ECE 661 HW09
Rehana Mahfuz (rmahfuz@purdue.edu)

Theory

1) Initial estimation of Fundamental Matrix using manual correspondences

At least 8 manual correspondences need to be found between the two images. Let one pair of correspondences be denoted by (x,y), (x',y'). To begin with, we normalize these points such that all pixel correspondences have zero mean and are a distance of sqrt(2) from the center. All manually selected points in the left image are multiplied by T, and all manually selected points in the right image are multiplied by T'

Then the following row vector is constructed for each correspondence:

[x'x,   x'y,   x,   y' , y'y,   y',   x,   y,   1]

All of these row vectors are stacked horizontally to obtain a matrix A. Since $x'^TFx = 0$, Af=0, where f=[f11, f12, f13, f21, f22, f23, f31, f32, 1]. f33 is set to 1, since we only need to calculate the matrix F up to a constant. The equation Af=0 is solved using the linear least squares method to get an initial estimate of F. This method involves finding the singular value decomposition (SVD) of A, and then selecting the eigenvector corresponding the smallest eigenvalue as the solution. Since we want the rank of the fundamental matrix to be 2, we further take the SVD of F as u,d,v^T = svd(F), zero out the smallest eigenvalue in d, and then calculate $F=u*d*v^T$, this time using the modified d. Finally, we denormalize the fundamental matrix as $F_{denormalized} = T'^TFT$. We refine F by nonlinear optimization, the details of which are in section 4 of the theory.

The epipole e of the left image is the right null vector of F, and the epipole e' of the right image is the left null vector of F. If P1, the camera projection matrix of the left image is

$P1 = [I_{3x3}|0]$, then the right camera projection matrix is $P2=[[e']_xFe']$, where $[e']_x$ is the cross product representation of the right epipole.

2) Rectification of images to send epipoles to infinity

Moving epipole e' to infinity:

The second image center is shifted to the origin using T1= $\begin{bmatrix} 1 & 0 & -0.5*width \\ 0 & 1 & -0.5*height \\ 0 & 0 & 1 \end{bmatrix}$

The angle of the epipole with respect to the +x-axis is found as theta=$\tan^{-1}(\frac{-(e'[1]-0.5*height)}{e'[0]-0.5*width}$

, so that the rotation matrix R = $\begin{bmatrix} \cos(theta) & -\sin(theta) & 0 \\ \sin(theta) & \cos(theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$ can be applied to make the epipole e' go to the x-axis. Next, G= $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.5 * f & 0 & 1 \end{bmatrix}$ is applied to make the epipole e' go to infinity along the x-axis. Another translation T2 is applied to move the center back to the original center. In short, the homography to be applied to the right image is H'=T2*G*R*T1.

Moving epipole e to infinity:

Find M=P'P^{+}. We find a,b,c to minimize the following sum of squares:

$\sum_i \left(ax_i^{hat} + by_i^{hat} + c - x_i'^{hat}\right)^2$, where x_i^{hat}=H0*x_i and x'^{hat}_i = H2*x'_I, and H0=H2*M. We calculate a,b,c to construct the following matrix: H_A = $\begin{bmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. In short, the homography to be applied to the first image is H = H_A * H0.

The resulting images are called rectified images, and we can expect the correspondences between these images to be in the same row, or in a row very close to its own row.

3) Finding correspondences between automatically generated interest points

The canny edge detector is used to find edges. One-twelfth of the pixels that are edges are considered as corners. The Normalized Cross Correlation (NCC) metric is used to establish correspondences between the corners. Moreover, the search for the correspondence of a particular point is limited to the same row and neighboring few rows to take advantage of the rectification.

4) Non-linear least squares refinement of the Fundamental Matrix

The Levenberg-Marquardt (LM) algorithm is used for nonlinear least squares refinement to make the fundamental matrix more accurate. For each of the correspondences calculated, the triangulation method outlined in section 5 of the theory is used to find the corresponding world point. That world point is projected back to the images using the camera projection matrices that we have from our existing estimate of the fundamental matrix. The measure we strive to minimize is:

$D^2_{geom} = \sum_i \left(\left|\left| x_i - x_i^{projected} \right|\right|^2 + \left|\left| x_i' - x_i'^{projected} \right|\right|^2 \right)$

5) 3D reconstruction of points

Let the corresponding points in the left and right images be (x,y) and (x'y') respectively. Let P_i be the ith row of the camera projection matrix of the left image, and let P'_I be the ith row of the camera projection matrix of the right image. Calculate

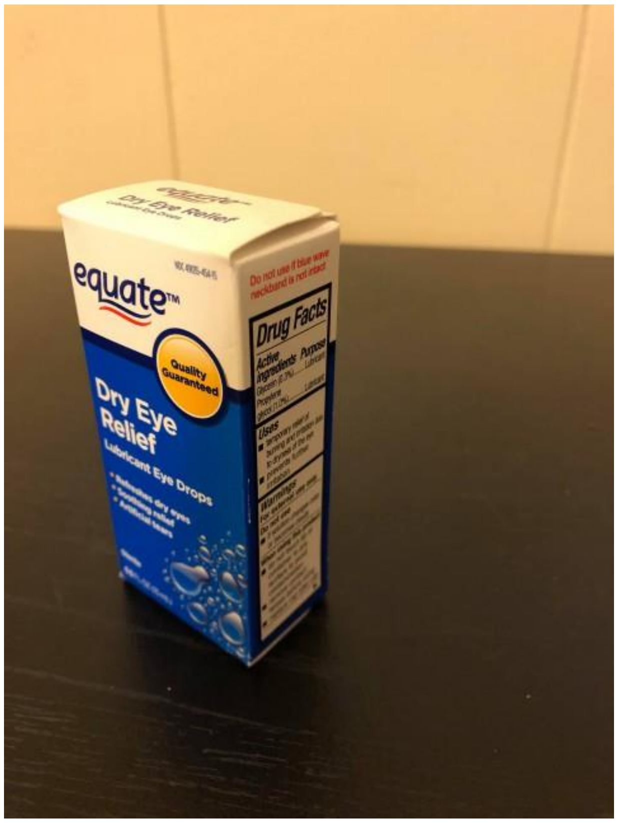
$$A = \begin{bmatrix} xP_3^T - P_1^T \\ yP_3^T - P_2^T \\ x'P_3'^T - P_1'^T \\ y'P_3'^T - P_2'^T \end{bmatrix}$$

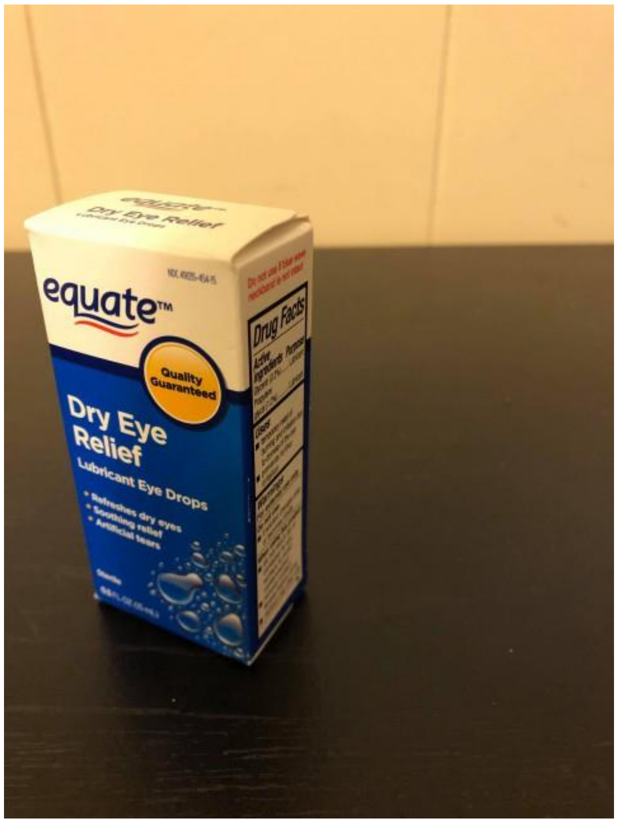
We want to minimize ||AX||, subject to ||X||=1, which is why we use the null vector of matrix A (eigenvector corresponding to smallest eigenvalue of singular value decomposition) as our solution, which is the 3-dimensional point we just reconstructed from two two-dimensional images of the same scene!

We specifically reconstruct in 3D the corner points, label them, and draw lines between them so that we can see the edges of the box in three dimensions.

Results

Images 1 and 2:

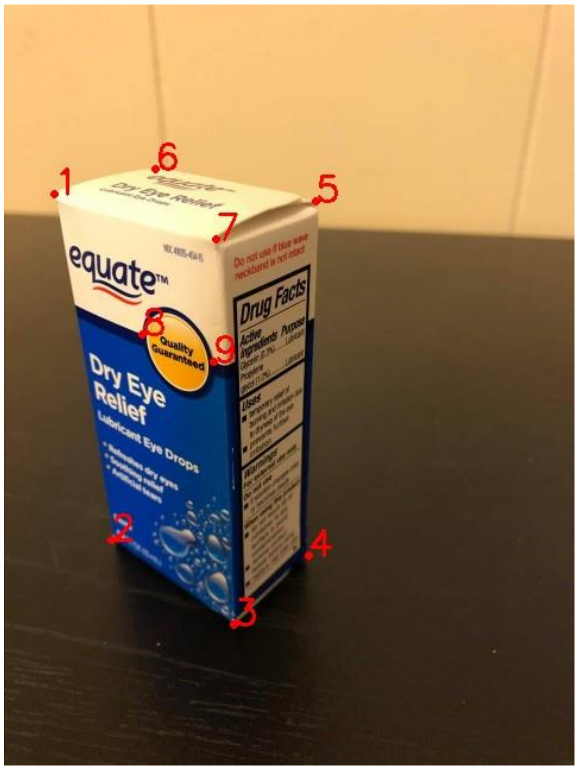


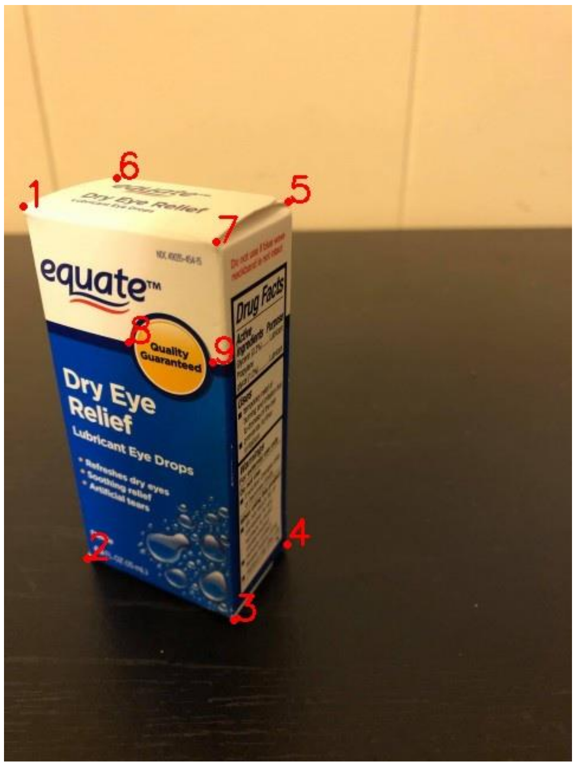


ECE 661 HW09

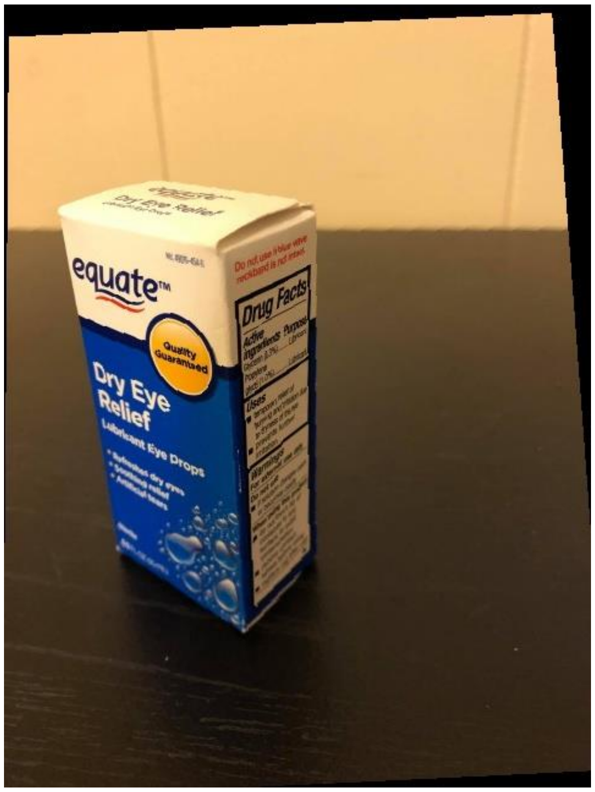
Rehana Mahfuz (rmahfuz@purdue.edu)

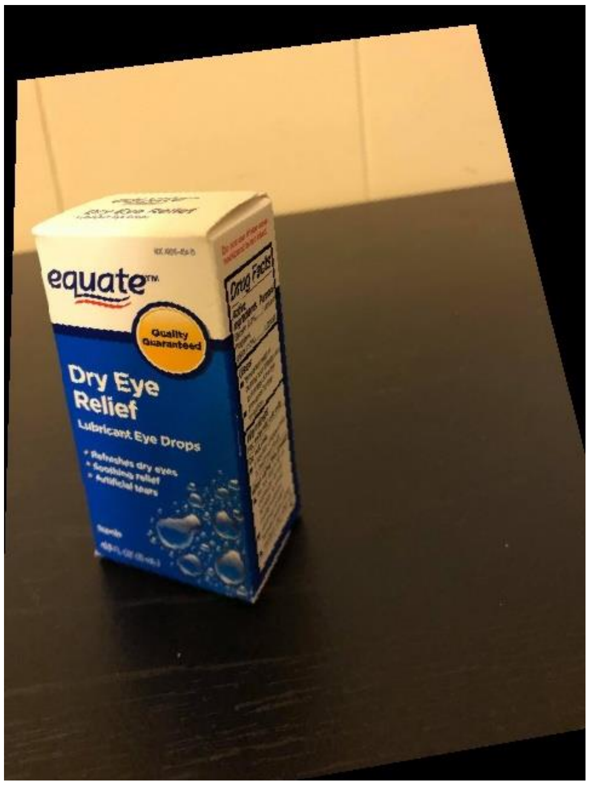
Images 1 and 2 with corners:





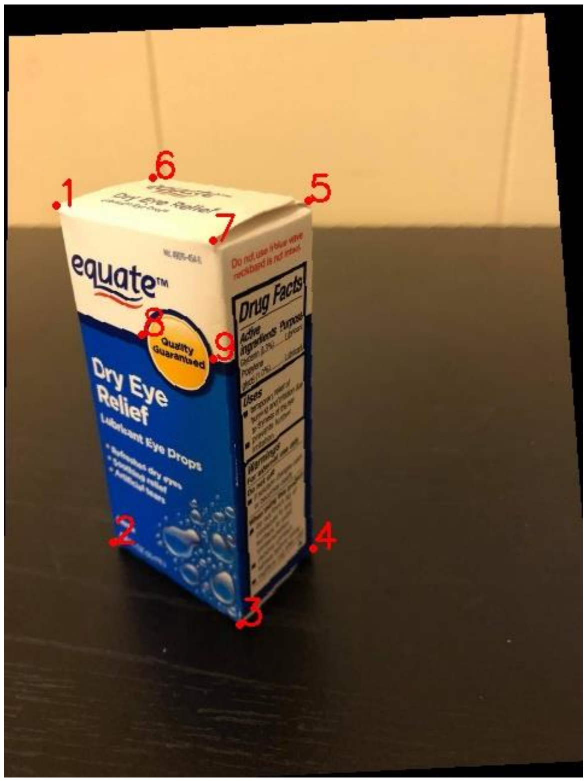
Rectified images 1 and 2:

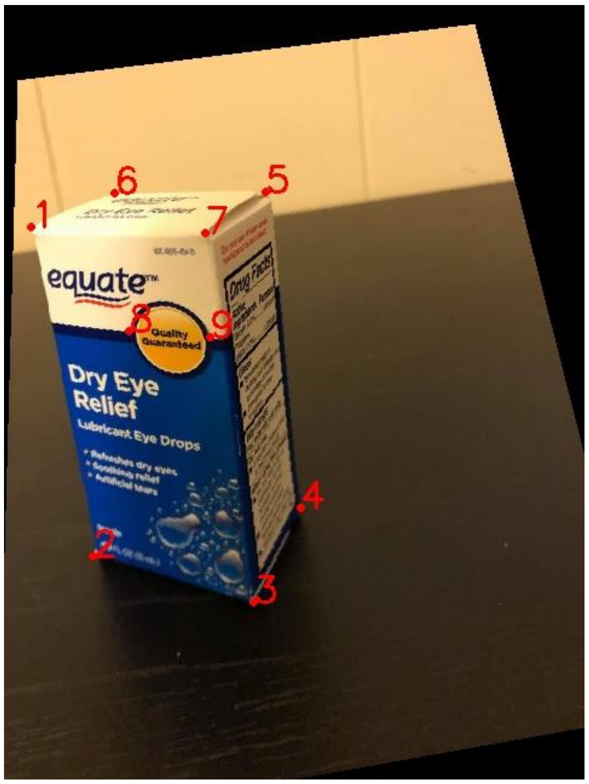




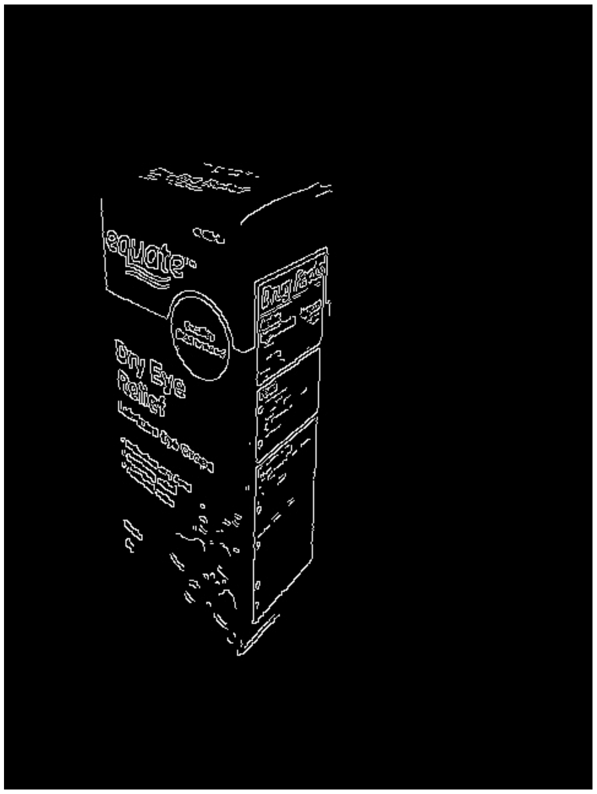
ECE 661 HW09
Rehana Mahfuz (rmahfuz@purdue.edu)

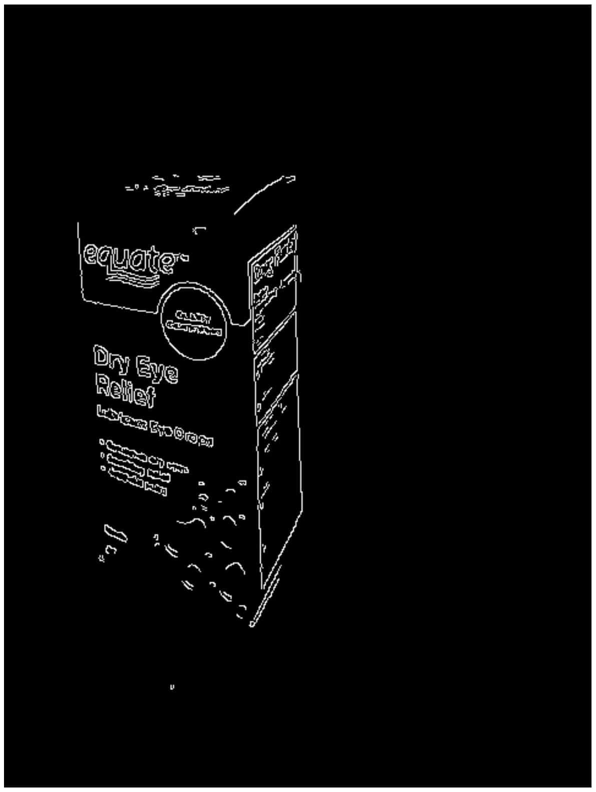
Rectified images with corners:





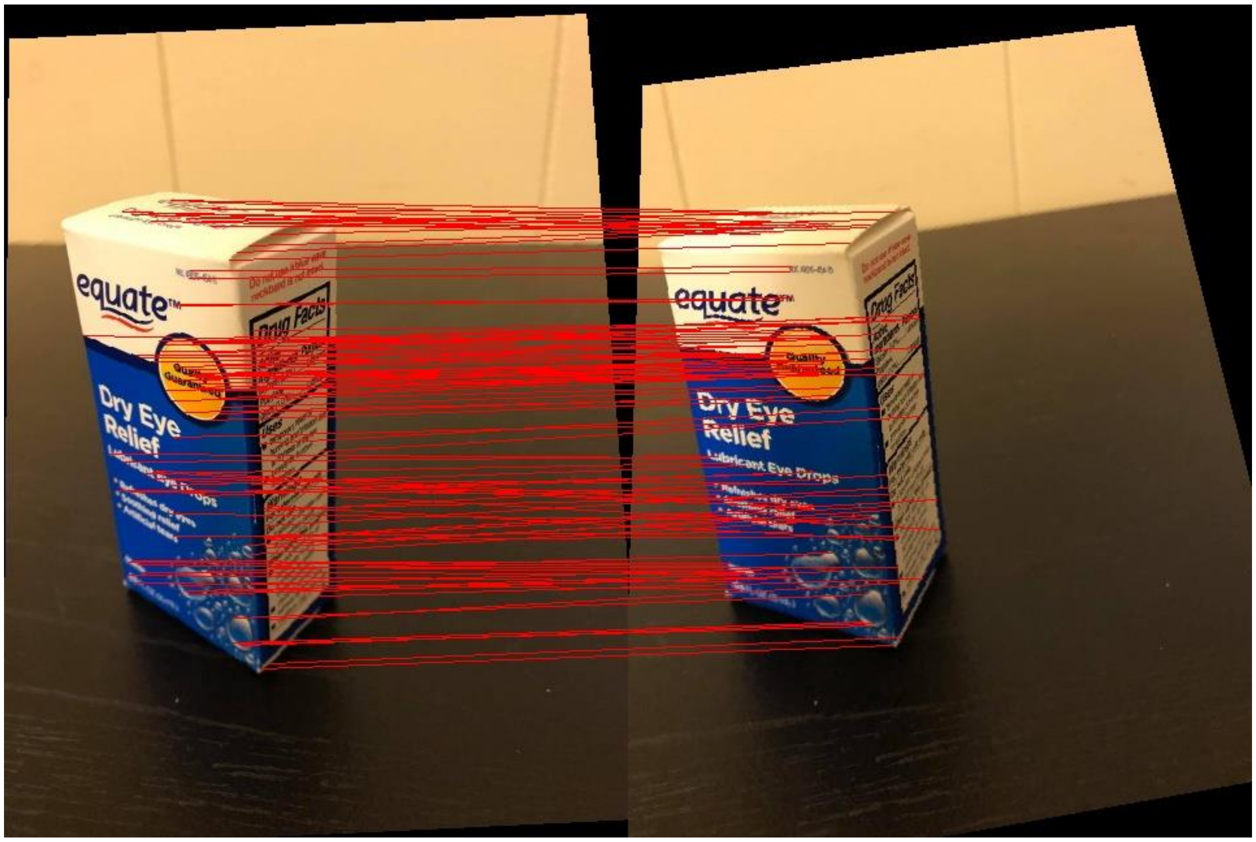
Edges detected using canny edge detector:





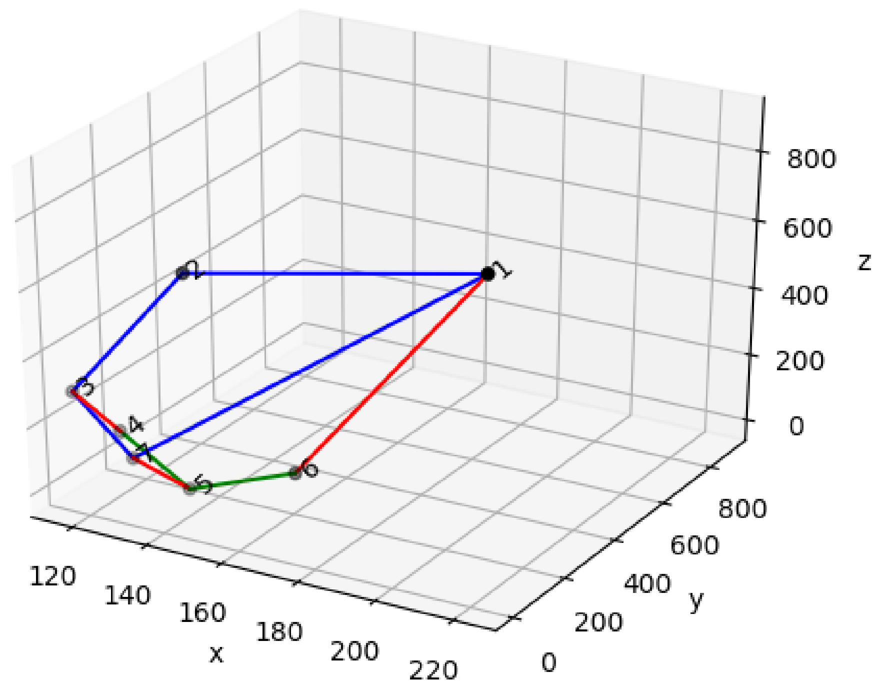
ECE 661 HW09
Rehana Mahfuz (rmahfuz@purdue.edu)

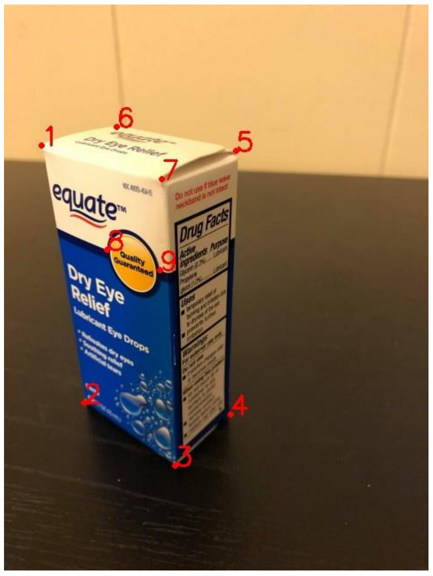
A subset of the pixels that make up the edges are chosen to be corners. Correspondences between corners using the NCC metric:

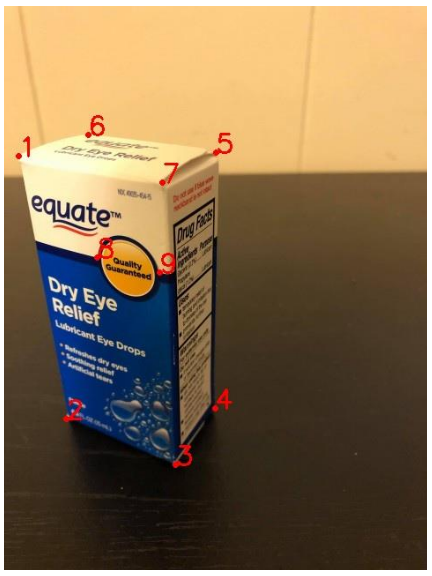


ECE 661 HW09
Rehana Mahfuz (rmahfuz@purdue.edu)

3D reconstruction with points labelled as numbers. Edges are drawn out. Edges that make up the front rectangle are blue. Edges making up the rectangle behind are green (only 2 of these can be seen). Edges joining the front rectangle with the back rectangle are red. Corners in the actual image are shown for comparison:

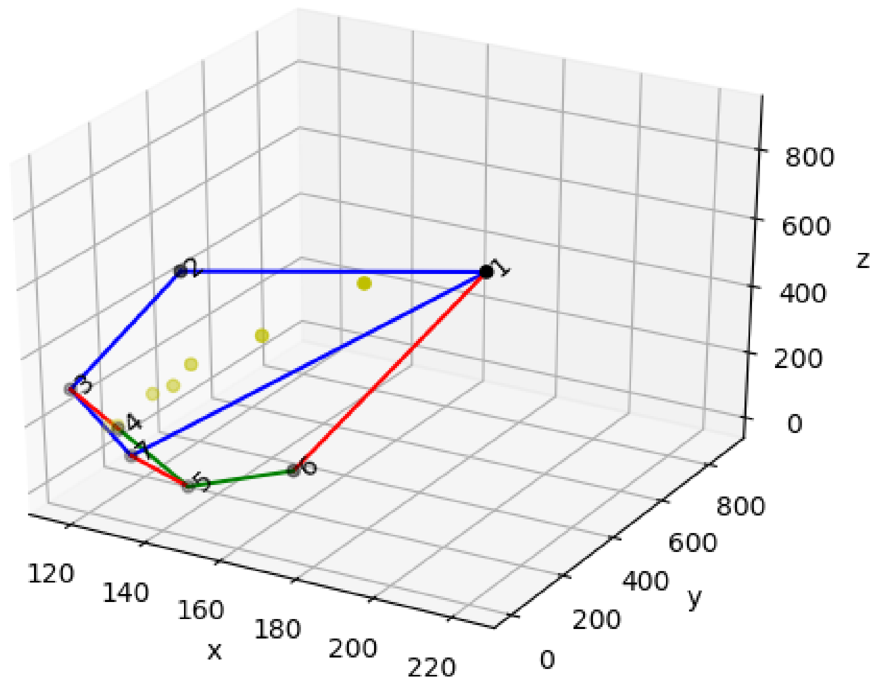


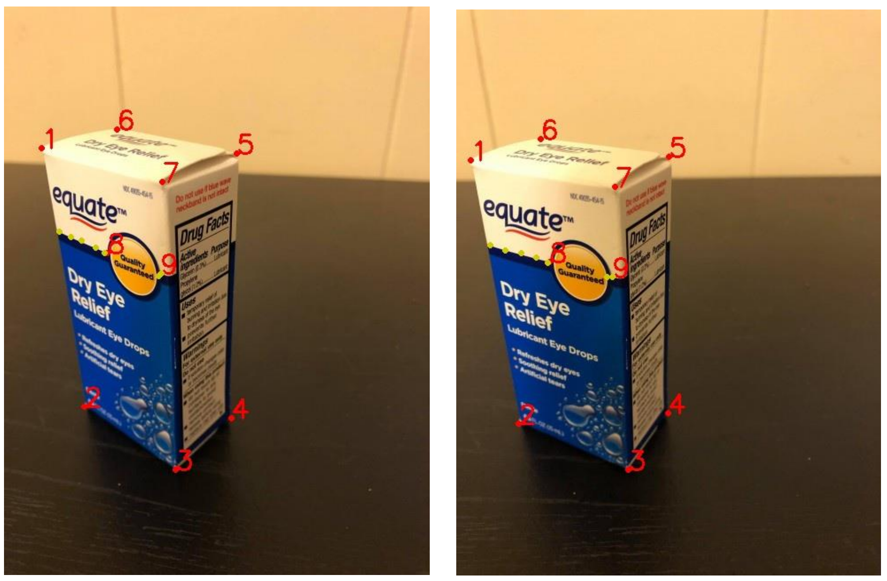




ECE 661 HW09
Rehana Mahfuz (rmahfuz@purdue.edu)

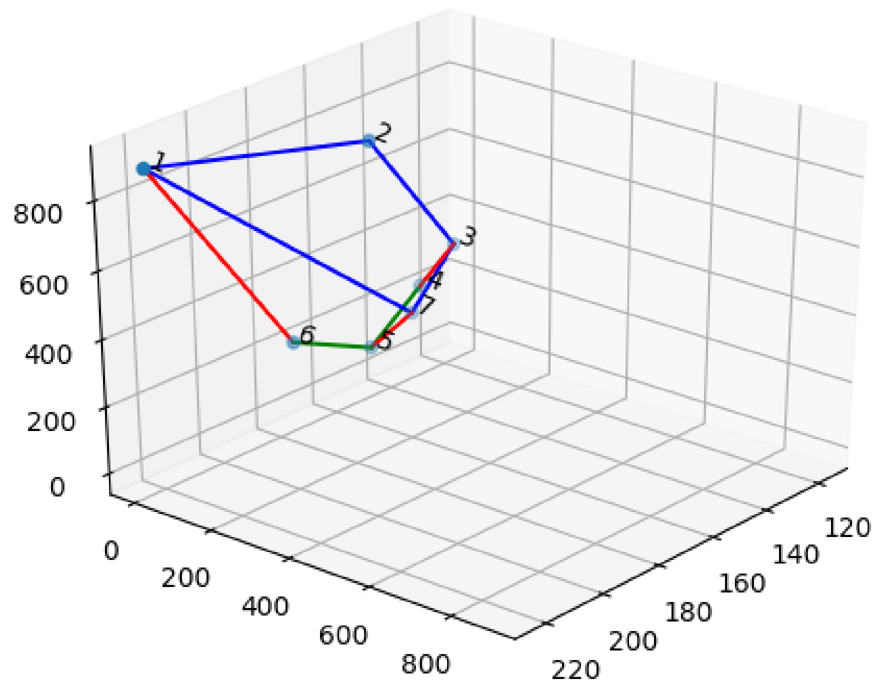
Some extra points on the front rectangle are shown to better represent the reconstruction. These points are along the black line that separates the lower blue part of the rectangle from the upper white part. Again, these yellow points are also shown on the original image for comparison:





ECE 661 HW09
Rehana Mahfuz (rmahfuz@purdue.edu)

Reconstruction from another point of view:



Since LM optimization did not make a noticeable improvement to the fundamental matrix, or to the position of the reconstructed 3D points, I have not included those images. They would be repetitive.

Observations

1) Rectifying the image did not work too well. Lines between correspondences are not necessarily very parallel to the x-axis, but make a small angle with the x-axis.
2) The number of corners found using canny edge detector is large. Using a subset of these pixels is a good way to find corners.
3) Non-linear optimization using the Levenberg-Marquardt method did not significantly improve the fundamental matrix. This is not surprising since LM only makes small refinements to the original fundamental matrix calculated using the original linear least squares method. The reason the initial estimate of the fundamental matrix is so accurate is probably because the manual point correspondences were accurate, and covered the 3-dimensional structure of the box as much as possible. Also, I used 9 manual correspondences instead of 8. After LM refinement, the maximum change in an element of the fundamental matrix was in the order of -10.

