ECE 661 - HW 9

Abhiram Gnanasambandam
agnanasa@purdue.edu

I. THEORY

A. Fundamental matrix

Fundamental matrix has a 7DoFs. We have to get at least 8 correspondences between the two images to estimate the Fundamental matrix. We have to normalize the pixels tuch thae mean is zero and the avergae distance from the center is $\sqrt{2}$ so the the Hartley and Zisserman algorithm gives better results. Denote the transformation which does this as T_1 and T_2.

Then using the normalized 8 points,we get an estimate of the F matix, given two sets of corrspondences x_i and x_i'/ The theory goes as follows. We have to solve the equation $Af = 0$, where

$$A = \begin{bmatrix} x_i'x_i & x_i'y_i & x_i' & y_i'x_i & y_i'y_i & x_i & y_i & 1 \end{bmatrix}$$

The matrix gets populated by 8 such rows given by 8 such correspondences.

$$F = \begin{bmatrix} F_{11} & F_{12} & F_{13} & F_{21} & F_{22} & F_{23} & F_{31} & F_{32} & F_{33} \end{bmatrix}^T$$

We can esaily solve for F by using SVD and finding the eigen-vector which corrresponds to the minimum eigen value. We have to make sure that rank of such a F matrix is always 2. We can ensure it by forcing the third and least eigen value to go to 0. Then we can denormalize the matrix F, by using $F_{de} = T_2^T F T_1$. Now, we can use the LM optimization to get better estimates.

Now, we have to estimate P and P' the Projection matrices to construct the world coordinated. As the cameras are not calibrated, we find the canonocal form. $P = [I|0]$ and $P' = [[e']_x F|e']$

The world co-orrdinates are obtained by finding the solution to $AX_{world} = 0$ given the constraint $||X_{world} = 1||$. Here A is given by

$$A = \begin{bmatrix} x_i P_3^T - P_1^T \\ y_i P_3^T - P_2^T \\ x_i' P_3'^T - P_1'^T \\ y_i' P_3'^T - P_2'^T \end{bmatrix}$$

Now, we generate the error function that will be used to optimize the F. The error function is projecting the world coordinate using P and P' to the respective image coordinates and finding the square of euclidean distance with the already existing coordinates. Now, we have to use LM to optimize the estimate of F again.

B. Image Rectification

We first rotate the image 2 by T_1 matrix. We then find the angle the eppole makes with x-axis and rotate the image so that the epipole beecomes parallel to x-axis.

$$e' = \begin{bmatrix} f \\ 0 \\ 1 \end{bmatrix}$$

The next step os tp send the epipole to infinity. For that we use the following G matrix.

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/f & 0 & 1 \end{bmatrix}$$

Now, we multiply with another transform T_2 to make image back to the center. So, the overall Homography is $H_2 = T_2 G R T_1$.

As we have the homography for Image2, we now have to find the homography for image 1. We use least squares mnimization to find it.

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

The details of the extended method can be found in Multiple View Geometry by Richard Hartley and Andrew Zisserman.

Once, the homographies are found, we can use them to get the rectified images.

C. Using SIFT and canny edge detectors

As we have already done HWs on Canny and SIFT, I am excluding the discussions on them here. We find the corresponding points between the two rectified images using one of two methods. And once we have to corresponding methods, we learnt in last two sections how to triangulate and find the world coordinates.

