hdl_graph_slam

Operating System:
ubuntu 18.04

Ros Version:
ros melodic

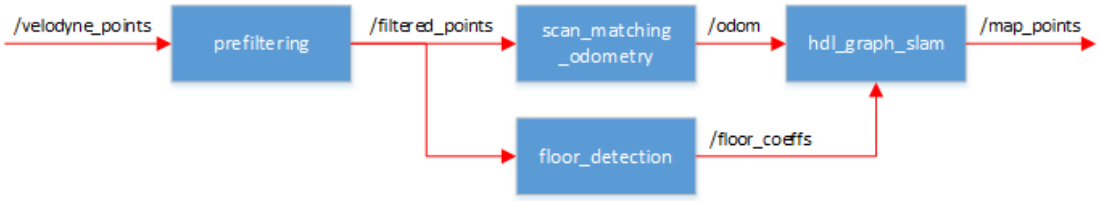
Software:
hdl_graph_slam

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

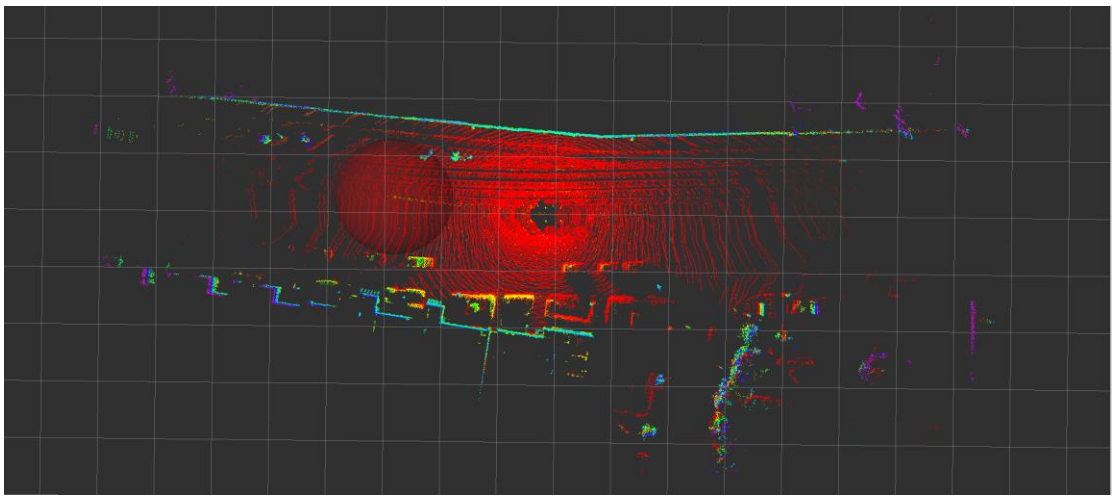
$ rosparam set use_sim_time true
$ roslaunch hdl_graph_slam hdl_graph_slam.launch

$ roscd hdl_graph_slam/rviz
$ rviz -d hdl_graph_slam.rviz

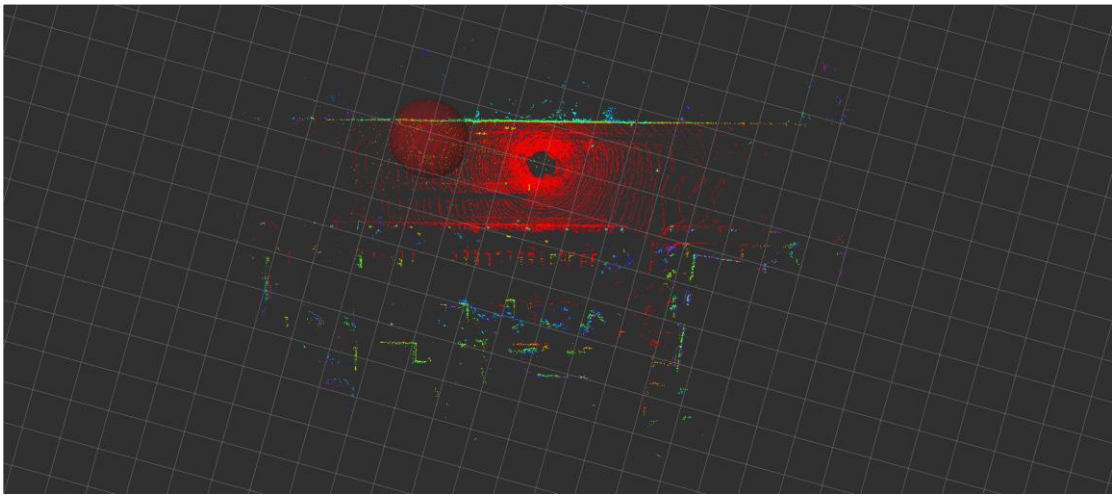
$ rosbag play --clock <kitti rosbag file> --topics <name of topic with message type >

