CAMERA CALIBRATION

DECEMBER 12, 2021

Submitted to: Prof. Dr. Mustafa Ünel

Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Computer Vision (EE 417)

TABLE OF CONTENTS

INTRODUCTION

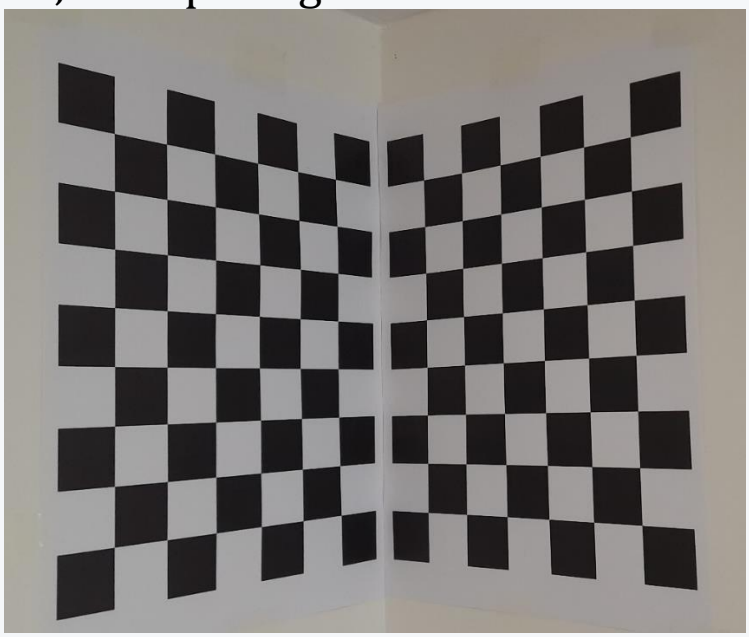
A camera, in the most basic terms, is a sensor that takes images from a 3D scene. There are myriads of camera types available today; however, they all rely on the basic principle. Mathematically, a camera could be modeled as a projective camera to get an idea of the processes involved in image formation. In modern cameras, contrary to the pinhole applied in projective cameras, lenses are used to focus images on the image plane. This brings us to the notion of perspective cameras.

This assignment explores two approaches to calibrating a camera. Firstly, the camera projection matrix is used. This procedure involves manually selecting corners to be used in solving the homogenous equation involving world coordinates and corresponding image coordinates. The approach applied was singular value decomposition which was used to estimate the projection matrix and then obtain the extrinsic and intrinsic camera parameters. In the second method, a series of images of a single calibration pattern were taken from different viewpoints. These images severed as inputs to the MATLAB camera calibration app, and the required parameters were obtained. Finally, a comparison of the above two methods was done.

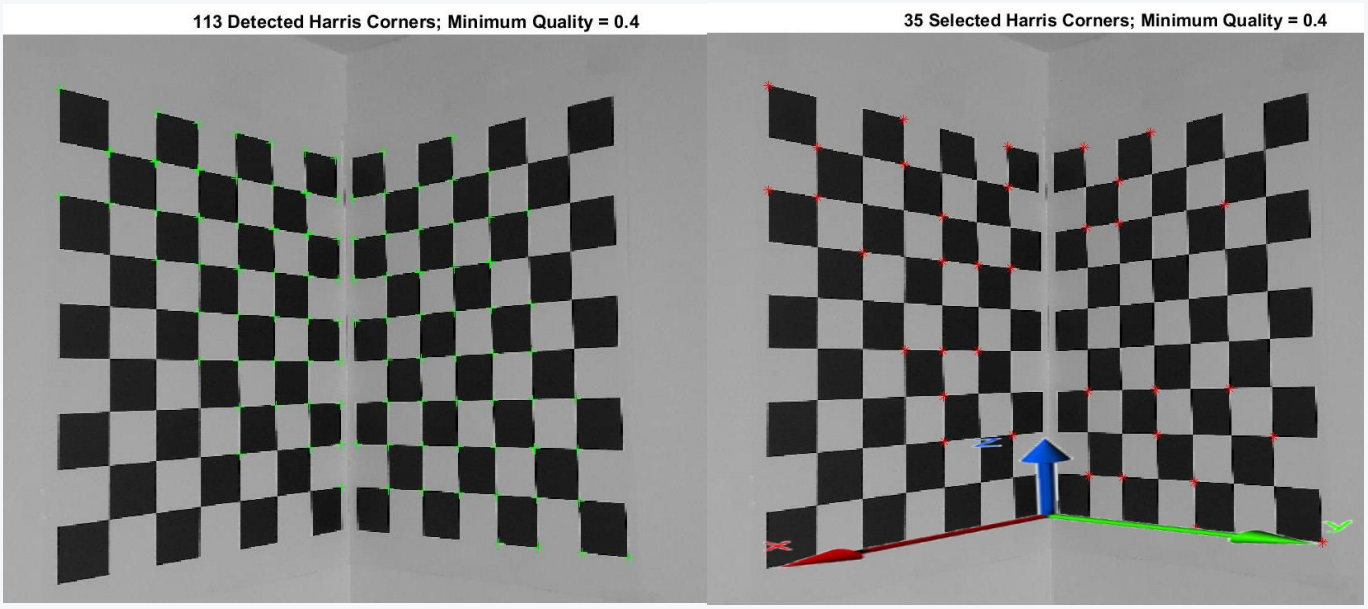
Camera calibration entails estimating the intrinsic and extrinsic parameters of a camera. Intrinsic or internal parameters include the effective focal lengths, α and β, which are scaled versions of the camera's focal length, the coordinates of the principal point u_0 and v_0, the skew parameter, s, and the radial distortion κ_1.

