Homework 1 - AutoCalib Report

Ameya Patil
Department of Computer Science
University of Maryland
College Park, Maryland 20740

I. Introduction

The aim of this task was to calibrate a camera using the technique described by Zhang. 13 images of a checkerboard captured from different perspectives, from a focus locked camera, were used for the purpose.

II. Implementation Details

A. Deciding the representation of object points

The coordinates of the corners in the model image, in world coordinates could be expressed in many ways depending on how large or how small we wanted the squares to be. The only restriction that had to be imposed was that the z coordinate of all those points should be zero since we are assuming the model image to be placed on the XY plane. So one possibility was to consider a digital image of the chessboard at some arbitrary magnification level, and apply the same procedure to detect the chessboard corners. Other possibility was to use the physical size of the chessboard box provided, and create a hypothetical grid and thus populate the corners of the model image. The guess was that either approach would work, that we just had to be consistent across all the 13 images.

It turned out that the guess was right, using either approach led to similar results for the camera parameters. The second approach was chosen because having a unit of measurement for the corner position meant we could ascribe some meaning to the computed camera parameters and errors.

cv2.findChessboardCorners() was used for detecting corners in the 13 images and cv2.cornerSubPix() was used to refine the detection.

B. Selection of points for the homography

Homography was estimated using cv2.getPerspectiveTransform() initially so as to ensure correctness of the remaining procedure. Once the remaining procedure was verified, homography was also computed using the LM optimization algorithm with initial estimate given by the DLT algorithm. Since only 4 corners are sufficient for the initial estimate and for the opencv API, the 4 corner points of the chessboard grid were chosen.

C. Scikit LM Optimize API

Using the scikit API for LM optimization took some time to understand because of the way it requires the cost function to return an array of error values to minimize - called a residual array. After that, there was a little confusion about which norm is to be used exactly, as most of the references used the notation for L1 norm. The source code of opencv camera calibration API was referred and L2 norm was used. Further, the initial estimate provided to the cost function has to be 1D array, and the parameters to be optimized were 2D matrices (K, ks). This meant some packing and unpacking had to be done to get the data in the required format.

D. Parameters to optimize

The initial implementation was trying to optimize all 9 values of the camera matrix(K). This resulted in the final refined matrix having non-zero values in the lower triangle. This is not desirable since it would give an incorrect estimate of the actual camera parameters. So later revisions ensured that only the upper triangle of the K matrix was optimized. This change also resulted in lower loss at convergence in the LM algorithm.

E. Distortion

As mentioned on the homework page, initial estimate for radial distortion parameters(ks) was assumed to be 0 and no tangential distortion was assumed at all. The estimated distortion parameters were also found out to be very small and so the rectification of the image did not result in any significant change. Visually, no distortion effect was visible in any of the images, so the results made sense. Rectification was performed using the opencv cv2.undistort() API

F. Results

Initial estimates:

$$K = \begin{vmatrix} 2050.24 & 1.36 & 753.28 \\ 0 & 2041.98 & 1368.86 \\ 0 & 0 & 1 \end{vmatrix}$$

$$ks = \begin{vmatrix} 0 & 0 \end{vmatrix}$$

After refinement using LM optimization:

$$K = \begin{vmatrix} 2048.66 & -6.68 & 723.28 \\ 0 & 2036.5 & 1361.96 \\ 0 & 0 & 1 \end{vmatrix}$$

$$ks = \begin{vmatrix} 3.38 * 10^{-8} & -3.46 * 10^{-14} \end{vmatrix}$$

Reprojection error after image rectification: 2.26 per pixel

The image dimensions are 1512 x 2688 (w x h) and the estimates pretty much convey the same

