HƯỚNG DẪN THỰC HÀNH XỬ LÝ ẢNH
Phân đoạn ảnh–Thresholding

Nguyễn Hải Triều

Khoa CNTT-Trường ĐH Nha Trang

trieunh@ntu.edu.vn

Ngày 25 tháng 11 năm 2023

Mục lục

1 Thresholding

Definition

Thresholding is the binarization of an image. In general, we seek to convert a grayscale image to a binary image, where the pixels are either 0 or 255.
Normally, we **use thresholding to focus on objects or areas** of particular interest in an image.

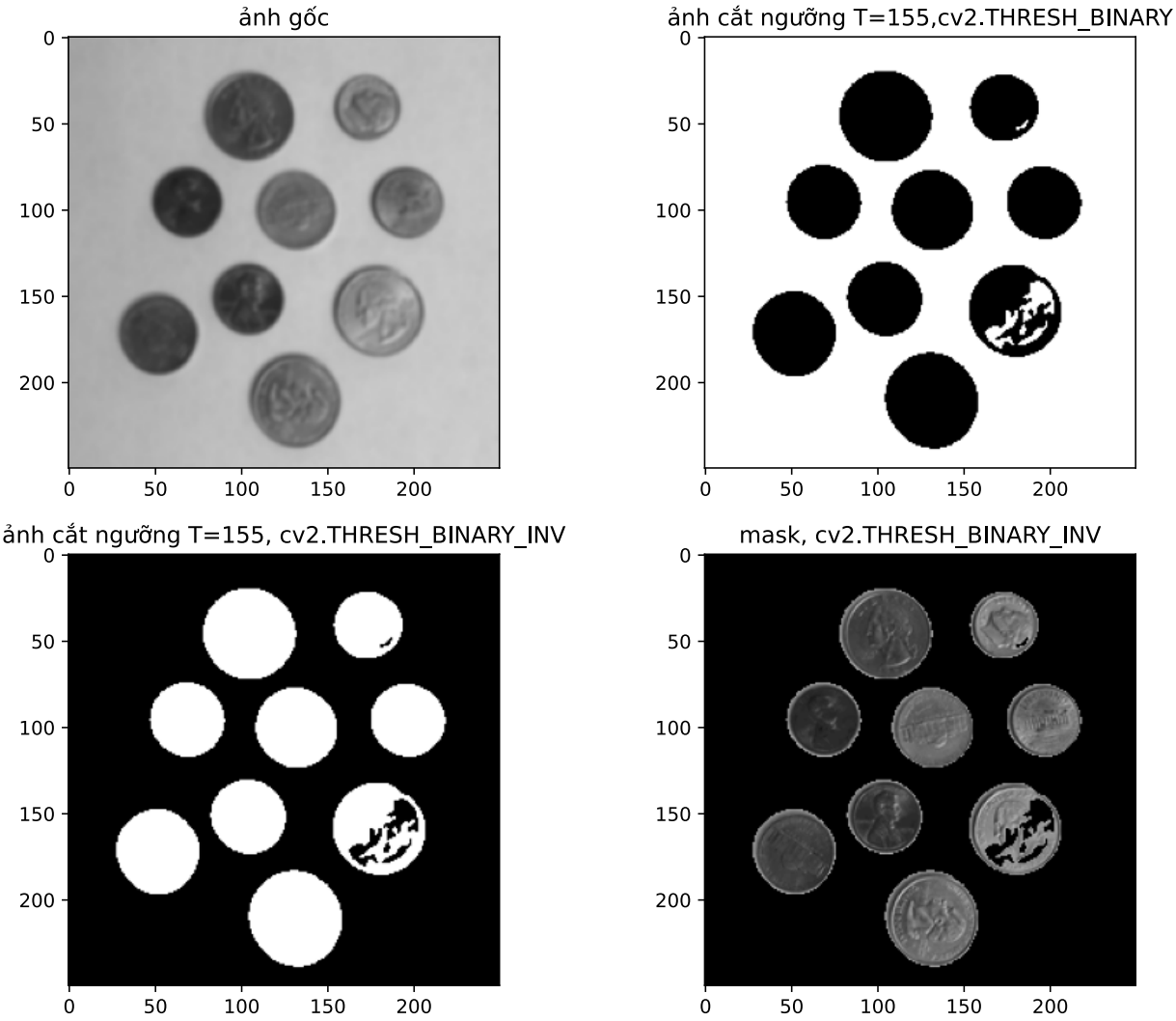
1.1 Simple Thresholding using OpenCV

Applying simple thresholding methods requires a threshold value T. All pixel intensities below T are set to 0. And all pixel intensities greater than T are set to 255.

