CV HW1 Report

-111065524 資應二 李懷

Question 1. Histogram equalization

1. Implementation:

a) implement histogram equalization:

Step 1.

First, we form a pixel-intensity frequency histogram by scanning through each pixel of the input image(grayscale) array and its levels (bit size, input image dtype is np.uint8, means 8-bit unsigned integer 0 to 255)

```
    levels = 2**(image.dtype.itemsize*8)
    # get the histogram distribution of pixel value:
    histo_array = form_histogram(image, levels)
```

```
def form_histogram(image, levels):
    # create an empty array of length G(gray-levels, 256 here) and form histogram of
    # the distribution of pixel value of the image
    histo_array = [0]*levels

    # scan through every pixel and count the frequency for each intensity gp
    for row in image:
        for gp in row:
            histo_array[gp] += 1
    return histo_array
```

Step 2.

計算 cumulative histogram， $\mathrm{Hc}[p] = \mathrm{Hc}[p-1] + \mathrm{H}[p]\ (p = 1,2,\ldots,255)$，也就是 0~255 每個 intensity 的累積 frequency(255 就代表 255 前所有 intensity 加總起來的 frequency)，並將其存入變數 histo_c_array:

```
    # form cumulative histogram:
    histo_c_array = histo_array.copy()
    for i, freq in enumerate(histo_c_array):
        if i != 0:
            histo_c_array[i] = freq + histo_c_array[i - 1]
```

Step 3.

接著，定義公式 $T[p] = round((G - 1)/N * M) * Hc[p])$ (refer to Algorithm 5.1 on textbook)，i 代表 N*M 就是我們的 image_size:

```python
# T[p]
def t_function(i: int, histo_c_array):
    return round(((levels-1)/image_size)*histo_c_array[i])
```

scan through each pixel 並套用 $T[p]$在當下 pixel 的 $gp(intensity)$上，that is $gq = T[gp]$，並將其存到 output_image 的 array 中 ：

```python
# histgoram equalization, rescan image every pixel intensity and apply T[gp]
output_image = image.copy()
for row in output_image:
    for i, gp in enumerate(row):
        row[i] = t_function(gp,histo_c_array)
```

We then test out our function:

We use cv2.imdecode instead of cv2.imread because the img_path contains Chinese characters (you can comment out imdecode and remove comment from imread if your img_path is all English)

```python
if __name__=='__main__':

    # read image img subfolder
    img_path = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), "img", "hw1-1.jpg")
    save_path = os.path.join(os.path.dirname(img_path)) # images save location

    # read image as grayscale
    #image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    # use this when path contains chinese char
    image = cv2.imdecode(np.fromfile(img_path, dtype=np.uint8), cv2.IMREAD_GRAYSCALE)

    # do histogram equalization here
    output_image = histo_equalize(image)
```

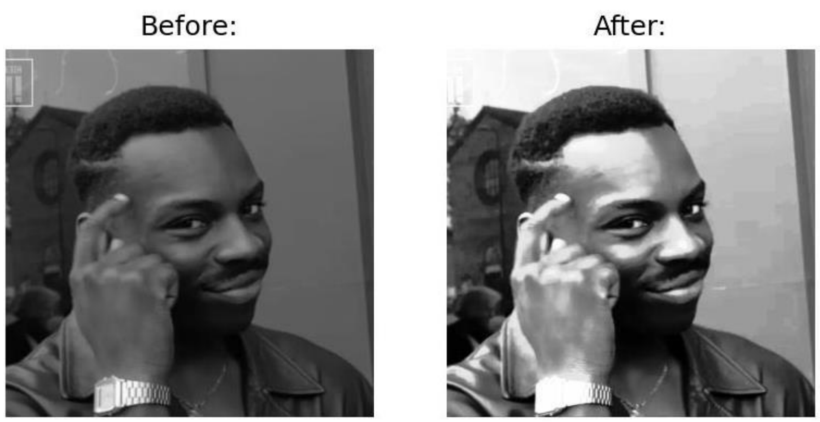
*Result Images: See below (b) and (c)

b) image before and after:

Code for drawing pictures side by side:

```python
    # plot 2 images side by side in one image
    fig = plt.figure('(b)',dpi=150)
    axs = fig.subplots(1,2)
    axs[0].set_title('Before:')
    # # since default matplotlib RGB, we have to specify it as grayscale,
    # if we want to show colored opencv image, convert to cv2.COLOR_BGR2RGB
    axs[0].imshow(image, cmap='gray')
    axs[0].axis('off')
    axs[1].set_title('After:')
    axs[1].imshow(output_image, cmap='gray')
    axs[1].axis('off')

    plt.suptitle('(b)before and after histogram equalization', fontsize=16, y=0.95)
    plt.savefig(os.path.join(save_path, "1-b.jpg"))
    plt.draw()
    print("Press Enter to close the window:")
    plt.waitforbuttonpress(0)
    plt.close()
```

Result Image:



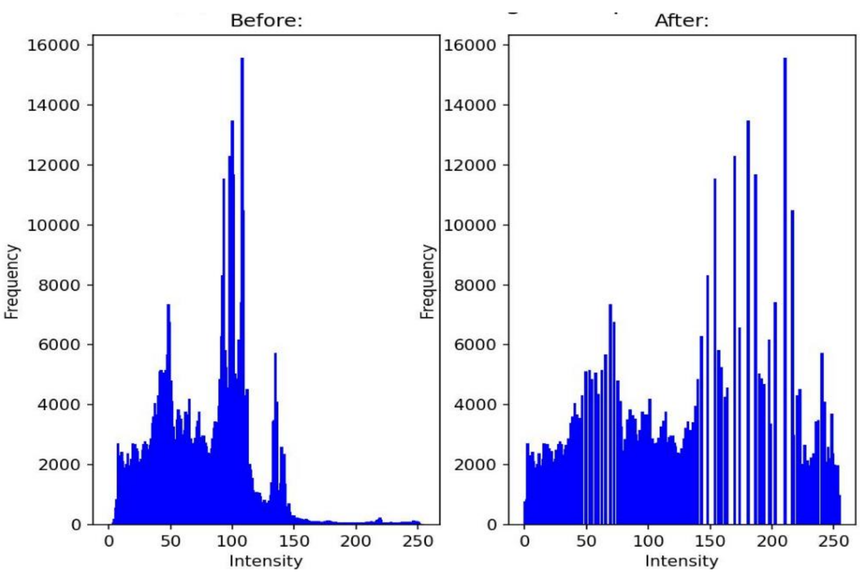
c) distribution of pixel value before and after:

Code for drawing pictures side by side:

也將 output_image 轉為 histogram array，然後再將原本的 histo_array 及 equalized 的 histo_array 分別畫在一張圖的左右

```
equalized_histo_array = form_histogram(output_image, levels)

# plot 2 histograms side by side in one image
fig = plt.figure('(c)', dpi=150,figsize=(8, 6))
axs = fig.subplots(1,2)
axs[0].set_title('Before:')
axs[0].hist(range(len(histo_array)), bins=len(histo_array), weights=histo_array, edgecolor='blue')
axs[0].set_xlabel('Intensity')
axs[0].set_ylabel('Frequency')

axs[1].set_title('After:')
axs[1].hist(range(len(equalized_histo_array)), bins=len(equalized_histo_array), weights=equalized_histo_array, edgecolor='blue')
axs[1].set_xlabel('Intensity')
axs[1].set_ylabel('Frequency')

plt.suptitle('(c)before and after histogram equalization', fontsize=16, y=0.95)

axs[0].yaxis.set_label_position("left")
plt.savefig(os.path.join(save_path, "1-c.jpg"))
plt.draw()
print("Press Enter to close the window:")
plt.waitforbuttonpress(0)
plt.close()
```

Result Image:



2. Discuss section:

-No discussion for this question.

Question 2. Harris corner detector

1. Implementation:

a) (i.) Grayscale and Gaussian Smooth (1 image):

定義 gaussian smoothing 的 function(輸入 image array, sigma 及 kernel size)

Step 1.

先用 cv2.getGaussianKernel 產出 1D 的 gaussian

Step 2.

再將 1D gaussian filter 外積產出 gaussian 2D filter (gaussian function is seperable)

Step 3.

再使用 cv2.filter2D 對 image 做 gaussian 的 convolution:

```
# (a)(i)Grayscale and Gaussian Smooth(1 image)
def gaussian_smooth(image, sigma=1, kernel_size=3):

    gaussian_1D_filter = cv2.getGaussianKernel(kernel_size,sigma)
    gaussian_2D_filter = np.outer(gaussian_1D_filter,gaussian_1D_filter)
    output_image = cv2.filter2D(image, -1, kernel=gaussian_2D_filter)

    return output_image
```

*Result Images: See below (b)

a) (ii.) Intensity Gradient (Sobel operator)(2 image):

定義 sobel operation 的 function(輸入 image array 及 sobel operator 的 size)

Step 1.

分別產出 3x3 的 x 與 y 的 sobel operators matrix，sobel_y 為 sobel_x transpose 後 row1 跟 3 交換。

Step 2.

將 image 轉為 float32，若為 uint8 的話範圍為 0~255，無法儲存負數，再分別做 convolution 並回傳。

```python
def sobel_operator(image):
    sobel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]], dtype=np.float32)
    sobel_y = sobel_x.T
    sobel_y[[0, 2]] = sobel_y[[2, 0]]

    image = image.astype(np.float32)
    gradient_x = cv2.filter2D(image, -1, kernel=sobel_x)
    gradient_y = cv2.filter2D(image, -1, kernel=sobel_y)

    return gradient_x, gradient_y
```

*Result Images: See below (b)

a) (iii.) Structure Tensor(1 image):

定義計算 structure tensor 及 response 的 function (輸入 image array, window_size, threshold 及 k(此僅$\det H - k(\text{trace}H)$^2計算 response 時會用到，Harris operator 的話不會用到))。

Step 1.

定義所需參數，response array 用來儲存結果，dx 及 dy 分別為 gradient in x 及 y direction，也就是Ix及Iy，dxy 代表$IxIy$。

Step 2.

