ECE 661 Homework 9

Zeeshan Nadir

email: **znadir@purdue.edu**

Due Date: November 27, 2014

1 Introduction

In this homework, we shall perform 3D scene reconstruction using stereo images. The reconstructions that we shall compute will be related to the world 3D with a projective distortion. Such reconstruction is called projective reconstruction. Obviously because of this, the reconstruction is going to look projectively distorted. If the scene is rich, we can remove projective distortion and then later affine distortion. There are quite a few many steps involved to perform the reconstruction and we shall explain all of them one by one. First we shall list all the main steps and then explain them.

1. Estimate the fundamental matrix F using manually selected points on both images through **linear least squares** optimization.

2. Using the estimated F, we shall rectify the images to send the epipoles to $e = [1 \quad 0 \quad 0]^T$.

3. Using the rectified images, we shall find interest points on the rectified images through **canny edge detector**.

4. Once the interest points are found on the rectified images, we apply the **non-linear least squares optimization** to improve the fundamental matrix, camera matrices and the 3D world points.

5. Finally we shall use triangulation to project the point correspondences to world 3D.

2 Linear Least Squares Estimation of Fundamental Matrix F

We estimate the fundamental matrix first using manual correspondences that are selected by the user. We denote the correspondences by $\left(x_i, {x'}_i^T\right)$. Here ${x'}_i$ is a pixel location in homogeneous coordinate in the right image and x_i is a pixel location in homogeneous coordinate in the left image. F is the fundamental matrix in homogeneous coordinates. We know from the theory of epipolar geometry that

$$ {x'}_i^T F x_i = 0 \tag{1} $$

We denote eq. (1) using the following form

$$ \left[x'_i x \quad x'_i y \quad x'_i \quad y'_i x_i \quad y'_i y_i \quad y'_i \quad x_i \quad y_i \quad 1\right] f = 0 \tag{2} $$

where $f = [F_{11} \quad F_{12} \quad F_{13} \quad F_{21} \quad F_{22} \quad F_{23} \quad F_{31} \quad F_{32} \quad F_{33}]$ We require 8 correspondences to use the normalized 8-point algorithm. This algorithm normalizes the data to improve the estimate of fundamental matrix F. Following are the main steps involved in estimating F.

(i) First we find normalization homographies T_1 and T_2 for the two images such that all the pixel correspondences are 0 mean and have a distance of $\sqrt{2}$ from the center i.e. $(0, 0)$. The homography T_1 is used for points x_i and homography T_2 is used for points x'_i.

(ii) When we have normalized all the points, we stack up all of them in the form of eq. (2) and make a matrix vector equation of the form $Af = 0$. The matrix A has all the stacked rows. This is solved using SVD to yield matrix F.

(iii) The rank of matrix F has to be 2. Therefore we may need to condition the matrix F to make its rank 2. This is done again by using Singular Value Decomposition. So if $F = UDV^T$, we set the smallest singular value in D equal to 0 and then denote it with $\widehat{D}$. F is then set to $F = U\widehat{D}V^T$.

(iv) Finally we denormalize the fundamental matrix F by using the following relation

$$F = T_2^T F T_1$$

(v) Compute the epipoles e and e' of the left and right images respectively. They are respetively the right and left null vectors of fundamental matrix F.

(vi) Finally we also calculate the camera matrices as follows

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3}$$

$$P' = \left[[e']_x F | e'\right] \tag{4}$$

3 Image Rectification

In order to compute our projectively distorted 3D scene structure, we must refine our estimate for matrix F. This needs finding large number of pixel correspondences $\left(x_i, x'^T_i\right)$ on the two images. It is often very helpful if we can simply lookup for pixel correspondences along the same rows (at best) or for each pixel in one image, we may look in a small number of adjoining rows in the second image. This is done by sending the epipoles in both the images to infinity. We compute the homographies H_1 and H_2 to send the epipoles e and e' to infinity. Following is procedure that is carried out to do this.

(i) First of all we shift the second image to origin using homography T_1. This makes the application of rotation straight forward which we would need later.

(ii) Find the angle of the epipole w.r.t x-axis and rotate the the entire image so that epipole goes to x-axis i.e.

$$e' = \begin{bmatrix} f \\ 0 \\ 1 \end{bmatrix} \tag{5}$$

(iii) Then use the homography $G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/f & 0 & 1 \end{bmatrix}$ to send the epipole to infinity along x-axis i.e.

$$e' = \begin{bmatrix} f \\ 0 \\ 0 \end{bmatrix} \tag{6}$$

(iv) Finally translate back the image to its original center point using homography T_2.

(v) The homography that shall be applied onto the second image to accomplish all these tasks is then given by

$$H_2 = T_2 G R T_1 \tag{7}$$

(vi) The homoraphy for first image is found using a linear least squares minimization procedure to minimize the sum of squares distances given by

$$\sum_i d\big(H_1 x_i, H_2 x_i'\big) \tag{8}$$

This is done in order to force the corresponding epipolar lines to be on the same rows.

(vii) The details of the procedure are given in Sec 11.12.12 of Hartley and Zisserman textbook. Here we shall only explain the procedure at a descriptive level.

(a) Let $M = P'P^+$

(b) Let $H_0 = H_2 M$ and $H_A = \begin{bmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

(c) Let $\widehat{x}_i = H_0 x_i$ and $\widehat{x}_i' = H_2 x_i'$

(d) Select a, b, c such that they minimize the following sum of squares

$$\sum_i \Big(\widehat{x}_i + b\widehat{y}_i + c - \widehat{x}_i'\Big)^2 \tag{9}$$

(e) Finally the homography H_1 for first image is then given by $H_1 = H_A H_0$.

4 Interest Point Detection Using Canny Edge Detector

Now that we have rectified the images, we can look for interest poitns in the two images using Canny Edge Detector. Finding large number of interest points is important. Since these shall be used to establish correspondences. This is relatively easy with rectified images since all the interest points are usually on the same rows or within a distance of couple of rows. Following is the procedure that we have adapted to find the interest points and the point correspondences.

(i) Use canny edge detector to find edges in both the images.

(ii) Use the pixels corresponding to edges as our interest points.

(iii) Usually this results in a very large number of pixels, and since I am coding in MATLAB, I shall randomly pick up 1500 pixels corresponding to edges in each point and then prune them for finding correspondences.

(iv) To find correspondence, for each interest point in one image, we look up for the interest point in the second image that gives the highest value of NCC score. We also check that that interest point in the second image should be withing a couple of rows of the interest point in the first image.

5 Projective Reconstruction using Triangulation and Refinement using Levenberg Marquardt algorithm

This section covers the details about how the final pixel correspondences are converted to points in world 3D. In general if we back project two corresponding pixels from images of the same scene into two different rays in world 3D, the two rays may not intersect at all. Therefore we need to refine the estimate for the fundamental matrix F and also the world 3D points that are projected back to the images for the calculation of square of the difference with the measured pixel locations. Following is the geometric distance that we need to minimize

$$d_{\text{geom}}^2 = \sum_i \left(\|x_i - \widehat{x}_i\|^2 + \|x'_i - \widehat{x}'_i\|^2 \right) \tag{10}$$

where $\widehat{x}_i$ and $\widehat{x}'_i$ are the projected points in the first and the second image respectively. We shall use Levenberg Marquardt (LM) Algorithm to perform this non-linear optimization. Following are important steps involved.

(i) Triangulate the 3D point X_i from the 2D points $\left(x_i, x'^T_i\right)$ using the linear triangulation method. This shall later be used as an initial condition for the nonlinear LM optimization.

