Computer Vision Hw1

```
from cmath import exp, pi
from operator import concat
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage
from scipy import signal
from skimage import color
from scipy.spatial import distance
```

Gaussian filter window

$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}$$

```
def Gaussian(x,y,sigma=5):
    gau = (1.0 / 2.0*pi*sigma**2 )* np.exp(-((x**2 + y**2) / (2.0 * sigma**2)))
    return gau/gau.sum()


def _filter(m,n):
    y,x = np.ogrid[-m//2+1:m//2+1,-n//2+1:n//2+1]
    G = Gaussian(x,y)
    return G
```

Sobel Operator

- Sobel gradient x
- Sobel gradient y
- Direction
- magnitude

```
def sobel(img):
    gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    gray_img = gray_img/255.0
    vertical = np.array([[-1 ,0,1], [-2,0,2], [-1,0,1]]) * 1.0/8.0
    # sobel gradient x
    Gx = signal.convolve2d(gray_img, vertical)
    horizontal = np.array([[1,2,1],[0,0,0],[-1,-2,-1]]) * 1.0/8.0
    # sobel gradient y
    Gy = signal.convolve2d(gray_img, horizontal)
    # magnitude
    G = np.sqrt(np.square(Gx) + np.square(Gy))
    G *= 255.0 / G.max()

    # Direction
    theta = np.arctan2(Gx, Gy)
    hsv = np.zeros((G.shape[0], G.shape[1], 3))
    hsv[..., 0] = (theta+ np.pi) / (2 * np.pi)
    hsv[..., 1] = np.ones((G.shape[0], G.shape[1]))
    hsv[..., 2] = (G - G.min()) / (G.max() - G.min())
    rgb = color.hsv2rgb(hsv)
    return Gx, Gy, G, theta, rgb
```

Structure Tensor

- Return Ixx Ixy Iyy and responce

```
def structure_matrix(size, dx, dy):
    # window size
    filter_ = np.ones((size,size))
    # Ixx
    Axx = signal.convolve2d(dx * dx, filter_, mode="same")
    # Iyy
    Ayy = signal.convolve2d(dy * dy, filter_, mode="same")
    # Ixy
    Axy = signal.convolve2d(dx * dy, filter_, mode="same")

    landa2, responce = cal_eigen(Axx, Axy, Ayy)
    landa2 *= 255.0
    landa2[responce > np.average(resopnce)] = 1

    return Axx, Axy, Ayy, responce, landa2
```

1. Eigen value = landa1 * landa2 / landa2
2. We can easily get determine approach to eigen value
3. Trace is approach to trace
4. Then we have landa2

```
def cal_eigen(Axx, Axy, Ayy, k=0.04):
    det = Axx * Ayy - Axy * Axy
    trace = Axx + Ayy

    landa2 = det/(trace+1e-9)
    responce = det - k * (trace * trace)

    return landa2, responce
```

Non-maximum suppresion

2 REQUIREMENTS

- Find points with large response
- Choose those points where landa- is a local maximum as features

```
def NMS(r, threshold = 0.04):
    # larger than threshold
    mask1 = (r > threshold)
    # local maximum
    mask2 = np.full((r.shape[0], r.shape[1]), True)
    size = 5
    for i in range(0, r.shape[0], size):
        for j in range(0, r.shape[1], size):
            local_window = r[i:i+5,j:j+5]
            x,y = np.unravel_index(local_window.argmax(), local_window.shape)
            local_mask = np.full((local_window.shape[0], local_window.shape[1]), False)
            local_mask[x,y] = True
            mask2[i:i+5, j:j+5] = local_mask
    mask =  (mask1 & mask2)

    # plot picture
    x,y = np.nonzero(mask)
    return x,y
```

COMBINE 2 IMAGES IN ONE

```
def plot_images(kp_left_img, kp_right_img):
    total_kp = np.concatenate((kp_left_img, kp_right_img), axis=1)
    plt.imshow(total_kp)
    return total_kp
```

PLOT THE MATCHES BETWEEN 2 IMAGES ON MERGED IMAGE

```
def plot_matches(matches, img):
    match_img = img.copy()
    offset = img.shape[1]/2
    fig, ax = plt.subplots()
    ax.set_aspect('equal')
    ax.imshow(np.array(match_img).astype('uint8'))

    ax.plot(matches[:, 0], matches[:, 1], 'xr')
    ax.plot(matches[:, 2] + offset, matches[:, 3], 'xr')

    ax.plot([matches[:, 0], matches[:, 2] + offset], [matches[:, 1], matches[:, 3]],
            'r', linewidth=0.5)
    plt.show()
```

SIFT

- Keypoints
- decribtor
- calculate similarity between each features **a.b / |a| * |b| **

