HW 04 – REPORT

소속 :

학번 :

이름 :

1. 서론

실습 목표 및 이론적 배경 기술 (1~2페이지)

이 실습의 목표는 local invariant features와 RANSAC을 이용해 image matching을 구현하고 homography와 keypoint projection을 이용하여 파노라마 이미지를 만드는 것이다.

local invariant features를 이용하여 두 이미지를 매칭시키는 방법은 크게 3단계로 구성된다. 먼저 Detection은 해당 이미지에서 keypoint를 찾는다. 예를 들면 harris corner detection을 이용하는 방법이 있다(앞으로 서술할 SIFT는 point를 찾기 위해 DoG(Difference of Gaussian)을 이용한다.

그 다음은 Description이다. Descriptor는 이미지가 변하더라도(transaltion, 2D rotation, scale, limited 3d rotation, limited affine transformation 등) 불변해야되고, 다른 descriptor와 구별가능해야한다. SIFT는 Scale Invariant Feature Transform의 약자로, description의 한 방법이다. 계산하는 방법은 먼저 interest point를 중심으로 하는 16x16의 사각형에서 각 점의 edge orientation을 계산하고, weak edge 제거한다. 그리고 16x16을 16개의 4x4로 쪼개고, 각 4x4 cell의 orientation historgram을 구한다. orientation historgram은 그림1과 같이 4x4 cell의 각 픽셀의 orientation을 8방향으로 분류하고, 각 방향의 magnitude를 합친 것이다. 각 descriptor는 16 x 8 = 128개의 값을 가지고 있다.

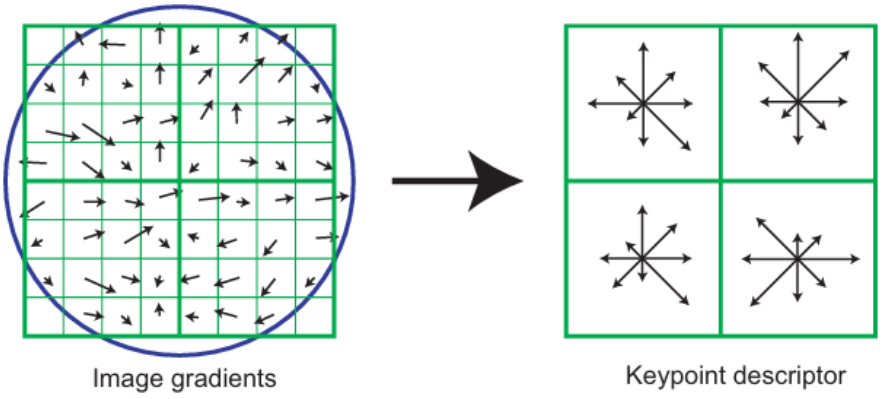

Figure 1 get Keypoint descriptor from 16x16 window

이러한 SIFT descriptor는 약 60도의 out of plane rotation에도 유지되고, illumination의 급격한 변화에도 유지된다.

matching은 두 이미지의 descriptor들을 서로 비교하여 가장 가까운 거리인 것끼리 매칭시킨다. 다만, 예를 들어 A와 B가 가장 가깝고, A와 C가 두번째로 가까울 때, A-B 거리와 A-C거리가 별로 차이가 나지 않는다면 A와 B를 매칭시키지 않는다.

RANSAC은 Random Sample Consensus의 약자로, outlier를 제거하는 방법 중 하나이다. matching 중 임의로 몇 개를 선택하여, 그 matching에 가장 알맞은 orientation 변화와 scale 변화를 계산(즉, 첫번째 이미지의 descriptor에서 두번째 이미지의 descriptor로의 orientation와 scale 변화를 계산)한다. 그리고 전체 matching 중 계산한 orientation변화와 scale 변화를 가진(오차범위 이내) matching을 선택한다. 이 실습에서는 1개의 matching을 임의로 선택한다.

이제 매칭된 keypoint를 이용하여 이미지1을 이미지2 공간으로 옮겨야 된다. 이때 Homography가 쓰인다. Homography matrix를 이용하여 이미지1의 좌표 (x,y)는 이미지2 공간의 (x′,y′)로 다음과 같이 매칭된다.

$$\begin{matrix} x' \\ y' \\ 1 \end{matrix} = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix} \begin{matrix} x \\ y \\ 1 \end{matrix} \quad (1.1)$$

매칭된 keypoint를 이용하여 homography matrix를 구하는 과정은 다음과 같다.

