ECE 661 Computer Vision (2018 Fall)
Homework 3

Runzhe Zhang

September 18, 2018

1. Using Point-to-Point Correspondences

The point p in a $2D$ plane to project into another plane is point p', canbe written as:

$$x' = Hx \quad \rightarrow \quad \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

We need to calculate getting the homography matrx H, there are 9 paremeters and 8 of the parameters need to be solved because we only care about the ratios of elements in H. Let's set the $h_{33} = 1$ first. Besides, we know the physical coordinates (x, y) and (x', y') can be get:

$$x = \frac{x_1}{x_3}\ ,\ y = \frac{x_2}{x_3} \quad and \quad x' = \frac{x'_1}{x'_3}\ ,\ y' = \frac{x'_2}{x'_3}$$

After that, we can get:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{H_{31}x + h_{32}y + 1}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{H_{31}x + h_{32}y + 1}$$

Every 2 equations for a single point and there are 8 parameters need to be solved. So, in total we need 8 equations. Four different pair points are required for calculating the homography uniquely. We can get this equation:

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4y'_4 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}$$

Use the above equation, we can get the H. After we get the point-to-point homography H, we can use it to remove the distortion. In the calculation process, I assume one pixel is equal to one centimeter.

2. 2-steps-method

2.1 Using Vanishing Line to Remove Projective Distortion

The second method to remove the projective distortion is to use the vanishing line in the image, based on the theory l_∞ to l_∞. So to remove the projective distortion is to find the homography that maps l_∞ in image back to l_∞ in world.

We should choose two pair of parallel lines in the world and to check the points on these lines in the images, for exmaple: I got p,q,r,s points pixel value in the image. Then, use cross product to get the tow pairs parallel lines in the image(not parallel in image, only parallel in world). Each pair parallel lines will get an ideal point through cross product and we also can use two ideal points to get vanishing line through cross product.

$$l_{pq} = p \times q \quad , \quad l_{rs} = r \times s;$$
$$l_{pr} = p \times r \quad , \quad l_{qs} = q \times s;$$
$$IdearPoint_{pq,rs} = l_{pq} \times l_{rs} \quad , \quad IdearPoint_{pr,qs} = l_{pr} \times l_{qs}$$
$$Vanishing_Line = IdearPoint_{pq,rs} \times IdearPoint_{pr,qs}$$

After we get the vanishing line, we can get the homography from l_∞ in image to l_∞ in world.

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

Use this homography to remove the projective distortion.

2.1.1 Extra Credit

To get the vanishing line use different two pairs of parallel lines. The first pair of parallel lines can be expressed by the cross product by these four points:

$$p = \begin{bmatrix} 220 \\ 1201 \\ 1 \end{bmatrix}, \quad q = \begin{bmatrix} 2229 \\ 181 \\ 1 \end{bmatrix}, \quad r = \begin{bmatrix} 125 \\ 1655 \\ 1 \end{bmatrix}, \quad s = \begin{bmatrix} 2323 \\ 1162 \\ 1 \end{bmatrix}$$

The two pairs of parallel lines are:

$$l_{pq} = \begin{bmatrix} 1020 \\ 2009 \\ -2637209 \end{bmatrix}, \quad l_{rs} = \begin{bmatrix} 493 \\ 2198 \\ -3699315 \end{bmatrix} \quad and \quad l_{pr} = \begin{bmatrix} -454 \\ -95 \\ 213975 \end{bmatrix}, \quad l_{qs} = \begin{bmatrix} -981 \\ 94 \\ 2169635 \end{bmatrix}$$

The two ideal point based on these two parallel lines are :

$$IdealPoint_{pq,rs} = \begin{bmatrix} -1306.67871 \\ 1976.11811 \\ 1 \end{bmatrix}, \quad and \quad IdealPoint_{pr,qs} = \begin{bmatrix} 1665.02767 \\ -5704.71120 \\ 1 \end{bmatrix}$$

The vanishing line based on the $IdealPoint_{pr,qs}$ and $IdealPoint_{pq,rs}$ are:

$$VanishingLine_1 = \begin{bmatrix} 0.00184460911 \\ 0.00071367708 \\ 1 \end{bmatrix}$$

The second pair of parallel lines can be expressed by the cross product by these four points:

$$P = \begin{bmatrix} 249 \\ 1241 \\ 1 \end{bmatrix}, \quad Q = \begin{bmatrix} 2034 \\ 399 \\ 1 \end{bmatrix}, \quad R = \begin{bmatrix} 165 \\ 1643 \\ 1 \end{bmatrix}, \quad S = \begin{bmatrix} 2077 \\ 1218 \\ 1 \end{bmatrix}$$

The two pairs of parallel lines are:

$$l_{PQ} = \begin{bmatrix} 842 \\ 1785 \\ -2424843 \end{bmatrix}, \quad l_{RS} = \begin{bmatrix} 425 \\ 1912 \\ -3211541 \end{bmatrix} \quad and \quad l_{PR} = \begin{bmatrix} -402 \\ -84 \\ 204342 \end{bmatrix}, \quad l_{QS} = \begin{bmatrix} -819 \\ 43 \\ 1648689 \end{bmatrix}$$

The two ideal point based on these two parallel lines are :

$$IdealPoint_{PQ,RS} = \begin{bmatrix} -1287.82793 \\ -5755.1739 \\ 1 \end{bmatrix}, \quad and \quad IdealPoint_{PR,QS} = \begin{bmatrix} 1710.88708 \\ -5755.1739 \\ 1 \end{bmatrix}$$

The vanishing line based on the $IdealPoint_{PR,QS}$ and $IdealPoint_{PQ,RS}$ are:

$$VanishingLine_2 = \begin{bmatrix} 0.00190730342 \\ 0.000740756207 \\ 1 \end{bmatrix}$$

Finally, we draw the two vanishing line like this:

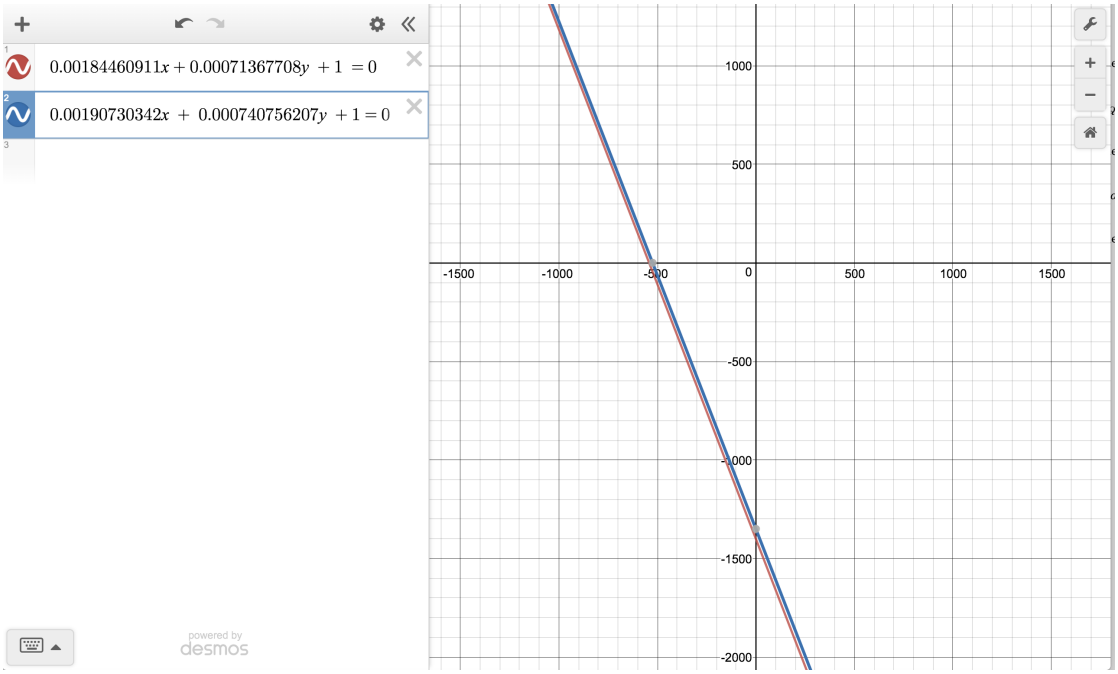

Figure 1: Vanishing Line

The red line are the first vanishing line and the blue line is the second vanishing line. We can see they are almost same, the little different may comes from the error to catch the pixels of the four points in parallel lines. So, until now, we can say the different set of parallel lines will get the same vanishing line.

2.2 Remove Affine Distortion

In this step, we need two pairs physically orthogonal lines to remove the affine distortion. Suppose we have two orthogonal lines $L = [l_1,\ l_2,\ l_3]^\top$ and $M = [m_1,\ m_2,\ m_3]^\top$ in the world plan. In the lecture 5, we know that the angle between two lines can be expressed by:

$$\cos\theta = \frac{L^\top C_\infty^* M}{\sqrt{(L^\top C_\infty^* L)\,(M^\top C_\infty^* M)}}, \qquad C_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The homography we define here are map world plane to image plane. We can get the lines in world plane by $L = H^\top L', \quad M = H^\top M'$. When $\cos\theta = 0$, we will get:

$$L'^\top H C_\infty^* H^\top M' = 0$$

We also denote the $H = \begin{bmatrix} A & 0 \\ 0 & 1 \end{bmatrix}$ in this form. So, we can get the equation like this:

$$\begin{bmatrix} l_1' & l_2' & l_3' \end{bmatrix} \begin{bmatrix} AA^\top & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} m_1' \\ m_2' \\ m_3' \end{bmatrix} = 0$$

Let's denote the:

$$S = AA^\top = \begin{bmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{bmatrix}$$

then, we have the equation like this:

$$s_{11} m_1' l_1' + s_{12}(l_1' m_2' + l_2' m_1') + s_{22} l_2' m_2' = 0$$

As we only care about the ratiors, we set $s_{22} = 1$ first. In the calculation process, we need two pairs orthogonal lines to get S. After we get S, we can use sigular value decomposition(SVD) on S to get A :

$$S = AA^\top = VD^2V^\top, \qquad A = VDV^\top$$

In my calculation process, I combine the projective homography and affine homography together. It means to multiply the inverse of projective homography H and affine homography H.

