CS512 Assignment 3: report

Gady Agam
Illinois Institute of Technology

Abstract

1. Problem statement

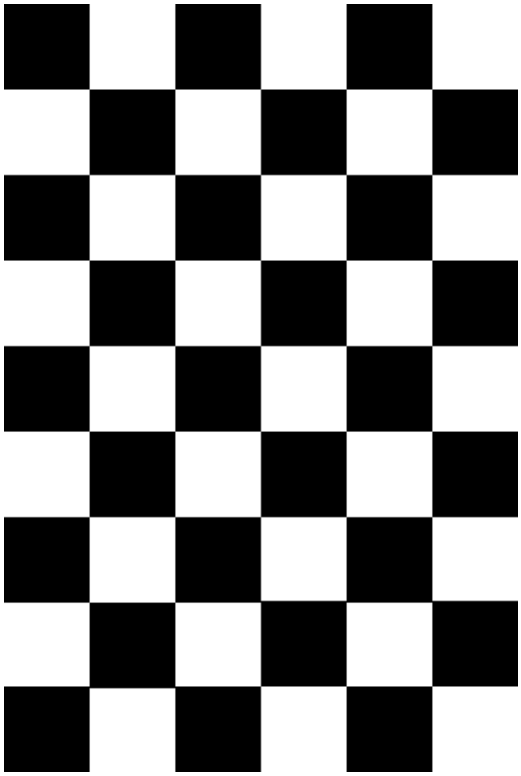
This assignment can be separated into two part: 1) detect feature points from the calibration target through camera, show them on the image and save them with corresponded world points. 2) use these 2D and 3D points to do camera calibration.

We separate these two part into two executable programs.

The camera calibration methods can be separated into two types: coplanar calibration and non-coplanar calibration. What I do in this assignment is coplanar calibration.

2. Proposed solution

The **first** program is to detect feature points and connect them with world points. Since what I do is coplanar calibration, the calibration target I use is a chessboard(a planar object) as following:



The edge length of each square is 20mm when printed. Although there are 7*10 feature points in this calibration target, there are only 5*8 points which are inners can be used. So there are 40 feature points and corresponding world points. We use OpenCV function "findChessboardCorners" to find the feature points, use "cornerSubPix" to make the detection more accurate, use "drawChessboardCorners" to draw the corners on the picture. We assume the first feature point that the "findChessboardCorners" find is the origin of world coordinate. The

coplanar calibration we use needs at least three photos of calibration target.

The **second** program is to calibrate the camera. The coplanar calibration method I used can be separate into three parts:

1) Input a world point set with three corresponding image point sets(each set represents feature points in one image). For each image point sets, compute its matrix H. Typically there are three image sets, so we get H1, H2 and H3.
2) Use {H1, H2, H3} to compute S. S is a matrix that gets from matrix K*. K* is a matrix of intrinsic parameters of the camera.
3) Use {H1, H2, H3} and S to get K*.
4) For each image, use corresponding H and K* to compute its [R*|T*]. [R*|T*] is a matrix of extrinsic parameters.
5) Calculate the error.

Since the calibration target—chessboard— we use is coplanar, the z-coordinate of the points on the chessboard are all the same. So we assume all these points has zero z-coordinate. Let Q be a 3DH world point, P be a 2DH image point, then:

$$P_i=\begin{bmatrix}x_i\\y_i\\z_i\end{bmatrix}=K^*\ast[R^*|T^*]\ast Q_i=K^*\ast[r_1r_2r_3|T^*]\ast\begin{bmatrix}x'_i\\y'_i\\0\\1\end{bmatrix}=K^*\ast[r_1r_2|T^*]\ast\begin{bmatrix}x'_i\\y'_i\\1\end{bmatrix}=K^*\ast H\ast Q'_i$$

Here we see the definition of matrix H. For each image, if we know at least 4 world-image corresponding point pairs, we can calculate its matrix H. But since the image points we know are in form (x_i / z_i, y_i / z_i), the matrix H we get is normalized (we don't know the const parameter of H, so normalized it that its last column is 1).

