REPORT ON FINDING EDGE OF OBJECT IN IMAGE

Name: Trần Trường Giang
Student ID: BI9-090

I. Objective

To create an image that contains only the edge of the object from the source image.

Edges are the outside limit of an object, area, surface. With this definition we can betwen

II. Method

In this lecture we will use Canny Edge Detection to do the task. This algorithm includes 5 step:

Step 1: Filter out noise using a Gaussian kernel K of size 5x5:

$$K = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Step 2: Finding intensity gradient of the image

Calculate two derivatives with kernel Gx for horizontal changes and Gy for vertical changes

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$Gy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Then we can find edge gradient G and direction θ :

$$G = \sqrt{G_x^2 + G_y^2}$$

Direction θ is rounded to 1 out of 4 angels: [0, 45, 90, 135]

$$= arctan(\frac{Gy}{Gx})$$

Step 3: Thresholding

Use a threshold to eliminate noise and small thin edges.

Step 4: Non-maximum suppression

After getting gradient magnitude and direction, a full scan of the image is done to remove any unwanted pixels which may not constitute the edge. Which mean for every pixel the algorithm check for the adjacent pixels, that aligned with the gradient direction, if they form a local maximum move to next step, otherwise, the adjacents are suppressed to zeros

Step 5: Tracing edge

At this step edges with intensity gradient more than setted maximum value are classified as sure-edge and those below setted minimum value are classified as non-edge. For the remaining in between, whether they are considered sure-edge or not is based on if the edge is connected to previously established sure-edge.

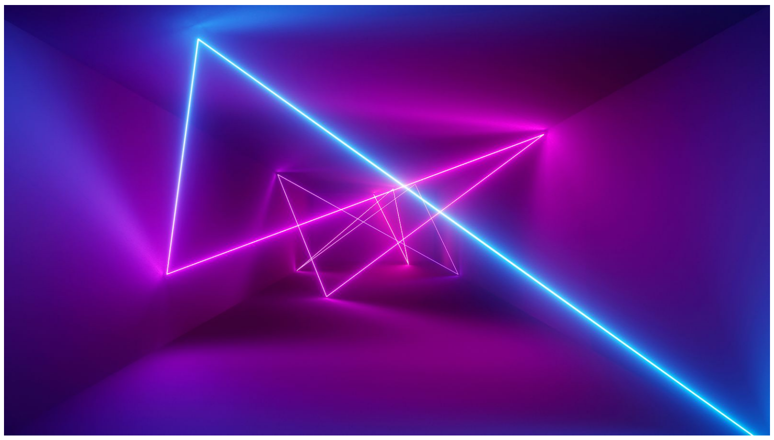
III. OpenCV Illustration

Code:

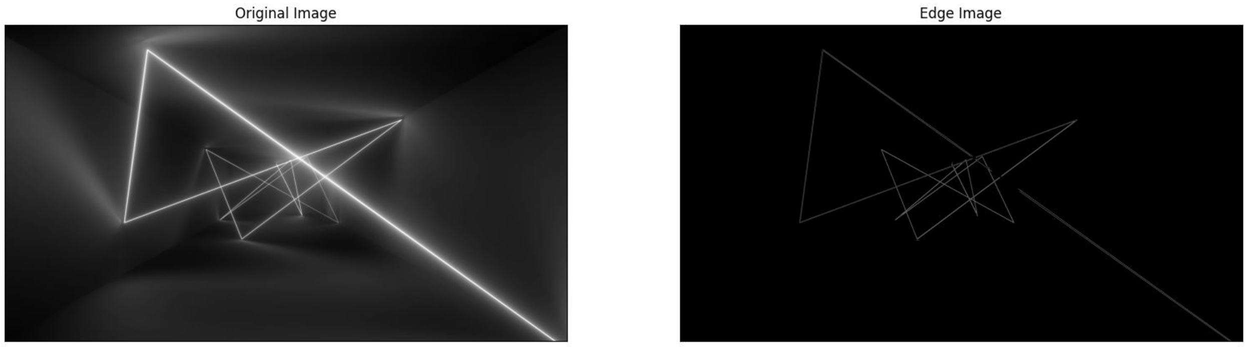
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('test.jpg',0)
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

For example 1 we are going to extract the edge from the neon light in the picture test.jpg bellow

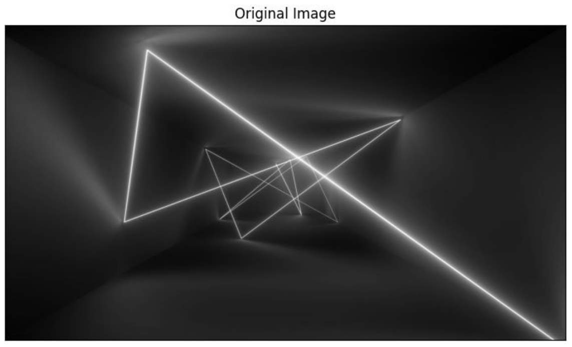
Example: test.jpg

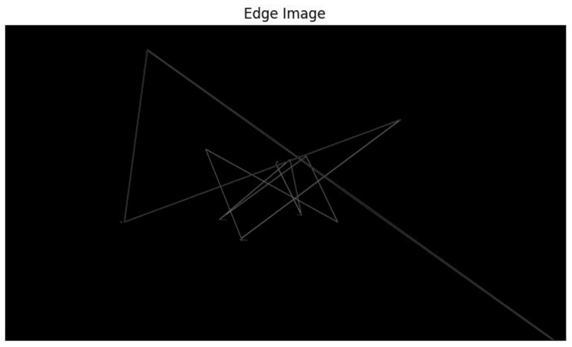


Canny(image,100,200)

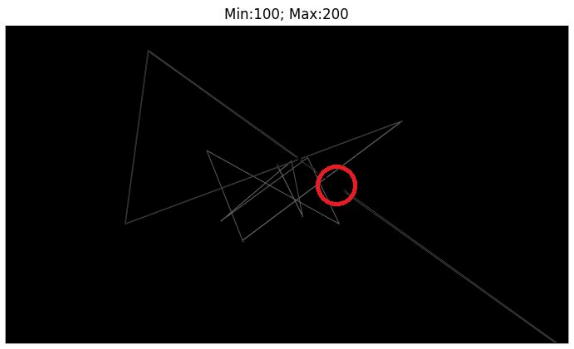


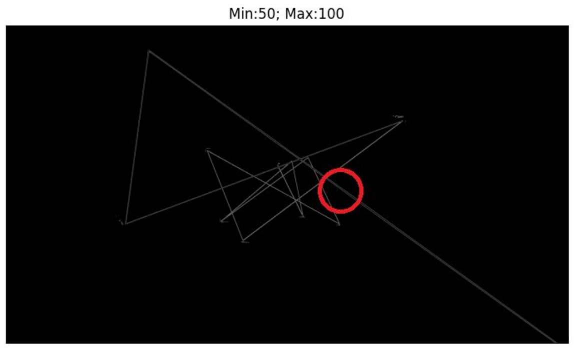
Canny(image,50,100)





With this simple test.jpg image we can see that to include more edges that we want we could simply reduce the maximum threshold so that more "sure-edges" were classified. If the result is still missing some edges that can be verified as "sure-edges" manually then also reduce the minimum threshold so that less edges are instantly classified as "non-edges".