3. One-Step Method

The third method to remove the projective and affine distortion is the One-Step Method. As we all know, the homography can be written like this:

$$H = \begin{bmatrix} A & 0 \\ v^\top & 1 \end{bmatrix}$$

Use the same conic function I show above in Two-Steps Method, the dual conic in the image plane can be written like this:

$$C_\infty^{*'} = HC_\infty^* H^\top = \begin{bmatrix} AA^\top & Av \\ v^\top A^\top & v^\top v \end{bmatrix} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

Then, we use the two orthogonal lines in the world can be expressed in the image plane is like this:

$$L^{'\top} C_\infty^{*'} M' = 0, \quad \begin{bmatrix} l_1' & l_2' & l_3' \end{bmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{bmatrix} m_1' \\ m_2' \\ m_3' \end{bmatrix} = 0$$

After all, to solve the H we need five pairs of orthogonal lines in the world, and expressed in the image plane to calculate the equation:

$$\begin{bmatrix} l_1'm_1', & \frac{l_1'm_2' + l_2'm_1'}{2}, & l_2'm_2', & \frac{l_1'm_3' + l_3'm_1'}{2}, & \frac{l_2'm_3' + l_3'm_2'}{2} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = -l_3'm_3'$$

Use the same method SVD to get A and then to get v by $v^\top A^\top$. Finally, we will get the homography to remove the projective and affine ditortion.

NOTE:
1. Catching point pixel is sensitive.
2. Nomalize the lines and homography.

4. Result

4.1 1.jpg Running Result

4.1.1 1.jpg coordinate image

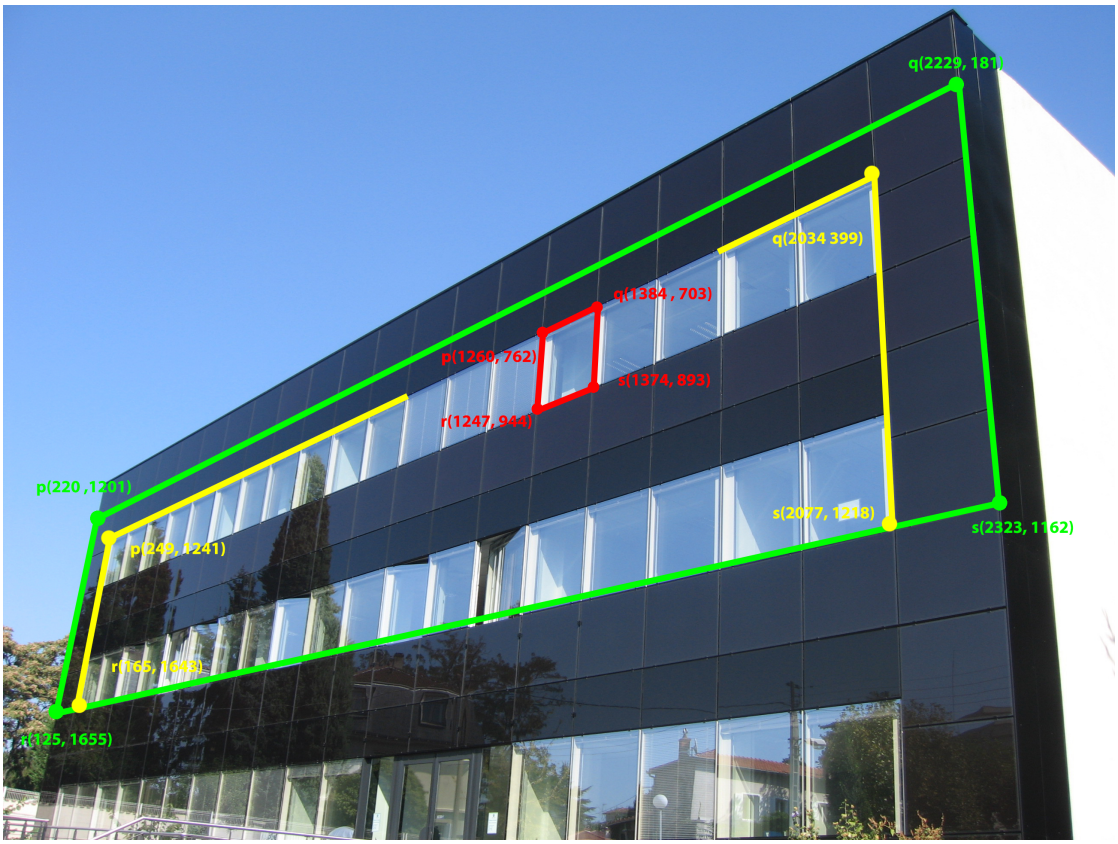

Figure 2: 1.jpg coordinate image

For 1.jpg image, I use the red p, q, ,r, s points to the Point-to-Point Correspondences Method, use green p, q, ,r, s points to do the 2-Steps- Method(include removing the projective and affine distortion), and use the green and yellow p, q, ,r, s points for the One-Step Method.

4.1.2 1.jpg Using Point-to-Point Correspondences Method

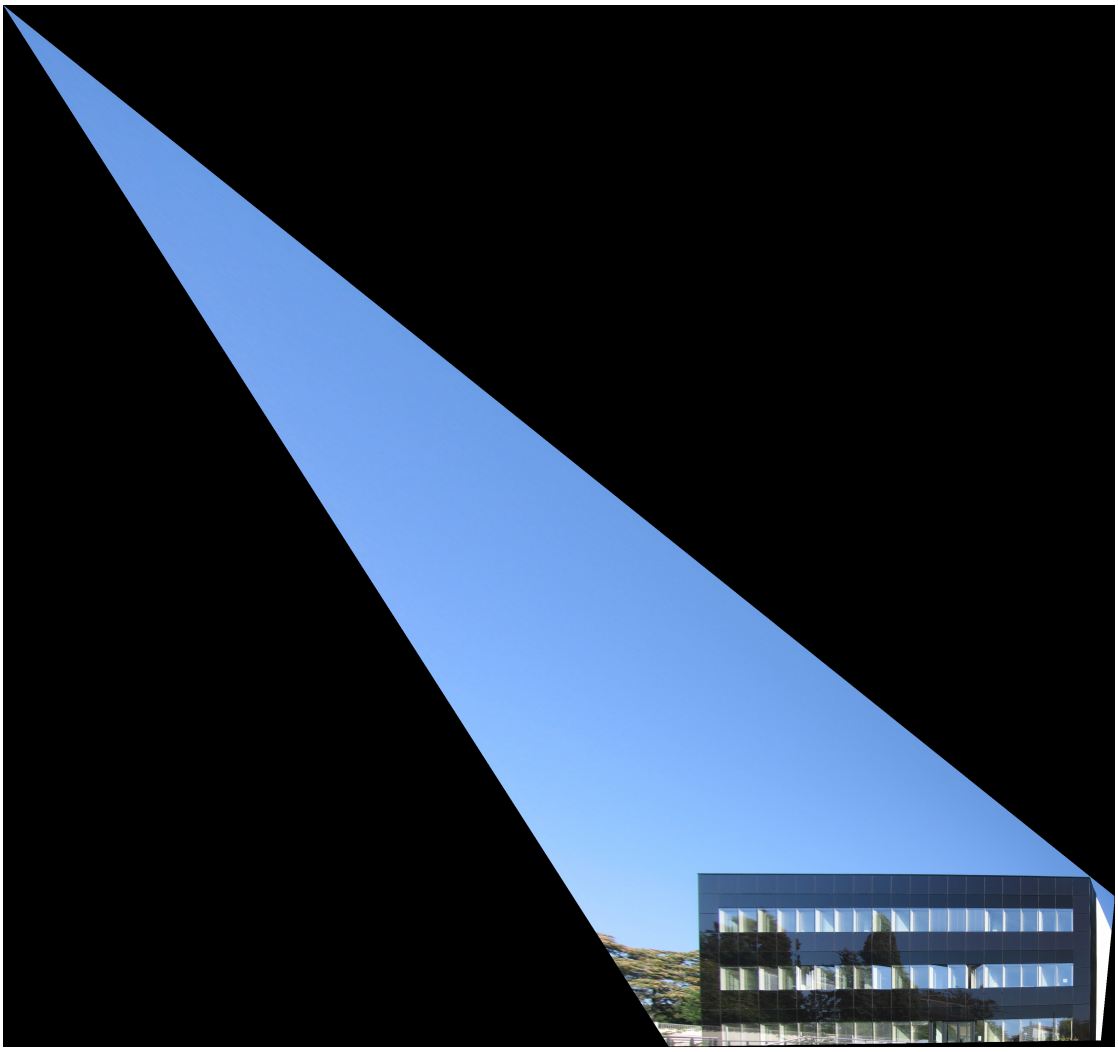

Figure 3: 1.jpg Using Point-to-Point Correspondences Method

4.1.3 1.jpg Using 2-Step Method - Remove Projective Distortion Using Vanishing Line

