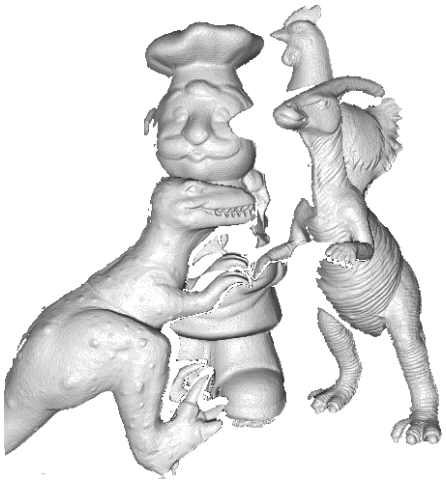
Exercise

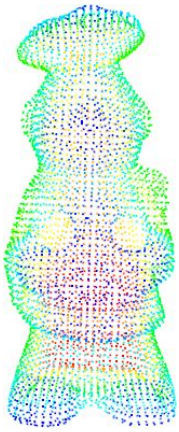
The task is to find the pose of an object within a scene. The object is a full model generated by a registration algorithm. The scene is a real scene captured with a laser scanner. The models are shown below.

Object

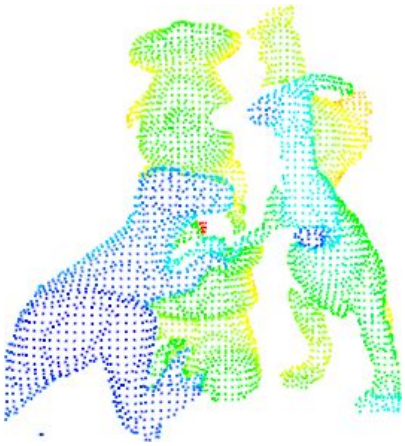


Scene

You will get both models as point clouds, downsampled to a resolution of 5 mm to limit the amount of data. Below are screenshots of the point cloud models.



Object point cloud



Scene point cloud

You are to solve both a local and a global pose estimation task. Before you start, make sure you have a working PCL installation (http://pointclouds.org/downloads/linux.html), and create a CMake project (http://pointclouds.org/documentation/tutorials/using_pcl_pcl_config.php). While programming, make use of the PCL API (http://docs.pointclouds.org/trunk).

Exercise 1: local alignment

In this exercise, the task is to perform local alignment using ICP. By some unknown process, the object has been brought into approximate alignment with the scene already. The pre-aligned object is given in the file **object-local.pcd**, and the scene in **scene.pcd**.

Write a C++ program for performing the following tasks:

- Load the two point clouds
    - HINTS: **pcl::io::loadPCDFile()**
- Show the initial state of the two models within the same view
    - HINTS: **pcl::visualization::PCLVisualizer**
- Implement ICP for bringing the models into accurate alignment. You are NOT allowed to use the **IterativeClosestPoint** class, but you are welcome to have a look inside the code for inspiration.
    - HINTS:**pcl::search::KdTree**, **pcl::registration::TransformationEstimationSVD**, **pcl::transformPointCloud()**
- Show the final state of the two models after applying your ICP algorithm

Exercise 2: global alignment

Now you are given a new object model in **object-global.pcd**, which is not close to be aligned with the scene.

Write a new C++ program for performing the following tasks:

- Load the two point clouds
- Show the initial state of the two models within the same view
- Compute surface normals using the *k* = 10 nearest neighbors around each point
    - HINTS: **pcl::NormalEstimation**
- Compute local Spin Image features with *r* = 5 cm for surface description
    - HINTS: **pcl::Histogram<153>**, **pcl::SpinImageEstimation**
- Find the nearest matching (*k* = 1) scene feature for each object feature
    - HINTS: for this task, you need to manually write a function that finds the best matching feature vector among all the candidates in the scene in terms of the squared Euclidean (L_2) distance. Have a look at the **Histogram** struct and use the **Histogram::histogram** member (a 153-dimensional float array) when accumulating distances.
- Implement RANSAC for finding an alignment of the object with a maximum of 10000 iterations, and with an inlier threshold of 5 mm
    - HINTS: use your feature matches to sample 3 random point pairs at a time (see **pcl::common::UniformGenerator**), use them for generating a candidate pose using **TransformationEstimationSVD**. Use a **KdTree** for the scene point cloud (initialized before your RANSAC loop) to count inliers within the threshold. Decide for yourself whether to use the RMSE or the inlier count as criterion for validating poses.
- Show the final state of the two models after applying your feature-based RANSAC algorithm