

EE 417 Computer Vision

Berkant Deniz Aktaş

19515

Homework 2

November 29, 2018

Sabancı University

Faculty of Engineering and Natural Sciences

Computer Science and Engineering

Introduction

In this homework, aim is calibrating the camera and finding intrinsic and extrinsic parameters by using two different methods. Two methods are used for this purpose. Second method is about extracting corner points then using SVD (Singular Value Decomposition) to create the related matrices. First method is about using a Camera Calibration Toolbox which is developed by Jean-Yves Bouguet in Cal. Tech. First method's main idea is using different photos of a 2D plane and reconstruct the 3D based on the images. Second method's main idea is detecting corner points and calculating rotation, transformation matrices and camera matrix with solving a linear equation with SVD.

Rig and Corner Detection

In order to compute these parameters, I build a very basic calibration object with using two 8x8 checkerboard images. After printing images, I attached both planes on a corner of a wall. Because of folding and attaching problems my object was not perfect enough but still both algorithms worked. One could build a perfect calibration object in order to achieve more robust results.



Figure 1. example image of checkerboard on a corner.

According to the problem definition, we were allowed to choose a corner detection methods from given 3 well known methods. For simplicity, I choose Harris Corner Detector. In the beginning I thought choosing Harris could be problematic because it can only compute integer valued pixels as corners. But computing 60 corner points with using intersection of lines method would be too hard in terms time. So, I stayed with Harris and decided errors due to integer pixel corners would be low.

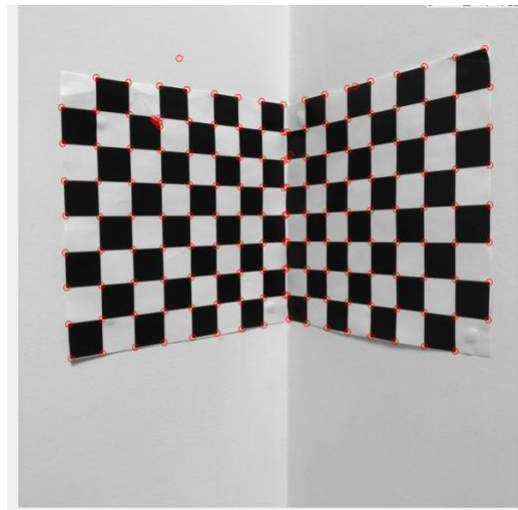


Figure 2. Corners detected as red dots

One can notice that there are misclassified corners due to not perfect construction of the rig. But I was planning to use 60 points for SVD and planning to compute all corners with toolbox so I decided to continue.

Camera Calibration Toolbox

I followed the steps denoted in :

http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html

According to the guide, I uploaded 18 images of the 3D rig. But the problematic part about this implementation is, toolbox working style. In order to detect parameters, toolbox computes different pictures of a 2D plane. Since my object is 3D I could not select all corners of the image, I tested left and right 2D planars of 3D calibration rig.

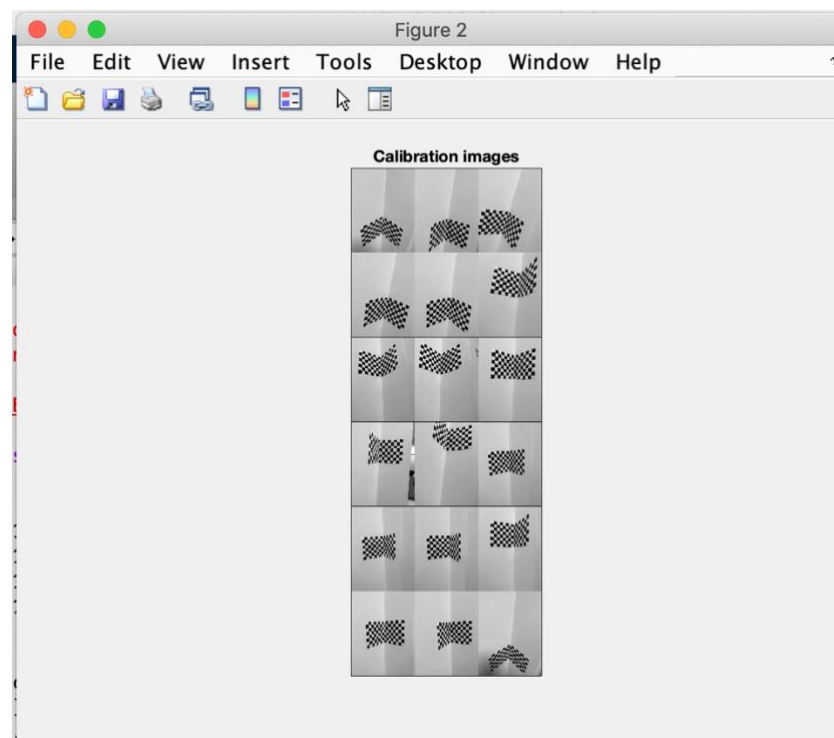


Figure 3. 18 images ready to process

After loading images, toolbox asked me to select main corners of each image, number of squares in a row and column and square length.

In this example I used, 14cm x 14cm checkerboard which as 8 squares on each row and column. Thus, one square is 17.5mm x 17.5mm length.

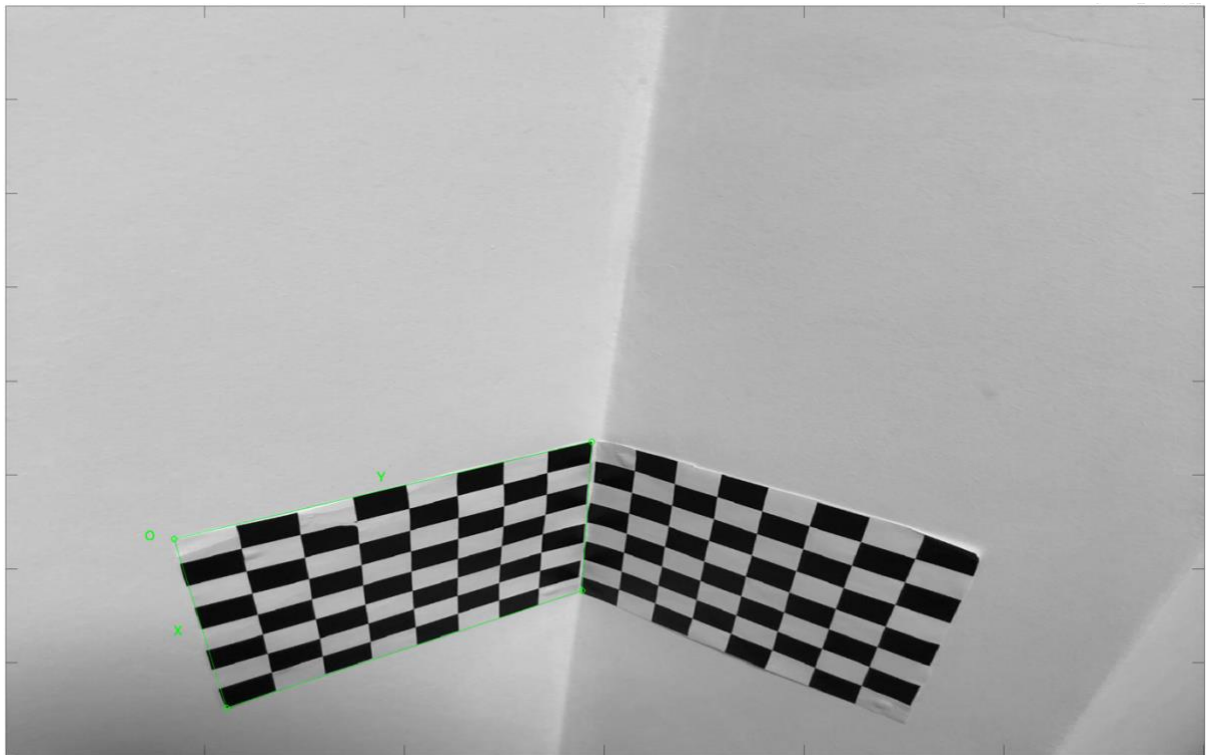


Figure 4. After extracting main corners

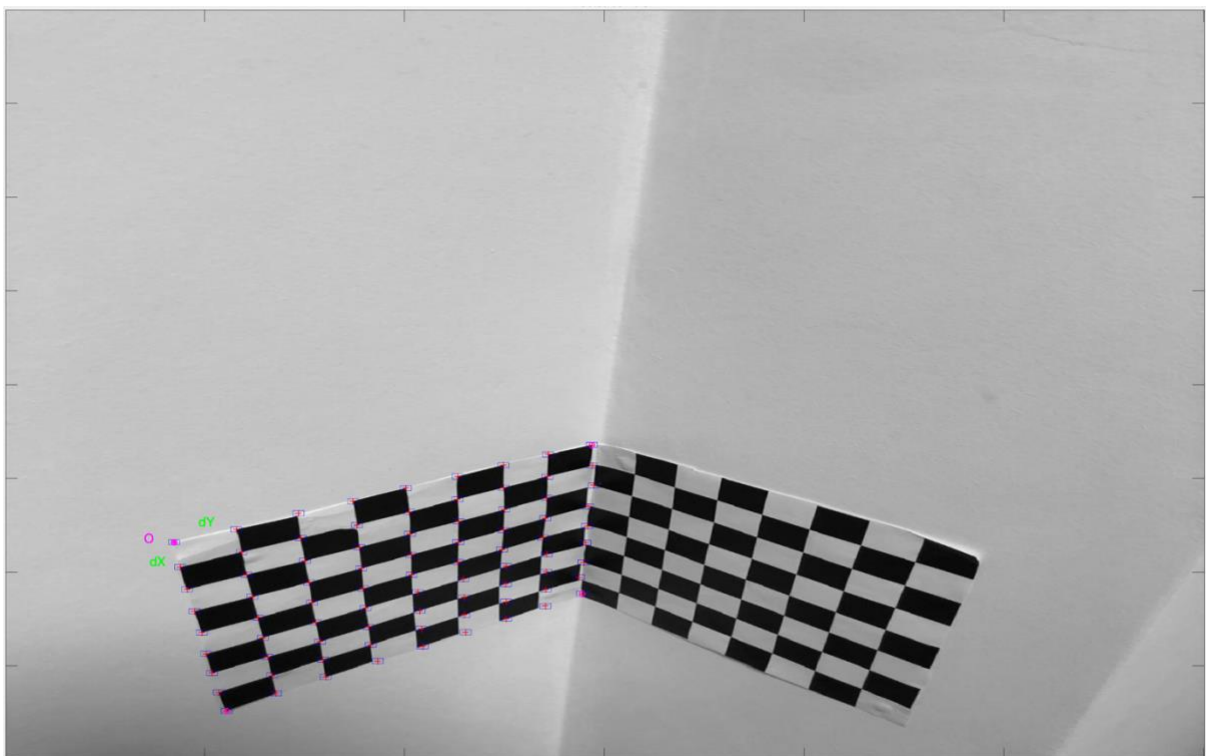


Figure 5. Corners of an image.



