ETH ZURICH, D-INFK

Computer Vision Exercise 2: Camera Calibration

Xingchang Huang

October 11, 2017

1 CAMERA CALIBRATION

The objective of exercise 2 is to implement our camera calibration method using DLT (Direct Linear Transform) and Golden Standard Algorithm in MATLAB, in order to find the intrinsic and extrinsic parameters (matrix) for the camera.

1.1 POINTS CHOOSING

With a chessboard photo taken from the camera, I first need to click at least 6 points to solve the equation system in DLT. I follow the world coordinate shown in tutorial and see each grid as 1 length of coordinate. The chosen points are shown in Figure 1.1. In order to avoid some noises when choosing points and typing the coordinate manually, I have chosen 6 points carefully and made multiple choices. In this way, the result will be more stable and I found that the chosen points are quite essential for the calibration performance.

1.2 DATA NORMALIZATION

After choosing points, I have to move the points to the origin of world coordinate. Thus, a matrix multiplication should be done. First, I calculate the centroid of chosen points and every point should subtract it. Then I have to normalize the average distance to the origin with a scale. My matrix is constructed as the following. T and U matrix are similar except for the dimension. I have checked that the average distances from points to the origin are $\sqrt{2}$ and $\sqrt{3}$ for image points and space points, respectively. After multiplied by the matrix, space points and image points are normalized around the origin.

If the normalization has not be done, the following result would be influenced and incorrect.

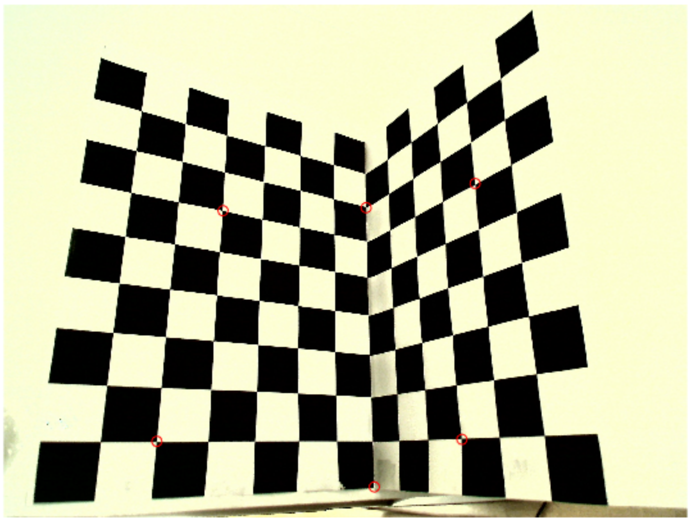

Figure 1.1: 6 Chosen Points for Calibration

In my mind, I suppose that it is because I clicked 6 points and give the coordinate (X,Y,Z) to them myself, whose order of magnitude is far different from the coordinates of image points. The problem when I ran it is that the re-projection points all disappeared and cannot be visualized.

$$T = \begin{bmatrix} S_{2D} & 0 & -S_{2D} * centroid_x \\ 0 & S_{2D} & -S_{2D} * centroid_y \\ 0 & 0 & 1 \end{bmatrix}$$

1.3 Direct Linear Transform

This algorithm uses the property of cross product and then change the variable to matrix P so that we can get elements of P matrix by solving the equation system. First I construct a matrix A, which has $2n$ (equal to 12 here) rows and 12 columns representing all the chosen points and their relevant values for rows. Rather than solve the equation directly, which is not realistic, we use SVD decomposition to get P because this is a minimization problem such that we should $min||Ax||$. Because zero vector should be an answer, we should add constraint $||x|| = 1$. Then from the properties of norm in SVD described in the reference book, we can deduce that the last column of matrix V, which is obtained from SVD(A), should be the minimal value of P. However, this is still not optimized well. So we should use golden standard algorithm to optimize the geometric error further. Also, note that the matrix P should be denormalized before decomposed into intrinsic matrix and extrinsic matrix.

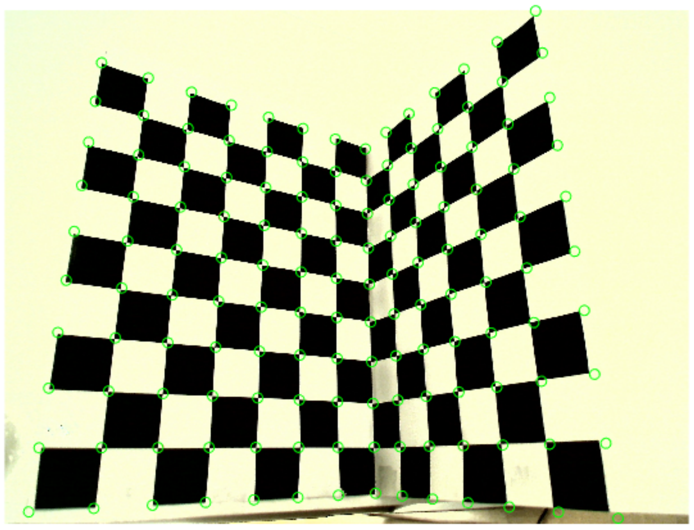

Figure 1.2: DLT Calibration Result. The re-projection error is 1.8939.

