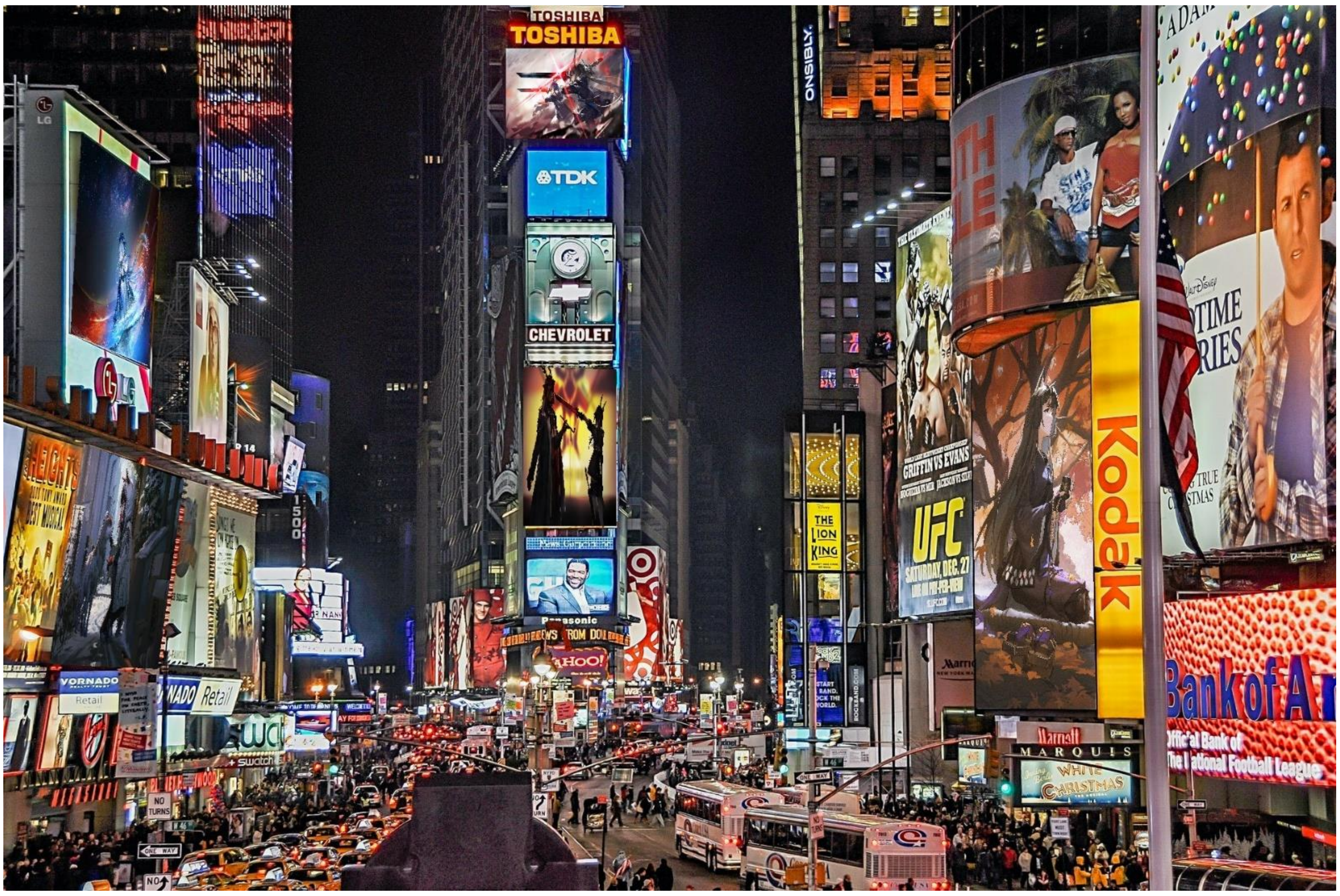
Computer Vision HW3 Report

Student ID: R11521201
Name: 廖沁柔

Part 1.

- **Paste your warped canvas**



Part 2.