For three images we can get {H1, H2, H3}. Notice that these three images have same intrinsic(K^*) camera parameters but different extrinsic camera parameters([$R^*|T^*$]). By using these Hs we can get S, which is:

$$S=\left(K^{*-1}\right)^T\ast K^{*-1}$$

By using {H1, H2, H3, S} we can get intrinsic parameters matrix K^*. Then [$R^*|T^*$]s for each images.

The calculations after we get {H1, H2, H3} (step 2-4) are straightforward, which will not induct errors. But the Hs we got may be wrong, because they are got from points which may be not accurate enough that have some noise or deviation in it.

We use RANSAC algorithm to try to get better Hs with points which are not that accurate. The RANSAC algorithm I use is:

1) random choose n (a small number) points, use these points to compute matrix H
2) update t=1.5 median distance error. Find all inliers by t
3) recompute matrix H with all these inliers if # of inliers >=d
4) update w and k
5) repeat step1-4 m times until m>=k
6) return the best H that has most inliers and least distance error summation (a function of these two parameters)

If the noise is impulsive noise that only affect few point with large difference, the RANSAC can handle it very well. But if the noise is gaussian noise that affect almost all point with small difference, the RANSAC can not handle it well.

The program does not address radial lens distortion.

To judge if the result intrinsic and extrinsic parameters is good, one method is to calculate the mean square error between the known and computed position of the image points. The program will output this error.

3. Implementation details

The first program just call OpenCV functions, so there's nothing need to talk about it.

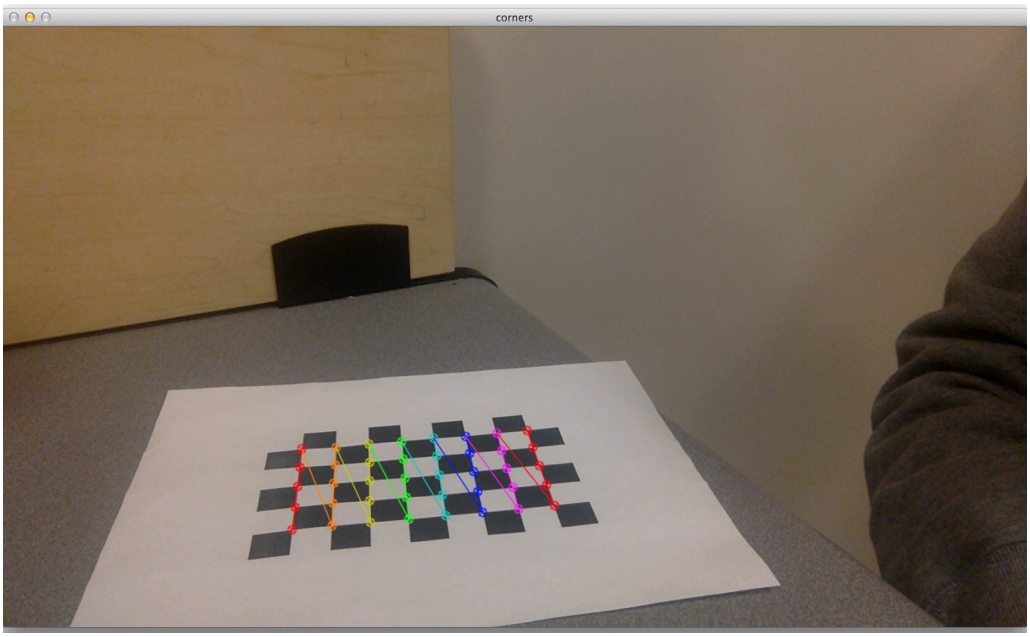
For second program, step 2-5 are just use the formulas to do some calculations, so also nothing need to talk. For the step 1: The RANSAC algorithm is more like an idea than a fixed algorithm, so many implement details can be changed in it. For example, the step 3 in RANSAC I described above can be cancelled; how to calculate t, w and k. Different version of RANSAC may have difference performance and precision on different set of input data. The "version" of RANSAC I used in this program is described above.

