ECE 661 Homework 10

Weichen Xu

xu1363@purdue.edu

Estimating the Fundamental Matrix

We need to manually obtain at least 8 pairs of correspondence points of two images to estimate the fundamental matrix.

To obtain the fundamental matrix F, the following equation should be solved using linear least square method and SVD.

$$Af = 0$$

Where A is in the form of n pairs of correspondences, as shown below

$$A = [x_i'x_i \quad x_i'y_i \quad x_i' \quad y_i'x_i \quad y_i'y_i \quad x_i \quad y_i \quad 1]$$

$$f = [F_{11} \quad F_{12} \quad F_{13} \quad F_{21} \quad F_{22} \quad F_{23} \quad F_{31} \quad F_{32} \quad F_{33}]$$

Solve f by using SVD and obtain the eigen vector with minimum eigen value. Since we need to restrict the rank of F is 2, we need to set the last eigen value to be 0, and then obtain the new F matrix using SVD.

To obtain the transformation matrix for the two images by the uncalibrated camera. Correspondence points in each image are t normalized to the zero mean and with a standard deviation of $\sqrt{2}$. Then the transformation matrix T can be denoted as

$$T = \begin{bmatrix} s & 0 & x \\ 0 & s & y \\ 0 & 0 & 1 \end{bmatrix}$$

We denote the mean value of x, y coordinates as $\bar{x}$, $\bar{y}$, we have following

$$s = \frac{\sqrt{2}}{\sum_{i=1}^{n} \sqrt{(\bar{x} - x_i)^2 + (\bar{y} - y_i)^2}/n}$$

$$x = -s * \bar{x}$$

$$y = -s * \bar{y}$$

The final estimate of the fundamental matrix F can be obtained by

$$F = T_2^T F T_1$$

Image Rectification

The image rectification process first rotates the second image with T_1 transformation matrix, then the epipole is sent to x-axis by apply R rotation matrix. The epipole is further sent to infinity by matrix G. Finally applying T_2 matrix will move the epipole back to the center. The overall homography matrix is expressed as

$$H_2 = T_2 G R T_1$$

To obtain the homography matrix for the first image, we need to use linear least square method to minimize the sum of distances by