- we compute the thresholded image using the **cv2.threshold** function
- The cv2.threshold function returns two values. The first is T, the value we manually specified for thresholding. The second is our actual thresholded image.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
# plt.rcParams.update({"text.usetex":True})
def cat_nguong_toan_cuc(image, T, inverse=False):
    '''
    NOTE: hàm cắt ngưỡng toàn cục, ngưỡng T chọn bằng tay.
    '''
    # các tham số khi gọi phương thức cv2.threshold:
        ## T là ngưỡng
        ## 255 là cường độsáng cực đại
        ## thresholding method: cv2.THRESH_BINARY-> nếu lớn hơn ngưỡng thì gán
            bằng cường độsáng cực đại; cv2.THRESH_BINARY_INV-> ngược lại nếu lớn
            hơn ngưỡng thì gán bằng 0;
    if inverse==True:
        (T, thresh) = cv2.threshold(image, T, 255, cv2.THRESH_BINARY_INV)
    else:
        (T, thresh) = cv2.threshold(image, T, 255, cv2.THRESH_BINARY)
    return thresh

if __name__=='__main__':
    img=cv2.imread('./images/coins.png',0)
    blurred = cv2.GaussianBlur(img, (5, 5), 0) # áp dụng Gaussian blurring với
        bán kính bằng 5 đểloại bỏ một vài cạnh có tần số cao mà chúng ta không
        quan tâm
    T=155 # thiết lập ngưỡng cho cắt ngưỡng toàn cục
    anh_cat_nguong=cat_nguong_toan_cuc(blurred, T)
    anh_cat_nguong_inv=cat_nguong_toan_cuc(blurred, T, inverse=True)
    # let's use our threshold as a mask and visualize only the the areas the
        coins in the image. we perform masking by using the cv2.bitwise_ and
        function
    mask_image=cv2.bitwise_and(img,img,mask=anh_cat_nguong_inv)

    # vẽ kết quả bằng matplotlib
    fig=plt.figure(figsize=(11,9))
    (ax1,ax2), (ax3,ax4)=fig.subplots(2,2)
    ax1.imshow(cv2.cvtColor(blurred,cv2.COLOR_BGR2RGB))
    ax1.set_title("ảnh gốc")

    ax2.imshow(cv2.cvtColor(anh_cat_nguong, cv2.COLOR_BGR2RGB))
    ax2.set_title("ảnh cắt ngưỡng T=155,cv2.THRESH_BINARY")

```

