VISUAL PERCEPTION

MULTIPLE VIEW GEOMETRY LAB REPORT

Gobichettipalayam Shriarulmozhivarman
Masters in Computer Vision

Submitted to Devesh Adlakha
Université de Bourgogne

22 April 2021

Chapter 1

Lab 1 Camera Calibration : DLT

1.1 Camera Calibration

Camera calibration is the process of extracting the parameters of a camera,intrinsic parameters and extrinsic parameters from 2D image coordinates and 3D world coordinates .Intrinsic parameter are the camera's internal properties such as, its focal length, skew angle,and image centre.Extrinsic parameters of the camera are the 3D position and orientation in the world.In total we have 11 parameter to describe the pin hole camera Model that can be used to form images from the real world.Let m be the homogeneous coordinates of the image points and M be the Corresponding points in the World coordinates then,$m_i \cong PM_i$ is in equality up to a scale for all the index of the image,where P is the Projective matrix.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = P_{3,4} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1.1}$$

where $P = KR[I| - T]$,K is the intrinsic parameter,R is the rotation matrix and T is the position of the camera in the 3D world.As P is the unknown that we to find,let is be filled with variable names. And P be $P = \begin{bmatrix} P11 & P12 & P13 & P14 \\ P21 & P22 & P23 & P24 \\ P31 & P23 & P33 & P34 \end{bmatrix}$ so,

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} P11 & P12 & P13 & P14 \\ P21 & P22 & P23 & P24 \\ P31 & P23 & P33 & P34 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1.2}$$

$$x = \frac{p_{11}X + p_{12}Y + p_{13}Z + p_{14}}{p_{31}X + p_{32}Y + p_{33} + p_{34}} \tag{1.3}$$

$$y = \frac{p_{21}X + p_{22}Y + p_{23}Z + p_{24}}{p_{31}X + p_{32}Y + p_{33} + p_{34}} \tag{1.4}$$

we have 2 equations for a point and 11 unknowns, so we at-least need 6 points to get all the parameters. this is can be done by two methods equation rearranging and Kronecker product methods.

1.1.1 Linear equation Method

For the linear equation to get the parameters we have to rearrange the equation in such a way the we solve for $AX = 0$.

Let us define $A = \begin{bmatrix} P_{11} \\ P_{12} \\ P_{13} \\ P_{14} \end{bmatrix}$,$B = \begin{bmatrix} P_{21} \\ P_{22} \\ P_{23} \\ P_{24} \end{bmatrix}$,$B = \begin{bmatrix} P_{31} \\ P_{32} \\ P_{33} \\ P_{34} \end{bmatrix}$ which implies that $m \cong PM$ will be

$$\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = \begin{bmatrix} A^T \\ B^T \\ C^T \end{bmatrix} Mi \tag{1.5}$$

and $x_i = A^T Mi$, $y_i = B^T Mi$,$w_i = C^T Mi$,so that $x_i = \frac{A^T Mi}{C^T Mi}$ and $y_i = \frac{B^T Mi}{C^T Mi}$ and rearrange it as,

$$-M_i^T A + 0 + x_i M_i^T C = 0$$
$$0 + -M_i^T B + x_i M_i^T C = 0$$

$$\alpha_{xi}^T p = 0$$
$$\alpha_{yi}^T p = 0$$

where α_{xi}^T is $[-M_i^T, 0^T, x_i M_i^T]$, α_{yi}^T is $[0^T, -M_i^T, y_i M_i^T]$ and p = $\begin{bmatrix} A \\ B \\ C \end{bmatrix}_{12,1}$

stacking the α_{xi}^T and α_{yi}^Tif we stack n points to have 2n x12 matrix,this can be solved by the Singular value decomposition(SVD), as this is of form $AX = 0$.stacking atleast 6 points,ensures a solution that can be used to extract the parameters from the projection Matrix.

1.1.2 Kronecker Product Method

We know that the m_i and M_i are in homogeneous coordinates and $m_i \cong PM_i$, and m_i,PM_i both of them are coliner.we known that the cross-product between two colinear vectors is zero.

$$m_i \times PM_i = 0 \tag{1.6}$$

$$[m_i]_\times \times PM_i = 0 \tag{1.7}$$

where $[m_i]_\times$is the skew-symmetric matrix for all i in the image points.Applying the Kronecker product to equation 1.7.

$$vec([m_i]_\times \times PM_i = 0) \tag{1.8}$$

$$(M_i^T \otimes [m_i]_\times)vec(P) = 0) \tag{1.9}$$

we get three equations for each point in the image and world coordinates, only two of them are linearly independent because it is product of a rank 1 matrix by a rank 2 matrix.we stack n points to have 2n x12 matrix,this can be solved by the Singular value decomposition(SVD), as this is of form $AX = 0$.stacking atleast 6 points,ensures a solution that can be used to extract the parameters from the projection Matrix.

1.1.3 Extracting Parameters from Projection matrix(P)

Intrinsic parameter and Extrinsic parameter can be extracted once we the projection matrix P.we can take $H = KR$ and this gives $P = KR[I| - T] = [H|h]$,and $h = -KRT$,so we get the camera location from $T = -inv(H)h$.we know that the K is a upper triangular matrix, and R is a rotational matrix (ie $R^T = R^{-1}$).

1.1.4 QR decomposition

we can use QR decomposition (QR decomposition of a matrix A = QR can be of an orthogonal matrix Q and R an upper triangular matrix),we cannot apply it directly as H in not in the corresponding form,but we can apply QR on the inverse of H so,

$$(transpose(R), inv(K)) = QRDecomposition(inv(H)) \tag{1.10}$$

but sometimes we have to insert two matrix such that their products are Identity matrix of size 3 to get the correct after the decomposition.

1.1.5 Cholesky decomposition

Cholesky decomposition gives the a product of a lower triangular matrix and its conjugate transpose.we use this to get the inverse K upper triangular matrix by inversing the product of H and H transpose.

$$inv(K)) = Choleskydecomposition(inv(H \times H.T)) \tag{1.11}$$

then we can get R from the H,as we now the K.This has the advantage of not inserting two matrix whose product are Identity.

1.1.6 Rotation angles

The rotational angles can be extracted from the rotational matrix with is of the "ZYX" form in the euler rotations.so we get the angles with the trigonometric operations on the vlues in the rotational matrix.

