Real Time Image Processing: Summary about Analog and Digital Camera

Guillaume Lemaître
Heriot-Watt University, Universitat de Girona, Université de Bourgogne
g.lemaitre58@gmail.com

I. Part 1

Implementation of Hall method

A. Step 1 - 2

The first was to define all parameters of intrinsic and extrinsics matrices. All these parameters are defined in the main function (Appendix-A) in the part *Step 1*. We assigned parameters to the values given in the guideline. Then, We need to compute intrinsic and extrinsic matrices correctly. We create a function named *createIntExtMat* (Appendix-B) which returns intrinsic and extrinsic matrix following the different parameters given. With the set of parameters given, the intrinsic and extrinsic matrices returned are:

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

Intrinsic Matrix

$$ext = \begin{pmatrix} -0.9511 & -0.1816 & -0.2500 & 100 \\ -0.2939 & 0.7817 & 0.5501 & 0 \\ 0.0955 & 0.5967 & -0.7968 & 1500 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Extrinsic Matrix

B. Step 3

The aim of this step is to create a set of 3D points include in the range [-480:480;-480:480;-480:480]. The function *createSetPoints* (Appendix-C) create a number of points defined by the user in the range defined by the user. In the main function (Appendix-A), in the part *Parameters*, three variables are defined allowing to run the function:

- $nbpoints$: number of points which will be generated
- $range\_min$: it is the minimum range
- $range\_max$: it is the maximum range

X	Y	Z
-191	-28	-259
331	-293	-263
-316	-261	-62
-181	106	-67
-303	389	461
-59	-373	-232

Table I
Results of a generation of six points in the range [-480:480;-480:480;-480:480]

The function *rand()* of Matlab is used to generate randomly the number (Appendix-C). Six points are generated using the function *createSetPoints* where the parameters for the range are defined in the guideline. The result of this generation is presented in the table I.

C. Step 4

In this step, we have to project the previous points generated, in the image using the camera transformation defined in the *Step 2*. We created the function *projection3Dto2D* (Appendix-D) which allow to project a set of 3D points using a transformation matrix given. In this case, we wanted to use the transformation defined by the intrinsic and extrinsic matrices found in the *Step 2*. This transformation matrix is:

$$K_{pinhole} = \begin{pmatrix} -0.3324 & 0.0624 & -0.2662 & 363.5215 \\ -0.1207 & 0.4903 & 0.1028 & 298.6679 \\ 0.0001 & 0.0004 & -0.0005 & 1.0000 \end{pmatrix}$$

Pinhole model transformation

We obtained this matrix using matrix multiplication between the intrinsic and the extrinsic matrices. To project a point, we used the following formula:

X	Y
443.5347	252.4974
292.2981	84.2785
515.7339	222.7152
393.7798	432.4515
410.7323	643.8410
434.1792	101.9778

Table II
PROJECTION OF THE 3D POINTS FOUND IN STEP 3 USING THE PINHOLE MODEL TRANSFORMATION

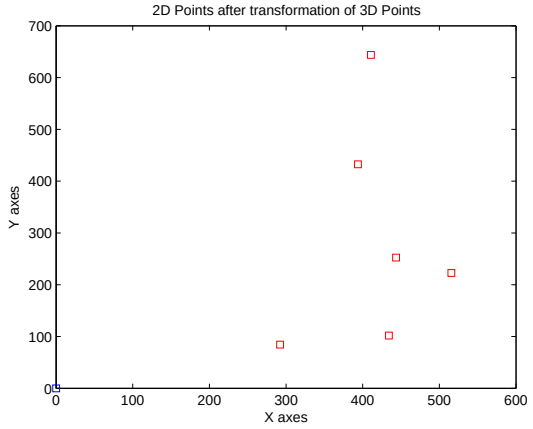

Figure 1. Plot of the 2D points projected

$$^{2d}p = ^{2d}K_{3d} \times ^{3d}p \tag{1}$$

where ^{2d}p is a 2D points, $^{2d}K_{3d}$ is the transformation matrix and ^{3d}p is a 3D points.

More precisely, we have:

$$\begin{pmatrix} s^I X_u \\ s^I Y_u \\ s \end{pmatrix} = ^{2d}K_{3d} \begin{pmatrix} ^W X_W \\ ^W Y_W \\ ^W Z_W \\ 1 \end{pmatrix}$$

If we apply the transformation on the 3D points generated in the *Step 3*, we obtained the points presented in the table II

D. Step 5

In this step, we want to plot the previous projected 2D points. The result is presented in figure 1.

The number of point is too small to see if the distribution of the points on the image is uniform and spread. To see the real distribution, we generated a large number of points (10,000 points) and see the real distribution. The result is shown on the figure 2.