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

```
def SIFT(img1, img2, gray1, gray2):
    # For same shape
    gray1.resize((2032,1536),refcheck=False)
    img1.resize((2032,1536,3),refcheck=False)
    # descriptor
    sift1 = cv2.SIFT_create(contrastThreshold=0.21, edgeThreshold=10, sigma=1.6)
    sift2 = cv2.SIFT_create(contrastThreshold=0.2, edgeThreshold=30, sigma=1.6)
    kp1, des1 = sift1.detectAndCompute(gray1.astype('uint8'),None)
    kp2, des2 = sift2.detectAndCompute(gray2.astype('uint8'),None)
    # plot merged image
    img3 = plot_images(img1, img2)
    img1 = cv2.drawKeypoints(gray1.astype('uint8'),kp1,img1, color=(255,100,255))
    img2 = cv2.drawKeypoints(gray2.astype('uint8'),kp2,img2, color=(255,100,255))

    # SIFT feature matching
    # similarity
    similarity = np.zeros((des1.shape[0], des2.shape[0]))
    for i in range(des1.shape[0]):
        for j in range(des2.shape[0]):
            similarity[i][j] = np.dot(des1[i], des2[j]) / (np.linalg.norm(des1[i]) * np.

    return similarity, img3, kp1, des1, kp2, des2, img1, img2
```

```
def match(similarity, kp1, kp2):
    # 2-nearest neighbor
    k = 2
    matches = []
    nn = []
    for idx in range(similarity.shape[0]):
        local = similarity[idx]
        max2min = sorted(local, reverse=True)
        k_nn = np.zeros(k)
        # find index
        best = np.where(local==max2min[0])[0]
        best = list(kp1[idx].pt + kp2[int(best)].pt)
        matches.append(best)
        for i in range(k-1):
            k_nn[i] = np.where(local==max2min[i+1])[0][0]
            match = list(kp1[idx].pt + kp2[int(k_nn[i])].pt)
            nn.append(match)
    matches = np.array(matches)
    return matches


def improve_match(similarity ,kp1, kp2):
    # 2-nearest-neighbor
    k = 2
    matches = []
    legal = True
    for idx in range(similarity.shape[0]):
        legal = True
        local = similarity[idx]
        max2min = sorted(local, reverse=True)
        k_nn = np.zeros(k)
        # find index
        best = np.where(local==max2min[0])[0]
        good = []
        good.append(list(kp1[idx].pt + kp2[int(best)].pt))
        for i in range(k-1):
            k_nn[i] = np.where(local==max2min[i])[0][0]
            if (max2min[0]*0.92 > max2min[i+1]):
                match = list(kp1[idx].pt + kp2[int(k_nn[i])].pt)
                good.append(match)
            else:
                legal=False
        if legal == True:
            matches= matches + good
    matches = np.array(matches)
    return matches
```

Homework To Do

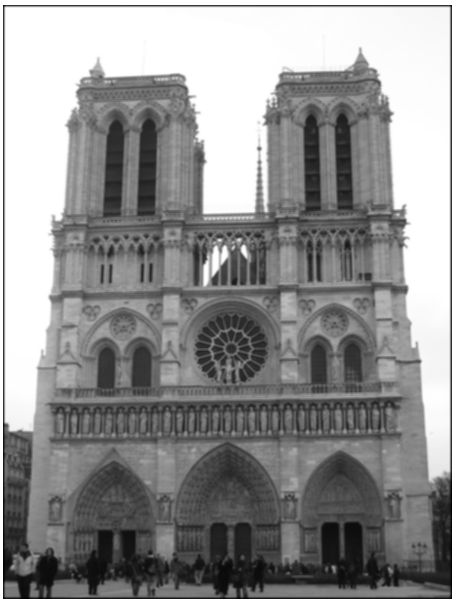
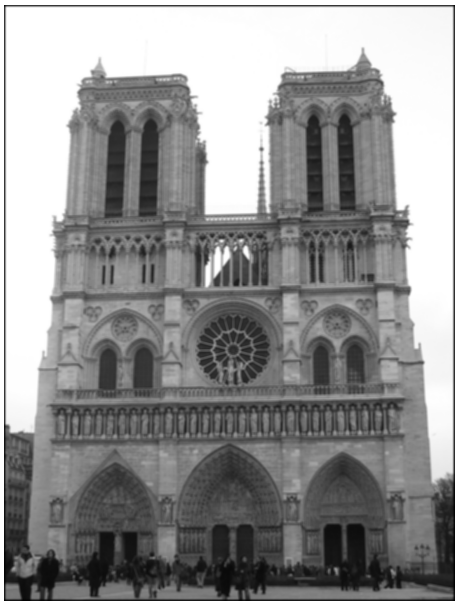
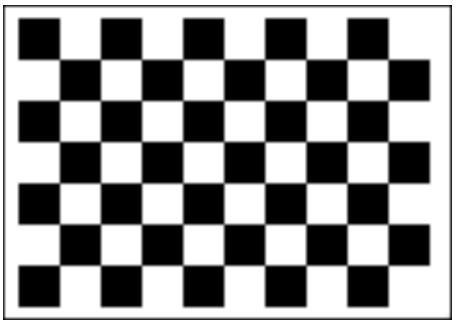
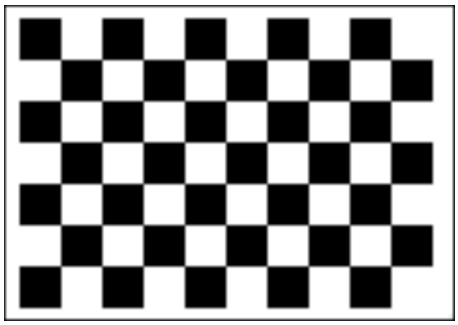
- Load images

```
chess_img = cv2.imread("chessboard-hw1.jpg")
notredame_a = cv2.imread("1a_notredame.jpg")
notredame_b = cv2.imread("1b_notredame.jpg")
chess_img = cv2.cvtColor(chess_img, cv2.COLOR_BGR2RGB)
chess_gray = cv2.cvtColor(chess_img, cv2.COLOR_RGB2GRAY)
notredame_a_gray = cv2.cvtColor(notredame_a, cv2.COLOR_RGB2GRAY)
notredame_b_gray = cv2.cvtColor(notredame_b, cv2.COLOR_RGB2GRAY)
```