計算每個 pixel 為中心的 3x3 window 的 structure tensor 合 (h_tensor_window

= $\sum_{(x,y)\in W} \begin{matrix} Ix^2 & IyIx \\ IxIy & Iy^2 \end{matrix})$。

```python
    # (iii.)Structure Tensor(1 image)
    def structure_tensor(image, window_size=3, k = 0.04):

        epsilon = 1e-6
        response_array = np.zeros_like(image).astype(np.float32)
        # response_array = cv2.cornerHarris(image, 3, 3, 0.04)
        dx, dy = sobel_operator(image)
        dx2 = dx*dx
        dy2 = dy*dy
        dxy = dx*dy
        # dxy == dyx
        print("calculating structure tensor and response...")
        offset = int(window_size/2)
        for y in range(offset, response_array.shape[0]-offset):
            for x in range(offset, response_array.shape[1]-offset):

                # sum of structure tensor within window
                Ix2 = np.sum(dx2[y-offset:y+1+offset, x-offset:x+1+offset])
                Iy2 = np.sum(dy2[y-offset:y+1+offset, x-offset:x+1+offset])
                IxIy = np.sum(dxy[y-offset:y+1+offset, x-offset:x+1+offset])
                h_tensor_window = np.array([[Ix2, IxIy], [IxIy, Iy2]])
```

※ 此為另一種計算 structure tensor 的方法(可替換上面 58~61 行)

```
        # different approach
        h_tensor_window = np.zeros((2,2))
        for i in range(-offset, offset+1):
            for j in range(-offset, offset+1):
                Ix, Iy = (dx[y+j, x+i], dy[y+j, x+i])
                a = np.array([[Ix], [Iy]])

                h_tensor = np.dot(a, a.T)
                h_tensor_window += h_tensor
```

Step 3.

再對每個 window 用 Harris operator 算出 response (由於 trace(H)可能為 0，加上ε)，計算完整張圖的 response (75 行為另一種計算 response 的方式，算出來的結果相較於 harris operator 來得少點，須另行調參數)。

```
74              # two ways of calculating Harris response using local structure tensor:
75              # response = np.linalg.det(h_tensor_window)-k*(h_tensor_window.trace()**2)
76              # Harris operator:
77              response = np.linalg.det(h_tensor_window)/(h_tensor_window.trace()+epsilon)
78              response_array[y, x] = response
79
80      return response_array
```

再定義一個 thresholding function (輸入 response 的 array 及 threshold)，整請見下方:

```
# do thresholding on corner response
def response_thresholding(response_array, threshold):
    response_array = np.where((response_array > threshold*response_array.max()), 255, 0)
    return response_array
```

*Result Images: See below (b)

a) (iv.) Non-maximal Suppression(1 image):

定義一個 NMS function，輸入為 response 的 array 及 nms 的 window_size，回傳 nms 後的 response array。

Step 1.

以 5x5 的 window 掃過 response array 每個 pixel。

Step 2.

並對 window 內的 response 找出 local maxima，response 小於此 local maxima 的 pixel 壓制為 0。

```python
def nms(response_array, nms_window_size = 3):
    offset = int(nms_window_size/2)
    for y in range(offset, response_array.shape[0]-offset):
        for x in range(offset, response_array.shape[1]-offset):
            # Find local maxima in a window
            window = response_array[y-offset:y+1+offset, x-offset:x+1+offset]
            local_max = np.max(window)
            for row in window:
                for i,each in enumerate(row):
                    if each<local_max:
                        row[i] = 0
    return response_array
```

*Result Images: See below (b)

a) Integration:

Step 1.

最後整合(i.)~(iv.)各 function 組成 harris corner detector function，輸入為 image array, gaussian sigma, gaussian kernel_size, harris operator 的 window_size 及 threshold，回傳為各 function 產出的 image。

```python
# (a)
def harris_corner_detector(image, sigma=3, kernel_size=3, window_size=3, threshold=0.1):
    # (i.)
    gaussian_blurred_image = gaussian_smooth(image, sigma, kernel_size)
    # gaussian_smooth_test(image, gaussian_blurred_image, 3, (3,3))
    gradient_image_x, gradient_image_y = sobel_operator(gaussian_blurred_image)
    response_array = structure_tensor(image, window_size)
    harris_response_image = response_thresholding(response_array, threshold)

    response_array_nms = nms(response_array, nms_window_size=5)
    harris_response_nms_image = response_thresholding(response_array_nms, threshold)

    return gaussian_blurred_image, gradient_image_x, gradient_image_y, harris_response_image, harris_response_nms_image
```

Step 2.

用(a)指定的參數來執行測試我們的 function。

```python
if __name__=='__main__':
    # read image img subfolder
    img_path = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), "img", "hw1-2.jpg")
    save_path = os.path.dirname(img_path) # images save location

    # read image as grayscale (if usage of cv2.IMREAD_GRAYSCALE is not allowed, use NTSC formula to convert RG
    #image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    # use this when path contains chinese char
    image = cv2.imdecode(np.fromfile(img_path, dtype=np.uint8), cv2.IMREAD_GRAYSCALE)

    # (a) implement Harris corner detector
    output_images = harris_corner_detector(image, sigma=3, kernel_size=3, window_size=3, threshold=0.1)
```

*Result Images: See below (b)

b) (i.) images processed after each step:

定義一個畫出及儲存(a)(i.)~(iii.)圖片的 function ，輸入為 image array 的 tuple (由 harris_corner_detector 回傳) 及儲存路徑。

```python
def draw_and_save_steps(*images, save_path):

    # save images to folder:
    plt.imsave(os.path.join(save_path, '2-(a)(i.)_gaussian_blur.jpg'), images[0], cmap='gray')
    plt.imsave(os.path.join(save_path, '2-(a)(ii.)_gradient_x.jpg'), images[1], cmap='gray')
    plt.imsave(os.path.join(save_path, '2-(a)(ii.)_gradient_y.jpg'), images[2], cmap='gray')
    plt.imsave(os.path.join(save_path, '2-(a)(iii.)_corner_response.jpg'), images[3], cmap='gray')
    plt.imsave(os.path.join(save_path, '2-(a)(iv.)_corner_response_nms.jpg'), images[-1], cmap='gray')
```

```python
    # (b)(i.) Show images after each step in (a.)
    draw_and_save_steps(*output_images, save_path=save_path)
```

Step 1.

