PA2 Structure-from-Motion(SfM)

20251155 박찬영

1. step1: Feature_maching.py

목적: 두 이미지에서 SIFT feature와 descripter를 추출하고, 매칭해서 대응점을 출력하는 것을 목적으로 한다.

overview:

1) 우선 두 이미지를 그레이스케일 이미지로 변환하였다. PA1에서 배운 방식을 활용
2) 다음 단계에서는 슬라이드와 주석에서 말한 SIFT 알고리즘을 활용하였다.[1]
    a) detect(.)로 keypoint를 계산하였다.
    b) compute(.)로 descriptor를 계산하였다.
3) 특징점 매칭을 위해 Brute-Force Matcher (BFM)[2]를 사용하였다.
    a) 우선 각 이미지들간의 거리를 일일이 계산을 한다.
    b) 이 거리 계산 결과 중 가장 가까운 2개를 뽑는다.
4) 2개의 매칭 후보에 대해 Lowe's ratio test를 적용한다.
    a) 두 후보가 있고, 첫번째 거리가 두번째 거리의 threshold_knn배보다 작은 경우를 조건으로 두었다.
5) 이 과정을 통해서 얻은 쌍으로된 대응점을 matches에 append하여 반환한다.

상세 구현내용:

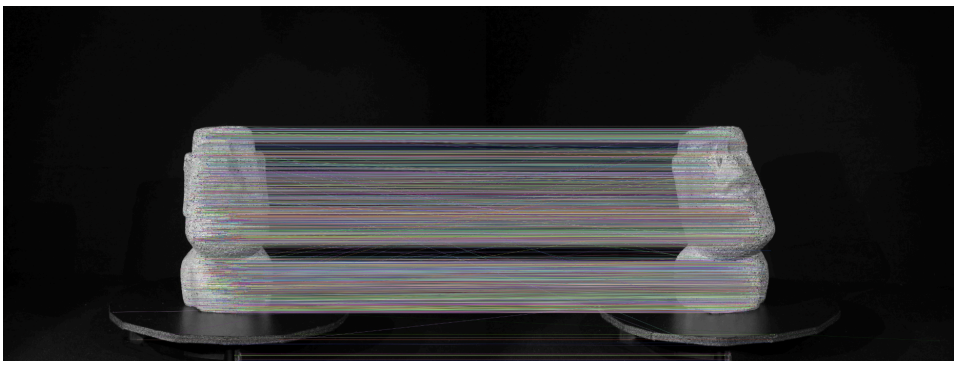
- 이 파트는 주어진 슬라이드와 주석에서 사용하라는 메서드를 공식 문서를 통해 확인해본 후 구현하였다.
- kmatch의 한 요소 값이 2 미만인 경우도 있을 수 있어서, 그 경우 continue로 안정성을 높였다.
- 여기서 k =2 로 설정한 이유는 Lowe's ratio test를 적용하기 위해서다. 이 방법은 가장 가까운 매칭인 첫번째 후보와 두 번째로 가까운 매칭 두 번째 후보의 거리 비율을 비교하여 신뢰도가 낮은 매칭을 걸러내는 방법으로 2개의 후보가 필요하여 k =2 로 설정하였다.

결과

```
(base) park_chanyoung@bagchan-yeong-ui-MacBookPro PA2 % python -u "/Users/park_chanyoung/Desktop/Gist_cv_PA/PA2/main_two_view.py"
Camera intrinsic matrix:
[[9.54641971e+03 0.00000000e+00 2.63605980e+03]
 [0.00000000e+00 9.52562444e+03 1.99502512e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Number of keypoints and matches (image0, image1):
kp1 : 44717, kp2 : 45789, matches : 2954
(base) park_chanyoung@bagchan-yeong-ui-MacBookPro PA2 %
```

[1] https://docs.opencv.org/4.x/d7/d60/classcv_1_1SIFT.html
[2] https://docs.opencv.org/4.x/d3/da1/classcv_1_1BFMatcher.html



- 이미지 1의 특징점 수 (kp1): 44,717개
- 이미지 2의 특징점 수 (kp2): 45,789개
- 매칭된 특징점 쌍 (matches): 2,954쌍
- 위 이미지는 특징점 중 매칭에 성공한 대응점을 매칭선으로 나타내었다.
- 2954 / 45789 = 6.5% 정도로 매칭 성공하였다.

