ECE 661 (Fall 2016) - Computer Vision - HW 10

Debasmit Das

November 29, 2016

1 Introduction

We shall perform 3D reconstruction using stereo images. The reconstruction we compute shall be related to the world 3D co-ordinates through a projective distortion and is called a projective reconstruction. There are quite a few stops to carry out projective reconstruction

- We estimate the fundamental matrix F using manually selected points on the 2 images through **linear least squares** optimization.
- Using the estimated F, we rectify the images so that the epipoles are at $e = [1 \quad 0 \quad 0]^T$.
- Using the rectified images, we shall find interest points using **SURF** Interest Point detector.
- Once these interest points are found on the rectified images we apply **non-linear** least squares optimization to improve the fundamental matrix, camera matrix and 3D world points.
- Finally, we use triangulation to project point correspondences to world 3D.

2 Linear Least Squares Estimate of Fundamental Matrix F

We estimate F using manual correspondences selected by the user. The correspondences are denoted by (x_i, x_i'). x_i', x_i are corresponding pixels in the right and left image respectively. From the theory of epipolar geometry, we know -

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

The above equation is denoted as

$$[x_i'x_i \quad x_i'y_i \quad y_i'x_i \quad y_i'y_i \quad y_i' \quad x_i \quad y_i \quad 1]f = 0 \tag{2}$$

where $f = [F_{11} \quad F_{12} \quad F_{13} \quad F_{21} \quad F_{22} \quad F_{23} \quad F_{31} \quad F_{32} \quad F_{33} \quad]$. We require 8 such correspondences. The steps are as following for the initial estimate of F.

- We have to find normalization homographies T_1 and T_2 for the 2 images such that pixel correspondences have zero mean and have distance of $\sqrt{2}$ from the center. The homography T_1 is used for points x_i and T_2 for points x_i'.
- Equation (2) is then stacked up to find the solution of f. f is solved using SVD.
- The rank of F is 2. So F needs to be conditioned before it can be used. So we do SVD $F = UDV^T$ and set the smallest singular value in D to be zero to obtain $\hat{D}$. Then we set $F = U\hat{D}V^T$.

- Finally, we denormalize the F using the relation $F = T_2^T F T_1$
- We also have to compute the epipoles e and e' for the two images. They are respectively the right and left null vectors of F.
- Finally, the camera matrices are formed using the canonical form.

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

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

3 Image Rectification

In order for to do projective reconstruction, we have to refine F. This needs finding the large number of pixel correspondences. It is best done if we search for the correspondence in the same row or adjoining row. This is done by sending epipoles of both images to infinity . We have to compute homographies H_1 and H_2 to transform e and e' to infinity.

- The second image is shifted to origin using T
- The angle of epipole w.r.t x-axis is found and rotated so that epipole goes to x-axis i.e. $e' = [f \quad 0 \quad 1]^T$.
- Then the homography $G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{-1}{f} & 0 & 1 \end{bmatrix}$ is applied to send the epipole to infinity along x-axis i.e. $e' = [f \quad 0 \quad 0]^T$.
- The image is translated back to its original centre point using T_2.
- The homography is then applied to second image and is given by $H_2 = T_2 G R T_1$.
- H_1 is found through a least squares minimization technique to minimize $\sum_i d(H_1 x_i, H_2 x_i')$. this is done to force the corresponding epipolar lines to be on the same rows. The details of the minimization procedure is given in Sec. 11.12.12 of Hartley Zisserman textbook.

4 Interest Point Dectection

For interest point detection, we use SURF. To find correspondence used is NCC. we do not go into the details because it has already been implemented a number of times in previous homework.

5 Projective Reconstruction using Triangulation and Refinement using Levenberg Marquardt (LM) algorithm

This deals with how the final pixel correspondences are converted to points in world 3D. If we back project two corresponding pixels from images the rays might not intersect at all. Therefore, there is a need to refine the F and also the world 3D points that are projected back to the images. Following is the geometric distance we need to minimize

$$d_{geom}^2 = \sum_i (\|x_i - \hat{x_i}\|^2 + \left\|x_i' - \hat{x_i'}\right\|^2) \tag{5}$$

where $\hat{x_i}$ and $\hat{x_i'}$ are the projected points in the first and second images respectively. We shall use the LM algorithm to perform the non-linear optimization. The steps used are as follows -

- we triangulate the 3D point X_i from 2D points $(x_i, x_i'^T)$ using Linear triangulation method. This will be used as initial condition for LM refinement.
  - For each correspondence $(x_i, x_i'^T)$, we form the matrix

$$G = \begin{bmatrix} x_i P^{3^T} - P^{1^T} \\ y_i P^{3^T} - P^{2^T} \\ x_i' P'^{3^T} - P'^{1^T} \\ y_i' P'^{3^T} - P'^{2^T} \end{bmatrix} \tag{6}$$

  - Our goal is to minimize $\|AX\|$ such that $\|X\| = 1$
  - This is given by the null vector of A. The world 3D point X_i is given by the null vector of matrix A.
- With this initial estimate, we apply LM algorithm to minimize the geometric distance
- We apply this to all the 3D points and visualize them

6 Observations

- We have to set manual correspondences properly for the initial estimate of F.
- As described in class, we will obtain reconstruction till a projective distortion of the original scene. This is because a canonical configuration of camera matrices are used.
- Rectification seems to be proper since we obtain corresponding points in rows that are very close to each other.
- LM algorithm is carried out to refine the fundamental matrix and the world points.
- The reconstruction performance kind of depends on how much our scene is rich in features. In our case the features are less and therefore the performance is not up to the mark.