Result

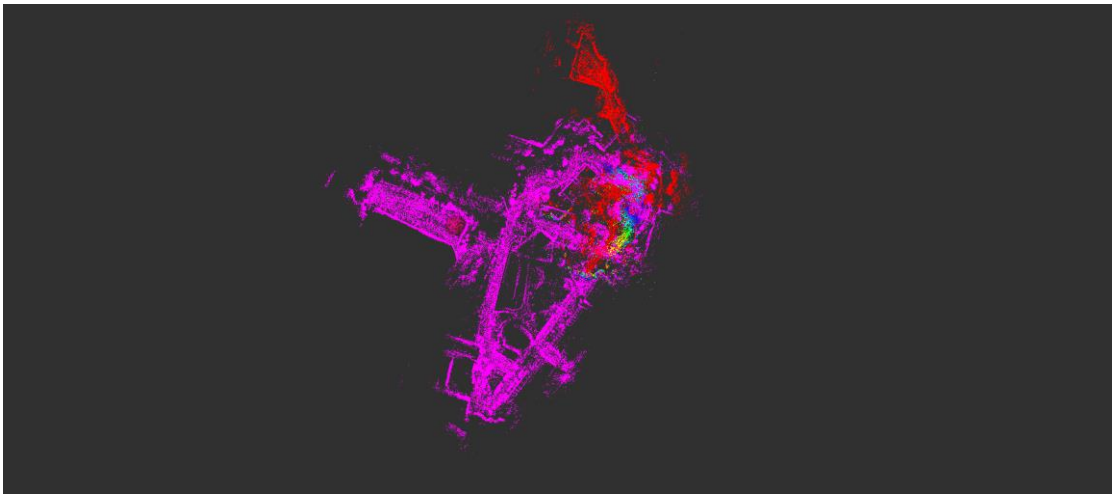
The point cloud:



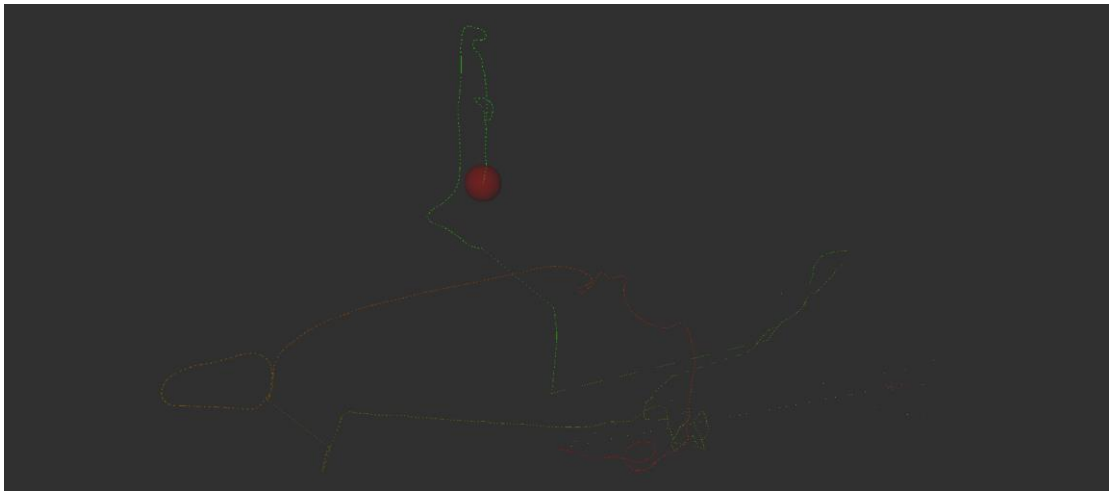
point cloud 1 of simpleKITTISDataset



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.

