cs512 Assignment 4

Chandni Patel
Department of Computer Science
Illinois Institute of Technology

November 28, 2020

Abstract

This report contains description of the feature extraction problem, the algorithms used for solving the Non-planar Camera Calibration problem, and analysis of the results obtained for the programming part of Assignment 4.

1 Problem Statement

The goal is to write a program to perform Camera Calibration with following requirements:

1. **Feature Extraction**: Extract feature points from calibration target and save them in a file by two ways:
    - Extract chessboard features using OpenCV and save 2D image points detected along with corresponding 3D world points automatically.
    - Interactively mark points on the image and save 2D image points marked in a file. Then enter 3D world points manually.
2. **Non-Planar Camera Calibration**: Perform Non-planar camera calibration as follows:
    - Compute camera parameters using point files generated in previous step.
    - Compute and display the mean square error between known image points and computed image points with projection matrix.
3. **RANSAC algorithm**: Perform RANSAC algorithm for robust estimation and read the parameter for RANSAC algorithm from a text file name "RANSAC.config".
4. **Testing**: Test with following provided files:
    - World points: ncc-worldPt.txt
    - Image points: ncc-imagePt.txt
    - Noisy Image points: ncc-noise-0-imagePt.txt, ncc-noise-1-imagePt.txt
    - Known parameter: ncc-params.txt

2 Proposed Solution

Solution for this programming assignment is created using Python with OpenCV. Major algorithms are implemented without using OpenCV functions to achieve various requirements like non-planar camera calibration, mean square error, and RANSAC.

In the first program, added the following **Support Keys** to apply main functions:

- Left mouse click – Mark feature point
- f – Extract Chessboard feature points
- h – Display help
- i – Reload the original image
- w – Save the current image and features extracted

3 Implementation Details

There are two program files, first for feature extraction and second for non-planar camera calibration, mean square error, RANSAC, and testing.

Feature Extraction

Once the image is display on running program 1, press 'f' to display chessboard features automatically. Further, on left mouse click, features can be marked on the image manually. Then press 'w' to save the image and features extracted.

I. When saving OpenCV generated chessboard feature points, corresponding 3D world points are also being saved in the file named "points_3Dto2D.txt" in the required format. Each line in the text file has 5 digits space separated, first three are the x, y, z coordinates of the 3D world points and next 2 digits are the x, y coordinates of the 2D image points.

II. When saving, marked feature points are saved in the file named "". Each line in the text file has 2 digits space separated, which are the x, y coordinates of the 2D image points. We can enter the corresponding 3D world points manually in the text file if want to use those feature points.

Camera Calibration

Program 2 contains all the requirements for non-planar camera calibration, RANSAC algorithm, and testing.

1. **Non-Planar Camera Calibration:** Following functions are implemented:

   I. Implemented GetFilePoint() function to read 3D world points and 2D image points from the file using the filename passed as parameter.

   II. Implemented GetMatrixA() function to build the required matrix for estimating the projection matrix M using all the points provided with given formula:

For a single point:

$$\underbrace{\begin{bmatrix} P_i^T & 0 & -x_i P_i^T \\ 0 & P_i^T & -y_i P_i^T \end{bmatrix}}_{2 \times 12} \underbrace{\begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}}_{12 \times 1} = \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{2 \times 1}$$

   III. Implemented GetMatrixM() function to get the estimated projection matrix using the matrix A by Singular Value Decomposition.

$$A = U\,D\,V^T$$

   IV. Implemented Non_Planar_Calibration() function to compute all the camera parameters using the following equations:

$$\begin{aligned}
|\rho| &= 1/|a_3| \\
u_0 &= |\rho|^2 a_1 \cdot a_3 \\
v_0 &= |\rho|^2 a_2 \cdot a_3 \\
\alpha_v &= \sqrt{|\rho|^2 a_2 \cdot a_2 - v_0^2} \\
s &= |\rho|^4/\alpha_v (a_1 \times a_3) \cdot (a_2 \times a_3) \\
\alpha_u &= \sqrt{|\rho|^2 a_1 \cdot a_1 - s^2 - u_0^2} \\
K^* &= \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \\
\epsilon &= \operatorname{sgn}(b_3) \\
T^* &= \epsilon|\rho|(K^*)^{-1} b \\
r_3 &= \epsilon|\rho| a_3 \\
r_1 &= |\rho|^2/\alpha_v a_2 \times a_3 \\
r_2 &= r_3 \times r_1 \\
R^* &= [r_1^T \;\; r_2^T \;\; r_3^T]^T
\end{aligned}$$

