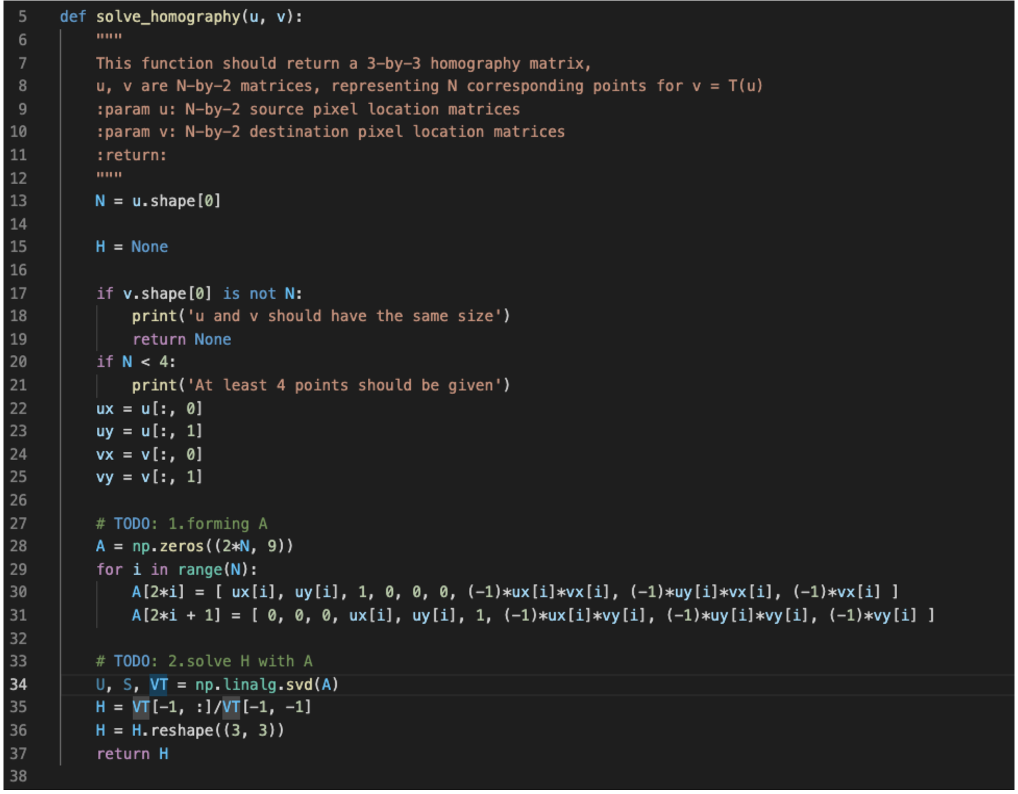
HW3 Report

Student ID: R10625016
Name: 許致銓

Part1:

1. Solve_homography():

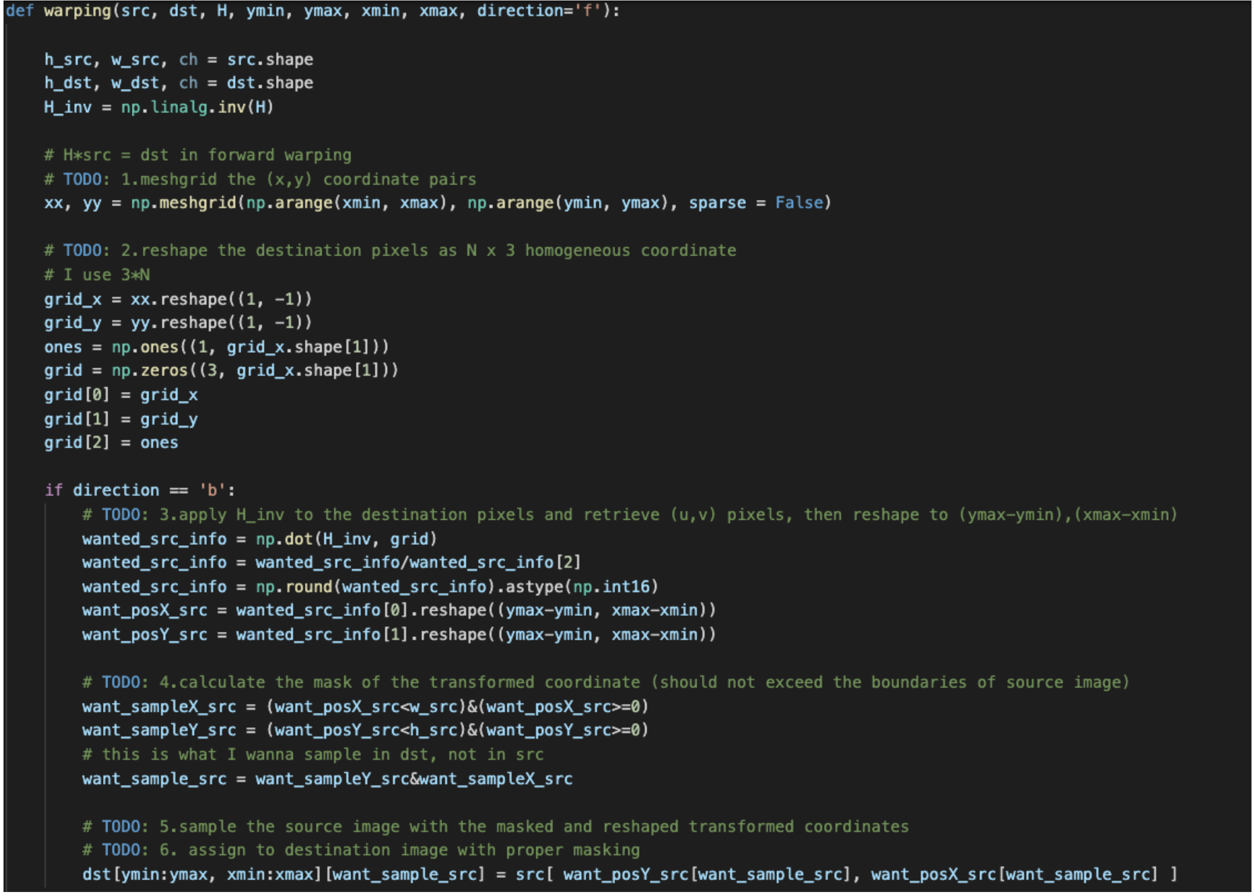


```
def solve_homography(u, v):
    """
    This function should return a 3-by-3 homography matrix,
    u, v are N-by-2 matrices, representing N corresponding points for v = T(u)
    :param u: N-by-2 source pixel location matrices
    :param v: N-by-2 destination pixel location matrices
    :return:
    """
    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')
    ux = u[:, 0]
    uy = u[:, 1]
    vx = v[:, 0]
    vy = v[:, 1]

    # TODO: 1.forming A
    A = np.zeros((2*N, 9))
    for i in range(N):
        A[2*i] = [ ux[i], uy[i], 1, 0, 0, 0, (-1)*ux[i]*vx[i], (-1)*uy[i]*vx[i], (-1)*vx[i] ]
        A[2*i + 1] = [ 0, 0, 0, ux[i], uy[i], 1, (-1)*ux[i]*vy[i], (-1)*uy[i]*vy[i], (-1)*vy[i] ]

    # TODO: 2.solve H with A
    U, S, VT = np.linalg.svd(A)
    H = VT[-1, :]/VT[-1, -1]
    H = H.reshape((3, 3))
    return H
```

