Reproducibility review of: Indoor localisation and location tracking in indoor facilities based on LiDAR point clouds and images of the ceilings

Nina Wiedemann

May 29, 2023

This report is part of the reproducibility review at the AGILE conference. For more information see https://reproducible-agile.github.io/. This document is published on OSF at https://doi.org/10.17605/osf.io/8t3bh. To cite the report use

Wiedemann, N. (2023). Reproducibility review of: Indoor localisation and location tracking in indoor facilities based on LiDAR point clouds and images of the ceilings. https://doi.org/10.17605/osf.io/8t3bh

1. Reviewed paper

Dardavesis, I.,Verbree, E., and Rafiee, A.: Indoor localisation and location tracking in indoor facilities based on LiDAR point clouds and images of the ceilings, AGILE GIScience Ser., 4, 4, https://doi.org/10.5194/agile-giss-4-4-2023, 2023

2. Summary

The paper is accompanied by a GitHub repository with front end and back end code of their indoor localization app. The code was not executable first, but installation instructions were added and file paths were fixed upon exchange with the authors. The results reported in the paper are partially reproducible with the provided code. The code mainly yields examples that are similar but not the same as in the paper, due to randomness in the algorithms. One script allows to reproduce a figure exactly, while other quantitative results (tables in the paper) can not be created with the code. However, given the instructions in the README and the example data, the repository can be very useful for researchers who want to apply the pipeline on their own data.

3. Reproducibility reviewer notes

3.1. Repository structure and installation

After first correspondence, the authors added a README file with a detailed description of the files included in the repository, which was useful to navigate the files. With my pull request, installation instructions for installing packages in a Python virtual environment via a requirements.txt file were added, as well as a license. The repository is structured and the code is partially documented. The exemplary data is provided within the repository but requires a while to download (258MB).

This review focuses on the code of the back end, since it generates the results shown in the paper. The back end code includes two scripts for generating examples for the registration and image-matching algorithms, and a script to generate scatter plots shown in two figures in the paper. All scripts must be executed from the folder "back_end", e.g., `cd back_end; python code/image_matching.py`.

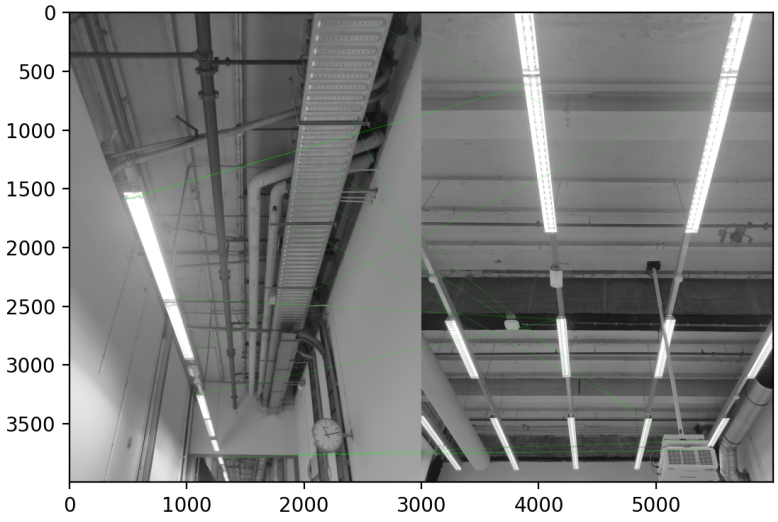

Figure 1: Reproducing image matching via SIFT and SURF. Note the green lines on the image which are hard to see.

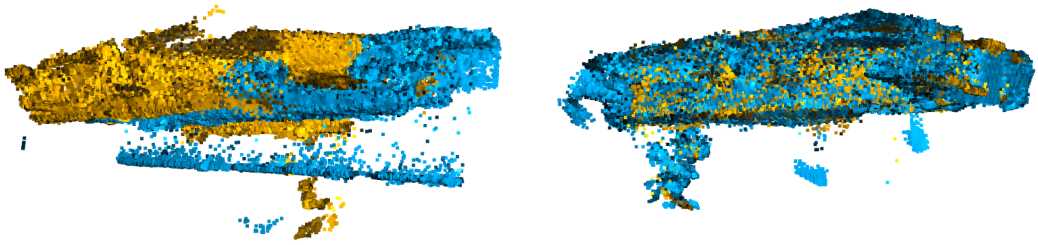

Figure 2: Reproducing registration results (Figure 11).

