ECE 661 Computer Vision

Homework 2

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

1. Logic

Manually select four points on each image and solve the homographies between them.

2. Step

a. Estimating projective homographies

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, 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 affine homographies

Affine homography H can be express as

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & h_{33} \end{bmatrix}.$$

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

$$x h_{11} + y h_{12} + h_{13} - x' h_{33} = 0,$$
$$x h_{21} + y h_{22} + h_{23} - y' h_{33} = 0.$$

Because only the ratio of H maters, we set $h_{33} = 1$. Therefore, if we selected four points, we could write down the equations as matrix

$$AX = B,$$

where

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_{21} & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \\ x_4 & y_4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_4 & y_4 & 1 \end{bmatrix},$$

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

The best estimate of X can be solved using least square

$$\hat{X} = (A^T A)^{-1} A^T B$$

Then the affine homography H can be estimated.

c. Applying homographies

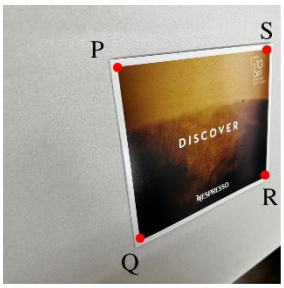
$$\mathrm{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}.$$

3. Result

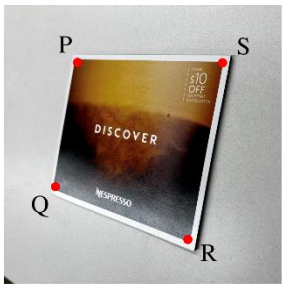
Task 1.1

The ROI of the car image is the entire image. The red dots are the points I use to obtain the homopraphies.

Input images:



(a)



(b)

(c)

(d)

Output images:

(a)

(b)

(c)

Task 1.2

Task 1.3

(a) (b) (c)

(c) has the better results than the others because the original image has almost only affine deformation.

Task 2.1

The ROI of the car image is the entire image. The red dots are the points I use to obtain the homopraphies.

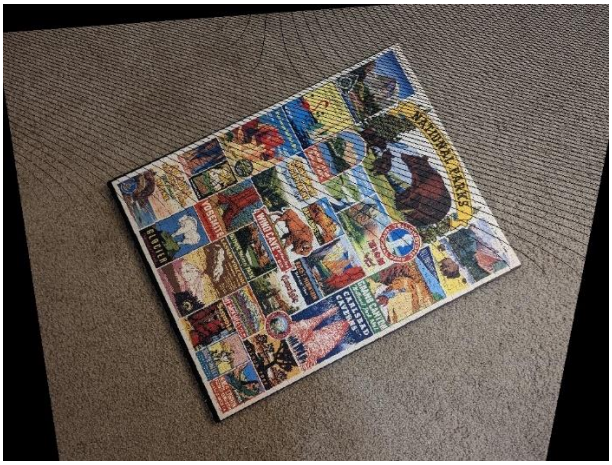
Input images:

(a) (b) (c) (d)

Output images:

(a) (b) (c)

Task 2.2



Task 2.3

(a)

(b)

(c)

(c) has the better results than the others because the original image has almost only affine deformation.

4. Source Code

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

# In[17]:


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


# In[18]:


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_affine_homography(X, Xp):
    n = X.shape[0]
    A = np.zeros((n*2, 6))
    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+1][3] = X[i][0]
        A[2*i+1][4] = X[i][1]
        A[2*i+1][5] = 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, [0, 0, 1])
    H = H.reshape(3, 3)
    return H

def apply_transform(img1, img2, ROI, H):
    h1 = img1.shape[0]
    w1 = img1.shape[1]
    h2 = img2.shape[0]
    w2 = img2.shape[1]
    imgOut = img1.copy()
    for i in range(w2):
        for j in range(h2):
            if ROI[j][i] > 0:
```

```
            pointH = np.array([i, j, 1])
            pointH2 = H.dot(pointH)
            pointH2 = pointH2/pointH2[2]
            pointH2 = np.around(pointH2).astype(int)
            if pointH2[0] < w1 and pointH2[0] >= 0 and pointH2[1] < h1 and pointH2[1] >= 0:
                imgOut[pointH2[1], pointH2[0]] = img2[j, i]
    return imgOut

if __name__ == '__main__':
    path = r'C:\Users\yhosc\Desktop\ECE661\hw2\hw2images\\'
    outputPath = r'C:\Users\yhosc\Desktop\ECE661\hw2\\'

    # Reading images in default mode
    img_card1 = cv2.imread(path+'card1.jpeg')
    img_card2 = cv2.imread(path+'card2.jpeg')
    img_card3 = cv2.imread(path+'card3.jpeg')
    img_car = cv2.imread(path+'car.jpg')

    # Array of points
    X = np.array([[0, 0],
                  [0, 558],
                  [760, 558],
                  [760, 0]])

    Xp1 = np.array([[526, 284],
                  [631, 1082],
                  [1210, 787],
                  [1228, 203]])

