Computer Vision HW3 Report

Student ID: R09942171
Name: 黃繼綸

Part 1. (2%)

1. Paste the function solve_homography() (1%)

```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 = np.ones(9)
    A = []

    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
    for i in range(N):
        A.append([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.append([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]])

    A = np.array(A)
    _, _, v_t = np.linalg.svd(A)

    # TODO: 2.solve H with A
    h_mat = v_t[-1, :].reshape(3,3)
    return h_mat
```

2. Paste your warped canvas (1%)

Part 2. (2%)

3. Paste the function code warping( ) (both forward & backward) (1%)

```
def warping(src, dst, H, 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 # (height, width) = (1275, 1920)
```

```
    H_inv = np.linalg.inv(H)

    # TODO: 1.meshgrid the (x,y) coordinate pairs


    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate


    warped = dst.copy()
    if direction == 'b':
        x = np.arange(0, w_dst, 1)
        y = np.arange(0, h_dst, 1)
        xx, yy = np.meshgrid(x, y)
        xx, yy = xx.flatten()[:, np.newaxis], yy.flatten()[:, np.newaxis]
        ones = np.ones((len(xx), 1))

        des_coor = np.concatenate((xx, yy, ones), axis=1).astype(np.int)
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then
reshape to (ymax-ymin),(xmax-xmin)
        Resource_pixel = H_inv.dot(des_coor.T).T # (N * 3)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the
boundaries of source image)
        Resource_pixel[:, :2] = Resource_pixel[:, :2] / Resource_pixel[:, 2][:, np.newaxis]

        out_boundary = []

        if (Resource_pixel[:, 0] < 0).any():
            out_boundary += np.where(Resource_pixel[:, 0] < 0)[0].tolist()
        if (Resource_pixel[:, 1] < 0).any():
            out_boundary += np.where(Resource_pixel[:, 1] < 0)[0].tolist()
        if (Resource_pixel[:, 0] > w_src-1).any():
            out_boundary += np.where(Resource_pixel[:, 0] > (w_src -1))[0].tolist()
        if (Resource_pixel[:, 1] > h_src-1).any():
            out_boundary += np.where(Resource_pixel[:, 1] > (h_src - 1))[0].tolist()

        # TODO: 5.sample the source image with the masked and reshaped transformed
coordinates
        if len(out_boundary):
            Resource_pixel = np.delete(Resource_pixel, out_boundary, 0)
            des_coor = np.delete(des_coor, out_boundary, 0)
```

```
        # TODO: 6. assign to destination image with proper masking
        tx = Resource_pixel[:, 0].astype(np.int)
        ty = Resource_pixel[:, 1].astype(np.int)
        dx = Resource_pixel[:, 0] - tx
        dy = Resource_pixel[:, 1] - ty

        # Bilinear interpolation
        ones = np.ones(len(dx)).astype(np.float)
        warped[des_coor[:, 1], des_coor[:, 0]] = ((((ones - dx) * (ones - dy))[:, np.newaxis]
* src[ty, tx]) \
                                                  + ((dx * (ones - dy))[:,
np.newaxis] * src[ty, tx+1]) \
                                                  + ((dx * dy)[:, np.newaxis] *
src[ty+1, tx+1]) \
                                                  + (((ones - dx) * dy)[:,
np.newaxis] * src[ty+1, tx]))


    elif direction == 'f':
        x = np.arange(0, w_src-1, 1)
        y = np.arange(0, h_src-1, 1)
        xx, yy = np.meshgrid(x, y)
        xx, yy = xx.flatten()[:, np.newaxis], yy.flatten()[:, np.newaxis]
        ones = np.ones((len(xx), 1))

        des_coor = np.concatenate((xx, yy, ones), axis=1).astype(np.int)
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to
(ymax-ymin),(xmax-xmin)
        Resource_pixel = H.dot(des_coor.T).T # (N * 3)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the
boundaries of destination image)
        Resource_pixel[:, :2] = Resource_pixel[:, :2] / Resource_pixel[:, 2][:, np.newaxis]
        out_boundary = []

        if (Resource_pixel[:, 0] < 0).any():
            out_boundary += np.where(Resource_pixel[:, 0] < 0)[0].tolist()
        if (Resource_pixel[:, 1] < 0).any():
            out_boundary += np.where(Resource_pixel[:, 1] < 0)[0].tolist()
        if (Resource_pixel[:, 0] > w_dst-1).any():
            out_boundary += np.where(Resource_pixel[:, 0] > (w_dst -1))[0].tolist()
        if (Resource_pixel[:, 1] > h_dst-1).any():
```

```
            out_boundary += np.where(Resource_pixel[:, 0] > (h_dst - 1))[0].tolist()

        # TODO: 5.filter the valid coordinates using previous obtained mask
        if len(out_boundary):
            Resource_pixel = np.delete(Resource_pixel, out_boundary, 0)
            des_coor = np.delete(des_coor, out_boundary, 0)

        # TODO: 6. assign to destination image using advanced array indicing
        tx = Resource_pixel[:, 0].astype(np.int)
        ty = Resource_pixel[:, 1].astype(np.int)
        dx = Resource_pixel[:, 0] - tx
        dy = Resource_pixel[:, 1] - ty

        # warped[Resource_pixel[:, 1], Resource_pixel[:, 0]] = src[des_coor[:, 1],
des_coor[:, 0]]

        ones = np.ones(len(dx)).astype(np.float)

        # Bilinear interpolation
        warped[ty, tx] = ((((ones - dx) * (ones - dy))[:, np.newaxis] * src[des_coor[:, 1],
des_coor[:, 0]]) \
                                                    + ((dx * (ones - dy))[:,
np.newaxis] * src[des_coor[:, 1], des_coor[:, 0]+1]) \
                                                    + ((dx * dy)[:, np.newaxis] *
src[des_coor[:, 1]+1, des_coor[:, 0]+1]) \
                                                    + (((ones - dx) * dy)[:,
np.newaxis] * src[des_coor[:, 1]+1, des_coor[:, 0]]))

    return warped
```

