Lab 4: Camera calibration

MCHA4400

Semester 2 2025

Introduction

In this lab, you will implement a planar camera model and perform camera calibration on two different cameras using the OpenCV API. You will process a set of image files (for the first camera), selected frames from a video file (for the second camera), calibrate each camera, and then test the calibration with a basic augmented reality demo.

The lab should be completed within 4 hours. The assessment will be done in the lab at the end of your enrolled lab session(s). Once you complete the tasks, call the lab demonstrator to start your assessment.

The lab is worth 5% of your course grade and is graded from 0–5 marks.

GenAI Tip

You are strongly encouraged to explore the use of Large Language Models (LLMs), such as GPT, Gemini or Claude, to assist in completing this activity. If you do not already have access to an LLM, bots are available to use via the UoN Mechatronics Slack team, which you can find a link to from Canvas. If you are are unsure how to make the best use of these tools, or are not getting good results, please ask your lab demonstrator for advice.

1 Camera calibration from images (2 marks)

The calibration grid shown in Figure 1 was used to obtain all calibration data. Each grid cell has a spacing of 22 mm.

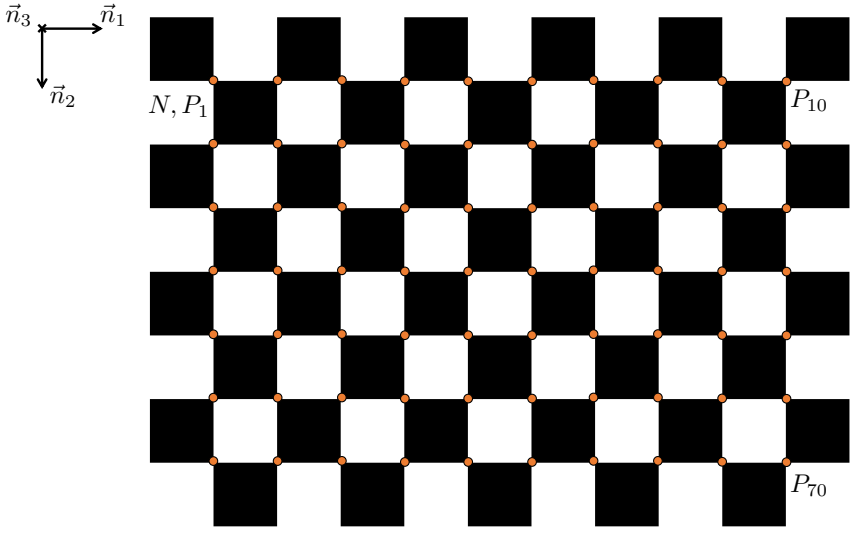

Figure 1: Camera calibration grid

Take the time to review the `struct` declarations within `src/Camera.h` and their implementations in `src/Camera.cpp`. In particular, note that

- `Pose` contains a rotation matrix and translation vector to represent 6-DOF poses in $\mathsf{SE}(3)$, along with functions for multiplication and inversion.
- `Camera` contains the intrinsic camera parameters and some functions to evaluate the camera model (e.g., `worldToPixel`).
- `Chessboard` contains the geometry and dimensions of the calibration chessboard.
- `ChessboardImage` contains an image of the chessboard and the extrinsic camera parameters (pose) for that image.
- `ChessboardData` contains one instance of `Chessboard` and a collection of `ChessboardImage` objects.

Complete the implementation in `src/Camera.cpp` in the tasks below.

Tasks

a) In the `ChessboardImage` constructor, detect the corners of the chessboard image using `cv::findChessboardCorners`. To improve the accuracy using subpixel refinement, you may also use `cv::cornerSubPix`.

b) In the `Camera::calibrate` function, use `cv::calibrateCamera` to calibrate the camera model and set the appropriate members containing the intrinsic and extrinsic camera parameters.

