NTU CV HW3

# Part 1

1. **Paste the function 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')

    # I prefer solution 2

    # TODO: 1.forming A

    A\_row\_list = []

    for point\_u, point\_v in zip(u, v):

        # print(point\_u, ', ', point\_v)

        arr1 = np.array([point\_u[0], point\_u[1], 1, 0, 0, 0, -1 \* point\_u[0] \* point\_v[0], -1 \* point\_u[1] \* point\_v[0], -1 \* point\_v[0]])

        arr2 = np.array([0, 0, 0, point\_u[0], point\_u[1], 1, -1 \* point\_u[0] \* point\_v[1], -1 \* point\_u[1] \* point\_v[1], -1 \* point\_v[1]])

        A\_row\_list.append(arr1)

        A\_row\_list.append(arr2)

    A\_mat = np.vstack(A\_row\_list)

    # TODO: 2.solve H with A

    (U, D, V) = np.linalg.svd(A\_mat)

    V = np.transpose(V)

    H = V[:, -1]

    H = H.reshape((3, 3))

    return H

1. **Paste your warped canvas**



# Part 2

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

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

    meshgrid\_x, meshgrid\_y = np.meshgrid(np.arange(xmin, xmax), np.arange(ymin, ymax))

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

    pixels\_idx = np.vstack([

        meshgrid\_x.reshape(meshgrid\_x.shape[0]\*meshgrid\_x.shape[1]),

        meshgrid\_y.reshape(meshgrid\_y.shape[0]\*meshgrid\_y.shape[1]),

        np.ones((meshgrid\_y.shape[0]\*meshgrid\_y.shape[1]), dtype=int)

    ])

    if direction == 'b':

        # TODO: 3.apply H\_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)

        new\_pixels\_idx = np.dot(H\_inv, pixels\_idx)

        new\_pixels\_idx[0, :] = np.divide(new\_pixels\_idx[0, :], new\_pixels\_idx[2, :])

        new\_pixels\_idx[1, :] = np.divide(new\_pixels\_idx[1, :], new\_pixels\_idx[2, :])

        new\_pixels\_idx[2, :] = np.ones\_like(new\_pixels\_idx[2, :])

        new\_pixels\_idx = new\_pixels\_idx.reshape((3, ymax-ymin, xmax-xmin))

        new\_pixels\_idx = np.round(new\_pixels\_idx).astype(int)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)

        mask = np.ones\_like(new\_pixels\_idx, dtype=bool)

        mask[0, :, :] = (new\_pixels\_idx[0, :, :] >= 0) & (new\_pixels\_idx[0, :, :] < w\_src)

        mask[1, :, :] = (new\_pixels\_idx[1, :, :] >= 0) & (new\_pixels\_idx[1, :, :] < h\_src)

        new\_mask = mask[0, :, :] & mask[1, :, :]

        new\_mask = new\_mask.reshape((ymax-ymin, xmax-xmin))

        # Turn invalid pixel index to (0, 0, 0), without changing the shape of pixel index array (keep it as (3, ymax-ymin, xmax-xmin))

        new\_pixels\_idx[:, ~new\_mask] = 0

        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates

        source\_img\_sample =  src[new\_pixels\_idx[1, :, :], new\_pixels\_idx[0, :, :], :]

        # TODO: 6. assign to destination image with proper masking

        dst[ymin:ymax, xmin:xmax, :][new\_mask, :] = source\_img\_sample[new\_mask, :]

    elif direction == 'f':

        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)

        new\_pixels\_idx = np.dot(H, pixels\_idx)

        new\_pixels\_idx[0, :] = np.divide(new\_pixels\_idx[0, :], new\_pixels\_idx[2, :])

        new\_pixels\_idx[1, :] = np.divide(new\_pixels\_idx[1, :], new\_pixels\_idx[2, :])

        new\_pixels\_idx[2, :] = np.ones\_like(new\_pixels\_idx[2, :])

        new\_pixels\_idx = new\_pixels\_idx.reshape((3, ymax-ymin, xmax-xmin))

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)

        mask = np.ones\_like(new\_pixels\_idx, dtype=bool)

        mask[0, :, :] = (new\_pixels\_idx[0, :, :] >= 0) & (new\_pixels\_idx[0, :, :] < w\_dst)

        mask[1, :, :] = (new\_pixels\_idx[1, :, :] >= 0) & (new\_pixels\_idx[1, :, :] < h\_dst)

        new\_mask = (mask[0, :, :] & mask[1, :, :])

        # TODO: 5.filter the valid coordinates using previous obtained mask

        valid\_coord\_idx = new\_pixels\_idx[:, new\_mask]

        valid\_coord\_idx = np.round(valid\_coord\_idx).astype(int)

        valid\_coord\_idx = valid\_coord\_idx[:2, :]

        valid\_coord\_idx = valid\_coord\_idx.reshape(2, ymax-ymin, xmax-xmin)

        # TODO: 6. assign to destination image using advanced array indicing

        dst[valid\_coord\_idx[1, :, :], valid\_coord\_idx[0, :, :], :] = \

        src[new\_mask, :].reshape((src.shape[0], src.shape[1], src.shape[2]))