V. Implemented GetMSE() function to compute the mean square error between known image points and computed image points with projection matrix M.

$$E = \frac{1}{m} \sum_i \left(\left\| x_i - \frac{m_1^T \ell_i}{m_3^T \ell_i} \right\|^2 + \left\| y_i - \frac{m_2^T \ell_i}{m_3^T \ell_i} \right\|^2 \right)$$

2. **RANSAC algorithm:** Following functions are implemented:

   I. Implemented GetDistance() function to compute distance between known image points and computed image points with projection matrix M.

   II. Implemented ApplyRANSAC() function for estimating best projection matrix M using the RANSAC algorithm. The function will read the parameter for RANSAC algorithm from "RANSAC.config" text file.

3. **Testing:** Following functions are implemented:

   III. Implemented ReadTestPoints() function to read 3D world points and 2D image points from the provided test file

   IV. Implemented CreateTestFile() function to merge points into one test file just like the required format.

Issues

- While extracting feature points, adding 3D world points manually was the key, but as I used a chessboard with fixed dimensions, it became easier.
- Initially, faced issues figuring out RANSAC algorithm, but it was soon clear enough.

Instructions to run the program

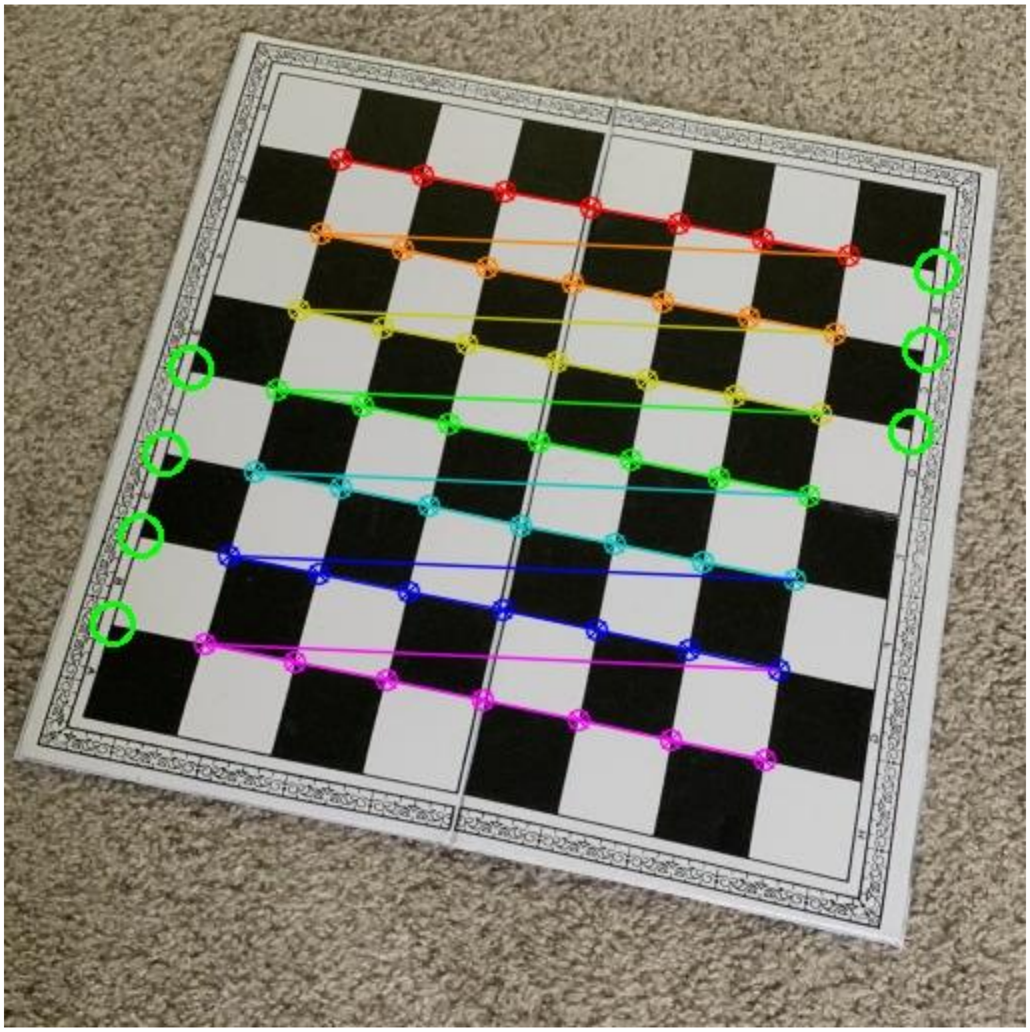
I. In program 1 and 2, I have hard coded input image name and input point file names in the programs, which can be modified in the function calls.

