Visual Odometry, Midterm assignment

120230196

조이준

1. Problem statement

   중간 대체 과제는 두 개의 입력 이미지를 이용한 파노라마 이미지 생성이다.

   해당 과제의 수행 흐름은 아래와 같다.

   A. 먼저 서강대학교 내부에서 서로 다른 시점의 두 이미지를 촬영한다.

   B. 다음 OpenCV 라이브러리를 이용하여 두 장의 이미지의 ORB 키포인트와 키포인트 디스크립터를 계산한다.

   C. 계산한 키포인트들에 Hamming distance 를 이용한 Brute Force matching 을 수행하여 매칭 리스트를 생성한다.

   D. 직접 RANSAC 알고리즘을 작성하여 이를 기반으로 Homography matrix 를 계산한다.

   E. 직접 작성한 코드로 파노라마 이미지 액자(틀)을 생성한다.

   F. 직접 작성한 코드로 두 장의 입력 이미지를 Homography matrix 를 사용하여 파노라마 틀에 올린다.

   G. 동일한 과정을 다른 두 이미지 셋에 적용하여 본다.

2. Algorithm

   전체 알고리즘 소스 코드는 기능에 따라 함수를 만들어 모듈화 하였다. 사용한 함수는 크게 5 가지이다. 각 함수의 구현과 기능은 아래 코드 이미지와 그에 따른 설명으로 나열한다.

   먼저 Global 코드로 입력 경로, 출력 경로, 총 RANSAC 반복횟수와 RANSAC threshold를 인자로 받아온다. 해당 인자들을 main 함수에 입력으로 주는 것이 소스 코드의 시작이다.

   입력 인자로 RANSAC의 반복횟수와 RANSAC의 threshold를 줌으로써 반복횟수와 threshold의 차이에 따라 키포인트 매칭의 결과가 달라지는 것을 확인할 수 있게 하였다.

```python
if __name__ == '__main__':
    # Set argument parser #
    parser = argparse.ArgumentParser(description='Use two images with dif view point to make panorama')
    parser.add_argument('--input_path', nargs="+", type=str, default=["Input/myseat1.jpg",
                                                                      "Input/myseat2.jpg"], help='Directory path to save captured images.')
    parser.add_argument('--output_path', type=str, default="Output/", help='Path to save panorama images')
    parser.add_argument('--max_iters', type=int, default=3000, help='Max iterations of RANSAC')
    parser.add_argument('--threshold', type=int, default=5, help='Threshold of RANSAC')

    args = parser.parse_args()
    main(args.input_path[0], args.input_path[1], args.output_path, args.max_iters, args.threshold)
```

<그림 1> Global Source Code

A. main

```python
# FUNCTION.
#
# DO FUNDAMENTAL WORKS, LOAD IMAGES, EXTRACT KEYPOINT, SAVE IMAGE
# CALL OTHER FUNCTIONS
#
def main(path1, path2, output_path, max_iters, threshold):
```

<그림 2> main function

main 함수는 입력으로 사용할 두 이미지의 경로와 최종 출력 이미지의 경로, RANSAC의 최대 반복횟수와 threshold를 인자로 받는다.

Main 함수의 기능은 ORB를 통한 키포인트 추출, Brute Force matching을 하고 이 결과를 다른 기능을 수행하는 함수들의 입력으로 주어 호출하는 것이다. 최종적으로는 호출한 함수를 통해 파노라마 이미지의 생성이 완료되면 주어진 출력 경로에 최종 파노라마 이미지를 저장한다.

ORB와 Brute Force matching은 Python의 OpenCV 라이브러리의 기능을 이용하여 구현했다.

과제 외의 부가적인 기능으로 OpenCV 라이브러리 기능을 이용하여 두 이미지의 ORB Keypoint matching 결과를 출력 디렉토리에 'keypoint_match' 이름의 이미지로 저장하였다.