One problem I found in testing is: if the n world points we choose to calibrate camera are "fortunately" on the same line, the algorithm will be failed. This is similar to the degenerate case of non-coplanar calibration. If {h1, h2, h3} are any combinations of the line expression of the line that these world points lay on, Ax will be 0 anyway. But since RANSAC will loop for k times and get the best H, these "error" Hs will be deleted anyway. So the possibility that all iterations get points in the same line is very low. Therefore if the input world points and image point are not on only one line, we don't have to consider it.

I write a third program which use the OpenCV function to do the camera calibration.

4. Results and discussion

Here is an example of output of first program:



Remember that the number of feature points is 5*8. This size is good because OpenCV can decide the direction without ambiguous (observe the color of the four corner squares). The output of this program is same as the date file on course web.

For the data in the course web side, there's the output of my second program:

A)

```
image 0:
w=1.000000, k=0.000000, # of inliers=121, average inlier distance error=0.000049
image 1:
w=1.000000, k=0.000000, # of inliers=121, average inlier distance error=0.000069
image 2:
w=1.000000, k=0.000000, # of inliers=121, average inlier distance error=0.000059
0th H:
[-0.91106004, 1.26071680, 182.70024109;
  0.85495085, 0.42747545, -67.01156616;
  -0.00070175, -0.00035088, 1.00000000]
1th H:
[-1.14384365, 1.04027700, 259.25888062;
  0.58434820, 0.43826103, -25.61271858;
  -0.00072727, -0.00054545, 1.00000000]
2th H:
[-0.56874096, 1.41147220, 100.66010284;
  1.05276453, 0.26319149, -61.28648758;
  -0.00063492, -0.00015873, 1.00000000]
Kstar:
[1304.36169434, -0.00080154, 319.99862671;
  0.00000000, 1304.35339355, 239.97830200;
  0.00000000, 0.00000000, 1.00000000]
0th RstarTstar:
[-0.44721374, 0.89442700, 0.00000000, -89.44194794;
  0.66666090, 0.33333051, -0.66667354, -199.98712158;
  -0.59629101, -0.29814556, -0.74534947, 849.71496582]
1th RstarTstar:
[-0.60000032, 0.79999983, -0.00000031, -39.99920273;
  0.49974877, 0.37481162, -0.78087622, -174.90142822;
  -0.62470061, -0.46852612, -0.62468600, 858.96453857]
2th RstarTstar:
[-0.24253553, 0.97014248, 0.00000022, -145.52059937;
  0.79955167, 0.19988813, -0.56635851, -199.87585449;
  -0.54944849, -0.13736199, -0.82415903, 865.38073730]
for image0, the error is 0.0001 pixels.
for image1, the error is 0.0001 pixels.
for image2, the error is 0.0001 pixels.
Program ended with exit code: 0
```

Images without noise

Comparing to the “know parameter” on the web, the error is less than 0.1%. We can think the algorithm is success in this situation.

