Assignment 4 Report

Harsh Vora
CWID : A20445400
CS512 – F20

November 29, 2020

1. Problem Statement

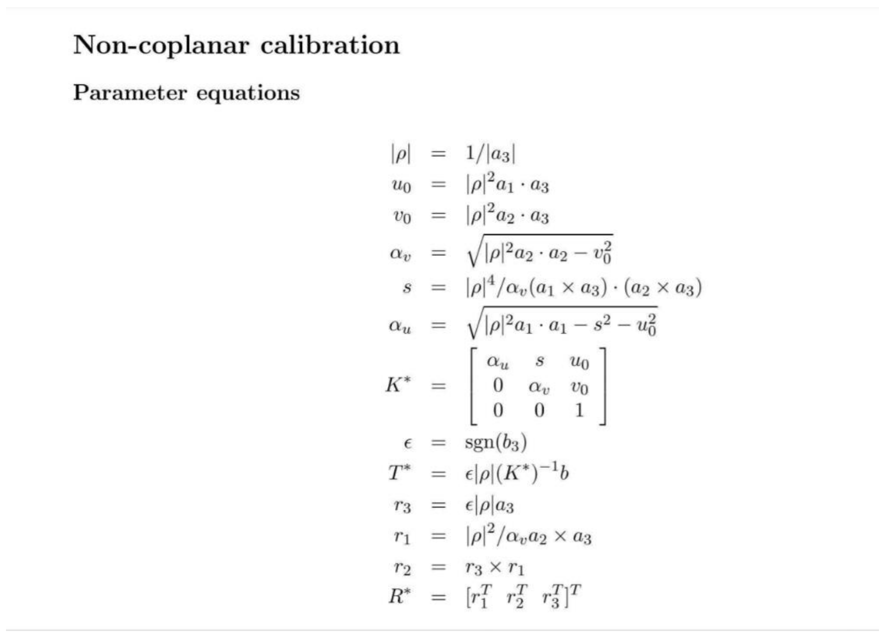
- Write a program to extract feature points from the calibration target and show them on the image using OpenCV functions, alternatively writing a program that allows you to interactively mark the points on the image. Also save the image points detected and/or manually entered in a file.

- Write a second program to compute camera parameters using a point correspondence file with a pair of corresponding points (3D-2D). The program should display the intrinsic and extrinsic parameters determined by the calibration process and display the mean square error between the known and computed position of the image. The computed position should be obtained by using the estimated camera parameters to project the 3D points onto the image plane.

- Implement RANSAC algorithm for robust estimation. The implementation of the RANSAC algorithm should include automatic estimation of the number of draws and of the probability that a data point is an inlier. The final values of these estimates should be displayed by the program. Parameters used in the RANSAC algorithm should be read from a text file named "RANSAC.config".

2. Proposed Solution

- I have used OpenCV functions such as cvFindChessBoardCorneres, cvFindCornersSubPix, cvDrawChessBoardCorners, to extract feature points from the calibration target.
- In the second Program we read from the correspondensePoints.txt file and create a estimation of projection matrix and then calculate the intrinsic and extrinsic parameters and also Mean Square Error(mse).
- To compute the camera parameters such as intrinsic and extrinsic parameters I made use of the following formulas:



Non-coplanar calibration

Parameter equations

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

mean square error:

$$mean\ square\ error = \frac{\sum_{i=1}^{n}(x_i - \frac{m_1^T p_i}{m_3^T P_i})^2 + (y_i - \frac{m_2^T p_i}{m_3^T P_i})^2}{n}$$

- After that for Implementation of RANSAC algorithm for robust estimation I used the following algorithm:
  Step1: Draw n points uniformly at random with replacement.
  Step2: Fit model to points.
  Step3: Find all inliners.
  Step4: If there are d inliners, recompute the model.
  Step5: Finally choose the best solution, when it has the largest number of inliners.

