Computer Vision HW3

612415013 電機碩一 蕭宥羽

一、 程式碼及方法

```
from math import pi
import numpy as np
import cv2
import matplotlib.pyplot as plt
import math
```

Import python libraries

```
def RGB2GRAY(image):
    return np.dot(image[...,:3], [0.21, 0.72, 0.07]).astype(np.uint8)
```

將原圖片轉為灰階圖片

```
def gaussian_filter(image, kernel_size, sd):
    height, width = image.shape[:2]
    matrix = np.zeros((height - kernel_size + 1, width - kernel_size + 1))

    # 生成高斯濾波器
    gaussian_filter = np.fromfunction(
        lambda x, y: (1/(2*np.pi*sd**2)) * np.exp(-((x-(kernel_size-1)/2)**2 + (y-
(kernel_size-1)/2)**2) / (2*sd**2)),
        (kernel_size, kernel_size)
    )

    # 歸一化濾波器
    gaussian_filter /= np.sum(gaussian_filter)



    #對每個窗口應用高斯濾波器
    for i in range(matrix.shape[0]):
        for j in range(matrix.shape[1]):
            window = image[i:i+kernel_size, j:j+kernel_size]
            matrix[i, j] = np.sum(window * gaussian_filter)

    return matrix.astype(np.uint8)
```

高斯模糊，可以控制 kernel size 跟標準差

```python
def apply_sobel(image):
    sobel_x_kernel = np.array([[-1, 0, 1],
                               [-2, 0, 2],
                               [-1, 0, 1]])

    sobel_y_kernel = np.array([[-1, -2, -1],
                               [0, 0, 0],
                               [1, 2, 1]])

    rows, cols = image.shape

    sobel_x_result = np.zeros((rows - 2, cols - 2))
    sobel_y_result = np.zeros((rows - 2, cols - 2))

    # 進行卷積操作
    for i in range(rows - 2):
        for j in range(cols - 2):
            sobel_x_result[i, j] = np.sum(image[i:i+3, j:j+3] * sobel_x_kernel)
            sobel_y_result[i, j] = np.sum(image[i:i+3, j:j+3] * sobel_y_kernel)

    # 計算邊緣強度和方向
    magnitude = np.sqrt(sobel_x_result**2 + sobel_y_result**2)
    angle = np.arctan2(sobel_y_result, sobel_x_result)

    return magnitude, angle
```

sobel_x、sobel_y 濾波器，將高斯模糊後的圖像作為輸入，並使用 Sobel 濾波器進行卷積操作，計算出每個像素的水平和垂直方向的梯度。接著，根據這些梯度值計算每個像素的邊緣強度（Magnitude）和梯度方向（Angle）。

```python
def non_max_suppression(magnitude, angle):
    rows, cols = magnitude.shape
    result = np.zeros_like(magnitude)

    angle = np.rad2deg(angle) % 180

    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            q = 255
            r = 255
```

```
            # 檢查梯度方向，並找出相鄰的兩個點
            if (0 <= angle[i, j] < 22.5) or (157.5 <= angle[i, j] <= 180):
                q = magnitude[i, j+1]
                r = magnitude[i, j-1]
            elif (22.5 <= angle[i, j] < 67.5):
                q = magnitude[i+1, j-1]
                r = magnitude[i-1, j+1]
            elif (67.5 <= angle[i, j] < 112.5):
                q = magnitude[i+1, j]
                r = magnitude[i-1, j]
            elif (112.5 <= angle[i, j] < 157.5):
                q = magnitude[i-1, j-1]
                r = magnitude[i+1, j+1]

            # 如果當前點的梯度是最大的，則保留，否則設為 0
            if (magnitude[i, j] >= q) and (magnitude[i, j] >= r):
                result[i, j] = magnitude[i, j]
            else:
                result[i, j] = 0

    return result
```