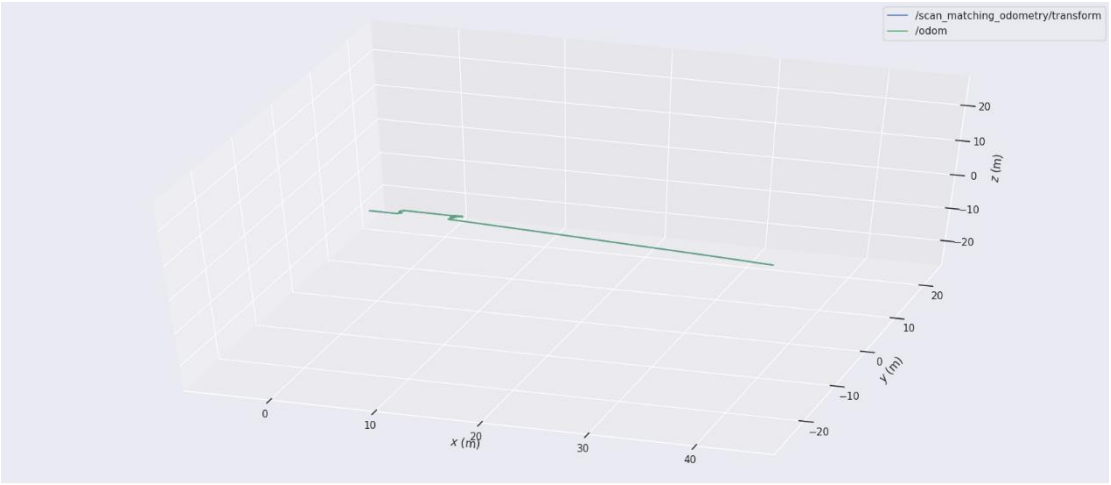
This package provides executables and a small library for handling, evaluating and comparing the trajectory output of odometry and SLAM algorithms.

Supported trajectory formats:

- 'TUM' trajectory files
- 'KITTI' pose files
- 'EuRoC MAV' (.csv groundtruth and TUM trajectory file)
- ROS bagfile with `geometry_msgs/PoseStamped`, `geometry_msgs/TransformStamped`, `geometry_msgs/PoseWithCovarianceStamped` or `nav_msgs/Odometry` topics or TF messages

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)

Name	Type	Default value	Description
registration_method	string	"NDT"	"NDT" or "GICP"
ndt_resolution	double	5.0	resolution size of voxel[m]
ndt_num_threads	int	0	threads using ndt(if `0` is set, maximum alloawble threads are used.) (The higher the number, the better, but reduce it if the CPU processing is too large to estimate its own position.)
gicp_corr_dist_threshold	double	5.0	the distance threshold between the two corresponding points of the source and target[m]
trans_for_mapupdate	double	1.5	moving distance of map update[m]
vg_size_for_input	double	0.2	down sample size of input cloud[m]
vg_size_for_map	double	0.05	down sample size of map cloud[m]
use_min_max_filter	bool	false	whether or not to use minmax filter
scan_max_range	double	100.0	max range of input cloud[m]
scan_min_range	double	1.0	min range of input cloud[m]
scan_period	double	0.1	scan period of input cloud[sec]
map_publish_period	double	15.0	period of map publish[sec]
num_targeted_cloud	int	10	number of targeted cloud in registration(The higher this number, the less distortion.)
debug_flag	bool	false	Whether or not to display the registration information
set_initial_pose	bool	false	whether or not to set the default pose value in the param file
initial_pose_x	double	0.0	x-coordinate of the initial pose value[m]
initial_pose_y	double	0.0	y-coordinate of the initial pose value[m]
initial_pose_z	double	0.0	z-coordinate of the initial pose value[m]
initial_pose_qx	double	0.0	Quaternion x of the initial pose value
initial_pose_qy	double	0.0	Quaternion y of the initial pose value
initial_pose_qz	double	0.0	Quaternion z of the initial pose value
initial_pose_qw	double	1.0	Quaternion w of the initial pose value
use_odom	bool	false	whether odom is used or not for initial attitude in point cloud registration
use_imu	bool	false	whether 9-axis imu is used or not for point cloud distortion correction
debug_flag	bool	false	Whether or not to display the registration information

backend(graph-based-slam)

Name	Type	Default value	Description
registration_method	string	"NDT"	"NDT" or "GICP"
ndt_resolution	double	5.0	resolution size of voxel[m]
ndt_num_threads	int	0	threads using ndt(if `0` is set, maximum alloawble threads are used.)
voxel_leaf_size	double	0.2	down sample size of input cloud[m]
loop_detection_period	int	1000	period of searching loop detection[ms]
threshold_loop_closure_score	double	1.0	fitness score of ndt for loop clousure
distance_loop_closure	double	20.0	distance far from revisit candidates for loop clousure[m]
range_of_searching_loop_closure	double	20.0	search radius for candidate points from the present for loop closure[m]
search_submap_num	int	2	the number of submap points before and after the revisit point used for registration
num_adjacent_pose_cnstraints	int	5	the number of constraints between successive nodes in a pose graph over time
use_save_map_in_loop	bool	true	Whether to save the map when loop close(If the map saving process in loop close is too heavy and the self-position estimation fails, set this to `false`.)

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: