Introduction to Computer Vision HW04

- RANSAC and Panorama Stitching -

2021-1학기 컴퓨터비전개론 059분반 201724437 김재현

Part 1. SIFT Keypoint Matching

1. FindBestMatches

찾고자 하는 source 이미지의 descriptor(여기서는 book, box 등)를 기준으로 모든 reference 이미지와의 각도 차이를 구한 뒤, 이 중 가장 작은 각도와 두 번째로 작은 각도의 비율이 threshold보다 작은 경우에만 해당 descriptor들의 인덱스를 추가하도록 하였습니다.

이 때 threshold를 높일수록 outlier가 많이 관찰되었고, 반대로 threshold를 낮추면 outlier가 줄어들지만 동시에 inlier의 숫자도 감소하는 것을 관찰할 수 있었습니다. 또한 0.1 단위로 threshold를 조정한 결과 과제에서 제시한 0.6보다 작은 0.5일 때부터 outlier의 숫자가 10개 이내로 줄어듦을 확인할 수 있었습니다. 해당 코드와 threshold = 0.5로 생성한 두 결과 이미지는 아래와 같습니다.

Line 55~83

```
def FindBestMatches(descriptors1, descriptors2, threshold):
    assert isinstance(descriptors1, np.ndarray)
    assert isinstance(descriptors2, np.ndarray)
    assert isinstance(threshold, float)
    ## START

    matched_pairs = []

    for j, desc2 in enumerate(descriptors2):
        angles = np.arccos(np.dot(descriptors1, desc2))
        sorted_angles = sorted(angles)
        if((sorted_angles[0]/sorted_angles[1]) <= threshold):
            best_match_index = np.where(angles == sorted_angles[0])[0][0]
            matched_pairs.append([best_match_index, j])
    ## END
    return matched_pairs
```

>>

그림 1 – scene-book.png

그림 2 – scene-box.png

2. RANSACFilter

각 match를 이루는 keypoint들의 속성 중 각도의 차이, scale의 비율을 계산한 뒤 이 두 지표가 다른 match들과 얼마나 차이가 나는지(두 match 간의 지표 차이가 threshold 안에 존재하는지)를 판단하여, outlier들을 필터링할 수 있었습니다. 이 때 orient_agreement의 값을 증가시킬 경우에는 다른 match line들과 큰 각도 차이를 보이는 outlier들이 증가하였습니다. scale_agreement의 값을 증가시킬 경우에는 source와 reference 이미지의 scale 비율을 벗어나는 keypoint들로 이루어진 outlier들이 많이 관찰되었습니다(예: 아래의 library 이미지에서 큰 창문의 그림자와 작은 외부 아치 장식의 그림자를 match시킨 경우 등).

또 ratio_threshold를 최대한 높이기 위해 orient_agreement, scale_agreement를 조정한 결과 ratio_thres=0.8, orient_agreement=10, scale_agreement=0.1의 값이 가장 적절한 것을 확인되었습니다. 해당 코드와 위의 세 가지 threshold 값으로 생성된 결과 이미지는 아래와 같습니다.

Line 9~51

```
def RANSACFilter(
      matched_pairs, keypoints1, keypoints2,
      orient_agreement, scale_agreement):
   assert isinstance(matched_pairs, list)
   assert isinstance(keypoints1, np.ndarray)
   assert isinstance(keypoints2, np.ndarray)
   assert isinstance(orient_agreement, float)
   assert isinstance(scale_agreement, float)
   ## START
   largest_set = []

   for _ in range(10):
      initial_orient_diff = abs(keypoints2[selected_match[1]][3]-
                            keypoints1[selected_match[0]][3])
      initial_scale_change = keypoints2[selected_match[1]][2] /
                            keypoints1[selected_match[0]][2]
      candidate_set = []
      for match in matched_pairs:
         orient_diff = abs(keypoints2[match[1]][3]-keypoints1[match[0]][3])
         scale_change = keypoints2[match[1]][2] / keypoints1[match[0]][2]
         orient_cond = (abs (orient_diff - initial_orient_diff) <=
 math.radians(orient_agreement))
         scale_cond = (initial_scale_change*(1.0-scale_agreement) <=
scale_change <= initial_scale_change*(1.0+scale_agreement))
         if(orient_cond and scale_cond):
```

```
            candidate_set.append(match)

        if(len(candidate_set) > len(largest_set)):
            largest_set = candidate_set

    ## END
    assert isinstance(largest_set, list)
    return largest_set
```

>>

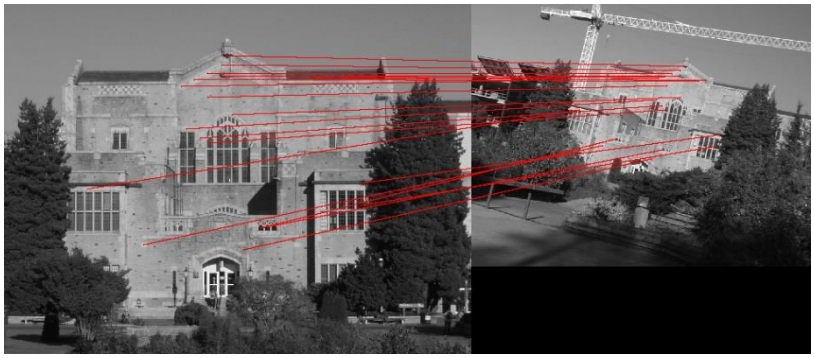

그림 3 – library-library2.png

Part 2. Panorama

1. Keypoint Projection

주어진 homography matrix의 transpose와 주어진 2d point의 좌표를 곱하여 project된 점들을 얻는 함수를 작성하였습니다. homography를 transpose한 이유는 주어진 점의 벡터 방향이 뒤집혀 있었기 때문이며, 이를 감안하여 homography를 뒤집는 동시에 곱하는 순서도 바꾸어 주었습니다. 해당 코드와 이 코드의 정상 작동을 보여주는 이미지는 아래와 같습니다.

Line 86~112

```
def KeypointProjection(xy_points, h):
    assert isinstance(xy_points, np.ndarray)
    assert isinstance(h, np.ndarray)
    assert xy_points.shape[1] == 2
    assert h.shape == (3, 3)
    # START
    xy_points_out = np.concatenate((xy_points, np.zeros((xy_points.shape[0],
 1))), axis=1)
    xy_points_out[:,2] = 1

    xy_points_out = np.dot(xy_points_out, h.T)

    xy_points_out[:,2][xy_points_out[:,2] == 0] = 1e-10

    xy_points_out = xy_points_out / xy_points_out[:,2:]
    xy_points_out = xy_points_out[:,0:2]

    # END
    return xy_points_out
```

>>

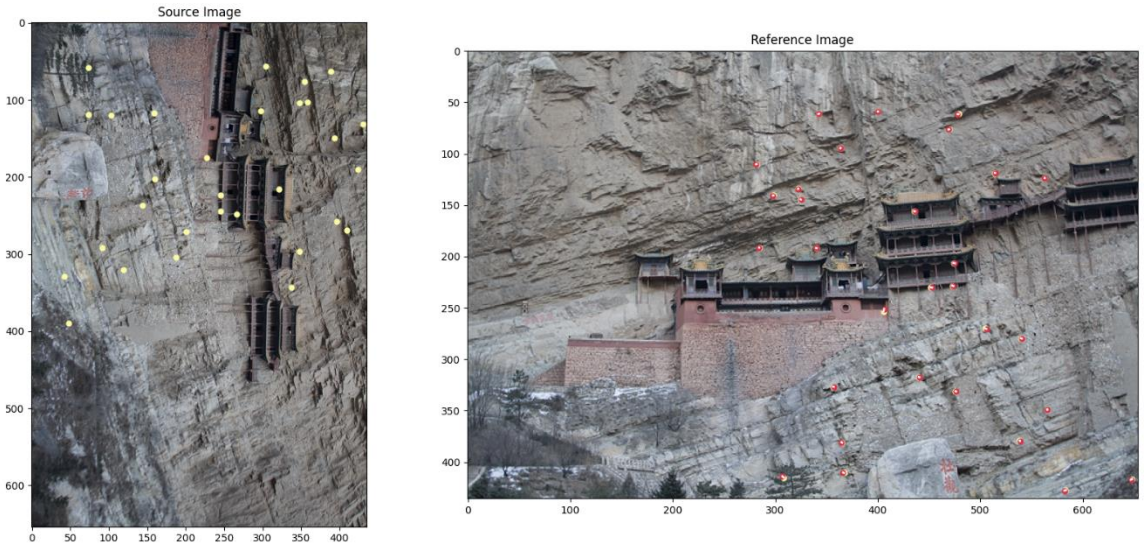

그림 4 – Hanging1-Hanging2.png

2. RANSAC Homography

이미지들로부터 얻어낸 match들에서 homography 후보를 얻어내고, 이 homography를 source 이미지의 keypoint에 적용했을 때 그 projection과 기존 reference 이미지의 차이가 제일 적은 것을 최종 homography로 삼는 과정을 다루고 있습니다. 이 때 4개의 점을 match로부터 무작위로 뽑은 다음 A 행렬을 만들고, A^TA의 고유벡터를 얻기 위해 특이값 분해(SVD)를 이용했습니다.

한편 이 함수의 매개변수인 num_iter는 후보 homography를 뽑아 적용해보는 횟수를 의미하며, 이 횟수가 늘어날수록 선택되는 keypoint의 조합이 늘어나 더 적합한 homography가 나온다는 것을 확인하였습니다. 다만 횟수에 따라 프로그램 시행시간도 같이 증가하므로, 여기서는 num_iter=200 정도로 설정했습니다.

tol은 projection point와 reference point 간의 거리에 대한 threshold 값으로써, 이 값을 줄일수록 최종 파노라마 이미지에서 두 이미지가 덜 분리되어 보였습니다. 여기서는 주어진 값 10보다 작은 tol=3을 적용하였습니다.

ratio_thres는 Part1.1에서 살펴본 바와 마찬가지로 FindBestMatches에서 적합한 match를 선정하는데 사용되었습니다. 이 값이 늘어날수록 outlier match가 늘어나 직관과 맞지 않는 이미지가 나왔습니다. 그러나 반대로 값을 너무 낮춘 경우에도 inlier의 수가 줄어들어 잘 연결되지 않은 이미지가 나왔습니다. 따라서 중간 값을 기준으로 적합한 값을 살펴본 결과 0.7일 때 가장 안정적인 결과를 확인할 수 있었습니다.

해당 코드와 num_iter=200, tol=3, ratio_thres=0.7로 설정하여 생성한 3개의 결과 이미지가 아래에 나와 있습니다.

Line 114~213

```
def RANSACHomography(xy_src, xy_ref, num_iter, tol):
    assert isinstance(xy_src, np.ndarray)
    assert isinstance(xy_ref, np.ndarray)
    assert xy_src.shape == xy_ref.shape
    assert xy_src.shape[1] == 2
    assert isinstance(num_iter, int)
    assert isinstance(tol, (int, float))
    tol = tol*1.0
```

```
    # START

    largest = 0

    for _ in range(num_iter):

        index_sample = random.sample(range(xy_ref.shape[0]),4)
        xy_src_sample = xy_src[index_sample]
        xy_ref_sample = xy_ref[index_sample]


        xy_ref_modified = np.repeat(xy_ref_sample.reshape(-1,1),3, axis=1)*-1
        xy_ref_modified[0::2,0] = 1
        xy_ref_modified[1::2,0] = 0
        xy_ref_modified[1::2,1] = 1
        xy_ref_modified[0::2,1] = 0
        xy_ref_modified = np.repeat(xy_ref_modified, 3, axis=1)

        xy_src_modified  =  np.concatenate((xy_src_sample,  xy_src_sample[:,1:]),
axis=1)
        xy_src_modified[:,2] = 1
        xy_src_modified = np.repeat(xy_src_modified, 2, axis=0)
        xy_src_modified = np.tile(xy_src_modified, 3)

        A = xy_ref_modified * xy_src_modified
        u,s,vt = np.linalg.svd(np.dot(A.T, A))

        h_cand = vt[s.argmin()].reshape((3,3))
        h_cand = h_cand / h_cand[2,2]


        well_proj_points = []
        xy_proj = KeypointProjection(xy_src, h_cand)

        point_diff = xy_proj - xy_ref
        proj_distances = np.hypot(point_diff[:,0:1], point_diff[:,1:])

        well_proj_points = xy_proj[np.where(proj_distances<=tol)]

        if(largest <= len(well_proj_points)):
            h = h_cand
            largest = len(well_proj_points)

    # END
    assert isinstance(h, np.ndarray)
    assert h.shape == (3, 3)
    return h
```

>>

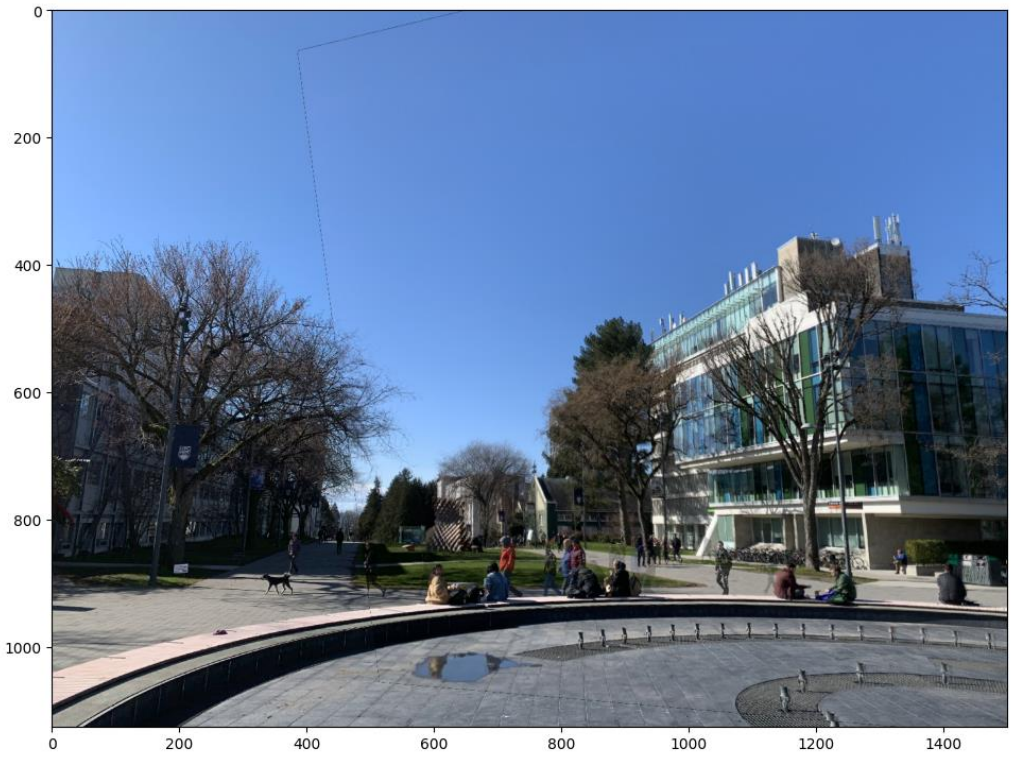

그림 5 – fountain04.png

그림 6 – garden034.png

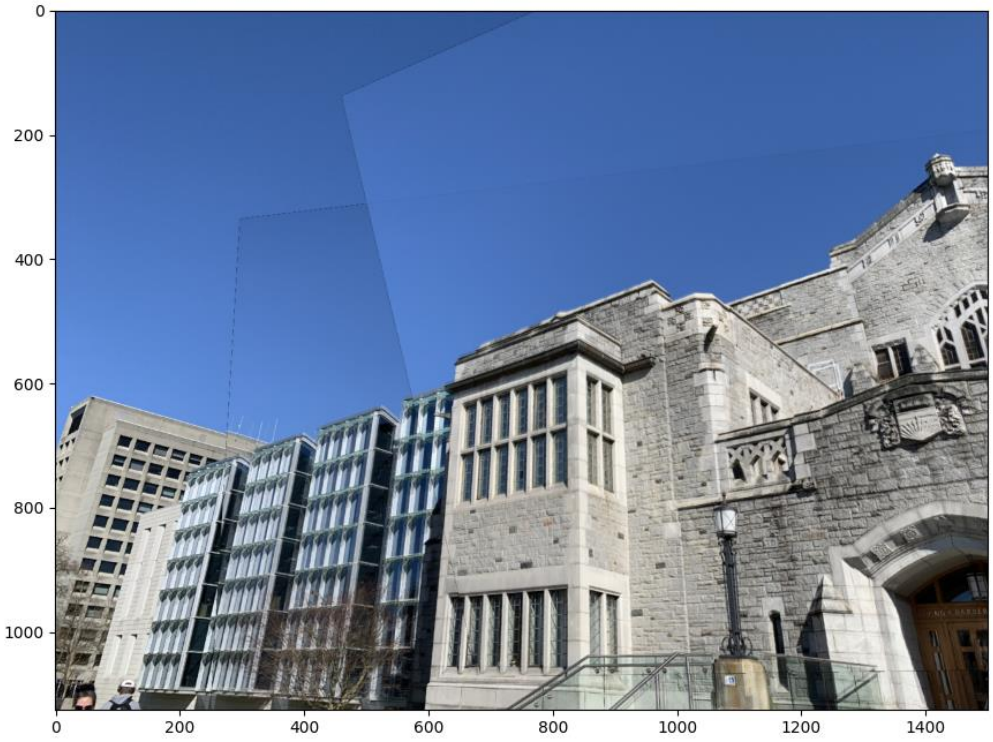

그림 7 – irving_out365.png