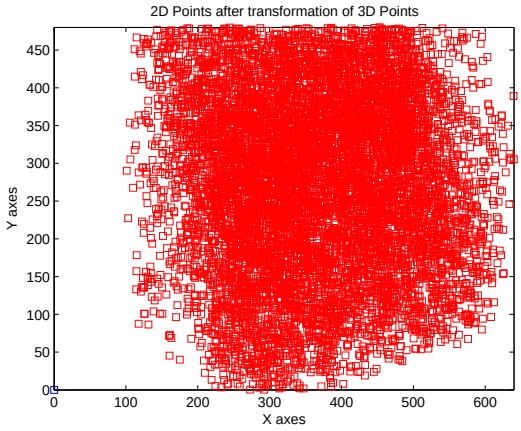

Figure 2. Distribution of the projection with a large number of points

We can see that the distribution of projected points is not uniform on the image. A vertical area, in the left of the image, on the range [0 100] is completely empty. We can conclude that the range fixed by the user is not enough large to fill the whole image.

However, the distribution of the points is not a matter because all points are generated non-linear and non-coplanar. Hence, each point gives a "new information" for the calibration.

E. Step 6

In this section, we computed the Hall transformation matrix. We created a function *calibration_Hall* (Appendix-E) which allow to compute the transformation using the equation defined by the Hall method. This function uses the set of 2D points and 3D points determined respectively at the *Step 3* and *Step 5*.

We will not show all equation which allow to determine the Hall transformation matrix. However, with the set of point (3D and 2D points), we determine two vectors B and Q. Then, we compute the Hall transformation following this equation:

$$A = (Q^t Q)^{-1} Q^t B; \tag{2}$$

Hence, using the previous set of points, we obtained the following Hall transformation matrix:

X	Y
443.4568	252.6355
292.1676	84.5002
515.9299	222.0899
393.3059	432.0810
410.4784	643.6807
434.1854	100.4632

Table III
POINTS 2D WITH NOISE ADDED

$$K_{hall} = \begin{pmatrix} -0.3324 & 0.0624 & -0.2662 & 363.5215 \\ -0.1206 & 0.4903 & 0.1028 & 298.6679 \\ 0 & 0.0004 & -0.0005 & 1 \end{pmatrix}$$

Hall transformation matrix

F. Step 7

In this step, we have to compare the difference of the Hall transformation matrix (*Step 6*) and the Pinhole transformation model (*Step 2*). To do it, we computed the difference between both matrices. We obtained the following difference matrix:

$$D_{ph} = \begin{pmatrix} 1.53e^{-14} & 7.76e^{-15} & 1.06e^{-14} & 5.12e^{-13} \\ 5.13e^{-15} & 1.03e^{-14} & 5.04e^{-15} & 2.22e^{-12} \\ 4.28e^{-17} & 2.09e^{-17} & 2.74e^{-17} & 0 \end{pmatrix}$$

Difference matrix between Hall transformation matrix and Pinhole transformation model

We can see that the difference between both matrices is very small.

G. Step 8

In this step, we have to add a Gaussian noise at all 2D points projected with a standard deviation of 0.5. It allows to have 95% of the noise generated in the range [-1 1]. *addNoise* function (Appendix-F) allows to generate and add a noise with a standard deviation set by the user.

We add the noise on the set of points projected in the *Step 5* presented in the table II. The coordinates of the points after added the noise is presented on the table III

Figure 3 shows the 2D points without noise (red) and the 2D points with noise (green).

Figure 4 shows more precisely the difference between 2D points with and without noise.

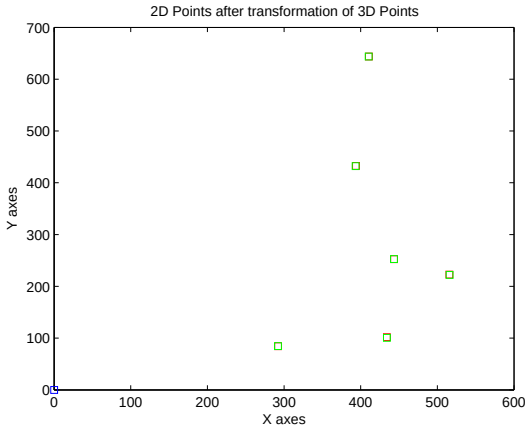

Figure 3. 2D points without noise (red) 2D points with noise (green)

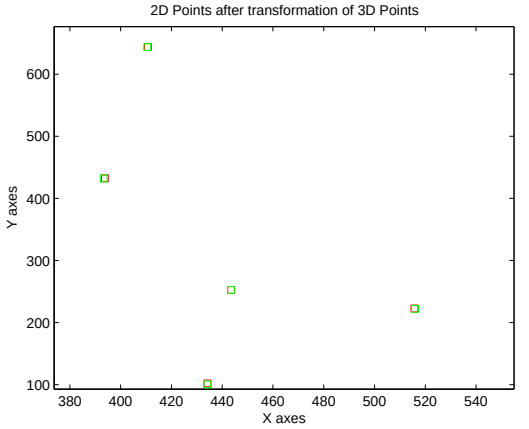

Figure 4. Zoom on: 2D points without noise (red) 2D points with noise (green)

