Computer Vision HW3 Report

Student ID: B11901123
Name: 張牲源

Part 1.

- **Paste your warped canvas**

Part 2.

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

```
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 = []
    for i in range(N):
         x, y = u[i]
         xp, yp = v[i]
```

```
        A.append([x, y, 1, 0, 0, 0, -x*xp, -y*xp, -xp])
        A.append([0, 0, 0, x, y, 1, -x*yp, -y*yp, -yp])

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

    return H
```

```
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
    xc, yc = np.meshgrid(np.arange(xmin, xmax, 1), np.arange(ymin, ymax, 1), sparse = False)

    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    xrow = xc.reshape(( 1,(xmax-xmin)*(ymax-ymin) ))
    yrow = yc.reshape(( 1,(xmax-xmin)*(ymax-ymin) ))
    onerow =  np.ones(( 1,(xmax-xmin)*(ymax-ymin) ))
    M = np.concatenate((xrow, yrow, onerow), axis = 0)

    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, M)
        Vx, Vy, _ = V/V[2]
        # then reshape to (ymax-ymin),(xmax-xmin)
        Vx = Vx.reshape(ymax-ymin, xmax-xmin)
        Vy = Vy.reshape(ymax-ymin, xmax-xmin)
        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the
boundaries of source image)
        h_src, w_src, ch = src.shape
        mask = (((Vx<w_src-1)&(0<=Vx))&((Vy<h_src-1)&(0<=Vy)))
        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
        mVx = Vx[mask]
        mVy = Vy[mask]
        mVxi = mVx.astype(int)
        mVyi = mVy.astype(int)
```

```
        dX = (mVx - mVxi).reshape((-1,1)) # calculate delta X
        dY = (mVy - mVyi).reshape((-1,1)) # calculate delta Y
        p = np.zeros((h_src, w_src, ch))
        p[mVyi, mVxi, :] += (1-dY)*(1-dX)*src[mVyi, mVxi, :]
        p[mVyi, mVxi, :] += (dY)*(1-dX)*src[mVyi+1, mVxi, :]
        p[mVyi, mVxi, :] += (1-dY)*(dX)*src[mVyi, mVxi+1, :]
        p[mVyi, mVxi, :] += (dY)*(dX)*src[mVyi+1, mVxi+1, :]
        # TODO: 6. assign to destination image with proper masking
        dst[ymin:ymax,xmin:xmax][mask] = p[mVyi,mVxi]


    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,M)
        V = (V/V[2]).astype(int)
        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)
        h_dst, w_dst, ch = dst.shape
        mask = ((Vx<w_dst)&(0<=Vx))&((Vy<h_dst)&(0<=Vy))
        # TODO: 5.filter the valid coordinates using previous obtained mask
        mVx = Vx[mask]
        mVy = Vy[mask]
        # TODO: 6. assign to destination image using advanced array indicing
        dst[mVy, mVx, :] = src[mask]
    return dst
```

