

Computer Vision Home Work 3

Name:許晉偉

Studentid:610415145

▪ Lane Lines Detection:

Q1. Gaussian Blur

```
: import numpy as np
import cv2
import matplotlib.pyplot as plt
img = cv2.imread('2.jpg')
Height, width, channels = img.shape[:3]
gray_img= np.zeros((Height, width, 1), np.uint16)
Gaussian_img= np.zeros((Height-2, width-2, 1), np.uint16)
Gaussian_Filter=np.zeros((3, 3, 1),np.float16)
GX=np.zeros((Height-2, width-2, 1), np.int32)
GY=np.zeros((Height-2, width-2, 1), np.int32)
G=np.zeros((Height-2, width-2, 1), np.int32)
cannyedge=np.zeros((Height-4, width-4, 1), np.int16)
theta=np.zeros((Height, width, 1), np.int16)
```

Read image and create temp_array we need to store the results in the next

```
: sigma =0.5**0.5
y, x = np.mgrid[-1:2, -1:2]
total=0
for i in range(3):
    for j in range(3):
        Gaussian_Filter[i][j]=(np.exp(-(x[i][j]**2+y[i][j]**2)/(2*sigma**2)))/(2*np.pi*sigma**2)
        total=total+Gaussian_Filter[i][j]
for i in range(3):
    for j in range(3):
        Gaussian_Filter[i][j]=np.round((Gaussian_Filter[i][j]/total),3)
```

Create an array like following image , first I created a 2-d array to store coordinate to the equation and do the normalization

$$\begin{bmatrix} 0.045 & 0.122 & 0.045 \\ 0.122 & 0.332 & 0.122 \\ 0.045 & 0.122 & 0.045 \end{bmatrix}$$

```
#grayscale image
for i in range(0,Height):
    for j in range(0,width):
        r=img[i,j,2]
        g=img[i,j,1]
        b=img[i,j,0]
        gray_img[i,j,0]=r*0.299+g*0.587+b*0.114
```

Convert the original image into a grayscale image

```
def convolution(x,y,img):
    result=0
    for i in range(0,3):
        for j in range(0,3):
            result=result+img[x+i,y+j,0]*Gaussian_Filter[i][j]
    return float(result)

for i in range(0,Height-3):
    for j in range(0,width-3):
        temp=convolution(i+1,j+1,gray_img)
        if(temp)<0 :
            temp=0
        if(temp)>255 :
            temp=255
        Gaussian_img[i,j,0]=np.round(temp,0)
cv2.imwrite('Gaussian.jpg', Gaussian_img)
```

Do the convolution for grayscale image by using Gaussian_Filter to achieve Gaussian blur

Q2. Canny Edge detection

1. Gradient calculation

```
# sobel detector
Gx = np.array([[ -1,  -2,  -1], [ 0,  0,  0], [ 1,  2,  1]])
Gy = np.array([[ -1,  0,  1], [-2,  0,  2], [-1,  0,  1]])
```

created a 2-d array to store the sobel detector, it used to calculate the approximate value of the gradient of the image intensity function

```
#Gradient calculation
def dot(x,y,img,direction):
    result=0
    for i in range(0,3):
        for j in range(0,3):
            if(direction=='x'):
                result=result+img[x+i,y+j,0]*Gx[i][j]
            elif((direction=='y')):
                result=result+img[x+i,y+j,0]*Gy[i][j]
    return result

for i in range(0,Height-4):
    for j in range(0,width-4):
        temp1=dot(i,j,Gaussian_img,'x')
        temp2=dot(i,j,Gaussian_img,'y')
        GX[i][j]=temp1
        GY[i][j]=temp2
for i in range(0,Height-2):
    for j in range(0,width-2):
        G[i][j]=np.sqrt(((GX[i][j]**2)+(GY[i][j]**2)))
        theta[i][j]=np.arctan2(GY[i][j],GX[i][j])*180/ np.pi
```

