ECE 661 – Homework 9

Ran Xu

xu943@purdue.edu

11/19/2018

1. Math Reductions and Algorithms

In this homework, we want to reconstruct the 3D scene using epipolar geometry and then find the point correspondences with image rectification.

1.1 Projective Reconstruction

Firstly, we manually label 8 point correspondences between the binocular stereo images, denoting $\vec{x}_i = (u_i, v_i, 1)^T$ and $\vec{x}_i' = (u_i', v_i', 1)^T, i = 0,1,\ldots,7$ in 3-dimentional HC.

Secondly, to derive the Fundamental Matrix F, we leverage the equation

$$\vec{x}_i'^T F \vec{x}_i = 0$$

We flatten the 3x3 Fundamental Matrix F in a vector f, where $f = [F_{11}, F_{12}, F_{13}, F_{21}, F_{22}, F_{23}, F_{31}, F_{32}, F_{33}]$. And thus each point correspondence $(u_i, v_i, 1)^T$ and $(u_i', v_i', 1)^T$ have one equation,

$$[u_i'u_i \quad u_i'v_i \quad u_i' \quad v_i'u_i \quad v_i'v_i \quad v_i' \quad u_i \quad v_i \quad 1] \cdot f = 0$$

Stacking all 8 equations, we have

$$Af = 0$$

The optimal solution of f that minimizes Af and subjects to $\|f\| = 1$ is given by the eigenvector of the A^TA corresponding to the smallest eigenvalue,

$$f = V_9, where\ A = UDV^T = U \begin{bmatrix} D_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & D_9 \end{bmatrix} \begin{bmatrix} V_1^T \\ \vdots \\ V_9^T \end{bmatrix} and\ D_1 \geq \cdots \geq D_9$$

Thirdly, we enforce the rank 2 of Fundamental Matrix F by zeroing its smallest singular value,

$$F_{refined} = U_f D_{refined} V_f^T = U \begin{bmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} V_f^T, where\ F = U_f D_f V_f^T = U \begin{bmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & D_3 \end{bmatrix} V_f^T$$

The right-image epipole $\vec{e}'$ is also given in this singular value decomposition because it is the left null-vector of $F_{refined}$ and thus it is the third column of U_f.

$$\vec{e}' = U_{3f}$$

We initialize the projection matrix to the left and right image plane P and P' as follows,

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, P' = [[\vec{e}']_x F \mid \vec{e}']$$

The estimated world coordinates $\vec{X}_i$ follows the following 4 equations in terms of image coordinates $\vec{x}_i = (u_i, v_i, 1)^T$, $\vec{x}_i' = (u_i', v_i', 1)^T$, and projection matrixes $P = \begin{bmatrix} \vec{P}_1^T \\ \vec{P}_2^T \\ \vec{P}_3^T \end{bmatrix}$, $P' = \begin{bmatrix} \vec{P}_1'^T \\ \vec{P}_2'^T \\ \vec{P}_3'^T \end{bmatrix}$

$$B\vec{X}_i = \begin{bmatrix} u_i\vec{P}_3^T - \vec{P}_1^T \\ v_i\vec{P}_3^T - \vec{P}_2^T \\ u_i'\vec{P}_3'^T - \vec{P}_1'^T \\ v_i'\vec{P}_3'^T - \vec{P}_2'^T \end{bmatrix} \vec{X}_i = 0$$

The optimal solution of world coordinates $\vec{X}_i$ that minimizes $B\vec{X}_i$and subjects to $\|\vec{X}_i\| = 1$ is given by the eigenvector of the $B^T B$ corresponding to the smallest eigenvalue,

$$\vec{X}_i = V_4, where\ B = UDV^T = U \begin{bmatrix} D_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & D_4 \end{bmatrix} \begin{bmatrix} V_1^T \\ \vdots \\ V_4^T \end{bmatrix}\ and\ D_1 \geq \cdots \geq D_4$$

1.2 LM optimization

To refine the right-projection matrix P' and the estimated world X_i, we use LM optimization to minimize the summed re-projected error.