$\alpha_x = \arctan 2d(R_{21}, R_{11}))$, $\beta_y = \arctan 2d(-R_{31}, \sqrt{R_{32}^2 + R_{33}^2})$ and $\gamma_z = \arctan 2d(R_{32}, R_{33}))$

1.2 Matlab Implementation

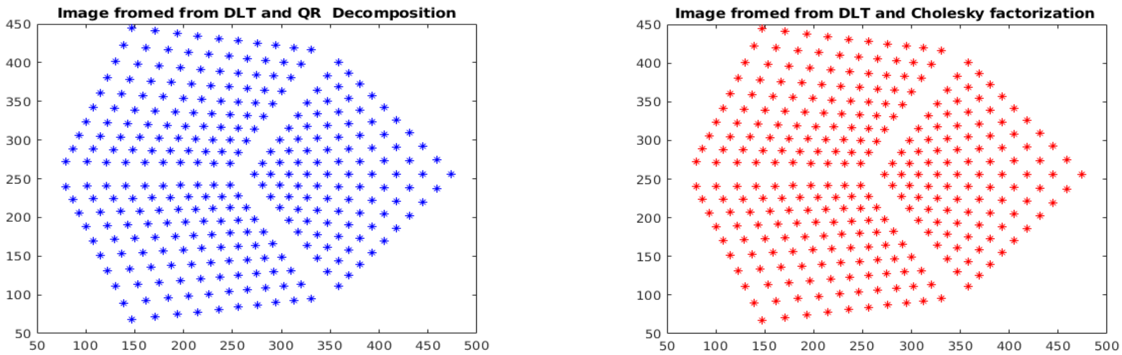
The main code implementation is done in the lab1.m and file,it contains 4 part, in this file have implemented the both the QR decomposition and Cholesky decomposition in the simulate from the data provide for the lab0.This code file uses the $make_3Dworld.m$ to simulate the 3D points,DltQR.m for QR Decomposition and DltChol.m for Cholesky decomposition.In the all the parts the parameters are take from lab0.

- Part-1 contains the code to load the data from lab0.
- Part-2 contains the code to simulate the 3D world Data.
- Part-3 contains the code to perform DLT by linear equation and QR Decmposition over the image created and world coordinates.
- Part-4 contains the code to perform DLT by Kronecker product and Cholesky factorization over the image created and world coordinates.

This file can be run by section in these combinations:

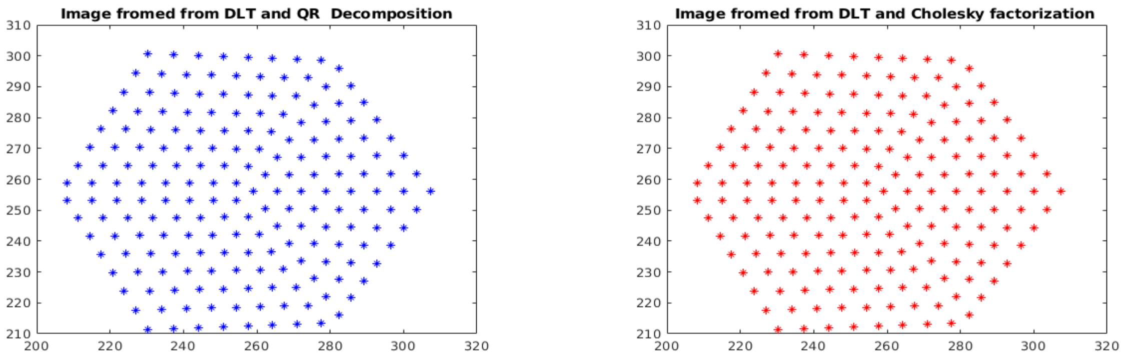
- part - 1 and part - 3 (Lab0 data and QR Decomposition)

- part - 2 and part - 4 (simulated data and Cholesky decomposition)
- part - 1 and part - 4 (lab0 data and Cholesky decomposition)
- part - 2 and part - 3 (simulated data and QR Decomposition)



(a) QR decomposition **(b)** Cholesky decomposition

Figure 1.1: camera calibration on Lab 0 data.



(a) QR decomposition **(b)** Cholesky decomposition

Figure 1.2: camera calibration on the simulated data.

```
P =

   -0.0006    0.0002    0.0009   -0.7071
   -0.0002    0.0011   -0.0002   -0.7071
    0.0000    0.0000    0.0000   -0.0028


K =

  800.0000    0.0000  256.0000
         0  800.0000  256.0000
         0         0    1.0000


R =

   -0.7071    0.0000    0.7071
   -0.4082    0.8165   -0.4082
    0.5774    0.5774    0.5774


T =

   1.0e+03 *

    1.2000
    1.2000
    1.2000
```

Figure 1.3: Parameters from the Decomposition

We can see the results on both the Decomposition the results are the same that I have used for creating the Projection matrix. with this information I create the $lab1\_gaussiansimulate.m$ file to simulate the data and add Gaussian noise with mean of 0 and standard deviation of 0.05 pixel to the simulate data and compute errors for the actual image and the noised image.

