Visual Odometry # Project01

2D homography computation with ORB and RANSAC

120230455 강필재

1. Compute ORB keypoint and descriptors and Bruteforce matching with Hanmming distance

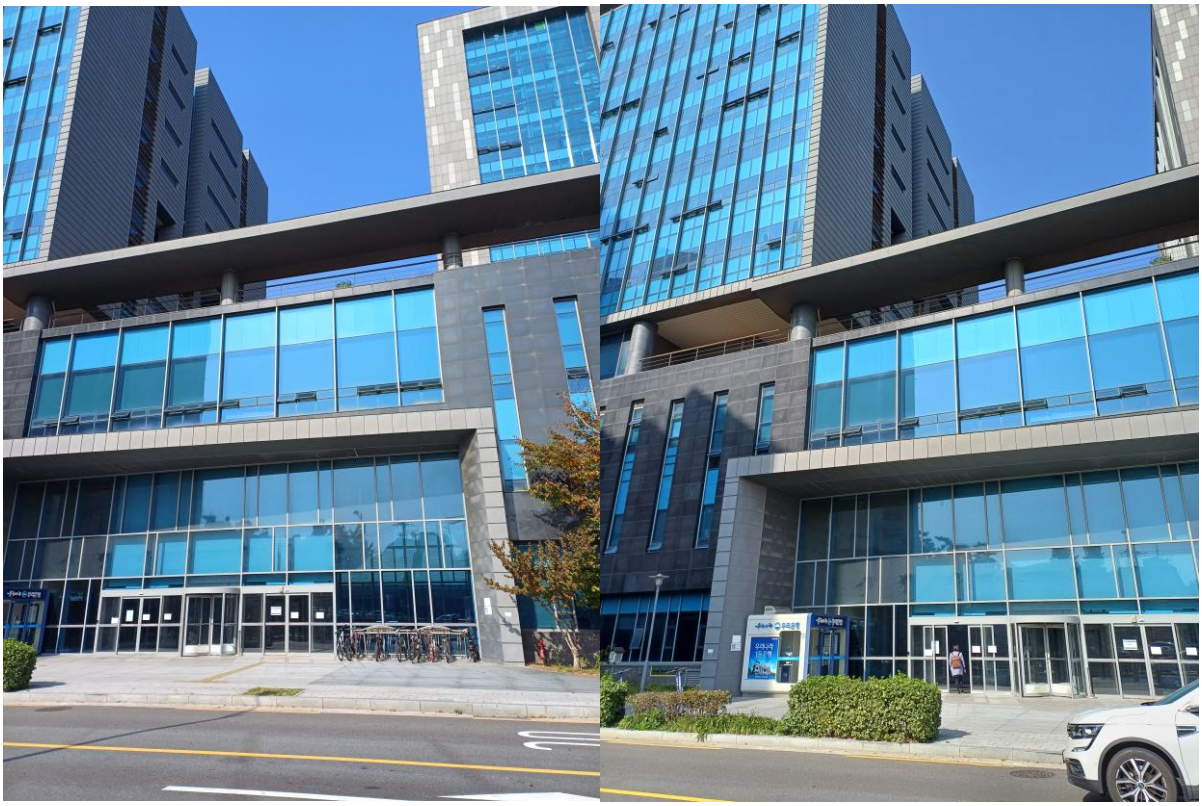

figure1. [left] image1, [right] image2

input image 로 figure1.을 사용하였다. 그리고 두 image 의 keypoint 를 찾는 알고리즘으로 ORB 를 사용하였다. 코드는 다음과 같다.

```
orb_detector = cv2.ORB_create()
#created our ORB(Oriented FAST and Rotated BRIEF) detector

key_points1, descriptors1 = orb_detector.detectAndCompute(image1_gray, None)
# obtained key points and descriptions for first gray image
key_points2, descriptors2 = orb_detector.detectAndCompute(image2_gray, None)
# obtained key points and descriptions for second gray image


#Drawing keypoints for Mountain images
keyImage1 = cv2.drawKeypoints(image1_gray, key_points1, np.array([]), (0, 0, 255))
keyImage2 = cv2.drawKeypoints(image2_gray, key_points2, np.array([]), (0, 0, 255))

cv2.imwrite('keyImage1.jpg', keyImage1)
cv2.imwrite('keyImage2.jpg', keyImage2)
```

Figure2. ORB keypoint and desciptor

figure2.와 같이 Opencv 에서 ORB 내장함수를 활용하여 각 image 의 keypoint 와 descripter 를 계산하였다. 검출의 효과를 올리기 위해 이미지들을 gray scale 로 변화하였다.

figure3. [left] image1 의 keypoint, [right] image2 의 keypoint

figure3.는 image1 과 image2 의 keypoint 를 ORB 로 검출한 결과를 이미지로 표현한 것이다. 이후 위의 keypoint 들을 Hamming distance 를 사용한 Bruteforce matching 를 하였다. 이 또한 opencv 의 내장함수 cv2.BFMatcher 를 사용하였다.

