Computer Vision HW3 Report

Student ID: B10505047
Name: 邱郁喆

Part 1.

- **Paste your warped canvas**

Part 2.

- **Paste the function code *solve_homography(u, v)* & *warping( )* (both forward & backward)**

1. solve_homography(u, v) :

```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
    A = np.zeros((2*N, 9))
    # deal with odd rows
    A[0::2, 0:2] = u[0::1, 0:2]
    A[0::2, 2] = 1
    A[0::2, 6:8] = -(u[0::1, 0:2])*(v[0::1, 0:1])
    A[0::2, 8] = -(v[0::1, 0])
    # deal with even rows
    A[1::2, 3:5] = u[0::1, 0:2]
    A[1::2, 5] = 1
    A[1::2, 6:8] = -(u[0::1, 0:2])*(v[0::1, 1:2])
    A[1::2, 8] = -(v[0::1, 1])

    # TODO: 2.solve H with A
    U, S, VT = np.linalg.svd(A)
    H = VT[-1, :].reshape((3, 3)) # Let H be the last column of V <=> last row of VT
    return H
```

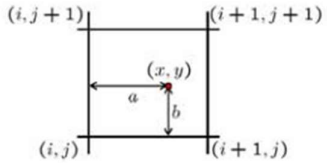
2. warping( ) (both forward & backward)

```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
    xx, yy = np.meshgrid(np.arange(xmin, xmax), np.arange(ymin, ymax))

    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    x = xx.reshape(((xmax-xmin)*(ymax-ymin), 1))
    y = yy.reshape(((xmax-xmin)*(ymax-ymin), 1))
    ones = np.ones(((xmax-xmin)*(ymax-ymin), 1))
    U = np.hstack((x, y, ones)) # N x 3

    if direction == 'b':
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        V = np.dot(H_inv, U.T) # mapping in src (change to 3 x N)
        V = V/V[2]
        Vx = V[0].reshape((ymax-ymin, xmax-xmin))
        Vy = V[1].reshape((ymax-ymin, xmax-xmin))

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)
        # mask = v (logical operation) image range
        mask = ((Vx >= 0) & (Vx <= w_src-1)) & ((Vy >= 0) & (Vy <= h_src-1))

        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
        # output = do something * mask
        outputx = Vx[mask] # shape: (n, )
        outputy = Vy[mask]
```

```python
        # Bilinear interpolation
        i = np.floor(outputx).astype(int)
        j = np.floor(outputy).astype(int)
        a = (outputx-i)[:, np.newaxis] # shape: (n, 1)
        b = (outputy-j)[:, np.newaxis]
        output = np.zeros((h_src, w_src, ch)) # same shape as src
        output[j, i] += (1-a)*(1-b)*src[j, i] + a*(1-b)*src[j, i] + \
                        a*b*src[j+1, i+1] + (1-a)*b*src[j, i+1]

        # TODO: 6. assign to destination image with proper masking
        dst[ymin:ymax,xmin:xmax][mask] = output[j, i]

    elif direction == 'f':
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        V = np.dot(H, U.T) # mapping in dst (change to 3 x N)
        V = np.round(V/V[2]).astype(int) # may be non-integer, so need to do round. Also, np.round() returns float64
        Vx = V[0].reshape((ymax-ymin, xmax-xmin))
        Vy = V[1].reshape((ymax-ymin, xmax-xmin))

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
        # mask = v (logical operation) image range
        mask = ((Vx >= 0) & (Vx <= w_dst-1)) & ((Vy >= 0) & (Vy <= h_dst-1))

        # TODO: 5.filter the valid coordinates using previous obtained mask
        # output = do something * mask
        outputx = Vx[mask]
        outputy = Vy[mask]

        # TODO: 6. assign to destination image using advanced array indicing
        dst[outputy, outputx] = src[mask]
        # print(dst.shape) # (1275, 1920, 3)
        # print(dst[outputy, outputx].shape) # (563000, 3)
        # print(outputy.shape) # (563000,)

    return dst
```

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

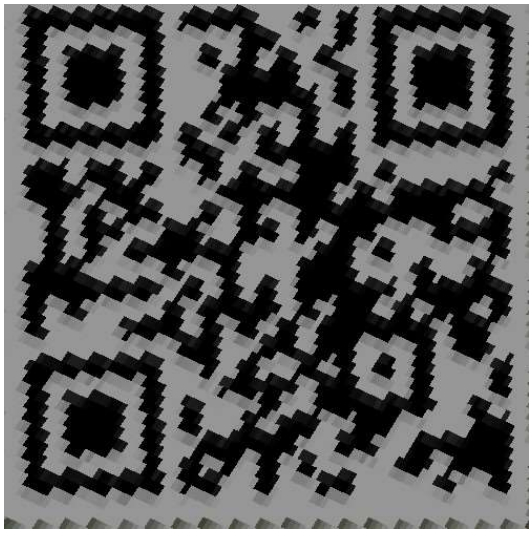
我使用的方法為 bilinear interpolation，其基本原理就是不管 warping 後得到的座標是多少，其一定會介於圖中某四個整數點之間，透過這四個點進行 interpolation 後即為所求(如下圖所示)



