Computer Vision HW1 - Image Feature

Part 1. Harris Corner Detector

A. Corner Detection

1. Gaussian Smooth

原理 :

1. 選定一個Guassian Kernel的size與Sigma，建出Filter

   當Kernel Size為 n，則 Kernel之Size為[1…n, 1…n]

   其中 Kernel[y, x] = $\frac{1}{2\pi\sigma^2}exp^{\frac{-(x-kernelCenter_x)(y-kernelCenter_y)}{2\sigma^2}}$

   , where $kernelCenter_x = \lfloor\frac{n}{2}\rfloor$ $kernelCenter_y = \lfloor\frac{n}{2}\rfloor$
2. 將Input Image與Guassian Kernel進行2D Convolution即可得到Gaussian Smoothing Image

Implementation :

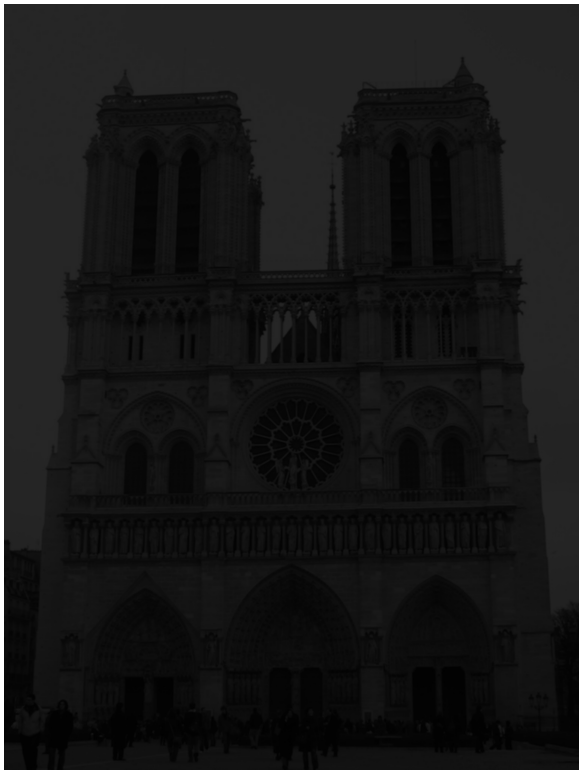
```
def GuassianSmooth(img, kernel_size, sigma):
    '''
    input :
        img: input an image with np array
        sigma: parameter for Gaussian Function
        kernel_size: size of kernel(filter)

    output:
        out: return an image that applies Gaussain Filter
    '''
    x_center = math.floor(kernel_size/2)
    y_center = math.floor(kernel_size/2)

    kernel = np.zeros((kernel_size, kernel_size))

    # 建Gaussian Kernel Filter
    for i in range(kernel_size):
        for j in range(kernel_size):
            x = j - x_center
```

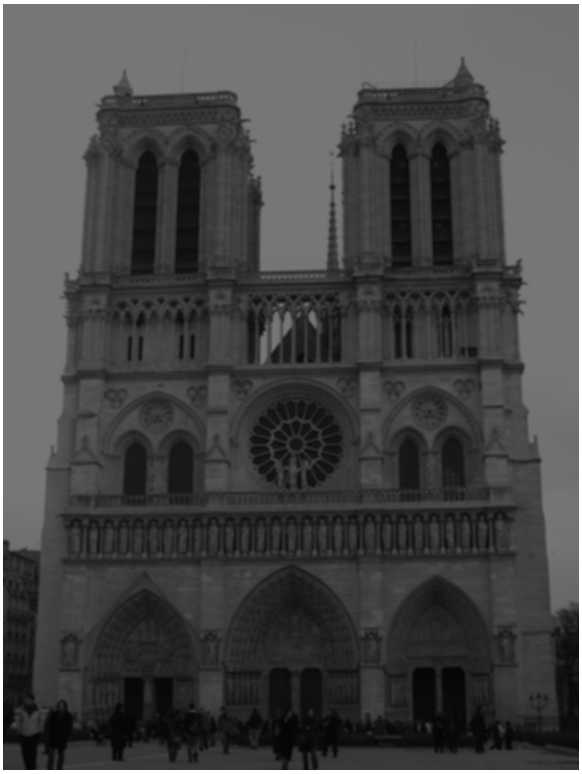
```
            y = i - y_center
            kernel[i][j] = math.exp((-(x**2 + y**2))/(2 * (sigma**2))) / (2 * math.pi * sigma**2)

    #將Input Image與Gaussian Kernel進行Convolution
    out = np.zeros(img.shape)
    out = np.round(convolve2d(img, kernel, boundary='symm', mode='same'))

    return out
```

Result :

1a_notredame

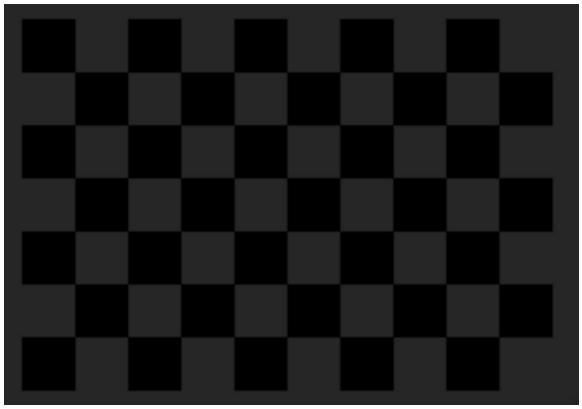
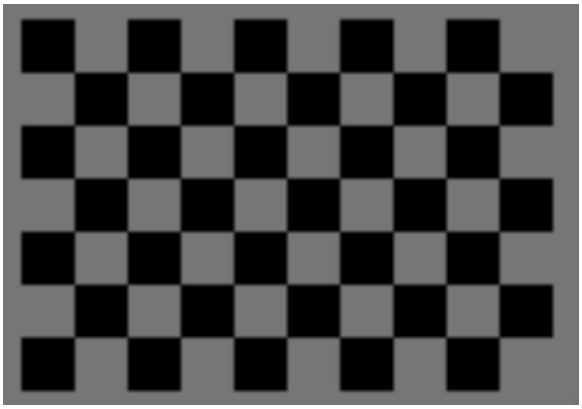
- 左圖為Sigma = 5，Kernel_Size = 5之Smoothhing之結果
- 左圖為Sigma = 5，Kernel_Size = 10之Smoothhing之結果





chessboard-hw1

- 左圖為Sigma = 5，Kernel_Size = 5之Smoothhing之結果
- 左圖為Sigma = 5，Kernel_Size = 10之Smoothhing之結果

2. Intensity Gradient (Sobel edge detection)

原理 :

1. Apply Gaussian Smoothing to get a blurred Image
2. Calculate the Gradient in X direction $\left(\frac{\partial f}{\partial x}\right)$ and in Y direction $\left(\frac{\partial f}{\partial y}\right)$ through Sobel Operator
   - Sobel Operator : Hx = $\frac{1}{8}\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$ Hy = $\frac{1}{8}\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix}$ -
   - Image與Sobel Operator之Convolution即為Gradient的近似解
   - $\frac{\partial f}{\partial x} \approx Image \circledast H_x \quad \frac{\partial f}{\partial y} \approx Image \circledast H_y$
   - 將Blurred Image from Gaussian Smoothing分別與Hx和Hy Convolution
3. 計算所有Pixel的Gradient之Manitude與Direction
   - $Magnitude(i,j) = \sqrt{(\frac{\partial f}{\partial x})^2_{(i,j)} + (\frac{\partial f}{\partial y})^2_{(i,j)}}$
   - $Direction(i,j) = \arctan((\frac{\partial f}{\partial x})_{(i,j)} / (\frac{\partial f}{\partial y})_{(i,j)})$

Implementation :

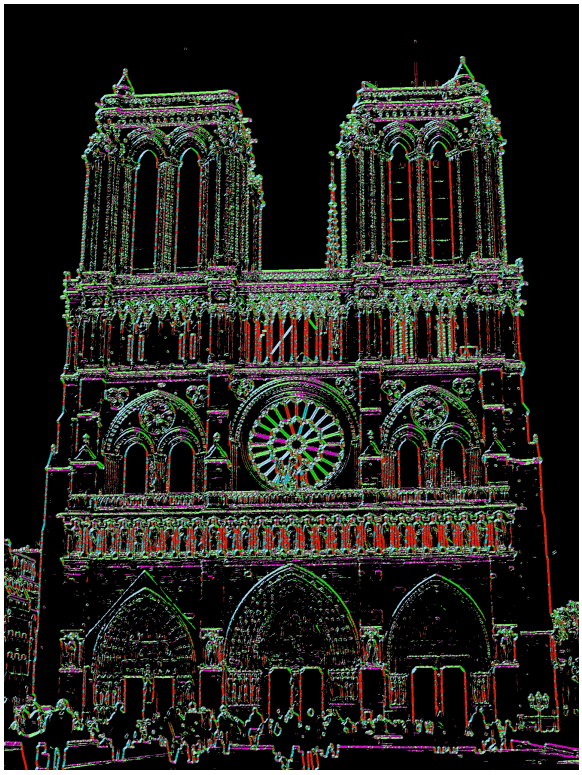
```
def ImageGradient(img):
    '''
    input:
        img : an "blured" gray image

    output:
        gradient_magnitude : the gradient magnitude of every pixel in image
        gradient_direction : the gradient direction of every pixel in image
    '''

    # Sobel Operator(Kernel)
    Hx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) #/ 8.
    Hy = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]]) #/ 8.

    # Apply sobel operator to the image => Thus we can get the Gradient of an image
    gradient_x = np.round(convolve2d(img, Hx, boundary='symm', mode='same'))
    gradient_y = np.round(convolve2d(img, Hy, boundary='symm', mode='same'))
```

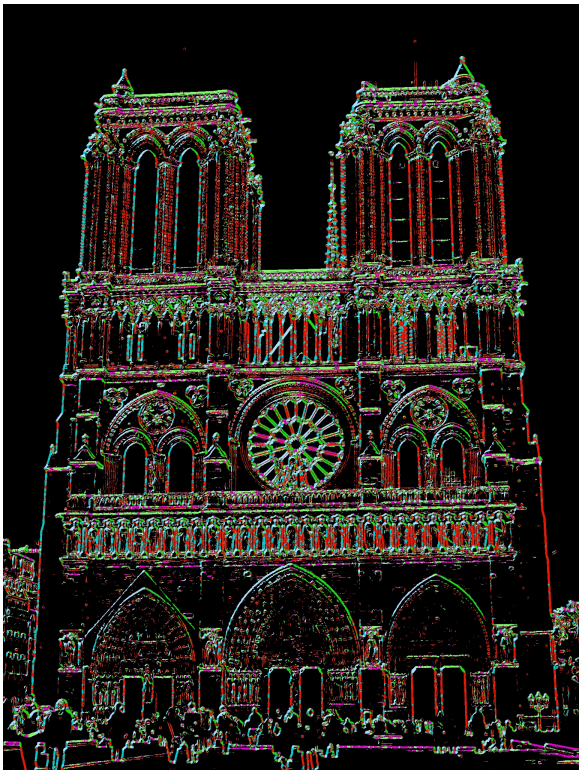
```
    gradient_magnitude = np.zeros(gradient_x.shape)
    gradient_direction = np.zeros(gradient_x.shape)

    # Calculate the gradient magnitude and direction of evey pixel in image
    for i in range(gradient_x.shape[0]):
        for j in range(gradient_x[i].shape[0]):
            gradient_magnitude[i][j] = (math.sqrt(gradient_x[i][j]**2 + gradient_y[i][j]**2))
            if gradient_x[i][j] != 0:
                gradient_direction[i][j] = math.atan(gradient_y[i][j]/gradient_x[i][j])
            else:
                # if gradient_x[i][j] == 0 => gradient_x[i][j] / gradient_y[i][j] = +inf or -inf
                if gradient_y[i][j] > 0:
                    gradient_direction[i][j] = math.pi /2
                else:
                    gradient_direction[i][j] = -math.pi /2

            #eliminate weak gradient
            if(gradient_magnitude[i][j] < 7):
                gradient_magnitude[i][j] = 0
                gradient_direction[i][j] = 0

    return gradient_magnitude, gradient_direction
```

Result :

1a_notredame - direction

- 左圖為Kernel_Size = 5 Gradient direction之結果
- 左圖為Kernel_Size = 10 Gradient direction之結果

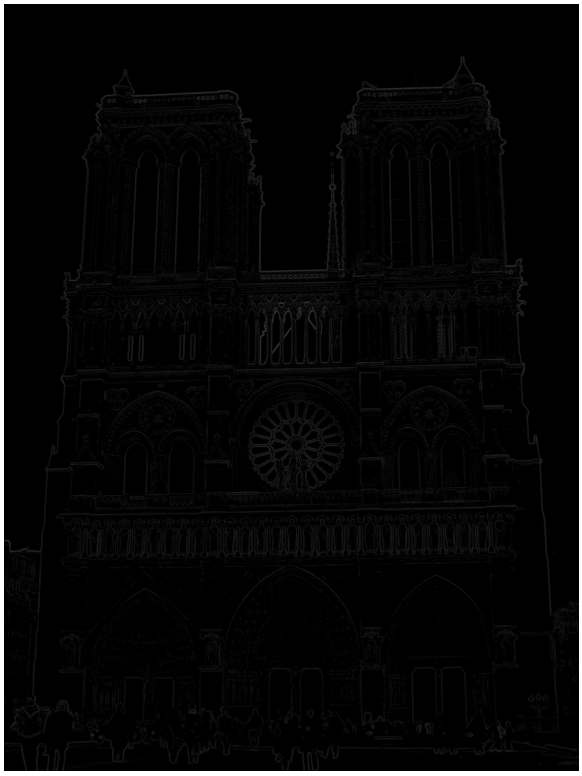
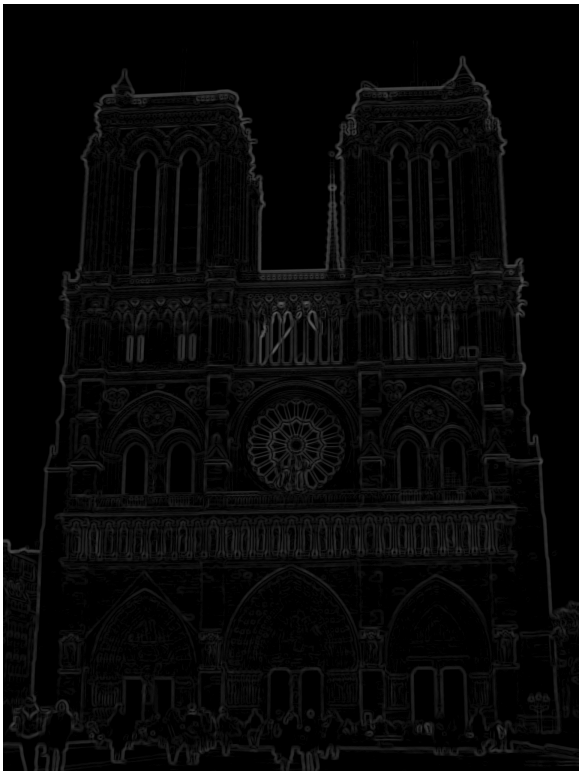




