ECE 661 Computer Vision

Homework 3

Yi-Hong Liao | liao163@purdue.edu | PUID 0031850073

1. Logic

Find the homographies by using the point-to-point correspondence, two-steps method and one-steps method and apply the homographies to the distorted images.

2. Step

a. Estimating homographies using the point-to-point correspondence

Homography H can be express as

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}.$$

For each selected points pair (x, y) and (x', y') we have two equations

$$xh_{11} + yh_{12} + h_{13} - xx'h_{31} - yx'h_{32} - x'h_{33} = 0,$$
$$xh_{21} + yh_{22} + h_{23} - xy'h_{31} - yy'h_{32} - y'h_{33} = 0.$$

Because only the ratio of H maters, we set $h_{33} = 1$. Therefore, if we selected four points on the distorted image and its four corresponding points on the undistorted image based on the physical dimension, we can write down the equations as matrix

$$AX = B,$$

where

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_{21} & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix},$$

$$X = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}, B = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}.$$

The best estimate of X can be solved using least square

$$\hat{X} = (A^TA)^{-1}A^TB.$$

Then the homography H can be estimated.

b. Estimating homographies using the two-steps method

First, remove the projective distortion by moving VL back to I_∞. Select two parallel line pairs in real-world (I_1, I_2), (I_3, I_4) by taking the cross products of points on those lines. Find the vanishing point P and Q by taking the cross products of the two parallel line pairs. Solve the vanishing line by taking the cross product of P and Q.

$$P \times Q = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix}$$

The homography to remove projective distortion is

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

Then, remove the affine distortion. Select two perpendicular line pairs in real-world (L_1, M_1), (L_2, M_2) from the image with the projective distortion removed. Solve the homography to remove affine transform using

$$LH_a C_\infty^* {H_a}^T M = 0,$$

and the SVD method on the lecture notes.

c. Estimating homographies using the one-steps method

Find at least five physical perpendicular line pairs (l, m). Solve

$$l^T C^{*\prime} m = 0,$$

where

$$C^{*\prime} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix},$$

$$C^{*\prime} = H C_\infty^* H^T$$

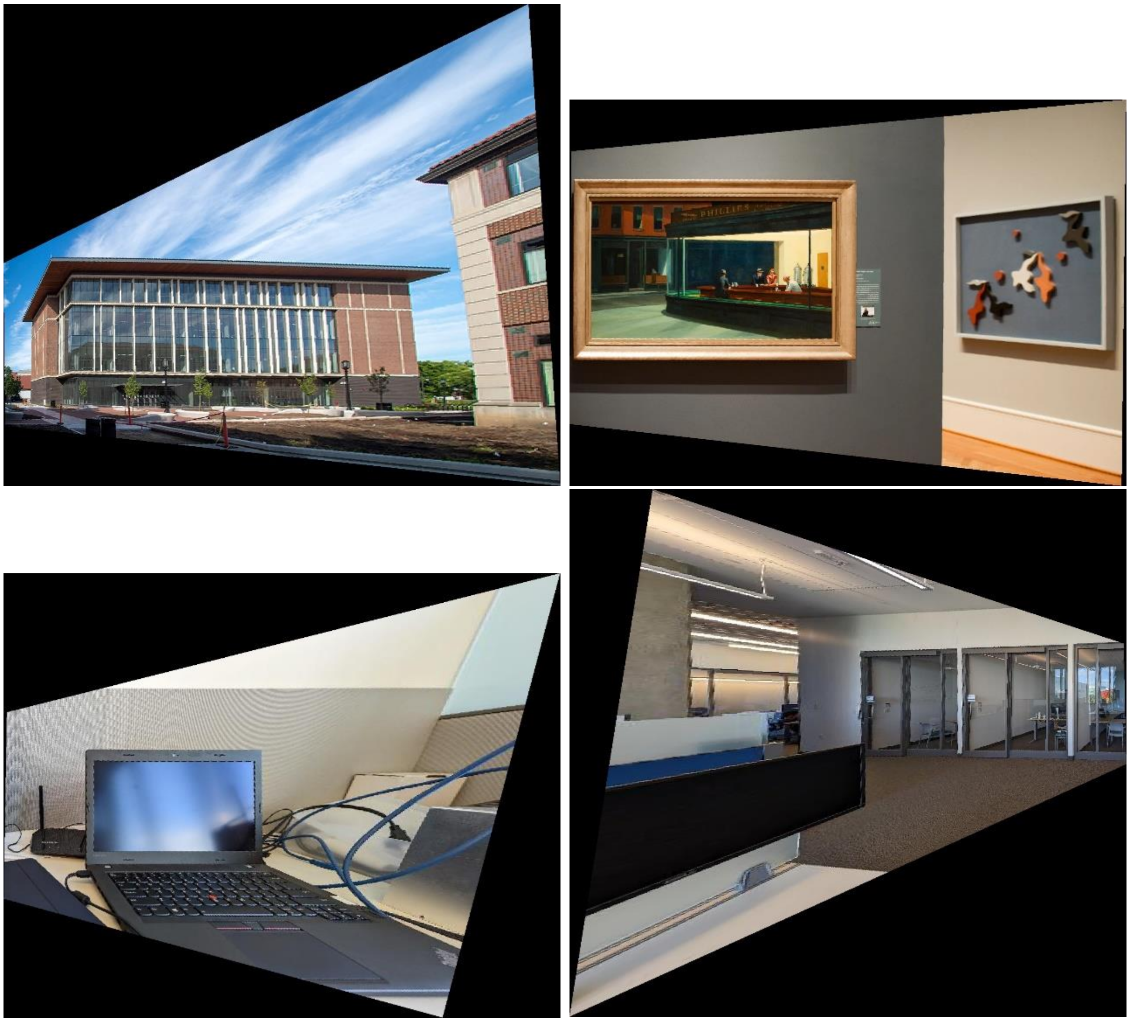
using the SVD approach on the lecture note and the equations on the homework sheet.

