Computer Vision HW3 Report

Student ID: R08222017
Name: 陳韋辰

- Part 1 (2%)

1. Paste the function solve_homography( ) (1%)

```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):
        #first row
        A[2*i][3] = -u[i][0]
        A[2*i][4] = -u[i][1]
        A[2*i][5] = -1

        A[2*i][6] = u[i][0]*v[i][1]
        A[2*i][7] = u[i][1]*v[i][1]
        A[2*i][8] = v[i][1]
        #second row
        A[2*i+1][0] = u[i][0]
        A[2*i+1][1] = u[i][1]
        A[2*i+1][2] = 1

        A[2*i+1][6] = -u[i][0]*v[i][0]
        A[2*i+1][7] = -u[i][1]*v[i][0]
        A[2*i+1][8] = -v[i][0]
    # TODO: 2.solve H with A
    u, s, vh = np.linalg.svd(A)
    #extract H
    H = vh[-1].reshape(3,3)
    #normalize H with |h|=1
    H = H/np.linalg.norm(H)

    return H
```

2. Paste your warped canvas (1%)

- Part 2 (2%)

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

```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
    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    X,Y = np.mgrid[xmin:xmax:1, ymin:ymax:1]
    W = np.ones((xmax-xmin)*(ymax-ymin)).astype(np.int32)
    homocoordinate_pairs = np.vstack((X.flatten(), Y.flatten(), W))
    # print(homocoordinate_pairs[-1])

    if direction == 'b':
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        #計算transform對應的位置
        transformed_homocoordinate_pairs = np.dot(H_inv, homocoordinate_pairs)
        #計算x-y實際值(除w項)
        xy_pairs = homocoordinate_pairs[0:2,:]
        xy_transformed_pairs = np.round(transformed_homocoordinate_pairs[0:2,:]/transformed_homocoordinate_pairs[2]).astype(np.int32)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)
        #在範圍外(要刪掉的)顯示True
        xy_pairs_mask = (xy_transformed_pairs[0]<0) + (xy_transformed_pairs[0]>(w_src-1)) \
            + (xy_transformed_pairs[1]<0) + (xy_transformed_pairs[1]>(h_src-1))

        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
        xy_pairs = xy_pairs[:,~xy_pairs_mask]
        xy_transformed_pairs = xy_transformed_pairs[:,~xy_pairs_mask]

        # TODO: 6. assign to destination image with proper masking
        #於dst覆蓋src圖片(img上的x-y會顛倒(row-col))
        dst[xy_pairs[1], xy_pairs[0]] =\
            src[xy_transformed_pairs[1], xy_transformed_pairs[0]]

        pass

    elif direction == 'f':
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        #計算transform對應的位置
        transformed_homocoordinate_pairs = np.dot(H, homocoordinate_pairs)
        #計算x-y實際值(除w項)
        xy_pairs = homocoordinate_pairs[0:2,:]
        xy_transformed_pairs = np.round(transformed_homocoordinate_pairs[0:2,:]/transformed_homocoordinate_pairs[2]).astype(np.int32)
        # print(xy_homocoordinate_pairs.shape, xy_transformed_homocoordinate_pairs.shape)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
        # x_pairs_mask = ma.masked_outside(xy_transformed_pairs[0], 0, w_dst-1).mask#x的boundary
        # y_pairs_mask = ma.masked_outside(xy_transformed_pairs[1], 0, h_dst-1).mask#y的boundary
        # xy_pairs_mask = ma.masked_where(xy_transformed_pairs[0]<0 or  xy_transformed_pairs[0]>w_dst-1\
        #     or xy_transformed_pairs[1]<0 or xy_transformed_pairs[1]>h_dst-1, xy_transformed_pairs)

        #在範圍外(要刪掉的)顯示True
        xy_pairs_mask = (xy_transformed_pairs[0]<0) + (xy_transformed_pairs[0]>(w_dst-1)) \
            + (xy_transformed_pairs[1]<0) + (xy_transformed_pairs[1]>(h_dst-1))

        # xy_pairs = ma.array(xy_pairs, mask = np.tile(xy_pairs_mask, (2,1)))
        # xy_transformed_pairs = ma.array(xy_transformed_pairs, mask = np.tile(xy_pairs_mask, (2,1)))
        # print(xy_pairs_mask.astype(np.int32).sum(), len(transformed_homocoordinate_pairs[0]))
        # xy_transformed_pairs.view(ma.MaskedArray)
        # print(xy_pairs.mask)

        # TODO: 5.filter the valid coordinates using previous obtained mask
        #把範圍外(顯示True的)刪掉
        xy_pairs = xy_pairs[:,~xy_pairs_mask]
        xy_transformed_pairs = xy_transformed_pairs[:,~xy_pairs_mask]

        # TODO: 6. assign to destination image using advanced array indicing
        #於dst覆蓋src圖片(img上的x-y會顛倒(row-col))
        dst[xy_transformed_pairs[1], xy_transformed_pairs[0]] =\
            src[xy_pairs[1], xy_pairs[0]]


        pass

    return dst
```