Code for drawing (a)(i.)Gaussian Blurred image(1 image) :

```python
    plt.figure('(a)(i.) Gaussian Smooth')
    plt.imshow(images[0], cmap='gray')
    plt.suptitle('(a)(i.) Gaussian blur:', fontsize=16, y=0.95)
    plt.draw()
    print("Press Enter to close the window:")
    plt.waitforbuttonpress(0)
    plt.close()
```

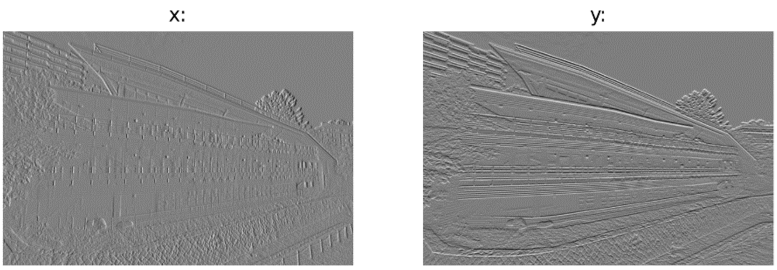
Result Image:

Step 2.

Code for drawing (a)(ii.) gradient intensity map in x and y direction(2 images) :

```
fig = plt.figure('(a)(ii.) Intensity Gradient (Sobel operator)', dpi=150)
axs = fig.subplots(1,2)
axs[0].set_title('x:')
axs[0].imshow(images[1], cmap='gray')
axs[0].set_axis_off()

axs[1].set_title('y:')
axs[1].imshow(images[2], cmap='gray')
axs[1].set_axis_off()
plt.suptitle('(a)(ii.) Show gradient intensity map of x and y direction:', fontsize=16, y=0.95)
plt.draw()
print("Press Enter to close the window:")
plt.waitforbuttonpress(0)
plt.close()
```

Result Image:



Step 3.

Code for drawing (a)(iii.) Harris response (1 image):

```
plt.figure('(a)(iii.) Harris Response', figsize=(8,6))
plt.imshow(images[3], cmap='gray')
plt.suptitle('(a)(iii.) Show the corner response R after thresholding the response:', fontsize=16, y=0.95)
plt.draw()
print("Press Enter to close the window:")
plt.waitforbuttonpress(0)
plt.close()
```

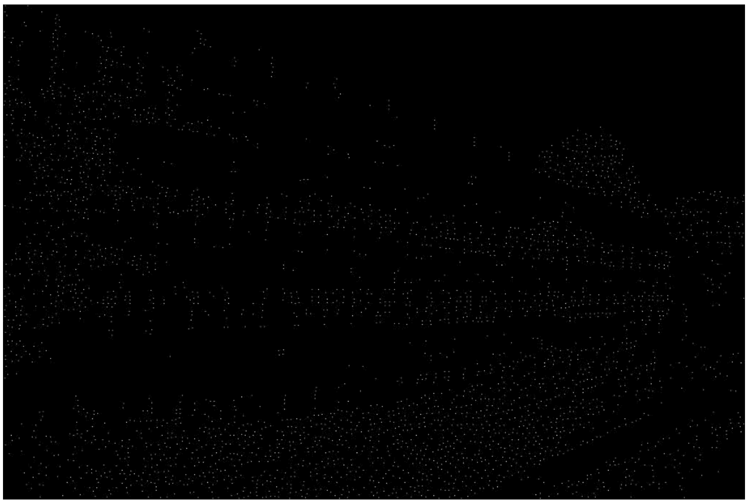
Result Image: 

Step 4.

Code for drawing (a)(iv.) Harris response after NMS(1 image):

```
    plt.figure('(a)(iv.) Harris Response NMS', figsize=(8,6))
    plt.imshow(images[-1], cmap='gray')
    plt.suptitle('(a)(iv.) Harris Response NMS:', fontsize=16, y=0.95)
    plt.draw()
    print("Press Enter to close the window:")
    plt.waitforbuttonpress(0)
    plt.close()
```

Result Image:



b) (ii.) original image(grayscale) with corner points overlaid (1 image):

Code for drawing Corner points overlaid (1 image):