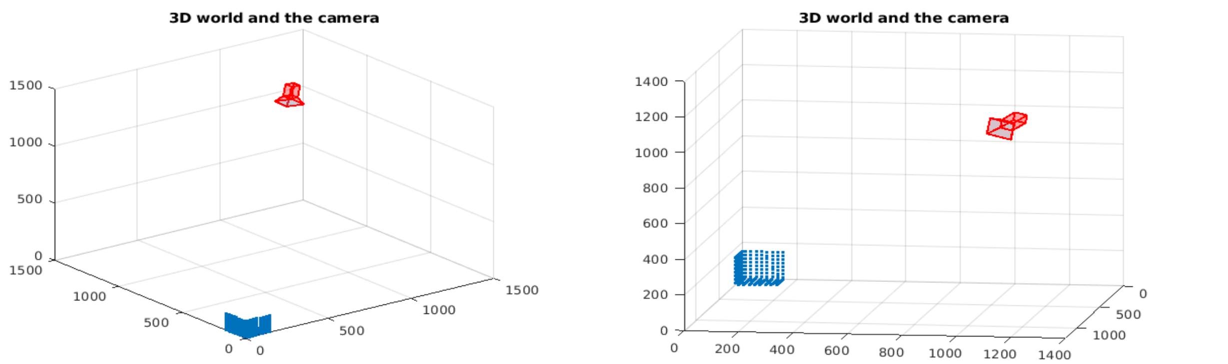


Figure 1.4: 3D World points and camera

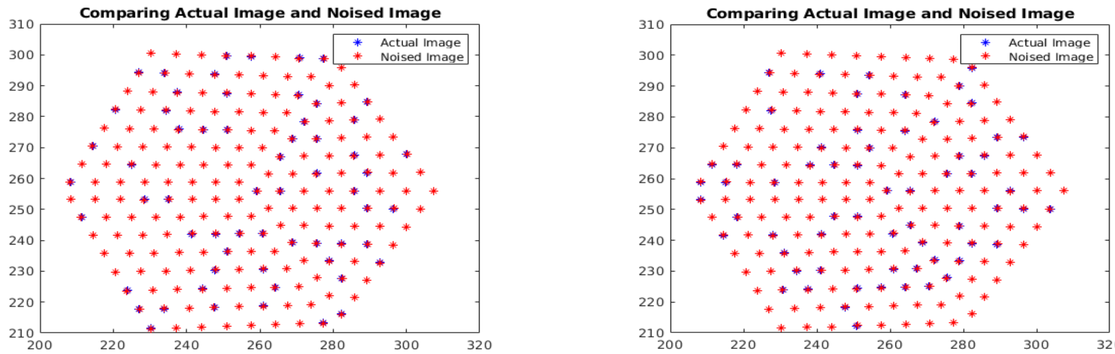


Figure 1.5: Actual and Gaussian noised images points with mean as 0 and std as 0.5

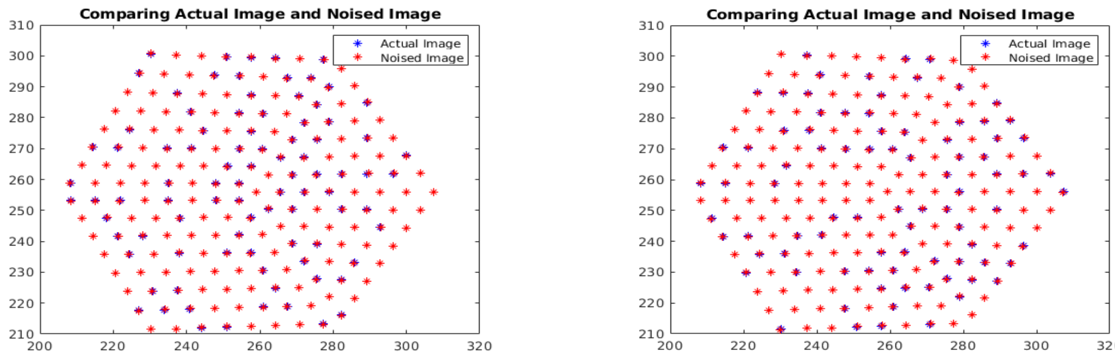


Figure 1.6: Actual and Gaussian noised images points with mean as 0 and std as 0.75

calculating the error after 15 iterations of this code i have the following error in the order of fx,fy,u,v,$T_{3,1}$,alpha,beta,gamma(all angels are in degree).

```
0.0660
0.0424
0.0302
0.1034
0.1656
0.0461
0.0784
60.0033
23.4960
-59.9943
```

(a) mean as 0 and std as 0.5

```
0.1703
0.2213
0.1781
0.0056
0.4396
0.1487
0.2267
71.9921
28.2058
-72.0095
```

(b) mean as 0 and std as 0.75

Figure 1.7: mean error after 15 iterations

```
errors = [actualfx - noisefx;
          actualfy - noisefy;
          actualu  - noiseu;
          actualv  - noisev;
          AcT      - NoT;
          actualalpha - noisealpha;
          actualbeta - noisebeta;
          actualgamma - noisegamma ];

 errors_cum = errors_cum+errors;
```

Figure 1.8: Error calculation

The error can the difference in the actual values and the real values from the noised image for this setup. The mean error is close to zero when Gaussian noise added to all the image points,except for the angles.I believe that this due the Coterminal Angles.The error increases with increase of the standard deviation.

Chapter 2

Lab 3 camera calibration Bouguet Toolbox

2.1 Bouguet Toolbox

Camera Calibration Bouguet Toolbox is matlab implementation for the calibrating the camera to extract parameters using checker borad images from the camera,developed by Jean-Yves Bouguet. This tool box can used to calibrate pin hole camera,stereo system, stereo image rectification and 3D stereo triangulation systems.

2.1.1 Calibrating images from Toolbox

calibrating with in examples in the toolbox, loading the images, and extract the grid corners for the selected images with defaults hyper parameters set in the toolbox.the toolbox hyper parameters such corner window size,size of the square along X direction and Y direction.Once the corner extraction is done we can calibrate the camera to it parameters and the estimated errors.

I choose 5 images to calibrate the camera image ([1 2 5 10 11]) for the process and did all the steps for calibration.

