NTU CV HW3

Part 1

1. Paste the function solve_homography( )

```
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')

    # I prefer solution 2
    # TODO: 1.forming A
    A_row_list = []
    for point_u, point_v in zip(u, v):
        # print(point_u, ', ', point_v)
        arr1 = np.array([point_u[0], point_u[1], 1, 0, 0,
0, -1 * point_u[0] * point_v[0], -
1 * point_u[1] * point_v[0], -1 * point_v[0]])
        arr2 = np.array([0, 0, 0, point_u[0], point_u[1],
1, -1 * point_u[0] * point_v[1], -
1 * point_u[1] * point_v[1], -1 * point_v[1]])
```

```
        A_row_list.append(arr1)
        A_row_list.append(arr2)
    A_mat = np.vstack(A_row_list)

    # TODO: 2.solve H with A
    (U, D, V) = np.linalg.svd(A_mat)
    V = np.transpose(V)
    H = V[:, -1]
    H = H.reshape((3, 3))
    return H
```

2. Paste your warped canvas

Part 2

1. **Paste the function code warping( ) (both forward & backward)**

```
def warping(src, dst, H, ymin, ymax, xmin, xmax, directi
on='b'):
    """
    Perform forward/backward warpping without for loops.
 i.e.
    for all pixels in src(xmin~xmax, ymin~ymax),  warp t
o 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 destinati
on(source, if forward warp) pixel coordinate
    :param xmax: upper horizontal bound of the destinati
on(source, if forward warp) pixel coordinate
    :param direction: indicates backward warping or forw
ard 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
    meshgrid_x, meshgrid_y = np.meshgrid(np.arange(xmin,
 xmax), np.arange(ymin, ymax))

    # TODO: 2.reshape the destination pixels as N x 3 ho
mogeneous coordinate
    pixels_idx = np.vstack([
        meshgrid_x.reshape(meshgrid_x.shape[0]*meshgrid_
x.shape[1]),
        meshgrid_y.reshape(meshgrid_y.shape[0]*meshgrid_
y.shape[1]),
```

```
        np.ones((meshgrid_y.shape[0]*meshgrid_y.shape[1]
), dtype=int)
    ])


    if direction == 'b':
        # TODO: 3.apply H_inv to the destination pixels
and retrieve (u,v) pixels, then reshape to (ymax-
ymin),(xmax-xmin)
        new_pixels_idx = np.dot(H_inv, pixels_idx)
        new_pixels_idx[0, :] = np.divide(new_pixels_idx[
0, :], new_pixels_idx[2, :])
        new_pixels_idx[1, :] = np.divide(new_pixels_idx[
1, :], new_pixels_idx[2, :])
        new_pixels_idx[2, :] = np.ones_like(new_pixels_i
dx[2, :])
        new_pixels_idx = new_pixels_idx.reshape((3, ymax
-ymin, xmax-xmin))
        new_pixels_idx = np.round(new_pixels_idx).astype
(int)

        # TODO: 4.calculate the mask of the transformed
coordinate (should not exceed the boundaries of source i
mage)
        mask = np.ones_like(new_pixels_idx, dtype=bool)
        mask[0, :, :] = (new_pixels_idx[0, :, :] >= 0) &
 (new_pixels_idx[0, :, :] < w_src)
        mask[1, :, :] = (new_pixels_idx[1, :, :] >= 0) &
 (new_pixels_idx[1, :, :] < h_src)
        new_mask = mask[0, :, :] & mask[1, :, :]
        new_mask = new_mask.reshape((ymax-ymin, xmax-
xmin))

        # Turn invalid pixel index to (0, 0, 0), without
 changing the shape of pixel index array (keep it as (3,
 ymax-ymin, xmax-xmin))
        new_pixels_idx[:, ~new_mask] = 0
```

```
        # TODO: 5.sample the source image with the maske
d and reshaped transformed coordinates
        source_img_sample =  src[new_pixels_idx[1, :, :]
, new_pixels_idx[0, :, :], :]

        # TODO: 6. assign to destination image with prop
er masking
        dst[ymin:ymax, xmin:xmax, :][new_mask, :] = sour
ce_img_sample[new_mask, :]

    elif direction == 'f':
        # TODO: 3.apply H to the source pixels and retri
eve (u,v) pixels, then reshape to (ymax-ymin),(xmax-
xmin)
        new_pixels_idx = np.dot(H, pixels_idx)
        new_pixels_idx[0, :] = np.divide(new_pixels_idx[
0, :], new_pixels_idx[2, :])
        new_pixels_idx[1, :] = np.divide(new_pixels_idx[
1, :], new_pixels_idx[2, :])
        new_pixels_idx[2, :] = np.ones_like(new_pixels_i
dx[2, :])

        new_pixels_idx = new_pixels_idx.reshape((3, ymax
-ymin, xmax-xmin))

        # TODO: 4.calculate the mask of the transformed
coordinate (should not exceed the boundaries of destinat
ion image)
        mask = np.ones_like(new_pixels_idx, dtype=bool)
        mask[0, :, :] = (new_pixels_idx[0, :, :] >= 0) &
 (new_pixels_idx[0, :, :] < w_dst)
        mask[1, :, :] = (new_pixels_idx[1, :, :] >= 0) &
 (new_pixels_idx[1, :, :] < h_dst)
        new_mask = (mask[0, :, :] & mask[1, :, :])

        # TODO: 5.filter the valid coordinates using pre
vious obtained mask
        valid_coord_idx = new_pixels_idx[:, new_mask]
```