非最大值抑制（Non-maximum Suppression）的操作，主要用於精簡邊緣圖像，僅保留梯度方向上的局部極大值。

```
def connect_weak_edges(magnitude, high_threshold, low_threshold):
    rows, cols = magnitude.shape
    result = np.zeros_like(magnitude)


    strong_edge = (magnitude >= high_threshold)
    weak_edge = (magnitude >= low_threshold) & (magnitude < high_threshold)

    result[strong_edge] = 255

    def recursive_connect(i, j):
        if i < 0 or i >= rows or j < 0 or j >= cols or result[i, j] == 255:
            return
        if weak_edge[i, j]:
            result[i, j] = 255
            for x in range(-1, 2):
                for y in range(-1, 2):
```

```
                    recursive_connect(i + x, j + y)

    for i in range(rows):
        for j in range(cols):
            if weak_edge[i, j]:
                recursive_connect(i, j)

    return result
```

connect Weak Edges，這是 Canny 邊緣檢測的最後一個步驟。將將低於高閾值但高於低閾值的弱邊緣透過連接的方式變成邊緣。

```
def canny_edge_detection(image, high_threshold, low_threshold):
    gradient_magnitude,gradient_angle=apply_sobel(image)

    non_max_suppressed=non_max_suppression(gradient_magnitude, gradient_angle)

    edge=connect_weak_edges(non_max_suppressed, high_threshold, low_threshold)

    return edge.astype(np.uint8)
```

Canny 邊緣檢測，包括 Sobel 濾波器、非最大值抑制和連接弱邊緣

```
def Hough_Transform(image, edge, threshold):
    height, width = edge.shape
    thetas = np.arange(0, 180, step=1)
    rhos = np.linspace(-int(np.ceil(np.sqrt(width * width + height * height))),
int(np.ceil(np.sqrt(width * width + height * height))), 2*int(np.ceil(np.sqrt(width *
width + height * height))))
    accumulator = np.zeros((len(rhos), len(rhos)))

    for y in range(height):
      for x in range(width):
        if edge[y][x] != 0:
          point = [y - (height / 2), x - (width / 2)]
          for i in range(len(thetas)):
            rho = (point[1] * np.cos(np.deg2rad(thetas))[i]) + (point[0] *
np.sin(np.deg2rad(thetas))[i])
            theta = thetas[i]
            rho_idx = np.argmin(np.abs(rhos - rho))
            accumulator[rho_idx][i] += 1
```

```
    for y in range(accumulator.shape[0]):
        for x in range(accumulator.shape[1]):
            if accumulator[y][x] > threshold:
                rho = rhos[y]
                theta = thetas[x]
                a = np.cos(np.deg2rad(theta))
                b = np.sin(np.deg2rad(theta))
                x0 = (a * rho) + (width / 2)
                y0 = (b * rho) + (height / 2)
                x1 = int(x0 + 1000 * (-b))
                y1 = int(y0 + 1000 * (a))
                x2 = int(x0 - 1000 * (-b))
                y2 = int(y0 - 1000 * (a))
                pt1 = (x1, y1)
                pt2 = (x2, y2)
                cv2.line(image, pt1, pt2, (0,0,255), 3, cv2.LINE_AA)

    return image.astype(np.uint8)
```