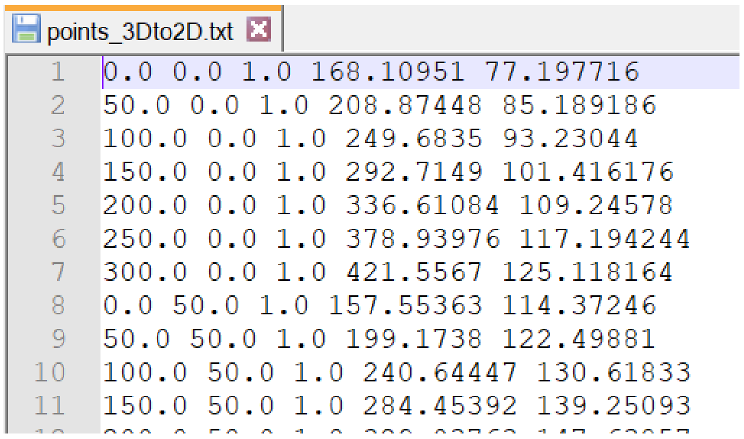
II. When running the programs, the Jupyter notebooks, images and files will have to be in the same folder location unless set otherwise.

4 Results and Discussion

Tried multiple test cases provided for testing to get different solutions. Here are the optimal results obtained using the image clicked from my phone for testing and evaluation of the algorithm implemented:

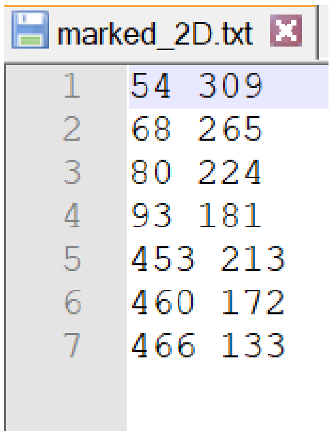
1. **Feature Extraction:** Using a chessboard image clicked by my phone and program 1, following features were detected on the image. The OpenCV detected the multicolor feature points which are stored in “points_3Dto2D.txt” text file. Further, I marked 7 feature points manually which are stored in “marked_2D.txt” text file and displayed by green circles.





points_3Dto2D.txt

```
0.0 0.0 1.0 168.10951 77.197716
50.0 0.0 1.0 208.87448 85.189186
100.0 0.0 1.0 249.6835 93.23044
150.0 0.0 1.0 292.7149 101.416176
200.0 0.0 1.0 336.61084 109.24578
250.0 0.0 1.0 378.93976 117.194244
300.0 0.0 1.0 421.5567 125.118164
0.0 50.0 1.0 157.55363 114.37246
50.0 50.0 1.0 199.1738 122.49881
100.0 50.0 1.0 240.64447 130.61833
150.0 50.0 1.0 284.45392 139.25093
```



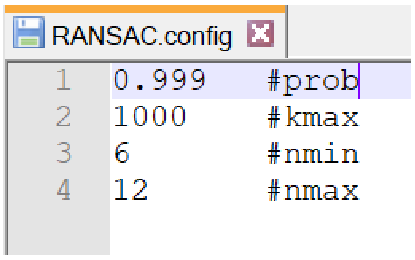
marked_2D.txt

```
54 309
68 265
80 224
93 181
453 213
460 172
466 133
```