    Xp2 = np.array([[327, 250],
                  [225, 843],
                  [855, 1089],
```

```
                [1010, 261]])

    Xp3 = np.array([[584, 80],
                [94, 588],
                [704, 1179],
                [1195, 675]])

    ROI = np.zeros([img_car.shape[0], img_car.shape[1]])
    cv2.fillPoly(ROI, pts = [X], color = 255)


# In[19]:


# Task 1.1
H1c = find_homography(X, Xp1)
H2c = find_homography(X, Xp2)
H3c = find_homography(X, Xp3)
img1c = apply_transform(img_card1, img_car, ROI, H1c)
img2c = apply_transform(img_card2, img_car, ROI, H2c)
img3c = apply_transform(img_card3, img_car, ROI, H3c)

cv2.imwrite(outputPath+'card1c.jpg', img1c)
cv2.imwrite(outputPath+'card2c.jpg', img2c)
cv2.imwrite(outputPath+'card3c.jpg', img3c)


# In[20]:


# Task 1.2
ROI1 = np.ones([img_card1.shape[0], img_card1.shape[1]])*255
```

```
H12 = find_homography(Xp1, Xp2)
H23 = find_homography(Xp2, Xp3)
H13 = H23.dot(H12)
blank_image = np.zeros(img_card3.shape, dtype='uint8')
img31 = apply_transform(blank_image, img_card1, ROI1, H13)
cv2.imwrite(outputPath+'card31.jpg', img31)


# In[21]:


# Task 1.3
H1c_affine = find_affine_homography(X, Xp1)
H2c_affine = find_affine_homography(X, Xp2)
H3c_affine = find_affine_homography(X, Xp3)
img1c_affine = apply_transform(img_card1, img_car, ROI, H1c_affine)
img2c_affine = apply_transform(img_card2, img_car, ROI, H2c_affine)
img3c_affine = apply_transform(img_card3, img_car, ROI, H3c_affine)

cv2.imwrite(outputPath+'card1c_affine.jpg', img1c_affine)
cv2.imwrite(outputPath+'card2c_affine.jpg', img2c_affine)
cv2.imwrite(outputPath+'card3c_affine.jpg', img3c_affine)


# In[22]:


# Reading images in default mode
img_puzzle1 = cv2.imread(path+'puzzle1.jpg')
img_puzzle2 = cv2.imread(path+'puzzle2.jpg')
img_puzzle3 = cv2.imread(path+'puzzle3.jpg')
img_f16 = cv2.imread(path+'f16.png')
```

```
# Array of points
X = np.array([[0, 0],
              [0, 742],
              [1117, 742],
              [1117, 0]])

Xp1 = np.array([[179, 229],
              [362, 670],
              [915, 382],
              [673, 51]])

Xp2 = np.array([[257, 139],
              [146, 541],
              [763, 650],
              [799, 104]])

Xp3 = np.array([[183, 299],
              [364, 698],
              [873, 437],
              [665, 67]])

ROI = np.zeros([img_f16.shape[0], img_f16.shape[1]])
cv2.fillPoly(ROI, pts = [X], color = 255)


# In[23]:


# Task 2.1
H1f = find_homography(X, Xp1)
H2f = find_homography(X, Xp2)
```

```
H3f = find_homography(X, Xp3)
img1f = apply_transform(img_puzzle1, img_f16, ROI, H1f)
img2f = apply_transform(img_puzzle2, img_f16, ROI, H2f)
img3f = apply_transform(img_puzzle3, img_f16, ROI, H3f)

cv2.imwrite(outputPath+'puzzle1f.jpg', img1f)
cv2.imwrite(outputPath+'puzzle2f.jpg', img2f)
cv2.imwrite(outputPath+'puzzle3f.jpg', img3f)


# In[24]:


# Task 1.2
ROI1 = np.ones([img_puzzle1.shape[0], img_puzzle1.shape[1]])*255
H12 = find_homography(Xp1, Xp2)
H23 = find_homography(Xp2, Xp3)
H13 = H23.dot(H12)
blank_image = np.zeros(img_puzzle1.shape, dtype='uint8')
img31 = apply_transform(blank_image, img_puzzle1, ROI1, H13)
cv2.imwrite(outputPath+'puzzle31.jpg', img31)


# In[25]:


# Task 1.3
H1f_affine = find_affine_homography(X, Xp1)
H2f_affine = find_affine_homography(X, Xp2)
H3f_affine = find_affine_homography(X, Xp3)
img1f_affine = apply_transform(img_puzzle1, img_f16, ROI, H1f_affine)
img2f_affine = apply_transform(img_puzzle2, img_f16, ROI, H2f_affine)
```

```
img3f_affine = apply_transform(img_puzzle3, img_f16, ROI, H3f_affine)

cv2.imwrite(outputPath+'puzzle1f_affine.jpg', img1f_affine)
cv2.imwrite(outputPath+'puzzle2f_affine.jpg', img2f_affine)
cv2.imwrite(outputPath+'puzzle3f_affine.jpg', img3f_affine)
```