Homework #3

May 28, 2021

Part 1

1. Paste the function solve_homography()

```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 = np.zeros((2*N, 8))
    for i in range(N):
        A[2*i, :] = np.array([u[i, 0], u[i, 1], 1, 0, 0, 0, -u[i, 0]*v[i,0], -u[i, 1]*v[i, 0]])
        A[2*i+1, :] = np.array([0, 0, 0, u[i, 0], u[i, 1], 1, -u[i, 0]*v[i, 1], -u[i, 1]*v[i, 1]])

    # TODO: 2.solve H with A
    b = v.reshape(-1)
    H, res, _, _ = np.linalg.lstsq(A, b, rcond=None)
    H = np.concatenate((H, np.array([1])))
    H = H.reshape(3,3)

    return H
```

2. Paste your warped canvas

Part 2

1. Paste the function code warping()

```
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 = np.linspace(xmin, xmax-1, xmax-xmin)
    y = np.linspace(ymin, ymax-1, ymax-ymin)
    x, y = np.meshgrid(x, y)
    x = x.reshape(-1).astype(int)
    y = y.reshape(-1).astype(int)
    u = np.vstack((x, y, np.ones(len(x))))

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

    if direction == 'b':
        # TODO: 3.apply H_inv to the destination pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        H_inv = np.linalg.inv(H)
        v = H_inv @ u
        vx = np.round(v[0] / v[2]).astype(int)
        vy = np.round(v[1] / v[2]).astype(int)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of source image)
        mask = (vx >= 0) & (vx < w_src) & (vy >= 0) & (vy < h_src)
```

```python

        # TODO: 5.sample the source image with the masked and reshaped transformed coordinates
        x = x[mask]
        y = y[mask]
        vx = vx[mask]
        vy = vy[mask]

        # TODO: 6. assign to destination image with proper masking
        dst[y, x] = src[vy, vx]

    elif direction == 'f':
        # TODO: 3.apply H to the source pixels and retrieve (u,v) pixels, then reshape to (ymax-ymin),(xmax-xmin)
        v = H @ u
        vx = np.round(v[0] / v[2]).astype(int)
        vy = np.round(v[1] / v[2]).astype(int)

        # TODO: 4.calculate the mask of the transformed coordinate (should not exceed the boundaries of destination image)
        mask = (vx >= 0) & (vx < w_dst) & (vy >= 0) & (vy < h_dst)

        # TODO: 5.filter the valid coordinates using previous obtained mask
        x = x[mask]
        y = y[mask]
        vx = vx[mask]
        vy = vy[mask]

        # TODO: 6. assign to destination image using advanced array indicing
        dst[vy, vx] = src[y, x]

    return dst
```

2. introduce the interpolation method

直接對 position 使用四捨五入取值，沒有做內插法。

Part 3

1. Paste the 2 warped QR code and the link you find

(a) secret1

(b) secret2

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

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

兩個照片投影方式不一樣，一個 Perspective，一個是 stereographic。兩個 warped result 一樣，認為是因為 QR code 貼在兩張圖片用的方法是一樣的，都是從角落四個點用 DLT 推算貼在圖片上的位置，所以可以找到 H inverse 使得兩張圖片的 QR code 可以還原成同一個。

Part 4

1. Paste your stitched panorama

2. Can all consecutive images be stitched into a panorama?

不一定可以，要能夠找到對應足夠個特徵點，以 homography 為例，需要 4 個才可以。

[Bonus]

Using homography to produce a “more than 2 images panorama”