Computer Vision HW3 Report

Student ID: R12521601

Name: 詹承諺

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')

    # TODO: 1.forming A
    A = np.zeros((2 * N, 9))
    for i in range(N):
        u_x, u_y = u[i]
        v_x, v_y = v[i]
        A[2*i] = np.array([u_x, u_y, 1, 0, 0, 0, -u_x*v_x, -u_y*v_x, -v_x])
        A[2*i+1] = np.array([0, 0, 0, u_x, u_y, 1, -u_x*v_y, -u_y*v_y, -v_y])

    # TODO: 2.solve H with A
    u, sigma, vt = np.linalg.svd(A, full_matrices=True)
    H = vt[-1]
    H = H.reshape(3, 3)
    H = H / H[2, 2]
    return H
```

```python
def warping(src, dst, H, ymin, ymax, xmin, xmax, direction='b'):
    """
    Perform forward/backward warpping without for loops. i.e.
    for all pixels in src(xmin~xmax, ymin~ymax),  warp to destination
          (xmin=0,ymin=0)  source                      destination
                          |--------|              |------------------------|
                          |        |              |                        |
                          |        |     warp     |                        |
    forward warp          |        |  ---------->  |                        |
                          |        |              |                        |
                          |--------|              |------------------------|
                                (xmax=w,ymax=h)

    for all pixels in dst(xmin~xmax, ymin~ymax),  sample from source
                             source                      destination
                          |--------|              |------------------------|
                          |        |              | (xmin,ymin)            |
                          |        |     warp     |           |--|         |
    backward warp         |        |  <----------  |           |__|         |
                          |        |              |             (xmax,ymax)|
                          |--------|              |------------------------|

    :param src: source image
    :param dst: destination output image
    :param H:
    :param ymin: lower vertical bound of the destination(source, if forward warp) pixel coordinate
    :param ymax: upper vertical bound of the destination(source, if forward warp) pixel coordinate
    :param xmin: lower horizontal bound of the destination(source, if forward warp) pixel coordinate
    :param xmax: upper horizontal bound of the destination(source, if forward warp) pixel coordinate
    :param direction: indicates backward warping or forward warping
    :return: destination output image
    """

    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, y = np.meshgrid(np.arange(xmin, xmax), np.arange(ymin, ymax))

    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    homo = np.vstack((x.flatten(), y.flatten(), np.ones(x.size))).astype(np.int32)

    if direction == 'b':
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        src_pixels = np.dot(H_inv, homo)
        src_pixels = src_pixels / src_pixels[2, :]
        src_pixels = np.round(src_pixels[:2, :].T).astype(np.int32)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)
        mask = (src_pixels[:, 0] >= 0) & (src_pixels[:, 0] < w_src) & (src_pixels[:, 1] >= 0) & (src_pixels[:, 1] < h_src)

        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
        src_pixels_with_mask = src_pixels[mask]
        coord_homo_with_mask = homo[:, mask]

        # TODO: 6. assign to destination image with proper masking
        dst[coord_homo_with_mask[1, :],coord_homo_with_mask[0, :]] = src[src_pixels_with_mask[:, 1], src_pixels_with_mask[:, 0]]

    elif direction == 'f':
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        dest_pixels = np.dot(H, homo)
        dest_pixels = dest_pixels / dest_pixels[2, :]
        dest_pixels = np.round(dest_pixels[:2, :].T).astype(np.int32)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
        mask = (dest_pixels[:, 0] >= 0) & (dest_pixels[:, 0] < w_dst) & (dest_pixels[:, 1] >= 0) & (dest_pixels[:, 1] < h_dst)


        # TODO: 5.filter the valid coordinates using previous obtained mask
        dest_pixels_with_mask = dest_pixels[mask]
        coord_homo_with_mask = homo[:, mask]