Find the gradients in the X and Y directions by using a sobel detector

To dot the Gaussian filtered image and find out the gradients intensity

and angle by following equation

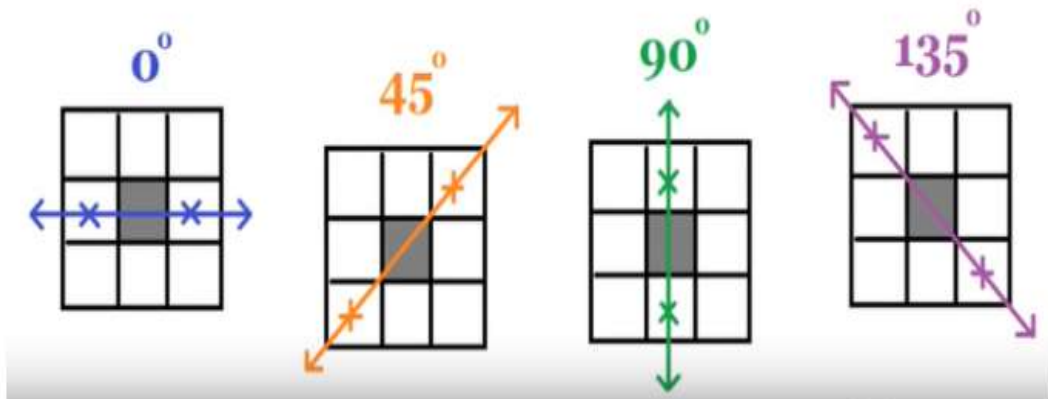
$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

2.Non-maximum suppression

```
#Non-maximum suppression
for i in range(0,Height-2):
    for j in range(0,width-2):
        if ((22.5> theta[i][j] >= -22.5) or (-157.5>= theta[i][j] >= -180) or (180>= theta[i][j] >= 157.5)):
            theta[i][j] = 0
        elif ((67.5> theta[i][j] >= 22.5) or (-112.5>= theta[i][j] >= -157.5)):
            theta[i][j] = 45
        elif ((112.5> theta[i][j] >= 67.5) or (-67.5>= theta[i][j] >= -112.5)):
            theta[i][j] = 90
        elif ((157.5> theta[i][j] >= 112.5) or (-22.5>= theta[i][j] >= -67.5)):
            theta[i][j] = 135
        else:
            print(theta[i][j])
```

In order to simplify the calculation, I transfer 360 angle to

0,45,90,and135 Same as picture



```

NEXT=255
BACK=255
for i in range(1,Height-4):
    for j in range(1,width-4):
        if(theta[i][j]==0):
            NEXT=G[i][j+1]
            BACK=G[i][j-1]
        elif(theta[i][j]==45):
            NEXT=G[i-1][j+1]
            BACK=G[i+1][j-1]
        elif(theta[i][j]==90):
            NEXT=G[i+1][j]
            BACK=G[i-1][j]
        elif(theta[i][j]==135):
            NEXT=G[i+1][j+1]
            BACK=G[i-1][j-1]
        if((G[i][j]>=NEXT) or (G[i][j]>=BACK)):
            cannyedge[i,j,0]=G[i][j]
        else:
            cannyedge[i,j,0]=0

```

Then do the Non-maximum suppression by using loop to find the degree

The angle is use to distinguish different gradient directions and compare the same directions element If it is larger than the other two, keep it, if not, abandon it

3.Double threshold

```

#DOUBLE THRESHOLD
threshold_high=120
threshold_low=60
for i in range(0,Height-4):
    for j in range(0,width-4):
        if(cannyedge[i,j,0]>=threshold_high):
            cannyedge[i,j,0]=255
        elif(cannyedge[i,j,0]<=threshold_low):
            cannyedge[i,j,0]=0

```

Set the high and low threshold and follow this mechanism to find the edge.

4.Edge Tracking by Hysteresis

```
#Edge Tracking by Hysteresis
for i in range(1,Height-6):
    for j in range(1,width-6):
        if ((threshold_high>cannyedge[i,j,0]) and (cannyedge[i,j,0]>threshold_low)):
            if ((cannyedge[i+1,j-1,0] == 255) or (cannyedge[i+1, j,0] == 255) or (cannyedge[i+1,j+1,0] == 255)
                or (cannyedge[i,j-1,0] == 255) or (cannyedge[i,j+1,0] == 255)
                or (cannyedge[i-1,j-1,0] == 255) or (cannyedge[i-1,j,0] == 255)
                or (cannyedge[i-1,j+1,0] == 255)):
                cannyedge[i,j,0]=255
            else:
                cannyedge[i, j,0]=0
```

This method is to consider the point between the high threshold and the low threshold if the nearby point is edge, then it is also regarded as an edge