Figure 6. Zoom in to a corner

As we can see in the Figure 4,5 and 6. Program succesfully extracted some pixels which are very close to corners of the image. But they are not perfect because of non-perfect rig and not selecting precise main corners. After selecting all corners and processing the images, programs asks us to compute calibration just by clicking Calibration button.

```

Command Window
Size of each square along the Y direction: dy=1/.5mm (Note: to reset the size of the squares, clear the variables dx and dy)
If the guessed grid corners (red crosses on the image) are not close to the actual corners,
it is necessary to enter an initial guess for the radial distortion factor kc (useful for subpixel detection)
Need of an initial guess for distortion? ([]=no, other=yes)
Corner extraction...
done

Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .
Initialization of the principal point at the center of the image.
Initialization of the intrinsic parameters using the vanishing points of planar patterns.

Initialization of the intrinsic parameters - Number of images: 18

Calibration parameters after initialization:

Focal Length:      fc = [ 1295.44770  1295.44770 ]
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 ]

Main calibration optimization procedure - Number of images: 18
Gradient descent iterations: 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...16...17...18...19...20...21...22...23...24...25...26...27...28...done
Estimation of uncertainties...done

Calibration results after optimization (with uncertainties):

Focal Length:      fc = [ 1400.71466  1383.67933 ] +/- [ 30.60034  35.71088 ]
Principal point:   cc = [ 638.19597  834.87600 ] +/- [ 45.53438  23.18954 ]
Skew:              alpha_c = [ 0.00000 ] +/- [ 0.00000 ] => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:        kc = [ 0.12952  -0.27898  0.02348  0.00721  0.00000 ] +/- [ 0.03654  0.06203  0.00557  0.00911  0.00000 ]
Pixel error:       err = [ 1.95588  2.33656 ]

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

```

Figure 7. First calibration results

As it is shown on the Figure 7. Program computed some results but the pixel error is too high as its shown on the Figure 6. According to guide we can improve our results by analyzing error and reducing it.

Now lets try to look all images and their error graph.

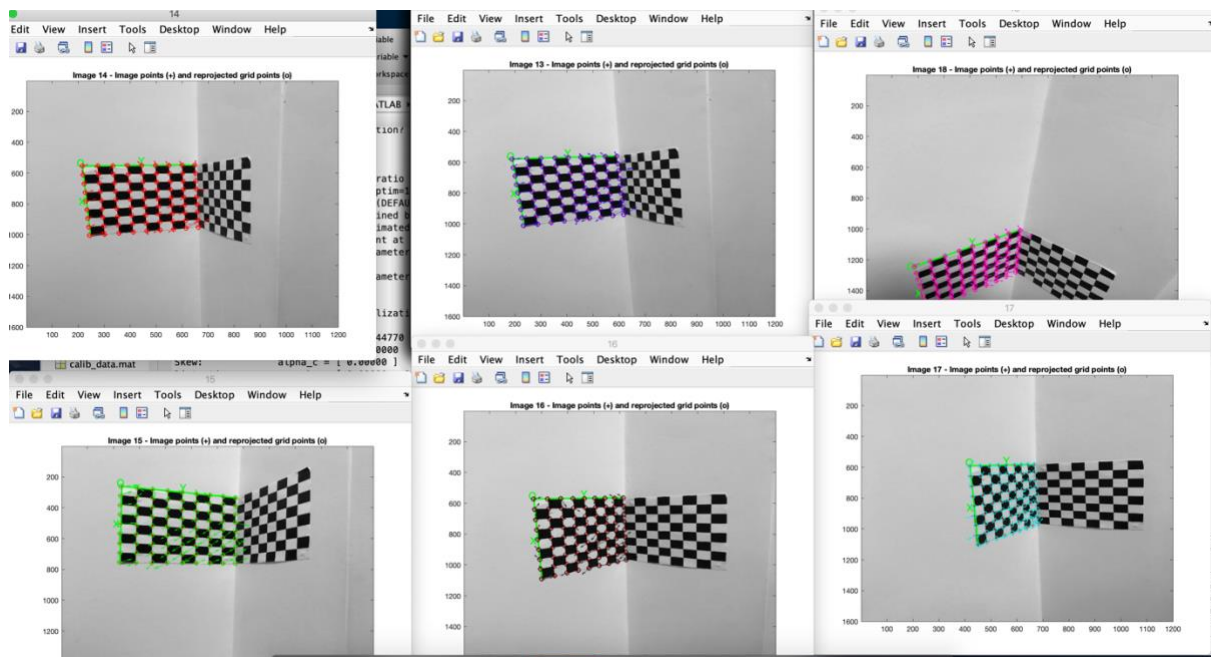


Figure 6. Detected corners of all images.

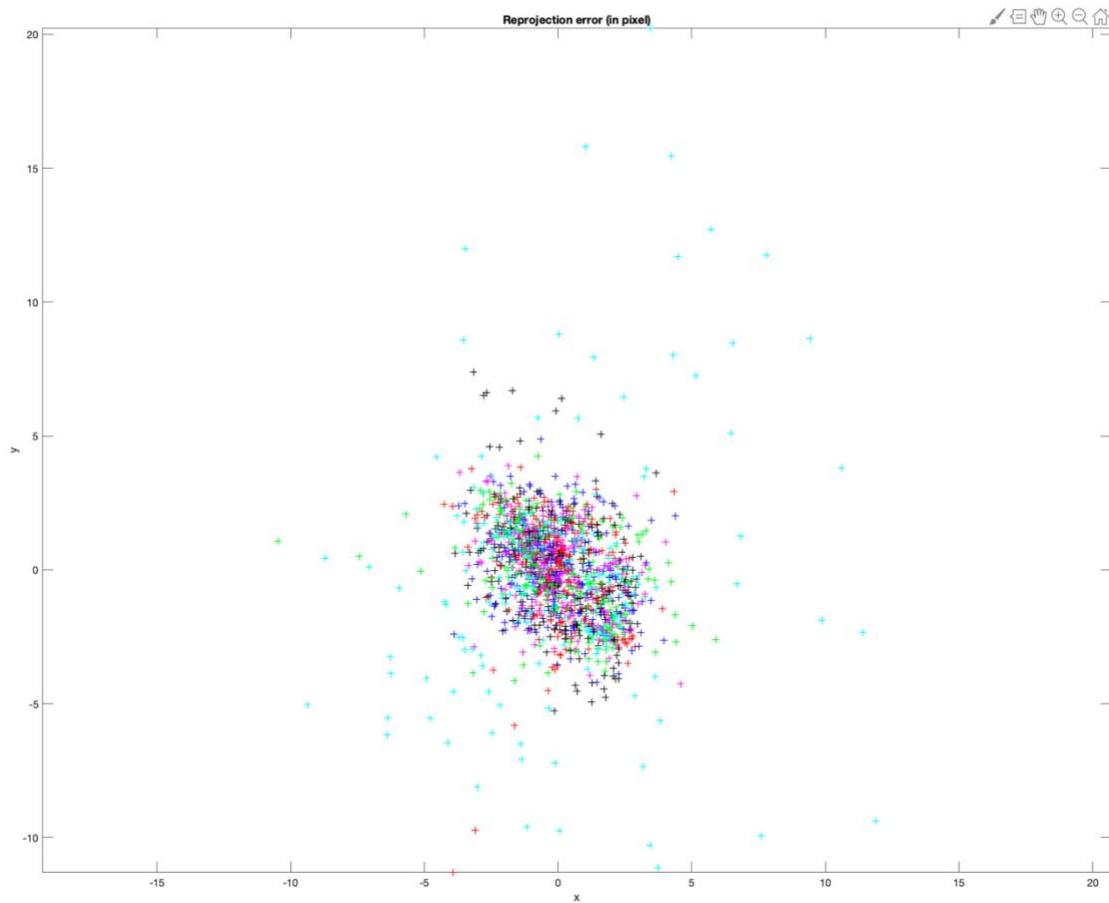


Figure 7. Projection error of the images in Figure 6.

We can show the interpreted camera parameters and extrinsic parameters of each image with a using toolbox.

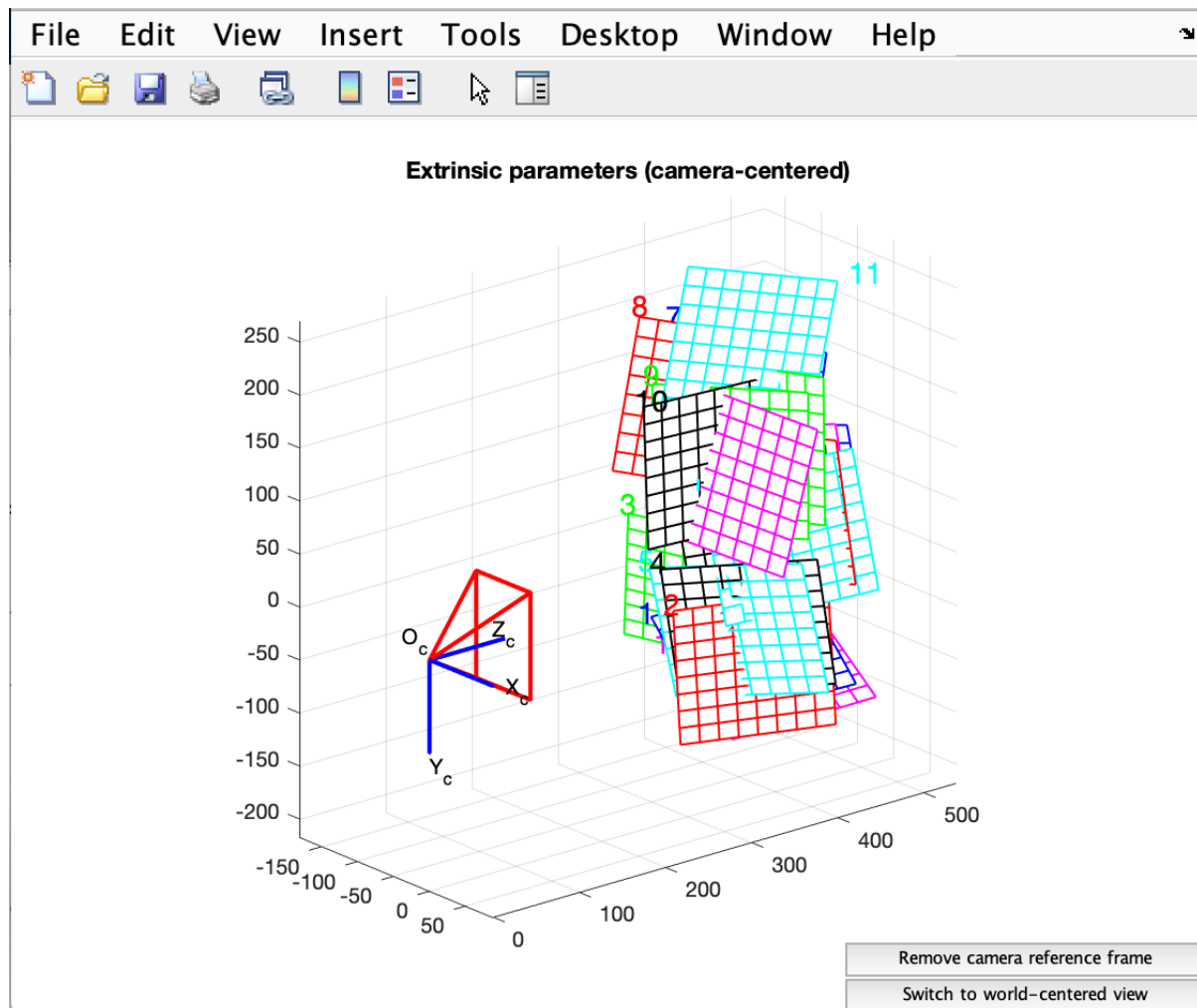


Figure 8. Camera centered view of the planes.

