RANSAC, Panorama

1.

[Code]

```
## START
## the following is just a placeholder to show you the output format
'''
num = 5
matched_pairs = [[i, i] for i in range(num)]
'''
# 모든 가능한 match를 비교하기 위해서
# print(descriptors1.shape, descriptors2.shape) # (694, 128), (579, 128)
matched_pairs = []

for i in range(descriptors1.shape[0]):
    temp = []
    for j in range(descriptors2.shape[0]):
        # 두 descriptors의 내적 값
        value = np.dot(descriptors1[i], descriptors2[j])
        temp.append([j, value])
    # 두 descriptors 내적 값을 기준으로 내림차순 정렬
    temp = sorted(temp, key = lambda x : -x[1])
    # math.acos를 이용해서 max, secondmax를 나누어 keypoint가 매칭되었는지 다시 확인(잘 매칭되었다
    if math.acos(temp[0][1])/math.acos(temp[1][1]) < threshold:
        matched_pairs.append([i, temp[0][0]])

## END
```

모든 가능한 match를 비교하기 위해서 전체 이중 loop를 통해 전체 경우를 고려했다.

max, secondmax를 이용해서 key point가 제대로 매칭되었는지를 다시 확인해서 잘 매칭된 경우에만 넣어주었다.

1) Threshold 변경해보면서 어떤 것들이 달라지는지 확인

threshold : 0.8

threshold : 0.6

threshold가 0.8인 경우에 0.6일때에 비해서 개수가 늘어남에 따라 걸러지지 않고 남은 outlier가 많이 있는 것을 확인할 수 있다.

2)

- scene basmati

- scene book

- scene box

세 경우 다 확인

2.

[Code]

```
## START
# match들 중 랜덤하게 무작위로 고름, 나머지와 유사한지를 판단.
# 10번 반복해서 제일 큰 match를 갖는 결과를 리턴
# keypoint1, 2 [x, y, orient_agreement, scale_agreement]

# 저장 후 리턴할 largest_set, 갱신을 위한 largest_num
largest_set = [-1, -1]
largest_num = 0
# print(matched_pairs[1]) # [184, 15].

for i in range(10):
    # matched_pair에서 랜덤하게 고름. [(i, j)] format이 저장되어 있다.
    k = random.choice(matched_pairs)
    print(k)
    x = k[0]
    y = k[1]
    print(keypoints1[x])
    print(keypoints2[y])
    num = 0

    for kx, ky in matched_pairs:
        # radian to degree
        degree1 = np.degrees(keypoints1[x][3] - keypoints1[kx][3])
        degree2 = np.degrees(keypoints2[y][3] - keypoints2[ky][3])
        # print(degree1)
        # print(degree2)
        scale1 = abs(keypoints1[x][2] - keypoints1[ky][3])
        scale2 = abs(keypoints2[y][2] - keypoints2[ky][3])
        # print(scale1)
        # print(scale2)

        # 1) degree1 - 30 < degree2 < degree1 + 30 - orient_agreement = 30
        # 2) scale1*0.5 < scale2 < scale1*1.5 - scale_agreement = 0.5
        if abs(degree1-degree2) % 360 < orient_agreement and scale1*scale_agreement < scale2 <
            num += 1
    if largest_num < num:
        largest_num = num
        largest_set = [x, y]
    print(largest_num, largest_set)

## END
```

matched_pair에서 랜덤하게 10번 골라서 orientation, degree의 범위를 고려하여 내부에 존재하는 것을

RANSAC의 후보로 넣고, inlier가 최대가 되는 것을 고른다.

scene - basmati

Case 1 : orient_agreement = 50, scale_agreement = 1.5

-> 10개 [478, 71]

Case 2 : orient_agreement = 30, scale_agreement = 0.5

-> 17개 [184, 15]

