ARO Homework 4: ICP with fast nearest-neighbour search

Karel Zimmermann Vladimir Kubelka
Ondrej Holesovsky Radoslav Skoviera

2018

You are given sequences of 2D pointclouds and of noisy odometry measurements (absolute translations and rotations of a mobile robot in the world coordinate frame). A code skeleton with methods for loading the data and computing certain coordinate transforms is available in the Jupyter notebook file `icp_2d_student.ipynb`. The notebook also contains more detailed instructions. Your task is to build a 2D pointcloud map using the several partial pointclouds collected from different robot poses. Utilize the ICP (Iterative Closest Point) algorithm for the mapping process.

Each iteration of the ICP algorithm requires to find the best point correspondences between two pointclouds by means of the nearest neighbour search. The simplest implementation of the nearest neighbour search has linear time complexity (it needs to iterate over all the points), which would be too slow for real-world pointclouds (hundreds of thousands of points). A k-d tree is a faster algorithm for nearest neighbour search suitable for low dimensional vectors.

1. Learn more about what a *k-d tree* is and what it is good for.
2. Look up the documentation of the SciPy *k-d tree* implementation.
3. Implement the ICP algorithm (e.g. in the empty *icp* function body of the Jupyter notebook). To make things simpler at this point, use only the sequence of pointclouds, not the odometry data. Employ a k-d tree for nearest neighbour search.
4. Test your algorithm on the provided data. Compare the quality of simultaneous robot pose tracking and 2D mapping based on the coordinate transforms provided by: a) odometry, b) ICP.
5. `Bonus:` There are cases when the ICP on its own can fail to track the motion of a robot correctly. Give a real-world example of such a case. Utilize the odometry sequence to make the ICP pose tracking and mapping more robust in these scenarios.