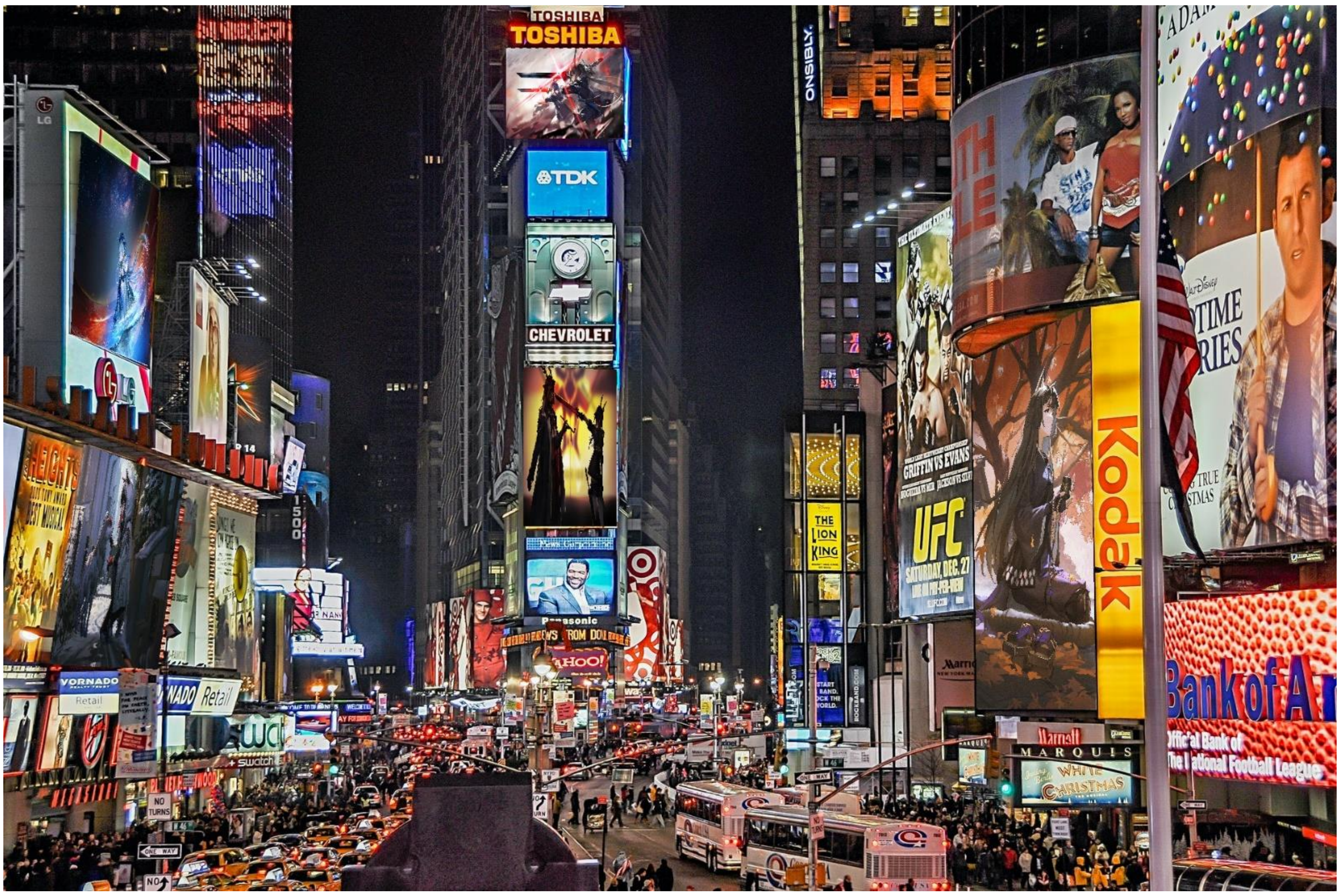
Computer Vision HW3 Report