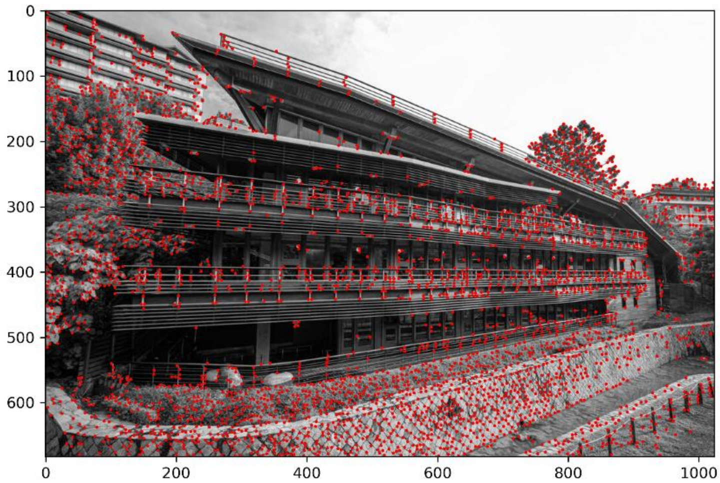
Use plt.scatter to do the point overlaying:

```
    # (b)(ii.) Shows the original image(grayscale) with corner points overlaid.
    draw_and_save_overlaid(image, output_images[-1], window_size=3, threshold=0.1, save_path=os.pat
```

```
def draw_and_save_overlaid(image, overlay, window_size, threshold, save_path, title = "(b)(ii.)"):

    plt.figure(f'{title} Corner Points Overlaid', figsize=(8,6))
    plt.imshow(image, cmap='gray')
    overlay = np.column_stack(np.where(overlay > 0))
    # Create a scatter plot to draw the points on the image
    plt.scatter(overlay[:, 1], overlay[:, 0], s=1, c='red')
    plt.suptitle(f'{title} Shows the original image(grayscale) with corner points overlaid.', fontsize=16, y=0.95)
    if window_size and threshold:
        plt.figtext(0.5, 0.02, f"window_size={window_size}, threshold={threshold}", ha="center", fontsize=12)
    plt.savefig(save_path, dpi=300)
    plt.draw()
    print("Press Enter to close the window:")
    plt.waitforbuttonpress(0)
    plt.close()
```

Result Image:

(b)(ii.) Shows the original image(grayscale) with corner points overlaid.



window_size=3, threshold=0.1

c) (i.) Try a different window size in computing the structure tensor H of each pixel (1 image) :

Code for drawing response array(after threshold) for c(i) and c(ii):

```
def draw_response_array(response_array, window_size, threshold, save_path, question):

    if question == 1:
        plt.figure('(c)(i.) Harris Response', figsize=(8,6))
        plt.suptitle('(c)(i.)Try a different window size in computing the structure tenso
    else:
        plt.figure('(c)(ii.) Harris Response', figsize=(8,6))
        plt.suptitle('(c)(ii.)Try a different threshold in thresholding corner response.'

    if window_size and threshold:
        plt.figtext(0.5, 0.02, f"window_size={window_size}, threshold={threshold}", ha="c
    plt.imshow(response_array, cmap='gray')
    plt.savefig(save_path, dpi=300)
    plt.draw()
    print("Press Enter to close the window:")
    plt.waitforbuttonpress(0)
    plt.close()
```

Step 1.

In __main__，執行 harris_corner_detector，將 window_size 設為 7，儲存 output 的 threshold response array。

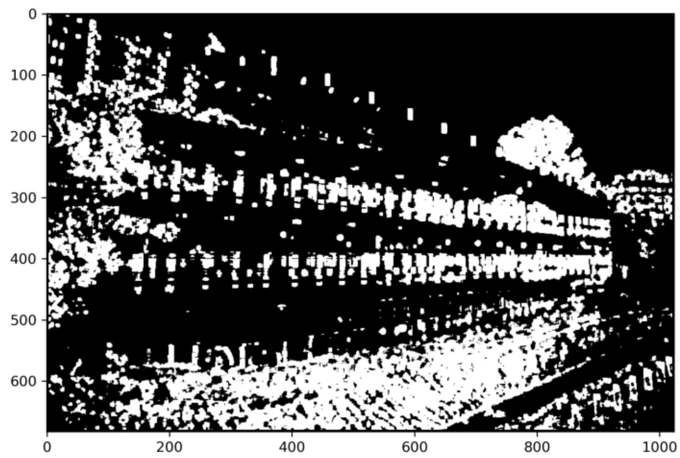
Step 2.

再將 response array after thresholding 及 corner points overlay 圖畫出來:

```
# (c)(i.) Try a different window size in computing the structure tensor H of each pixel.
print("(c)(i.)")
window_size = 7
output_images = harris_corner_detector(image, sigma=3, kernel_size=3, window_size=window_size, threshold=threshold)
draw_response_array(output_images[-2], window_size=7, threshold=0.1, save_path=os.path.join(save_path, "2-(c)(i.)_corner_respo
draw_and_save_overlaid(image, output_images[-1], window_size=7, threshold=0.1, save_path=os.path.join(save_path, "2-(c)(i.)_co
```

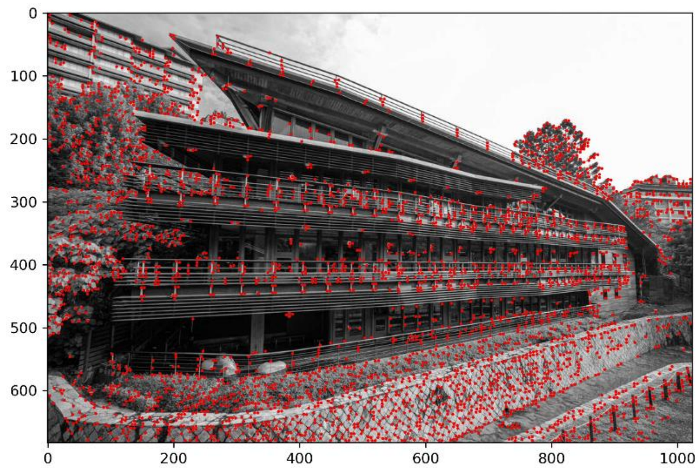
Result Image:

Try a different window size in computing the structure tensor ▯ of each



window_size=7, threshold=0.1

(c)(i.) Shows the original image(grayscale) with corner points overlaid.



window_size=7, threshold=0.1

c) (ii.) Try a different threshold in thresholding corner response (1 image):

Step 1.