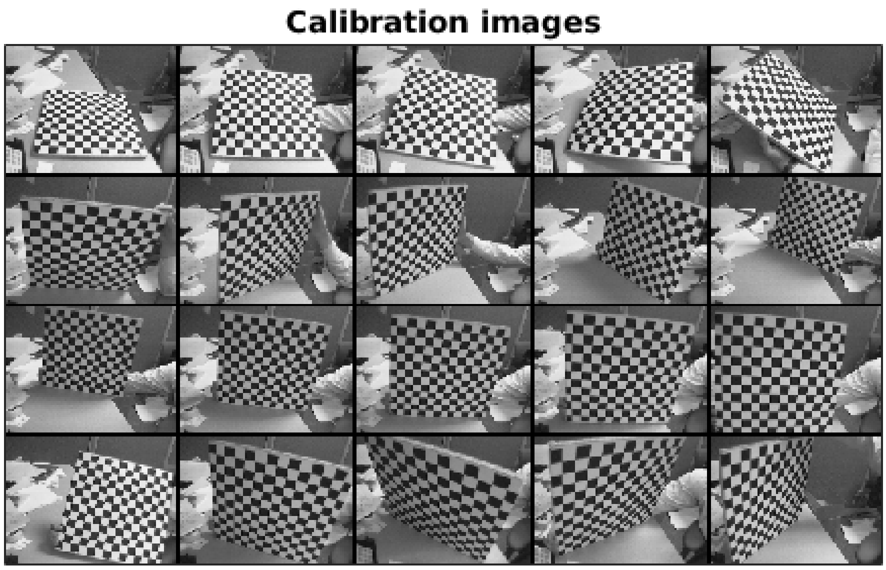
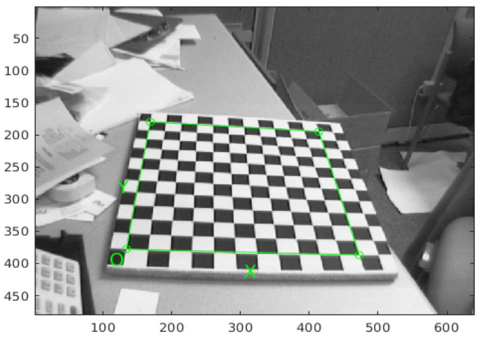
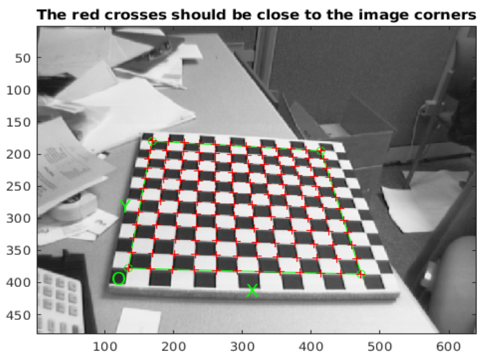


Figure 2.1: Tool box example calibration images

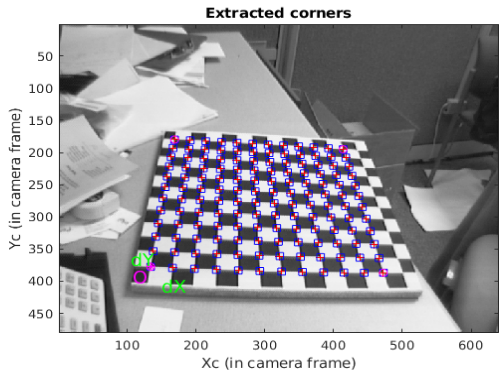


(a) Corner Selection

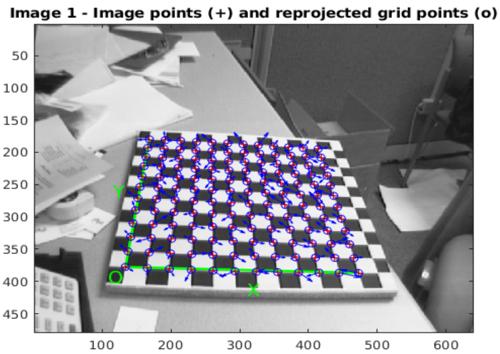


(b) Corner detection

Figure 2.2: Cornor selection and detection on image(no:5)

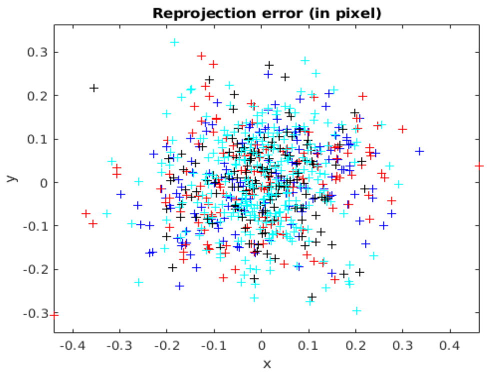


(a) corner extraction

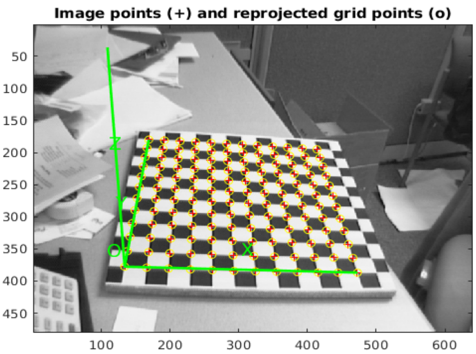


(b) Reprojection on the image points

Figure 2.3: Calibration on image(no:5)

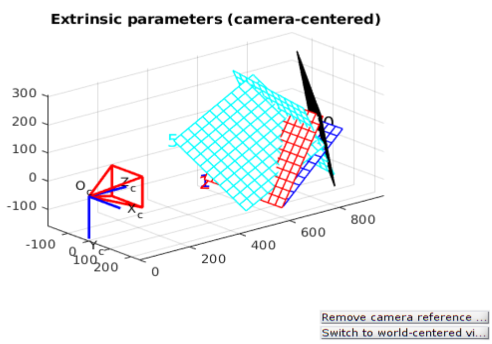


(a) Reprojection Error



(b) Reprojection on the image points

Figure 2.4: Rprojection on image(no:5)



