Lab 1 - Camera Calibration
Pramita Winata

1 Introduction

In this lab, calibration of a simulated camera is implemented in MATLAB. Two methods, Halls and Faugeras methods are used to do the calibration. The calibration is done in 12 steps on MATLAB environment. In the following sections, each of the steps will be discussed in detail.

2 Implementation

2.1 Step 1 & 2

In this step, intrinsics and extrinsics parameters are given. From the given parameters, intrinsics and extrinsics transformation matrices can be formed. The following MATLAB code, shows the composition of the intrinsic and extrinsic matrices. From the given parameters, the obtained matrices are :

$$intrinsic = \begin{pmatrix} 557.0943 & 0 & 326.3819 & 0 \\ 0 & 712.9824 & 298.6679 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$extrinsics = \begin{pmatrix} -0.9511 & -0.2939 & -0.25 & 100 \\ -0.2939 & 1.1102 & 0.5501 & 0 \\ 0.0955 & 0.4899 & -0.7968 & 1500 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2.2 Step 3

In this step, we need to generate random 3D points as our points in the world. The points will lie in the range of [-480:480;-480:480;-480:480]. *rand* function in MATLAB is used to generate this. Function *create3DPoints* is created since we will generate more points later on. Generated 3D Points shows in Table 1

X	Y	Z
-252.121901082644	29.6373667468113	-392.161217914164
-90.8971969146324	-379.347602768874	-372.207396330214
273.050775114156	-200.092494809346	99.3921012008514
445.845760527265	-64.8144057884534	186.962106833223
247.775304277876	-64.6633668987835	149.278118211396
-374.635151305870	416.409454449919	-300.037625835181

Tab. 1: Random 3D point in the range of [-480 480]

2.3 Step 4

After having a set of points in the world system, we need to project the points to the image plane using the camera transformation matrix. First, we need to create the transformation matrix which is the matrix multiplication of extrinsics and intrinsics parameters of the camera. We have obtained these matrices from Section 2.1. The resut obtained is

$$transformation_matrix = \begin{pmatrix} -498.661466084079 & -3.82737110079675 & -399.328511955333 & 545282.28 \\ -181.020023319185 & 937.901756949879 & 154.258004987770 & 448001.85 \\ 0.0954915028125263 & 0.489912387106340 & -0.796781123448736 & 1500 \end{pmatrix}$$

The 2D projection can be calculated by multiplied the 3D points with this $transformation_matrix$, using this formula:

$$\begin{pmatrix} s^I X_u \\ s^I Y_u \\ s \end{pmatrix} =^C K_W \begin{pmatrix} {}^W X_W \\ {}^W Y_W \\ {}^W Z_W \\ 1 \end{pmatrix} \tag{1}$$

where X_u is the projection in X, Y_u is projection in Y, and s is the scaling factor. Thus, we need to divide the resulting values from the matrix with the scaling factor to obtained the correct coordinates.

The result is shown in Table 2

X	**Y**
458.976366026981	255.666489901269
462.344109067495	31.9900144134019
274.454109760419	167.727100854786
182.504945403792	246.242172569165
263.912932003601	266.218949802448
445.824409942766	450.945827133137

Tab. 2: Projection of 3D points

2.4 Step 5

In order to make clearer, we will visualize the projected points in this step. Furthermore, to be able to determine that the points are distributed in the image plane, we will draw the image window ($640x480$). The plot is shown in Figure 1. It can be seen that the points are covering most of the part in the image plane. However some parts, i.e left-up and left-down corner, are not covered. We can try to generate more points to cover this sections. It is important that the points are well-distributed since we want to calibrate the camera such that all the section in the image plane is calibrated, thus making the computation more accurate.

2.5 Step 6

In this step we will perform Hall calibration by generating the Hall transformation matrix, A. In this method, 2 matrices need to be generated, Q and B, where

$$QA = B \tag{2}$$

$$Q_{2i-1} = ({}^W X_{wi} \quad {}^W Y_{wi} \quad {}^W Z_{wi} \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad -^I X_{Ui}^W X_{wi} \quad -^I X_{Ui}^W Y_{wi} \quad -^I X_{Ui}^W Z_{wi}) \tag{3}$$

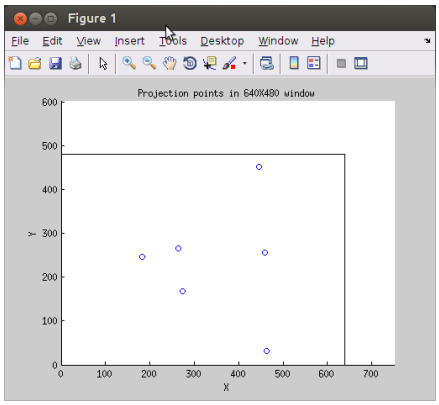

Fig. 1: Projection in image plane

$$Q_{2i} = (0 \quad 0 \quad 0 \quad 0 \quad {}^{W}X_{wi} \quad {}^{W}Y_{wi} \quad {}^{W}Z_{wi} \quad 1 \quad -{}^{I}Y_{Ui}^{W}X_{wi} \quad -{}^{I}Y_{Ui}^{W}Y_{wi} \quad -{}^{I}YX_{Ui}^{W}Z_{wi}) \tag{4}$$