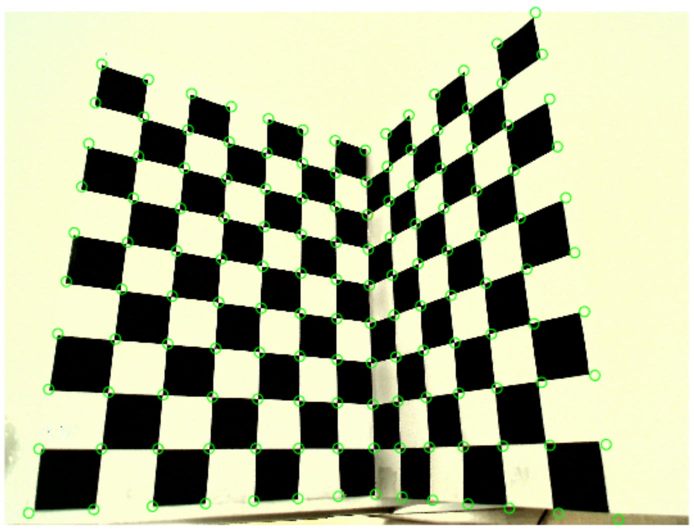

Figure 1.3: Golden Standard Algorithm Calibration Result. The re-projection error is 1.7769.

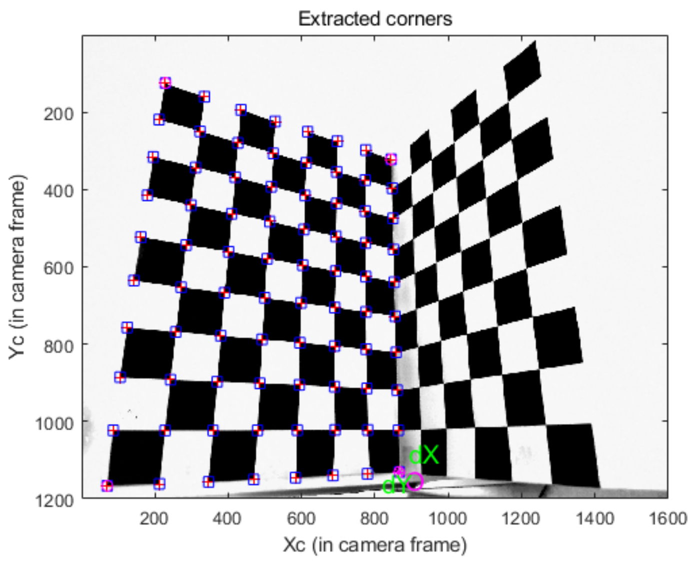

Figure 1.4: Chosen Rectangle and Points using Camera Calibration Toolbox.

1.4 Golden Standard Algorithm

In MATLAB, there is a fminsearch function for searching the minimal value of function and its variables. Thus, I just need to define the cost function as geometric error and initialize the P matrix using DLT above. After optimization, the result looks better and the error is lower than just using DLT. The comparison of DLT and Golden Standard has been shown in Figure 1.2 and 1.3, including the re-projection errors.

1.5 Bouget Calibration Toolbox

This toolbox is used for camera calibration shown in GUI, which is more convenient to use for us. The step should be the following. First, make sure you are in the right path containing images (.jpg). Then run the calib_gui and select images from image names and formats. Next extract the grid corners for further calibration. The extraction of point is kind of different from my previous method by clicking 4 extreme points. Because I can not find a plane by choosing 4 extreme points, I choose to click 4 extreme corners in 1 of 2 planes, as shown in Figure 1.4. Also, I have used the 3D calibration in the examples of toolbox and make a comparison with each other as shown in Figure 1.6. The last step is calibration and we can see the re-projection results, optimized intrinsic parameters and pixel errors. Therefore, the re-projection of the first one is shown in Figure 1.5.

Also, I have used my own camera and printed pattern for the camera calibration toolbox and the calibration result is shown in Figure 1.7.

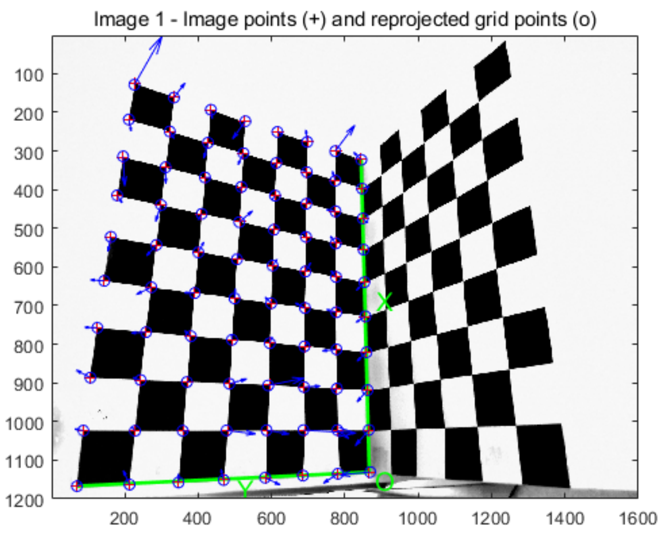

Figure 1.5: Re-projection of points, the shown Pixel Error by the toolbox is [0.05912, 0.04146].