2. **Non-planar Camera Calibration:** Using "points_3Dto2D.txt" text file as input, non-planar camera calibration was performed to obtain the camera parameters.

```
***************************************************************

Total Point Pairs =  49

Camera Parameters from Non-Planar Camera Calibration:

M^ =  [[0.0063 -0.0020 0.6415 0.6415]
 [0.0012 0.0054 0.2973 0.2973]
 [-0.0000 -0.0000 0.0038 0.0038]]

u0 =  167.3234          v0 =  77.5282

alpha u =  1.7093       alpha v =  1.482

s =  -0.0457            ρ =  260.8117

K* =  [[1.7093 -0.0457 167.3234]
 [0.0000 1.4820 77.5282]
 [0.0000 0.0000 1.0000]]

T* =  [0.0000 0.0007 1.0000]

R* =  [[0.9773 -0.2119 0.0000]
 [0.2119 0.9773 0.0007]
 [-0.0001 -0.0006 1.0000]]

M =  [[1.6373 -0.5148 167.3233 167.3233]
 [0.3031 1.3983 77.5292 77.5292]
 [-0.0001 -0.0006 1.0000 1.0000]]

Mean Square Error =  0.5734
```

3. **RANSAC algorithm:** Using “points_3Dto2D.txt” text file as input, non-planar camera calibration was performed along with RANSAC algorithm to get the best projection matrix M and obtain rest of the camera parameters. Here, the mean square error is almost the same as before.

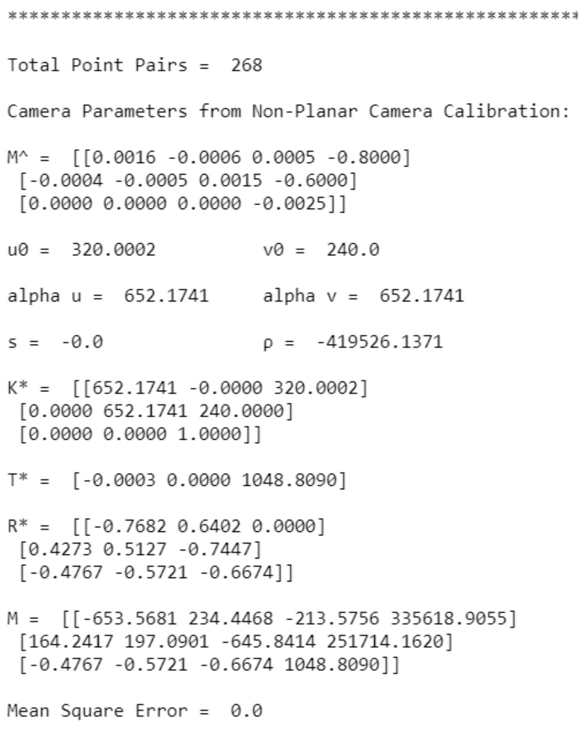

RANSAC.config

```
0.999       #prob
1000        #kmax
6           #nmin
12          #nmax
```

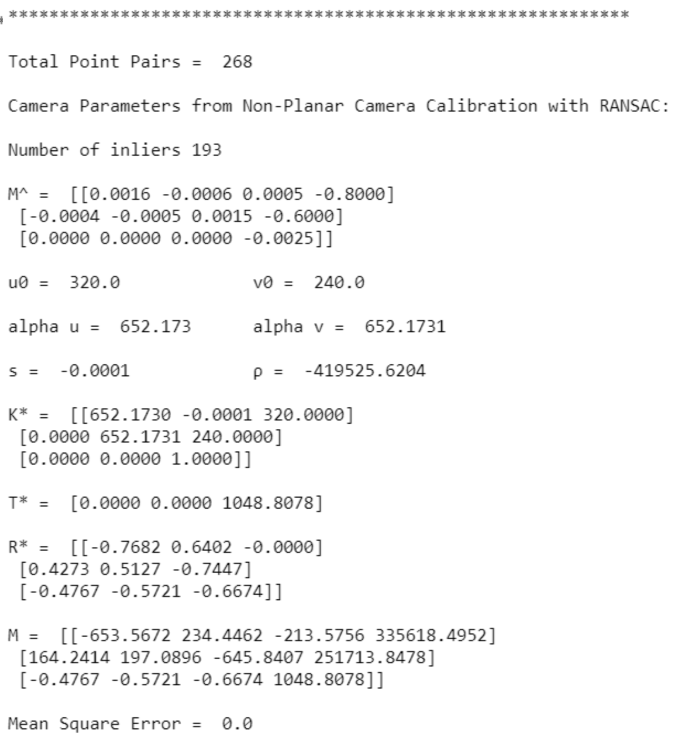
```
*************************************************************

Total Point Pairs =  49

Camera Parameters from Non-Planar Camera Calibration with RANSAC:

Number of inliers 28

M^ =  [[0.0062 -0.0020 0.6413 0.6413]
 [0.0011 0.0053 0.2978 0.2978]
 [-0.0000 -0.0000 0.0038 0.0038]]

u0 =  167.4372          v0 =  77.7658

alpha u =  1.704        alpha v =  1.4773

s =  -0.0482            ρ =  261.099

K* =  [[1.7040 -0.0482 167.4372]
 [0.0000 1.4773 77.7658]
 [0.0000 0.0000 1.0000]]

T* =  [0.0000 0.0007 1.0000]

R* =  [[0.9776 -0.2106 0.0000]
 [0.2106 0.9776 0.0007]
 [-0.0002 -0.0007 1.0000]]

M =  [[1.6295 -0.5155 167.4371 167.4371]
 [0.2990 1.3933 77.7668 77.7668]
 [-0.0002 -0.0007 1.0000 1.0000]]

Mean Square Error =  0.594
```