$$\min_{H_1} \sum d(H_1 x_i, H_2 x_i')$$

Experiment Results

Task 1

Figure 1: Correspondence points on the image 1

Figure 2: Correspondence points on the image 2

Task 2

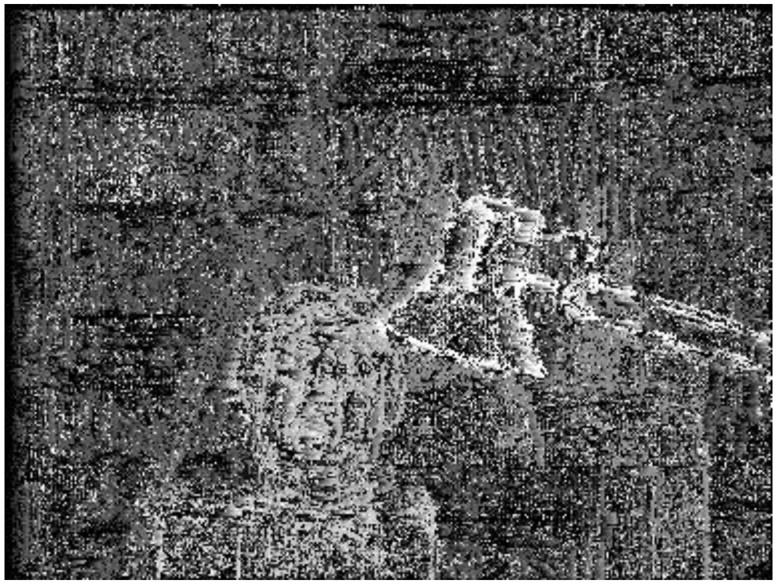

Figure 3: Disparity map when $M = 3$

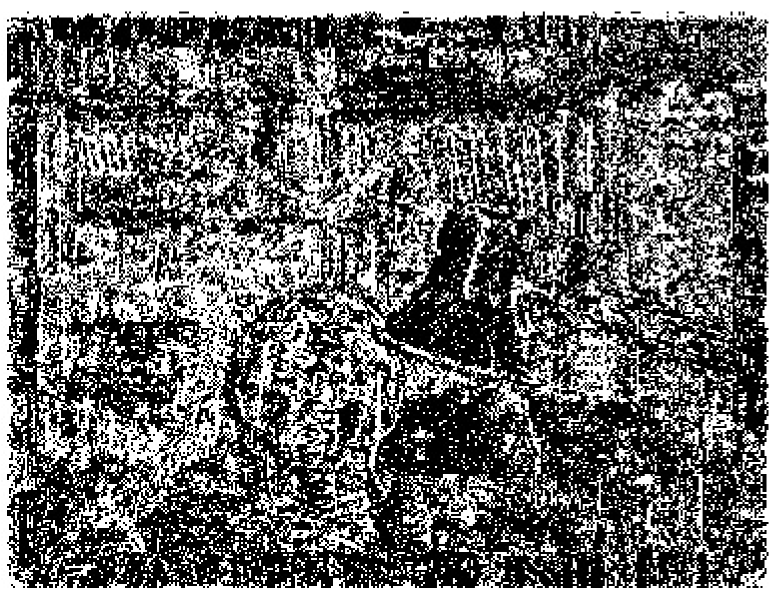

Figure 4: Binary error mask when $M = 3$ (accuracy is 36.8%)

Figure 5: Disparity map when $M = 7$

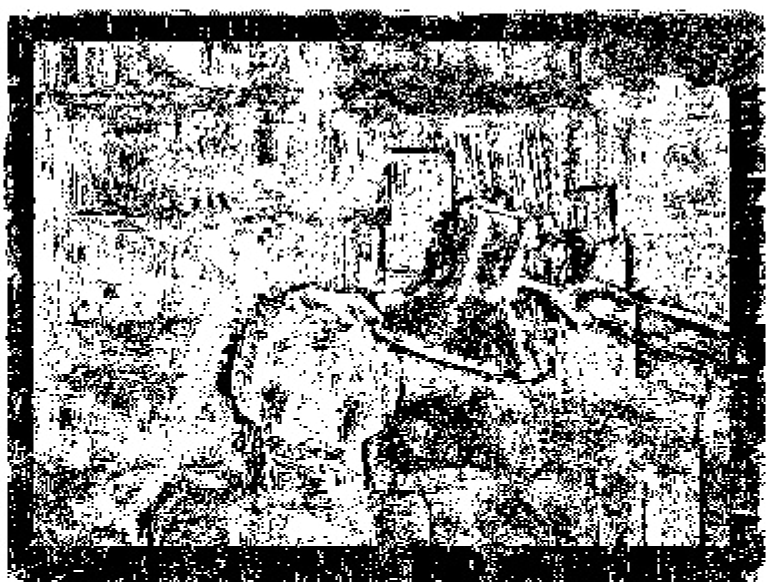

Figure 6: Binary error mask when $M = 7$ (accuracy is 56.1%)

Observation

From the experiment, I increase the M value from 3 to as large as 11, the large the M value, the better quality of the disparity map, while it also takes much more time since the $M * M$ area is expanded significantly. With the M value increases, it gets closer to the given ground truth disparity map.

