CV HW3

R11921041　蔣沅均

1. Homography estimation – warped canvas

2. Marker-Based Planar AR

- Code of solve_homography

```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))
    A[0::2, :] = np.hstack((u, np.ones((N, 1)), np.zeros((N, 3)), -u[:, 0:1] * v[:, 0:1], -u[:, 1:2] * v[:, 0:1], -v[:, 0:1]))
    A[1::2, :] = np.hstack((np.zeros((N, 3)), u, np.ones((N, 1)), -u[:, 0:1] * v[:, 1:2], -u[:, 1:2] * v[:, 1:2], -v[:, 1:2]))

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

    return H
```

- Code of warping

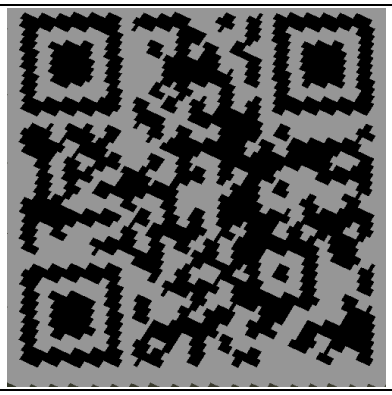
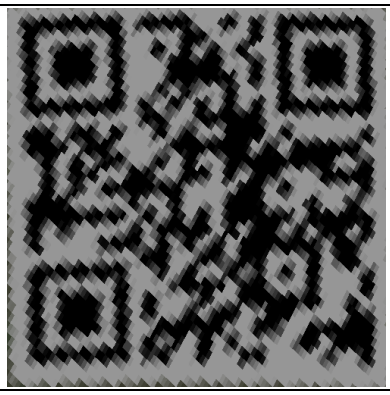
```python
def warping(src, dst, H, ymin, ymax, xmin, xmax, direction='b', blending=False):
    """
    Perform forward/backward warpping without for loops. i.e.
    for all pixels in src(xmin~xmax, ymin~ymax),  warp to destination
        (xmin=0,ymin=0)  source                      destination
    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

```python
    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
    mesh = np.meshgrid(np.arange(xmin, xmax), np.arange(ymin, ymax))
    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    pixels = np.hstack((mesh[0].reshape(-1, 1), mesh[1].reshape(-1, 1), np.ones((mesh[0].size, 1)))) # x, y, 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)
        src_pixels = np.dot(H_inv, pixels.T).T
        src_pixels = src_pixels[:, :2] / src_pixels[:, 2:]
        src_pixels = np.round(src_pixels).astype(np.int)
        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)
        mask = np.ones(src_pixels.shape[0], dtype=np.bool)
        mask = np.logical_and(mask, src_pixels[:, 0] >= 0)
        mask = np.logical_and(mask, src_pixels[:, 0] < w_src)
        mask = np.logical_and(mask, src_pixels[:, 1] >= 0)
        mask = np.logical_and(mask, src_pixels[:, 1] < h_src)
        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
        src_pixels = src_pixels[mask]
        dst_pixels = np.round(np.stack((mesh[0], mesh[1]), axis=-1).reshape(-1, 2)[mask]).astype(np.int)

        # TODO: 6. assign to destination image with proper masking
        if not blending:
            dst[dst_pixels[:, 1], dst_pixels[:, 0]] = src[src_pixels[:, 1], src_pixels[:, 0]]
```

```
        # TODO: blending
        elif blending:
            src_cover = np.zeros((h_dst, w_dst))
            src_cover[dst_pixels[:, 1], dst_pixels[:, 0]] = 1
            dst_cover = np.zeros((h_dst, w_dst))
            dst_cover[np.sum(dst, axis=-1) != 0] = 1
            overlap = src_cover * dst_cover

            overlap_start = np.where(overlap.sum(axis=0)>5)[0][0]
            overlap_end = np.where(overlap.sum(axis=0)>5)[0][-1]+1

            dst_overlap = overlap.copy()
            src_overlap = overlap.copy()

            def sigmoid(x):
                return 1 / (1 + np.exp(-x))
            dst_overlap[:, overlap_start:overlap_end] *= sigmoid(np.linspace(-20, 20, num=overlap_end-overlap_start))
            src_overlap[:, overlap_start:overlap_end] *= sigmoid(np.linspace(20, -20, num=overlap_end-overlap_start))

            dst_weight = dst_cover - dst_overlap
            src_weight = src_cover - src_overlap

            warped_src = np.zeros_like(dst)
            warped_src[dst_pixels[:, 1], dst_pixels[:, 0]] = src[src_pixels[:, 1], src_pixels[:, 0]]

            dst = dst_weight[..., np.newaxis] * dst + src_weight[..., np.newaxis] * warped_src


