컴퓨터공학과 201702081 최재범 6주차_과제

- 구현 코드 및 설명

◆ L2 distance : L2 distance 공식을 그대로 사용

```
def L2_distance(vector1, vector2):
    '''
    # TODO
    #vector1과 vector2의 거리 구하기 (L2 distance)
    #distance 는 스칼라
    #np.sqrt(), np.sum() 를 잘 활용하여 구하기
    #L2 distance를 구하는 내장함수로 거리를 구한 경우 감점
    '''
    distance = np.sqrt( np.sum((vector1 - vector2) ** 2) )
    return distance
```

◆ A, B 완성 : 변환 전 과 후 좌표에 각각 해당하는 A, B 행렬을 채움

```
for idx, point in enumerate(points):
    '''
    #ToDo
    #A, B 완성
    # A.append(???) 이런식으로 할 수 있음
    # 결과만 잘 나오면 다른방법으로 해도 상관없음
    '''
    A.append( [point[0][0], point[0][1], 1, 0, 0, 0] )
    A.append( [0, 0, 0, point[0][0], point[0][1], 1] )
    B.append( [point[1][0]] )
    B.append( [point[1][1]] )
```

◆ X 완성 : Ax = b 에서 x를 구하기 위해 공식 사용 : $(A^TA)^{-1}A^Tb$

```
'''
#ToDo
#X 완성
#np.linalg.inv(V) : V의 역행렬 구하는것
#np.dot(V1, V2) : V1과 V2의 행렬곱
# V1.T : V1의 transpose
'''
atainv = np.linalg.inv(np.dot(np.transpose(A), A))
atb = np.dot(np.transpose(A), B)

X = np.dot(atainv, atb)
```

◆ M 완성 : 위에서 공식으로 구한 X의 요소들을 재배치 하고 0, 0, 1 추가

```
'''
# ToDo
# 위에서 구한 X를 이용하여 M 완성
'''
M = [ [X[0][0], X[1][0], X[2][0]],
      [X[3][0], X[4][0], X[5][0]],
      [0, 0, 1]          ]
```

