**hdl\_graph\_slam**

**Operating System:**

ubuntu 18.04

**Ros Version:**

ros melodic

**Software:**

hdl\_graph\_slam

**Introduction:**

***Summary***

hdl\_graph\_slam is an open source ROS package for real-time 6DOF SLAM using a 3D LIDAR. It is based on 3D Graph SLAM with NDT scan matching-based odometry estimation and loop detection. It also supports several graph constraints, such as GPS, IMU acceleration (gravity vector), IMU orientation (magnetic sensor), and floor plane (detected in a point cloud). We have tested this package with Velodyne (HDL32e, VLP16) and RoboSense (16 channels) sensors in indoor and outdoor environments.

***Nodelets***

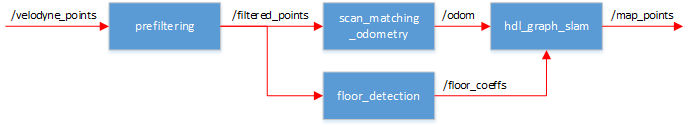
hdl\_graph\_slam consists of four nodelets.

**·**prefiltering\_nodelet

**·**scan\_matching\_odometry\_nodelet

**·**floor\_detection\_nodelet

**·**hdl\_graph\_slam\_nodelet

The input point cloud is first downsampled by prefiltering\_nodelet, and then passed to the next nodelets. While scan\_matching\_odometry\_nodelet estimates the sensor pose by iteratively applying a scan matching between consecutive frames (i.e., odometry estimation), floor\_detection\_nodelet detects floor planes by RANSAC. The estimated odometry and the detected floor planes are sent to hdl\_graph\_slam. To compensate the accumulated error of the scan matching, it performs loop detection and optimizes a pose graph which takes various constraints into account.

***Constraints (Edges)***

You can enable/disable each constraint by changing params in the launch file, and you can also change the weight (\*\_stddev) and the robust kernel (\*\_robust\_kernel) of each constraint.

**·**Odometry

**·**Loop closure

**·**GPS

·/gps/geopoint (geographic\_msgs/GeoPointStamped)

·/gps/navsat (sensor\_msgs/NavSatFix)

·/gpsimu\_driver/nmea\_sentence (nmea\_msgs/Sentence)

hdl\_graph\_slam supports several GPS message types. All the supported types contain (latitude, longitude, and altitude). hdl\_graph\_slam converts them into the UTM coordinate, and adds them into the graph as 3D position constraints. If altitude is set to NaN, the GPS data is treated as a 2D constrait. GeoPoint is the most basic one, which consists of only (lat, lon, alt). Although NavSatFix provides many information, we use only (lat, lon, alt) and ignore all other data. If you're using HDL32e, you can directly connect hdl\_graph\_slam with velodyne\_driver via /gpsimu\_driver/nmea\_sentence.

**·**IMU acceleration (gravity vector)

·/gpsimu\_driver/imu\_data (sensor\_msgs/Imu)

This constraint rotates each pose node so that the acceleration vector associated with the node becomes vertical (as the gravity vector). This is useful to compensate for accumulated tilt rotation errors of the scan matching. Since we ignore acceleration by sensor motion, you should not give a big weight for this constraint.

**·**IMU orientation (magnetic sensor)

·/gpsimu\_driver/imu\_data (sensor\_msgs/Imu)

If your IMU has a reliable magnetic orientation sensor, you can add orientation data to the graph as 3D rotation constraints. Note that, magnetic orientation sensors can be affected by external magnetic disturbances. In such cases, this constraint should be disabled.

**·**Floor plane

·/floor\_detection/floor\_coeffs (hdl\_graph\_slam/FloorCoeffs)

This constraint optimizes the graph so that the floor planes (detected by RANSAC) of the pose nodes becomes the same. This is designed to compensate the accumulated rotation error of the scan matching in large flat indoor environments.

***Parameters***

All the configurable parameters are listed in launch/hdl\_graph\_slam.launch as ros params.

***Services***

**·**/hdl\_graph\_slam/dump (hdl\_graph\_slam/DumpGraph)

·save all the internal data (point clouds, floor coeffs, odoms, and pose graph) to a directory.

