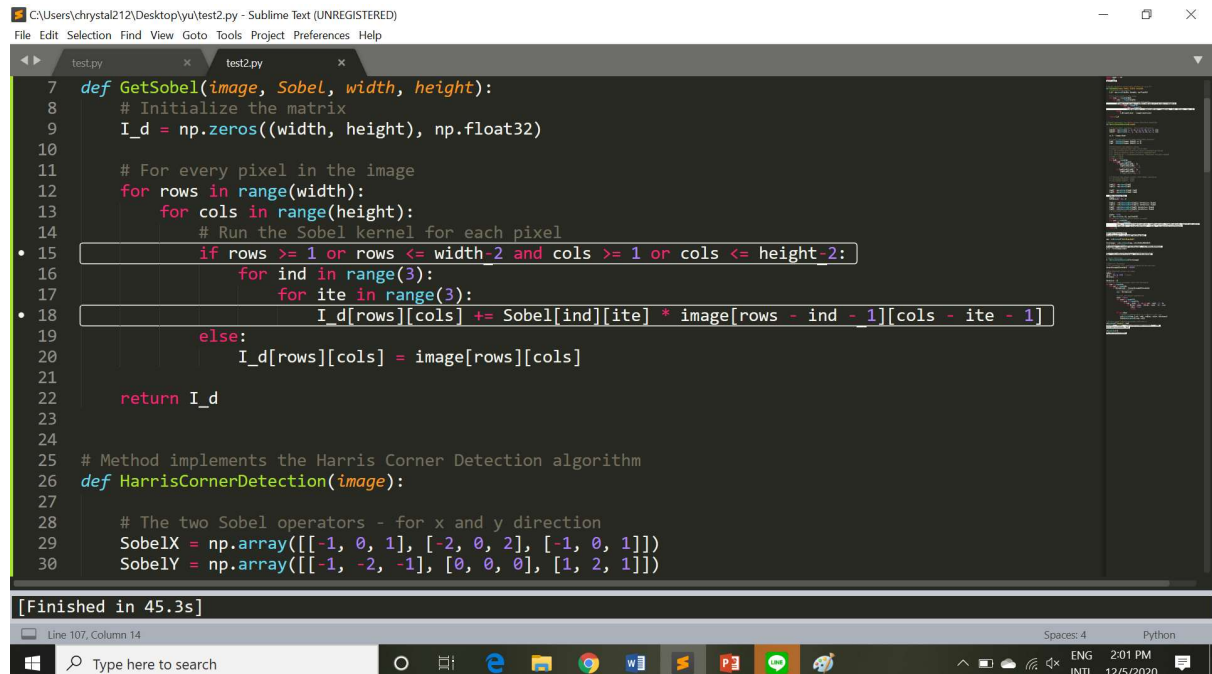


系級:資科所碩一 學號:108753208 姓名:葉冠宏

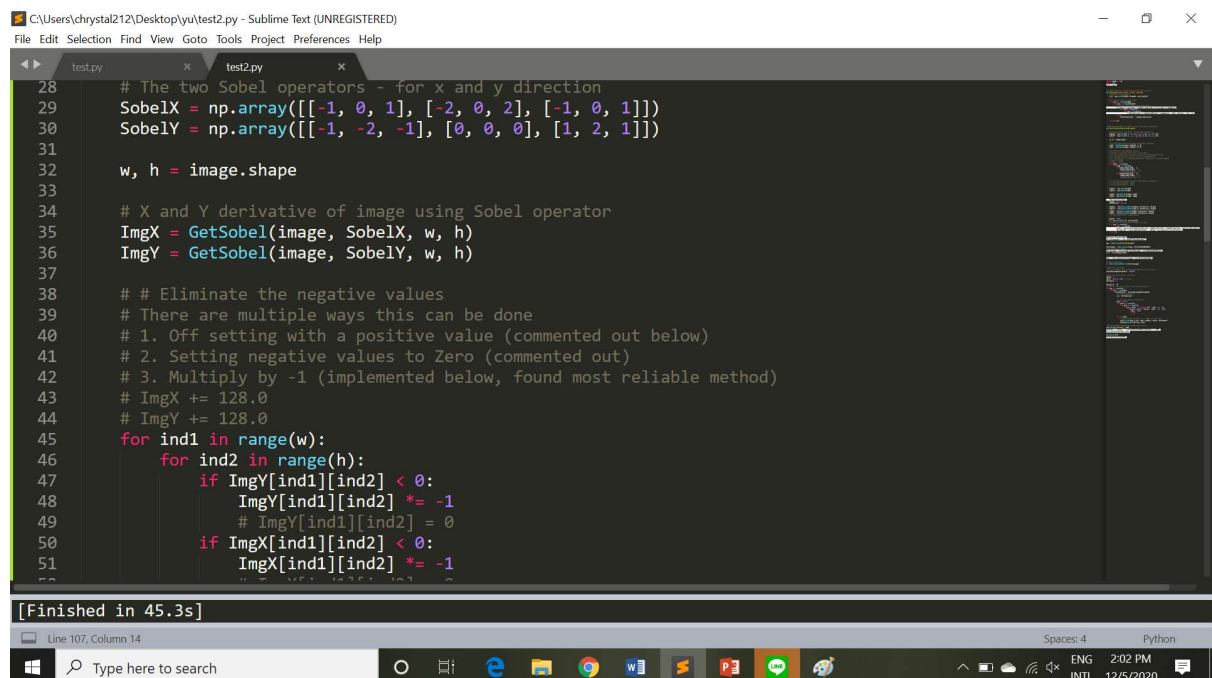
Implement the Harris corner detector and experiment with at least one of the following settings:

1. Different ways of computing the derivative I_x , I_y

法一: Approximation to a derivative of an image by Sobel operator



```
7 def GetSobel(image, Sobel, width, height):
8     # Initialize the matrix
9     I_d = np.zeros((width, height), np.float32)
10
11     # For every pixel in the image
12     for rows in range(width):
13         for cols in range(height):
14             # Run the Sobel kernel for each pixel
15             if rows >= 1 or rows <= width-2 and cols >= 1 or cols <= height-2:
16                 for ind in range(3):
17                     for ite in range(3):
18                         I_d[rows][cols] += Sobel[ind][ite] * image[rows - ind - 1][cols - ite - 1]
19             else:
20                 I_d[rows][cols] = image[rows][cols]
21
22     return I_d
23
24
25 # Method implements the Harris Corner Detection algorithm
26 def HarrisCornerDetection(image):
27
28     # The two Sobel operators - for x and y direction
29     SobelX = np.array([[ -1,  0,  1], [ -2,  0,  2], [ -1,  0,  1]])
30     SobelY = np.array([[ -1, -2, -1], [ 0,  0,  0], [ 1,  2,  1]])
```



```
28 # The two Sobel operators - for x and y direction
29 SobelX = np.array([[ -1,  0,  1], [ -2,  0,  2], [ -1,  0,  1]])
30 SobelY = np.array([[ -1, -2, -1], [ 0,  0,  0], [ 1,  2,  1]])
31
32 w, h = image.shape
33
34 # X and Y derivative of image using Sobel operator
35 ImgX = GetSobel(image, SobelX, w, h)
36 ImgY = GetSobel(image, SobelY, w, h)
37
38 # Eliminate the negative values
39 # There are multiple ways this can be done
40 # 1. Off setting with a positive value (commented out below)
41 # 2. Setting negative values to Zero (commented out)
42 # 3. Multiply by -1 (implemented below, found most reliable method)
43 # ImgX += 128.0
44 # ImgY += 128.0
45 for ind1 in range(w):
46     for ind2 in range(h):
47         if ImgY[ind1][ind2] < 0:
48             ImgY[ind1][ind2] *= -1
49             # ImgY[ind1][ind2] = 0
50         if ImgX[ind1][ind2] < 0:
51             ImgX[ind1][ind2] *= -1
```

```
C:\Users\chrystal21\Desktop\yu\test2.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

test.py x test2.py x
58 ImgX_2 = np.square(ImgX)
59 ImgY_2 = np.square(ImgY)
60
61 ImgXY = np.multiply(ImgX, ImgY)
62 ImgYX = np.multiply(ImgY, ImgX)
63
64 #Use Gaussian Blur
65 Sigma = 1.4
66 kernelsize = (3, 3)
67
68 ImgX_2 = cv2.GaussianBlur(ImgX_2, kernelsize, Sigma)
69 ImgY_2 = cv2.GaussianBlur(ImgY_2, kernelsize, Sigma)
70 ImgXY = cv2.GaussianBlur(ImgXY, kernelsize, Sigma)
71 ImgYX = cv2.GaussianBlur(ImgYX, kernelsize, Sigma)
72 # print(ImgXY.shape, ImgYX.shape)
73
74 alpha = 0.06
75 R = np.zeros((w, h), np.float32)
76 # For every pixel find the corner strength
77 for row in range(w):
78     for col in range(h):
79         M_bar = np.array([ImgX_2[row][col], ImgXY[row][col], ImgYX[row][col], ImgY_2[row][col]])
80         R[row][col] = np.linalg.det(M_bar) - (alpha * np.square(np.trace(M_bar)))
81     return R

[Finished in 45.3s]
Line 107, Column 14
Spaces: 4 Python
Type here to search
ENG INTL 2:03 PM 12/5/2020
```