7 Results

We shall show the results step by step and explain as necessary. We shall also explain the process of rectifying images, using LM optimization and 3D reconstruction. The dataset used is from the images in http://vasc.ri.cmu.edu//idb/html/stereo/book/index.html

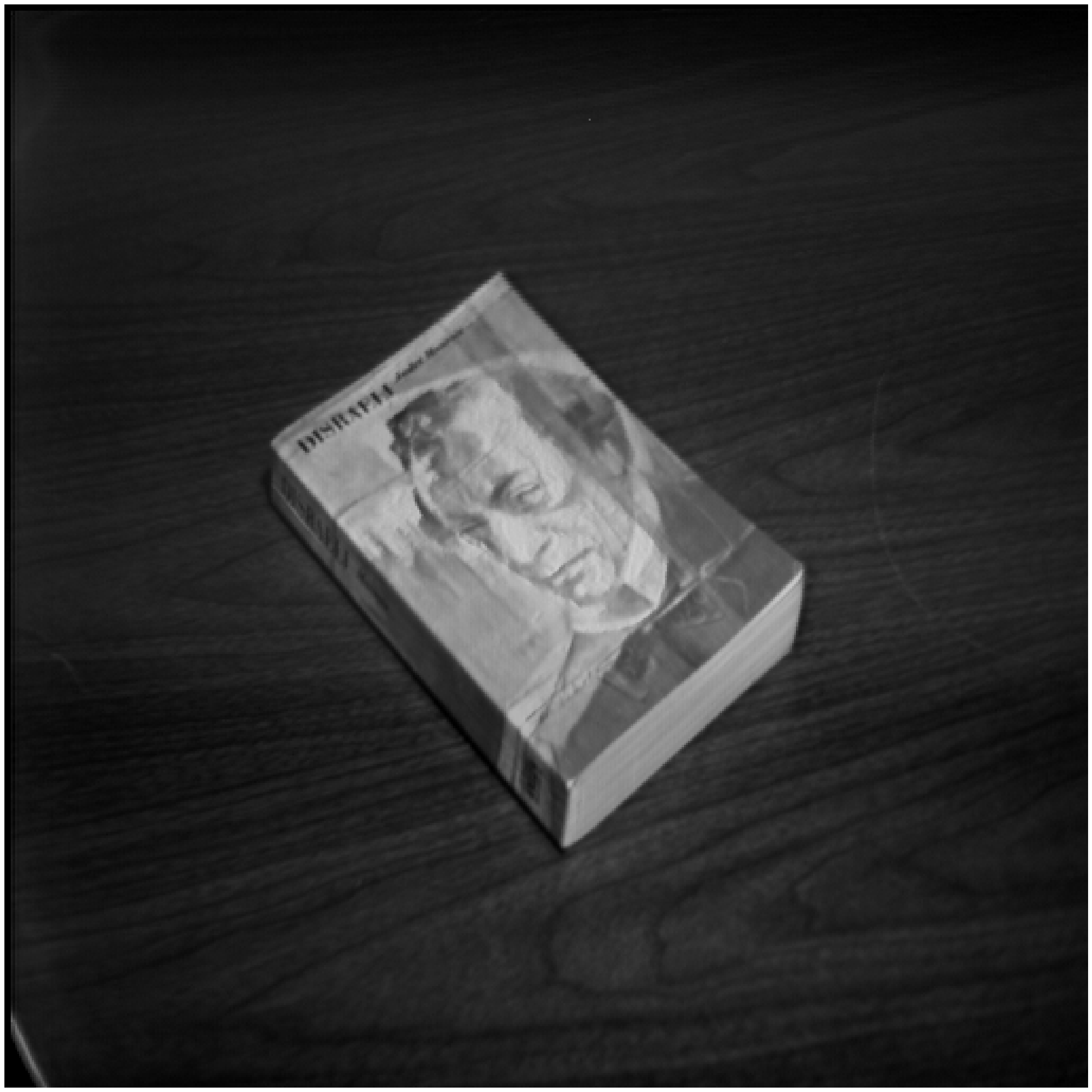

Figure 1: Input Image 1

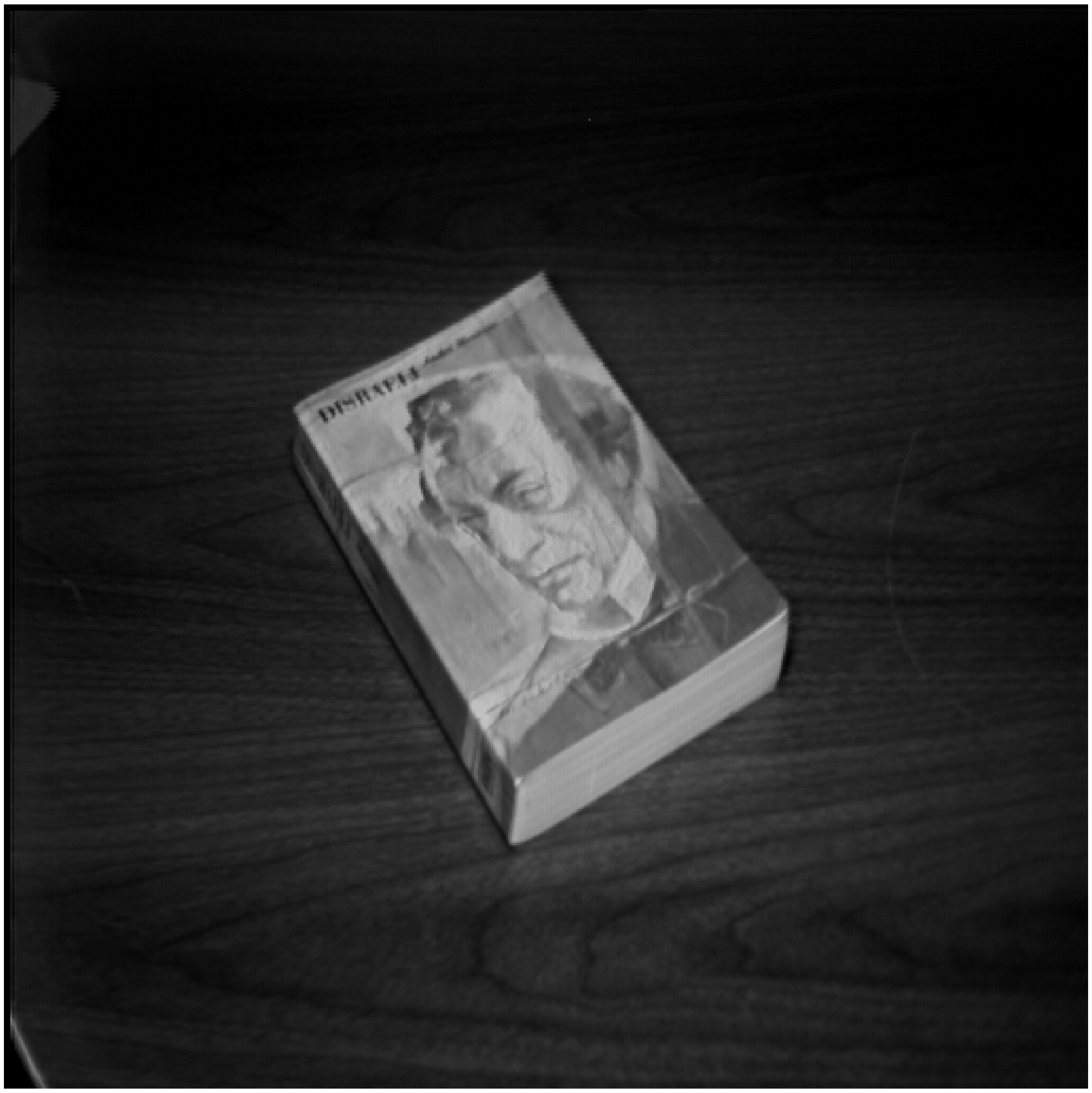

Figure 2: Input Image 2

Now we would have to select manual correspondences and try to show the correspondences