Case 3 : orient_agreement = 60, scale_agreement = 0.4

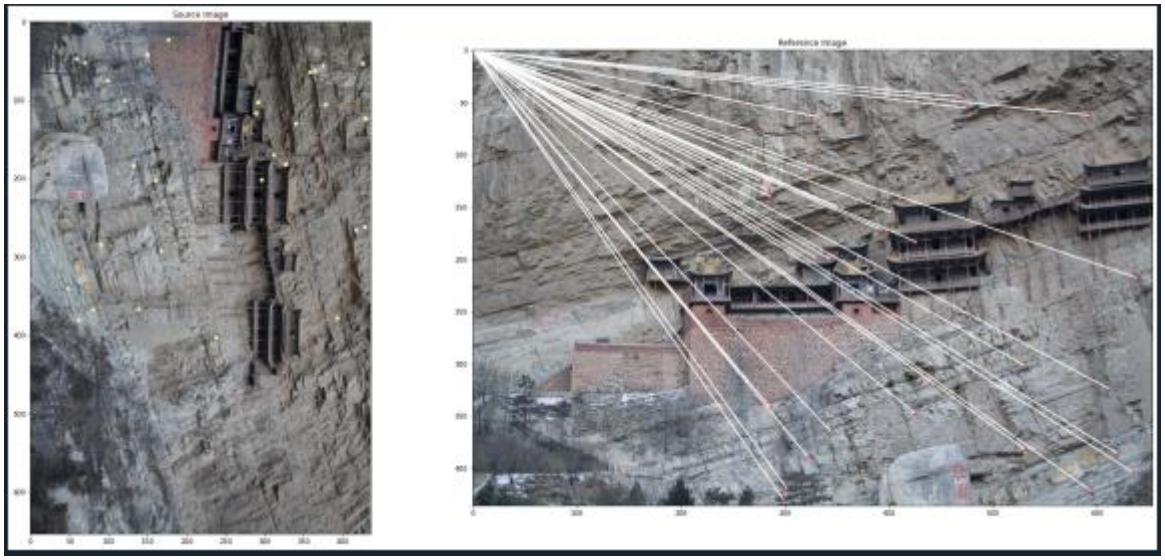
-> 16개 [565, 56]

3.

[Code]

```
# START
# 0으로 나누지 않도록 constant_value를 1e-10으로 설정한다.
'''
xy_points[29][2] =  1
xy_points_out = xy_points.dot(h)
xy_points_out = np.delete(xy_points_out, 2, 1)
'''

# 위, 아래, 왼은 padding 0, 오른쪽은 padding 1개. constant_value는 1
xy_points = np.pad(xy_points, ((0, 0), (0, 1)), constant_values = 1)
xy_points_out = xy_points.dot(h)

for i in range(30):
    # 나눠야 할 값(w)가 0일 경우 1e-10으로 설정.
    if xy_points_out[i][2] == 0:
        xy_points_out[i][2] = 1e-10
    # X, Y각각을 W로 나눠줘야 한다.
    xy_points_out[i][0] = xy_points_out[i][0] / xy_points_out[i][2]
    xy_points_out[i][1] = xy_points_out[i][1] / xy_points_out[i][2]

xy_points_out = np.delete(xy_points_out, 2, 1)

# END
return xy_points_out
```

맨 오른쪽에 열을 1개 추가해주고 해당하는 것과 homography를 연산함으로써 xy_points_out 리턴값을 얻는다. 이 때, 0으로 나눠서 발생하는 오류를 없애기 위해서 분모(W)가 0이 될 수 있는 경우에는 1e-10으로 처리해주었다.



4.

[Code]

```
# START
# num_iter에 해당하는 수만큼 iteration을 수행
max_num = 0
for i in range(num_iter):
    # 4 matches를 랜덤하게 뽑는다.
    num = 0

    src_arr = []
    ref_arr = []
    for _ in range(4):
        # idx를 랜덤하게 뽑는다.
        idx = random.randrange(len(xy_src))
        src_arr.append(xy_src[idx])
        ref_arr.append(xy_ref[idx])

    homography =
    for j in range(len(xy_src)):
        kx, ky = np.dot(xy_src[j], homography)
        val = math.sqrt((xy_ref[j][0] -kx)**2 + (xy_ref[j][1] - ky)**2)
        if val < tol:
            num += 1
    if num > max_num:
        h = homography

# 1. 랜덤하게 4개 뽑아서 xy_src -> xy_ref로 h를 구하고,
# 2. h로 projected된 것이랑 실제 xy_ref의 거리(distance)를 구하고,
# 3. tol보다 적은거 개수가 최댓값이 되는 h를 리턴한다.
```

[미완료]