LAB REPORT FOR CHRISTOPHE BOBDA'S GROUP

WEEK 2

Xianhan Li

INTRODUCTION

This report includes 2 tasks: The first one is about edges detection and corner detector, I have set up a Canny Edge Detection with Sobel Operator, and used Harris Corner Detector; The second one is about comparing SIFT and SURF, including comparing the computation time, the total number of keypoints found and the percentage of matches.

TASK1

STEP1: Gaussian filtering is applied to smooth the image with the aim of removing noise.

Gaussian filter is the discretization of Gaussian function, the corresponding transverse and longitudinal index of the filter into the Gaussian function, so as to obtain the corresponding value.

```
# Gaussian filter (5*5, σ = 1.4)
k = 5 // 2
gaussian = np.zeros([5, 5])
for i in range(5):
    for j in range(5):
        gaussian[i, j] = np.exp(-((i - k) ** 2 + (j - k) ** 2) / (2 * 1.4 ** 2))
gaussian /= 2 * np.pi * 1.4 ** 2
# Batch Normalization
gaussian = gaussian / np.sum(gaussian)
# Use Gaussian Filter
W, H = image.shape
gaussian_image = np.zeros([W - k * 2, H - k * 2])
for i in range(W - 2 * k):
    for j in range(H - 2 * k):
        # convolution operation
        gaussian_image[i, j] = np.sum(image[i:i + 5, j:j + 5] * gaussian)
gaussian_image = np.uint8(gaussian_image)
cv.imshow("Smooth", gaussian_image)
```

STEP2: Calculate gradient strength and direction.

Look for the edge, that is, the location where the intensity of the gray changes the most (a black edge and a white edge in the middle is the edge, which has the largest change in gray value). In the image, the gradient is used to represent the change degree and direction of the gray value.

```
# Use Sobel to compute gradients and direction
W_0, H_0 = gaussian_image.shape
Gx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
Gy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
gradients = np.zeros([W_0 - 2, H_0 - 2])
directions = np.zeros([W_0 - 2, H_0 - 2])
for i in range(W_0 - 2):
    for j in range(H_0 - 2):
        dx = np.sum(gaussian_image[i:i + 3, j:j + 3] * Gx)
        dy = np.sum(gaussian_image[i:i + 3, j:j + 3] * Gy)
        gradients[i, j] = np.sqrt(dx ** 2 + dy ** 2)
        if dx == 0:
            directions[i, j] = np.pi / 2
        else:
            directions[i, j] = np.arctan(dy / dx)
gradients = np.uint8(gradients)
```

STEP3: Apply non-maximum suppression technique NMS to eliminate edge false detection.

The aim is to make a blurred border sharp. In popular terms, the maximum value of the gradient intensity on each pixel is retained, and the other values are deleted.

```
# Non-Maximum Suppression
W_1, H_1 = gradients.shape
nms = np.copy(gradients[1:-1, 1:-1])
for i in range(1, W_1 - 1):
    for j in range(1, H_1 - 1):
        theta = directions[i, j]
        weight = np.tan(theta)
        if theta > np.pi / 4:
            d1 = [0, 1]
            d2 = [1, 1]
            weight = 1 / weight
        elif theta >= 0:
            d1 = [1, 0]
            d2 = [1, 1]
        elif theta >= - np.pi / 4:
            d1 = [1, 0]
            d2 = [1, -1]
            weight *= -1
        else:
            d1 = [0, -1]
            d2 = [1, -1]
            weight = -1 / weight
        g1 = gradients[i + d1[0], j + d1[1]]
        g2 = gradients[i + d2[0], j + d2[1]]
        g3 = gradients[i - d1[0], j - d1[1]]
        g4 = gradients[i - d2[0], j - d2[1]]
        grade_count1 = g1 * weight + g2 * (1 - weight)
        grade_count2 = g3 * weight + g4 * (1 - weight)
        if grade_count1 > gradients[i, j] or grade_count2 > gradients[i, j]:
            nms[i - 1, j - 1] = 0
```

