Exercise 4: Camera calibration

02504 Computer vision

Morten R. Hannemose, mohan@dtu.dk, DTU Compute

February 25, 2022

These exercises will take you through:

Direct linear transform (DLT), linear algorithm for camera calibration and

checkerboard calibration, and bundle adjustment from Zhang (2000).

You should be able to perform camera calibration using both methods.

Mathematical exercises: Direct linear transform (DLT)

In this section consider the 3D points

$$\boldsymbol{Q}_{ijk} = \begin{bmatrix} i \\ j \\ k \end{bmatrix}, \tag{1}$$

where $i = 0, 1$, $j = 0, 1$, and $k = 0, 1$. Consider also a camera with $f = 1000$ and a resolution of 1080×1920. Furthermore, the camera is transformed such that

$$\boldsymbol{R} = \begin{bmatrix} \sqrt{1/2} & -\sqrt{1/2} & 0 \\ \sqrt{1/2} & \sqrt{1/2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \boldsymbol{t} = \begin{bmatrix} 0 & 0 & 10 \end{bmatrix}. \tag{2}$$

Exercise 4.1

Find $\boldsymbol{\mathcal{P}}$ and the projections $\boldsymbol{q}$.

Exercise 4.2

Using only $\boldsymbol{Q}$ and $\boldsymbol{q}$ estimate the projection matrix $\boldsymbol{\mathcal{P}}$ with DLT. Do not normalize your points to begin with.

Use the estimated projection matrix to reproject the points $\boldsymbol{Q}$. What is the reprojection error $\sqrt{\sum ||\boldsymbol{q}_{\text{est}} - \boldsymbol{q}||^2}$? Does normalizing your points change the results?

Programming exercises: Checkerboard calibration

Here we will prepare calibration with checkerboards. We do not yet have the ability to detect checkerboards, so for now we will define the points ourselves.

Exercise 4.3

Define a function `checkerboard_points(n, m)` that returns the 3D points

$$\boldsymbol{Q}_{ij} = \begin{bmatrix} i - \frac{n-1}{2} \\ j - \frac{m-1}{2} \\ 0 \end{bmatrix}, \tag{7}$$

where $i = 0, \dots, n-1$ and $j = 0, \dots, m-1$. The points should be returned as a $3 \times n \cdot m$ matrix and their order does not matter. These points lie in the $z = 0$ plane by definition.

Exercise 4.4

Let $\boldsymbol{Q}_\Omega$ define a set of corners on a checkerboard. Then define three sets of checkerboard points $\boldsymbol{Q}_a$, $\boldsymbol{Q}_b$, and $\boldsymbol{Q}_c$, where

$$\begin{aligned}
\boldsymbol{Q}_a &= \boldsymbol{\mathcal{R}}\left(\tfrac{\pi}{10}, 0, 0\right)\boldsymbol{Q}_\Omega, && (8)\\
\boldsymbol{Q}_b &= \boldsymbol{\mathcal{R}}(0, 0, 0)\boldsymbol{Q}_\Omega, && (9)\\
\boldsymbol{Q}_c &= \boldsymbol{\mathcal{R}}\left(-\tfrac{\pi}{10}, 0, 0\right)\boldsymbol{Q}_\Omega, && (10)\\
& && (11)
\end{aligned}$$

where

$$\boldsymbol{\mathcal{R}}(\theta_x, \theta_y, \theta_z) = \begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{bmatrix}. \tag{12}$$

Recall that you can compute $\boldsymbol{\mathcal{R}}$ with `scipy` as follows:

```
from scipy.spatial.transform import Rotation
R = Rotation.from_euler('xyz', [θx, θy, θz]).as_matrix()
```

The points should look like Figure 1.

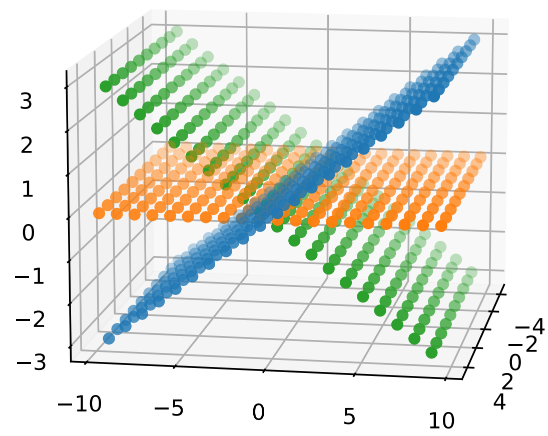

Figure 1: A 3D plot of $\boldsymbol{Q}_a$, $\boldsymbol{Q}_b$, and $\boldsymbol{Q}_c$. In this case $n = 10$, $m = 20$.

Using the projection matrix from Exercise 4.1, project all the checkerboard points to the image plane, obtaining: $\boldsymbol{q}_a$, $\boldsymbol{q}_b$, and $\boldsymbol{q}_c$.

We will now go through the method outlined in Zhang's method[1] step by step.

