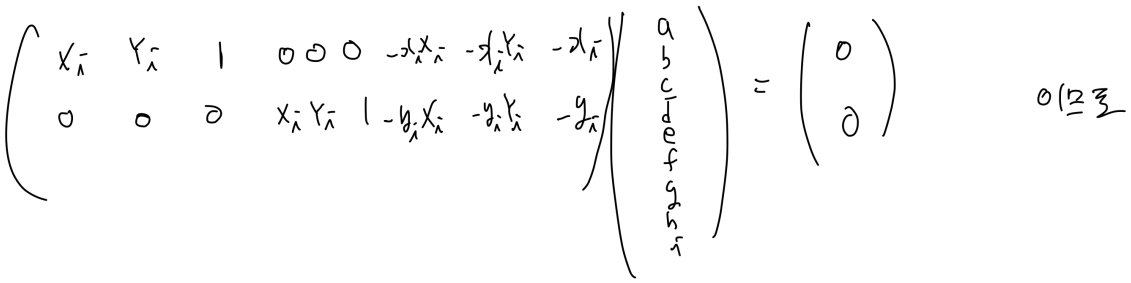
Part 1.

1) H = compute_h(p1,p2)

이번 과제의 이론 부분의 문제 3.(a) 번의 수식



에 따라서 A 행렬 ,h 벡터를 생각하고 Ah = 0 을 Singular Value Decomposition(넘파이의 svd 함수이용)을 A에 적용해서 h 벡터를 찾고, 이를 3 X 3의 H 행렬로 변환하였다. 이때 || h || = 1이다.

```
H = np.zeros((3, 3))
N, _ = p1.shape

# A matrix. ref: theory Question 3 (a)
A = np.zeros((1, 9))
for i in range(N):
    xi = p1[i, 0]
    yi = p1[i, 1]
    Xi = p2[i, 0]
    Yi = p2[i, 1]
    first_row = [Xi, Yi, 1, 0, 0, 0, -xi * Xi, -xi * Yi, -xi]      You, seconds ago • Uncommitted changes
    second_row = [0, 0, 0, Xi, Yi, 1, -yi * Xi, -yi * Yi, -yi]
    two_rows = np.vstack((first_row, second_row))
    if i == 0:
        A = two_rows
    else:
        A = np.vstack((A, two_rows))

_, Sig, V_transposed = np.linalg.svd(A)
V = V_transposed.T

# 이론문제에서 풀었던 문제처럼 Sigma 행렬의 대각 성분 중 가장 작은 eigenvalue를 찾아서, 해당하는 인덱스의 V의 column을 찾는다. 이 열 벡터가 우리가 찾는 h임.
# 이후 h 벡터를 H(행렬)로 변환.
v_col = V[:, -1]
H = np.reshape(v_col, (3, 3))
return H
```

2) H = compute_h_norm(p1, p2)

각 좌표(p1과 p2)에 각각의 최솟값을 빼고, 최댓값 – 최솟값의 크기를 나누어서 정규화하고, 좌표의 값들이 0 ~ 1 사이의 값을 가지도록 만들었다. 이 과정은 정규화 행렬을 사용했고, homogeneous 좌표계에서 이루어졌다. 이후 앞서 구현한 compute_h를 호출하였다. 그 다음 compute_h에서 반환되는 H행렬을 p1좌표를 올바르게 구할 수 있도록 보정해주었다.

Normalization을 포함한 Homograpy 행렬식은 $(normalization_mat_p1)p1 = (H_{normalized})(normalization_mat_p2)p2$인데, 올바른 p1을 구하려면 $p1 = \left(normalization_{mat_{p1}}\right)^{-1}(H_{normalized})(normalization_mat_p2)p2$의 식이 되어야 한다.
따라서 compute_h를 통해 받아온 H_normalized에 $(normalization_mat_p1)^{-1}$을 좌측에 행렬을 곱해주고, 우측에는 $(normalization\_mat\_p2)$ 행렬을 곱해주어서 p1을 정상적으로 구할 수 있는 H를 만들어 주었다.

```python
# normalization matrix
N, _ = p1.shape

max_p1, min_p1 = np.max(p1), np.min(p1)
normalization_mat_p1 = np.array(
    [[1.0, 0.0, -min_p1], [0.0, 1.0, -min_p1], [0.0, 0.0, (max_p1 - min_p1)]]
) / (max_p1 - min_p1)
max_p2, min_p2 = np.max(p2), np.min(p2)
normalization_mat_p2 = np.array(
    [[1.0, 0.0, -min_p2], [0.0, 1.0, -min_p2], [0.0, 0.0, (max_p2 - min_p2)]]
) / (max_p2 - min_p2)
normalized_p1 = np.zeros(p1.shape)
normalized_p2 = np.zeros(p2.shape)
for i in range(N):
    x_1, y_1 = p1[i]
    x_2, y_2 = p2[i]
    homogeneous_p1 = np.array([x_1, y_1, 1.0])
    homogeneous_p2 = np.array([x_2, y_2, 1.0])
    normalized_p1[i] = (normalization_mat_p1 @ homogeneous_p1.T)[:2]
    normalized_p2[i] = (normalization_mat_p2 @ homogeneous_p2.T)[:2]

H_normalized = compute_h(normalized_p1, normalized_p2)

# 정규화 해서 구했던 H를 원래 p1를 구할 수 있도록 변형
normalization_mat_p1_inverse = np.linalg.inv(normalization_mat_p1)
norm_mat_inv_mul_H = normalization_mat_p1_inverse @ H_normalized
H = norm_mat_inv_mul_H @ normalization_mat_p2

return H
```

Part 2.