CALIBRATION USING THE CAMERA PROJECTION MATRIX

- A 2D camera calibration rig was place on two walls, and using a phone camera, a crisp image was taken.



- In Matlab, the image was converted to grayscale, and the "detectHarrisFeatures()" function was used to extract corners with a minimum quality of 0.4.
- 35 corner points were chosen being sure to avoid any points lying on the intersection curve of two quadric surfaces.
- Also, a world coordinate frame was attached as follows:



- The size of each checkerboard square is 39.5mm.
- The distance from origin to the corner of the first square along the x- or y-axis is 9.5mm.

Camera Calibration

- The image coordinates and world coordinates were converted to homogeneous coordinates by simply appending ones in the last column.

```
H_ones = ones(N,1);
H_corners = [corners H_ones];
H_world = [world_coordinate H_ones];
```

- Two equations were obtained for each of the 35 selected corner points and used to form the P-matrix.

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

known, unknown

$$\mathcal{P}\boldsymbol{m}=0$$

Homogenous linear system

$$\mathcal{P} \overset{\text{def}}{=} \begin{pmatrix} \boldsymbol{P}_1^T & \mathbf{0}^T & -u_1\boldsymbol{P}_1^T \\ \mathbf{0}^T & \boldsymbol{P}_1^T & -v_1\boldsymbol{P}_1^T \\ \cdots & \cdots & \cdots \\ \boldsymbol{P}_n^T & \mathbf{0}^T & -u_n\boldsymbol{P}_n^T \\ \mathbf{0}^T & \boldsymbol{P}_n^T & -v_n\boldsymbol{P}_n^T \end{pmatrix}_{2n \times 12}$$

1x4

$$\boldsymbol{m} \overset{\text{def}}{=} \begin{pmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{pmatrix}_{12\times 1}$$

4x1

- The homogenous equation was solved using singular value decomposition and the estimates of the components of m were obtained from the last column orthogonal matrix on the right-hand side.

$$\mathcal{P}\boldsymbol{m}=0$$

Compute SVD decomposition of P

$$\mathbf{U}_{2n\times 12}\ \mathbf{D}_{12\times 12}\ \mathbf{V}^{\mathrm{T}}{}_{12\times 12}$$

- The column is reshaped to yield the 3x4 projective matrix.
- Now, the camera parameters could be extracted using the QR factorization function in MATLAB; however, I opted to use algebraic and matrix formulars as shown below:

$$A = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \mathbf{a}_3^T \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad K = \begin{bmatrix} \alpha & -\alpha\cot\theta & u_o \\ 0 & \frac{\beta}{\sin\theta} & v_o \\ 0 & 0 & 1 \end{bmatrix}$$

- A corresponds to the 3x3 matrix of the first three columns of the 3x4 matrix obtained in the last step, while b is the 4th column.

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

$$\alpha = \rho^2 |\mathbf{a}_1 \times \mathbf{a}_3| \sin\theta \quad \rightarrow \text{f}$$

$$\beta = \rho^2 |\mathbf{a}_2 \times \mathbf{a}_3| \sin\theta$$

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

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

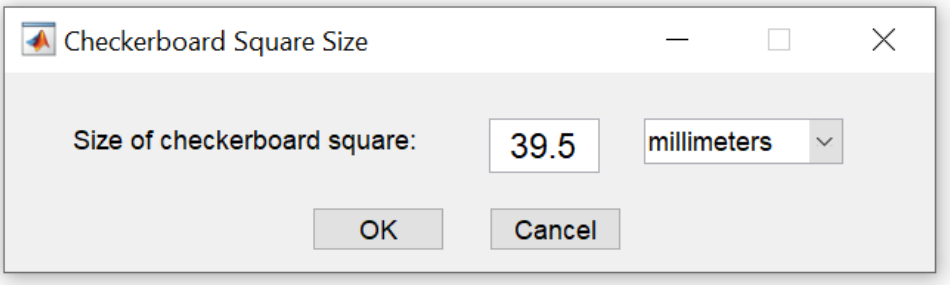
$$\mathbf{r}_2 = \mathbf{r}_3 \times \mathbf{r}_1 \qquad T = \rho K^{-1}\mathbf{b}$$