4. **Testing:** Three tests were conduction using different files provided.

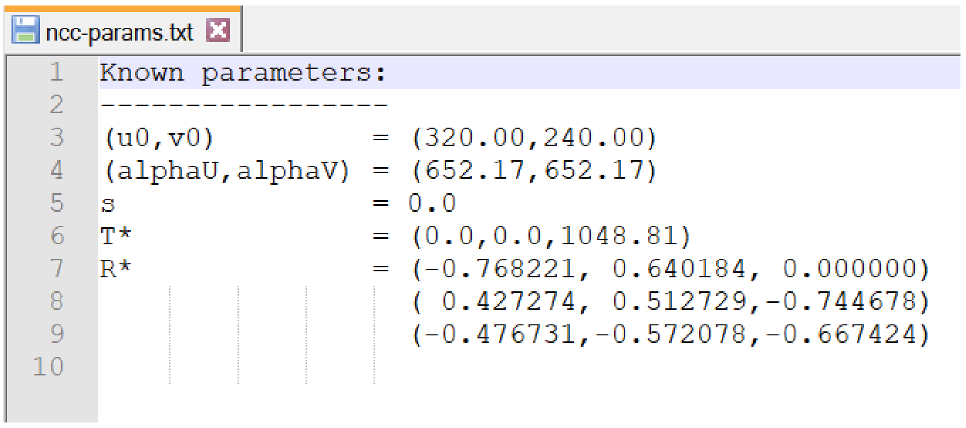
   I. **Test 1 without noise:** Using ncc-worldPt.txt and ncc-imagePt.txt, a text file named “test1_3Dto2D.txt” was created and passed as parameter for non-planar camera calibration with and without RANSAC algorithm. Here the mean square error was 0 without noise. The camera parameters match the known parameters provide in ncc-params.txt.



```
*******************************************************

Total Point Pairs =  268

Camera Parameters from Non-Planar Camera Calibration:

M^ =  [[0.0016 -0.0006 0.0005 -0.8000]
 [-0.0004 -0.0005 0.0015 -0.6000]
 [0.0000 0.0000 0.0000 -0.0025]]

u0 =  320.0002          v0 =  240.0

alpha u =  652.1741     alpha v =  652.1741

s =  -0.0               ρ =  -419526.1371

K* =  [[652.1741 -0.0000 320.0002]
 [0.0000 652.1741 240.0000]
 [0.0000 0.0000 1.0000]]

T* =  [-0.0003 0.0000 1048.8090]

R* =  [[-0.7682 0.6402 0.0000]
 [0.4273 0.5127 -0.7447]
 [-0.4767 -0.5721 -0.6674]]

M =  [[-653.5681 234.4468 -213.5756 335618.9055]
 [164.2417 197.0901 -645.8414 251714.1620]
 [-0.4767 -0.5721 -0.6674 1048.8090]]

Mean Square Error =  0.0
```