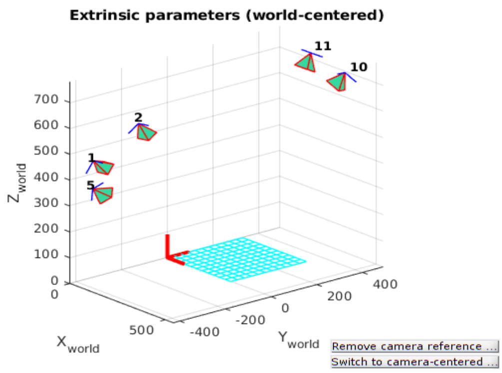


Figure 2.5: Extrinsic Parameter

```
Calibration results (with uncertainties):

Focal Length:          fc = [ 659.09968   659.48265 ] ± [ 1.08091   1.19309 ]
Principal point:       cc = [ 305.12140   247.13433 ] ± [ 2.00207   1.94383 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.26589   0.20387   0.00092   0.00014  0.00000 ] ± [ 0.00980   0.06024   0.000
Pixel error:          err = [ 0.11818   0.10329 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Figure 2.6: Calibration Results images in the tool box

I have got the results form the example images and compared to the it with the results in the web page of the toolbox and its near the actual results.

For testing the toolbox I took 9 pictures with the phone with the checker board with 30 mm in the square size.

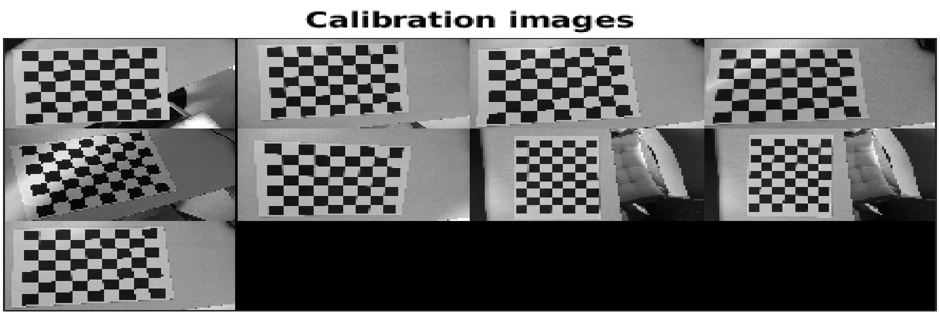
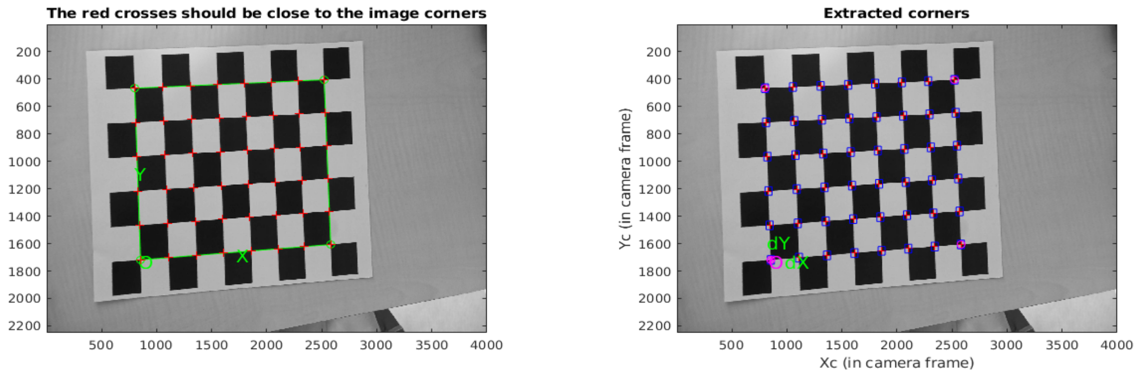


Figure 2.7: calibration images captured with my phone



(a) Corner Selection **(b)** Corner detection

Figure 2.8: corner detecting and extracting

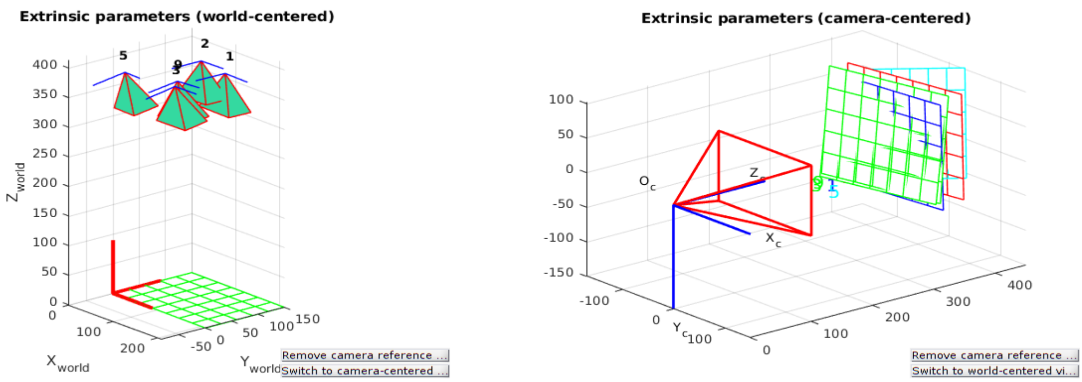
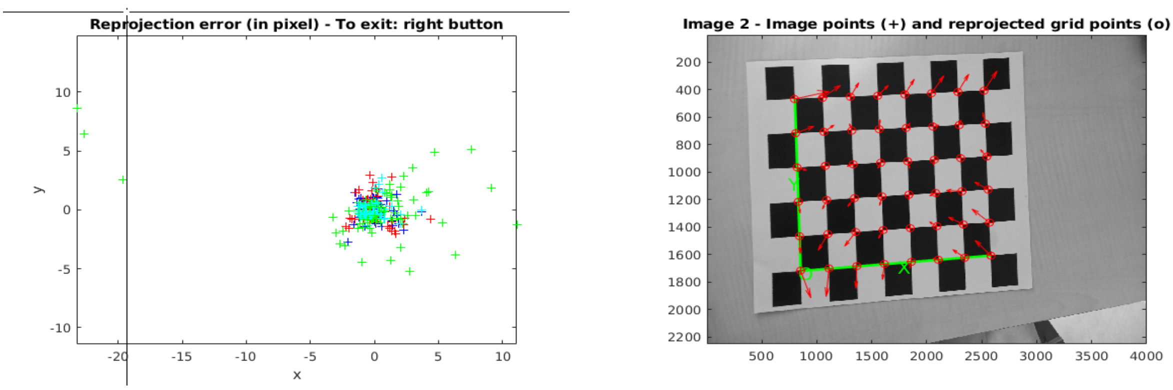


Figure 2.9: Extrinsic Parameters

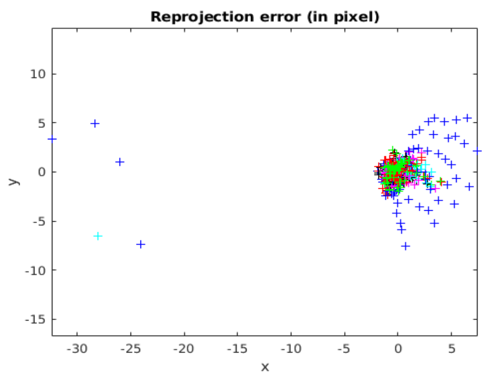


(a) Reprojection Error **(b)** Reprojection on the image points

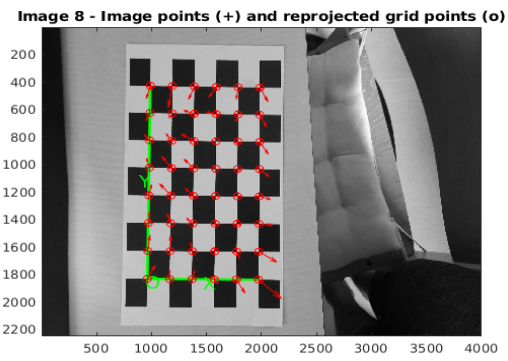
Figure 2.10: Reprojection error