```
37      ax3.imshow(cv2.cvtColor(anh_cat_nguong_inv, cv2.COLOR_BGR2RGB))
38      ax3.set_title("ảnh cắt ngưỡng T=155, cv2.THRESH_BINARY_INV")
39
40      ax4.imshow(cv2.cvtColor(mask_image, cv2.COLOR_BGR2RGB)) # sử dụng cắt
            ngưỡng toàn cục ngược như một mặt nạ mask để che đi các phần không phải
            là các coins. Ảnh chỉ hiển thị khu vực có coins.
41      ax4.set_title("mask, cv2.THRESH_BINARY_INV")
42
43      plt.savefig("simple_thresholding.pdf",bbox_inches='tight')
44      plt.show()
```



Hình 1: Ví dụ simple thresholding trong OpenCV.

1.2 Adaptive thresholding

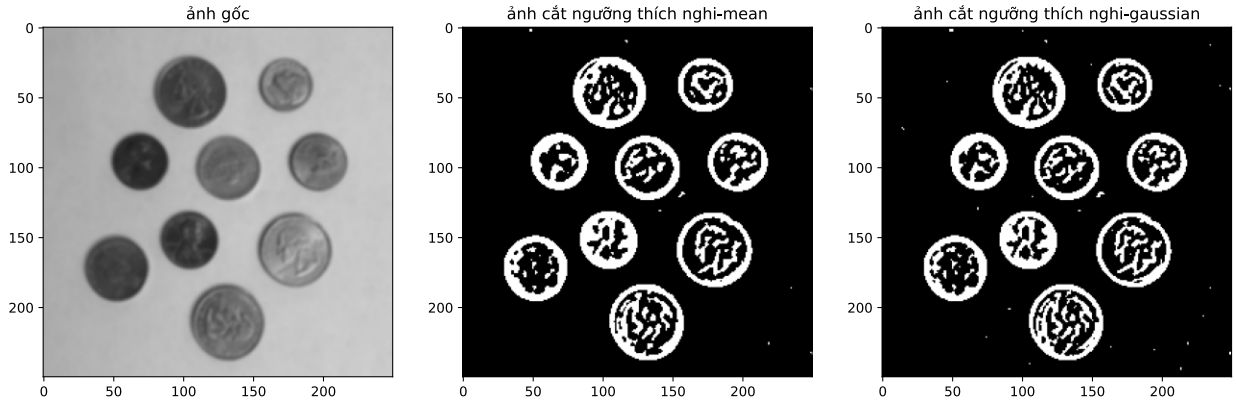
Cắt ngưỡng dựa trên thuộc tính vùng ảnh cục bộ

One of the downsides of using simple thresholding methods is that we **need to manually supply our threshold value T**. In order to overcome this problem, we can **use adaptive thresholding**, which considers **small neighbors of pixels** and then **finds an optimal threshold value T** for each neighbor.

- using the **cv2.adaptiveThreshold** function
- The first parameter: image we want to threshold
- The second parameter: maximum values
- The third parameter: compute the threshold for the current neighborhood of pixels. We indicate that we want to compute the mean of the neighborhood of pixels
  cv2.ADAPTIVE_THRESH_MEAN_C,
  cv2.ADAPTIVE_THRESH_GAUSSIAN_C
- The fourth parameter: thresholding method
  cv2.THRESH_BINARY,
  cv2.THRESH_BINARY_INV
- The fifth parameter: neighborhood size (kích thước vùng ảnh con Sxy, kích thước phải là 1 số nguyên lẻ)
- Finally, a parameter simply called an integer C. This value is an integer that is subtracted from the mean.

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
def cat_nguong_thich_nghi(image, neighborhood='mean',ksize=3, int_c=4):
    '''
    NOTE: hàm cắt ngưỡng thích nghi, chọn ngưỡng T tối ưu.
    '''
    # các tham số khi gọi phương thức cv2.threshold:
        ## 255 là cường độsáng cực đại
        ## thresholding method: cv2.THRESH_BINARY-> nếu lớn hơn ngưỡng thì gán
            bằng cường độsáng cực đại; cv2.THRESH_BINARY_INV-> ngược lại nếu lớn
            hơn ngưỡng thì gán bằng 0;
    if neighborhood=='mean':
        thresh = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
            cv2.THRESH_BINARY_INV, ksize, int_c)
    elif neighborhood=='gaussian':
        thresh = cv2.adaptiveThreshold(image, 255,
            cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, ksize, int_c)
    else:
        return image
    return thresh

```

```
if __name__=='__main__':
    img=cv2.imread('./images/coins.png',0)
    blurred = cv2.GaussianBlur(img, (5, 5), 0) # áp dụng Gaussian blurring với
        bán kính bằng 5 đểloại bỏ một vài cạnh có tần số cao mà chúng ta không
        quan tâm

    anh_cat_nguong_mean=cat_nguong_thich_nghi(blurred, ksize=11, int_c=4)
    anh_cat_nguong_gauss=cat_nguong_thich_nghi(blurred, 'gaussian', ksize=15,
        int_c=3)

    # vẽ kết quả bằng matplotlib
    fig=plt.figure(figsize=(16,9))
    ax1,ax2,ax3=fig.subplots(1,3)
    ax1.imshow(cv2.cvtColor(blurred,cv2.COLOR_BGR2RGB))
    ax1.set_title("ảnh gốc")

    ax2.imshow(cv2.cvtColor(anh_cat_nguong_mean, cv2.COLOR_BGR2RGB))
    ax2.set_title("ảnh cắt ngưỡng thích nghi-mean")

    ax3.imshow(cv2.cvtColor(anh_cat_nguong_gauss, cv2.COLOR_BGR2RGB))
    ax3.set_title("ảnh cắt ngưỡng thích nghi-gaussian")

    plt.savefig("adaptive_thresholding.pdf",bbox_inches='tight')
    plt.show()
```



Hình 2: Ví dụ adaptive thresholding trong OpenCV.