II. Results

Figure 1: View 1

Figure 2: View 2

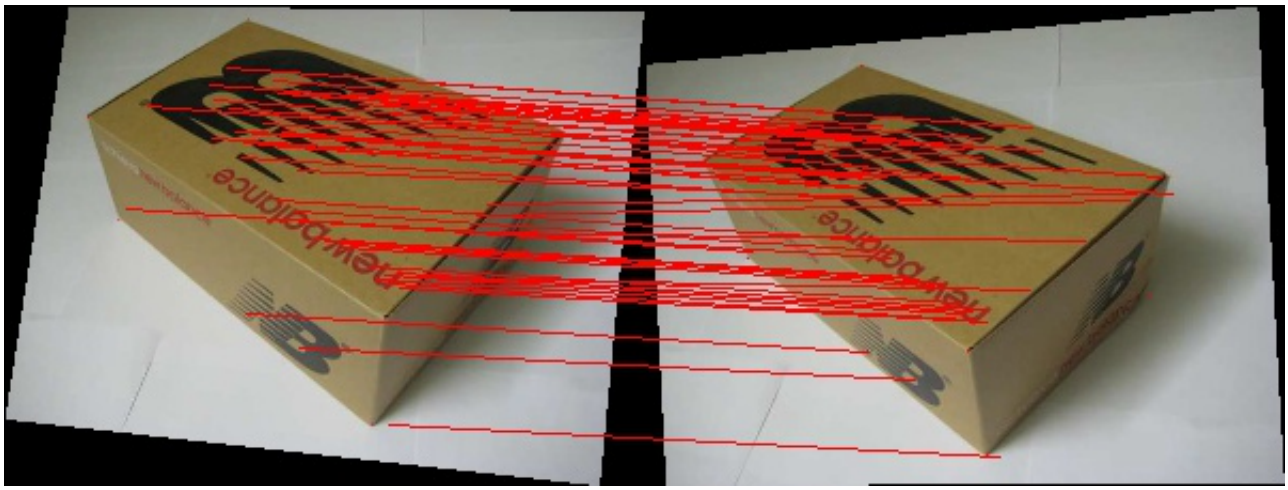

Figure 3: Rectified Images and the SIFT correspondences

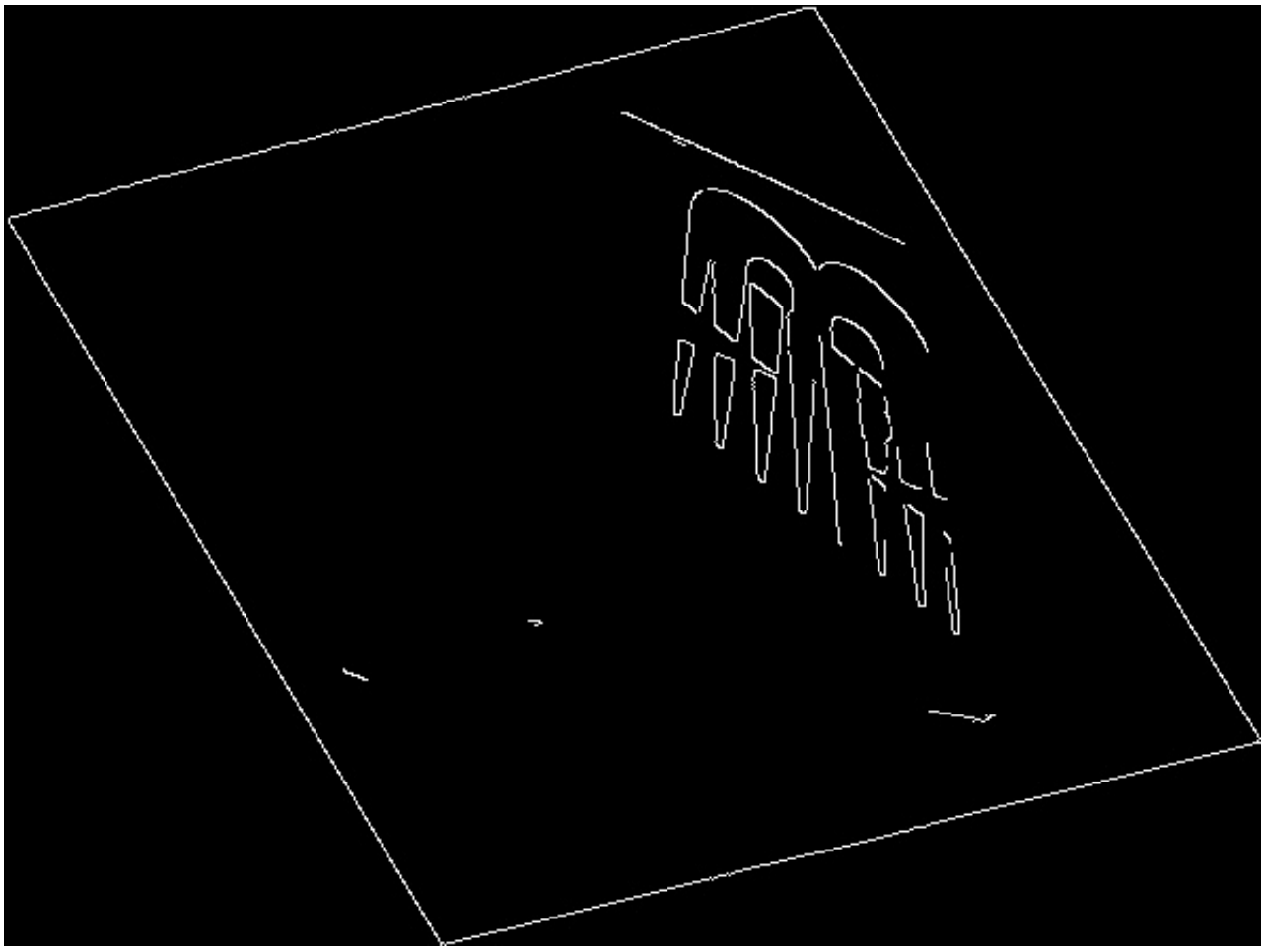

Figure 4: Result of canny on rectified image 1

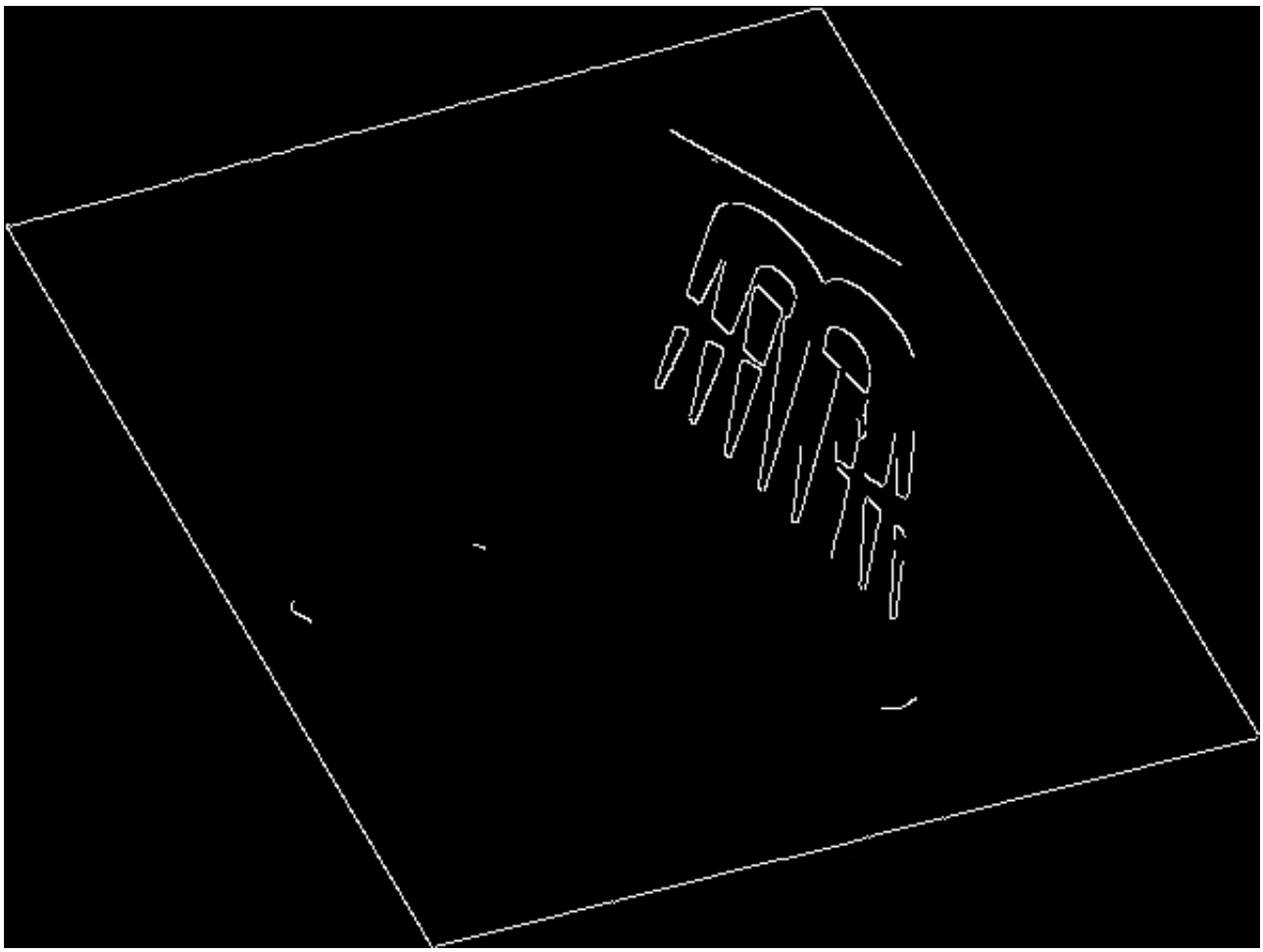