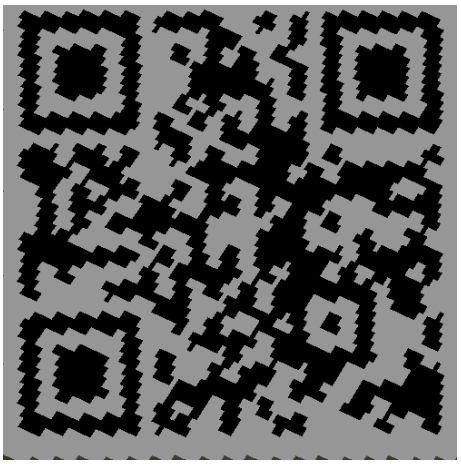
2. Briefly introduce the interpolation method you use (1%)

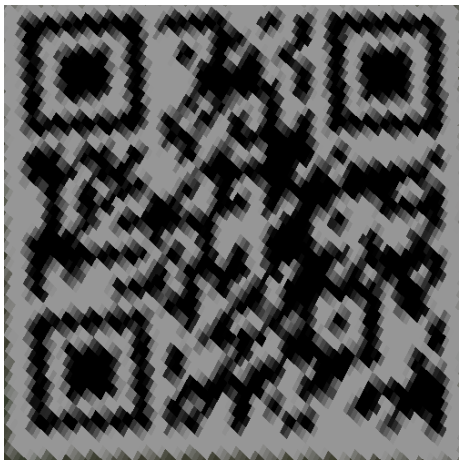
我**使用了 NumPy 中的 numpy.round()進行插值**，也就是會利用四捨五入到整數位，選擇離計算出來的值最接近 pixel 的 **Nearest neighbor** 算法。

- Part 3 (8%)

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

BL_secret1 BL_secret2





Link:

BL_secret1: https://qrgo.page.link/jc2Y9

BL_secret2: https://qrgo.page.link/jc2Y9

2. Discuss the difference between 2 source images, are the warped results the same or different? (3%)

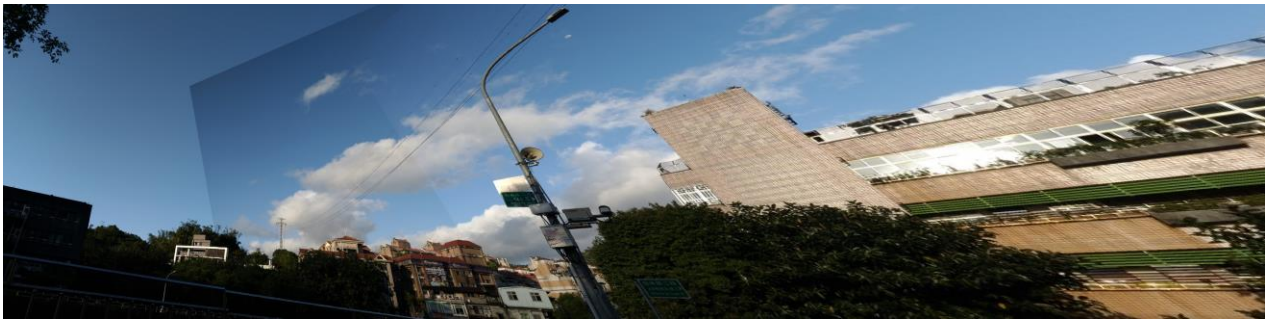
兩者 **warping 結果是不一樣的**，**BL_secret2 的 QR code 形狀是歪曲的**，因此並無法利用 homography 轉換成直線的 QR code，不符合 projective transformation 的線到線轉換的 invariant 特性，而 **BL_secret1 的 QR code 則是直線**，因此符合 homography 的轉換。也因此 BL_secret1 的 QR code 雖然有點扭曲，但大致上還算可以辨認；而相反的 BL_secret2 的 QR code 則嚴重失真，homography 轉換的效果不佳，但如使用 iPhone12 手機相機內建的 QR code 掃，兩者都掃得出連結。