Note: The MATLAB code is found in the appendix.

- The results are included in the results section.

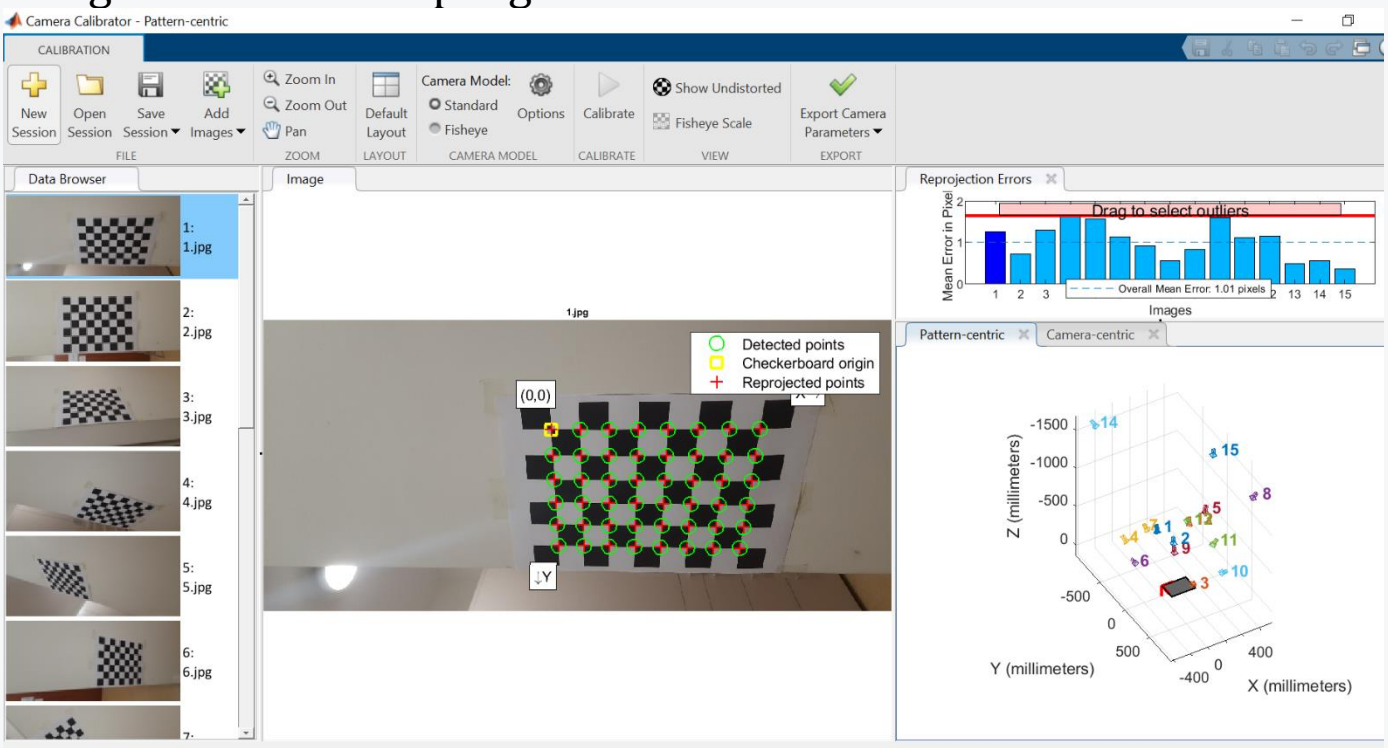
CALIBRATION USING THE MATLAB CALIBRATION TOOLBOX

For this section, 15 images (attached in the appendix) of a single plane calibration pattern were taken (from different viewpoints) and used as the inputs for the calibration process. The size of each square (39.5mm) was also defined.