Figure 5: Result of canny on rectified image 2

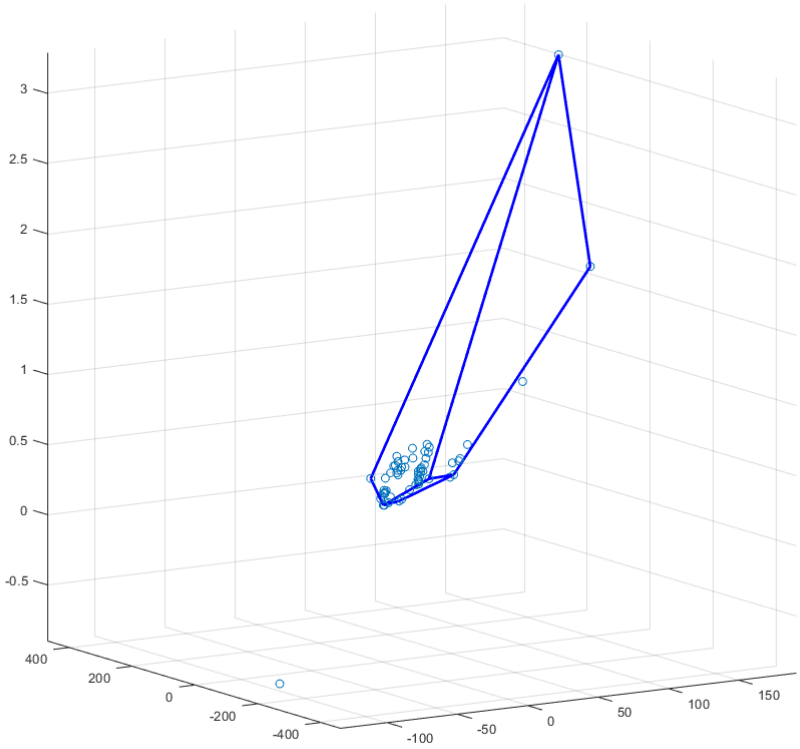

Figure 6: The projected 3D points with SIFT on the rectified images

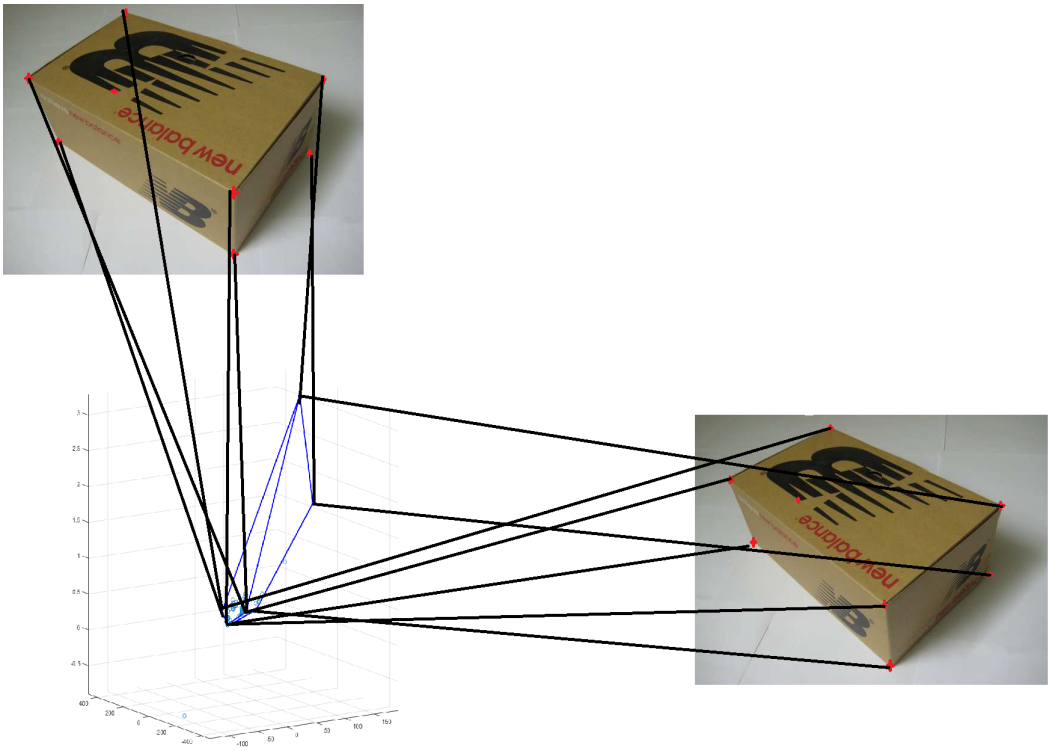

Figure 7: The projected 3D points with SIFT on the rectified images , with details for understanding the scene structure

