**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 , the following equation should be solved using linear least square method and SVD.

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

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

We denote the mean value of coordinates as , , we have following

The final estimate of the fundamental matrix can be obtained by

**Image Rectification**

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

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

**Experiment Results**

**Task 1**

A picture containing computer

Description automatically generated

Figure 1: Correspondence points on the image 1

A picture containing street, sign, sitting, box

Description automatically generated

Figure 2: Correspondence points on the image 2

**Task 2**

**A picture containing photo, standing, building, field

Description automatically generated**

Figure 3: Disparity map when

A picture containing standing, people, photo, front

Description automatically generated

Figure 4: Binary error mask when (accuracy is 36.8%)

A picture containing building, outdoor, standing, front

Description automatically generated

Figure 5: Disparity map when

A picture containing graffiti

Description automatically generated

Figure 6: Binary error mask when (accuracy is 56.1%)

**Observation**

From the experiment, I increase the 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 area is expanded significantly. With the 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  
 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]]*#,[3120,1575]]*pts2 = [[499,721],[1973,2368],[2815,1722],[3015,1553],[3627,1119],[1806,138],[824,1315],[2127,2883]]*#,[3431,1799]]*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)  
  
  
  
  
  
  
*# 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)  
  
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)