B)

```
image 0:
w=0.793388, k=9.127175, # of inliers=96, average inlier distance error=1.573140
image 1:
w=1.000000, k=0.000000, # of inliers=121, average inlier distance error=0.000069
image 2:
w=1.000000, k=0.000000, # of inliers=121, average inlier distance error=0.000059
0th H:
[-0.91809618, 1.28148031, 181.54078674;
  0.85149831, 0.43559560, -66.83169556;
  -0.00071941, -0.00031381, 1.00000000]
1th H:
[-1.14384365, 1.04027700, 259.25888062;
  0.58434820, 0.43826103, -25.61271858;
  -0.00072727, -0.00054545, 1.00000000]
2th H:
[-0.56874096, 1.41147220, 100.66010284;
  1.05276453, 0.26319149, -61.28648758;
  -0.00063492, -0.00015873, 1.00000000]
Kstar:
[1576.84948730, -21.96019363, 317.32684326;
  0.00000000, 1216.28039551, -348.23065186;
  0.00000000, 0.00000000, 1.00000000]
0th RstarTstar:
[-0.44244131, 0.90380502, 0.04007432, -85.17388916;
  0.50772440, 0.27568281, -0.81078815, 237.73446655;
  -0.73923051, -0.32245609, -0.58085734, 1027.55102539]
1th RstarTstar:
[-0.59524775, 0.79917848, -0.00000049, -34.28302383;
  0.28167820, 0.21125831, -0.93739259, 274.47058105;
  -0.75255406, -0.56441635, -0.35086221, 1034.76330566]
2th RstarTstar:
[-0.23282377, 0.96870929, 0.00000034, -139.78506470;
  0.71266359, 0.17816624, -0.67955393, 245.88568115;
  -0.66174299, -0.16543558, -0.73184514, 1042.24438477]
for image0, the error is 2.0092 pixels.
for image1, the error is 0.0001 pixels.
for image2, the error is 0.0001 pixels.
Program ended with exit code: 0
```

Image 1 has gaussian noise

By observing the K^* matrix, v_0 is negative and s is large. So we can see: If the image has gaussian noise, the RANSAC can not do well. Actually when I use all 121 points to compute H without RANSAC algorithm, the output parameters are also unreasonable. One conclusion I get is: the calibration algorithm that we use to find parameters are originally very sensitive to noise. What RANSAC can do is just eliminate outliers but not eliminate general noise. So for this calibration algorithm, what every iteration of the RANSAC does is selecting some “noise points” to find H. Although the noise is small, but they're large enough to make the sensitive calibration algorithm failed. Since all points are “deadly noise point” for this calibration algorithm, RANSAC can do nothing about it. That' why RANSAC cannot handling well with gaussian noise for this coplanar calibration algorithm.

C)

```
image 0:
w=0.917355, k=3.739008, # of inliers=111, average inlier distance error=0.000144
image 1:
w=1.000000, k=0.000000, # of inliers=121, average inlier distance error=0.000069
image 2:
w=1.000000, k=0.000000, # of inliers=121, average inlier distance error=0.000059
0th H:
[-0.91105956, 1.26071680, 182.70018005;
  0.85495025, 0.42747524, -67.01151276;
  -0.00070175, -0.00035088, 1.00000000]
1th H:
[-1.14384365, 1.04027700, 259.25888062;
  0.58434820, 0.43826103, -25.61271858;
  -0.00072727, -0.00054545, 1.00000000]
2th H:
[-0.56874096, 1.41147220, 100.66010284;
  1.05276453, 0.26319149, -61.28648758;
  -0.00063492, -0.00015873, 1.00000000]
Kstar:
[1304.35412598, 0.00001756, 319.99911499;
  0.00000000, 1304.35229492, 239.99250793;
  0.00000000, 0.00000000, 1.00000000]
0th RstarTstar:
[-0.44721359, 0.89442718, -0.00000009, -89.44220734;
  0.66666389, 0.33333209, -0.66667026, -199.99539185;
  -0.59628791, -0.29814422, -0.74535292, 849.71020508]
1th RstarTstar:
[-0.60000032, 0.79999977, -0.00000031, -39.99942017;
  0.49975309, 0.37481487, -0.78087169, -174.90991211;
  -0.62469703, -0.46852344, -0.62469137, 858.95959473]
2th RstarTstar:
[-0.24253586, 0.97014242, 0.00000022, -145.52081299;
  0.79955375, 0.19988863, -0.56635529, -199.88430786;
  -0.54944539, -0.13736121, -0.82416117, 865.37579346]
for image0, the error is 2.4391 pixels.
for image1, the error is 0.0001 pixels.
for image2, the error is 0.0001 pixels.
Program ended with exit code: 0
```

Image 1 has impulsive noise

We can see that: RANSAC handles very well on impulsive noise. Because the inliers have no deviation in them, after eliminating the outliers, RANSAC gives “pure” input data to find-H function, and it finds the “correct” H. Without RANSAC, using all 121 points to find H results to a bad matrix K^* (large s, negative u_0 and v_0, large difference of a_u and a_v).