  (a) For each correspondence $\left(x_i, x'^T_i\right)$, form the matrix

$$A = \begin{bmatrix} x_i P^{3^T} - P^{1^T} \\ y_i P^{3^T} - P^{2^T} \\ x'_i P'^{3^T} - P'^{1^T} \\ y'_i P'^{3^T} - P'^{2^T} \end{bmatrix}$$

  (b) Our goal is to minimize $\|AX\|$ subject to $\|X\| = 1$.

  (c) This is given by the null vector of A. Hence the world 3D point X_i is given by the null vector of matrix A.

(ii) Using this as an initial estimate for our world points, we apply Levenberg Marquardt Algorithm to minimize the geometric distance given in eq. (10).

(iii) Finally once we have found all the world 3D points, we plot them.

6 Observations

- Setting the manual correspondences accurately is important for good estimate of fundamental matrix F.
- The 3D scene that we shall reconstruct would appear to be projectively distorted as described in class notes. This is because we are using canonical configuration of the camera and this results in reconstruction upto a projective distortion. We are not removing this distortion.
- The canny edge detector gives a very large number of edge pixels. This can make the overall reconstruction process very slow.
- Canny edge detector can lead to spurious correspondences because a lot of pixels on the edges may appear to be very similar. Specially in case when we have smooth regions on both sides of the edges. In such cases SIFT or SURF may prove to be better.
- Image rectification seems to be working pretty well since after we rectify the images, the correspondences appear on rows that are very close to each other.
- LM optimization improves the estimate of fundamental matrix and the world points.
- Note that the accuracy of end results also depend upon the kind of the scene we are trying to capture. The rich the scene is in structure, the easier it is to find interest points and make sense out of results.

7 Results

Now we shall show our results step by step and explaining them as necessary. We shall also explain the process of rectifying the images, benefit of LM optimization and 3D scene reconstruction.

Figure 1: Input Image 1

Figure 2: Input Image 2

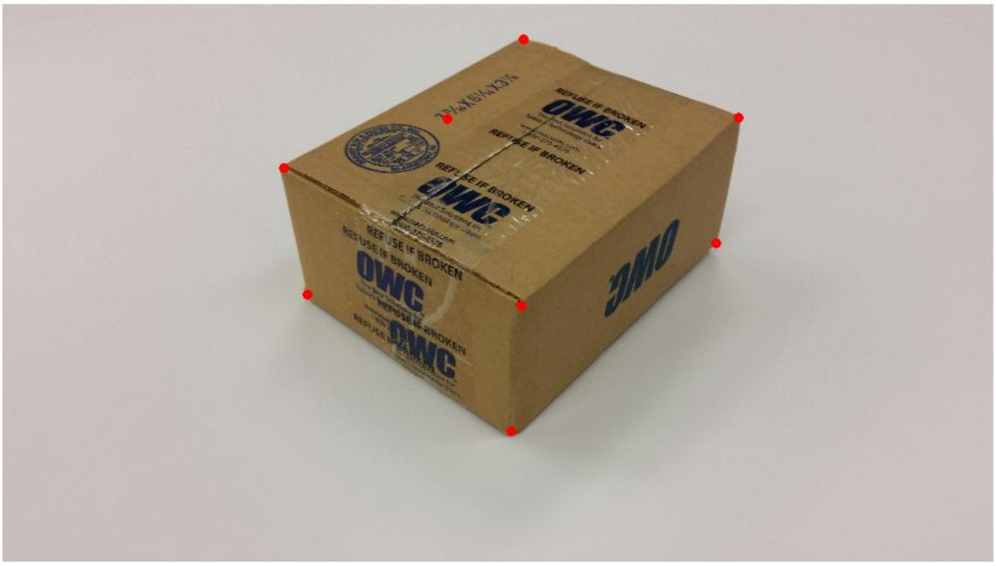

Figure 3: Manually selected interest points on Input Image 1

Figure 4: Manually selected interest points on Input Image 2

Now we shall show the pixel correspondence between the interest points in Fig. 3 and Fig. 4 to first see the correspondences before the rectification is done and later see after rectification is done.

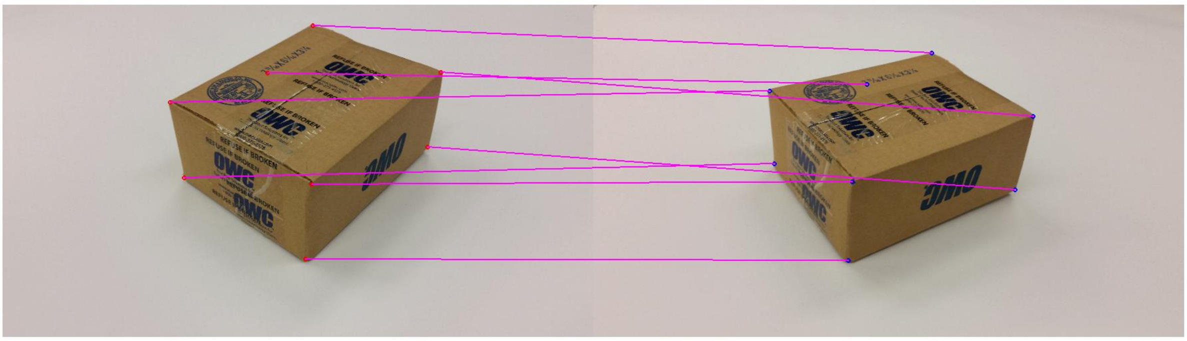

Figure 5: Manual Point Correspondences between the two images **before** image rectification. Note that since epipoles are at finite locations, interest points don't appear to be horizontally aligned.

Figure 6: Rectified Image 1

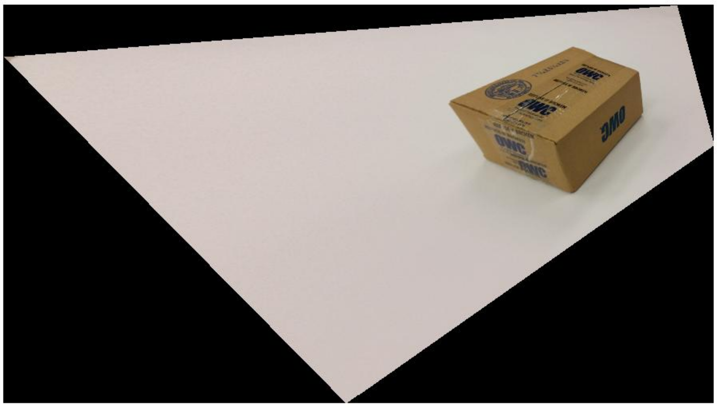

Figure 7: Rectified Image 2

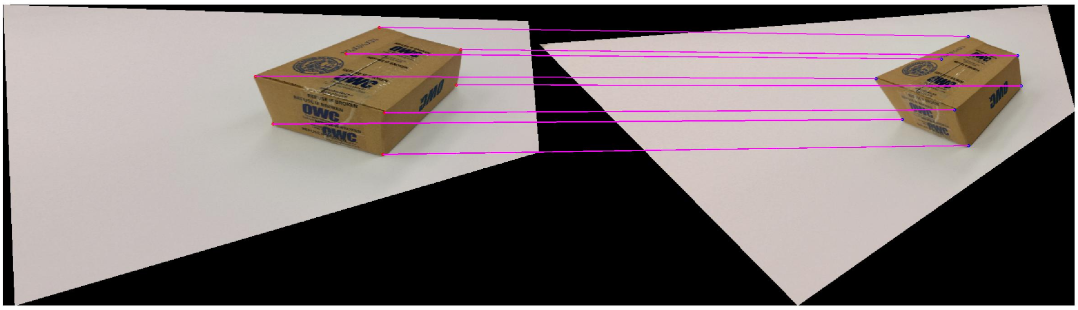

Figure 8: Rectified Images displayed side by side along with point correspondences

