ECE661 Fall 2024: Homework 8

TA: Rahul Deshmukh (deshmuk5@purdue.edu)

Due Date: Midnight, 09 Nov 2024

Late submissions will be accepted with penalty: -10 points per-late-day, up to 5 days.

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. This can be a challenging homework, **start early**!

1 Theory Question

1. Why is the following theoretical observation fundamental to Zhang's algorithm for camera calibration?

   If π denotes the plane that contains the calibration pattern, we can show that π samples the Absolute Conic Ω_∞ at exactly two points that are the Circular Points for π.

2. As you might suspect, the image of the Absolute Conic Ω_∞ on the camera sensor plane is also a conic that is typically denoted ω. How would you derive the algebraic form of ω from Ω_∞? Can you prove that ω does not contain any real pixel locations?

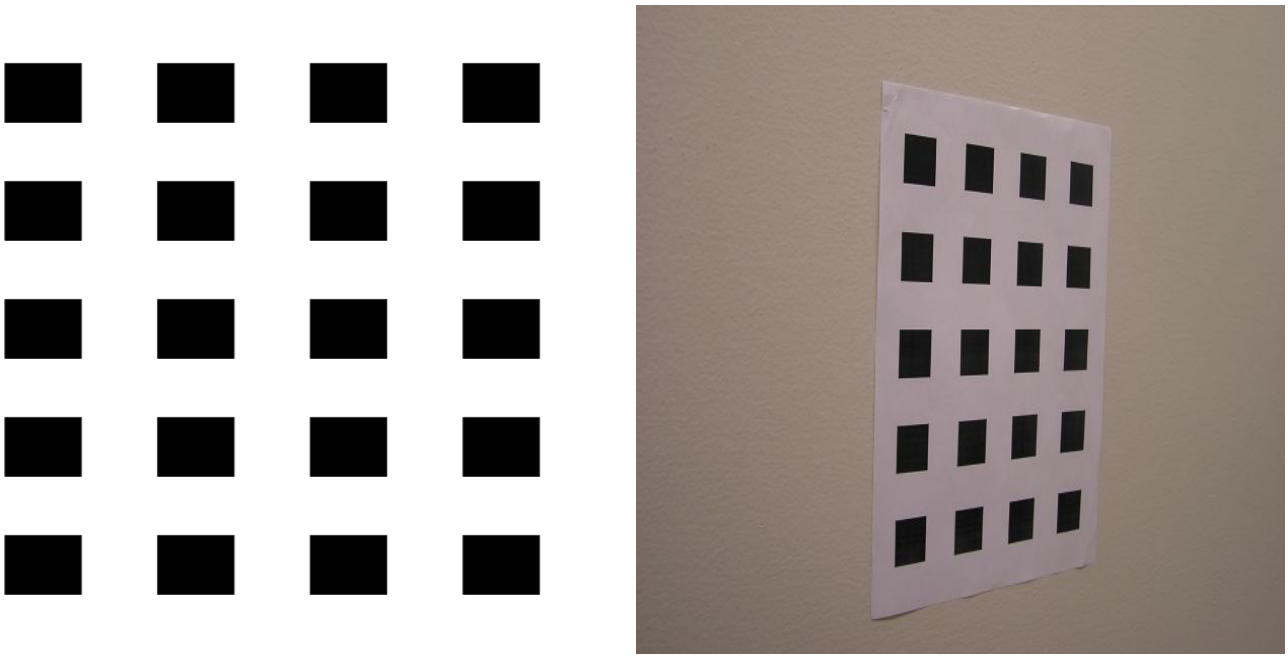
2 Introduction

The goal in this homework is to implement the popular Zhang's algorithm for camera calibration. A formal description of the algorithm can be found in the Zhang's technical report [1].

For this assignment, you can assume the camera to be a pin-hole camera. This implies that a complete calibration procedure will involve estimating all the 5 intrinsic parameters and the 6 extrinsic parameters that determine the position and orientation of the camera with respect to a reference world coordinate system. For this you need to to establish correspondences between image points and their world coordinates. To this end, you will use the provided checkerboard pattern consisting of alternating black and white squares, as shown in Fig. 1(a). We will be using the corners of these squares in our calibration procedure.

3 Programming Tasks

You will use two datasets for this homework. The provided dataset contains 40 images of the calibration pattern taken from varying viewpoints and orientations. Additionally, you will create the second dataset



(a) Calibration Pattern (b) Example View

Figure 1: The checkerboard calibration pattern and an example view in the provided dataset.

on your own. Your assignment consists of the following steps:

3.1 Creating Your Own Dataset

- Print out the calibration pattern and mount it on a wall or a large piece of cardboard. Now choose one of the corners on the pattern as your world origin and measure the world coordinates of all the other corner points on the pattern with a ruler. Number the corners appropriately. A particular corner should get the same number label in all the images. (This does not mean that you need to write any numbers on the actual calibration pattern).