Code

```
import numpy as np
import cv2
from matplotlib.pyplot import imread
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.linalg import null_space


def get_fundamental_F(pts1, pts2):
    N = len(pts1)
    A = np.zeros((N,9))
    for i in range(N):
        x1, y1 = pts1[i][0], pts1[i][1]
        x2, y2 = pts2[i][0], pts2[i][1]
        A[i] = [x2*x1, x2*y1, x2, y2*x1, y2*y1, y2, x1, y1, 1]

    # use SVD to obtain the f then reshape to 3*3 F
    u,d,v_t = np.linalg.svd(A)
    v = v_t.T
    F = v[:,v.shape[1]-1]
    F = F.reshape(3,3)

    # restrict the rank of F to be 2 by setting last eigenvalue to be 0
    u,d,v_t = np.linalg.svd(F)
    D = np.array([[d[0],0,0],[0,d[1],0],[0,0,0]])
    F = np.dot(u, np.dot(D, v_t))

    T1 = get_T(pts1)
    T2 = get_T(pts2)
    F = np.dot(T2.T, np.dot(F, T1))
    #F = F / F[-1,-1]
    return F

def get_T(pts):
    # obtain the transformation matrix T by the correspondence points
    pts = np.array(pts)
    x = pts[:,0]
    y = pts[:,1]
    avg_x = np.mean(x)
    avg_y = np.mean(y)
    square_x = np.square(x-avg_x)
    square_y = np.square(y-avg_y)

    mean = np.sum(np.sqrt(np.add(square_x,square_y))) / len(pts)
    scale = np.sqrt(2)/mean
    x0 = -1*scale*avg_x
```

```python
    y0 = -1*scale*avg_y
    T = np.array([[scale, 0, x0], [0, scale, y0], [0, 0, 1]])
    #print(T)
    return T


def mapping(img_target,H):
    # Mapping the image using homography matrix H
    P_distort = np.array([0,0,1])
    Q_distort = np.array([0,img_target.shape[0]-1,1])
    R_distort = np.array([img_target.shape[1]-1,img_target.shape[0]-1,1])
    S_distort = np.array([img_target.shape[1]-1,0,1])


    P_world = np.matmul(H,P_distort)
    P_world = P_world / P_world[2]
    Q_world = np.matmul(H,Q_distort)
    Q_world = Q_world / Q_world[2]
    R_world = np.matmul(H,R_distort)
    R_world = R_world / R_world[2]
    S_world = np.matmul(H,S_distort)
    S_world = S_world / S_world[2]

    xmin = np.int32(np.round(np.amin([P_world[0],Q_world[0],R_world[0],S_world[0]])))
    xmax = np.int32(np.ceil(np.amax([P_world[0],Q_world[0],R_world[0],S_world[0]])))
    ymin = np.int32(np.round(np.amin([P_world[1],Q_world[1],R_world[1],S_world[1]])))
    ymax = np.int32(np.ceil(np.amax([P_world[1],Q_world[1],R_world[1],S_world[1]])))

    xlen = xmax-xmin
    ylen = ymax-ymin

    img_new = np.zeros((ylen,xlen,3), dtype=np.uint8)
    print('The output image size is',xlen,ylen)
    Hinv = np.linalg.inv(H)

    for i in range(xlen):
        for j in range(ylen):
            input = np.array([i+xmin,j+ymin,1])
            output = np.matmul(Hinv,input)
            x = np.int(np.round(output[0]/output[2]))
            y = np.int(np.round(output[1]/output[2]))

            if x>0 and x<img_target.shape[1]-1 and y>0 and y<img_target.shape[0]-1:
                img_new[j,i,:] = img_target[y,x,:]
    return img_new


def img_rectify(img1, img2, pts1, pts2, F):
    h, w = img1.shape[0], img1.shape[1]

    # get the null vector from the F matrix
    e = null_space(F)
    e /= e[2]
    ep = null_space(F.T)
    ep /= ep[2]
```

```
    # obtain the second image's homography matrix H2
    theta = np.arctan(-1*(ep[1] - h/2)/(ep[0] - w/2))
    theta = theta[0]
    f = np.cos(theta)*(ep[0] - w/2) - np.sin(theta) * (ep[1] - h/2)
    G = np.array([[1,0,0],[0,1,0],[-1/f, 0, 1]], dtype=np.float)
    R = np.array([[np.cos(theta), -1*np.sin(theta), 0], [np.sin(theta), np.cos(theta), 0], [0,0,1]], dtype=np.float)
    T1 = np.array([[1,0, -1*w/2], [0,1, -1*h/2], [0,0,1]], dtype=np.float)