Tip

The OpenCV function `cv::calibrateCamera` returns the camera pose, for each frame, in camera coordinates. The k^{th} element of argument `tvecs` corresponds to the translational vector $\mathbf{r}^c_{N/C}$ for the k^{th} calibration image and the k^{th} element of argument `rvecs` corresponds to the exponential parameterisation $\boldsymbol{\Theta}^c_n$ of the rotation matrix $\mathbf{R}^c_n = \exp \mathbf{S}(\boldsymbol{\Theta}^c_n)$ for the k^{th} calibration image, which can be found by calling `cv::Rodrigues` on the `rvecs[k]` variable. Note that unlike OpenCV, we will represent the camera position and orientation in world-fixed coordinates by $\mathbf{r}^n_{C/N}$ and $\mathbf{R}^n_c$, respectively.

c) In the `Camera::worldToVector` function, return the unit vector $\mathbf{u}^c_{P/C} = \frac{\mathbf{r}^c_{P/C}}{\|\mathbf{r}^c_{P/C}\|}$ that corresponds to the provided world position $\mathbf{r}^n_{P/N}$ and camera pose.

d) In the `Camera::vectorToPixel` function, use `cv::projectPoints` to map from a vector in camera coordinates, $\mathbf{r}^c_{P/C}$, to a pixel location in image coordinates, $\mathbf{r}^i_{Q/O}$.

e) In the `Camera::pixelToVector` function, return the unit vector $\mathbf{u}^c_{P/C}$ that corresponds to the provided pixel coordinates $\mathbf{r}^i_{Q/O}$. This can be done with the help of the `cv::undistortPoints` function.

f) In the `Camera::calcFieldOfView` function, calculate the horizontal, vertical and diagonal field of view of the camera, using the intrinsic parameters of the camera. This can be done by finding the angles between direction vectors that correspond to some of the edge and corner pixels in the image with the help of `Camera::pixelToVector`.

g) In the `Camera::isVectorWithinFOV` function, determine if the provided vector $\mathbf{r}^{c}_{P/C}$ lies within the camera field of view.

h) Configure and build the application and ensure all the provided unit tests pass:

Terminal

```
nerd@basement:~/MCHA4400/lab4$ cmake -G Ninja -B build && cd build
nerd@basement:~/MCHA4400/lab4/build$ ninja
```

i) Build and run the application with verbosity level 1 to display the calibration images and review the calibration results printed to the terminal:

Terminal

```
nerd@basement:~/MCHA4400/lab4/build$ ninja && ./lab4 -c -v=1 ../data/images/config.xml
```

Images similar to Figure 2 should be displayed.