**·**/hdl\_graph\_slam/save\_map (hdl\_graph\_slam/SaveMap)

·save the generated map as a PCD file.

***Requirements***

hdl\_graph\_slam requires the following libraries:

**·**OpenMP

**·**PCL 1.7

**·**g2o

**·**suitesparse

The following ROS packages are required:

**·**geodesy

**·**nmea\_msgs

**·**pcl\_ros

**·**ndt\_omp

**Examples**

***Dataset***

Kitti dataset

***Tools***

<https://github.com/ethz-asl/kitti_to_rosbag>

This toos is used to convert the kitti data into rosbag file.

***Process:***

\*Remember to source the ros installation and the hdl\_graph\_slam installation in each new terminal.

$ roscore

$ rosparam set use\_sim\_time true

$ roslaunch hdl\_graph\_slam hdl\_graph\_slam.launch

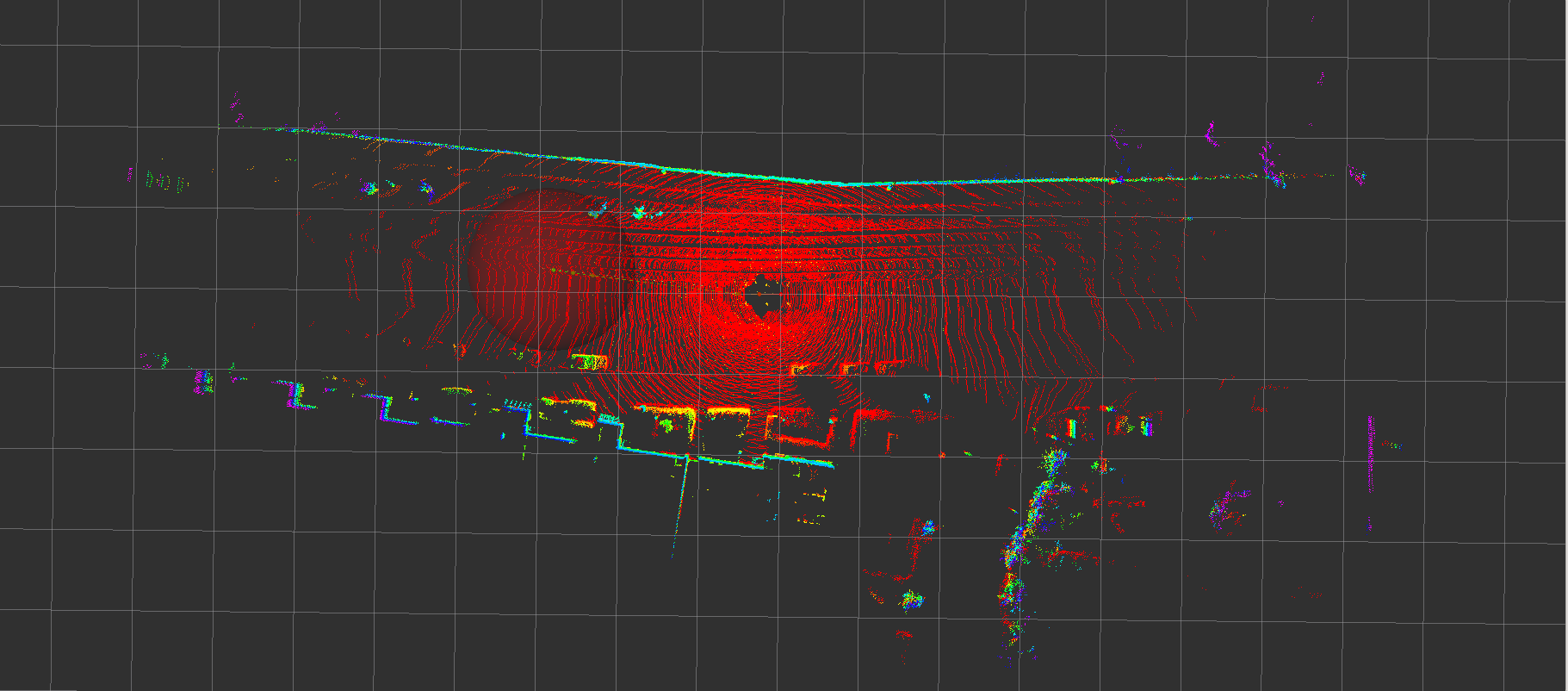
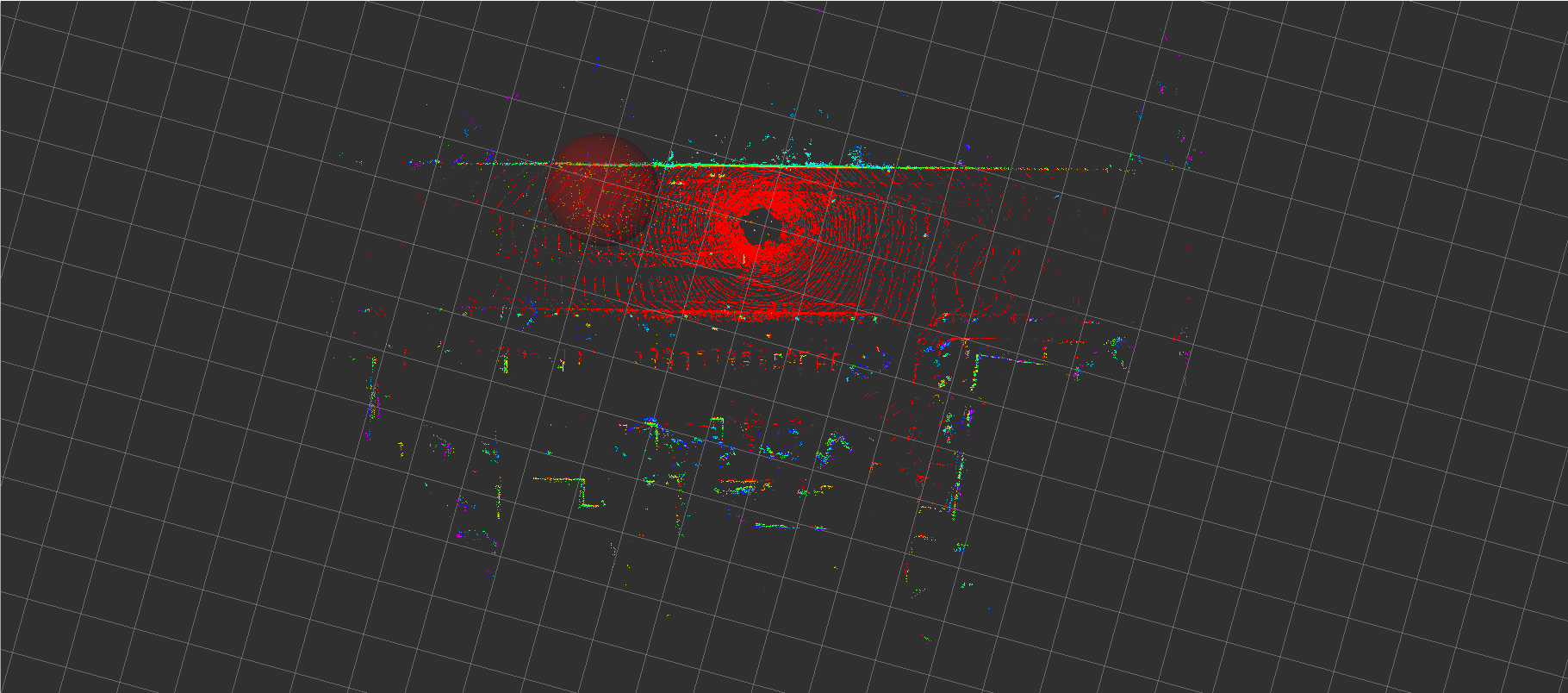
$ roscd hdl\_graph\_slam/rviz

