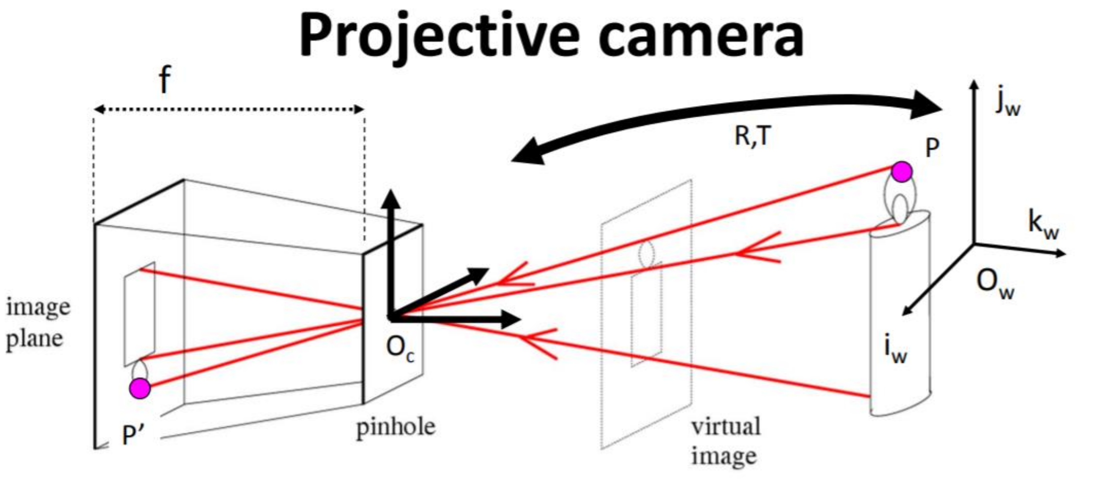
EE417 ASSIGNMENT II

CAMERA CALIBRATION

1. Camera Calibration by SVD



$$P' = M\,P_w$$
$$= K\left[R \quad T\right]P_w$$

Internal (intrinsic) parameters

External (extrinsic) parameters

f = focal length

u_o, v_o = offset

$\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ $\rightarrow$ non-square pixels

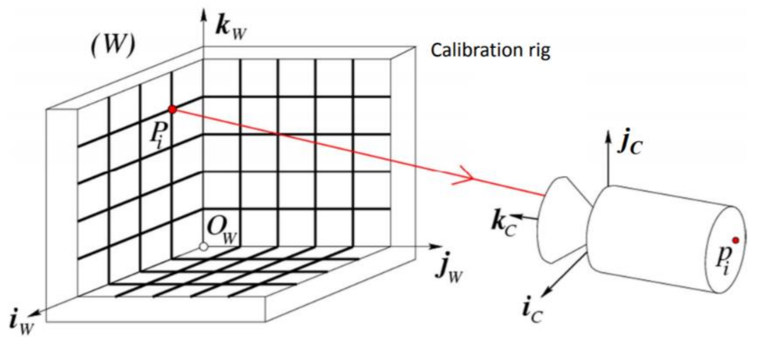
θ = skew angle

R,T = rotation, translation

$$P' = M\,P_w \quad = K\left[R \quad T\right]P_w$$

$$\boxed{\mathcal{M} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha\cot\theta\boldsymbol{r}_2^T + u_0\boldsymbol{r}_3^T & \alpha t_x - \alpha\cot\theta t_y + u_0 t_z \\ \dfrac{\beta}{\sin\theta}\boldsymbol{r}_2^T + v_0\boldsymbol{r}_3^T & \dfrac{\beta}{\sin\theta}t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix}_{3\times 4}}$$

$$K = \begin{bmatrix} \boldsymbol{\alpha} & -\boldsymbol{\alpha}\cot\boldsymbol{\theta} & u_o \\ 0 & \dfrac{\boldsymbol{\beta}}{\sin\boldsymbol{\theta}} & v_o \\ 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} \mathbf{r}_1^T \\ \mathbf{r}_2^T \\ \mathbf{r}_3^T \end{bmatrix} \qquad T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$



- $P_1 \dots P_n$ with known positions in $[O_w, i_w, j_w, k_w]$
- $p_1, \dots p_n$ known positions in the image

Goal: compute intrinsic and extrinsic parameters

$$\begin{cases} -u_1(\mathbf{m}_3\, P_1) + \mathbf{m}_1\, P_1 = 0 \\ -v_1(\mathbf{m}_3\, P_1) + \mathbf{m}_2\, P_1 = 0 \\ \vdots \\ -u_n(\mathbf{m}_3\, P_n) + \mathbf{m}_1\, P_n = 0 \\ -v_n(\mathbf{m}_3\, P_n) + \mathbf{m}_2\, P_n = 0 \end{cases} \longrightarrow \boxed{\underset{\text{known}}{\mathcal{P}}\,\underset{\text{unknown}}{\boldsymbol{m}} = 0}$$

Homogenous linear system

$$\mathcal{P} \overset{\text{def}}{=} \begin{pmatrix} \boldsymbol{P}_1^T & \mathbf{0}^T & -u_1 \overset{1\times4}{\boldsymbol{P}_1^T} \\ \mathbf{0}^T & \boldsymbol{P}_1^T & -v_1\boldsymbol{P}_1^T \\ \dots & \dots & \dots \\ \boldsymbol{P}_n^T & \mathbf{0}^T & -u_n\boldsymbol{P}_n^T \\ \mathbf{0}^T & \boldsymbol{P}_n^T & -v_n\boldsymbol{P}_n^T \end{pmatrix}_{2n\times 12} \qquad \boldsymbol{m} \overset{\text{def}}{=} \begin{pmatrix} \overset{4\times1}{\mathbf{m}_1^T} \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{pmatrix}_{12\times1}$$

$$\mathcal{P}\boldsymbol{m} = 0$$ Compute SVD decomposition of P

$$\downarrow$$

$$\boxed{\mathrm{U}_{2n\times12}\ \mathbf{D}_{12\times12}\,\mathrm{V}^{\mathrm{T}}{}_{12\times12}}$$

Last column of V gives $\boldsymbol{m}$

$$\downarrow$$

$$M \qquad \mathrm{M\,P_i} \rightarrow \mathrm{p_i}$$

$$\frac{\mathcal{M}}{\rho} = \left(\underbrace{\begin{matrix} \alpha \boldsymbol{r}_1^T - \alpha\cot\theta\,\boldsymbol{r}_2^T + u_0\boldsymbol{r}_3^T \\ \dfrac{\beta}{\sin\theta}\boldsymbol{r}_2^T + v_0\boldsymbol{r}_3^T \\ \boldsymbol{r}_3^T \end{matrix}}_{A} \;\; \underbrace{\begin{matrix} \alpha t_x - \alpha\cot\theta\, t_y + u_0 t_z \\ \dfrac{\beta}{\sin\theta} t_y + v_0 t_z \\ t_z \end{matrix}}_{\mathbf{b}} \right) = \mathrm{K}[\mathrm{R} \quad \mathrm{T}]$$

$$\mathrm{K} = \begin{bmatrix} \boldsymbol{\alpha} & -\boldsymbol{\alpha}\cot\boldsymbol{\theta} & \mathrm{u_o} \\ 0 & \dfrac{\boldsymbol{\beta}}{\sin\boldsymbol{\theta}} & \mathrm{v_o} \\ 0 & 0 & 1 \end{bmatrix}$$

Intrinsic

$$\boldsymbol{\alpha} = \boldsymbol{\rho}^2 |\mathbf{a}_1 \times \mathbf{a}_3| \sin\boldsymbol{\theta}$$
$$\boldsymbol{\beta} = \boldsymbol{\rho}^2 |\mathbf{a}_2 \times \mathbf{a}_3| \sin\boldsymbol{\theta}$$
$$\longrightarrow \mathrm{f}$$

$$\mathrm{A} = \begin{bmatrix} \mathbf{a}_1^{\mathrm{T}} \\ \mathbf{a}_2^{\mathrm{T}} \\ \mathbf{a}_3^{\mathrm{T}} \end{bmatrix} \qquad \mathrm{b} = \begin{bmatrix} \mathrm{b}_1 \\ \mathrm{b}_2 \\ \mathrm{b}_3 \end{bmatrix}$$