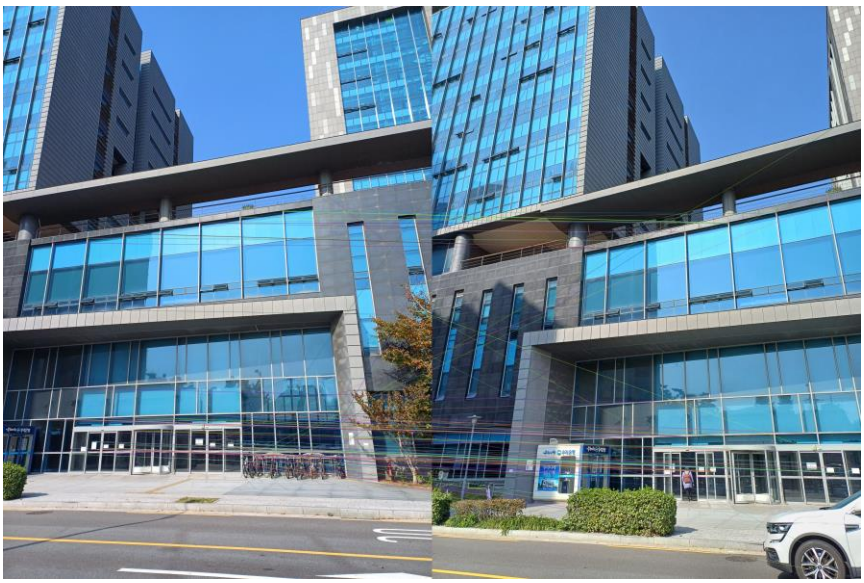

figure4. Bruteforce matching of hamming distance 의 결과

figure4.는 Hamming distance 를 활용한 Bruteforce matching 의 결과이다.

2. Implement RANSAC algorithm to compute the homography matrix

```python
def ransac(kp1, kp2, matches, num_iterations, inlier_threshold):
    H = None
    best_inliers = 0

    for _ in range(num_iterations):
        random_matches = np.random.choice(matches, 4, replace=False)
        h = compute_homography(kp1, kp2, random_matches)

        inliers = 0
        for match in matches:
            src_pt = kp1[match.queryIdx].pt
            dst_pt = kp2[match.trainIdx].pt
            src_pt = np.array([src_pt[0], src_pt[1], 1])
            projected_pt = np.dot(h, src_pt)
            projected_pt /= projected_pt[2]

            dist = np.sqrt((projected_pt[0] - dst_pt[0]) ** 2 + (projected_pt[1] -

            if dist <= inlier_threshold:
                inliers += 1

        if inliers > best_inliers:
            best_inliers = inliers
            H = h

    return H
```

```python
def compute_homography(kp1, kp2, matches):
    src_pts = []
    dst_pts = []

    for match in matches:
        src_pt = kp1[match.queryIdx].pt
        dst_pt = kp2[match.trainIdx].pt
        src_pts.append(src_pt)
        dst_pts.append(dst_pt)

    src_pts = np.array(src_pts, dtype=np.float32)
    dst_pts = np.array(dst_pts, dtype=np.float32)

    A = []
    for i in range(4):
        x, y = src_pts[i]
        u, v = dst_pts[i]
        A.append([x, y, 1, 0, 0, 0, -x * u, -y * u, -u])
        A.append([0, 0, 0, x, y, 1, -x * v, -y * v, -v])

    A = np.array(A)
    _, _, V = np.linalg.svd(A)

    H = V[-1].reshape(3, 3)
    H /= H[2, 2]

    return H
```

figure5. RANSAC 함수 구현 [left] ransac function [right] homography matrix 계산

figure5.는 최적의 Homography matrix 를 찾기 위한 RANSAC 함수이다. Iteration 은 1000, inlier threshold 는 5.0 으로 설정하였다. 우선 homography 를 구하기 위해서는 4 개의 point 가 필요하므로 random 함수를 사용하여 4point 를 무작위로 고른뒤에 compute_homography 함수를 호출하여 parameter 로 준 4point 를 이용하여 homography matrix 를 구한다. 그리고 이를 image1 에 적용한 결과와 image2 의 결과를 비교하여 distance 를 구한다. 이 distance 가 inlier threshold 보다 작으면 inlier 이다. 이 inlier 의 개수를 구하여 가장 inlier 의 개수가 많은 homography matrix 를 구하는 것이 바로 RANSAC 을 사용한 최적의 homography matrix 를 구하는 알고리즘이다.