G. Images

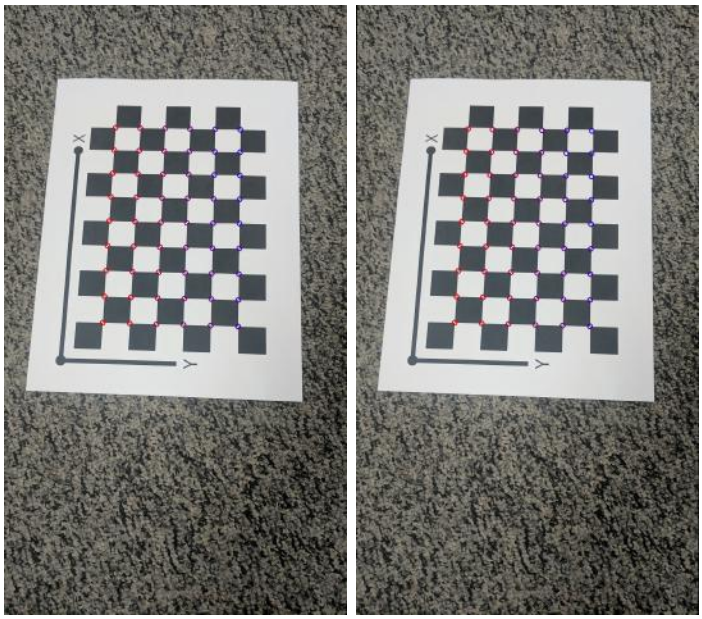
As mentioned before, since the distortion parameters after the refinement step were very close to 0, meaning that there was not much radial distortion in the images captured and there was negligible difference between the original and rectified images. Further, in all the rectified images with reprojected corners, the corners marked in blue are slightly off, which is probably the reason for the reprojection error

III. Conclusion

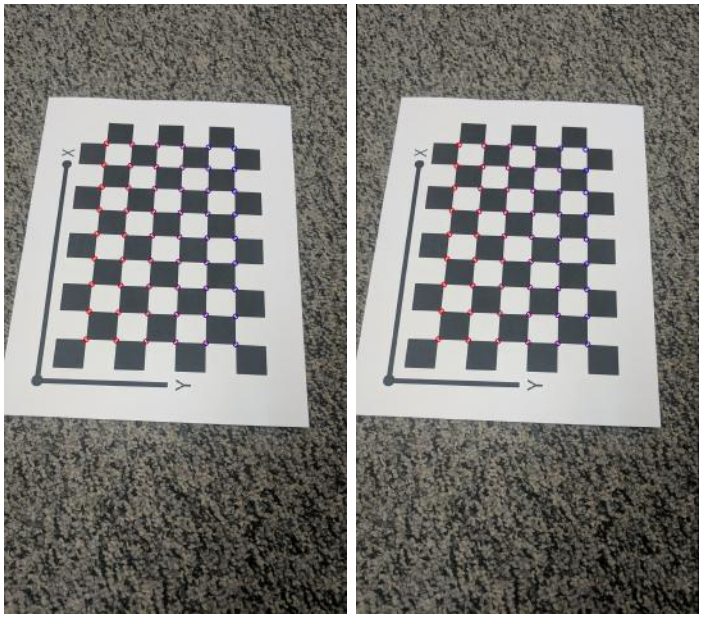
This task has helped in gaining an understanding of camera calibration, why it is essential and also the ingenuous simplifications used by Zhang to make the process faster.

References

[1] https://www.cs.umd.edu/class/spring2016/cmsc426/lectures/camera-calibration.pdf
[2] https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf
[3] https://prateekvjoshi.com/2014/05/31/understanding-camera-calibration/
[4] https://opencv-python-tutroals.readthedocs.io/en/latest/pytutorials/pycalib3d/pycalibration/pycalibration.html
[5] https://webserver2.tecgraf.puc-rio.br/ mgattass/calibration/zhanglatex/zhang.pdf
[6] https://lmfit.github.io/lmfit-py/fitting.html
[7] http://staff.fh-hagenberg.at/burger/publications/reports/2016Calibration/Burger-CameraCalibration-20160516.pdf
[8] https://opencv-python-tutroals.readthedocs.io/en/latest/pytutorials/pycalib3d/pycalibration/pycalibration.html