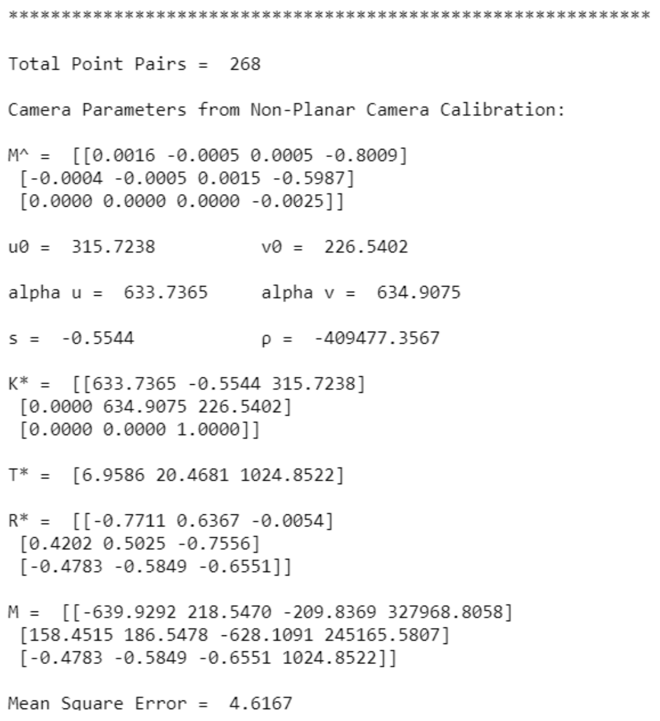


```
************************************************************

Total Point Pairs =  268

Camera Parameters from Non-Planar Camera Calibration with RANSAC:

Number of inliers 193

M^ =  [[0.0016 -0.0006 0.0005 -0.8000]
 [-0.0004 -0.0005 0.0015 -0.6000]
 [0.0000 0.0000 0.0000 -0.0025]]

u0 =  320.0             v0 =  240.0

alpha u =  652.173      alpha v =  652.1731

s =  -0.0001            ρ =  -419525.6204

K* =  [[652.1730 -0.0001 320.0000]
 [0.0000 652.1731 240.0000]
 [0.0000 0.0000 1.0000]]

T* =  [0.0000 0.0000 1048.8078]

R* =  [[-0.7682 0.6402 -0.0000]
 [0.4273 0.5127 -0.7447]
 [-0.4767 -0.5721 -0.6674]]

M =  [[-653.5672 234.4462 -213.5756 335618.4952]
 [164.2414 197.0896 -645.8407 251713.8478]
 [-0.4767 -0.5721 -0.6674 1048.8078]]

Mean Square Error =  0.0
```

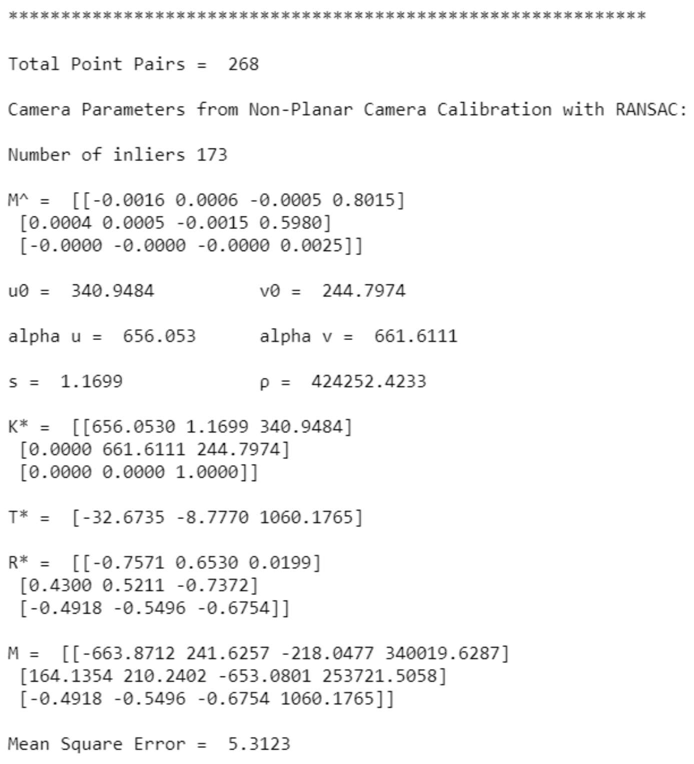


```
Known parameters:
------------------
(u0,v0)          = (320.00,240.00)
(alphaU,alphaV)  = (652.17,652.17)
s                = 0.0
T*               = (0.0,0.0,1048.81)
R*               = (-0.768221, 0.640184, 0.000000)
                   ( 0.427274, 0.512729,-0.744678)
                   (-0.476731,-0.572078,-0.667424)
```

II. **Test 2:** Using ncc-worldPt.txt and ncc-noise-0-imagePt.txt, a text file named "test2_noise_3Dto2D.txt" was created and passed as parameter for non-planar camera calibration with and without RANSAC algorithm. The mean square error increases due to the noisy points to 4.6167 and 5.3123 without and with RANSAC respectively, which is not too bad.