1) set_cor_mosiac()

porto1, porto2 사진에서 대응하는 corner에 해당하는 픽셀을 육안으로 확인하고 6개의 쌍을 선택했다.

```
def set_cor_mosaic():
    """
    H 사이즈 3 x 3
    min correspond pair: 4
    (x,y)
    """
    # TODO 1
    p_in = np.array(
        [[1119, 765], [1254, 964], [980, 451], [1284, 514], [1282, 416], [1258, 257]]
    )
    p_ref = np.array(
        [[373, 769], [509, 954], [224, 440], [536, 517], [535, 423], [512, 268]]
    )

    return p_in, p_ref
```

2) warp_image(igs_in, igs_ref, H)

이 함수는 inverse warping 방법을 사용하여, 구현하였다. 그 과정에서 bilinear interpolation을 사용하였다. igs_merge는 원래 크기의 4배가 되도록, 즉, 가로 길이 2배, 세로 길이 2배로 설정했다. igs_warp는 igs_ref, 즉 proto 2와 같은 크기(1200,1600)이 되도록 생성하였다.

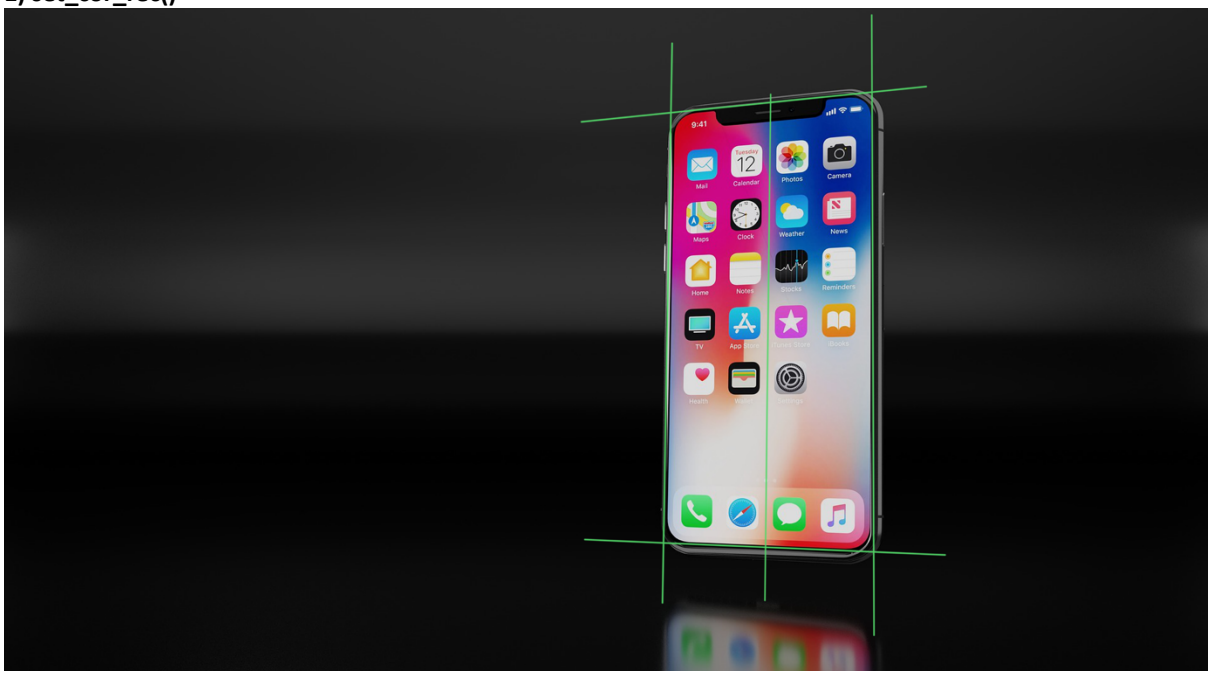
먼저, Inverse warping을 위해 H의 역행렬을 구한 다음, homogeneous 좌표계에서 igs_merge -> igs_in로의 좌표를 찾아내었다. 이렇게 output의 좌표를 모두 고려함으로써, aliasing과 sub-sampling 이슈를 해결할 수 있었다.

```
# proto 2 size
igs_warp = np.zeros((1200, 1600, 3), dtype=np.uint8)
igs_merge = np.pad(
    igs_ref,
    ((600, 600), (1600, 0), (0, 0)),
    mode="constant",
)       You, seconds ago • Uncommitted changes

for r in range(-600, 1800):
    for c in range(-1600, 1600):
        # inverse warping (in -> ref)
        igs_merge_coordinate = np.array([c, r, 1])
        igs_in_coordinate_homo = H_inverse @ igs_merge_coordinate.T
        in_y, in_x, in_z = igs_in_coordinate_homo
        in_x, in_y = in_x / in_z, in_y / in_z

        # bilinear 보간(interpolation)
        if 0 <= in_x < R_in and 0 <= in_y < C_in:
            org_r, org_c = int(np.floor(in_x)), int(np.floor(in_y))

            # (r,c) 기준으로 어디에 위치해 있는지 0~1 스케일로 표현 => (a,b)
            a = in_x - np.floor(in_x)
            b = in_y - np.floor(in_y)
            if org_r + 1 < R_in and org_c + 1 < C_in:
                rc = igs_in[org_r, org_c]
                r1_c = igs_in[org_r + 1, org_c]
                r_c1 = igs_in[org_r, org_c + 1]
                r1_c1 = igs_in[org_r + 1, org_c + 1]

                interpolated_pixel = (
                    (1 - a) * (1 - b) * rc
                    + a * (1 - b) * r1_c
                    + a * b * r1_c1
                    + (1 - a) * b * r_c1
                )

                igs_merge[r + 600, c + 1600] = interpolated_pixel
                # full size warpped image
                # igs_warp[r+600,c +1600] = interpolated_pixel

                # Case warpped size = proto 2 size
                if (0 <= (r) < R_ref) and (0 <= c < C_ref):
                    igs_warp[r, c] = interpolated_pixel

return igs_warp, igs_merge
```