Note that once we rectify the images, the point correspondences are very close in row number. We have yet to select a large number of point correspondences later with canny edge detector. However this is the appropriate place to show how the epipoles go to infinity along x-axis

Initial Fundamental Matrix before images are rectified is given by

$$F = \begin{bmatrix} 0.000002545830537 & 0.000017246081557 & 0.000909294100453 \\ -0.000014110404458 & 0.000007499340541 & -0.010209160424781 \\ -0.000024816877215 & 0.006022136207114 & -0.319278007547744 \end{bmatrix} \tag{11}$$

Initial First Camera Matrix is given by

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{12}$$

Initial Second Camera Matrix is given by

$$P' = \begin{bmatrix} 0.0004316768 & -0.0009689965 & 0.11178361271 & 1.39818781565 \\ 0.00006248474 & 0.00412699751 & -0.20757452638 & 0.25987560446 \\ 0.0000087557 & -0.00000236433 & 0.00695571184 & -0.00432772621 \end{bmatrix} \tag{13}$$

Initial epipoles are given as

$$e = \begin{bmatrix} -696.8688 \\ 50.1456 \\ 1.0000 \end{bmatrix} \text{ and } e' = \begin{bmatrix} -323.0768 \\ -60.0490 \\ 1.0000 \end{bmatrix} \tag{14}$$

Final Fundamental Matrix after images are rectified is given by

$$F = \begin{bmatrix} -0.000000000000000 & -0.000000000000000 & 0.000000000000000 \\ 0.000000000000000 & 0.000000000000000 & -0.066690020983087 \\ -0.000000000000000 & 0.057357725258866 & 1.362867393896769 \end{bmatrix} \times 10^2 \tag{15}$$

Final Second Camera Matrix is given by

$$P' = \begin{bmatrix} -0.0000000000 & 0.0000000000 & 0.0000000000 & 0.010000000 \\ 0.0000000000 & -0.0573577252 & -1.3628673938 & 0.000000000 \\ 0.0000000000 & 0.0000000000 & -0.0666900209 & 0.000000000 \end{bmatrix} \times 10^2 \tag{16}$$

Final epipoles after image rectification are given as

$$e = \begin{bmatrix} 0.061 \\ 0 \\ 0 \end{bmatrix} \text{ and } e' = \begin{bmatrix} -458.0768 \\ 0 \\ 0 \end{bmatrix} \tag{17}$$

Please note that all the above entities are in homogeneous coordinates. So their absolute value does not matter a lot. Also note that there is no point of showing the final value of P matrix since that remains the same.

Next we shall the results of Canny Edge Detector.