Firstly, there are $3n + 12$ optimization variables in the LM optimization, where n is the number of point correspondence ($n = 8$ in this homework) and 12 is the number of unknown parameters in P'. Note that P stays unchanged in the optimization.

Secondly, the cost function in LM optimization is defined as the sum of re-projecting the estimated world coordinates to the left image plane and re-projecting the estimated world coordinates to the right image plane as follows,

$$d_{geom}^2 = \sum_{i=0}^{i<8} \|P\vec{X}_i - \vec{x}_i\|_2 + \|P'\vec{X}_i - \vec{x}_i'\|_2, where\ \|\cdot\|_2\ represents\ the\ l2 - norm\ of\ physical\ corrdinate$$

Finally, we solve the optimization using LM method and minimize the reprojection error.

1.3 Image Rectification and Interest Point Detection

Firstly, we use SIFT interest point detector to find interest point in the left and right images.

Secondly, to rectify the right image, the purpose is to map the right epipole to $[1,0,0]^T$. The homography H' is the dot product of four transformation matrices:

$$H' = T_2 GRT$$

where T sends the image center to origin, R rotates the right epipole to $[f, 0,1]^T$, G sends $[f, 0,1]^T$ to $[1,0,0]^T$ and T_2 recovers origin back to the image center.

Thirdly, to rectify the left image, the purpose is to map the left epipole to $[1,0,0]^T$. The homography H is given by

$$H = \begin{bmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $a, b, and\ c$ minimize,

$$\sum (au_i + bv_i + c - u_i')^2$$

Fourthly, given a pair of feature vector $f_1(i)$ and $f_2(i)$ extracted from left and right image respectively, we define the Sum of Squared Differences (SSD) as

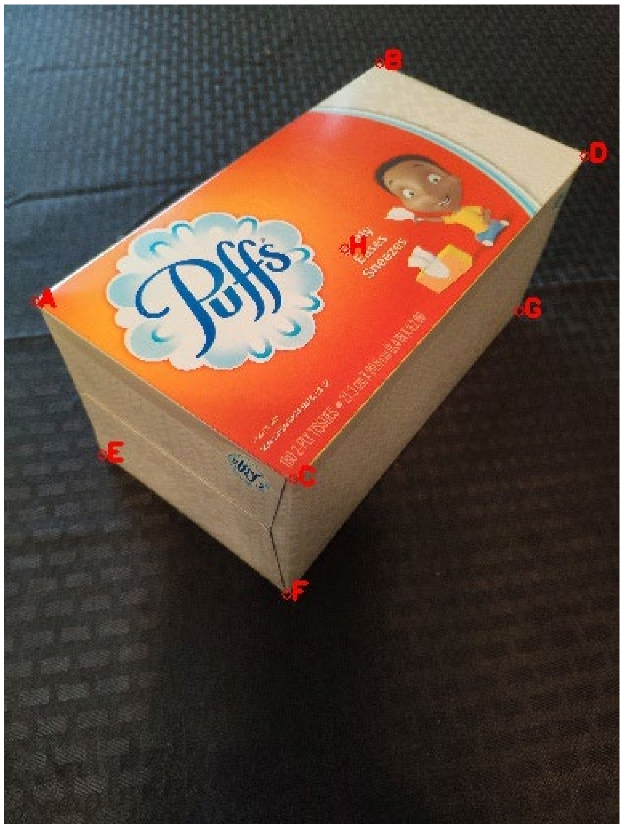
$$SSD = \sum \left(f_1(i) - f_2(i)\right)^2$$

Finally, after the image is rectified, for each interest point in the left image, we search for the minimal SSD value of the interest point in the right image within 3 nearby rows to serve as the matched point correspondences. The matched SSD metric is noted down for final screen process. The screen process selects the point correspondences with the highest quality, i.e. the smallest SSD value.