2. step2: E_estimation.py

목적**:** Essential Matrix E를 추정하고, 이를 사용하여 정규화된 특징점 간의 관계(Epipolar Constraint)를 기반으로 inlier 쌍을 구하는 것이 이 step이 목적이다.

Overview 및 상세구현**:**

1) 우선 입력으로 받은 각 이미지의 특징점과 매칭점을 numpy형식으로 변환해준다.
2) RANSAC을 위한 사전 단계로서, 대응점을 Homogeneous 형태로 변환, 정규화 과정을 수행하였다.
3) MATLAB을 활용하여 RANSAC 알고리즘을 수현하였다.
    a) 모든 매칭점들 중 무작위로 5쌍을 선택하고, 5쌍을 형태와 타입을 변환하여, calibrated_fivepoint(.)에 입력으로 넣어서 E (Essential Matrix) 후보를 E_cand로 추정하였다. E_cand를 numpy로 변환 후, shape과 출력을 확인해보았다. (N, 9) 형식으로, 각 열마다 각각 후보 E들을 나타낸다.
    b) 그 후, Epipolar Constraint 관계를 이용하여, inlier 쌍을 구하였다. 이때 아래 식처럼 0으로 수렴할 수는 없고, 코드에서 주어진 threshold=1e-5 값 이하인 점들을 inlier로 판단하여 구했다.

$$\hat{x}'^{T} E \hat{x} = 0$$

    c) 찾은 모델 중 가장 많은 inlier를 가지는 E를 업데이트 하였다. 후보들 중 가장 정확하다고 생각하는 E를 업데이트
4) 이 과정을 max_iter=5000 번 반복하였다.

결과**:**

```
Number of keypoints and matches (image0, image1):
kp1 : 44717, kp2 : 45789, matches : 2954
[[-4.73927003e-03  1.36223657e-01  1.95681468e-03]
 [ 1.51621255e-01  4.97375915e-03  6.90639334e-01]
 [ 2.62250882e-02 -6.93348746e-01 -3.91369608e-04]]
Number of inlier points: 2831
```

```
[[-4.73927003e-03  1.36223657e-01  1.95681468e-03]
 [ 1.51621255e-01  4.97375915e-03  6.90639334e-01]
 [ 2.62250882e-02 -6.93348746e-01 -3.91369608e-04]]
```

- 2번째 사진은 Essential Matrix E의 결과이며,두 정규화된 이미지 좌표 사이의 Epipolar Constraint를 만족하는 행렬로, 이 행렬을 통해 두 이미지 간의 회전 및 평행이동 관계를 유도할 수 있다.
- 이전 단계의 2954개의 매칭 중, RANSAC 기반해, 정합된 매칭쌍인 inlier이 2831개로 결정된다.

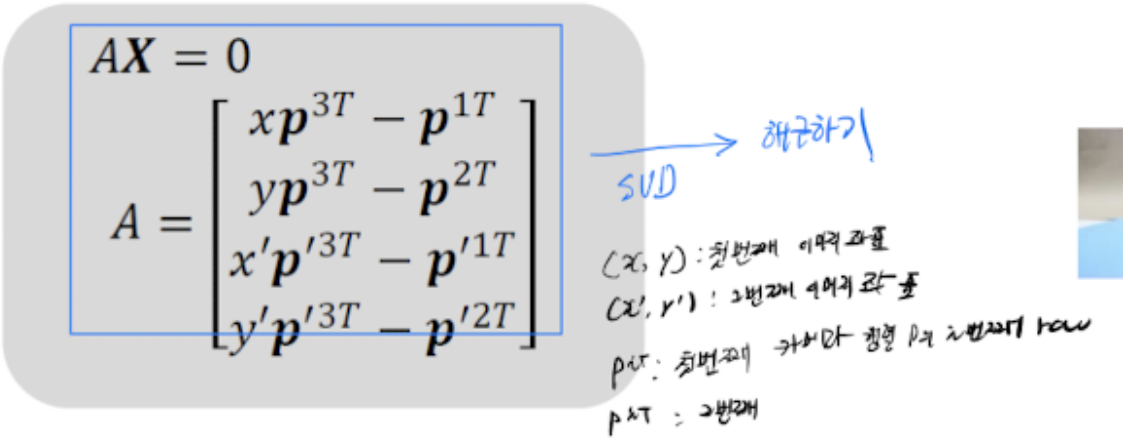
step4: [triangulation.py]