After generate noisy points, we repeated *Step 6* to find the Hall transformation matrix with noise and we obtained the following matrix:

$$K_{hall} = \begin{pmatrix} -0.3296 & 0.0714 & -0.2681 & 363.4586 \\ -0.1172 & 0.4981 & 0.1030 & 299.6868 \\ 0 & 0.0004 & -0.0005 & 1 \end{pmatrix}$$

Hall transformation matrix with noise

Then, we computed the difference matrix between Hall transformation with noise and without noise. This matrix is as follow:

$$D_{hnh} = \begin{pmatrix} 0.0029 & 0.0090 & 0.0019 & 0.0629 \\ 0.0035 & 0.0077 & 0.0002 & 1.0189 \\ 0.0000 & 0.0000 & 0.0000 & 0 \end{pmatrix}$$

X	Y
443.1487	252.5450
292.1652	84.4652
516.1130	222.0681
393.5039	432.1376
410.3552	643.6619
434.2419	100.5734

Table IV
POINTS 2D PROJECTED USING THE HALL TRANSFORMATION MATRIX WITH NOISE

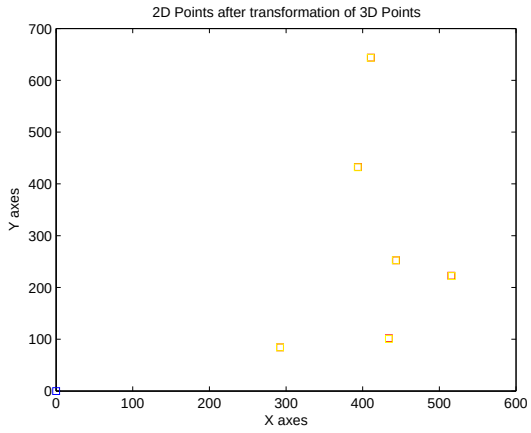

Figure 5. 2D points without noise (red) 2D points projected with the Hall transformation matrix with noise (yellow)

Difference matrix between Hall transformation matrix with noise and Hall transformation matrix without noise

We can see that the difference is not negligible.

Now, we have to recompute the projection of 2D points using the 3D points of the *Step 3* and the Hall transformation matrix with noise. The results obtained are presented in the table IV.

Figure 5 shows the 2D points without noise (red) and the 2D points projected with the Hall transformation matrix with noise (yellow).

Figure 6 shows more precisely the difference between 2D points without noise (red) and the 2D points projected with the Hall transformation matrix with noise (yellow).

To compute the accuracy of the Hall transformation matrix with noise, we created a function *compute_Err* (Appendix-G) which allows to compute first the Euclidean distance between 2D points found with the model and the previous transformation and then allows to compute the mean and standard deviation of the Euclidean

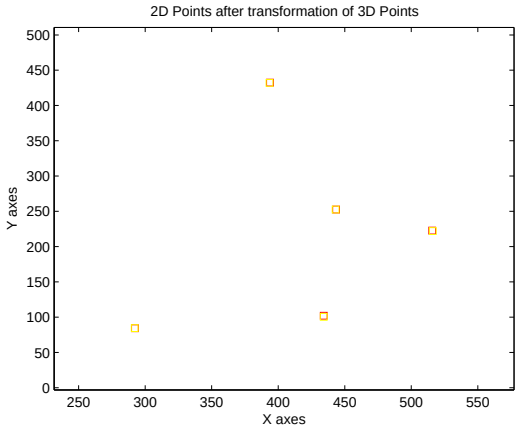

Figure 6. Zoom on: 2D points without noise (red) and the 2D points projected with the Hall transformation matrix with noise (yellow)

distance. Hence, we obtained:

- Mean: 0.6016
- Standard deviation: 0.4290

H. Step 9

In this step, we have to check that if the number of 3D points generated increase, the accuracy of Hall transformation matrix with better. We increased the number of points at 50 and computed the mean and the standard deviation of the Euclidean distance between 2D points found with the model and the Hall transformation matrix model and we found a better accuracy than previously:

- Mean: 0.1643
- Standard deviation: 0.0879

II. PART 2

Implementation of Faugeras method

A. Step 10

In this section, we have to implement the Faugeras method. We created the function *calibration_faugeras* (Appendix-H) which allows to compute the vector X containing all parameters to compute the transformation matrix. We will not present all equations in this part. However, with the set of point (3D and 2D points), we determine two vectors B and Q. Then, we compute the Faugeras vector X following this equation:

$$X = (Q^t Q)^{-1} Q^t B; \tag{3}$$

After to find the X vector, we have to extract all parameters to construct intrinsic and extrinsic matrices. (extract_param_faugeras) (Appendix-I)) allows to extract all parameters to construct intrinsic and extrinsic matrices. Then, we can use *createIntExtMat* (Appendix-B) function to create intrinsic and extrinsic matrices with the previous parameters determined.