- For the very first image of the calibration pattern on the wall, position the camera such that its Principal Axis is approximately perpendicular to the plane of the wall on which you mounted the calibration pattern. Also make sure that the x-axis of the image is very roughly along the horizontal axis of the calibration pattern and the y-axis of the image very roughly along the vertical axis of the pattern. These conditions are meant to be satisfied only approximately. Despite the approximations involved, the distance between the camera and calibration pattern that you can measure manually will serve as a check on your calculations of the calibration parameters. In the following discussion, this image will be referred to as 'Fixed Image'.

- Now move your camera in different directions and capture images of the calibration pattern. Obviously you will need to rotate/tilt the camera in order to capture the calibration pattern from different positions. A minimum of 20 different poses of the camera is required for good camera calibration.

3.2 Zhang's Algorithm

Implement the following steps for each of the datasets.

3.2.1 Corner Detection

- Extract edges using the Canny edge detector. You can use any open-source implementation of Canny edge detector like the `cv2.Canny` function from OpenCV, or `skimage.feature.canny` from scikit-image.
- Fit straight lines to the edges using the Hough transform. You can again use any open-source implementation of the Hough line transform like the `cv2.HoughLines` or the more efficient `cv2.HoughLinesP` functions from OpenCV. Note that the `skimage.transform` module also has the equivalent function calls.
- The corners will be the intersection points of these lines.
- Depending on the accuracy of your corner detection, you might wish to improve your results. You can refer to the previous year solutions for improving this accuracy. However do note that it is not necessary to detect 100% of the true corners in every image. Since you will be using the Levenberg-Marquardt (LM) non-linear optimization to refine your calibration, you should have robust calibration as long as you detect sufficiently high number of corners with good accuracy in each image.
- Assign labels to the corners using the same numbering scheme that you used in the previous section. These labels should be indicated on every output image in your report.

Proceed further only when you are sure that your corner detection algorithm is working correctly.

3.2.2 Camera Calibration

- Establish correspondences between the extracted corners in each image and their world coordinates.
- Implement Zhang's calibration algorithm.
- Use the Levenberg-Marquadt algorithm for non-linear optimization. You can either use the `scipy.optimize.least_squares` function or your own implementation from your previous homeworks.
- To measure the accuracy of your camera-calibration, reproject the corner points from two or more views back into the 'Fixed Image'. You can do a visual comparison of the locations of the original corners vis-a-vis the reprojected corner points. In each of the images, measure the reprojection error for each point using the Euclidean Distance measure. Calculate an estimate of the mean and variance of this error.

- Show at least two image pairs where one can see the improvement of your calibration estimate by using the LM optimization.
- Compare your estimated camera pose for the 'Fixed Image' with the measured ground-truth.

3.3 Plotting the Camera Poses

In Lecture-18 and 19 you learned how to model the camera and some properties of the projection matrix respectively. For the camera calibration calculations you must have used the world coordinate frame as the coordinate system for your world points and for the camera extrinsics. However, as you will see shortly, to plot the camera poses you will be using the camera frame as opposed to the world coordinates.

To plot the camera pose, you need two things – (1) The camera center, and (2) The directions of the camera's coordinate axes (i.e. $X_{cam}, Y_{cam}, Z_{cam}$ axis). In Lecture-18, we proved that the camera center is given by Eq. 1. Now all that is needed is to compute the directions of the camera's coordinate axes. The task of finding the axis directions is the same as finding the global coordinates of the points which are **1 unit** away from the camera center along the three camera axes. These three points are given by Eq. 4.

Let's say you are given a camera with the pose parameters $(\boldsymbol{R}, \boldsymbol{t})$. From Lecture-18, we know that Eq. 2 can be used to convert any world coordinate to camera frame 3D coordinate. Using this equation we can obtain the reverse conversion as shown in Eq. 3.

Finally, using Eq. 3 you can convert the three points shown in Eq. 4 and get their corresponding world coordinates. You are now ready to plot the camera pose.

Plot three arrows for the camera axis with different colors (X_{cam} - red, Y_{cam} - green, Z_{cam} - blue) for the three axes. Also plot the camera principal plane using a small 3D rectangle. Assign random colors to each camera and make sure the 3D rectangle is semi-transparent. See Fig. 2 for example plot. Plot all the camera poses in a single plot. Also try plotting the camera calibration pattern in this 3D plot. The camera calibration pattern is a grid of black rectangles on the Z=0 plane.

