ECE 661: Homework 3

Xingguang Zhang

(Fall 2020)

Task 1

1. Point to point correspondences

The projective transformation H from the source coordinates in homogeneous representational space $X = (x, y, 1)$ to target coordinates $Y = (u, v, 1)$ is:

$$Y = HX$$

H is a non-singular matrix and can be represented as:

$$H = \begin{bmatrix} a_{11} & a_{12} & t_1 \\ a_{21} & a_{22} & t_2 \\ v_1 & v_2 & 1 \end{bmatrix}$$

Thus, for each pair of X and Y, we have:

$$\begin{bmatrix} a_{11} & a_{12} & t_1 \\ a_{21} & a_{22} & t_2 \\ v_1 & v_2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

We can form 3 functions by this, they are:

$$\begin{aligned} a_{11}x + a_{12}y + t_1 &= u \\ a_{21}x + a_{22}y + t_2 &= v \\ v_1x + v_2y + 1 &= 1 \end{aligned}$$

Or,

$$\begin{aligned} a_{11}x + a_{12}y + t_1 &= v_1xu + v_2yu + u \\ a_{21}x + a_{22}y + t_2 &= v_1xv + v_2yv + v \end{aligned}$$

Then we have the equation that can be easily combined with equations from other pairs of points:

$$\begin{aligned} a_{11}x + a_{12}y + t_1 \qquad\qquad\qquad - v_1xu - v_2yu &= u \\ a_{21}x + a_{22}y + t_2 - v_1xv - v_2yv &= v \end{aligned}$$

It can be represented by matrix multiplication for coordinate pairs (x_i, y_i) and (u_i, v_i):

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_iu_i & -y_iu_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_iv_i & -y_iv_i \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ t_1 \\ a_{21} \\ a_{22} \\ t_2 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \end{bmatrix}$$

To solve the 8 unknown parameters, we need four pairs of associated points, the equations can be represented by:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -y_3u_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3v_3 & -y_3v_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4v_4 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ t_1 \\ a_{21} \\ a_{22} \\ t_2 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix}$$

Define

$$T = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -y_3u_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3v_3 & -y_3v_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4v_4 \end{bmatrix} \quad P = \begin{bmatrix} a_{11} \\ a_{12} \\ t_1 \\ a_{21} \\ a_{22} \\ t_2 \\ v_1 \\ v_2 \end{bmatrix} \quad C = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{bmatrix}$$

We can solve P by

$$P = T^{-1}C$$

Once we get P, we can get H, actually the point-to-point correspondence method is the same as the method in homework2, undistorted images are set as the same size as the original image (height, width are defined as the height and width of the original image). To achieve this, we first compute the homographic transform matrix $\mathbf{A}_1$ which transform the annotated 4 points $PQSR$ to (0,0), (0, width), (height, 0), and (height, width). Afterwards, we compute the corresponding points of (0,0), (0, width), (height, 0), and (height, width) of the original image using $\mathbf{A}_1$, the four points will define an irregular convex quadrilateral. We then use another similarity transform $\mathbf{A}_2$ to project the quadrilateral into a maximal area which have the same size as the original image. The final projection matrix will be $\mathbf{A} = \mathbf{A}_1\mathbf{A}_2$.
The results are shown below (PQSR are shown as the vertices of the green bounding boxes):

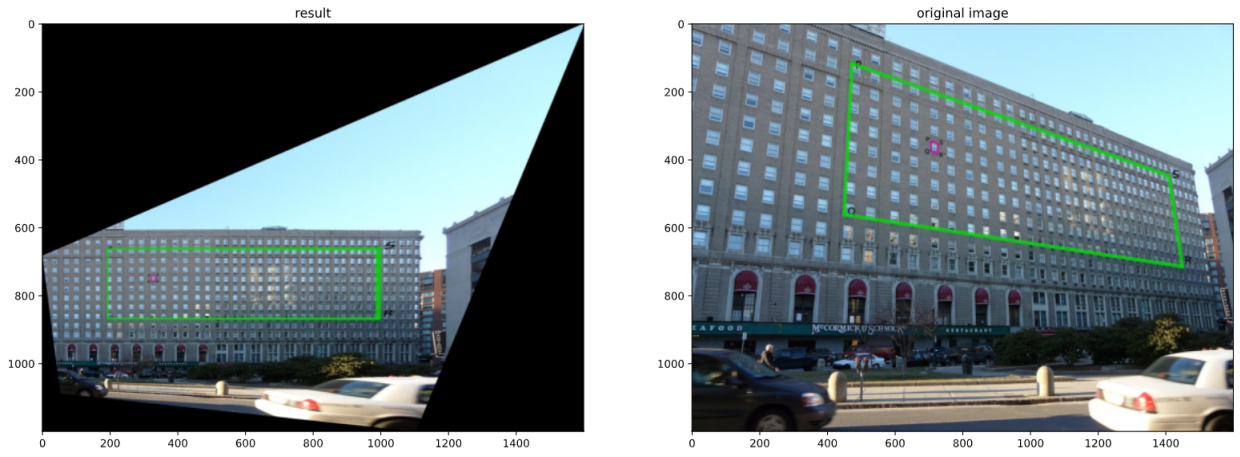

Figure 1: The undistorted image and the original image1

Figure 2: The undistorted image and the original image2

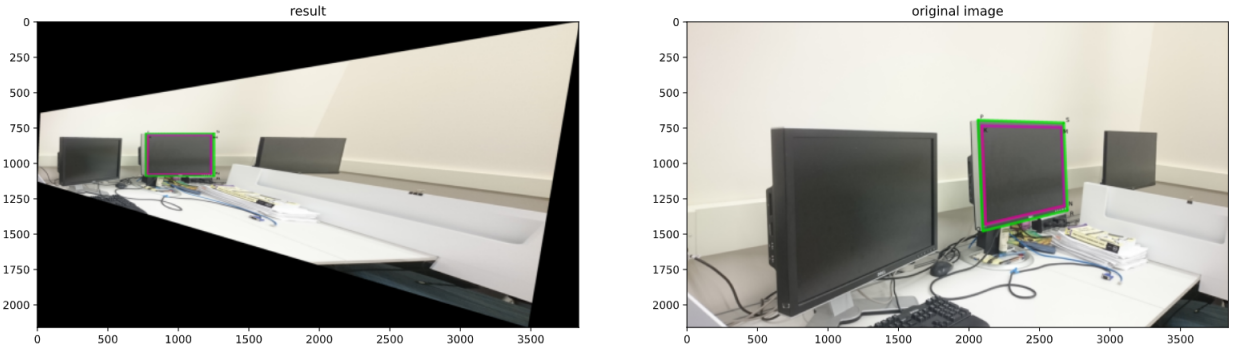

Figure 3: The undistorted image and the original image3

2. Two-step method

Vanishing line method for removing the projective distortion

First we find two pairs of parallel lines $\mathbf{l}_1$, $\mathbf{l}_2$ and $\mathbf{m}_1$, $\mathbf{m}_2$ in the world coordinates and $\mathbf{l}'_1$, $\mathbf{l}'_2$ and $\mathbf{m}'_1$, $\mathbf{m}'_2$ on the distorted image plane, by points $PQSR$ under representational space:

$$\begin{aligned}\mathbf{l}'_1 &= P \times Q \\ \mathbf{l}'_2 &= S \times R \\ \mathbf{m}'_1 &= P \times S \\ \mathbf{m}'_2 &= R \times Q\end{aligned}$$

Then we can find two vanishing points VP_1 and VP_2 on the image plane by the distorted parallel lines:

$$VP_1 = \mathbf{l}_1 \times \mathbf{l}_2 \quad VP_2 = \mathbf{m}_1 \times \mathbf{m}_2$$

The vanishing line defined by two vanishing points are:

$$VL \equiv (l_1, l_2, l_3) = VP_1 \times VP_2$$

The projective transform matrix will be given by:

$$A_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

Removing affine distortion by dual degenerate conic $\mathbf{C}^*_\infty$

After removing the projective distortion, we need to remove the affine distortion. If we have two intersected lines $\mathbf{l} = (l_1, l_2, l_3)$ and $\mathbf{m} = (m_1, m_2, m_3)$, they have angle θ, the cosine of which can be calculated by:

$$cos\theta = \frac{l_1 m_1 + l_2 m_2}{\sqrt{(l_1^2 + l_2^2)(m_1^2 + m_2^2)}} = \frac{\mathbf{l}^T \mathbf{C}^*_\infty \mathbf{m}}{\sqrt{(\mathbf{l}^T \mathbf{C}^*_\infty \mathbf{l})(\mathbf{m}^T \mathbf{C}^*_\infty \mathbf{m})}}$$

where

$$\mathbf{C}^*_\infty = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

If the $\mathbf{l}$ and $\mathbf{m}$ are chosen perpendicular to each other in the undistorted image, we have $cos\theta = 0$, which means that $\mathbf{l}^T \mathbf{C}^*_\infty \mathbf{m} = 0$.
Assume the affine transform from the **given image space** to the **undistorted image space** is $\mathbf{H}_a^{-1}$, the associated lines of $\mathbf{l}$ and $\mathbf{m}$ on the given image space are $\mathbf{l} = \mathbf{H}_a^T \mathbf{l}'$ and $\mathbf{m} = \mathbf{H}_a^T \mathbf{m}'$, so that we have

$$\mathbf{l'}^T \mathbf{H}_a \mathbf{C}^*_\infty \mathbf{H}_a^T \mathbf{m}' = 0 \tag{1}$$