Student ID: R12921059
Name: 鄧雅文

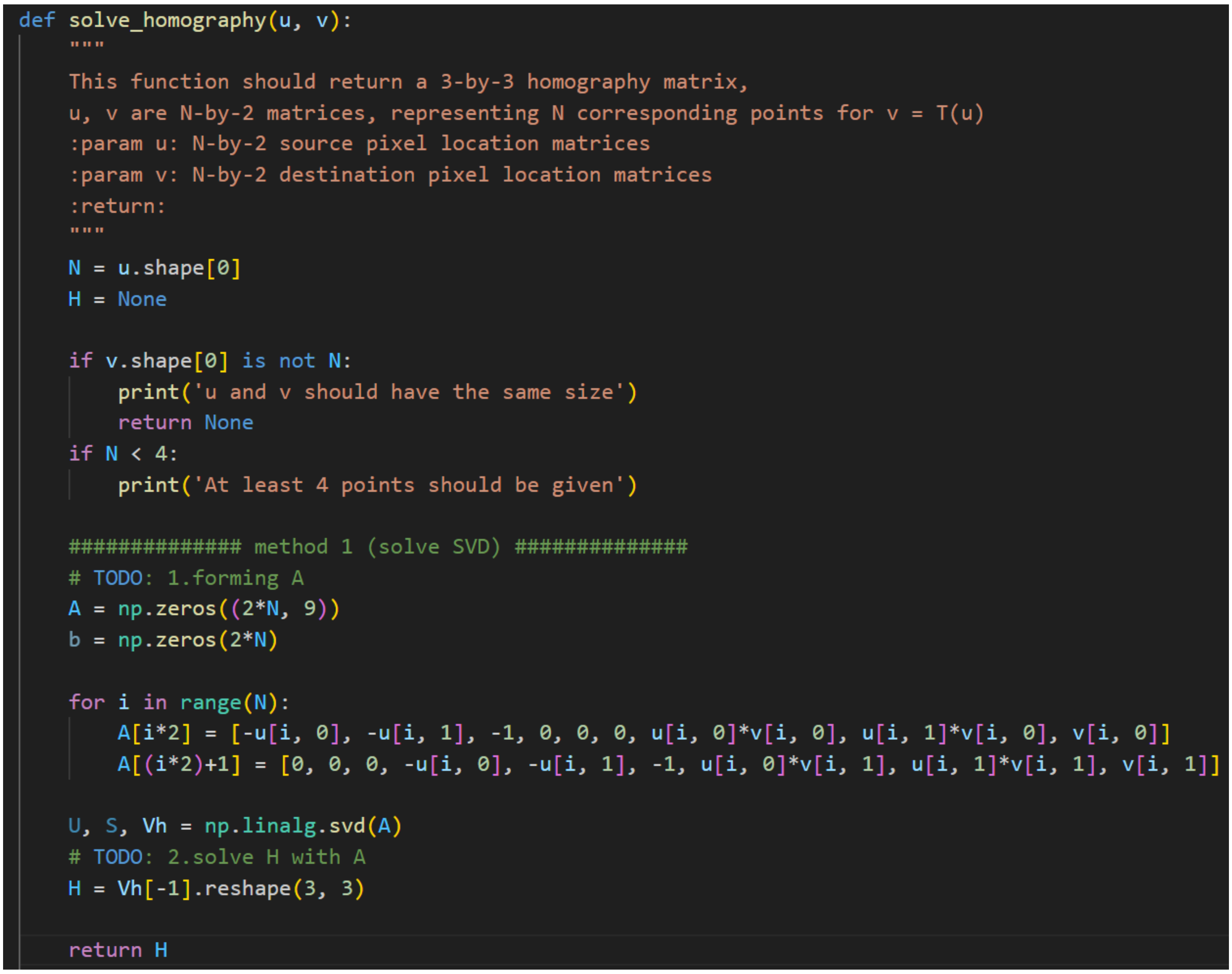
Part 1.