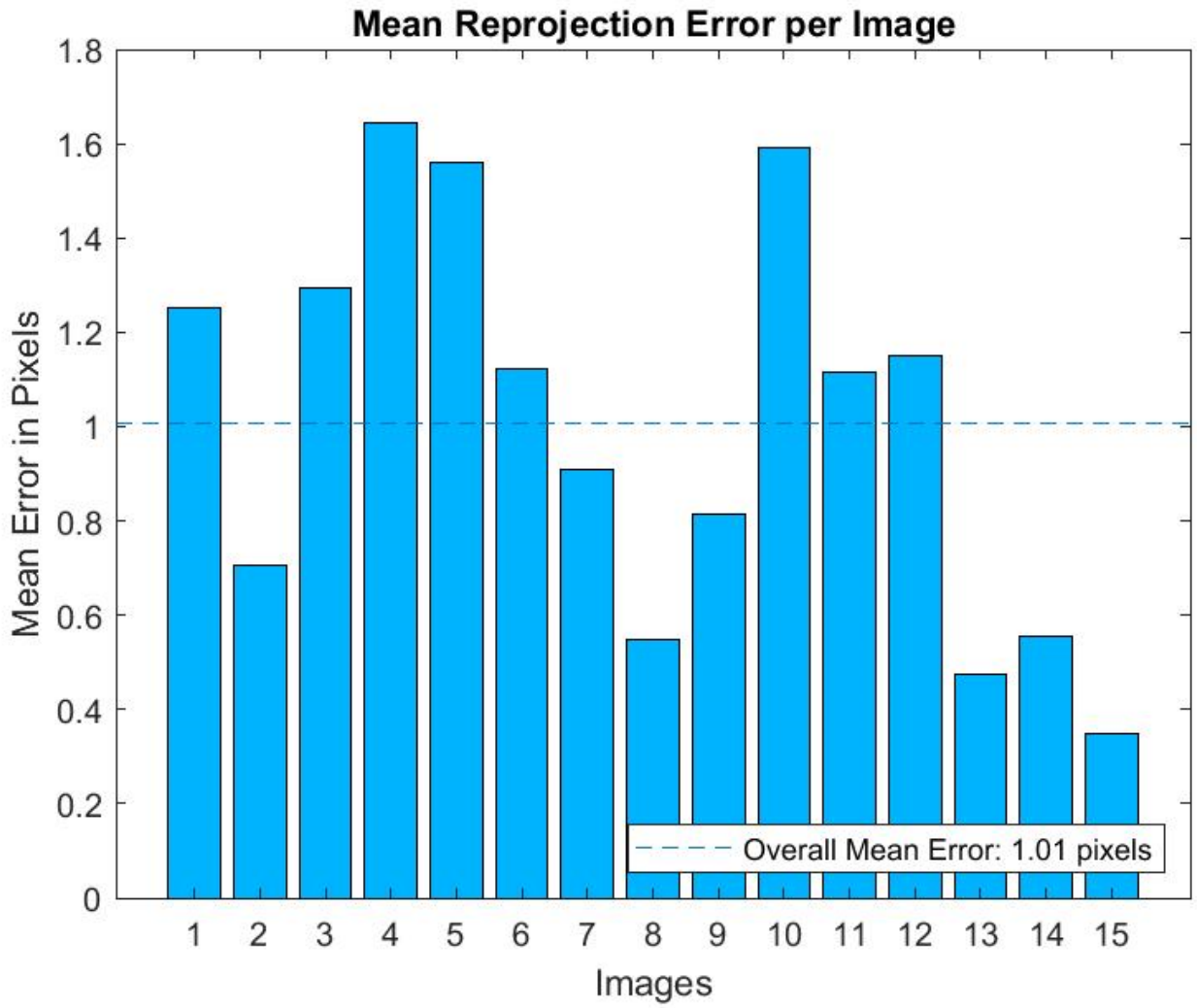


picked four images obtained for each data set.

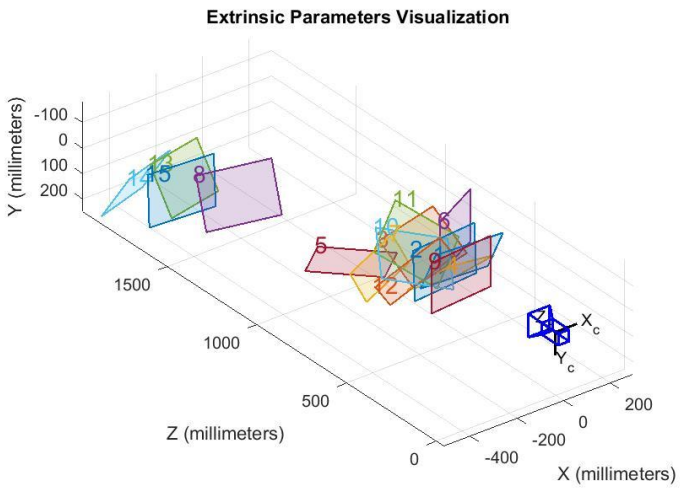
- The origin and axes were automatically included by the algorithm at the top-right corner.