Expanding it we get:

$$\begin{pmatrix} l'_1 & l'_2 & l'_3 \end{pmatrix} \begin{bmatrix} \mathbf{A} & \vec{\mathbf{0}} \\ \vec{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \vec{\mathbf{0}} \\ \vec{\mathbf{0}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A}^T & \vec{\mathbf{0}}^T \\ \vec{\mathbf{0}}^T & 1 \end{bmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} = \begin{pmatrix} l'_1 & l'_2 & l'_3 \end{pmatrix} \begin{bmatrix} \mathbf{A}\mathbf{A}^T & \vec{\mathbf{0}} \\ \vec{\mathbf{0}} & 1 \end{bmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} = 0$$

Assume $\mathbf{S} = \mathbf{A}\mathbf{A}^T = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix}$, we have:

$$\begin{pmatrix} l'_1 & l'_2 \end{pmatrix} \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \begin{pmatrix} m'_1 \\ m'_2 \end{pmatrix} = 0$$

Since $\mathbf{S}$ is symmetric, $s_{12} = s_{21}$, the following equation for $\mathbf{S}$ has 3 unknowns s_{11}, s_{21}, s_{22}:

$$s_{11} l'_1 m'_1 + s_{12}(l'_1 m'_2 + l'_2 m'_1) + s_{22} l'_2 m'_2 = 0$$

We don't need 3 equations to solve this function, because only ratio matters, we can set $s_{22} = 1$, so we just need 2 pairs of orthogonal lines $\mathbf{l'}_{PQ}$, $\mathbf{m'}_{QR}$ and $\mathbf{l'}_{RS}$, $\mathbf{m}_{PS}$, and the equation to solve $\mathbf{S}$ is:

$$\begin{bmatrix} l'_{PQ1} m'_{QR1} & l'_{PQ1} m'_{QR2} + l'_{PQ2} m'_{QR1} \\ l'_{RS1} m'_{PS1} & l'_{RS1} m'_{PS2} + l'_{RS2} m'_{PS1} \end{bmatrix} \begin{bmatrix} s_{11} \\ s_{12} \end{bmatrix} = \begin{bmatrix} l'_{PQ2} m'_{QR2} \\ l'_{RS2} m'_{PS2} \end{bmatrix}$$

When we get $\mathbf{S}$, we can use singular value decomposition(SVD) to find $\mathbf{A}$. Since $\mathbf{S}$ is positive semi-definite, $\mathbf{S} = \mathbf{V}\mathbf{D}\mathbf{V}^T$, where $\mathbf{V}$ is orthonormal, we finally get $\mathbf{A}$ by

$$\mathbf{A} = \mathbf{V}\mathbf{D}^{\frac{1}{2}}\mathbf{V}^T$$

After we get $\mathbf{A}$, the affine transform matrix is

$$\mathbf{H}_a = \begin{bmatrix} \mathbf{A} & \vec{\mathbf{0}} \\ \vec{\mathbf{0}} & 1 \end{bmatrix}$$

Combining the projective transform and affine transform, the two step method restore the original image plane by transform:

$$\mathbf{H} = \mathbf{H}_a^{-1}\mathbf{A}_p$$

In practice, the result is sensitive to point selection, I used two sets of 4 points for each image to compute the projective and affine distortion respectively. The results obtained by the two-step method are shown below (green bounding box for projective transform and purple bounding box for affine):

