Computer Vision HW3 Report

B09901075 陳駿瑋

Part 1: Homography estimation

Part 2: Marker-Based Planar AR

1. ***solve_homography(u, v)***

```python
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
    A_for_vx = np.concatenate((u, np.ones((N, 1)), np.zeros((N, 3)),
                               -1 * np.multiply(u[:, 0], v[:, 0]).reshape((N, 1)), # -ux * vx
                               -1 * np.multiply(u[:, 1], v[:, 0]).reshape((N, 1)), # -uy * vx
                               -1 * v[:, 0].reshape((N, 1))), axis=1) # -vx
    A_for_vy = np.concatenate((np.zeros((N, 3)), u, np.ones((N, 1)),
                               -1 * np.multiply(u[:, 0], v[:, 1]).reshape((N, 1)), # -ux * vy
                               -1 * np.multiply(u[:, 1], v[:, 1]).reshape((N, 1)), # -uy * vy
                               -1 * v[:, 1].reshape((N, 1))), axis=1) # -vy
    A = np.concatenate((A_for_vx, A_for_vy), axis=0)

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

    return H
```

2. *warping()*

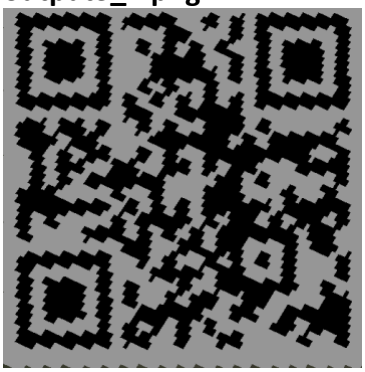
- The interpolation method I use is “nearest neighbor” (np.rint() in the code).

Code:

```
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(range(xmin, xmax), range(ymin, ymax))

# TODO: 2.reshape the destination pixels as N x 3 homogeneous coordinate
N = (xmax - xmin) * (ymax - ymin)
U = np.concatenate((x.reshape((1, N)), y.reshape((1, N)), np.ones((1, N))), axis=0)

if direction == 'b':
    # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
    V = np.dot(H_inv, U)
    V = np.divide(V, V[-1, :])
    Vx = np.rint(V[0, :].reshape(((ymax-ymin),(xmax-xmin))))
    Vy = np.rint(V[1, :].reshape(((ymax-ymin),(xmax-xmin))))
    # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)
    x_mask = np.bitwise_and((Vx >= 0), (Vx < w_src))
    y_mask = np.bitwise_and((Vy >= 0), (Vy < h_src))
    mask = np.bitwise_and(x_mask, y_mask)
    # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
    # TODO: 6. assign to destination image with proper masking
    valid_Vx = Vx[mask].astype(int)
    valid_Vy = Vy[mask].astype(int)
    dst[y[mask], x[mask]] = src[valid_Vy, valid_Vx]
    # pass

elif direction == 'f':
    # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
    V = np.dot(H, U)
    V = np.divide(V, V[-1, :])
    Vx = np.rint(V[0, :].reshape(((ymax-ymin),(xmax-xmin))))
    Vy = np.rint(V[1, :].reshape(((ymax-ymin),(xmax-xmin))))
    # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
    x_mask = np.bitwise_and((Vx >= 0), (Vx < w_dst))
    y_mask = np.bitwise_and((Vy >= 0), (Vy < h_dst))
    mask = np.bitwise_and(x_mask, y_mask)
    # TODO: 5.filter the valid coordinates using previous obtained mask
    # TODO: 6. assign to destination image using advanced array indicing
    valid_Vx = Vx[mask].astype(int)
    valid_Vy = Vy[mask].astype(int)
    dst[valid_Vy, valid_Vx, :] = src[mask]
    # pass

return dst
```