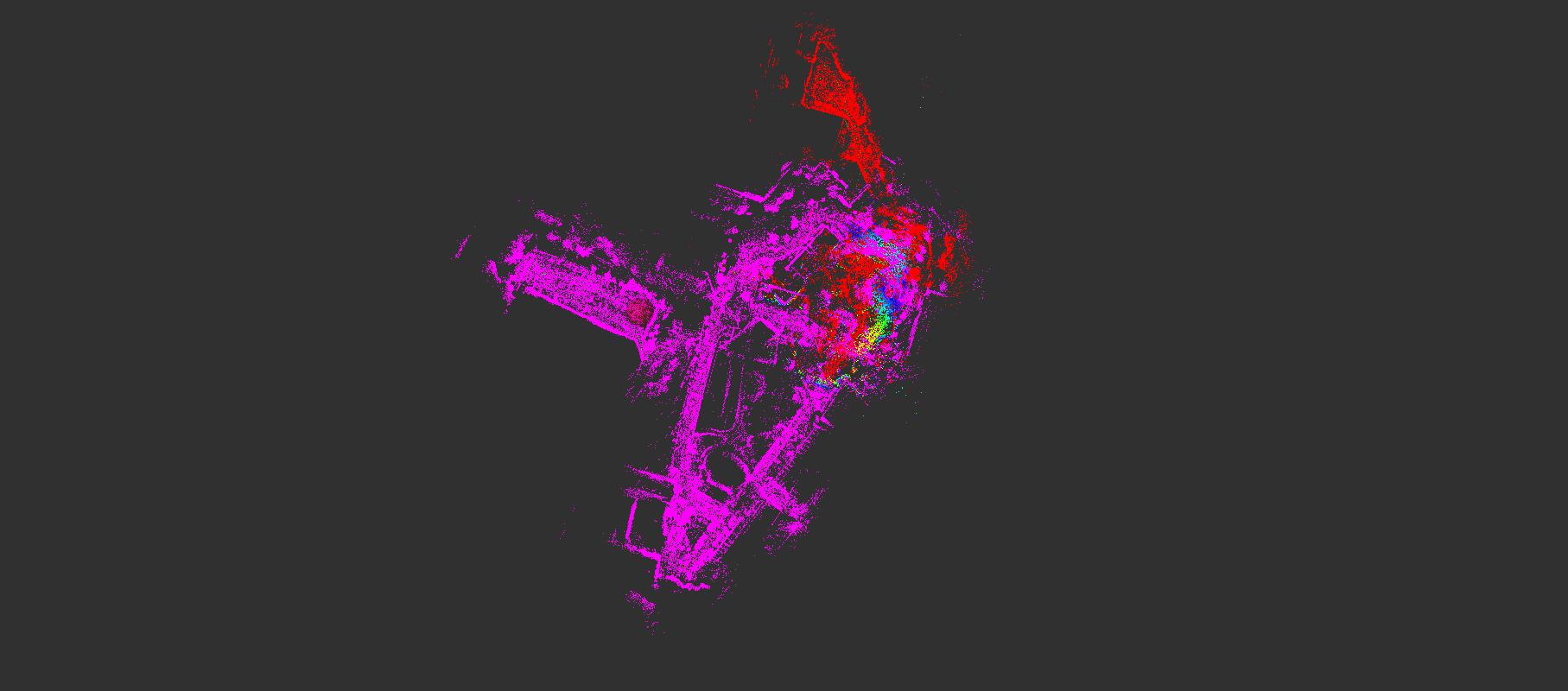
$ rviz -d hdl\_graph\_slam.rviz

$ rosbag play --clock <kitti rosbag file> --topics <name of topic with message type >

