비주얼 오도메트리와 증강 현실

중간시험과제: automatic stitching of two images

120230252 허승회

컴퓨터공학과

1. Introduction

   본 과제에서는 컴퓨터 비전 분야의 두 가지 핵심 알고리즘 ORB와 RANSAC을 사용한 파노라마 이미지 스티칭 기술을 구현하는 것을 목적으로 합니다.

2. Method

   A. Input images

      i. Set1


      ii. Set2


B. ORB feature extract

```
orb = cv2.ORB_create()
kp1, des1 = orb.detectAndCompute(img1, None)
kp2, des2 = orb.detectAndCompute(img2, None)
```

Opencv에 존재하는 ORB_create 라이브러리를 사용하여 input image들의 key point를 추출하였다.

ORB 알고리즘은 Oriented FAST and Rotated BRIEF의 약자로 이미지에서 빠르게 특징점을 추출하고 설명하는데 사용되는 알고리즘이다. ORB는 기본적으로 FAST 키포인트 탐지기와 BRIEF 특징점 기술자에 기반을 두며 회전 불변성과 빠르고 효율적이라는 특징이 있다.

C. bruteforce matching with Hamming distance

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1, des2)
matches = sorted(matches, key=lambda x: x.distance)
```

ORB와 마찬가지로 Bruteforce matcher 또한 opencv의 라이브러리를 사용했다. ORB를 통해 추출한 point들의 거리를 Hamming distance를 사용하여 일치하는 point를 찾아냈으며 distance 값을 기준으로 정렬했다.

D. Homography

```
def compute_homography(src_pts, dst_pts):
    A = []
    for (x1, y1), (x2, y2) in zip(src_pts, dst_pts):
        A.append([-x1, -y1, -1, 0, 0, 0, x1*x2, y1*x2, x2])
        A.append([0, 0, 0, -x1, -y1, -1, x1*y2, y1*y2, y2])
    A = np.array(A)
    _, _, Vt = np.linalg.svd(A)
    H = Vt[-1].reshape(3, 3)
    return H / H[2, 2]
```

4개의 대응점을 사용하여 homography 행렬을 계산하는 함수를 구현한다.

E. RANSAC

```python
def ransac_for_homography(matches, kp1, kp2, threshold=5.0, iterations=1000):
    src_pts = np.float32([kp1[m.queryIdx].pt for m in matches])
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in matches])

    max_inliers = []
    final_H = None

    if len(src_pts) < 4:
        print("Not enough points to find homography!")
        return None

    for i in range(iterations):
        # Randomly select 4 pairs of points
        indices = np.random.choice(len(src_pts), 4, replace=False)
        src = src_pts[indices]
        dst = dst_pts[indices]

        # Compute the homography matrix
        H = compute_homography(src, dst)

        # Calculate inliers
        transformed_pts = np.dot(H, np.vstack((src_pts.T, np.ones(src_pts.shape[0]))))
        transformed_pts /= transformed_pts[2]
        transformed_pts = transformed_pts[:2].T

        dist = np.sqrt(np.sum((dst_pts - transformed_pts)**2, axis=1))
        inliers = [idx for idx, d in enumerate(dist) if d < threshold]

        if len(inliers) > len(max_inliers):
            max_inliers = inliers
            final_H = H

    good_matches = [matches[i] for i in max_inliers]

    return final_H, good_matches
```

C에서 얻은 match points와 두 이미지의 특징점을 입력으로 받아 사용한다. 이후 D에서 정의한 함수를 하용해서 4개의 대응점에 해당하는 homography 행렬을 계산한다. 해당 행렬을 사용해서 src point를 변환하고, dst points 사이의 거리가 threshold보다 작으면 inlier로 간주하여 사용한다.

RANSAC 알고리즘은 이상치가 포함된 데이터에서 모델 파라미터를 견고하게 추정하기 위한 반복적인 방법이다. RANSAC은 아래와 같은 작동 방식을 가지고 있으며 이상치에 견고하고 모델에 특정하지 않다는 특징이 있다.

➔ 데이터 셋에서 임의로 최소한의 데이터 포인트를 선택해 모델을 적합시킴

➔ 적합된 모델에 대해 모든 데이터 포인트를 테스트하여 인라이어 집합을 탐색

➔ 임의의 선택과 적합, 테스트 과정을 iteration만큼 반복

➔ 가장 많은 인라이어를 가진 모델을 최종 모델로 선택

i. Set1 match points image

ii. Set2 match points image

F. Warp images

```python
def apply_homography(pt, H):
    x, y = pt
    transformed_pt = np.dot(H, [x, y, 1])
    transformed_pt /= transformed_pt[2]
    return transformed_pt[:2]

def warp_image_simple(img, H, output_shape):
    h, w, _ = img.shape
    output_img = np.zeros(output_shape, dtype=np.uint8)