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

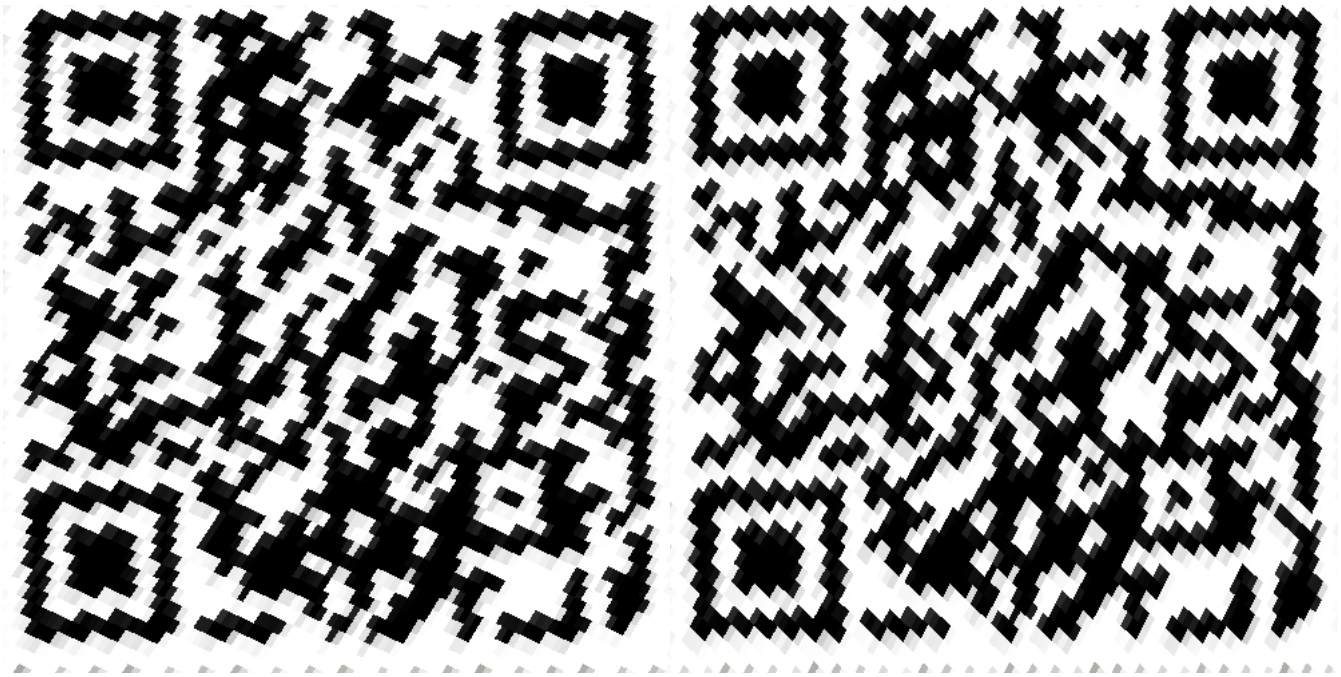
1. Backward Warping (direction='b'):
   - For each destination pixel, the corresponding source pixel location is calculated (which is typically non-integer)
   - The algorithm computes weights for the four nearest source pixels based on the fractional distances:
     - dX = (mVx - mVxi).reshape((-1,1)) # fractional x-distance
     - dY = (mVy - mVyi).reshape((-1,1)) # fractional y-distance
   - Each destination pixel gets a weighted average from the four neighboring source pixels:
     - Top-left: weight = (1-dY)*(1-dX)
     - Bottom-left: weight = (dY)*(1-dX)
     - Top-right: weight = (1-dY)*(dX)
     - Bottom-right: weight = (dY)*(dX)

This bilinear interpolation ensures smooth transitions between pixels, avoiding aliasing artifacts.

2. Forward Warping (direction='f'):
   In forward warping, a simpler nearest-neighbor approach is used:
   - Source pixel locations are mapped to destination coordinates
   - The resulting coordinates are rounded to integers using .astype(int)

Part 3.

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



http://media.ee.ntu.edu.tw/courses/cv/25S/

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

The source images are captured from slightly different angles or positions. In the code, different sets of corner coordinates are defined for each image (corners1 and corners2), identifying specific quadrilateral regions that contain hidden information. These two warped images clearly show that the left image is sharper, while the right one appears more blurred.

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

The results are different.

The cause of the difference:

- the input images (secret1 vs secret2) are different: the first is more upright, and its aspect ratio is closer to the original, making it easier to restore accurately. In contrast, the QR code in the second source image is more distorted and elongated.
- They were each warped using a different homography matrix, derived from distinct source corner coordinates.

Part 4.

• Paste your stitched panorama

• Can all consecutive images be stitched into a panorama?

The three images can be stitched together, but the seams between the images are not be perfectly aligned and some information may be lost. The combined image will still be visually recognizable as a panorama.

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

For any stitched panorama:

- If the panorama is projected onto a flat surface, the maximum rotation angle cannot exceed 180 degrees.
- If the panorama is projected onto a cylindrical surface, the maximum rotation angle cannot exceed 360 degrees.

When projecting a panorama onto a flat surface, a rotation angle of 180 degrees already pushes the physical limits, as the left and right edges are projected to infinity. Therefore, it is impossible for the rotation angle to exceed 180 degrees.