A1: Calibration

1. Overview

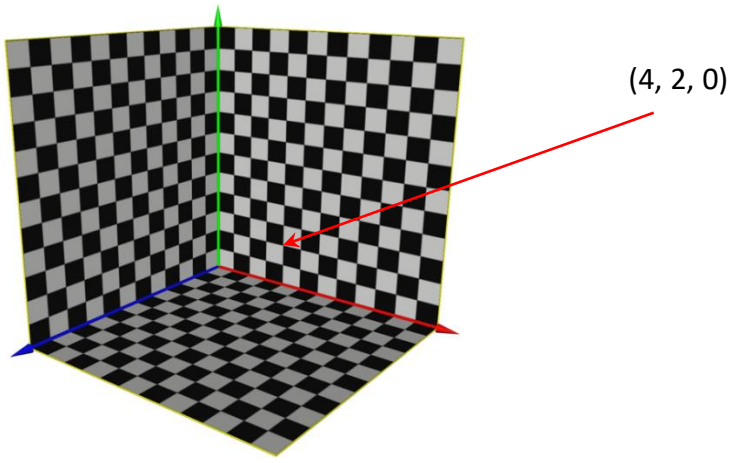
In this assignment, you will gain hands-on experience in camera calibration. Specifically, you will practice:

1) Basic linear algebra operations (e.g., matrix-vector product and SVD);
2) Collecting a set of 3D-2D corresponding points using a (virtual) apparatus;
3) Extracting intrinsic camera parameters;
4) Extracting extrinsic camera parameters;
5) Using open-source libraries to visualize and validate your calibration results.

2. The tasks

Task #1: **Collect at least 6 pairs of 3D-2D corresponding points (10%)**

Compile and run the viewer. You will see a virtual calibration rig as follows



In this rig, the **red**, **green**, and **blue** arrows denote the ***X***, ***Y***, and ***Z*** axes, respectively. With the grid texture, you can easily figure out the 3D coordinates of any points on the grid.

You can use your mouse to manipulate the scene (i.e., move and reorient your camera).

Take a picture of the scene by pressing the "s" key. Then you will see a dialog asking for a file name. You can provide any name, e.g., "image_01.png", and save it to your disk.

With the image, pick ***n (n>=6)*** points on the virtual scene, and figure out their corresponding image points. To make access to the data easier, please store these correspondences in a "***.txt***" file in which each line has 5 coordinates separated by space, i.e., the first 3 coordinates represent a 3D point and the subsequent 2 coordinates represent the corresponding 2D image point. See the two examples in "/resources/data".

Data: use the provided data for testing your camera calibration algorithm. In your submission, you must include the test results using your collected 3D-2D correspondences. We assume every team collects different data for the experiment.

Image coordinates: The image below shows the image coordinate system. Image coordinates are denoted in pixels, with the origin point (0, 0) corresponding to the top-

left corner of the image. The X-axis starts at the left edge of an image and goes towards the right edge. The Y-axis starts at the top of the image towards the image's bottom. All image pixels have nonnegative coordinates.

Task #2: **Implement the camera calibration algorithm (50%)**

In the file "/A1_Calibration_Code/Calibration/calibration_method.cpp", the function "calibration" is defined but not implemented. Comments and guidance are left in the function. Example code is also provided within the function for you to get familiar with the necessary data structures and algorithms.

Camera calibration can be achieved by tackling the following subtasks:

- Construct the P matrix. (10%)
- Solve for M (the whole projection matrix, i.e., $M = K * [R, \mathbf{t}]$). (10%)
- Extract intrinsic parameters from M. (10%)
- Extract extrinsic parameters from M. (10%).
- Your code must explicitly handle the sign of ρ. (10%)

Task #3: **Evaluation (10%)**

- Evaluate your calibration results both qualitatively and quantitatively. (5%)
- Invite student(s) of another group to review your code. Incorporate the received feedback in your implementation and reflect on the changes in the report. (5%)

Note: You're encouraged to discuss and share experiences with other groups, but coping codes from others will be penalized.

3. Submission. Your submission should include:

(1) Report (max 3 pages excluding figures and tables) **(30%)**

You're expected to deliver a serious scientific report. Make every sentence, equation, and notation precise and clear to avoid misinterpretation. Meanwhile, the report should

be as concise as possible, but it should provide key information to reimplement the method to reproduce your results, and it should include:

- Description of the methodology. Please use mathematical language (e.g., equations) as much as possible for the description. (5%)
- Calibration results of your algorithm on both provided data and your data. (5%)
- Snapshots of your registration results (shown next to the input images). Please use the snapshot function provided by the viewer to take snapshots. **Note**: the resolution of the snapshots may be affected by the DPI scaling factor of your display (e.g., a MacBook with a retina display will generate images with doubled resolution, which may result in different intrinsic parameters). Therefore, it is recommended to take the input and resulting images on the same machine or machines with the same DPI scaling factor. (3%)
- Explanation of how you verified the intermediate results (if necessary). (2%)
- Discussion on how you determined the sign of ρ. (5%)
- Discussion on the accuracy of your results, e.g., where do the errors come from? Can you visualize and quantify the errors? How to improve accuracy? (5%)
- Reflection on how the feedback received from others helped to improve your implementation. (5%)
- A short description of "who did what".

To promote formal scientific writing, the following rules apply for assessment:

- Any misunderstanding or misconception of main concepts: 10% deduction.
- Multiple unclear or ambiguous descriptions: 10% deduction.
- Multiple typos, grammar issues, format issues (e.g., a figure/table without a caption or multiple figures/tables with the same caption, unindexed or unreferenced figures/tables), consistency issues (e.g., upper case and lower case used interchangeably, normal font and italic font used interchangeably), misuse of symbols in notations: 10% deduction.
- The report is over length: 10% deduction.

(2) Data

- Your input 3D-2D point pairs (in the *txt* format).
- The input image (must be taken using the built-in snapshot function of the viewer).

(3) Source code

Please submit **only** the following file:

- *A1_Calibration_Code/Calibration/calibration_method.cpp*

This file contains your implementation of the '*calibration(…)*' function.

Your source code should compile and reproduce your results without modification.

Please compress the above files into an archive file named in the following format:

GEO1016_Assignment_1_Group_XX.zip

where 'XX' is a 2-digit number of your group ID, which can be found here:

https://docs.google.com/document/d/1qNMGZlcXL10sY5ntmgh3l3ZswYm1TKtbQYA-14SiVyg/edit?usp=sharing

Appendix 1: About the code framework

The code framework is based on *Easy3D*[1]. The viewer provides visualization of 3D scenes and cameras. It also serves as a means to intuitively validate the camera calibration results.

The usage of the viewer:

Left button:	*rotate the camera*
Right button:	*move the camera*
Key 's':	*snapshot (i.e., take a picture)*
Key 't':	*show/hide the virtual camera*
Key 'space':	*run the camera calibration algorithm*

Students are supposed to have some experience with C++ and can already compile and run the code of this assignment. If not, please have a look at:

https://3d.bk.tudelft.nl/courses/geo1016/resources/build_C++.html

Appendix 2: Image comparison tool

Some tools can visualize the differences between two images, such as ImageDiff:

https://github.com/LiangliangNan/ImageDiff

Pre-built executables are available for Mac, Windows, and Linux users.

[1] https://github.com/LiangliangNan/Easy3D. The code framework uses a stripped earlier version of Easy3D, which is not compatible with its latest version.