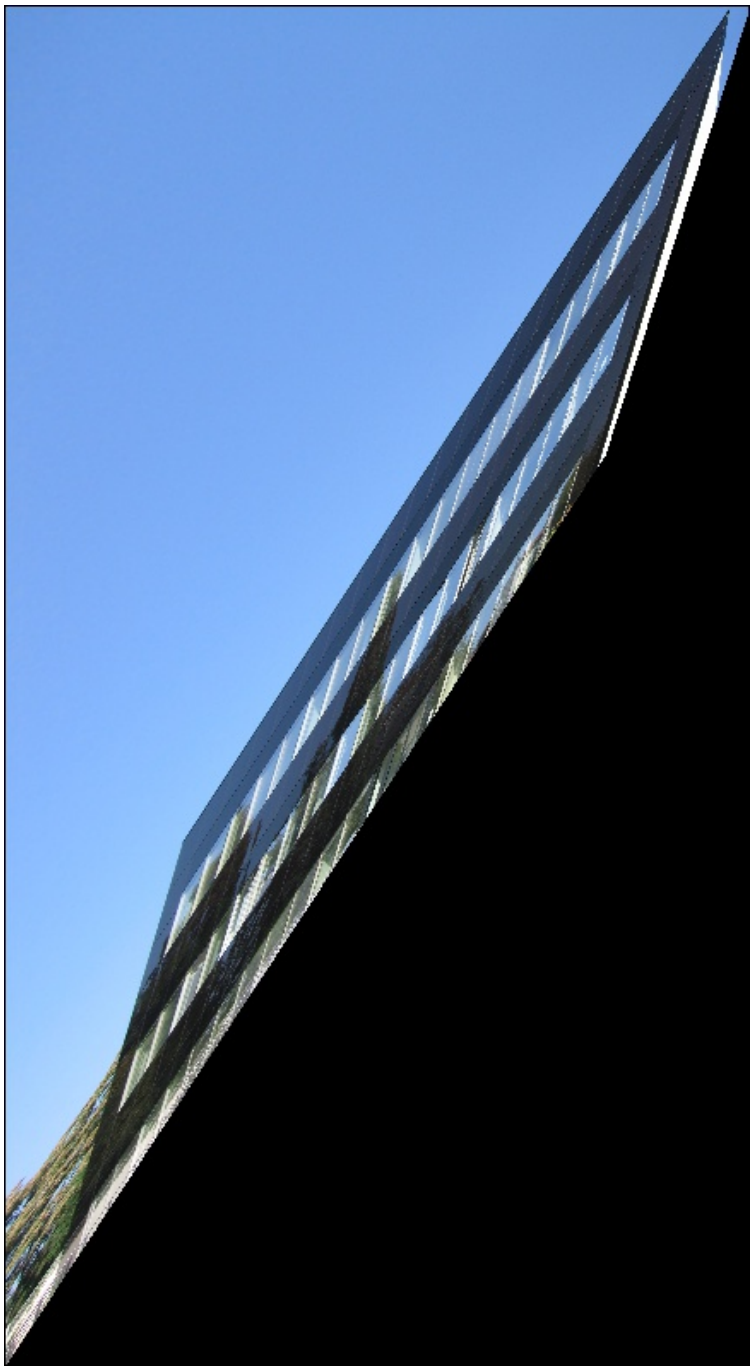

Figure 4: 1.jpg Using 2-Step Method Remove Projective Distortion Using Vanishing Line

4.1.4 1.jpg Using 2-Step Method -Remove Affine Distortion

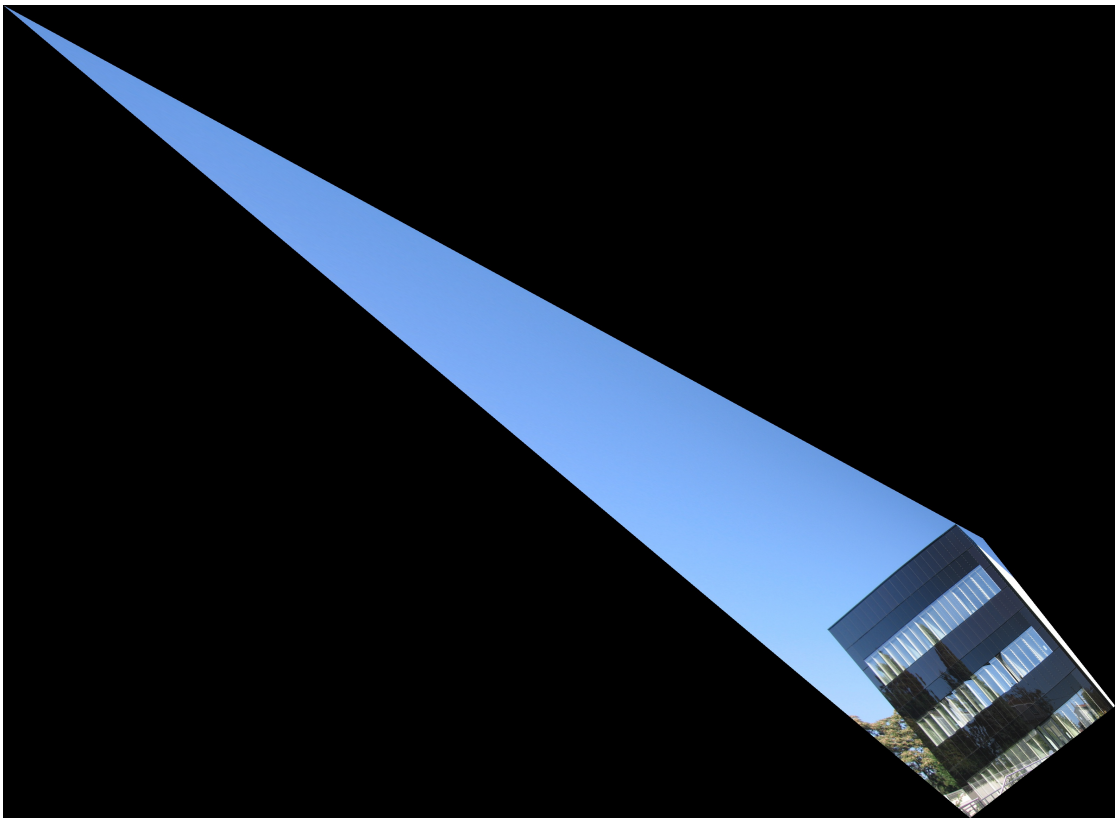

Figure 5: 1.jpg Using 2-Step Method Remove Affine Distortion

4.1.5 1.jpg Using 1-Step Method

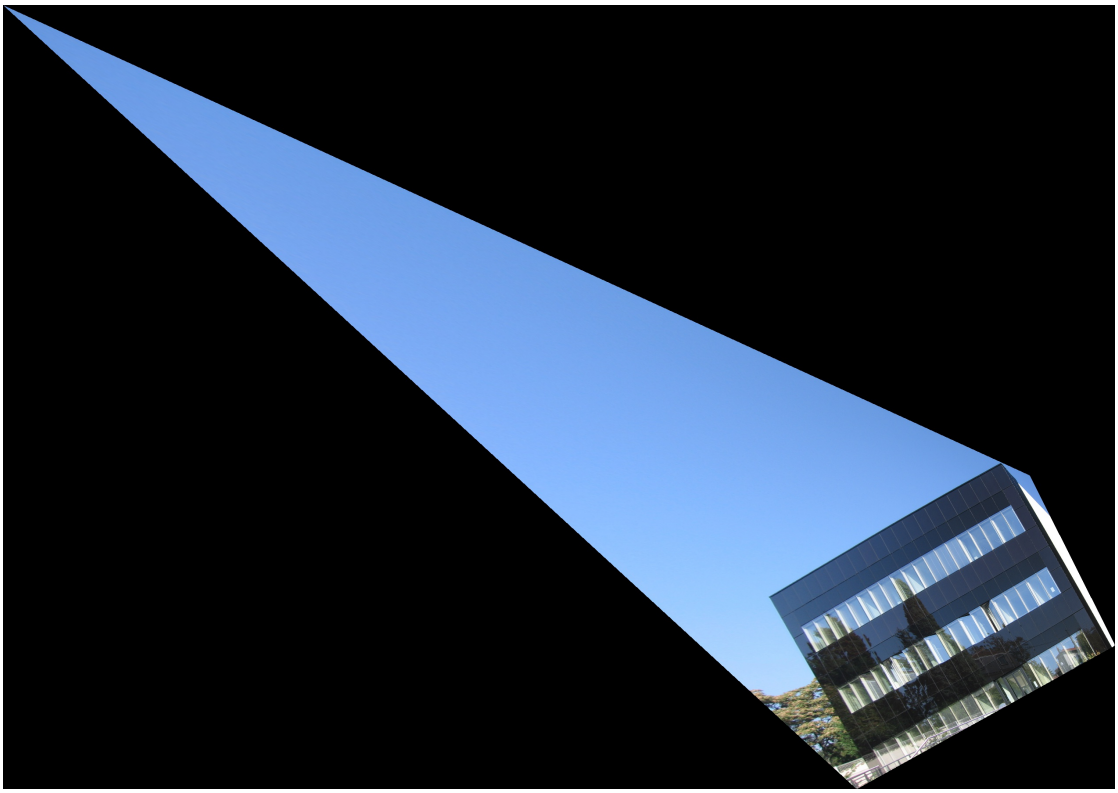

Figure 6: 1.jpg Using 1-Step Method

4.2 2.jpg Running Result

4.2.1 2.jpg coordinate image

Figure 7: 2.jpg coordinate image

For 2.jpg image, I use the green p, q, ,r, s points to the Point-to-Point Correspondences Method, and the 2-Steps- Method(include removing the projective and affine distortion), and use the green and yellow p, q, ,r, s points for the One-Step Method.

4.2.2 2.jpg Using Point-to-Point Correspondences Method

Figure 8: 2.jpg Using Point-to-Point Correspondences Method

4.2.3 2.jpg Using 2-Step Method Remove Projective Distortion Using Vanishing Line

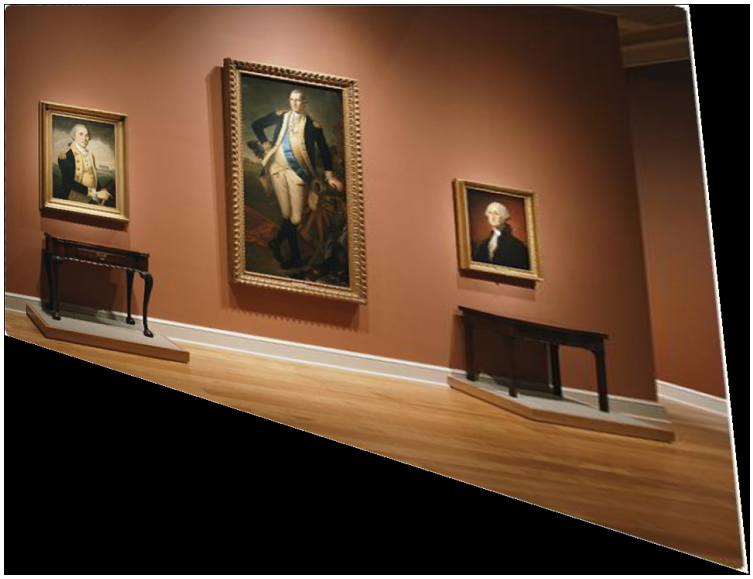

Figure 9: 2.jpg Using 2-Step Method Remove Projective Distortion Using Vanishing Line