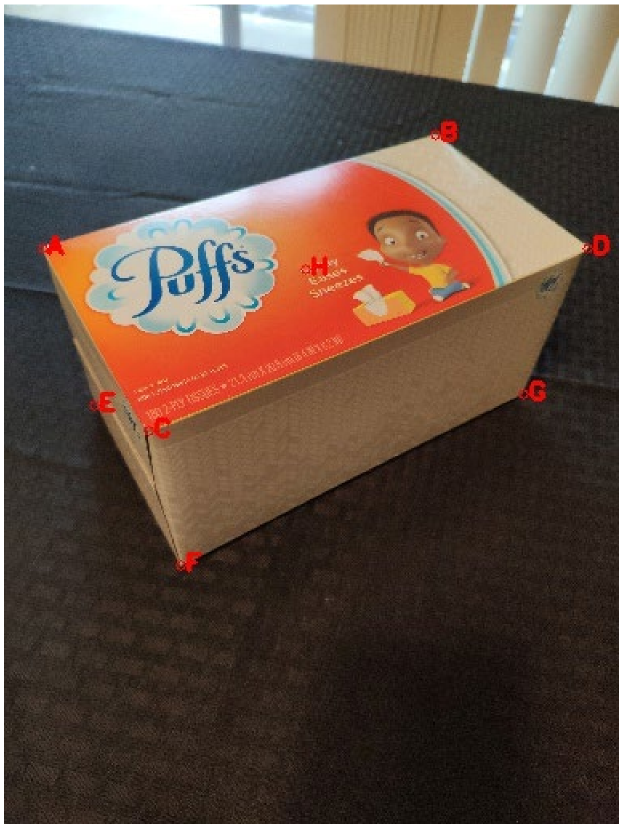
2. Evaluations

2.1 Projective Reconstruction

I manually selected 8 points in both left and right images to estimate the fundamental matrix F. The point correspondences are labeled with same letter.



(a)



(b)

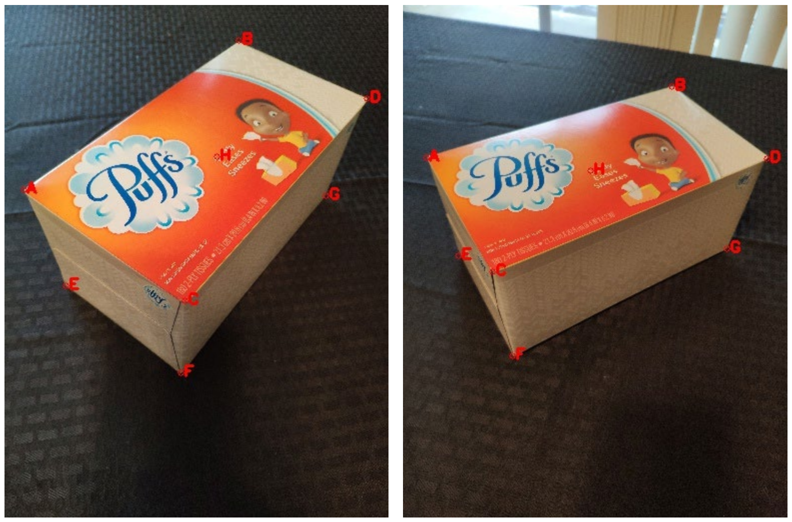
Figure 1: The 8 manually selected 8 points labelled "A" to "H" to indicate the correspondences in (a) left image (b) right image

The estimated fundamental matrix F is,

$$F = \begin{bmatrix} 7.517 \times 10^{-6} & -7.874 \times 10^{-5} & -4.477 \times 10^{-2} \\ 1.176 \times 10^{-5} & -4.923 \times 10^{-5} & 1.497 \times 10^{-1} \\ 1.360 \times 10^{-2} & -1.148 \times 10^{-1} & 1 \end{bmatrix}$$