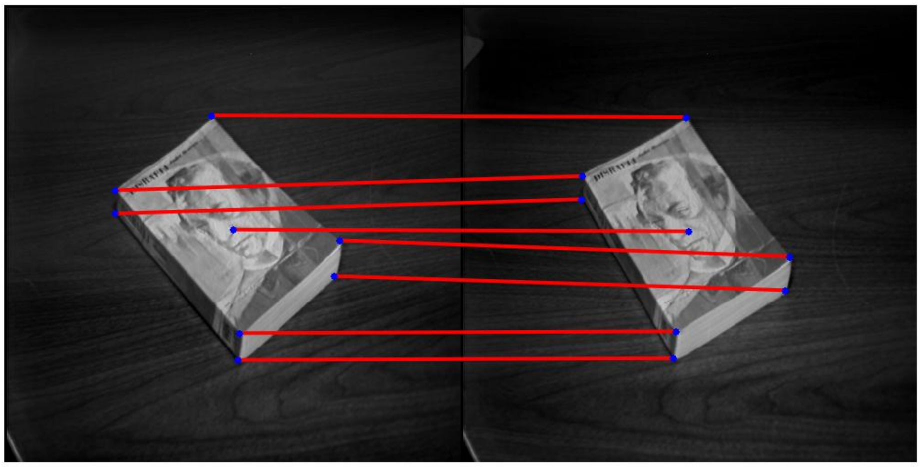

Figure 3: Manual Correspondences between the images

Before Rectification

$$F = \begin{bmatrix} 0.0032 & 0.0021 & 0.0074 \\ 0.0000 & 0.0000 & 0.0030 \\ -0.0074 & -0.0011 & -0.1294 \end{bmatrix}$$

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

$$P' = \begin{bmatrix} -0.0070 & -0.0011 & -0.1226 & -0.3197 \\ -0.0024 & -0.0004 & -0.0413 & 0.9475 \\ 0 & 0 & -0.0080 & 0.0036 \end{bmatrix}$$