$$B_{2i-1} = ({}^{I}X_{Ui}) \tag{5}$$

$$B_2 = ({}^{I}Y_{Ui}) \tag{6}$$

We used pseudo-inverse to solve the equation

$$A = (Q^tQ)^{-1}Q^tB \tag{7}$$

Function in MATLAB to compute this is created *hall.m*. The resulting matrix is shown in below

$$A = \begin{pmatrix} -0.332440977385970 & -0.00255158073526873 & -0.266219007968996 & 363.521520001266 \\ -0.120680015523622 & 0.625267837975171 & 0.102838669972363 & 298.667899998170 \\ 6.36610019384538e-05 & 0.000326608258108549 & -0.000531187415664940 & 1 \end{pmatrix}$$

2.6 Step 7

Now we have 2 calibration matrices, one obtained in Section 2.1 and one obtained in Section 2.5. Comparing these matrices by taking its differences we obtained

$$error\_hall = 1.0e^{-08} * \begin{pmatrix} 0.0003 & -0.0001 & 0.0001 & 0.1266 \\ 0.0023 & 0.0009 & -0.0019 & -0.1830 \\ 0.0000 & 0.0000 & -0.0000 & 0 \end{pmatrix}$$

After normalized the error, the difference between these two matrices is $2.2249xe^{-9}$, which is very small.

2.7 Step 8

In this step, we will give apply Gaussian noise to the projection points in the range [-1, +1] then compute the Hall calibration matrix again. We obtained

With the same technique, we calculate the Hall transformation matrix, A_noisy.

$$A\_noisy \begin{pmatrix} -0.326321744887294 & 0.00729453108048483 & -0.281541448221668 & 365.754258676637 \\ -0.0983350249729365 & 0.644474396291619 & 0.0771001611523039 & 298.936066925949 \\ 0.000145748220820647 & 0.000378181494020234 & -0.000624994860327783 & 1 \end{pmatrix}$$

X	Y
458.061948059879	256.358739156947
462.312745670475	32.2144749734115
274.272480524186	167.216809158221
180.968451134149	246.555312068015
263.769589736995	266.120278349022
446.027212629086	450.236152902869

Tab. 3: Noisy points with Gaussian

X	Y
458.077184024014	256.316314086929
462.340933207176	32.2491101720203
274.066161265893	167.176601014366
180.977948452559	246.570644311923
263.948835667053	266.133638712366
445.999165695872	450.254994013098

Tab. 4: Projection using noisy transformation matrix

Computing the differences obtained from the non-noisy Hall transformation matrix, we obtained:

$$A\_noisy \begin{pmatrix} 0.0061 & 0.0223 & 0.0001 & 0.0098 \\ 0.0192 & 0.0001 & -0.0153 & -0.0257 \\ -0.0001 & 2.2327 & 0.2682 & 0 \end{pmatrix}$$

Normalizing the error, we obtained 2.249207714123728 of error factor.

Then, we project the 3D Points using this new noisy matrix. The resulting noisy projection is

We can compute the distance between this noisy points with non-noisy points and averaging them. The distance is computed using eucledian distance.

$$Eucledian\_distances = \begin{pmatrix} 1.27163538363805 \\ 0.00449134473630293 \\ 0.548642022462221 \\ 2.15949979813704 \\ 0.0507754477938485 \\ 0.247141956145488 \end{pmatrix}$$

$$Average\_distances = 0.7137.$$

We can see that the distance between the projection points using non-noisy Hall matrix with the projection points using noisy Hall matrix cannot be ignored. It is important to compute transformation matri as accurate as possible to get closer to the right projection point.

Figure 4 shows the visualization of this discrepancy, where blue is non-noisy projection and green is noisy-projction.

2.8 Step 9

In this step we need to increase the number of 3D points to 10 and 50 points and repeat step 8.

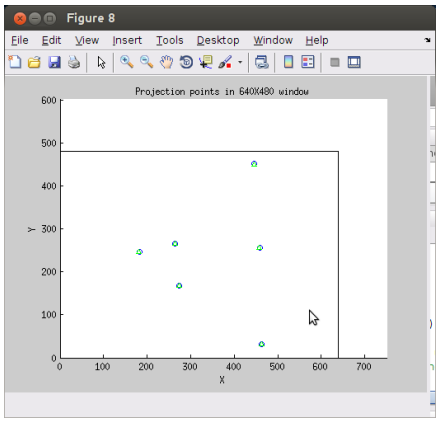

Fig. 2: Non-noisy projection vs noisy-projection

Number of points	**6**	**10**	**50**	**100**	**500**
Transformation matrix error	2.24920	0.18432	0.14815	0.041929	7.56115e-10
Average eucledian distances	0.7137	0.46077	0.16866	0.07994	0.06965

It is noticed that the more points used the more accurate calibration matrix is.