1-1

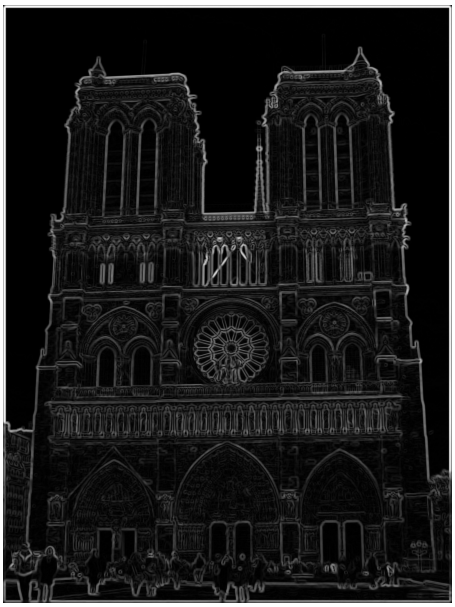
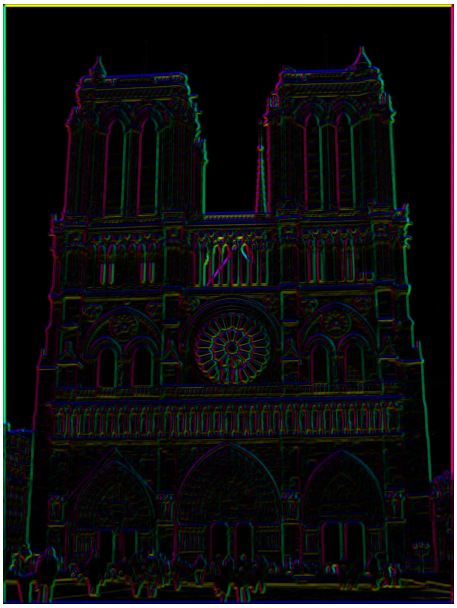
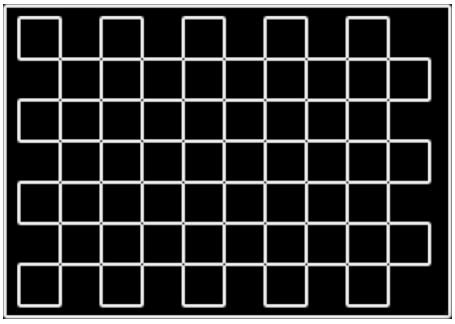
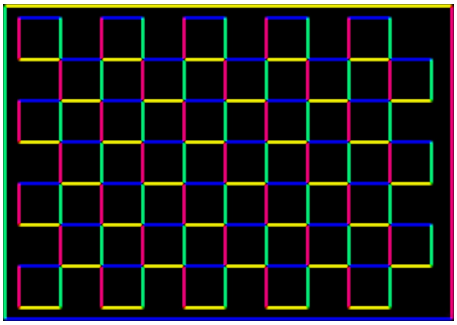
(a) Gaussian Smooth: Show the results of Gaussian smoothing for σ=5 and kernel size=5 and 10 respectively.

```
# ### Hw1-1 ###
"""
# 1(a)
# Gaussian smoothing
"""
# size10
G_10 = _filter(10,10)
# size5
G_5 = _filter(10,10)
chess_conv_5 = signal.convolve2d(chess_gray, G_5)
notredame_a_conv_5 = signal.convolve2d(notredame_a_gray, G_5)
chess_conv_10 = signal.convolve2d(chess_gray, G_10)
```

window size 5	window size 10
	
	

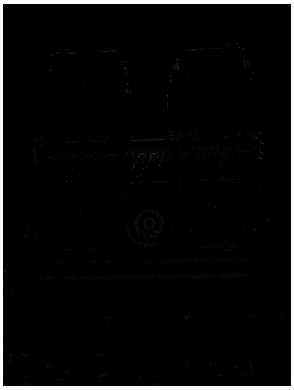
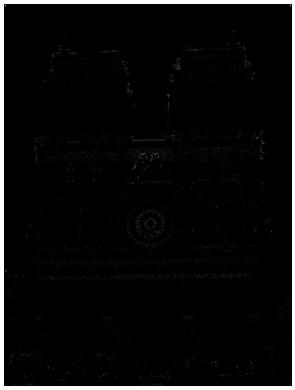
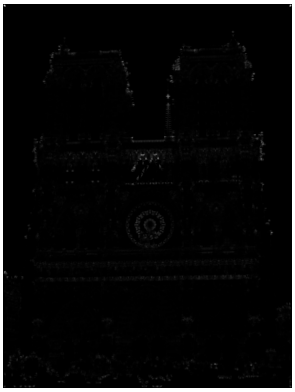
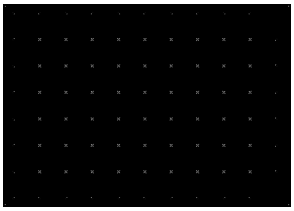
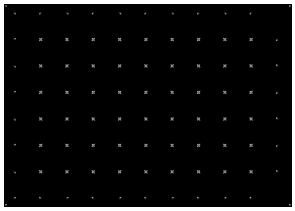
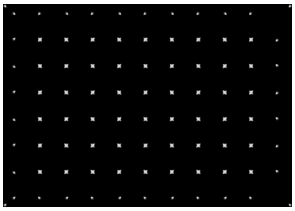
(b) Intensity Gradient (Sobel edge detection): Apply the Sobel filters to the blurred images and compute the magnitude (2 images) and direction (2 images) of gradient. (You should eliminate weak gradients by proper threshold.)

```
"""
1(b)
"""
# Intensity Gradient (Sobel edge detection)
dx_c, dy_c, magnitude_c, theta_c, direction_c = sobel(chess_img)
dx_a, dy_a, magnitude_a, theta_a, direction_a = sobel(notredame_a)
```

Magnitude	Direction
	
	