Estimated values

Intrinsic

$$\boldsymbol{\rho} = \frac{\pm 1}{|\mathbf{a}_3|} \qquad \mathrm{u_o} = \boldsymbol{\rho}^2(\mathbf{a}_1 \cdot \mathbf{a}_3) \qquad \mathrm{v_o} = \boldsymbol{\rho}^2(\mathbf{a}_2 \cdot \mathbf{a}_3)$$

$$\cos\theta = \frac{(\mathbf{a}_1 \times \mathbf{a}_3)\cdot(\mathbf{a}_2 \times \mathbf{a}_3)}{|\mathbf{a}_1 \times \mathbf{a}_3| \cdot |\mathbf{a}_2 \times \mathbf{a}_3|}$$

Extrinsic

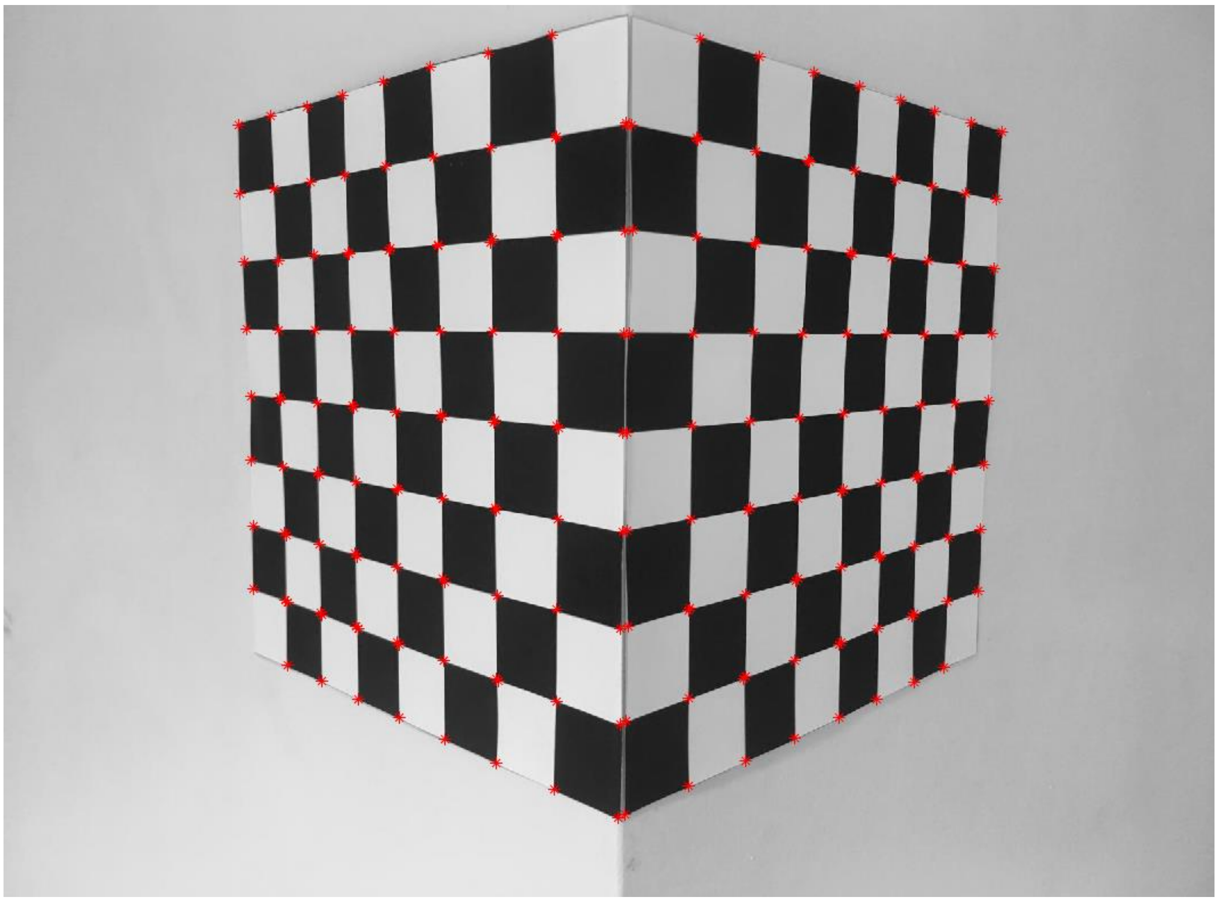
$$\mathbf{r}_1 = \frac{(\mathbf{a}_2 \times \mathbf{a}_3)}{|\mathbf{a}_2 \times \mathbf{a}_3|} \qquad \mathbf{r}_3 = \frac{\pm 1}{|\mathbf{a}_3|}$$

$$\mathbf{r}_2 = \mathbf{r}_3 \times \mathbf{r}_1 \qquad \mathrm{T} = \boldsymbol{\rho}\,\mathrm{K}^{-1}\mathbf{b}$$

The equations above are summarizing the steps of camera calibration method. We will try to implement these equations in our assignment.

We will use a two checkerboards that is post on the wall as a calibration object.

Then, we used Harris method to detect the corners in the image as can be seen below:



Now, we need to to match some correspondence points in real worlds by creating a coordinate system and in camera pixels.

This was a hard task because we entered each corners’ pixels and real world coordinates manually We used 91 corners and created the following correspondence matrix (x pixel, ypixel, xrealworld, yrealworld, zrealworld):

```
% pix, piy, x,y,z
corresp = [ 827 172  0 0 0; 913 188  1 0 0; 991 206  2 0 0; 1060 218 3 0 0; 1132 232 4 0 0; 1175 244 5 0 0;
1223 253 6 0 0;
            829 309  0 0 1; 913 319  1 0 1; 991 328  2 0 1; 1059 334 3 0 1; 1116 339 4 0 1; 1170 345 5 0 1;
1217 350 6 0 1;
            824 446  0 0 2; 988 445  1 0 2; 1052 446 2 0 2; 1111 446 3 0 2; 1163 446 4 0 2; 1213 447 5 0 2;
1258 447 6 0 2;
            827 576  0 0 3; 907 568  1 0 3; 983 562  2 0 3; 1048 557 3 0 3; 1107 551 4 0 3; 1159 546 5 0 3;
1254 537 6 0 3;
            821 706  0 0 4; 905 691  1 0 4; 983 675  2 0 4; 1047 664 3 0 4; 1106 651 4 0 4; 1156 643 5 0 4;
1203 635 6 0 4;
            823 832  0 0 5; 904 808  1 0 5; 978 789  2 0 5; 1046 769 3 0 5; 1103 754 4 0 5; 1153 740 5 0 5;
1200 726 6 0 5;
            820 956  0 0 6; 902 926  1 0 6; 978 898  2 0 6; 1042 876 3 0 6; 1103 852 4 0 6; 1151 835 5 0 6;
1199 815 6 0 6;
            727 188 0 1 0; 642 201 0 2 0; 504 225 0 3 0; 450 238 0 4 0; 405 246 0 5 0; 356 355 0 6 0;
            728 316 0 1 1; 644 322 0 2 1; 572 329 0 3 1; 510 334 0 4 1; 454 339 0 5 1; 409 343 0 6 1;
            730 445 0 1 2; 644 443 0 2 2; 576 443 0 3 2; 514 443 0 4 2; 458 442 0 5 2; 410 441 0 6 2;
            731 571 0 1 3; 648 560 0 2 3; 576 552 0 3 3; 517 549 0 4 3; 460 540 0 5 3; 413 537 0 6 3;
            730 691 0 1 4; 648 676 0 2 4; 578 663 0 3 4; 520 652 0 4 4; 464 641 0 5 4; 416 632 0 6 4;
            728 809 0 1 5; 650 791 0 2 5; 581 773 0 3 5; 519 753 0 4 5; 463 736 0 5 5; 417 724 0 6 5;
            728 930 0 1 5; 649 900 0 2 5; 580 876 0 3 5; 521 856 0 4 5; 467 835 0 5 5; 418 813 0 6 5;
            ];
```

But these coordinates were needed to be converted into real world coordinates, we measured our checkerboard and observed that each square is 18mm. Therefore, we multiplied each realworld coordinates by 18.