STEP4: Apply the method of double threshold to determine possible boundaries.

```
# Double Threshold
visited = np.zeros_like(nms)
output_image = nms.copy()
W_2, H_2 = output_image.shape
def dfs(i, j):
    if i >= W_2 or i < 0 or j >= H_2 or j < 0 or visited[i, j] == 1:
        return
    visited[i, j] = 1
    if output_image[i, j] > 50:
        output_image[i, j] = 255
        dfs(i - 1, j - 1)
        dfs(i - 1, j)
        dfs(i - 1, j + 1)
        dfs(i, j - 1)
        dfs(i, j + 1)
        dfs(i + 1, j - 1)
        dfs(i + 1, j)
        dfs(i + 1, j + 1)
    else:
        output_image[i, j] = 0
```

```
for w in range(W_2):
    for h in range(H_2):
        if visited[w, h] == 1:
            continue
        if output_image[w, h] >= 100:
            dfs(w, h)
        elif output_image[w, h] <= 50:
            output_image[w, h] = 0
            visited[w, h] = 1
for w in range(W_2):
    for h in range(H_2):
        if visited[w, h] == 0:
            output_image[w, h] = 0
cv.imshow("outputImage", output_image)
```

STEP5: Apply Harris Corner Detector to identify features.

```
# Harris Corner Detector
window_size=3
k=0.04
threshold=0.1
dx = cv.Sobel(output_image, cv.CV_64F, 1, 0, ksize=3)
dy = cv.Sobel(output_image, cv.CV_64F, 0, 1, ksize=3)
Ixx = dx * dx
Iyy = dy * dy
Ixy = dx * dy
Sxx = cv.GaussianBlur(Ixx, (window_size, window_size), 0)
Syy = cv.GaussianBlur(Iyy, (window_size, window_size), 0)
Sxy = cv.GaussianBlur(Ixy, (window_size, window_size), 0)
det_M = Sxx * Syy - Sxy * Sxy
trace_M = Sxx + Syy
R = det_M - k * trace_M * trace_M
corner_points = np.zeros_like(R)
corner_points[R > threshold * R.max()] = 1
image_with_corners = cv.cvtColor(output_image, cv.COLOR_GRAY2BGR)
image_with_corners[corner_points == 1] = [0, 0, 255]
```

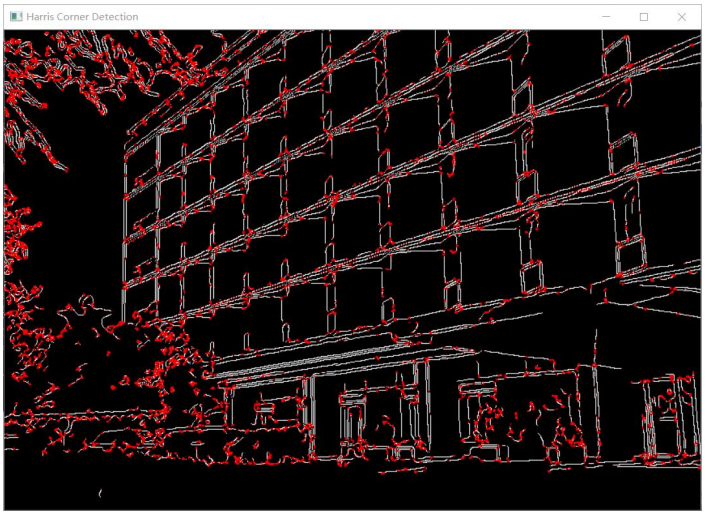
Results of Implement:

Original image:

Smoothed image:

Edge detected image:

Corner detected image:



TASK2

STEP1: Read in the images and change them to grayscale images.

```
img_1 = cv2.imread('15_1.png')
img_2 = cv2.imread('15_2.png')
gray_1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2GRAY)
gray_2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2GRAY)
```

STEP2: Build SIFT or SURF objects, then detect and compute the key points and descriptor.