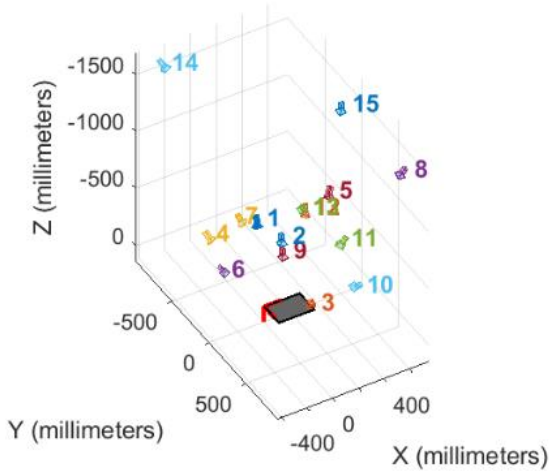


- A mean reprojection error of around 1.01 was observed. This was mostly due to images taken from extremely tight angles.



- Extrinsic Parameter Visualization





Note: The results are included in the next section.

RESULTS

Using the Camera Projection Equation

The intrinsic parameter matrix, the rotation matrix, and the translation vector:

K — 3x3 double

	1	2	3
1	3.4273e+03	0.5946	2.4922e+03
2	0	3.3329e+03	1.0437e+03
3	0	0	1

R — 3x3 double

	1	2	3
1	-0.6105	0.7917	-0.0232
2	0.0449	0.0639	0.9969
3	0.7908	0.6076	-0.0746

T — 3x1 double

	1
1	106.3040
2	-196.7081
3	-872.4246

From the following:

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

Theta = 1.570969824045283rad ~ 90 degrees. Therefore, from the above result, the camera has no skew. This is of course expected because the advancement in manufacturing processes has greatly reduced the likelihood of produced skewed lenses.

Alpha, u_0 and v_0 can be directly read from the above image.

Using the Camera Calibration Toolbox

calibrationSession | calibrationSession.CameraParameters

calibrationSession.CameraParameters

Property	Value
ImageSize	[2136,4624]
RadialDistortion	[0.1827,-0.4827]
TangentialDistortion	[0,0]
WorldPoints	*48x2 double*
WorldUnits	'millimeters'
EstimateSkew	1
NumRadialDistortionCoefficients	2
EstimateTangentialDistortion	0
TranslationVectors	*15x3 double*
ReprojectionErrors	*48x2x15 double*
RotationVectors	*15x3 double*
NumPatterns	15
Intrinsics	*1x1 cameraIntrinsics*
IntrinsicMatrix	[3.2738e+03,0,0;3.2062,3.2788e+03,0;2.3633e+03,1.0171e+03,1]
FocalLength	[3.2738e+03,3.2788e+03]
PrincipalPoint	[2.3633e+03,1.0171e+03]
Skew	3.2062
MeanReprojectionError	1.0055
ReprojectedPoints	*48x2x15 double*
RotationMatrices	*3x3x15 double*

Intrinsic parameter matrix (transpose) – K

calibrationSession.CameraParameters.Intrinsics.IntrinsicMatrix

	1	2	3	4	5
1	3.2738e+03	0	0		
2	3.2062	3.2788e+03	0		
3	2.3633e+03	1.0171e+03	1		

calibrationSession.CameraParameters.Intrinsics

Property	Value
FocalLength	[3.2738e+03,3.2788e+03]
PrincipalPoint	[2.3633e+03,1.0171e+03]
ImageSize	[2136,4624]
RadialDistortion	[0.1827,-0.4827]
TangentialDistortion	[0,0]
Skew	3.2062
IntrinsicMatrix	[3.2738e+03,0,0;3.2062,3.2788e+03,0;2.3633e+03,1.0171e+03,1]