2.2 Improvements using LM optimization

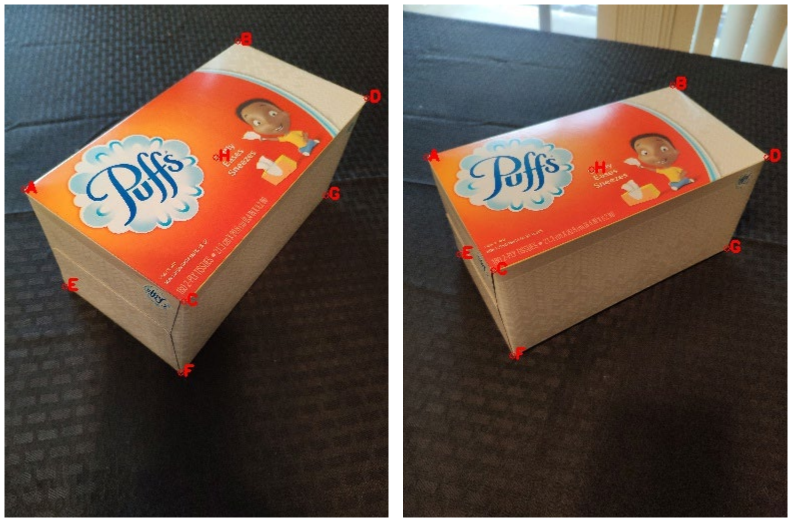
Before the LM optimization, the re-projection of the 3D reconstructed scene onto 2 image planes are shown in Figure 2. There is some small reprojection error in the right image and cost function $d_{geom}^2 = 1.37372$.



(a) (b)

Figure 2: Before the LM optimization, the re-projection of the 3D reconstructed scene onto (a) left image plane (b) right image plane

After the LM optimization, the re-projection of the 3D reconstructed scene onto 2 image planes are shown in Figure 3. It is obvious that now all the 8 points precisely match the one that human annotates in Figure 1. The cost function after optimization is $d_{geom}^2 = 0.00515$, which is less than 1% of the error before LM optimization. The improvement through LM optimization is significant.