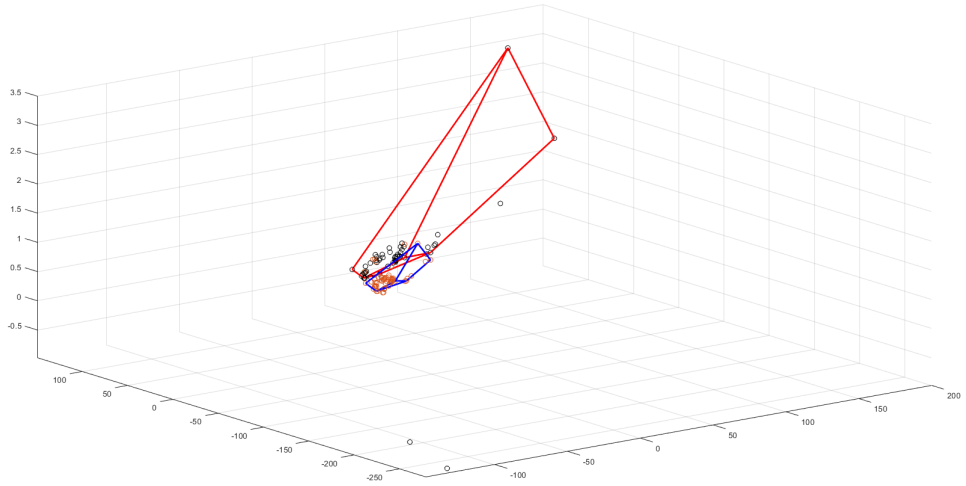

Figure 8: The projected 3D points with SIFT on the rectified images , before (red lines and black circles) and after(blue lines and red dots) using LM optimization.