$$e = \begin{bmatrix} -0.2067 \\ 0.9784 \\ 0.0032 \end{bmatrix}$$

$$e' = \begin{bmatrix} -0.3197 \\ 0.9475 \\ 0.0036 \end{bmatrix}$$

7.1 Rectification

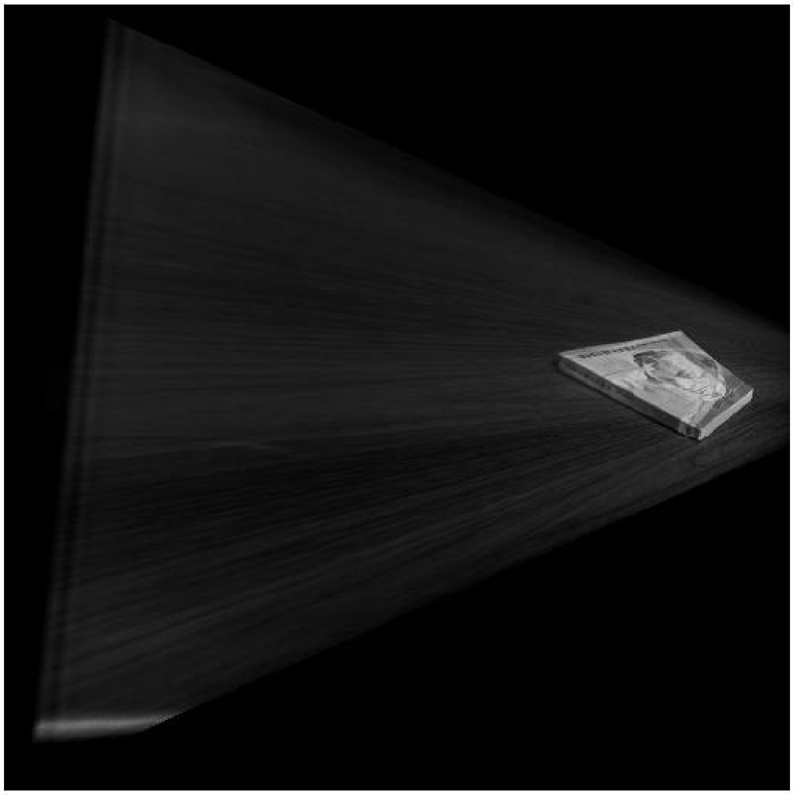

Figure 4: After rectification of first image

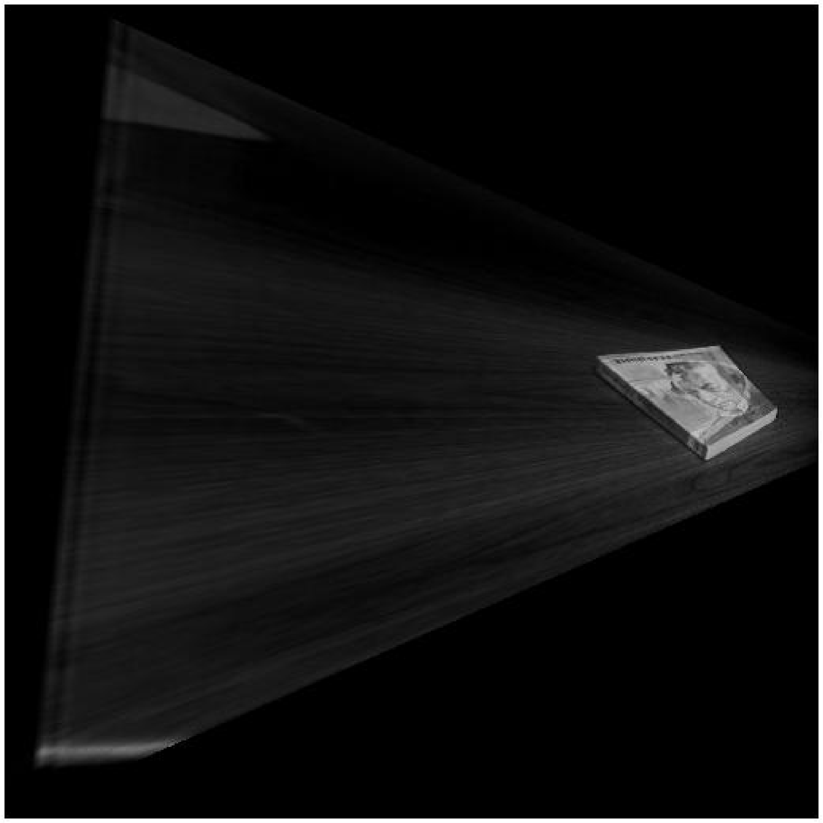

Figure 5: After rectification of second image

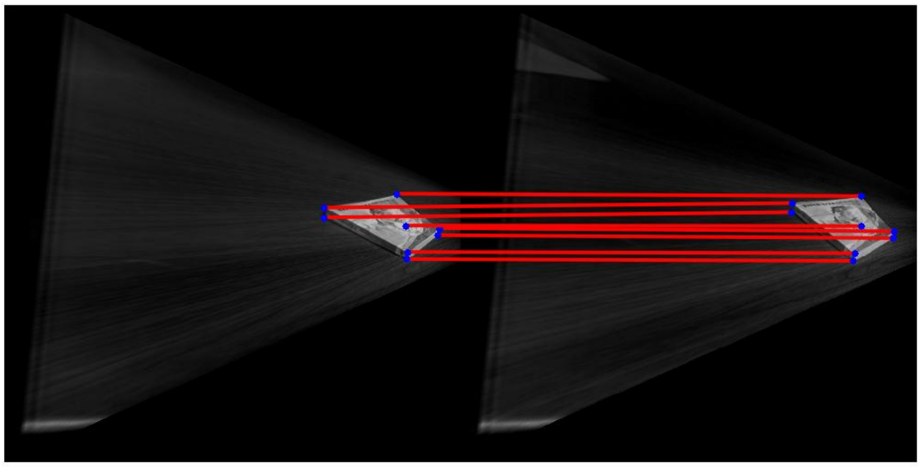

Figure 6: After rectification and with manual correspondences

After Rectification

$$F = \begin{bmatrix} 0.0025 & 0.0013 & 0.0000 \\ 0.0013 & 0.0001 & 0.0439 \\ -0.0056 & -0.0182 & -1.4375 \end{bmatrix}$$

$$P' = \begin{bmatrix} -0.0001 & -0.0003 & -0.0007 & 1.0000 \\ 0.0056 & 0.0182 & 1.4375 & 0.0000 \\ 0.0000 & -0.0001 & 0.0439 & 0.0000 \end{bmatrix}$$

$$e = \begin{bmatrix} 0.9675 \\ 0.0000 \\ 0.0000 \end{bmatrix}$$

$$e' = \begin{bmatrix} 1.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}$$