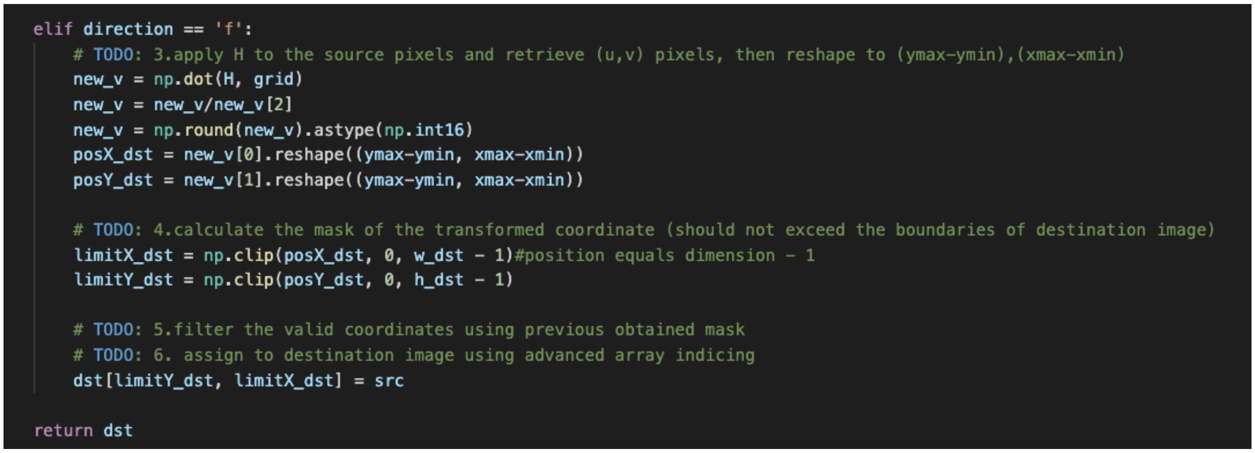
2. Warped canvas

Part2:

1. Warping()



```python
def warping(src, dst, H, ymin, ymax, xmin, xmax, direction='f'):

    h_src, w_src, ch = src.shape
    h_dst, w_dst, ch = dst.shape
    H_inv = np.linalg.inv(H)

    # H*src = dst in forward warping
    # TODO: 1.meshgrid the (x,y) coordinate pairs
    xx, yy = np.meshgrid(np.arange(xmin, xmax), np.arange(ymin, ymax), sparse = False)

    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    # I use 3*N
    grid_x = xx.reshape((1, -1))
    grid_y = yy.reshape((1, -1))
    ones = np.ones((1, grid_x.shape[1]))
    grid = np.zeros((3, grid_x.shape[1]))
    grid[0] = grid_x
    grid[1] = grid_y
    grid[2] = ones

    if direction == 'b':
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        wanted_src_info = np.dot(H_inv, grid)
        wanted_src_info = wanted_src_info/wanted_src_info[2]
        wanted_src_info = np.round(wanted_src_info).astype(np.int16)
        want_posX_src = wanted_src_info[0].reshape((ymax-ymin, xmax-xmin))
        want_posY_src = wanted_src_info[1].reshape((ymax-ymin, xmax-xmin))

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)
        want_sampleX_src = (want_posX_src<w_src)&(want_posX_src>=0)
        want_sampleY_src = (want_posY_src<h_src)&(want_posY_src>=0)
        # this is what I wanna sample in dst, not in src
        want_sample_src = want_sampleY_src&want_sampleX_src

        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
        # TODO: 6. assign to destination image with proper masking
        dst[ymin:ymax, xmin:xmax][want_sample_src] = src[ want_posY_src[want_sample_src], want_posX_src[want_sample_src] ]
```

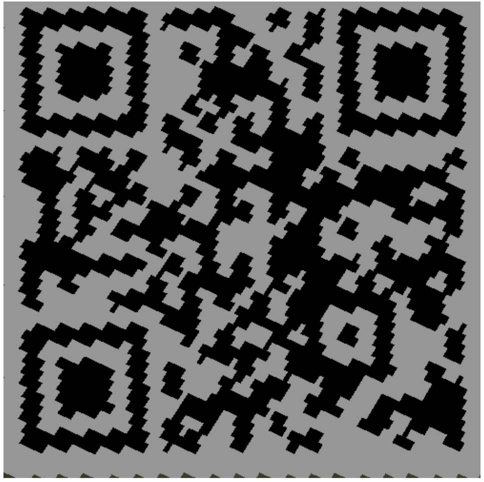


```
elif direction == 'f':
    # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
    new_v = np.dot(H, grid)
    new_v = new_v/new_v[2]
    new_v = np.round(new_v).astype(np.int16)
    posX_dst = new_v[0].reshape((ymax-ymin, xmax-xmin))
    posY_dst = new_v[1].reshape((ymax-ymin, xmax-xmin))

    # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
    limitX_dst = np.clip(posX_dst, 0, w_dst - 1)#position equals dimension - 1
    limitY_dst = np.clip(posY_dst, 0, h_dst - 1)

    # TODO: 5.filter the valid coordinates using previous obtained mask
    # TODO: 6. assign to destination image using advanced array indicing
    dst[limitY_dst, limitX_dst] = src

return dst
```