2.9 Step 10

Now, we will use other method to calibrate the camera, Faugeras method. First we need to compute the transformation matrix of Faugeras. This can be done using two method least-square method and SVD . MATLAB function $faugeras_LS.m$ and $faugeras_SVD.m$ are created for this. The resulted matrix then used to extract the intrinsic and extrinsic parameters which will be used to form the transformation matrics. Function $faugeras_calib_matrix.m$ is created for this. We obtained the Faugeras transformation matrix as follow:

$$m\_F = \begin{pmatrix} -0.332440977389353 & -0.00255158073383211 & -0.266219007970274 & 363.521520000007 \\ -0.120680015546076 & 0.625267837966632 & .102838669991796 & 298.667900000004 \\ 6.36610018751645e-05 & 0.000326608258070652 & -0.000531187415632749 & 1 \end{pmatrix}$$

Computing the difference between this matrix and the calibration matrix and then normalized it, we obtained an error of $8.136489615831788e-12$. This error is considered very small.

2.10 Step 11

Step 11. Add Gaussian noise to the 2D points (produce noise so that the 95[-1,1], then in the range [-2,2] and finally in the range [-3,3]) and compute vector X (repeat step 10) for each rang. Which method is more accurate (Faugeras or Hall) with such noise, compute the accuracy from 2D point discrepancy? We then apply noises in 3 different ranges to the 2D points, and repeat Step 10. We calculate the error of the calibration matrix, which is computed by taking the difference of the transformation matrix with the original matrix in Step 2.

Noise	**[0 0]**	**[-1 1]**	**[-2 2]**	**[-3 3]**
Faugeras	8.136489615831788e-12	1.032264446488363	2.447590196945333	2.654129136855054
Hall	2.224952940303527e-09	1.032264447108434	2.447590198988834	2.654129139308839

Tab. 5: Error in calibration matrix

In order to compare which method is better, we repeat Step 8 for every range of noises with both method. We project the 3D points with each of the matrices, compute the discrepancy points' distance, and get the average of it.

Noise	**[-1 1]**	**[-2 2]**	**[-3 3]**
Faugeras	0.455940352150220	1.105659958418729	2.715037024787530
Hall	0.455940353210494	1.105659960282629	2.715037026432457

Tab. 6: Average of eucledian distances of noisy points using Hall and Faugeras method

It can be seen that both method give approximately the same discrepancy. Thus, Faugeras and Hall are equivalent.

2.11 Step 12

In this last step, we will draw the setting of the calibration environment to verify that the projection points is in the optical ray of the 3D point. The most important thing to understand in this step is to be able to transform a point to different coordinate system. We need to choose one coordinate system to be drawn. World coordinate system is chosen for this. Thus the camera points and image points need to be converted to the world coordinate system. The implementation of this environment drawing is done in *DrawSetting.m*. The drawing steps can be divided into 4 main parts:

1. Drawing 3D Points :
   This is fairly straigthforward as the points already in the world coordinate system.

2. Drawing the Camera points :
   To convert camera points to world coordinate system, we use the inverse extrinsic matrix of the calibration matrix. Th multiplication between inverse extrinsic parameter and the point will change the point reference system from camera to the world coordinate system.
   Function *getCameraPointInWorld.m* illustrates this.

3. Drawing the image plane
   For this case, we convert the point to camera coordinate system first and then convert it to the world coordinate system. We utilize the parameters in intrinsic matrix and the focal length of the camera.

$$ {}^{C}P_{I} \begin{pmatrix} -(u0-x)*f/au \\ -(v0-x)*f/av \\ f \\ 1 \end{pmatrix} $$

4. Drawing the projection points Drawing the projection points have the same flow with drawing the image plane. We need to convert the points to refer to camera coordinate system then convert it again to the world coordinate system.

After drawing all of the necessary points and also the optical ray. It is noticed that the projection points lie in the optical ray defined by the 3D points and it is crossing in the focal point (origin of the camera). It means that the calibration and conversion is done correctly. It is also noticed that when trying to draw the center point retinal plane, which is [u0, v0], with focal length as the distance from the focal point (origin of camera), the image border limit does not cover the whole 640 x 480 dimension , as shown in