霍夫轉換，並畫在原圖上面

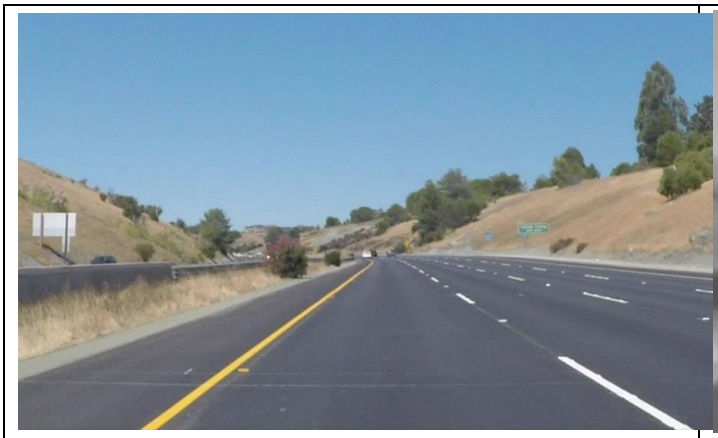
```
test_img=cv2.imread("test_img/1.jpg")

gray_img = RGB2GRAY(test_img)

#gaussian_img=gaussian_filter(gray_img, kernel_size=7, sd=5)
gaussian_img=gaussian_filter(gray_img, kernel_size=3, sd=1)
cv2.imwrite("result_img/result1_img1.png", gaussian_img)

edge_img = canny_edge_detection(gaussian_img, low_threshold=165, high_threshold=180)
cv2.imwrite("result_img/result1_img2.png", edge_img)

hough_img = Hough_Transform(test_img, edge_img,90)
cv2.imwrite("result_img/result1_img3.png", hough_img)


test_img=cv2.imread("test_img/2.jpg")
gray_img = RGB2GRAY(test_img)

gaussian_img=gaussian_filter(gray_img, kernel_size=9, sd=7)
cv2.imwrite("result_img/result2_img1.png", gaussian_img)


edge_img = canny_edge_detection(gaussian_img, low_threshold=70, high_threshold=100)
```

```
cv2.imwrite("result_img/result2_img2.png", edge_img)

hough_img = Hough_Transform(test_img, edge_img,95)
cv2.imwrite("result_img/result2_img3.png", hough_img)


test_img=cv2.imread("test_img/3.jpg")
gray_img = RGB2GRAY(test_img)

gaussian_img=gaussian_filter(gray_img, kernel_size=9, sd=7)
cv2.imwrite("result_img/result3_img1.png", gaussian_img)

edge_img = canny_edge_detection(gaussian_img, low_threshold=50, high_threshold=100)
cv2.imwrite("result_img/result3_img2.png", edge_img)

hough_img = Hough_Transform(test_img, edge_img,95)
cv2.imwrite("result_img/result3_img3.png", hough_img)
```

輸入圖片，並調整高斯模糊及閥值等參數，最後輸出並儲存

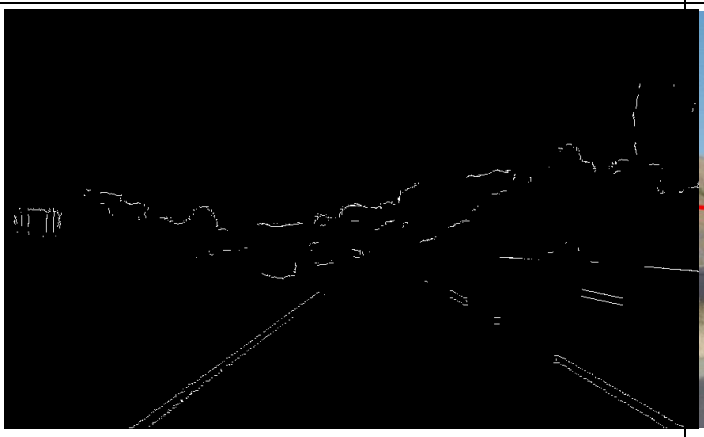
二、 結果

<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>Test image1</td><td>Gaussian Blur 1</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>Canny Edge Detection 1</td><td>Hough Transform 1</td></tr>
</table>



