TSBB09 Computer Exercise D
Camera Calibration 2

Code development: Andreas Robinson. Advice: Per-Erik For Forssén.
Exercise instructions: Andreas Robinson and Maria Magnusson.

Computer Vision Laboratory, Linköping University, Sweden

November 2018 - November 2022

Contents

1 Preliminaries **1**
1.1 Data 2

2 Calibration with OpenCV **2**
2.1 Accessing Python 2
2.2 Detecting calibration patterns 2
2.3 Calibrating 4
2.4 Analyzing OpenCV's calibration 5

3 Your implementation of Zhang's calibration method **6**

4 Validation of the OpenCV calibration **8**

1 Preliminaries

Before attending the computer exercise it is necessary to read through this guide to the exercise. The goal of this computer exercise is to get a complete understanding of Zhang's calibration method [?]. The guide contains one home exercise to be answered before the session. It is clearly marked with a pointing finger. The last step in Zhang's calibration method is nonlinear minimization. To get a better understanding, beyond this computer exercise, we recommend: `http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3215/pdf/imm3215.pdf`

A computer symbol indicates that an image has to be demonstrated during the computer exercise.
Important: You must be able to show the teacher all plots. It is a good idea associate your plots with the questions. E.g question 6 could have figure numbers 60, 61, 62 ... etc.

1.1 Data

All the data that you need for the computer exercise are located here:
`/courses/TSBB09/CameraCalibration2`

Make a new directiory `CameraCalibration2` on your home directory and copy the directories `matlab` and `python` there. The `data/input` and `data/validation` directory contains a lot of data (all the calibration images) and is better not to copy.

QUESTION 1: Look at all the images at `data/input` and check that they all contain a chessboard pattern. How many calibration images are there?

---

2 Calibration with OpenCV

2.1 Accessing Python

The OpenCV library has a camera calibration toolbox that should be sufficient for most camera calibration needs. You have been provided with some software written in Python, that takes advantage of this toolbox.

To access Python from the ISY lab computers, open a terminal and invoke these two commands:

```
bash
export PATH=/courses/TSBB09/anaconda3/bin:$PATH
```
The correct version of Python will now be accessible.

2.2 Detecting calibration patterns

In the `python` directory you will find `find_points.py`. This is a small program that can detect, locate and store coordinates of the inner corners of a

chessboard (not necessarily 8-by-8 squares) in a set of images. Follow the steps **a)-c)**:

a) Program parameters
Find the program parameters in the "main" section. Here you need to specify the input directory in `images_path`, where the calibration patterns are located. Make sure the `file_pattern` parameter matches the image file extensions. The output directory in `data_path` should point to a place in your own home directory.

QUESTION 2: Also check the chessboard parameters to match the shape of the chessboard in your images. How many chessboard inner corners, (columns, rows) is specified in `find_points.py`? What is the chessboard tile size?

---

b) Execute
Execute `python find_points.py`. Note that the first time you call python, it will take rather long time. If all went well, `data_path` contains a set of images and some data files. The images are the same as the input, but with the detected corners and their ordering painted on top, in a zig-zag pattern. Every calibration image is accompanied with one ".npz" and one ".mat" file, that contain the geometry of the calibration points and the model points. Note that the model points are the same for all calibration images.

QUESTION 3: In MATLAB, plot the model points and draw lines between them. Check that they are given in a zig-zag pattern. How many model points are there and do they agree with the number of chessboard inner corners specified in `find_points.py`?

---

c) Check
Ensure that for every image, the points are drawn from left-to-right and top-to-bottom in a zig-zag pattern and that they coincide with the actual chessboard pattern. If they don't, you need to leave the image out. In that case, remove the corresponding ".npz" and ".mat" files, and/or rerun the program with a better image.

QUESTION 4: How many calibration points are there in each output image?

2.3 Calibrating

Once corners have been detected, the command `python calibrate.py` will perform the actual calibration method. Find the `data_path` parameter in `calibrate.py` and point it to the output directory from the detection step above. When you run the program, it will compute and print the intrinsic camera matrix, the distortion coefficients, and the root mean square error (RMSE), `rpe`, which is based on absolute reprojection errors. The calibration is good if the latter is less than 1.0 pixels.

QUESTION 5: What values do you get in the A-matrix, the distortion coefficients, and RMSE for reprojection, `rpe`?

QUESTION 6: Zhang's method estimates two distortion coefficients, but `calibrate.py`, which is an extension of Zhang's method, returns more than two. What are these additional parameters?
Hint: To answer this question, you will have to study
`calib3d. Camera Calibration and 3D Reconstruction`,
in OpenCV's documentation. We suggest looking at version 2.4 of the library at:
`https://docs.opencv.org/2.4/`
(Other version, like 3.4, may be interesting beyond this computer exercise.)
A direct link is:
`https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`

2.4 Analyzing OpenCV's calibration

For the remainder of this assignment, you should switch to working in MATLAB, or if you prefer, keep working in Python. In both cases, you have a set of helper functions that you are free to use. For simplicity, we are defaulting to MATLAB in the remaining sections, but the Python functions are mostly the same.