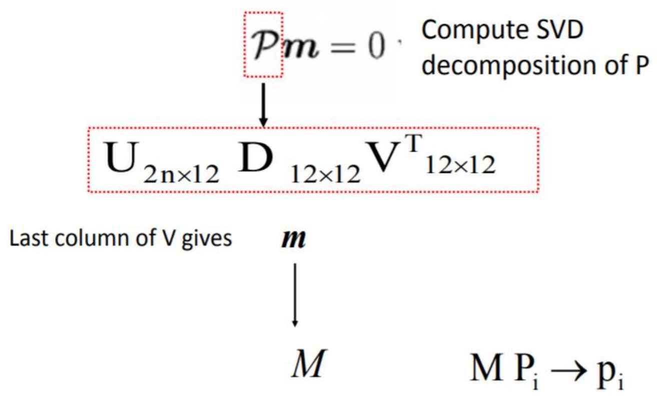
```
N = 91; % Number of corners that are used to calibrate our
camera
L = 18; % Length of each square in mm
Pw = []; % Real world coordinates
Pc = []; % Camera coordinates in pixels
for j = 1:1:N
   Pc = [Pc; corresp(j,1) corresp(j,2) 1];
   Pw = [Pw; corresp(j,3)*L corresp(j,4)*L corresp(j,5)*L
1];
End
```

Ax=0 form

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ & & & & & & \vdots & & & & & \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

After that we constructed our P matrix

```
for i = 1:1:N
P = [   P;
        Pw(i,:)   zeros(1,4)  -Pc(i,1)*Pw(i,:);
        zeros(1,4)   Pw(i,:)  -Pc(i,2)*Pw(i,:);
    ];
end
```


$\mathcal{P}m = 0$
Compute SVD decomposition of P
$U_{2n\times12}\, D_{12\times12}\, V^T_{12\times12}$
Last column of V gives m
M
$M P_i \rightarrow p_i$

Now we can use SVD to get the m matrix that will be used to find the intrinsic and extrinsic parameters.

```
[U S V] = svd(P);
% Last column of V gives m
m = V(:,end);
m = transpose(reshape(m,[],3));
```

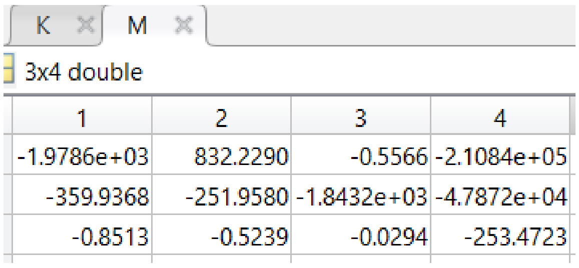
Rest of the codes are the implementation of the other equations to get the intrinsic and extrinsics.

(For this part I used this resource: http://www.sci.utah.edu/~gerig/CS6320-S2013/Materials/hw1_abhishek_projectreport_b.pdf)

```
% Normalizing of M to make the norm of third rotation
vector unity
norm_31 = norm(m(3,1:3));
M = m / norm_31;

a1 = M(1,1:3);
a2 = M(2,1:3);
a3 = M(3,1:3);
b = M(1:3, 4);
r3 = a3;

% Computation of the intrinsic parameters
u_0 = transpose(a1*transpose(a3));
v_0 = transpose(a2*transpose(a3));
cross_a1a3 = cross(a1,a3);
cross_a2a3 = cross(a2,a3);
theta = acos (-
1*cross_a1a3*transpose(cross_a2a3)/(norm(cross_a1a3)*norm(
cross_a2a3)));
alpha = norm(cross_a1a3) * sin(theta);
beta = norm(cross_a2a3) * sin(theta);

% Computation of the extrinsic parameters
r1 = cross_a2a3/norm(cross_a2a3);
r2 = cross(r3, r1);
K = [alpha -1*alpha*cot(theta) u_0
     0 beta/sin(theta) v_0
     0  0  1];
t = inv(K) * b; % Translation vector

% Rotation matrix
R(1,1:3) = r1;
R(2,1:3) = r2;
R(3,1:3) = r3;
```

RESULTS:

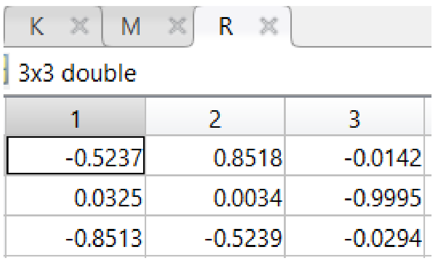
M Matrix:


K M

3x4 double

1	2	3	4
-1.9786e+03	832.2290	-0.5566	-2.1084e+05
-359.9368	-251.9580	-1.8432e+03	-4.7872e+04
-0.8513	-0.5239	-0.0294	-253.4723

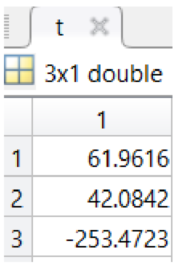
Rotation Matrix:


K M R

3x3 double

1	2	3
-0.5237	0.8518	-0.0142
0.0325	0.0034	-0.9995
-0.8513	-0.5239	-0.0294

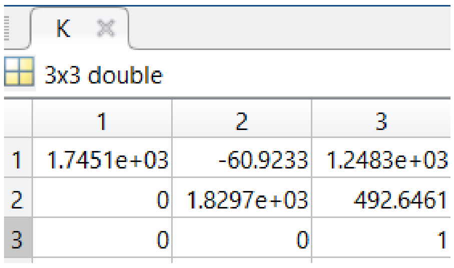
T translation Matrix:


t

3x1 double

	1
1	61.9616
2	42.0842
3	-253.4723

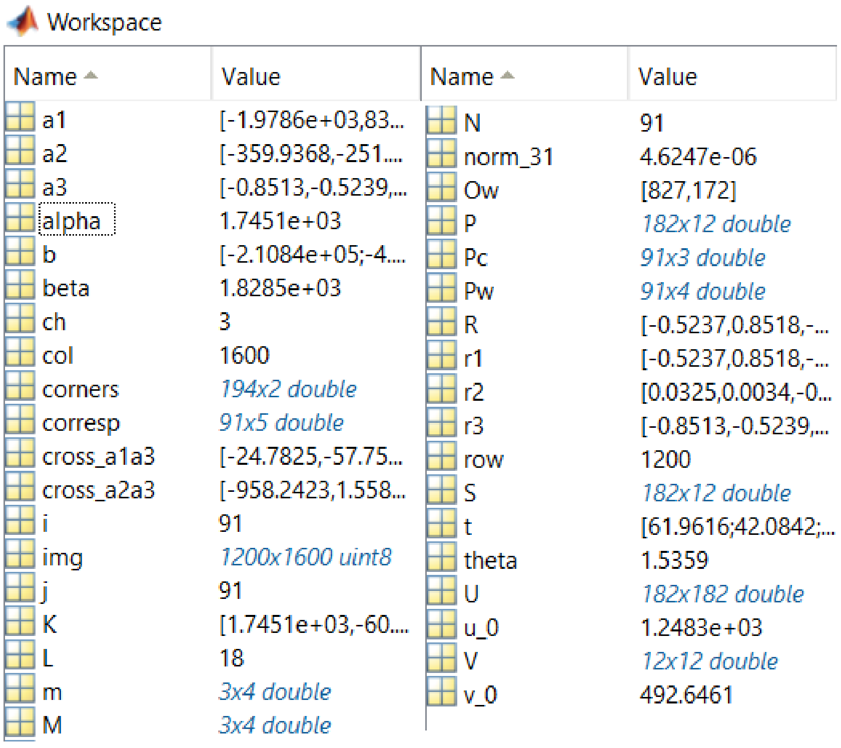
K camera matrix:


K

3x3 double

	1	2	3
1	1.7451e+03	-60.9233	1.2483e+03
2	0	1.8297e+03	492.6461
3	0	0	1

Other Parameters:

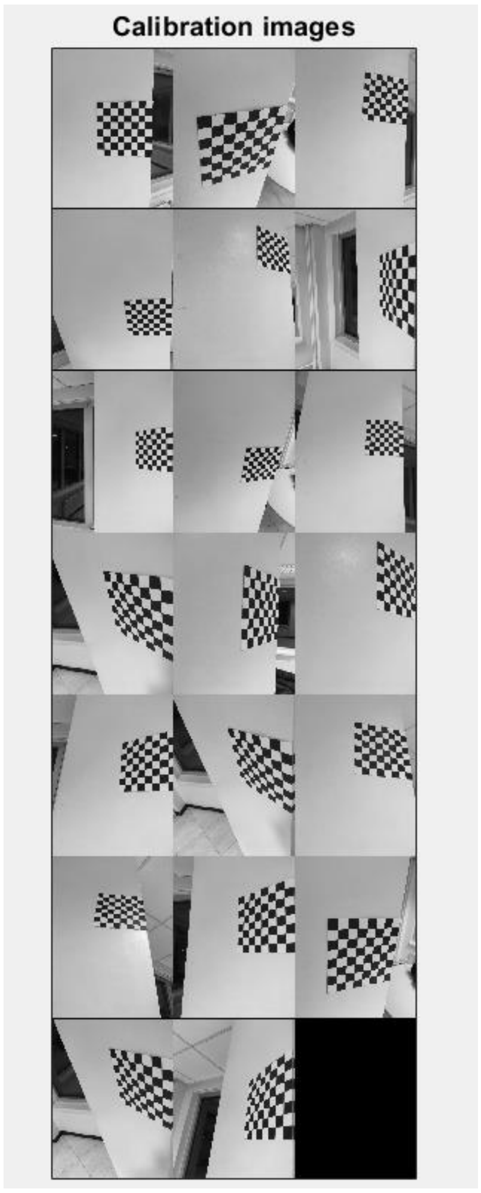

Workspace

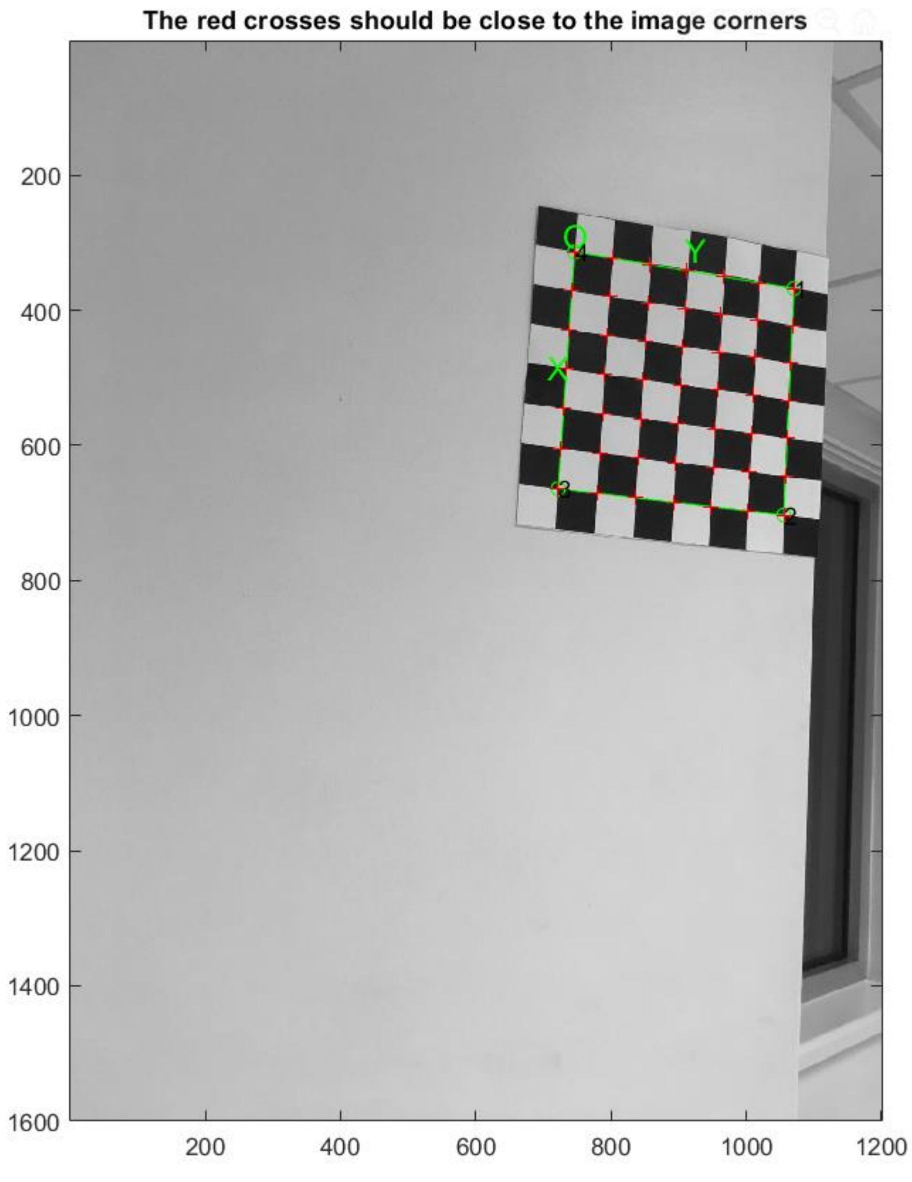
Name	Value	Name	Value
a1	[-1.9786e+03,83...	N	91
a2	[-359.9368,-251....	norm_31	4.6247e-06
a3	[-0.8513,-0.5239,...	Ow	[827,172]
alpha	1.7451e+03	P	*182x12 double*
b	[-2.1084e+05;-4....	Pc	*91x3 double*
beta	1.8285e+03	Pw	*91x4 double*
ch	3	R	[-0.5237,0.8518,-...
col	1600	r1	[-0.5237,0.8518,-...
corners	*194x2 double*	r2	[0.0325,0.0034,-0...
corresp	*91x5 double*	r3	[-0.8513,-0.5239,...
cross_a1a3	[-24.7825,-57.75...	row	1200
cross_a2a3	[-958.2423,1.558...	S	*182x12 double*
i	91	t	[61.9616;42.0842;...
img	*1200x1600 uint8*	theta	1.5359
j	91	U	*182x182 double*
K	[1.7451e+03,-60....	u_0	1.2483e+03
L	18	V	*12x12 double*
m	*3x4 double*	v_0	492.6461
M	*3x4 double*		