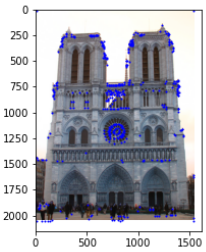
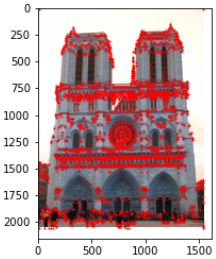
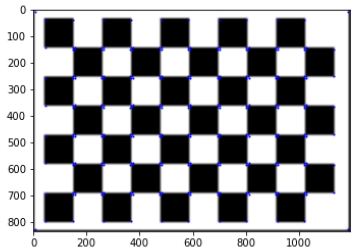
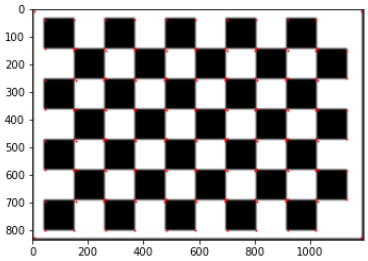
© Structure Tensor: Use the Sobel gradient magnitude (with Gaussian kernel size=10) above to compute the structure tensor H of each pixel. Show the images of the smaller eigenvalue of H with window size 3x3 and 5x5. (2 images)

```
"""
1(c)
"""

# Structure Tensor
# Chess image
Axx_c_3,Axy_c_3,Ayy_c_3,responce_c_3, landa2_c_3 = structure_matrix(size = 3, dx=dx_c, d
Axx_c_5,Axy_c_5,Ayy_c_5,responce_c_5, landa2_c_5 = structure_matrix(size = 5, dx=dx_c, d
Axx_c_10,Axy_c_10,Ayy_c_10,responce_c_10, landa2_c_10 = structure_matrix(size = 10, dx=d

# 1a_notredame
Axx_a_3,Axy_a_3,Ayy_a_3,responce_a_3, landa2_a_3 = structure_matrix(size = 3, dx=dx_a, d
Axx_a_5,Axy_a_5,Ayy_a_5,responce_a_5, landa2_a_5 = structure_matrix(size = 5, dx=dx_a, d
Axx_a_10,Axy_a_10,Ayy_a_10,responce_a_10, landa2_a_10 = structure_matrix(size = 10, dx=d
```

3X3	5X5	10X10
		
		
		

(d) Non-maximal Suppression: Perform non-maximal suppression on the results above along with appropriate thresholding for corner detection. (2 images)

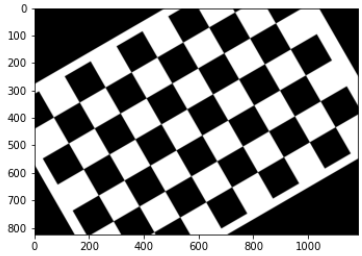
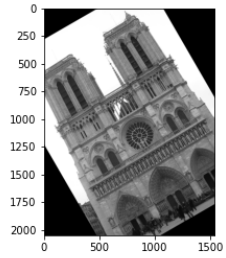
```
"""
1(d)
"""
# Chess image
#Non-maximal Suppression
# size=5
r,c =  NMS(responce_c_5)
# size=10
r,c =  NMS(responce_c_10)

# 1a_notredame image
#Non-maximal Suppression
# size=5
r,c =  NMS(responce_a_5)
# size=10
r,c =  NMS(responce_a_10)
```

5X5	10X10
	
	
	

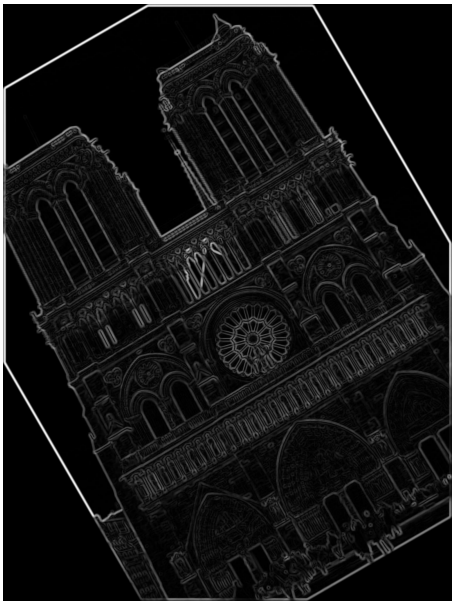
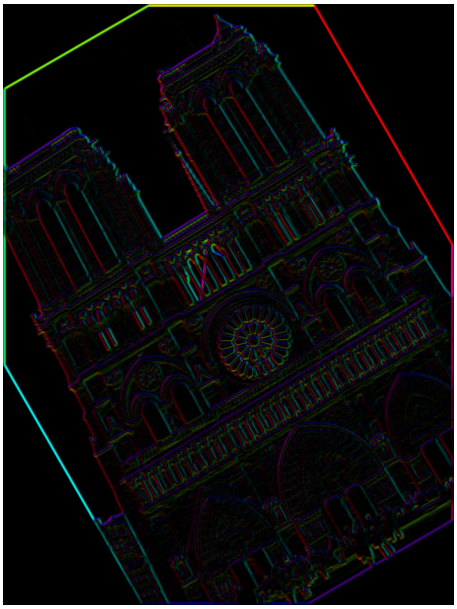
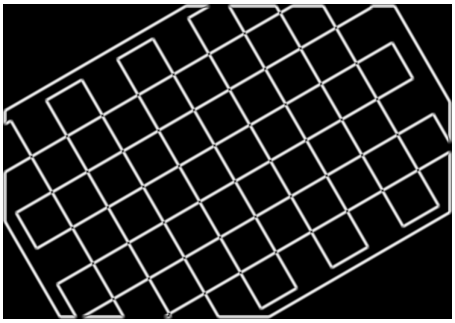
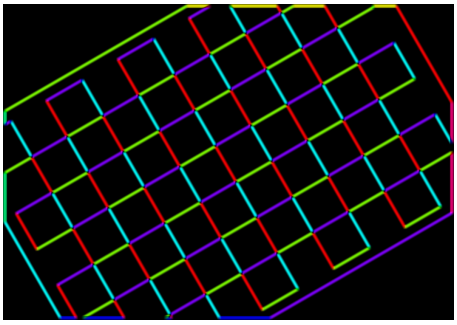
(b) Experiments (Rotate and Scale):Apply the same corner detection algorithm to the rotated (by 30°) and scaled (to 0.5x) images. (2 images)