3. Result

Original Images (Input Images)

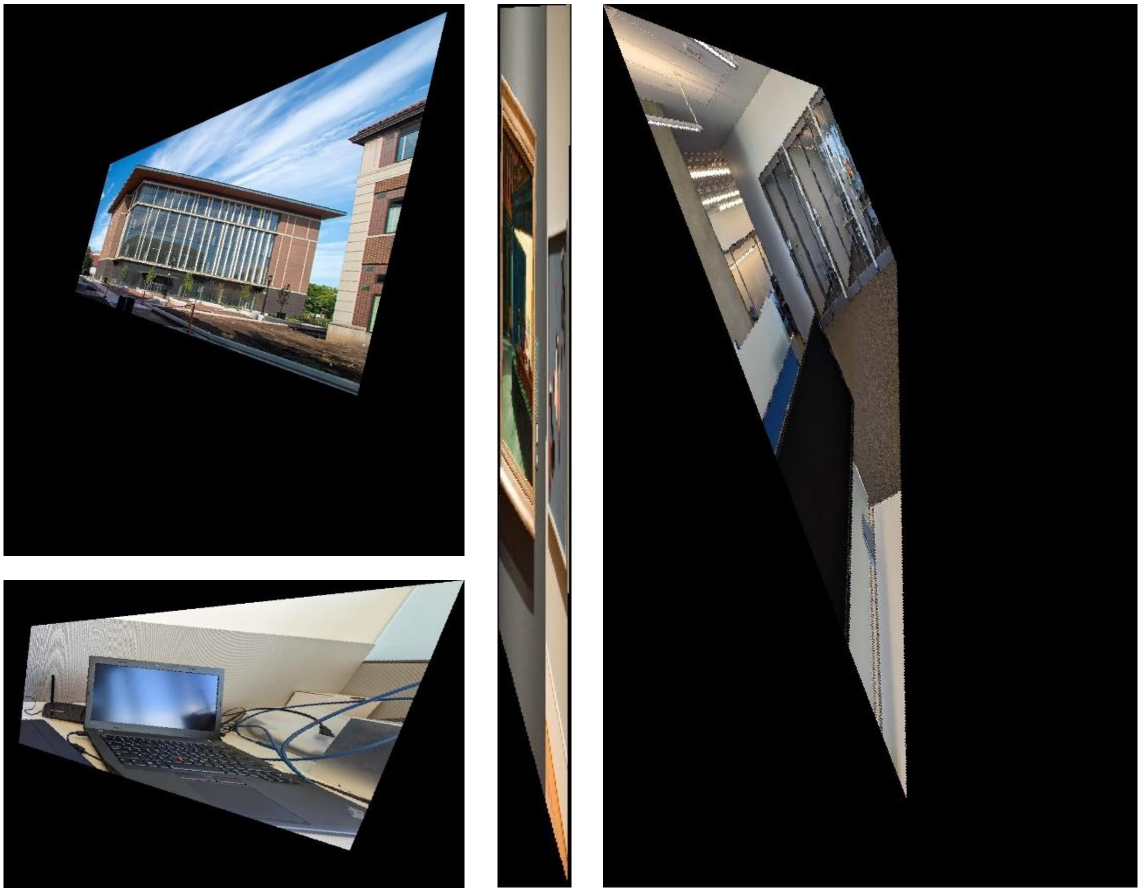
The red lines are the parallel / orthogonal lines I chose for homography calculations.