4.2.4 2.jpg Using 2-Step Method Remove Affine Distortion

Figure 10: 2.jpg Using 2-Step Method Remove Affine Distortion

4.2.5 2.jpg Using 1-Step Method

Figure 11: 2.jpg Using 1-Step Method

4.3 My own image 1 Running Result

4.3.1 My own image 1 coordinate image

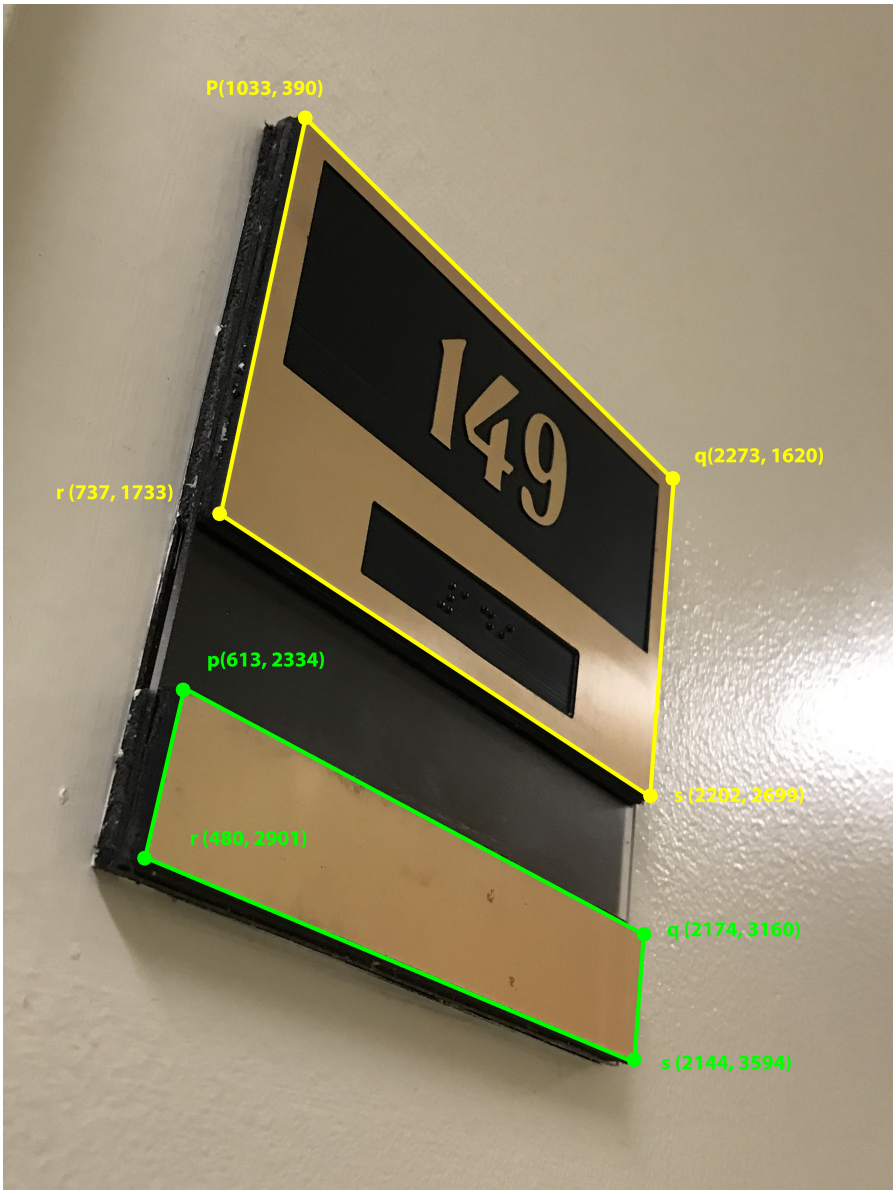

Figure 12: My own image 1 coordinate image

For my own image 1, I use the yellow p, q, ,r, s points to the Point-to-Point Correspondences Method, and the 2-Steps- Method(include removing the projective and affine distortion), and use the green and yellow p, q, ,r, s points for the One-Step Method.

4.3.2 My own image 1 using Point-to-Point Correspondences Method

Figure 13: My own image 1 Using Point-to-Point Correspondences Method

4.3.3 My own image 1 using 2-Step Method Remove Projective Distortion Using Vanishing Line

Figure 14: My own image 1 using 2-Step Method Remove Projective Distortion Using Vanishing Line