1.3 Otsu

We can automatically compute the threshold value of T is to **use Otsu's method**. OpenCV does not directly provide built-in support for Otsu's method. we use the otsu function in the **mahotas.thresholding package**.

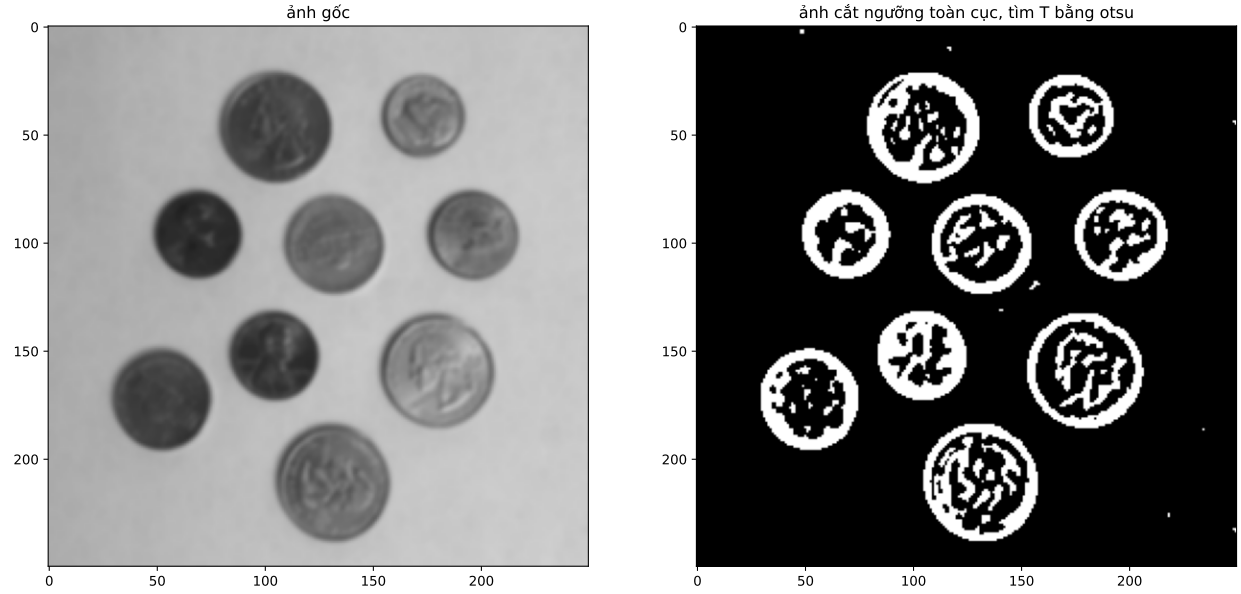
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import mahotas
```