Point-to-point correspondence



The point-to-point correspondence method removes the projective and affine transform and gives the best result among other approaches.

Two-steps approach



The two-step approach works well on the building laptop and office images, but not on the Nighthawks image. This is due to the large condition number in the estimated matrix A.

One-step approach

The one-step approach generates decent results. The projective and the affine distortion are mostly removed. However, the resolution of the office image drops significantly. This is due to the error in the $C^{*\prime}$ matrix estimation.

4. Source Code

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:



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



# In[2]:



def find_homography(X, Xp):
    n = X.shape[0]
    A = np.zeros((n*2, 8))
    B = np.zeros((n*2, 1))
    for i in range(n):
        A[2*i][0] = X[i][0]
        A[2*i][1] = X[i][1]
        A[2*i][2] = 1
        A[2*i][6] = -X[i][0]*Xp[i][0]
        A[2*i][7] = -X[i][1]*Xp[i][0]
        A[2*i+1][3] = X[i][0]
        A[2*i+1][4] = X[i][1]
        A[2*i+1][5] = 1
        A[2*i+1][6] = -X[i][0]*Xp[i][1]
        A[2*i+1][7] = -X[i][1]*Xp[i][1]
        B[2*i] = Xp[i][0]
        B[2*i+1] = Xp[i][1]
```

```
    H = np.linalg.inv(A.transpose().dot(A)).dot(A.transpose()).dot(B)
    H = np.append(H, 1)
    H = H.reshape(3, 3)
    return H

def find_projective_homography(l1, m1, l3, m3):
    # remove projective distortion
    vp1 = np.cross(l1, l3)
    vp2 = np.cross(m1, m3)
    vl = np.cross(vp1/vp1[2], vp2/vp2[2])
    Hp = np.array([[1, 0, 0], [0, 1, 0], vl/vl[2]])
#    img_unprojective = create_transformed_image(img_building, ROI, Hp, 2)
#    cv2.imwrite(outputPath+'building_unprojective.jpg', img_unprojective)
    return Hp

def find_affine_homography(l1p, m1p, l3p, m3p):
    a = np.array([[l1p[0]*m1p[0], l1p[0]*m1p[1]+l1p[1]*m1p[0]],
                  [l3p[0]*m3p[0], l3p[0]*m3p[1]+l3p[1]*m3p[0]]])
    b = np.array([-l1p[1]*m1p[1],
                  -l3p[1]*m3p[1]])
    X = np.linalg.inv(a.transpose().dot(a)).dot(a.transpose()).dot(b)
    X = X/np.max(X)
    S = np.array([[X[0], X[1]],
                 [X[1], 1]])

    w, v = np.linalg.eig(S)
    w_ab = np.absolute(w)
    w_ab_diag = np.diag(w_ab)
    S_new = np.dot(np.dot(v, w_ab_diag), v.transpose())
#    S_new = S

    u, s, vh = np.linalg.svd(S_new, full_matrices=True)
```

```
    lambdas = np.sqrt(np.diag(s))
    A = np.dot(np.dot(u, lambdas), u.transpose())
    Ha = np.linalg.inv(np.array([[A[0, 0], A[0, 1], 0], [A[1, 0], A[1, 1], 0], [0, 0, 1]]))