    H_inv = np.linalg.inv(H)
    for y in range(output_shape[0]):
        for x in range(output_shape[1]):
            src_x, src_y = apply_homography([x, y], H_inv)
            src_x, src_y = int(src_x), int(src_y)
            if 0 <= src_x < w and 0 <= src_y < h:
                output_img[y, x] = img[src_y, src_x]

    return output_img

def warpImages(img1, img2, H):
    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]

    img1_points = np.float32([[0, 0], [0, h1], [w1, h1], [w1, 0]])
    temp_points = np.float32([[0, 0], [0, h2], [w2, h2], [w2, 0]])

    img2_points = np.array([apply_homography(pt, H) for pt in temp_points])

    points = np.vstack([img1_points, img2_points])

    [x_min, y_min] = np.int32(np.min(points, axis=0))
    [x_max, y_max] = np.int32(np.max(points, axis=0))

    output_shape = (y_max - y_min + 1, x_max - x_min + 1, 3)

    H_translation = np.array([
        [1, 0, -x_min],
        [0, 1, -y_min],
        [0, 0, 1]
    ])

    final_H = np.dot(H_translation, H)
    output_img = warp_image_simple(img2, final_H, output_shape)
    output_img[-y_min:-y_min+h1, -x_min:-x_min+w1] = img1

    return output_img
```

Img1의 모서리와 img2의 모서리를 변환하여 결과 이미지의 최소 및 최대 경계를 계산한다. 변환된 img2가 저장될 출력 이미지의 크기를 결정하고 변환 행렬을 사

용하여 결과 이미지 내 올바른 위치에 img2를 배치한다. 마지막으로 img1을 그대로 복사한다.

i. Apply homography

입력값 pt에 호모그래피 행렬을 적용하여 변환된 포인트 반환

ii. Warp image simple

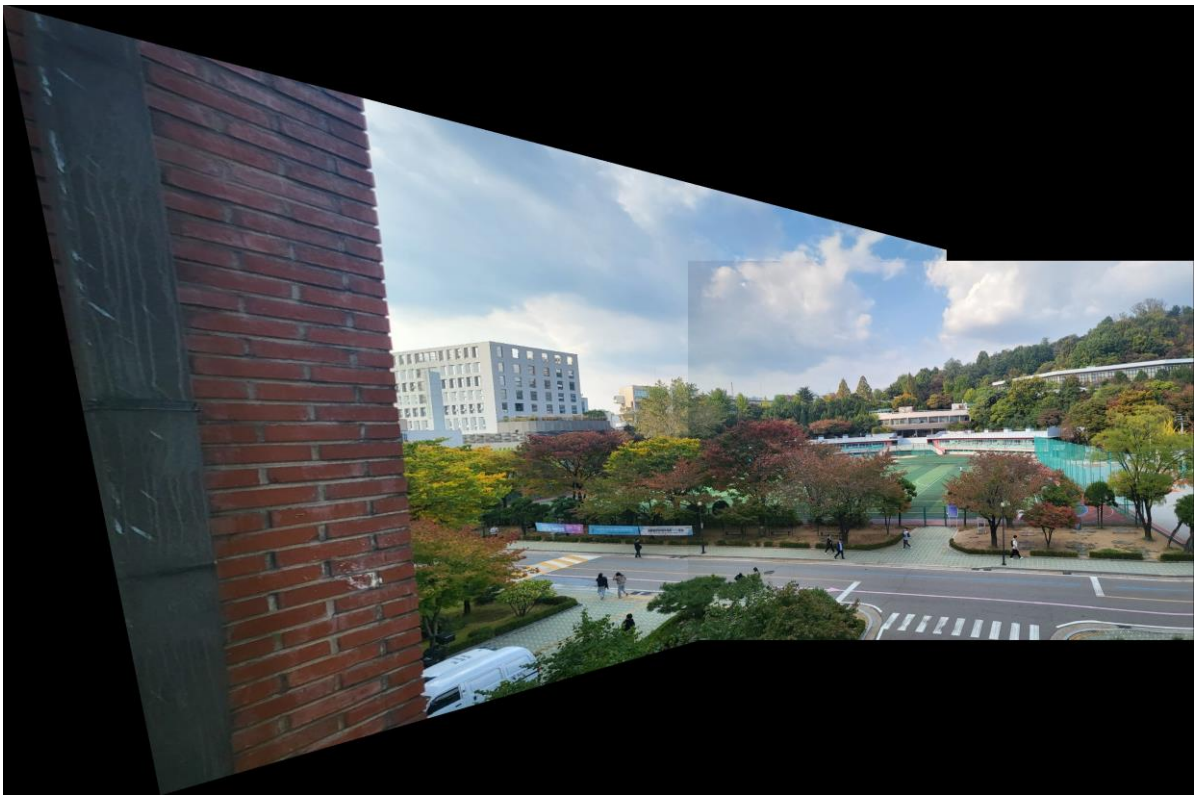
입력된 이미지에 homography 행렬을 적용하여 변환된 이미지 반환

iii. WarpImages

입력된 두 이미지를 homography 행렬을 사용하여 합성하고 결과물을 반환

3. Results

A. Set1 Results



B. Set2 Results