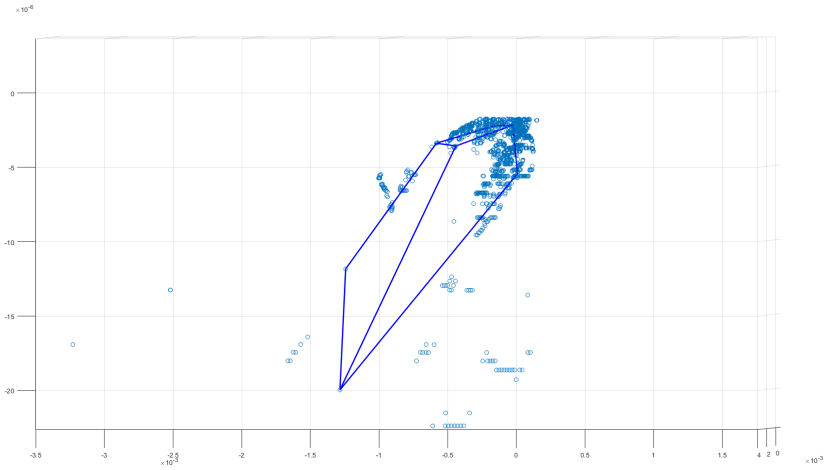

Figure 9: The projected 3D points with Canny on the rectified images

```
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 18 12:49:46 2018

@author: abhir
"""

import cv2 as cv
import numpy as np
from scipy.optimize import least_squares # LM algorithm from here
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def return_correspondences(edges1,edges2):
    p1 = []
    p2 = []
    for i in range(3,len(edges1)-3):
        for j in range(3,len(edges1[0])-3):
            if(edges1[i,j] == 255):
                dist =1e10
                i1 = i
                for j1 in range(3,len(edges2[0])- 3):
                    if(np.linalg.norm(im1[i-2:i+3 , j-2:j+3] - im2[i1-2:i1+3, j1-2:j1+3
                        t = j1
                        dist = np.linalg.norm(im1[i-2:i+3 , j-2:j+3] - im2[i1-2:i1+3, j1

                p1.append([i,j])
                p2.append([i1,t])

    return p1,p2

def draw_lines(image1,image2,points1,points2,points11,points21,number):
    s1 = np.shape(image1)
    s2 = np.shape(image2)

    nimage = np.zeros((s1[0] , s1[1]+s2[1],3));
    nimage[:,0:s1[1]] = image1
    nimage[:,s1[1]:] = image2

    #nimage = nimage.astype('uint8')

    if(number == 0):
        for i in range(0,len(points1)):
            cv.line(nimage,(points1[i][1],points1[i][0]),(s1[1]+points2[i][1], points2[i

    if(number == 1):
        for i in range(0,len(points11)):
            cv.line(nimage,(points11[i][1],points11[i][0]),(s1[1]+points21[i][1], points


    return nimage
```

```
def return_corres_points(SSd, harris1 , harris2 , SSDthresh , NCCthresh):
    l1 = len(SSd)

    points1_NCC = []
    points2_NCC = []

    points1_SSD = []
    points2_SSD = []

    point2 = []


    for i in range(0,l1):
        if(np.sum(SSd[i] < SSDthresh) > 0) :
            j1 = np.argmin(SSd[i])
            if(j1 not in point2):
                if(np.abs(harris1[i][0] - harris2[j1][0]) < 30):
                    point2.append(j1)
                    points1_SSD.append([harris1[i][0],harris1[i][1]])
                    points2_SSD.append([harris2[j1][0],harris2[j1][1]])


    return points1_SSD,points2_SSD,points1_NCC,points2_NCC
def p2p_corres(des1,des2):

    l1 = len(des1)
    l2 = len(des2)


    SSD = np.zeros((l1,l2))


    for i in range(0,l1):
        for j  in range(0,l2):
            SSD[i,j] = np.linalg.norm(des1[i] - des2[j])

    return SSD

def rectifyimage(im1,H1,fn,kkkk):
    Hinv = H1
    #Finding the size of the output image
    tl = np.matmul(Hinv,[0.0,0.0,1.0]); #Map for Top left corner
    tr = np.matmul(Hinv,[len(im1[0]),0.0,1.0]); #Map for Top right corner
    br = np.matmul(Hinv,[len(im1[0]),len(im1),1.0]); #Map for bottom right corner
    bl = np.matmul(Hinv,[0.0,len(im1),1.0]); #Map for bottom left




    tl = tl/tl[2];
    tr = tr/tr[2];
    br = br/br[2];
    bl = bl/bl[2];


    print tl,tr,br,bl
```

```
    #Find x length of the output image
    minx = np.min([tl[0] , tr[0] , br[0] , bl[0] ]);
    maxx = np.max([tl[0] , tr[0] , br[0] , bl[0] ]);
    xlen = np.round(maxx - minx).astype('int');


    #Find y length of the output image
    miny = np.min([tl[1] , tr[1] , br[1] , bl[1] ]);
    maxy = np.max([tl[1] , tr[1] , br[1] , bl[1] ]);
    ylen = np.round(maxy - miny).astype('int');

#    print ylen,xlen
    ###
    #Initialise the ouput image

    H_scaling = np.eye(3);
    print 'asd'
    print minx,miny ,maxx,maxy, xlen,ylen

#    minx = -15.5947476171
#    miny = -138.06632525
#    maxx = 623.459010284
#    maxy = 585.306281335
#    xlen = 639
#    ylen = 723

    H_scaling[0,0] = float(np.shape(im1)[1])/xlen/2
    H_scaling[1,1] = float(np.shape(im1)[0])/ylen/2

    Hinv = np.matmul(H_scaling,Hinv)

#    print H_scaling/H_scaling[2,2]

    tl = np.matmul(Hinv,[0.0,0.0,1.0]); #Map for Top left corner
    tr = np.matmul(Hinv,[len(im1[0]),0.0,1.0]); #Map for Top right corner
    br = np.matmul(Hinv,[len(im1[0]),len(im1),1.0]); #Map for bottom right corner
    bl = np.matmul(Hinv,[0.0,len(im1),1.0]); #Map for bottom left



    tl = tl/tl[2];
    tr = tr/tr[2];
    br = br/br[2];
    bl = bl/bl[2];
#    print tl,tr,br,bl
    #Find x length of the output image
    minx = np.min([tl[0] , tr[0] , br[0] , bl[0] ]);
    maxx = np.max([tl[0] , tr[0] , br[0] , bl[0] ]);
    xlen = np.round(maxx - minx).astype('int');

    #Find y length of the output image
    miny = np.min([tl[1] , tr[1] , br[1] , bl[1] ]);
    maxy = np.max([tl[1] , tr[1] , br[1] , bl[1] ]);
    ylen = np.round(maxy - miny).astype('int');
```

```
    im3 = np.zeros((ylen,xlen,3)).astype('uint8');

    H1 = np.linalg.inv(Hinv)


#     print H1/H1[2][2]
    #Get the image
    for i in range(0,len(im3)):
        for j in range(0,len(im3[0])):
                newindex = np.matmul(H1,[j + minx,i+miny ,1.0])