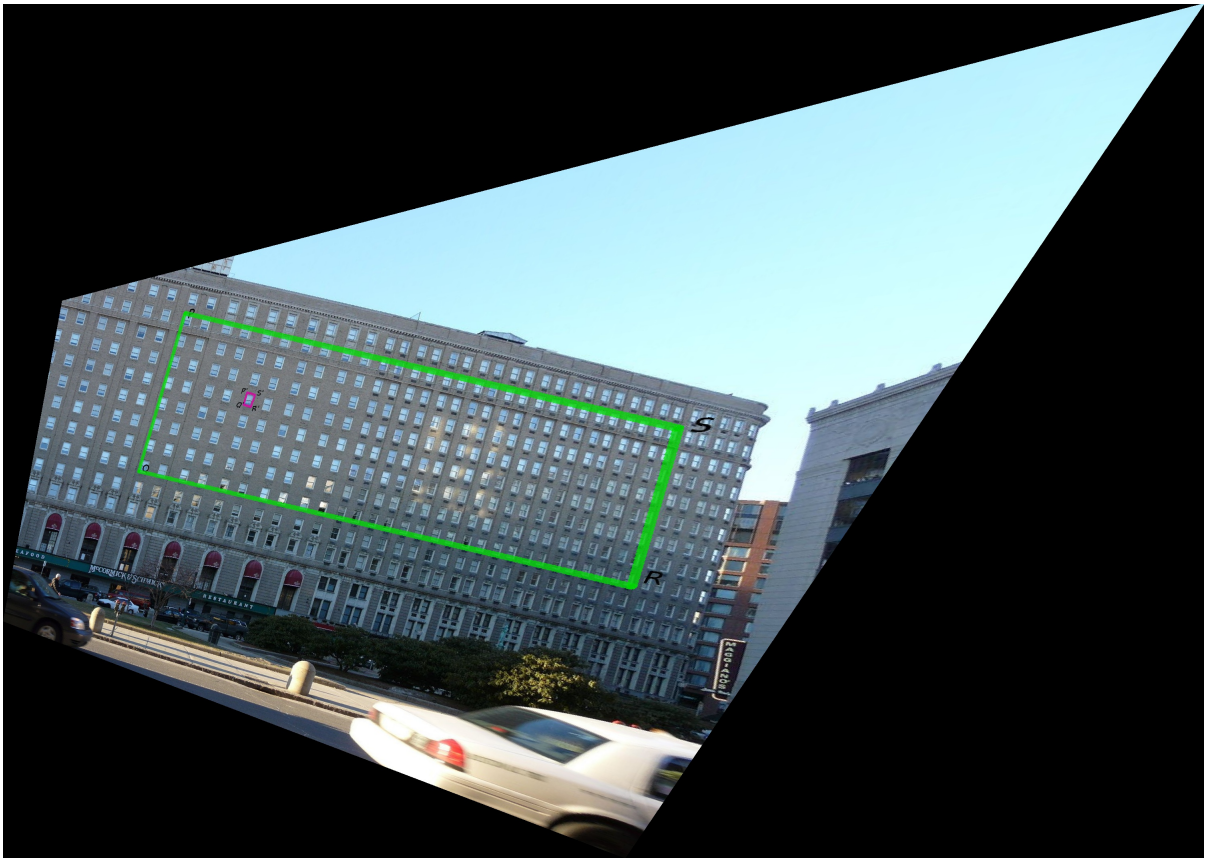

Figure 4: The undistorted image1 using the two-step method

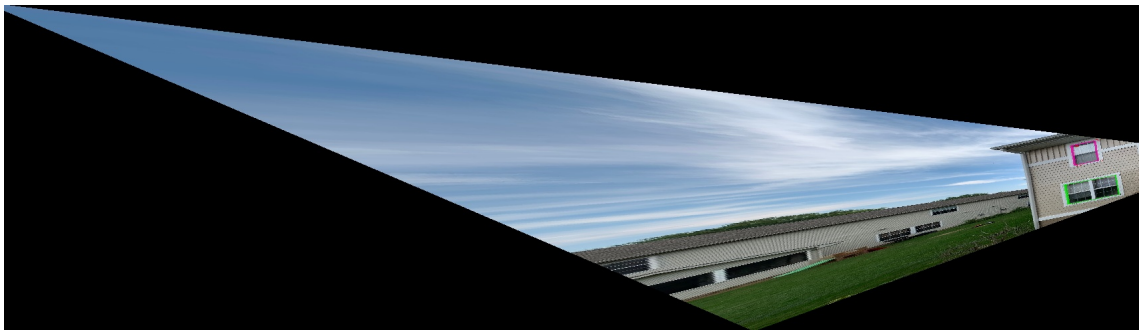

Figure 5: The undistorted image2 using the two-step method

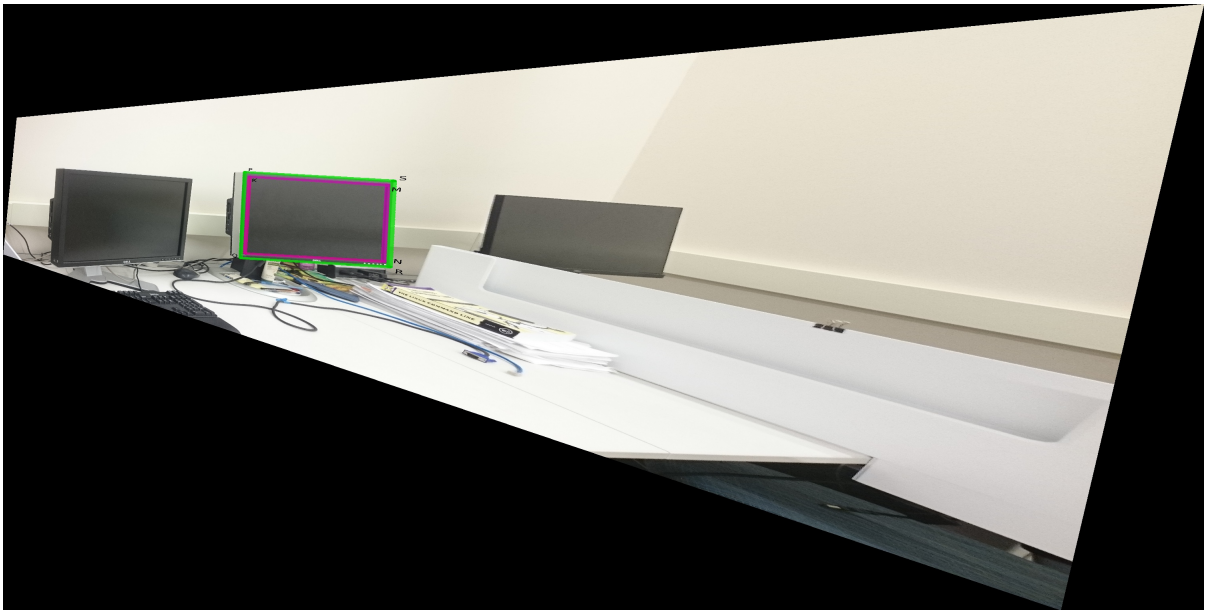

Figure 6: The undistorted image3 using the two-step method

2. One-step method

We directly combine the projective and affine transform together to the transform

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \vec{\mathbf{0}} \\ \vec{\mathbf{v}}^T & 1 \end{bmatrix}$$

For the one-step approach, let $\mathbf{C}_\infty^{*'}$ be a projection of the $\mathbf{C}_\infty^*$ and it's represented as

$$\mathbf{C}_\infty^{*'} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

Let $\mathbf{C}_\infty^{*'} = \mathbf{H}\mathbf{C}_\infty^*\mathbf{H}^T$, just as equation (1), we have

$$\mathbf{l}'^T\mathbf{C}_\infty^{*'}\mathbf{m}' = 0$$

That is,

$$\begin{pmatrix} l'_1 & l'_2 & l'_3 \end{pmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{pmatrix} m'_1 \\ m'_2 \\ m'_3 \end{pmatrix} = 0$$

Since only ratio matters, if we set $f = 1$, the equation is:

$$\begin{pmatrix} l'_1 m'_1 & \frac{l'_1 m'_2 + l'_2 m'_1}{2} & l'_2 m'_2 & \frac{l'_1 m'_3 + l'_3 m'_1}{2} & \frac{l'_2 m'_3 + l'_3 m'_2}{2} \end{pmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = -l'_3 m'_3$$

There are 5 unknowns, so we need at least 5 pairs of orthogonal lines to solve a, b, c, d, e. Once we find $\mathbf{C}_\infty^{*'}$, since

$$\mathbf{C}_\infty^{*'} = \begin{bmatrix} \mathbf{A} & \vec{\mathbf{0}} \\ \vec{\mathbf{v}}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \vec{\mathbf{0}} \\ \vec{\mathbf{0}}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A}^T & \vec{\mathbf{v}} \\ \vec{\mathbf{0}}^T & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A}\mathbf{A}^T & \mathbf{A}\vec{\mathbf{v}} \\ \vec{\mathbf{v}}^T \mathbf{A}^T & 1 \end{bmatrix} \tag{2}$$

Assume $\mathbf{S} = \mathbf{A}\mathbf{A}^T$, using the same SVD decomposition as two-step method, we can find $\mathbf{A}$, then by

$$\mathbf{A}\mathbf{v} = \begin{bmatrix} d/2 \\ e/2 \end{bmatrix} \tag{3}$$

we can solve $\mathbf{v}$, and finally get $\mathbf{H}$ Since we need at least 5 pairs of orthogonal lines, I used 4 from the green bounding box and other pair selected from the purple bounding box, the results using the one-step method is shown below:

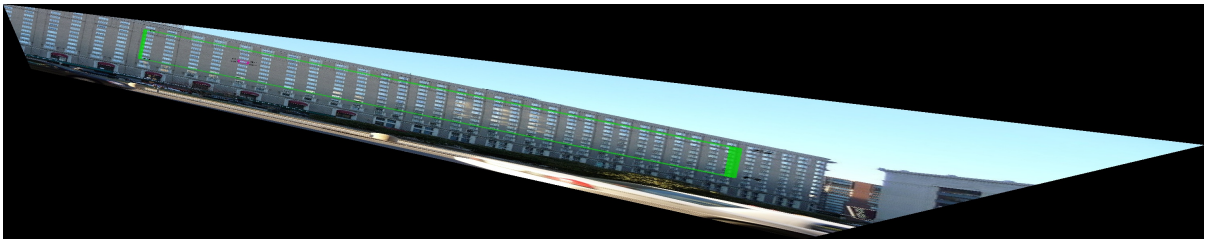

Figure 7: The undistorted image1 using the one-step method

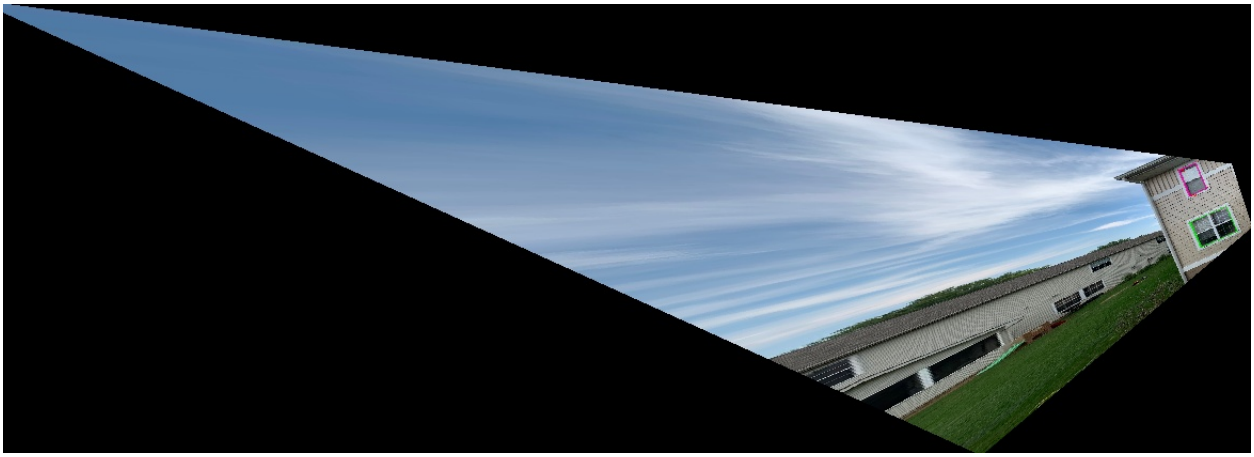

Figure 8: The undistorted image2 using the one-step method

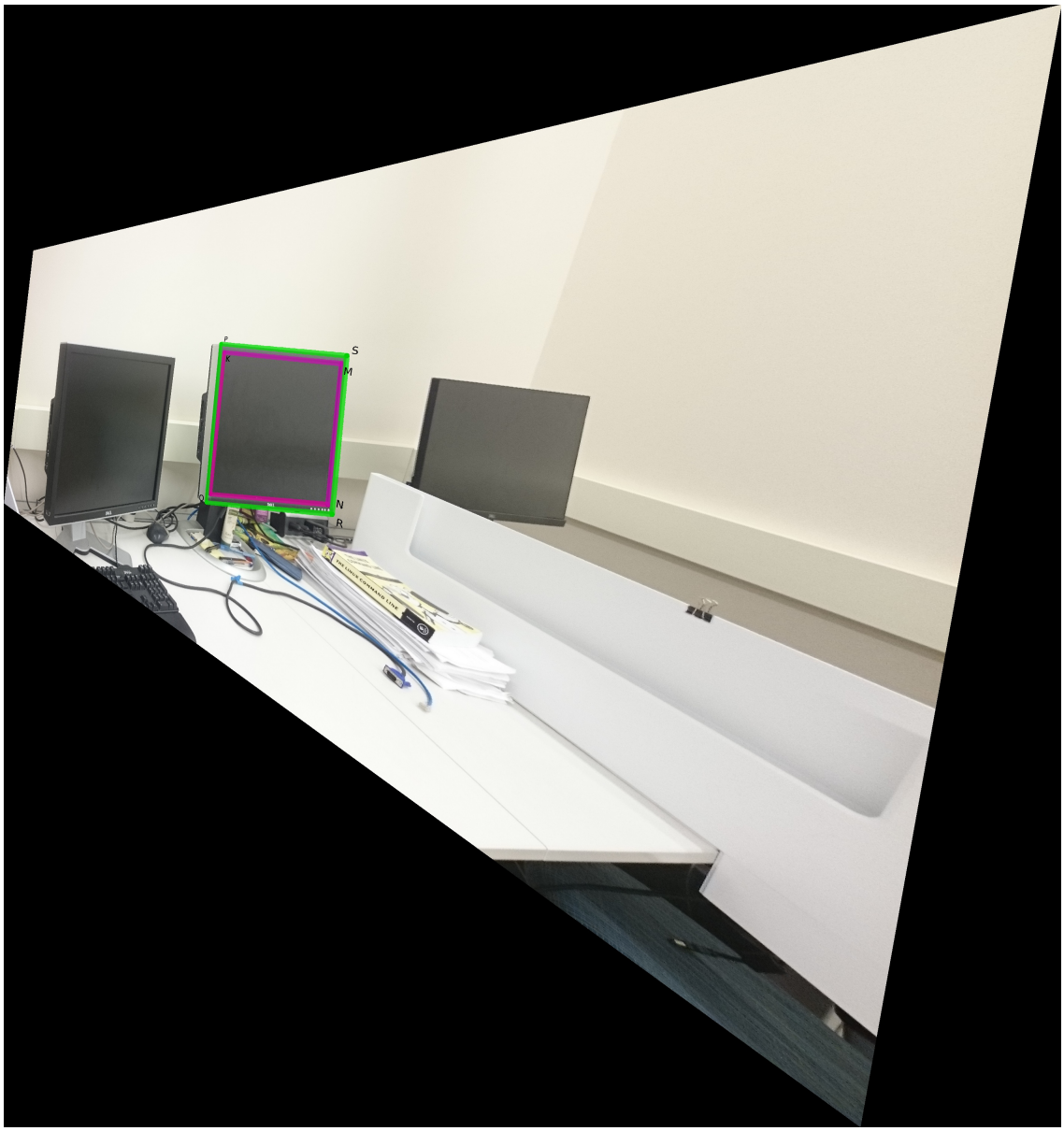

Figure 9: The undistorted image3 using the one-step method

3. Observations on the results obtained using the above three methods.

Among the 3 methods, the point-to-point correspondence method is the most robust and the results of which look the most natural. The robustness of the first method may come from the fixed output size, as we initially transform the 4 anchor points in the distorted image to a reasonable area. But for the two-step and one-step methods, sometimes the anchor points will be transformed to an extremely small area or area with an extreme length ratio.

Besides, I found that in the two-step method, choosing two sets of points could make the computation more stable. My assumption is that if we use only 4 points, removing the projective distorting will add a constraint to make $\mathbf{m}_1$ and $\mathbf{m}_2$ parallel, $\mathbf{l}_1$ and $\mathbf{l}_2$ parallel, when we compute the affine transform still from the same sets, the parallel relationship enforced by the first step will

make the two pairs of orthogonal collapsed to just 1 pair. Using different set of points (at least an additional pair of orthogonal lines) for removing affine distortion could avoid this ill-conditional scenario.

We always assume that the transform from the world coordinates to the image plane is linear, but actually the nonlinear distortion can't be neglected in some condition. With the field of view becomes larger (far from the central area of the image plane), the non-linearity affects more on our computation. In this homework, I noticed that if we select the points from the marginal area, for example if we select the first monitor in image 3, the results got from all 3 methods will look terrible more easily then the results computed from the central area.

Task 2

I tested another 3 images using the 3 methods above, the results are shown below:

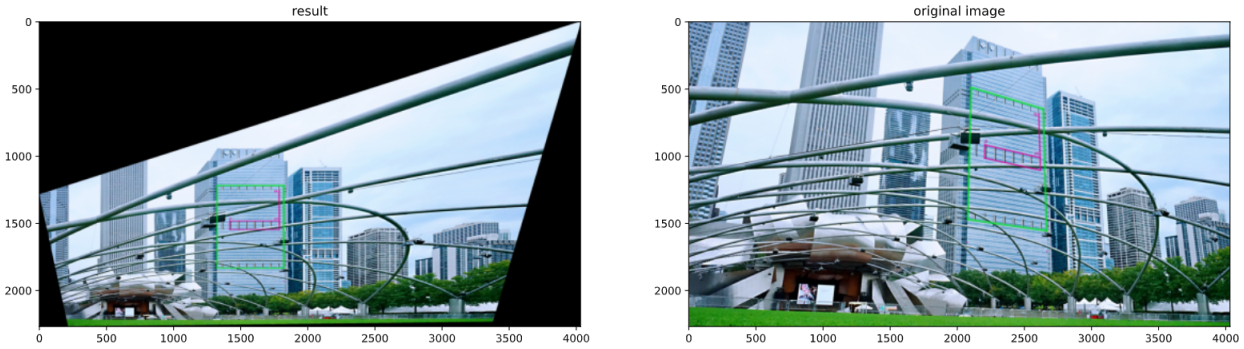

Figure 10: The undistorted image and the original image4

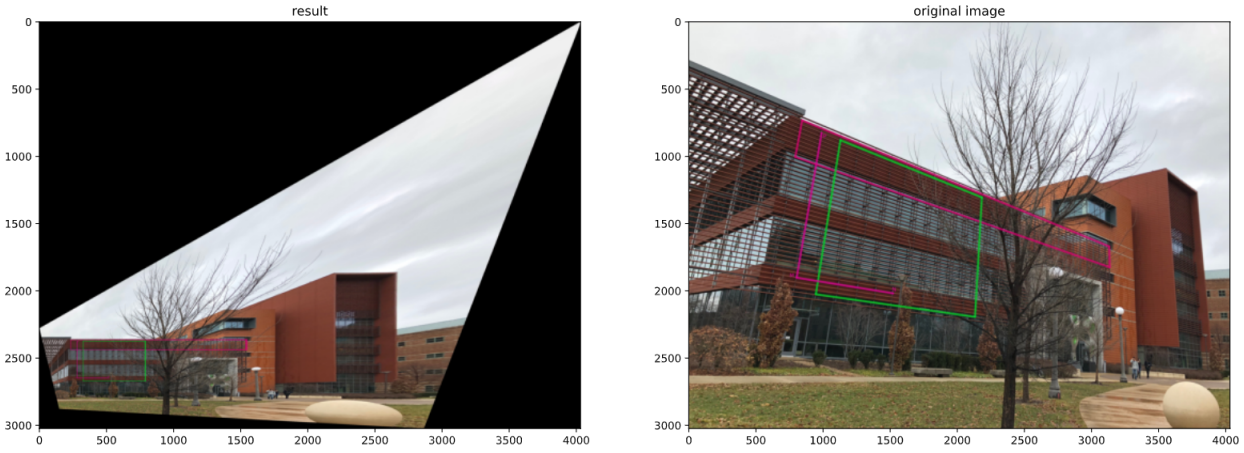

Figure 11: The undistorted image and the original image5

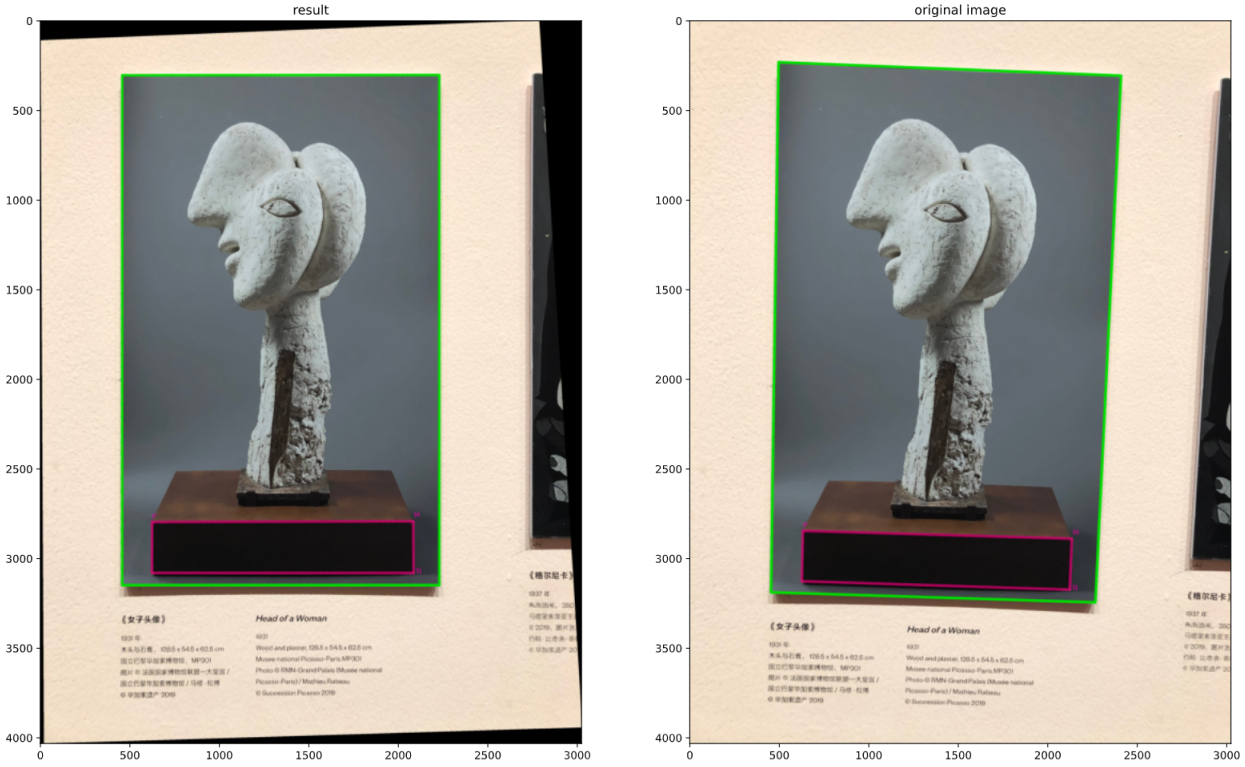

Figure 12: The undistorted image and the original image6

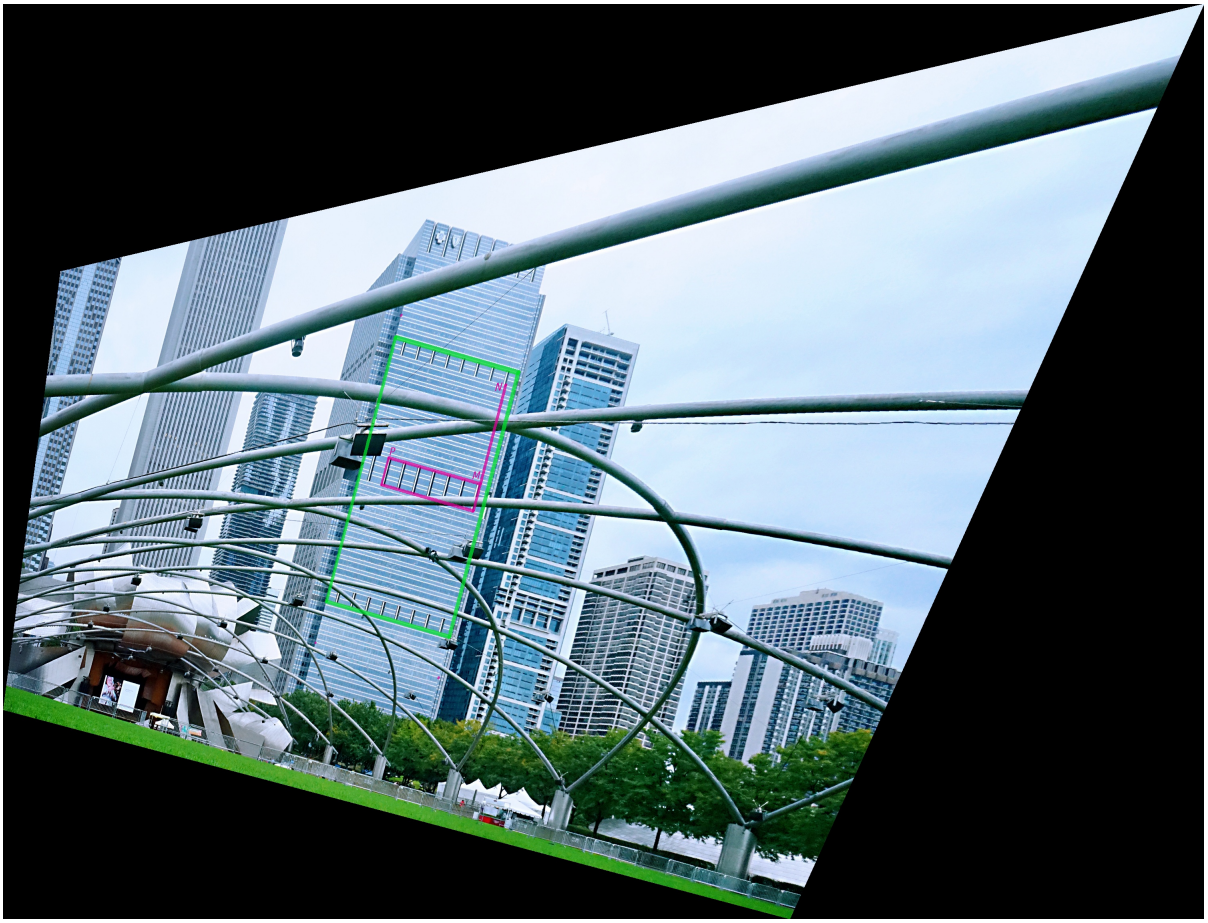

Figure 13: The undistorted image4 using the two-step method

Figure 14: The undistorted image5 using the two-step method

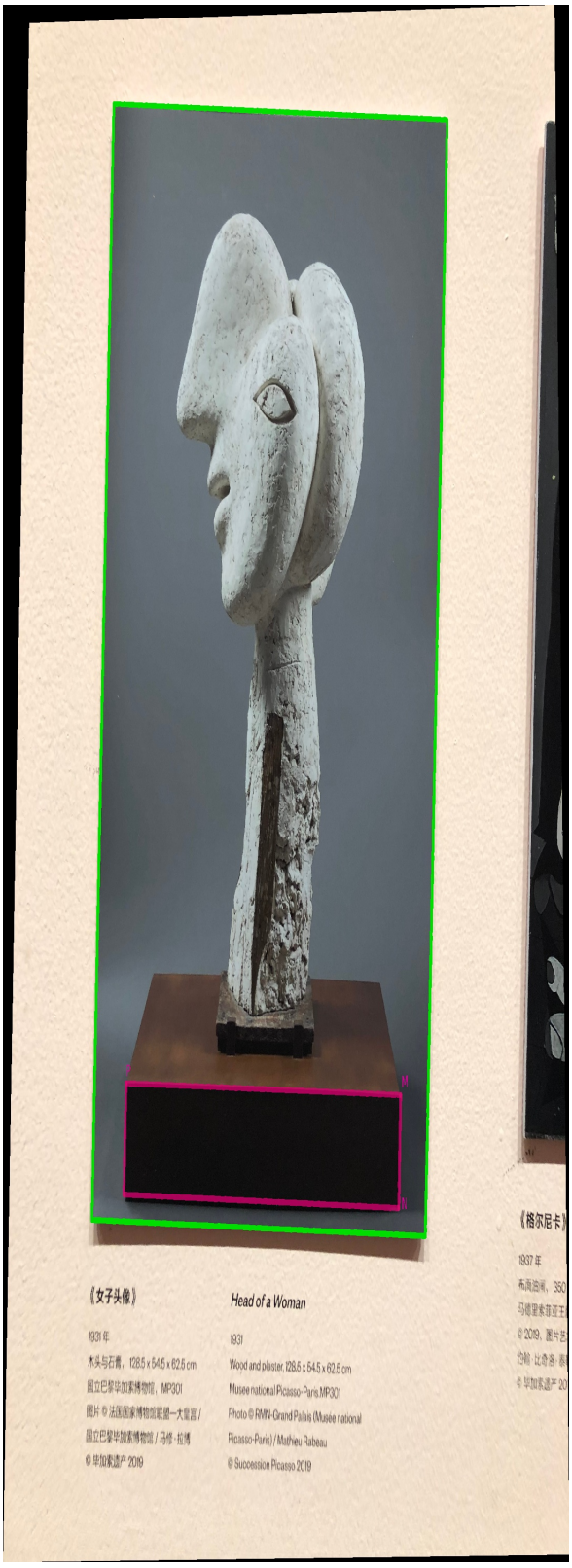

Figure 15: The undistorted image6 using the two-step method

Figure 16: The undistorted image4 using the one-step method

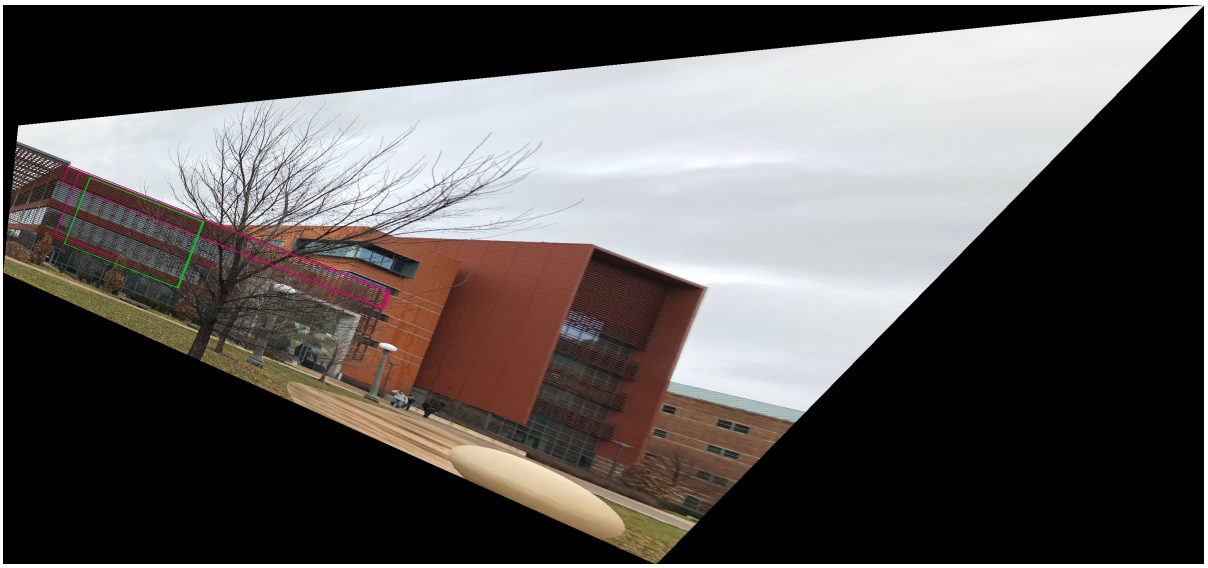

Figure 17: The undistorted image5 using the one-step method

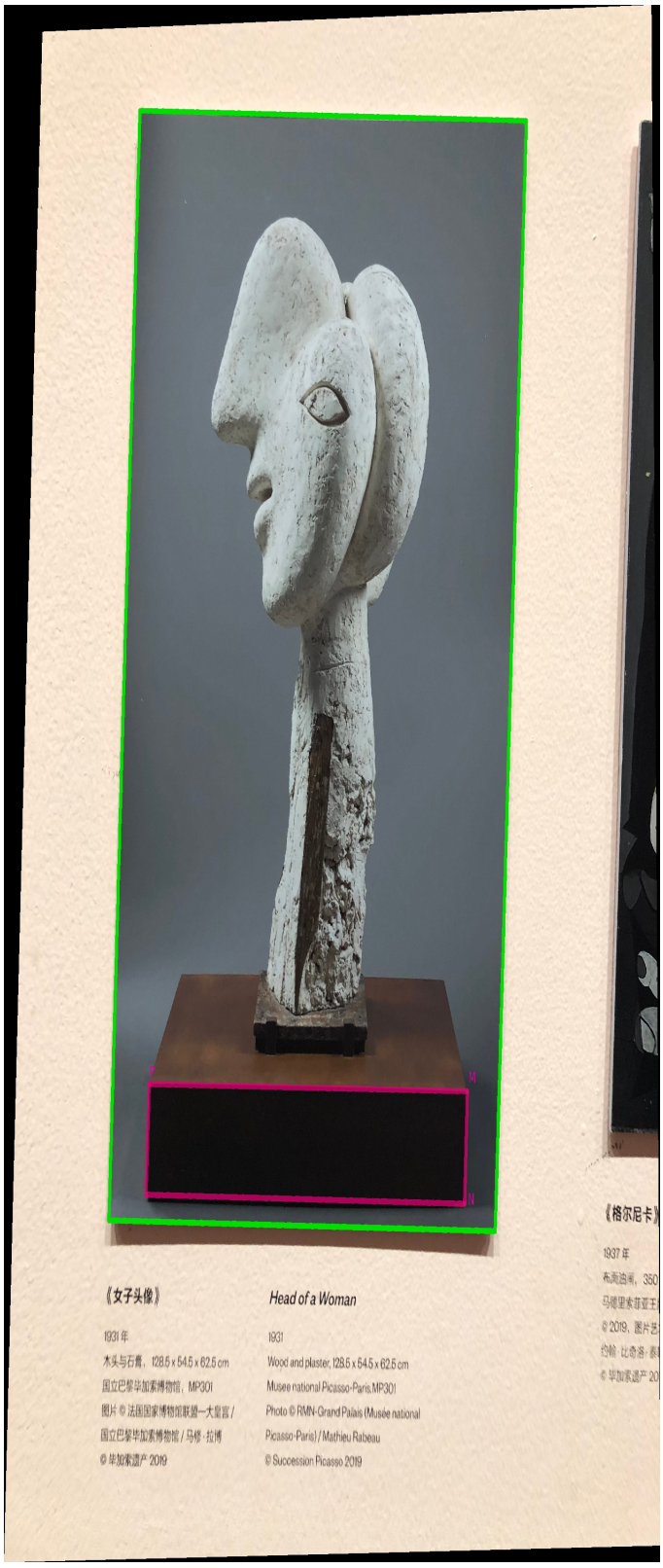

Figure 18: The undistorted image6 using the one-step method

Codes

point-to-point correspondences

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os

folder_task = '/home/xingguang/Documents/ECE661/hw3/hw3_Task1_Images/Images'
path_1 = os.path.join(folder_task, '1.jpeg')
path_2 = os.path.join(folder_task, '2.jpeg')
path_3 = os.path.join(folder_task, '3.jpeg')
outpath_1 = os.path.join(folder_task, 'ptop1.jpeg')
outpath_2 = os.path.join(folder_task, 'ptop2.jpeg')
outpath_3 = os.path.join(folder_task, 'ptop3.jpeg')