    return Ha

def one_step_method(ls, ms):
    a = np.empty((0,5), float)
    b = np.empty((0,1), float)

    for i in range(5):
        l = ls[i]/ls[i, 2]
        m = ms[i]/ms[i, 2]
        a = np.append(a, [[l[0]*m[0], l[1]*m[0]+l[0]*m[1], l[1]*m[1], l[0]*m[2]+l[2]*m[0],
l[1]*m[2]+l[2]*m[1]]], axis=0)
        b = np.append(b, -l[2]*m[2])

    X = np.linalg.inv(a.transpose().dot(a)).dot(a.transpose()).dot(b)

    X = X/np.max(X)
    C = np.array([[X[0], X[1], X[3]], [X[1], X[2], X[4]], [X[3], X[4], 1]])

    w, v = np.linalg.eig(C)
    w_ab = np.absolute(w)
    w_ab_diag = np.diag(w_ab)
    C_new = np.dot(np.dot(v, w_ab_diag), v.transpose())

    u, s, vh = np.linalg.svd(C_new[0:2, 0:2], full_matrices=True)
    lambdas = np.sqrt(np.diag(s))

    AH = np.dot(np.dot(u, lambdas), u.transpose())
    V = np.linalg.inv(AH.transpose().dot(AH)).dot(AH.transpose()).dot(C_new[0:2, 2])
```

```
    H_one = np.linalg.inv(np.array([[AH[0, 0], AH[0, 1], 0], [AH[1, 0], AH[1, 1], 0], [V[0], V[1], 1]]))
    return H_one


def create_transformed_image(img, ROI_p, H, ds):
    h = img.shape[0]
    w = img.shape[1]

    new_ROI_p = np.zeros(shape=(ROI_p.shape[0], 3), dtype='int32')

    for i in range(ROI_p.shape[0]):
        pointH = np.array([ROI_p[i, 0], ROI_p[i, 1], 1])
        pointHp = H.dot(pointH)
        pointHp = pointHp/pointHp[2]
        pointHp = np.around(pointHp).astype(int)
        new_ROI_p[i, :] = pointHp

    max = np.amax(new_ROI_p, axis=0)
    min = np.amin(new_ROI_p, axis=0)

    new_ROI_p_offset = new_ROI_p
    new_ROI_p_offset[:, 0] = new_ROI_p_offset[:, 0]-min[0]
    new_ROI_p_offset[:, 1] = new_ROI_p_offset[:, 1]-min[1]

    ho = max[1]-min[1]+1
    wo = max[0]-min[0]+1
    print("new image height and width", ho, wo)
    imgOut = np.zeros([np.int32(np.floor(ho/ds))+1, np.int32(np.floor(wo/ds))+1, 3], dtype='uint8')
    new_ROI = np.zeros([ho, wo])
    cv2.fillPoly(new_ROI, pts = np.int32([new_ROI_p_offset[:, 0:2]]), color = 255)

    for i in range(0, wo, ds):
```

```
        for j in range(0, ho, ds):
            if new_ROI[j,i] > 0:
                pointH = np.array([i+min[0], j+min[1], 1])
                pointHp = np.linalg.inv(H).dot(pointH)
                pointHp = pointHp/pointHp[2]
                pointHp = np.around(pointHp).astype(int)
                if pointHp[0] < w and pointHp[0] >= 0 and pointHp[1] < h and pointHp[1] >= 0:
                    imgOut[np.int32(np.floor(j/ds)), np.int32(np.floor(i/ds))] = img[pointHp[1],
pointHp[0]]

    return imgOut



# In[3]:



if __name__ == '__main__':
    path = r'C:\Users\yhosc\Desktop\ECE661\HW3\hw3images\\'
    outputPath = r'C:\Users\yhosc\Desktop\ECE661\HW3\\'

    # Reading images in default mode
    img_building = cv2.imread(path+'building.jpg')
    img_nighthawks = cv2.imread(path+'nighthawks.jpg')
    img_laptop = cv2.imread(path+'laptop.jpg')
    img_office = cv2.imread(path+'office.jpg')