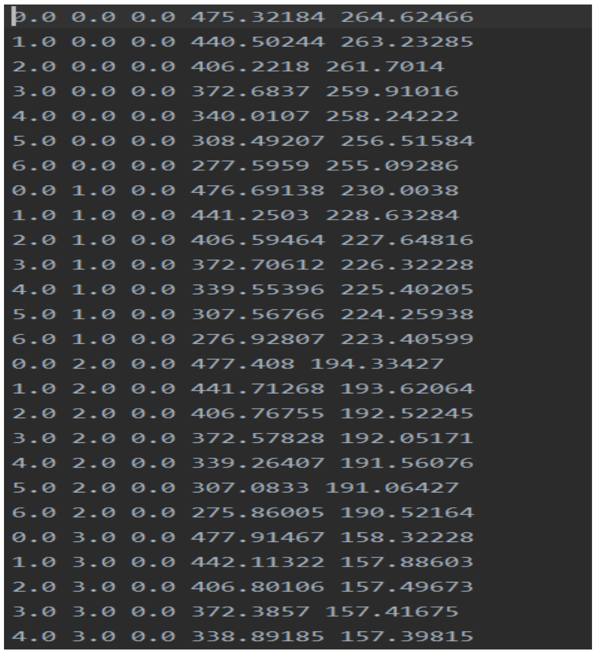
3. Implementation Details

- I used zip() function to iterate two list at same time , which helps to compute faster.
- np.concatenate() make it easier to combine two np array.
- Managing the datatype of the arguments between functions was bit hard and I had to make sure that they are converted to the right datatype in the beginning of the function.
- Had many errors like divide by zero error and matrix dimension error while multiplying 2 matrix.

4. Results and Discussion

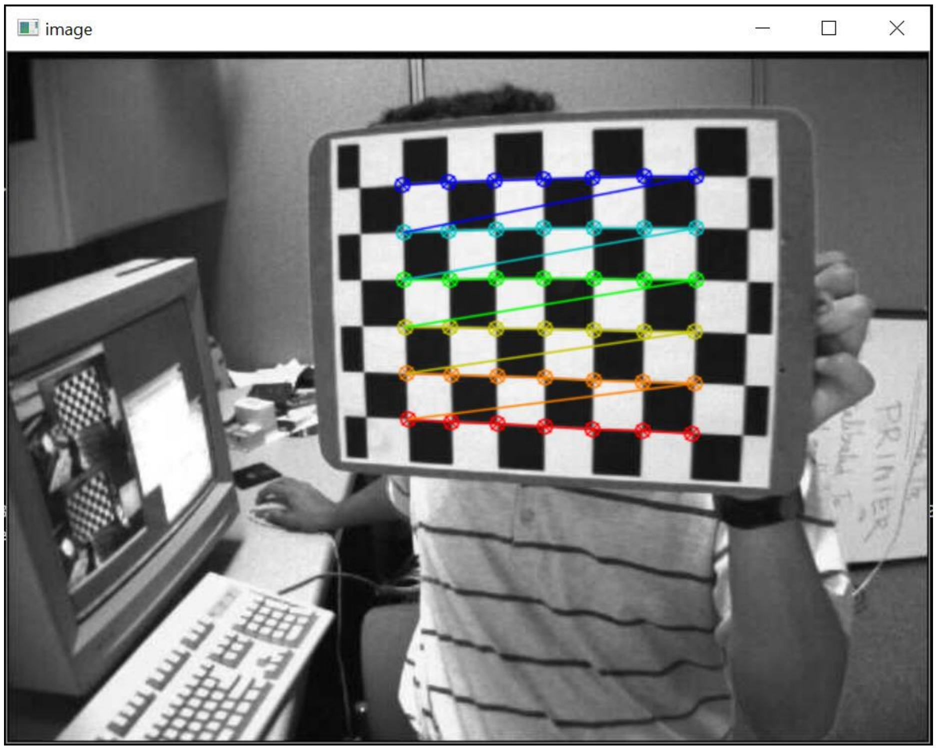
1st Program: Extract feature points from a image.

a) Correspondence file generated for the program.



