Computer Vision HW3 Report

Student ID:	R11921038
Name:	江讀晉

Part 1. Homography Estimation (1%)

• Paste the forward warped canvas. (1%)

Ans:

Part 2. Marker-Based Planar AR (3%)

• Paste the function code *solve_homography(*)* and *warping(*)* (both forward and backward). (1+1%)

Ans:

solve_homography(u, v)

```
1  def solve_homography(u, v):
2      """
3      This function should return a 3-by-3 homography matrix,
4      u, v are N-by-2 matrices, representing N corresponding points for v = T(u)
5      :param u: N-by-2 source pixel location matrices
6      :param v: N-by-2 destination pixel location matrices
7      :return:
8      """
9      N = u.shape[0]
10     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')

    # 0. Transform u, v to homogeneous coordinate
    u = np.hstack((u, np.ones((N, 1))))
    v = np.hstack((v, np.ones((N, 1))))

    # TODO: 1.forming A
    A = np.zeros((2 * N, 9))
    A[::2, :3] = u
    A[::2, 3:6] = 0
    A[::2, 6:] = -u * v[:, 0].reshape((-1, 1))
    A[1::2, :3] = 0
    A[1::2, 3:6] = u
    A[1::2, 6:] = -u * v[:, 1].reshape((-1, 1))

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

    return H
```

warping(src, dst, H, ymin, ymax, xmin, xmax, direction)

```
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
    x, y = np.meshgrid(np.arange(xmin, xmax), np.arange(ymin, ymax))

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

    if direction == 'b':
        '''
        backward warping
            :var interpolation:    'bilinear' or 'nearest'
            :var warped_homo_coord: (3, N). N equals to (ymax-ymin) * (xmax-xmin)
            :var homo_coord:       (3, N)
            :var coord:           (N, 2)
            :var mask:            (N,)
            :var filtered_coord    (M, 2). M is the number of elements passing the mask.
```

```
        '''
        interpolation = 'bilinear'
        warped_homo_coord = np.vstack((x.flatten(), y.flatten(), np.ones((1, x.size))))    # (3, N)

        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        homo_coord = np.dot(H_inv, warped_homo_coord)
        homo_coord /= homo_coord[-1, :]
        homo_coord[[0, 1]] = homo_coord[[1, 0]] # exchange x and y

        coord = np.round((homo_coord.T)[:, :-1])

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)
        upper_bound = np.zeros_like(coord)
        upper_bound[:, 0], upper_bound[:, 1] = h_src, w_src

        if interpolation == 'bilinear':
            # sloppily, ignore the pixels on the boundary :-/
            mask = np.logical_and(coord >= 1, coord < (upper_bound - 1))
        else:
            mask = np.logical_and(coord >= 0, coord < upper_bound)
        mask = np.logical_and(mask[:, 0], mask[:, 1])

        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
        filtered_coord = coord[mask]
        floor_coord = (np.floor(filtered_coord)).astype(np.int32)
        delta = filtered_coord - floor_coord
        delta = np.expand_dims(delta, axis=2)

        # TODO: 6. assign to destination image with proper masking
        output = (np.copy(dst[ymin:ymax, xmin:xmax, :])).reshape(-1, ch)
        if interpolation == 'bilinear':
            output[mask] = \
                src[floor_coord[:, 0], floor_coord[:, 1], :] * (1 - delta[:, 0]) * (1 - delta[:, 1]) + \
                src[floor_coord[:, 0], floor_coord[:, 1] + 1, :] * delta[:, 0] * (1 - delta[:, 1]) + \
                src[floor_coord[:, 0] + 1, floor_coord[:, 1], :] * (1 - delta[:, 0]) * delta[:, 1] + \
                src[floor_coord[:, 0] + 1, floor_coord[:, 1] + 1, :] * delta[:, 0] * delta[:, 1]
        else:
            filtered_coord = filtered_coord.astype(np.int32)
            output[mask] = src[filtered_coord[:, 0], filtered_coord[:, 1], :]

        dst[ymin:ymax, xmin:xmax, :] = output.reshape(ymax-ymin, xmax-xmin, ch)

    elif direction == 'f':
        '''
        forward warping
            :var homo_coord:       (3, N).  N equals to (ymax-ymin) * (xmax-xmin)
            :var warped_homo_coord: (3, N)
            :var warped_coord:     (N, 2)
            :var mask:            (N,)
            :var filtered_coord    (M, 2). M is the number of elements passing the mask.
        '''
        homo_coord = np.vstack((x.flatten(), y.flatten(), np.ones((1, x.size))))

        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        warped_homo_coord = np.dot(H, homo_coord)
        warped_homo_coord /= warped_homo_coord[-1, :]
        warped_homo_coord[[0, 1]] = warped_homo_coord[[1, 0]]   # exchange x and y

        warped_coord = np.round((warped_homo_coord.T)[:, :-1]).astype(np.int32)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
        upper_bound = np.zeros_like(warped_coord)
        upper_bound[:, 0], upper_bound[:, 1] = h_dst, w_dst
        mask = np.logical_and(warped_coord >= 0, warped_coord < upper_bound)
        mask = np.logical_and(mask[:, 0], mask[:, 1])

        # TODO: 5.filter the valid coordinates using previous obtained mask
        mask = mask.reshape(ymax-ymin, xmax-xmin)
        warped_coord = warped_coord.reshape(ymax-ymin, xmax-xmin, -1)
        filtered_coord = warped_coord[mask]

        # TODO: 6. assign to destination image using advanced array indicing
        dst[filtered_coord[:, 0], filtered_coord[:, 1], :] = src[mask]

    return dst
```