1a_notredame - magnitude

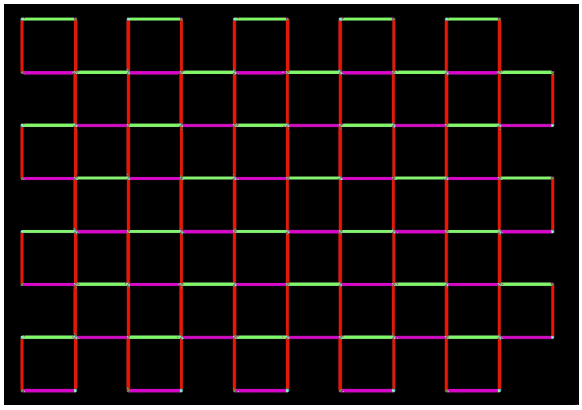
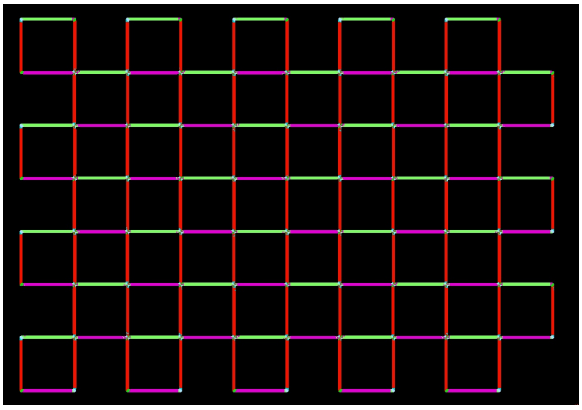
- 左圖為Kernel_Size = 5 Gradient magnitude之結果

- 左圖為Kernel_Size = 10 Gradient magnitude之結果

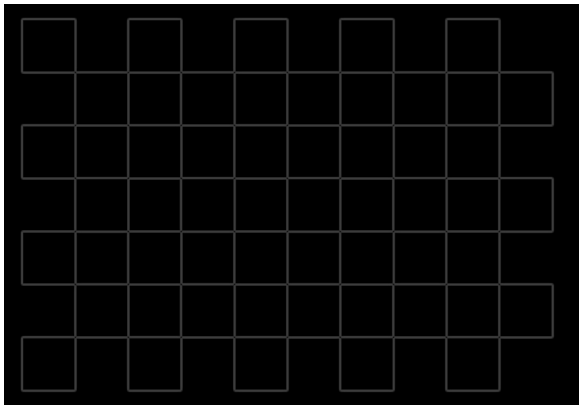
chessboard-hw1 - direction

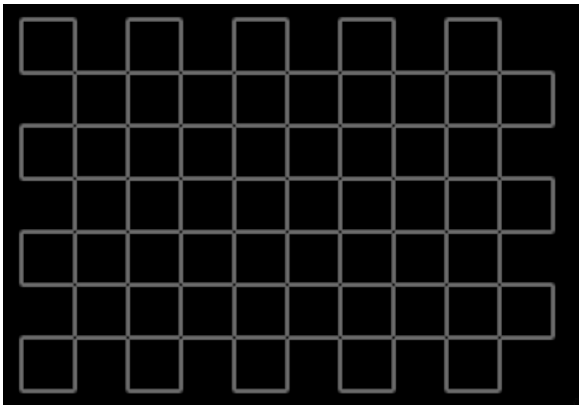
- 左圖為Kernel_Size = 5 Gradient direction之結果
- 左圖為Kernel_Size = 10 Gradient direction之結果

chessboard-hw1 - magnitude

- 左圖為Kernel_Size = 5 Gradient magnitude之結果
- 左圖為Kernel_Size = 10 Gradient magnitude之結果





3. Structure Tensor

原理 :

Implementation of Corner and edge detection

- First, Calculate the local Structure Matrix for evey pixel in image.
  - Local Structure Matrix : H(x, y) = $\sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$, where W is the local window of x and y. 舉例來說，欲計算Local Structure Matrix在(x, y)的值且Window size為3，則 $W = \{(x-1,y-1),(x-1,y),(x-1,y+1),(x,y-1),(x,y),(x,y+1),(x+1,y-1),(x$
- Second, Implement Harris Detection : 計算Image中每個Pixel的Local Structure Matrix之Response - R(x, y)。
  - Response Function : $R(x,y) = det(H(x,y)) - K(Trace(H(x,y))^2)$, where H(x, y) is a local Structure Matrix of a pixel in an image and k = 0.04~0.06 (by experiment)
  - 若R(x, y)>0 且 R(x, y)大於某個Threshold，則代表(x, y)有可能為Corner (在 Part 1 A-4 Non maximum Suppresion會找出所有Corner)

    若R(x, y)<0 且 R(x, y)小於某個Threshold，則代表(x, y)為Edge

    若R(x,y) 為一個很小的正數或者為一個很小的負數，則代表(x, y)為所在位置為Flat Region

Implementation :

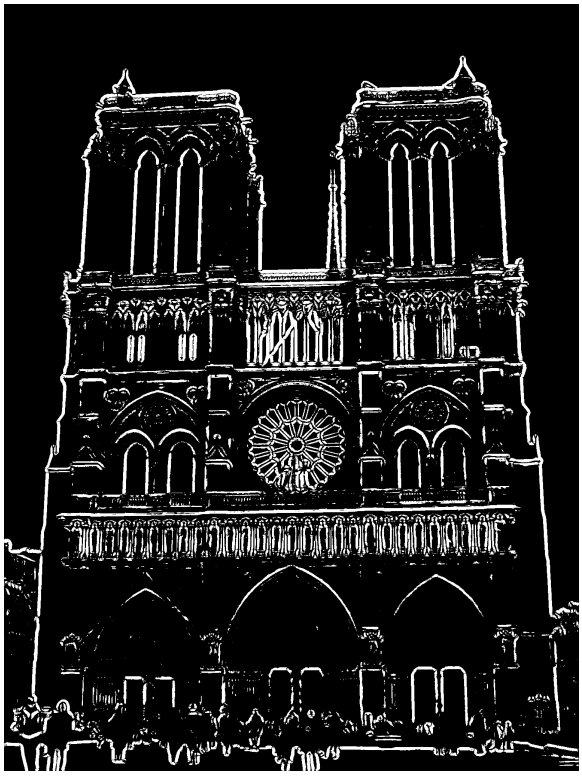
```
def StructureTensor(img, win_size):
    '''
    Input:
        img: GrayScale Input image
        win_size: The window size of the local Stucture Matrix
    Output:
        R: The matrix that stores the response of every pixel in image
    '''
    # sobel operator
    Hx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) / 8.
    Hy = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]]) / 8.

    # Calculate the Gradient in both x and y  direction
    Ix = (convolve2d(img, Hx, boundary='symm', mode='same'))
    Iy = (convolve2d(img, Hy, boundary='symm', mode='same'))
```