NOTE: Make sure that your camera axes and principal plane are scaled properly so as to discern them easily.

$$\boldsymbol{C} = -\boldsymbol{R}^T\boldsymbol{t} \tag{1}$$

$$\boldsymbol{X}_{cam} = \boldsymbol{R}\boldsymbol{X} + \boldsymbol{t} \tag{2}$$

$$\begin{aligned} \boldsymbol{X} &= \boldsymbol{R}^{-1}\boldsymbol{X}_{cam} - \boldsymbol{R}^{-1}\boldsymbol{t} \\ &= \boldsymbol{R}^T\boldsymbol{X}_{cam} - \boldsymbol{R}^T\boldsymbol{t} = \boldsymbol{R}^T\boldsymbol{X}_{cam} + \boldsymbol{C} \end{aligned} \tag{3}$$

$$\boldsymbol{X}_{cam}^{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \boldsymbol{X}_{cam}^{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \boldsymbol{X}_{cam}^{z} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4}$$

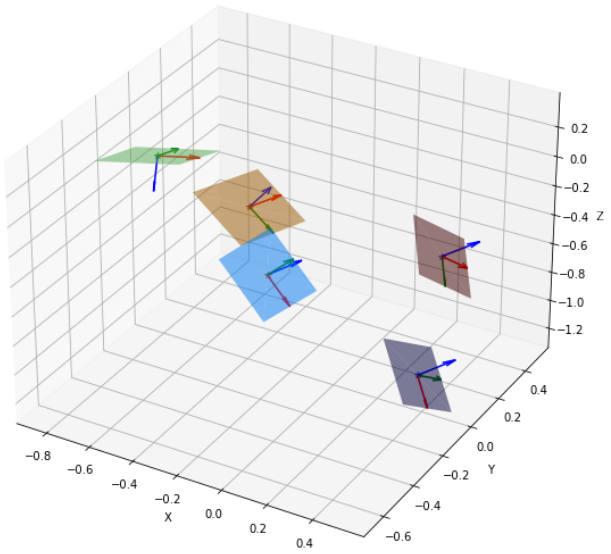


Figure 2: 3D plot showing random camera poses for five cameras

3.4 Extra Credits (10 points)

- Estimate the radial distortion parameters $\boldsymbol{k} = [k_1, k_2]^T$. You can refer to page 6 of Lecture 21 or Section 3.3 of Zhang's report [1].
- Quantitatively demonstrate good reduction in the reprojection error by using the LM optimization simultaneously on all the intrinsic, extrinsic and radial distortion parameters. Show two image pairs with and without considering the radial distortion parameters. Tabulate the improvement of your calibration estimate by including the radial distortion parameters in terms of the reprojection error.

4 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. Turn in the following: (a) a typed self-contained pdf report with source code and results and (b) zipped source code files (only .py files are accepted). Rename your .zip file as hw8_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.

There should be two items in your submission - (1) Homework PDF (2) ZIP file containing source code (*.py files) and text files (if any).

2. Your pdf must include a description of

   - Your answer to the theoretical question in Sec. 1.
   - A clear description of how you implemented each of the given programming tasks, with relevant equations.
   - For each dataset, you should:
     - Show at least two output images for the edge-detection, Hough line fitting and corner detection steps. Clearly label the detected corners so that the reader should be able to visually verify the correct correspondences.
     - In separate plots, show the Fixed Image with reprojected corners from at least two views, along with the original corners. This is to give a visual measure of the accuracy of your calibration procedure. Label the corners. Use different colors to differentiate the reprojected corners from the original corners.
     - Show at least two image pairs before and after using the LM optimization. The reprojected corners should appear visibly more accurate after LM.
     - Include your estimates for the intrinsic camera matrix $\boldsymbol{K}$ and for the external camera calibration matrix $[\boldsymbol{R}|\boldsymbol{t}]$ for at least two images.
     - Plot the camera poses as explained in Section 3.3.
     - For extra credits, show at least two image pairs where the improvement by considering $\boldsymbol{k}$ is visible (even slightly). Also tabulate the corresponding reductions in reprojection errors.
   - Your source code. Make sure that your source code files are adequately commented and cleaned up and are readable in the PDF as well.
   - Use lstlisting package to include your source code in the report.

3. To help better provide feedbacks to you, make sure to **number your figures**.

4. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

Homework File Size

For your homework submissions, please ensure that your reports (PDFs) are **under 10Mb**.

Some ways to reduce your report sizes are:

- Downsample your input/output image when including in the reports.
- When including plots in your reports, save them as PDFs instead of PNG/JPEG. PDFs are vector formats which are lightweight and do not pixelate when zooming into the plot.

Strike a balance between the file size and image quality displayed in your report. This will help your TA in easy distribution of homeworks for grading without maxing out the disk space. Moreover, this will help you when you upload your homeworks to your individual git repos. You don't want to upload 40 Mb PDF to your git repo!

Finally, do not include the images in your ZIP files for your homework submissions. Ideally, the ZIP file should be under 1Mb because it only contains ASCII (*.py /*.txt) files.

References

[1] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, December 2000. MSR-TR-98-71, Updated March 25, 1999. 1, 5