```
Calibration results (with uncertainties):

Focal Length:          fc = [ 3326.09290   3327.03315 ] ± [ 342.21274   343.73006 ]
Principal point:       cc = [ 2010.19721   1051.94403 ] ± [ 74.85859   87.83258 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ 0.08626   -0.21707   -0.00112   -0.01013  0.00000 ] ± [ 0.06227   0.22373   0.0053
Pixel error:          err = [ 3.01525   1.47118 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Figure 2.11: Calibration Results with images [1 2 3 5 9]

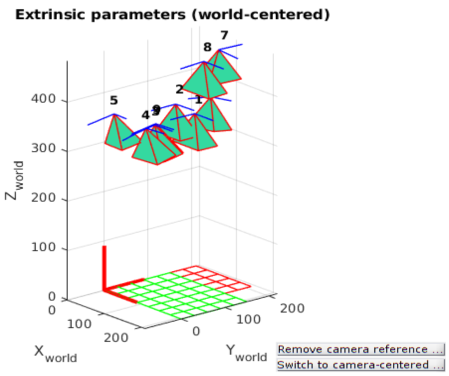


(a) Prepojection error



(b) Reprojection on the image points

Figure 2.12: Calibration with on all image



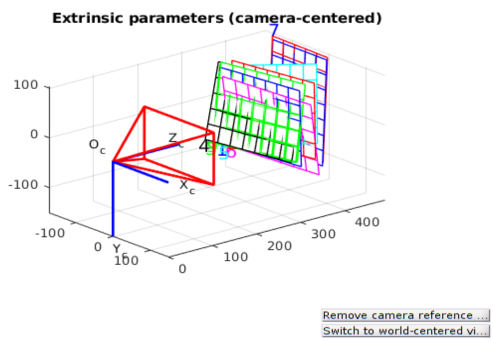


Figure 2.13: Extrinsic Parameter

```
Calibration results (with uncertainties):

Focal Length:          fc = [ 3199.11397   3213.20677 ] ± [ 174.93539   180.07480 ]
Principal point:       cc = [ 2003.61578   1060.64536 ] ± [ 55.64854   45.67771 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ 0.07608   -0.25064   -0.00352   -0.00996  0.00000 ] ± [ 0.04333   0.14351   0.00403
Pixel error:          err = [ 3.34735   1.45687 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Figure 2.14: Calibration Results with images all the images

By calibrating with all the images I got better results.I think giving too many images to calibrate is also not guaranteed to give better results.As the number of images increase we are bound to make mistakes in the image corner detection and extraction which leads to the calibration errors.

Matlab Code

Listing 2.1: DltQR.m DLT with QR Decomposition

```
function [P,K,R,T] = DltQR(image,world_coordinates)
%% DltQR to perform the extraction of camera Parameter
        %form the image coordinates with its corrosponding world coordinates

%   Input
%       image - Image coordinates of the corrosponding world points
%       world_coordinates - World coordiantes
%
%   Output
%       P - Projection matrix
%       K - intrinsic Parameters
%       R - Rotation matrix
%       T - camera 3D pose
%% Function starts here
    % Scale correction on the image.
    imag2D_X = image(1,:)./ image(3,:);
    imag2D_Y = image(2,:)./image(3,:);
    % intlize an empty varibale to store the equations to solve
    L = [];
    % looping over all the points
    for i = 1:size(image,2)
%       alphax
```

```
        alphax = [-world_coordinates(i,:),0,0,0,0,imag2D_X(1,i)*world_coordinates(i,:)];
        L = vertcat(L,alphax);
%       alphay
        alphay = [0,0,0,0,-world_coordinates(i,:),imag2D_Y(1,i)*world_coordinates(i,:)];
        L = vertcat(L,alphay);
    end
    %solving the 2n x 12 matrix to get the Projection matrix
    [~,~,V] = svd(L);
    % extracting the last columns with the least eigenvalue
    J = V(:,end);
    A = J(1:4)';
    B = J(5:8)';
    C = J(9:12)';
    %forming the Projection matrix
    P = [A;B;C];

    %extracting the 3D pose of the camera
    h = P(:,4);
    H = P(:,1:3);
    T = -inv(H)*h ;

    %extracting the Rotation and intrinsic matrix
    [Q,K] = qr(inv(H));
    K = inv (K)*[-1 0 0;0 1 0;0 0 -1];
    K = K./(K(3,3))
    R = [-1 0 0;0 1 0;0 0 -1]*Q'
end
```

Listing 2.2: DltChol.m DLT with Cholesky Decomposition

```
function [P,K,R,T] = DltChol(image,world_coordinates)
%% DltChol to perform the extraction of camera Parameter
        %form the image coordinates with its corrosponding world coordinates
```

```

%   Input
%       image - Image coordinates of the corrosponding world points
%       world_coordinates - World coordiantes
%
%   Output
%       P - Projection matrix
%       K - intrinsic Parameters
%       R - Rotation matrix
%       T - camera 3D pose
%% Function starts here
    % intlize an empty varibale to store the equations to solve
    L = [];
    for i = 1:size(image,2)
        % image coordinates in the form to perform the cross product
        mi = [0 -image(3,i) image(2,i);image(3,i) 0 -image(1,i);-image(2,i) image(1,i) 0 ];
        % world point of the corrosponding image
        Mi = world_coordinates(i,:);
        % performing the Kronecker Tensor Product
        K = kron(Mi,mi);
        L = vertcat(L,K);
    end
    %solving the 2n x 12 matrix to get the Projection matrix
    [~,~,V] = svd(L);
    % extracting the last columns with the least eigenvalue
    J = V(:,end);
    % reshape the last column to get the projection matrix
    P = reshape(J,[3,4]);
    h = P(:,4);
    H = P(:,1:3);

```