- **Paste the function code**

solve_homography(u, v)

```
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 (參考 Lec09 p.12 DLT Algorithm)
    # 公式參照 p.35 Importance of Normalization
    A = []
    for i in range(N):
        x, y = u[i] # 原點
        x_p, y_p = v[i] # 投影點(Projective)
        A.append([x, y, 1, 0, 0, 0, -x*x_p, -y*x_p, -x_p])
        A.append([0, 0, 0, -x, -y, -1, x*y_p, y*y_p, y_p])
    A = np.array(A)

    # TODO: 2.solve H with A
    # 求解參照 p.12 (iii) Obtain SVD of A.
    # U, S, V = np.linalg.svd(A)
    # U: 左奇異向量 / S: 由大到小排序的奇異值 / V: 右奇異向量
    _, _, V = np.linalg.svd(A)

    # 解 H 只需要用到 V 的最後一行
    H = V[-1].reshape(3,3)

    return H
```

warping( ) (both forward &   backward) **(為方便排版，將調整部分註解)**

```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 # height & width of source img
    h_dst, w_dst, ch = dst.shape # height & width of destination output img
    H_inv = np.linalg.inv(H) # calculate the inverse of H

    # TODO: 1.meshgrid the (x,y) coordinate pairs
    # 生成網格：透過 meshgrid 生成座標網格
    xc, yc = np.meshgrid(np.arange(xmin, xmax, 1), np.arange(ymin, ymax, 1))
    # 指定範圍內的 pixel 總數
    xrow = xc.reshape(( 1,(xmax-xmin)*(ymax-ymin) ))
    yrow = yc.reshape(( 1,(xmax-xmin)*(ymax-ymin) ))
    # 為構成齊次矩陣 -> 需要一個全為 1 的 row
    all_onerow =  np.ones(( 1,(xmax-xmin)*(ymax-ymin) ))

    # TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
    # 建立齊次座標：生成 Nx3 三維的(x, y, 1)矩陣 M
    # 縱向拼接 -> axis = 0
    M = np.concatenate((xrow, yrow, all_onerow), axis = 0)

    if direction == 'b': # back warping
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels,
        # then reshape to (ymax-ymin),(xmax-xmin)
        # back warping: destination * H_inv
        V = np.dot(H_inv, M)
        Vx, Vy, _ = V/V[2] # 進行 normalize (將第三行都賦值 1，故為"_")
        Vx = Vx.reshape(ymax-ymin, xmax-xmin)
        Vy = Vy.reshape(ymax-ymin, xmax-xmin)

        # TODO: 4.calculate the mask of the transformed coordinate
        h_src, w_src, ch = src.shape
        # Vx < w_src-1: 不超過原圖右邊介；0 <= Vx: 確保不會是負數
        # Vy < h_src-1: 不超過原圖下邊界；0 <= Vy: 確保不會是負數
        # index 從 0 開始
        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
        mask_Vx = Vx[mask]
        mask_Vy = Vy[mask]
        # 處理 sub-pixel location
        # Step 1: Nearest neighbor
        mask_Vxint = mask_Vx.astype(int)
        mask_Vyint = mask_Vy.astype(int)
        # Step 2: Bilinear interpolation
        dX = (mask_Vx - mask_Vxint).reshape((-1,1))
        dY = (mask_Vy - mask_Vyint).reshape((-1,1))
        p = np.zeros((h_src, w_src, ch))
        # 找尋鄰近的 4 個點
        # 左上角、左下角、右上角、右下角
        p[mask_Vyint, mask_Vxint, :] += (1-dY)*(1-dX)*src[mask_Vyint, mask_Vxint, :]
        p[mask_Vyint+1, mask_Vxint, :] += dY*(1-dX)*src[mask_Vyint+1, mask_Vxint, :]
        p[mask_Vyint, mask_Vxint+1, :] += (1-dY)*dX*src[mask_Vyint, mask_Vxint+1, :]
        p[mask_Vyint+1, mask_Vxint+1, :] += dY*dX*src[mask_Vyint+1, mask_Vxint+1, :]

        # TODO: 6. assign to destination image with proper masking
        dst[ymin:ymax,xmin:xmax][mask] = p[mask_Vyint,mask_Vxint]

        pass

    elif direction == 'f': # forward warping
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels,
        # then reshape to (ymax-ymin),(xmax-xmin)
        # forward warping: source * H # 較為直觀
        V = np.dot(H,M)
        V = (V/V[2]).astype(int)
        Vx, Vy, _ = V
        Vx = Vx.reshape(ymax-ymin, xmax-xmin)
        Vy = Vy.reshape(ymax-ymin, xmax-xmin)
        # TODO: 4.calculate the mask of the transformed coordinate
        mask = (((Vx<w_dst-1)&(0<=Vx))&((Vy<h_dst-1)&(0<=Vy)))

        # TODO: 5.filter the valid coordinates using previous obtained mask
        mask_Vx = Vx[mask]
        mask_Vy = Vy[mask]

        # TODO: 6. assign to destination image using advanced array indicing
        dst[mask_Vy, mask_Vx, :] = src[mask]

        pass

    return dst
```

• Briefly introduce the interpolation method you use

Step 1: Nearest Neighbor

First, it determines the nearest integer pixel coordinates for each sub-pixel location. This is done by converting the floating-point coordinates (mask_Vx, mask_Vy) to integer values (mask_Vxint, mask_Vyint). These integer coordinates refer to the top-left corner of the 2x2 pixel grid surrounding the actual sub-pixel point.