執行 harris_corner_detector，將 threshold 提高到 0.5(window_size 承接上題為 7)。

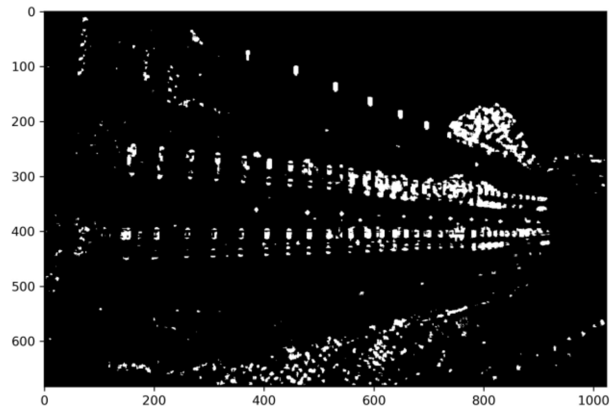
Step 2.

再將 response array after thresholding 及 corner points overlay 圖畫出來:

```
# (c)(ii.) Try a different threshold in thresholding corner response.
print("(c)(ii.)")
threshold = 0.3
output_images = harris_corner_detector(image, sigma=3, kernel_size=3, window_size=window_size, threshold=threshold)
draw_response_array(output_images[-2], window_size, threshold, save_path=os.path.join(save_path, "2-(c)(ii.)_corner_response.
draw_and_save_overlaid(image, output_images[-1], window_size, threshold, save_path=os.path.join(save_path, "2-(c)(ii.)_corner_
```

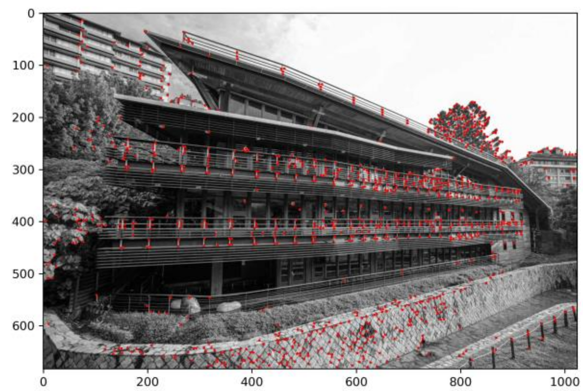
Result Image:

(c)(ii.)Try a different threshold in thresholding corner response.



window_size=7, threshold=0.3

(c)(ii.) Shows the original image(grayscale) with corner points overlaid.



window_size=7, threshold=0.3

2. Discuss section:

d) Discuss the result of (b.) and (c.):

(i.) How does window size affect the result?

With larger window size, here we increase window_size from 3 to 7, we did discover that before NMS, the harris response array pixels value passed the thresholding are much more, as for after NMS, the number of points are not noticeably different.

However, more points passing threshold is not good if we want to have more refined result, but this does not mean larger window size is bad, it is because we did not retune our threshold for new window size, after c(ii) where we increase the threshold, the result is much more refined(see c(ii) image).

This is because the window size has to be large enough to have enough gradient information to calculate and thus properly find the best corner location.

When we increase window size when computing structure tensors, it should:

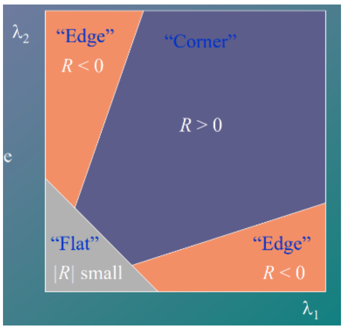
- Be more noise-tolerant as it allows more neighbor pixels to be taken into account and more smoothing average can be calculated.
- Allow larger-scale corners to be detected

However, larger window slightly increases the computation time as it scans through more pixels in a window.

(ii.) How does threshold affect the result?

If we increase the threshold, the corner points detected would be less, since obviously less points would have passed the threshold, only keeping the points with large response, we can better visualize it by using the below image:

Increase the threshold would reduce the region of valid points, pushing the valid corner point region closer to the top-right corner.



Question 3. SIFT object recognition

1. Implementation:

定義一個 sift function，輸入為兩個 image arrays 及 k(object 數量)

a) Grayscale the image, detect keypoint and extract SIFT feature:

Step 1.

Create sift features using opencv function, and use sift.detectAndCompute to compute image's keypoints and their descriptors:

```
def sift(image1, image2, k):
    sift = cv2.SIFT_create()
    keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
    # keypoints2, descriptors2 = sift.detectAndCompute(image2, None)
```

b) Brute-force matching and ratio test(20 matches for each object):

Before matching, we want to segment keypoints of image1(scene/query image) or image2(target/train image) into groups with number of objects (k=3 in this case) since we want to find 20 matches for "each object".