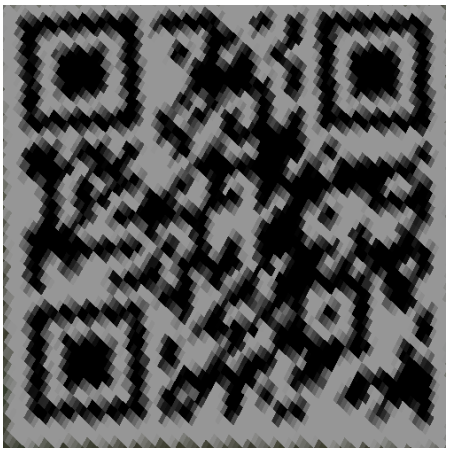
```
        valid_coord_idx = np.round(valid_coord_idx).asty
pe(int)
        valid_coord_idx = valid_coord_idx[:2, :]
        valid_coord_idx = valid_coord_idx.reshape(2, yma
x-ymin, xmax-xmin)

        # TODO: 6. assign to destination image using adv
anced array indicing
        dst[valid_coord_idx[1, :, :], valid_coord_idx[0,
 :, :], :] = \
        src[new_mask, :].reshape((src.shape[0], src.shap
e[1], src.shape[2]))

    return dst
```

2. **Briefly introduce the interpolation method you use**
   我沒有做特別的 interpolation 方法，單純在 backward warpping 遇到小數時，就直接取整數，才能當作 index 使用

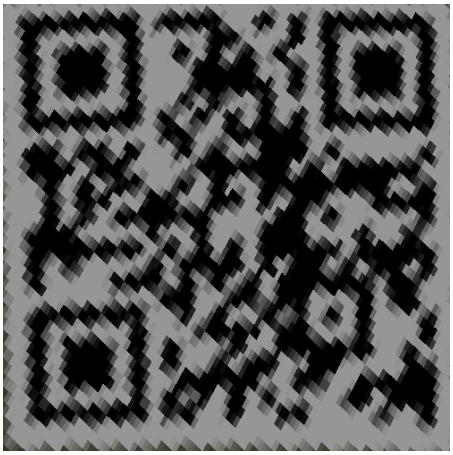
Part 3

1. Paste the 2 warped QR code and the link you find

I. QR code from BL_secret1.png



II. QR code from BL_secret2.png



III. Link of the QR codes (Scan by Android Line app)

https://qrgo.page.link/jc2Y9

➔ 這堂課的課程網頁

➔ 兩個 QR code 掃描結果一樣

➔ 要拿遠一點掃描才會偵測得出來

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

   兩張 source image 是不同的，BL_secret1 感覺上是 one point perspective 的照片，部分原本真實建築物上是平行的線，在照片上依然是平行的。而 BL_secret2 感覺原本的直線都彎曲了，特別是在照片的中心，所以猜測是 Barrel distortion，可能是相機鏡片較厚導致的。

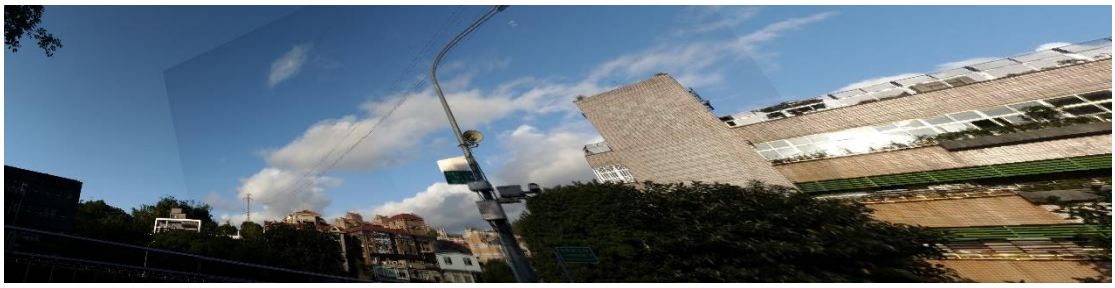
   兩張 source image warp 的結果都可以掃描出相同的 QR code link，所以是相同的。