- **Paste your warped canvas**

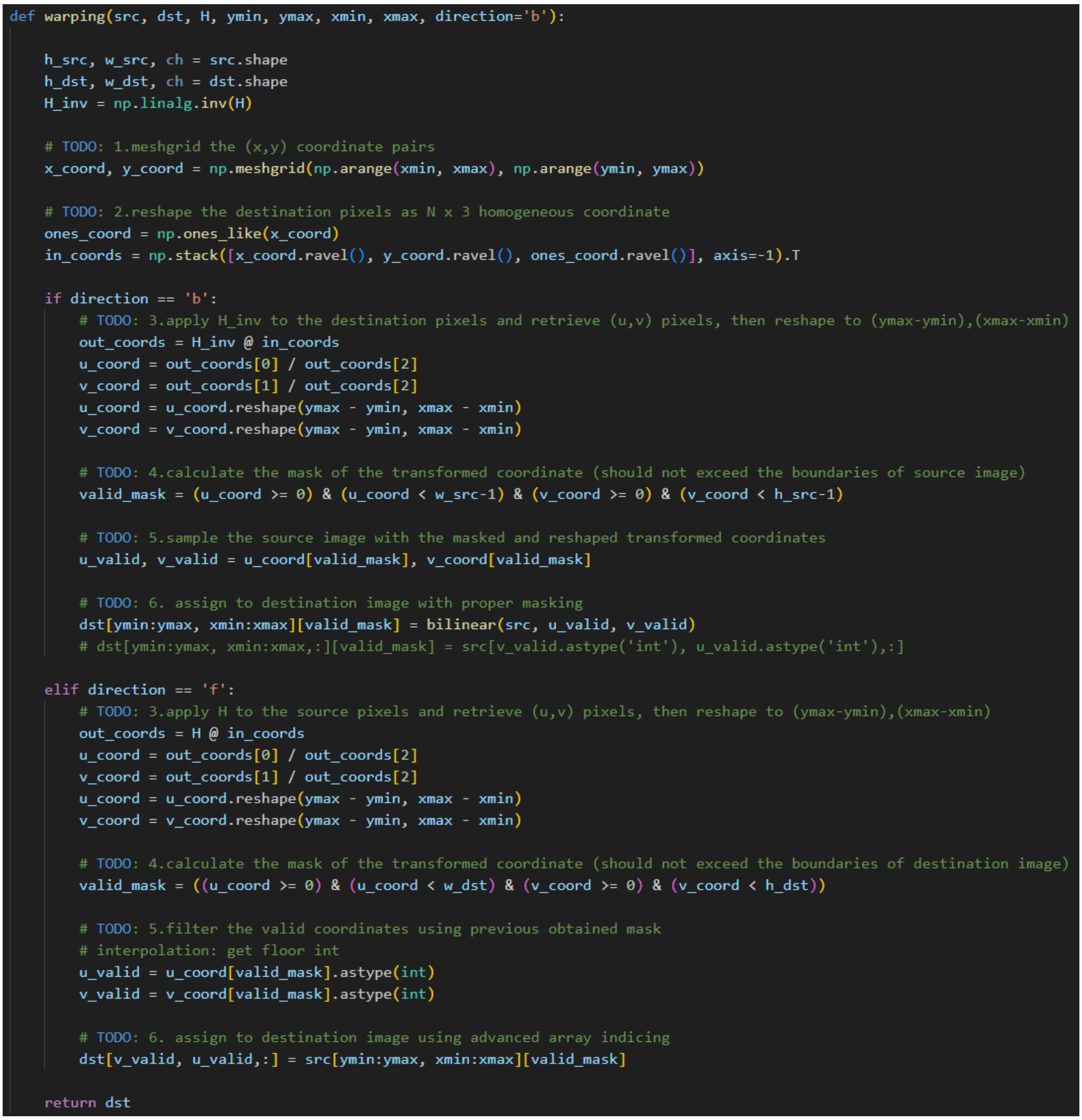


Part 2.