                newindex = newindex/newindex[2];

                #Use bilinear approximation to get the output pixel value
                if( newindex[1] > 0 and newindex[1] < len(im1)-1 and newindex[0] > 0 and
                        floorx = np.floor(newindex[1]);
                        floorxi = floorx.astype('int');
                        floory = np.floor(newindex[0]);
                        flooryi = floory.astype('int');

                        ceilx = np.ceil(newindex[1]);
                        ceilxi = ceilx.astype('int');
                        ceily = np.ceil(newindex[0]);
                        ceilyi = ceily.astype('int');


                        pixel_value = (ceilx - newindex[1] )*(ceily - newindex[0] )*im1[

                        im3[i][j] = pixel_value.astype('uint8')

    #Write the image
    cv.imwrite(fn,im3)

    kkk=np.zeros((len(kkkk),3))

    for i in range(0,len(kkkk)):
        kkk[i] = np.matmul(Hinv,[kkkk[i,0],kkkk[i,1],1.0])
        kkk[i] = kkk[i]/kkk[i,2]

        kkk[i,0] = kkk[i,0] - minx
        kkk[i,1] = kkk[i,1] - miny


    return im3,kkk




#def rectify_images(image,H,points):

#Find homography for rectification
def find_homo(e1,e2,P1,P2,cp1,cp2,F1):
    x1 = cp1;
    x2 = cp2;
    ss = np.shape(im1);

    angle = math.atan(-(e2[1]- ss[0]/2 )/(e2[0]- ss[1]/2 ))
```

```
    print angle
#     print angle

    f = math.cos(angle)*(e2[0]-ss[1]/2)-math.sin(angle)*(e2[1]-ss[0]/2);
#     print f

    G = np.zeros((3,3))
    G[0,0] = 1
    G[1,1] = 1
    G[2,0] = -1/f
    G[2,2] = 1;

    R = np.zeros((3,3))
    R[0][0] = math.cos(angle)
    R[0,1] = -math.sin(angle)
    R[1,0] = math.sin(angle)
    R[1,1] = math.cos(angle)
    R[2,2] = 1

    T = np.zeros((3,3))

    T[0,0] = 1;
    T[0,2] = -ss[1]/2
    T[1,1] = 1
    T[1,2] = -ss[0]/2
    T[2,2] = 1;

    H2 = np.matmul(G,np.matmul(R,T));


    im_c = np.zeros(3)

    im_c[0] = ss[1]/2 ;
    im_c[1] = ss[0]/2
    im_c[2] = 1;

    c_rec = np.matmul(H2,im_c)
    c_rec = c_rec/c_rec[2]

    T2 = np.eye(3)
    T2[0,2] = ss[1]/2-c_rec[0]
    T2[1,2] = ss[0]/2-c_rec[1]




    H2 = np.matmul(T2,H2) ##########


    #First epipole
#     M = np.matmul(P2,np.linalg.pinv(P1))
    sec_m = np.ones((3,3))

    sec_m[:,0] = e2;
    sec_m[:,1] = e2;
    sec_m[:,2] = e2;
```

```
    F1 = F1/F1[2,2]

#     print a_x(e2), e2,F1
    M = np.matmul(a_x(e2),F1) + sec_m
    # M = exF + evt

#     print M/M[2,2]

#     M = np.matmul(P2,np.linalg.pinv(P1))
    H0 = np.matmul(H2,M)

#     print H0
    l = len(cp1)

    x1_dash = np.zeros((l,3));
    x2_dash = np.zeros((l,3));

    for i in range(0,l):
        x1_dash[i] = np.matmul(H0,x1[i])
        x1_dash[i] = x1_dash[i] / x1_dash[i,2]

        x2_dash[i] = np.matmul(H2,x2[i])
        x2_dash[i] = x2_dash[i] / x2_dash[i,2]


    A = x1_dash
    b = x2_dash[:,0]

#     print A,b

    x = np.matmul(np.linalg.pinv(A),b)



    Ha = np.eye(3)
    Ha[0,0] = x[0]
    Ha[0,1] = x[1]
    Ha[0,2] = x[2]

    H1 = np.matmul(Ha,H0)

    c_rec = np.matmul(H1,im_c)
    c_rec = c_rec/ c_rec[2]

    T1 = np.eye(3)
    T1[0,2] = ss[1]/2-c_rec[0]
    T1[1,2] = ss[0]/2-c_rec[1]

    H1 = np.matmul(T1,H1)

    F_rec = np.matmul(np.linalg.pinv(np.transpose(H2)),np.matmul(F1,np.linalg.pinv(H1)))


    x1_rec = np.zeros((l,3));
    x2_rec = np.zeros((l,3));
```

```
    for i in range(0,1):
        x1_rec[i] = np.matmul(H1,x1[i])
        x1_rec[i] = x1_rec[i] / x1_rec[i,2]

        x2_rec[i] = np.matmul(H2,x2[i])
        x2_rec[i] = x2_rec[i] / x2_rec[i,2]


    u,s,vh = np.linalg.svd(F_rec)

    e1_rec = np.transpose(vh[2])
    e1_rec = e1_rec / e1_rec[2]
    e2_rec = u[:,2]
    e2_rec = e2_rec/e2_rec[2]

    return x1_rec,x2_rec,H1,H2,F_rec,e1_rec,e2_rec

# LM cost for F matrix
def costf(f):
    F = np.zeros((3,3))
    F[0,0] = f[0]
    F[0,1] = f[1]
    F[0,2] = f[2]
    F[1,0] = f[3]
    F[1,1] = f[4]
    F[1,2] = f[5]
    F[2,0] = f[6]
    F[2,1] = f[7]
    F[2,2] = f[8]

    u,s,vh = np.linalg.svd(F)

#    s[2] = 0
#
#    F = np.matmul(u,np.matmul(np.diag(s),vh))
#    u,s,vh = np.linalg.svd(F)

    e1 = np.transpose(vh[2])

    e2 = u[:,2]

#    print e1,e2

    P = np.zeros((3,4))

    P[0:3,0:3] = np.eye(3)

    e2x = a_x(e2)

    P_dash = np.zeros((3,4))


    P_dash[0:3,0:3] = np.matmul(e2x,F)
    P_dash[0:3,3] = e2
#    print e2x
#    print '\n\n\n'
```

```
#     print P,P_dash
    C = []
    l = len(cp1)
    for i in range(0,l):
        A = np.zeros((4,4));
        A[0] = cp1[i][0] * P[2,:] - P[0,:]
        A[1] = cp1[i][1] * P[2,:] - P[1,:]
        A[2] = cp2[i][0] * P_dash[2,:] - P_dash[0,:]
        A[3] = cp2[i][1] * P_dash[2,:] - P_dash[1,:]


        u,s,vh = np.linalg.svd(A)

        xw = np.transpose(vh[3])

        xw = xw / np.linalg.norm(xw)

        x1 = np.matmul(P,xw)
        x2 = np.matmul(P_dash,xw)