```
5
6  def otsu_threshold(image):
7      '''
8      NOTE: hàm chọn ngưỡng T tối ưu bằng phương pháp otsu.
9      '''
10     T = mahotas.thresholding.otsu(image)
11     print("Otsu's threshold: {}".format(T))
12     thresh = image.copy()
13     thresh[thresh > T] = 255
14     thresh[thresh < 255] = 0
15     thresh = cv2.bitwise_not(thresh) #We then invert our threshold by using
           cv2.bitwise_not. This is equivalent to applying a cv2.THRESH_BINARY_INV
           thresholding type
16     return thresh
17
18 if __name__=='__main__':
19     img=cv2.imread('./images/coins.png',0)
20     blurred = cv2.GaussianBlur(img, (5, 5), 0) # áp dụng Gaussian blurring với
           bán kính bằng 5 đểloại bỏ một vài cạnh có tần số cao mà chúng ta không
           quan tâm
21
22     anh_cat_nguong_otsu=otsu_threshold(blurred)
23     # vẽ kết quả bằng matplotlib
24     fig=plt.figure(figsize=(16,9))
25     ax1,ax2=fig.subplots(1,2)
26     ax1.imshow(cv2.cvtColor(blurred,cv2.COLOR_BGR2RGB))
27     ax1.set_title("ảnh gốc")
28
29     ax2.imshow(cv2.cvtColor(anh_cat_nguong_mean, cv2.COLOR_BGR2RGB))
30     ax2.set_title("ảnh cắt ngưỡng toàn cục, tìm T bằng otsu")
31
32     plt.savefig("otsu_thresholding.pdf",bbox_inches='tight')
33     plt.show()
```

2 Gradients and edge detection

Definition

- **Edge detection** embodies mathematical methods to find points in an image where the brightness of pixel intensities changes distinctly.
- **Canny edge detection**, a multi-stage process of noise reduction (blurring), finding the gradient of the image (utilizing the Sobel kernel in both the horizontal and vertical direction), non-maximum suppression, and hysteresis thresholding.

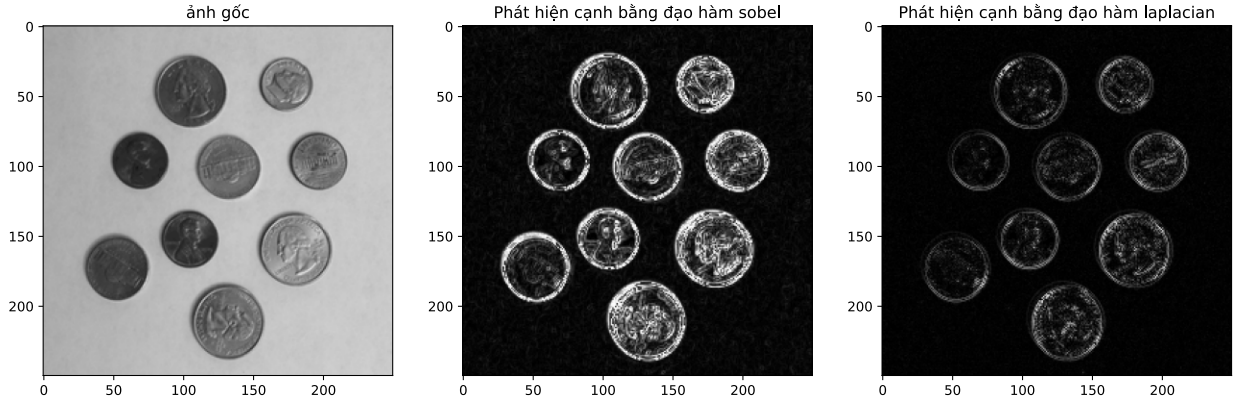


Hình 3: Ví dụ otsu's thresholding.

2.1 Laplacian and Sobel

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

def sobel_filter(image):
    '''
    NOTE: lọc sắc nét sobel.
    '''
    # Compute gradients along the X and Y axis, respectively
    sobelX = cv2.Sobel(image, cv2.CV_64F, 1, 0)
    sobelY = cv2.Sobel(image, cv2.CV_64F, 0, 1)
    # The sobelX and sobelY images are now of the floating
    # point data type -- we need to take care when converting
    # back to an 8-bit unsigned integer that we do not miss
    # any images due to clipping values outside the range
    # of [0, 255]. First, we take the absolute value of the
    # graident magnitude images, THEN we convert them back
    # to 8-bit unsigned integers
    sobelX = np.uint8(np.absolute(sobelX))
    sobelY = np.uint8(np.absolute(sobelY))
    # We can combine our Sobel gradient images using our
    # bitwise OR
    sobelCombined = cv2.bitwise_or(sobelX, sobelY)
    return sobelCombined
def laplacian_filter(image):
    # Compute the Laplacian of the image
    lap = cv2.Laplacian(image, cv2.CV_64F)
    lap = np.uint8(np.absolute(lap))
    return lap

```

```
if __name__=='__main__':
    img=cv2.imread('./images/coins.png',0)

    sobel_img=sobel_filter(img)
    lap_img=laplacian_filter(img)
    # vẽ kết quả bằng matplotlib
    fig=plt.figure(figsize=(16,9))
    ax1,ax2,ax3=fig.subplots(1,3)
    ax1.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
    ax1.set_title("ảnh gốc")

    ax2.imshow(cv2.cvtColor(sobel_img, cv2.COLOR_BGR2RGB))
    ax2.set_title("Phát hiện cạnh bằng đạo hàm sobel")

    ax3.imshow(cv2.cvtColor(lap_img, cv2.COLOR_BGR2RGB))
    ax3.set_title("Phát hiện cạnh bằng đạo hàm laplacian")

    plt.savefig("gradient_edges.pdf",bbox_inches='tight')
    plt.show()
```