B. DIY_HOMOGRAPHY

```python
# FUNCTION.
#
# FIND THE BEST HOMOGRAPHY MATRIX WITH GREAT KEYPOINT MATCHING USING RANSAC
# ARGUMENT: MATHCED KEYPOINTS (MATCHING RESULT, KEYPOINTS, ITERATIONS, THRESHOLD)
#
def DIY_HOMOGRAPHY(k1, k2, matches, max_iters, threshold):
```

<그림 3> DIY_HOMOGRAPHY function

DIY_HOMOGRAPHY 함수는 입력으로 두 이미지의 키포인트와 해당 키포인트의 매칭 결과 그리고 RANSAC 알고리즘을 수행하기 위한 RANSAC 알고리즘 최대 반복횟수와 threshold를 인자로 받는다.

각 이미지의 키포인트에서 매칭된 포인트를 리스트화 하여 결과를 DIY_RANSAC 함수에 인자로 전달한다. DIY_RANSAC 함수를 통해서 Homography matrix와 가장 좋은 매칭 결과를 받아 main 함수에 이를 반환한다.

C. DIY_RANSAC

```
# FUNCTION.
#
# RANSAC ALGORITHM
# ARGUMENT: IMG1 KEYPOINTS, IMG2 KEYPOINTS
#
def DIY_RANSAC(src, dst,  max_iters, threshold):
```

<그림 4> DIY_RANSAC function

DIY_RANSAC 함수는 두 이미지의 매칭된 키포인트와 최대 반복횟수와 threshold를 인자로 전달받는다.

매칭된 키포인트에서 랜덤으로 4 개를 뽑고 최적의 Homography를 찾기 위해 생성한 Homography matrix를 적용하여 하나의 셋으로 다른 셋을 예측한다.

예측한 포인트와 기존의 포인트의 거리를 계산하여 threshold와의 비교를 하고 inlier가 가장 많이 예측된 Homography matrix를 best Homography matrix로 판단하여 반환한다. 이 때, 해당 inliers도 같이 반환하여 최적의 matching 결과를 확인할 수 있게 한다.

D. DIY_PANORAMA

```
# FUNCTION.
#
# GENERATE PANORAM IMAGE BY HOMOGRAPHY MATRIX
# ARGUMENT: INPUT IMAGES, HOMOGRAPHY MATRIX
#
def DIY_PANORAMA(im1, im2, homography):
```

<그림 5> DIY_PANORAMA function

DIY_PANORAMA는 두 이미지와 위에서 계산한 Homography matrix를 입력 인자로 받는다. 이 입력 인자를 통해서 적절한 크기의 파노라마 액자를 만들고 각 입력 이미지를 이어 붙이기 위해 최적의 포인트를 Homography matrix를 통해 찾는다.

이후에 image 1을 DIY_WARPING 함수로 보내어 파노라마 액자의 왼쪽 부분에 이를 적절히 붙인다.

이후에 파노라마 액자에 image 2를 위의 결과의 옆에 붙인다.

이렇게 생성한 파노라마 이미지를 main 함수에 반환한다.

E. DIY_WARPING

```
# FUNCTION.
#
# WARPING IMAGE TO PANORAMA FRAME BY WARPING MATRIX
# ARGUMENT: ONE IMAGE, WARPING MATRIX, SIZE
#
def DIY_WARPING(matrix, im2, panorama_size):
```

<그림 6> DIY_WARPING function

DIY_HOMOGRAPHY 함수로부터 이미지와 파노라마 액자의 크기, 그리고 image warping에 사용할 행렬을 인자로 받는다.

행렬과 포인트 연산을 통해 해당 포인트가 이미지 내에 존재하는 영역이면 해당 픽셀 값을 파노라마 액자에 값으로 지정한다.

이 과정을 통해서 인자로 받은 이미지를 액자에 출력한다.

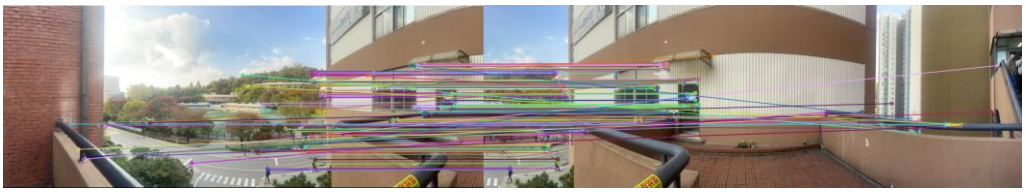
3. Experiment

A. 실험1 결과

아래의 두 이미지는 입력으로 사용한 서강대학교 이미지이다.

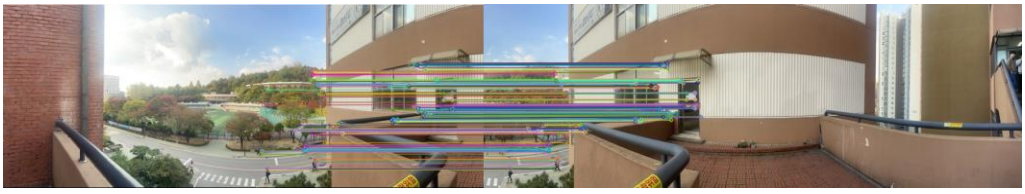
<그림 7> Two input images for first experiment

아래의 이미지는 Python 라이브러리 OpenCV를 활용한 ORB 키포인트 매칭 결과이다.



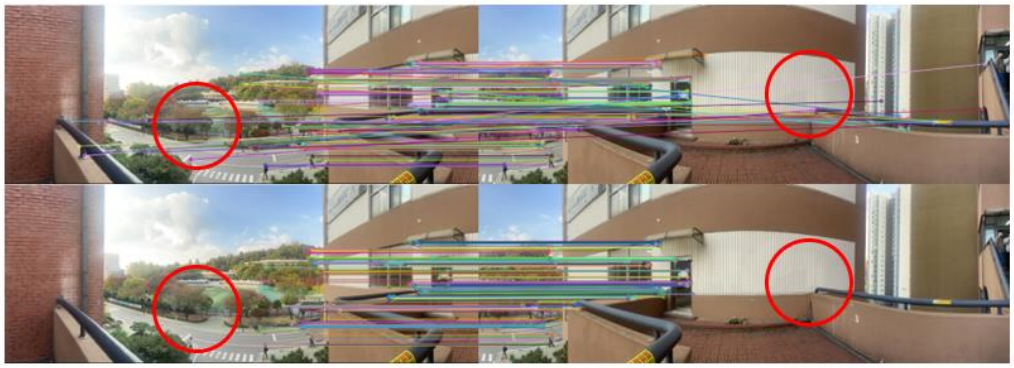
<그림 8> ORB Keypoint matching result

Homography matrix를 통해서 최적의 matching point를 찾은 뒤의 결과는 다음과 같다.



<그림 9> RANSAC Keypoint matching result (1000 iterations, 5 threshold)

Threshold에 따라 불필요한 매칭 결과가 일부 사라진 것을 볼 수 있다.



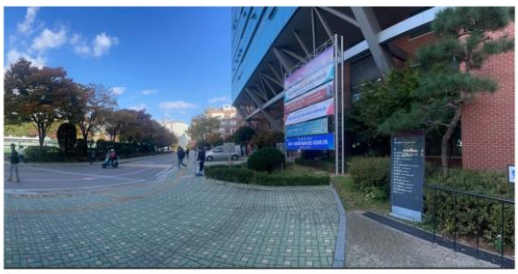
<그림 10> Compare two matching results

최종적으로 실험을 통해 얻은 파노라마 이미지는 아래와 같다.

<그림11> Final panorama image result

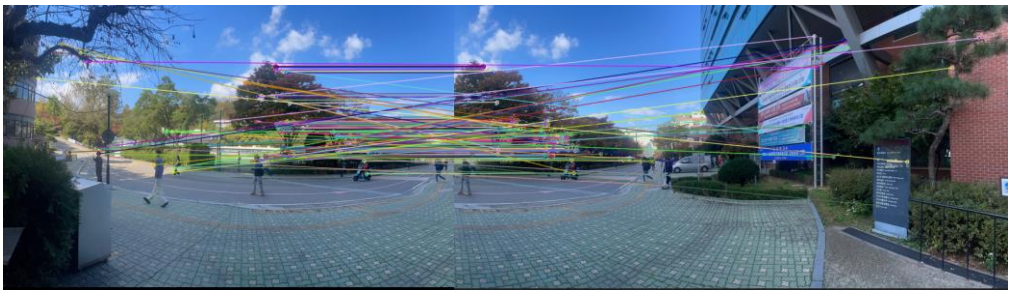
B. 실험2 결과

아래의 두 이미지는 두번째 실험의 입력으로 사용한 서강대학교 이미지이다.



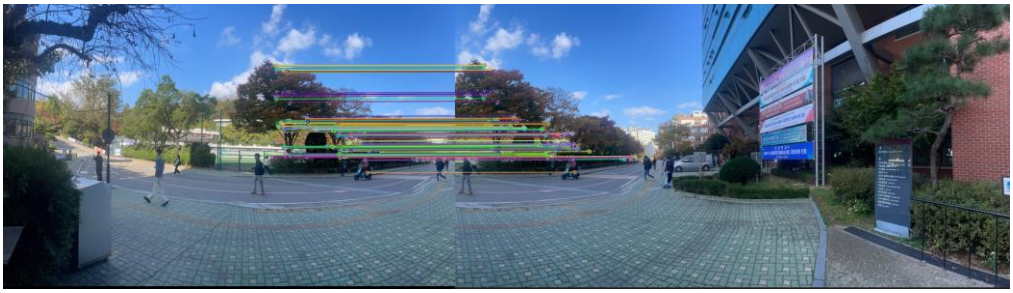
<그림 12> Two input images for second experiment

아래의 이미지는 Python 라이브러리 OpenCV를 이용한 ORB 키포인트 매칭 결과이다.



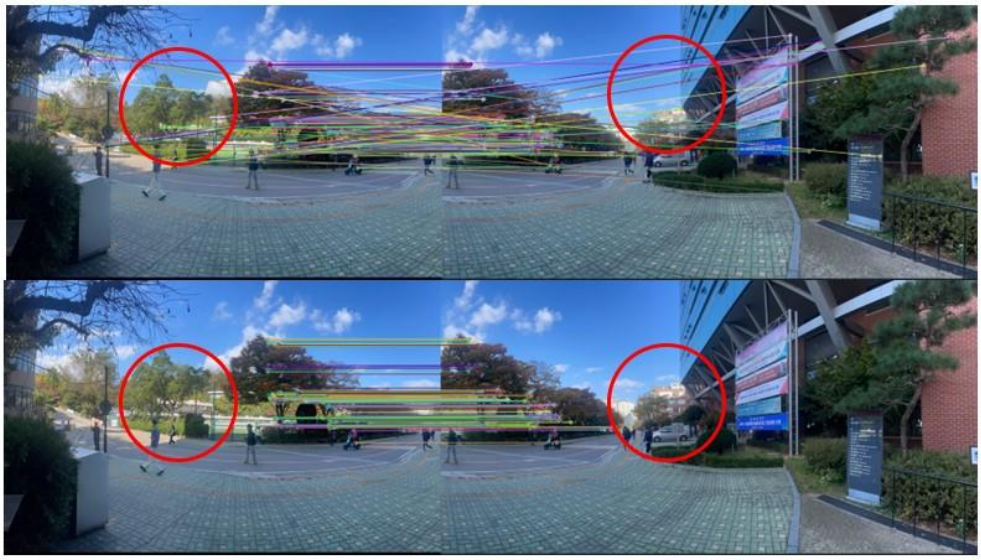
<그림 13> ORB Keypoint matching result

Homography matrix를 이용한 최적의 매칭 결과는 다음과 같다.



<그림 14> RANSAC matching result (30000 iterations, 5 threshold)

Threshold에 따라 불필요한 매칭이 지워진 것을 볼 수 있다.



<그림 15> Compare two matching result

최종적으로 실험을 통해 얻은 파노라마 이미지는 다음과 같다.

<그림 16> Final panorama image result