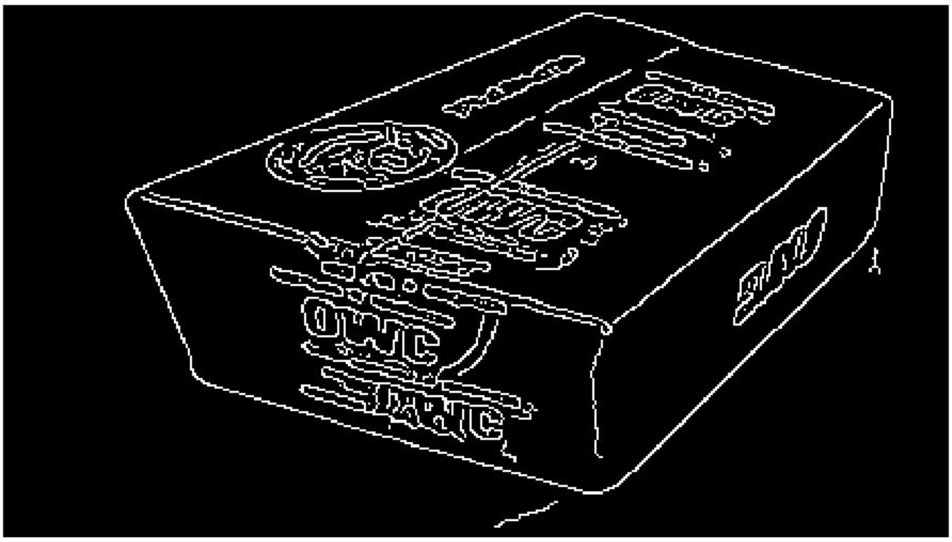

Figure 9: Edges/Interest Points found through Canny Edge Detector in Image 1

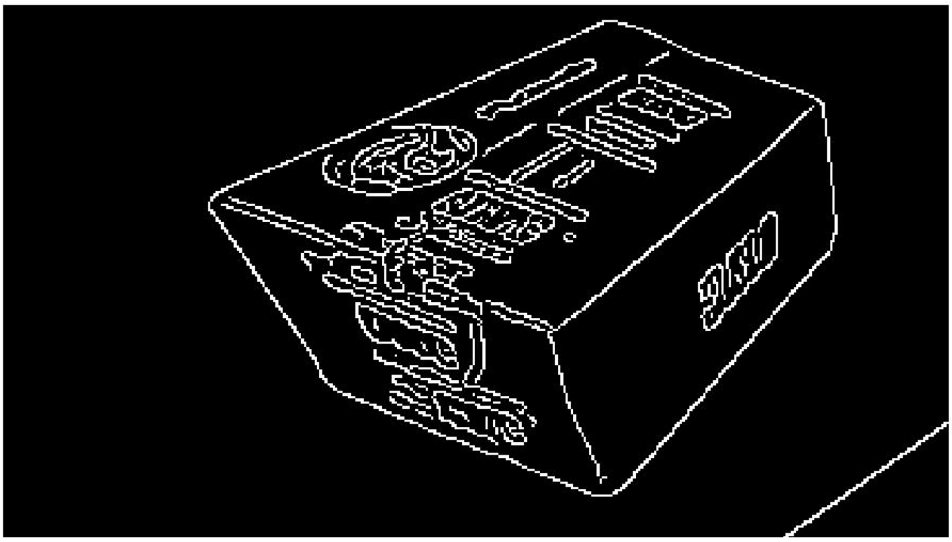

Figure 10: Edges/Interest Points found through Canny Edge Detector in Image 2

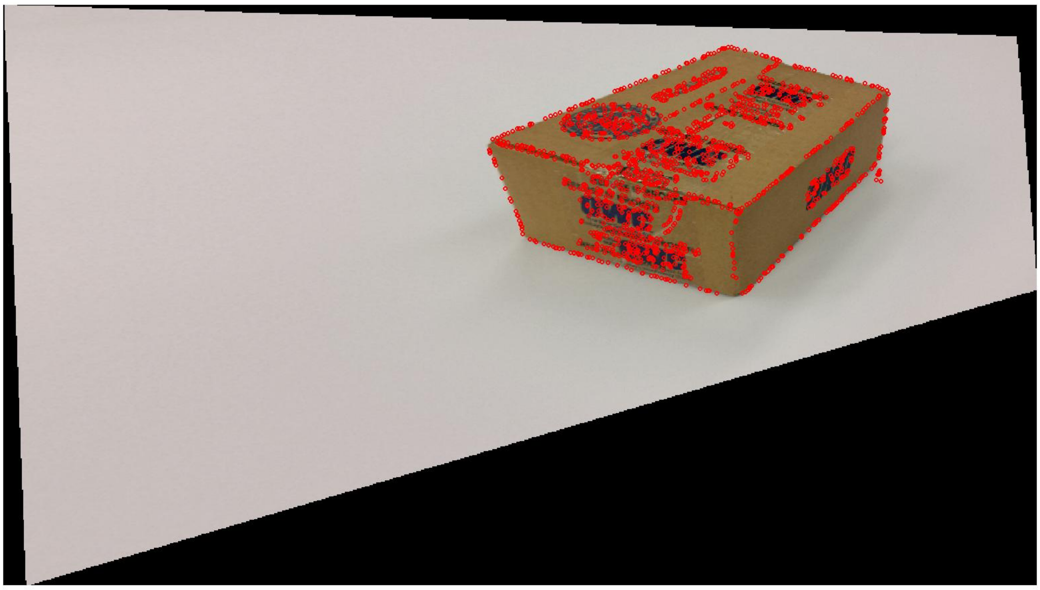

Figure 11: Interest Points found through Canny Edge Detector of Image 1 shown on the image

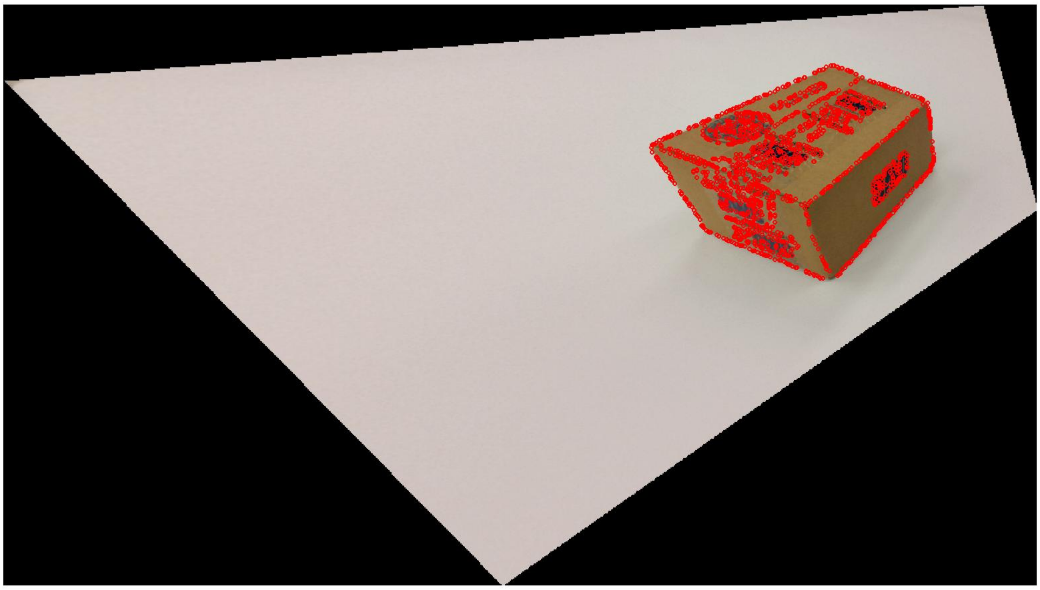

Figure 12: Interest Points found through Canny Edge Detector of Image 2 shown on the image

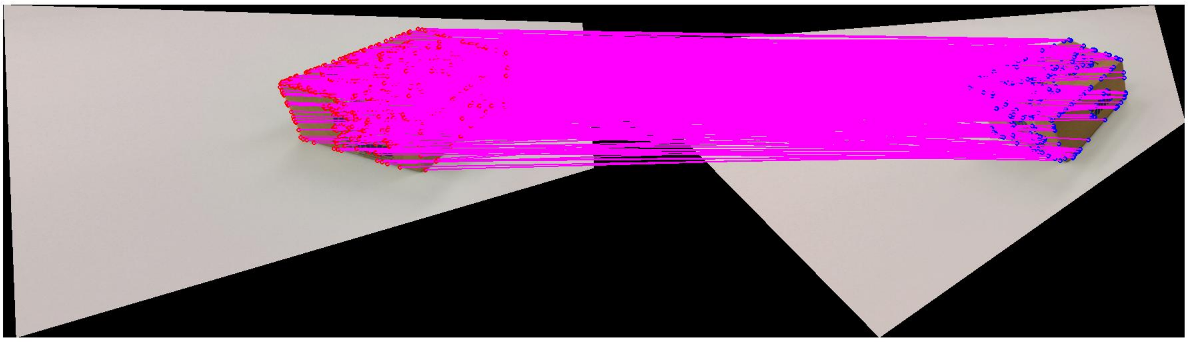

Figure 13: Point Correspondences

We can see that most of the point correspondences lie on horizontal lines which means that there row numbers is pretty close to each other. This is the benefit we get from image rectification.

Next we shall show the 3D scene reconstruction using triangulation. Of course we use Levenberg Marquardt algorithm to refine the world points.

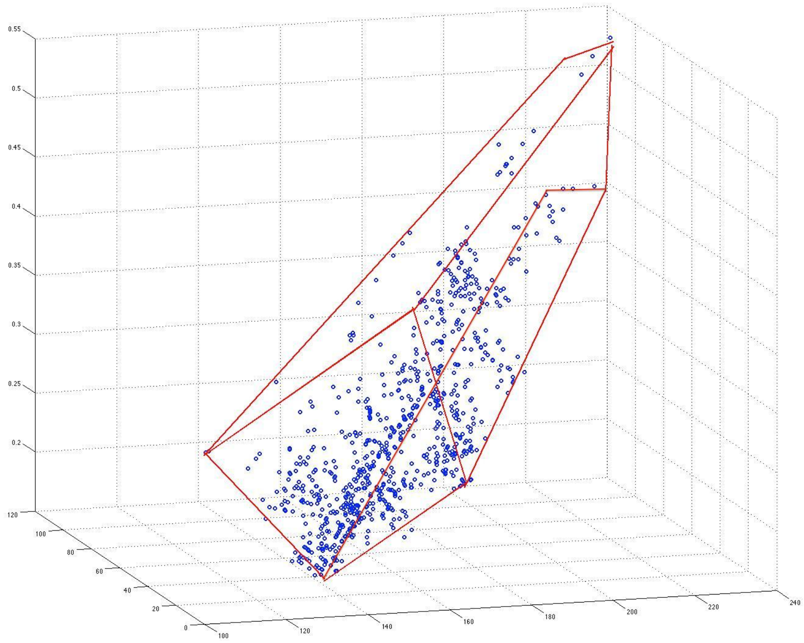

Figure 14: 3D Scene Reconstruction using scatter plot in MATLAB

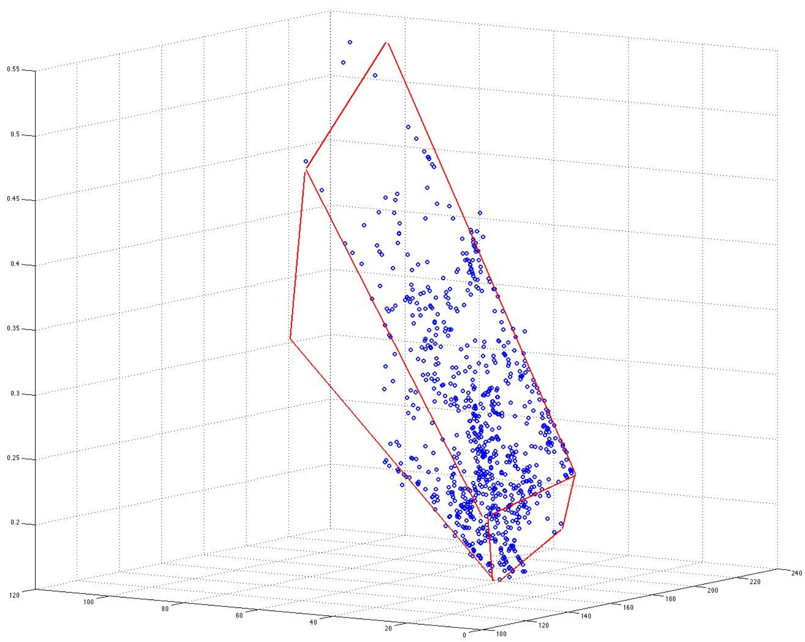

Figure 15: 3D Scene Reconstruction using scatter plot in MATLAB in another view

Note that the reconstructed image looks distorted since we haven't removed the projective distortion. Besides one has to rotate the 3D scatter plot to get a clear picture of the 3D scene in mind.

We shall first show the two images side by side like page 267 of Hartley and Zisserman text.

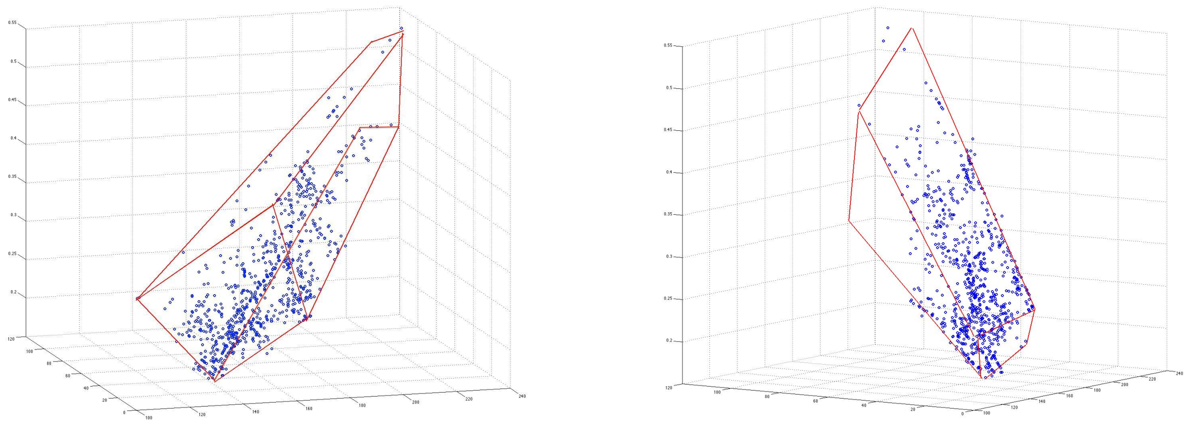

Figure 16: Two different views of 3D projective scene reconstruction displayed side by side. We can easily notice that the the 3D reconstruction we have is projectively distorted version of real 3D scene.

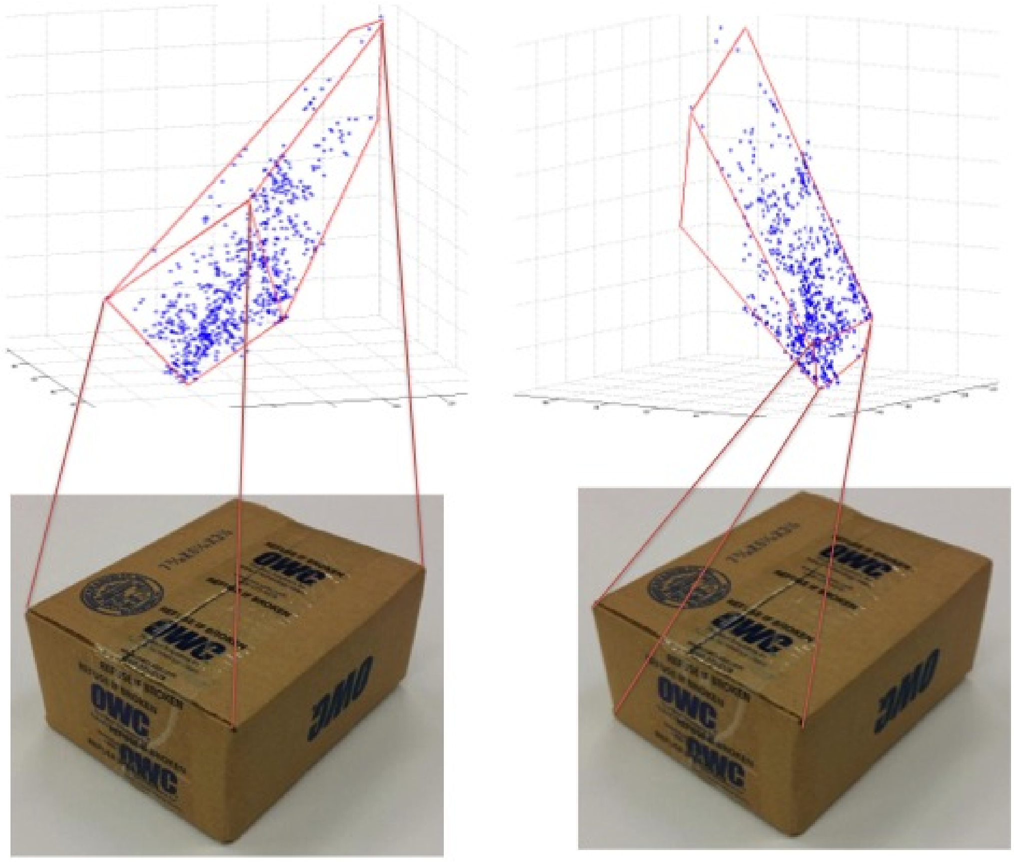

Figure 17: 3D Scene Point Correspondences with the Image 1

Improvement by the use of Levenberg Marquardt

The improvement that I noticed because of the use of Levenberg Marquardt Algorithm is in the estimate of F matrix and hence P' matrix. If I don't use Levenberg Marquardt Algorithm, then once condition the fundamental matrix F to make it rank 2, everything breaks down. Therefore its absolutely essential to perform non-linear optimization to fine tune the parameters.

Finally we show the correspondences between points on the images and also between those points and the 3D world points.

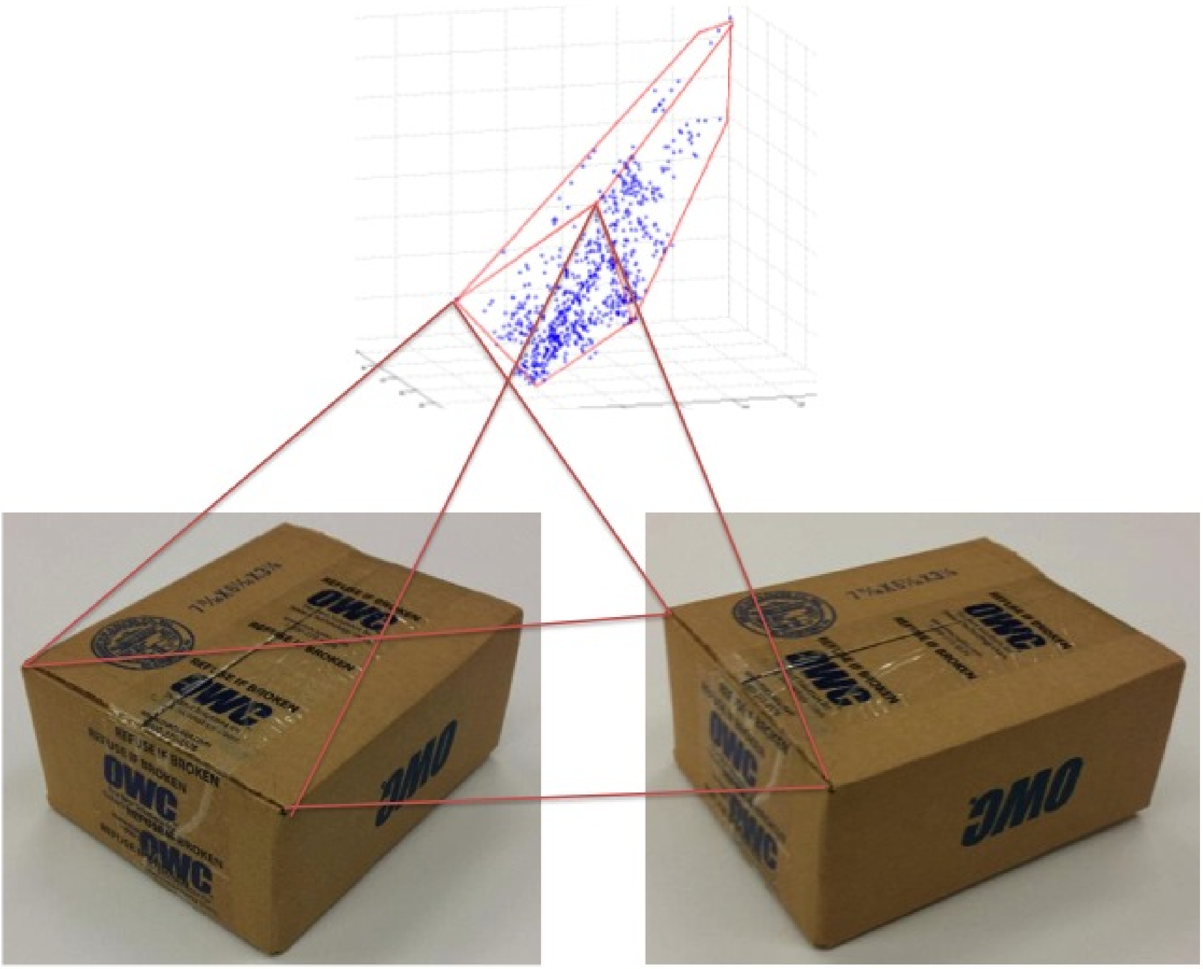

Figure 18: Point Correspondence between images and 3D world points

8 Source Code

Following are the codes of all the functions we have used.

Main Script

```
close all; clear;
path_load = ['/Users/zeeshannadir/purdue/ECE661/Fall 2014/hw 9/'];

im1 = imread([path_load '5.jpg']);
im2 = imread([path_load '6.jpg']);

% Downsample the 16 Megapixel images
factor = 6;
im1 = DownSampling(im1,factor);
im2 = DownSampling(im2,factor);

% show the images
figure;
imshow(im1); axis on; f1 = gcf;
figure;
imshow(im2); axis on; f2 = gcf;
% ------------------------------ Important Parameters ...
    -----------------------------------

Total_Corresp = 8; % Total no. of manual correspondences needed
T_ncc = 0.70;    % Threshold for NCC
W_ncc = 11;      % Window for NCC
r_ncc = 0.99;    % Ratio for getting rid of false correspondences
thresh_F = 1e-16;% Threshold for numerical purposes since we may not get ...
    x'^T F x exactly equal to 0
% ...
    ------------------------------------------------------------------------------------

% Get the 8 corresponding points on each of the two images
%[x1 y1 x2 y2] = Select_Correspondences (f1,f2, Total_Corresp);
load ('Correspondences3.mat');
% Form vectors of correspondence points
x1 = [x1 y1 ones(Total_Corresp,1)];
x2 = [x2 y2 ones(Total_Corresp,1)];

[T1] = Normalize_Points (x1);
[T2] = Normalize_Points (x2);

% Plot the correspondences
Plot_Correspondences (im1,im2,x1,x2)

% First we find the fundamental matrix using linear least squares method
```

```
F = Find_Fundamental_Matrix_LLS (x1,x2,T1,T2);

% epipoles are given by e_1 and e_2
e_1 =  null(F);   % Right Null Vector
e_2 =  null(F.'); % Left Null Vector
% Find matrix form of e_2
e_2_x = [0 -e_2(3) e_2(2);e_2(3) 0 -e_2(1);-e_2(2) e_2(1) 0];

% Compute Camera Projection Matrices
P1 = [1 0 0 0;0 1 0 0;0 0 1 0];
P2 = [e_2_x*F , e_2];

% Apply Non-Linear least squares estimation to improve the estimate
% of F and P2 so far
[P2 F X] = nonLinearLeastSquaresOpt (@error_fun , x1, x2, P1, P2);

% Recalculate the epipoles
e_1 =  null(F);   % Right Null Vector
e_2 =  null(F.'); % Left Null Vector

% Rectify the Images
% I should also plot correspondences within the rectify images function
[im1_rect im2_rect F x1_new x2_new H1 H2] = Rectify_Images ...
    (e_1,e_2,x1,x2,im1,im2,P1,P2,F);
figure;
imshow([im1_rect im2_rect]);

% Convert the rectified images to Gray scale for NCC criterion
im1_rect_gray = single(rgb2gray(im1_rect));
im2_rect_gray = single(rgb2gray(im2_rect));

Plot_Correspondences (im1_rect,im2_rect,x1_new,x2_new)

[Edges1]=cannyEdgeDetector(im1_rect,0.05);
[Edges2]=cannyEdgeDetector(im2_rect,0.05);

% Make Descriptor function gives interest points along with a window around
% the interest points so that we could apply the NCC criterion later
[interest_points1] = Make_Descriptor(Edges1,im1_rect_gray,W_ncc,1);
[interest_points2] = Make_Descriptor(Edges2,im2_rect_gray,W_ncc,2);

Plot_Interest_points(im1_rect,interest_points1);
Plot_Interest_points(im2_rect,interest_points2);


% Establish Correspondence
[C] = Establish_Correspondence_NCC (interest_points1,interest_points2);
[x1_final x2_final] = Get_Final_InterestPoints_Using_NCC (C, ...
    interest_points1, interest_points2, r_ncc, T_ncc, F, thresh_F);
```

```

Plot_Correspondences (im1_rect,im2_rect,x1_final,x2_final);

[T1] = Normalize_Points (x1_final);
[T2] = Normalize_Points (x2_final);

% First we find the fundamental matrix using linear least squares method
F = Find_Fundamental_Matrix_LLS (x1_final,x2_final,T1,T2);

% epipoles are given by e_1 and e_2
e_1 =  null(F);    % Right Null Vector
e_2 =  null(F.'); % Left Null Vector
% Find matrix form of e_2
e_2_x = [0 -e_2(3) e_2(2);e_2(3) 0 -e_2(1);-e_2(2) e_2(1) 0];

% Compute Camera Projection Matrices
P1 = [1 0 0 0;0 1 0 0;0 0 1 0];
P2 = [e_2_x*F , e_2];

[P2 F X] = nonLinearLeastSquaresOpt (@error_fun , x1_final, x2_final, P1, P2);
%scatter3(X(:,1),X(:,2),X(:,3));
PlotWorldPoints(X,x1_final,x2_final,P1,P2,F,thresh_F);
```