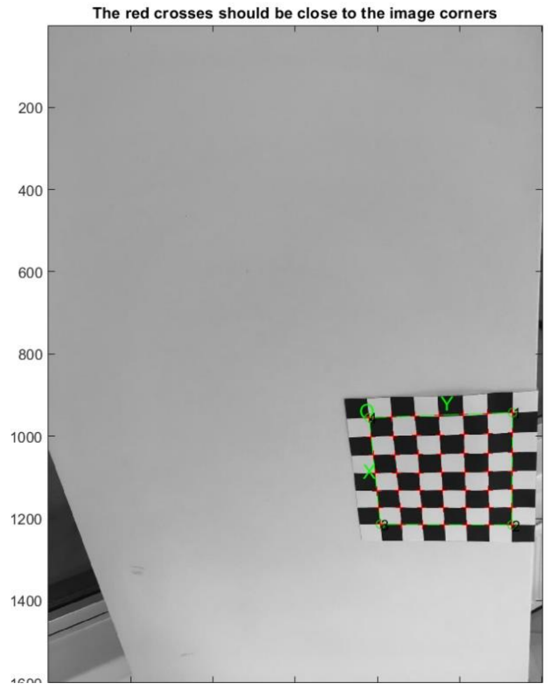
2. Camera Calibration by MATLAB Toolbax

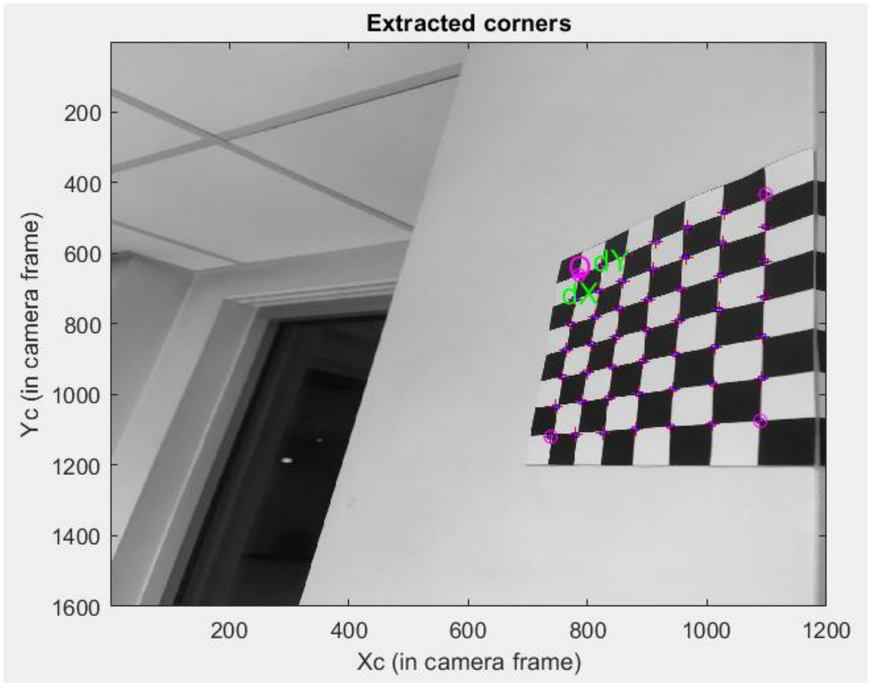
In this section of the lab, we will use a MATLAB toolbox to calibrate our camera. I took 20 photos of a checkerboard from different angles and load them into the toolbox. Then, we manually plotted the coordinate plane and toolbox automatically detected the corners in the each image. Actually, we applied this steps on all of 20 images.

Some results can be seen below:









Then we used these corners to initialize and calibrate our camera.

After first calibration we got the following results:

Calibration parameters after initialization:

Focal Length: fc = [1379.95263   1379.95263]

Principal point: cc = [599.50000   799.50000]

Skew: alpha_c = [0.00000]   => angle of pixel = 90.00000 degrees

Distortion: kc = [0.00000   0.00000   0.00000   0.00000   0.00000]

Calibration results after optimization (with uncertainties):

Focal Length: fc = [1378.79002   1377.11766] +/- [11.23310   10.08762]

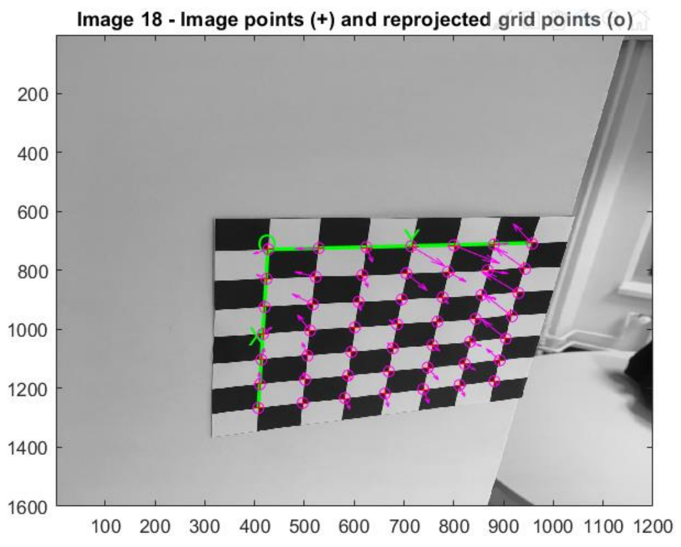
Principal point: cc = [617.34454   802.33128] +/- [19.25324   22.77023]

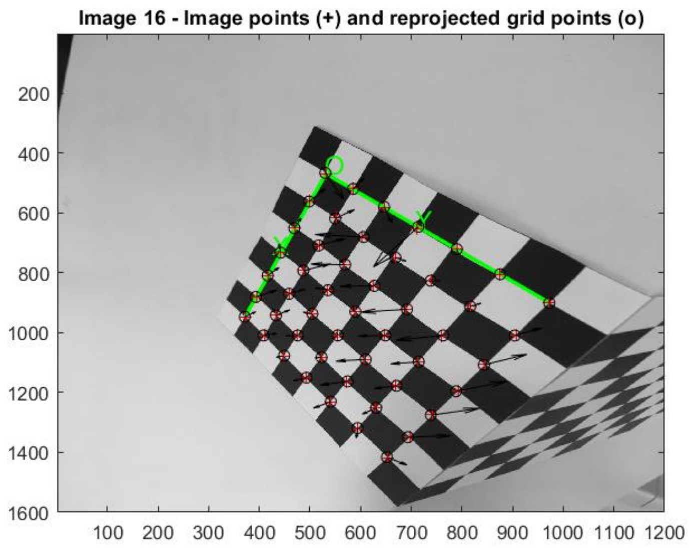
Skew: alpha_c = [0.00000] +/- [0.00000]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees

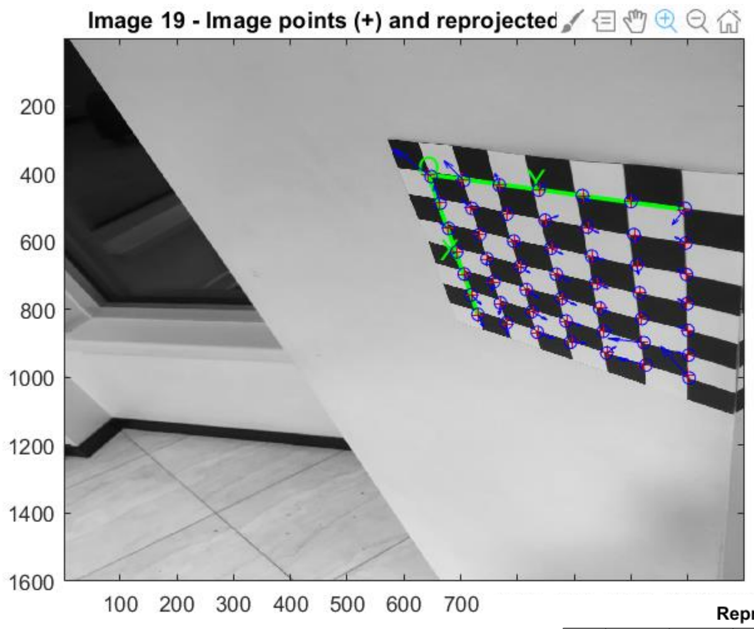
Distortion: kc = [0.00543   0.02250   0.00492   0.00105 0.00000] +/- [0.04260   0.17789   0.00627   0.00550 0.00000]

Pixel error: err = [1.43095   1.65710]

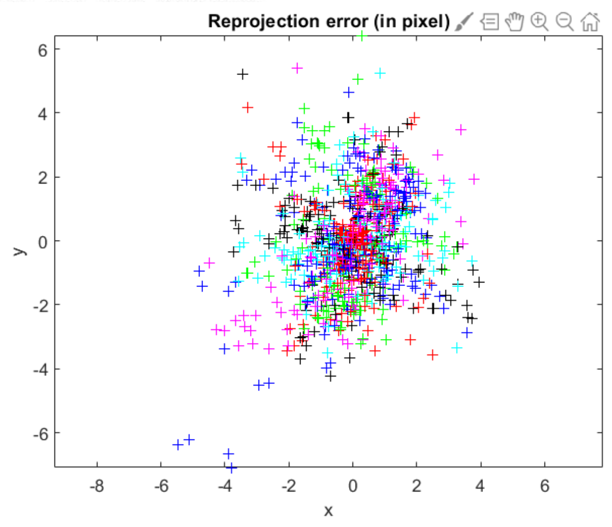
We want to decrease this pixel error, in order to achieve this goal we need to examine images separately. For examination, reprojection of grid points can be useful.