```
    %extracting the 3D pose of the camera
    T = -inv(H)*h;

    %extracting the Rotation and intrinsic matrix
    K = inv(chol(inv(H*H')));
    R = inv(K)*H;
    K = K./(K(3,3));
end
```

Listing 2.3: lab1.m camera calibration without noise

```
%% Part - 0 Instructions for running the lab

% Part - 1 contains the code to load the data from lab0
% Part - 2 contains the code to simulate the 3D world Data
% Part - 3 contains the code to perform DLT by linear
    equation
%           and QR Decmposition
%         over the image created and world coordinates
% Part - 4 contains the code to perform  DLT by Kronecker
    product
%           and Cholesky factorization
%         over the image created and world coordinates

%%%%%%%%   Run   %%%%%%%%
%      1. part - 1 and part - 3
%      2. part - 2 and part - 4
%      3. part - 1 and part - 4
%      4. part - 2 and part - 3

%% Part - 1 world points from lab 0
close all
clear
%%%loading data for the lab

%world coordinates to image
```

```
world_coordinates = load('pts3D.txt');
%intrinsic parameter of the camera
intrinsic_param = load('K.txt');
%wordl coordinate of the camera
optical_center = load('C.txt');
%3D roataion of the camera
rotation_matrix = load('R.txt');
%trainslation from the origin
translation = -rotation_matrix*optical_center;

%exterinic parameter of the camera from the 3D rotation and
    translation
exterinic_param = [rotation_matrix translation];

% Projection matrix from the intrinsic and exterinic
    parameter
P = intrinsic_param * exterinic_param

% converting the world coordinates into the homogeneous
    coordinates
world_coordinates(:,4) = 1 ;

% creating the image from the projection matrix
image = (intrinsic_param*exterinic_param*transpose(
    world_coordinates));


%% Part - 2 custom world points
clear
close all
%%%loading data for the lab

%intrinsic parameter of the camera
intrinsic_param = load('K.txt');
%wordl coordinate of the camera
```

```
optical_center = load('C.txt');
%3D roataion of the camera
rotation_matrix = load('R.txt');
%trainslation from the origin
translation = -rotation_matrix*optical_center;

%exterinic parameter of the camera from the 3D rotation and
    translation
exterinic_param = [rotation_matrix translation];

% creating Homogeneous world coordinates
world_coordinates = make_3Dworld(150,150,150);

% Projection matrix from the intrinsic and exterinic
    parameter
P = intrinsic_param * exterinic_param

% creating the image from the projection matrix
image = (intrinsic_param*exterinic_param*transpose(
    world_coordinates));
%% part - 3  DLT with the QR Decomposition

% solving to get the paramters of camera from
% QR decomposition and DLT equation
[P,K,R,T] = DltQR(image,world_coordinates)

%forming the image from the results of DLT
image = P * (world_coordinates');

% Scale correction on the image to plot
imag2D_X1 = image(1,:)./ image(3,:);
imag2D_Y1 = image(2,:)./image(3,:);

% ploting the image
figure
```

```
plot(imag2D_Y1,imag2D_X1,'*b')
title('Image fromed from DLT and QR  Decomposition')

% extracting intrinsic parameters
fx = K(1,1)
fy = K(2,2)
u  = K(1,3)
v  = K(2,3)

% extracting the euler angles from the rotation matrix
alpha = atan2d(R(2,1),R(1,1))
beta = atan2d(-R(3,1),sqrt(R(3,2)^2+R(3,3)^2))
gamma = atan2d(R(3,2),R(3,3))
%% Part - 4 DLT with Kronecker product and Cholesky factorization

% solving to get the paramters of camera from
% Cholesky factorization and DLT (Kronecker product)
[P,K,R,T] = DltChol(image,world_coordinates)

%forming the image from the results of DLT
image = P * (world_coordinates');

% Scale correction on the image to plot
imag2D_X2 = image(1,:)./ image(3,:);
imag2D_Y2 = image(2,:)./image(3,:);
% ploting the image
figure
plot(imag2D_Y2,imag2D_X2,'*r')
title('Image fromed from DLT and Cholesky factorization ')

% extracting intrinsic parameters
fx = K(1,1)
fy = K(2,2)
u  = K(1,3)
```

```
v  = K(2,3)

% extracting the euler angles from the rotation matrix
alpha = atan2d(R(2,1),R(1,1))
beta = atan2d(-R(3,1),sqrt(R(3,2)^2+R(3,3)^2))
gamma = atan2d(R(3,2),R(3,3))
```

Listing 2.4: *make_3Dworld.m* Create 3D world for simulation

```
%% Part - 0 Instructions for running the lab

% Part - 1 contains the code to load the data from lab0
% Part - 2 contains the code to simulate the 3D world Data
% Part - 3 contains the code to perform DLT by linear
   equation
%            and QR Decmposition
%          over the image created and world coordinates
% Part - 4 contains the code to perform  DLT by Kronecker
   product
%            and Cholesky factorization
%          over the image created and world coordinates

%%%%%%%%  Run  %%%%%%%%
%      1. part - 1 and part - 3
%      2. part - 2 and part - 4
%      3. part - 1 and part - 4
%      4. part - 2 and part - 3

%% Part - 1 world points from lab 0
close all
clear
%%%loading data for the lab

%world coordinates to image
world_coordinates = load('pts3D.txt');
%intrinsic parameter of the camera
```

```
intrinsic_param = load('K.txt');
%wordl coordinate of the camera
optical_center = load('C.txt');
%3D roataion of the camera
rotation_matrix = load('R.txt');
%trainslation from the origin
translation = -rotation_matrix*optical_center;

%exterinic parameter of the camera from the 3D rotation and translation
exterinic_param = [rotation_matrix translation];

% Projection matrix from the intrinsic and exterinic parameter
P = intrinsic_param * exterinic_param

% converting the world coordinates into the homogeneous coordinates
world_coordinates(:,4) = 1 ;

% creating the image from the projection matrix
image = (intrinsic_param*exterinic_param*transpose(world_coordinates));


%% Part - 2 custom world points
clear
close all
%%%loading data for the lab

%intrinsic parameter of the camera
intrinsic_param = load('K.txt');
%wordl coordinate of the camera
optical_center = load('C.txt');
%3D roataion of the camera
```