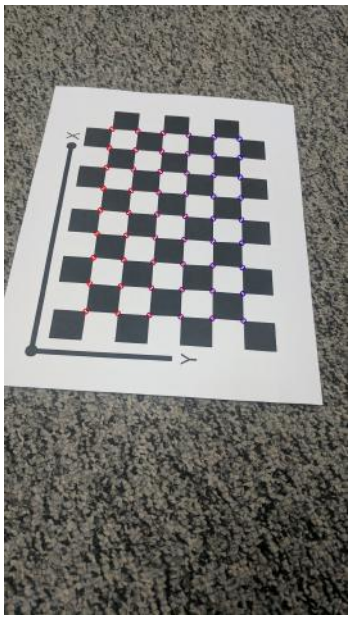


(a) Original image(left) with corners and rectified image(right) with reprojected corners

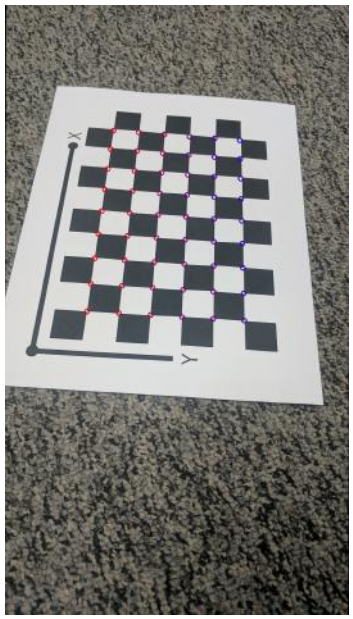


(b) Original image(left) with corners and rectified image(right) with reprojected corners

Fig. 1. Results for 1st image (top) and 2nd image (bottom)

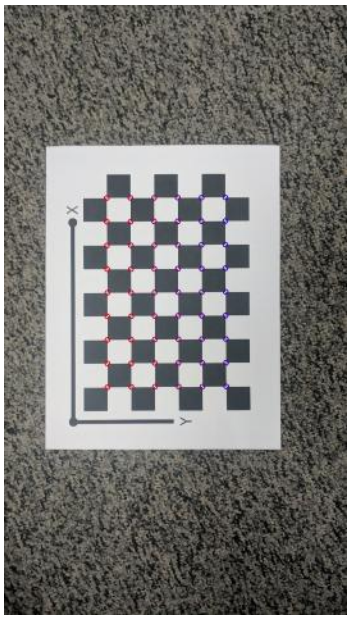


(a) Original image(left) with corners and rectified image(right) with reprojected corners

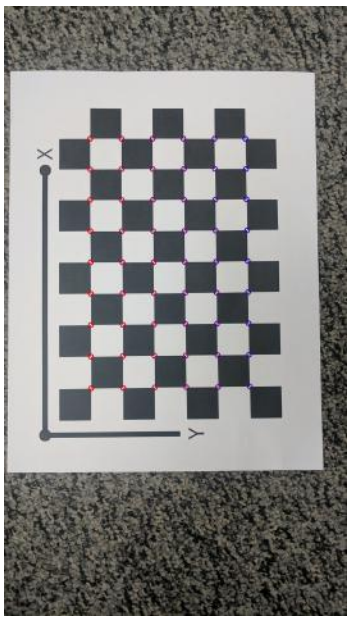


(b) Original image(left) with corners and rectified image(right) with reprojected corners

Fig. 2. Results for 3rd image (top) and 4th image (bottom)

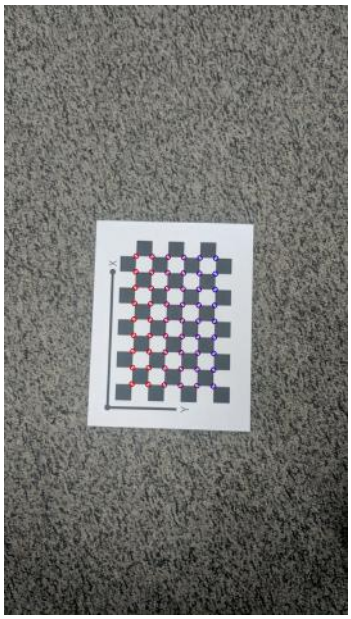


(a) Original image(left) with corners and rectified image(right) with reprojected corners