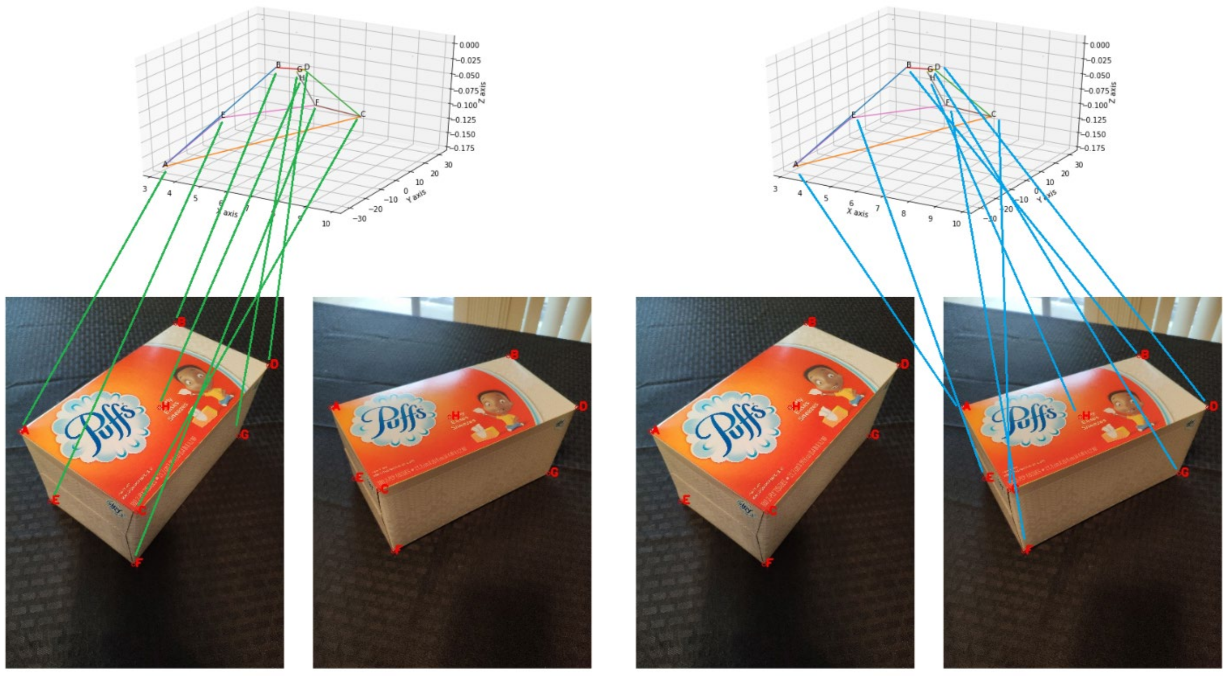


(a) (b)

Figure 3: After the LM optimization, the re-projection of the 3D reconstructed scene onto (a) left image plane (b) right image plane

2.3 3D Visual Inspection

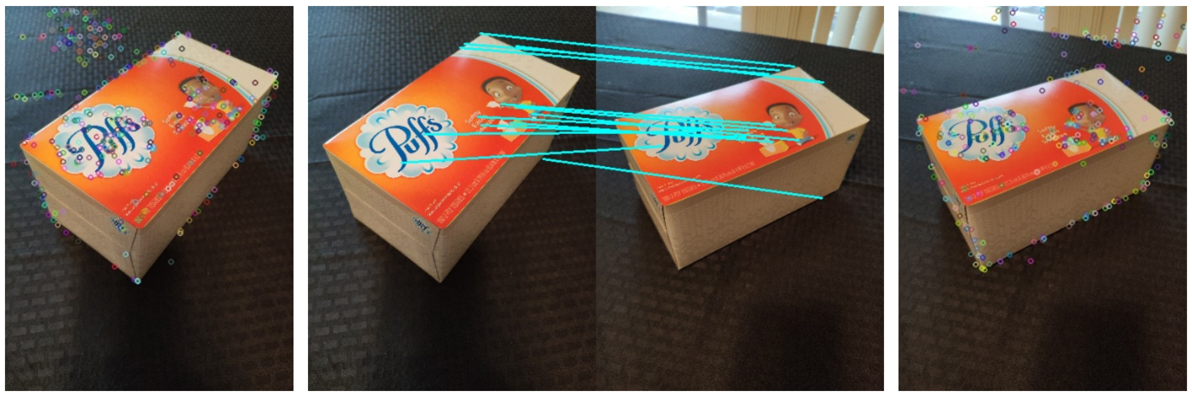
In Figure 4, I showed the reconstructed scene in 3D view, due to the projective distortion in reconstructed 3D scene, it is very hard to image its original 3D shape by watching the reconstructed scene. However, we can still observe the general structure by pairing the 8 correspondence and the 3D topology is preserved in the 3D reconstructed scene.



(a) (b)

Figure 4: Reconstructed 3D scene with point correspondence with (a) the left image and (b) the right image

2.4 Matched SIFT Correspondences after Image Rectification



(a) (b) (c)

Figure 5: (a) detected SIFT key points in the left image (b) 15 pairs of top-matched SIFT correspondences after image rectification (c) detected SIFT key points in the right image

In Figure 5, I showed the original SIFT key points in both left and right images and the top 15 matched results after the image rectification. All the correspondences are matched correctly with ease after the image rectification because we can match the correspondence in the nearby rows of right image form that of left image.

3. Source code

The left and right images are named “s1.jpg” and “s2.jpg”.