        x1 = x1/x1[2]
        x2 = x2/x2[2]

#         print x1,x2
#         print np.linalg.norm(x1 - cp1)**2,np.linalg.norm(x2 - cp2)**2
#         print cp1,cp2
        C.append(np.linalg.norm(x1 - cp1[i])**2);
        C.append(np.linalg.norm(x2 - cp2[i])**2)
#     print np.linalg.norm(C)

    return np.asarray(C)

def a_x(w):
    wx = w[0]
    wy = w[1]
    wz = w[2]
    Wx = np.zeros((3,3))
    Wx[0][1] = -wz
    Wx[0][2] = wy
    Wx[1][0] = wz
    Wx[1][2] = -wx
    Wx[2,0] = -wy
    Wx[2,1] = wx

    return Wx
#Find the Fundamental Matrix
def findF(np1,np2):
    #Set he matrices for solving for F
    l = len(cp1)
    A = np.zeros((l,9));

    for i in range(0,l):
        A[i,0] = np2[i][0]*np1[i][0]
        A[i,1] = np2[i][0]*np1[i][1]
        A[i,2] = np2[i][0]
        A[i,3] = np2[i][1]*np1[i][0]
```

```
        A[i,4] = np2[i][1]*np1[i][1]
        A[i,5] = np2[i][1]
        A[i,6] = np1[i][0]
        A[i,7] = np1[i][1]
        A[i,8] = 1

    print A[:,6],np1[:,0]
    u,s,vh = np.linalg.svd(A);

    f = np.transpose(vh[8])

    F = np.zeros((3,3))
    F[0,0] = f[0]
    F[0,1] = f[1]
    F[0,2] = f[2]
    F[1,0] = f[3]
    F[1,1] = f[4]
    F[1,2] = f[5]
    F[2,0] = f[6]
    F[2,1] = f[7]
    F[2,2] = f[8]

    u,s,vh = np.linalg.svd(F);

    s[2] = 0
    F = np.matmul(u,np.matmul(np.diag(s),vh))

    return F


#This function normalizes the points
def normalize_points(points):
    m1 = np.mean(points[:,0]);
    m2 = np.mean(points[:,1]);

    print np.shape(points),np.shape(m1)

    dist1 = (points[:,0]-m1)*(points[:,0]-m1)
    dist2 = (points[:,1]-m2)*(points[:,1]-m2)

    dist = np.zeros(len(points))

    for i in range(0,len(points)):
        dist[i] = np.sqrt(dist1[i] + dist2[i])

    md = np.sum(dist)/len(points)

    scale = np.sqrt(2) / md;

    new_points = np.ones(np.shape(points))

    T = np.zeros((3,3))

    T[0][0] = scale
    T[0][2] = -scale * m1
    T[1][1] = scale;
```

```
    T[1][2] = -scale * m2;
    T[2][2] = 1;

    for i in range(0,len(points)):
        new_points[i][0] = (points[i][0] - m1)*scale
        new_points[i][1] = (points[i][1] - m2)*scale

#    new_points1 = np.ones(np.shape(points))
#
#    for i in range(0,8):
#        new_points1[i] = np.matmul(T,points[i]);

    return new_points,T

im1 = cv.imread('11.png')
im2 = cv.imread('12.png')


cp1 = np.zeros((8,3))
cp2 = np.zeros((8,3))


#cp1[0] = [1286,2158,1]
#cp1[1] = [1731,2621,1]
#cp1[2] = [2261,2503,1]
#cp1[3] = [2317,1320,1]
#cp1[4] = [1785,1326,1]
#cp1[5] = [1303,1237,1]
#cp1[6] = [1492,1626,1]
#cp1[7] = [1497,1389,1]
#
#
#
#cp2[0] = [1369,2120,1]
#cp2[1] = [1786,2584,1]
#cp2[2] = [2322,2469,1]
#cp2[3] = [2368,1280,1]
#cp2[4] = [1825,1291,1]
#cp2[5] = [1374,1203,1]
#cp2[6] = [1554,1592,1]
#cp2[7] = [1558,1358,1]

cp1[0] = [94,231,1]
cp1[1] = [384,415,1]
cp1[2] = [42,123,1]
cp1[3] = [384,311,1]
cp1[4] = [205,13,1]
cp1[5] = [535,121,1]
cp1[6] = [510,244,1]
cp1[7] = [186,147,1]



cp2[0] = [100,212,1]
cp2[1] = [331,419,1]
cp2[2] = [60,107,1]
```

```
cp2[3] = [318,316,1]
cp2[4] = [228,22,1]
cp2[5] = [510,154,1]
cp2[6] = [490,268,1]
cp2[7] = [176,141,1]


#Img1 = []
#Img2 = []
#Img1.append([1859.0,447.0,1.0])
#Img1.append([3905.0,1011.0,1.0])
#Img1.append([481.0,897.0,1.0])
#Img1.append([3001.0,2033.0,1.0])
#Img1.append([737.0,2093.0,1.0])
#Img1.append([2701.0,3341.0,1.0])
#Img1.append([2073.0,1937.0,1.0])
#Img1.append([2777.0,2785.0,1.0])
#
#
#
#
#Img2.append([1921.0,459.0,1.0])
#Img2.append([3977.0,985.0,1.0])
#Img2.append([557.0,853.0,1.0])
#Img2.append([3093.0,1929.0,1.0])
#Img2.append([793.0,2077.0,1.0])
#Img2.append([2769.0,3329.0,1.0])
#Img2.append([2153.0,1869.0,1.0])
#Img2.append([2857.0,2725.0,1.0])
#
#cp1 = np.asarray(Img1)
#cp2 = np.asarray(Img2)

kkkk1 = cp1[0:7];
kkkk2 = cp2[0:7];

for i in range(0,8):
    cv.circle(im1, (int(cp1[i,0]),int(cp1[i,1])) , 1, (0,0,255), -1)
    cv.circle(im2, (int(cp2[i,0]),int(cp2[i,1])) , 1, (0,0,255), -1)

#normalized points
np1,T1 = normalize_points(cp1)
np2,T2 = normalize_points(cp2)

Fde = findF(np1,np2)

F = np.matmul(np.transpose(T2), np.matmul(Fde,T1));

F = F/F[2,2]
u,s,vh = np.linalg.svd(F)

e1 = np.transpose(vh[2])
e2 = u[:,2]

P = np.zeros((3,4))
```

```
P[0:3,0:3] = np.eye(3)

e2x = a_x(e2)

P_dash = np.zeros((3,4))


P_dash[0:3,0:3] = np.matmul(e2x,F)
P_dash[0:3,3] = e2
F = F / F[2,2]
f = np.zeros(9)
f[0] = F[0][0]
f[1] = F[0][1]
f[2] = F[0][2]
f[3] = F[1][0]
f[4] = F[1][1]
f[5] = F[1][2]
f[6] = F[2][0]
f[7] = F[2][1]
f[8] = F[2][2]


#res = least_squares(costf , f , method='lm')
#for i in range(0,8)
f1 = f

F1 = np.zeros((3,3))
F1[0,0] = f1[0]
F1[0,1] = f1[1]
F1[0,2] = f1[2]
F1[1,0] = f1[3]
F1[1,1] = f1[4]
F1[1,2] = f1[5]
F1[2,0] = f1[6]
F1[2,1] = f1[7]
F1[2,2] = f1[8]

F1 = F1/F1[2,2]


#u,s,vh = np.linalg.svd(F1)
#
#s[2] = 0
#
#F1 = np.matmul(u,np.matmul(np.diag(s),vh))