    elif direction == 'f':
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        dst_pixels = np.dot(H, pixels.T).T
        dst_pixels = dst_pixels[:, :2] / dst_pixels[:, 2:]
        dst_pixels = np.round(dst_pixels).astype(np.int) # N, 2
        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
        mask = np.ones(dst_pixels.shape[0], dtype=np.bool)
        mask = np.logical_and(mask, dst_pixels[:, 0] >= 0)
        mask = np.logical_and(mask, dst_pixels[:, 0] < w_dst)
        mask = np.logical_and(mask, dst_pixels[:, 1] >= 0)
        mask = np.logical_and(mask, dst_pixels[:, 1] < h_dst)
        # TODO: 5.filter the valid coordinates using previous obtained mask
        src_pixels = np.round(np.stack((mesh[0], mesh[1]), axis=-1).reshape(-1, 2)[mask]).astype(np.int)
        dst_pixels = dst_pixels[mask]
        # TODO: 6. assign to destination image using advanced array indicing
        dst[dst_pixels[:, 1], dst_pixels[:, 0]] = src[src_pixels[:, 1], src_pixels[:, 0]]

    return dst
```

- Briefly introduce the interpolation method you use
  - 這邊使用的是 nearest neighbor，即直接將目標座標四捨五入至整數，並取該整數位置之值。

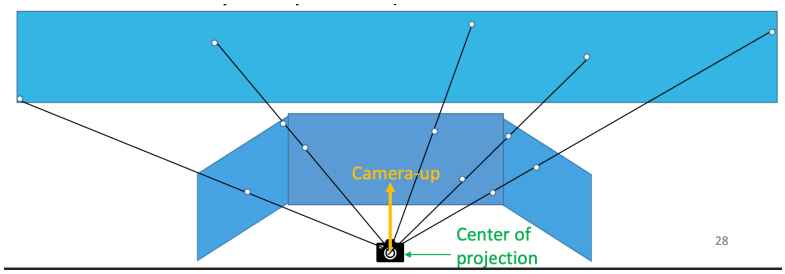
3. Unwarp the secret
   - 2 warped images

3_1	3_2
	

   - the link you find
     - 兩張 QR code 都可以解出以下網址 http://media.ee.ntu.edu.tw/courses/cv/21S/
   - (a) Discuss the difference between 2 source images, are the warped results the same or different?
     (b) If the results are the same, explain why. If the results are different, explain why.
     - 兩張圖都能成功解出連結，推測這是因 QR code 的設計上就有一定的容錯能力，因此即使 3_2 較為模糊仍然能解析出一樣的結果。
     - 兩張圖視覺上有顯著差異，這是因為 BL_secret2 的圖片中有相當嚴重的 distortion，造成 QR code 在此圖中呈現弧狀，而 warping 只能夠映射四邊形，所以在 warp 有弧度的圖片時就會產生瑕疵，因此如果要產生如 3_1 的結果，應該要先進行 undistort 再 warp。

4. Panorama
   1. Paste your stitched panorama

2. Can all consecutive images be stitched into a panorama
   a. 不行。
   b. 反例:
      當拍照片時旋轉超過 180 度時，即無法如下圖（作業說明投影片 p28）所示，將所有照片投影至一個平面。



Bonus:

先計算出兩張圖的重疊區域，再延 x 軸方向依照 sigmoid 函數遞增/遞減 alpha blending 的比例，最後再依照此比例將兩圖融合。使用 sigmoid 函數即可以減小兩圖比例相近的區域面積，進而減少明顯的疊影。

程式碼:

```
# TODO: blending
elif blending:
    src_cover = np.zeros((h_dst, w_dst))
    src_cover[dst_pixels[:, 1], dst_pixels[:, 0]] = 1
    dst_cover = np.zeros((h_dst, w_dst))
    dst_cover[np.sum(dst, axis=-1) != 0] = 1
    overlap = src_cover * dst_cover

    overlap_start = np.where(overlap.sum(axis=0)>5)[0][0]
    overlap_end = np.where(overlap.sum(axis=0)>5)[0][-1]+1

    dst_overlap = overlap.copy()
    src_overlap = overlap.copy()

    def sigmoid(x):
        return 1 / (1 + np.exp(-x))
    dst_overlap[:, overlap_start:overlap_end] *= sigmoid(np.linspace(-20, 20, num=overlap_end-overlap_start))
    src_overlap[:, overlap_start:overlap_end] *= sigmoid(np.linspace(20, -20, num=overlap_end-overlap_start))

    dst_weight = dst_cover - dst_overlap
    src_weight = src_cover - src_overlap

    warped_src = np.zeros_like(dst)
    warped_src[dst_pixels[:, 1], dst_pixels[:, 0]] = src[src_pixels[:, 1], src_pixels[:, 0]]

    dst = dst_weight[..., np.newaxis] * dst + src_weight[..., np.newaxis] * warped_src
```