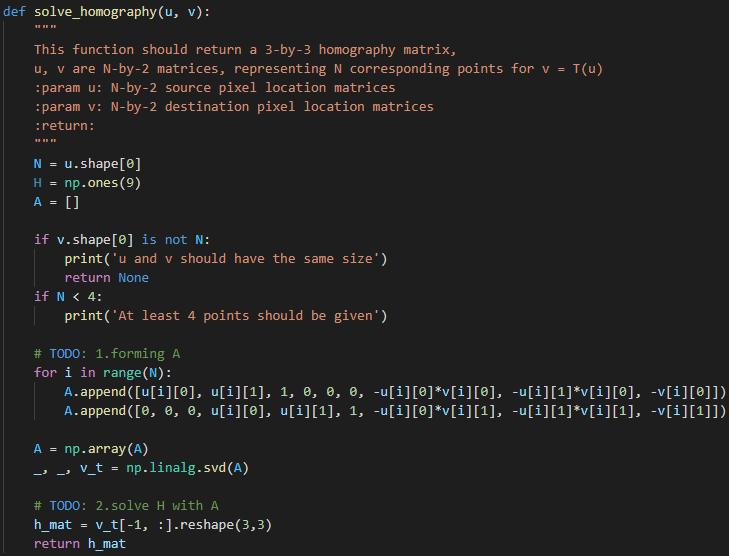
**Computer Vision HW3 Report**

Student ID: R09942171

Name: 黃繼綸

**Part 1. (2%)**

1. **Paste the function solve\_homography() (1%)**



1. **Paste your warped canvas (1%)**



**Part 2. (2%)**

1. **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

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

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

**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/**

****

****

1. **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.

1. **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%)**

****

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

No

1. **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.