

Computer Vision HW3

111598066 資工碩一 許哲維

I 、 Explain Program and Method

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

Import four Python libraries.

```
def RGB2Gray(image):
    return np.dot(image[...,:3], [0.21, 0.72, 0.07]).astype(np.uint8)
```

Convert every RGB image's bit to [0.21, 0.72, 0.07] to get the grayscale image. (Base on luminosity method)

```
def Gaussian_Blur(image):
    x, y = np.mgrid[-1:2, -1:2]
    kernel = np.exp(-(x * x + y * y))
    kernel = kernel / kernel.sum()
    imagepad = np.zeros((int(image.shape[0]+2), int(image.shape[1]+2)))
    imagepad[1:image.shape[0]+1, 1:image.shape[1]+1] = image
    result = np.zeros((int(image.shape[0]), int(image.shape[1])))

    for y in range(0, result.shape[1]):
        for x in range(0, result.shape[0]):
            result[x, y] = (kernel * imagepad[x: x + 3, y: y + 3]).sum()

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

The Gaussian Blur operation is using a 3*3 box to take turns create the image blur.

```
def Gradient_Calculation(image):
    imagepad = np.zeros((int(image.shape[0]+2), int(image.shape[1]+2)))
    imagepad[1:image.shape[0]+1, 1:image.shape[1]+1] = image
    Gx_kernel = np.array([[ -1, 0, 1], [ -2, 0, 2], [ -1, 0, 1]])
    Gy_kernel = np.array([[ -1, -2, -1], [ 0, 0, 0], [ 1, 2, 1]])
    Gx = np.zeros((int(image.shape[0]), int(image.shape[1])))
    Gy = np.zeros((int(image.shape[0]), int(image.shape[1])))
    gradient = np.zeros((int(image.shape[0]), int(image.shape[1])))
    direction = np.zeros((int(image.shape[0]), int(image.shape[1])))

    for y in range(0, Gx.shape[1]):
```

```

        for x in range(0, Gx.shape[0]):
            Gx[x, y] = (Gx_kernel * imagepad[x: x + 3, y: y + 3]).sum()
            Gy[x, y] = (Gy_kernel * imagepad[x: x + 3, y: y + 3]).sum()
            direction[x, y] = np.arctan2(Gy[x, y], Gx[x, y])
            gradient[x, y] = abs(Gx[x, y]) + abs(Gy[x, y])

    return gradient, direction

def Nonmaximum_Suppression(image, direction):
    result = np.zeros((image.shape[0], image.shape[1]))
    angle = direction * 180.0 / np.pi

    for i in range(1, image.shape[0]-1):
        for j in range(1, image.shape[1]-1):
            try:
                x = 255
                y = 255

                if((0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180)):
                    x = image[i, j+1]
                    y = image[i, j-1]
                elif(22.5 <= angle[i,j] < 67.5):
                    x = image[i+1, j-1]
                    y = image[i-1, j+1]
                elif(67.5 <= angle[i,j] < 112.5):
                    x = image[i+1, j]
                    y = image[i-1, j]
                elif(112.5 <= angle[i,j] < 157.5):
                    x = image[i-1, j-1]
                    y = image[i+1, j+1]

                if((image[i,j] >= x) and (image[i,j] >= y)):
                    result[i,j] = image[i,j]
                else:
                    result[i,j] = 0

            except IndexError as e:
                pass

    return result

def Double_Threshold(image):
    highthreshold = image.max() * 0.1
    lowthreshold = highthreshold * 0.008

    result = np.zeros((image.shape[0], image.shape[1]), dtype=np.int32)

```

```

    strong_i, strong_j = np.where(image >= highthreshold)
    weak_i, weak_j = np.where((image <= highthreshold) & (image >=
lowthreshold))

    result[strong_i, strong_j] = np.int32(255)
    result[weak_i, weak_j] = np.int32(1)

    return result

def Edge_Tracking(image):
    for i in range(1, image.shape[0]-1):
        for j in range(1, image.shape[1]-1):
            if (image[i,j] == np.int32(1)):
                try:
                    if ((image[i+1, j-1] == np.int32(255)) or (image[i+1, j] ==
np.int32(255))
                        or (image[i+1, j+1] == np.int32(255)) or (image[i, j-1] ==
np.int32(255))
                        or (image[i, j+1] == np.int32(255)) or (image[i-1, j-1] ==
np.int32(255))
                        or (image[i-1, j] == np.int32(255)) or (image[i-1, j+1] ==
np.int32(255)))):
                        image[i, j] = np.int32(255)
                    else:
                        image[i, j] = 0

                except IndexError as e:
                    pass

    return image

def Canny_Edge_Detection(image):
    gradient, direction = Gradient_Calculation(image)
    nonmaximum = Nonmaximum_Suppression(gradient, direction)
    threshold = Double_Threshold(nonmaximum)
    result = Edge_Tracking(threshold)

    return result.astype(np.uint8)

```

The function will turn the inputted image into edge image. (Base on Canny Edge Detection)

```

def Hough_Transform(image, edge):
    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] > 135:
                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)

```

The function will turn the inputted data into Hough Transform image.

```

def main():
    path = input("Please Enter Path:> ")
    filename = input("Please Enter Filename:> ")
    if not(os.path.isfile(path+filename)):
        print("Error : Cannot Find File ! ")
        return
    image = cv2.imread(path + filename)

```





```
grayscale = RGB2Gray(image) #轉灰階
gaussian = Gaussian_Blur(grayscale) #高斯模糊
print('Write File:> Gaussian_Blur')
cv2.imwrite(path + 'result_img1.png', gaussian)
canny = Canny_Edge_Detection(gaussian) #Canny Edge Detection
print('Write File:> Canny_Edge_Detection')
cv2.imwrite(path + 'result_img2.png', canny)
hough = Hough_Transform(image.copy(), canny) #Hough Transform
print('Write File:> Hough_Transform')
cv2.imwrite(path + 'result_img3.png', hough)
```

```
main()
```




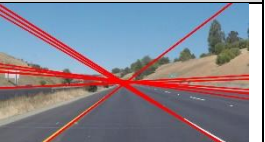
First, the program will ask you to enter the path and then the file name. If the file cannot be found, the program will jump out and end. Otherwise, it will perform grayscale conversion, Gaussian blur, Canny edge detection, Hough transform, write all the files to the path.

II 、 Result Images

1. 1.jpg

Origin	Gaussian blur	Canny edge detection	Hough transform
			

2. 2.jpg

Origin	Gaussian blur	Canny edge detection	Hough transform
			

3. 3.jpg

Origin	Gaussian blur	Canny edge detection	Hough transform
			

4. 4.jpg

Origin	Gaussian blur	Canny edge detection	Hough transform
			

※ All the original outputs can be seen from the attached file.