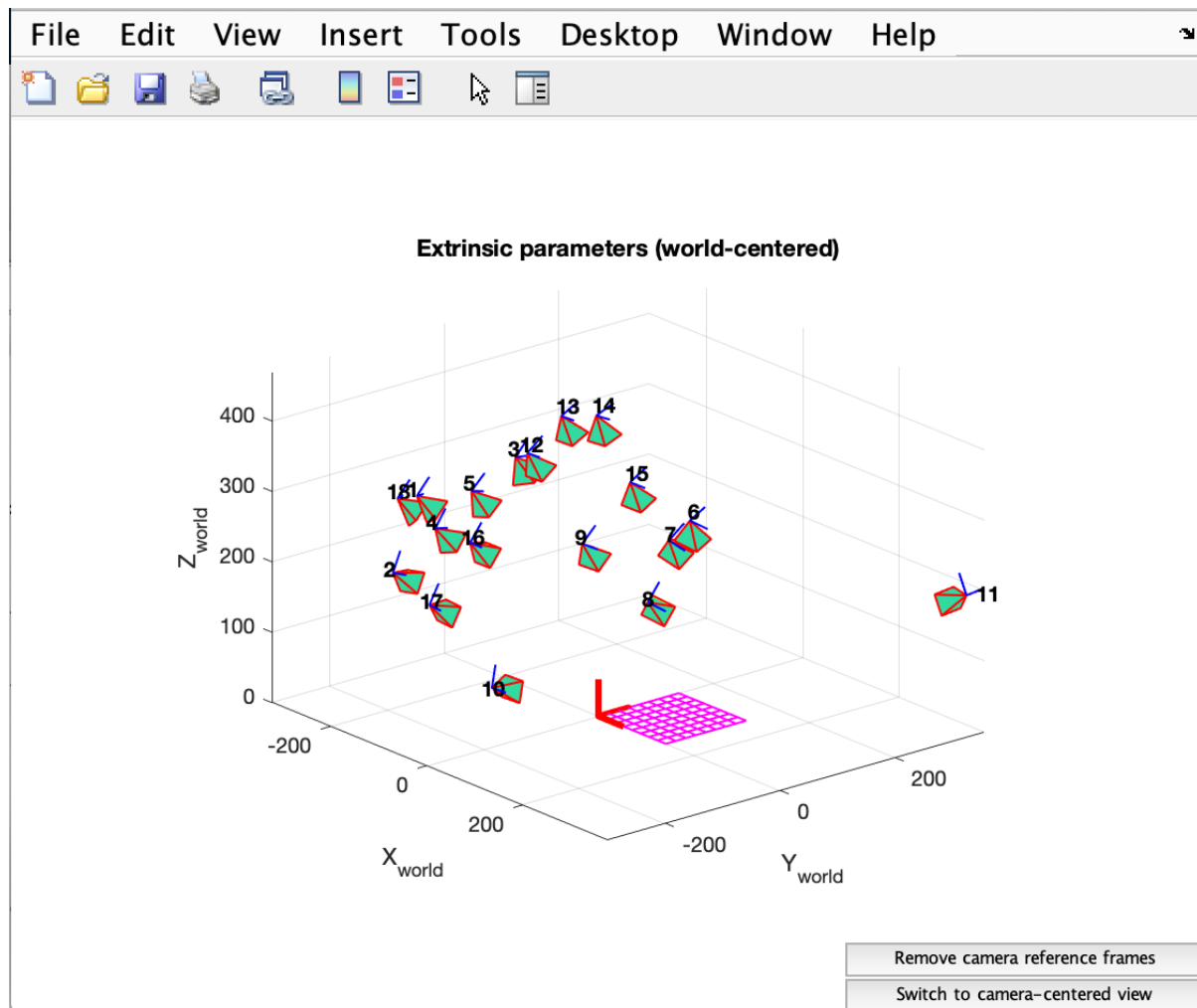


Figure 9. Plane centered view of cameras

As its shown in the figure 7. Some images have high error e.g cyan colored marks. Now our aim is decreasing this error. With recomputing the corners instead of manually marking them with using the previous calculated results.

Calibration results after optimization (with uncertainties):

```
Focal Length:      fc = [ 1367.54242  1350.32443 ] +/- [ 18.12769  21.14302 ]
Principal point:   cc = [ 601.09794   824.90280 ] +/- [ 26.79482   12.50287 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000 ] => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:       kc = [ 0.12742  -0.28063  0.01818  0.01032  0.00000 ] +/- [ 0.02155  0.03571  0.00303  0.00523  0.00000 ]
Pixel error:      err = [ 1.35297  1.24090 ]
```

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

Figure 10. Recomputing corners with previous results

We can see that Figure 10 and Figure 7. Has different results. The pixel error is reduced and different results of parameters are achieved. Both methods we used 5x5 window but toolbox guide suggests us to change different window sizes to increase accuracy and decrease pixel error. Lets analyze the results on achieved on Figure 10.

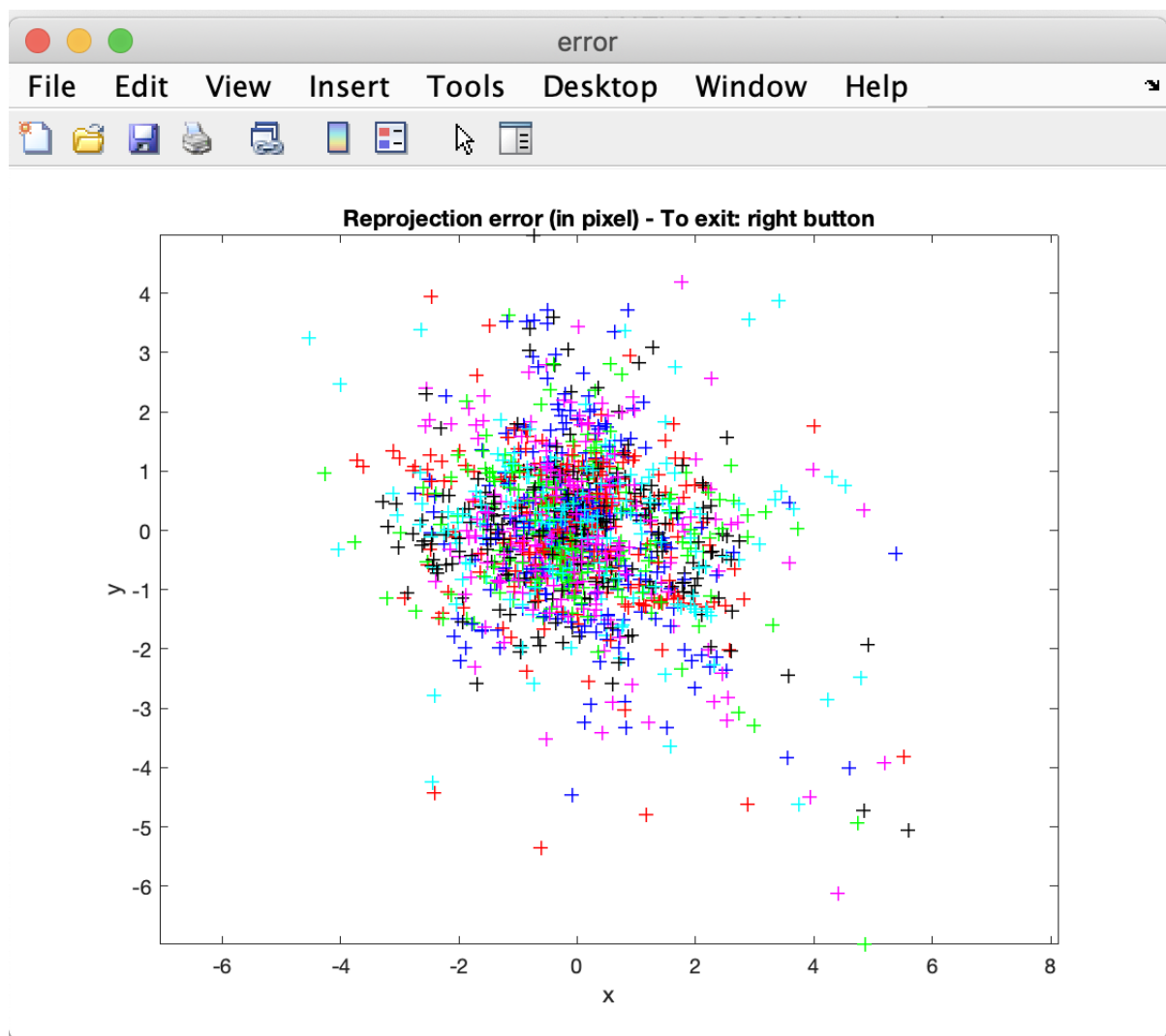


Figure 11. Error matrix achieved in second computation.