Exercise 4.5

Define a function `estimateHomographies(Q_omega, qs)` which takes the following input arguments:

- `Q_omega`: an array original un-transformed checkerboard points in 3D, for example $\boldsymbol{Q}_\Omega$.
- `qs`: a list of arrays, each element in the list containing $\boldsymbol{Q}_\Omega$ projected to the image plane from different views, for example `qs` could be $[\boldsymbol{q}_a, \boldsymbol{q}_b, \boldsymbol{q}_c]$.

The function should return the homographies that map from `Q_omega` to each of the entries in `qs`. The homographies should work as follows:

$$\boldsymbol{q} = H\tilde{\boldsymbol{Q}}_\Omega, \tag{13}$$

where $\tilde{\boldsymbol{Q}}_\Omega$ is $\boldsymbol{Q}_\Omega$ without the z-coordinate, in homogeneous coordinates. Remember that we need multiple orientations of checkerboards e.g. rotated and translated.

Use your function `hest` from week 2 to estimate the individual homographies. Your should return a list of homographies; one homography for each checkerboard orientation.

Test your function using $\boldsymbol{Q}_\Omega$, $\boldsymbol{q}_a$, $\boldsymbol{q}_b$, and $\boldsymbol{q}_c$. Check that the estimated homographies are correct with Equation 13.

Exercise 4.6

[1] Zhang, Zhengyou. "A flexible new technique for camera calibration." IEEE Transactions on pattern analysis and machine intelligence 22.11 (2000): 1330-1334.

Now, define a function `estimate_b(Hs)` that takes a list of homographies `Hs` and returns the vector $\boldsymbol{b}$ from Equation 6 in the paper. Use Equations 8 and 9 in the paper to form the matrix $\boldsymbol{V}$. This is the coefficient matrix used to estimate $\boldsymbol{b}$ using SVD.

Test your function with the homographies from previous exercise. See if you get the same result as by constructing $\boldsymbol{B}_{\text{true}} = \boldsymbol{K}^{-\text{T}}\boldsymbol{K}^{-1}$, and converting this into $\boldsymbol{b}_{\text{true}}$.

Is $\boldsymbol{b}$ a scaled version of $\boldsymbol{b}_{\text{true}}$?

Suggestions for debugging:

- Check that $\boldsymbol{v}_{11} \cdot \boldsymbol{b}_{\text{true}} = \boldsymbol{h}_1^{\text{T}} \boldsymbol{B}_{\text{true}} \boldsymbol{h}_1$

Exercise 4.7

Next, define a function `estimateIntrisics(Hs)` that takes a list of homographies `Hs` and returns a camera matrix $\boldsymbol{K}$. Use your `estimate_b` from the previous exercise. From $\boldsymbol{b}$, estimate the camera matrix $\boldsymbol{K}$ (they use $\boldsymbol{A}$ in the paper). Find the solution in Appendix B from the paper.

Test your function with the homographies from Exercise 4.5. Do you get the original camera matrix?

Exercise 4.8

Now, define a function `Rs, ts = estimateExtrinsics(K, Hs)` that takes the camera matrix $\boldsymbol{K}$ and the homographies `Hs` and returns the rotations `Rs` and translations `ts` of each checkerboard. Use the formulas given in the paper but you do not need to bother with Appendix C — we can live with the error.

What kind of rotations do you get, and are they valid?

Join the functions to make a larger function `K, Rs, ts = calibratecamera(qs, Q)` that finds the camera intrinsics and extrinsics from the checkerboard correspondences `q` and `Q`.

Solutions

Answer of exercise 4.1

The camera matrix is

$$\begin{bmatrix} 1000 & 0 & 960 \\ 0 & 1000 & 540 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3}$$

Note that it's not obvious how to have x and y in the camera matrix, as you can also flip 960 and 540 around. Having them the other way around is fine, as long as you are consistent, but you will of course get different projections.

The projection matrix is

$$\boldsymbol{\mathcal{P}} = \begin{bmatrix} 707.11 & -707.11 & 960 & 9600 \\ 707.11 & 707.11 & 540 & 5400. \\ 0 & 0 & 1 & 10 \end{bmatrix}, \tag{4}$$

and the projections are

$$\boldsymbol{q}_{000} = \begin{bmatrix} 960 \\ 540 \end{bmatrix}, \quad \boldsymbol{q}_{001} = \begin{bmatrix} 960 \\ 540 \end{bmatrix}, \quad \boldsymbol{q}_{010} = \begin{bmatrix} 889.29 \\ 610.71 \end{bmatrix}, \quad \boldsymbol{q}_{011} = \begin{bmatrix} 895.72 \\ 604.28 \end{bmatrix}, \tag{5}$$

$$\boldsymbol{q}_{100} = \begin{bmatrix} 1030.71 \\ 610.71 \end{bmatrix}, \quad \boldsymbol{q}_{101} = \begin{bmatrix} 1024.28 \\ 604.28 \end{bmatrix}, \quad \boldsymbol{q}_{110} = \begin{bmatrix} 960 \\ 681.42 \end{bmatrix}, \quad \text{and } \boldsymbol{q}_{111} = \begin{bmatrix} 960. \\ 668.56 \end{bmatrix}. \tag{6}$$

Answer of exercise 4.2

The projection matrix and the reprojections are identical to the original within machine precision. The reprojection error on my system is approximately 10^{-11} without normalization and 10^{-14} with normalization.