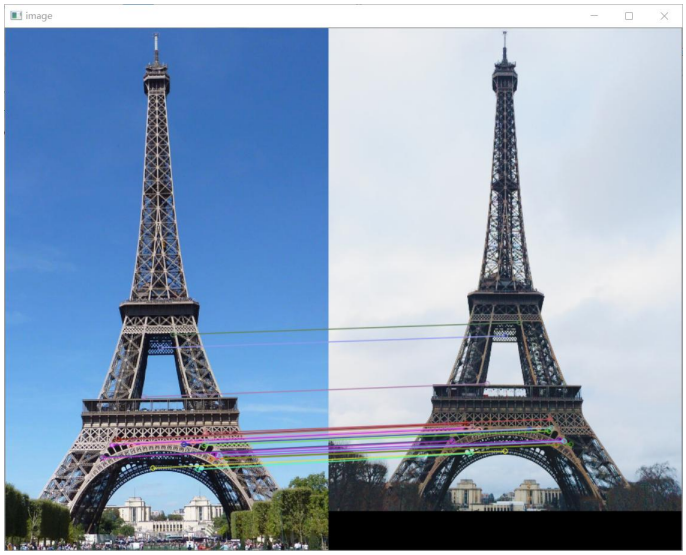
```
# SIFT特征计算
sift = cv2.xfeatures2d.SIFT_create()
# sift = cv2.xfeatures2d.SURF_create()
psd_kp1, psd_des1 = sift.detectAndCompute(gray_1, None)
psd_kp2, psd_des2 = sift.detectAndCompute(gray_2, None)
print(len(psd_kp1))
print(len(psd_kp2))
```

STEP3: Use FLANN to match the key points and draw them out on the images.

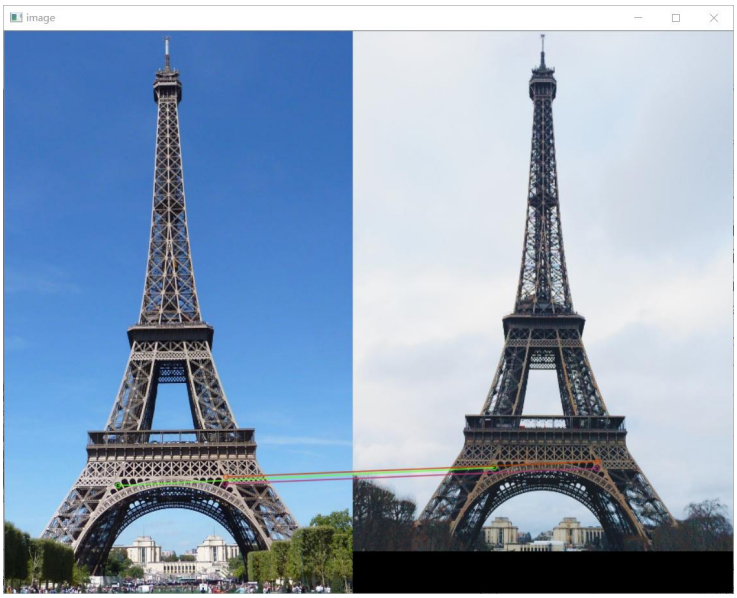
```python
# Flann特征匹配
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(psd_des1, psd_des2, k=2)
goodMatch = []
for m, n in matches:
    if m.distance < 0.50 * n.distance:
        goodMatch.append(m)
print(len(goodMatch))

# 增加一个维度
goodMatch = np.expand_dims(goodMatch, 1)
print(goodMatch[:])
img_out = cv2.drawMatchesKnn(img_1, psd_kp1,
                             img_2, psd_kp2,
                             goodMatch[:], None, flags=2)
```

Results of Implement:

SIFT:

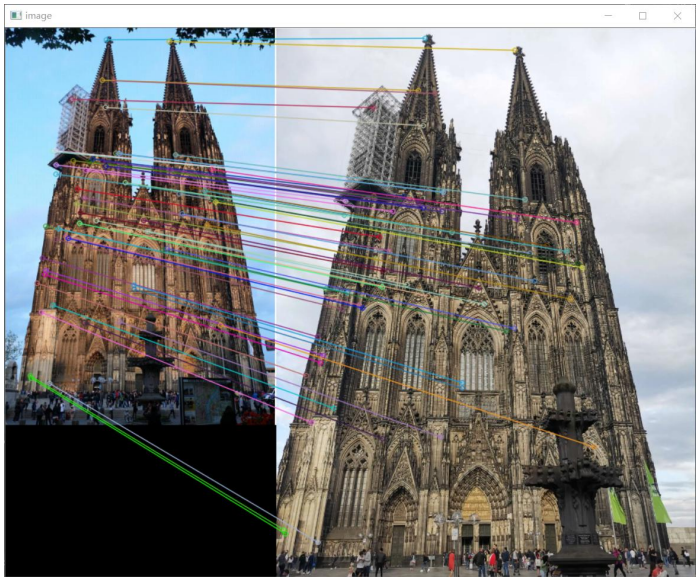


SURF:

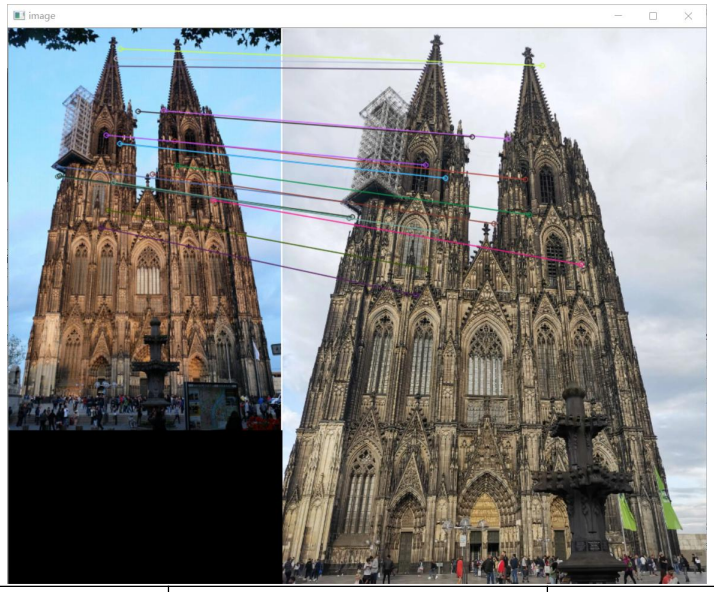


	SIFT	SURF
Time cost	0.13660812377929688 s	0.3820044994354248 s
Total key points (small amount)	1385	1573
Matched key points	26	3
The percentage of matches	1.88%	0.191%

SIFT:

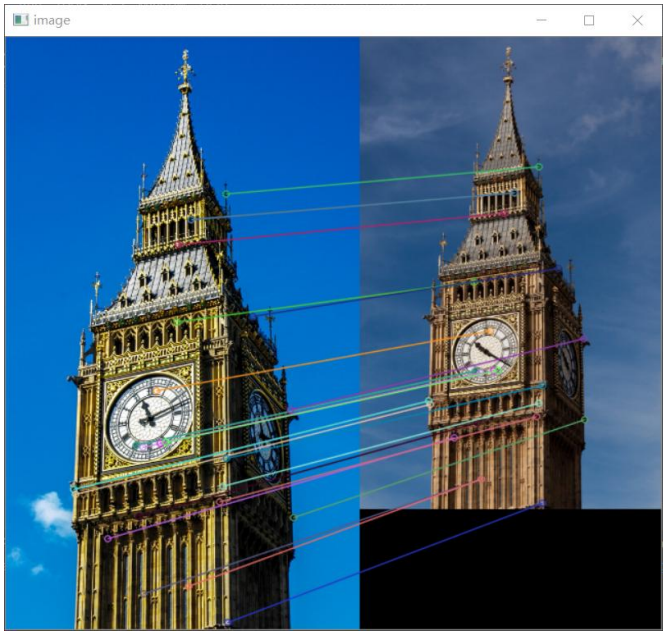


SURF:

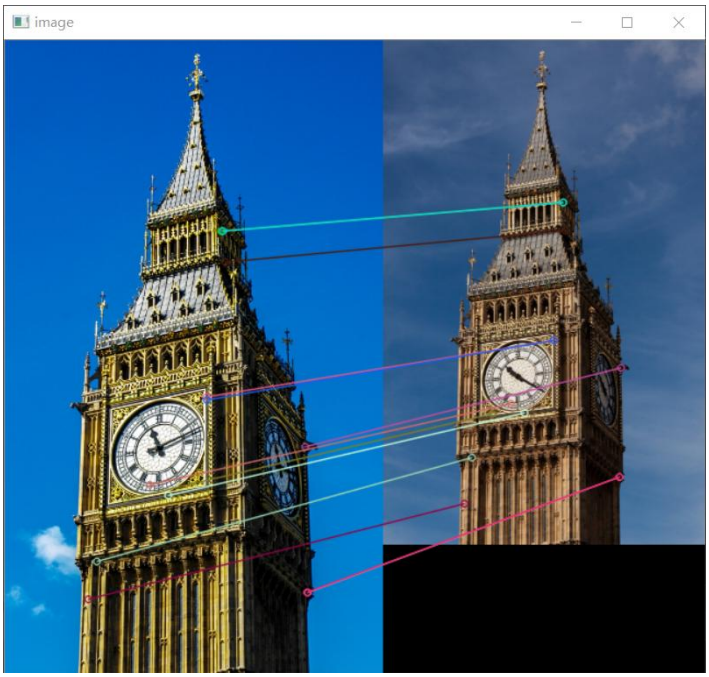


	SIFT	SURF
Time cost	0.2161414623260498 s	0.4819209575653076 s
Total key points (small amount)	2493	2371
Matched key points	62	18
The percentage of matches	2.49%	0.759%

SIFT:

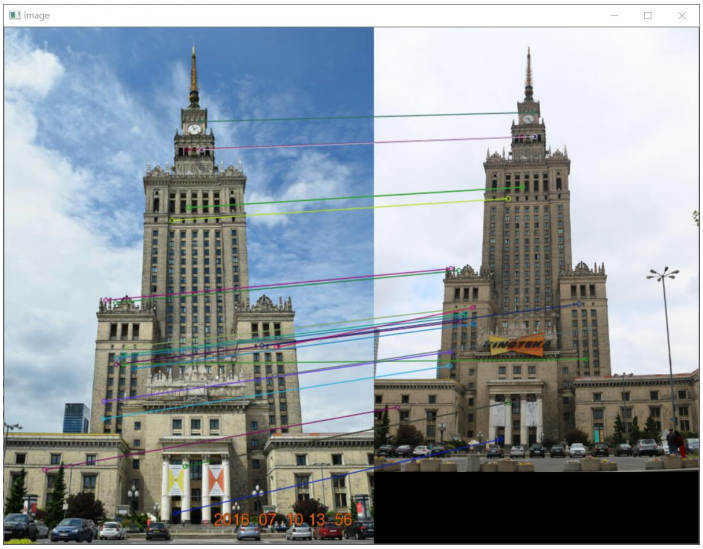


SURF:

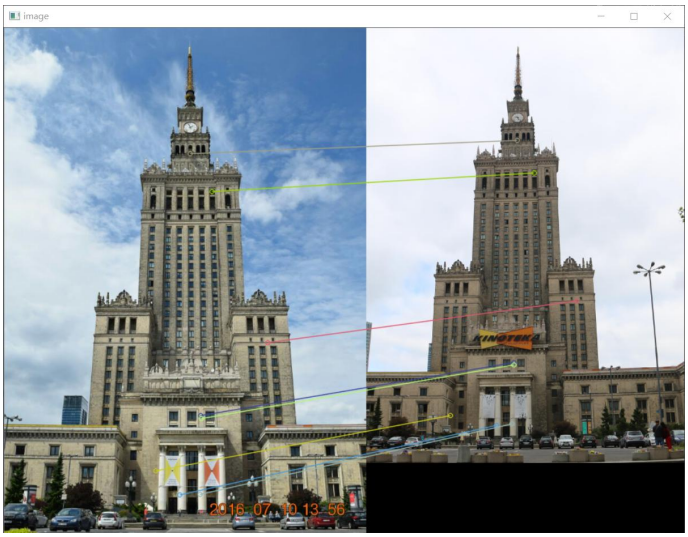


	SIFT	SURF
Time cost	0.08487749099731445 s	0.36838197708129883 s
Total key points (small amount)	738	1018
Matched key points	25	13
The percentage of matches	3.39%	1.28%

SIFT:

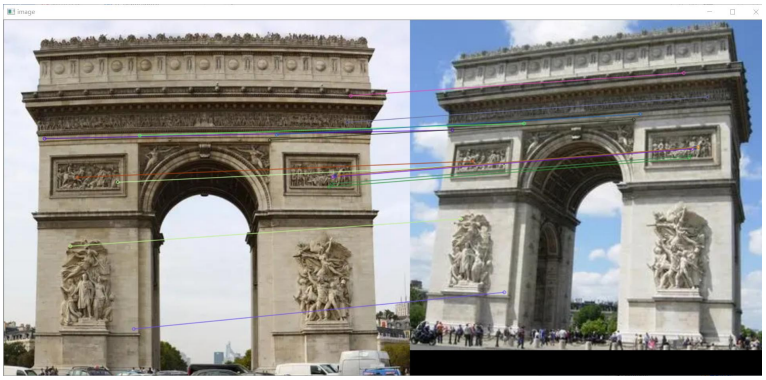


SURF:

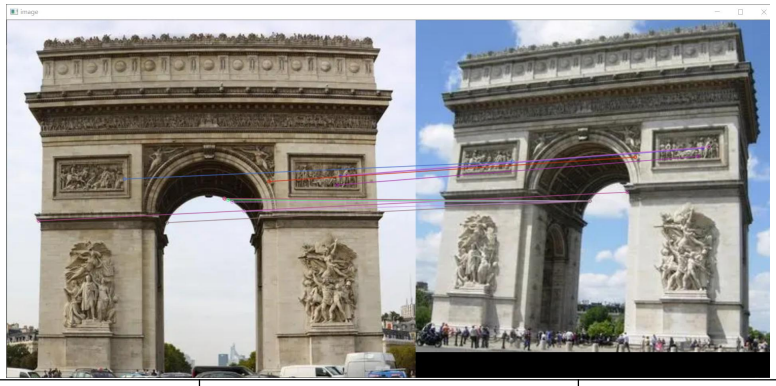


	SIFT	SURF
Time cost	0.16718769073486328 s	0.53013014793396 s
Total key points (small amount)	1594	2054
Matched key points	23	8
The percentage of matches	1.44%	0.389%

SIFT:



SURF:



Time cost	0.3048267364501953 s	0.5895392894744873 s
Total key points (small amount)	3313	3654
Matched key points	15	12
The percentage of matches	0.453%	0.328%

Conclusion

From the tables, I can find that SIFT always has batter performance in terms of time. SIFT also has batter percentages of matches. However, for the most time, SURF algorithm can detect more key points.