• Briefly introduce the implemented interpolation method. (1%)

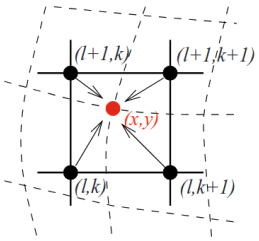
Ans:

對於 forward warping，interpolation 的方式為 *nearest neighbor*，亦即將所求的座標四捨五入至整數，且沒有特別處理「hole」問題，如第 1 頁 Part 1 輸出結果所示。

至於 backward warping，interpolation 的方式為 *nearest neighbor* 和 *bilinear*，繳交的程式碼預設使用 bilinear interpolation。Bilinear interpolation 參考所求座標周邊的四個點，依據各點的距離決定對應權重，如下圖所示。此外，如果所求座標恰位於邊界上，則無法找出周遭四個點，較好的解決方式為改找鄰近點，然而，此次作業使用的圖片尺寸較大，故僅忽略邊界上的所求座標。

$$
\begin{aligned}
f(x,y) &= (1-a)(1-b)\, g_s(l,k) + a\,(1-b)\, g_s(l+1,k) \\
&+ b\,(1-a)\, g_s(l,k+1) + ab\, g_s(l+1,k+1)\,,
\end{aligned}
$$

$$
\begin{aligned}
l &= \text{ceil}\,(x)\,, \quad a = x - l\,, \\
k &= \text{ceil}\,(y)\,, \quad b = y - k\,.
\end{aligned}
$$

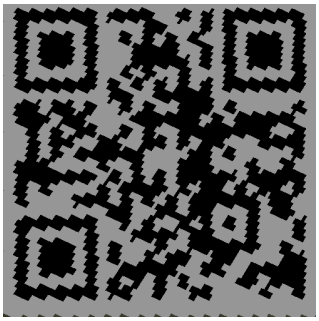
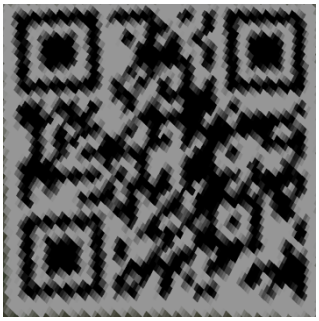


https://stackoverflow.com/questions/50086717/bilinear-interpolation-artifacts

Part 3. Unwarp the Secret (8%)

• Paste the 2 warped images and get the correct QR code link from both images.

Ans:

Source image	QR code	Link
 BL_secret1.png		http://media.ee.ntu.edu.tw /courses/cv/21S/
 BL_secret2.png		http://media.ee.ntu.edu.tw /courses/cv/21S/

• 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. (3+4%)

Ans:

兩個經由 warping 得到的影像結果並不相同：BL_secret1.png 大致沒有 distortion，因此 warped result 較為清晰；BL_secret2.png 則像是使用廣角鏡頭或魚眼鏡頭，因此產生類似 barrel distortion，使得影像中原本為直線的地方皆有扭曲，因此 warped results 較為模糊，且可以看出經由 QR code 邊界所形成的 envelope 水平線部分略為上凸。不過兩者生成的 QR code 皆可掃描並連到 2021 年春季的課程網頁。