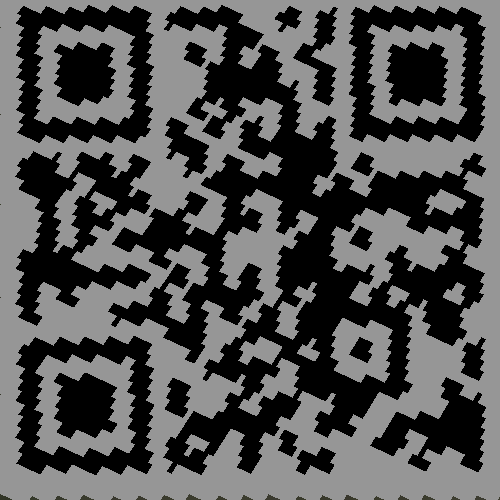
    return dst

1. **Briefly introduce the interpolation method you use**

我沒有做特別的interpolation方法，單純在backward warpping遇到小數時，就直接取整數，才能當作index使用，應該算是Nearest Neighbor。

# Part 3

1. **Paste the 2 warped QR code and the link you find**
2. QR code from BL\_secret1.png



1. QR code from BL\_secret2.png



1. Link of the QR codes (Scan by Android Line app)

<https://qrgo.page.link/jc2Y9>

* 這堂課的課程網頁
* 兩個QR code掃描結果一樣
* BL\_secret2.png比較模糊，所以要拿遠一點掃描才會偵測得出來

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

兩張source image是不同的，BL\_secret1感覺上是one point perspective的照片，部分原本真實建築物上是平行的線，在照片上依然是平行的。而BL\_secret2感覺原本的直線都彎曲了，特別是在照片的中心，所以猜測是Barrel distortion，可能是相機鏡片較厚導致的。

兩張source image warp的結果是不同的，最明顯的差異應該會是在右上方的大正方形，BL\_secret1是正方形，但是BL\_secret2有一點變形成平行四邊形。還有BL\_secret2的圖片比較模糊，感覺某些pixel的顏色錯位了，所以有灰色的部分。但是都可以掃描出相同的QR code link。

1. **If the results are the same, explain why. If the results are different, explain why.**

原因就是這part做homography的時候是使用四個source的點map到destination中，這樣做會假設source的QR code是平行四邊形，所以才能透過homography拉伸成正方形。而BL\_secret2的QR code並非平行四邊形，他的邊線因為distortion的關係，並非直線而是曲線，所以warp的結果才會有變形的現象。

# Part 4

1. **Paste your stitched panorama**



1. **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?**

就算是連續影像也**不一定**能夠組成一張全景圖。

Homography會假設三張不同照片是從同一焦點卻是不同角度拍攝一個更遠的平面景象。但若是在室內拍攝多張影像，只要碰到轉角就不是同一平面了，也就會讓原本是平面的牆壁，變成像是圓弧狀的牆面。

1. **[Bonus] Using homography to produce a “more than 2 images panorama”**

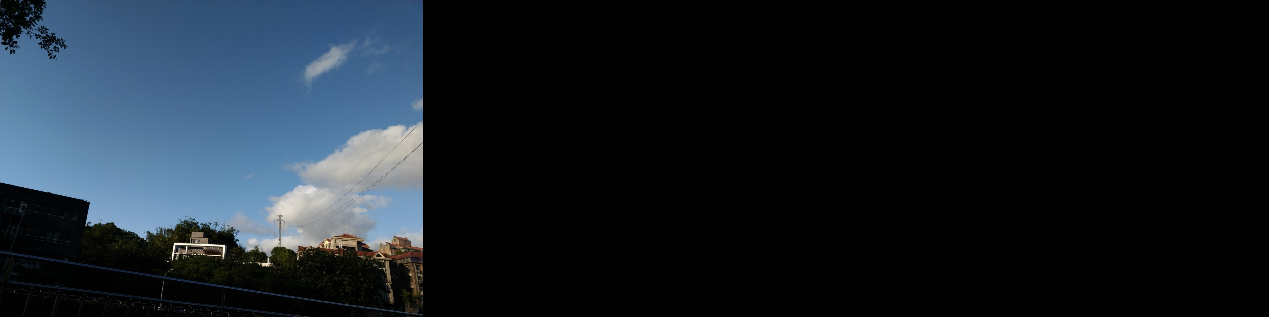
我使用了3張影像合成一張全景圖

1. **[Bonus] Using blending techniques (simple alpha blending)\***

我實作了簡單的alpha blending，會將重疊到的部分相加後取平均。

假設已經求得三張圖片的homography了

1. 先將3個影像分別投影在不同的3個canvas上，但是同樣大小且背景都是0 (黑色)







1. 建立一個大小為canvas的mask，將重疊的部分、只有一張圖片的部分和沒有圖片的部分分別給予不同的值。
2. 根據mask的數值做不同的alpha blending，重疊的部分各自的權重為0.5；只有一張圖片的部分，有圖片的權重為1，沒有圖片的為0；完全沒有圖片的部分就維持0(黑色)。

需要將重疊與菲重疊區隔開來是因為，如果只有單張影像在canvas的某個位置上，如果取平均就只是將那張影像的數值除以2變暗而已，並不是理想中的效果。

\*Reference of alpha blending: <https://inst.eecs.berkeley.edu/~cs194-26/fa17/upload/files/proj6B/cs194-26-abw/>