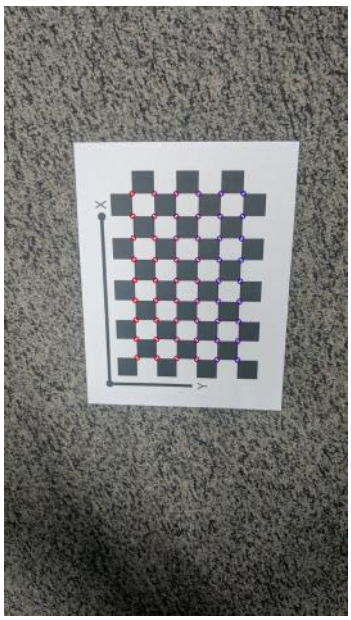


(b) Original image(left) with corners and rectified image(right) with reprojected corners

Fig. 3. Results for 5th image (top) and 6th image (bottom)

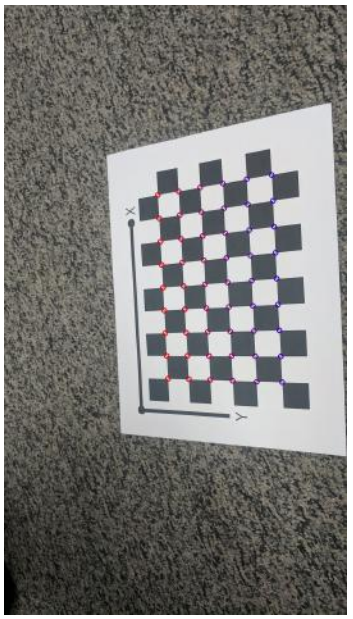


(a) Original image(left) with corners and rectified image(right) with reprojected corners

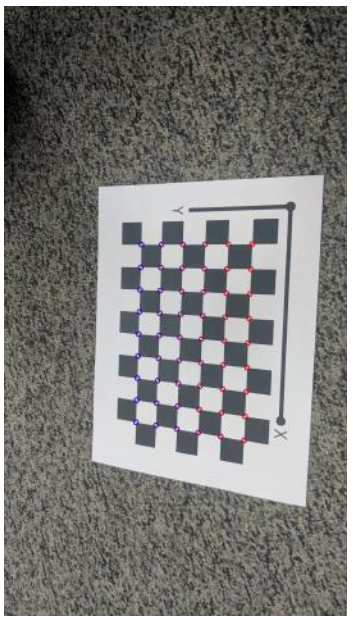


(b) Original image(left) with corners and rectified image(right) with reprojected corners

Fig. 4. Results for 7th image (top) and 8th image (bottom)

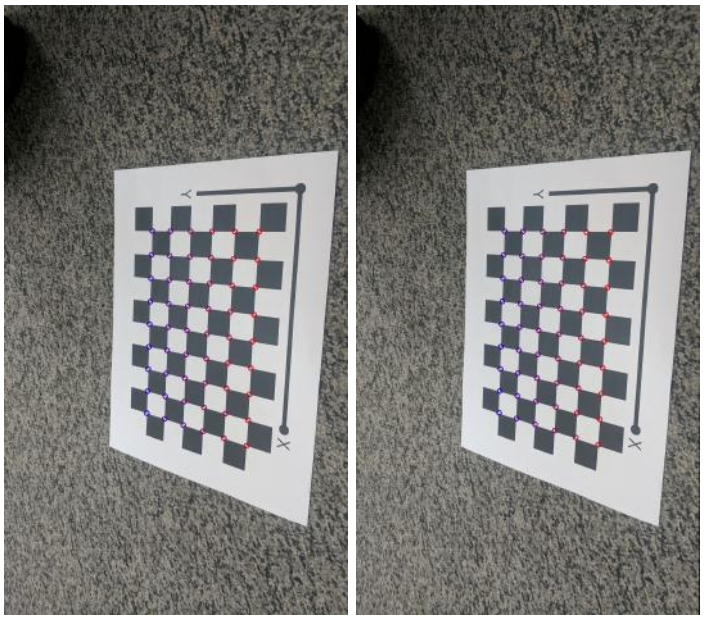


(a) Original image(left) with corners and rectified image(right) with reprojected corners