影像產生 distortion 的原因通常與透鏡曲率或鏡頭組合瑕疵有關，並造成 warping 的結果產生較大誤差。常見的解決方式為藉由取得足夠 calibration pattern 的影像，計算出以及相機內參數 distortion 係數。

Part 4. Panorama (8%)

• Paste the stitched panorama. (1%)

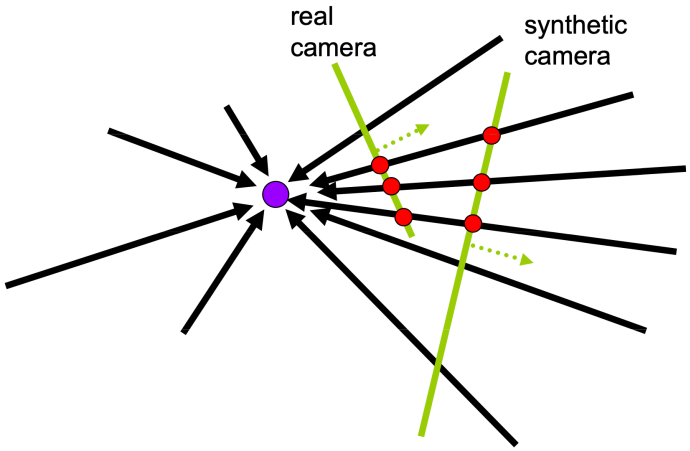
- Without blending.

- With blending.

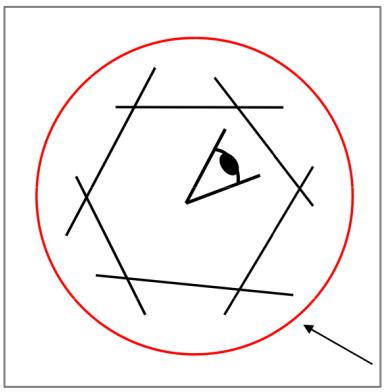
• Can all consecutive images be stitched into a panorama? If yes, explain the reason. If not, explain under what conditions will result in a failure? (3+4%)

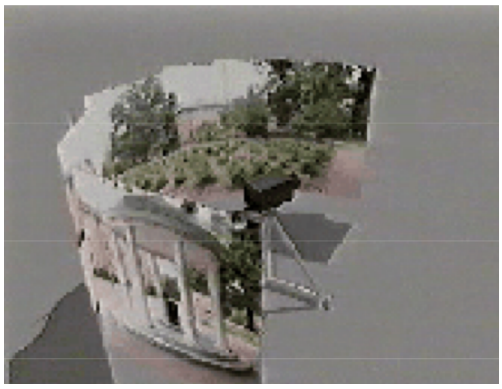
Ans:

不行。要形成 panorama（也就是 synthetic camera view），每個 projection 需要相同的中心，需要 planar scene 或是近似效果的 far-away scene，如下一頁第一張圖所示。也就是說，camera center 只能有 rotation 以及可忽略的 translation，即對應 homography matrix 的定義。

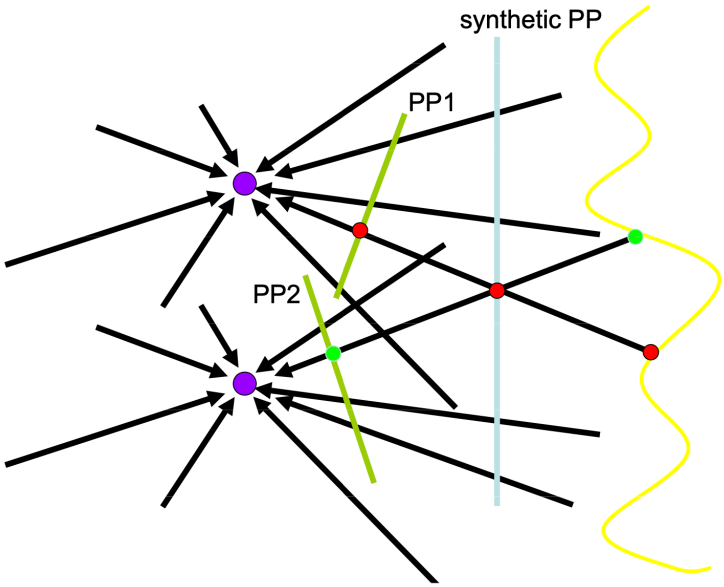


如果需要 360 度的 panorama，則可將每個 projection 視為投影至一個 cylinder 上，透過適當的 aligning 以及 blending 即可得到全景，如下方兩張圖所示。