Source code

```
import numpy as np
```

ECE 661 HW09
Rehana Mahfuz (rmahfuz@purdue.edu)

```
import cv2
from scipy.linalg import null_space
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy import optimize


path = '../Users/rmahfuz/Desktop/661/HW09/'
#================================================================================
def ncc(cor_pts1, cor_pts2, img1, img2):
    #returns a list of point correspondences, where every element is of the form [(pt_x1, pt_y1), (pt_x2, pt_y2)]
    fil_win = 31 #filter window length
    pad_len = int(fil_win/2)
    ncc_store = np.zeros((len(cor_pts1), len(cor_pts2))) #array to store all NCCs
    ncc_store = ncc_store - 2
    for i in range(len(cor_pts1)):
        for j in range(len(cor_pts2)):
            cor1 = cor_pts1[i]; cor2 = cor_pts2[j]

            x_lo1 = max(0, cor1[0]-pad_len)
            x_hi1 = min(cor1[0]+pad_len+1, img1.shape[0])
            y_lo1 = max(0,cor1[1]-pad_len)
            y_hi1 = min(cor1[1]+pad_len+1, img1.shape[1])

            x_lo2 = max(0, cor2[0]-pad_len)
            x_hi2 = min(cor2[0]+pad_len+1, img2.shape[0])
            y_lo2 = max(0,cor2[1]-pad_len)
            y_hi2 = min(cor2[1]+pad_len+1, img2.shape[1])
            if x_hi1 - x_lo1 == x_hi2 - x_lo2 and y_hi1 - y_lo1 == y_hi2 - y_lo2:
                mean1 = np.mean(img1[x_lo1:x_hi1,y_lo1:y_hi1])
                mean2 = np.mean(img2[x_lo2:x_hi2,y_lo2:y_hi2])
                term1 = np.subtract(img1[x_lo1:x_hi1,y_lo1:y_hi1], mean1)
                term2 = np.subtract(img2[x_lo2:x_hi2,y_lo2:y_hi2], mean2)
                ncc_store[i,j] = np.divide(np.sum(np.multiply(term1, term2)),
                                 np.sqrt(np.multiply(np.sum(np.square(term1)), np.sum(np.square(term2)))))
                #np.sum(np.square(np.subtract(img1[x_lo1:x_hi1,y_lo1:y_hi1],
img2[x_lo2:x_hi2,y_lo2:y_hi2])))


    to_ret = [];nccs = []
    #thresh = np.min(ssd_store[ncc_store > 0])
    track = np.ones(len(cor_pts2))
    for i in range(len(ncc_store)):
        cur = ncc_store[i]
        to_find = cur[cur>=-1]
        if len(to_find) > 0:
            j = np.argmax(to_find)
            if abs(cor_pts1[i][0] - cor_pts2[j][0]) < 30 and abs(cor_pts1[i][1] - cor_pts2[j][1]) < 60 and track[j]
== 1:# and max(to_find) > 0.45:
                if track[j] == 1 and max(to_find) > 0.6:
                    nccs.append(max(to_find))
                    to_ret.append([cor_pts1[i], cor_pts2[j]])
                    track[j] = 0
    sorted_idx = np.argsort(nccs)
    #to_ret = to_ret[sorted_idx]
    return (to_ret, nccs)
#================================================================================
def findT(pts):
    '''pts is a list (of length >=8) of tuples, each tuple containing a point'''
    x_only = list(map(lambda x: x[0], pts))
    y_only = list(map(lambda x: x[1], pts))
    mu_x = np.mean(x_only)
    mu_y = np.mean(y_only)
    term1 = np.square(x_only-mu_x)
    term2 = np.square(y_only-mu_y)
    meann = (1.0/len(pts))*np.sum(np.sqrt(np.add(term1,term2)))
    scale = np.sqrt(2)/meann
    x = -1*scale*mu_x
    y = -1*scale*mu_y
    T = np.array([[scale, 0, x], [0, scale, y], [0, 0, 1]])
    return T
#==============================================================================
def findF(pts1, pts2, T1, T2):
    #print('pts1 = {}, pts2 = {}'.format(pts1, pts2))
    A = []
    for i in range(len(pts1)):
        to_app = [pts2[i][0]*pts1[i][0], pts2[i][0]*pts1[i][1], pts2[i][0], pts2[i][1]*pts1[i][0],
                  pts2[i][1]*pts1[i][1], pts2[i][1], pts1[i][0], pts1[i][1], 1.0]
        A.append(to_app)
    #print('A.shape = {}'.format(np.array(A).shape))
    u,d,v_t = np.linalg.svd(A) #Linear least sq solution:
    v = v_t.T; #print('v = \n{}'.format(v))
    F = v[:,v.shape[1]-1];#print('F = \n{}'.format(F))
    #print('F = \n{}'.format(F))
    #F = np.append(F,1)
    assert len(F) == 9
    F = F.reshape(3,3); #F = F.T
    u,d,v_t = np.linalg.svd(F) #Constraint enforncement that rank(F)=2

    #print('u.shape = {}, d.shape = {}, v_t.shape = {}'.format(u.shape, d.shape, v_t.shape))

    #print('d = \n{}'.format(d))

    d = np.array([[d[0],0,0],[0,d[1],0],[0,0,0]])

    F = np.matmul(u, d); F = np.matmul(F, v_t) #F = u*d*v_t

    F = np.matmul(T2.T,F); F = np.matmul(F, T1)#F = T2'*F*T1 (denormalization)

    #F = F/F[2,2]

    #print('F = \n{}'.format(F))

    return F

#=================================================================================

def apply_h(H, img, fname = None):

    height = img.shape[0]; width = img.shape[1]; #print('height = {}, width = {}'.format(height, width))

    #Scaling:

    cor_old = [[0,0], [img.shape[1],0], [0, img.shape[0]], [img.shape[1], img.shape[0]]]

    cor = np.matmul(H,
np.array([[0,img.shape[1],0,img.shape[1]],[0,0,img.shape[0],img.shape[0]],[1,1,1,1]]))

    cor /= cor[2]; #print('cor = \n{}'.format(cor))

    min_val = [min(cor[0]), min(cor[1])]

    max_val = [max(cor[0]), max(cor[1])]

    Dim = np.array(max_val) - np.array(min_val); Dim = [int(Dim[0]), int(Dim[1])]; #print('scaling Dim =
\n{}'.format(Dim))

    scale = np.array([[width/Dim[0],0,0], [0,height/Dim[1],0], [0,0,1]])

    H = np.matmul(scale, H)

    #Translation:

    cor = np.matmul(H,
np.array([[0,img.shape[1],0,img.shape[1]],[0,0,img.shape[0],img.shape[0]],[1,1,1,1]]));

    cor /= cor[2]; #print('cor = \n{}'.format(cor))

    min_val = [min(cor[0]), min(cor[1])]

    Dim = min_val; Dim = [int(Dim[0]), int(Dim[1])]; #print('translation Dim = \n{}'.format(Dim))

    T = np.array([[1,0,-1*Dim[0]+1], [0,1,-1*Dim[1]+1], [0,0,1]])
    H_n = np.matmul(T, H); Hinv = np.linalg.pinv(H_n);

    new_img = np.zeros((img.shape[0], img.shape[1],3))

    for i in range(height):

        for j in range(width):

            tmp = np.matmul(Hinv, np.array([[i],[j],[1]])); tmp/=tmp[2]

            if tmp[0] >= 0 and tmp[0] < height and int(tmp[1]) >= 0 and int(tmp[1]) < width:

                new_img[i,j] = img[int(tmp[0]), int(tmp[1])]

    #new_img = new_img.T

    if fname == None:

        fname = 'rectified_img.jpg'

    cv2.imwrite(path+fname, new_img)

    return [new_img, H_n]

#==================================================================================

def rectify(pts1, pts2, img1, img2, F):

    height = img1.shape[0]; width = img1.shape[1]

    num_pts = len(pts1)

    e1 = null_space(F);e1/=e1[2];#print('e1 = ', e1)

    e2 = null_space(F.T);e2/=e2[2];#print('e2.shape = ', e2.shape)

    #print('e1=\n{}\ne2=\n{}'.format(e1,e2))


    #To calculate H2:

    theta = np.arctan(-1*(e2[1] - height/2.0)/(e2[0] - width/2.0))

    f = np.cos(theta)*(e2[0] - width/2.0) - np.sin(theta) * (e2[1] - height/2.0); f = f[0]; #print('f = 
{}'.format(f))

    R = np.array([[np.cos(theta)[0], -1*np.sin(theta)[0], 0], [np.sin(theta)[0], np.cos(theta)[0], 0], [0,0,1]])

    T = np.array([[1,0, -1*width/2.0], [0,1, -1*height/2.0], [0,0,1]])

    G = np.array([[1,0,0],[0,1,0],[-1.0/f, 0, 1]]);#print('G = \n{}\nR = \n{}'.format(G,R))

    H2 = np.matmul(G,R);

    H2 = np.matmul(H2,T); assert H2.shape == (3,3)
    center = np.array([[width/2.0,height/2.0,1]]);

    center_new = np.matmul(H2, center.T);

    center_new /= center_new[2]; #print('STOP HERE: In rectify, center = ', center);

    T2 = np.array([[1,0, (width/2.0) - center_new[0]], [0, 1, (height/2.0)-center_new[1]], [0,0,1]])

    H2 = np.matmul(T2,H2); #print('H2 = \n{}'.format(H2))

    #To calculate H1:

    theta = np.arctan(-1*(e1[1] - height/2.0)/(e1[0] - width/2.0))

    f = np.cos(theta)*(e1[0] - width/2.0) - np.sin(theta) * (e1[1] - height/2.0); f = f[0];