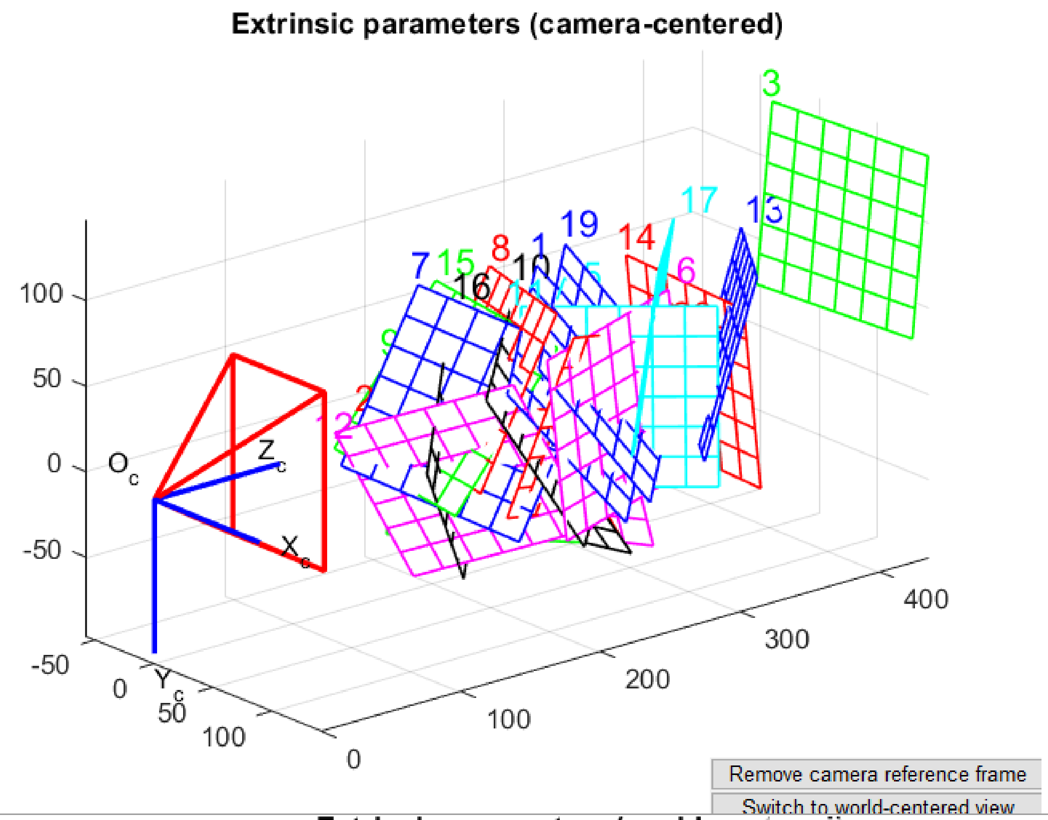
By clicking the analyze error, we will look at the reprojection error:

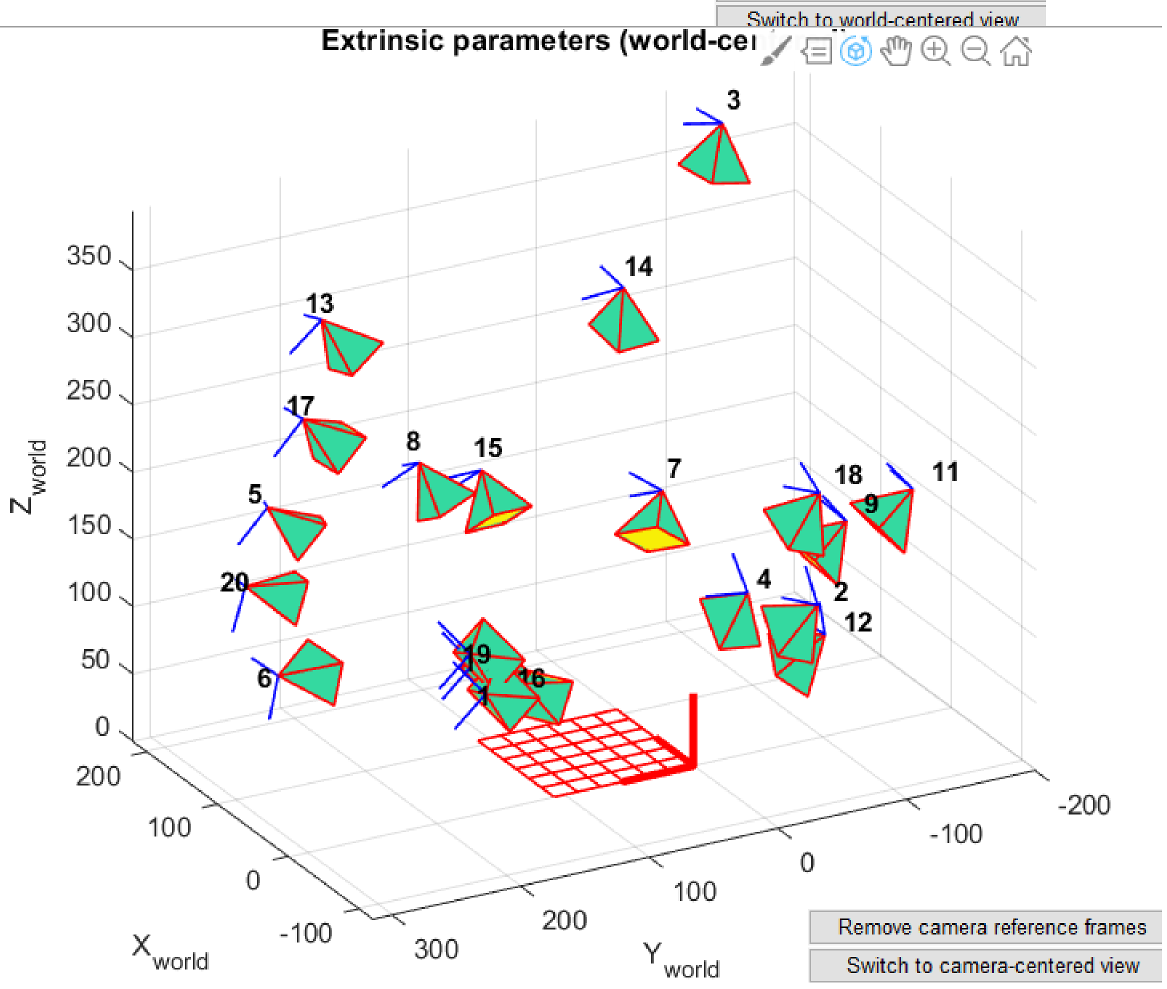


Pixel error: err = [1.43095   1.65710]

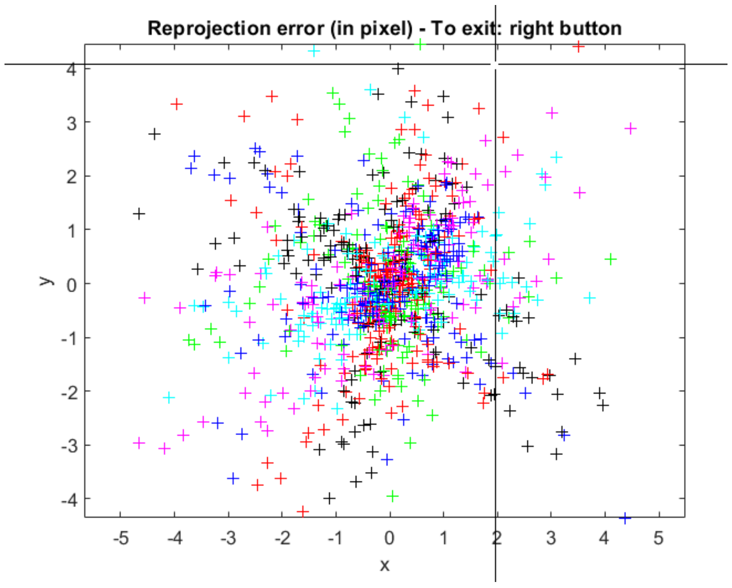
(all active images)

In this tool we can also see the representation of extrinsic parameters in 3D plane which shows from which angles the images were taken in real life coordinates.





We will use the reprojection error graph that can be seen on the right to detect which images are causing the most error in the camera calibration. In order to do that, we find the right-top image that has the maximum pixel error and we get the following resut:



Selected image: 12

Selected point index: 23

Pattern coordinates (in units of (dX,dY)): (X,Y)=(1,3)

Image coordinates (in pixel): (817.58,775.94)

Pixel error = (4.47517,2.87768)

Window size: (wintx,winty) = (5,5)

We see that (wintx,winty) = (5,5) can cause a problem and we decreased this number to (wintx,winty) = (3,3).