Function for normalizing the interest points to be used in normalized 8 point algorithm.

```
function [T] = Normalize_Points (x)
mean_x = mean(x(:,1));
mean_y = mean(x(:,2));
% First find current distance to the mean
temp_std = 0;
for i = 1:size(x,1)
    temp_std = temp_std + sqrt((x(i,1)-mean_x)^2+(x(i,2)-mean_y)^2);
end
% Now normalize all the points using mean and distance
temp_std = temp_std/size(x,1);
scale = sqrt(2)/temp_std;
xtr = -scale*mean_x;
ytr = -scale*mean_y;
T = [scale 0 xtr;0 scale ytr;0 0 1];
end
```

Fundamental matrix estimation using linear least squares method

```
function [F] = Find_Fundamental_Matrix_LLS (x_1,x_2,T1,T2)
total_corresp = size(x_1,1);

% Find the normalized correspondences
```

```
x_1_t = (T1 * x_1.').';
x_2_t = (T2 * x_2.').';

A = zeros(total_corresp,9); % A is the total_corresp x 9 vector
for i=1:1:total_corresp
    A(i,:) = [x_2_t(i,1)*x_1_t(i,1) x_2_t(i,1)*x_1_t(i,2) x_2_t(i,1) ...
        x_2_t(i,2)*x_1_t(i,1) x_2_t(i,2)*x_1_t(i,2) ...
        x_2_t(i,2) x_1_t(i,1) x_1_t(i,2) 1];
end
% Perform SVD on A
[U D V] = svd(A);
% % F is the last column vector in V
F = reshape (V(:,end),3,3).';
% Condition the F matrix
[U1 D1 V1] = svd(F);
D1(end,end) = 0; % Make it a rank 2 by zeroing the last singular value
F = U1 * D1 * V1.' ;
F = T2.' * F * T1;
end
```

Non linear least squares optimization to fine tune the parameters of the fundamental matrix and finding the world points. Note that we are finding the initial condition for LM algorithm using eq. (11) within the non-linear least squares minimization. This is just to find the initial condition. The optimization procedure to find tune the 3D world points and the fundamental matrix is Lavenberg Marquardt Algorithm.

```
function [P2 F X] = nonLinearLeastSquaresOpt (error_fun_handle , x1, x2, P1, P2)
% Create a vector of all the parameters
%total parameters are 12 + 3*total_correspondences
p = [reshape(P2.',1,12)];
total_corresp = size(x1,1);
X = zeros(total_corresp,4); % Create a matrix of world points that shall be ...
    optimized over
X_temp = zeros(total_corresp,4);
for i = 1:size(x1,1)
    A = getA (P1,P2,x1(i,:),x2(i,:));
    [U,D,V] = svd(A);
    Xn = give_physical( V(:,4) ); % give_physical function converts from ...
        homog. to physical
    X_temp(i,:) = Xn.';
    p = [p Xn(1:3).'];
end

options = ...
    optimset('Algorithm','levenberg-marquardt','MaxFunEvals',1000,'MaxIter',1000);
p_updated = lsqnonlin(error_fun_handle,p,[],[],options,x1,x2);

```

```
P2 = reshape(p_updated(1:12),4,3)';
t = P2(:,4); %is this e2? it is
ex = [0 -t(3) t(2); t(3) 0 -t(1); -t(2) t(1) 0];
M = P2(:,1:3);
F = ex*M;

% return the world 3D points
counter = 13;
for i=1:1:total_corresp
X(i,:) = [p_updated(counter:counter+2) 1];
counter = counter+3;
end
end

function [A] = getA (P1,P2,x1,x2)
A =      [ (x1(1)*P1(3,:) - P1(1,:));
           (x1(2)*P1(3,:) - P1(2,:));
           (x2(1)*P2(3,:) - P2(1,:));
           (x2(2)*P2(3,:) - P2(2,:)) ];
end
```

Function for image rectification is given as follows

```
function [im1_rect im2_rect F x1_new x2_new H1 H2] = Rectify_Images ...
    (e1,e2,x1,x2,im1,im2,P1,P2,F)
[h w temp] = size(im1);
total_points = size(x1,1);
% Convert e2 from homogenous to physical coordinates.
e2 = give_physical(e2);
angle = atan (-(e2(2)-h/2)/(e2(1)-w/2));
f = cos(angle)*(e2(1) - w/2) - sin(angle)*(e2(2) - h/2);
R = [cos(angle) -sin(angle) 0;sin(angle) cos(angle) 0;0 0 1];
T = [1 0 -w/2;0 1 -h/2;0 0 1];
G = [1 0 0;0 1 0;-1/f 0 1];
H2 = G*R*T;

% Preserves the center after applying homography
center_point = [w/2 h/2 1].';
new_center = give_physical( H2 * center_point );
T2 = [1 0 w/2 - new_center(1);0 1 h/2 - new_center(2);0 0 1];
H2 = T2 * H2;


% Now compuate the homography for first image
M = P2 * ( P1.' * (P1 * P1.')^-1 );
H0= H2 * M;

x_1_hat = ones(size(x1));
```

```matlab
x_2_hat = ones(size(x2));

for i=1:1:total_points
    x_1_hat(i,:) = give_physical((H0 * (x1(i,:).')).');
    x_2_hat(i,:) = give_physical((H2 * (x2(i,:).')).');
end

% Perform the Linear Least Squares Estimation for HA
A = zeros(total_points,3);
b = zeros(total_points,1);
for i=1:1:total_points
    A(i,:) = [x_1_hat(i,1) x_1_hat(i,2) 1];
    b(i)   = x_2_hat(i,1);
end
x = (A.' * A )^-1 * A.' * b; % Least squares estimate step
HA = [x(1) x(2) x(3);0 1 0;0 0 1];
H1 = HA * H0;
% Preserves the center after applying homography
center_point = [w/2 h/2 1].';
new_center = give_physical( H1 * center_point );
T1 = [1 0 w/2 - new_center(1);0 1 h/2 - new_center(2);0 0 1];
H1 = T1 * H1;

% Update the fundamental matrix accordingly

[im1_rect H1]= applyHomography (H1,im1);
[im2_rect H2]= applyHomography (H2,im2);
F = (H2.')^-1 * F * (H1)^-1;

% update the interest points on the new plane
x1_new = zeros(size(x1));
x2_new = zeros(size(x2));

for i=1:1:total_points
    temp = give_physical(H1 * x1(i,:).');
    x1_new(i,:) = temp;
end
for i=1:1:total_points
    temp = give_physical(H2 * x2(i,:).');
    x2_new(i,:) = temp;
end

end
```

Canny Edge Detector Function

```matlab
function [BW] = cannyEdgeDetector(im,thresh)
img = rgb2gray(im);
```

```
BW = edge(img,'canny',thresh);
figure
imshow(BW)
end
```