    R = np.array([[np.cos(theta)[0], -1*np.sin(theta)[0], 0], [np.sin(theta)[0], np.cos(theta)[0], 0], [0,0,1]])

    T = np.array([[1,0, -1*width/2.0], [0,1, -1*height/2.0], [0,0,1]])

    G = np.array([[1,0,0],[0,1,0],[-1.0/f, 0, 1]])

    H1 = np.matmul(G,R); H1 = np.matmul(H1,T); assert H1.shape == (3,3)

    center = np.array([width/2.0,height/2.0,1]); center_new = np.matmul(H1, center.T);

    center_new /= center_new[2]; #print('STOP HERE: In rectify, center = ', center);

    T1 = np.array([[1,0, (width/2.0) - center_new[0]], [0, 1, (height/2.0)-center_new[1]], [0,0,1]])

    H1 = np.matmul(T1,H1); #print('H1 = \n{}'.format(H1))

    #To calculate H1 again:

    P = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0]]); #print('e2 = {}'.format(e2))

    e2_x = np.array([[0,-1*e2[2],e2[1]], [e2[2], 0, -1*e2[0]], [-1*e2[1], e2[0], 0]]); #print('e2_x =
\n{}'.format(e2_x))

    P_dash = np.matmul(e2_x, F);

    P_dash = np.append(P_dash, np.array([e2[0], e2[1], e2[2]]), axis = 1);#  print('P_dash =
\n{}'.format(P_dash))

    '''e1 /= e1[2]; e2 /= e2[2]; #print('e2 = {}'.format(e2))

    M = np.matmul(P_dash, np.linalg.pinv(P))

    if np.linalg.matrix_rank(M) == 2:

        #print('rank of M is 2')

        M = np.array([[P_dash[0,0], P_dash[0,2], P_dash[0,3]],

                [P_dash[1,0], P_dash[1,2], P_dash[1,3]],
                    [P_dash[2,0], P_dash[2,2], P_dash[2,3]]])

    H0 = np.matmul(H2, M)



    pts1_hat = list(map(lambda x: [x[0],x[1],1], pts1)); pts1_hat = np.array(pts1_hat).T; #print('pts1_hat = 
\n{}'.format(pts1_hat))

    pts2_hat = list(map(lambda x: [x[0],x[1],1], pts2)); pts2_hat = np.array(pts2_hat).T

    pts1_hat = np.matmul(H0, pts1_hat); pts1_hat /= pts1_hat[2];#print('pts1_hat = 
\n{}'.format(pts1_hat));

    pts2_hat = np.matmul(H2, pts2_hat); pts2_hat /= pts2_hat[2];#print('pts2_hat = 
\n{}'.format(pts2_hat))



    abc = np.matmul(np.linalg.pinv(np.array(pts1_hat).T), 
np.array(pts2_hat[0,:]).T);#print('np.array(pts2_hat[0,:]).T = \n{}'.format(np.array(pts2_hat[0,:]).T))

    #print('abc = \n{}'.format(abc))

    Ha = np.array([[abc[0], abc[1], abc[2]], [0,1,0], [0,0,1]])

    H1 = np.matmul(Ha, H0)

    center = np.array([width/2.0, height/2.0, 1]).T

    center_new = np.matmul(H1, center);#print('center_new = \n{}'.format(center_new));

    center_new /= center_new[2]

    T1 = np.array([[1,0,width/2.0-center_new[0]], [0,1,height/2.0-center_new[1]], [0,0,1]])

    H1 = np.matmul(T1,H1)'''

    #print('H1 = \n{}\nH2=\n{}'.format(H1, H2))

    [rect_img1, H1] = apply_h(H1, img1, '1rect.jpg')

    [rect_img2, H2] = apply_h(H2, img2, '2rect.jpg')

    F = np.matmul(np.linalg.pinv(H2.T), F); F = np.matmul(F, np.linalg.inv(H1))

    tmp_pts1 = list(map(lambda x: [x[1],x[0],1], pts1)); pts1 = np.matmul(H1, np.array(tmp_pts1).T);

    pts1 /= pts1[2]; pts1 = pts1.T; pts1 = np.array(list(map(lambda x: [x[1],x[0],x[2]], pts1)))

    tmp_pts2 = list(map(lambda x: [x[1],x[0],1], pts2)); pts2 = np.matmul(H2, np.array(tmp_pts2).T)

    pts2 /= pts2[2]; pts2 = pts2.T; pts2 = np.array(list(map(lambda x: [x[1],x[0],x[2]], pts2)))
    #print('pts1 = \n{}\npts2=\n{}'.format(pts1, pts2))


    stacked_img = np.hstack((rect_img1, rect_img2))

    for i in range(len(pts1)):

        cv2.line(stacked_img, (int(pts1[i,0]), int(pts1[i,1])), (int(pts2[i,0])+img1.shape[1], int(pts2[i,1])), color
= (0,0,255), thickness = 2)

    cv2.imwrite(path + 'stacked.jpg', stacked_img)

    print('just wrote stacked.jpg')


    return [rect_img1, rect_img2, F, pts1, pts2, H1, H2, P_dash]

#===============================================================================

def make_gray(img):

    new_img = []

    for i in range(img.shape[0]): #Making the image grayscale

        new_img.append(list(map(lambda x: sum(x)/3, img[i])))

    img = np.array(new_img, dtype=np.uint8)

    return img

#===============================================================================

def remove_corner_edges(edges):

    for i in range(edges.shape[0]):

        for j in range(edges.shape[1]):

            if j < 40 or j > 350 or i < 100:

                edges[i,j] = 0

    return edges

#===============================================================================

def get_corners(img):

    '''finds indices of corners from output of canny edge detector'''

    corners=[]; cnt = 0

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i,j] != 0:
                cnt += 1
                if cnt%12 == 0:
                    corners.append([i,j])
    return corners
#===============================================================================
def triangulate(corresp, P_dash):
    world_pts=[]
    #print('corresp = {}'.format(corresp))
    P = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0]])
    for (pt1,pt2) in corresp:
        #print('pt1 = {}, pt2 = {}'.format(pt1,pt2))
        A=np.array([pt1[0]*P[2] - P[0],
                    pt1[1]*P[2] - P[1],
                    pt2[0]*P_dash[2] - P_dash[0],
                    pt2[1]*P_dash[2] - P_dash[1]])
        #print('A = \n{}'.format(A))
        u,d,v_t = np.linalg.svd(A)
        v=v_t.T
        tmp=v[:,-1]; tmp/=tmp[3];#print('tmp = \n{}'.format(tmp));
        world_pts.append(tmp)
    return world_pts
#===============================================================================
def lm(pts1, pts2, F):
    #Change pts1 and pts2 to 3 x num_pts:
    #print('pts1 = ',pts1)
    assert np.array(pts1).shape[1] == 3
    pts1=np.array(pts1).T
    pts2=np.array(pts2).T

    #print('F given to us = {}'.format(F))

    F = F.flatten(); F/= F[-1];F = F[:-1]; F = F.reshape(8,1);#print('F right before lm = {}'.format(F))


    def errfunc(F,pts1,pts2):

        #print('F in lm= {}'.format(F))

        P1 = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0]]);

        F = F.flatten(); F = np.append(F,1).reshape(3,3);#print('F in LM's errfunc= \n{}'.format(F))'''

        #F = np.array(list(F.flatten()).append(1)).reshape(3,3);print('F = \n{}'.format(F))

        #F = F.reshape(3,3)

        e2 = null_space(F.T);e2/=e2[2];#print('e2.shape = ', e2.shape)

        e2_x = np.array([[0,-1*e2[2],e2[1]], [e2[2], 0, -1*e2[0]], [-1*e2[1], e2[0], 0]]); #print('e2_x =
\n{}'.format(e2_x))

        P2 = np.matmul(e2_x, F);

        P2 = np.append(P2, np.array([e2[0], e2[1], e2[2]]), axis = 1);


        corresps=[]

        for i in range(len(pts1[0])):

            corresps.append([[pts1[0,i],pts1[1,i]],[pts2[0,i],pts2[1,i]]])

        world_pts=triangulate(corresps, P2)

        world_pts=np.array(world_pts).T; #print('world_pts.shape = {}'.format(world_pts.shape))


        proj1 = np.matmul(P1,world_pts); proj1/=proj1[2]; #(3x4) x (4x8) = (3x8)

        proj2 = np.matmul(P2,world_pts); proj2/=proj2[2]; #(3x4) x (4x8) = (3x8)

        #print('proj1.shape = {}, pts1.shape = {}'.format(proj1.shape, pts1.shape))

        assert proj1.shape == pts1.shape

        err = np.append(proj1[:2]-pts1[:2], proj2[:2]-pts2[:2])

        #print('err = {}'.format(err))

        #err = np.sum(np.square(proj1-pts1)) + np.sum(np.square(proj2-pts2))
        return err


    ans=optimize.leastsq(errfunc,F,args=(pts1,pts2))

    F=F.flatten(); print('change = ',ans[0] - F)

    #print('Success of lm: {}, message: {}'.format(ans.ier,ans.mesg))

    #print('ans = \n{}'.format(ans))

    F=np.append(ans[0],1).reshape(3,3)

    e2 = null_space(F.T);e2/=e2[2];#print('e2.shape = ', e2.shape)

    e2_x = np.array([[0,-1*e2[2],e2[1]], [e2[2], 0, -1*e2[0]], [-1*e2[1], e2[0], 0]]); #print('e2_x =
\n{}'.format(e2_x))