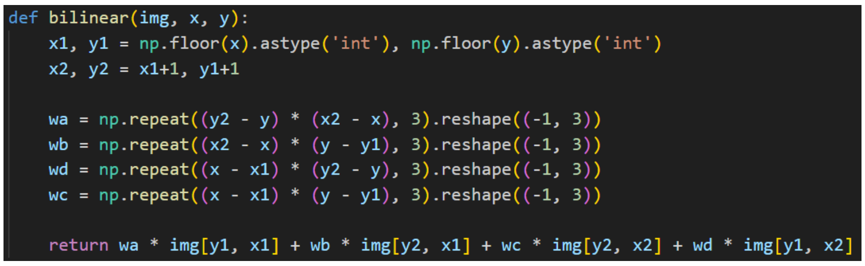
- **Paste the function code *solve_homography(u, v)* & *warping( )* (both forward & backward)**



```python
def solve_homography(u, v):
    """
    This function should return a 3-by-3 homography matrix,
    u, v are N-by-2 matrices, representing N corresponding points for v = T(u)
    :param u: N-by-2 source pixel location matrices
    :param v: N-by-2 destination pixel location matrices
    :return:
    """
    N = u.shape[0]
    H = None

    if v.shape[0] is not N:
        print('u and v should have the same size')
        return None
    if N < 4:
        print('At least 4 points should be given')

    ############## method 1 (solve SVD) ##############
    # TODO: 1.forming A
    A = np.zeros((2*N, 9))
    b = np.zeros(2*N)

    for i in range(N):
        A[i*2] = [-u[i, 0], -u[i, 1], -1, 0, 0, 0, u[i, 0]*v[i, 0], u[i, 1]*v[i, 0], v[i, 0]]
        A[(i*2)+1] = [0, 0, 0, -u[i, 0], -u[i, 1], -1, u[i, 0]*v[i, 1], u[i, 1]*v[i, 1], v[i, 1]]

    U, S, Vh = np.linalg.svd(A)
    # TODO: 2.solve H with A
    H = Vh[-1].reshape(3, 3)

    return H
```



```python
def warping(src, dst, H, ymin, ymax, xmin, xmax, direction='b'):

    h_src, w_src, ch = src.shape
    h_dst, w_dst, ch = dst.shape
    H_inv = np.linalg.inv(H)

    # TODO: 1.meshgrid the (x,y) coordinate pairs
    x_coord, y_coord = np.meshgrid(np.arange(xmin, xmax), np.arange(ymin, ymax))

    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    ones_coord = np.ones_like(x_coord)
    in_coords = np.stack([x_coord.ravel(), y_coord.ravel(), ones_coord.ravel()], axis=-1).T

    if direction == 'b':
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        out_coords = H_inv @ in_coords
        u_coord = out_coords[0] / out_coords[2]
        v_coord = out_coords[1] / out_coords[2]
        u_coord = u_coord.reshape(ymax - ymin, xmax - xmin)
        v_coord = v_coord.reshape(ymax - ymin, xmax - xmin)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)
        valid_mask = (u_coord >= 0) & (u_coord < w_src-1) & (v_coord >= 0) & (v_coord < h_src-1)

        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
        u_valid, v_valid = u_coord[valid_mask], v_coord[valid_mask]

        # TODO: 6. assign to destination image with proper masking
        dst[ymin:ymax, xmin:xmax][valid_mask] = bilinear(src, u_valid, v_valid)
        # dst[ymin:ymax, xmin:xmax,:][valid_mask] = src[v_valid.astype('int'), u_valid.astype('int'),:]

    elif direction == 'f':
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        out_coords = H @ in_coords
        u_coord = out_coords[0] / out_coords[2]
        v_coord = out_coords[1] / out_coords[2]
        u_coord = u_coord.reshape(ymax - ymin, xmax - xmin)
        v_coord = v_coord.reshape(ymax - ymin, xmax - xmin)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
        valid_mask = ((u_coord >= 0) & (u_coord < w_dst) & (v_coord >= 0) & (v_coord < h_dst))

        # TODO: 5.filter the valid coordinates using previous obtained mask
        # interpolation: get floor int
        u_valid = u_coord[valid_mask].astype(int)
        v_valid = v_coord[valid_mask].astype(int)

        # TODO: 6. assign to destination image using advanced array indicing
        dst[v_valid, u_valid,:] = src[ymin:ymax, xmin:xmax][valid_mask]

    return dst
```



```python
def bilinear(img, x, y):
    x1, y1 = np.floor(x).astype('int'), np.floor(y).astype('int')
    x2, y2 = x1+1, y1+1

    wa = np.repeat((y2 - y) * (x2 - x), 3).reshape((-1, 3))
    wb = np.repeat((x2 - x) * (y - y1), 3).reshape((-1, 3))
    wd = np.repeat((x - x1) * (y2 - y), 3).reshape((-1, 3))
    wc = np.repeat((x - x1) * (y - y1), 3).reshape((-1, 3))

    return wa * img[y1, x1] + wb * img[y2, x1] + wc * img[y2, x2] + wd * img[y1, x2]
```

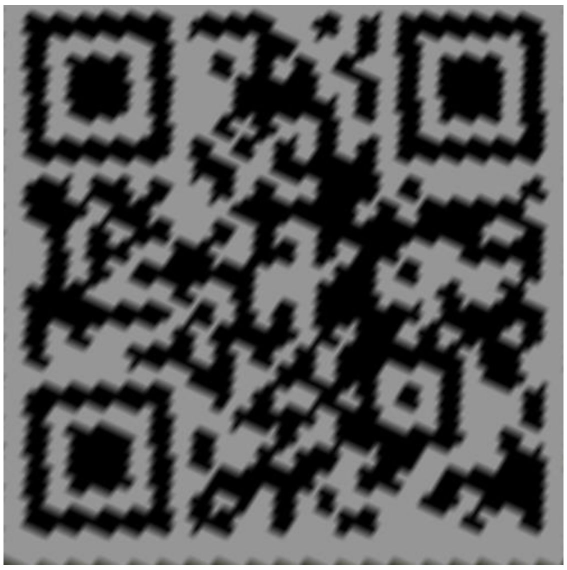
- **Briefly introduce the interpolation method you use**

在 backward warping 使用 bilinear interpolation (參考 Reference [1] [2])，這樣可以讓 part3 的結果看起來比較 smooth 一點。(補充: 若把 call bilinear 那註解，換成下面那行，就會變成直接取整數）
forward warping 的部分則是直接取整數。

Part 3.