        # TODO: 6. assign to destination image using advanced array indicing
        dst[dest_pixels_with_mask[:, 1], dest_pixels_with_mask[:, 0]] = src[coord_homo_with_mask[1, :], coord_homo_with_mask[0, :]]

    return dst
```

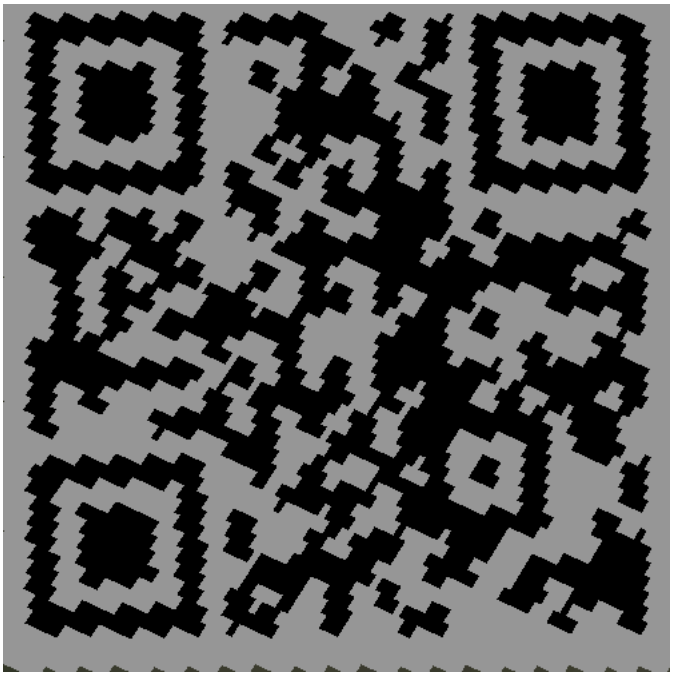
- **Briefly introduce the interpolation method you use**

Interpolation 主要是用 nearest neighbor 的方式。方法是透過 np.round().astype(np.int32)來取最近整數

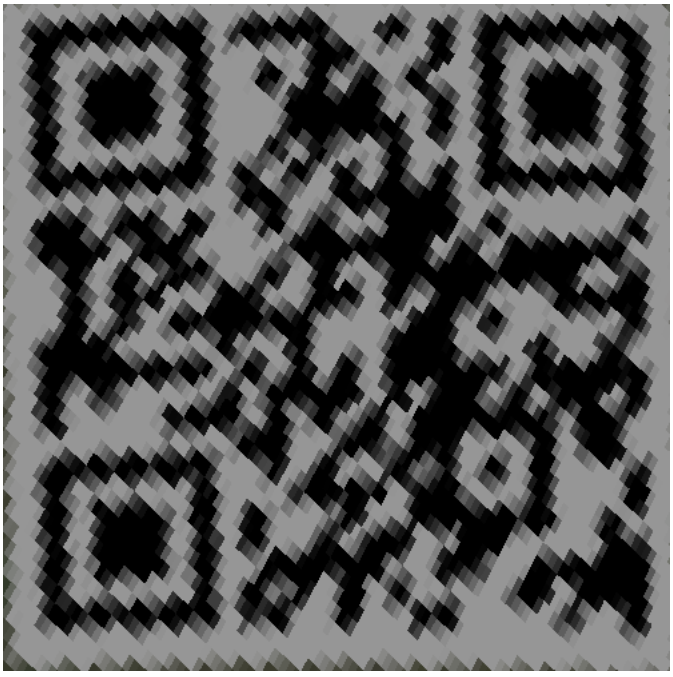
Part 3.

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

Output3_1.png (link: http://media.ee.ntu.edu.tw/courses/cv/21S/)



Output3_2.png



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

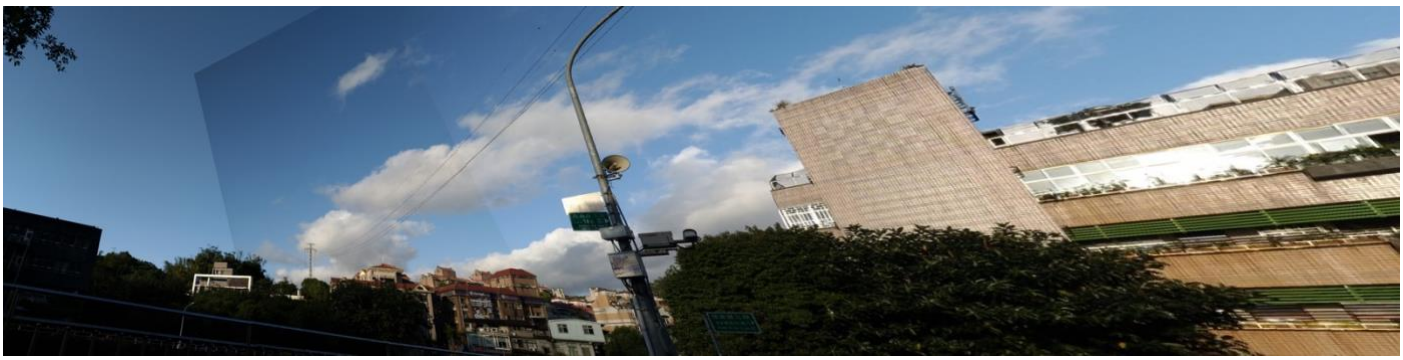
以畫質來說 output3_1 較為清晰，反之，output3_2 較為模糊。但就結果而論兩者皆能成功掃描出同一個網站（**http://media.ee.ntu.edu.tw/courses/cv/21S/**），表示兩張源圖皆能還原成可用的 QR code。