Function for making the descriptors require for each of the interest points to be used later in NCC criterion.

```
function [D] = Make_Descriptor ( C_Points, f, W, label)
%% First Define Descriptor Structure
D(1).m      = uint32(0); % row number of interest point
D(1).n      = uint32(0); % column number of interest point
D(1).feat   = zeros(W,W); % feature window around the interest point
D(1).avg    = 0;          % Mean value of the window

% ------------ Make an Array of the location points from the logical matrix ...
    C_points ------------
height = size(C_Points,1);
width  = size(C_Points,2);
Total_Interest_Points = 1500;

% label is used to identify if the image is the left or right
% this is just to fool the algorithm and get rid of the edges
% that show up due to applying homography because we do zero filling
temp = logical(zeros(size(C_Points)));
if (label==1)
    temp(37:249,415:761) = C_Points(37:249,415:761);
end
if (label == 2)
    temp(51:235,554:798) = C_Points(51:235,554:798);
end
C_Points = temp;

Loc = find(C_Points(:)==1);
total_points = length(Loc);
temp = randperm(total_points);

for j=1:1:Total_Interest_Points
    i = temp(j);
    [ D(j).m D(j).n]=Find_Location(Loc(i),height);
end
% since there are thousands of interest points, we randomly select 1000
for i=1:1:Total_Interest_Points
    x = D(i).n;
    y = D(i).m;
    for n= -(W-1)/2 : 1 : (W-1)/2
        for m = -(W-1)/2 : 1 : (W-1)/2
           % if ( (x+n ≥ 1) && (x+n ≤ width) && (y+m ≥ 1) && (y+m ≤height))
```

```
                D(i).feat(m+(W-1)/2 + 1,n+(W-1)/2 + 1) = f (y+m,x+n);
                %else
                %     D(i).feat(m+(W-1)/2 + 1,n+(W-1)/2+1) =0;
                %end
            end
        end
        D(i).avg = mean(mean(D(i).feat));
    end
end
```

Function for establishing correspondence using NCC criterion

```
function [C] = Establish_Correspondence_NCC (D_1,D_2)
% Final interest points
x1=[];
x2=[];

C = zeros(length(D_1),length(D_2));

for i=1:1:size(C,1)
    i
    for j=i:1:size(C,2)
        C(i,j) = ( sum(sum(  (D_1(i).feat - D_1(i).avg) .* (D_2(j).feat - ...
            D_2(j).avg) )) ) / ...
            sqrt ( (sum(sum( (D_1(i).feat - D_1(i).avg).^2))) *  (sum(sum( ...
                (D_2(j).feat - D_2(j).avg).^2))) );

    end
end
end
```

Function that prunes all the correspondences and checks for NCC criterion and the constrain $x'^T F x = 0$ and returns the correspondences that pass this crteria.

```
function [x1 x2]= Get_Final_InterestPoints_Using_NCC ...
    (C,D_1,D_2,r_ncc,T_ncc,F,thresh_F)
x1=[];
x2=[];
last = size(C,2);
for i = 1:1:size(C,1)
    i
    [b1,i1] = max(C(i,:));
    [b2,i2] = max(C(i,[(1:i1-1) i1+1:last]));
    if (i2 >= i1)
        i2 = i2+1;
    end
    if  (b1 < T_ncc)
```

```
            % DO NOTHING;
        elseif ( (b2/b1) > r_ncc )
            % DO NOTHING;

        % elseif ...
            (epipolar_constraint(D_1(i).n,D_1(i).m,D_2(i).n,D_2(i).m,F,thresh_F)==0)
        % Since image rectifying is working well, we just define a row
        % threshold. When the interest points are within threshold, we accept
        % the correspondence

        elseif (abs(D_1(i).m - D_2(i).m) > 50)
            % DO NOTHING;
        else
            x1 = [x1;double(D_1(i).n) double(D_1(i).m) 1];
            x2 = [x2;double(D_2(i1).n) double(D_2(i1).m) 1];
        end
    end
end

function [c] = epipolar_constraint (a,b,c,d,F,thresh)
x1=double([a b 1].');
x2=double([c d 1].');
result = x2.' * F * x1;
if (result < thresh)
    c = 1;
else
    c=0;
end
c=1;
end
```

Function for the plotting the points in world 3D.

```
function [] = PlotWorldPoints (X,x1_orig,x2_orig,P1,P2,F,thresh_F)
x_world = [];
for i=1:1:size(X,1)
    x_1 = give_physical(P1 * X(i,:).');
    x_2 = give_physical(P2 * X(i,:).');
    x_world = [x_world;X(i,:)];
end
figure;
scatter3(x_world(:,1),x_world(:,2),x_world(:,3),'o','Linewidth',2);
disp(['Total points in world 3D = ' num2str(size(x_world,1))]);
end
```

Following is the function that applies homography

```
function [Y H_new] = applyHomography(H,X)
% This function applies a homography H onto the image X
% And returns the result in Y
% The important thing is that there may be scaling and shifting needed
% Rather than doing that explicitly,it combines that into a new homography H_new
% And returns that new homography for keeping record
X = single (X);
[height_orig width_orig temp] = size(X);

% First find the boundary of the resulting image
a = [1 1];
b = [size(X,2) 1];
c = [1 size(X,1) ];
d = [size(X,2) size(X,1)];

[i]=give_physical ( H * [a.';1] ); a_(1) = round(i(1));  a_(2) = round(i(2));
[i]=give_physical ( H * [b.';1] ); b_(1) = round(i(1)); b_(2) = round(i(2));
[i]=give_physical ( H * [c.';1] ); c_(1) = round(i(1)); c_(2) = round(i(2));
[i]=give_physical ( H * [d.';1] ); d_(1) = round(i(1)); d_(2) = round(i(2));

tx1 = min([a_(1) b_(1) c_(1) d_(1)]);
tx2 = max([a_(1) b_(1) c_(1) d_(1)]);

ty1 = min([a_(2) b_(2) c_(2) d_(2)]);
ty2 = max([a_(2) b_(2) c_(2) d_(2)]);

% compute the height and width of projected image into the world plane
height = (ty2-ty1);
width  = (tx2-tx1);
disp(['total height = ' num2str(height)]);
disp(['total width = ' num2str(width)]);

H_scale = [width_orig/width 0 0;0 height_orig/height 0;0 0 1];
H = H_scale * H;

[i]=give_physical ( H * [a.';1] ); a_(1) = round(i(1));  a_(2) = round(i(2));
[i]=give_physical ( H * [b.';1] ); b_(1) = round(i(1)); b_(2) = round(i(2));
[i]=give_physical ( H * [c.';1] ); c_(1) = round(i(1)); c_(2) = round(i(2));
[i]=give_physical ( H * [d.';1] ); d_(1) = round(i(1)); d_(2) = round(i(2));

tx1 = min([a_(1) b_(1) c_(1) d_(1)]);
tx2 = max([a_(1) b_(1) c_(1) d_(1)]);

ty1 = min([a_(2) b_(2) c_(2) d_(2)]);
ty2 = max([a_(2) b_(2) c_(2) d_(2)]);

```

```
tx = tx1; % now we have the proper offsets that could be added
ty = ty1; % now we have the proper offsets that could be added

T=[1 0 -tx+1;0 1 -ty+1;0 0 1];
H_new = T * H;
H_inv = H_new^-1;

Y = zeros(height_orig,width_orig,3);
for m = 1:1:height_orig
    m
    for n=1:1:width_orig
        [s]=give_physical ( H_inv * [n;m;1] );
        temp = biLinear(s(1),s(2),X);
        % check if bilinear didn't return zero, it may return zero if the index
        % where we want to interpolate is outside the domain of image plane
        Y(m,n,:) = temp; % note all three channels (rgb) are copied at once
    end
end

Y = uint8(Y);
end
```