    # image building
    # corresponding points
    PQRS1_building = np.array([[241, 203, 1], [238, 370, 1], [295, 374, 1], [298, 217, 1]])
    PQRS2_building = np.array([[237, 196, 1], [232, 378, 1], [722, 411, 1], [719, 326, 1]])
    PQRS1_building_p = np.array([[241, 203, 1], [241, 373, 1], [298, 373, 1], [298, 203, 1]])

    l1 = np.cross(PQRS1_building[0], PQRS1_building[3])
```

```
    m1 = np.cross(PQRS1_building[0], PQRS1_building[1])
    l2 = np.cross(PQRS1_building[0], PQRS1_building[1])
    m2 = np.cross(PQRS1_building[1], PQRS1_building[2])
    l3 = np.cross(PQRS1_building[1], PQRS1_building[2])
    m3 = np.cross(PQRS1_building[2], PQRS1_building[3])
    l4 = np.cross(PQRS1_building[2], PQRS1_building[3])
    m4 = np.cross(PQRS1_building[0], PQRS1_building[3])
    l5 = np.cross(PQRS2_building[0], PQRS2_building[3])
    m5 = np.cross(PQRS2_building[0], PQRS2_building[1])

    ls = np.array([l1, l2, l3, l4, l5])
    ms = np.array([m1, m2, m3, m4, m5])

    ROI = np.array([[0, 0, 1], [0, 532, 1], [799, 532, 1], [799, 0, 1]])

    # Using point-to-point correspondences
    H_point = find_homography(PQRS1_building, PQRS1_building_p)

    img_undistorted1 = create_transformed_image(img_building, ROI, H_point, 1)
    cv2.imwrite(outputPath+'building_undistorted1.jpg', img_undistorted1)

#     # debug
#     cv2.imshow('image', img_undistorted1)
#     cv2.waitKey(0)
#     cv2.destroyAllWindows()


# In[4]:


# Using two-step method
Hp = find_projective_homography(l1, m1, l3, m3)
```

```
img_unprojective = create_transformed_image(img_building, ROI, Hp, 2)
cv2.imwrite(outputPath+'building_unprojective.jpg', img_unprojective)

ROI_p = np.array([[0, 0, 1], [0, 822, 1], [1235, 822, 1], [1235, 0, 1]])
l1p = np.cross([152, 128, 1], [200, 145, 1])
m1p = np.cross([152, 128, 1], [148, 232, 1])
l3p = np.cross([148, 232, 1], [197, 250, 1])
m3p = np.cross([197, 250, 1], [200, 145, 1])

Ha = find_affine_homography(l1p, m1p, l3p, m3p)
img_undistorted2 = create_transformed_image(img_unprojective, ROI_p, Ha, 1)
cv2.imwrite(outputPath+'building_undistorted2.jpg', img_undistorted2)
 # debug
#     cv2.imshow('image', img_undistorted2)
#     cv2.waitKey(0)
#     cv2.destroyAllWindows()



# In[5]:



# Using one-step method
H_one = one_step_method(ls, ms)
img_undistorted3 = create_transformed_image(img_building, ROI, H_one, 5)
cv2.imwrite(outputPath+'building_undistorted3.jpg', img_undistorted3)

#     # debug
#     cv2.imshow('image', img_undistorted3)
#     cv2.waitKey(0)
#     cv2.destroyAllWindows()
```

```
# In[6]:



# image nighthawks
# corresponding points
PQRS1_nighthawks = np.array([[76, 182, 1], [79, 653, 1], [807, 620, 1], [804, 220, 1]])
PQRS2_nighthawks = np.array([[13, 99, 1], [12, 727, 1], [865, 676, 1], [862, 160, 1]])
PQRS1_nighthawks_p = np.array([[76, 182, 1], [76, 652, 1], [906, 652, 1], [906, 182, 1]])