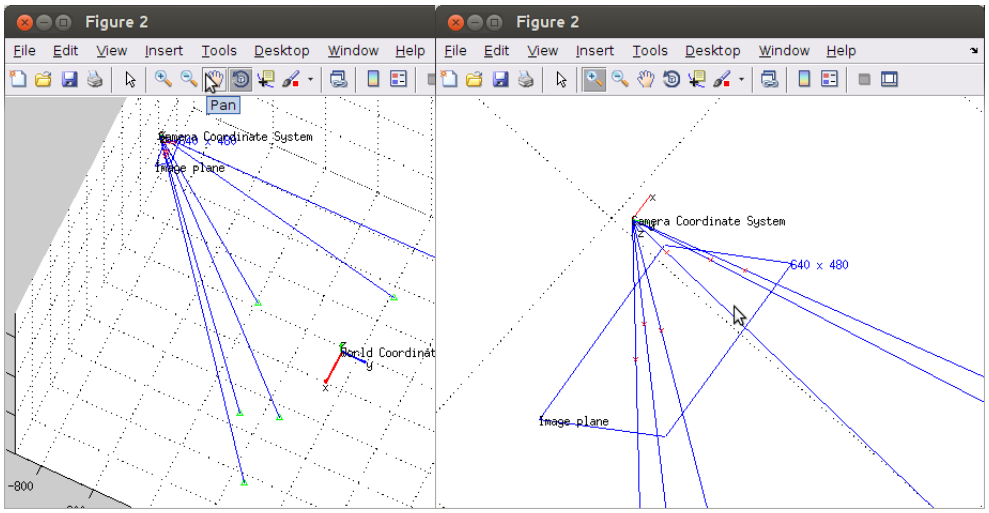

Fig. 3: Optical ray

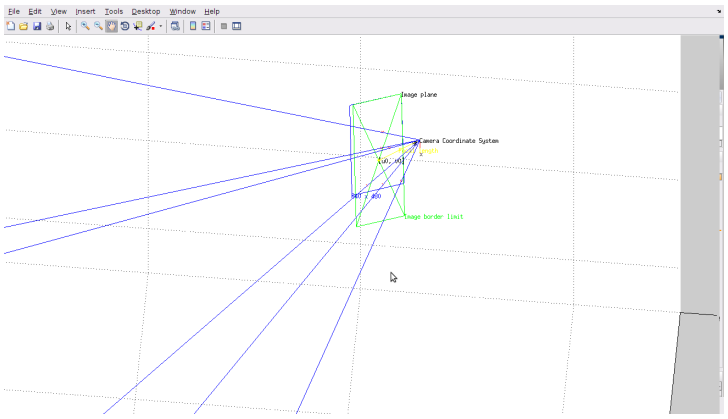

Fig. 4: Focal length

3 General comments

The lab has been proved very usefull to better understanding the camera calibration. I definetly have much better understanding in camera calibration after doing the lab and the report. However, the time given seems not enough. Suggestion would be to make this work in three lab session instead of two.