Non-planar scene 之所以無法形成 panorama，是因為其對應的 epipolar line 和 epipolar 會形成交點，因此無法將 projection 都投影至連續的 synthetic view 上，如下圖所示。

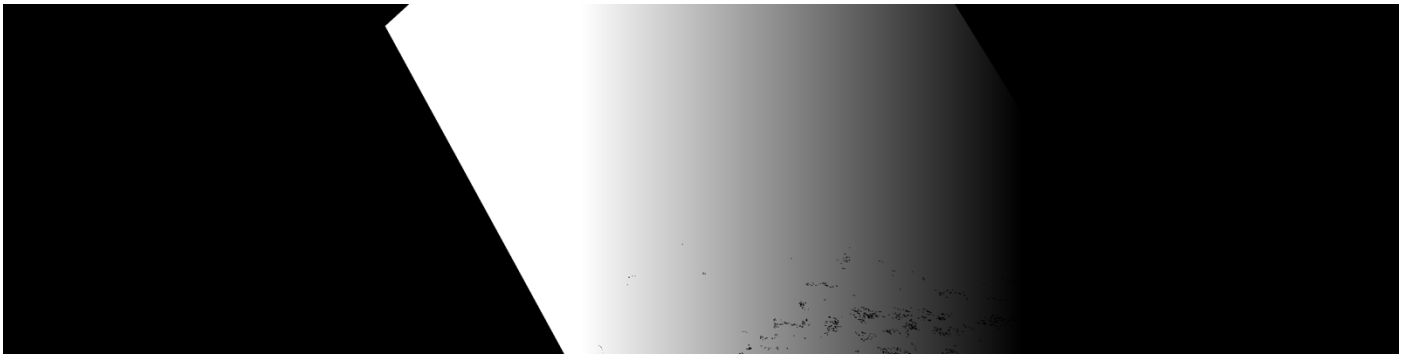


- Reference: Yung-Yu Chuang (2010). Image stitching in digital visual effects. https://www.csie.ntu.edu.tw/~cyy/courses/vfx/10spring/lectures/handouts/lec07_stitching.pdf

• [Bonus] Using blending techniques (e.g. alpha blending) (5%)

Ans:

我實作的方法為 alpha blending，成果如第 5 頁所示。主要流程為：1. 找出兩張 projection（左圖和右圖）重疊的像素座標。2. 將重疊部分水平分為四等份，最左側的等份使用左圖，最右側的等份使用右圖，中間的部分則是算出相對左圖由 1 逐漸遞減至 0 的 alpha 值，如下圖 mask 所示。3. 最後，重疊部分，左圖以 alpha 值、右圖以(1-alpha)值相疊加。程式碼如下方所示。



Frame 2 與 Frame 3 重疊部分所形成的 alpha mask

alpha_blend(img1, img2)

```python
def alpha_blend(img1, img2):
    """
        Alpha blending for two images
        :param img1:    image 1
        :param img2:    image 2
        :return:       blended image
    """
    assert img1.shape == img2.shape

    # find the overlapping region and the corresponding coordinates
    overlap = np.logical_and(np.sum(img1, axis=2) != 0, np.sum(img2, axis=2) != 0).astype(np.uint8)
    x_min = np.min(np.where(overlap == 1)[1])
    x_max = np.max(np.where(overlap == 1)[1])
    x_min += (x_max - x_min) // 4   # only blend the middle part
    x_max -= (x_max - x_min) // 4

    # create mask for alpha blending
    mask = (np.copy(overlap)).astype(np.float32)    # (H, W)
    mask[:, :x_min] *= 1.0  # left part
    mask[:, x_min:x_max] *= (1.0 - np.arange(x_max - x_min) / (x_max - x_min))  # middle part
    mask[:, x_max:] *= 0.0  # right part
    mask = np.stack([mask] * 3, axis=2) # (H, W, 3)

    # alpha blending
    blend = np.zeros_like(img1)
    blend[overlap == 1] = img1[overlap == 1] * mask[overlap == 1] \
                          + img2[overlap == 1] * (1 - mask[overlap == 1])
    blend[overlap == 0] = img1[overlap == 0] + img2[overlap == 0]
    return blend
```