위 수식 1.1에 매칭된 keypoint $(x_i, y_i) \rightarrow (x_i', y_i')$ 를 대입하고 계산하면 아래와 같이 나타난다.

$$x_i'(h_{20}x_i + h_{21}y_i + h_{22}) = h_{00}x_i + h_{01}y_i + h_{02}$$

$$y_i'(h_{20}x_i + h_{21}y_i + h_{22}) = h_{10}x_i + h_{11}y_i + h_{22}$$

이를 matrix로 나타내고, 매칭된 모든 keypoint를 대입하면 아래와 같이 나타난다.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 & -x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 & -y_1' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n & -x_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n & -y_n' \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = Ah = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \quad (1.2)$$

여기서 homography matrix h를 구하는 방법은 A^TA 의 eigenvalue를 구하고, 가장 작은 eigenvalue에 해당하는 eigenvector를 구하면 된다.

2. 본론

실습 내용 및 결과 기술 (2페이지 이상)

Part1: SIFT Keypoint Matching

두 이미지의 keypoint를 서로 매칭시킨다.

Part1-1: FindBestMatches

두 keypoint의 descriptor의 각도를 측정한다.

두 descriptor 사이의 각도를 측정하는 방법은 다음과 같다.

$$\theta = \frac{\arccos(descriptor_i \cdot descriptor_j)}{|descriptor_i||descriptor_j|} \quad (2.1)$$

image2의 keypoint 중에서 image1의 한 keypoint와의 각도가 가장 작은 것을 선택하여 매칭시킨다. 단, 각도가 가장 작은 keypoint와 이루는 각도와 그 다음으로 작은 keypoint와 이루는 각도 차이가 거의 없으면, 매칭시키지 않는다.

즉 다음과 같을 때만 best_match를 선택한다.

$$\frac{angle(best\ match)}{angle(second\ match)} < threshold \quad (2.2)$$

solution.py에 FindBestMatches 함수에 구현되어있으며, main_match.py를 통해 실행 결과와 적용한 threshold는 아래와 같다.

Figure 2 matches for scene and basmati(threshold=0.6)

Figure 3 matches for scene and book(threshold=0.4)

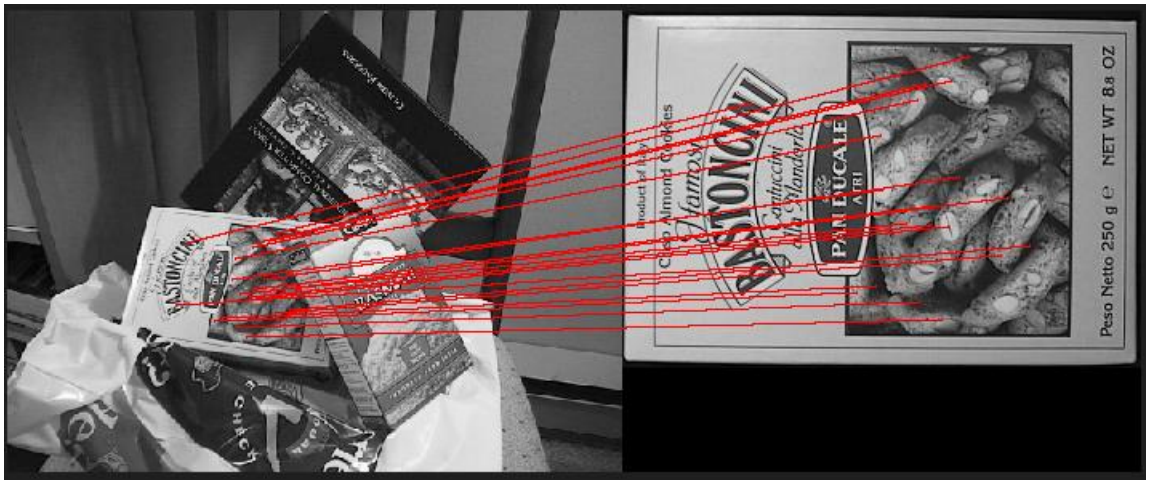

Figure 4 matches for scene and box(threshold=0.5)