Step1. Divide Keypoints into groups

In this case, since in image2, 3 objects are merged vertically, we can simply segment them by vertical height, like so:

```
    segments = np.array_split(image2, k)

    keypoints2, descriptors2 = [], []
    for i in range(k):
        start_height = i * segments[i].shape[0]
        end_height = (i+1) * segments[i].shape[0]

        black_mask = np.zeros_like(image2)
        black_mask[start_height:end_height, :] = segments[i]
```

We use np.array_split to split image2 into k=3 groups vertically, and iterate through each group, while iterating, we assign the segment array to their relative position on the blackmask of the original image2, this is because later on when we are doing detectAndCompute(), we want the keypoints to be on their original y position of image2, and we blackmask everything outside current segment so when detecting keypoints, it only focuses on the current segment.

Then we detectAndCompute keypoints and descriptors for each segement, assign their class_id(this attribute is used to identify what object the keypoint belongs to), then we extend and append the current segment's keypoints and descriptors to keypoints2 and descriptors2 varaibles that store all keypoints/descriptors of the entire image2.

```python
        black_mask[start_height:end_height, :] = segments[i]

        keypoints_segment, descriptor_segment = sift.detectAndCompute(black_mask, None)
        # assign class_id so annotate the group
        for each in keypoints_segment:
            each.class_id = i

        # combine all segment's keypoints and desciprtors
        keypoints2.extend(keypoints_segment)
        descriptors2.append(descriptor_segment)

    keypoints2, descriptors2 = tuple(keypoints2), np.vstack(descriptors2)
```

Another approach for Step1(Optional):

We also tried sklearn kmeans to cluster object's keypoints as the other approch, because it is possible that there are times we don't want to segment the second image (although it is much easier and accurate to segment) and only want to segment the first, this is because we sometimes treat the first image as a query/scene image (which means we use it as input and match it with target image(2nd image) that is maybe not an input but stored in the system), of course this is not the best solution and we can use other more advanced instance segmentation methods, but since this is not the main focus of this problem? we decided to just tried kmeans:

Code for Kmeans clustering, this assigns cluster labels to each keypoint's class_id(default is-1) which tells us what group each keypoint belong to :

```
def clusterKeypoints(keypoints, k):
    points = [keypoint.pt for keypoint in keypoints]
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(points)
    cluster_ids = kmeans.labels_

    for keypoint, cluster_id in zip(keypoints, cluster_ids):
        keypoint.class_id = int(cluster_id)
```

```
    clusterKeypoints(keypoints1, k)
```

Step 2. Do brute-force matching

Before we start matching, we can rank keypoints along with its descriptor by keypoint's response, this will improve the matching result if we only do matches on stronger keypoints:

```
    # sort by keypoints and descriptors of image1 by keypoint response
    keypoints1, descriptors1 = zip(*sorted(zip(keypoints1, descriptors1), key=lambda pair: pair[0].response, reverse=True))
```

We loop through descriptors1, and nested for loop each descriptor of descriptors2, we then use np.linalg.norm to compute the Euclidean distance(which is L2 norm, default of np.linalg.norm) of two descriptors(each is 128D vector), and add the index (j, the index of descriptors2 to distances list), the complexity of this matching is O(descriptors1.shape[0], descriptors1.shape[0]):

```
    print("matching...")
    ratio = 0.7
    matches = []
    #  for each 128D vector in desc1, find a match in desc2
    for i, desc1 in enumerate(descriptors1):
        # i used as queryIdx(first image descriptor vector index)
        distances = []
        for j, desc2 in enumerate(descriptors2):
            # j used as trainIdx(second image descriptor vector index)
            # Calculate distance between desc1 and desc2, L2 norm
            distance = np.linalg.norm(desc1 - desc2)
            distances.append((j, distance))
        distances.sort(key = lambda x:x[1]) # sort by distance
```

Step 3. Ratio Test

We then sort the distances list by the distance in ascending order, the then find top2 best matches, and do the ratio test(ratio set to 0.7) on the best match, this will filter out those bad quality matches, then we append the best_match index (best_match[0], this is the best match index of descriptor in descriptors2) along with current i to the matches list,

```
        distances.sort(key = lambda x:x[1]) # sort by distance
        best_match, second_best_match = distances[0], distances[1]

        # pass the ratio test to get matched
        if best_match[1] < ratio * second_best_match[1]:
            matches.append((i, best_match[0])) # image index pair
```

Step 4.

After existing the loop, we then finally convert each (i, j) in matches to openCV DMatch object, this will allow us to us them to drawMatches on the image:

```
    matches = [(cv2.DMatch(i, j, 0)) for i,j in matches] # convert to DMatch object, imgIdx = 0
```

We then retrieve the top-20 best matches from each object(filter by each class_id of keypoints2 we assigned when doing segmentation), then finally draw them on the image:

```
    # find top 20 matches for each object
    matches_obj = []
    for object in range(k):
        matches_obj.extend([match for match in matches if keypoints2[match.trainIdx].class_id == object][:20])

    # matched_image = cv2.drawMatchesKnn(image1, keypoints1, image2, keypoints2, matches[:60], None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGL
    matched_image = cv2.drawMatches(image1, keypoints1, image2, keypoints2, matches_obj, None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POIN
```