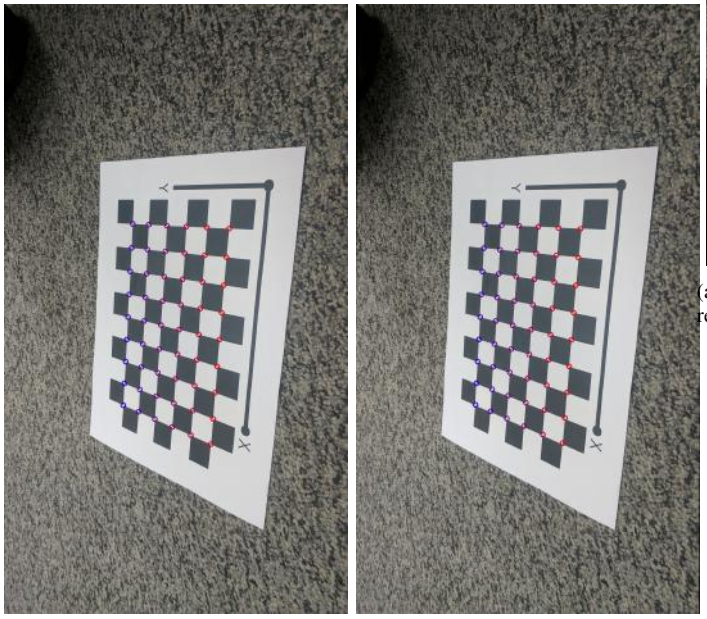


(b) Original image(left) with corners and rectified image(right) with reprojected corners

Fig. 5. Results for 9th image (top) and 10th image (bottom)

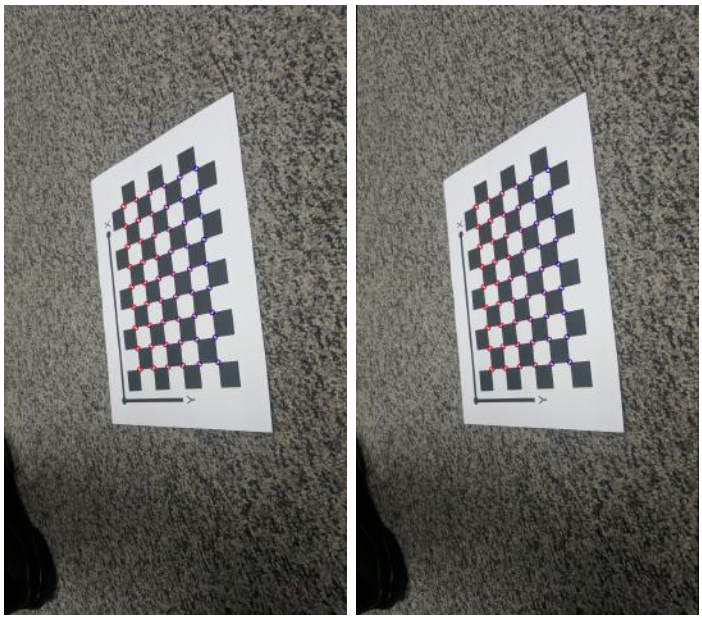


(a) Original image(left) with corners and rectified image(right) with reprojected corners



(b) Original image(left) with corners and rectified image(right) with reprojected corners

Fig. 6. Results for 11th image (top) and 12th image (bottom)



(a) Original image(left) with corners and rectified image(right) with reprojected corners

Fig. 7. Results for 13th image