img_1 = cv2.cvtColor(cv2.imread(path_1), cv2.COLOR_BGR2RGB)
img_2 = cv2.cvtColor(cv2.imread(path_2), cv2.COLOR_BGR2RGB)
img_3 = cv2.cvtColor(cv2.imread(path_3), cv2.COLOR_BGR2RGB)

def choosePQRS(img=1):
    if img == 1:
        P = [472, 118]
        Q = [449, 562]
        S = [1406, 448]
        R = [1447, 715]
    if img == 2:
        P = [382, 575]
        Q = [379, 834]
        S = [593, 551]
        R = [606, 922]
    if img == 3:
        # P = [686, 744] #1st monitor
        # Q = [736, 2100]
        # S = [1772, 783]
        # R = [1775, 1616]
        P = [2062, 698] #2nd
        Q = [2096, 1479]
        S = [2668, 717]
        R = [2694, 1333]
    return np.asarray([P, Q, S, R])

def definePQRS(width, height):
    P = [0, 0]
    Q = [0, height]
    S = [width, 0]
    R = [width, height]
    return np.asarray([P, Q, S, R])

def findHomoproj(source, target):
    def F_unit(source_point, target_point):
        x, y = source_point[0], source_point[1]
        x_, y_ = target_point[0], target_point[1]
        return np.asarray([[x, y, 1, 0, 0, 0, -x*x_, -y*x_],
                           [0, 0, 0, x, y, 1, -x*y_, -y*y_]])
    F_list = [F_unit(source[i], target[i]) for i in range(4)]
    F = np.concatenate(F_list, axis=0)
```

```
    T_span = target.reshape((-1,1))
    H_param = np.matmul(np.linalg.inv(F), T_span)
    H = np.ones((9, 1))
    H[:8, :] = H_param
    return H.reshape((3, 3))

def newHomography(H, r, width, height):
    expanded = projTransform(np.linalg.inv(H), r)
    l, r = np.amin(expanded[:, 0]), np.amax(expanded[:, 0])
    t, b = np.amin(expanded[:, 1]), np.amax(expanded[:, 1])
    s_1 = (r - l) / width
    s_2 = (b - t) / height
    H_new = np.asarray([[s_1, 0, l], [0, s_2, t], [0, 0, 1]])
    return np.matmul(H, H_new)

def projTransform(H, source):
    nps = source.shape[0]
    source_rep = np.concatenate((source, np.ones((nps,1))), axis=1)
    t_homo = np.matmul(H, source_rep.T).T
    t_norm = t_homo / t_homo[:,2].reshape((nps,1))
    return t_norm[:,:2]

def projRange(source, H):
    out = np.zeros_like(source)
    height, width, _ = source.shape
    for i in range(out.shape[0]):
        for j in range(out.shape[1]):
            coor_source = np.asarray([[j, i]])
            coor_target = projTransform(H, coor_source).squeeze()
            h, w = int(coor_target[1]), int(coor_target[0])
            if 0 <= h < height and 0 <= w <width:
                out[i,j,:] = source[h, w,:]
    return out

range_1 = choosePQRS(1)
range_2 = choosePQRS(2)
range_3 = choosePQRS(3)
range_41 = definePQRS(img_1.shape[1], img_1.shape[0])
range_42 = definePQRS(img_2.shape[1], img_2.shape[0])
range_43 = definePQRS(img_3.shape[1], img_3.shape[0])
H_41 = findHomoproj(range_41, range_1)
H_42 = findHomoproj(range_42, range_2)
H_43 = findHomoproj(range_43, range_3)
H_14 = findHomoproj(range_1, range_41)
H_24 = findHomoproj(range_2, range_42)
H_34 = findHomoproj(range_3, range_43)

H_1 = newHomography(H_41, range_41, img_1.shape[1], img_1.shape[0])
H_2 = newHomography(H_42, range_42, img_2.shape[1], img_2.shape[0])
H_3 = newHomography(H_43, range_43, img_3.shape[1], img_3.shape[0])

result = projRange(img_1, H_1)
plt.figure(figsize=(20,40))
plt.subplot(121)
plt.title('result')
plt.imshow(result)
plt.subplot(122)
plt.title('original image')
plt.imshow(img_1)
```

```
result = projRange(img_2, H_2)
plt.figure(figsize=(20,40))
plt.subplot(121)
plt.title('result')
plt.imshow(result)
plt.subplot(122)
plt.title('original image')
plt.imshow(img_2)

result = projRange(img_3, H_3)
plt.figure(figsize=(20,40))
plt.subplot(121)
plt.title('result')
plt.imshow(result)
plt.subplot(122)
plt.title('original image')
plt.imshow(img_3)
```

Two step method

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os

folder_task = '/home/xingguang/Documents/ECE661/hw3/hw3_Task1_Images/Images'
path_1 = os.path.join(folder_task, '1.jpeg')
path_2 = os.path.join(folder_task, '2.jpeg')
path_3 = os.path.join(folder_task, '3.jpeg')
outpath_1 = os.path.join(folder_task, 'twostep1.jpeg')
outpath_2 = os.path.join(folder_task, 'twostep2.jpeg')
outpath_3 = os.path.join(folder_task, 'twostep3.jpeg')
outpathproj_1 = os.path.join(folder_task, 'twostepproj1.jpeg')
outpathproj_2 = os.path.join(folder_task, 'twostepproj2.jpeg')
outpathproj_3 = os.path.join(folder_task, 'twostepproj3.jpeg')
img_1 = cv2.cvtColor(cv2.imread(path_1), cv2.COLOR_BGR2RGB)
img_2 = cv2.cvtColor(cv2.imread(path_2), cv2.COLOR_BGR2RGB)
img_3 = cv2.cvtColor(cv2.imread(path_3), cv2.COLOR_BGR2RGB)

def choosePQRS(img=1):
    if img == 1:
        P = [472, 118]
        Q = [449, 562]
        S = [1406, 448]
        R = [1447, 715]

        P_ = [705, 346]
        Q_ = [705, 376]
        S_ = [726, 353]
        R_ = [725, 383]


    if img == 2:
        P = [382, 576]
        Q = [379, 834]
        S = [593, 551]
        R = [606, 922]
```

```
        P_ = [427, 118]
        Q_ = [425, 346]
        S_ = [511, 40]
        R_ = [514, 298]

    if img == 3:
        P = [2062, 698] #2nd
        Q = [2096, 1479]
        S = [2668, 717]
        R = [2694, 1333]

        P_ = [2089, 738]
        Q_ = [2115, 1434]
        S_ = [2652, 749]
        R_ = [2674, 1301]
    return np.asarray([P, Q, S, R]), np.asarray([P_, Q_, S_, R_])

def definePQRS(width, height):
    P = [0, 0]
    Q = [0, height]
    S = [width, 0]
    R = [width, height]
    return np.asarray([P, Q, S, R])

range_1, aff_1 = choosePQRS(1)
range_2, aff_2 = choosePQRS(2)
range_3, aff_3 = choosePQRS(3)

def findProjTrans(PQSR):
    PQSR = np.concatenate((PQSR, np.ones((4,1))), axis=1)
    pq = np.cross(PQSR[0], PQSR[1])
    sr = np.cross(PQSR[2], PQSR[3])
    ps = np.cross(PQSR[0], PQSR[2])
    qr = np.cross(PQSR[1], PQSR[3])
    point_1 = np.cross(pq, sr)
    point_2 = np.cross(ps, qr)
    point_1 = point_1/point_1[-1]
    point_2 = point_2/point_2[-1]
    VL = np.cross(point_1, point_2)
    H = np.identity(3)
    H[-1,:] = VL/VL[-1]
    return H


def findAffineTrans(PQSR):
    PQSR = np.concatenate((PQSR, np.ones((4,1))), axis=1)
    pq = np.cross(PQSR[0], PQSR[1])
    sr = np.cross(PQSR[2], PQSR[3])
    ps = np.cross(PQSR[0], PQSR[2])
    qr = np.cross(PQSR[1], PQSR[3])

    pq = pq/pq[-1]
    sr = sr/sr[-1]
    ps = ps/ps[-1]
    qr = qr/qr[-1]
    LM_1 = np.matmul(pq[np.newaxis,:].T, ps[np.newaxis,:])
    LM_2 = np.matmul(sr[np.newaxis,:].T, qr[np.newaxis,:])
    A = np.asarray([[LM_1[0,0], LM_1[0,1]+LM_1[1,0]], [LM_2[0,0], LM_2[0,1]+LM_2
    [1,0]]])
```

```
    C = np.asarray([[LM_1[1,1]], [LM_2[1,1]]])
    s_ = np.matmul(np.linalg.pinv(A), -C).squeeze()
    S = np.asarray([[s_[0],s_[1]], [s_[1], 1]])

    _, D, VT = np.linalg.svd(S, full_matrices=True)
    d = np.asarray([[np.sqrt(D[0]), 0], [0, np.sqrt(D[1])]])
    H = np.identity(3)
    H[0:2,0:2] = np.matmul(np.matmul(VT.T,d), VT)
    return H

def projTransform(H, source):
    nps = source.shape[0]
    source_rep = np.concatenate((source, np.ones((nps,1))), axis=1)
    t_homo = np.matmul(H, source_rep.T).T
    t_norm = t_homo / t_homo[:,2].reshape((nps,1))
    return t_norm[:,:2]

def projResizeRange(source, H, limit):
    r = definePQRS(source.shape[1], source.shape[0])
    expanded = projTransform(H, r)
    l, r = np.amin(expanded[:, 0]), np.amax(expanded[:, 0])
    t, b = np.amin(expanded[:, 1]), np.amax(expanded[:, 1])
    # s_1 = (r - l) / source.shape[1]
    # s_2 = (b - t) / source.shape[0]

    # s = s_1 if s_1>s_2 else s_2

    height, width = (b-t), (r-l)
    if height > limit or width > limit:
        s_r = limit / height if height>width else limit/width
    else:
        s_r = 1
    H = np.linalg.inv(H)
    out = np.zeros((int((b-t)*s_r), int((r-l)*s_r), 3), dtype=source.dtype)
    print("Size of undistorted image is:", out.shape)
    for i in range(out.shape[0]):
        for j in range(out.shape[1]):
            coor_source = np.asarray([[j/s_r+l, i/s_r+t]])
            coor_target = projTransform(H, coor_source).squeeze()
            h, w = int(coor_target[1]), int(coor_target[0])
            if 0 <= h < source.shape[0] and 0 <= w <source.shape[1]:
                out[i,j,:] = source[h, w,:]
    return out

H_proj = findProjTrans(range_1)
result = projResizeRange(img_1, H_proj, 3000)
cv2.imwrite(outpathproj_1, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
plt.figure(figsize=(20,40))
plt.title('result')
plt.imshow(result)

H_proj = findProjTrans(range_1)
Noproj = projTransform(H_proj, aff_1)
H_aff = findAffineTrans(Noproj)
H = np.matmul(np.linalg.inv(H_aff), H_proj)

result = projResizeRange(img_1, H, 3000)
cv2.imwrite(outpath_1, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
plt.figure(figsize=(20,40))
```

```
plt.title('result')
plt.imshow(result)

H_proj = findProjTrans(aff_2)
Noproj = projTransform(H_proj, range_2)
H_aff = findAffineTrans(Noproj)
H = np.matmul(np.linalg.inv(H_aff), H_proj)

result = projResizeRange(img_2, H, 3000)
cv2.imwrite(outpath_2, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
plt.figure(figsize=(20,40))
plt.title('result')
plt.imshow(result)

H_proj = findProjTrans(range_3)
result = projResizeRange(img_3, H_proj, 3000)
cv2.imwrite(outpathproj_3, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
plt.figure(figsize=(20,40))
plt.title('result')
plt.imshow(result)

H_proj = findProjTrans(range_3)
Noproj = projTransform(H_proj, aff_3)
H_aff = findAffineTrans(Noproj)
H = np.matmul(np.linalg.inv(H_aff), H_proj)

result = projResizeRange(img_3, H, 3000)
cv2.imwrite(outpath_3, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
plt.figure(figsize=(20,40))
plt.title('result')
plt.imshow(result)
```