```
rotation_matrix = load('R.txt');
%trainslation from the origin
translation = -rotation_matrix*optical_center;

%exterinic parameter of the camera from the 3D rotation and
    translation
exterinic_param = [rotation_matrix translation];

% creating Homogeneous world coordinates
world_coordinates = make_3Dworld(150,150,150);

% Projection matrix from the intrinsic and exterinic
    parameter
P = intrinsic_param * exterinic_param

% creating the image from the projection matrix
image = (intrinsic_param*exterinic_param*transpose(
    world_coordinates));
%% part - 3  DLT with the QR Decomposition

% solving to get the paramters of camera from
% QR decomposition and DLT equation
[P,K,R,T] = DltQR(image,world_coordinates)

%forming the image from the results of DLT
image = P * (world_coordinates');

% Scale correction on the image to plot
imag2D_X1 = image(1,:)./ image(3,:);
imag2D_Y1 = image(2,:)./image(3,:);

% ploting the image
figure
plot(imag2D_Y1,imag2D_X1,'*b')
title('Image fromed from DLT and QR  Decomposition')
```

```
% extracting intrinsic parameters
fx = K(1,1)
fy = K(2,2)
u  = K(1,3)
v  = K(2,3)

% extracting the euler angles from the rotation matrix
alpha = atan2d(R(2,1),R(1,1))
beta = atan2d(-R(3,1),sqrt(R(3,2)^2+R(3,3)^2))
gamma = atan2d(R(3,2),R(3,3))
%% Part - 4 DLT with Kronecker product and Cholesky
   factorization

% solving to get the paramters of camera from
% Cholesky factorization and DLT (Kronecker product)
[P,K,R,T] = DltChol(image,world_coordinates)

%forming the image from the results of DLT
image = P * (world_coordinates');

% Scale correction on the image to plot
imag2D_X2 = image(1,:)./ image(3,:);
imag2D_Y2 = image(2,:)./image(3,:);
% ploting the image
figure
plot(imag2D_Y2,imag2D_X2,'*r')
title('Image fromed from DLT and Cholesky factorization ')

% extracting intrinsic parameters
fx = K(1,1)
fy = K(2,2)
u  = K(1,3)
v  = K(2,3)

```

```
% extracting the euler angles from the rotation matrix
alpha = atan2d(R(2,1),R(1,1))
beta = atan2d(-R(3,1),sqrt(R(3,2)^2+R(3,3)^2))
gamma = atan2d(R(3,2),R(3,3))
```

Listing 2.5: *lab1_gaussiansimulate.m* simulate 3d world with noise

```
clc
clear all
close all

%%%loading data for the lab
%intrinsic parameter of the camera
intrinsic_param = load('K.txt');
%wordl coordinate of the camera
optical_center = load('C.txt');
%3D roataion of the camera
rotation_matrix = load('R.txt');
%creating the 3D world for simulation
world_coordinates = make_3Dworld(150,150,150);

%trainslation from the origin
translation = -rotation_matrix*optical_center;

%exterinic parameter of the camera from the 3D rotation and
    translation
exterinic_param = [rotation_matrix translation];

% Projection matrix from the intrinsic and exterinic
    parameter
P = intrinsic_param * exterinic_param;

%ploting the world coordinates with plot3d
plot3(world_coordinates(:,1),world_coordinates(:,2),
    world_coordinates(:,3),".")
hold on
```

```
grid on
title('3D world and the camera')

%plot the camera to visulize the in the 3d World
absPose = rigid3d(rotation_matrix,optical_center');
plotCamera('AbsolutePose',absPose,'Size',50)

actualImage = (intrinsic_param*exterinic_param*transpose(
   world_coordinates));
%%
% scaled image coordinates
imag2D_X = actualImage(1,:)./ actualImage(3,:);
imag2D_Y = actualImage(2,:)./actualImage(3,:);

errors_cum = zeros(10,1);

for i = 1:15

    %creating the gaussian noise
    gaussian = normrnd(0,0.75,size(actualImage));
    %noise image
    noisedImage = actualImage+gaussian;

    imag2D_X1 = noisedImage(1,:)./noisedImage(3,:);
    imag2D_Y1 = noisedImage(2,:)./noisedImage(3,:);

    %ploting the image and comparing the actual and noised
    figure
    plot(imag2D_Y,imag2D_X,'b*')
    hold on
    plot(imag2D_Y1,imag2D_X1,'r*')
    title('Comparing Actual Image and Noised Image')
    legend('Actual Image','Noised Image')


```

```
    %sloving to get the parametes from actual image with the Cholesky decomposition
    [AcP,AcK,AcR,AcT] = DltChol(actualImage,world_coordinates);

    %sloving to get the parametes from Noised image with the Cholesky decomposition
    [NoP,NoK,NoR,NoT] = DltChol(noisedImage,world_coordinates);



    % extracting intrinsic parameters
    actualfx = AcK(1,1);
    actualfy = AcK(2,2);
    actualu  = AcK(1,3);
    actualv  = AcK(2,3);

    % extracting the euler angles from the rotation matrix
    actualalpha = atan2d(AcR(2,1),AcR(1,1));
    actualbeta = atan2d(-AcR(3,1),sqrt(AcR(3,2)^2+AcR(3,3)^2));
    actualgamma = atan2d(AcR(3,2),AcR(3,3));



    % extracting intrinsic parameters
    noisefx = NoK(1,1);
    noisefy = NoK(2,2);
    noiseu  = NoK(1,3);
    noisev  = NoK(2,3);

    % extracting the euler angles from the rotation matrix
    noisealpha = atan2d(NoR(2,1),NoR(1,1));
    noisebeta = atan2d(-NoR(3,1),sqrt(NoR(3,2)^2+NoR(3,3)^2))
```

```
        ;
    noisegamma = atan2d(NoR(3,2),NoR(3,3));

    errors = [actualfx - noisefx;
             actualfy - noisefy;
             actualu  - noiseu;
             actualv  - noisev;
             AcT      - NoT;
             actualalpha - noisealpha;
             actualbeta - noisebeta;
             actualgamma - noisegamma ];

     errors_cum = errors_cum+errors;
end
disp('average error after 15 iterations')
% calculating the mean error
errors_cum/15
```