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

從源圖來看的話，第一張 QR code 的圖片形狀較為方正且比例的調整較小。第二張源圖的 QR code 線條較為彎曲且比例有做調整，因此可能會導致轉換效果較第一張來得差。

Part 4.

- **Paste your stitched panorama**



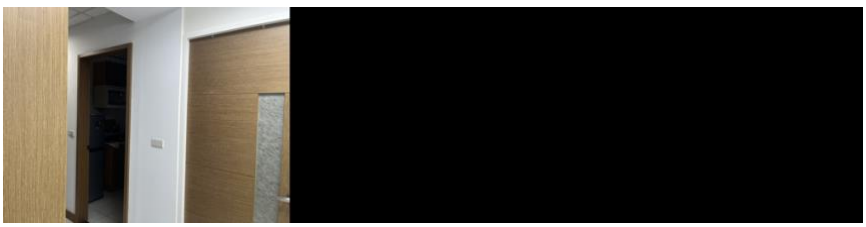
- **Can all consecutive images be stitched into a panorama?**

不是所有的連續圖片都能夠貼成全景圖。

1. Images photoed with camera translation: 以相機有位移的狀況為例

a. 沒有位移：

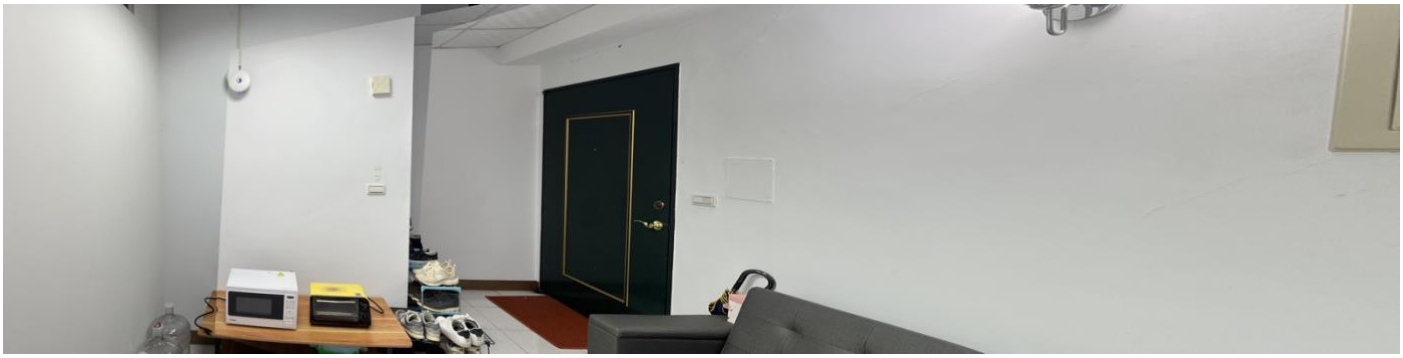
b. 有位移：



可看出有位移可能會導致圖與圖之間無法連接成功。

2. Non-planar scene (scene of in-door view)

將以下三張室內照片，可以獲得下圖的全景照片，因此可知室內場景仍然可用相同方式取得全景照片。



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

雖然作業的三張圖能成功連接成全景圖，但並不是所有的連續圖片都能夠貼成全景圖。最影響成功可能性的因素是照片 overlap 的比例。以投影到平面來說，旋轉的角度不能超過 180 度，否則會無法抓取 feature。也不能平移過多導致無法抓取圖與圖之間的 feature，進而無法連接（如上題的 b，因為平移過多而導致無法成功連接連續圖片）。

而室內場景只要相機不要位移過多或旋轉角度過大，仍可將連續圖片投出全景照片，只要有辦法從連續的圖片中圖與圖之間的共同 feature 抓出來，就可以做出全景照片。