◆ Dst 완성 : bilinear(Backward) 방식으로 목표 이미지 완성. 매칭되지 않는 부분은 검정색으로 나타냄

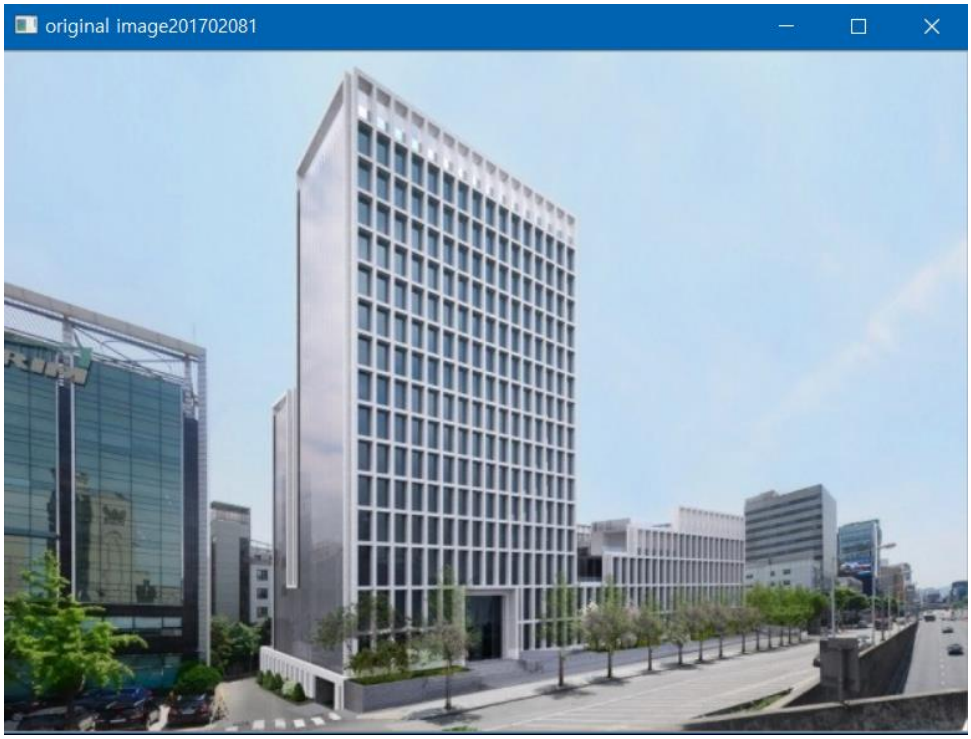
```
'''
# ToDo
# backward 방식으로 dst완성
'''
# Backward 방식 : bilinear
src = img1  # 원본
h, w = img1.shape[:2]

h_, w_ = img2.shape[:2]
dst = np.zeros((h_, w_, 3))

for row in range(h_):
    for col in range(w_):
        # bilinear
        vec = np.dot(M_, np.array([[col, row, 1]]).T)
        c = vec[0, 0]
        r = vec[1, 0]
        c_left = int(c)
        c_right = min(int(c + 1), w - 1)
        r_top = int(r)
        r_bottom = min(int(r + 1), h - 1)
        s = c - c_left
        t = r - r_top

        if 0 <= r_bottom and r_bottom < h and \
                0 <= r_top and r_top < h and \
                0 <= c_right and c_right < w and \
                0 <= c_left and c_left < w:
            intensity = (1 - s) * (1 - t) * src[r_top, c_left] \
                        + s * (1 - t) * src[r_top, c_right] \