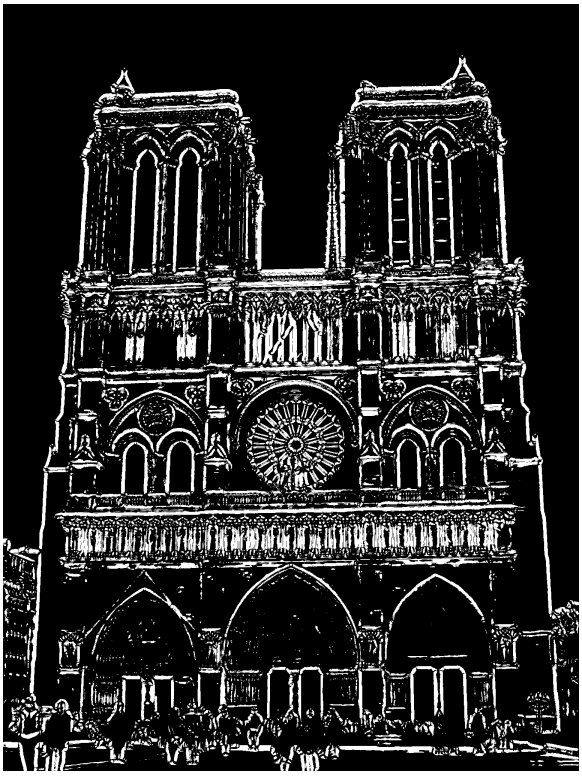
```
    # Padding
    Ix_pad = np.zeros((Ix.shape[0]+int(round(win_size/2)), Ix.shape[1]+int(round(win_size/2))))
    Ix_pad[int(round(win_size/2)/2):Ix.shape[0]+int(round(win_size/2)/2), int(round(win_size/2)/2):Ix.shape[1]+
    Iy_pad = np.zeros((Iy.shape[0]+2, Iy.shape[1]+2))
    Iy_pad[int(round(win_size/2)/2):Iy.shape[0]+int(round(win_size/2)/2), int(round(win_size/2)/2):Iy.shape[1]+


    # round the Gradient values in both x and y direction to integer
    for i in range(Ix_pad.shape[0]):
        for j in range(Ix_pad.shape[1]):
            Ix_pad[i][j] = int(round(Ix_pad[i][j]))
    Ix_pad = Ix_pad.astype(int)

    for i in range(Iy_pad.shape[0]):
        for j in range(Iy_pad.shape[1]):
            Iy_pad[i][j] = int(round(Iy_pad[i][j]))
    Iy_pad = Iy_pad.astype(int)

    # R stores the output Response function (which was already mentioned above)
    # of evey pixel in the image.
    R = []

    k = 0.04

    # Calculate the Response Value of every pixel in the image
    for i in range(img.shape[0]):
        l = []
        for j in range(img.shape[1]):
            IxIx = convolve2d(Ix_pad[i:i+win_size, j:j+win_size], Ix_pad[i:i+win_size, j:j+win_size], mode='val
            IxIy = convolve2d(Ix_pad[i:i+win_size, j:j+win_size], Iy_pad[i:i+win_size, j:j+win_size], mode='val
            IyIy = convolve2d(Iy_pad[i:i+win_size, j:j+win_size], Iy_pad[i:i+win_size, j:j+win_size], mode='val
            h = [[IxIx, IxIy], [IxIy, IyIy]]

            sign, logdet = np.linalg.slogdet(h)
            determinate = np.exp(logdet)
            Trace = (np.trace(h))
            r = determinate - k * (Trace ** 2)
            l.append(r)

        R.append(l)

    return R
```

Result :

1a_notredame

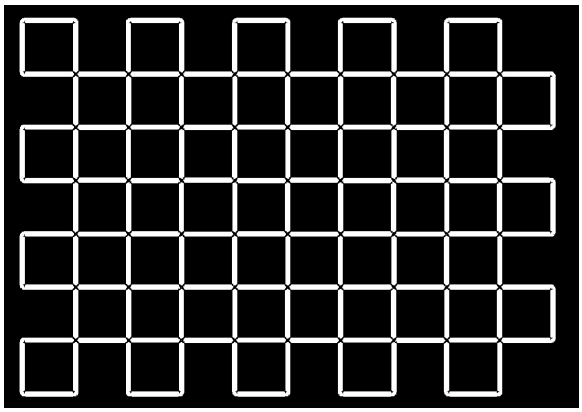
- 左圖為Kernel_Size = 10 Window Size = 3之結果
- 左圖為Kernel_Size = 10 Window Size = 5之結果

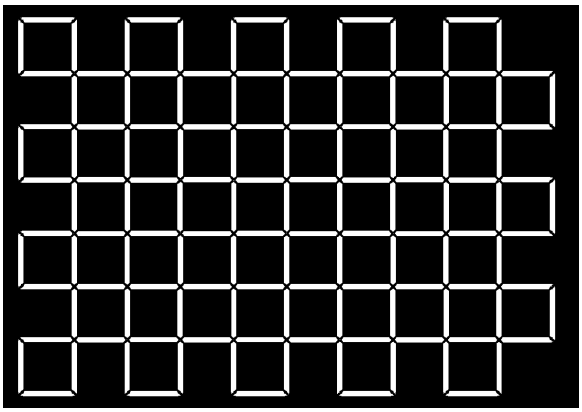




chessboard-hw1

- 左圖為Kernel_Size = 10 Window Size = 3之結果
- 左圖為Kernel_Size = 10 Window Size = 5之結果





4. Non Maximum Suppresion

原理 :

1. First, Get the Response Value of Harris Detection From the Previous Question.
    - StructureTensor Function Retruns the Response Value of Harris Detection
    - If Response Value R(x, y)>0 and R(x, y) are greater than a certain Threshold, then (x, y) is one of the best candidates to be a Corner in the Image.

- However, after calculating the Response Value by Harris Detection, we only know which points are the candidates to be the Corner. In other word, we don't know which points are actually coner.
- Thus, we have to Implement the Non Maximum Suppression Algorithm, we can judge which points are actually corner in the image.

2. Second, Implement Non Maximum Suppression.
   - 從Response Value R(x, y)>0 and R(x, y) are greater than a certain Threshold的這些Candidate Pixels中找出Corner。
   - 尋找Corner的方法 : 針對所有Candidate Pixels，檢查此Pixel沿著Gradient的Direction之R(x, y)是否為Local Maximum。

     舉例來說，若(x, y)之Gradient角度為45度，我們需要檢查(x, y)沿45度方向之R(x, y)是否為local Maximum，因此我們需要檢查R(x, y)是否大於R(x-1, y-1)與R(x+1, y+1)。

     若R(x, y)同時大於R(x-1, y-1)與R(x+1, y+1)，則R(x, y)為Local Maximum ⇒ R(x, y)為 Corner。

     若R(x, y)沒有同時大於R(x-1, y-1)與R(x+1, y+1)，則R(x, y)不是Local Maximum ⇒ R(x, y)不會是Corner。