Rotation matrices for each of the 15 input images:

```
val(:,:,1) =

    0.9983    0.0026   -0.0585
    0.0250    0.8849    0.4652
    0.0530   -0.4658    0.8833

val(:,:,2) =

    0.9978   -0.0215    0.0622
    0.0230    0.9994   -0.0242
   -0.0617    0.0256    0.9978

val(:,:,3) =

    0.9802   -0.0568    0.1895
    0.1791    0.6619   -0.7279
   -0.0841    0.7474    0.6590

val(:,:,4) =

    0.9154    0.2883    0.2809
   -0.4023    0.6315    0.6629
    0.0137   -0.7198    0.6940

val(:,:,5) =

    0.8591    0.1665   -0.4840
    0.1647    0.8055    0.5693
    0.4847   -0.5688    0.6645

val(:,:,6) =

    0.8605    0.0640    0.5054
   -0.0420    0.9976   -0.0548
   -0.5077    0.0259    0.8611

val(:,:,7) =

    0.5375    0.7599    0.3656
   -0.8351    0.5399    0.1057
   -0.1170   -0.3621    0.9248

val(:,:,8) =

    0.9614   -0.0355   -0.2729
   -0.0782    0.9155   -0.3947
    0.2638    0.4008    0.8774

val(:,:,9) =

    0.9939    0.0152    0.1091
   -0.0137    0.9998   -0.0138
   -0.1093    0.0122    0.9939

val(:,:,10) =

    0.8952    0.3422   -0.2856
   -0.4345    0.5267   -0.7306
   -0.0996    0.7781    0.6202

val(:,:,11) =

    0.7109    0.6073   -0.3548
   -0.7019    0.6445   -0.3031
    0.0446    0.4645    0.8844

val(:,:,12) =

    0.8052   -0.5373   -0.2509
    0.5930    0.7258    0.3488
   -0.0053   -0.4297    0.9030

val(:,:,13) =

    0.9514    0.0819    0.2970
    0.0827    0.8608   -0.5022
   -0.2968    0.5024    0.8121

val(:,:,14) =

    0.9241    0.0781    0.3740
   -0.2554    0.8543    0.4527
   -0.2842   -0.5139    0.8094

val(:,:,15) =

    0.9989    0.0308   -0.0355
   -0.0341    0.9948   -0.0956
    0.0324    0.0967    0.9948
```

1	2	3
-45.5261	-39.0299	609.1780
-123.7166	-71.7555	635.0372
-158.8959	-58.8967	778.7779
18.7380	31.8357	618.6517
-355.0261	-64.1372	865.5726
56.6870	-85.5199	717.4731
-141.2330	-100.4528	728.4252
-316.3587	5.0733	1.5962e+03
-142.0396	-90.9151	505.8231
-171.5237	-113.8838	780.5781
-75.7444	-180.7396	795.7903
-243.9200	53.1847	689.0628
-342.4132	55.6295	1.8186e+03
-461.8974	52.2166	1.7793e+03
-455.9643	-6.7599	1.6846e+03

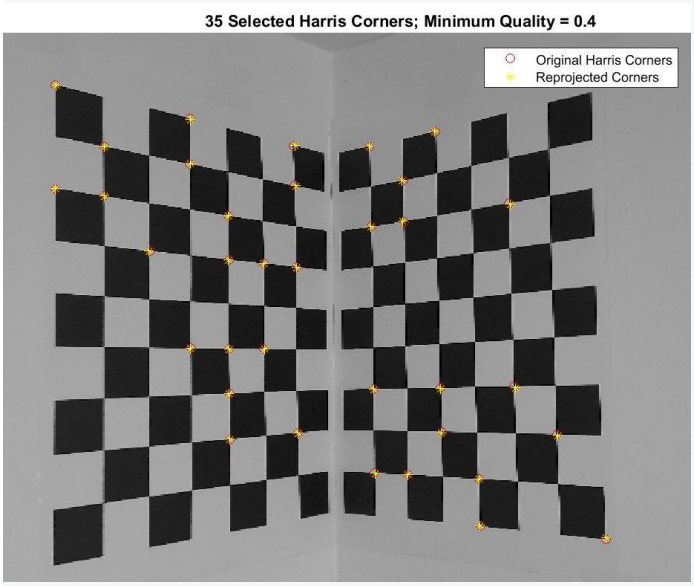
DISCUSSION

Comparison of Results

- Firstly, we notice that the camera parameters obtained from each method are not exactly the same.
- Method 1 – $[u_0,v_0]$ = [2.492175868365855e+03 , 1.043706879038802e+03]
- Method 2 – $[u_0,v_0]$ = [2.3633e+03 , 1.0171e+03]
- The discrepancy here is not much.
- For the skewness factor, method 2 returns a factor of about 3.2 while method 1 returns a factor of about 0.5. I believe this is due to the multiple viewpoints used for the 15 images in method 2.
- The alphas and betas in both cases are also close in value.

