Computer Vision HW3 Report

Student ID: R12K41025
Name: 杜冠廷

Part 1.

- **Paste your warped canvas**

Part 2.

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

solve_homography(u, v)

```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):
        A[2*i, :] = np.array([u[i][0], u[i][1], 1, 0, 0, 0, -u[i][0] * v[i][0], -u[i][1] * v[i][0], -v[i][0]])
        A[2*i+1, :] = np.array([0, 0, 0, u[i][0], u[i][1], 1, -u[i][0] * v[i][1], -u[i][1] * v[i][1], -v[i][1]])

    # TODO: 2.solve H with A
    _, _, Vt=np.linalg.svd(A)
    V = np.transpose(Vt)
    H = V[:, -1]
    H = (H/np.sum(H)).reshape((3,3))
    return H
```

forward warping( )

```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
    X, Y = np.meshgrid(np.arange(xmin, xmax),np.arange(ymin, ymax))

    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    homogeneous = np.concatenate((X.reshape(-1, 1), Y.reshape(-1, 1), np.ones_like(X.reshape(-1, 1))), axis = 1)
```

```python
    elif direction == 'f':
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        # (3,3) * (N,3)^T = (3,N)
        transformed_coord = H @ homogeneous.T # np.dot(H, homogeneous.T)
        # inhomogeneous, (N,3)
        transformed_coord = (transformed_coord / transformed_coord[2]).T
        transformed_coord = transformed_coord.astype(int)
        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
        mask = (transformed_coord[:,0] >= 0) & (transformed_coord[:,0] < w_dst) & \
               (transformed_coord[:,1] >= 0) & (transformed_coord[:,1] < h_dst)
        # TODO: 5.filter the valid coordinates using previous obtained mask
        valid_src = homogeneous[mask][:,:-1].T
        valid_dst = transformed_coord[mask][:,:-1].T
        # TODO: 6. assign to destination image using advanced array indicing
        dst[valid_dst[1],valid_dst[0]] = src[valid_src[1],valid_src[0]]

    return dst
```

backward warping( )

```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
    X, Y = np.meshgrid(np.arange(xmin, xmax),np.arange(ymin, ymax))

    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    homogeneous = np.concatenate((X.reshape(-1, 1), Y.reshape(-1, 1), np.ones_like(X.reshape(-1, 1))), axis = 1)
```

```python
    if direction == 'b':
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        # (3,3) * (N,3)^T = (3,N)
        transformed_coord =  H_inv @ homogeneous.T # np.dot(H_inv, homogeneous.T)
        # inhomogeneous, (N,3)
        transformed_coord = (transformed_coord / transformed_coord[2]).T
        transformed_coord = transformed_coord.astype(int)
        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
        mask = (transformed_coord[:,0] >= 0) & (transformed_coord[:,0] < w_src) & \
               (transformed_coord[:,1] >= 0) & (transformed_coord[:,1] < h_src)
        # TODO: 5.filter the valid coordinates using previous obtained mask
        valid_dst = homogeneous[mask][:,:-1].T
        valid_src = transformed_coord[mask][:,:-1].T
        # TODO: 6. assign to destination image using advanced array indicing
        dst[valid_dst[1],valid_dst[0]] = src[valid_src[1],valid_src[0]]
```

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

We use the nearest neighbor method by rounding the coordinates to the closest integer to get the pixel index. In forward (or backward) warping, we choose the nearest pixel after mapping to the destination (or source) image. In Python, this can be done using the ".astype(int)" function to convert the coordinates to integers.

Part 3.

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

BL_secret1	BL_secret2
Link	http://media.ee.ntu.edu.tw/courses/cv/25S/

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

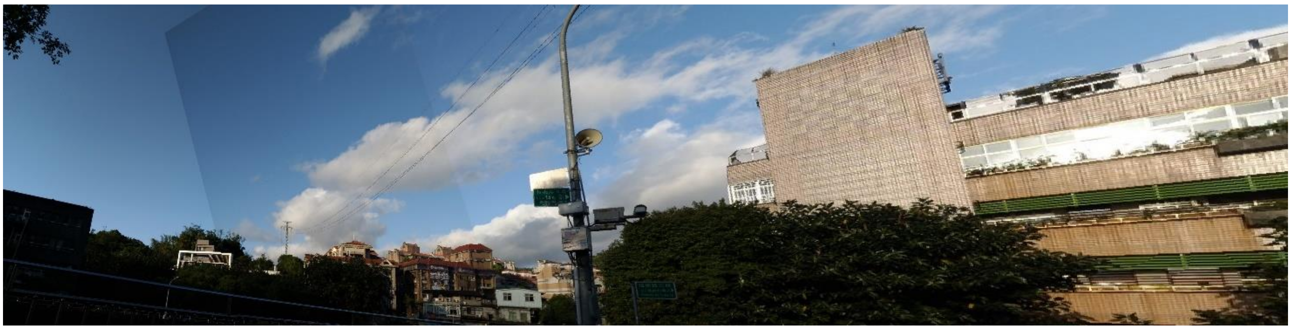
The first image (BL_secret1) is projected onto a "planar" projection plane. However, the second image (BL_secret2) is not projected onto a flat projection plane (for example, straight lines in the real world appear curved in the image). To address the issues of distortion and irregularity, we want to process the images on a planar projection plane. Therefore, we can see that the warped result of the BL_secret1 image (on the left) is clearer and more accurate than that of the BL_secret2 image (on the right). Although the scanning results are the same, the two images are not exactly identical.

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

Due to the distortion present in the second image, the warped results of the two images are noticeably different. However, the decoded link remains the same. This is likely because QR codes are designed with error correction capabilities, and our mobile phone cameras can handle a certain level of distortion. In other words, as long as the distortion isn't too severe, the phone can still accurately decode the QR code.

Part 4.

- **Paste your stitched panorama**



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

No. In some condition (e.g., when there are too few matched feature points between images), it's not possible to successfully stitch consecutive images into a panorama.

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

To create a panorama by stitching images together, we need to detect features, match them between images, and find a homography matrix that works best for each pair. But if there aren't enough matching points between two images, or if the matches are incorrect, we can't figure out the right transformation between them. This can lead to strange warping and heavy distortion in the final stitched result. To avoid this problem, we should choose image pairs with plenty of good matches and try not to use images that have many similar-looking objects, which can confuse the matching process.