l1 = np.cross(PQRS1_nighthawks[0], PQRS1_nighthawks[3])
m1 = np.cross(PQRS1_nighthawks[0], PQRS1_nighthawks[1])
l2 = np.cross(PQRS1_nighthawks[0], PQRS1_nighthawks[1])
m2 = np.cross(PQRS1_nighthawks[1], PQRS1_nighthawks[2])
l3 = np.cross(PQRS1_nighthawks[1], PQRS1_nighthawks[2])
m3 = np.cross(PQRS1_nighthawks[2], PQRS1_nighthawks[3])
l4 = np.cross(PQRS1_nighthawks[2], PQRS1_nighthawks[3])
m4 = np.cross(PQRS1_nighthawks[0], PQRS1_nighthawks[3])
l5 = np.cross(PQRS2_nighthawks[0], PQRS2_nighthawks[3])
m5 = np.cross(PQRS2_nighthawks[0], PQRS2_nighthawks[1])

ls = np.array([l1, l2, l3, l4, l5])
ms = np.array([m1, m2, m3, m4, m5])

ROI = np.array([[0, 0, 1], [0, 957, 1], [1439, 957, 1], [1439, 0, 1]])

# Using point-to-point correspondences
H_point = find_homography(PQRS1_nighthawks, PQRS1_nighthawks_p)

img_undistorted1 = create_transformed_image(img_nighthawks, ROI, H_point, 1)
cv2.imwrite(outputPath+'nighthawks_undistorted1.jpg', img_undistorted1)
```

```
# In[7]:



# Using two-step method
Hp = find_projective_homography(l1, m1, l3, m3)
img_unprojective = create_transformed_image(img_nighthawks, ROI, Hp, 2)
cv2.imwrite(outputPath+'nighthawks_unprojective.jpg', img_unprojective)

ROI_p = np.array([[0, 0, 1], [0, 675, 1], [1019, 675, 1], [1019, 0, 1]])
l1p = np.cross([38, 93, 1], [481, 132, 1])
m1p = np.cross([38, 93, 1], [40, 329, 1])
l3p = np.cross([40, 329, 1], [480, 370, 1])
m3p = np.cross([480, 370, 1], [480, 132, 1])

Ha = find_affine_homography(l1p, m1p, l3p, m3p)
img_undistorted2 = create_transformed_image(img_unprojective, ROI_p, Ha, 1)
cv2.imwrite(outputPath+'nighthawks_undistorted2.jpg', img_undistorted2)


# In[8]:



# Using one-step method
H_one = one_step_method(ls, ms)
img_undistorted3 = create_transformed_image(img_nighthawks, ROI, H_one, 5)
cv2.imwrite(outputPath+'nighthawks_undistorted3.jpg', img_undistorted3)


# In[9]:



# image laptop
```

```
# corresponding points

PQRS1_laptop = np.array([[235, 151, 1], [247, 355, 1], [504, 325, 1], [489, 175, 1]])
PQRS2_laptop = np.array([[215, 123, 1], [233, 388, 1], [517, 346, 1], [496, 158, 1]])
PQRS1_laptop_p = np.array([[235, 151, 1], [235, 331, 1], [555, 331, 1], [555, 151, 1]])

l1 = np.cross(PQRS1_laptop[0], PQRS1_laptop[3])
m1 = np.cross(PQRS1_laptop[0], PQRS1_laptop[1])
l2 = np.cross(PQRS1_laptop[0], PQRS1_laptop[1])
m2 = np.cross(PQRS1_laptop[1], PQRS1_laptop[2])
l3 = np.cross(PQRS1_laptop[1], PQRS1_laptop[2])
m3 = np.cross(PQRS1_laptop[2], PQRS1_laptop[3])
l4 = np.cross(PQRS1_laptop[2], PQRS1_laptop[3])
m4 = np.cross(PQRS1_laptop[0], PQRS1_laptop[3])
l5 = np.cross(PQRS2_laptop[0], PQRS2_laptop[3])
m5 = np.cross(PQRS2_laptop[0], PQRS2_laptop[1])

ls = np.array([l1, l2, l3, l4, l5])
ms = np.array([m1, m2, m3, m4, m5])

ROI = np.array([[0, 0, 1], [0, 528, 1], [701, 528, 1], [701, 0, 1]])

# Using point-to-point correspondences
H_point = find_homography(PQRS1_laptop, PQRS1_laptop_p)

img_undistorted1 = create_transformed_image(img_laptop, ROI, H_point, 1)
cv2.imwrite(outputPath+'laptop_undistorted1.jpg', img_undistorted1)