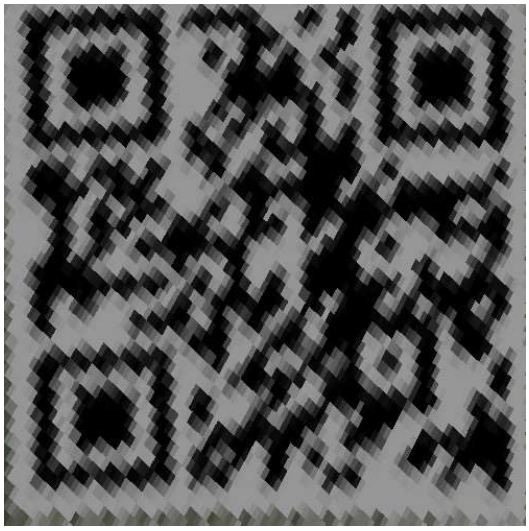
$$
\begin{aligned}
f(x,y) = \; & (1-a)(1-b) \;\; f[i,j] \\
& +a(1-b) \;\; f[i+1,j] \\
& +ab \;\; f[i+1,j+1] \\
& +(1-a)b \;\; f[i,j+1]
\end{aligned}
$$

Part 3.

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



output3_1.png



output3_2.png

link I found : https://qrgo.page.link/jc2Y9 (兩張圖都是此 link)

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

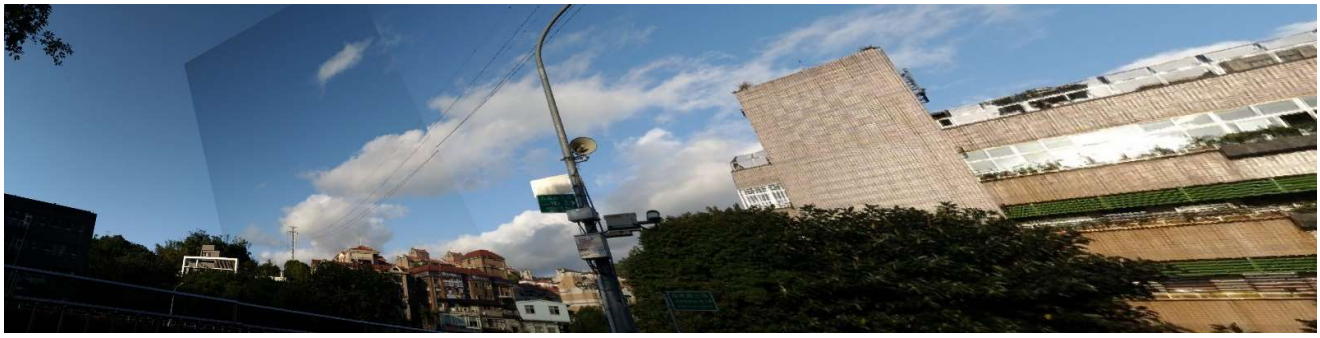
第一張原圖(BL_secret1.png)中的邊界是直的，而第二張原圖(BL_secret1.png)中的邊界則是彎曲的。因為如此，所以兩張圖得到的 warped results 也並不相同，output3_1.png 的 QR code 較為清晰，而 output3_2.png 的 QR code 則較為模糊。

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

造成兩張圖得到的 warped results 不相同的原因可能是由於第一張原圖(BL_secret1.png)中的邊界是直的，因此圖片保留較多資訊以及 homography 將直線轉換直線能得到較正確的結果；而由於第二張原圖(BL_secret1.png)中的邊界是彎曲的，因此可能有圖片較被壓縮以及 homography 較不能將曲線轉換直線的問題，故得到較模糊的結果。

Part 4.

- **Paste your stitched panorama**



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

No, not all consecutive images can be stitched into a panorama.

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

若是要拼接成 panorama，則圖片之間需要有一些共同的特徵點，也就是說圖片之間需要足夠的重疊部分，如此才能將它們拼接起來。因此，若圖片之間重疊部分不夠多，則無法成功拼接成 panorama。