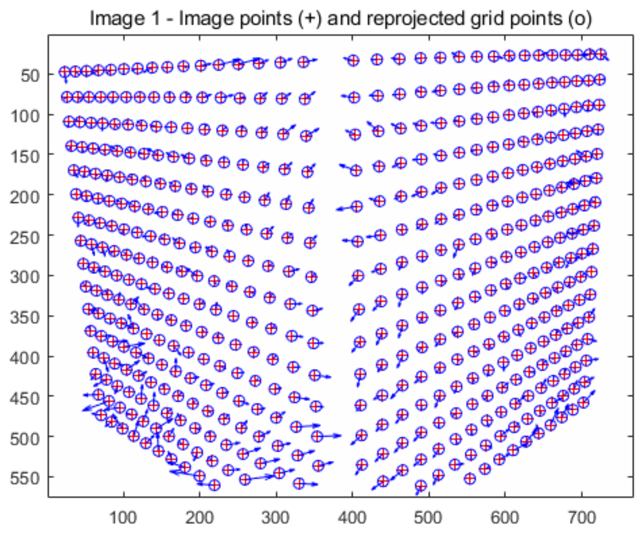

Figure 1.6: Calibration Toolbox for 3D result.

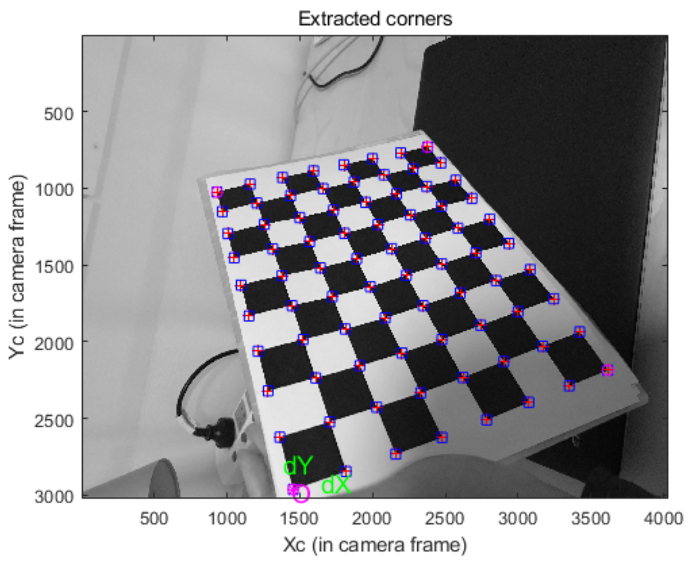

Figure 1.7: Calibration Toolbox using my own camera.

2 Bonus

Here, I should take the radial distortion into account as well, which is represented as k1, k2 and r. My way to optimize this distortion factor is that for each loop I first optimize the P matrix, and then optimize k1, k2 with initial values (both -0.1), iteratively. The decomposition of P in the previous step would be useful here as we need the camera center or principle center to compute the distortion factor. For each point, we should get the radial distortion factor r using the image points and principle points, where $r = -R \times C$. My result is shown in Figure 2.1 with no obvious optimization. All the details of implementation are in the code with comments.

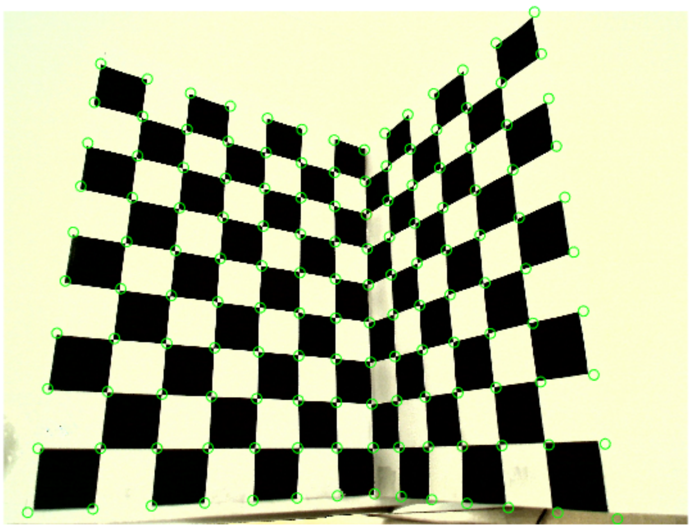

Figure 2.1: Calibration considering Radial Distortion.