3. Warpping 2 image to panorama image using homography matrix

```python
def warp_perspective(image1, image2, H):

    pts1 = np.float32([[0, 0], [0, image2.shape[0]], [image2.shape[1], image2.shape[0]], [image2.shape[1], 0]]).reshape(-1, 1, 2)
    pts2 = np.float32([[0, 0], [0, image1.shape[0]], [image1.shape[1], image1.shape[0]], [image1.shape[1], 0]]).reshape(-1, 1, 2)
    pts2_ = cv2.perspectiveTransform(pts2, homographyMat)
    pts = np.concatenate((pts1, pts2_), axis=0)

    #Finding the minimum and maximum coordinates
    [xmin, ymin] = np.int32(pts.min(axis=0).ravel() - 0.5)
    [xmax, ymax] = np.int32(pts.max(axis=0).ravel() + 0.5)

    #Translating
    Ht = np.array([[1, 0, -xmin],
                   [0, 1, -ymin],
                   [0, 0, 1]])

    #Warping the first image on the second image using Homography Matrix
    result = cv2.warpPerspective(image1, Ht.dot(homographyMat), (xmax-xmin, ymax-ymin))
    result[-ymin:image2.shape[0]-ymin, -xmin:image2.shape[1]-xmin] = image2

    return result
```

figure6. Use a homographic matrix to warp two images with panoramic images

figure6.은 두 이미지를 panoramic 이미지로 만들기 위해 homography matrix 를 사용하여 새로운 크기의 이미지를 생성하고 image1 을 homography matrix 연산을 통해 새로운 이미지에 투영하고 나머지 image2 를 이어 붙였다.

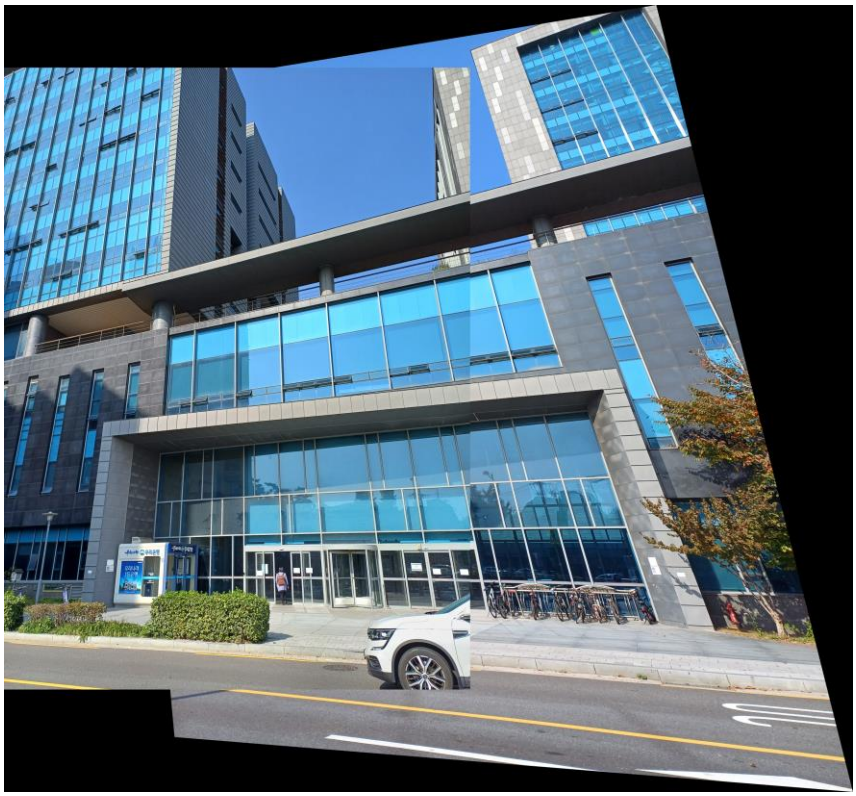

figure7. Panorama image

figure 7.은 최종적으로 얻은 panorama image 의 결과이다.