We then test out our function:

```
if __name__=='__main__':
    # read image img subfolder
    img_path = os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), "img")
    # read image as grayscale (if usage of cv2.IMREAD_GRAYSCALE is not allowed, use NTSC formula to convert RGB int
    #image1 = cv2.imread(os.path.join(img_path,"hw1-3-1.jpg"), cv2.IMREAD_GRAYSCALE)
    #image1 = cv2.imread(os.path.join(img_path,"hw1-3-2.jpg"), cv2.IMREAD_GRAYSCALE)
    # use this when path contains chinese char
    image1 = cv2.imdecode(np.fromfile(os.path.join(img_path,"hw1-3-1.jpg"), dtype=np.uint8), cv2.IMREAD_GRAYSCALE)
    image2 = cv2.imdecode(np.fromfile(os.path.join(img_path,"hw1-3-2.jpg"), dtype=np.uint8), cv2.IMREAD_GRAYSCALE)

    matched_image = sift(image1, image2, 3)
    draw_and_save(matched_image, os.path.join(img_path,"sift_match.jpg"))
```

c) Plot matching result on the images (1 image):

Code for drawing matching result:

```
def draw_and_save(matched_image, save_path):
    # Plot matching result on the images
    plt.figure('(c) sift')
    plt.imshow(matched_image, cmap='gray')
    plt.imsave(save_path, matched_image, cmap='gray')
    plt.draw()
    print("Press Enter to close the window:")
    plt.waitforbuttonpress(0)
    plt.close()
```

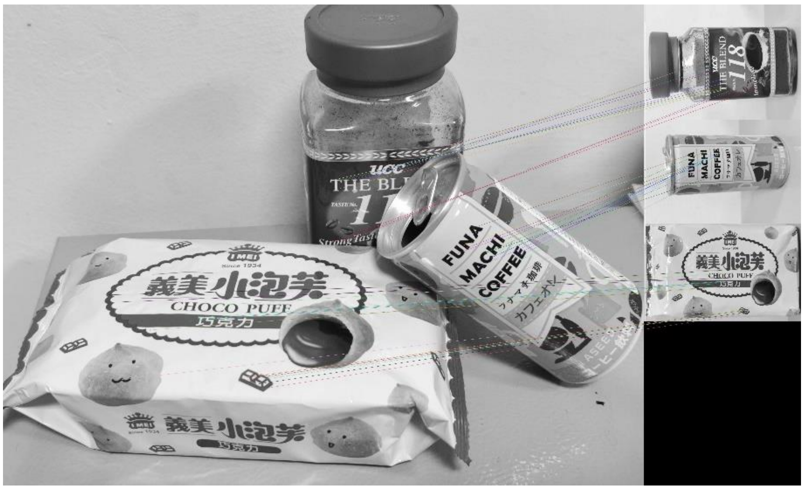
Result image:

d) Scale the scene image to 2.0x and redo (a.)(b.)(c.)

Resize the first image (query image) 2x, then call our sift function do redo all process of (a) & (b), then draw our new matched image.

```
    scaled_image = cv2.resize(image1, (2 * image1.shape[1], 2 * image1.shape[0]))
    matched_image = sift(scaled_image, image2, 3)
    draw_and_save(matched_image, os.path.join(img_path,"sift_match_2xscaled.jpg"))
```

Result image :



2. Discuss section:

e)

- Discuss the cases of mis-matching in the point correspondences in (c.)or(d.):
  Since we already ranked our keypoints/descriptors by their keypoint response before matching, this means our matches result will be relatively accurately, but if we discard the line:

```
    # sort by keypoints and descriptors of image1 by keypoint response
    keypoints1, descriptors1 = zip(*sorted(zip(keypoints1, descriptors1), key=lambda pair: pair[0].response, reverse=True))
```

  then the mismatches will be significantly more and easier for us to observe, for example, it matches the feature on “義” letter on the 泡芙外包裝 to the feature on the can of the second image:

- Discuss the difference between the results before and after the scale:
  Although SIFT is scale invariant, if we scale up the image, the original keypoints remains, but new keypoints may be introduced (we can see this by printing the shape of keypoints/descriptors, their number increases), so it may allow better matches to be computed, and the result can potentially be better and should not be worse than before.

一些注意事項：

※ Code 中可能留有些有助教指名不能用的 opencv function，但它們僅被用來測試及驗證我的 implementation，都有把它們註解掉，且在跑 __main__時不會被實際執行。

※ 使用 cv2.imdecod 而非 cv2.imread 主要是因為路徑問題，若為全英文路徑應該也是能照常執行，也可以 comment 掉並改成用 imread。

※ 用 matplotlib 而非 opencv 來 show 圖片主要是因為 matplotlib 比較好在圖上註記文字，且有些圖需要 x,y 軸及其他標記，為求一致性全程使用 matplotlib 的 plt.show()。

※ 執行程式時會依序顯示圖片並將圖片儲存至 img 資料夾，跳出圖片視窗任案一鍵即可關閉並繼續執行 (透過點擊視窗右上的 x 來關閉會造成程式卡住，須強制中斷)。

※ 執行 hw1-2 及 hw1-3 時因為計算量較大，所以可能耗時比較久，大概 1 分鐘內。

※ cv2.cvtColor 的 cv2.BGR2GRAY 跟 cv2.imread 的 cv2.IMREAD_GRAYSCALE 接 能將圖轉為灰階，但值好像不太一樣。