HW 04 – REPORT

소속 : 정보컴퓨터공학부

학번 : 201824487

이름 : 박현수

1. 서론

이번 과제에서는 수업 중에 배운 RANSAC과 Projection을 직접 코드로 작성해보는 것이다. 먼저 SIFT(Scale-Invariant Feature Transform)은 image의 크기와 회전에 불변하는 특징을 추출하는 알고리즘이다. 올바른 matching 할 수 있지만 계산이 많이 필요해서 오랜 시간이 필요하다. 이걸 해결하기 위해 만든 알고리즘이 RANSAC(Random sample consensus)이다. 랜덤으로 몇 개의 sample들을 선택하여 matching해보는 것이다. 그 중 오차 범위 내로 비슷한 matching을 inlier이라 하고 그 외에는 outlier라고 한다. 무작위로 선택한 sample들 중 inlier가 가장 많은 vector로 matching하는 알고리즘이다. RANSAC을 이용하면 계산 수를 줄여 효율적으로 matching을 진행할 수 있다. 이 matching algorithm과 homography matrix를 이용하여 projection을 진행한다. 한 장면을 여러 사진으로 찍은 Image들을 이용하여 매칭된 점을 이어 하나의 큰 Image를 만드는 것을 해볼 것이다. 그 과정에서 필요한 Homography matrix에 대한 정보와 계산 방법은 강의 자료를 참고하면 알 수 있다.

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix}}_{\mathbf{A}_{2n \times 9}} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\mathbf{h}_{9}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{0}_{2n}}$$

"Solving for homographies"

2. 본론

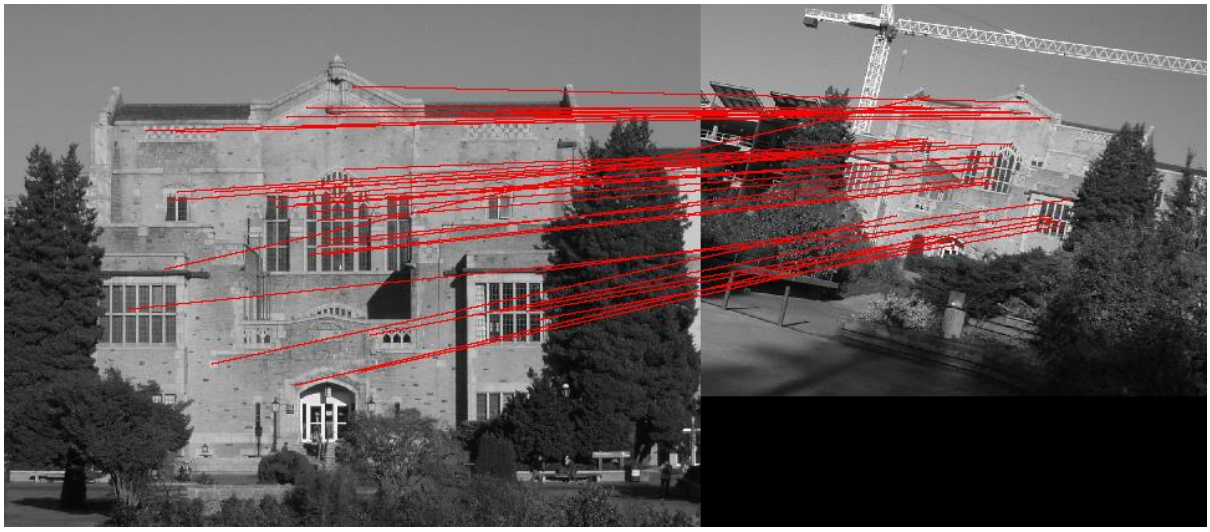
Part1-1은 SIFT algorithm을 이용하여 Keypoint Matching을 진행한다. Descriptor 2개를 모든 경우에 대해 각도를 acos()함수를 통해 저장한다. 그리고 각각 각도의 값을 비교하여 최적의 값을 찾는다. 이 과정을 FindBestMatches()함수에서 구현한다. 이 과정은 과제 설명 영상에 포함되어있다.

"scene-book"

"scene-basmati"

FindBestMatches()함수에서 ratio_threshold값을 0.7로 하여 진행하였다. Outlier의 개수는 10개 이하로 하여 진행해야 한다. 값을 0.8로 진행하였을 때, 육안으로 보기엔 outlier의 개수가 8개 정도로 보였지만, 확실치 않았기 때문에 0.7값이 안전하다고 판단되어 진행하였다.

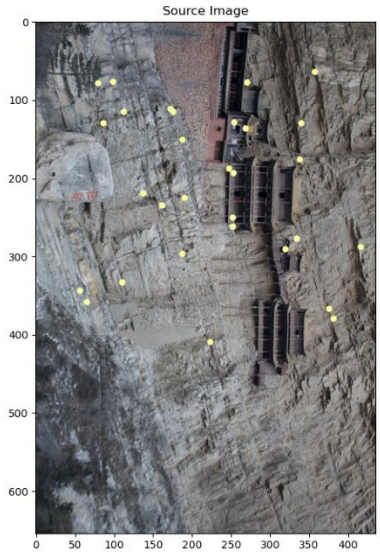
Part1-2는 RANSAC함수를 구현하는 것이다. 난수를 선택하여 첫 번째 orientation과 scale-ratio를 계산한다. 다음으론 모든 경우에 대해서 두 번째 orientation과 scale-ratio를 계산하고, degree와 scale이 error범주 내에 있는지 확인한다. 이 방법을 10번 반복하여 최고의 match를 찾는 알고리즘이다. 위와 마찬가지고 과제 강의를 통해 코드를 작성하고 파라미터만 수정한다. 이 부분도 같은 ratio_threshold값을 사용하였다.



“library-library2”

Outlier가 거의 보이지 않는 결과물을 얻을 수 있다.

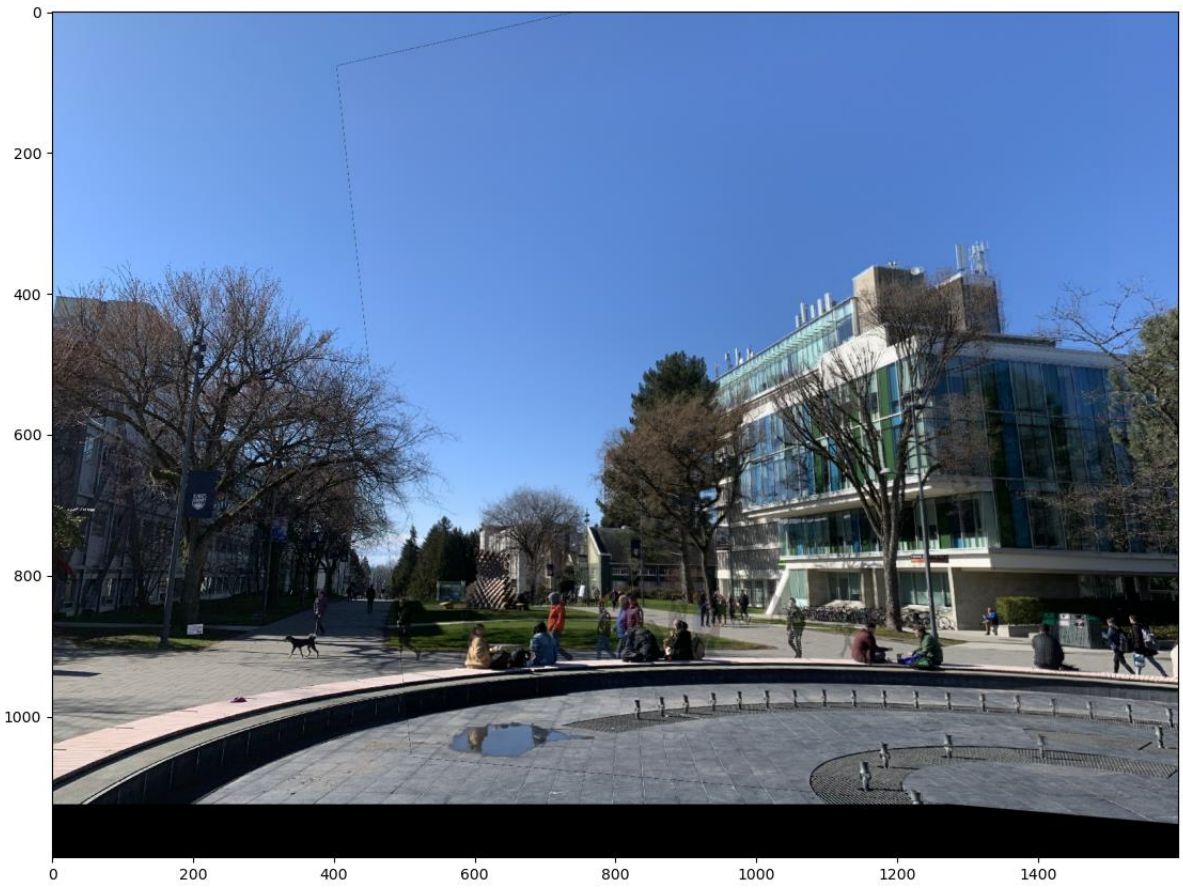
Part2-1은 KeypointProjection()함수에 주어진 homography matrix를 이용해 projection하는 코드를 작성하는 것이다. 먼저 값이 1인 column을 저장하여 points에 추가하여 차원을 증가시킨다. 그 후 homography matrix와의 내적을 통해 값을 추출한다. 마지막으로 추가해준 차원 z의 값으로 x, y를 나누어 준다. 다만, z값이 0이면 나누어 줄 수 없기 때문에 0 대신에 1e-10값을 대체하여 사용한다.



“Hanging1-Hanging2”

Matching된 점들을 통해 잘 진행되었음을 알 수 있다.

Part2-2와 이 전 문제의 차이점은 homography matrix를 직접 계산의 유무이다. RANSAC을 추가한 Homography() 함수를 구현하는 것이다. 먼저 projection을 위해 필요한 최소의 match points 4개를 random으로 선정한다. 그 후 강의자료에 나와있는 A matrix를 만들어 준다. A matrix를 만들었다면 A^T matrix와 A matrix를 내적하고, 내적한 값을 이용하여 eigen value와 eigen vector를 찾는다. 이 때 가장 작은 eigenvalue의 eigenvector를 구하여 shape를 3*3으로 맞춰준다. 이렇게 homography matrix를 생성하였으면, projection을 진행하여 matching된 점의 길이가 threshold보다 길면 inlier로 판단하여 최종적으로 inlier의 개수가 가장 많은 h matrix를 선정한다.



“fountain4-fountain0”

“garden0-garden3-garden4”

“irving_out3-irving_out6-irving_out5”

마지막 문제에서는 canvas의 높이와 너비, num_iter, tol, ratio_thres 의 많은 파라미터들의 값을 변경하여 최적의 projection을 찾아야 한다. 여러 번 반복한 결과,

1) fountain에서의 파라미터: num_iter = 60, tol = 10, ratio_thres = 0.75
2) garden에서의 파라미터: num_iter = 70, tol = 10, ratio_thres = 0.7
3) irving에서의 파라미터: num_iter = 50, tol = 10, ratio_thres = 0.7

로 적절한 결과를 얻을 수 있었다.

3. 결론

이번 과제에선 강의 시간에 배운 matching algorithm의 개념에 대해 다시 확인하고, 직접 코드를 통해 성능을 확인할 수 있었다. 그 뒤 작성한 matching code를 이용하여 homography matrix가 주어져 있는 함수로 projection을 하고 마지막으로는 직접 최적의 homography matrix를 찾아보았다. 앞의 matching algorithm에 대한 코드는 과제 영상을 통해 빠르게 이해하고 구현하여 진행할 수 있었다. 이를 바탕으로 강의 pdf에 나와있는 homography matrix와 projection 계산법을 통해 코드를 작성하면 어려움 없이 진행할 수 있었다. 코드 작성하는 과정은 앞의 과제들과 난이도 부분에서 큰 차이는 없었다. 하지만 작성한 solution.py가 끝이 아니라 main code에서 직접 파라미터를 수정하며 최적의 결과값을 찾아가는 과정에서 많은 시간이 필요했다. 하지만 모든 결과들 다 적절한 파라미터를 찾아서 만족스러운 결과를 얻었다. 이번 과제를 통해 파라미터의 설정에 있어 개념을 완벽히 이해하고 있어야 의미 없이 값을 바꾸는 경우를 줄일 수 있다는 것을 느꼈다.