                        + (1 - s) * t * src[r_bottom, c_left] \
                        + s * t * src[r_bottom, c_right]
        else:
            intensity = 0

        dst[row, col] = intensity

dst = dst.astype(np.uint8)
```

◆ 예상 point 구하여, inlier 판단 : M으로 구한 point와 실제 point를 구하고, 둘 사이의 거리를 구하여 threshold와 비교, 카운트

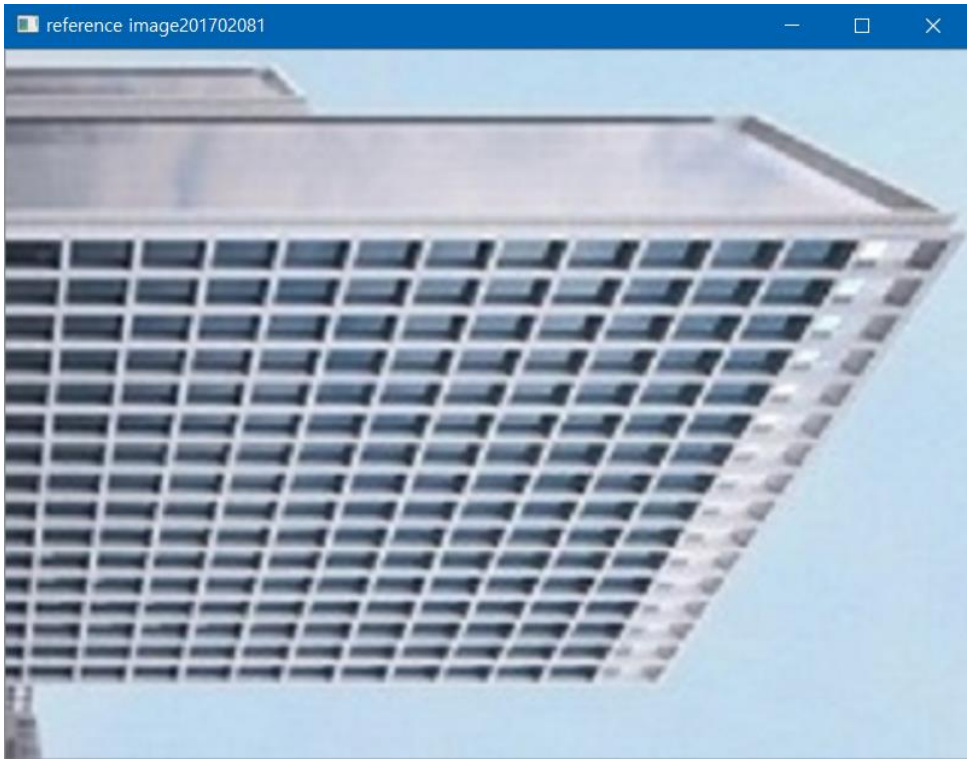
```
count_inliers = 0
for idx, point in enumerate(points):
    '''
    # ToDo
    # 위에서 구한 M으로(3개의 point로 만든 M) 모든 point들에 대하여 예상 point 구하기
    # 구해진 예상 point와 실제 point간의 L2 distance 를 구해서 threshold_distance보다 작은 값이 있는 경우 inlier로 판단
    '''
    # M으로 구한 point
    point_predict = np.dot( M, np.array([point[0][0], point[0][1], 1]) )

    # 실제 point
    point_real = np.array([point[1][0], point[1][1], 1])


    if L2_distance(point_predict, point_real) < threshold_distance:
        count_inliers += 1
```