4. Briefly introduce the interpolation method you use (1%)

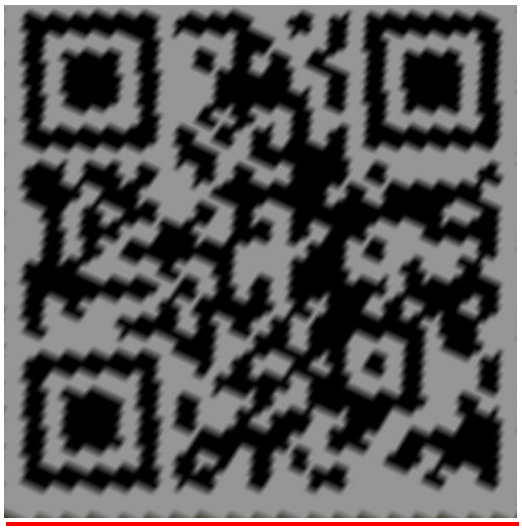
I used the bilinear interpolation method. The formulation is as follow:

$$\big((1-dx)*(1-dy)\big)*src[y,x]+\big((\mathrm{dx})*\ (1\ -\ \mathrm{dy})\big)*src[y,\ x+1]+(dx*dy)*$$

$$src[y+1,x+1]+\big((1-dx)*dy\big)*src[y+1,x]$$

Part 3. (8%)

1. **Paste the 2 warped QR code and the link you find (1%)**
   Link: http://media.ee.ntu.edu.tw/courses/cv/21S/



2. **Discuss the difference between 2 source images, are the warped results the same or different? (3%)**
   The results are different. The top figure is more apparent than the down figure.
3. **If the results are the same, explain why. If the results are different, explain why. (4%)**
   The original image of the down figure is slightly curved, causing the information to be compressed. While the original image of the top figure is a perfect planar, so the transformed image is clearer than the down figure.

Part 4. (8%)

1. **Paste your stitched panorama (1%)**

2. **Can all consecutive images be stitched into a panorama? (3%)**

   No

3. **If yes, explain your reason. If not, explain under what conditions will result in a failure? (4%)**

   If all consecutive images have no overlapping and similar features, which might not find the matched points to stitch the images into a panorama.