We determined intrinsic and extrinsic matrices with the 3D points and 2D points generated in *Step 3* and *Step 4*. We computed differences matrices to see the differences between parameters and we obtained:

$$int = \begin{pmatrix} 8,52e^{-12} & 0 & -2,84e^{-12} & 0 \\ 0 & 6,82e^{-12} & -1,02e^{-12} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Difference of intrinsic Matrix

$$ext = \begin{pmatrix} 1,11e^{-16} & 1,83e^{-15} & 9,71e^{-16} & 7,07e^{-12} \\ 7,77e^{-16} & 1,11e^{-15} & 2,10e^{-15} & 2,03e^{-12} \\ 2,42e^{-15} & 1,33e^{-15} & 7,77e^{-16} & 1,59e^{-11} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Difference of extrinsic Matrix

We can see that the difference between parameters is negligible.

B. Step 11

In this part, we repeat the previous step but using 2D noisy points of the *Step 8*. We computed the Faugeras transformation matrix and computed 2D projected points. We obtain the following accuracy:

- Mean: 0.6016
- Standard deviation: 0.4290

The mean and standard deviation is exactly the same than Hall method because we do not use distorted image and Pinhole model. We present the results for different standard deviation when we generate the noise: With a standard deviation of 1.0:

1) Hall
   - Mean: 1.7509
   - Standard deviation: 1.2671
2) Faugeras
   - Mean: 1.7509
   - Standard deviation: 1.2671

With a standard deviation of 1.5:

1) Hall
   - Mean: 1.4275
   - Standard deviation: 0.6490
2) Faugeras
   - Mean: 1.4275
   - Standard deviation: 0.6490

We can see that for different noise level, Faugeras and Hall are equivalent.

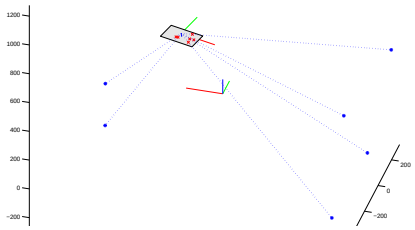

Figure 7. Representation in the 3D world

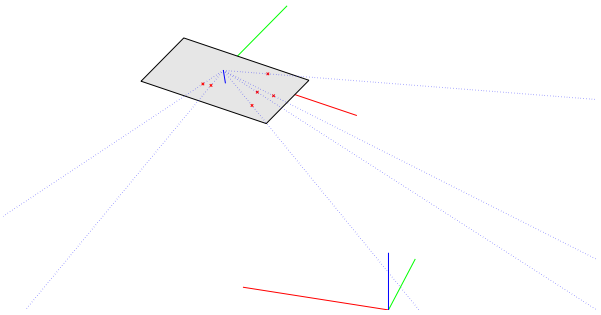

Figure 8. Zoom on the image to check if the optical across the image in good points (Bottom view)

III. PART 3

A. Step 12

In this part, we decided to plot in 3D the environment. These drawing are in the main function (Appendix-A) in *Step 12*.

The world coordinate system is the system defined in [0,0,0]. To draw camera, and 3D points, we inverse the extrinsic matrix to find W_C^K. To plot the 2D points, we use the parameters of the intrinsic matrix.

We wanted to check if the line from the camera center to the 3D points across the image exactly on projection computed. Figures 7 and 8 shows a good results.

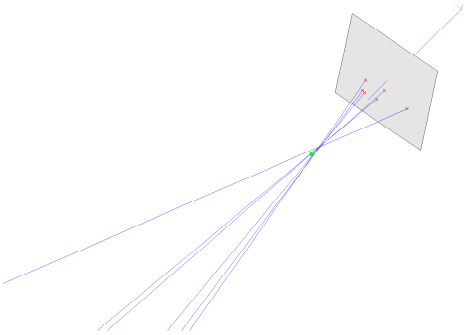

Figure 9. Optical rays do not across exactly the focal point

However, the figure 9 shows that the optical rays do not across exactly the focal point (yellow point) but pass near of this point.

IV. Conclusion

In this paper, we implemented two linear methods to calibrate camera: Hall and Fagueras methods. The main difficulties of this work is that we only simulate the calibration. It should preferable to understand to apply in the real world with real camera and different type of images to compare the result (undistorted and distorted images).

A. Main function

This code is the main function.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Guillaume Lemaitre
%%% main file
%%% Calibration cameras
%%% Universitat de Girona - Heriot-Watt University - Universit de
%%% Bourgogne
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Matlab Initialisation
clc;
clear all; close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Parameters
nbpoints = 6;
f = 80;
height_im = 640;
width_im = 480;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PART 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Model camera given

%%% Step 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Variables to define intrinsic and extrinsic matrices
%%% Intrasec parameters
au=557.0943;
av=712.9824;
u0=326.3819;
v0=298.6679;

%%% Extrinsic parameters
Phix=0.8*pi/2;
Phiy=-1.8*pi/2;
Phix1=pi/5;
tx=100;
ty=0;
tz=1500;

%%% Computation of rotational matrices XYX Euler
%%% Rotation R(X,phix)
%%% Allocation RX1
RX1 = zeros(3,3);
%%% Compute parameters
RX1(1,1) = 1;
RX1(2,2) = cos(Phix);
RX1(3,2) = sin(Phix);
RX1(2,3) = - sin(Phix);
RX1(3,3) = cos(Phix);