# In[10]:
```

```
# Using two-step method
Hp = find_projective_homography(l1, m1, l3, m3)
img_unprojective = create_transformed_image(img_laptop, ROI, Hp, 2)
cv2.imwrite(outputPath+'laptop_unprojective.jpg', img_unprojective)

ROI_p = np.array([[0, 0, 1], [0, 538, 1], [892, 538, 1], [892, 0, 1]])
l1p = np.cross([143, 91, 1], [402, 145, 1])
m1p = np.cross([143, 91, 1], [145, 207, 1])
l3p = np.cross([145, 207, 1], [405, 261, 1])
m3p = np.cross([405, 261, 1], [402, 145, 1])

Ha = find_affine_homography(l1p, m1p, l3p, m3p)
img_undistorted2 = create_transformed_image(img_unprojective, ROI_p, Ha, 1)
cv2.imwrite(outputPath+'laptop_undistorted2.jpg', img_undistorted2)


# In[11]:


# Using one-step method
H_one = one_step_method(ls, ms)
img_undistorted3 = create_transformed_image(img_laptop, ROI, H_one, 5)
cv2.imwrite(outputPath+'laptop_undistorted3.jpg', img_undistorted3)


# In[12]:


# image office
# corresponding points

PQRS1_office = np.array([[235, 139, 1], [240, 350, 1], [304, 367, 1], [300, 126, 1]])
```

```
PQRS2_office = np.array([[232, 127, 1], [236, 323, 1], [455, 432, 1], [455, 64, 1]])
PQRS1_office_p = np.array([[235, 139, 1], [235, 400, 1], [340, 400, 1], [340, 139, 1]])

l1 = np.cross(PQRS1_office[0], PQRS1_office[3])
m1 = np.cross(PQRS1_office[0], PQRS1_office[1])
l2 = np.cross(PQRS1_office[0], PQRS1_office[1])
m2 = np.cross(PQRS1_office[1], PQRS1_office[2])
l3 = np.cross(PQRS1_office[1], PQRS1_office[2])
m3 = np.cross(PQRS1_office[2], PQRS1_office[3])
l4 = np.cross(PQRS1_office[2], PQRS1_office[3])
m4 = np.cross(PQRS1_office[0], PQRS1_office[3])
l5 = np.cross(PQRS2_office[0], PQRS2_office[3])
m5 = np.cross(PQRS2_office[0], PQRS2_office[1])

ls = np.array([l1, l2, l3, l4, l5])
ms = np.array([m1, m2, m3, m4, m5])

ROI = np.array([[0, 0, 1], [0, 528, 1], [701, 528, 1], [701, 0, 1]])

# Using point-to-point correspondences
H_point = find_homography(PQRS1_office, PQRS1_office_p)

img_undistorted1 = create_transformed_image(img_office, ROI, H_point, 1)
cv2.imwrite(outputPath+'office_undistorted1.jpg', img_undistorted1)



# In[13]:



# Using two-step method
Hp = find_projective_homography(l1, m1, l3, m3)
img_unprojective = create_transformed_image(img_office, ROI, Hp, 1)
```

```
cv2.imwrite(outputPath+'office_unprojective.jpg', img_unprojective)

ROI_p = np.array([[0, 0, 1], [0, 635, 1], [183, 635, 1], [183, 0, 1]])
l1p = np.cross([120, 71, 1], [134, 56, 1])
m1p = np.cross([120, 71, 1], [126, 181, 1])
l3p = np.cross([126, 181, 1], [139, 168, 1])
m3p = np.cross([139, 168, 1], [134, 56, 1])

Ha = find_affine_homography(l1p, m1p, l3p, m3p)
img_undistorted2 = create_transformed_image(img_unprojective, ROI_p, Ha, 1)
cv2.imwrite(outputPath+'office_undistorted2.jpg', img_undistorted2)


# In[14]:


# Using one-step method
H_one = one_step_method(ls, ms)
img_undistorted3 = create_transformed_image(img_office, ROI, H_one, 1)
cv2.imwrite(outputPath+'office_undistorted3.jpg', img_undistorted3)
```