Part1-2: RANSACFilter

이 함수는 RANSAC을 이용하여 outlier를 제거하는 함수이다. match 중 임의로 하나를 골라 이 match와 orientation변화와 scale 변화가 비슷한 match를 consensus set으로 만들고, 이를 10번 반복하여 consensus set이 가장 큰 것을 고른다. 여기서 비슷하다는 말은 orientation와 scale 변화에서 다음과 같이 정의한다.

match1의 orientation 변화를 d1, match2의 orientation 변화를 d2라 할 때, match1을 기준으로 match2가 다음을 만족한다:

$$|d1 - d2| < tolerance \quad (2.3)$$

match1의 scale변화를 s1, match2의 scale 변화를 s2라 할 때, match1을 기준으로 match2가 다음을 만족한다:

$$s1 \times (1 - agreement) < s2 < s1 \times (1 + agreement) \quad (2.4)$$

이 함수는 solution.py에 RANSACFilter 함수에 구현되어있고, main_match.py를 이용하여 실행한 결과는 아래와 같다.

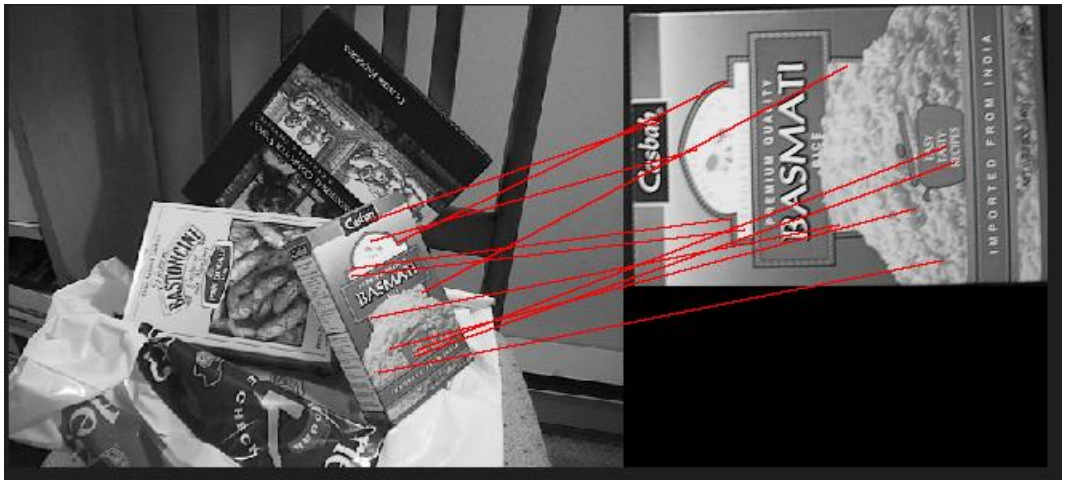

Figure 5 matches for scene and basmati with RANSAC(ratio=0.6, orient=30, scale=50%)