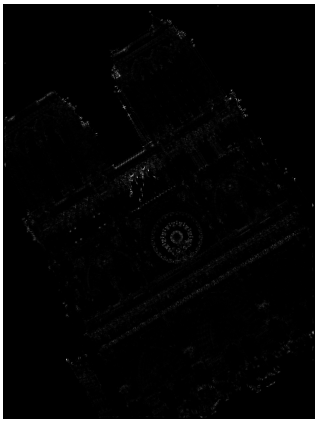
rotate

```
M = cv2.getRotationMatrix2D((chess_img.shape[1]/2,chess_img.shape[0]/2),30,1)
chess_image_rotate = cv2.warpAffine(chess_img,M,(chess_img.shape[1],chess_img.shape[0]))

#a_notredame = cv2.rotate(notredame_a, cv2.ROTATE_30)
M = cv2.getRotationMatrix2D((notredame_a.shape[1]/2,notredame_a.shape[0]/2),30,1)
notredame_a_rotate = cv2.warpAffine(notredame_a,M,(notredame_a.shape[1],notredame_a.shap
```

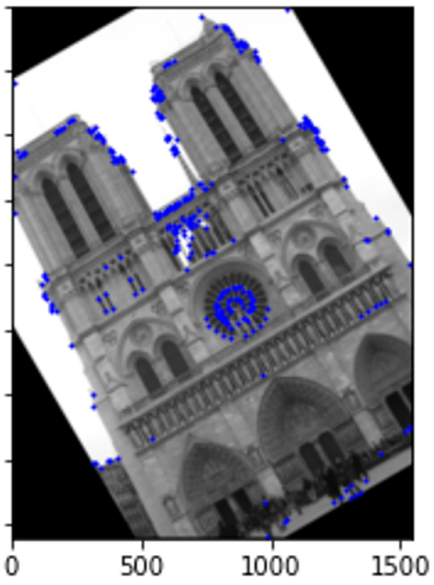
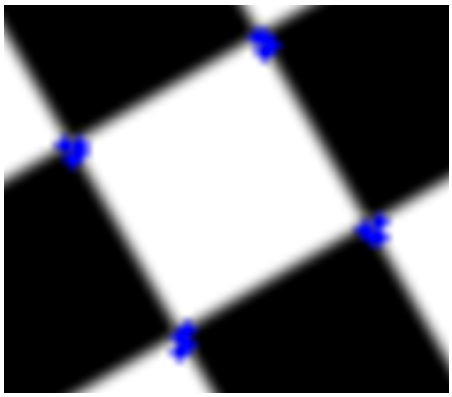
rotate Chess	rotate notredame
	

```
dx_c_r, dy_c_r, magnitude_c_r, theta_c_r, direction_c_r = sobel(chess_image_rotate)
dx_a_r, dy_a_r, magnitude_a_r, theta_a_r, direction_a_r = sobel(notredame_a_rotate)

Axx_c_5_r,Axy_c_5_r,Ayy_c_5_r,responce_c_5_r, landa2_c_5_r = structure_matrix(size = 5,
Axx_a_5_r,Axy_a_5_r,Ayy_a_5_r,responce_a_5_r, landa2_a_5_r = structure_matrix(size = 5,
```

Rotate Maginitude	Rotate Direction
	
	

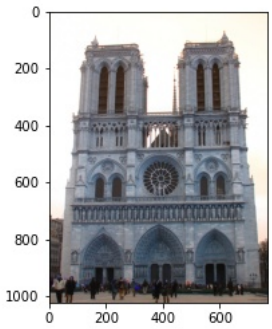
Rotate Chess Structure Tensor	Rotate notredame Structure Tensor
	

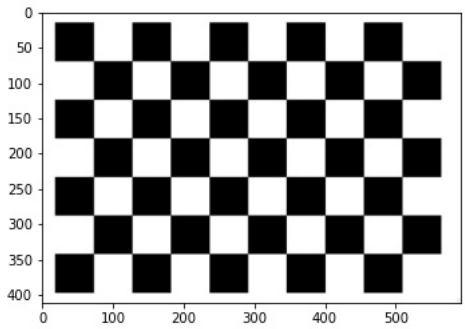
```
# Chess rotated image
#Non-maximal Suppression
r,c =  NMS(responce_c_5_r)
# notredame_a rotated image
#Non-maximal Suppression
r,c =  NMS(responce_a_5_r)
```