4.3.4 My own image 1 using 2-Step Method Remove Affine Distortion

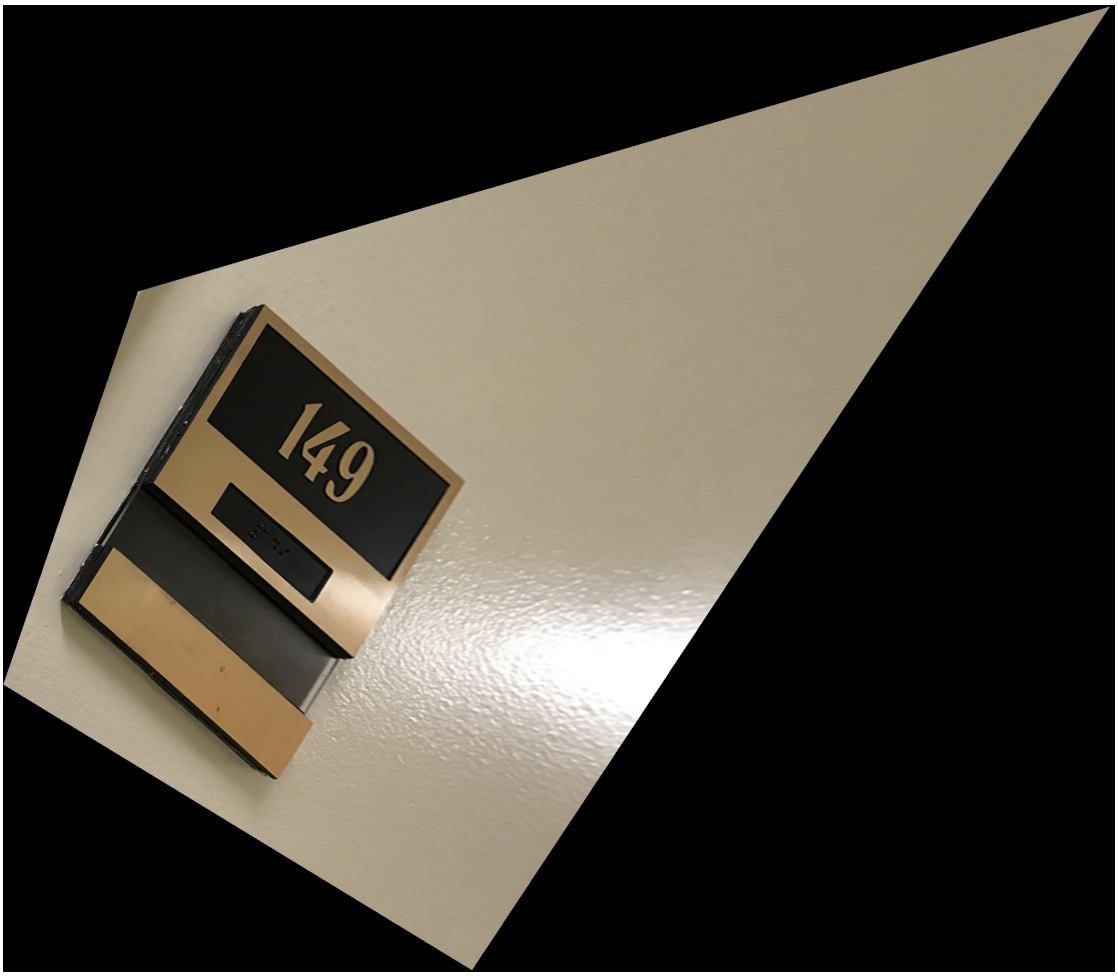

Figure 15: My own image 1 using 2-Step Method Remove Affine Distortion

4.3.5 My own image 1 using 1-Step Method

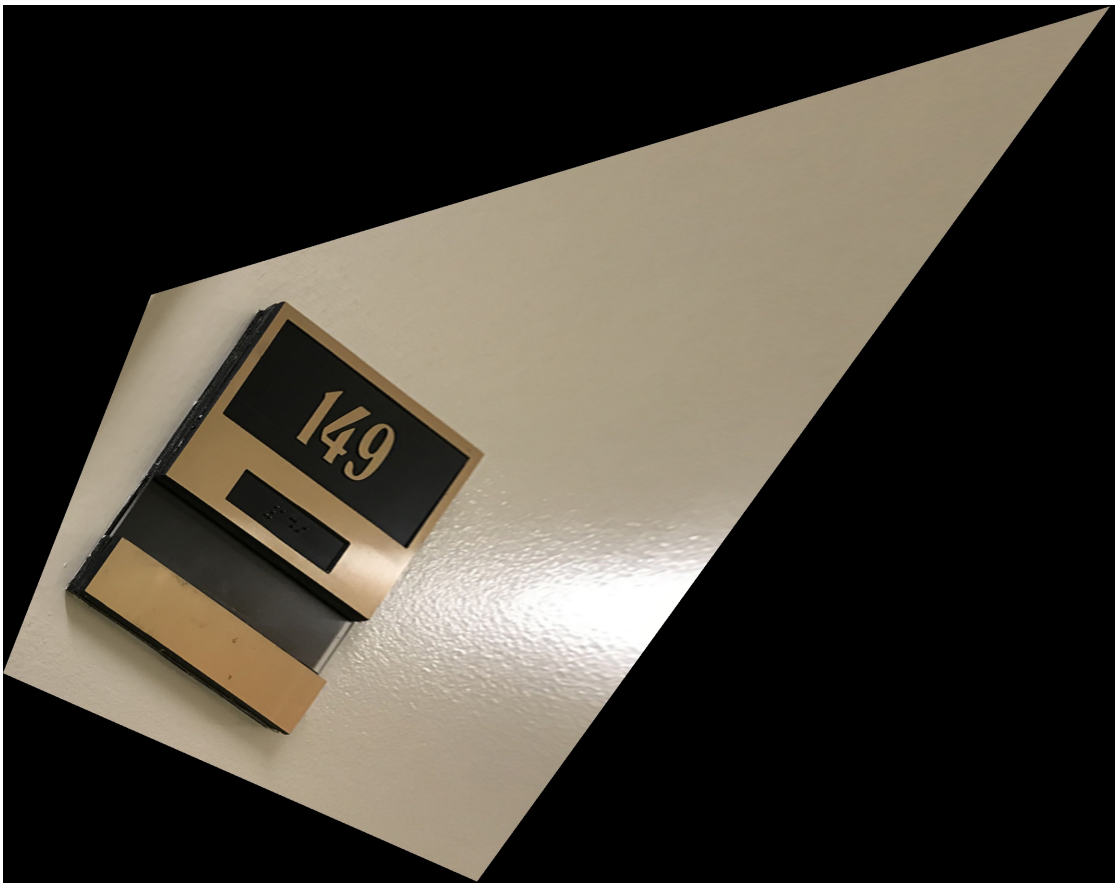

Figure 16: My own image 1 using 1-Step Method

4.4 My own image 2 Running Result

4.4.1 My own image 2 coordinate image

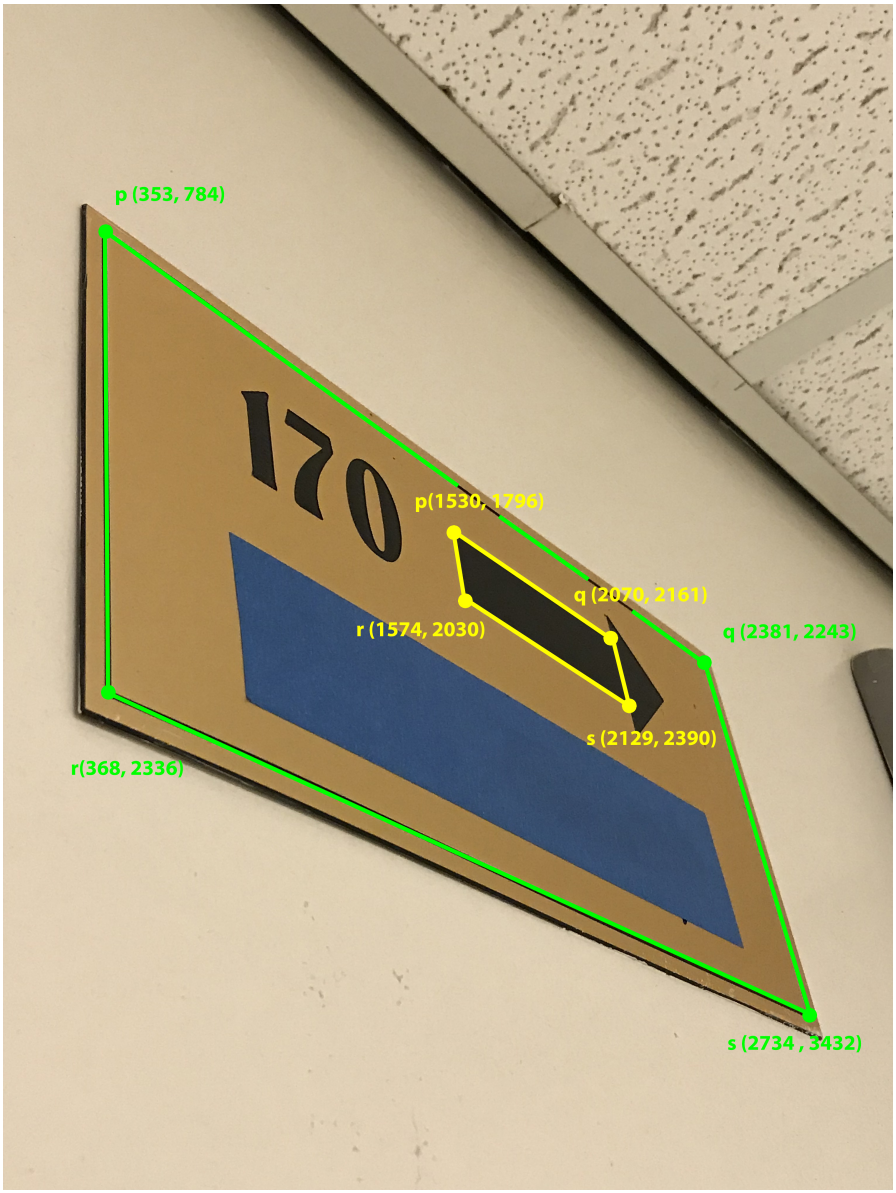

Figure 17: My own image 2 coordinate image

For my own image 2, I use the green p, q, ,r, s points to the Point-to-Point Correspondences Method, and the 2-Steps- Method(include removing the projective and affine distortion), and use the green and yellow p, q, ,r, s points for the One-Step Method.

4.4.2 My own image 2 using Point-to-Point Correspondences Method

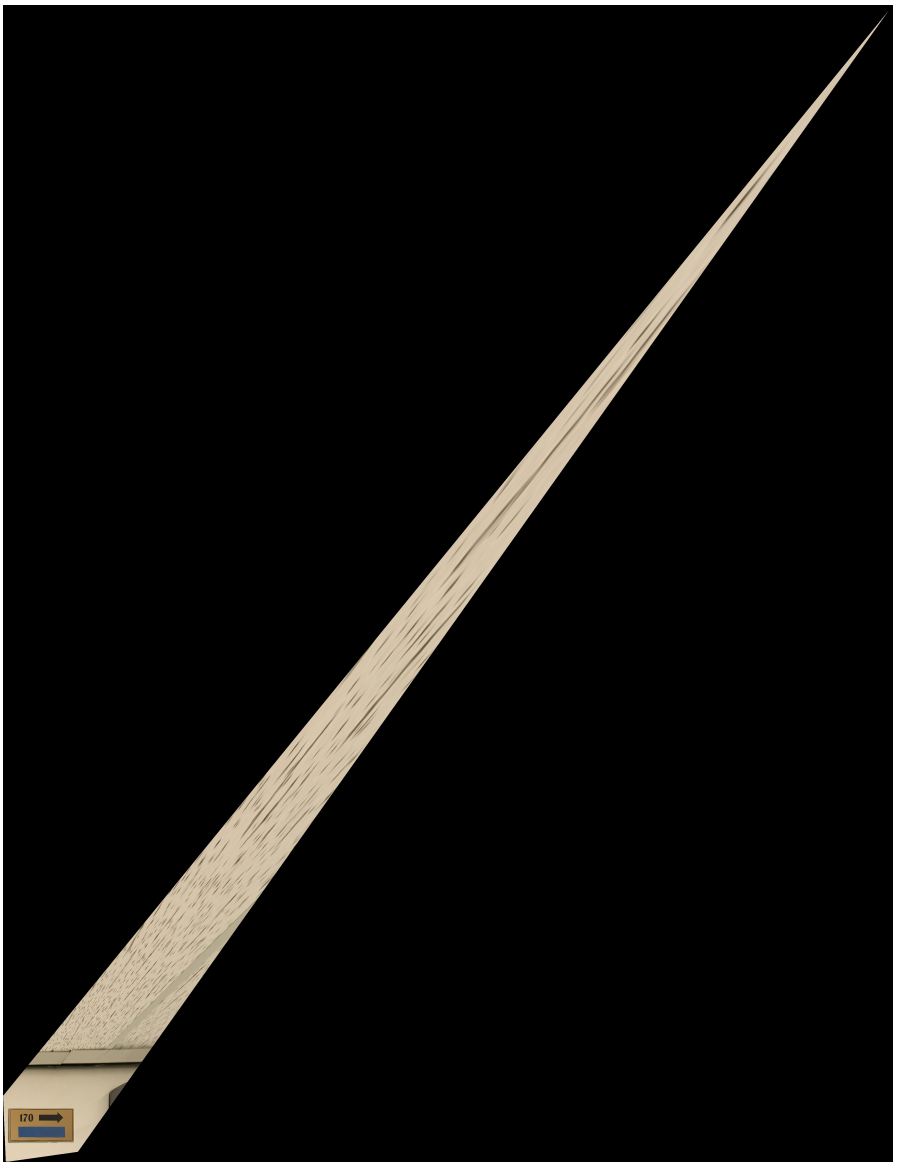

Figure 18: My own image 2 Using Point-to-Point Correspondences Method

Figure 19: My own image 2 Using Point-to-Point Correspondences Method Zoom In Detail Image

4.4.3 My own image 2 using 2-Step Method Remove Projective Distortion Using Vanishing Line

Figure 20: My own image 2 using 2-Step Method Remove Projective Distortion Using Vanishing Line