The “calibrationCamera” function in OpenCV consider radial distortion coefficients, but there's no s in its K^* matrix, so its s always equals to zero. I think it use different algorithm from my program. I test its outputs with above three types of inputs and find: if there's no noise in the points, the calibrationCamera() function does very good job. But if there's noise in it, it does not.

Here are the results of the calibrationCamera() function with noise points:

```
Kstar:
[1306.87353516, 0.00000000, 316.30734253;
  0.00000000, 1299.74621582, 233.01281738;
  0.00000000, 0.00000000, 1.00000000]
distCoeffs:
[-0.2378317616938083, 15.8176798528202269,
0.0026572748478269, -0.0013579030129155,
-239.5976996866248498]
0th RstarTstar:
[-0.44690990, 0.89457345, 0.00314369, -87.56484222;
  0.66529047, 0.33471018, -0.66735125, -196.36468506;
  -0.59804696, -0.29615444, -0.74473649, 851.20446777]
1th RstarTstar:
[-0.60175800, 0.79867649, -0.00178459, -37.56243896;
  0.49928686, 0.37443879, -0.78135026, -171.33378601;
  -0.62337786, -0.47107479, -0.62409019, 860.35632324]
2th RstarTstar:
[-0.24404356, 0.96976161, -0.00227262, -143.07174683;
  0.80116504, 0.20029414, -0.56392986, -196.40650940;
  -0.54642230, -0.13944419, -0.82581955, 866.44543457]
for image0, the error is 1.9422 pixels.
for image1, the error is 0.1043 pixels.
for image2, the error is 0.1091 pixels.
Program ended with exit code: 0
```

gaussian noise

```
Kstar:
[1315.22839355, 0.00000000, 264.61715698;
  0.00000000, 1306.93359375, 249.81231689;
  0.00000000, 0.00000000, 1.00000000]
distCoeffs:
[-2.3420582215155408, 68.7034356062757467,
0.0015794228693243, -0.0032361719830112,
-547.3679401212530138]
0th RstarTstar:
[-0.48013029, 0.87640136, -0.03735793, -50.53727341;
  0.67065722, 0.33929762, -0.65961808, -206.98146057;
  -0.56541467, -0.34175697, -0.75067198, 845.52868652]
1th RstarTstar:
[-0.62499803, 0.78020668, -0.02559327, -4.02535009;
  0.50742847, 0.38113332, -0.77282196, -182.55142212;
  -0.59320641, -0.49599895, -0.63410664, 851.19598389]
2th RstarTstar:
[-0.26494288, 0.96363932, -0.03470706, -108.94016266;
  0.80828720, 0.20231543, -0.55293781, -207.61557007;
  -0.52581084, -0.17455022, -0.83249938, 859.34191895]
for image0, the error is 5.1709 pixels.
for image1, the error is 2.9029 pixels.
for image2, the error is 2.9808 pixels.
Program ended with exit code: 0
```

impulsive noise

We can see that: the K^* matrix is not accurate as the one we computed. The square errors and the disCoeff parameters are not small. Maybe the calibrationCamera() function trend the noise as radial lens distortion.