Q3. *Hough Transform*

```
theta_h =(np.linspace(-90, 90, 181))
d = np.sqrt((cannyedge_f.shape[0]-1)**2 + (cannyedge_f.shape[1]-1)**2)
D= int(np.ceil(d))
r = np.linspace(-D, D,2*D+1)
parameter_space= np.zeros((len(r), len(theta_h), 1))
threshold = 100
Range =[0,540,0,960]
result_img = img.copy()
```

Variable D is diagonal, The variable theta_h is the angle from $-\pi/2 \sim \pi/2$, use the two Variable to create Parameter space to map Cartesian coordinates

The threshold in here, Make Hough's conversion line more accurate

```

: for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if (cannyedge_f[i,j] == 255): #偵測到邊緣
            for thIdx in range(len(theta_h)): #90度跑到90度
                rVal = j * np.cos(theta_h[thIdx]*np.pi/180.0) + i * np.sin(theta_h[thIdx]*np.pi/180.0)
                rIdx = np.nonzero(np.abs(r-rVal) == np.min(np.abs(r-rVal)))
                parameter_space[rIdx[0], thIdx] += 1
    for p in range(parameter_space.shape[0]):
        for t in range(parameter_space.shape[1]):
            if parameter_space[p,t] >= threshold:
                if ((-60 <= theta_h[t] <= 50) or (50 <= theta_h[t] <= 60)):
                    a = np.cos(theta_h[t]*np.pi/180.0)
                    b = np.sin(theta_h[t]*np.pi/180.0)
                    x0 = a * r[p]
                    y0 = b * r[p]
                    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)
result_img[Range[0] : Range[1] , Range[2] : Range[3]] = img[Range[0] : Range[1] , Range[2] : Range[3]]
cv2.imwrite('Hough4.jpg',result_img.astype('uint8'))

```

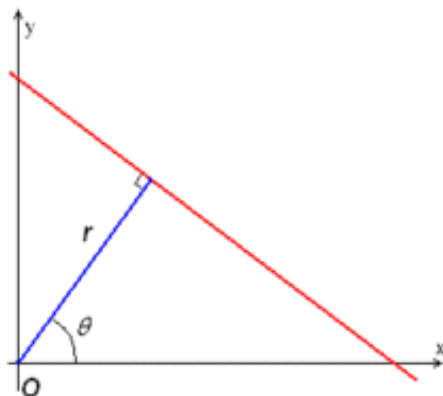
First we need to transform the coordinate system into a (r-theta) Polar coordinates system

$R = X * \cos(\theta) + Y * \sin(\theta)$:

A point in the original coordinate system converted to (r-theta) will become a line In the (r-theta) coordinate system, each point represents a line

The more points the lines overlap, the higher reliability of the line in the original coordinate system.

And set the start point and end point to draw the picture



Result images:

Image 1.jpg:



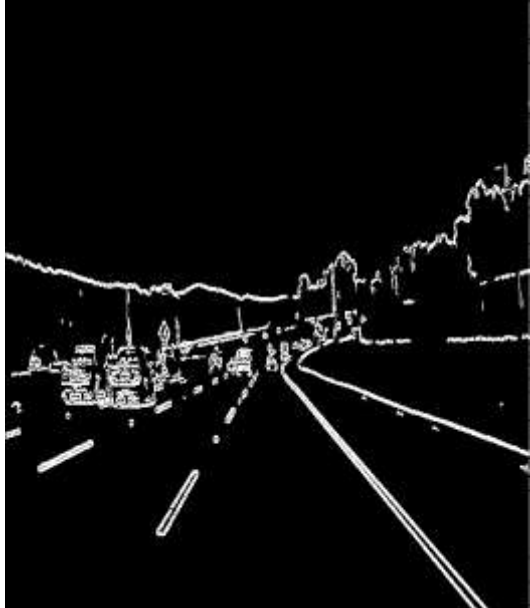
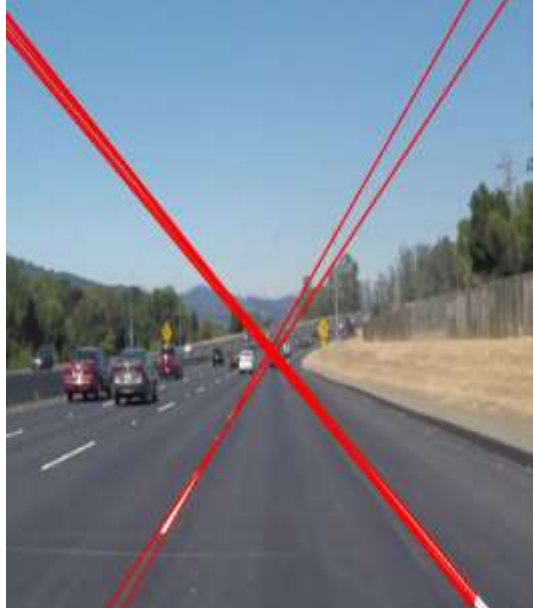
Original	Gaussian
	