- **Paste the 2 warped images and the link you find**

output3_1



output3_2

兩張 QR code 掃出來都是跳同一個連結: http://media.ee.ntu.edu.tw/courses/cv/21S/

- **Discuss the difference between 2 source images, are the warped results the same or different?**

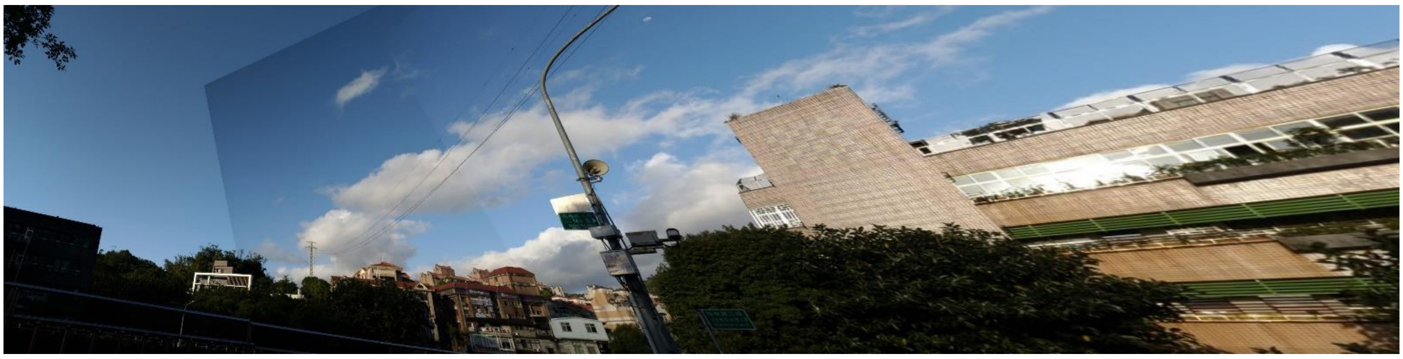
BL_secret2 明顯看上去比較模糊，感覺是因為變形導致些許放大。BL_secret1 的格子天花板因為單點透視，所以在距拍攝者較遠的格子看起來有變小，且拍攝角度還是偏向往正面拍，所以格子的橫向 boarder 看起來還是比較像平行線。BL_secret2 格子的橫向 boarder 則是變形明顯，已經面變成曲線了，應該是鏡頭導致的 Barrel Distortion。
Warp 後的 QR code 掃出來都是跳同一個連結: http://media.ee.ntu.edu.tw/courses/cv/21S/。

- **If the results are the same, explain why. If the results are different, explain why?**

output3_1 & output3_2 模糊程度不同，而且 output3_2 左下角角落的黑邊比 output3_1 多一點。
分析原因: homography matrix 是由四個點得到的，但是 BL_secret2 的變形使得天花板個子邊界已經呈現曲線，單用四個點已經沒辦法非常精準描述，所以使用 homography matrix 做 backward wraping 的結果比較模糊且還是有些微變形。

Part 4.

- **Paste your stitched panorama**



- **Can all consecutive images be stitched into a panorama?**

No

- **If yes, explain your reason. If not, explain under what conditions will result in a failure?**

1. 當圖片變形太多(像是 part3 那種 barrel distortion)，結果會不好
2. 當兩張圖片內的同一個物體若有移動，做 stitching 後會產生 ghost

Reference

[1] https://stackoverflow.com/a/12729229/13904312
[2] https://github.com/Offliners/NTUEE-CV-2022Spring/tree/main/homework3
[3] https://math.stackexchange.com/questions/494238/how-to-compute-homography-matrix-h-from-corresponding-points-2d-2d-planar-homog/4841217#4841217
[4] https://github.com/linrl3/Image-Stitching-OpenCV/blob/new/Image_Stitching.py
[5] https://github.com/Spheluo/Projective-Geometry/blob/main/src/utils.py
[6] https://github.com/Louislar/NTU_CV_HW