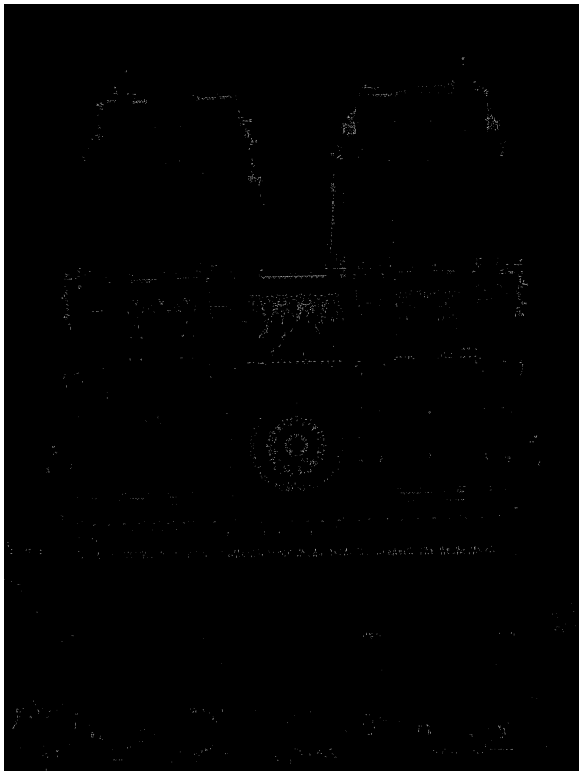
Implementation

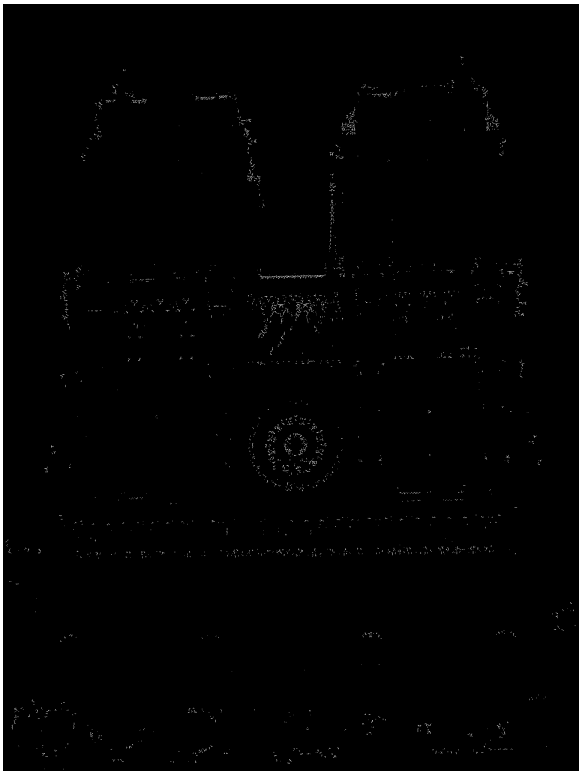
```
# LOOP Image中所有Pixel
for i in range(1, R.shape[0]-1):
    for j in range(1, R.shape[1]-1):
        # 當R大於某個Threshold且R大於0時，(i, j)有可為corner
        if R[i][j] > threshold:

            # 檢查(i, j)沿著Gradient之方向是否為Local Maximum
            # 若R[i][j]為local Maximum，則代表(i, j)為Corner
            # 當(i,j)為Corner時，則將Corner[i][j]assign為1，紀錄Corner
            if (0 <= angle[i][j] < 22.5) or (157.5 <= angle[i][j] <= 180):
                # angle 0 and 180
                if R[i][j] >= R[i][j-1] and R[i][j] >= R[i][j+1]:
                    Corner[i][j] = 1
            elif (22.5 <= angle[i][j] < 67.5):
                # angle 45
                if R[i][j] >= R[i-1][j-1] and R[i][j] >= R[i+1][j+1]:
```

Result : (將Corner以白點表示，其餘的點以黑色背景表示)

1a_notredame

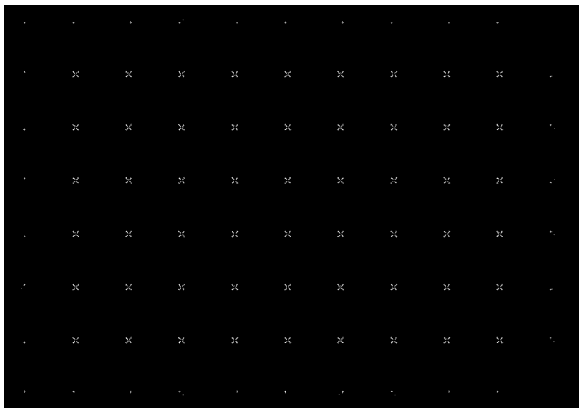
- 左圖為Kernel_Size = 10 Window Size = 3之結果
- 左圖為Kernel_Size = 10 Window Size = 5之結果

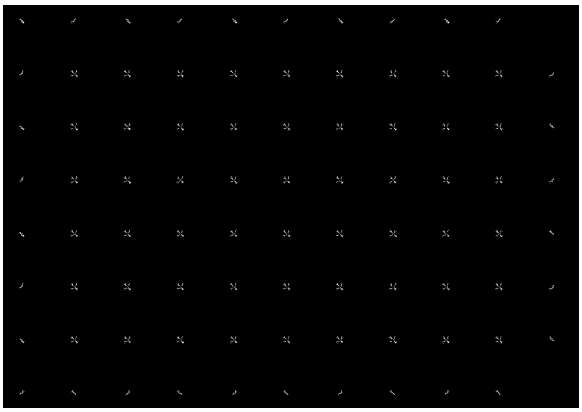




chessboard-hw1

- 左圖為Kernel_Size = 10 Window Size = 3之結果
- 左圖為Kernel_Size = 10 Window Size = 5之結果



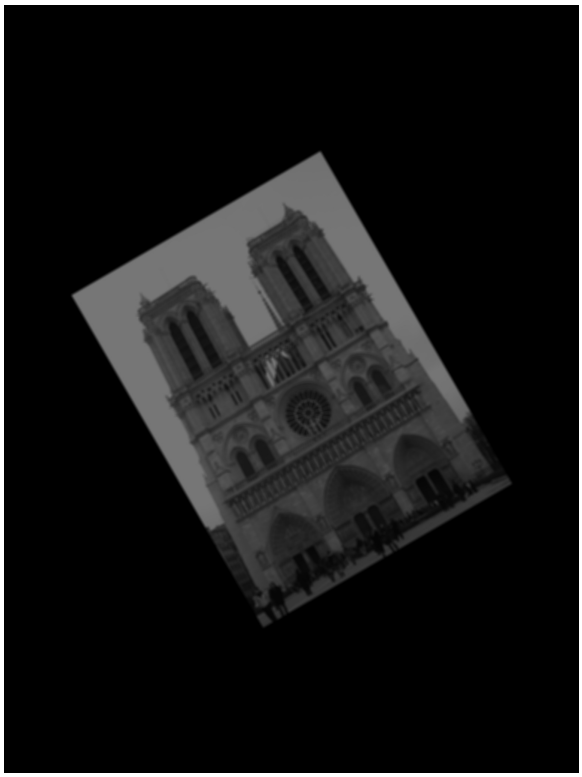


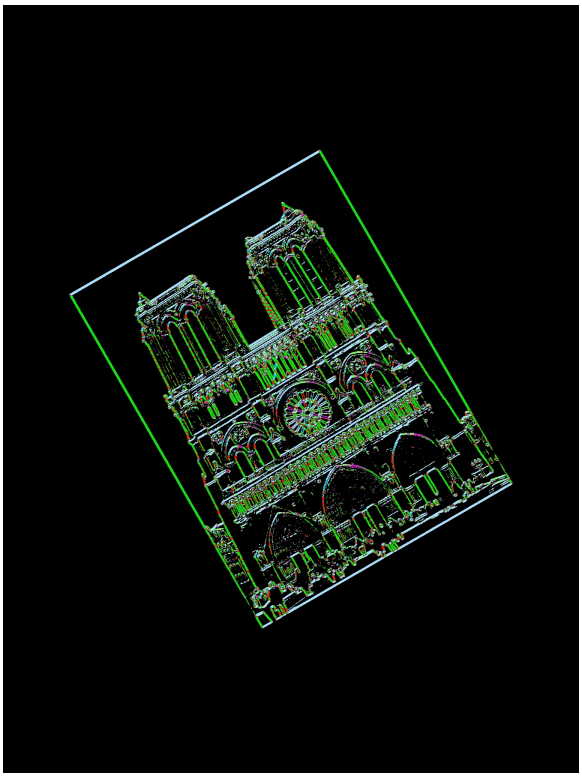
B. Experiments (Rotate and Scale):