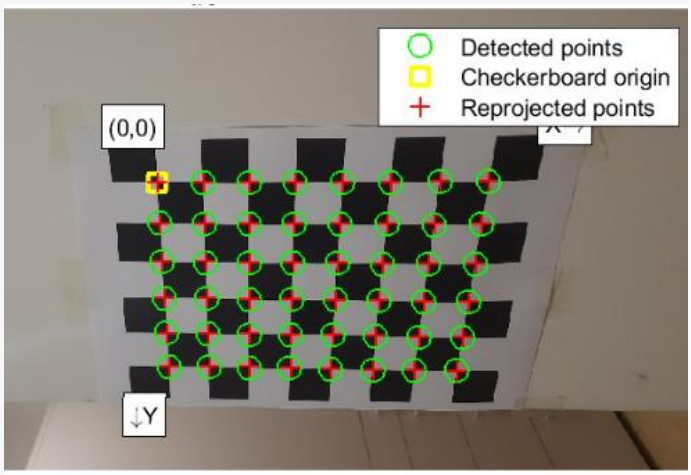
Comparison of Results after Reprojection

Method 1:



I included a short code to check whether points obtained using the projection matrix will match with the originally detected Harris corners. The image on the left was the output:
Notice how the results generally coincide well *except for a number of non-overlapping detections*.

Method 2:
The algorithm in the toolbox automatically reprojects the corners to produce the following:



Comments

- The precision of calibration depends on how accurately the world and image points are located.
- Error propagation could be a culprit for discrepancies in the results obtained.
- In method 1, rho was assumed to be positive. Further optimization could be performed to determine the actual of this parameter.

$$\rho = \frac{\pm 1}{|\mathbf{a}_3|}$$

- Method 2 is a much more convenient approach to calibrating cameras. It is also more robust and advantageous given that there are multiple tools to calculate a plethora of results and optimize them.
- Also, while method 1 requires at least two planes for calibration, method 2 requires only one plane.
- Method 2 would be preferred because detailed results can be obtained in real-time.
- For method 1, it might be better to use a subpixel accurate corner detection method like the intersection of Hough lines.
- In this work, a high-end mobile phone camera was used and a skew angle of 90 degrees was found. This proves that modern cameras usually come without skew owning to the high precision manufacturing processes they undergo.
- From the reprojection results, we notice that there are some reprojected corners that don't coincide with the original corners for method 1. However, with method, there's 100% overlapping of original and reprojected corners.

Better Methods for Corner Extraction

- As found in the literature, we can apply certain methods to already existing corner detectors (like Harris, Kanade-Tomasi, etc.) to achieve sub-pixel accuracy.
- **These methods can be classified into interpolation method, classification method, and fitting method.**
- *The fitting method is widely used in the field of digital image processing. There are various kinds of surface fitting, such as Gaussian surface fitting, polynomial fitting method, ellipse fitting method. Gaussian surface fitting algorithm is particularly common in achieving positioning precision at level of the sub-pixel.*

REFERENCES

Reinhard Klette. 2014. Concise Computer Vision: An Introduction into Theory and Algorithms. Springer Publishing Company, Incorporated.

http://www.vision.caltech.edu/bouguetj/calib_doc/

https://www.mathworks.com/help/vision/ref/detectharrisfeatures.html

https://www.mathworks.com/matlabcentral/answers/26796-how-do-i-add-a-column-to-a-matrix

https://www.youtube.com/watch?v=3NcQbZu6xt8

https://www.youtube.com/watch?v=x6YIwoQBBxA&t=66s

https://www.cse.unr.edu/~bebis/CS791E/Notes/CameraCalibration.pdf

https://github.com/begumcig/Camera-Calibration

https://github.com/alaattinyilmaz/camera-calibration/blob/master/CameraCalibration.pdf