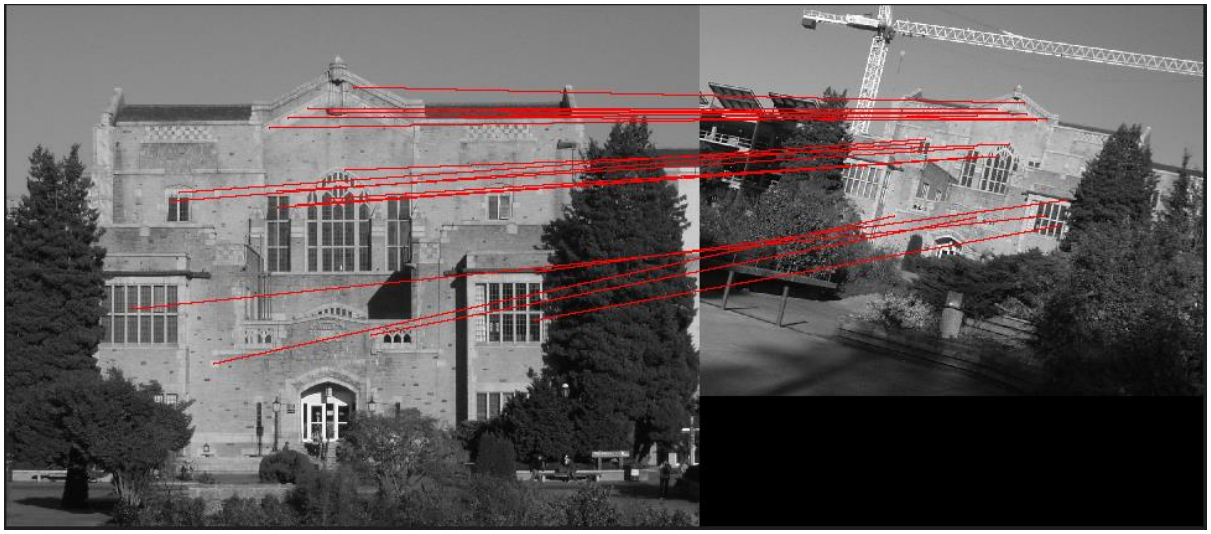

Figure 6 matches for library and library2 with RANSAC(ratio=0.6, orient=45, scale=60%)

Part2: Panorama

Homography를 이용하여 파노라마 이미지를 만드는 프로그램을 구현한다.

Part2-1: Keypoint projection

주어진 (x,y) 좌표를 homography matrix를 이용하여 projection한다.

먼저 (x,y) 좌표를 (x,y,1)로 변환 후, homography matrix와 행렬곱을 연산한다. 만약 결과값의 z값이 0이면, 오류를 방지하기 위해 1e-10으로 변경 후, homogeneous coordinate를 regular coordinate로 변경하기 위해 z값으로 나눈 후, z축을 제거한다.

solution.py의 KeypointProjection 함수에 구현되어 있으며, main_proj.py를 이용하여 실행한 결과는 아래와 같다.

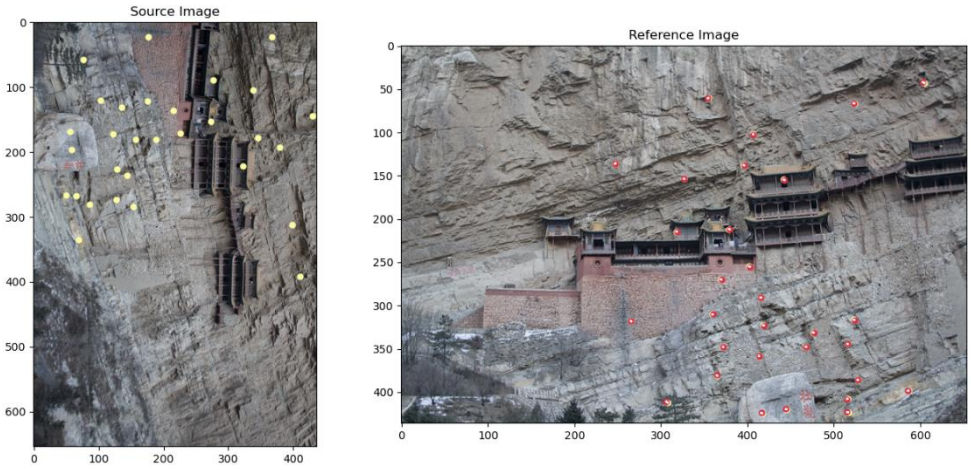

Figure 7 projection from Hanging1 to Hanging2

Part2-2: making panorama image with RANSAC Homography

이 함수는 주어진 match들(src 좌표에서 dst 좌표로 매칭됨)을 이용하여 homography matrix를 계산한다. 임의로 4개의 match를 선택한 후, 각 match 즉 (x,y)->(x′,y′)의 x, y, x′, y′값을 이용하여 식 1.2의 A matrix를 만든다. A^TA의 eigenvalue와 eigenvector를 numpy의 np.linalg.eig 함수를 이용하여 계산한다. np.linalg.eig의 결과값은 (eigenvalue,eigenvector)로 주어진다. 주의할 점은, 첫번째 eigenvalue에 해당하는 eigenvector는 eigenvector[:,0]이다

(즉, [eigenvector[0,0],eigenvector[1,0]…,eigenvector[8,0]]. eigenvector[0]이 아니다). 계산한 eigenvalue에서 가장 작은 값을 찾고, 그 값에 해당하는 eigenvector가 homography vector이고, 이를 3x3 matrix로 변환하면 된다.