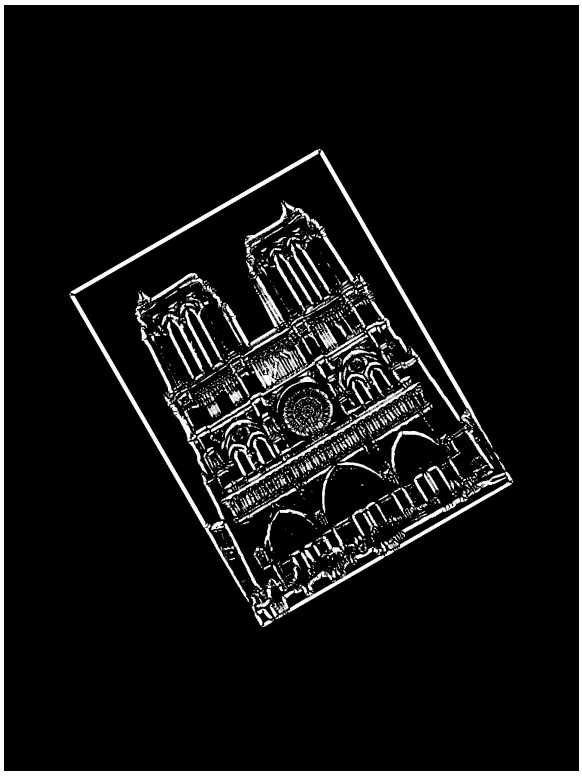
Implementation

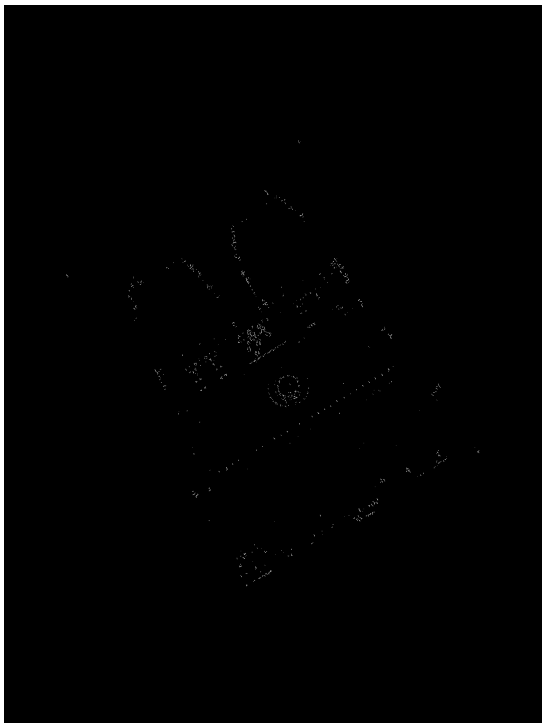
```
h = Grayimage.shape[0]
w = Grayimage.shape[1]

M = cv2.getRotationMatrix2D((w//2, h//2), 30, 0.5)
Grayimage = cv2.warpAffine(Grayimage, M, (w,h))
saved_file = 'transformed/' + file_name
```

以下四張圖依序為1a_notredame Part 1 - A a~d之結果

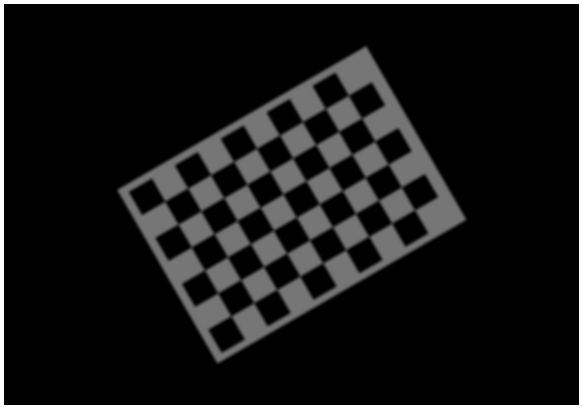
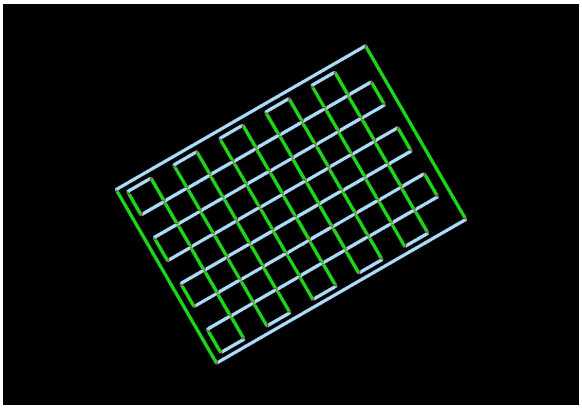
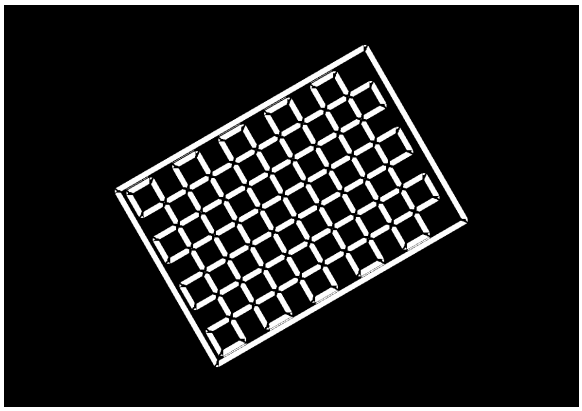
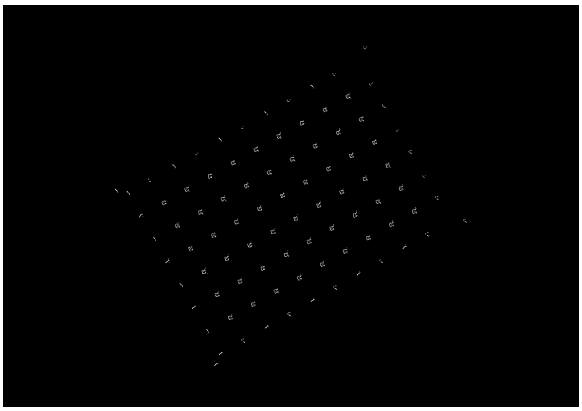








chessboard-hw1 Part 1 - A a~d之結果

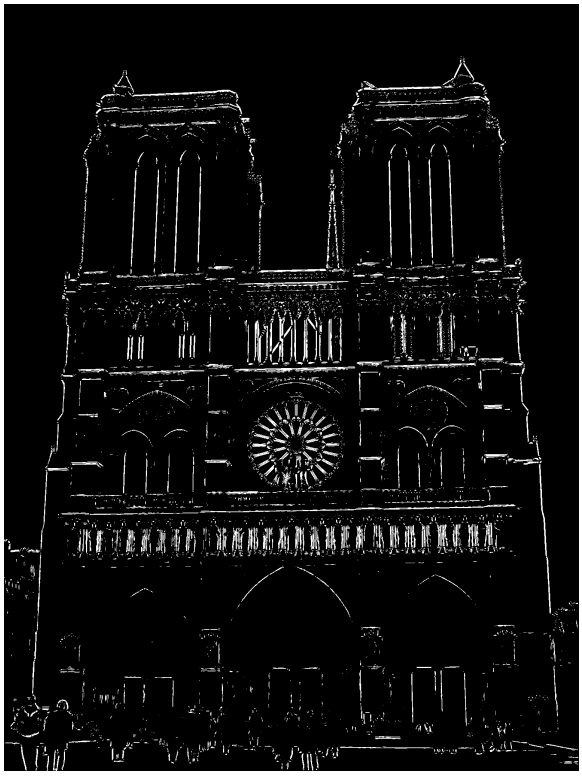
C. Discussion:

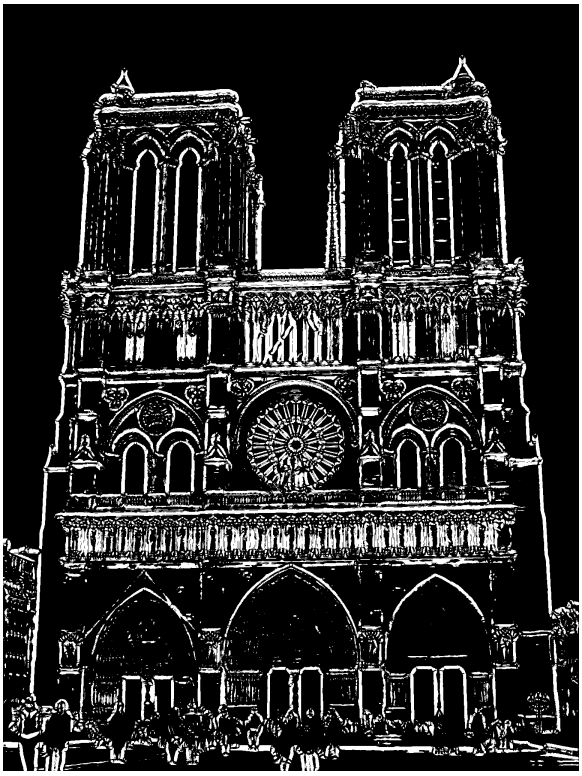
a. Discuss the results of blurred images and detected edge between different kernel sizes of Gaussian filter.

- blurred images between different kernel sizes of Gaussian filter - 從Part 1 A-a之結果可以看出與Kernel Size較大之filter進行Convolution會較模糊

- 左圖為Kernel size = 5之圖片進行Detect Edge，右圖為Kernel size = 10之圖片進行Detect Edge

  ⇒ 從下圖可以看出Smoothing之Kernel Size越大，Detect Edge之效果越好





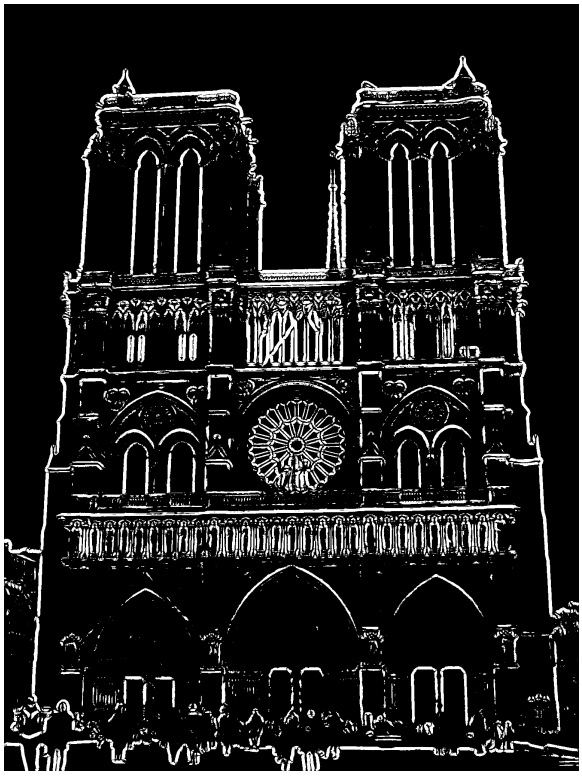
b. Difference between 3x3 and 5x5 window sizes of structure tensor.

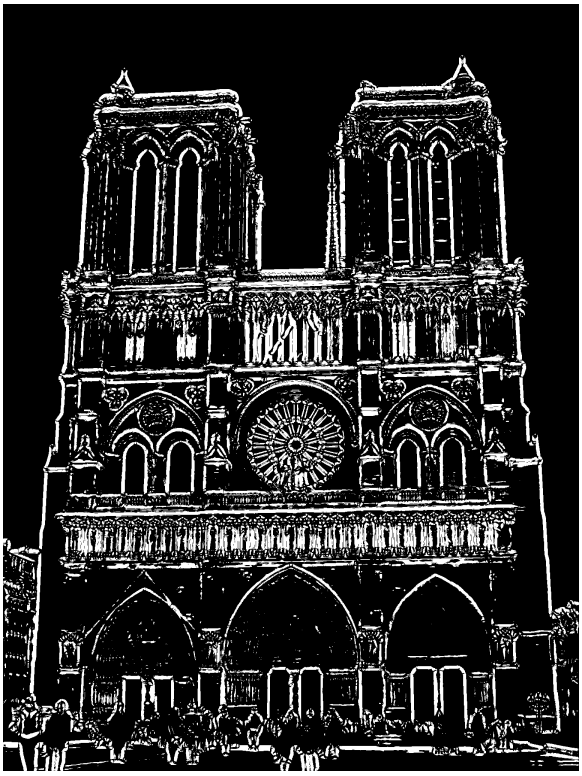
以下兩張圖將Edge以白色顯示，其餘點以黑色顯示

左圖為使用3x3 window進行Structure Tensor進行Edge Detect的結果

右圖為使用5x5 window進行Structure Tensor進行Edge Detect的結果

可以發現使用5x5 window會較使用3x3 window進行Structure Tensor Detect Edge得到更細節的結果





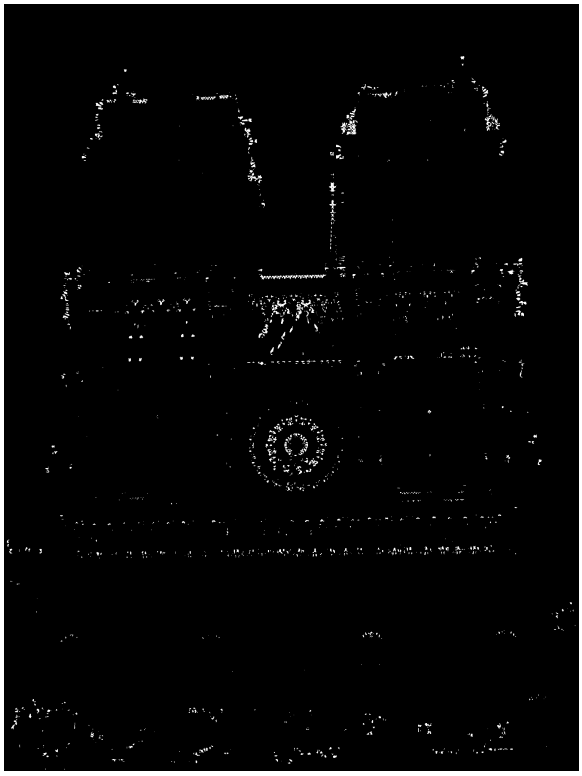
c. The effect of non-maximal suppression.