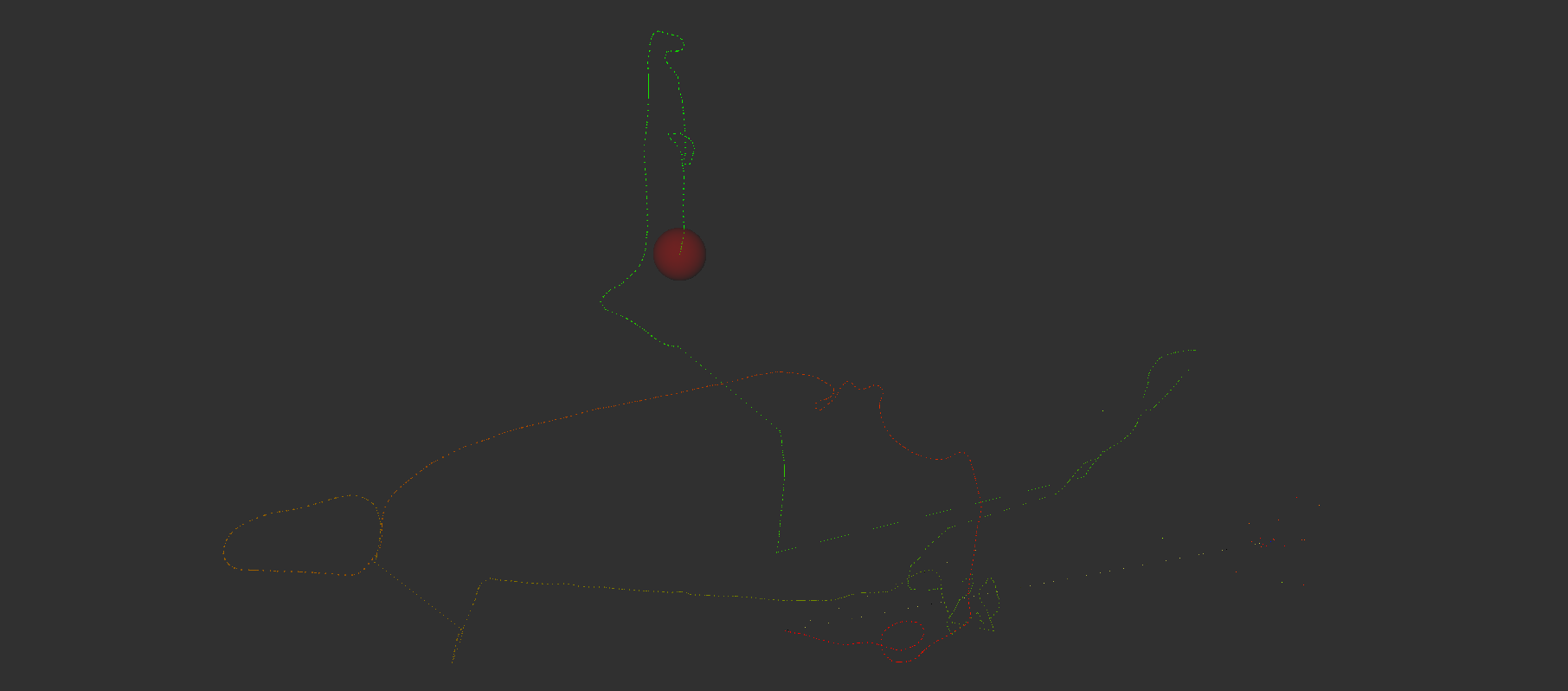
***Result***

The point cloud:

point cloud 1 of simpleKITTISDataset

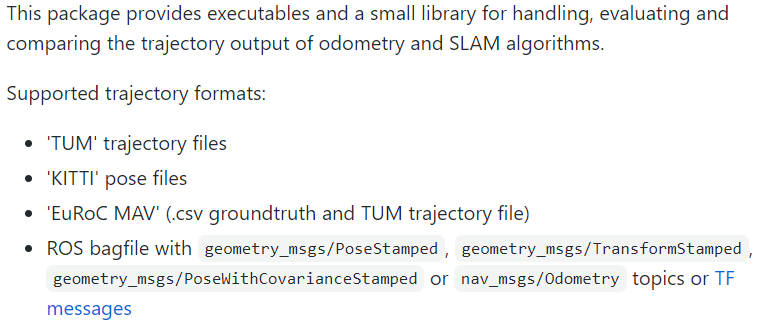
point cloud 2 of simpleKITTISDataset

point cloud 3 of StenvenDataset

path of StevenDataset

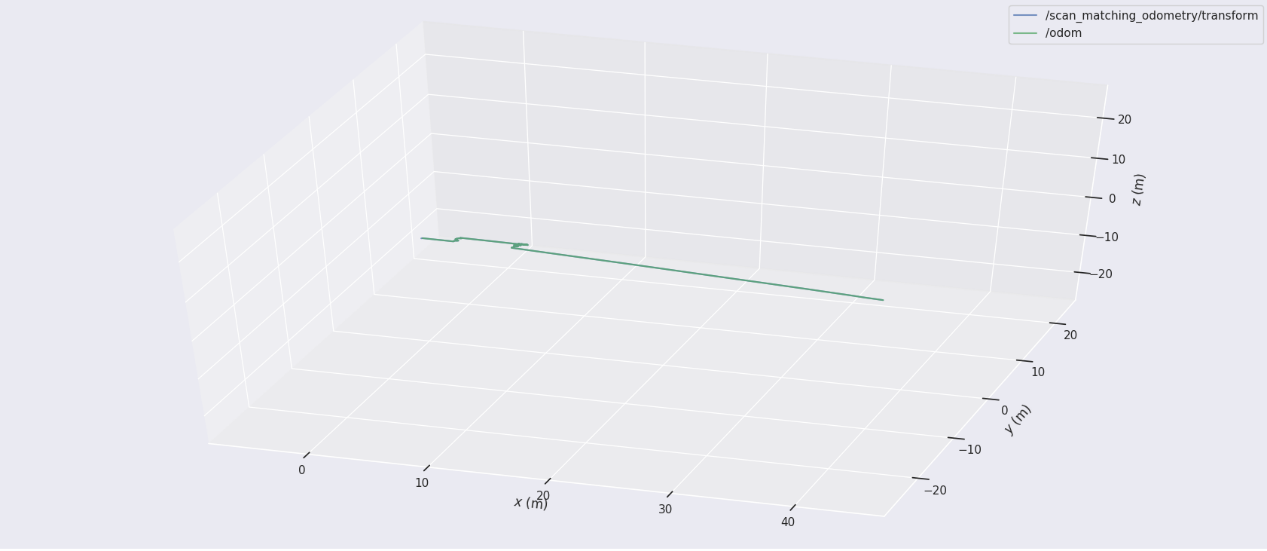
evo evaluate:

the evo tools: <https://github.com/MichaelGrupp/evo>

\*Remember to source the ros installation and the evo installation in each new terminal.

Find the topic with upper message type by $ rostopic list. Then record them.

$ rosbag record <topic name>

Use the bag file to plot.

Trajectory of simpleKITTIDataset

Get the ground truth.

Use the evo\_traj to caculate the error.

$

**lidarslam\_ros2**

**Operating System:**

ubuntu 18.04

**Ros Version:**

ros melodic + ros2 eloquent

**Software:**

lidarslam\_ros2

**Introduction:**

***Summary***

ros2 slam package of the frontend using OpenMP-boosted gicp/ndt scan matching and the backend using graph-based slam. lidarslam\_ros2 is a ROS2 package of the frontend using OpenMP-boosted gicp/ndt scan matching and the backend using graph-based slam. I found that even a four-core laptop with 16GB of memory could work in outdoor environments for several kilometers with only 16 line LiDAR. (WIP)