%%% Rotation R(Y,phiy)
%%% Allocation RY1
RY1 = zeros(3,3);
%%% Compute parameters
RY1(1,1) = cos(Phiy);
RY1(3,1) = - sin(Phiy);
```

```
RY1(2,2) = 1;
RY1(1,3) = sin (Phiy);
RY1(3,3) = cos(Phiy);

%%% Rotation R(X,phix1)
%%% Allocation RX2
RX2 = zeros(3,3);
%%% Compute parameters
RX2(1,1) = 1;
RX2(2,2) = cos(Phix1);
RX2(3,2) = sin(Phix1);
RX2(2,3) = -sin(Phix1);
RX2(3,3) = cos(Phix1);

%%% Computation of the rotationnal matrice
rMat = RX1*RY1*RX2;

%%% Step 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Compute Intrinsic and Extrinsic matrix
[int_Mat_mod,ext_Mat_mod] = createIntExtMat(au,av,u0,v0,rMat(1,1),rMat(1,2),
rMat(1,3),rMat(2,1),rMat(2,2),rMat(2,3),rMat(3,1),rMat(3,2),rMat(3,3),tx,ty,tz);

%%% Compute transformation matrice of the model
model = int_Mat_mod*ext_Mat_mod;
disp('Pinhole Model: ')
disp(model/model(3,4))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Step 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Creation of set of points

points3d = createSetPoints(nbpoints,-480,480);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Step 4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Computation of projection points on the image

points2d = projectiion3Dto2D(points3d,model);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Step 5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot the 2D Points
%%% Plot (0,0)
figure(1);
plot(0,0,'bs');
hold on;
%%% Plot all 2D points
plot(points2d(:,1),points2d(:,2),'rs');
title('2D Points after transformation of 3D Points')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Step 6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Compute Hall Matrix

hall_Mat = calibration_Hall(points2d, points3d);
disp('Hall Matrix without noise')
disp(hall_Mat)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Step 7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Compare both matrix computing the difference
diff_model_hall_Mat = (model/model(3,4) - hall_Mat);
disp('Difference between Pinhole Model and Hall Calibration without noise')
disp(diff_model_hall_Mat)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Step 8 - Step 9 (Changing nbpoints parameters)%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Adding gaussian noise of range [-1,1]
points2dnoise = addNoise(points2d,nbpoints,0.5);

%%% Plot the 2D Points with noise
figure(1);
hold on;
plot(points2dnoise(:,1),points2dnoise(:,2),'gs');

%%% Compute the noisy Hall matrix
hall_noise_Mat = calibration_Hall(points2dnoise, points3d);
disp('Hall Matrix with noise')
disp(hall_noise_Mat)

%%% Compute the projection using the noisy hall matrix
points2dnoise_hall = projectiion3Dto2D(points3d,hall_noise_Mat);

%%% Plot the 2D Points projected with hall calibration and noise
figure(1);
hold on;
plot(points2dnoise_hall(:,1),points2dnoise_hall(:,2),'ys');

%%% Compute the distance mean and std of original projection and hall
%%% calibration projection
[eucl_dist_hall_noise,mean_hall_noise,std_hall_noise] = compute_Err(points2d,points2dnoise_hall);
%%% Display the results
%disp('Euclidean distance between projection with model and projection with hall projection')
%disp(eucl_dist_hall_noise)
disp('Mean between projection with model and projection with hall projection')
disp(mean_hall_noise)
disp('Standard deviation between projection with model and projection with hall projection')
disp(std_hall_noise)

%%% Compare both matrix computing the difference
diff_model_hall_noise_Mat = (model/model(3,4) - hall_noise_Mat);
disp('Difference between Pinhole Model and Hall Calibration with noise')
disp(diff_model_hall_noise_Mat)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PART 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Step 10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Compute vector X for faugeras calibration
X_vec_faugeras = calibration_faugeras(points2d, points3d);

%%% Extracts parameters from faugeras vector
[u0_fig,v0_fig,au_fig,av_fig,r1_fig,r2_fig,r3_fig,tx_fig,ty_fig,tz_fig] =
extract_param_faugeras(X_vec_faugeras);

%%% Constructs the matrices with parameters
[int_Mat_fig,ext_Mat_fig] = createIntExtMat(au_fig,av_fig,u0_fig,v0_fig,r1_fig(1),r1_fig(2),
r1_fig(3),r2_fig(1),r2_fig(2),r2_fig(3),r3_fig(1),r3_fig(2),r3_fig(3),tx_fig,ty_fig,tz_fig);
disp('Intrinsic Matrix found with faugeras calibration')
```

```
disp(int_Mat_fig)
disp('Extrinsic Matrix found with faugeras calibration')
disp(ext_Mat_fig)

%%% Compute the difference between the paramters
%%% Intrinsic matrix
diff_model_fig_int = (int_Mat_mod - int_Mat_fig);
disp('Difference between Intrinsic matrix: Pinhole parameters and faugeras paramters without noise')
disp(diff_model_fig_int)
%%% Extrinsic matrix
diff_model_fig_ext = (ext_Mat_mod - ext_Mat_fig);
disp('Difference between Extrinsic matrix: Pinhole parameters and faugeras paramters without noise')
disp(diff_model_fig_ext)