4.4.4 My own image 2 using 2-Step Method Remove Affine Distortion

Figure 21: My own image 2 using 2-Step Method Remove Affine Distortion

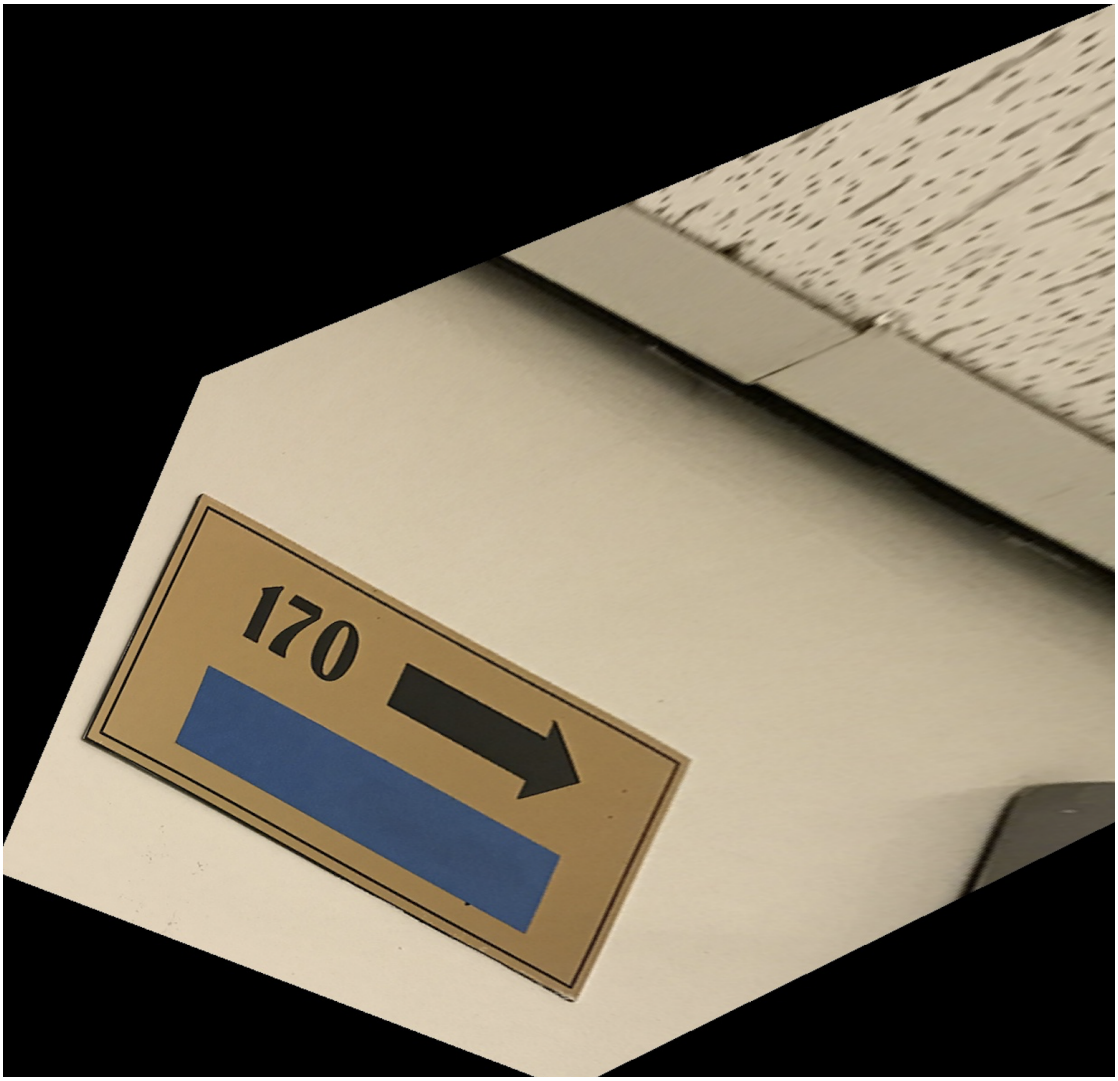

Figure 22: My own image 2 using 2-Step Method Remove Affine Distortion Zoom In Detail Image