    H2 = np.dot(T1, np.dot(G, R))

    center = np.array([w/2, h/2, 1])
    center_shift = np.dot(H2, center)
    center_shift /= center_shift[2]
    T2 = np.array([[1, 0, w/2 - center_shift[0]], [0, 1, w/2 - center_shift[1]], [0,0,1]], dtype=np.float)
    H2 = np.dot(T2, H2)


def plot_corner(img, pts):
    img_plot = img.copy()

    for i in range(len(pts)):
        loc = tuple([pts[i][0], pts[i][1]])
        cv2.circle(img_plot, loc, 2, (0,0,255), 30)
        cv2.putText(img_plot, str(i+1),loc, cv2.FONT_HERSHEY_COMPLEX, 3, (0,0,255), 1)

    return img_plot



path = '/home/xu1363/Documents/ECE 661/hw10/Task1_Images/'
file1 = 'img1.jpg'
file2 = 'img2.jpg'

img1 = imread(path + file1)
img2 = imread(path + file2)

pts1 =
[[160,695],[2264,2121],[2839,1488],[2968,1339],[3344,947],[1422,114],[489,1363],[2223,2667]]#,[3120,15
75]]
pts2 =
[[499,721],[1973,2368],[2815,1722],[3015,1553],[3627,1119],[1806,138],[824,1315],[2127,2883]]#,[3431,1
799]]


img_plot1 = plot_corner(img1, pts1)
cv2.imwrite(path + 'img1_corners.jpg', img_plot1)
img_plot2 = plot_corner(img2, pts2)
cv2.imwrite(path + 'img2_corners.jpg', img_plot2)

F = get_fundamental_F(pts1, pts2)
img_rectify(img1, img2, pts1, pts2, F)
```

```python
# Task 2

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



def block_xor(block1, block2):
    center1 = block1[1, 1]
    center2 = block2[1, 1]
    line1 = block1.flatten()
    line2 = block2.flatten()

    for i in range(len(line1)):
        if center1 < line1[i]:
            line1[i] = 1
        else: line1[i] = 0

    for i in range(len(line2)):
        if center2 < line2[i]:
            line2[i] = 1
        else: line2[i] = 0

    cost = 0
    for i in range(len(line1)):
        if line1[i] != line2[i]:
            cost += 1

    return cost


file_truth = 'left_truedisp.pgm'
img_truth = plt.imread(path+file_truth)
img_truth = np.array(img_truth, dtype = np.float32)
img_truth /= 16
img_truth = np.array(img_truth, dtype = np.int16)
print(np.max(img_truth))

plt.imshow(img_truth, cmap = 'gray')
print(img_truth.shape)


path = '/home/xu1363/Documents/ECE 661/hw10/Task2_Images/'
file1 = 'Left.ppm'
file2 = 'Right.ppm'

img1 = plt.imread(path+file1)
img2 = plt.imread(path+file2)
gray1 = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)
```

```python
dmax = 14
M = 3
edge = int((M-1)/2)


h = img1.shape[0]
w = img1.shape[1]

img1new = np.zeros((h+2*edge, w+2*edge))
img2new = np.zeros((h+2*edge, w+2*edge))
img1new[edge:-edge,edge:-edge] = gray1
img2new[edge:-edge,edge:-edge] = gray2

disparity_map = np.zeros((h, w))
for j in range(h):
  for i in range(w):
    block1 = img1new[j:j+M, i:i+M]
    candidate = 0
    cost_min = 100

    for k in range(dmax+1):
      i2 = i - k
      if i2 > 0:
        block2 = img2new[j:j+M, i2:i2+M]
        cost = block_xor(block1, block2)
        if cost < cost_min:
          candidate = k
          cost_min = cost
    disparity_map[j, i] = candidate


img = np.array(disparity_map, dtype = np.float32)
img = img / np.max(img) * 255
img = np.array(img, dtype = np.uint16)

plt.imshow(disparity_map, cmap = 'gray')

cv2.imwrite(path + 'disparity_map_'+ str(M) + '.jpg', img)

print(np.max(img))


img_dif = abs(disparity_map - img_truth)
img_mask = np.zeros((h,w))
true = 0
for j in range(h):
  for i in range(w):
    if img_dif[j, i] <= 1:
      true += 1
      img_mask[j, i] = 255
print('accuracy is ', true/h/w)

plt.imshow(img_mask, cmap = 'gray')
cv2.imwrite(path + 'img_mask_'+ str(M) + '.jpg', img_mask)
```