3.1 Helper Functions

```
def Solve_F(KPL, KPR):  # KPL/KPR is (3,n) shape, third row must be 1
    Npts = KPL.shape[1]
    A = np.zeros((Npts, 9))
    for idx in range(Npts):
        A[idx, 0] = KPR[0, idx]*KPL[0, idx] # x' * x
        A[idx, 1] = KPR[0, idx]*KPL[1, idx] # x' * y
        A[idx, 2] = KPR[0, idx]             # x' * 1
        A[idx, 3] = KPR[1, idx]*KPL[0, idx] # y' * x
        A[idx, 4] = KPR[1, idx]*KPL[1, idx] # y' * y
        A[idx, 5] = KPR[1, idx]             # y' * 1
        A[idx, 6] = KPL[0, idx]             # x
        A[idx, 7] = KPL[1, idx]             # y
        A[idx, 8] = 1                       # 1
    # Solve F
    u, s, vh = np.linalg.svd(A)
    f = vh[-1, :]   # The eigenvector corresponding to the smallest eigen value
    F = np.array([[f[0], f[1], f[2]], [f[3], f[4], f[5]], [f[6], f[7], f[8]]])/f[8]
    # Enforce F to be rank 2
    u, s, vh = np.linalg.svd(F)
    s[-1] = 0
    F = np.dot(np.dot(u, np.diag(s)), vh)
    LeftNull = u[:, 2]
    return F, LeftNull

import scipy.optimize
def LossFunc(p, KPL, KPR, F, PL):  # Shapes: (12), (3,N)
    PR = p[0:12].reshape(3,4)
    Npts = int(p.shape[0]-12)/3
    X = np.vstack([p[12:].reshape(3, Npts), np.ones((1, Npts))])

    # Compute Error
    Est_KPL = np.dot(PL, X)
    Est_KPL = Est_KPL/Est_KPL[2,:]
    Est_KPR = np.dot(PR, X)
    Est_KPR = Est_KPR/Est_KPR[2,:]
    DiffL = Est_KPL - KPL
    DiffR = Est_KPR - KPR
    Diff = np.hstack([DiffL.flatten(), DiffR.flatten()])
    Cost = np.sum(Diff**2)/2
    return Diff

def PrintKP(img_name, KP, prefix): # Print the 8 key pts on images and label with "A"
to "H"
    # img is (H, W, 3), KP is (3, 8) shape
    KP2D = KP[0:2,:]/KP[2,:]
    img = cv.imread('%s.jpg' %img_name)
    Labels = ["A", "B", "C", "D", "E", "F", "G", "H"]
    for idx in range(8):
        # Draw key points: KP2D[0, idx], KP2D[1, idx]
        x_kp = KP2D[0, idx]
        y_kp = KP2D[1, idx]
        cv.circle(img, center = (int(x_kp), int(y_kp)), radius = 3, color = (0, 0, 25
5))
        # Draw labels: Labels[idx]
        cv.putText(img, Labels[idx], (int(x_kp+3), int(y_kp+3)),
                   cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
    cv.imwrite('save_8kp_%s_%s.jpg' %(prefix,img_name), img)

def SSD(f1, f2):
    return np.sum((f1-f2)**2)

def KPDist(KP1, KP2, F, th = 3): # (3,1), (3,1), (3,3)
    # Epipolar line in image 2
    KPL = np.array([[KP1[0]], [KP1[1]], [1]])
    KPR = np.array([[KP2[0]], [KP2[1]], [1]])
    EpiLineR = np.dot(F, KPL)
    Dist = np.dot(EpiLineR.reshape(1,3), KPR)/np.sqrt(EpiLineR[0]**2+EpiLineR[1]**2)
    return np.abs(Dist[0][0]) > th

def Correspondence_Dist(FM1, Coord1, FM2, Coord2, F):
    #(#,128), [i].pt=(2,), (##,128), [i].pt=(2,)
    N1 = FM1.shape[0]
    N2 = FM2.shape[0]
    Pairs = []
    Scores = []
    for idx1 in range(N1):
        Score = np.array([SSD(FM1[idx1,:], FM2[idx2,:]) for idx2 in range(N2)])
        OutRange = [KPDist(Coord1[idx1].pt, Coord2[idx2].pt, F, th = 3) for idx2 in ra
nge(N2)]
        Score = [x+1e10*y for (x,y) in zip(Score, OutRange)]
        idx2 = np.argmin(Score)
        Scores.append(np.min(Score))
        Pairs.append((idx1,idx2))
    BestPairs = [x for _,x in sorted(zip(Scores,Pairs))]  # Incresing order
    # Select 15 pairs
    return BestPairs[0:int(np.min([15,len(BestPairs)]))]

def Correspondence(FM1, Coord1, FM2, Coord2):
    #(#,128), [i].pt=(2,), (##,128), [i].pt=(2,)
    N1 = FM1.shape[0]
    N2 = FM2.shape[0]
    Pairs = []
    Scores = []
    for idx1 in range(N1):
        Score = np.array([SSD(FM1[idx1,:], FM2[idx2,:]) for idx2 in range(N2)])
        idx2 = np.argmin(Score)
        Scores.append(np.min(Score))
        Pairs.append((idx1,idx2))
    BestPairs = [x for _,x in sorted(zip(Scores,Pairs))]  # Incresing order
    # Select 15 pairs
    return BestPairs[0:int(np.min([15,len(BestPairs)]))]

def DrawPairs(input_image1, input_image2, Coord1, Coord2, Pairs, prefix):
    left = cv.imread("%s.jpg" %input_image1)
    right = cv.imread("%s.jpg" %input_image2)
    output = np.concatenate((left, right), axis = 1)
    for (idx1, idx2) in Pairs:
        x1 = int(Coord1[idx1].pt[0])
        y1 = int(Coord1[idx1].pt[1])
        x2 = int(Coord2[idx2].pt[0]) + left.shape[1]
        y2 = int(Coord2[idx2].pt[1])
        cv.line(output, (x1,y1),(x2,y2), (255,255,0), 2)
    cv.imwrite("%sRect_%s_%s.png" %(prefix, input_image1, input_image2), output)
```