4.4.5 My own image 2 using 1-Step Method

Figure 23: My own image 2 using 1-Step Method

Figure 24: My own image 2 using 1-Step Method Zoom In Detail Image

5. Code

```
#####
#####    ECE 661 Computer Vision(2018 Fall) Homework 3
#####          Sep 03 2018       Runzhe Zhang
#####                    Task    1
############################################################


import numpy as np
import cv2
import math


def getrgb(pt, img):
    p = img[math.floor(pt[0,1]), math.floor(pt[0,0])]
    q = img[math.floor(pt[0,1]), math.ceil(pt[0,0])]
    r = img[math.ceil(pt[0,1]), math.floor(pt[0,0])]
    s = img[math.ceil(pt[0,1]), math.ceil(pt[0,0])]
    x = pt[0,1] - math.floor(pt[0,1])
    y = pt[0,0] - math.floor(pt[0,0])
    wp = 1/np.linalg.norm(np.array([x,y]))
    wq = 1/np.linalg.norm(np.array([x,1-y]))
    wr = 1/np.linalg.norm(np.array([1-x,y]))
    ws = 1/np.linalg.norm(np.array([1-x,1-y]))
    color = (p*wp+q*wq+r*wr+s*ws)/(wp+wq+wr+ws)
    return color

img1 = cv2.imread('m11.jpg')

# 1.jpg
# p = np.matrix('1260 762 1', dtype=float)
# q = np.matrix('1384 703 1', dtype=float)
# r = np.matrix('1247 944 1', dtype=float)
# s = np.matrix('1374 893 1', dtype=float)
#
# P = np.matrix('0 0 1', dtype=float)
# Q = np.matrix('60 0 1', dtype=float)
# R = np.matrix('0 80 1', dtype=float)
# S = np.matrix('60 80 1', dtype=float)

# 2.jpg
# p = np.matrix('246 71 1', dtype=float)
```

```
# q = np.matrix('326 82 1', dtype=float)
# r = np.matrix('245 269 1', dtype=float)
# s = np.matrix('323 265 1', dtype=float)
#
# P = np.matrix('0 0 1', dtype=float)
# Q = np.matrix('40 0 1', dtype=float)
# R = np.matrix('0 80 1', dtype=float)
# S = np.matrix('40 80 1', dtype=float)

# m1.jpg
p = np.matrix('1033 390 1', dtype=float)
q = np.matrix('2273 1620 1', dtype=float)
r = np.matrix('737 1733 1', dtype=float)
s = np.matrix('2202 2699 1', dtype=float)

P = np.matrix('0 0 1', dtype=float)
Q = np.matrix('125 0 1', dtype=float)
R = np.matrix('0 75 1', dtype=float)
S = np.matrix('125 75 1', dtype=float)

equation_maxtrix = np.matrix(np.zeros((8, 8)), dtype=float)
equation_maxtrix[0, 0:3] = P
equation_maxtrix[0, 6:8] = -P[0, 0:2] * p[0, 0]
equation_maxtrix[1, 3:6] = P
equation_maxtrix[1, 6:8] = -P[0, 0:2] * p[0, 1]
equation_maxtrix[2, 0:3] = Q
equation_maxtrix[2, 6:8] = -Q[0, 0:2] * q[0, 0]
equation_maxtrix[3, 3:6] = Q
equation_maxtrix[3, 6:8] = -Q[0, 0:2] * q[0, 1]
equation_maxtrix[4, 0:3] = R
equation_maxtrix[4, 6:8] = -R[0, 0:2] * r[0, 0]
equation_maxtrix[5, 3:6] = R
equation_maxtrix[5, 6:8] = -R[0, 0:2] * r[0, 1]
equation_maxtrix[6, 0:3] = S
equation_maxtrix[6, 6:8] = -S[0, 0:2] * s[0, 0]
equation_maxtrix[7, 3:6] = S
equation_maxtrix[7, 6:8] = -S[0, 0:2] * s[0, 1]
destination_vector = np.matrix('0 0 0 0 0 0 0 0', dtype=float)
destination_vector[0, 0:2] = p[0, 0:2]
destination_vector[0, 2:4] = q[0, 0:2]
destination_vector[0, 4:6] = r[0, 0:2]
destination_vector[0, 6:8] = s[0, 0:2]
# Calculate the Homography
```

```
param = np.transpose(np.linalg.inv(equation_maxtrix) * np.transpose(destination_vec
H = np.zeros((3, 3), dtype=float)
H[0] = param[0, 0:3]
H[1] = param[0, 3:6]
H[2, 0:2] = param[0, 6:8]
H[2, 2] = 1

img_p = np.matrix('0 0 1',dtype = float)
img_q = np.matrix('0 0 1',dtype = float)
img_q[0,0] = img1.shape[1]
img_r = np.matrix('0 0 1',dtype = float)
img_r[0,1] = img1.shape[0]
img_s = np.matrix('0 0 1',dtype = float)
img_s[0,0] = img1.shape[1]
img_s[0,1] = img1.shape[0]

world_P = np.transpose(np.linalg.inv(H) * np.transpose(img_p)); world_P = world_P /
world_Q = np.transpose(np.linalg.inv(H) * np.transpose(img_q)); world_Q = world_Q /
world_R = np.transpose(np.linalg.inv(H) * np.transpose(img_r)); world_R = world_R /
world_S = np.transpose(np.linalg.inv(H) * np.transpose(img_s)); world_S = world_S /

max_point = np.maximum(np.maximum(np.maximum(world_P,world_Q),world_R),world_S)
min_point = np.minimum(np.minimum(np.minimum(world_P,world_Q),world_R),world_S)

img_result = np.zeros((int(max_point[0,1]-min_point[0,1]), int(max_point[0,0]-min_p

tmp = np.matrix('0 0 1', dtype=float)
for row in range(int(min_point[0,1]), int(max_point[0,1]) - 1):
    for col in range(int(min_point[0,0]), int(max_point[0,0]) - 1):

        tmp[0, 0] = col
        tmp[0, 1] = row
        tr = np.transpose(H * np.transpose(tmp))
        tr = tr / tr[0, 2]
        if tr[0, 0] > 0 and tr[0, 1] > 0 and tr[0, 1] < img1.shape[0] - 1 and tr[0,
            img_result[(row - int(min_point[0,1])), (col - int(min_point[0,0]))] =

cv2.imwrite('Task_1_imgM1_result1.jpg', img_result)

# Begin to remove affine distortion

p_PP = np.transpose(np.rint(np.linalg.inv(H) * np.transpose(p)))
q_PP = np.transpose(np.rint(np.linalg.inv(H) * np.transpose(q)))
```

```
r_PP = np.transpose(np.rint(np.linalg.inv(H) * np.transpose(r)))
s_PP = np.transpose(np.rint(np.linalg.inv(H) * np.transpose(s)))

pq_PP = np.cross(p_PP,q_PP)
pr_PP = np.cross(p_PP,r_PP)
qs_PP = np.cross(q_PP,s_PP)
rs_PP = np.cross(r_PP,s_PP)

s_equation = np.matrix([[pq_PP[0,0]*pr_PP[0,0] , pq_PP[0,1]*pr_PP[0,0]+pq_PP[0,0]*p
                        [qs_PP[0,0]*rs_PP[0,0] , qs_PP[0,1]*rs_PP[0,0]+qs_PP[0,0]*r

b_equation = np.matrix([[-pq_PP[0,1]*pr_PP[0,1]] , [-qs_PP[0,1]*rs_PP[0,1]]])

ans_equation = np.linalg.inv(np.transpose(s_equation)*s_equation)*(np.transpose(s_e

SS = np.matrix([[np.asscalar(np.array(ans_equation[0,0])) , np.asscalar(np.array(an
                [np.asscalar(np.array(ans_equation[1,0])), 1]])

U,D,VT = np.linalg.svd(SS)

Ds = np.matrix([[np.sqrt(D[0]) , 0],[0 , np.sqrt(D[1])]])

A = np.transpose(VT) * Ds * VT

H_affine = np.matrix([[A[0,0], A[0,1], 0],[A[1,0] , A[1,1] , 0],[0 , 0 , 1]])

H_final = np.dot(H, H_affine)

final_P = np.transpose(np.linalg.inv(H_final) * np.transpose(img_p)); final_P = fina
final_Q = np.transpose(np.linalg.inv(H_final) * np.transpose(img_q)); final_Q = fina
final_R = np.transpose(np.linalg.inv(H_final) * np.transpose(img_r)); final_R = fina
final_S = np.transpose(np.linalg.inv(H_final) * np.transpose(img_s)); final_S = fina

fmax_point = np.maximum(np.maximum(np.maximum(final_P,final_Q),final_R),final_S)
fmin_point = np.minimum(np.minimum(np.minimum(final_P,final_Q),final_R),final_S)

fimg_result = np.zeros((int(fmax_point[0,1]-fmin_point[0,1]), int(fmax_point[0,0]-f
#fimg_result = np.zeros((img1.shape[0], img1.shape[1], 3))
# Looping over the image
tmp = np.matrix('0 0 1', dtype=float)
for row in range(int(fmin_point[0,1]), int(fmax_point[0,1]) - 1):
    for col in range(int(fmin_point[0,0]), int(fmax_point[0,0]) - 1):
        # if img1fill[row,column,1] > 0:
```

```
            tmp[0, 0] = col
            tmp[0, 1] = row
            tr = np.transpose(H_final * np.transpose(tmp))
            tr = tr / tr[0, 2]
            if tr[0, 0] > 0 and tr[0, 1] > 0 and tr[0, 1] < img1.shape[0] - 1 and tr[0,
                fimg_result[(row - int(fmin_point[0,1])), (col - int(fmin_point[0,0]))]

cv2.imwrite('Task_1_fresult1.jpg', fimg_result)
```

```
#####
#####    ECE 661 Computer Vision(2018 Fall) Homework 3
#####          Sep 03 2018       Runzhe Zhang
#####                    Task   2
############################################################

import numpy as np
import cv2
import math

def getrgb(pt, img):
    p = img[math.floor(pt[0,1]), math.floor(pt[0,0])]
    q = img[math.floor(pt[0,1]), math.ceil(pt[0,0])]
    r = img[math.ceil(pt[0,1]), math.floor(pt[0,0])]
    s = img[math.ceil(pt[0,1]), math.ceil(pt[0,0])]
    x = pt[0,1] - math.floor(pt[0,1])
    y = pt[0,0] - math.floor(pt[0,0])
    wp = 1/np.linalg.norm(np.array([x,y]))
    wq = 1/np.linalg.norm(np.array([x,1-y]))
    wr = 1/np.linalg.norm(np.array([1-x,y]))
    ws = 1/np.linalg.norm(np.array([1-x,1-y]))
    color = (p*wp+q*wq+r*wr+s*ws)/(wp+wq+wr+ws)
    return color

img1 = cv2.imread('t22.jpg')

# 1.jpg
# p = np.matrix('1260 762 1', dtype=float)
# q = np.matrix('1384 703 1', dtype=float)
# r = np.matrix('1247 944 1', dtype=float)
# s = np.matrix('1374 893 1', dtype=float)
#
# P = np.matrix('0 0 1', dtype=float)
# Q = np.matrix('60 0 1', dtype=float)
# R = np.matrix('0 80 1', dtype=float)
# S = np.matrix('60 80 1', dtype=float)

# 2.jpg
# p = np.matrix('246 71 1', dtype=float)
# q = np.matrix('326 82 1', dtype=float)
# r = np.matrix('245 269 1', dtype=float)
# s = np.matrix('323 265 1', dtype=float)
#
```

```
# P = np.matrix('0 0 1', dtype=float)
# Q = np.matrix('40 0 1', dtype=float)
# R = np.matrix('0 80 1', dtype=float)
# S = np.matrix('40 80 1', dtype=float)

# m1.jpg
# p = np.matrix('170 77 1', dtype=float)
# q = np.matrix('373 319 1', dtype=float)
# r = np.matrix('79 575 1', dtype=float)
# s = np.matrix('352 711 1', dtype=float)

p = np.matrix('70 157 1', dtype=float)
q = np.matrix('471 445 1', dtype=float)
r = np.matrix('73 463 1', dtype=float)
s = np.matrix('541 680 1', dtype=float)

p = np.matrix('258 177 1', dtype=float)
q = np.matrix('449 345 1', dtype=float)
r = np.matrix('193 436 1', dtype=float)
s = np.matrix('342 666 1', dtype=float)

p = np.matrix('206 209 1', dtype=float)
q = np.matrix('364 73 1', dtype=float)
r = np.matrix('212 357 1', dtype=float)
s = np.matrix('381 275 1', dtype=float)

# p = P; q = Q; r = R; s = S;

pq = np.cross(p,q)
pr = np.cross(p,r)
qs = np.cross(q,s)
rs = np.cross(r,s)

idea_point1 = np.cross(pq,rs); idea_point1 = idea_point1 / idea_point1[0, 2]
idea_point2 = np.cross(pr,qs); idea_point2 = idea_point2 / idea_point2[0, 2]

vanishing_line = np.cross(idea_point2,idea_point1); vanishing_line = vanishing_line

H_projective = np.matrix([[1, 0, 0],[0, 1, 0],[vanishing_line[0,0], vanishing_line[

img_p = np.matrix('0 0 1',dtype = float)
img_q = np.matrix('0 0 1',dtype = float)
img_q[0,0] = img1.shape[1]
```

```
img_r = np.matrix('0 0 1',dtype = float)
img_r[0,1] = img1.shape[0]
img_s = np.matrix('0 0 1',dtype = float)
img_s[0,0] = img1.shape[1]
img_s[0,1] = img1.shape[0]

affine_P = np.transpose(H_projective * np.transpose(img_p)); affine_P = affine_P /
affine_Q = np.transpose(H_projective * np.transpose(img_q)); affine_Q = affine_Q /
affine_R = np.transpose(H_projective * np.transpose(img_r)); affine_R = affine_R /
affine_S = np.transpose(H_projective * np.transpose(img_s)); affine_S = affine_S /

affine_max_point = np.maximum(np.maximum(np.maximum(affine_P,affine_Q),affine_R),af
affine_min_point = np.minimum(np.minimum(np.minimum(affine_P,affine_Q),affine_R),af

img_result = np.zeros((int(affine_max_point[0,1]-affine_min_point[0,1]), int(affine_
#img_result = np.zeros((img1.shape[0], img1.shape[1], 3))
# Looping over the image
tmp = np.matrix('0 0 1', dtype=float)
for row in range(int(affine_min_point[0,1]), int(affine_max_point[0,1]) - 1):
    print(row)