Appendices

A MATLAB Code

```
clear all; clc; close all;
%Step 1
au = 557.0943;
av =712.9824;
u0 =326.3819;
v0 =298.6679;
f =80.;
Tx =100;
Ty =0;
Tz =1500;
Phix =0.8*pi/2;
Phiy =-1.8*pi/2;
Phix1 =pi/5;
image_size = [640 480];

%Step 2
rotation_on_x = [1 0 0; 0 cos(Phix) -sin(Phix);0 sin(Phix) cos(Phix)];
rotation_on_y = [cos(Phiy) 0 sin(Phiy); 0 1 0; -sin(Phiy) 0 cos(Phiy)];
rotation_on_x1 = [1 0 0; 0 cos(Phix1) -sin(Phix1);0 sin(Phix) cos(Phix1)];
rotation_matrix =  rotation_on_x * rotation_on_y * rotation_on_x1;
translation_matrix = [Tx; Ty; Tz];
intrinsic_param = [au 0 u0 0; 0 av v0 0; 0 0 1 0];
extrinsic_param = [rotation_matrix , translation_matrix ; 0 0 0 1 ];


%Step 3- generate six 3D random points
threshold_max = 480;
threshold_min = -480;
points_total = 6;
xyz = create3Dpoints( points_total, threshold_max, threshold_min );

%Step 4 - project 3D points to 2D point
calib_param = intrinsic_param * extrinsic_param;
xyz = [xyz; ones(1,points_total)];
projection = calib_param * xyz;

s = projection(3,:);
sX = projection (1,:);
sY = projection(2,:);

Xu = sX ./ s;
Yu = sY ./ s;
p_2d = [Xu; Yu];

%Step 6 - Method of Hall
A = hall( xyz, p_2d )
```

```
% Step 7.
% Compare the matrix obtained in Step 6 to the one defined in step 2.
calib_matrix_normalized = calib_param / calib_param(3,4)
error = A(:) - calib_matrix_normalized(:);
error_calib_hall = norm(error,2)

%step 8 -  Add some Gaussian noise to all the 2D points
%producing discrepancies between the
%range [-1,+1] pixels for the 95% of points
noise = randn(2,points_total)*0.5;
noisy_points = p_2d + noise;
% Again repeat step 6 with the noisy 2D points and the ones
%defined in step 3.
% Compare the obtained matrix to the one you got in step 6 with the
% non-noisy points
A_noisy = hall( xyz, noisy_points )
error = A_noisy(:) - calib_matrix_normalized(:);
error_noisy_hall = norm(error,2);
%Now compute the 2D points with the obtained matrix
% and compare them to those obtained in step 4 (you can check accuracy
% computing the discrepancy between points)?
%step 4
p_2d_noisy = get2dProjection( xyz, A_noisy );
eucl_dist = compute_points_distance( p_2d_noisy(),p_2d());
error_p2d_noisy_norm = mean(eucl_dist);

figure(1)
hold on

axis([0 750 0 600]);
xlabel('X'); ylabel('Y');
title('Projection points in 640X480 window');
rectangle('Position',[0,0,640,480]);hold on;scatter(p_2d(1,:),p_2d(2,:)....
,'o','b');hold on; scatter(p_2d_noisy(1,:), p_2d_noisy(2,:),'^','g')

% Step 10. Define the vector X of the method of Faugeras. Compute X
%by using both least-squares (LS) and Singular Value Decomposition (SVD)
%by using the points of
% Step 3 and Step 4, without noise.
X_SVD = faugeras_SVD( xyz, p_2d );
X = faugeras_LS( xyz, p_2d );

% Extract the camera parameters from both
% computations. Compare the obtained parameters with the ones defined in Step 1.
[calib_matrix_F, intrinsic_param_F, extrinsic_param_F] ....
= faugeras_calib_matrix(X);
calib_matrix_normalized_F = calib_matrix_F / calib_matrix_F(3,4)
%Step 11.
%[-1,1]
noise_11 = randn(2,points_total)*0.5;
p_2d_noise_11 = p_2d + noise_11;
X_n11 = faugeras_LS( xyz, p_2d_noise_11 );
calib_matrix_F_n11 = faugeras_calib_matrix(X_n11);
```

```
calib_matrix_normalized_F_n11 = calib_matrix_F_n11/calib_matrix_F_n11(3,4)
error_Fn11 = calib_matrix_normalized_F_n11(:) - calib_matrix_normalized(:);
error_Fn11 = norm(error_Fn11,2);

p_2d_Fn1 = get2dProjection( xyz, calib_matrix_normalized_F_n11 );
eucl_dist_1 = compute_points_distance( p_2d_Fn1,p_2d);
error_p_2d_Fn1 = mean(eucl_dist_1)

A_noisy_11 = hall( xyz, p_2d_noise_11 )
error_11 = A_noisy_11(:) - calib_matrix_normalized(:);
error_noisy_hall_11 = norm(error_11,2)

p_2d_A1 = get2dProjection( xyz, A_noisy_11 );
eucl_dist_A1 = compute_points_distance( p_2d_A1,p_2d);
error_p_2d_A1 = mean(eucl_dist_A1)


%[-2,2]
noise_22 = randn(2,points_total)*1;
p_2d_noise_22 = p_2d + noise_22;
X_n22 = faugeras_LS( xyz, p_2d_noise_22 );
calib_matrix_F_n22 = faugeras_calib_matrix(X_n22);