If we compare Figure 11 and Figure 7 we can see that the distribution is better and error is reduced but lets select a point which is not close to (0,0) in error matrix and look for it in image. With clicking and looking for the image corners :

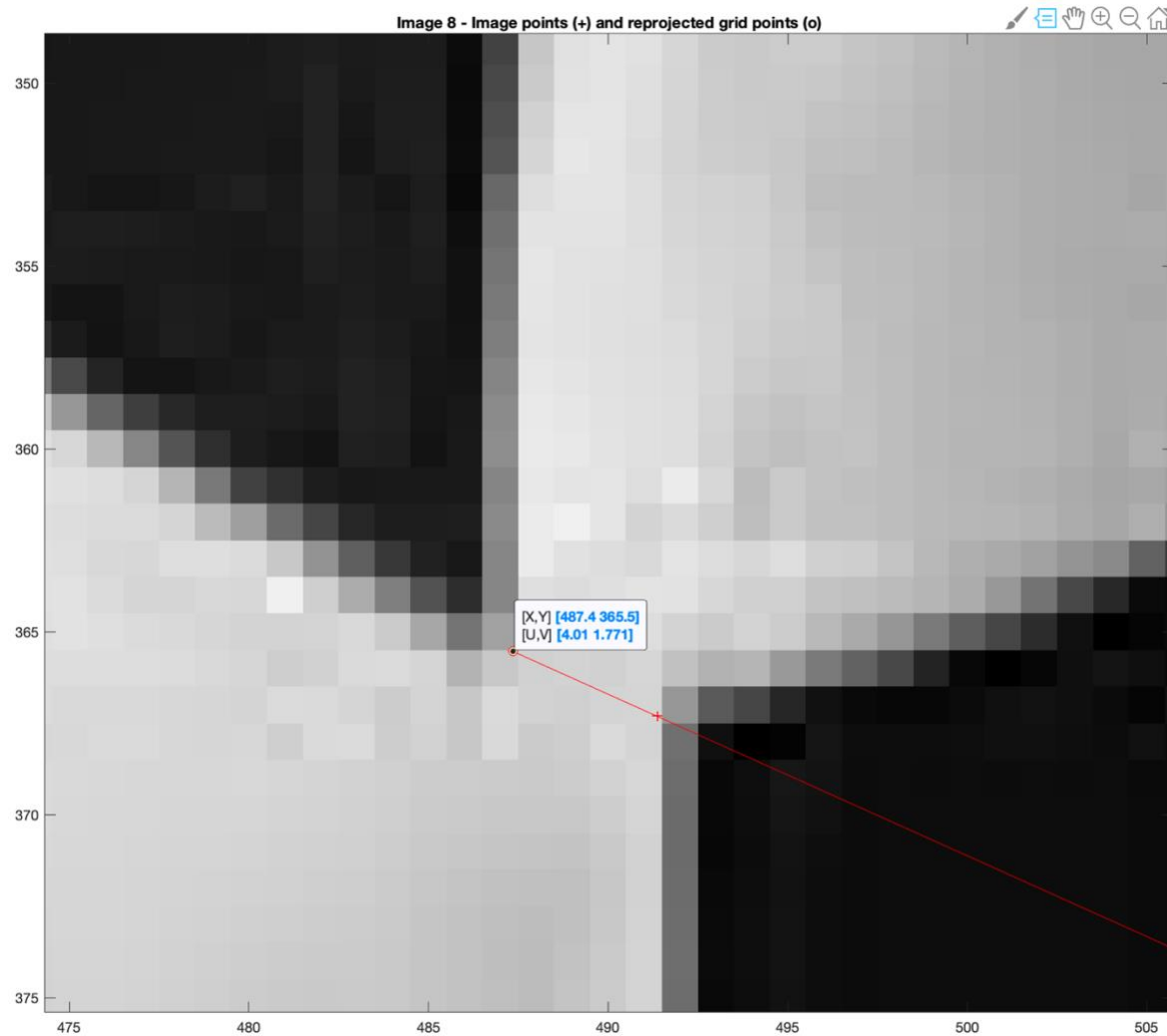


Figure 12. Pixel error of the one corner in image 8

We can see that there is difference in the results. Now lets try to recompute it with a smaller window size 3x3 and check for results.

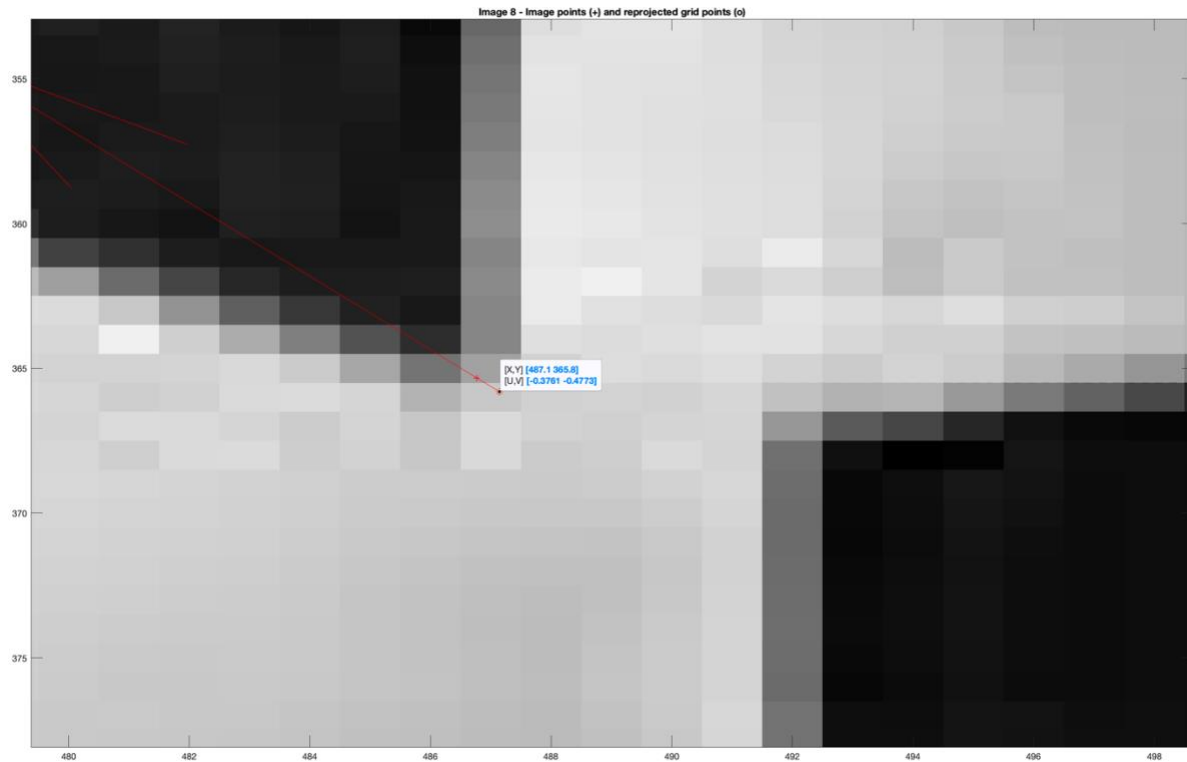


Figure 13. After computing with smaller window size

Comparing Figure 13 and Figure 12, we can see that we reduced error for this corner point.

Let's check general pixel error and calibration results:

```

Re-extraction of the grid corners on the images (after first calibration)
Window size for corner finder (wintx and winty):
wintx ([]) = 5) = 3
winty ([]) = 5) = 3
Window size = 7x7
Number(s) of image(s) to process ([]) = all images) =
Use the projection of 3D grid or manual click ([]) = auto, other = manual):
Processing image 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...16...17...18...
done

Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .

Main calibration optimization procedure - Number of images: 18
Gradient descent iterations: 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...16...17...18...19...done
Estimation of uncertainties...done

Calibration results after optimization (with uncertainties):

Focal Length:      fc = [ 1354.14490   1356.46518 ] +/- [ 12.05866   14.63924 ]
Principal point:   cc = [ 608.00155   823.87523 ] +/- [ 18.71257    8.63815 ]
Skew:              alpha_c = [ 0.00000 ] +/- [ 0.00000 ] => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:        kc = [ 0.10309   -0.26541   0.01583   0.00188   0.00000 ] +/- [ 0.01495   0.02447   0.00205   0.00371   0.00000 ]
Pixel error:        err = [ 0.96873   0.83513 ]

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

```

Figure 14. Calibration results with 3x3 window

We can see that Figure 14, which has 3x3 window and bases previous results as corners has smaller error than previous computations shown in Figure 7 and Figure 10. We can use Figure 14's results for our intrinsic parameters.

Now lets put an image which is not in the dataset to compute extrinsic parameters.

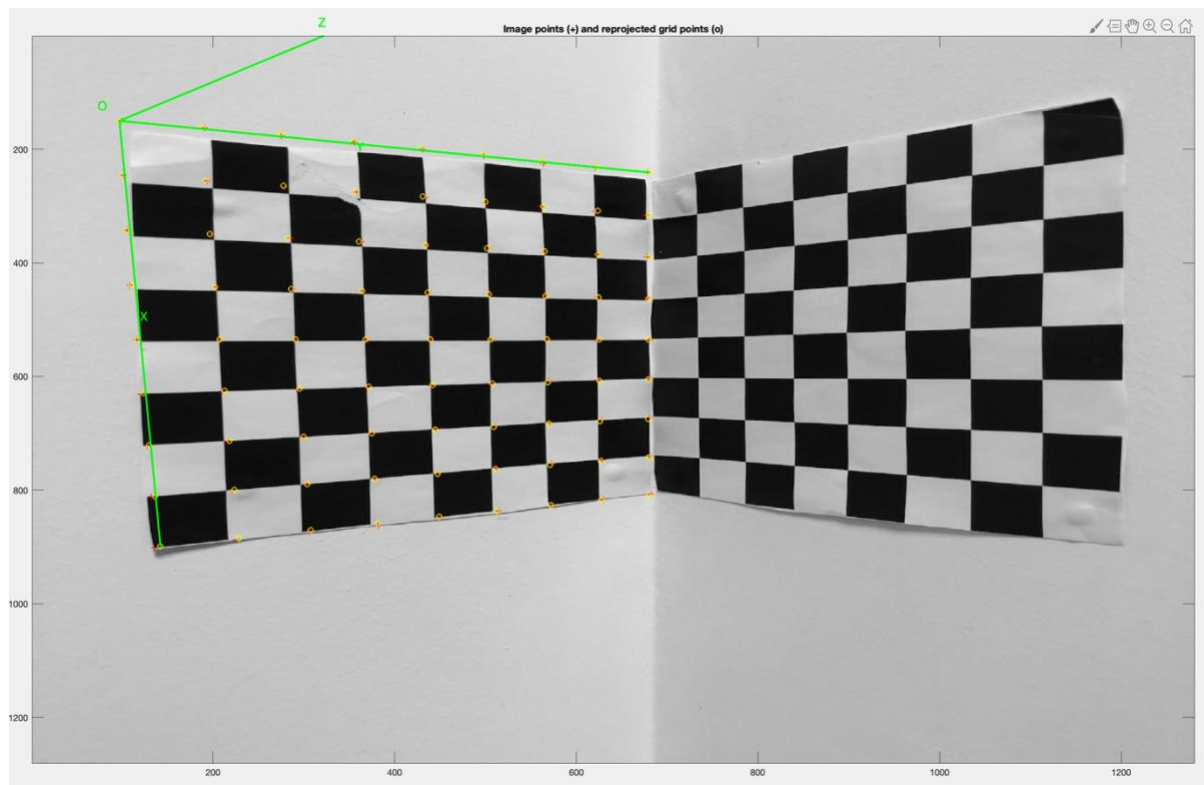


Figure 15. New image for extrinsic

```

Interrupt while evaluating UIControl Callback.

.          12.jpg          17.jpg          4.jpg          9.jpg
..         13.jpg          18.jpg          5.jpg          Calib_Results.m
.DS_Store  14.jpg          19.jpg          6.jpg          Calib_Results.mat
10.jpg     15.jpg          2.jpg           7.jpg          calib_data.mat
11.jpg     16.jpg          3.jpg           8.jpg          img.jpg

Computation of the extrinsic parameters from an image of a pattern
The intrinsic camera parameters are assumed to be known (previously computed)

Image name (full name without extension): img
Image format: (['r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg', 'm'='ppm']) j

Extraction of the grid corners on the image
Window size for corner finder (wintx and winty):
wintx ([]) = 5) = 3
winty ([]) = 5) = 3
Window size = 7x7
Click on the four extreme corners of the rectangular complete pattern (the first clicked corner is the origin)...
Could not count the number of squares in the grid. Enter manually.
Number of squares along the X direction ([]) = 10) = 8
Number of squares along the Y direction ([]) = 10) = 8
Size dX of each square along the X direction ([]) = 30mm) = 17.5
Size dY of each square along the Y direction ([]) = 30mm) = 17.5
Corner extraction...

Extrinsic parameters:

Translation vector: Tc_ext = [ -92.742977      -123.353320      239.678719 ]
Rotation vector:   omc_ext = [  1.965189      1.815964      0.840544 ]
Rotation matrix:   Rc_ext = [  0.010533      0.782774      0.622217
                             0.980919      -0.128894      0.145550
                             0.194133      0.608811     -0.769195 ]

Pixel error:      err = [ 1.93504   1.49100 ]
fx >> |