3. **If the results are the same, explain why. If the results are different, explain why.**

   Warp 出來的結果一樣是合理的，因為從助教給的範例 code 就可以發現，兩張圖片做 backward warpping 時給的 destination 大小都是(500, 500)，所以可以猜測兩個 QR code 原始的大小都是(500, 500)。

Part 4

1. **Paste your stitched panorama**



2. **Can all consecutive images be stitched into a panorama? If yes, explain your reason. If not, explain under what conditions will result in a failure?**

   就算是連續影像也<u>不一定</u>能夠組成一張全景圖。

   Homography 會假設三張不同照片是從同一焦點卻是不同角度拍攝一個更遠的平面景象。但若是在室內拍攝多張影像，只要碰到轉角就不是同一平面了，也就會讓原本是平面的牆壁，變成像是圓弧狀的牆面。

3. **[Bonus] Using homography to produce a “more than 2 images panorama”**

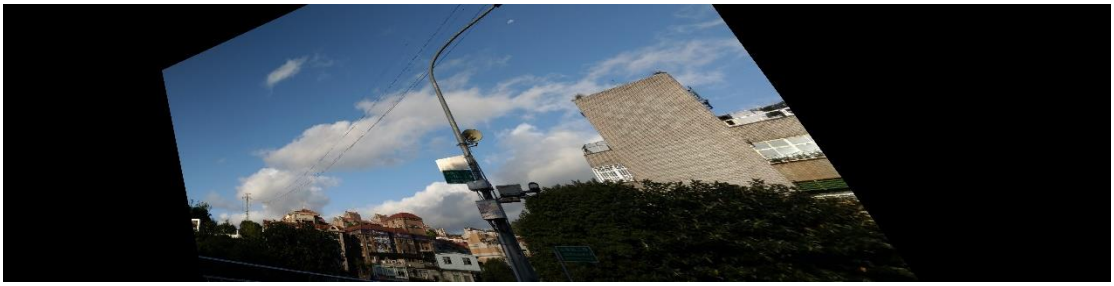
   我使用了 3 張影像合成一張全景圖

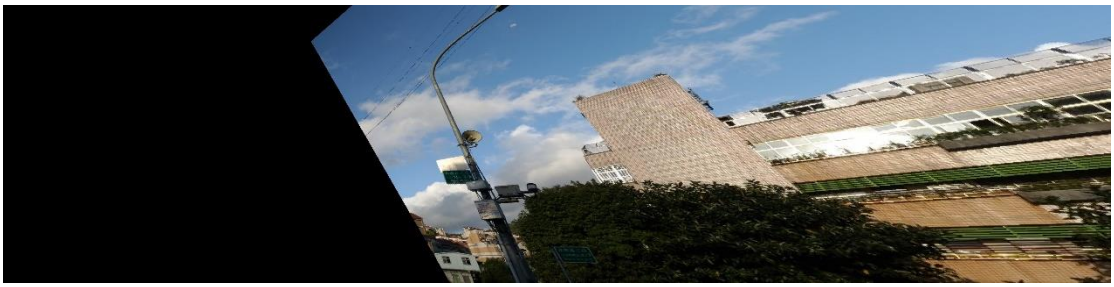
4. **[Bonus] Using blending techniques (simple alpha blending)***

   我實作了簡單的 alpha blending，會將重疊到的部分相加後取平均。

   假設已經求得三張圖片的 homography 了

   I. 先將 3 個影像分別投影在不同的 3 個 canvas 上，但是同樣大小且背景都是 0 (黑色)





II. 建立一個大小為 canvas 的 mask，將重疊的部分、只有一張圖片的部分和沒有圖片的部分分別給予不同的值。

III. 根據 mask 的數值做不同的 alpha blending，重疊的部分各自的權重為 0.5；只有一張圖片的部分，有圖片的權重為 1，沒有圖片的為 0；完全沒有圖片的部分就維持 0(黑色)。

需要將重疊與菲重疊區隔開來是因為，如果只有單張影像在 canvas 的某個位置上，如果取平均就只是將那張影像的數值除以 2 變暗而已，並不是理想中的效果。

*Reference of alpha blending: https://inst.eecs.berkeley.edu/~cs194-26/fa17/upload/files/proj6B/cs194-26-abw/