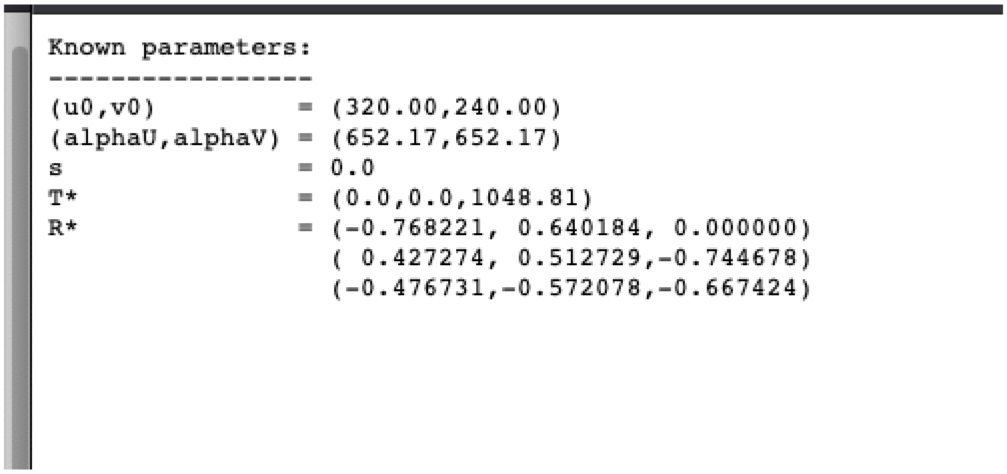
```
0.0 0.0 0.0 475.32184 264.62466
1.0 0.0 0.0 440.50244 263.23285
2.0 0.0 0.0 406.2218 261.7014
3.0 0.0 0.0 372.6837 259.91016
4.0 0.0 0.0 340.0107 258.24222
5.0 0.0 0.0 308.49207 256.51584
6.0 0.0 0.0 277.5959 255.09286
0.0 1.0 0.0 476.69138 230.0038
1.0 1.0 0.0 441.2503 228.63284
2.0 1.0 0.0 406.59464 227.64816
3.0 1.0 0.0 372.70612 226.32228
4.0 1.0 0.0 339.55396 225.40205
5.0 1.0 0.0 307.56766 224.25938
6.0 1.0 0.0 276.92807 223.40599
0.0 2.0 0.0 477.408 194.33427
1.0 2.0 0.0 441.71268 193.62064
2.0 2.0 0.0 406.76755 192.52245
3.0 2.0 0.0 372.57828 192.05171
4.0 2.0 0.0 339.26407 191.56076
5.0 2.0 0.0 307.0833 191.06427
6.0 2.0 0.0 275.86005 190.52164
0.0 3.0 0.0 477.91467 158.32228
1.0 3.0 0.0 442.11322 157.88603
2.0 3.0 0.0 406.80106 157.49673
3.0 3.0 0.0 372.3857 157.41675
4.0 3.0 0.0 338.89185 157.39815
```

b) Image with detected points.



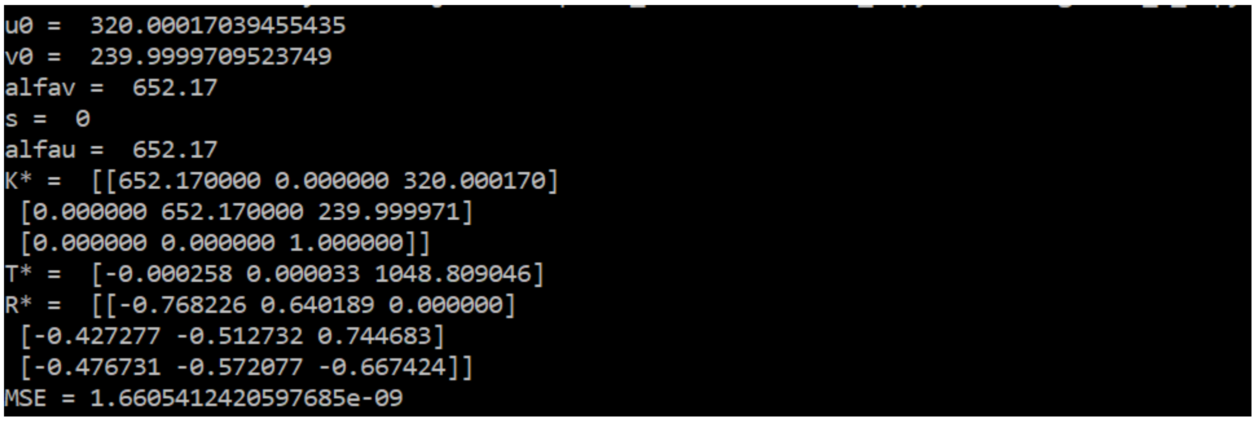
2nd Program: Computing camera intrinsic and extrinsic parameters as well as Mean Square Error.
The generated value are almost same with the values given by the professor.

Professors known Parameters:



```
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)
```

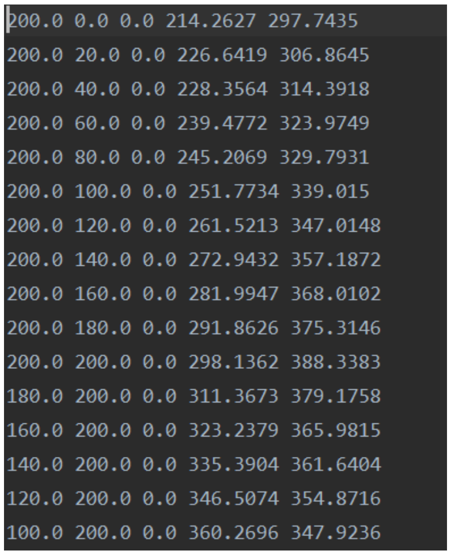
My Parameters:



```
u0 =  320.00017039455435
v0 =  239.9999709523749
alfav =  652.17
s =  0
alfau =  652.17
K* =  [[652.170000 0.000000 320.000170]
 [0.000000 652.170000 239.999971]
 [0.000000 0.000000 1.000000]]
T* =  [-0.000258 0.000033 1048.809046]
R* =  [[-0.768226 0.640189 0.000000]
 [-0.427277 -0.512732 0.744683]
 [-0.476731 -0.572077 -0.667424]]
MSE = 1.6605412420597685e-09
```