Canny	Hough
	

Image 2.jpg:

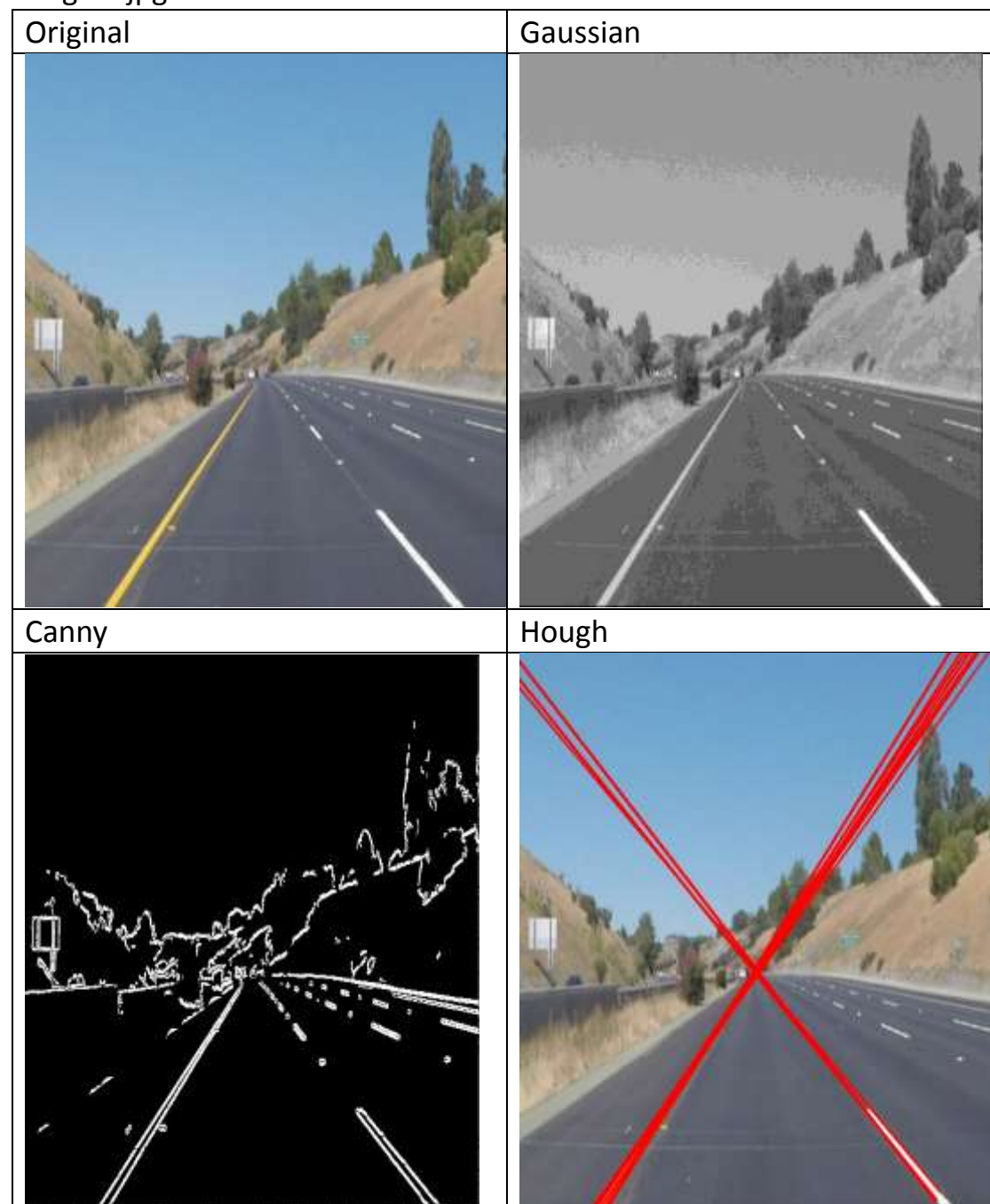


Image 3.jpg:

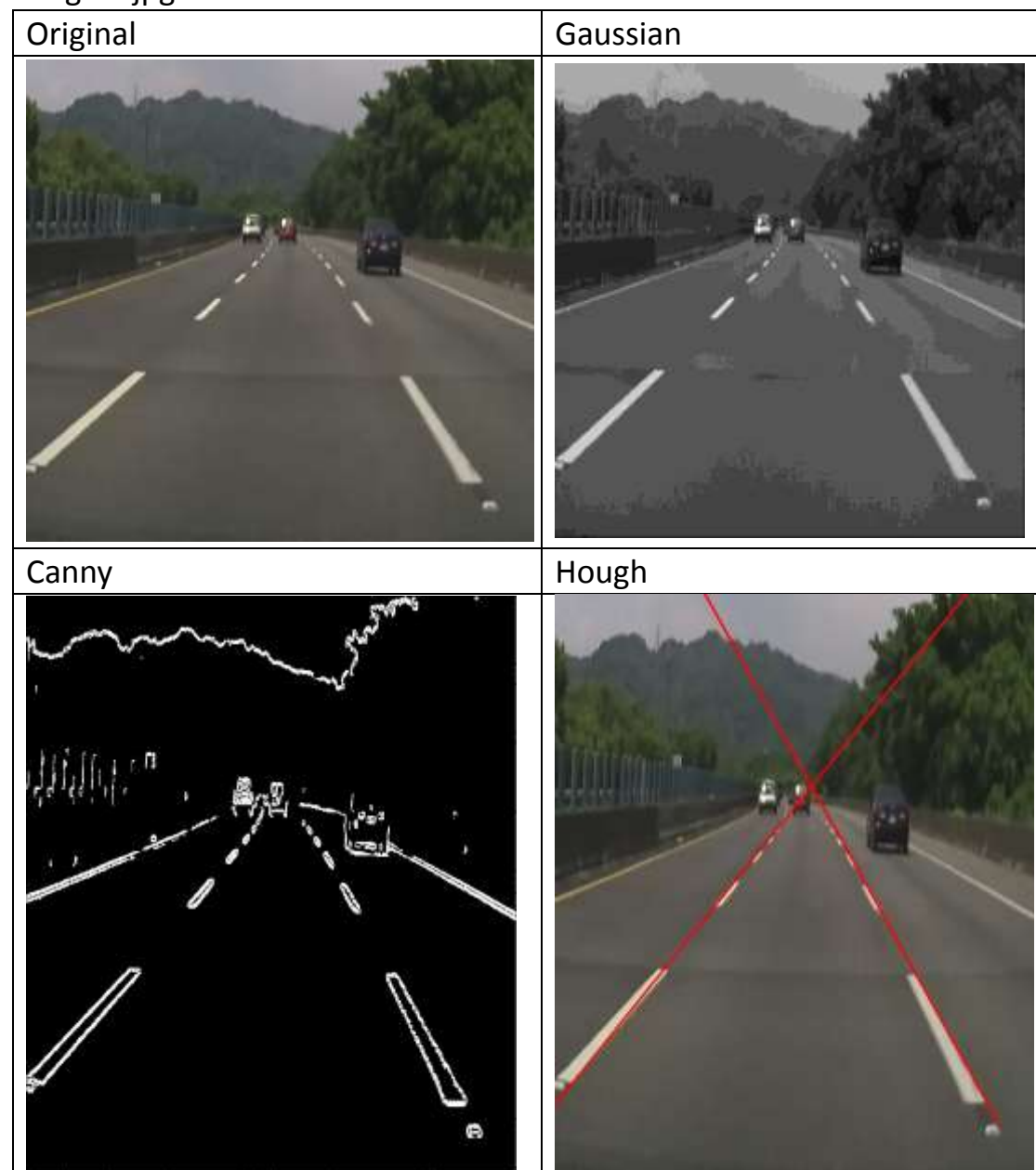




Image 4.jpg:

Original	Gaussian
	
Canny	Hough