Rotate Chess NMS	Rotate notredame NMS
	

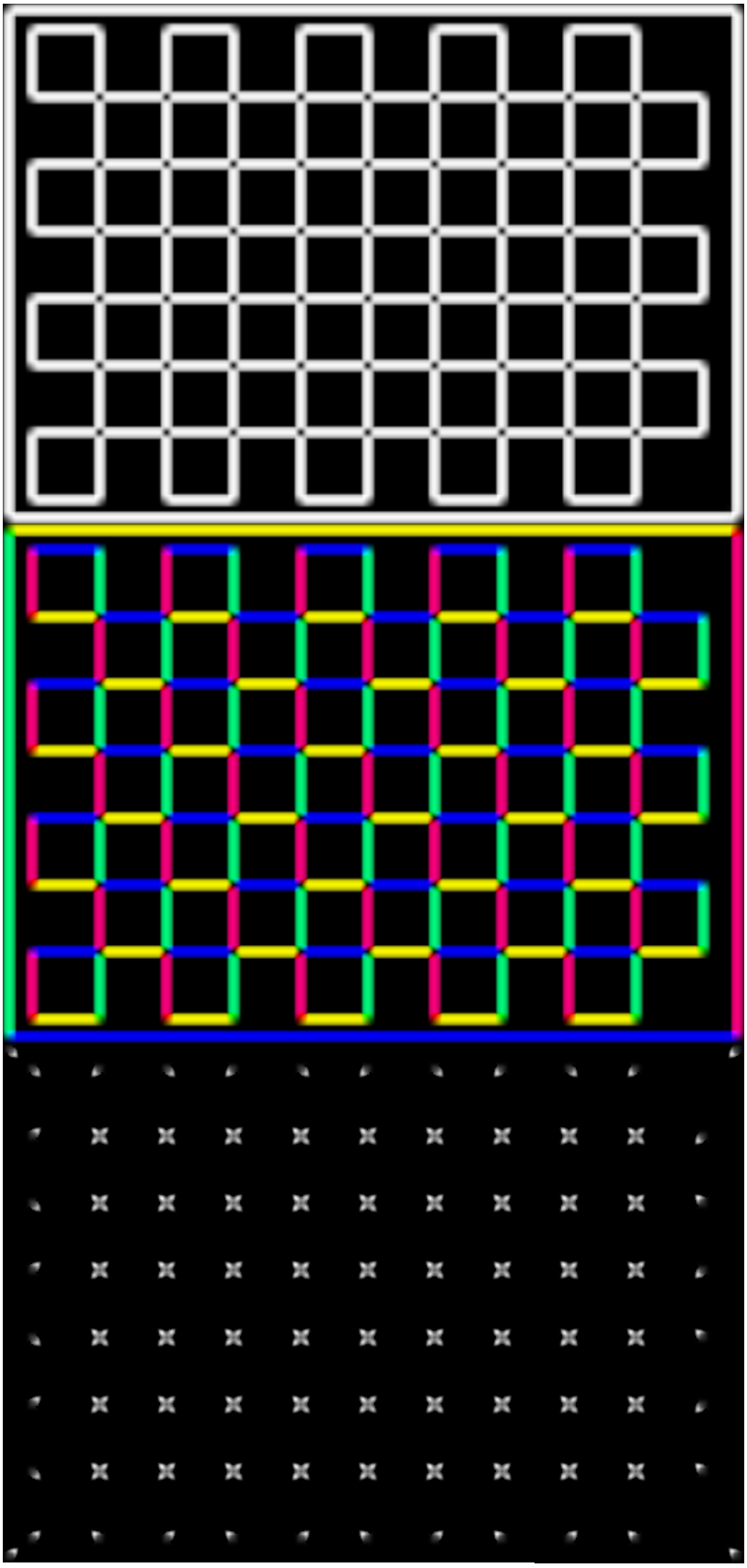
scale

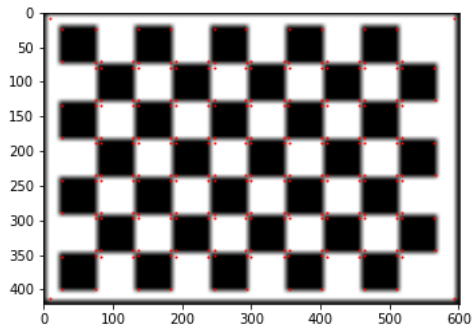
```
# Chess scale
chess_scale = cv2.resize(chess_img, (0,0), fx=0.5, fy=0.5)
notredame_scale = cv2.resize(notredame_a, (0,0), fx=0.5, fy=0.5)
```