7.2 SURF Features of rectified images

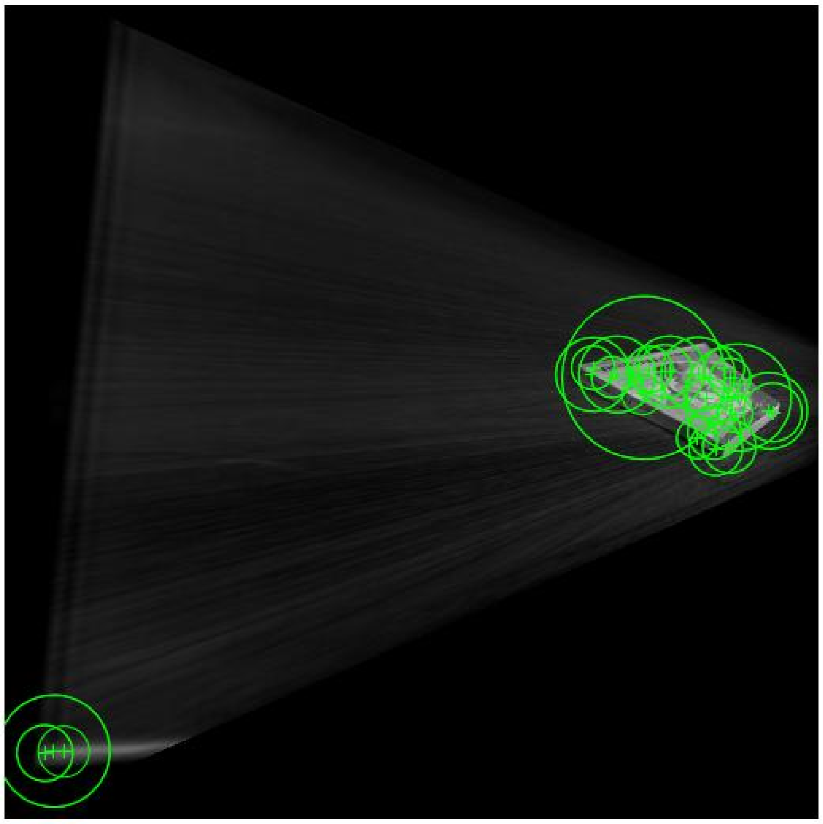

Figure 7: SURF Features of rectified first image

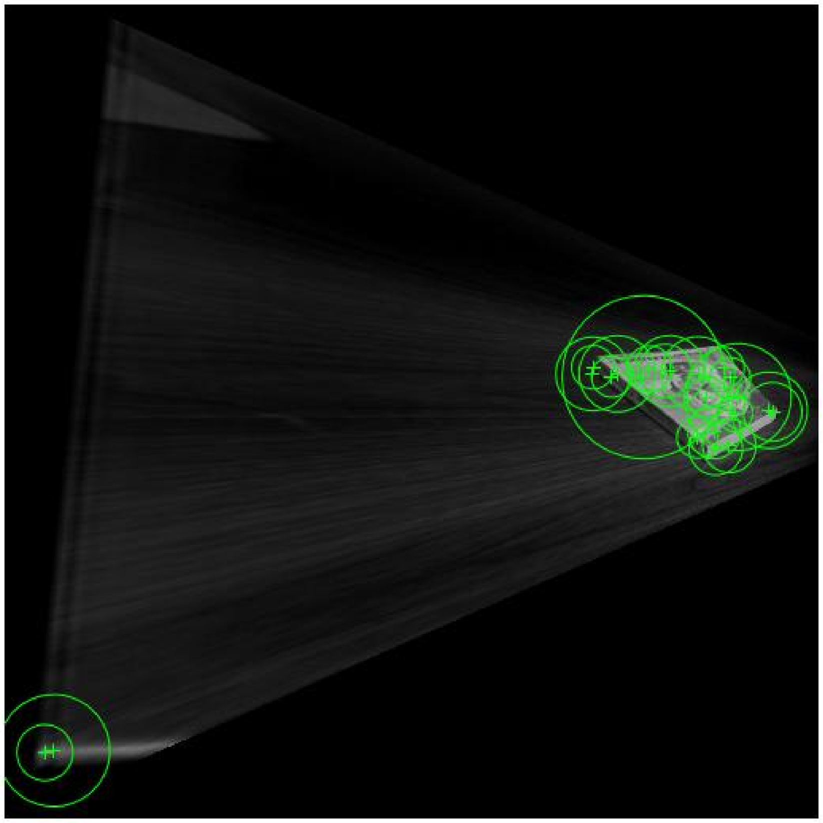

Figure 8: SURF Features of rectified second image

Now the correspondences are matched for the features,

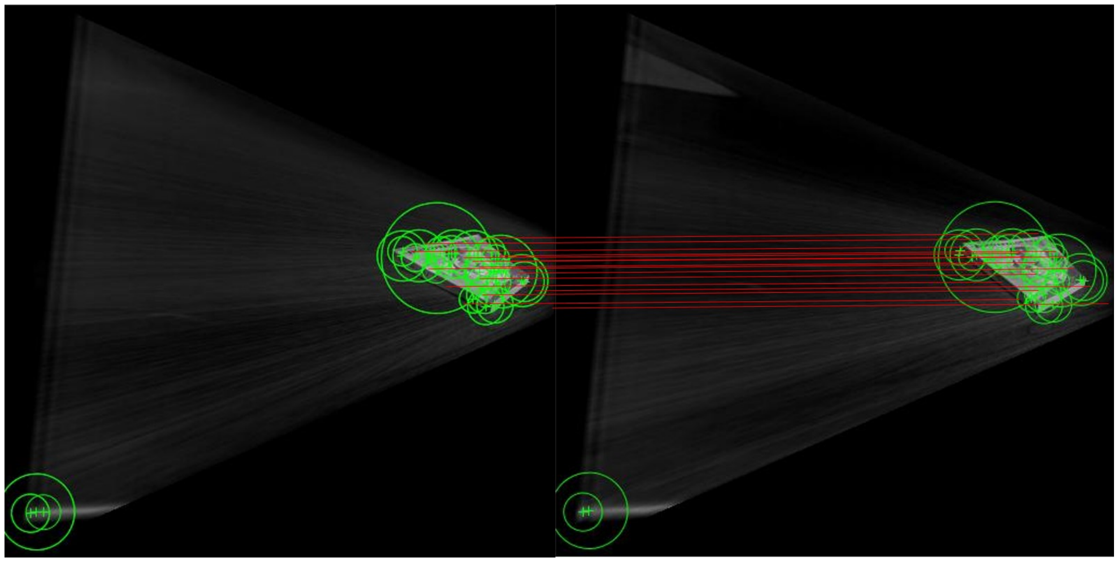

Figure 9: SURF Features Correspondence

For 3D reconstruction, we try out refining the F, P and the the conditioning of the F such that the rank of F, using the LM algorithm. Also we need to refine the world points according to the condition described in theory.