찾아낸 float 형태의 좌표를 정수 형태로 버림(floor)하여 변환한 뒤, 그 좌표의 근처의 3개의 점(r,r+1,c,c+1)과 이루는 사각형에서 수업시간에 배웠던 bilinear interpolation을 적용하여, igs_warp와 igs_merge에 들어갈 픽셀 값을 계산했다. 이 과정에서 igs_merge에는 warp에 들어가는 이미지가 나중에 덮어쓰기 되므로, 과제 요구 조건처럼 igs_warp이 우선적으로 merge에 나타난다.

Part 3.

1) set_cor_rec()



그림과 같이 초록선이 만나는 점 6개를 c_in으로 고른 다음, c_ref는 수평선 상의 가장 왼쪽 점의 y 좌표를 기준으로 설정하였다.(x좌표는 c_in과 동일)

```
def set_cor_rec():
    """
    (x,y): 이미지 상에서 가로, 세로      You, seconds ago • Uncommitted changes
    """
    c_in = np.array(
        [[1063, 166], [1222, 150], [1384, 133], [1051, 852], [1214, 859], [1389, 866]]
    )
    c_ref = np.array(
        [[1063, 166], [1222, 166], [1384, 166], [1051, 852], [1214, 852], [1389, 866]]
    )

    return c_in, c_ref
```

2) rectify(igs, p1, p2)

함수의 파라미터 p1과 p2는 각각 c_in(출발지)과 c_ref(목적지)에 해당한다. 즉 compute_h_norm의 p1(목적지),p2(출발지)와 반대이므로 compute_h_norm(p2,p1)을 호출한다. 그리고 igs와 동일한 크기의 igs_rec(output에 해당)을 생성한다.

이후에는 warp_image함수처럼 igs_ref -> igs_in의 좌표를 inverse warping으로 찾아내고, bilinear interpolation을 수행한다.

```
R, C, _ = igs.shape

for r in range(R):
    for c in range(C):
        # inverse warping (in -> ref)
        igs_ref_coordinate = np.array([c, r, 1])
        igs_in_coordinate_homo = H_inverse @ igs_ref_coordinate.T
        in_y, in_x, in_z = igs_in_coordinate_homo
        in_x, in_y = in_x / in_z, in_y / in_z

        # bilinear 보간(interpolation)
        if 0 <= in_x < R and 0 <= in_y < C:
            org_r, org_c = int(np.floor(in_x)), int(np.floor(in_y))

            # (r,c) 기준으로 어디에 위치해 있는지 0~1 스케일로 표현 => (a,b)
            a = in_x - np.floor(in_x)
            b = in_y - np.floor(in_y)
            if org_r + 1 < R and org_c + 1 < C:
                rc = igs[org_r, org_c]
                r1_c = igs[org_r + 1, org_c]
                r_c1 = igs[org_r, org_c + 1]
                r1_c1 = igs[org_r + 1, org_c + 1]

                interpolated_pixel = (
                    (1 - a) * (1 - b) * rc
                    + a * (1 - b) * r1_c
                    + a * b * r1_c1
                    + (1 - a) * b * r_c1
                )
                igs_rec[r, c] = interpolated_pixel
```