Computer Vision HW3 Report

Student ID: R13945050
Name: 張祐嘉

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):
    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
    x = u[:, 0]
    y = u[:, 1]
    x_p = v[:, 0]
    y_p = v[:, 1]

    zeros = np.zeros_like(x)
    ones = np.ones_like(x)
    A1 = np.stack([x, y, ones, zeros, zeros, zeros, -x * x_p, -y * x_p, -x_p], axis=1)
    A2 = np.stack([zeros, zeros, zeros, x, y, ones, -x * y_p, -y * y_p, -y_p], axis=1)
    A = np.vstack([A1, A2])  # shape: (2N, 9)
    # TODO: 2.solve H with A
    U, S, Vt = np.linalg.svd(A)
    h = Vt[-1, :]
    h = h / h[-1]
    H = h.reshape((3,3))
    return H
```

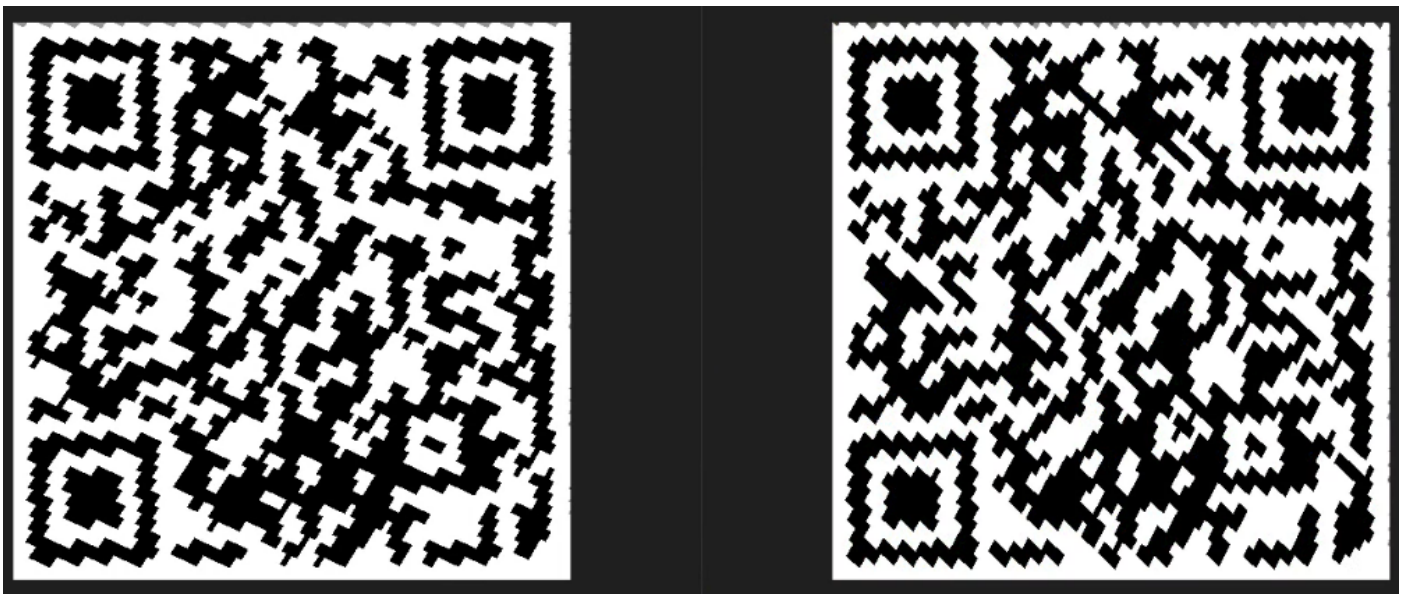
```python
def warping(src, dst, H, ymin, ymax, xmin, xmax, direction='b'):
    h_src, w_src, ch = src.shape
    h_dst, w_dst, ch = dst.shape
    H_inv = np.linalg.inv(H)

    # TODO: 1.meshgrid the (x,y) coordinate pairs
    x = np.arange(xmin, xmax)
    y = np.arange(ymin, ymax)
    xv, yv = np.meshgrid(x, y)
    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    num_pixels = xv.size
    ones = np.ones((num_pixels,))

    if direction == 'b':
        dst_coords = np.stack([xv.ravel(), yv.ravel(), ones], axis=1)
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels,
        src_coords = (H_inv @ dst_coords.T).T  # shape: (N, 3)
        src_coords = src_coords / src_coords[:, 2:3]  # Normalize by the third (ho
        ux = src_coords[:, 0].reshape((ymax - ymin), (xmax - xmin))
        uy = src_coords[:, 1].reshape((ymax - ymin), (xmax - xmin))
        # TODO: 4.calculate the mask of the transformed coordinate (should not exc
        valid_mask = (
            (ux >= 0) & (ux < w_src) &
            (uy >= 0) & (uy < h_src)
        )
        # TODO: 5.sample the source image with the masked and reshaped transformed
        valid_x = ux[valid_mask].astype(int)
        valid_y = uy[valid_mask].astype(int)