계산한 homography matrix를 모든 source point에 적용하여 나온 projection 결과(즉, result point)와 실제 대응되는 점의 거리를 측정하고, 이 거리가 tolerance보다 작으면 inlier로 취급하고, 이 inlier가 가장 많이 나오게되는 homography matrix를 선택한다.

solution.py에 RANSACHomography 함수에 구현되어있으며, main_pano.py를 통해 실행한 결과는 아래와 같다 RANSAC의 특성상, 실행할 때마다 결과가 약간씩 달라질 수 있고, 터무니 없는 결과가 나올 수도 있다. 아래는 실행한 것 중 가장 잘 구현된 panorama 사진이다.

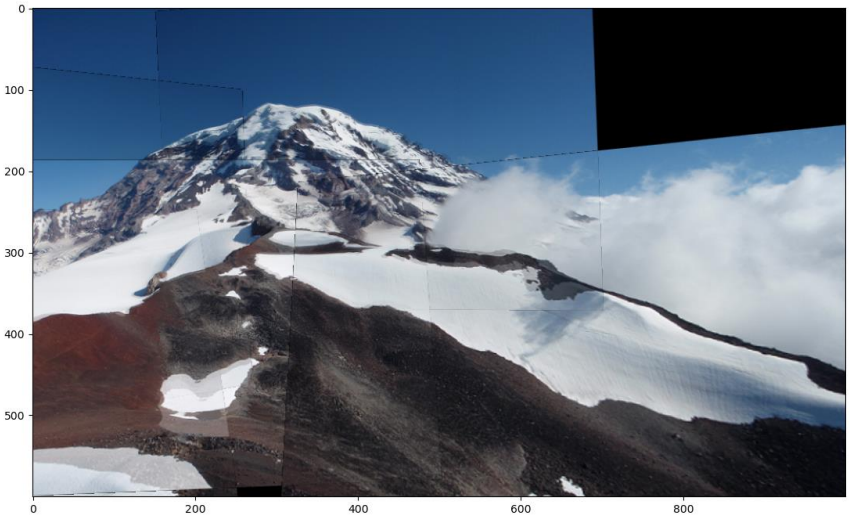

Figure 8 panorama image with Rainier123456 (num_iter=50, tol=10, ratio=0.9)

Figure 9 panorama image with fountain4 – fountain0(num_iter=50, tol=10, ratio=0.9)

Figure 10 panorama image with garden0-garden3-garden4(num_iter=200, tol=10, ratio=0.9)

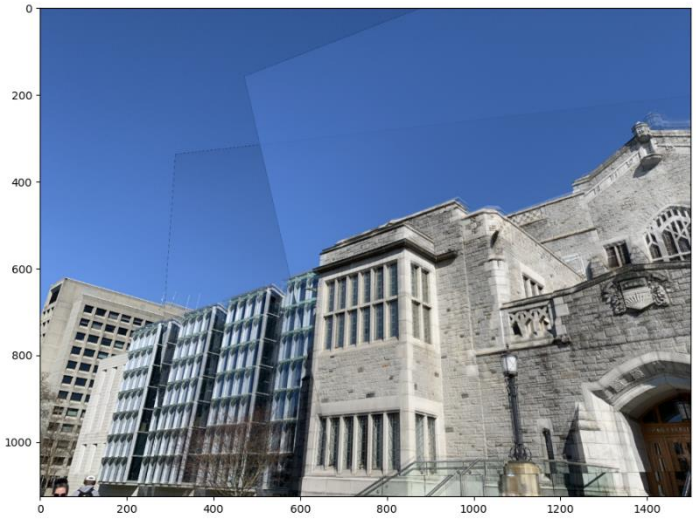

Figure 11 panorama image with irving_out3,6,5(num_iter=200, tol=10, ratio=0.9)

3. 결론

토의 및 결론 (1페이지)

keypoint를 descriptor를 이용하여 서로 매칭시키고, homography matrix를 계산하여 panorama image를 만들었다. RANSAC의 특성상 임의로 match를 선택하기 때문에 실행할 때마다 결과가 다르게 나올 때가 있다. 처음에는, eigenvector값을 불러올 때,eigenvector[i]로 해서 inlier가 0개로 나왔고, 이를 eigenvector[:,i]로 바꾸어 해결하였다. 이 실습을 통해 opencv를 이용하지 않고,

homography matrix를 구하고 panorama image를 만드는 방법을 알게 되었다.

참고한 자료는 아래와 같다.

- Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94
- https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html