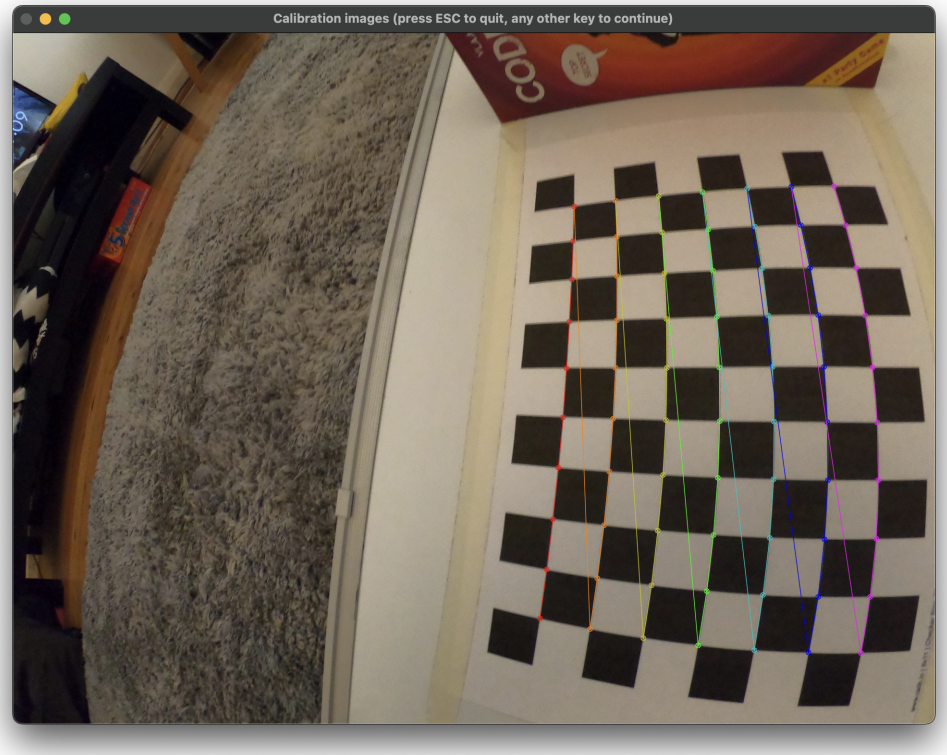

Figure 2: Example of calibration image with drawn chessboard corners in sequence.

VS Code tip

When debugging the application in Visual Studio Code, you can pass command line parameters by modifying the `launch.json` file in the `.vscode` folder. Add or modify the

`args` array in the configuration for your debug target. For example:

```
"args": ["-c", "-v=1", "../data/images/config.xml"],
```

This will pass the arguments `-c -v=1 ../data/images/config.xml` to the application when launched from the debugger.

2 Camera calibration from video (1 mark)

In this task, we will perform camera calibration by processing frames from a video file instead of using individual image files. This offers some advantages for visual navigation applications. It enables calibration under conditions that better match operational use, including camera settings (e.g., resolution, exposure, etc.) and any dynamic effects (e.g., image stabilisation, auto-focus behaviour). It can also be more convenient and time-efficient to record a short video rather than capturing numerous individual images.

We'll extend the existing code to handle video input, select appropriate frames for calibration, and perform the calibration process using these video frames.

Complete the implementation in `src/Camera.cpp` in the tasks below.

Tasks

a) Modify the `ChessboardData` constructor to handle video files in the configuration XML. To do this, select a subset of appropriate frames from the video file to use to detect the chessboard corners.

Info

It is usually not wise to use all frames from a video for calibration, since `cv::calibrateCamera` doesn't cope well with a large number of calibration images.

b) Build and run the application with verbosity level 1 to display the calibration images and review the calibration results printed to the terminal:

Terminal

```
nerd@basement:~/MCHA4400/lab4/build$ ninja && ./lab4 -c -v=1 ../data/video/config.xml
```

Based on the obtained intrinsic camera parameters, how does this camera compare to the one in Problem 1?

3 Augmented reality (2 marks)

In the previous tasks, you found the camera matrix & distortion coefficients (intrinsic parameters) and camera poses (extrinsic parameters). In this task, you will add some 3D geometry to the scene to validate the calibration of your camera. Complete the implementation in `src/Camera.cpp` in the tasks below.

Tasks

a) In the `ChessboardImages::drawBox` function, draw a rectangular prism with the base spanning the interior corners of the chessboard and height equal to 0.23 m (same height as the CodeNames box). To do this, generate a set of vertices for each line to draw in world coordinates and check that they lie within the camera field of view using `Camera::isWorldWithinFOV`. If so, find the image coordinates using `Camera::worldToPixel` and then draw them on the image using `cv::line`.

> **Hint**
>
> Due to the distortion of the lens, straight lines in the world now become curved in the image, therefore you may need to draw several line segments per box edge to obtain a geometrically consistent render of the box.

> **Tip**
>
> The line thickness can be set to a fixed size in pixels in the image (easy to implement), or a fixed size in space to give the lines a more realistic appearance (harder to implement).
>
> To implement the latter option, let 0 and 1 denote the endpoints of a line segment and let $\mathbf{r}^n_{1/0} = \mathbf{r}^n_{1/N} - \mathbf{r}^n_{0/N}$ be the line segment vector expressed in world coordinates. Let T be a point offset from 0 by a small distance w in a direction perpendicular to the line segment and the camera's optical axis, i.e., $\mathbf{r}^n_{T/N} = \mathbf{r}^n_{0/N} + w\,\mathbf{u}^n_{T/0}$, where w is a small constant (e.g., 3 mm) and $\mathbf{u}^n_{T/0} = \frac{\mathbf{r}^n_{1/0} \times \mathbf{c}^n_3}{\|\mathbf{r}^n_{1/0} \times \mathbf{c}^n_3\|}$ is a unit vector perpendicular to both the line segment vector and the camera's optical axis expressed in world coordinates.
>
> The thickness of the line segment in pixels can then be calculated as $\|\mathbf{r}^i_{T/O} - \mathbf{r}^i_{0/O}\|$, where $\mathbf{r}^i_{T/O} = \texttt{w2p}(\mathbf{r}^n_{T/N})$ and $\mathbf{r}^i_{0/O} = \texttt{w2p}(\mathbf{r}^n_{0/N})$ are the image coordinates corresponding to points T and 0, respectively, and $\texttt{w2p}(\cdot)$ is the world-to-pixel function.