**io**

***frontend(scan-matcher)***

*input*

**·**/input\_cloud (sensor\_msgs/PointCloud2)

**·**/tf(from "base\_link" to LiDAR's frame)

**·**/initial\_pose (geometry\_msgs/PoseStamed)(optional)

**·**/imu (sensor\_msgs/Imu)(optional)

**·**/tf(from "odom" to "base\_link")(Odometry)(optional)

*output*

**·**/current\_pose (geometry\_msgs/PoseStamped)

**·**/map (sensor\_msgs/PointCloud2)

**·**/path (nav\_msgs/Path)

**·**/tf(from "map" to "base\_link")

**·**/map\_array(lidarslam\_msgs/MapArray)

***backend(graph-based-slam)***

*input*

**·**/map\_array(lidarslam\_msgs/MapArray)

*output*

**·**/modified\_path (nav\_msgs/Path)

**·**/modified\_map (sensor\_msgs/PointCloud2)

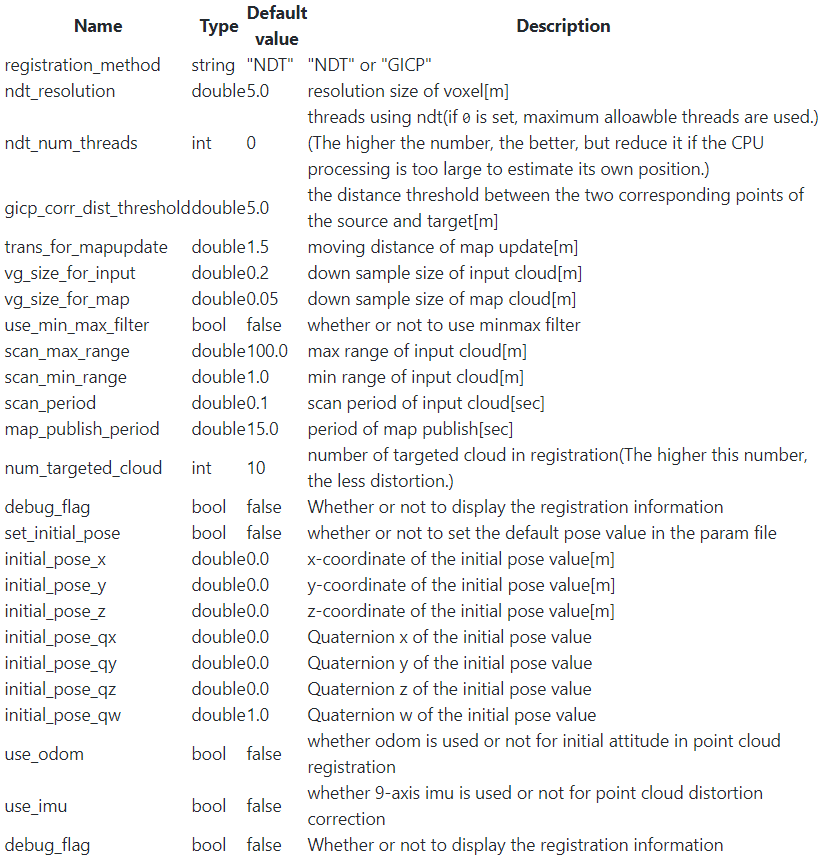
*srv*

**·**/map\_save (std\_srvs/Empty)

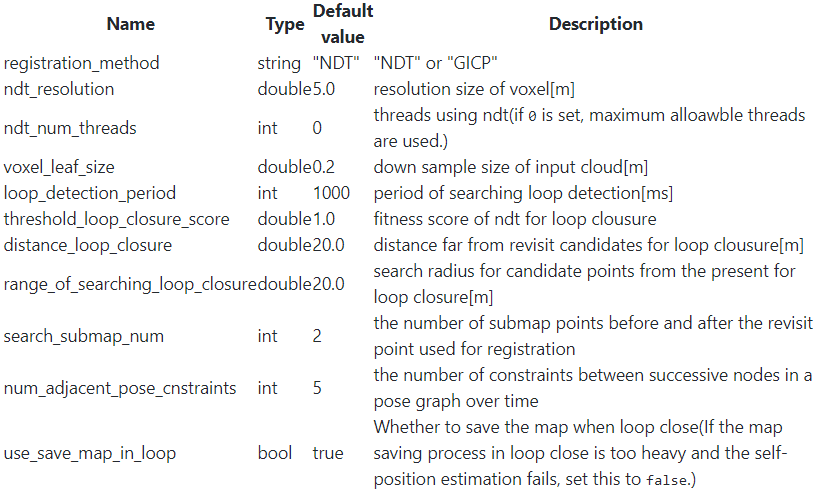
pose\_graph.g2o and map.pcd are saved in loop closing or using the following service call.

ros2 service call /map\_save std\_srvs/Empty

***params***

*frontend(scan-matcher)*

*backend(graph-based-slam)*



***Process:***

\*Remember to source the ros1 installation firstly, ros2 installation secondly, and the hdl\_graph\_slam installation thirdly in each new terminal.

\*The ros2 cannot play the rosbag file directly. To play a rosbag of ros1 on ros2, we need to install not only ros2 but also ros1.

For ubuntu 20.04, the ros1 we are installing is ros noetic. Next, we need to install not only rosbag2 but also rosbag2\_bag\_v2.

https://github.com/ros2/rosbag2\_bag\_v2

$ sudo apt install -y ros-foxy-rosbag2-bag-v2-plugins

See README.md in rosbag2\_bag\_v2 for details.

$ rviz2 -d src/lidarslam\_ros2/lidarslam/rviz/mapping.rviz

$ ros2 launch lidarslam lidarslam.launch.py

$ ros2 bag play -s rosbag\_v2 <bag file>

***Result:***