        # Get the corresponding destination coordinates
        dst_x_valid = xv[valid_mask]
        dst_y_valid = yv[valid_mask]
        # TODO: 6. assign to destination image with proper masking
        dst[dst_y_valid, dst_x_valid] = src[valid_y, valid_x]

    elif direction == 'f':
        src_coords = np.stack([xv.ravel(), yv.ravel(), ones], axis=1)# Shape: (N,
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then re
        dst_coords = (H @ src_coords.T).T  # shape: (N, 3)
        dst_coords = dst_coords / dst_coords[:, 2:3]
        vx = dst_coords[:, 0].reshape((ymax-ymin),(xmax-xmin))
        vy = dst_coords[:, 1].reshape((ymax-ymin),(xmax-xmin))
        # TODO: 4.calculate the mask of the transformed coordinate (should not exc
        valid_mask = (
            (vx >= 0) & (vx < w_dst) &
            (vy >= 0) & (vy < h_dst)
        )
        # TODO: 5.filter the valid coordinates using previous obtained mask
        x_valid = vx[valid_mask].astype(int)
        y_valid = vy[valid_mask].astype(int)
        src_x_valid = xv[valid_mask]
        src_y_valid = yv[valid_mask]
        # TODO: 6. assign to destination image using advanced array indicing
        dst[y_valid, x_valid] = src[src_y_valid, src_x_valid]
    return dst
```

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

Forward 和 **backward** 我都採用 **Nearest neighbor** 的方式 **(**取最接近整數**)**，主要理由是程式上看起來比較簡單運算速度也較快，有嘗試 **bilinear interpolation** 的方式，雖然會使影像特別是邊緣的地方較滑順，但肉眼上似乎看不出跟 **nearest neighbor** 有太大的區別，因此最後還是採用 **nearest neighbor**