Move to `CameraCalibration2/matlab` on your home directory and execute `addpath functions` so that the functions directory is added to your MATLAB path. Execute `cal_help_all` to get the full list of functions.

The task is now to load and plot the reprojection errors for the OpenCV calibration. You will need to call the following functions:
`load_calibration_input`,
`load_calibration`
`plot_reprojection_errors`
You will want to load the files in the output directory created by `find_points.py` (`data_path`) and the file `calibration_cv.mat` generated by `calibrate.py`.

The reprojection error plot shows the distances between all observed chessboard corner points and their corresponding reprojected model points, that have been used in the calibration.

Longer lines segments indicate larger errors. Note however, that MATLAB's `quiver` function which is responsible for the plotting, exaggerates the line lengths to enhance the legibility. You can use MATLAB's data cursor to get numbers out.

QUESTION 7: Display the reprojection error plot. Which calibration point has the worst reprojection error? Why is that, do you think?

QUESTION 8: The function `reprojection_errors` computes reprojection errors of all points in the calibration. Call it and plot a histogram (`histogram(..., 100, 'BinLimits', [0,5])`) of the output. At which error interval is the peak located? How is this value related to the RMSE for reprojection, `rpe`? Also, calculate the mean of the reprojection_errors.

QUESTION 9: Look up `undistort_image` and use it on one of the calibration images. (We suggest '14.jpg'). Display both the original and the resulting image. What do you see? Use a digital or physical ruler.

QUESTION 10: Modify the last parameter (B) in `undistort_image` so that it produces a "zoomed out" image, by scaling some elements (which ones?). Display the results. What do you see?

---

3 Your implementation of Zhang's calibration method

Your task is to start from `calibrate.m` and `calibrate_zhang.m` and build Zhang's calibration algorithm with the functions in `matlab/functions`.

QUESTION 11: First look at the MATLAB function `find_homography.m`. Is it the homogeneous or inhomogeneous solution?

When your implementation is ready, run it on the same calibration data as before.

QUESTION 12: What values do you get in the A-matrix and the radial distortion coefficients?

QUESTION 13: Call `reprojection_errors` and calculate RMSE for reprojection, `rpe2`? Also, calculate the mean of the reprojection_errors.

QUESTION 14: Comment out `refine_homography`. How does A change?

QUESTION 15: Compare A before and after `refine_calibration`. Try to explain the difference.

QUESTION 16: Compare the reprojection errors for your Zhang implementation with the reprojection errors from the previous implementation in OpenCVs, by plotting them in a histogram.
Tip: To show two histograms of `rpes1` and `rpes2` in the same figure:

```
histogram(rpes1, 100, 'displaystyle', 'stairs', 'BinLimits', [0,5]);
hold on;
histogram(rpes2, 100, 'displaystyle', 'stairs', 'BinLimits', [0,5]);
hold off;
legend('OpenCV', 'Zhang')
```

Comment on the result!

QUESTION 17: Similarly as before with OpenCV, apply `undistort_image` to '14.jpg'. Produce one normal and one "zoomed out" image. Compare with the results you got previously with OpenCV!

---

4 Validation of the OpenCV calibration

The best calibration result we have got so far is OpenCV's calibration, where we obtained an A-matrix and the radial distortion parameters d (k1, k2 and k3). This gave the RMSE for reprojection, `rpe` = 0.47 pixels. Now we will validate this calibration (A, d) with new test images taken with the same camera, but on a different calibration pattern. (It would have worked fine with the same calibration pattern as well.) These images are here: `/courses/TSBB09/CameraCalibration2/data/validation`

QUESTION 18: How many chessboard inner corners are there in the new calibration pattern?

Copy `find_points.py` to `find_validation_points.py`. Modify it so that correct paths are set up for input and output data. The size of the chessboard tiles does not need to be changed. (A question about this will come later.)

QUESTION 19: However, there is something else that needs to be changed. What?

Execute `python find_validation_points.py`. Check that the images have received the colorful zigzag pattern.

Investigate the function `validate_finished.py`. Modify it so that correct paths are set up. Remember that A and d should be taken from OpenCV's calibration.

QUESTION 20: Consult the documentation for cv2.solvePnP. What does the function do? What is the output?

Finally, the reprojection errors are calculated and displayed with an image and a histogram. The mean value and RMSE (rpe) are also calculated.

QUESTION 21: Write down RMSE (rpe) and mean and look at the histogram. How do you see that they seem reasonable?

QUESTION 22: Compare the histogram, RMSE (rpe) and mean with the previous results in question 8. Why are the values worse now?

QUESTION 23: The size of the chessboard tiles was kept as (`cb_tile_size = 0.03475`), in `find_validation_points.py` despite this was not true. Why did the intrinsic calibration work anyway?

Extra

Calibrate one of the cameras on your mobile phone using either OpenCV or Zhang's method. Unwarp an image with straight edges near the camera. Can you tell whether the distortion was reduced?