calib_matrix_normalized_F_n22  = calib_matrix_F_n22/calib_matrix_F_n22(3,4)
error_Fn22 = calib_matrix_normalized_F_n22(:) - calib_matrix_normalized(:);
error_Fn22 = norm(error_Fn22,2);

p_2d_Fn22 = get2dProjection( xyz, calib_matrix_normalized_F_n22 );
eucl_dist_22 = compute_points_distance( p_2d_Fn22,p_2d);
error_p_2d_Fn22 = mean(eucl_dist_22);

A_noisy_22 = hall( xyz, p_2d_noise_22 )
error_22 = A_noisy_22(:) - calib_matrix_normalized(:);
error_noisy_hall_22 = norm(error_22,2)

p_2d_A22 = get2dProjection( xyz, A_noisy_22 );
eucl_dist_A22 = compute_points_distance( p_2d_A22,p_2d);
error_p_2d_A22 = mean(eucl_dist_A22)

%[-3,3]
noise_33 = randn(2,points_total)*1.5;
p_2d_noise_33 = p_2d + noise_33;
X_n33 = faugeras_LS( xyz, p_2d_noise_33 );
calib_matrix_F_n33 = faugeras_calib_matrix(X_n33);
calib_matrix_normalized_F_n33 = calib_matrix_F_n33/calib_matrix_F_n33(3,4)
error_Fn33 = calib_matrix_normalized_F_n33(:) - calib_matrix_normalized(:);
error_Fn33 = norm(error_Fn33,2);

p_2d_Fn33 = get2dProjection( xyz, calib_matrix_normalized_F_n33 );
eucl_dist_33 = compute_points_distance( p_2d_Fn33,p_2d);
error_p_2d_Fn33 = mean(eucl_dist_33);


A_noisy_33 = hall( xyz, p_2d_noise_33 )
```

```
error_33 = A_noisy_33(:) - calib_matrix_normalized(:);
error_noisy_hall_33 = norm(error_33,2)

p_2d_A33 = get2dProjection( xyz, A_noisy_33 );
eucl_dist_A33 = compute_points_distance( p_2d_A33,p_2d);
error_p_2d_A33 = mean(eucl_dist_A33)

projection_F = calib_matrix_normalized_F * xyz;
s = projection_F(3,:);
sX = projection_F(1,:);
sY = projection_F(2,:);

Xu_F = sX ./ s;
Yu_F = sY ./ s;
p_2d_F = [Xu; Yu];

%Step 12
DrawSetting(f, intrinsic_param_F, extrinsic_param_F, xyz, p_2d_F );
```

```
function A = hall( p3d, p2d  )
%HALL Summary of this function goes here
%   Detailed explanation goes here
Q = [];
B = [];

for row = 1:size(p2d,2)
   idx = 2*row-1;

   r1 = [ p3d(1,row) p3d(2,row) p3d(3,row) 1 0 0 0 0  ....
       -p2d(1,row)*p3d(1,row) -p2d(1,row)*p3d(2,row) ....
       -p2d(1,row)*p3d(3,row) ];
   r2 = [ 0 0 0 0 p3d(1,row) p3d(2,row) p3d(3,row) 1 ....
       -p2d(2,row)*p3d(1,row) -p2d(2,row)*p3d(2,row) ....
       -p2d(2,row)*p3d(3,row) ];
   Q(idx,: ) = r1;
   Q(idx+1, : ) = r2;

   B(idx,: ) = p2d(1,row);
   B(idx+1, : ) = p2d(2,row);

end
%Get the calibration matrix using Least Squares
A = pinv(Q)*B;
A = [A; 1];
A = reshape(A, 4, 3)';

end
```

```
function points3D = create3Dpoints( points_total,threshold_max,threshold_min)
%CREATE3DPOINTS Summary of this function goes here
%   Detailed explanation goes here

points3D = rand(3, points_total) * (threshold_max-threshold_min) + threshold_min;
```

```
end

function eucl_dist = compute_points_distance(points2d,points2dnoise)

npoints = size(points2d,2);
eucl_dist = zeros(npoints,1);
for i = 1:npoints
    eucl_dist(i) = sqrt((points2d(1,i) - points2dnoise(1,i))^2 +....
    (points2d(1,i) - points2dnoise(1,i))^2);
end



end
```

```
function X = faugeras_LS( p3d, p2d )
%FAugeras Summary of this function goes here
%   Detailed explanation goes here
Q = [];
B = [];

for row = 1:size(p2d,2)
   idx = 2*row-1;

   r1 = [ p3d(1,row) p3d(2,row) p3d(3,row) -p2d(1,row)*p3d(1,row) ....
       -p2d(1,row)*p3d(2,row) -p2d(1,row)*p3d(3,row) 0 0 0 1 0 ];
   r2 = [ 0 0 0 -p2d(2,row)*p3d(1,row) -p2d(2,row)*p3d(2,row) ....
       -p2d(2,row)*p3d(3,row) p3d(1,row) p3d(2,row) p3d(3,row) 0 1  ];
   Q(idx,: ) = r1;
   Q(idx+1, : ) = r2;

   B(idx,: ) = p2d(1,row);
   B(idx+1, : ) = p2d(2,row);

end

%Get the calibration matrix using Least Squares
X = pinv(Q)* B;

end
```

```
function X = faugeras_LS( p3d, p2d )
%FAugeras Summary of this function goes here
%   Detailed explanation goes here
Q = [];
B = [];

for row = 1:size(p2d,2)
   idx = 2*row-1;

   r1 = [ p3d(1,row) p3d(2,row) p3d(3,row) -p2d(1,row)*p3d(1,row)....
       -p2d(1,row)*p3d(2,row) -p2d(1,row)*p3d(3,row) 0 0 0 1 0 ];
```

```
    r2 = [ 0 0 0 -p2d(2,row)*p3d(1,row) -p2d(2,row)*p3d(2,row)....
        -p2d(2,row)*p3d(3,row) p3d(1,row) p3d(2,row) p3d(3,row) 0 1  ];
