Modern C++ SoSe2025

Course Project Assignment

3D Occupancy Grid Mapping

As your final project for this course, you will be making an occupancy mapping pipeline given a sequence of posed LiDAR scans.

We will provide you with utility functions that you can use for specific parts of the entire pipeline. Apart from this, you will be using two third-party dependencies: Eigen3 and Open3D.

Eigen3 is something you will be familiar with already (linear algebra library) and Open3D is a library for 3D data processing. While you can do much more with Open3D, for this project you will use it to visualize the final map you will produce. And even then, you do not need to use Open3D in code you write, but you need it to use the visualization function we provide. Nevertheless, this will require linking Open3D in your build system and the build system is something we expect you to do.
Hint: https://www.open3d.org/docs/release/cpp_project.html

On which note, you are also expected to use *modern* cmake for your build system. If not modern cmake = fail.

In terms of data, you will be provided a `pose_file.txt` and a sequence of `.ply` files for the point clouds. Each point cloud will have a timestamp in it's name, and the corresponding pose will be provided in the `pose_file.txt`. You will be able to read both the poses and the raw point clouds using the provided function

```cpp
using Vector3dVector = std::vector<Eigen::Vector3d>;
using PoseAndCloud = std::pair<Eigen::Matrix4d, Vector3dVector>;
const dataloader::Dataset dataset = Dataset("path_to_data_dir");
const PoseAndCloud pose_and_coud = dataset[0];
// max index is given by dataset.size();
```

The `Eigen::Matrix4d` part of `PoseAndCloud` is a 6-DoF transform from the sensor frame to the map frame. The `Vector3dVector` part is the raw point cloud in the sensor frame.

You must then perform 3D occupancy grid mapping using Bresenham's line algorithm (implemented by you).

Once your map is built, you can visualize the final output as a point cloud using Open3D's visualization tool. You can use the given function

```cpp
void visualize(const Vector3dVector& cloud);
```

That's it. While you are otherwise free to implement the mapping pipeline in any way, the following have to be satisfied at least once:

- Implement a custom class
- Use an STL algorithm
- Use an STL container
- Use a lambda

Additionally, you should time your code and compute the average time taken to integrate each scan into the map, and also the total time for execution.

You can refer to the following to refresh yourself on how 2D grid mapping works: https://www.youtube.com/watch?v=fO4euQVfMso

Scan this QR code to get to a sciebo page where you can download the data and utility code:

All the best!