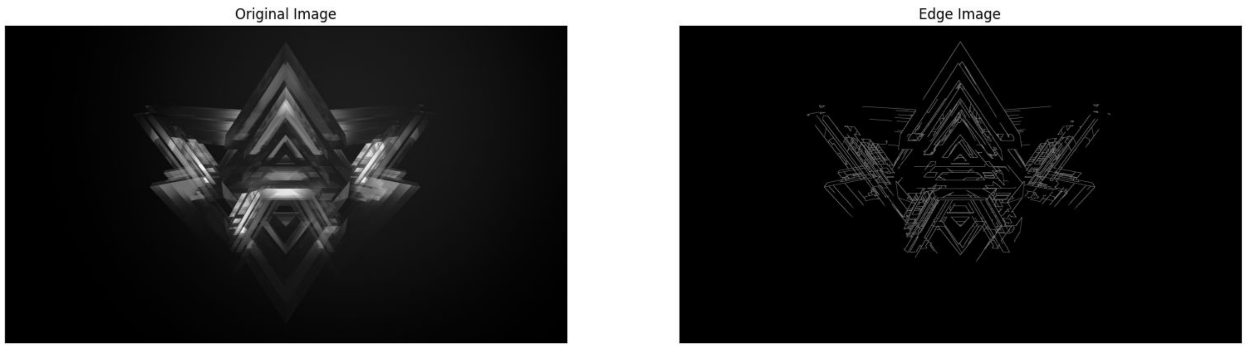




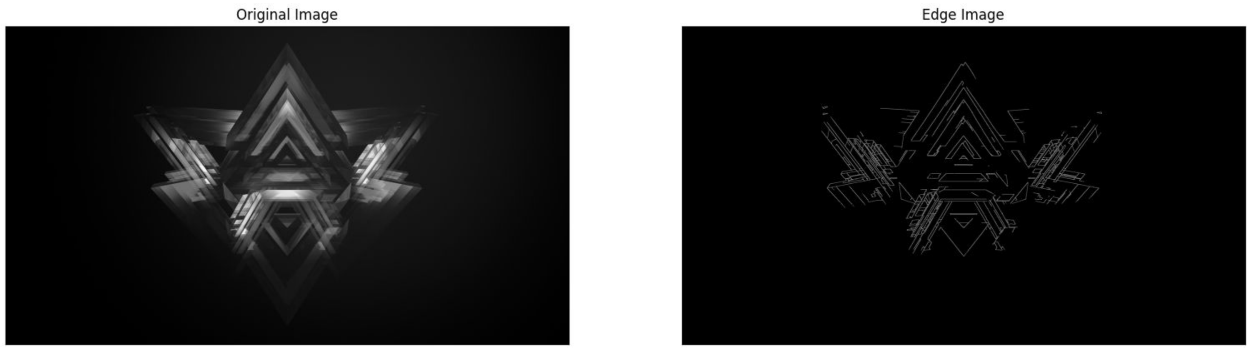
For example 2 we are going to extract the edge from the object in the picture test2.jpg below without taking any detail in the object itself.

Example: test2.jpg

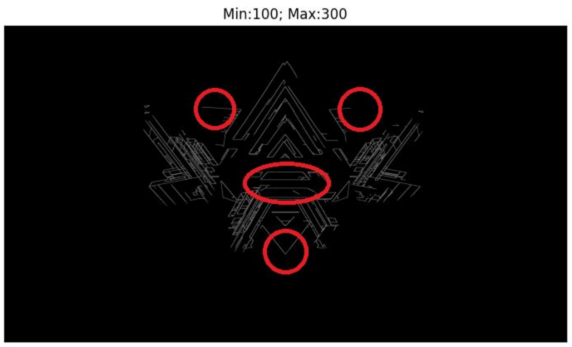
Canny(image,50,100)

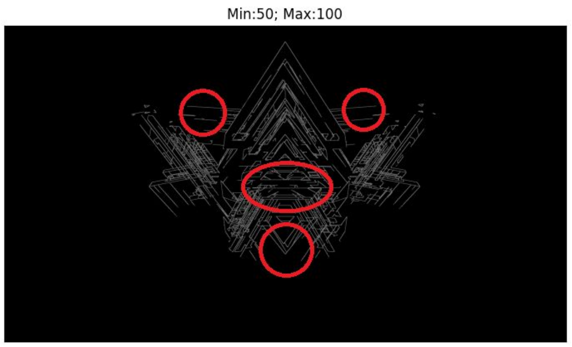


Canny(image,100,300)



As this example shows, if you want to eliminate the unwanted non-edges, one solution is to lower the minimum or the maximum threshold or both of them at the same time.





IV. Conclude

For canny() function with parameter:

cv.Canny(image,minVal,maxVal,L2gradient=true/false)

- We can change minVal and maxVal to change 2 minimum and maximum gradient threshold in step 5 to find required edges.
- We can change L2greadient to change how the algorithm do normalization to find edge gradient :
  + True : $G = \sqrt{G_x^2 + G_y^2}$
  + False : $G = |Gx| + |Gy|$