https://github.com/jahnavi-chowdary/CV-Assignment-2/blob/master/DLT.m

APPENDIX

CAMERA CALIBRATION

Assignment2.m

```
clear all; close all; clc;

% Read the image to be preprocessed
im = imread('CalibrationRig.jpg');

% The row, column, and channels of the image are obtained
along with the cardinality of the image.
[r, c, ch] = size(im);
Card = r*c;

img = im;

% This is added in case the image introduced is an RGB
image.
% It functions to convert it to a gray-scale image.
if ch == 3
    img = rgb2gray(img);
end

% Find the Harris corners
C_H = detectHarrisFeatures(img, 'MinQuality', 0.4);
imshow(img)
hold on
% Use the following to show all detected corners
% plot(C_H);
% title('113 Detected Harris Corners; Minimum Quality =
0.4')

N = 35; % Number of selected corners

% Coordinates of the selected corner points from the image
corners = [1015.07 197.849;
           1015.52 586.523;
           1670.91 1526.18;
           2612.77 1850.76;
           3084.11 1901.32;
           2900.62 1508.57;
           2738.39 1327.41;
```

```
            2443.12 371.954;
            2194.47 428.608;
            2202.66 727.34;
            1920.74 879.125;
            1798.35 865.796;
            1370.31 820.117;
            1665.87 1186.28;
            1523.53 1183.98;
            1669.33 1354.58;
            1667.23 852.664;
            1519.28 322.695;
            1197.54 428.103;
            1197.16 613.537;
            2323.96 709.427;
            2339.14 1659.43;
            2212.93 1652.87;
            2210.28 1337.98;
            2719.05 641.073;
            2323.27 553.502;
            1908.45 425.628;
            1926.47 1503.11;
            2461.53 1332.38;
            1520.5 492.202;
            1911.85 573.718;
            2605.73 1675.47;
            2466.27 1498.64;
            1665.49 687.469;
            1798    1186.48];

% Corner points in world coordinates
world_coordinate = [286.5   0   355.5;
                    286.5   0   276.5;
                    128.5   0     79;
                     0     168    0;
                     0     286.5  0;
                     0     247    79;
                     0     207.5 118.5;
                     0     128.5 355.5;
                     0     49.5  355.5;
                     0     49.5  276.5;
                    49.5    0    237;
```

```
89      0    237;
207.5   0    237;
128.5   0    158;
168     0    158;
128.5   0    118.5;
128.5   0    237;
168     0    355.5;
247     0    316;
247     0    276.5;
 0     89    276.5;
 0     89    39.5;
 0     49.5  39.5;
 0     49.5  118.5;
 0     207.5 276.5;
 0     89    316;
49.5    0    355.5;
49.5    0    79;
 0     128.5 118.5;
168     0    316;
49.5    0    316;
 0     168   39.5;
 0     128.5 79;
128.5   0    276.5;
89      0    158];

% Convert the Image Corner Points world coordinates to
homogeneous
% coordinates by appending an Nx1 column of ones.
H_ones = ones(N,1);
H_corners = [corners H_ones];
H_world = [world_coordinate H_ones];

plot(H_corners(:,1),H_corners(:,2),'ro');
title('35 Selected Harris Corners; Minimum Quality = 0.4')

% Create the 2nx12 P-matrix
P = zeros(2*N, 12);

j = 1;
zero_T = zeros(1,4);
for i = 1:2:2*N
```

```
    P(i,:) =   [H_world(j,:)    zero_T       -
H_corners(j,1)*H_world(j,:)];
    P(i+1,:) = [zero_T       H_world(j,:)    -
H_corners(j,2)*H_world(j,:)];
    j = j+1;
end

% Compute the Singular Value Decomposition of P
[U, D, V] = svd(P);

estimated_m = V(:,12);
m1 = transpose(estimated_m(1:4,1));
m2 = transpose(estimated_m(5:8,1));
m3 = transpose(estimated_m(9:12,1));

projection_matrix = [m1; m2; m3];

submatrix_m = projection_matrix(:,1:3);

a1 = submatrix_m(1,1:3);
a2 = submatrix_m(2,1:3);
a3 = submatrix_m(3,1:3);
b = projection_matrix(1:3,4);

% Find the scaling factor since our estimation was only up
to scale.
rho = 1/norm(a3);

% Find the intrinsic parameters
u0 = rho^2*dot(a1,a3);
v0 = rho^2*dot(a2,a3);
theta = acos(dot(cross(a1,a3),transpose(cross(a2,a3))) /
norm(cross(a1,a3)*norm(cross(a2,a3))));
alpha = rho^2*norm(cross(a1,a3))*sin(theta);
beta = rho^2*norm(cross(a2,a3))*sin(theta);

% Intrinsic parameter matrix, K
K = [alpha  -1*alpha*cot(theta) u0;
     0       beta/sin(theta)    v0;
     0              0           1];

% Find the extrinsic parameters
r1 = (cross(a2,a3)) / norm(cross(a2,a3));
```

```
r3 = a3/norm(a3);
r2 = cross(r3,r1);
% Rotation Matrix
R = [r1; r2; r3];
% Translation Vector
T = rho*(K\b);

% These final lines of code are used to obtain the
reprojected corner for
% the sake comparison
% check= projection_matrix*H_world';
% x = check(1,:)./check(3,:);
% y = check(2,:)./check(3,:);

% hold on;
% plot(x,y,'y*');
% legend('Original Harris Corners', 'Reprojected Corners')
```

IMAGES FOR METHOD 2