3.2 Main Function - Projective Reconstruction & LM Optimization

```
# HW09 -- Projective Reconstruction
# Author: Ran Xu (xu943@purdue.edu)

# Manual selection of 8 key points on both image
import numpy as np
img1_name, img2_name = ("s1", "s2")
KPL = np.array([[156, 1845,1422,2853,483 ,1386,2532,1671],    # X cord. (--)
                [1464,288, 2337,741, 2214,2901,1506,1197],    # Y cord. (|)
                [1,   1,   1,   1,   1,   1,   1,   1]])      # HC: all ones
KPR = np.array([[195, 2124,714, 2874,447, 870 ,2556,1485],    # X cord. (--)
                [1203,639, 2100,1200,1974,2757,1917,1305],    # Y cord. (|)
                [1,   1,   1,   1,   1,   1,   1,   1]])      # HC: all ones
KPL[0:2, :] = KPL[0:2, :]/7.875
KPR[0:2, :] = KPR[0:2, :]/7.875

import cv2 as cv
# Unit test -- read image
img1 = cv.imread("%s.jpg" %img1_name)
img2 = cv.imread("%s.jpg" %img2_name)
print("Image shapes (H, W, C) = ", img1.shape, img2.shape)

PrintKP(img1_name, KPL, "Init")
PrintKP(img2_name, KPR, "Init")
F, eR = Solve_F(KPL, KPR)  # (3,3), (3,)
print("F=", F)

# Solve P, P' with F, eR
PL = np.array([[1,0,0,0], [0,1,0,0], [0,0,1,0]])  # [I_3x3, 0]
eRx = np.array([[0, -eR[2], eR[1]], [eR[2], 0, -eR[0]], [-eR[1], eR[0], 0]])
PR = np.hstack([np.dot(eRx, F), eR.reshape(3,1)]) # [e']xF | e'

# Estimate X
Npts = KPL.shape[1]
X = np.zeros((4, Npts))
for idx in range(Npts):
    xL, yL = (KPL[0, idx], KPL[1, idx])
    xR, yR = (KPR[0, idx], KPR[1, idx])
    A = np.zeros((4, 4))
    A[0, :] = xL * PL[2, :] - PL[0, :]
    A[1, :] = yL * PL[2, :] - PL[1, :]
    A[2, :] = xR * PR[2, :] - PR[0, :]
    A[3, :] = yR * PR[2, :] - PR[1, :]
    u, s, vh = np.linalg.svd(A)
    X[:, idx] = vh[-1, :]/vh[-1, -1]

# PreLM
Est_KPL = np.dot(PL, X)
Est_KPR = np.dot(PR, X)
PrintKP(img1_name, Est_KPL, "PreLM")
PrintKP(img2_name, Est_KPR, "PreLM")

p_init = np.hstack([PR.flatten(), X[0:3,:].flatten()])
Diff = LossFunc(p_init, KPL, KPR, F, PL); Cost = np.sum(Diff**2)/2
print("LM optimization starts! Initial Cost = %.5f" %Cost)

# PostLM
sol = scipy.optimize.least_squares(LossFunc, p_init, method = 'lm', args = [KPL, KPR,
F, PL])
Diff = LossFunc(sol.x, KPL, KPR, F, PL); Cost = np.sum(Diff**2)/2
print("LM optimization ends! Initial Cost = %.5f" %Cost)
```

```
p = sol.x
PR = p[0:12].reshape(3,4)
Npts = int(p.shape[0]-12)/3
X = np.vstack([p[12:].reshape(3, Npts), np.ones((1, Npts))])
Est_KPL = np.dot(PL, X)
Est_KPR = np.dot(PR, X)
PrintKP(img1_name, Est_KPL, "PostLM")
PrintKP(img2_name, Est_KPR, "PostLM")
```