#
u,s,vh = np.linalg.svd(F1)
e1 = np.transpose(vh[2])
e2 = u[:,2]

P = np.zeros((3,4))

P[0:3,0:3] = np.eye(3)

e1 = e1/e1[2]
e2 = e2/e2[2]
```

```
e2x = a_x(e2)


P_dash = np.zeros((3,4))
P_dash[0:3,0:3] = np.matmul(e2x,F1)
P_dash[0:3,3] = e2

x1_rec,x2_rec,H1,H2,F_rec,e1_rec,e2_rec = find_homo(e1,e2,P,P_dash,cp1,cp2,F1)#x1_rec,x2

image1clr,kkk1 = rectifyimage(im1,H1,'rec1.png',kkkk1)
image2clr,kkk2 = rectifyimage(im2,H2,'rec2.png',kkkk2)

kkk1 = kkk1.astype('int')
kkk2 = kkk2.astype('int')
#SIFT
#sift = cv.xfeatures2d.SIFT_create()
#image1clr = cv.resize(image1clr, (0,0), fx=0.2, fy=0.2)
#image1gray= cv.cvtColor(image1clr,cv.COLOR_BGR2GRAY)
#
#kp1,des1 = sift.detectAndCompute(image1gray,None)
#
#harris1 = []
#
#for i in kp1:
#    harris1.append([i.pt[1],i.pt[0]])
#
#
##image2clr = cv.imread('m4.jpg')
##image2clr = cv.resize(image2clr, (0,0), fx=0.2, fy=0.2)
#image2gray = cv.cvtColor(image2clr,cv.COLOR_BGR2GRAY)
#kp2, des2 = sift.detectAndCompute(image2gray,None)
#
#harris2 = []
#
#for i in kp2:
#    harris2.append([i.pt[1],i.pt[0]])
#
#SSD = p2p_corres(des1,des2)
#
##for i in range(1,1000):
##    if(np.sum(SSD <i  ) >= 1000):
##        ll = i
##        break;
#
#points1_SSD,points2_SSD,points1_NCC,points2_NCC = return_corres_points(SSD,harris1,harr
#
#points1_SSD = np.asarray(points1_SSD).astype('int')
#points2_SSD = np.asarray(points2_SSD).astype('int')
#points1_NCC = np.asarray(points1_SSD).astype('int')
#points2_NCC = np.asarray(points2_SSD).astype('int')
#
#
#
#nimage1 = draw_lines(image1clr,image2clr,points1_SSD,points2_SSD,points1_NCC,points2_NC
#
#stri = 'sift.jpg'
```

```
#cv.imwrite(stri,nimage1)


#Canny
image1gray= cv.cvtColor(image1clr,cv.COLOR_BGR2GRAY)
edges1 = cv.Canny(image1gray,255*1.5,255)

image2gray= cv.cvtColor(image2clr,cv.COLOR_BGR2GRAY)
edges2 = cv.Canny(image2gray,255*1.5,255)

points1_SSD,points2_SSD = return_correspondences(edges1,edges2);

points1_SSD = np.asarray(points1_SSD).astype('int')
points2_SSD = np.asarray(points2_SSD).astype('int')
points1_NCC = np.asarray(points1_SSD).astype('int')
points2_NCC = np.asarray(points2_SSD).astype('int')

nimage1 = draw_lines(image1clr,image2clr,points1_SSD,points2_SSD,points1_NCC,points2_NCC

stri = 'sift.jpg'
cv.imwrite(stri,nimage1)


#############
Img1 = []
Img2 = []
for i in range(len(points1_SSD)):
    Img1.append([points1_SSD[i][0] , points1_SSD[i][1] , 1.0 ] )
    Img2.append([points2_SSD[i][0] , points2_SSD[i][1] , 1.0 ])

for j in range(len(kkk1)):
    Img1.append([kkk1[j][1] , kkk1[j][0] , 1.0 ] )
    Img2.append([kkk2[j][1] , kkk2[j][0] , 1.0 ])


cp1 = np.asarray(Img1)
cp2 = np.asarray(Img2)

#for i in range(0,8):
#    cv.circle(im1, (int(cp1[i,0]),int(cp1[i,1])) , 1, (0,0,255), -1)
#    cv.circle(im2, (int(cp2[i,0]),int(cp2[i,1])) , 1, (0,0,255), -1)

#normalized points
np1,T1 = normalize_points(cp1)
np2,T2 = normalize_points(cp2)

Fde = findF(np1,np2)

F = np.matmul(np.transpose(T2), np.matmul(Fde,T1));

F = F/F[2,2]
u,s,vh = np.linalg.svd(F)

e1 = np.transpose(vh[2])
e2 = u[:,2]
```

```
P = np.zeros((3,4))

P[0:3,0:3] = np.eye(3)

e2x = a_x(e2)

P_dash = np.zeros((3,4))


P_dash[0:3,0:3] = np.matmul(e2x,F)
P_dash[0:3,3] = e2
F = F / F[2,2]
f = np.zeros(9)
f[0] = F[0][0]
f[1] = F[0][1]
f[2] = F[0][2]
f[3] = F[1][0]
f[4] = F[1][1]
f[5] = F[1][2]
f[6] = F[2][0]
f[7] = F[2][1]
f[8] = F[2][2]


#res = least_squares(costf , f , method='lm')
#for i in range(0,8)
f1 = f

F1 = np.zeros((3,3))
F1[0,0] = f1[0]
F1[0,1] = f1[1]
F1[0,2] = f1[2]
F1[1,0] = f1[3]
F1[1,1] = f1[4]
F1[1,2] = f1[5]
F1[2,0] = f1[6]
F1[2,1] = f1[7]
F1[2,2] = f1[8]

F1 = F1/F1[2,2]


#u,s,vh = np.linalg.svd(F1)
#
#s[2] = 0
#
#F1 = np.matmul(u,np.matmul(np.diag(s),vh))
#
u,s,vh = np.linalg.svd(F1)
e1 = np.transpose(vh[2])
e2 = u[:,2]

P = np.zeros((3,4))

P[0:3,0:3] = np.eye(3)
```

```
e1 = e1/e1[2]
e2 = e2/e2[2]
e2x = a_x(e2)


P_dash = np.zeros((3,4))
P_dash[0:3,0:3] = np.matmul(e2x,F1)
P_dash[0:3,3] = e2

xw = np.zeros((len(cp1) , 3))
for i in range(0,len(cp1)):
    A = np.zeros((4,4));
    A[0] = cp1[i][0] * P[2,:] - P[0,:]
    A[1] = cp1[i][1] * P[2,:] - P[1,:]
    A[2] = cp2[i][0] * P_dash[2,:] - P_dash[0,:]
    A[3] = cp2[i][1] * P_dash[2,:] - P_dash[1,:]


    u,s,vh = np.linalg.svd(A)

    xw1 =   np.transpose(vh[3])
    xw1 = xw1 /xw1[3]

    xw[i] = xw1[0:3]

fig = plt.figure()
ax = fig.add_subplot(111,projection = '3d')
ax.scatter(xw[:,0],xw[:,1],xw[:,2])
#ax.plot(xw[50:52,0],xw[50:52,1],xw[50:52,2])
#ax.plot(xw[52:54,0],xw[52:54,1],xw[52:54,2])
#ax.plot(xw[54:56,0],xw[54:56,1],xw[54:56,2])
#ax.plot([xw[50,0],xw[52,0]],[xw[50,1],xw[52,1]],[xw[50,2],xw[52,2]])
#ax.plot([xw[52,0],xw[54,0]],[xw[52,1],xw[54,1]],[xw[52,2],xw[54,2]])
#ax.plot([xw[51,0],xw[53,0]],[xw[51,1],xw[53,1]],[xw[51,2],xw[53,2]])
#ax.plot([xw[53,0],xw[55,0]],[xw[53,1],xw[55,1]],[xw[53,2],xw[55,2]])
```