Now we test the algorithm with real image (token from camera):

```
image 0:
w=0.800000, k=8.739211, # of inliers=32, average inlier distance error=0.522524
image 1:
w=1.000000, k=0.000000, # of inliers=40, average inlier distance error=0.753699
image 2:
w=0.825000, k=7.401153, # of inliers=33, average inlier distance error=0.487396
0th H:
[-0.13225105, 0.62260693, 896.13641357;
  0.28438964, -2.76660395, 652.30963135;
  0.00225306, -0.00013450, 1.00000000]
1th H:
[-2.09173322, 1.26234138, 948.60375977;
  -0.65645254, -3.00491691, 633.79113770;
  0.00118466, 0.00006613, 1.00000000]
2th H:
[-3.03793955, -1.37081575, 1108.89782715;
  1.44044924, -2.49915242, 519.01385498;
  -0.00048354, 0.00053137, 1.00000000]
Kstar:
[1022.73297119, 0.41321588, 614.90515137;
  0.00000000, 1024.51037598, 364.30538940;
  0.00000000, 0.00000000, 1.00000000]
0th RstarTstar:
[-0.53991622, 0.25134224, 0.80071384, 100.02202606;
  -0.19052652, -0.96525925, 0.17964379, 102.29545593;
  0.81987202, -0.04894218, 0.56904650, 363.89309692]
1th RstarTstar:
[-0.86613458, 0.37563527, 0.33873147, 102.46798706;
  -0.33362758, -0.92879868, 0.15778995, 82.63366699;
  0.37216067, 0.02077422, 0.92978692, 314.15036011]
2th RstarTstar:
[-0.85152680, -0.52698159, -0.04364610, 153.43136597;
  0.50130135, -0.83500266, 0.22470474, 47.97428513;
  -0.15361981, 0.16881433, 0.97520375, 317.69528198]
for image0, the error is 0.8718 pixels.
for image1, the error is 0.7537 pixels.
for image2, the error is 0.8125 pixels.
Program ended with exit code: 0
```

The image centre should be (640, 360); The a_u and a_v should be 1024. Here we can see it can get some reasonable result, but seems not that accurate. I think part of the reason comes from the camera itself or the accuracy of location of points when detected. Actually, the result is quiet unstable when using real image. Sometimes it get “good” result as above, sometimes the results are bad (large s). Except the reason that the data may have different kind of noise in it, I think the other reason is: the function I use to decide which H model is better. It's a hard question to find a perfect compare function to do it. My compare function is the betterModel(), check it if you want.

By the way, this is the result of real image from OpenCV calibrationCamera function:

```
Kstar:
[1037.44348145, 0.00000000, 635.02905273;
  0.00000000, 1057.39526367, 414.25741577;
  0.00000000, 0.00000000, 1.00000000]
distCoeffs:
[0.2894361005777911, -2.5965636821758018,
0.0166793922349426, 0.0303364346587954,
5.9673806251527237]
0th RstarTstar:
[-0.55746150, 0.24823517, 0.79222214, 94.25663757;
  -0.22276314, -0.96398312, 0.14530347, 84.06668854;
  0.79975826, -0.09547681, 0.59268117, 389.80776978]
1th RstarTstar:
[-0.85372984, 0.37326023, 0.36307329, 95.72570038;
  -0.33981055, -0.92768604, 0.15468474, 66.25609589;
  0.39455569, 0.00868284, 0.91883099, 330.34405518]
2th RstarTstar:
[-0.84276658, -0.53400767, -0.06767775, 147.75520325;
  0.51602960, -0.83729452, 0.18069679, 31.54751205;
  -0.15315968, 0.11736150, 0.98120761, 338.59799194]
for image0, the error is 7.7956 pixels.
for image1, the error is 9.3515 pixels.
for image2, the error is 8.6017 pixels.
Program ended with exit code: 0
```

Here we can see: the square error is large. And the parameters are not that accurate. After several experiments: if the OpenCV function or my program is more accurate is depends on the input data points.

Summary: This coplanar calibration algorithm is very sensitive to noise, making that RANSAC algorithm cannot handle well if all points have noise in it. What RANSAC can do is eliminate outliers but not eliminate general derivations. So if it's impulsive noise, RANSAC can handle it well and improve the accuracy of the algorithm. The calibrationCamera() function in OpenCV uses different algorithm and calculate different parameters, so it's useless to compare with it if the input points have noise or derivation in them.

5. References