3.3 Main Function - 3D Reconstruction

```
# 3D print
Est_KPL = Est_KPL/Est_KPL[2,:]
Est_KPR = Est_KPR/Est_KPR[2,:]
np.set_printoptions(3)

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

lines = [[1, 2], [1, 3], [3, 4], [2, 4], [1, 5], [3, 6], [5, 6], [6, 7], [4, 7]]
fig = plt.figure(figsize=(10,6))
ax = fig.gca(projection='3d')
Labels = ["A", "B", "C", "D", "E", "F", "G", "H"]

for idx in range(8):
    plt.scatter(X[0,idx], X[1,idx], X[2,idx])
    ax.text(X[0,idx], -X[1,idx], -X[2,idx], Labels[idx], zdir = None)
for line in lines:
    Xs = [X[0, idx-1] for idx in line]
    Ys = [-X[1, idx-1] for idx in line] # Rotate y so that it matches original directi
on
    Zs = [-X[2, idx-1] for idx in line] # Rotate z so that it matches original directi
on
    plt.plot(Xs, Ys, Zs)
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')
plt.savefig("Recon3D.png")
ax.view_init(azim=-70)
```

3.4 Main Function – Image Rectification and Interest Point Matching

```
# Unit test -- read image, get SIFT KPs, and draw
sift = cv.xfeatures2d.SIFT_create()

img = cv.imread('%s.jpg' %img1_name)
img1_gray = cv.imread('%s.jpg' %img1_name, cv.IMREAD_GRAYSCALE)
kp1, des1 = sift.detectAndCompute(img1_gray, None)
cv.drawKeypoints(img,kp1,img)
cv.imwrite('KP_%s.jpg' %img1_name, img)

img = cv.imread('%s.jpg' %img2_name)
img2_gray = cv.imread('%s.jpg' %img2_name, cv.IMREAD_GRAYSCALE)
kp2, des2 = sift.detectAndCompute(img2_gray, None)
cv.drawKeypoints(img,kp2,img)
cv.imwrite('KP_%s.jpg' %img2_name, img)

Pairs = Correspondence(des1, kp1, des2, kp2)
DrawPairs(img1_name, img2_name, kp1, kp2, Pairs, "Pre")
Pairs = Correspondence_Dist(des1, kp1, des2, kp2, F)
DrawPairs(img1_name, img2_name, kp1, kp2, Pairs, "Post")
```