    for col in range(int(affine_min_point[0,0]), int(affine_max_point[0,0]) - 1):
        # if img1fill[row,column,1] > 0:
        tmp[0, 0] = col
        tmp[0, 1] = row
        tr = np.transpose(np.linalg.inv(H_projective) * np.transpose(tmp))
        tr = tr / tr[0, 2]
        if tr[0, 0] > 0 and tr[0, 1] > 0 and tr[0, 1] < img1.shape[0] - 1 and tr[0,
            img_result[(row - int(affine_min_point[0,1])), (col - int(affine_min_po

cv2.imwrite('Task_2_imgT2_result1.jpg', img_result)

p_AF = np.transpose(H_projective * np.transpose(p))
q_AF = np.transpose(H_projective * np.transpose(q))
r_AF = np.transpose(H_projective * np.transpose(r))
s_AF = np.transpose(H_projective * np.transpose(s))

pq_AF = np.cross(p_AF,q_AF)
pr_AF = np.cross(p_AF,r_AF)
qs_AF = np.cross(q_AF,s_AF)
rs_AF = np.cross(r_AF,s_AF)

s_equation = np.matrix([[pq_AF[0,0]*pr_AF[0,0] , pq_AF[0,1]*pr_AF[0,0]+pq_AF[0,0]*p
                        [pq_AF[0,0]*qs_AF[0,0] , pq_AF[0,1]*qs_AF[0,0]+pq_AF[0,0]*q
```

```
b_equation = np.matrix([[-pq_AF[0,1]*pr_AF[0,1]] , [-pq_AF[0,1]*qs_AF[0,1]]])

s_invs = np.linalg.inv(s_equation)
ans_equation = np.matmul(s_invs,b_equation)

SS = np.matrix([[np.asscalar(np.array(ans_equation[0,0])) , np.asscalar(np.array(an
                [np.asscalar(np.array(ans_equation[1,0])), 1]])

U,D,VT = np.linalg.svd(SS)

Ds = np.matrix([[np.sqrt(D[0]) , 0],[0 , np.sqrt(D[1])]])

A = np.transpose(VT) * Ds * VT ; A = A / np.max(A)

H_affine = np.matrix([[A[0,0], A[0,1], 0],[A[1,0] , A[1,1] , 0],[0 , 0 , 1]])

H_final = np.dot(np.linalg.inv(H_projective),H_affine)


final_P = np.transpose(np.linalg.inv(H_final) * np.transpose(img_p)); final_P = fina
final_Q = np.transpose(np.linalg.inv(H_final) * np.transpose(img_q)); final_Q = fina
final_R = np.transpose(np.linalg.inv(H_final) * np.transpose(img_r)); final_R = fina
final_S = np.transpose(np.linalg.inv(H_final) * np.transpose(img_s)); final_S = fina

fmax_point = np.maximum(np.maximum(np.maximum(final_P,final_Q),final_R),final_S)
fmin_point = np.minimum(np.minimum(np.minimum(final_P,final_Q),final_R),final_S)

fimg_result = np.zeros((int(fmax_point[0,1]-fmin_point[0,1]), int(fmax_point[0,0]-f

tmp = np.matrix('0 0 1', dtype=float)
for row in range(int(fmin_point[0,1]), int(fmax_point[0,1]) - 1):
    print(row)
    for col in range(int(fmin_point[0,0]), int(fmax_point[0,0]) - 1):

        tmp[0, 0] = col
        tmp[0, 1] = row
        tr = np.transpose(H_final * np.transpose(tmp))
        tr = tr / tr[0, 2]
        if tr[0, 0] > 0 and tr[0, 1] > 0 and tr[0, 1] < img1.shape[0] - 1 and tr[0,
            fimg_result[(row - int(fmin_point[0,1])), (col - int(fmin_point[0,0]))]

cv2.imwrite('Task_2_imgT2_result2.jpg', fimg_result)
```

```
#####
#####    ECE 661 Computer Vision(2018 Fall) Homework 3
#####          Sep 03 2018       Runzhe Zhang
#####                    Task   3
############################################################

import numpy as np
import cv2
import math

def getrgb(pt, img):
    p = img[math.floor(pt[0,1]), math.floor(pt[0,0])]
    q = img[math.floor(pt[0,1]), math.ceil(pt[0,0])]
    r = img[math.ceil(pt[0,1]), math.floor(pt[0,0])]
    s = img[math.ceil(pt[0,1]), math.ceil(pt[0,0])]
    x = pt[0,1] - math.floor(pt[0,1])
    y = pt[0,0] - math.floor(pt[0,0])
    wp = 1/np.linalg.norm(np.array([x,y]))
    wq = 1/np.linalg.norm(np.array([x,1-y]))
    wr = 1/np.linalg.norm(np.array([1-x,y]))
    ws = 1/np.linalg.norm(np.array([1-x,1-y]))
    color = (p*wp+q*wq+r*wr+s*ws)/(wp+wq+wr+ws)
    return color

img1 = cv2.imread('t22.jpg')

img_p = np.matrix('0 0 1',dtype = float)
img_q = np.matrix('0 0 1',dtype = float)
img_q[0,0] = img1.shape[1]
img_r = np.matrix('0 0 1',dtype = float)
img_r[0,1] = img1.shape[0]
img_s = np.matrix('0 0 1',dtype = float)
img_s[0,0] = img1.shape[1]
img_s[0,1] = img1.shape[0]

# 1.jpg
# p = np.matrix('220 1201 1', dtype=float)
# q = np.matrix('2229 181 1', dtype=float)
# r = np.matrix('125 1655 1', dtype=float)
# s = np.matrix('2323 1162 1', dtype=float)
#
# P = np.matrix('249 1241 1', dtype=float)
# Q = np.matrix('2034 399 1', dtype=float)
```

```
# R = np.matrix('165 1643 1', dtype=float)
# S = np.matrix('2077 1218 1', dtype=float)

# p = np.matrix('248 1242', dtype=float)
# q = np.matrix('2033 398 1', dtype=float)
# r = np.matrix('224 1357 1', dtype=float)
# s = np.matrix('2047 620 1', dtype=float)

# 2.jpg
# p = np.matrix('246 71 1', dtype=float)
# q = np.matrix('326 82 1', dtype=float)
# r = np.matrix('245 269 1', dtype=float)
# s = np.matrix('323 265 1', dtype=float)
#
# P = np.matrix('0 0 1', dtype=float)
# Q = np.matrix('40 0 1', dtype=float)
# R = np.matrix('0 80 1', dtype=float)
# S = np.matrix('40 80 1', dtype=float)

# m1.jpg
# p = np.matrix('170 77 1', dtype=float)
# q = np.matrix('373 319 1', dtype=float)
# r = np.matrix('79 575 1', dtype=float)
# s = np.matrix('352 711 1', dtype=float)

# m1.jpg
# p = np.matrix('170 77 1', dtype=float)
# q = np.matrix('373 319 1', dtype=float)
# r = np.matrix('79 575 1', dtype=float)
# s = np.matrix('352 711 1', dtype=float)

# P = np.matrix('180 106 1', dtype=float)
# Q = np.matrix('366 324 1', dtype=float)
# R = np.matrix('157 243 1', dtype=float)
# S = np.matrix('358 434 1', dtype=float)

# p = np.matrix('70 157 1', dtype=float)
# q = np.matrix('471 445 1', dtype=float)
# r = np.matrix('73 463 1', dtype=float)
# s = np.matrix('541 680 1', dtype=float)
#
# P = np.matrix('305 360 1', dtype=float)
# Q = np.matrix('410 431 1', dtype=float)
```

```
# R = np.matrix('313 401 1', dtype=float)
# S = np.matrix('421 472 1', dtype=float)

p = np.matrix('258 177 1', dtype=float)
q = np.matrix('449 345 1', dtype=float)
r = np.matrix('193 436 1', dtype=float)
s = np.matrix('342 666 1', dtype=float)

P = np.matrix('305 309 1', dtype=float)
Q = np.matrix('420 430 1', dtype=float)
R = np.matrix('245 517 1', dtype=float)
S = np.matrix('342 666 1', dtype=float)

p = np.matrix('206 209 1', dtype=float)
q = np.matrix('364 73 1', dtype=float)
r = np.matrix('212 357 1', dtype=float)
s = np.matrix('381 275 1', dtype=float)

P = np.matrix('214 400 1', dtype=float)
Q = np.matrix('387 338 1', dtype=float)
R = np.matrix('216 449 1', dtype=float)
S = np.matrix('393 407 1', dtype=float)

# P = np.matrix('234 394 1', dtype=float)
# Q = np.matrix('386 339 1', dtype=float)
# R = np.matrix('216 449 1', dtype=float)
# S = np.matrix('391 391 1', dtype=float)


pq = np.cross(p,q) #; pq = pq / pq[0, 2]
pr = np.cross(p,r) #; pr = pr / pr[0, 2]
qs = np.cross(q,s) #; qs = qs / qs[0, 2]
rs = np.cross(r,s) #; rs = rs / rs[0, 2]

PQ = np.cross(P,Q) #; PQ = PQ / PQ[0, 2]
PR = np.cross(P,R) #; PR = PR / PR[0, 2]
QS = np.cross(Q,S) #; QS = QS / QS[0, 2]
RS = np.cross(R,S) #; RS = RS / RS[0, 2]

# RS = PQ;
equation_matrix = np.matrix([[pq[0,0]*pr[0,0], (pq[0,1]*pr[0,0]+pq[0,0]*pr[0,1])/2,
                             [qs[0,0]*rs[0,0], (qs[0,1]*rs[0,0]+qs[0,0]*rs[0,1])/2,
                             [QS[0,0]*RS[0,0], (QS[0,1]*RS[0,0]+QS[0,0]*RS[0,1])/2,
```

```
                                [pq[0,0]*qs[0,0], (pq[0,1]*qs[0,0]+pq[0,0]*qs[0,1])/2,
                                [rs[0,0]*pr[0,0], (rs[0,1]*pr[0,0]+rs[0,0]*pr[0,1])/2,

b_equation = np.matrix([[ -pq[0,2]*pr[0,2] ],[ -qs[0,2]*rs[0,2] ],[ -PR[0,2]*RS[0,2

# equation_matrix = np.matrix([[PQ[0,0]*PR[0,0], (PQ[0,1]*PR[0,0]+PQ[0,0]*PR[0,1])/2
#                               [PR[0,0]*RS[0,0], (PR[0,1]*RS[0,0]+PR[0,0]*RS[0,1])/2
#                               [PQ[0,0]*PR[0,0], (PQ[0,1]*PR[0,0]+PQ[0,0]*PR[0,1])/2
#                               [RS[0,0]*QS[0,0], (RS[0,1]*QS[0,0]+RS[0,0]*QS[0,1])/2
#                               [rs[0,0]*pr[0,0], (rs[0,1]*pr[0,0]+rs[0,0]*pr[0,1])/2
#
# b_equation = np.matrix([[ -PQ[0,2]*PR[0,2] ],[ -PR[0,2]*RS[0,2] ],[ -PQ[0,2]*PR[0

ans_equation = np.linalg.inv(np.transpose(equation_matrix)*equation_matrix)*(np.tra
ans_equation = ans_equation / np.max(ans_equation)

SS = np.matrix([[ ans_equation[0,0], ans_equation[1,0]/2 ],[ ans_equation[1,0]/2, a

U,D,VT = np.linalg.svd(SS)

Ds = np.matrix([[np.sqrt(D[0]) , 0],[0 , np.sqrt(D[1])]])

A = np.transpose(VT) * Ds * VT

v = np.linalg.inv(A) * np.matrix([[ans_equation[3,0]/2], [ans_equation[4,0]/2]])

H = np.matrix([[A[0,0], A[0,1], 0], [A[1,0], A[1,1], 0], [v[0,0], v[1,0], 1]]); # H

final_P = np.transpose(np.linalg.inv(H) * np.transpose(img_p)); final_P = final_P /
final_Q = np.transpose(np.linalg.inv(H) * np.transpose(img_q)); final_Q = final_Q /
final_R = np.transpose(np.linalg.inv(H) * np.transpose(img_r)); final_R = final_R /
final_S = np.transpose(np.linalg.inv(H) * np.transpose(img_s)); final_S = final_S /


fmax_point = np.maximum(np.maximum(np.maximum(final_P,final_Q),final_R),final_S)
fmin_point = np.minimum(np.minimum(np.minimum(final_P,final_Q),final_R),final_S)

fimg_result = np.zeros((int(fmax_point[0,1]-fmin_point[0,1]), int(fmax_point[0,0]-fm

tmp = np.matrix('0 0 1', dtype=float)
for row in range(int(fmin_point[0,1]), int(fmax_point[0,1]) - 1):
    print(row)
    for col in range(int(fmin_point[0,0]), int(fmax_point[0,0]) - 1):
```

```
            tmp[0, 0] = col
            tmp[0, 1] = row
            tr = np.transpose(H * np.transpose(tmp))
            tr = tr / tr[0, 2]
            if tr[0, 0] > 0 and tr[0, 1] > 0 and tr[0, 1] < img1.shape[0] - 1 and tr[0,
                fimg_result[(row - int(fmin_point[0,1])), (col - int(fmin_point[0,0]))]


cv2.imwrite('Task_3_imgT2_result1.jpg', fimg_result)
```

6. Observation

1. After completing these three method to remove the projective and affine distortion, I think the point-to-point is the best method. It's quickly, accurately and easily to get a good result. I will describe the advantage and disadvantage for these three method below.

2. The Point-to-Point method: It's easily catch the point and calculation process is easily and accurately. I like this method. The homography matrix may be more difficult to calculate comparing the 2-Step and 1-Step Method because it need to solve a 3x3 matrix directly.

3. The 2-Step Method: The advantage: it's more robust than 1-Step method. Some points we can get good result in 2-Step Method but we can't in 1-Step method. The disadvantage: it need two steps to remove the projective and affine distortion. In this process, we also need calculate the vanishing line using two pairs of parallel lines.

4. The 1-Step Method: The advantage: It don't need too much calculation, just need 5 pairs of orthogonal lines in the physical space. The disadvantage: The points of the 5 pairs of orthogonal lines are very sensitive. We need catch them very carefully.

5. The 2-Step and 1-Step method are still include other distortion in the result image.