

# HW05 - REPORT

정보컴퓨터공학부 201624536 이국현

May 5, 2022

# Chapter 1

## 서론

- Projection
- Homography
- RANSAC

### 1.1 Projection

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

homogeneous image coordinates                      homogeneous scene coordinates

Figure 1.1: Image to homogeneous

Source image를 reference image의 coordinate로 변환하기 위해 Homography matrix를 곱하려면 먼저 Source image를 homogeneous coordinate로 변환해야 한다. 이는 그림과 같이 Additional dimension을 주어 변환할 수 있다.

Homography matrix 연산을 수행하였다면, 다시 Homogeneous coordinate를 Image coordinate로 변환해야 한다. 이는 그림과 같이 Additional dimension을 제거하고, 그 값으로 Nomalize 하면 된다.

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

homogeneous image coordinates                      homogeneous scene coordinates

Figure 1.2: Image to homogeneous

## 1.2 Homography

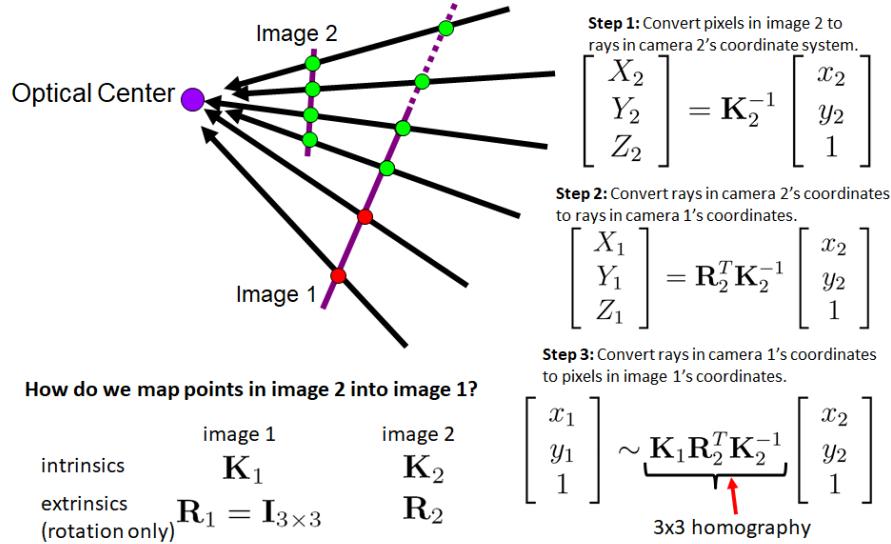


Figure 1.3: Homography matrix

Homography matrix는 Intrinsics과 Transformation을 표현하는 Extrinsics으로 구성되어 있으며, 한 Image coordinate에서 다른 Image coordinate로 변환하는 과정을 포함하고 있다.

## 1.3 RANSAC

Panorama를 생성하기 위해서는 Image 간의 Homography matrix를 계산해야 하며, 이를 위해 Image 간의 Feature를 매치하지만, Outlier가 생길 수 있다. 이 Outlier는 Panorama를 생성하는 데 큰 장애물이 될 수 있기 때문에 RANSAC을 이용

하여 Outlier를 제거한다. RANSAC의 알고리즘은 다음과 같다.

1. Feature Matches에서 랜덤으로 필요한 만큼의 Match set을 선택하여 Model 생성  
이때 필요한 Match의 수는 Transformation의 특성에 따라 다르다.
2. 각 Model에서 Transformation 구한다 (Homography matrix)
3. 각 Model에서의 Inlier counting
4. 1 - 3을 반복
5. Inlier가 가장 많은 Model 선택
6. 새로운 Model에서 Outlier 제거
7. 추출된 Inlier로 다시 최종 Model 생성

# Chapter 2

## 본론

### 2.1 Keypoint Projection

---

```
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
    h_points = []
    for xy_point in xy_points:
        # image coordinate를 homogeneous coordinate로 변환
        h_point = np.append(xy_point, 1)
        h_points.append(h_point)
    h_points = np.array(h_points)

    xy_points_out = []
    for h_point in h_points:
        # homography matrix를 곱하여 reference coordinate로 변환
        xy_point_out = h.dot(h_point)
        nomalizer = xy_point_out[2] if (xy_point_out[2] != 0) else 1e-10
        # additional dimension을 없애고 그 값으로 나누어 projection
        xy_point_out = xy_point_out[:2] / nomalizer
        xy_points_out.append(xy_point_out)
    xy_points_out = np.array(xy_points_out)
    # END
    return xy_points_out
```

---

Source image를 Reference image의 coordinate로 변환하기 위해 먼저, Source image를 Homogeneous coordinate로 변환하고, 여기에 Homography matrix를 곱하여 Reference image의 Coordinate로 변환하였다. 마지막으로 Additional dimension을 없애면서 해당 값으로 나누어 Nomalize 해주었다.

## 2.2 RANSAC

Panorama를 생성하기 위해서 Image 간의 Feature를 매치하지만, Outlier가 생길 수 있고, 이 Outlier가 Panorma를 생성하는 데 큰 장애물이 될 수 있기 때문에, RANSAC을 이용하여 Outlier를 제거한다.