在這個方法中，我們用 Sobel Operator 來去估計圖像的梯度變化。而 y 方向的 filter 和 x 方向的 filter 具有類似 transpose matrix 的關係。藉由 convolution 的相乘相加，我們可以判斷 kernel 的數值的相異程度。如果在 x 方向的 kernel 中心上下 row 變化類似，那最後得到的相加值就會是 0。如果相差頗大，那就會是正數或負數。

最後，再經過 Gaussian blur 的處理後，就可以再之後針對每個 pixel 算出其 corner strength 以便之後用來判斷是否有大於 corner strength 的 threshold，做 corner detection。



可以觀察到，從結果來看，corner 的確有被明確偵測出來，而且效果會比傳統做相鄰的 gradient 效果還要好。

法二: Compute the gradient over a small region

```
dy, dx = np.gradient(img)

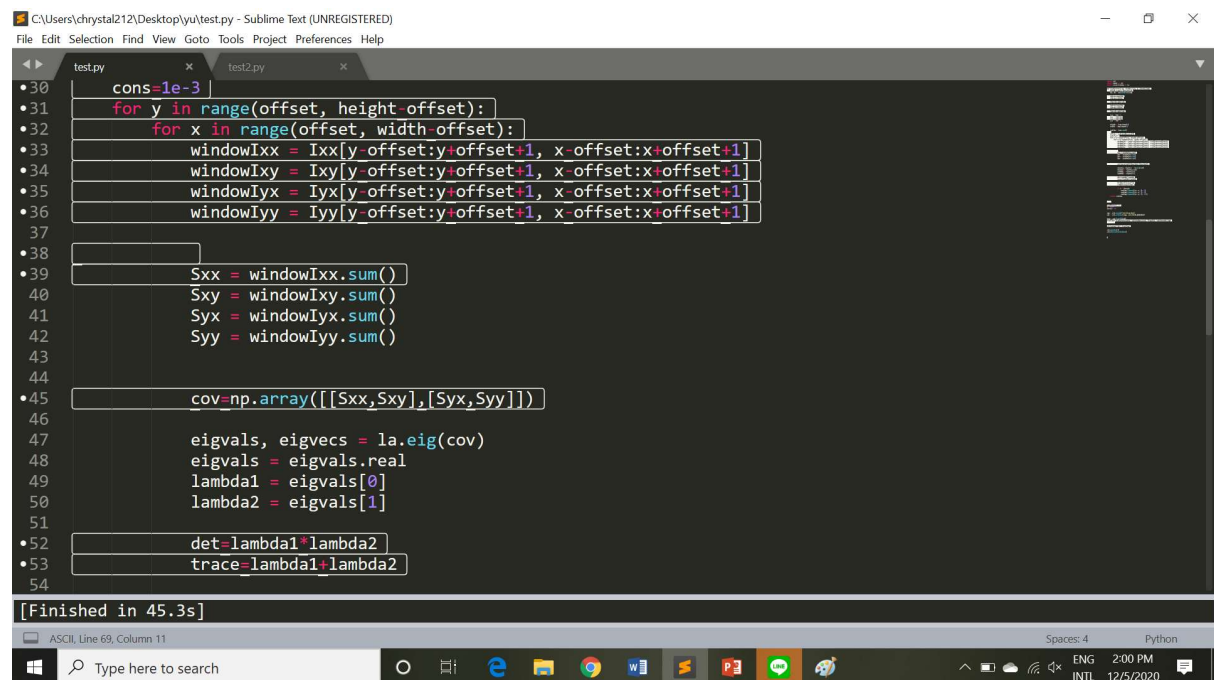
mdx=np.mean(dx)
sdx=np.std(dx)

fdx=(dx-mdx)/sdx

mdy=np.mean(dy)
sdy=np.std(dy)

fdy=(dy-mdy)/sdy

Ixx = fdx**2
Ixy = fdx*fdy
Iyx = fdy*fdx
Iyy = fdy**2
```



```
C:\Users\chrysal212\Desktop\yu\test.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

test.py x test2.py x
30 cons=1e-3
31 for y in range(offset, height-offset):
32     for x in range(offset, width-offset):
33         windowIxx = Ixx[y_offset:y_offset+1, x_offset:x_offset+1]
34         windowIxy = Ixy[y_offset:y_offset+1, x_offset:x_offset+1]
35         windowIyx = Iyx[y_offset:y_offset+1, x_offset:x_offset+1]
36         windowIyy = Iyy[y_offset:y_offset+1, x_offset:x_offset+1]
37
38
39         Sxx = windowIxx.sum()
40         Sxy = windowIxy.sum()
41         Syx = windowIyx.sum()
42         Syy = windowIyy.sum()
43
44
45         cov=np.array([[Sxx,Sxy],[Syx,Syy]])
46
47         eigvals, eigvecs = la.eig(cov)
48         eigvals = eigvals.real
49         lambda1 = eigvals[0]
50         lambda2 = eigvals[1]
51
52         det=lambda1*lambda2
53         trace=lambda1+lambda2
54
[Finished in 45.3s]
ASCII, Line 69, Column 11
Spaces: 4 Python
```