3. If the results are the same, explain why. If the results are different, explain why. (4%)

Warping 結果是不一樣的，主要原因如上述所提到，**homography 是線性的投射，有線到線投影的 collinearity invariant 特性，並無法將曲線轉成直線**。因此 BL_secret2 的 QR code 的邊是彎曲的，warping 結果會較差，有奇怪的失真抖動，較不銳利，相較之下 BL_secret1 的 QR code 的邊原本就是直線，理論上可以 homography 成方形的 QR code，雖然圖像也有抖動，但比較清楚。

- Part 4 (8%)

1. Paste your stitched panorama (1%)



2. Can all consecutive images be stitched into a panorama? (3%)

不行，像是**視角超過 180 度**或是**相機鏡頭出現平移**就無法使用 homography 的方式連接 consecutive images。

3. If yes, explain your reason. If not, explain under what conditions will result in a failure? (4%)

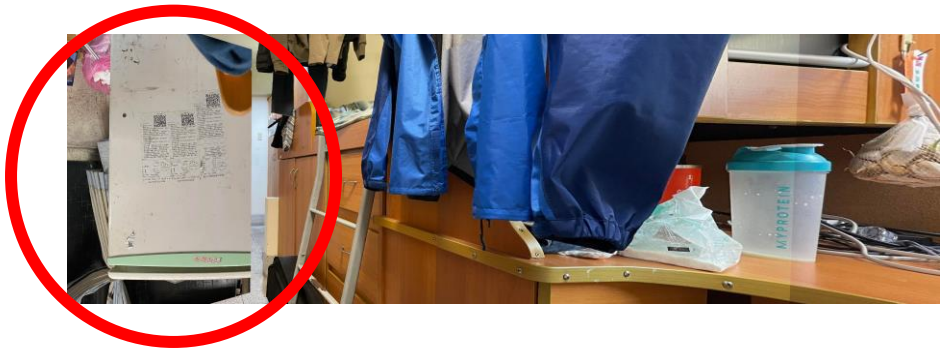
由於 homography 的全景拼貼的是利用投射點將全景球的影像投射到平面上，在 **0 和 180 度的角度將會投射到平面的無窮遠處**，所以 homography 最大只能投射 0 到 180 度的範圍，實驗如下：

使用 4 張照片，代表 90~180 度，進行 homography 的全景合成(寬度 36000 pixel)

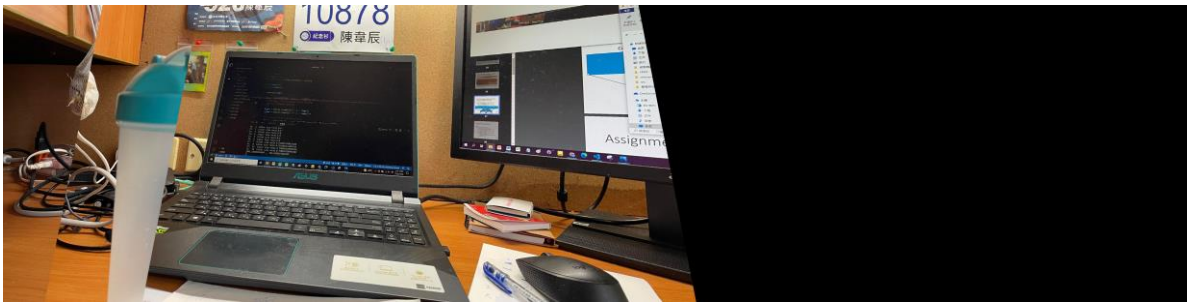
可以發現在靠近 180 的影像已經出現嚴重變形，甚至範圍已經超過輸出圖像，理論上可以顯示到無窮遠處。

使用 6 張照片(額外兩張)，代表 90~225 度，進行 homography 的全景合成(寬度 36000 pixel)

在超過 180 度後，全景拼貼將會依照投射點，投射到反向的位置，如下圖為全景圖最左側放大，可以發現 225 度右側的位置(約 270 度)被上下顛倒左右相反投影到全景圖左側，覆蓋掉原本圖像：



另一個例子是相機平移，由於 homography 的全景拼貼需要固定投射點，因此**平移後就無法得到同一個平面的圖像**(也就是出現不同視角，會看到 3 維空間中物體的不同角度)，自然無法正常拼貼，實驗如下：



可以發現 warping 的結果出現很多切線，代表 homography 的結果無法連接不同視角的照片，無法有效全景拼貼。