---

```
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
    minMatchCount = 4 # homography matrix를 구하기 위해 최소 4 개의
                       # match가 필요
    maxCount = 0
    maxMatches = 0
    for k in range(num_iter):
        matches = []
        # random으로 4 개의 match 선택
        for _ in range(minMatchCount):
            match = random.randint(0, len(xy_src) - 1)
            matches.append(match)
        # 선택한 match로 homography matrix를 구하고 proejction
        h, _ = cv2.findHomography(xy_src[matches], xy_ref[matches])
        xy_proj = KeypointProjection(xy_src, h)

        inlierCount = 0
        # 모든 match를 순회하여 inlier counting
        for i in range(xy_proj.shape[0]):
            dist = np.linalg.norm(xy_proj[i] - xy_ref[i])
            if dist < tol:
                inlierCount += 1
        # inlier가 가장 많은 match set 선택
        if inlierCount > maxCount:
            maxCount = inlierCount
            maxMatches = matches

    # inlier가 가장 많은 match set으로 homography matrix를 구하고
    # proejction
    h, _ = cv2.findHomography(xy_src[maxMatches], xy_ref[maxMatches])
    xy_proj = KeypointProjection(xy_src, h)
    # inlier 추출
```

```

inliers = []
for i in range(xy_proj.shape[0]):
    dist = np.linalg.norm(xy_proj[i] - xy_ref[i])
    if dist < tol:
        inliers.append(i)
# inlier를 가지고 다시 homography matrix를 구해서 return
h, _ = cv2.findHomography(xy_src[inliers], xy_ref[inliers])
# END
assert isinstance(h, np.ndarray)
assert h.shape == (3, 3)
return h

```

---

수행한 RANSAC의 단계는 다음과 같다.

1. Matches에서 랜덤으로 4개 이상의 Match 선택
2. 선택한 Match set으로 Homography matrix 생성
3. Euclidean distance를 통해 Inlier counting (tol보다 작을 때)
4. 1 - 3을 num\_iter 번 반복
5. Inlier가 가장 많은 Homography matrix 선택
6. 새로운 Homography matrix에서 Outlier 제거
7. 추출된 Inlier로 다시 최종 Homography matrix 생성

# Chapter 3

## 결론

### 3.1 Keypoint Projection

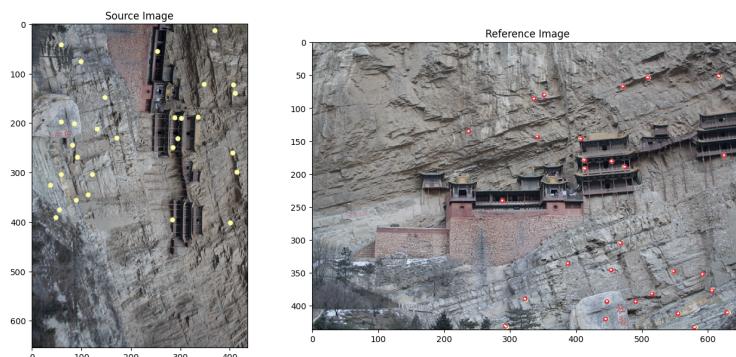


Figure 3.1: Keypoint Projection

Source Image의 Feature들이 Reference frame으로 Projection 되어 매칭된 것을 확인할 수 있다.

### 3.2 RANSAC

Fountain의 경우 num\_iter=70, tol=20으로 설정하였을 때 좋은 Panorama image를 얻을 수 있었다.

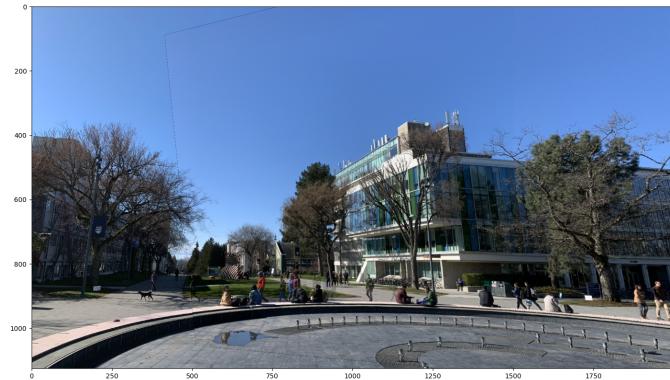


Figure 3.2: fountain



Figure 3.3: garden

Garden의 경우 num\_iter=400, tol=15으로 설정하였을 때 좋은 Panorama image를 얻을 수 있었다.

Irving의 경우 Outlier Ratio는 num\_iter=70, tol=20으로 설정하였을 때 좋은 Panorama image를 얻을 수 있었다.



Figure 3.4: irving

### 3.3 Parameter setting

$$N >= \frac{\log(1-p)}{\log(1-(1-e)^s)}$$

- p : Target Accuracy
- e : Proportion of outliers
- s : Least sample size

Interation number N은 위와 같은 수식으로 결정할 수 있다. Outlier ratio는 설정한 Interation number와 Tolerance에 따라 달라질 수 있기 때문에, 자동화를 통해 따라 여러 값을 테스트해 봐야 할 것이다. Accuracy rate를 0.7 이상으로 설정하여 구한 Interation number를 사용하였을 때, 시각적으로 잘 연결된 Panorama image를 얻을 수 있었다.