Step 2: Bilinear Interpolation

Compute fractional parts:

The code calculates the fractional parts of the original floating-point coordinates:

- **dX = (mask_Vx - mask_Vxint):**
  Horizontal distance from the actual sub-pixel point to the left border of the pixel.
- **dY = (mask_Vy - mask_Vyint):**
  Vertical distance from the actual sub-pixel point to the top border of the pixel.

Determine weights:

- **(dY)*(1-dX): Weight for the top-left pixel.**
- **dY*(1-dX): Weight for the bottom-left pixel.**
- **(dY)*dX: Weight for the top-right pixel.**
- **dY*dX: Weight for the bottom-right pixel.**

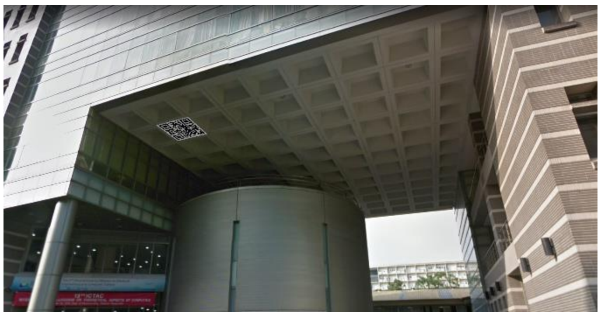
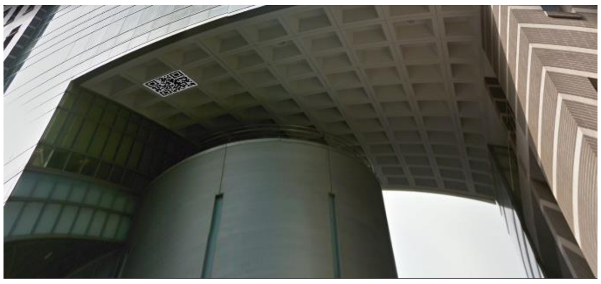
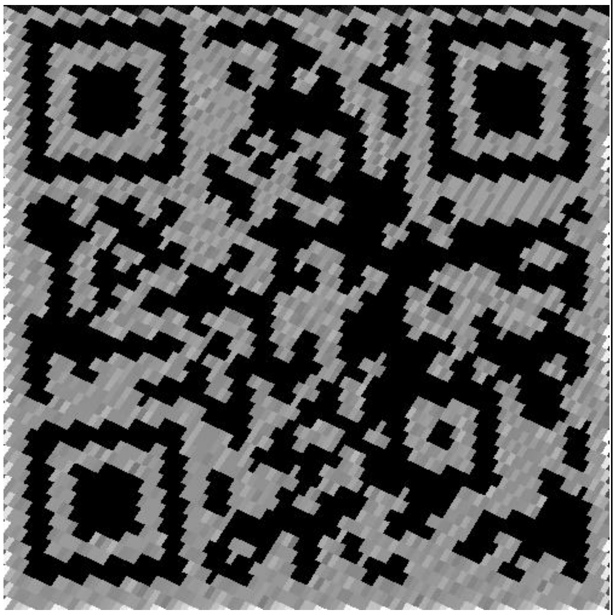
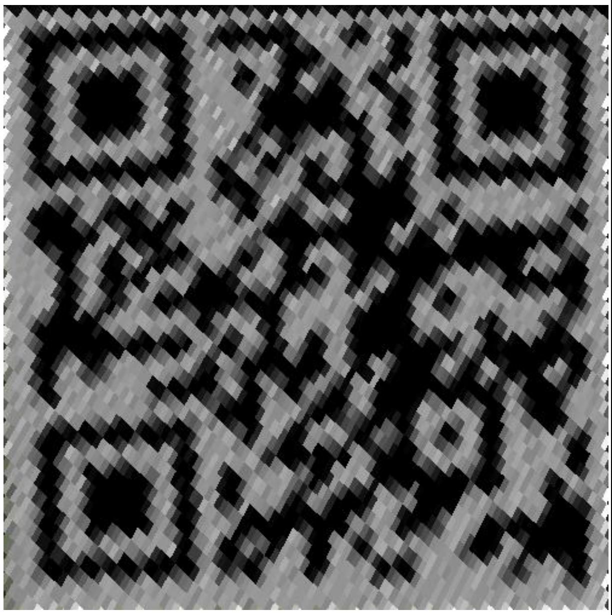
These weights are proportional to the area opposite the corner being considered, effectively a measure of how close the sub-pixel point is to each corner of the 2x2 grid.

Weighted sum:

- Each corner pixel's intensity is multiplied by its corresponding weight to get a contribution to the final interpolated value.
- The contributions from all four neighboring pixels are summed up to get the interpolated value at the sub-pixel location. This sum is stored in the output array ‘p’.

Part 3.

• Paste the 2 warped images and the link you find

SRC name	**BL_secret1**	**BL_secret2**
SRC Image		
Output name	**output3_1**	**output3_2**
Output Image		
Link	Computer Vision | Spring 2021 (ntu.edu.tw)	Computer Vision | Spring 2021 (ntu.edu.tw)

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

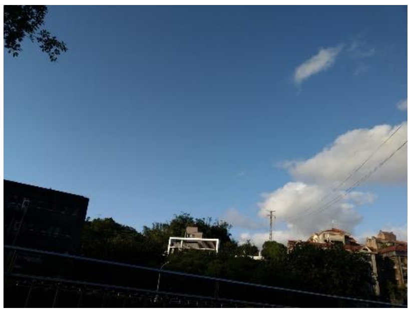
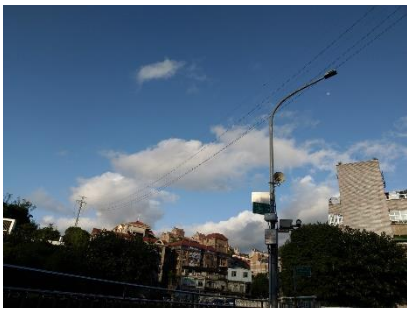
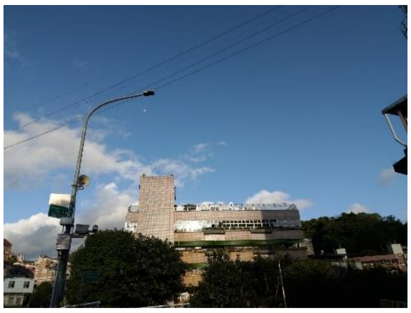
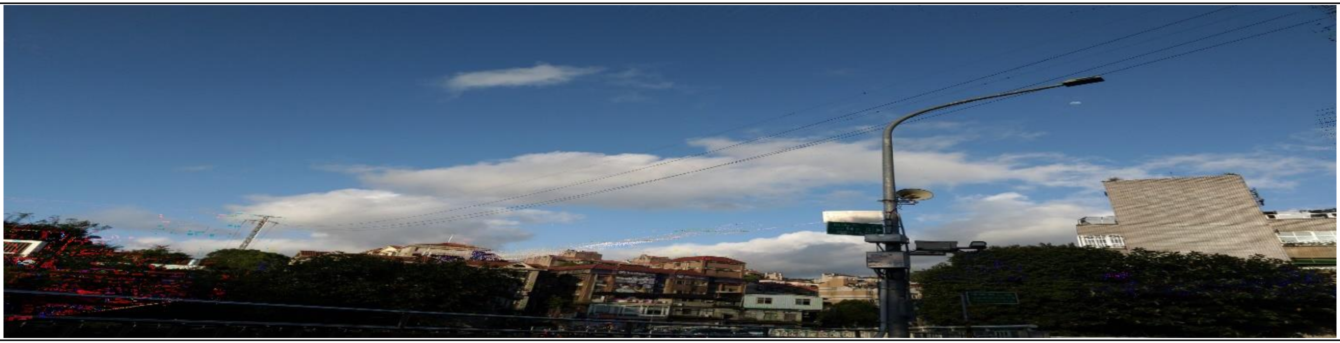
• If the results are the same, explain why. If the results are different, explain why?

Ans:

Both transformed images of QR codes link to the computer vision course website. However, output3_1 is noticeably clearer. Looking back at the original images, it was observed that BL_secret1 is more squared (linear), while BL_secret2 is distorted (curved). Therefore, under the same transformation settings, the latter appears relatively blurry.

Part 4.

- **Paste your stitched panorama**

Frame 1	Frame 2	Frame 3
		
Stitched Panorama		
		

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

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

Ans:

Upon visually inspecting the stitched panorama, **the process seems to have been carried out successfully**. However, there is a discernible loss of information along the edges, particularly on the left and right sides of the image.

This phenomenon is likely attributed to the method used for stitching, which involves selectively filtering and matching keypoints. Because of this selective process, any keypoints that do not find a match in the corresponding images are excluded from the final composition. This approach, while effective in ensuring the alignment and congruity of matched points, unfortunately results in the exclusion of potentially valuable image data where no direct keypoint correlations exist.

This effect could be mitigated by employing a stitching method that incorporates a more inclusive keypoint retention strategy or by enhancing the keypoint matching algorithm to increase the likelihood of finding matches across all regions of the images involved.