```

Figure 17. Extrinsic parameters of the image in Figure 16

Figure 17 shows the extrinsic parameters. Translation vector's norm will result us the distance between camera and image center. The distance is approximately 284 meters which is a logical number. Unfortunately I could not remember the distance when I was taking this picture and I can not take another one because I dismantled the rig from the wall but it should be between 20cm-30cm. Notice that we are working with the left plane and took the photne from the left of the left plane. According to toolbox guide:

$$Rc_ext = rodrigues(omc_ext).$$

$$XX_c = Rc_ext * XX + Tc_ext$$

Where P is a point in a space of coordinate vector $XX = [X;Y;Z]$ in the grid reference frame (O,X,Y,Z) . Let $XX_c = [X_c;Y_c;Z_c]$ be the coordinate vector of P in the camera reference frame (O_c,X_c,Y_c,Z_c) . (This definition is taken from the toolbox website directly).

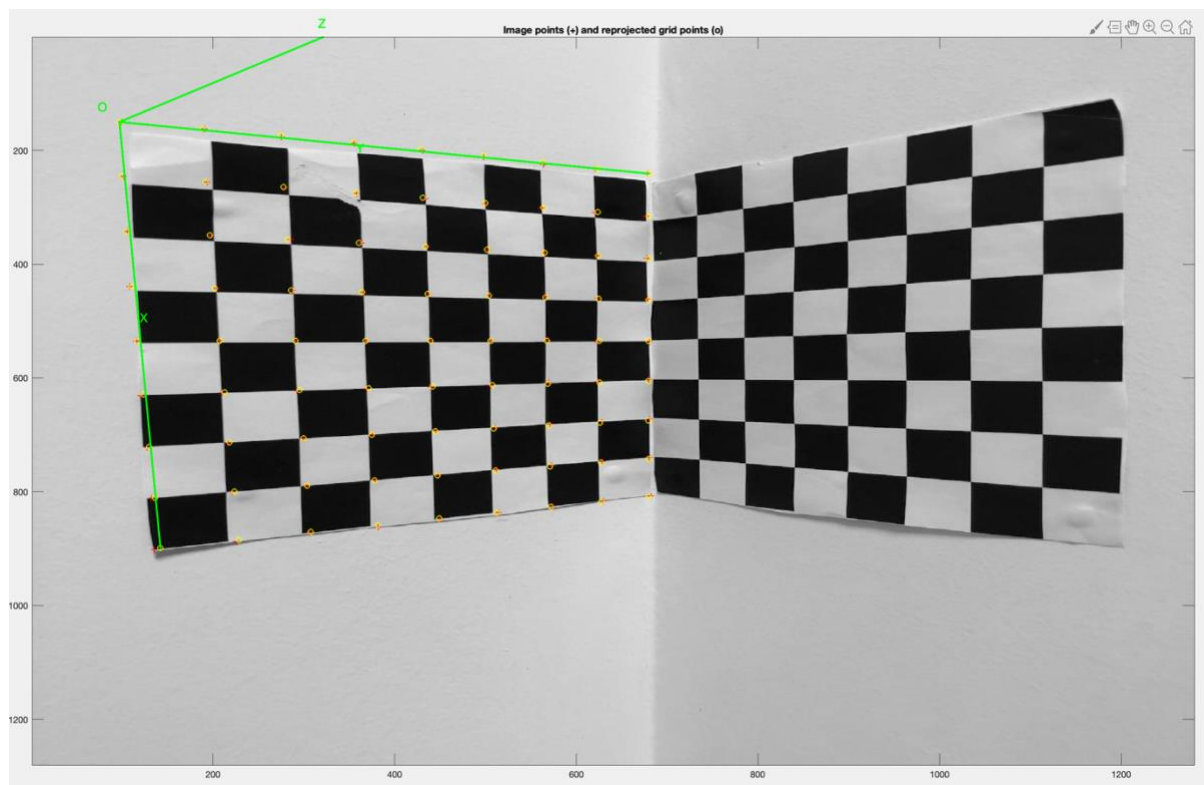


Figure 18. (O,X,Y,Z) parameters in image grid

Results

With this method we computed intrinsic parameters of the camera and computed extrinsic parameters of the camera given an image. One can increase number of photos and increase accuracy of results. 20 Images seems enough but working with more images will conclude better results. While doing this job, most time consuming part was selecting the main corners of the image. I thought using a simple corner detector and finding maximum minimum x and y coordiantes of the corners for automatic calibration would be really good.

Camera Projection Matrix

In this method, we will work with a single 3D plane and try to compute parameters with building 3D world. While searching for this homework, I found a very useful link and implemented similar method in order to understand the topic.

http://www.sci.utah.edu/~gerig/CS6320-S2013/Materials/hw1_abhishek_projectreport_b.pdf

This guy from Utah University implemented a similar method for his own 3D calibration rig. I read the paper and tried to follow same methods for my calibration object. I would like to mention that I did not directly copy paste the code and I tried to understand all of its aspects with comparing the formulas in the lecture slides and I implemented the code with different parameters which are suitable for my calibration object. First, I found the Harris corners of the 3D calibration rig. As its shown on the Figure 19 there are misclassified corners due to some problems as I stated before. But since we have 12 unknowns and we can use atleast 60 points to estimate them choosing a 66 of the detected corners would be enough.

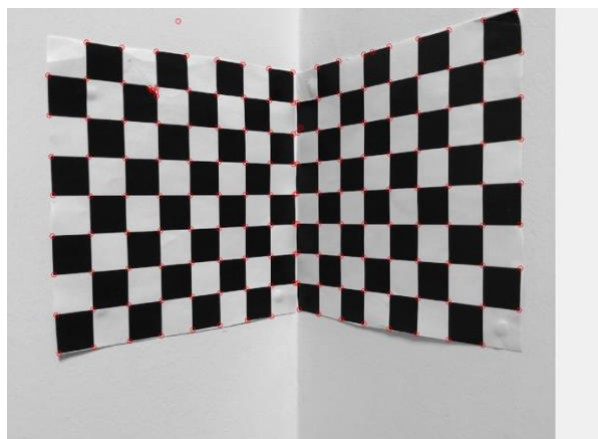


Figure 19. Harris Corners of my 3D calibration object

After detecting corners, I took a window in the image shown on Figure 20.

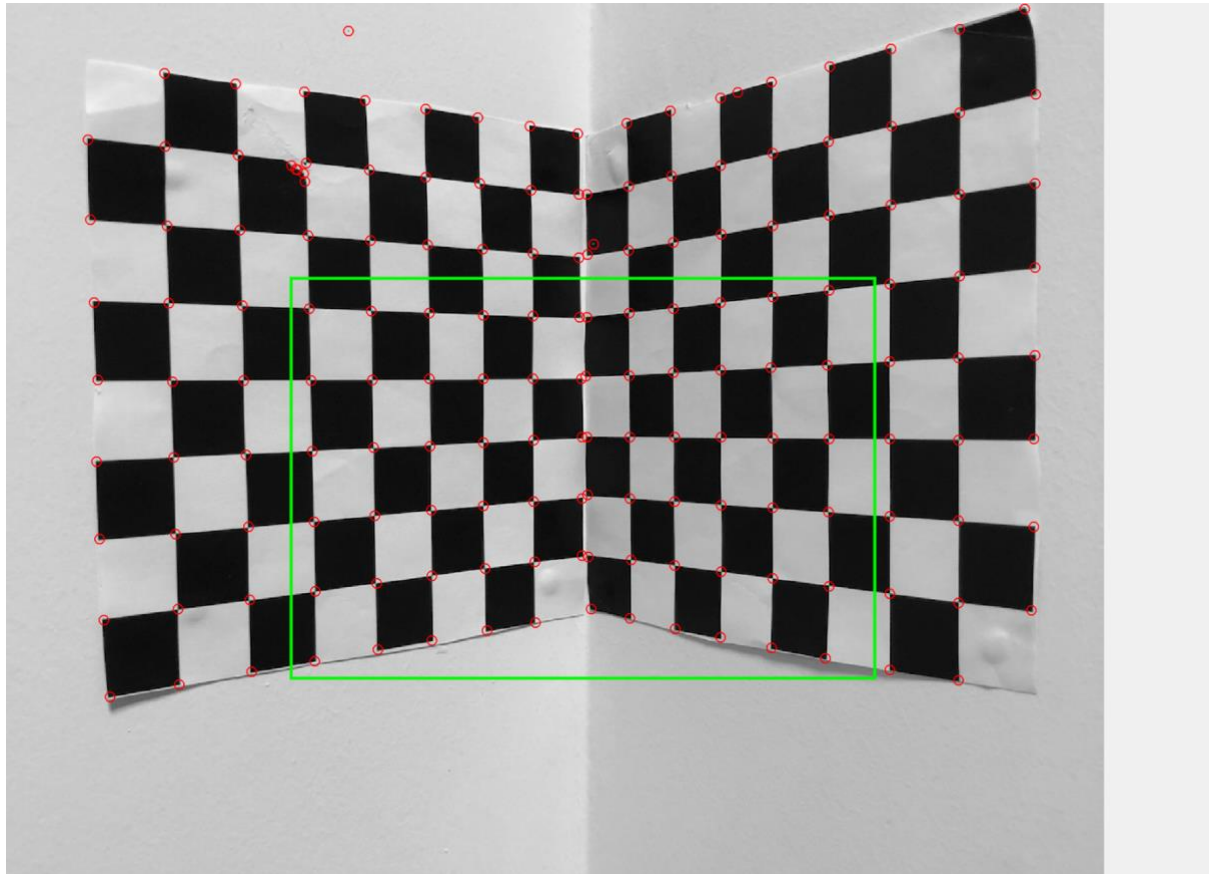


Figure 20. Subset of corners

I decided to use 66 corners in the green window. Note that there are some duplicate corners in the middle. I discarded one of the duplicate ones and selected one of them.