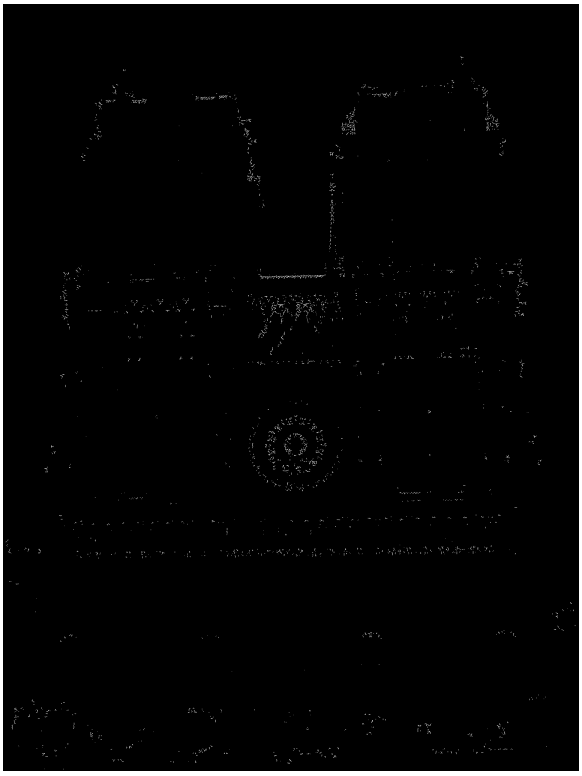
以下內容在Part 1 A-d有提及過

- StructureTensor Function Retruns the Response Value of Harris Detection
- If Response Value R(x, y)>0 and R(x, y) are greater than a certain Threshold, then (x, y) is one of the best candidates to be a Corner in the Image.
- However, after calculating the Response Value by Harris Detection, we only know which points are the candidates to be the Corner. In other word, we don’t know which points are actually coner.
- Thus, we have to Implement the Non Maximum Suppression Algorithm, we can judge which points are actually corner in the image.
- Non-maximal Suppression會從Response Value R(x, y)>0 and R(x, y) are greater than a certain Threshold的這些Candidate Pixels中找出Corner。

左圖為有Apply Non-Maximal Supression Detect 到 Corner之結果

右圖為Harris Detection中$R(x,y) = det(H(x,y)) - K(Trace(H(x,y))^2) > 0$之Edge Candidate，未Apply Non-Maximal Supression Detect Local Maximum之結果





d. Discuss the result from (B). Is Harris detector rotation-invariant or scale-invariant?

Yes, Harris Detector is rotation-invariant and scale-invariant. (旋轉與縮小Edge仍可被Detect)

Part 2. SIFT interest point detection and matching

A. SIFT interest point detection

a. Apply SIFT interest point detector (functions from OpenCV) to the following two images

Implementation :

After calling the function sift.detect(), we'll get all the interest points and the interest points will be saved in variable kp.

```
sift = cv2.SIFT_create()
kp = sift.detect(gray,None)
```

b. Adjust the related thresholds in SIFT detection such that there are around 100 interest points detected in each image .

Implementation :

To get around 100 interest points, we need to add an additional threshold when creating the SIFT Detection Object. The variable kp will record the interests points that were found in SIFT Detection

```
sift = cv2.SIFT_create(ontrastThreshold = 0.03)
kp = sift.detect(gray,None)
```

c. Plot the detected interest points on the corresponding images

以下兩張圖片為Part2 A-a之結果(沒設Threshold)

- 左圖為1a_notredame.jpg進行SIFT Detection之後的結果，一共找到23657個interest points
- 右圖為1b_notredame.jpg進行SIFT Detection之後的結果，一共找到18751個interest points

以下兩張圖片為Part2 A-b之結果(有設Threshold，Threshold = 0.2)

- 左圖為1a_notredame.jpg進行SIFT Detection之後的結果，一共找到312個interest points
- 右圖為1b_notredame.jpg進行SIFT Detection之後的結果，一共找到500個interest points

B. SIFT feature matching

STEP 1 : 找出img1與img2的keypoints與discriptor

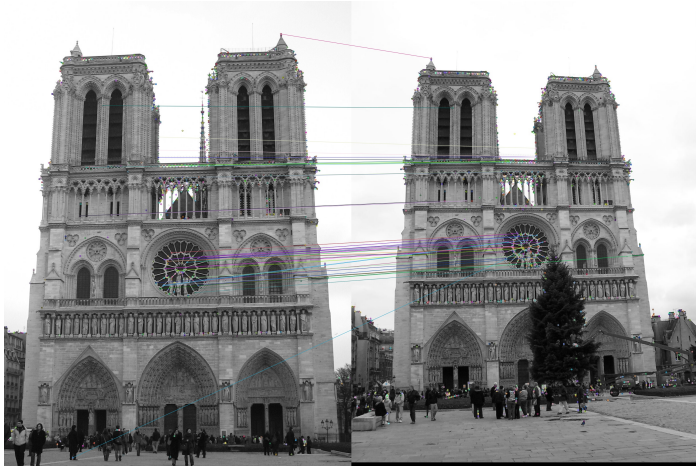
STEP 2 : BruteForce計算img1與img2中任兩個discriptor之間的L1 Norm(distance) ⇒distance function

- 每次取img1之其中一個discriptor，與img2中所有discriptor計算distance。取最小的兩個distance相除，若相除結果小於某個Threshold，則該次計算的img1之discriptor與img2中和img1 distance最小的keypoint為Match Keypoint。⇒ Neighbor Matching

STEP 3 : 將Match的keypoints紀錄下來，並畫圖

```
def SIFTFeaturMatching(img1, img2):
    sift = cv2.SIFT_create(contrastThreshold = 0.16)

    keypoints1, descriptors1 = sift.detectAndCompute(img1,None)
    keypoints2, descriptors2 = sift.detectAndCompute(img2,None)

    matches = []

    # brute force find euclidian distance between two keypoints(discriptor)
    for idx1, val1 in enumerate(descriptors1):

        distance = []

        for idx2 in range(0, descriptors2.shape[0]):
            dis = np.linalg.norm(descriptors2[idx2] - val1, ord=1)
            distance.append([dis, idx2])

        distance = sorted(distance, key = lambda x: x[0])

        # 取距離最短的兩個值相除，小於threshold才match
        if  distance[0][0] > 0.2 * distance[1][0]: #and distance[1][0] / distance[0][0] < 1.3:
            matches.append(cv2.DMatch(idx1, distance[0][1], distance[0][0]))

        del distance
```

```
matches = sorted(matches, key = lambda x:x.distance)

result = cv2.drawMatches(img1, keypoints1, img2, keypoints2, matches[:50], None)

cv2.imwrite('2/2b.jpg', result)
```



C. Discussion

a. Discuss the cases of mis-matching in the point correspondences

下圖之屋頂有許多條線Mis-Matching，並未連到對應點

b. Discuss and implement possible solutions to reduce the mis-matches, and show your results.

調整Part 2 B 提及之Step2的Threshold (每次計算img1其中一個discriptor與img2所有discriptor中distane最小的兩個值相除。若相除值小於Threshold，則img1該次計算的discriptor與和img2距離最短的discriptor才會配對成功)，即可減少mismatch

左圖的Threshold為0.9，可以看出屋頂有許多mis-matching的點

右圖的Threshold為0.2，可以看mis-matching的點少很多