step3을 하기 전에, step 4를 설명하는 이유는 step3에 4가지 P1 후보를 선택할때, cheirality condition 조건을 확인하기 위해서, 3D point 위치를 알아야하기 때문이다. 슬라이드 상에서도 4step을 참조하라고 나와있다.

목적**:** inlier에 있는 대응점을 활용해서 3D 공간상으로 점을 재구성하는 것을 목적으로 하고 있다.

Overview & 상세구현**:**

1) 우선 아래 Figure에 있는 수식을 활용하여 A를 구했다.
2) 그 후 A를 SVD하였다.
    a) 선형방정식인 AX = 0의 해는 Non-trivial solution이다. 즉, X는 A의 null space에 있는 벡터여야한다.
    b) 그래서 가장 안정적인 해를 찾기 위해 SVD를 적용한다.
    c) 여기서 V의 마지막 열(Vt의 마지막 행)은 A의 null space를 생성하는 벡터이다.
    d) homogeneous로 변환을 위해 (x,y,z,w) → (x/w, y/w, z/w, 1)로 변환해줘야한다.
3) 위 가정을 inlier에서 반복한다.



step3: E_decomposition.py

목적: essential Matrix E를 SVD분해하여, 두 카메라의 상대적인 위치와 방향 (즉, 카메라 포즈: R, t)을 추정하고 cheirality condition을 이용해 P1(두번째 카메라 포즈)를 선택하여 추정하는 것입니다.

Overview 및 특별 구현:

1. 기존 카메라 P0 포즈 정하기
2. E를 SVD 분해하기 & W 정하기
    a. 후에 4가지 조합을 위해 필요한 행렬을 구하기 위해서이다.
3. 4가지 후보 생성하기
    a. 아래 슬라이드 식을 참조하였고 concat으로 R,t를 연결하는 것에 사용하였다.

$$P_1 = [UWV^T | +u_3]$$
$$P_2 = [UWV^T | -u_3]$$
$$P_3 = [UW^TV^T | +u_3]$$
$$P_4 = [UW^TV^T | -u_3]$$

4. 제일 중요한것은 cheirality condition을 사용하였다.
   우선 앞선 step4로 3d 포인터를 구하고, 두카메라에서 3d point값을 투영시키고 z > 0 보다 크다는 조건을 활용하여 P1을 선정하였다.

결과: 아래는 결과 값이다.

```
(base) park_chanyoung@bagchan-yeong-ui-MacBookPro PA2 % python -u "/Users/park_chanyoung/Desktop/Gist_cv_PA/PA2/main_two_view.py"
Camera intrinsic matrix:
[[9.54641971e+03 0.00000000e+00 2.63605980e+03]
 [0.00000000e+00 9.52562444e+03 1.99502512e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Number of keypoints and matches (image0, image1):
kp1 : 44717, kp2 : 45789, matches : 2954
[[-4.73650625e-03  1.32734535e-01  1.74292635e-03]
 [ 1.49569137e-01  5.06595194e-03  6.91086521e-01]
 [ 2.66619649e-02 -6.94008582e-01 -3.59619270e-04]]
Number of inlier points: 2830
Camera 1 Pose (P0):
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]]
Camera 2 Pose (P1):
[[-0.91941511 -0.0379486   0.3914534   0.98219761]
 [ 0.04061641 -0.99917374 -0.00146609 -0.00237937]
 [-0.39118559 -0.01455149 -0.92019677  0.18783556]]
Number of triangulated 3D points: 2830
```