7.3 Result of 3D Reconstruction

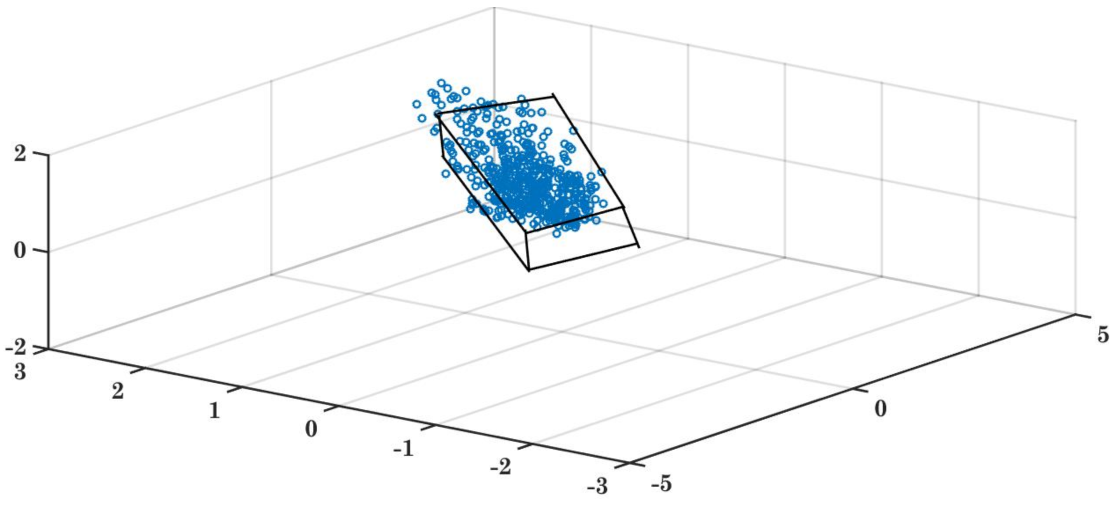

Figure 10: View 1 of 3D reconstruction

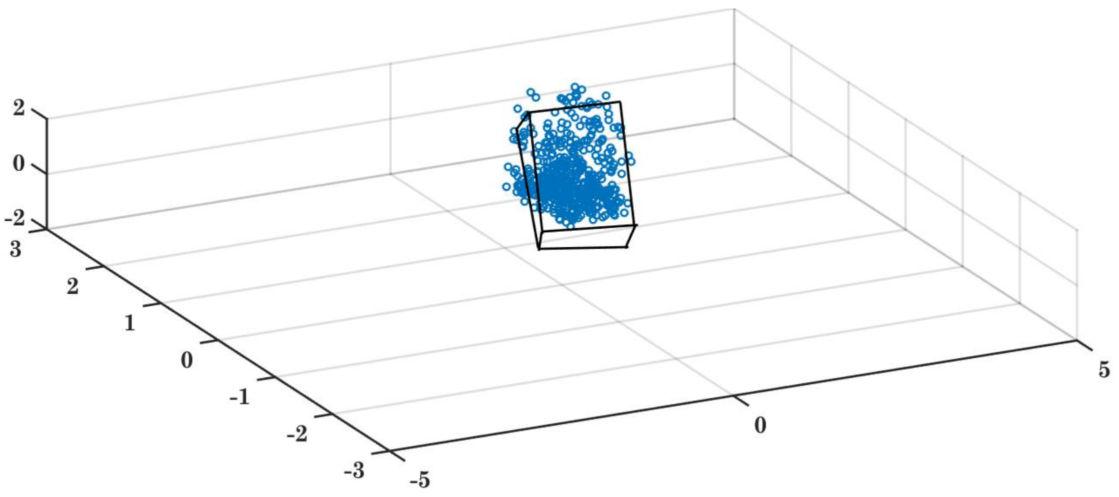

Figure 11: View 2 of 3D reconstruction

Showing the corresponding corner points

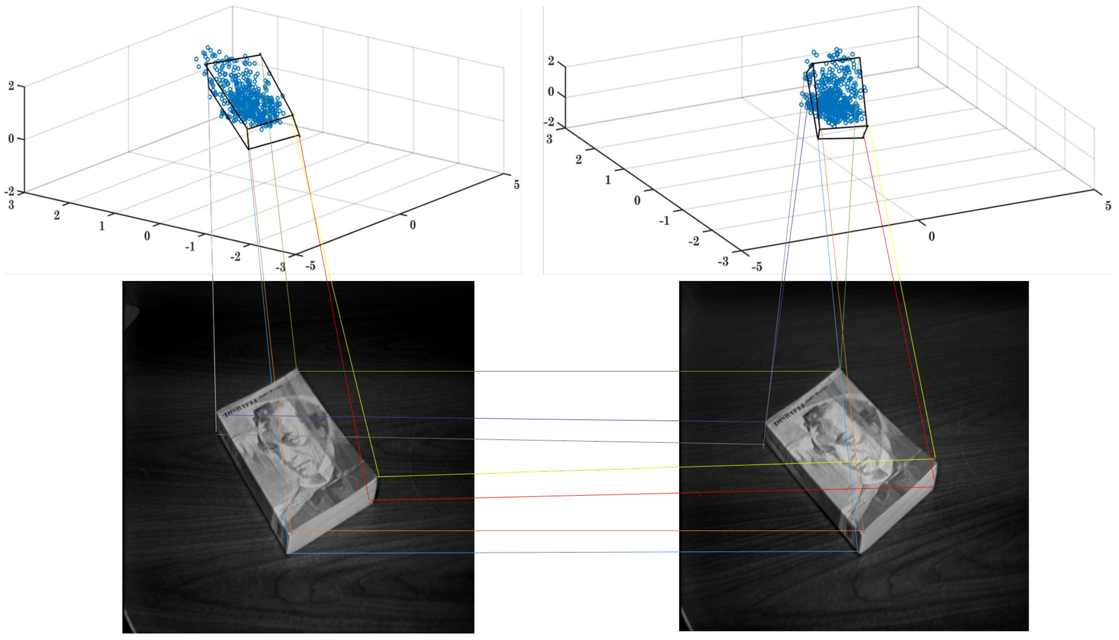

Figure 12: Showing the corresponding Corner Points. Each color represents each corner point

Code

The script is in MATLAB 2016a and is self-explanatory

Main Script

```
close all; clear; warning off
```

```
im1=imread('left.png');
im2=imread('right.png');

% Downsampling is down if the resolution is too high
fact=1
im1 = imresize(im1, fact);
im2 = imresize(im2, fact);

% Parameters
n_corresp=8; % Total no. of manual correspondences
T_ncc=0.7; % Threshold 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'^TFx

%% Selecting the manual correspondences
pt1=[124 234 ; 124 208 ; 232 124 ; 376 264 ; 370 304 ; 262 398 ; 264 368 ; 257 252];
pt2=[135 218 ; 136 192 ; 252 126 ; 369 282 ; 364 320 ; 238 396 ; 241 366 ; 255 254];


% Making them homogeneous coordinates
x1=[pt1 ones(n_corresp,1)];
x2=[pt2 ones(n_corresp,1)];

% Plot the correspondences
plot_corresp(im1,im2,x1,x2);


%% Finding F

%Use the 8-point normalization algorithm
[T1]= norm_point(x1);
[T2]= norm_point(x2);

%Find F using Linear Least Squares
F=Find_F_LLS(x1,x2,T1,T2);

%

%Compute the epipoles and the projection  matrices
e1=null(F); % Right null vector
e2=null(F'); % Left null vector

e2x = [0 -e2(3) e2(2);e2(3) 0 -e2(1);-e2(2) e2(1) 0];


P1=[1 0 0 0;0 1 0 0;0 0 1 0];
P2=[e2x*F, e2];

% Apply Non-linear least  squares to imporve estimate of F and P2;
[P2 F X]=NLLSOpt(@errorFunc,x1,x2,P1,P2);
```

```
%Recalculate the epipoles with refined estimates
e1=null(F);
e2=null(F');

%% Rectification to be done

%Rectification of images need to be done
[im1rect im2rect F x1new x2new H1 H2] = Rect_Img(e1,e2,x1,x2,im1,im2,P1,P2,F);


plot_corresp(im1rect,im2rect,x1new,x2new);

%% Extracting SURF Features

Ig1=rgb2gray(im1rect);
Ig2=rgb2gray(im2rect);

pts1=detectSURFFeatures(Ig1);
pts2=detectSURFFeatures(Ig1);

[feat1,valpts1]= extractFeatures(Ig1,pts1);
[feat2,valpts2]= extractFeatures(Ig2,pts2);

% This is for finding the NCC matrix
[C]=EstCorrespNCC(feat1,feat2,valpts1,valpts2);
[x1f,x2f]=GetFinFeatNCC(C,feat1,feat2,valpts1,valpts2, r_ncc, T_ncc, F, thresh_F );

% Plot the Correspondences for SURF
plot_corresp(im1rect,im2rect,x1f,x2f);

%% Beginning of 3D reconstruction

[T1]= norm_point(x1f);
[T2]= norm_point(x2f);

%Find F using Linear Least Squares
F=Find_F_LLS(x1f,x2f,T1,T2);

%Compute the epipoles and the projection  matrices
e1=null(F); % Right null vector
e2=null(F'); % Left null vector

e2x = [0 -e2(3) e2(2);e2(3) 0 -e2(1);-e2(2) e2(1) 0];

%Compute Camera Projection Matrices
P1=[1 0 0 0;0 1 0 0;0 0 1 0];
P2=[e2x*F, e2];

% Apply Non-linear least  squares to imporve estimate of F and P2;
[P2 F X]=NLLSOpt(@errorFunc,x1f,x2f,P1,P2);
```

```
PlotWorPts(X,x1f,x2f,P1,P2,F,thresh_F);
```