Before computing I created 3D World of these 66 points by manually computing x,y and z values.

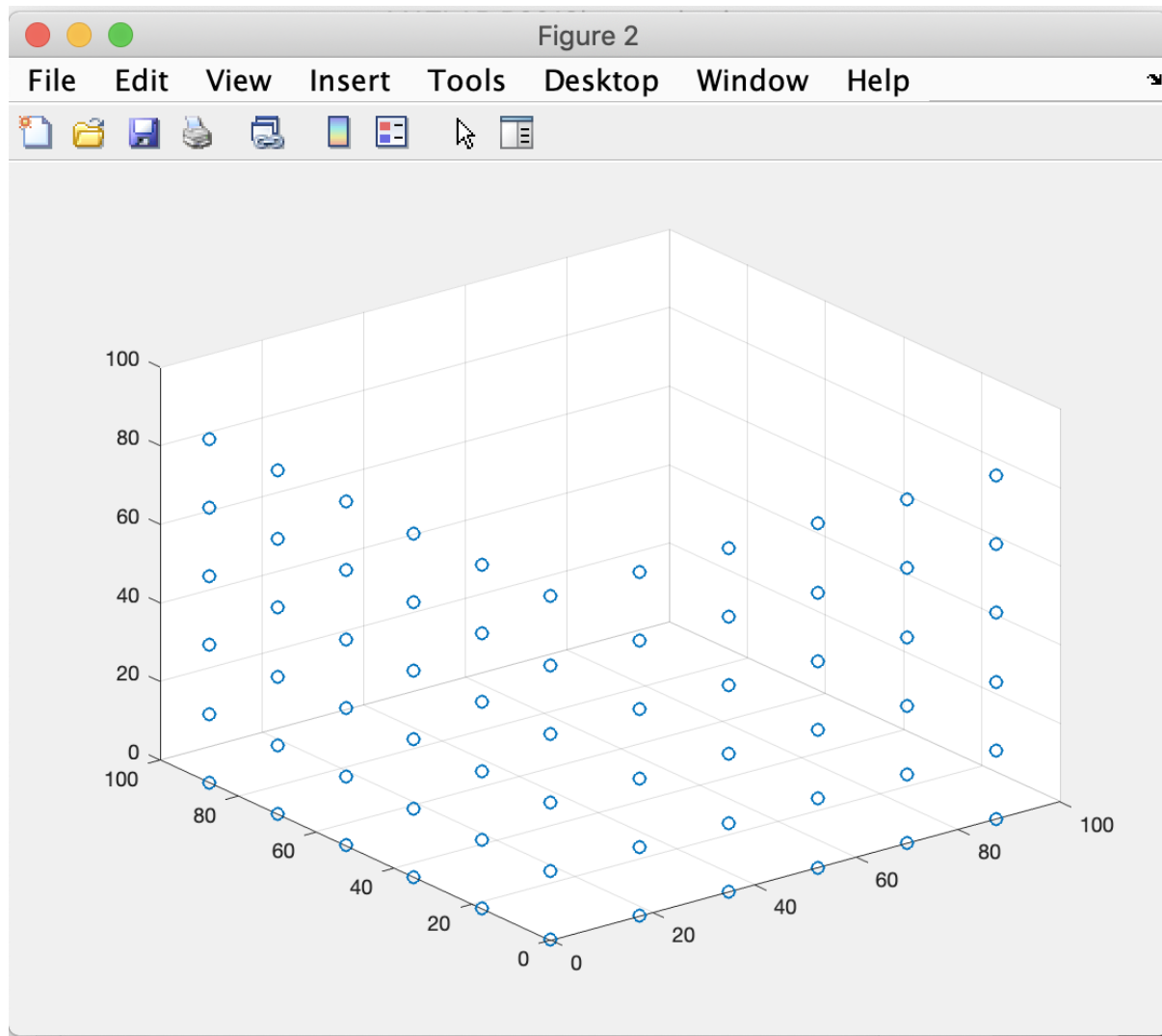


Figure 21.

Note that different than previous method, I used different coordinate system (right handed).

After that I extracted all corner points on a paper and created ix and iy based on the corners.

Calibration Problem

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

Homogenous linear system

$\mathcal{P} \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{P}_1^T & \mathbf{0}^T & -u_1 \mathbf{P}_1^T \\ \mathbf{0}^T & \mathbf{P}_1^T & -v_1 \mathbf{P}_1^T \\ \dots & \dots & \dots \\ \mathbf{P}_n^T & \mathbf{0}^T & -u_n \mathbf{P}_n^T \\ \mathbf{0}^T & \mathbf{P}_n^T & -v_n \mathbf{P}_n^T \end{pmatrix}$

$2n \times 12$

$\mathbf{m} \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{pmatrix}$

12×1

1x4
known
unknown

With this equation we need to create a P matrix: $P(1:2*n,1:12) = 0$;

where n is number of points and 66 in this case. We created 132x12 matrix with 12 unknowns.

If we want to fill the inside of the matrix:
for i=1:2:132

$P(i,1) = wx(j); P(i+1,5) = wx(j);$

$P(i,2) = wy(j); P(i+1,6) = wy(j);$

$P(i,3) = wz(j); P(i+1,7) = wz(j);$

$P(i,4) = 1; P(i+1,8) = 1;$

$P(i,9:12) = P(i,1:4)*-1*ix(j);$

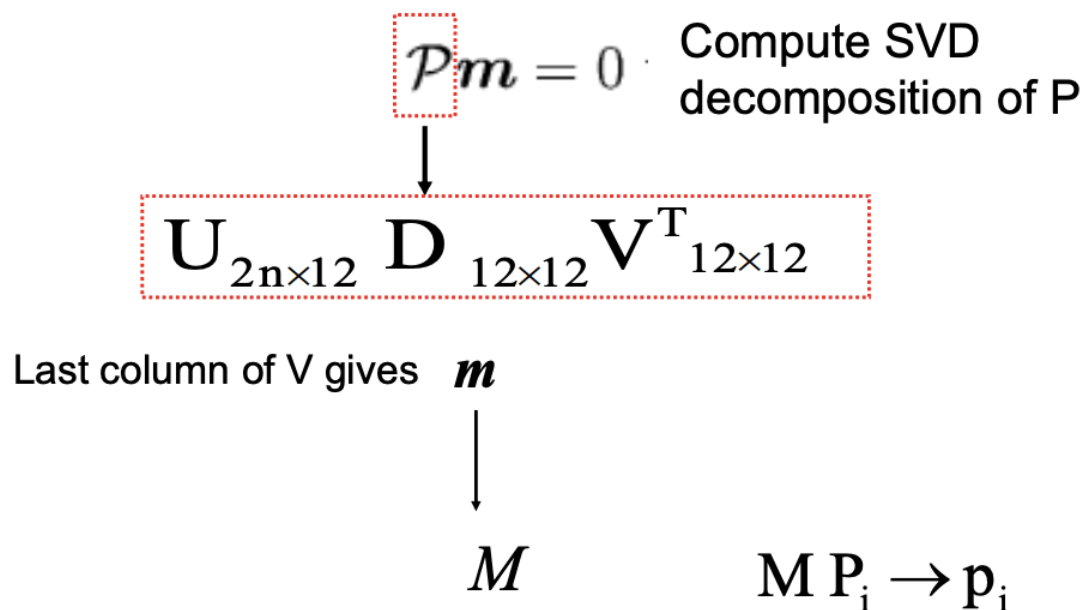
$P(i+1,9:12) = P(i,1:4)*-1*iy(j);$

$j = j+1;$

end

Now we fit World coordinates and pixel coordinates in the matrix. We can use SVD to find 12 unknowns of the linear equation.

General Calibration Problem



```
[U S V] = svd(P);
[min_val, min_index] = min(diag(S(1:12,1:12)));

% m is given by right singular vector of min. singular value
m = V(1:12, min_index);
```

We can normalize m by :

%normalize M to make the norm of third rotation vecto unity

```
norm_31 = norm(m(9:11));
```

```
m_canonical = m / norm_31;
```

```
M(1,1:4) = m_canonical(1:4);
```

```
M(2,1:4) = m_canonical(5:8);
```

```
M(3,1:4) = m_canonical(9:12);
```

We succesfully found M .

```
a1 = M(1,1:3);
```

```
a2 = M(2,1:3);
```

```
a3 = M(3,1:3);
```

```
b = M(1:3, 4);
```

```
r3 = a3;
```

Extracting camera parameters

$$\frac{\mathcal{M}}{\rho} = \begin{pmatrix} \alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T & \alpha t_x - \alpha \cot \theta t_y + u_0 t_z \\ \frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T & \frac{\beta}{\sin \theta} t_y + v_0 t_z \\ \mathbf{r}_3^T & t_z \end{pmatrix} = \mathbf{K} [\mathbf{R} \quad \mathbf{T}]$$

\mathbf{A}

\mathbf{b}

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

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \mathbf{a}_3^T \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Estimated values

Intrinsic

$$\rho = \frac{\pm 1}{\|\mathbf{a}_3\|} \quad \begin{aligned} u_o &= \rho^2 (\mathbf{a}_1 \cdot \mathbf{a}_3) \\ v_o &= \rho^2 (\mathbf{a}_2 \cdot \mathbf{a}_3) \end{aligned}$$

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

We can divide M to two as intrinsic and extrinsic parameters.

```
%compute the intrinsic parameters
```

```
u_0 = a1*a3';
```

```
v_0 = a2*a3';
```

```
cross_a1a3 = cross(a1,a3);
```

```
cross_a2a3 = cross(a2,a3);
```

```
theta = acos (-1*cross_a1a3*cross_a2a3'/(norm(cross_a1a3)*norm(cross_a2a3)));
```

Applying following formulas we can find u0 and v0 and costheta.

Extracting camera parameters

$$\frac{\mathcal{M}}{\rho} = \begin{pmatrix} \boxed{\alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T} & \boxed{\alpha t_x - \alpha \cot \theta t_y + u_0 t_z} \\ \boxed{\frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T} & \boxed{\frac{\beta}{\sin \theta} t_y + v_0 t_z} \\ \mathbf{r}_3^T & t_z \end{pmatrix} = \mathbf{K} [\mathbf{R} \quad \mathbf{T}]$$

\mathbf{A} \mathbf{b}

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \mathbf{a}_3^T \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Estimated values

Intrinsic

$$\begin{aligned} \alpha &= \rho^2 |\mathbf{a}_1 \times \mathbf{a}_3| \sin \theta \\ \beta &= \rho^2 |\mathbf{a}_2 \times \mathbf{a}_3| \sin \theta \end{aligned} \quad \rightarrow \quad \mathbf{f}$$

```
alpha = norm(cross_a1a3) * sin(theta);
```

```
beta = norm(cross_a2a3) * sin(theta);
```

Now we can find alpha and beta.