    Q(idx,: ) = r1;
    Q(idx+1, : ) = r2;

    B(idx,: ) = p2d(1,row);
    B(idx+1, : ) = p2d(2,row);

end

%Get the calibration matrix using SVD
[U,S,V]= svd(Q);
%B = QX
%B = USV' X
%VSU'B = X
X=(V   *   pinv(S) * U' )     *   B;


end
```

```
function [calib_param_F, intrinsic_param_F, extrinsic_param_F ] ....
= faugeras_calib_matrix( X )
%FAUGERAS_CALIB_MATRIX Summary of this function goes here
%   Detailed explanation goes here
T1 = X(1:3); T1=T1';
T2 = X(4:6); T2=T2';
T3 = X(7:9); T3=T3';
C1 = X(10);
C2 = X(11);
%Intrinsics
u0_F = ( T1*transpose(T2) ) / (norm(T2))^2 ;
v0_F = ( T2*transpose(T3) ) / (norm(T2))^2 ;
au_F = (norm(cross(transpose(T1),transpose(T2)))) / norm(T2)^2;
av_F = (norm(cross(transpose(T2),transpose(T3)))) / norm(T2)^2;

%Extrinsics
Tx_F = (norm(T2)/(norm(cross(transpose(T1),transpose(T2)))))*(C1-u0_F);
Ty_F = (norm(T2)/(norm(cross(transpose(T2),transpose(T3)))))*(C2-v0_F);
Tz_F = 1 / norm(T2);
r1_F = (norm(T2)/(norm(cross(transpose(T1),transpose(T2)))))*(T1-u0_F*T2 );
r2_F = (norm(T2)/(norm(cross(transpose(T2),transpose(T3)))))*(T3-v0_F*T2 );
r3_F = T2 / norm(T2);

rotation_matrix_F =  [r1_F; r2_F; r3_F];
translation_matrix_F = [Tx_F; Ty_F; Tz_F];
intrinsic_param_F = [au_F 0 u0_F 0; 0 av_F v0_F 0; 0 0 1 0];
extrinsic_param_F = [rotation_matrix_F , translation_matrix_F ; 0 0 0 1 ];
calib_param_F = intrinsic_param_F * extrinsic_param_F;
%calib_matrix_normalized_F = calib_param_F / calib_param_F(3,4);

end
```

```
function [] = DrawSetting( f, intrinsic_param_F, extrinsic_param_F, xyz, p2d )

%Part 3
%Step 12
figure(2)
hold on
grid on

%Drawing plane
xlim([-500 500]);xlabel('x');
ylim([-900 500]);ylabel('y');
zlim([-500 1300]);zlabel('z');

%World Coordinate System
%Define the points
x_orig_w=[100 0 0]';
y_orig_w=[0 100 0]';
z_orig_w=[0 0 100]';
orig_w=[0 0 0 1]';
%Draw the vectors
 quiver3(orig_w(1),orig_w(2),orig_w(3),x_orig_w(1),x_orig_w(2),x_orig_w(3)....
 ,'Color','r','LineWidth',2);
 quiver3(orig_w(1),orig_w(2),orig_w(3),y_orig_w(1),y_orig_w(2),y_orig_w(3)....
 ,'Color','b','LineWidth',2);
 quiver3(orig_w(1),orig_w(2),orig_w(3),z_orig_w(1),z_orig_w(2),z_orig_w(3)....
 ,'Color','g','LineWidth',2);
%Label the vectors
 text(orig_w(1),orig_w(2),orig_w(3),'World Coordinate System');
text(x_orig_w(1),x_orig_w(2),x_orig_w(3),'x');
 text(y_orig_w(1),y_orig_w(2),y_orig_w(3),'y');
text(z_orig_w(1),z_orig_w(2),z_orig_w(3),'z');
 hold on;

%Camera Coordinate System
cKw = extrinsic_param_F;
%Origin of camera referring to camera coordinate system
x_orig_c = [10 0 0 1]';
y_orig_c = [0 10 0 1]';
z_orig_c = [0 0 10 1]';
orig_c   = [0 0 0 1]';
% Change the reference point to World coordinate system
%Origin of camera referring to world coordinate system
c_to_w = getCameraPointInWorld( cKw, orig_c )
x_orig_c_to_w= getCameraPointInWorld( cKw, x_orig_c);
y_orig_c_to_w= getCameraPointInWorld( cKw, y_orig_c);
z_orig_c_to_w= getCameraPointInWorld( cKw, z_orig_c);
%%% Plot
hold on;
plot3([c_to_w(1) x_orig_c_to_w(1)],[c_to_w(2) x_orig_c_to_w(2)],[c_to_w(3) ....
    x_orig_c_to_w(3)],'r');
hold on;
plot3([c_to_w(1) y_orig_c_to_w(1)],[c_to_w(2) y_orig_c_to_w(2)],[c_to_w(3)....
    y_orig_c_to_w(3)],'g');
```

```
hold on;
plot3([c_to_w(1) z_orig_c_to_w(1)],[c_to_w(2) z_orig_c_to_w(2)],[c_to_w(3) ....
    z_orig_c_to_w(3)],'b');
hold on;
%Label
text(c_to_w(1),c_to_w(2),c_to_w(3),'Camera Coordinate System');
text(x_orig_c_to_w(1),x_orig_c_to_w(2),x_orig_c_to_w(3),'x');
text(y_orig_c_to_w(1),y_orig_c_to_w(2),y_orig_c_to_w(3),'y');
text(z_orig_c_to_w(1),z_orig_c_to_w(2),z_orig_c_to_w(3),'z');

%hold on;
%Image plane
o_i=[0 0]';
a_i=[0 640]';
b_i=[ 480 640]';
c_i=[ 480 0]';
%Image plane to camera
o_i_to_c = getImagePlaneInCamera( o_i(1), o_i(2), f, intrinsic_param_F );
a_i_to_c = getImagePlaneInCamera( a_i(1), a_i(2), f, intrinsic_param_F );
b_i_to_c = getImagePlaneInCamera( b_i(1), b_i(2), f, intrinsic_param_F );
c_i_to_c = getImagePlaneInCamera( c_i(1), c_i(2), f, intrinsic_param_F );
%Camera to World