Part 3.

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



兩個連結相同: **http://media.ee.ntu.edu.tw/courses/cv/25S/**

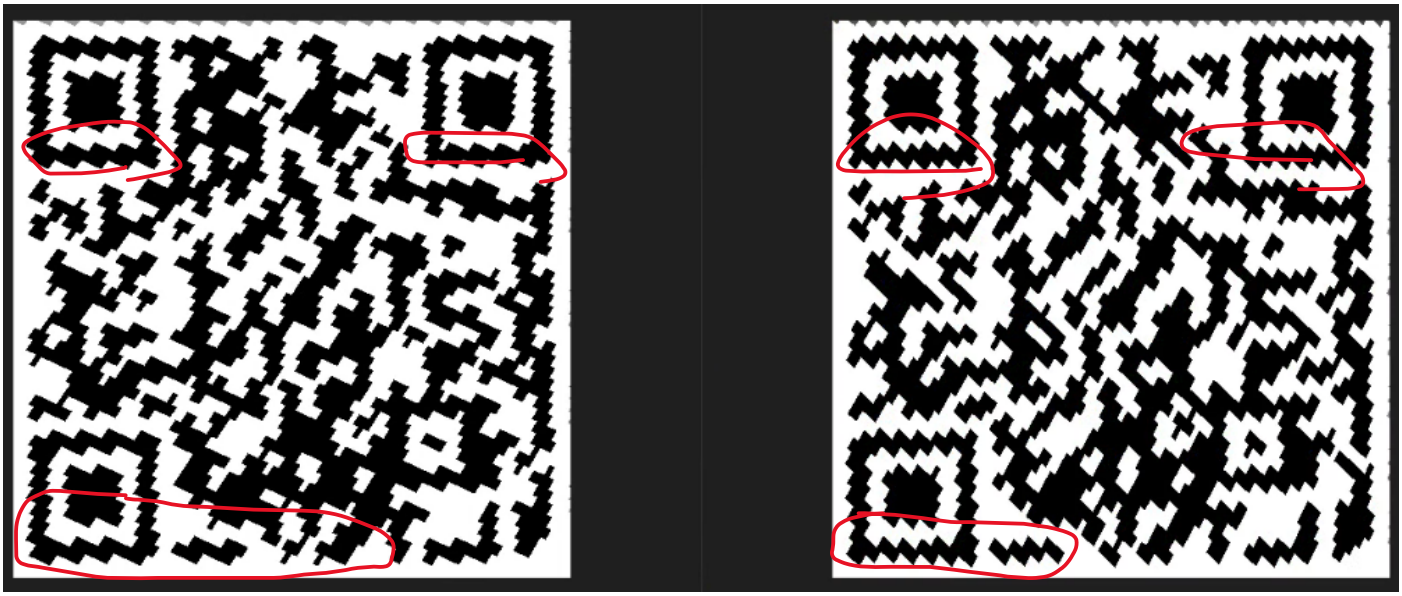
- **Discuss the difference between 2 source images, are the warped results the same or different?**

BL_secret2 的圖具有視角上的些微變形，以 **qrcode** 的形狀而言跟另一張圖相比有被壓縮到的感覺**(**長寬比較大**)**，但也因為沒有到差的很多**(**原圖長寬比**)**，所以轉換出的 **qrcode** 看不出太明顯的差異，可能硬要說的話，右圖**(BL_secret2)** 感覺有稍微被擠壓到**(**左圖延左右方向內擠壓**)**，從每一格小長方型方塊可以看出，斜率有變大的現象

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

大致上是相同的，但細節上可以發現右圖每個小方塊明顯較斜 **ex:**

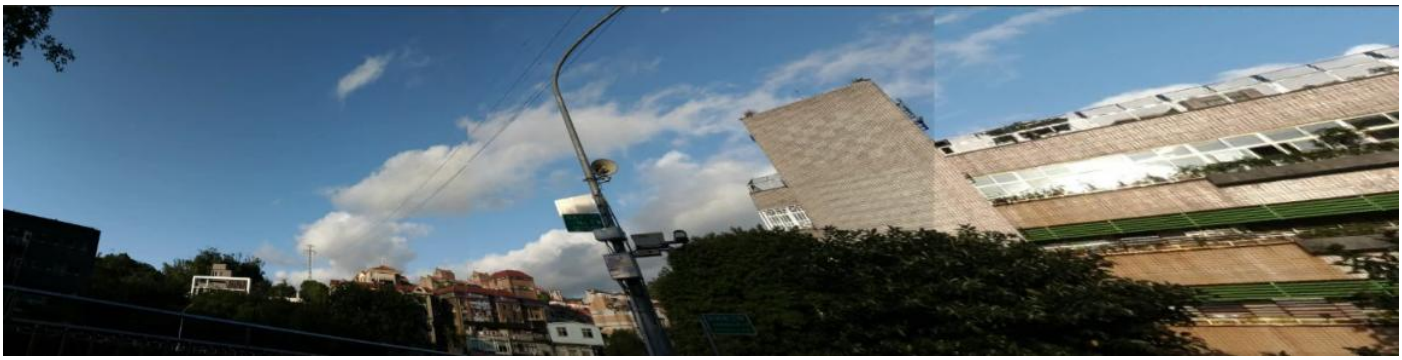
如紅色圈選處



推測可能在 **homography** 轉換上 **BL_secret2** 會有拉伸的問題(因為需轉換到正方形的 **size)**，也就是代表原圖的長寬比如果越大轉換成正方形時就會有越嚴重的變形，反之若原圖沒有使長寬發生畸變轉換後應維持正常

Part 4.

- **Paste your stitched panorama**



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

1. **Images photoed with camera translation Ans: No (但在距離足夠遠時可以)**
2. **Non planar scene (scene of indoor view) Ans: No**

主要理由皆為 **homography** 矩陣的轉換方法有假設整個場景為單一平面的前提，因此上述兩種做法會使得影像有不同的角度/深度，導致無法正常拼接

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

failure?

第一是若相機位置有平移或前進/後退(不維持在中心)，會有視角差的狀況發生(因同一物體在不同照片的大小或方向可能不同導致無法很好的拼接)，但經過測試發現若場景夠遠相機的平移對特徵點的視差影響較小，仍可拼接成功，而室內場景則是因為距離太近導致物體有前後景的現象(場景不再單一平面上)，而不同深度上的 **pixel** 無法僅透過單一個 **homography** 矩陣做轉換，因此可能出現扭曲或特徵無法對齊的狀況，另外再測試時也發現，重疊太少可能造成找不到理想的轉換矩陣，或是造成剩下的影像被拉伸(目前的寫法會固定轉換後影像的大小，即全圖的 **size** 是固定的，但我想應可以克服)如最右圖只剩房子