%%% Compute faugeras matrix
faugeras_Mat = int_Mat_fig*ext_Mat_fig;
disp('faugeras matrix without noise')
disp(faugeras_Mat)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Step 11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Compute faugeras with noise
X_vec_faugeras_noise = calibration_faugeras(points2dnoise, points3d);

%%% Extracts parameters from faugeras vector
[u0_fig_noise,v0_fig_noise,au_fig_noise,av_fig_noise,r1_fig_noise,r2_fig_noise,
r3_fig_noise,tx_fig_noise,ty_fig_noise,tz_fig_noise] = extract_param_faugeras(X_vec_faugeras_noise);

%%% Constructs the matrices with parameters
[int_Mat_fig_noise,ext_Mat_fig_noise] = createIntExtMat(au_fig_noise,av_fig_noise,u0_fig_noise,
v0_fig_noise,r1_fig_noise(1),r1_fig_noise(2),r1_fig_noise(3),r2_fig_noise(1),r2_fig_noise(2),
r2_fig_noise(3),r3_fig_noise(1),r3_fig_noise(2),r3_fig_noise(3),tx_fig_noise,ty_fig_noise,tz_fig_noise);
disp('Intrinsic Matrix found with faugeras calibration noisy')
disp(int_Mat_fig_noise)
disp('Extrinsic Matrix found with faugeras calibration noisy')
disp(ext_Mat_fig_noise)

%%% Compute the difference between the paramters
%%% Intrinsic matrix
diff_model_fig_int_noise = (int_Mat_fig_noise - int_Mat_fig);
disp('Difference between Intrinsic matrix: faugeras with noise parameters and faugeras paramters without noise')
disp(diff_model_fig_int_noise)
%%% Extrinsic matrix
diff_model_fig_ext_noise = (ext_Mat_fig_noise - ext_Mat_fig);
disp('Difference between Extrinsic matrix: faugeras with noise and faugeras paramters without noise')
disp(diff_model_fig_ext_noise)

%%% Compute faugeras matrix
faugeras_Mat_noise = int_Mat_fig_noise*ext_Mat_fig_noise;
disp('faugeras matrix with noise')
disp(faugeras_Mat_noise)

%%% Compute the projection
%%% Compute the projection using the noisy hall matrix
points2dnoise_faugeras = projectiion3Dto2D(points3d,faugeras_Mat_noise);

%%% Compute the distance mean and std of original projection and hall
%%% calibration projection
[eucl_dist_fig_noise,mean_fig_noise,std_fig_noise] = compute_Err(points2d,points2dnoise_faugeras);
%%% Display the results
%disp('Euclidean distance between projection with model and projection with faugeras projection')
%disp(eucl_dist_fig_noise)
disp('Mean between projection with model and projection with faugeras projection')
disp(mean_fig_noise)
disp('Standard deviation between projection with model and projection with faugeras projection')
disp(std_fig_noise)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PART 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Step 12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Plot World coordinates
figure(2);
draw_axis([0 0 0],100,100,100);

%%% Plot Camera Coordinates
%%% Compute transformation WORLD CAMERA
Rext = [ext_Mat_mod(1,1) ext_Mat_mod(1,2) ext_Mat_mod(1,3) ; ext_Mat_mod(2,1) ext_Mat_mod(2,2) ext_Mat_mod(2,3) ;
Text = [ext_Mat_mod(1,4) ext_Mat_mod(2,4) ext_Mat_mod(3,4)]';
WRC = Rext';
WTC = -WRC*Text;
WKC = [WRC(1,:) WTC(1) ; WRC(2,:) WTC(2) ; WRC(3,:) WTC(3) ; 0 0 0 1];
%%% Compute the camera center
wpc = WKC*[0 0 0 1]';
%%% Compute vectors to draw axis
wpc2 = WKC*[100 0 0 1]';
wpc3 = WKC*[0 100 0 1]';
wpc4 = WKC*[0 0 100 1]';
%%% Plot
hold on;
plot3([wpc(1) wpc2(1)],[wpc(2) wpc2(2)],[wpc(3) wpc2(3)],'r');
hold on;
plot3([wpc(1) wpc3(1)],[wpc(2) wpc3(2)],[wpc(3) wpc3(3)],'g');
hold on;
plot3([wpc(1) wpc4(1)],[wpc(2) wpc4(2)],[wpc(3) wpc4(3)],'b');
hold on;

%%% Plot Focal length
%%% Compute Focal point see by the world
wpcfocal = WKC*[0 0 2*f 1]';
hold on;
plot3(wpcfocal(1),wpcfocal(2),wpcfocal(3),'y*');