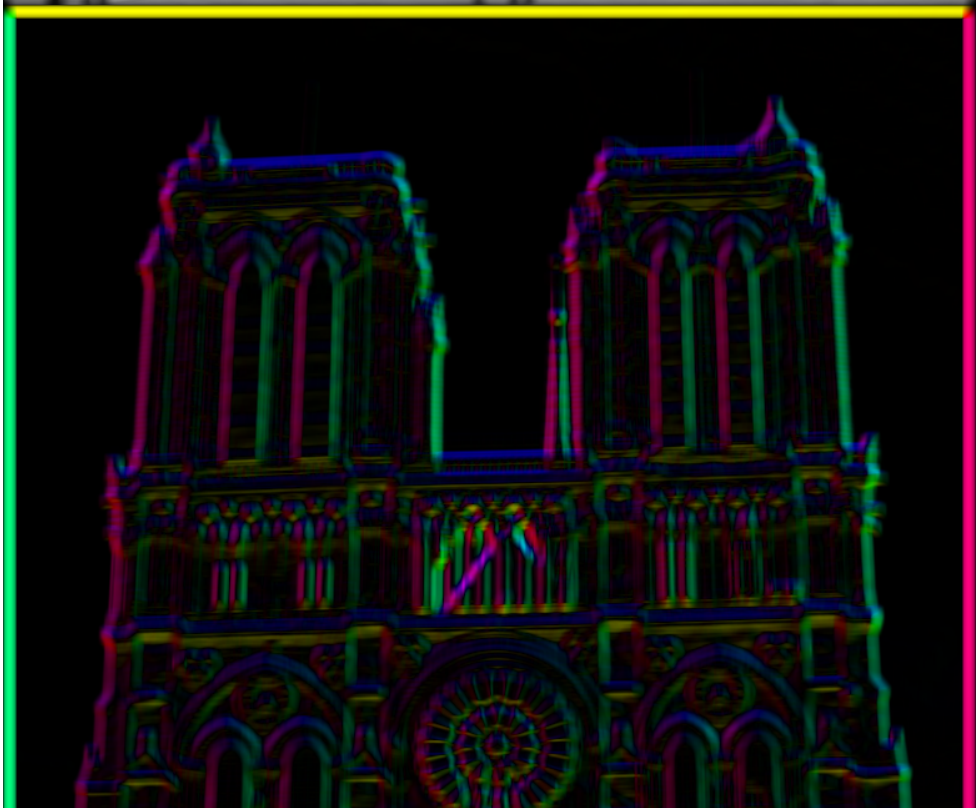


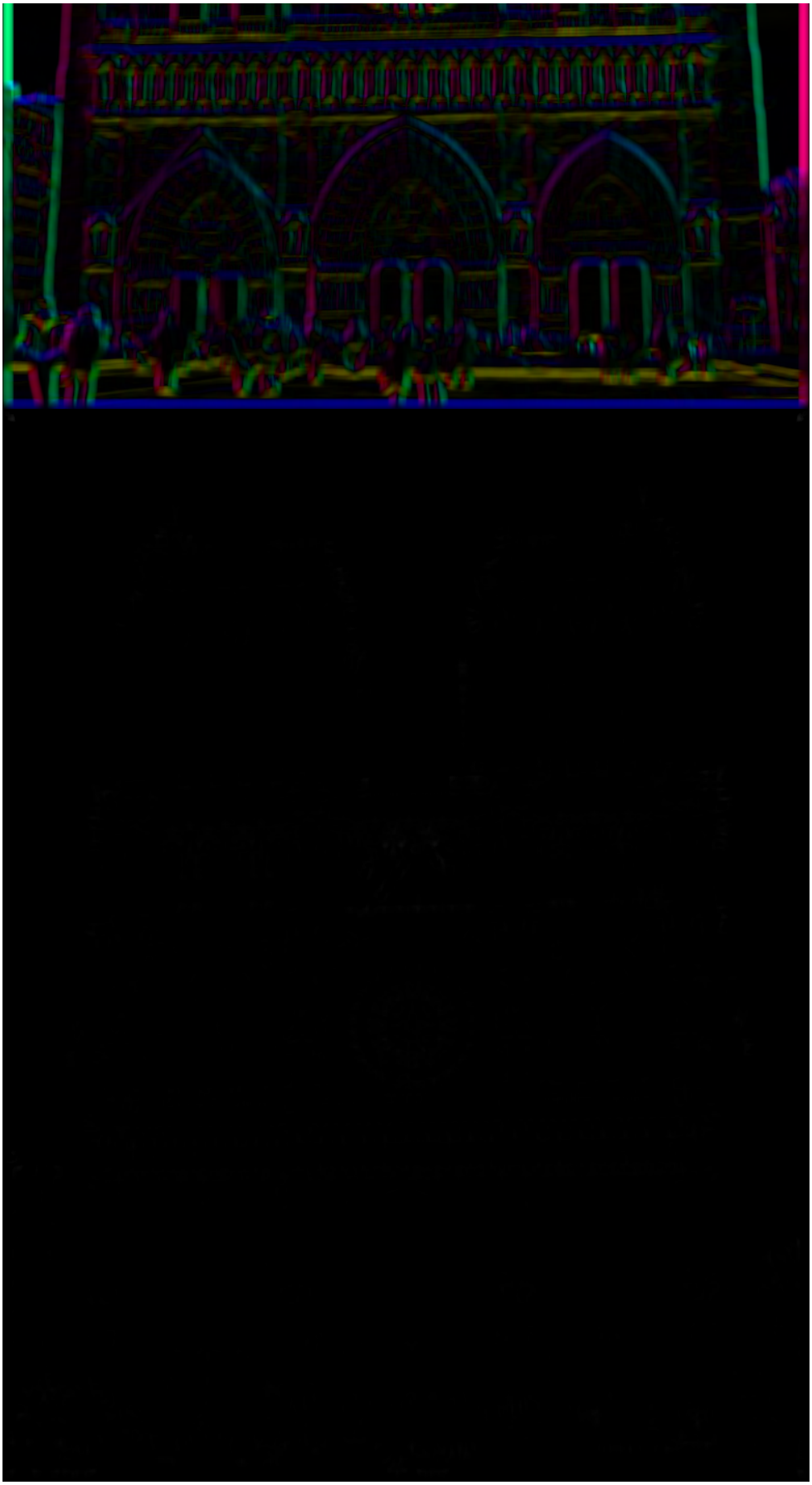
```
# Chess scale corner detection
dx_c_s, dy_c_s, magnitude_c_s, theta_c_s, direction_c_s = sobel(chess_scale)
Axx_c_5_s,Axy_c_5_s,Ayy_c_5_s,responce_c_5_s, landa2_c_5_s = structure_matrix(size = 5,
r,c =  NMS(responce_c_5_s)
```

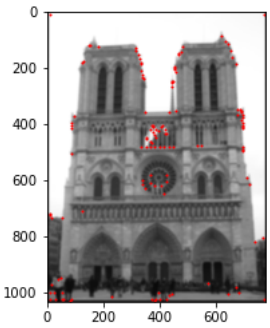



0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600

```
# notredame scale corner detection
dx_a_s, dy_a_s, magnitude_a_s, theta_a_s, direction_a_s = ![](https://i.imgur.com/Zsl1G)
Axx_a_5_s,Axy_a_5_s,Ayy_a_5_s,responce_a_5_s, landa2_a_5_s = structure_matrix(size = 5,
r,c =  NMS(responce_a_5_s)
```







Discussion

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

- 從上面實作結果能夠觀察到，Gaussian filter其window size 越大，對於平滑效果與去躁化效果更為顯著，同時模糊效果也更加明顯。
- 然而在 detected edge 方面卻恰其相反，從結果得知，windoe size越小越能夠有較清晰的邊緣化偵測。

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

- 相比3X3 window size 下，5X5 window size 最終呈現的 2X2 pixel 值包含更大範圍的運算，這也導致之後所做的 harris corner response 能夠更加明顯的偵測到gradient的劇烈變化而發現corner

c. The effect of non-maximal suppression.