b) Build and run the application with verbosity level 2 (box display) to validate each calibration set:

Terminal

```
nerd@basement:~/MCHA4400/lab4/build$ ninja && ./lab4 -c -v=2 ../data/images/config.xml
nerd@basement:~/MCHA4400/lab4/build$ ninja && ./lab4 -c -v=2 ../data/video/config.xml
```

Images similar to Figure 3 should be displayed.

> **Tip**
>
> To troubleshoot issues with drawing the box, try drawing only the base rectangle in the plane of the chessboard. Its vertices are guaranteed to be in the field of view for all the images used for camera calibration.

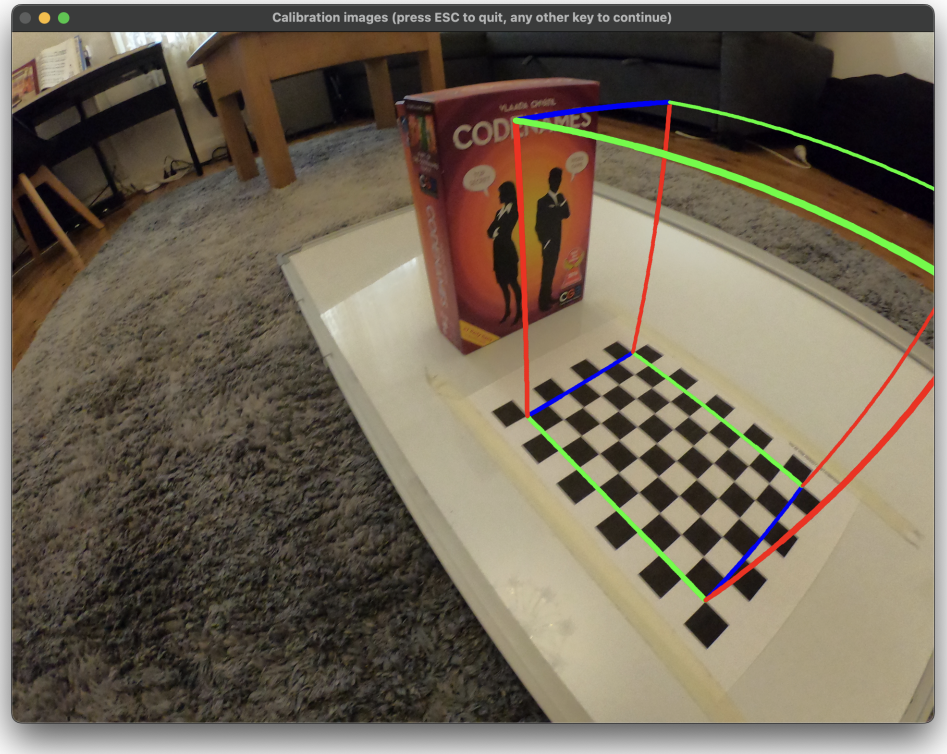

Figure 3: Box validation of camera calibration.

c) Export all of the calibration images with the box visualisation to the `out` directory by using the following commands:

Terminal

```
nerd@basement:~/MCHA4400/lab4/build$ ninja && ./lab4 -c -e -v=2 ../data/images/config.xml
nerd@basement:~/MCHA4400/lab4/build$ ninja && ./lab4 -c -e -v=2 ../data/video/config.xml
```