● 이미지

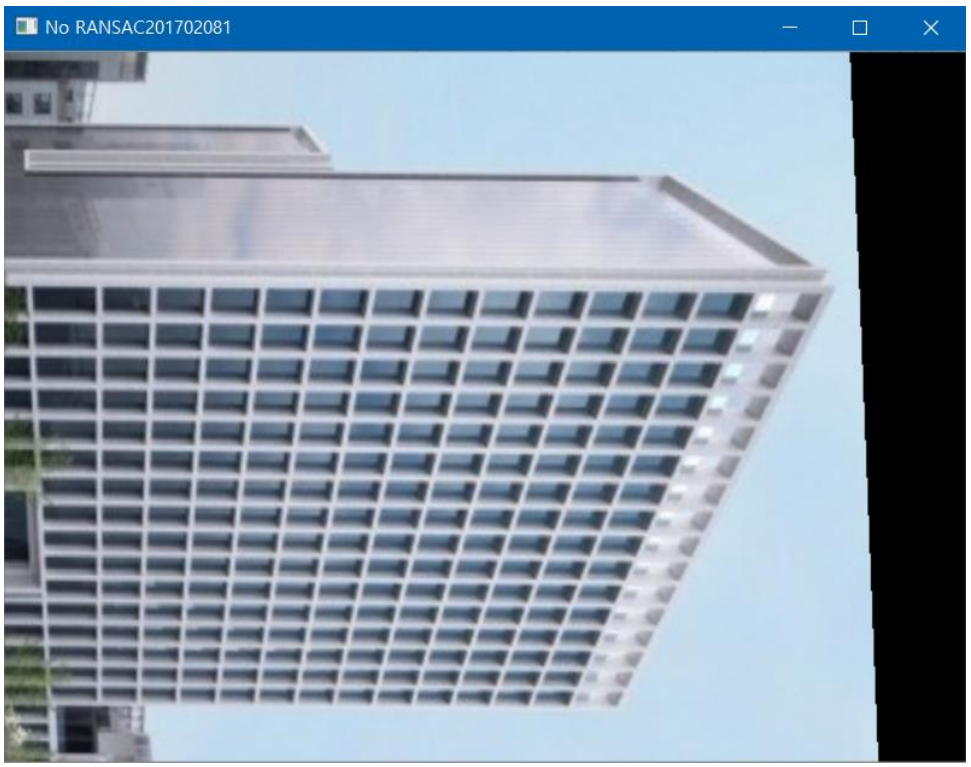
◆ Original



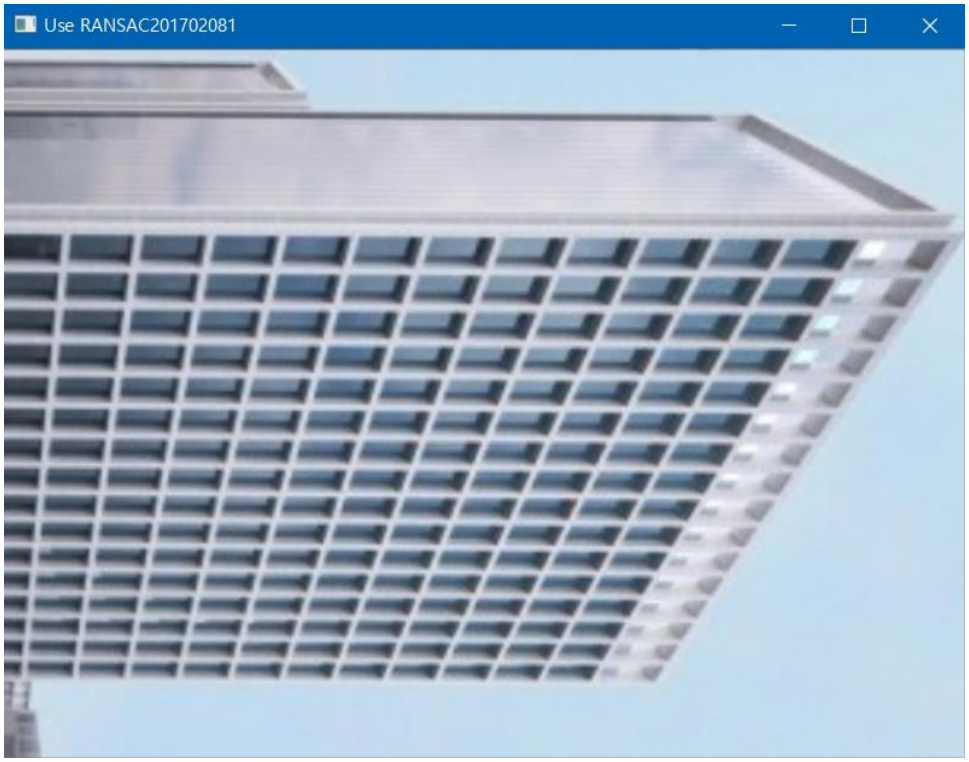
◆ Reference



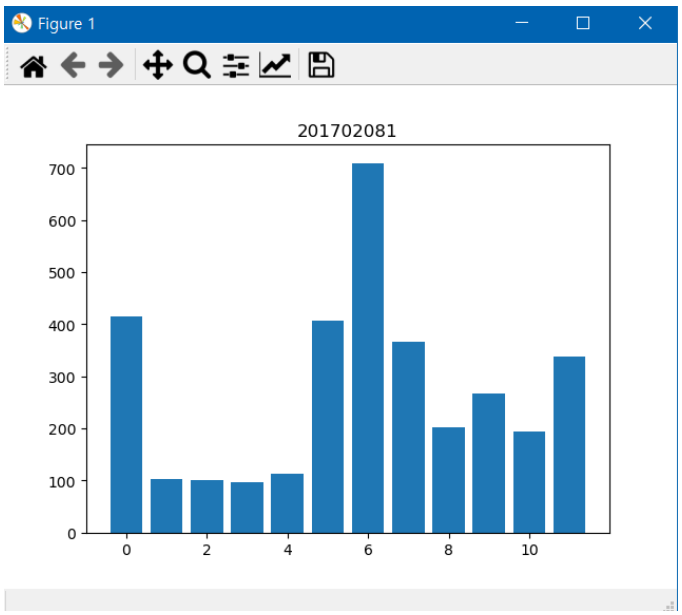
◆ No RANSAC



◆ Use RANSAC



◆ 실습 : 히스토그램



- 느낀 점

빌딩이 자꾸 일부분만 나오는데 어떻게 전체를 나오게 할 지 모르겠다

- 과제 난이도

어려웠다