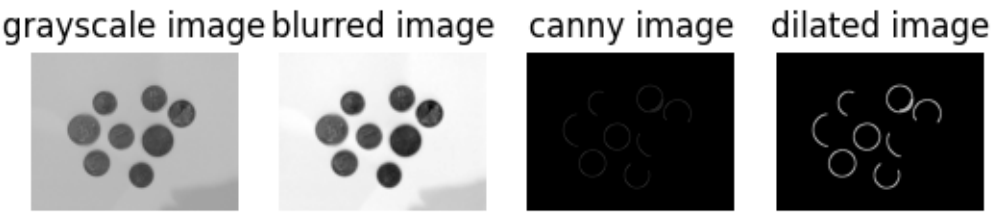
[ ]:
```
def find_and_number_coins(image):
    # Tìm contours trong ảnh
    (cnts, _) = cv2.findContours(enhance_coins(image), cv2.RETR_EXTERNAL, cv2.
 ↪CHAIN_APPROX_SIMPLE)

    # In ra số đồng xu tìm được trong ảnh
    print("Có {} coins trong ảnh".format(len(cnts)))

    # Đánh số thứ tự các đồng xu và vẽ đường biên bên ngoài
    for (i, c) in enumerate(cnts):
        # Tính diện tích của contour để lọc các contour nhỏ
```

```
        area = cv2.contourArea(c)
        if area > 100:
            # Vẽ đường biên bên ngoài đồng xu
            cv2.drawContours(image, [c], -1, (0, 255, 0), 2)

            # Lấy tọa độ trung tâm của contour
            M = cv2.moments(c)
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])

            # Vẽ đường tròn tại tọa độ trung tâm và đánh số thứ tự vào đồng xu
            cv2.circle(image, (cX, cY), 10, (0, 255, 0), -1)
            cv2.putText(image, f"Coin {i+1}", (cX - 20, cY - 20),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

    return image, cnts
```

Hàm này nhận vào ảnh và thực hiện việc tìm contours (đường biên) trong ảnh đã được xử lý bởi hàm enhance_coins. Sau đó, nó đánh số thứ tự và vẽ đường biên bên ngoài cho các đồng xu được tìm thấy. Kết quả trả về là ảnh đã được xử lý và đánh số.

0.5 Đọc ảnh, hiển thị

[ ]:
```
# Đọc ảnh từ file
image = cv2.imread('coins.jpg')

# Kiểm tra xem ảnh có được đọc thành công không
if image is None:
    raise ValueError("Ảnh không được load thành công")

# Hiển thị ảnh gốc với các đồng xu đã được đánh số thứ tự và có đường biên bên␣
 ↪ngoài
result_image, cnts = find_and_number_coins(image)
cv2_imshow(result_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```




```
Có 8 coins trong ảnh
```


Coin 7
Coin 8
Coin 6
Coin 5
Coin 4
Coin 3
Coin 2
Coin 1