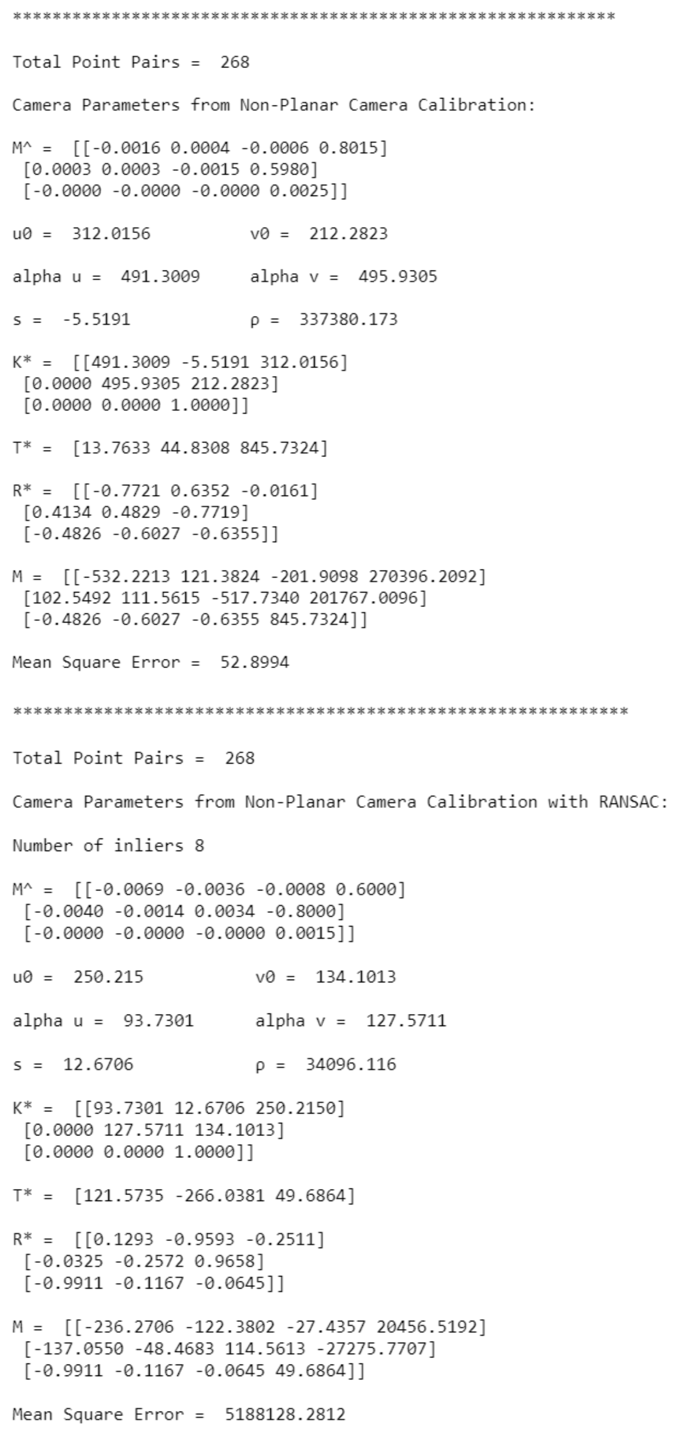


```
************************************************************

Total Point Pairs =  268

Camera Parameters from Non-Planar Camera Calibration:

M^ =  [[0.0016 -0.0005 0.0005 -0.8009]
 [-0.0004 -0.0005 0.0015 -0.5987]
 [0.0000 0.0000 0.0000 -0.0025]]

u0 =  315.7238          v0 =  226.5402

alpha u =  633.7365     alpha v =  634.9075

s =  -0.5544            ρ =  -409477.3567

K* =  [[633.7365 -0.5544 315.7238]
 [0.0000 634.9075 226.5402]
 [0.0000 0.0000 1.0000]]

T* =  [6.9586 20.4681 1024.8522]

R* =  [[-0.7711 0.6367 -0.0054]
 [0.4202 0.5025 -0.7556]
 [-0.4783 -0.5849 -0.6551]]

M =  [[-639.9292 218.5470 -209.8369 327968.8058]
 [158.4515 186.5478 -628.1091 245165.5807]
 [-0.4783 -0.5849 -0.6551 1024.8522]]

Mean Square Error =  4.6167
```



```
************************************************************

Total Point Pairs =  268

Camera Parameters from Non-Planar Camera Calibration with RANSAC:

Number of inliers 173

M^ =  [[-0.0016 0.0006 -0.0005 0.8015]
 [0.0004 0.0005 -0.0015 0.5980]
 [-0.0000 -0.0000 -0.0000 0.0025]]

u0 =  340.9484          v0 =  244.7974

alpha u =  656.053      alpha v =  661.6111

s =  1.1699             ρ =  424252.4233

K* =  [[656.0530 1.1699 340.9484]
 [0.0000 661.6111 244.7974]
 [0.0000 0.0000 1.0000]]

T* =  [-32.6735 -8.7770 1060.1765]

R* =  [[-0.7571 0.6530 0.0199]
 [0.4300 0.5211 -0.7372]
 [-0.4918 -0.5496 -0.6754]]

M =  [[-663.8712 241.6257 -218.0477 340019.6287]
 [164.1354 210.2402 -653.0801 253721.5058]
 [-0.4918 -0.5496 -0.6754 1060.1765]]

Mean Square Error =  5.3123
```