Test image2

Gaussian Blur 2

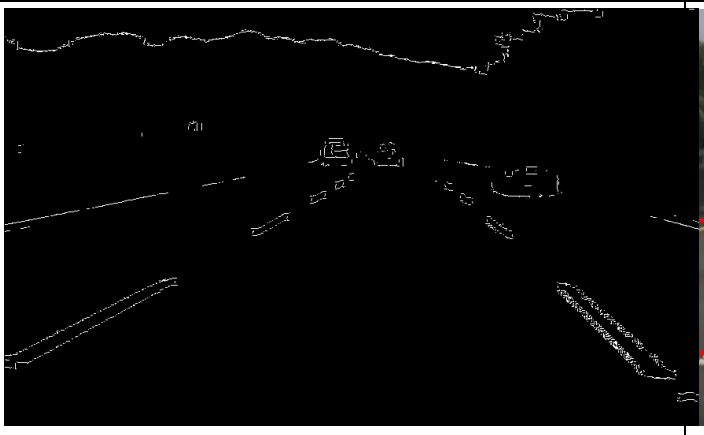


Canny Edge Detection 2

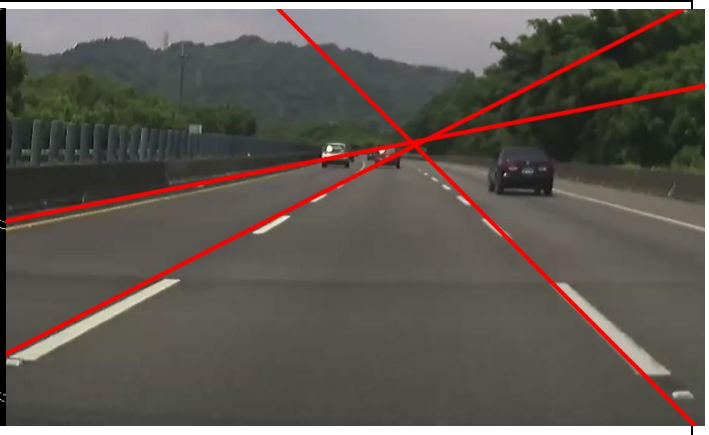
Hough Transform 2

Test image3

Gaussian Blur 3



Canny Edge Detection 3

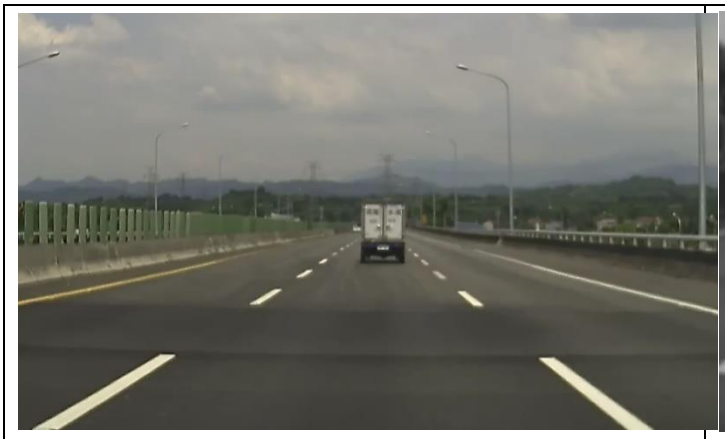


Hough Transform 3

三、

四、

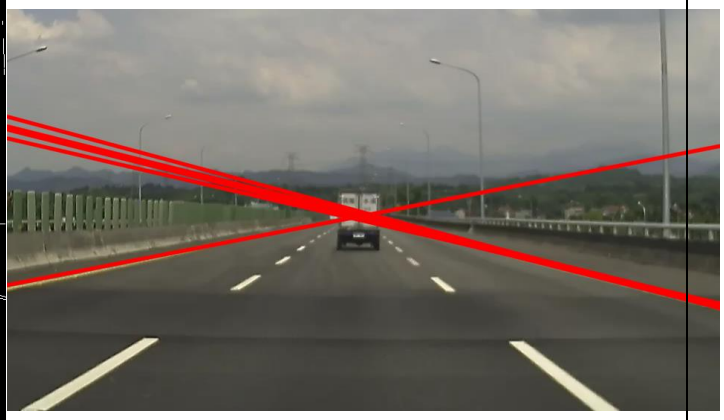
五、



Test image4

Gaussian Blur 4

Canny Edge Detection 4



Hough Transform 4