    P2 = np.matmul(e2_x, F);

    P2 = np.append(P2, np.array([e2[0], e2[1], e2[2]]), axis = 1);

    return [F,P2]



#==========================================================================================


def main():

    img1 = cv2.imread(path + '1.jpg')

    img2 = cv2.imread(path + '2.jpg')

    #Manually extracted points

    pts1 = [[159,42],[450,90],[521,193],[463,256],[165,262],[138,127],[197,178],[277,116],[300,176]]

    pts2 = [[170,16],[468,70],[520,194],[456,239],[166,240],[146,95 ],[200,179],[285,105],[302,176]]

    '''img1_with_corners=img1; i = 0

    for pt in pts1:

        i+=1

        cv2.circle(img1_with_corners,(int(pt[1]),int(pt[0])),2,color=(0,0,255),thickness=2)

cv2.putText(img1_with_corners,str(i),(int(pt[1]),int(pt[0])),fontFace=cv2.FONT_HERSHEY_SIMPLEX,fontS
cale=1,color=(0,0,255),thickness=2)
```

```
    cv2.imwrite(path+'img1_with_corners.jpg',img1_with_corners)

    img2_with_corners=img2; i=0;

    for pt in pts2:

        i+=1

        cv2.circle(img2_with_corners,(int(pt[1]),int(pt[0])),2,color=(0,0,255),thickness=2)

cv2.putText(img2_with_corners,str(i),(int(pt[1]),int(pt[0])),fontFace=cv2.FONT_HERSHEY_SIMPLEX,fontS
cale=1,color=(0,0,255),thickness=2)

    cv2.imwrite(path+'img2_with_corners.jpg',img2_with_corners)

    print('just wrote images with corners')'''




    '''stacked_img = np.hstack((img1, img2))

    for i in range(len(pts1)):

        cv2.line(stacked_img, (int(pts1[i][0]), int(pts1[i][1])), (int(pts1[i][0])+img1.shape[1], int(pts2[i][1])),
color = (0,0,255), thickness = 2)

    cv2.imwrite(path + 'stacked_old.jpg', stacked_img)'''


    '''yellow_pts1 = [(62,253),(75,259),(87,263),(103,271),(113,277),(177,300),(174,303)]

    yellow_pts2=[(38,266),(55,272),(73,275),(90,280),(103,286),(176,302),(170,302)]

    i=0

    for pt in yellow_pts1:

        i+=1

        cv2.circle(img1_with_corners,(int(pt[0]),int(pt[1])),2,color=(0,204,204),thickness=2)

    cv2.imwrite(path+'img1_selected_pts.jpg',img1_with_corners)

    i=0

    for pt in yellow_pts2:

        i+=1

        cv2.circle(img2_with_corners,(int(pt[0]),int(pt[1])),2,color=(0,204,204),thickness=2)

    cv2.imwrite(path+'img2_selected_pts.jpg',img2_with_corners)
    print('just wrote yellow pts')'''


    pts1 = list(map(lambda x: [x[1],x[0]], pts1));pts2 = list(map(lambda x: [x[1],x[0]], pts2))


    #Rectification::

    T1 = findT(pts1);# print('T1 = \n', T1)

    pts1_n = np.matmul(T1, np.array(list(map(lambda x: [x[0], x[1], 1], pts1))).T);

    T2 = findT(pts2);

    pts2_n = np.matmul(T2, np.array(list(map(lambda x: [x[0], x[1], 1], pts2))).T)

    F = findF(pts1_n.T, pts2_n.T, T1, T2) #Initial estimate

    [F,P_dash] = lm(np.array(list(map(lambda x: [x[0],x[1],1],pts1))),np.array(list(map(lambda
x:[x[0],x[1],1],pts2))),F)

    [rect_img1, rect_img2, F, pts1, pts2, H1, H2, P_dash] = rectify(pts1, pts2, img1, img2, F)

    #print('F = \n{}\nP2=\n{}'.format(F,P_dash))

    '''img1_with_corners=rect_img1; i = 0

    for pt in pts1:

        i+=1

        cv2.circle(img1_with_corners,(int(pt[0]),int(pt[1])),2,color=(0,0,255),thickness=2)

cv2.putText(img1_with_corners,str(i),(int(pt[0]),int(pt[1])),fontFace=cv2.FONT_HERSHEY_SIMPLEX,fontS
cale=1,color=(0,0,255),thickness=2)

    cv2.imwrite(path+'rectimg1_with_corners.jpg',img1_with_corners)

    img2_with_corners=rect_img2; i=0;

    for pt in pts2:

        i+=1

        cv2.circle(img2_with_corners,(int(pt[0]),int(pt[1])),2,color=(0,0,255),thickness=2)

cv2.putText(img2_with_corners,str(i),(int(pt[0]),int(pt[1])),fontFace=cv2.FONT_HERSHEY_SIMPLEX,fontS
cale=1,color=(0,0,255),thickness=2)

    cv2.imwrite(path+'rectimg2_with_corners.jpg',img2_with_corners)
    print('just wrote rect imgs with corners')'''


    #Finding edges, corners and correspondences

    gray_rect1 = make_gray(rect_img1); gray_rect2 = make_gray(rect_img2)

    edges1 = cv2.Canny(gray_rect1, 255*1.5, 255)

    edges1 = remove_corner_edges(edges1);cv2.imwrite(path + 'edges1.jpg', edges1)

    edges2 = cv2.Canny(gray_rect2, 255*1.5, 255)

    edges2 = remove_corner_edges(edges2);cv2.imwrite(path + 'edges2.jpg', edges2)

    corners1 = get_corners(edges1); corners2 = get_corners(edges2);

    [corresp,nccs] = ncc(corners1,corners2,rect_img1, rect_img2)

    '''corresp_img = np.hstack((rect_img1, rect_img2))

    for i in range(len(corresp)):

        cv2.line(corresp_img, (corresp[i][0][1], corresp[i][0][0]),

        (corresp[i][1][1]+img1.shape[1], corresp[i][1][0]), color = (0,0,255), thickness = 1)

    cv2.imwrite(path + 'corresps.jpg', corresp_img)'''

    #Do LM with these correspondences:

    corr1=[];corr2=[]

    for i in range(len(corresp)):

        corr1.append([corresp[i][0][0],corresp[i][0][1],1])

        corr2.append([corresp[i][1][0],corresp[i][1][1],1])

    #print('corr1[0] = ',corr1[0])

    [F,P_dash] = lm(corr1,corr2,F)


    #world_pts = triangulate(corresp, P_dash)

    world_pts =
triangulate([[(62,253),(38,266)],[(75,259),(55,272)],[(87,263),(73,275)],[(103,271),(90,280)],

                        [(113,277),(103,286)],[(177,300),(176,302)],[(174,303),(170,302)]], P_dash) #selected
points on front face of box

    x_world_c = np.array(list(map(lambda x: x[0], world_pts)))
    y_world_c = np.array(list(map(lambda x: x[1], world_pts)))
    z_world_c = np.array(list(map(lambda x: x[2], world_pts)))


    border_pts = []
    for i in range(len(pts1)):
        border_pts.append([pts1[i],pts2[i]])
    #print('border_pts=\n{}'.format(border_pts))
    border_world_pts = triangulate(border_pts, P_dash)
    #print('border_world_pts=\n{}'.format(border_world_pts))
    x_world = np.array(list(map(lambda x: x[0], border_world_pts)))
    y_world = np.array(list(map(lambda x: x[1], border_world_pts)))
    z_world = np.array(list(map(lambda x: x[2], border_world_pts)))
    #print('x_world=\n{}\ny_world=\n{}\nz_world=\n{}'.format(x_world,y_world,z_world))


    #print('world_pts = \n{}'.format(world_pts))


    #Plotting those points:
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    #ax.scatter(x_world_c,y_world_c,z_world_c,color='y',zdir='y')
    ax.scatter(x_world[:7],y_world[:7],z_world[:7],color='k',zdir='y')
    #for i in range(len(x_world)):
    for i in range(7):
      ax.text(x_world[i],z_world[i],y_world[i],str(i+1),zdir='y')
    ax.plot([x_world[0],x_world[1]],[y_world[0],y_world[1]],[z_world[0],z_world[1]],color='b',zdir='y')
    ax.plot([x_world[6],x_world[2]],[y_world[6],y_world[2]],[z_world[6],z_world[2]],color='b',zdir='y')
    ax.plot([x_world[4],x_world[3]],[y_world[4],y_world[3]],[z_world[4],z_world[3]],color='g',zdir='y')
    ax.plot([x_world[0],x_world[6]],[y_world[0],y_world[6]],[z_world[0],z_world[6]],color='b',zdir='y')
    ax.plot([x_world[1],x_world[2]],[y_world[1],y_world[2]],[z_world[1],z_world[2]],color='b',zdir='y')
    ax.plot([x_world[2],x_world[3]],[y_world[2],y_world[3]],[z_world[2],z_world[3]],color='r',zdir='y')

    ax.plot([x_world[6],x_world[4]],[y_world[6],y_world[4]],[z_world[6],z_world[4]],color='r',zdir='y')

    ax.plot([x_world[0],x_world[5]],[y_world[0],y_world[5]],[z_world[0],z_world[5]],color='r',zdir='y')

    ax.plot([x_world[5],x_world[4]],[y_world[5],y_world[4]],[z_world[5],z_world[4]],color='g',zdir='y')

    ax.view_init(None,30)

    ax.set_xlabel('x');ax.set_ylabel('y');ax.set_zlabel('z')

    plt.savefig(path+'border_pts.png')

#================================================================================

if __name__ == '__main__':

    main()

#================================================================================
```