After decreasing to (wintx,winty) = (3,3) we get the following image that has the maximum error

After Window size = 3

Selected image: 18

Selected point index: 15

Pattern coordinates (in units of (dX,dY)): (X,Y)=(0,4)

Image coordinates (in pixel): (801.50,715.43)

Pixel error = (2.84799,2.01666)

Window size: (wintx,winty) = (3,3)

Observe that only six iterations were necessary for convergence, and no initialization step was performed (the optimization started from the previous calibration result).

Observe that the uncertainties on the calibration parameters are also estimated.

Calibration results after optimization (with uncertainties):

Focal Length: fc = [1366.96496   1363.47282] +/- [9.42235   8.45198]

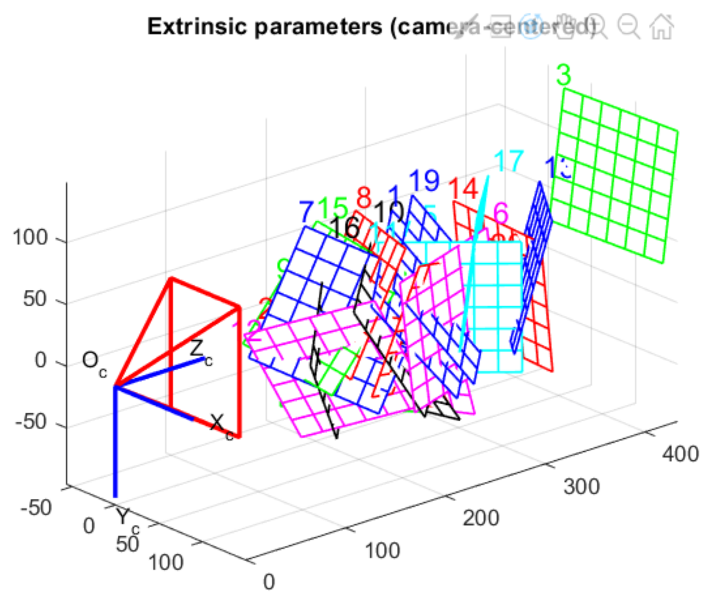
Principal point: cc = [600.87055   802.21071] +/- [16.05872   18.88992]

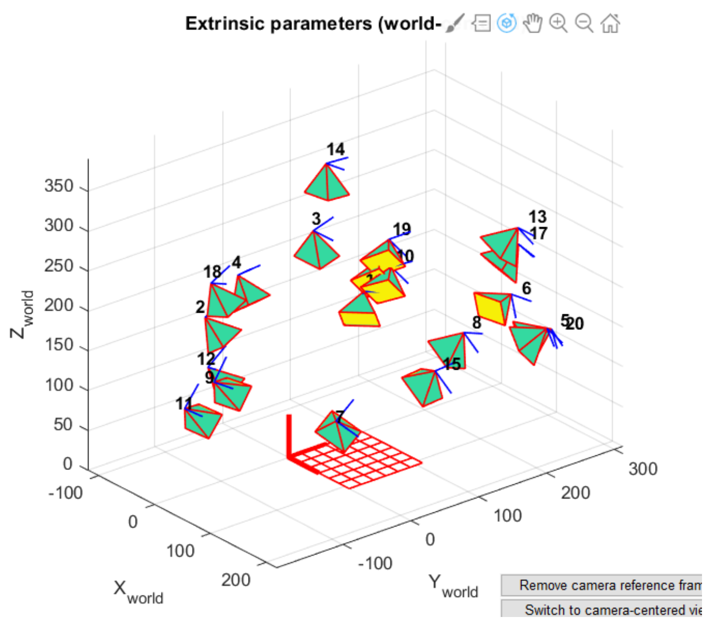
Skew: alpha_c = [0.00000] +/- [0.00000]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees

Distortion: kc = [0.01956   0.01099   0.00439   -0.00026 0.00000] +/- [0.03469   0.13611   0.00541   0.00481 0.00000]

Pixel error: err = [1.33728   1.29067]

Let us check new representation of extrinsic parameters:





We followed this logic and decreased window size again for the images that has maximum errors.

Improvements can be seen below:

Calibration results after optimization (with uncertainties):

Focal Length: fc = [1362.21732   1361.58376] +/- [6.82167   6.17159]

Principal point: cc = [594.45286   812.38695] +/- [11.56384   13.54440]

Skew: alpha_c = [0.00000] +/- [0.00000]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees

Distortion: kc = [0.02301   0.02883   0.00687   -0.00114 0.00000] +/- [0.02418   0.09190   0.00396   0.00357 0.00000]

Pixel error: err = [0.98320   0.92485]

Selected image: 12

Selected point index: 30

Pattern coordinates (in units of (dX,dY)): (X,Y)=(1,2)

Image coordinates (in pixel): (715.82,823.37)

Pixel error = (2.86614,2.46788)

Window size: (wintx,winty) = (3,3)

We used the (wintx,winty) = (1,1):

Window size = 1

Calibration results after optimization (with uncertainties):

Focal Length: fc = [1363.92852   1361.47742] +/- [2.27985   2.03903]

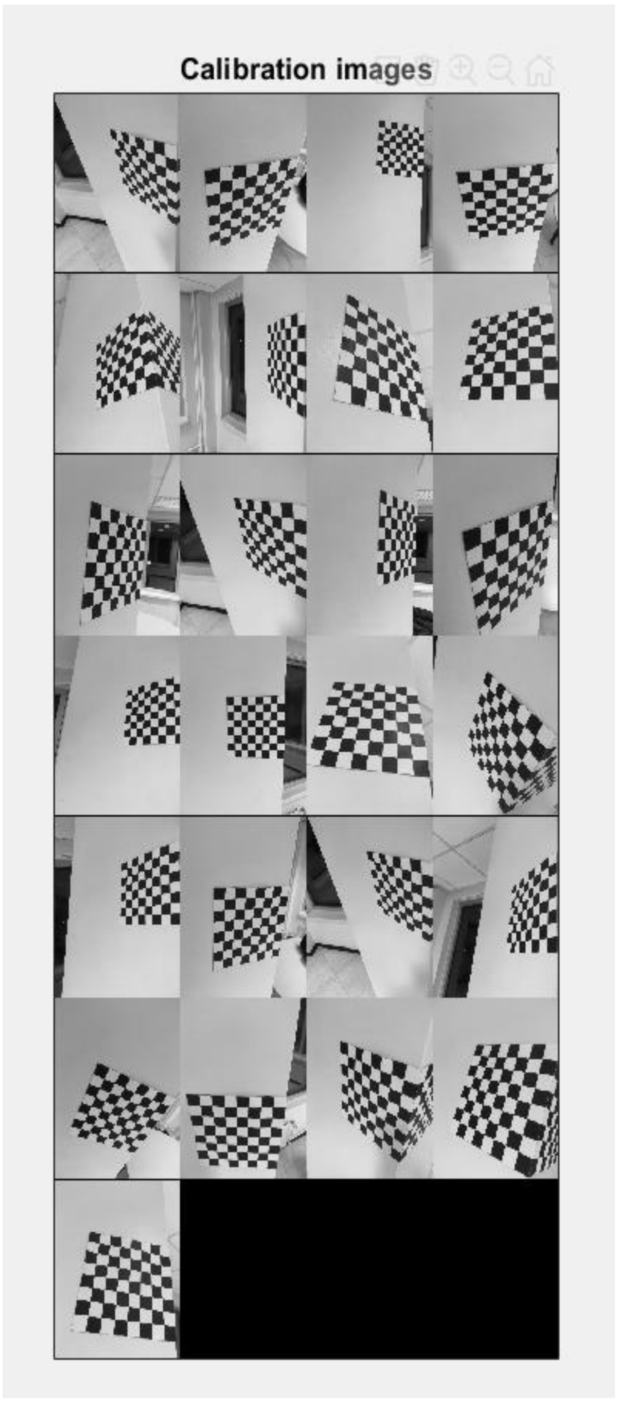
Principal point: cc = [602.50215   789.18370] +/- [3.90626   4.60368]

Skew: alpha_c = [0.00000] +/- [0.00000]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees

Distortion: kc = [0.02926   0.01535   -0.00054   0.00202 0.00000] +/- [0.00869   0.03420   0.00137   0.00119 0.00000]

Pixel error: err = [0.31620   0.32470]

Actually we have a good pixel error but to improve more we added 5 extra images:



The final results are below:

Focal Length: **fc = [1359.62628 1358.18255] +/- [1.94758 1.80501]**

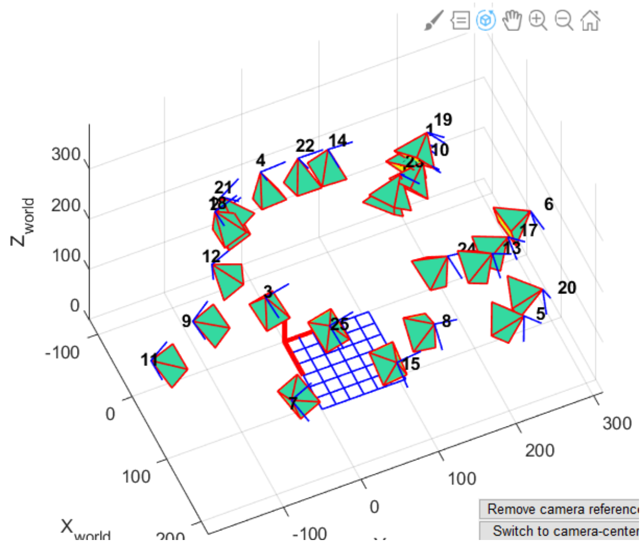
Principal point: **cc = [610.89195 784.34440] +/- [3.31461 3.73085]**

Skew: alpha_c = [0.00000] +/- [0.00000] => angle of pixel axes = 90.00000 +/- 0.00000 degrees

Distortion: kc = [0.03470 -0.00515 -0.00232 0.00402 0.00000] +/- [0.00681 0.02545 0.00113 0.00101 0.00000]

Pixel error: **err = [0.31614 0.32320]**

We can say that distortion is not so high therefore we can conclude that our camera is calibrated with a 0.316 and 0.323 pixel error.



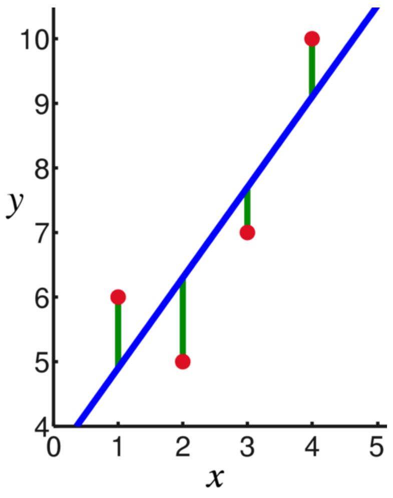
DISCUSSION & COMPARISON

After applying both methods, now we can discuss about the results we got and compare the performances of these techniques.

In the Direct Camera Calibration method (the first one), we used SVD method by the help of extracted corners. If we use more corners (in our case we used 91), we can calibrate our camera more precise. By use of K camera matrix we can say that focal length of our camera is around fx =1745mm fy=1829mm, u0 = 1248 v0=492. Although these results are different than the results we got from the second toolbox method, they are showing consistency.

Actually, I would use the toolbox method to calibrate my camera because it gave us a more robust solution because of its characteristics. To compare, this method is using the initializing and making improvements on each steps eventually reaching a convergent solution.

On the other hand, SVD method uses linear least square method in order to achieve decomposition as can be seen below:



This graph actually makes an approximation and tries to create a linear model that gives the minimum Euclidian distances between the corresponding points (i.e. minimum error).

However, toolbox method is more flexible than SVD model because in this model we can actually regulate the parameters in each separate images and then combine of them. We have done it in a iterative manner and improved our performance.

CODES

cameracalibration.m

```
img = imread('calobject.jpg');
[row,col,ch] = size(img);
img = rgb2gray(img);

corners = corner(img,'Harris');
imshow(img);
hold on;
plot(corners(:,1),corners(:,2),'r*');

% pix, piy, x,y,z
corresp = [ 827 172  0 0 0; 913 188  1 0 0; 991 206  2 0 0; 1060 218
3 0 0; 1132 232 4 0 0; 1175 244 5 0 0; 1223 253 6 0 0;
            829 309  0 0 1; 913 319  1 0 1; 991 328  2 0 1; 1059 334
3 0 1; 1116 339 4 0 1; 1170 345 5 0 1; 1217 350 6 0 1;
            824 446  0 0 2; 988 445  1 0 2; 1052 446 2 0 2; 1111 446
3 0 2; 1163 446 4 0 2; 1213 447 5 0 2; 1258 447 6 0 2;
            827 576  0 0 3; 907 568  1 0 3; 983 562  2 0 3; 1048 557
3 0 3; 1107 551 4 0 3; 1159 546 5 0 3; 1254 537 6 0 3;
            821 706  0 0 4; 905 691  1 0 4; 983 675  2 0 4; 1047 664
3 0 4; 1106 651 4 0 4; 1156 643 5 0 4; 1203 635 6 0 4;
            823 832  0 0 5; 904 808  1 0 5; 978 789  2 0 5; 1046 769
3 0 5; 1103 754 4 0 5; 1153 740 5 0 5; 1200 726 6 0 5;
            820 956  0 0 6; 902 926  1 0 6; 978 898  2 0 6; 1042 876
3 0 6; 1103 852 4 0 6; 1151 835 5 0 6; 1199 815 6 0 6;
            727 188 0 1 0; 642 201 0 2 0; 504 225 0 3 0; 450 238 0 4
0; 405 246 0 5 0; 356 355 0 6 0;
            728 316 0 1 1; 644 322 0 2 1; 572 329 0 3 1; 510 334 0 4
1; 454 339 0 5 1; 409 343 0 6 1;
            730 445 0 1 2; 644 443 0 2 2; 576 443 0 3 2; 514 443 0 4
2; 458 442 0 5 2; 410 441 0 6 2;
            731 571 0 1 3; 648 560 0 2 3; 576 552 0 3 3; 517 549 0 4
3; 460 540 0 5 3; 413 537 0 6 3;
            730 691 0 1 4; 648 676 0 2 4; 578 663 0 3 4; 520 652 0 4
4; 464 641 0 5 4; 416 632 0 6 4;
            728 809 0 1 5; 650 791 0 2 5; 581 773 0 3 5; 519 753 0 4
5; 463 736 0 5 5; 417 724 0 6 5;
            728 930 0 1 5; 649 900 0 2 5; 580 876 0 3 5; 521 856 0 4
5; 467 835 0 5 5; 418 813 0 6 5;
            ];

N = 91; % Number of corners that are used to calibrate our camera
L = 18; % Length of each square in mm
Pw = []; % Real world coordinates
Pc = []; % Camera coordinates in pixels
for j = 1:1:N
   Pc = [Pc; corresp(j,1) corresp(j,2) 1];
   Pw = [Pw; corresp(j,3)*L corresp(j,4)*L corresp(j,5)*L 1];
end
```

```
P = [];

for i = 1:1:N
P = [  P;
       Pw(i,:)  zeros(1,4) -Pc(i,1)*Pw(i,:);
       zeros(1,4)  Pw(i,:) -Pc(i,2)*Pw(i,:);
    ];
end

[U S V] = svd(P);
% Last column of V gives m
m = V(:,end);
m = transpose(reshape(m,[],3));

% Normalizing of M to make the norm of third rotation vector unity
norm_31 = norm(m(3,1:3));
M = m / norm_31;

a1 = M(1,1:3);
a2 = M(2,1:3);
a3 = M(3,1:3);
b = M(1:3, 4);
r3 = a3;

% Computation of the intrinsic parameters
u_0 = transpose(a1*transpose(a3));
v_0 = transpose(a2*transpose(a3));
cross_a1a3 = cross(a1,a3);
cross_a2a3 = cross(a2,a3);
theta = acos (-
1*cross_a1a3*transpose(cross_a2a3)/(norm(cross_a1a3)*norm(cross_a2a3
)));
alpha = norm(cross_a1a3) * sin(theta);
beta = norm(cross_a2a3) * sin(theta);

% Computation of the extrinsic parameters
r1 = cross_a2a3/norm(cross_a2a3);
r2 = cross(r3, r1);
K = [alpha -1*alpha*cot(theta) u_0
     0 beta/sin(theta) v_0
     0 0 1];
t = inv(K) * b; % Translation vector

% Rotation matrix
R(1,1:3) = r1;
R(2,1:3) = r2;
R(3,1:3) = r3;
```