%%% Plot the Image limits
%%% Compute Borders of the image
%%%Compute ku and kv;
ku = -(au/f); kv = -(av/f);
wpcim1 = WKC*[(-u0)/ku (-v0)/kv f 1]';
wpcim2 = WKC*[(-u0)/ku (v0)/kv f 1]';
wpcim3 = WKC*[(u0)/ku (-v0)/kv f 1]';
wpcim4 = WKC*[(u0)/ku (v0)/kv f 1]';
%%% Plot the image
fill3([wpcim1(1) wpcim2(1) wpcim4(1) wpcim3(1)],[wpcim1(2) wpcim2(2) wpcim4(2)
 wpcim3(2)],[wpcim1(3) wpcim2(3) wpcim4(3) wpcim3(3)],[1 1 1]*0.9);
hold on;

%%% Plot 3D Points
plot3(points3d(:,1),points3d(:,2),points3d(:,3), 'b*');

tempPoints = zeros(4,1);
%%% Transform 2D points in pixels then transform see form the world
ppoints2d = zeros(nbpoints,4);
for i = 1:nbpoints
    tempPoints(1)= (u0 - points2d(i,1))/ku;
    tempPoints(2)= (v0 - points2d(i,2))/kv;
    tempPoints(3) = f;
    tempPoints(4) = 1;
    ppoints2d(i,:) = WKC*tempPoints;
    plot3(ppoints2d(i,1),ppoints2d(i,2),ppoints2d(i,3),'xr');
end
```

```
%%% Plot the lines between en focal points and 3D points
for i = 1:nbpoints
    hold on;
    line([wpc(1) points3d(i,1)],[wpc(2) points3d(i,2)],[wpc(3) points3d(i,3)],'LineStyle',':');
    hold on;
    %line([ppoints2d(i,1) points3d(i,1)],[ppoints2d(i,2) points3d(i,2)],[ppoints2d(i,3)
    % points3d(i,3)],'LineStyle',':');
end
```

B. *createIntExtMat function*

This function create and intrinsic and extrinsic matrices using parameters given in arguments

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function corresponding to the step 2
%%% Allow to obtain intrinsic et extrinsic matrices
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [int_Mat,ext_Mat] = createIntExtMat(au,av,u0,v0,r11,r12,r13,r21,r22,
r23,r31,r32,r33,tx,ty,tz)

%%% Computation of the intrinsic matrix
%%% Allocation of the matrix
int_Mat = zeros(3,4);
%%% Fill the matrix with parameters defined previously
int_Mat(1,1) = au;
int_Mat(2,2) = av;
int_Mat(1,3) = u0;
int_Mat(2,3) = v0;
int_Mat(3,3) = 1;

%%% Computation of the extrinsic matrix
%%% Allocation of the matrix
ext_Mat = zeros(4,4);
%%% Fill the matrix with parameters defined previously
ext_Mat(1,1) = r11;
ext_Mat(1,2) = r12;
ext_Mat(1,3) = r13;
ext_Mat(1,4) = tx;
ext_Mat(2,1) = r21;
ext_Mat(2,2) = r22;
ext_Mat(2,3) = r23;
ext_Mat(2,4) = ty;
ext_Mat(3,1) = r31;
ext_Mat(3,2) = r32;
ext_Mat(3,3) = r33;
ext_Mat(3,4) = tz;
ext_Mat(4,4) = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C. *createSetPoints function*

This function create a set of 3D points in the range [minRange maxRange]

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function corresponding at the step 3
%%% Function to create a set of number
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function setPoints = createSetPoints(numberPoints, minRange, maxRange)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Initialisation

%%% Allocation of the set of points
setPoints = zeros(numberPoints,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Loop function

for i = 1:numberPoints
    setPoints(i,:) = minRange + (maxRange - minRange).*rand(1,3);
end

%%% Conversion in integer matrice
setPoints = round(setPoints);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

D. projectiion3Dto2D function

This function project 3D points in 2D points on the image

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function corresponding to the step 4
%%% Compute the projection of 3D points with given method matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Points2D] = projectiion3Dto2D(points3D,matrix)

%%% Intialisation of 2D Points
sizetemp = size(points3D);
nbpoints = sizetemp(1);
Points2Dtemp = zeros(3,nbpoints);
Points2D = zeros(nbpoints,2);
%%% Compute the 2D Points
for i = 1:nbpoints
    Points2Dtemp(:,i) = matrix*[points3D(i,:)' ; 1];
    Points2D(i,1) = Points2Dtemp(1,i)/Points2Dtemp(3,i);
    Points2D(i,2) = Points2Dtemp(2,i)/Points2Dtemp(3,i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

E. calibration Hall function

This function allow to construct the calibration of the Hall method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function corresponding to the step 6
%%% Compute Matrix of Hall
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [hall_Mat] = calibration_Hall(Points2D, setPoints)

%%% Initialisation
sizetemp = size(setPoints);
nbpoints = sizetemp(1);
Q = zeros(2*nbpoints, 11);

%%% Computation of the matrix Q
```

```
for i = 1:nbpoints
    Q((2*i -1),:) = [setPoints(i,1) setPoints(i,2) setPoints(i,3)
     1 0 0 0 0 (-Points2D(i,1)*setPoints(i,1)) (-Points2D(i,1)*setPoints(i,2))
      (-Points2D(i,1)*setPoints(i,3))];
    Q((2*i),:) = [0 0 0 0 setPoints(i,1) setPoints(i,2)
     setPoints(i,3) 1 (-Points2D(i,2)*setPoints(i,1)) (-Points2D(i,2)*setPoints(i,2))
      (-Points2D(i,2)*setPoints(i,3))];
end

%%% Initialisation
B = zeros(2*nbpoints, 1);
%%% Computation of B matrix
for i = 1:nbpoints
    B(2*i -1) = Points2D(i,1);
    B(2*i) = Points2D(i,2);
end

%%% Computation of A matrix
Ascal = (Q'*Q)\Q'*B;
hall_Mat = [Ascal(1:4)';Ascal(5:8)';Ascal(9:11)' 1];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

F. addNoise function

This function add noise to a set of 2D points