One step method

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os

folder_task = '/home/xingguang/Documents/ECE661/hw3/hw3_Task1_Images/Images'
path_1 = os.path.join(folder_task, '1.jpeg')
path_2 = os.path.join(folder_task, '2.jpeg')
path_3 = os.path.join(folder_task, '3.jpeg')
outpath_1 = os.path.join(folder_task, 'onestep1.jpeg')
outpath_2 = os.path.join(folder_task, 'onestep2.jpeg')
outpath_3 = os.path.join(folder_task, 'onestep3.jpeg')

img_1 = cv2.cvtColor(cv2.imread(path_1), cv2.COLOR_BGR2RGB)
img_2 = cv2.cvtColor(cv2.imread(path_2), cv2.COLOR_BGR2RGB)
img_3 = cv2.cvtColor(cv2.imread(path_3), cv2.COLOR_BGR2RGB)

def choosePQRS(img=1):
    if img == 1:
        P = [472, 118]
        Q = [449, 562]
        S = [1406, 448]
        R = [1447, 715]
        K = [705, 346]
        M = [705, 376]
```

```
        N = [725, 383]
        # 726, 353
        # K = [525, 242]
        # M = [511, 635]
        # N = [879, 682]

    if img == 2:
        P = [382, 576]
        Q = [379, 834]
        S = [593, 551]
        R = [606, 922]
        K = [425, 231]
        M = [425, 346]
        N = [515, 298]

    if img == 3:
        P = [2062, 698] #2nd
        Q = [2096, 1479]
        S = [2668, 717]
        R = [2694, 1333]
        K = [2089, 738]
        M = [2652, 749]
        N = [2674, 1301]

    return np.asarray([P, Q, S, R, K, M, N])


def definePQRS(width, height):
    P = [0, 0]
    Q = [0, height]
    S = [width, 0]
    R = [width, height]
    return np.asarray([P, Q, S, R])

range_1 = choosePQRS(1)
range_2 = choosePQRS(2)
range_3 = choosePQRS(3)

def HomoOnestep(points):
    PQSRKMN = np.concatenate((points, np.ones((7, 1))), axis=1)
    PQ=np.cross(PQSRKMN[0], PQSRKMN[1])
    PS=np.cross(PQSRKMN[0], PQSRKMN[2])
    SR=np.cross(PQSRKMN[2], PQSRKMN[3])
    RQ=np.cross(PQSRKMN[3], PQSRKMN[1])
    KM=np.cross(PQSRKMN[4], PQSRKMN[5])
    MN=np.cross(PQSRKMN[5], PQSRKMN[6])
    # KM=np.cross(PQSRKMN[0], PQSRKMN[3])
    # MN=np.cross(PQSRKMN[1], PQSRKMN[2])

    A=np.zeros((5,5))
    B=np.zeros((5,1))
    A[0,:]=[PQ[0]*PS[0], 0.5*(PQ[1]*PS[0]+PQ[0]*PS[1]), PQ[1]*PS[1], 0.5*(PQ[0]*PS
    [2]+PQ[2]*PS[0]), 0.5*(PQ[2]*PS[1]+PQ[1]*PS[2])]
    A[1,:]=[PS[0]*SR[0], 0.5*(PS[1]*SR[0]+PS[0]*SR[1]), PS[1]*SR[1], 0.5*(PS[0]*SR
    [2]+PS[2]*SR[0]), 0.5*(PS[2]*SR[1]+PS[1]*SR[2])]
    A[2,:]=[SR[0]*RQ[0], 0.5*(SR[1]*RQ[0]+SR[0]*RQ[1]), SR[1]*RQ[1], 0.5*(SR[0]*RQ
    [2]+SR[2]*RQ[0]), 0.5*(SR[2]*RQ[1]+SR[1]*RQ[2])]
    A[3,:]=[RQ[0]*PQ[0], 0.5*(RQ[1]*PQ[0]+RQ[0]*PQ[1]), RQ[1]*PQ[1], 0.5*(RQ[0]*PQ
    [2]+RQ[2]*PQ[0]), 0.5*(RQ[2]*PQ[1]+RQ[1]*PQ[2])]
```

```
    A[4,:]=[KM[0]*MN[0], 0.5*(KM[1]*MN[0]+KM[0]*MN[1]), KM[1]*MN[1], 0.5*(KM[0]*MN
    [2]+KM[2]*MN[0]), 0.5*(KM[2]*MN[1]+KM[1]*MN[2])]

    B[0]=-PQ[2]*PS[2]
    B[1]=-PS[2]*SR[2]
    B[2]=-SR[2]*RQ[2]
    B[3]=-RQ[2]*PQ[2]
    B[4]=-KM[2]*MN[2]

    C = np.matmul(np.linalg.pinv(A),B).squeeze()
    C = C/np.max(C)

    S = np.asarray([[C[0], 0.5*C[1]], [0.5*C[1], C[2]]])
    U, D, VT = np.linalg.svd(S)
    Dsquare = np.diag(np.sqrt(D))
    K = np.dot(VT, np.dot(Dsquare, VT))
    s = np.asarray([[0.5*C[3]], [0.5*C[4]]])
    v = np.dot(np.linalg.inv(K), s)
    H = np.asarray([[K[0,0],K[0,1],0],[K[1,0],K[1,1],0],[v[0,0],v[1,0],1]])
    return H

def projTransform(H, source):
    nps = source.shape[0]
    source_rep = np.concatenate((source, np.ones((nps,1))), axis=1)
    t_homo = np.matmul(H, source_rep.T).T
    t_norm = t_homo / t_homo[:,2].reshape((nps,1))
    return t_norm[:,:2]

def projResizeRange(source, H, limit):
    r = definePQRS(source.shape[1], source.shape[0])
    expanded = projTransform(H, r)
    l, r = np.amin(expanded[:, 0]), np.amax(expanded[:, 0])
    t, b = np.amin(expanded[:, 1]), np.amax(expanded[:, 1])

    height, width = (b-t), (r-l)
    if height > limit or width > limit:
        s_r = limit / height if height>width else limit/width
    else:
        s_r = 1
    H = np.linalg.inv(H)
    out = np.zeros((int((b-t)*s_r), int((r-l)*s_r), 3), dtype=source.dtype)
    print("Size of undistorted image is:", out.shape)
    for i in range(out.shape[0]):
        for j in range(out.shape[1]):
            coor_source = np.asarray([[j/s_r+l, i/s_r+t]])
            coor_target = projTransform(H, coor_source).squeeze()
            h, w = int(coor_target[1]), int(coor_target[0])
            if 0 <= h < source.shape[0] and 0 <= w <source.shape[1]:
                out[i,j,:] = source[h, w,:]
    return out

H = HomoOnestep(range_1)
result = projResizeRange(img_1, np.linalg.inv(H), 3000)
cv2.imwrite(outpath_1, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
plt.figure(figsize=(20,40))
plt.title('result')
plt.imshow(result)

H = HomoOnestep(range_2)
```

```
result = projResizeRange(img_2, np.linalg.inv(H), 3000)
cv2.imwrite(outpath_2, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
plt.figure(figsize=(20,40))
plt.title('result')
plt.imshow(result)

H = HomoOnestep(range_3)
result = projResizeRange(img_3, np.linalg.inv(H), 3000)
cv2.imwrite(outpath_3, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
plt.figure(figsize=(20,40))
plt.title('result')
plt.imshow(result)
```