8-point Normalization Script

```
function [T] = norm_point(x)

% The 8-point normalization algorithm is applied
mean_x=mean(x(:,1));
mean_y=mean(x(:,2));

std_tmp = 0;
for i = 1:size(x,1)
std_tmp = std_tmp + sqrt((x(i,1)-mean_x)^2+(x(i,2)-mean_y)^2);
end

std_tmp=std_tmp/size(x,1);
scale=sqrt(2)/std_tmp;
xtr= -scale*mean_x;
ytr= -scale*mean_y;
T=[scale 0 xtr; 0 scale ytr; 0 0 1];
end
```

Plot Correspondence Script

```
function []=plot_corresp(im1,im2,x1,x2)

im=[im1 im2];
imshow(im);
hold on
n_rows=size(im1,1);
n_cols=size(im1,2);

for i=1:size(x1,1)
    plot([x1(i,1)/x1(i,3);x2(i,1)/x2(i,3)+n_cols], [x1(i,2)/x1(i,3);x2(i,2)/x2(i,3)], ...
    'Color','r','linewidth',3)
    scatter([x1(i,1)/x1(i,3);x2(i,1)/x2(i,3)+n_cols], [x1(i,2)/x1(i,3);x2(i,2)/x2(i,3)], ...
    'blue', 'filled', 'linewidth', 5)
end

hold off
```

Linear Least Squares Script

```
function [F]=Find_F_LLS(x1,x2,T1,T2)

n_corresp=size(x1,1);

%The correspondences are normalized;

x1t=(T1*x1')';
x2t=(T2*x2')';
```

```
A=zeros(n_corresp,9);

for i=1:1:n_corresp
A(i,:) = [x2t(i,1)*x1t(i,1) x2t(i,1)*x1t(i,2) x2t(i,1) ...
x2t(i,2)*x1t(i,1) x2t(i,2)*x1t(i,2) ...
x2t(i,2) x1t(i,1) x1t(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
[UF DF VF] = svd(F);
DF(end,end)=0; % Make rank 2 by zeroing the last singular value

F=UF*DF*VF';
F=T2'*F*T1;
end
```

Non-Linear Least Squares Script

```
function [P2 F X]= NLLSOpt(errorFunchandle,x1,x2,P1,P2)
%First we need to create a parameter of vectors
% Total number of parameters should be 12+3*totalcoresspondences
size(P2);
p = [reshape(P2',1,12)];
n_corresp=size(x1,1);
X=zeros(n_corresp,4);
Xtmp=zeros(n_corresp,4);

for i=1:size(x1,1)
    Xn=getWorld(P1,P2,x1(i,:),x2(i,:));
    Xtmp(i,:)=Xn;
    p=[p Xn(1:3)];
end
p=double(p);
x1=double(x1);
x2=double(x2);

options = optimoptions('lsqcurvefit','Algorithm','levenberg-marquardt');

p_updated=lsqnonlin(errorFunchandle,p,[],[],options,x1,x2);

P2=reshape(p_updated(1:12),4,3)';
e2=P2(:,4);
ex=[0 -e2(3) e2(2); e2(3) 0 -e2(1); -e2(2) e2(1) 0];
M=P2(:,1:3);
F=ex*M;
```

```
% Return World 3D co-ordinates
cnt=13;
for i=1:n_corresp
    X(i,:)=[p_updated(cnt:cnt+2) 1];
    cnt=cnt+3;
end
end
```

LM algorithm Error Function Script

```
function e=errorFunc(p,x1,x2)

P2=reshape(p(1:12),4,3)';
n_corresp=size(x1,1);
% Return World 3D co-ordinates
cnt=13;
for i=1:n_corresp
    X(i,:)=[p(cnt:cnt+2) 1];
    cnt=cnt+3;
end
e=0;
% Error func is described that is to be used for LM refinement
for i=1:n_corresp
    x1est=([eye(3,3) zeros(3,1)]*X(i,:)')'
    x1est=x1est/x1est(end);
    x2est=(P2*X(i,:)')'
    x2est=x2est/x2est(end);
    e=e+(norm(x1(i,:)-x1est))^2 + (norm(x2(i,:)-x2est))^2
end
```