法二這邊是用傳統的相鄰 x 方向及 y 方向來做 gradient。得到這張灰階圖的矩陣後，我們針對每個 element 去做 normalize 的動作。然後分別計算 x 方向梯度矩陣的 square (Ixx)、y 方向梯度矩陣的 square (Iyy)、x 方向及 y 方向梯度矩陣的 dot (Ix*Iy)。我們用這些值組成一個 covariance matrix，然後計算其 eigen value，用這個 eigen value 計算 corner strength。



從傳統的 gradient derivative 來做 corner detection，發現其 corner detection 沒有比法一用 Sobel operator 做來的細緻。

2. Different window sizes

Sobel operator:

Window size:5



Window size:3



Window size:9



Gradient over a small region:

Window size:3



Window size:5



Window size:7



可以從上面的結果觀察到，當你的 window size 越大，你的 detection 越粗糙，因為在 window 中你可能把不是這個 corner 附近的 corner strength 也考量進來了。反之，window size 越小，你對 corner 的判斷越精確。

3. Different thresholds for f

Gradient over a small region:

Corner strength 計算方式一:

```
cov=np.array([[Sxx,Sxy],[Syx,Syy]])

eigvals, eigvecs = la.eig(cov)
eigvals = eigvals.real
lambda1 = eigvals[0]
lambda2 = eigvals[1]

det=lambda1*lambda2
trace=lambda1+lambda2

#r=det-k*(trace**2)
r=det/(trace+cons)
```



```

if r > thresh:
    newImg.itemset((y, x, 0), 0)
    newImg.itemset((y, x, 1), 0)
    newImg.itemset((y, x, 2), 255)

```

Nobel (1998)

$$R = \frac{\det(M)}{\text{trace}(M) + \epsilon}$$

Threshold:10



Threshold:2



Threshold:20



可以從上述的實驗結果觀察到當你的 threshold 越低，你越容易對 corner strength sensitive，所以你可以看到圖中有許多的紅點，對於 corner 的判斷越不精準。反之，當 threshold 越高，你對 corner strength 的標準越高，需要較高的梯度變化才會被偵測到。

Corner strength 計算方式二:

Harris & Stephens (1988)

$$R = \det(M) - \kappa \text{trace}^2(M)$$

```
r=det-k*(trace**2)  
#r=det/(trace+cons)
```

Threshold:300

K=0.04



Threshold:300

K=0.1



Threshold:300

K=0.15



這邊是另一種計算 corner strength 的方式。可以觀察到 k 在 $0.1 \sim 0.15$ 之間在一樣的 threshold 下對 corner detection 上有顯著的影響。而當 k 值越大，你的 corner strength 會越小，大於一樣 threshold 值的機率越小，所以你的 corner detection points 越少。

Reference:

<https://medium.com/datadriveninvestor/understanding-edge-detection-sobel-operator-2aada303b900>

https://github.com/ShivamChourey/Harris-Corner-Detection/blob/master/Corner_Detection.py

http://www.cs.cmu.edu/~16385/s17/Slides/6.2_Harris_Corner_Detector.pdf

<https://github.com/hughesj919/HarrisCorner/blob/master/Corners.py>