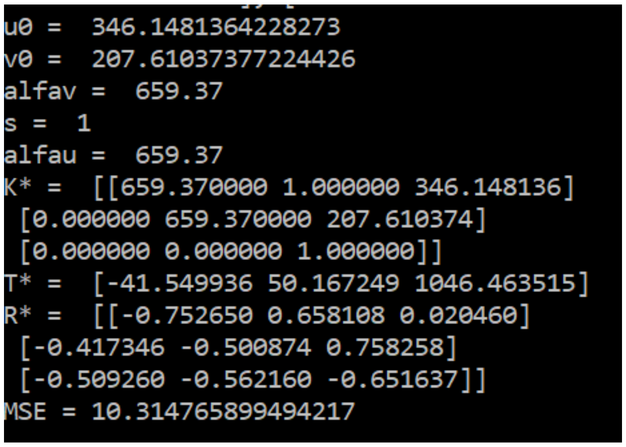
3. Parameters Influences: (RANSAC.config)

Noise 1 File:



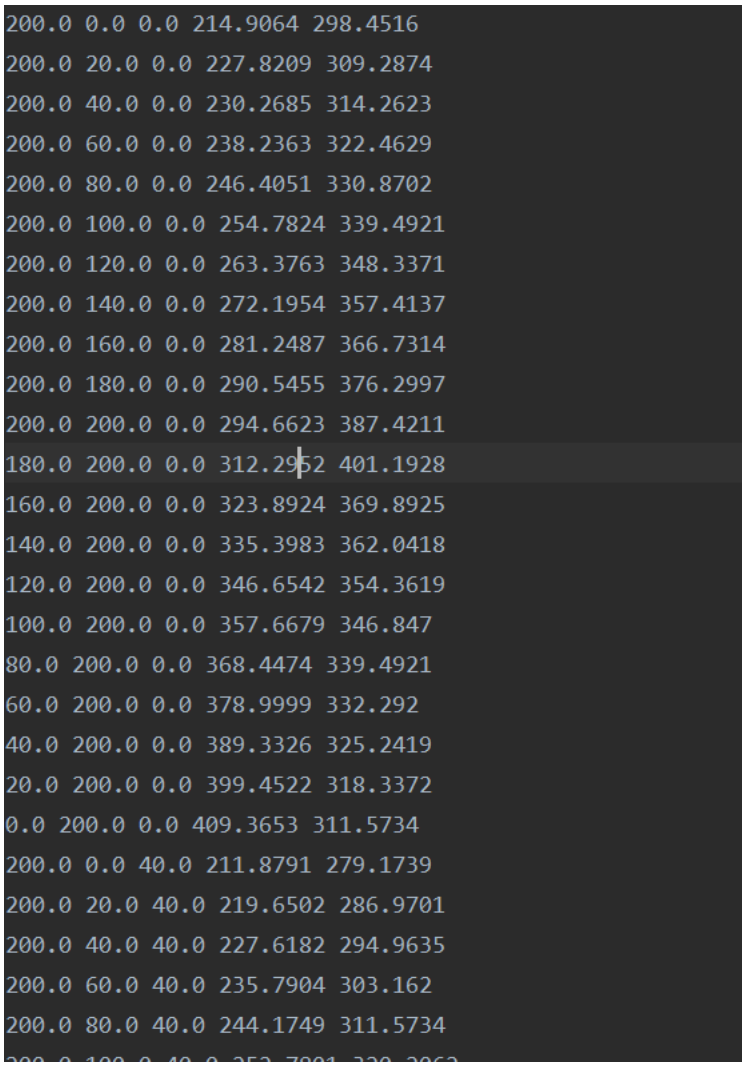
```
200.0 0.0 0.0 214.2627 297.7435
200.0 20.0 0.0 226.6419 306.8645
200.0 40.0 0.0 228.3564 314.3918
200.0 60.0 0.0 239.4772 323.9749
200.0 80.0 0.0 245.2069 329.7931
200.0 100.0 0.0 251.7734 339.015
200.0 120.0 0.0 261.5213 347.0148
200.0 140.0 0.0 272.9432 357.1872
200.0 160.0 0.0 281.9947 368.0102
200.0 180.0 0.0 291.8626 375.3146
200.0 200.0 0.0 298.1362 388.3383
180.0 200.0 0.0 311.3673 379.1758
160.0 200.0 0.0 323.2379 365.9815
140.0 200.0 0.0 335.3904 361.6404
120.0 200.0 0.0 346.5074 354.8716
100.0 200.0 0.0 360.2696 347.9236
```