o_i_to_w= getCameraPointInWorld( cKw, o_i_to_c);
a_i_to_w= getCameraPointInWorld( cKw, a_i_to_c);
b_i_to_w= getCameraPointInWorld( cKw, b_i_to_c);
c_i_to_w= getCameraPointInWorld( cKw, c_i_to_c);

line([o_i_to_w(1) a_i_to_w(1)],[o_i_to_w(2) a_i_to_w(2)],[o_i_to_w(3) ....
    a_i_to_w(3)],'Color','bl','LineWidth',1);
line([o_i_to_w(1) c_i_to_w(1)],[o_i_to_w(2) c_i_to_w(2)],[o_i_to_w(3) ....
    c_i_to_w(3)],'Color','bl','LineWidth',1);
line([b_i_to_w(1) c_i_to_w(1)],[b_i_to_w(2) c_i_to_w(2)],[b_i_to_w(3) ....
    c_i_to_w(3)],'Color','bl','LineWidth',1);
line([b_i_to_w(1) a_i_to_w(1)],[b_i_to_w(2) a_i_to_w(2)],[b_i_to_w(3) ....
    a_i_to_w(3)],'Color','bl','LineWidth',1);
%Label
text(b_i_to_w(1),b_i_to_w(2),b_i_to_w(3),'640 x 480','Color','b');



u0 = intrinsic_param_F(1,3);
v0 = intrinsic_param_F(2,3);
o_ix=[0 0]';
a_ix=[0 2*v0]';
b_ix=[2*u0 0]';
c_ix=[2*u0 2*v0]';
%Image plane to camera
o_i_to_cx = getImagePlaneInCamera( o_ix(1), o_ix(2), f, intrinsic_param_F );
a_i_to_cx = getImagePlaneInCamera( a_ix(1), a_ix(2), f, intrinsic_param_F );
b_i_to_cx = getImagePlaneInCamera( b_ix(1), b_ix(2), f, intrinsic_param_F );
c_i_to_cx = getImagePlaneInCamera( c_ix(1), c_ix(2), f, intrinsic_param_F );
%Camera to World
o_i_to_wx= getCameraPointInWorld( cKw, o_i_to_cx);
a_i_to_wx= getCameraPointInWorld( cKw, a_i_to_cx);
```

```
b_i_to_wx= getCameraPointInWorld( cKw, b_i_to_cx);
c_i_to_wx= getCameraPointInWorld( cKw, c_i_to_cx);
line([o_i_to_wx(1) a_i_to_wx(1)],[o_i_to_wx(2) a_i_to_wx(2)],[o_i_to_wx(3)....
    a_i_to_wx(3)],'Color','g','LineWidth',1);
line([o_i_to_wx(1) c_i_to_wx(1)],[o_i_to_wx(2) c_i_to_wx(2)],[o_i_to_wx(3) ....
    c_i_to_wx(3)],'Color','g','LineWidth',1);
line([b_i_to_wx(1) c_i_to_wx(1)],[b_i_to_wx(2) c_i_to_wx(2)],[b_i_to_wx(3)....
    c_i_to_wx(3)],'Color','g','LineWidth',1);
line([b_i_to_wx(1) a_i_to_wx(1)],[b_i_to_wx(2) a_i_to_wx(2)],[b_i_to_wx(3)....
    a_i_to_wx(3)],'Color','g','LineWidth',1);
line([b_i_to_wx(1) o_i_to_wx(1)],[b_i_to_wx(2) o_i_to_wx(2)],[b_i_to_wx(3) ....
    o_i_to_wx(3)],'Color','g','LineWidth',1);
line([c_i_to_wx(1) a_i_to_wx(1)],[c_i_to_wx(2) a_i_to_wx(2)],[c_i_to_wx(3) ....
    a_i_to_wx(3)],'Color','g','LineWidth',1);
%Label
text(b_i_to_wx(1),b_i_to_wx(2),b_i_to_wx(3),'Image border limit','Color','g');

hold on;
%Label
text(o_i_to_w(1),o_i_to_w(2),o_i_to_w(3),'Image plane');

%Focal point- from camera to world
focal_point = [0 0 f 1]';
focal_point_c_to_w = getCameraPointInWorld( cKw, focal_point );
hold on;
plot3(focal_point_c_to_w(1),focal_point_c_to_w(2),focal_point_c_to_w(3),'y*');
hold on;
text(focal_point_c_to_w(1),focal_point_c_to_w(2),focal_point_c_to_w(3),'[u0, v0]');
hold on
line([c_to_w(1) focal_point_c_to_w(1)],[c_to_w(2) focal_point_c_to_w(2)],....
[c_to_w(3) focal_point_c_to_w(3)],'Color','y','LineWidth',1);
focal_point_l_text = [0 0 f/2 1 ]';
focal_point_l_text_w = getCameraPointInWorld( cKw, focal_point_l_text );
hold on;
text(focal_point_l_text_w(1),focal_point_l_text_w(2),focal_point_l_text_w(3),....
'Focal length','Color','y');

%2D Projection points
for i = 1: size(p2d,2)
    point_i_to_c = getImagePlaneInCamera( p2d(1,i), p2d(2,i), f, intrinsic_param_F );
    point_i_to_w = getCameraPointInWorld( cKw, point_i_to_c);
    hold on
    plot3(point_i_to_w(1),point_i_to_w(2),point_i_to_w(3),'xr');
    hold on
    plot3(xyz(1,i),xyz(2,i),xyz(3,i), '^g');
    hold on
    line([xyz(1,i) c_to_w(1)],[xyz(2,i) c_to_w(2)],[xyz(3,i) c_to_w(3)],....
    'Color','b','LineWidth',1);
end

end

function p_in_world = getCameraPointInWorld(  cKw, p_in_c )
```

```
%GE Summary of this function goes here
%   Detailed explanation goes here
p_in_world = cKw\p_in_c; %inv(cKw) * p_in_c;
end
```

```
function point = getImagePlaneInCamera( x, y, f, intrinsic_param )

u0 = intrinsic_param(1,3);
v0 = intrinsic_param(2,3);
au = intrinsic_param(1,1);
av = intrinsic_param(2,2);

ku = -au / f;
kv = -av / f;

xI = (-x+u0) / ku;
yI = (-y+v0) / kv;
zI = f;

point = [  xI; yI; zI; 1 ];


end
```