```
function [Points2Dnoise] = addNoise(Points2D,nbpoints,std)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Adding gaussian noise to all points
%%% Allocate set of points
Points2Dnoise = zeros(nbpoints,2);
for i = 1:nbpoints
    Points2Dnoise(i,1) = Points2D(i,1) + std.*randn(1);
    Points2Dnoise(i,2) = Points2D(i,2) + std.*randn(1);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

G. compute Err function

This function that compute statistic errors between to set of 2D points

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function corresponding to the step 8
%%% Compute the errors between original points and created after
%%% calibration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [eucl_dist,meanv,stdv] = compute_Err(points2d,points2dothers)

%%% Initialisation
sizetemp = size(points2d);
nbpoints = sizetemp(1);
eucl_dist = zeros(nbpoints,1);

%%% Compute the euclidean distances
for i = 1:nbpoints
    eucl_dist(i) = sqrt((points2d(i,1) - points2dothers(i,1))^2 + (points2d(i,2) - points2dothers(i,2))^2);
end

meanv = mean(eucl_dist);
stdv = std(eucl_dist);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

H. calibration faugeras function

This function allow to compute the X vector of the Figueras calibration

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function corresponding to the step 10
%%% Compute Matrix of Figueras
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [X] = calibration_faugeras(points2d, points3d)


%%% Initialisation
sizetemp = size(points3d);
nbpoints = sizetemp(1);
Q = zeros(2*nbpoints, 11);

%%% Computation of the matrix Q
for i = 1:nbpoints
    Q((2*i -1),:) = [points3d(i,1) points3d(i,2) points3d(i,3)
    (-points2d(i,1)*points3d(i,1)) (-points2d(i,1)*points3d(i,2))
    (-points2d(i,1)*points3d(i,3)) 0 0 0 1 0];
    Q((2*i),:) = [0 0 0 (-points2d(i,2)*points3d(i,1)) (-points2d(i,2)*points3d(i,2))
     (-points2d(i,2)*points3d(i,3)) points3d(i,1) points3d(i,2) points3d(i,3) 0 1];
end

%%% Initialisation
B = zeros(2*nbpoints, 1);
%%% Computation of B matrix
for i = 1:nbpoints
    B(2*i -1) = points2d(i,1);
    B(2*i) = points2d(i,2);
end

%%% Computation of X vector
X = (Q'*Q)\Q'*B;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

I. extract param faugeras function

This function allow to extract parameters from the X vector of the Faugeras method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function corresponding to the step 10
%%% Extract parameters of Figueras Vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [u0,v0,au,av,r1,r2,r3,tx,ty,tz] = extract_param_faugeras(X)

T1 = X(1:3)';
T2 = X(4:6)';
T3 = X(7:9)';
C1 = X(10);
C2 = X(11);

%%% Compute extrinsics parameters
u0 = (T1*T2')/(norm(T2)^2);
v0 = (T2*T3')/(norm(T2)^2);
au = (norm(cross(T1',T2')))/(norm(T2)^2);
av = (norm(cross(T2',T3')))/(norm(T2)^2);

%%% Rotation parameters
r1 = (norm(T2)/(norm(cross(T1',T2'))))*(T1 - (((T1*T2')/(norm(T2)^2))*T2));
r2 = (norm(T2)/(norm(cross(T2',T3'))))*(T3 - (((T2*T3')/(norm(T2)^2))*T2));
```

```
r3 = T2/(norm(T2));

%%% Transformation parameters
tx = (norm(T2)/(norm(cross(T1',T2'))))*(C1 - (((T1*T2')/(norm(T2)^2))));
ty = (norm(T2)/(norm(cross(T2',T3'))))*(C2 - (((T2*T3')/(norm(T2)^2))));
tz = 1/(norm(T2));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

J. draw axis function

This function draw axis

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function corresponding to the step 6
%%% Compute Matrix of Hall
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function draw_axis(center, X, Y, Z)

plot3([center(1) X],[center(2) center(2)],[center(3) center(3)],'r');
hold on;
plot3([center(1) center(1)],[center(2) Y],[center(3) center(3)],'g');
hold on;
plot3([center(1) center(1)],[center(2) center(2)],[center(3) Z],'b');
hold on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```