- 再corner detection時，會有許多點都有很高的gradient，NMS能夠透過threshold先將過低的corner responce淘汰掉，再透過local中找尋最大值
- 因此，當threshold越高時，corner被NMS篩選出來的就會越少；同時若取的local size越高，也會因為在更大的範圍選maximum導致corner 適量減少

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

- 從實驗結果來看，在相同的參數下，scale後的圖片其 corner 會相對減少許多
- 因此，Harris detector 為 scale-invariant

1-2

(a) SIFT interest point detection

- Apply SIFT interest point detector (functions from OpenCV) to the following two images
- Adjust the related thresholds in SIFT detection such that there are around 100 interest points detected in each image .
- Plot the detected interest points on the corresponding images

```
### HW 1-2 ###
"""
2(A)
"""
similarity, img3, kp1, des1, kp2, des2, img1, img2 = SIFT(notredame_a, notredame_b, notr
```

```
[INFO] la_notredame of keypoints detected: 200
[INFO] lb_notredame of keypoints detected: 559
[INFO] feature vector shape: (200, 128)
[INFO] feature vector shape: (559, 128)
```

(b) SIFT feature matching

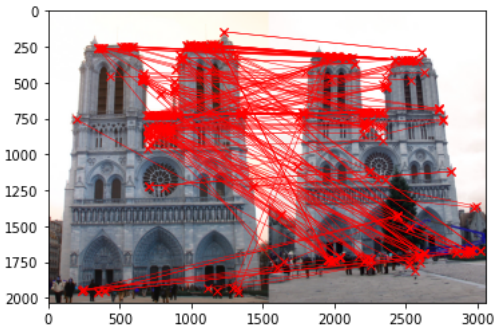
- Compare the similarity between all the pairs between the detected interest points from each of the two images based on a suitable distance function between two SIFT feature vectors
- Implement a function that finds a list of interest point correspondences based on nearest-neighbor matching principle
- Plot the point correspondences (from the previous step) overlaid on the pair of original images

```
"""
2(b)
"""
# Similarity
# Calculated by SIFT function
# Similarity = a.b / |a| * |b|
# match function
matches = match(similarity, kp1, kp2)
```

```
[[0.6024242  0.56047851 0.83430612 ... 0.73799318 0.76495177 0.76144153]
 [0.65043896 0.49992377 0.73422754 ... 0.72099304 0.73266935 0.7260772 ]
 [0.48084068 0.57046998 0.79172659 ... 0.7185362  0.82158959 0.82026953]
 ...
 [0.3757976  0.37014523 0.26550868 ... 0.20637999 0.19979922 0.18715909]
 [0.36106679 0.33766204 0.29832292 ... 0.22490242 0.22682631 0.22453961]
 [0.35434353 0.33512551 0.25046563 ... 0.19918947 0.18448707 0.17338404]]
The number of matches: 200
```

Plot the image

```
plot_matches(matches, img3)
```



© Discussion

- Discuss the cases of mis-matching in the point correspondence
- Discuss and implement possible solutions to reduce the mis-matches, and show your results.

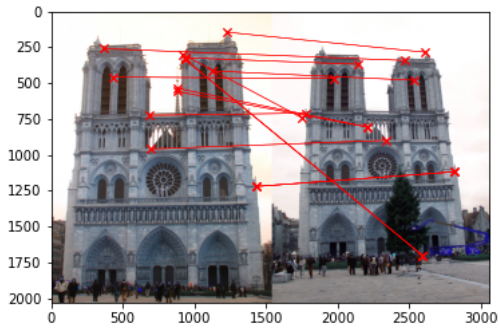
```
"""
2(c) Discuss the cases of mis-matching in the point correspondence
"""
```

```
'\n2(c) Discuss the cases of mis-matching in the point correspondence\n'
```

- Those which are matching better

```
"""
2(c) Discuss and implement possible solutions to reduce the mis-matches, and show your r
"""
matches = improve_match(similarity, kp1, kp2)
plot_matches(matches, img3)
```

```
The number of matches: 28
```



Discussion

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

- 在許多相似的區塊，例如左右兩邊的塔，有許多點皆被錯誤的match，像是左邊的塔連到右邊的塔，或是第二根短柱連到第五根短柱
- 同時也有許多黑色區塊被互相配對但皆不是正確的匹配

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

- 再經過k個最近鄰居的探索後，得到距離最短的k個描述特徵距離。
- 我實作了ratio test 的作法
- 我們將第二最短距離以上的點視為隨機配對結果，意即視為negative的配對
- 倘若最短距離與negative差距大於某個threshold，代表此最短距離與negative相距甚大，我們就將此點視為示好的match
- 反之，若距離不夠遠，意味著此點與negative蠻接近的，我們就是此點為不好的match並且不採用此點
- 從結果來看，非常大幅度的去掉了mis-matching的點，留下的大部分皆為正確的配對。然而也有許多正確的match被犧牲了