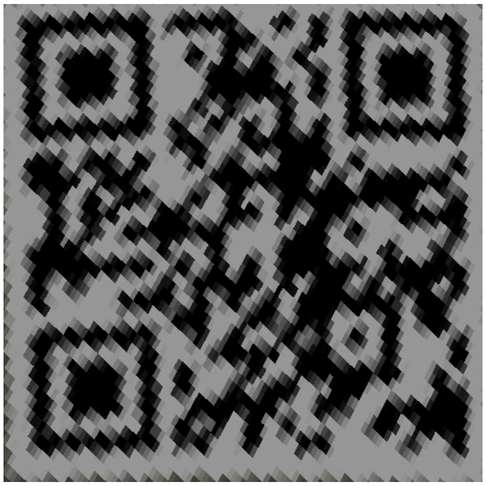
2. **Interpolation introduction:**
   For forward warping, I use the numpy function "round" as the KNN method. It is the same as the KNN when K = 1, which means the pixel value equals to the nearest pixel.
   As for backward warping, I also use the round function to implement the KNN when K = 1.

Part3:

1. **2 QR code:**



output3-1



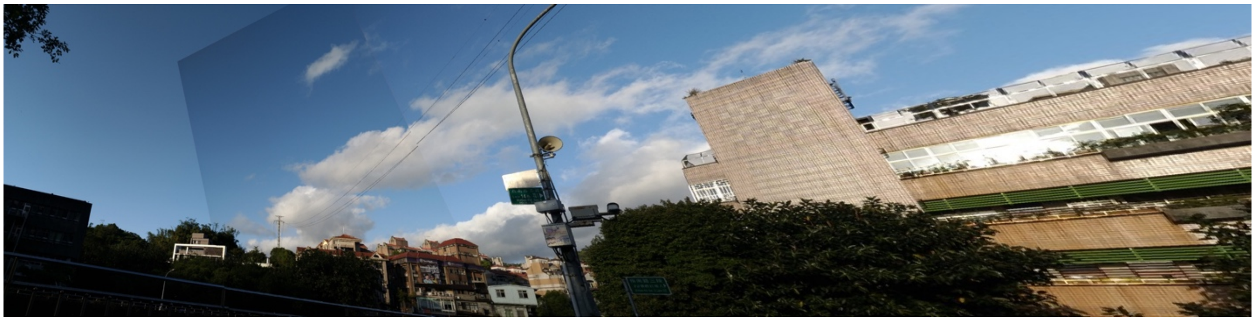
output3-2

2. **Links:** Both of them are http://media.ee.ntu.edu.tw/courses/cv/21S/.
3. **Discussion:**
   No, they are not the same. The first one is clearer, while the second one is a bit blurred.
4. **Explanation:**
   Because source 1 is in a more suitable ratio of the height and width, the picture can contain more information, and the height and width of the picture are more complete. Conversely, the source 2 picture is more compressed and confined to a smaller relative space though the width and the height are both larger or near

to the source 1 picture. To sum up, if QR code picture is as the same height & width ratio as the original picture, then it can contain more correct information. However, if you compress the picture or change the height & width ratio, there must be some transformation and reduction of the picture information.

Part4:

1. **Panorama:**



2. **Can all consecutive images be stitched into a panorama?**
   No, not all of them.
3. **Explanation:**
   The availability of stitching depends on the way we want to make a panorama. There are several types of panoramas such as rectilinear, cylindrical, circular, etc. Nevertheless, they share the common idea that it is impossible to exceed 360 FOV due to the physical world limitation, and for each method, there will be different limitation of panorama.
   Take methods below for example:
   ①. Rectilinear:
      It is so called flat projection, which is the same as our vision and camera projection.
      Pictures can’t be stitched if the angle exceeds 180 degrees, both vertically and horizontally. Noticeably, if the angle is more than 120 degrees, then the projection will be distorted significantly.
   ②. Cylindrical:
      In this method, we can stitch pictures within 360 degrees horizontally. However, the vertical limiting angles is 180 degrees, and property is similar to Rectilinear.

Reference:

- https://ptgui.com/man/projections.html
- https://en.wikipedia.org/wiki/Image_stitching
- https://en.wikipedia.org/wiki/Map_projection#Cylindrical