Image Rectification Script

```
function [im1rect im2rect F x1new x2new H1 H2] = Rect_Img(e1,e2,x1,x2,im1,im2,P1,P2,F)

h=size(im1,1);
w=size(im1,2);
npts=size(x1,1);

%Convert from homogeneous to physical coordinates
e2=e2/e2(end);

ang=atan(-(e2(2)-h/2)/(e2(1)-w/2));
f=cos(ang)*(e2(1)-w/2)-sin(ang)*(e2(2)-h/2);
R=[cos(ang) -sin(ang) 0;sin(ang) cos(ang) 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;

%Preserve Centre after applying Homography
cpt=[w/2 h/2 1]';
ncpt=H2*cpt;
```

```
ncpt=ncpt/ncpt(end);

T2=[1 0 w/2-ncpt(1);0 1 h/2-ncpt(2);0 0 1];
H2=T2*H2;



% H1 computation
%Compute homography for first image
M=P2*pinv(P1);
H0=H2*M;
H0=H1;
x1hat=ones(size(x1));
x2hat=ones(size(x2));

for i=1:1:npts
    tmp=(H0*x1(i,:)')'
    x1hat(i,:)=tmp/tmp(end);
    tmp=(H2*x2(i,:)')';
    x2hat(i,:)=tmp/tmp(end);
end

% %Linear Least Squares for finding HA
A=zeros(npts,3);
b=zeros(npts,1);
for i=1:npts
    A(i,:)=[x1hat(i,1) x1hat(i,2) 1];
    b(i)= x2hat(i,1);
end
x=pinv(A)*b; % Least squares step
HA=[x(1) x(2) x(3); 0 1 0; 0 0 1];
H1=HA*H0;

cpt=[w/2 h/2 1]';
ncpt=H1*cpt;
ncpt=ncpt/ncpt(end);

T1=[1 0 w/2-ncpt(1);0 1 h/2-ncpt(2);0 0 1];
H1=T1*H1;

% Update the Fundamental Matrix

[im1rect H1]=appHomo(H2,im1);
[im2rect H2]=appHomo(H2,im2);
F=inv(H2')*F*inv(H1);

% Update the interest points
x1new =zeros(size(x1));
x2new=zeros(size(x2));

for i=1:npts
    tmp=(H1*x1(i,:)')';
    x1new(i,:)=tmp/tmp(end);
    tmp=(H2*x2(i,:)')';
    x2new(i,:)=tmp/tmp(end);
end
end
```

Applying Homography Script

```
function [Xnew Hnew]=appHomo(H,X)
% This function is used to apply homography to an image
% The output is the new image and the required homography
%X=single(X);
hOrig=size(X,1);
wOrig=size(X,2);

% To find the boundary of the resulting image
a=[1 1];
b=[wOrig 1];
c=[1 hOrig];
d=[wOrig hOrig];

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

% To find the height and width of projected image into the world plane
ht=(ty2-ty1);
wt=(tx2-tx1);

H_scale=[wOrig/wt 0 0;0 hOrig/ht 0;0 0 1];
H=H_scale*H;

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

%Now we have found the offsets
tx=tx1;
```

```
ty=ty1;

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

%Now the output is an image
Xnew=zeros(hOrig,wOrig,3);
for m=1:hOrig
    for n=1:wOrig
        tmp=H_inv*[n;m;1];
        tmp=tmp/tmp(end);
        tmp=biLinear(tmp(1),tmp(2),X);

        Xnew(m,n,:)=tmp;
    end
end

Xnew=uint8(Xnew);
end
```

Bilinear Transformation Script

```
function [f]=biLinear(x,y,Z)
% Z is the image from where we got values
% x,y is the location form which we need to do bilinear interpolation
S = size(Z);
if ((y<1)||(y>S(1)) )
f=zeros(1,1,3);
elseif ((x<1)||(x>S(2)))
f = zeros(1,1,3);
else
x1 = floor(x);
x2 = ceil(x);
y1 = floor(y);
y2 = ceil(y);
f = ( (x2-x)*(y2-y) ) / ( (x2-x1)*(y2-y1) ) * Z(y1,x1,:) + ...
( (x-x1)*(y2-y) ) / ( (x2-x1)*(y2-y1) ) * Z(y1,x2,:) + ...
( (x2-x)*(y-y1) ) / ( (x2-x1)*(y2-y1) ) * Z(y2,x1,:) + ...
( (x-x1)*(y-y1) ) / ( (x2-x1)*(y2-y1) ) * Z(y2,x2,:);
end
end
```

Triangulation Script

```
function [Xn] = getWorld (P1,P2,x1,x2)
% This code is for getting the World co-ordiantes
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,:)) ];
```

```
[U D V]=svd(A);
Xn=V(:,4);
Xn=Xn/Xn(end);
Xn=Xn';

end
```

Plot World Coordinates Script

```
function []=PlotWorPts(X,x1f,x2f,P1,P2,F,thresh_F)
x_w=[];
%Triangulation is done here
for i=1:size(X,1)
    x1=(P1*X(i,:)')';
    x1=x1/x1(end);
    x2=(P2*X(i,:)')';
    x2=x2/x2(end);
    x_w=[x_w;X(i,:)];
end
figure;
scatter3(x_w(:,1),x_w(:,2),x_w(:,3),'o', 'Linewidth',2);
disp(['Total points in world 3D = ' num2str(size(x_w,1))]);
end
```

NCC Matrix Script

```
function [C] = EstCorrespNCC(feat1,feat2,valpts1,valpts2)

% The NCC Matrix is created
C=zeros(size(feat1,1),size(feat2,1));

for i=1:size(C,1)
    for j=1:size(C,2)
        C(i,j)=sum((feat1(i,:)-mean(feat1(i,:))).*(feat2(i,:)-mean(feat2(i,:)))) ...
            /sqrt(sum((feat1(i,:)-mean(feat1(i,:))).^2)*sum((feat2(i,:)-mean(feat2(i,:))).^2));
end
end
end
```

Final NCC Correspondence Script

```
function [x1,x2]=GetFinFeatNCC(C,feat1,feat2,valpts1,valpts2,r_ncc,T_ncc,F,thresh_F)

x1=[];
x2=[];
lst=size(C,2);

for i=1:size(C,1)
    [b1,i1]=max(C(i,:));
    [b2,i2]=max(C(i,[(1:i1-1) i1+1:lst]));
    if (i2>=i1)
        i2=i2+1;
    end
    if (b1 <T_ncc)
        % Do not do anything;
    elseif ((b2/b1)>r_ncc)
        % Do nothing
    elseif (abs(norm(feat1(i,:)-feat2(i,:)))>50)
        % Do nothing
    else
        x1=[x1;valpts1.Location(i,1) valpts1.Location(i,2) 1];
        x2=[x2;valpts2.Location(i,1) valpts2.Location(i,1) 1];
    end
end
end

% This is for declaring the epipole constraints
function [c]=epi_constraint(a,b,c,d,F,thresh)
x1=double([a,b,1]');
x2=double([c,d,1]');
result=x1'*F*x1;
if (result < thresh)
    c=1;
else
    c=0;
end
c=1;
end
```