3.2. Reproducing exemplary figures

Figure 11, 16 and 17 in the paper show examples for registration and image matching. When running the script image_matching.py, a figure similar to Figure 16 and 17 of the paper is shown, as shown in Figure 1. When executing point_cloud_registration.py, the registration results for some data is visualized as in Figure 11 in the paper, but apparently with different data. The output in our tests for reproducibility is shown in Figure 2. Upon request, the authors state that the global registration algorithm uses RANSAC to pick points on the cloud each time the algorithm runs. In that way, the registration results might differ each time even when using the same user point cloud.

3.3. Reproducing scatter plot in Figure 13

After the initial exchange, a script was added to generate the scatter plots of Figure 13, showing the user point clouds after matching. The script could be executed and the plots

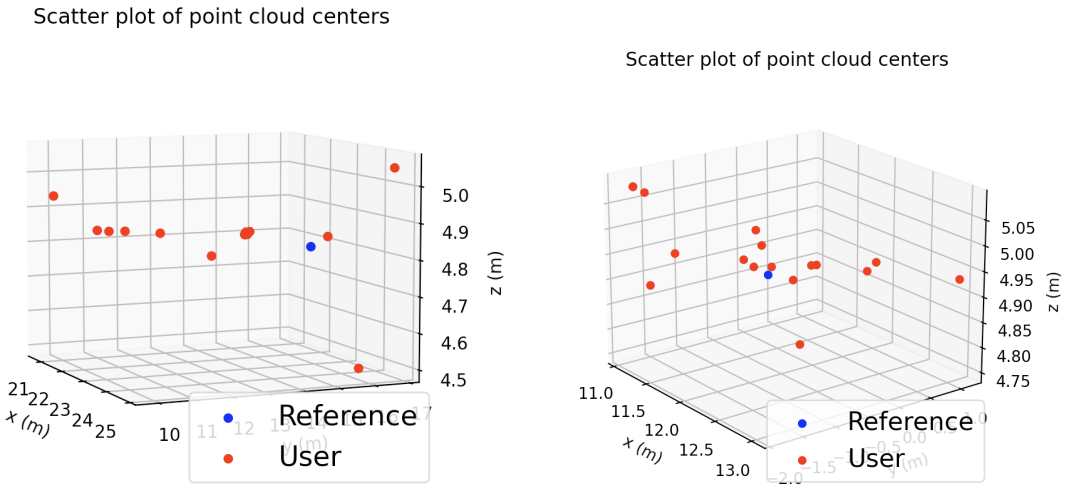

Figure 3: Reproducing registration results (Figure 11).

are the same as in the paper, after navigating the 3D plot (zooming and rotating). The plots produced in our tests are shown in Figure 3.

3.4. Runtime plots

Figure 12 and 14 show the runtimes of the algorithms and, according to the authors, they were created with Excel using the console outputs when running the algorithms. Indeed, the runtime is printed to the console when executing `python image_matching.py` or `python point_cloud_registration.py`. For example, when reproducing the latter, it states *Time elapsed: 57.1175*. However, this runtime does not match the one reported in the tables, which is due to the different hardware used. It is therefore generally not recommended to report runtimes in seconds. Furthermore, when executing the script without any changes, one figure is shown at a time and the script waits for the user to close the figure. This time counts towards the runtime as far as I can see, which is problematic for a runtime comparison that should be user-independent.

3.5. Algorithm comparison

Table 2, Table 3 and Figure 15 evaluate the tested algorithm in terms of correct matches. These results are **not reproducible** with the provided code, since they were created with separate software that is not publicly available. The authors were not able to add this code, at least not in the time frame of the reproducibility review.

4. Recommendations

- Describe the runtimes in the README file
- Relate the figures in the paper to your code in the README file
- The code to generate the missing tables and figures should be added if possible.
- Provide hardware specifications in paper and README to reproduce runtimes.