Output:



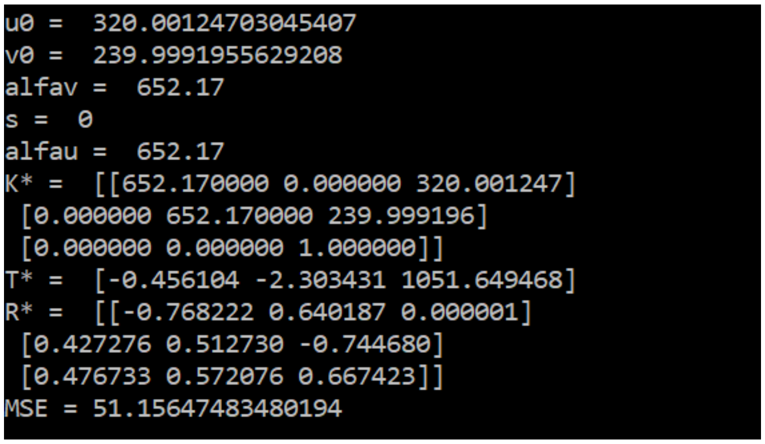
```
u0 =  346.1481364228273
v0 =  207.61037377224426
alfav =  659.37
s =  1
alfau =  659.37
K* =  [[659.370000 1.000000 346.148136]
 [0.000000 659.370000 207.610374]
 [0.000000 0.000000 1.000000]]
T* =  [-41.549936 50.167249 1046.463515]
R* =  [[-0.752650 0.658108 0.020460]
 [-0.417346 -0.500874 0.758258]
 [-0.509260 -0.562160 -0.651637]]
MSE = 10.314765899494217
```

Noise 2 File:



```
200.0 0.0 0.0 214.9064 298.4516
200.0 20.0 0.0 227.8209 309.2874
200.0 40.0 0.0 230.2685 314.2623
200.0 60.0 0.0 238.2363 322.4629
200.0 80.0 0.0 246.4051 330.8702
200.0 100.0 0.0 254.7824 339.4921
200.0 120.0 0.0 263.3763 348.3371
200.0 140.0 0.0 272.1954 357.4137
200.0 160.0 0.0 281.2487 366.7314
200.0 180.0 0.0 290.5455 376.2997
200.0 200.0 0.0 294.6623 387.4211
180.0 200.0 0.0 312.2952 401.1928
160.0 200.0 0.0 323.8924 369.8925
140.0 200.0 0.0 335.3983 362.0418
120.0 200.0 0.0 346.6542 354.3619
100.0 200.0 0.0 357.6679 346.847
80.0 200.0 0.0 368.4474 339.4921
60.0 200.0 0.0 378.9999 332.292
40.0 200.0 0.0 389.3326 325.2419
20.0 200.0 0.0 399.4522 318.3372
0.0 200.0 0.0 409.3653 311.5734
200.0 0.0 40.0 211.8791 279.1739
200.0 20.0 40.0 219.6502 286.9701
200.0 40.0 40.0 227.6182 294.9635
200.0 60.0 40.0 235.7904 303.162
200.0 80.0 40.0 244.1749 311.5734
```

Output:



```
u0 =  320.00124703045407
v0 =  239.9991955629208
alfav =  652.17
s =  0
alfau =  652.17
K* =  [[652.170000 0.000000 320.001247]
 [0.000000 652.170000 239.999196]
 [0.000000 0.000000 1.000000]]
T* =  [-0.456104 -2.303431 1051.649468]
R* =  [[-0.768222 0.640187 0.000001]
 [0.427276 0.512730 -0.744680]
 [0.476733 0.572076 0.667423]]
MSE = 51.15647483480194
```

After comparing the result of noise 1 and noise 2 with mse value I can see noise 1 output is having less mse then compared to noise 2 .Therefore, I think RANSAC was not able to clean the noise in noise2 file because the noise is much higher. It was able to handle the noise in noise1 file because the noise is less. This shows that RANSAC cannot clean files having lots of noise.

According to me RANSAC algorithm is not performing well with noise2 data because Noise1 has less number of outliers for RANSAC to filter out which is why we got less projection error for noise1 data. Moreover noise1 has less data and it is performing better.