III. **Test 3:** Using ncc-worldPt.txt and ncc-noise-1-imagePt.txt, a text file named "test3_noise_3Dto2D.txt" was created and passed as parameter for non-planar camera calibration with and without RANSAC algorithm. With more noisy points, the mean square error increases to 52.8994 without RANSAC. Whereas with RANSAC, it increases to a huge value with a divide by zero error, which is extremely bad performance.



```
************************************************************

Total Point Pairs =  268

Camera Parameters from Non-Planar Camera Calibration:

M^ =  [[-0.0016 0.0004 -0.0006 0.8015]
 [0.0003 0.0003 -0.0015 0.5980]
 [-0.0000 -0.0000 -0.0000 0.0025]]

u0 =  312.0156          v0 =  212.2823

alpha u =  491.3009     alpha v =  495.9305

s =  -5.5191            ρ =  337380.173

K* =  [[491.3009 -5.5191 312.0156]
 [0.0000 495.9305 212.2823]
 [0.0000 0.0000 1.0000]]

T* =  [13.7633 44.8308 845.7324]

R* =  [[-0.7721 0.6352 -0.0161]
 [0.4134 0.4829 -0.7719]
 [-0.4826 -0.6027 -0.6355]]

M =  [[-532.2213 121.3824 -201.9098 270396.2092]
 [102.5492 111.5615 -517.7340 201767.0096]
 [-0.4826 -0.6027 -0.6355 845.7324]]

Mean Square Error =  52.8994

*************************************************************

Total Point Pairs =  268

Camera Parameters from Non-Planar Camera Calibration with RANSAC:

Number of inliers 8

M^ =  [[-0.0069 -0.0036 -0.0008 0.6000]
 [-0.0040 -0.0014 0.0034 -0.8000]
 [-0.0000 -0.0000 -0.0000 0.0015]]

u0 =  250.215           v0 =  134.1013

alpha u =  93.7301      alpha v =  127.5711

s =  12.6706            ρ =  34096.116

K* =  [[93.7301 12.6706 250.2150]
 [0.0000 127.5711 134.1013]
 [0.0000 0.0000 1.0000]]

T* =  [121.5735 -266.0381 49.6864]

R* =  [[0.1293 -0.9593 -0.2511]
 [-0.0325 -0.2572 0.9658]
 [-0.9911 -0.1167 -0.0645]]

M =  [[-236.2706 -122.3802 -27.4357 20456.5192]
 [-137.0550 -48.4683 114.5613 -27275.7707]
 [-0.9911 -0.1167 -0.0645 49.6864]]

Mean Square Error =  5188128.2812
```

5. **Reason for RANSAC not being able to handle noisy data in test 3:**

   RANSAC is basically random sample consensus. For second noisy data, we get huge mean square error with RANSAC as it fails completely. This is because of 2 main reasons:

   I. RANSAC does not work well with low inlier ratios as every point in the dataset gets a vote just for being in the model. This makes the model more robust towards outliers. So, we need more inliers and less outlier for RANSAC to work properly. It does not work well as especially when there are more than 50% outliers.

   II. RANSAC is also not reliable when there are a lot of parameters to tune. Here, we have 12 parameters which makes it more complicated.

5 References

OpenCV Chessboard Corners: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html