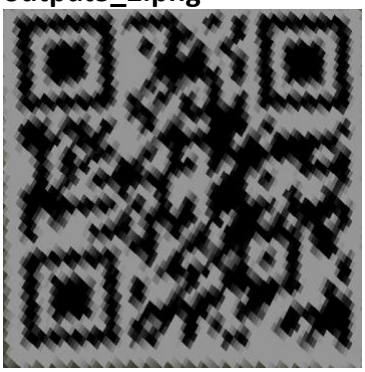
Part 3: Unwarp the secret

1. **output3_1.png**



http://media.ee.ntu.edu.tw/courses/cv/21S/

2. **output3_2.png**



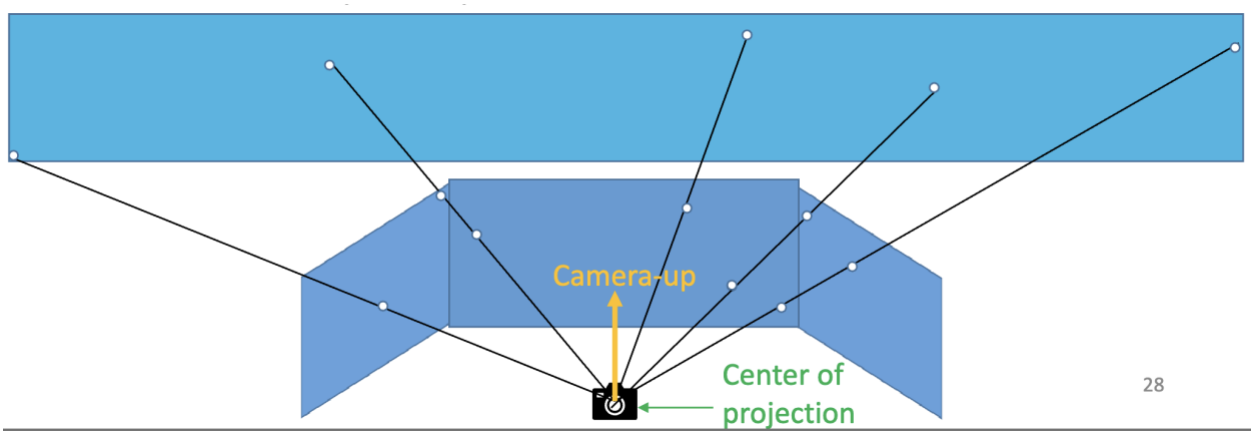
http://media.ee.ntu.edu.tw/courses/cv/21S/

兩張 source images 產生的結果中，第一張比第二張清楚，第二張明顯比較模糊，但兩者都能掃出來同樣的網址。比較原圖，可以觀察到第一張的 QR Code 還算方正，看起來像單純旋轉的正方形，不過因為 pixel 被轉成斜的，所以轉回來的圖沒法像正常的 QR Code 一樣都是直線；第二張原圖可以看出 QR Code 的形狀有變，被拉成接近長方形，邊界也有點弧度，要轉回正方形的 QR Code 就可能會相較困難，pixels 要找對應位置會比較容易偏移，在黑灰邊界就容易有模糊現象。
不過雖然第二張比較不清楚，但黑灰分別的圖案還是有保留下來，所以即使邊界模糊，還是能掃出一樣的網址。

Part 4: Panorama

Can all consecutive images be stitched into a panorama?

不行。以這次作業使用的方式，投射到平面，那圖片的視角不能旋轉超過 180 度。
從下圖（擷取自作業投影片）中可看到，假設圖片上方為 0 度，那正負 90 度的地方就會投影到無限遠處，超過這總共 180 度的範圍，就會往圖片下方投影，無法與圖片上的點共平面。



實際比較例子可看作業得到的 output4.png 和下圖。output4.png 中視角在範圍內，可以看到除了接面處因為不同圖片亮度不同，所以看得出接線外，圖形都有連在一起，形成一張不錯的全景圖。另外下面前四張圖片是素材，第五張是全景圖結果，可以看到如果正常接起來，最右邊應該會有鏡子、架子，但或許因為視角差太大，導致無法完整產生全景圖。