ECE661: Homework 9

Fall 2020
Due Date: Nov 02, 2020 (11:59 PM)

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace.

1 Theory Question

In Lecture 18, we showed that the image of the Absolute Conic Ω_∞ is given by $\omega = K^{-T}K^{-1}$. As you know, the Absolute Conic resides in the plane π_∞ at infinity. Does the derivation we went through in Lecture 18 mean that you can actually see ω in a camera image? Give reasons for both 'yes' and 'no' answers. Also, explain in your own words the role played by this result in camera calibration.

2 Introduction

The goal in this homework is to implement the popular Zhang's algorithm for camera calibration. A format description of the algorithm can be found in the Zhang's technical report [2].

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 can download and use the checkerboard pattern available in the given .zip archive. The checkerboard pattern consists of alternating black and white squares. We will be using the corners of these squares in our calibration procedure.

3 Programming Tasks

You will use two datasets for this homework. One dataset consisting of 40 images of the calibration pattern taken from varying viewpoints and orientations which is available in the given zip archive. You will create the second dataset on your own. Your assignment consists of the following steps:

3.1 Creating your own dataset

- Print out the calibration pattern that is available in the zip archive 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 very, very 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 cvCanny function from OpenCV. Scikit-image also has a function for the same, i.e., skimage.feature.canny.

- Fit straight lines to the edges using the Hough transform. You can again use any open-source implementation of Hough lines transform like the cvHoughLines or the more efficient HoughLinesP functions from OpenCV. Note that 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-Marquadt (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. The Levmar package [1] is a very good resource for this. It can be used with C++ and Python.
- To measure the accuracy of your camera-calibration, reproject the corner points from 4 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 (4 or more) images, measure the error for each point using the Euclidean Distance measure. Calculate an estimate of the mean and variance of this error.
- Show at least two images 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.

- **Extra credit tasks**
  1. Estimate the radial distortion parameters. You can refer to Section 3.3 of Zhang's report [2].
  2. Quantitatively demonstrate good reduction in error by using the LM optimization, on at least two images.

4 Submission Instructions

Include the results on both datasets in your report.

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

1. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files (.py, .cpp,.c). Rename your .zip file as hw9_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.

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, show at least two output images for the edge-detection, hough-line fitting and corner detection steps. Clearly label the detected corners as in it should be possible to visually verify the correct correspondences.
   - For each dataset, show at least three images (corresponding to 3 different views) for the reprojected corners and the original corners in the Fixed Image. This is to give a visual measure of the accuracy of your calibration procedure. Label the corners. Use different colours to differentiate the reprojected corners from the original corners.
   - Show at least two images to verify the accuracy obtained using the LM optimization.
   - Include your estimates for the intrinsic camera matrix (K) and for the external camera calibration matrix for at least four images.
   - Your source code. Make sure that your source code files are adequately commented and cleaned up.

References

[1] Levmar. URL https://users.ics.forth.gr/~lourakis/levmar/levmar.pdf.

[2] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, December 2000. URL https://www.microsoft.com/en-us/research/publication/a-flexible-new-technique-for-camera-calibration/. MSR-TR-98-71, Updated March 25, 1999.