Extracting camera parameters

$$\frac{\mathcal{M}}{\rho} = \begin{pmatrix} \boxed{\alpha \mathbf{r}_1^T - \alpha \cot \theta \mathbf{r}_2^T + u_0 \mathbf{r}_3^T} & \boxed{\alpha t_x - \alpha \cot \theta t_y + u_0 t_z} \\ \boxed{\frac{\beta}{\sin \theta} \mathbf{r}_2^T + v_0 \mathbf{r}_3^T} & \boxed{\frac{\beta}{\sin \theta} t_y + v_0 t_z} \\ \mathbf{r}_3^T & t_z \end{pmatrix} = \mathbf{K} [\mathbf{R} \quad \mathbf{T}]$$

A
b

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \mathbf{a}_3^T \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Estimated values

Extrinsic

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

$$\mathbf{r}_2 = \mathbf{r}_3 \times \mathbf{r}_1 \quad \mathbf{T} = \rho \mathbf{K}^{-1} \mathbf{b}$$

Applying above equations will result:

%compute 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];
```

Now we can find translation vector and rotation vector as extrinsic parameters:

%translation vector

```
t = inv(K) * b;
```

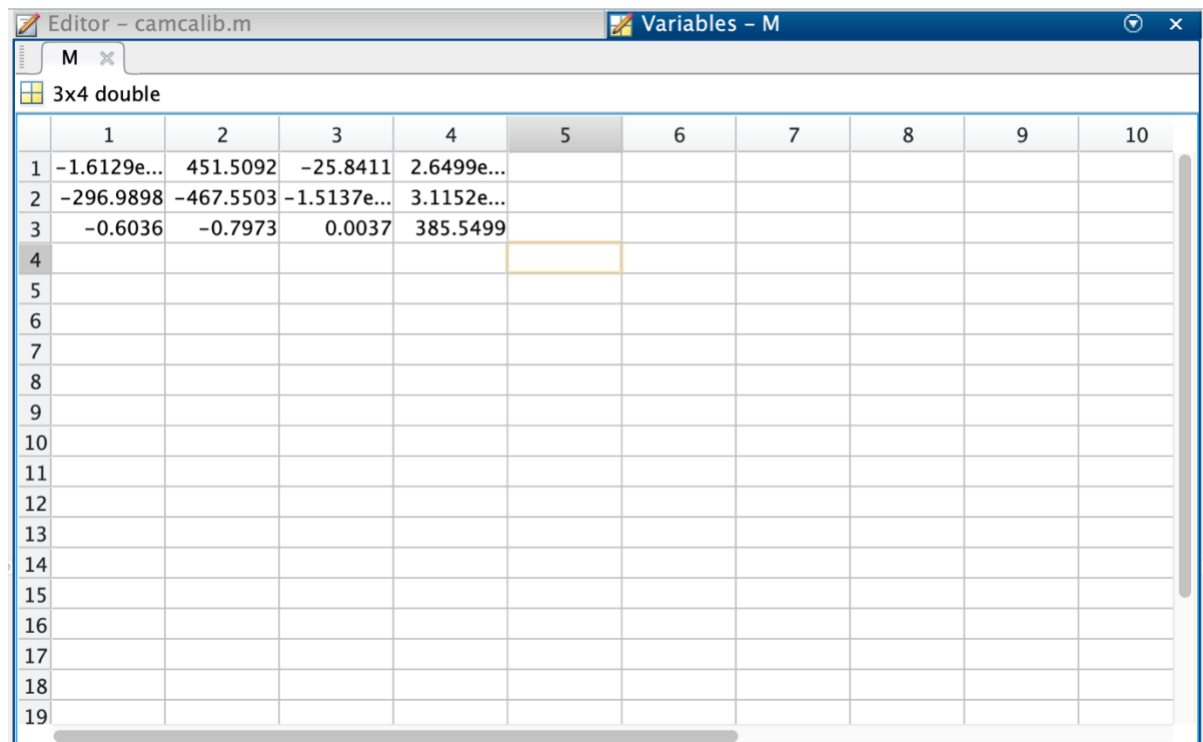
%rotation matrix

```
R(1,1:3) = r1;
```

```
R(2,1:3) = r2;
```

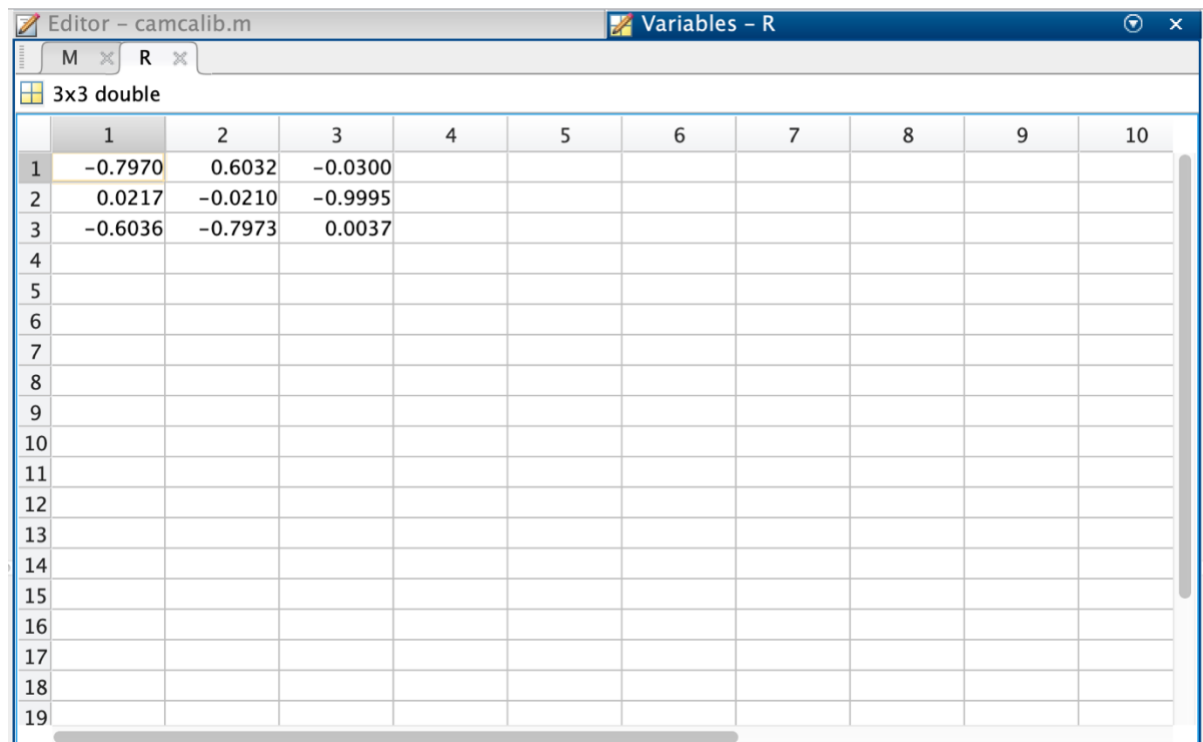
$R(3,1:3) = r3;$

Results :



	1	2	3	4	5	6	7	8	9	10
1	-1.6129e...	451.5092	-25.8411	2.6499e...						
2	-296.9898	-467.5503	-1.5137e...	3.1152e...						
3	-0.6036	-0.7973	0.0037	385.5499						
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										

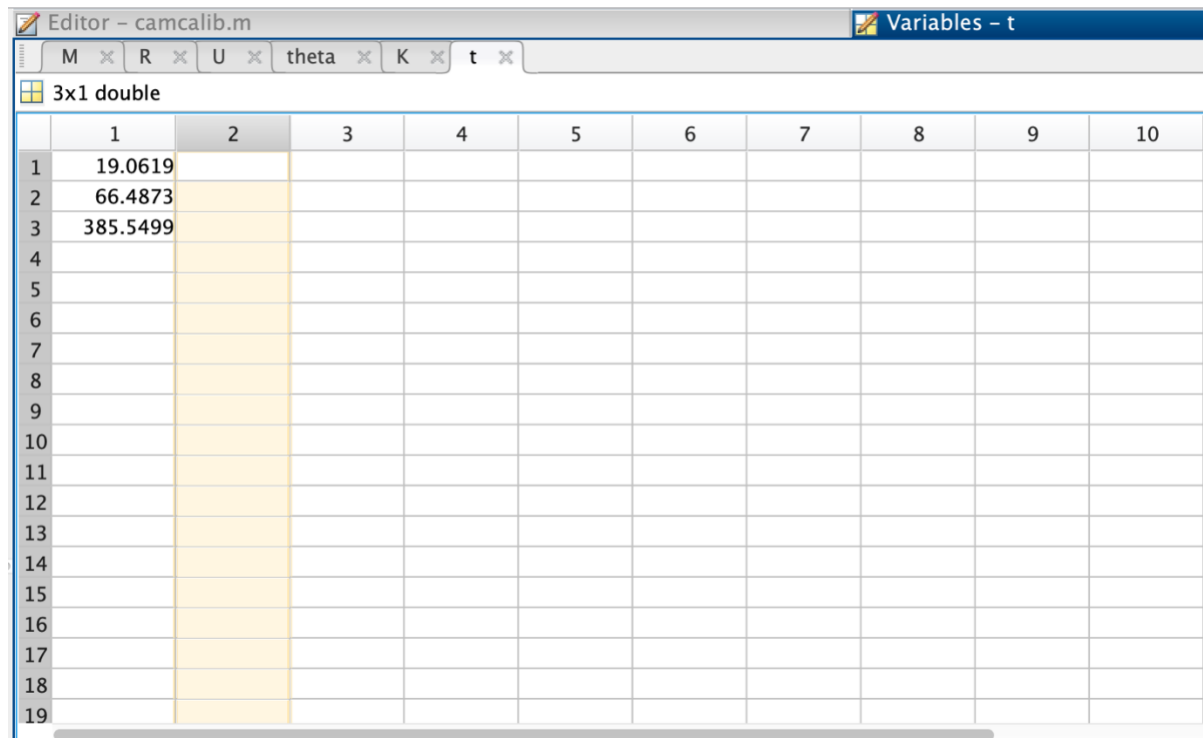
Calibration matrix M



The image shows a MATLAB Editor window titled "Editor - camcalib.m" with a "Variables - R" tab. The variable R is a 3x3 double matrix. The matrix is displayed in a grid with columns 1 through 10 and rows 1 through 19. The first three rows contain the values of the 3x3 matrix, and the rest are empty.

	1	2	3	4	5	6	7	8	9	10
1	-0.7970	0.6032	-0.0300							
2	0.0217	-0.0210	-0.9995							
3	-0.6036	-0.7973	0.0037							
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										

Rotation Matrix R



The image shows a MATLAB Editor window titled "Editor - camcalib.m" with a "Variables - t" tab. The variable t is a 3x1 double matrix. The matrix is displayed in a grid with columns 1 through 10 and rows 1 through 19. The first three rows contain the values of the 3x1 matrix, and the rest are empty.

	1	2	3	4	5	6	7	8	9	10
1	19.0619									
2	66.4873									
3	385.5499									
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										

Transformation Matrix T

Editor - camcalib.m

Variables - K

M R U theta K

3x3 double

	1	2	3	4	5	6	7	8	9
1	1.5586e+03	-18.6132	613.4450						
2	0	1.5163e+03	546.4881						
3	0	0	1						
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									

Camera Matrix K

Workspace	
Name ▲	Value
az	[-290.9898,-407...
a3	[-0.6036,-0.797...
alpha	1.5586e+03
b	[2.6499e+05;3.1...
beta	1.5162e+03
C	166x2 double
ch	3
col	1280
cross_a1a3	[-18.9492,21.50...
cross_a2a3	[-1.2085e+03,91...
i	131
img	1280x1280 uint8
imgGrey	1280x1280 uint8
imgRGB	1280x1280x3 ui...
ix	1x66 double
iy	1x66 double
j	67
K	[1.5586e+03,-18...
m	12x1 double
M	3x4 double
m_canonical	12x1 double
min_index	12
min_val	0.0493
n	66
norm_31	2.4451e-06
P	132x12 double
R	[-0.7970,0.6032,...
r1	[-0.7970,0.6032,...
r2	[0.0217,-0.0210,...
r3	[-0.6036,-0.797...
row	1280
S	132x12 double
t	[19.0619;66.487...
theta	1.5589
U	132x132 double
u_0	613.4450
V	12x12 double
v_0	546.4881
wx	1x66 double
wy	1x66 double
wz	1x66 double

All parameters

We can see that u_0 and u_v are computed as 613 and 546 for 1280x1280 image. The real center is 640x640 and the difference can be counted as offset. We can see that transformation matrix's norm is around 40 cm. Which is close to expected result.

Note that previous method computed norm different because we were working with different planes. All other related parameters can be found by looking parameter image.

Comparision

We implemented different methods and achieved different but close results on the same 3D object. The main difference is due to 2D object and moving camera compared with single image of 3D object which has known world coordinates. Personally I liked first method better because It allows user more flexibility. The second method is the basic idea and probably will work better if we know all corner points of the image. Also, this corners should be marked with little amount of error. I think one can compute the intrinsic and extrinsic parameters of the camera on an image with using very good corner detector (minimum misclassified corners) then applying SVD method. It will result very close approximation to real numbers but It will be time costly because of extracting all corners and matching them in world coordinate frame. Disadvantage of the SVD method is, computation of every coordinate point for different pictures. But with the first method, we can easily compute a new image's extrinsic parameters by looking previous results. Maybe we can try to use different decomposition methods such as QR decomposition to test the result and compare it.

Codes and Computations