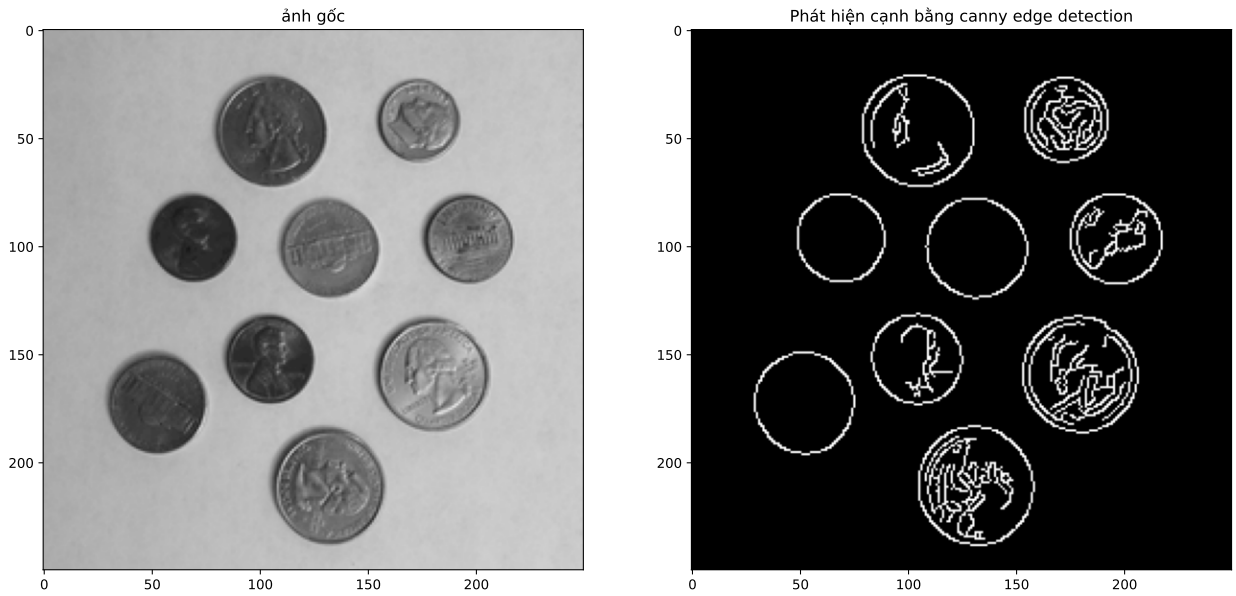


Hình 4: Ví dụ phát hiện cạnh bằng đạo hàm.

2.2 Canny edge detection

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

def canny_edge_detection(img, threshold1, threshold2):
    '''
    NOTE: phát hiện cạnh bằng canny_edge.
    '''
    # slightly to remove high frequency edges that we aren't interested in
    image = cv2.GaussianBlur(img, (5, 5), 0)
    # When performing Canny edge detection we need two values
    # for hysteresis: threshold1 and threshold2. Any gradient
    # value larger than threshold2 are considered to be an
```

```
    # edge. Any value below threshold1 are considered not to
    # ben an edge. Values in between threshold1 and threshold2
    # are either classified as edges or non-edges based on how
    # the intensities are "connected". In this case, any gradient
    # values below 30 are considered non-edges whereas any value
    # above 150 are considered edges.
    canny = cv2.Canny(image, threshold1, threshold2)
    return canny


if __name__=='__main__':
    img=cv2.imread('./images/coins.png',0)

    canny_img=canny_edge_detection(img,30,150)

    # vẽ kết quả bằng matplotlib
    fig=plt.figure(figsize=(16,9))
    ax1,ax2=fig.subplots(1,2)
    ax1.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
    ax1.set_title("ảnh gốc")

    ax2.imshow(cv2.cvtColor(canny_img, cv2.COLOR_BGR2RGB))
    ax2.set_title("Phát hiện cạnh bằng canny edge detection")

    plt.savefig("canny_edges.pdf",bbox_inches='tight')
    plt.show()
```



Hình 5: Ví dụ phát hiện cạnh bằng canny edge detection.

Tài liệu

[1] Ravishankar Chityala, Sridevi Pudipeddi. Image Processing and Acquisition using

Python (2020), Second Edition, Chapman & Hall/CRC.

[2] Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing (2018), Fourth Edition, Global Edition, Pearson.

[3] Adrian Rosebrock. Practical Python and OpenCV: An Introductory, Example Driven Guide to Image Processing and Computer Vision, Third Edition.