```
img = imread('img.jpg');
```

```

imgRGB = img;
[row, col, ch] = size(img);
if(ch==3)
    img = rgb2gray(img);
end

imgGrey = img;

pause(1);
figure, imshow(imgGrey), hold on
C = corner(imgGrey);
plot(C(:,1),C(:,2),'ro', 'MarkerSize', 8)

%world coordinates in mms.
%z coordinates
wz(1:6) = 5*17.5; wz(37:41) = 5*17.5;
wz(7:12) = 4*17.5; wz(42:46) = 4*17.5;
wz(13:18) = 3*17.5; wz(47:51) = 3*17.5;
wz(19:24) = 2*17.5; wz(52:56) = 2*17.5;
wz(25:30) = 1*17.5; wz(57:61) = 1*17.5;
wz(31:36) = 0*17.5; wz(62:66) = 0*17.5;

%x coordinates
wx(1:66) = 0;
wx(1:6:36) = 5*17.5;
wx(2:6:36) = 4*17.5;
wx(3:6:36) = 3*17.5;
wx(4:6:36) = 2*17.5;
wx(5:6:36) = 1*17.5;
wx(6:6:36) = 0*17.5;
%y coordinates
wy(1:66) = 0;
wy(37:5:66) = 1*17.5;
wy(38:5:66) = 2*17.5;
wy(39:5:66) = 3*17.5;
wy(40:5:66) = 4*17.5;
wy(41:5:66) = 5*17.5;

ix= [366 437 504 566 623 677 368 438 504 566 623 679 369 440 505 566 624
680 371 441 506 566 624 680 373 443 507 568 625 680 372 445 507 570 626 690
734 785 839 899 904 734 785 839 897 962 735 786 840 899 964 735 786 840 899
963 736 786 840 899 964 735 787 839 898 960] ;
iy= [455 458 460 463 464 465 538 537 536 536 536 536 619 615 613 609 606
603 700 693 687 682 677 673 780 771 762 752 746 739 860 847 836 825 816 800
460 455 448 442 434 532 529 526 523 520 603 603 603 604 605 673 677 681 684
689 744 750 757 765 774 811 823 833 845 867] ;

n = 66;

figure;

```

```

scatter3(wx,wy,wz);

P(1:2*n,1:12) = 0;
j=1;
%construct matrix P
for i=1:2:132
P(i,1) = wx(j); P(i+1,5) = wx(j);
P(i,2) = wy(j); P(i+1,6) = wy(j);
P(i,3) = wz(j); P(i+1,7) = wz(j);
P(i,4) = 1; P(i+1,8) = 1;
P(i,9:12) = P(i,1:4)*-1*ix(j);
P(i+1,9:12) = P(i,1:4)*-1*iy(j);
j = j+1;
end

%Perform SVD of P

[U S V] = svd(P);
[min_val, min_index] = min(diag(S(1:12,1:12)));

%m is given by right singular vector of min. singular value
m = V(1:12, min_index);

%normalize M to make the norm of third rotation vecto unity
norm_3l = norm(m(9:11));
m_canonical = m / norm_3l;
M(1,1:4) = m_canonical(1:4);
M(2,1:4) = m_canonical(5:8);
M(3,1:4) = m_canonical(9:12);


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

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

%compute 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];

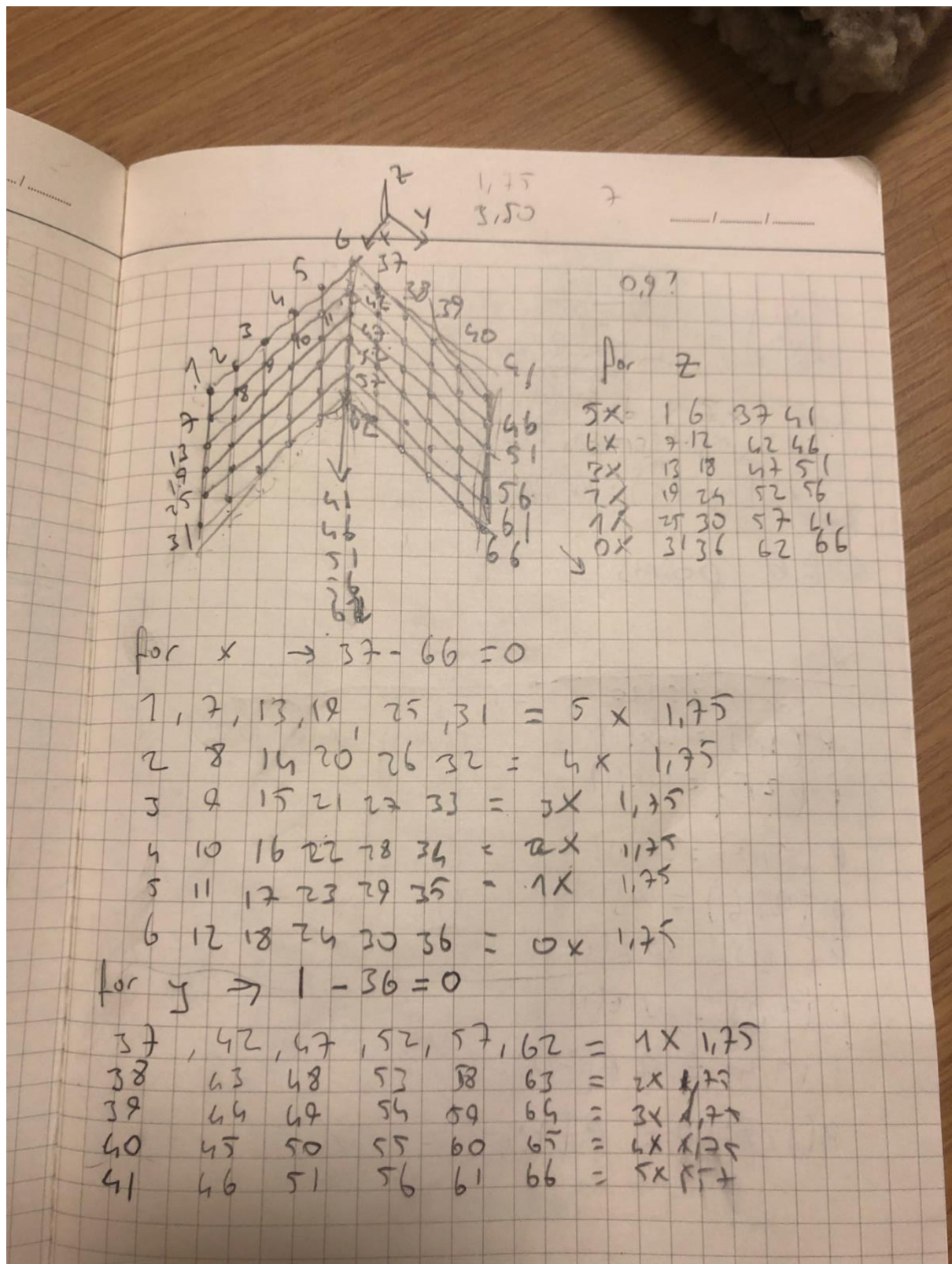
%translation vector
t = inv(K) * b;

```



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

%print all the values
%intrinsic
u_0;
v_0;
alpha;
beta;
%extrinsic
R;
t;
```



$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} \frac{m_1 p_i}{m_3 p_i} \\ \frac{m_2 p_i}{m_3 p_i} \end{bmatrix}$$

$$u_i: (m_3 \cdot p_i) - m_1 p_i = 0$$

$$v_i: (m_3 \cdot p_i) - m_2 p_i = 0$$

66 points

✓

for $i = 1:120$

$i = 1$

$$p(1,1) = w_x(1)$$

$$p(2,5) = w_x(1)$$

$$p(1,2) = w_y(1)$$

$$p(2,6) = w_y(1)$$

$$p(1,3) = w_z(1)$$

$$p(2,7) = w_z(1)$$

$$p(1,4) = 1 \quad p(2,8) = 1$$

$$p(1,9:12) = p(1,4) \cdot -1 \cdot i_x(1)$$

$$\begin{bmatrix} w_x(1) & w_x(1) & w_x(1) \\ - & - & - \\ - & - & - \end{bmatrix} \cdot 1$$

1	3
2	4
3	5
4	5
5	6
6	6
7	36
8	4
9	5
10	5
11	6
12	6
13	3
14	4
15	5
16	5
17	6
18	6
19	3
20	4
21	5
22	5
23	6
24	1
25	1
26	1
27	1
28	1
29	1
30	1

