Implementation and Evaluation of Odometry for Rover SLAM

Wang Zhiwei[1] and Yan Yihui[1]

A project of the 2019 Robotics Cource of the School of Information Science and Technology (SIST) of Shanghaitech University

Abstract. This project is mainly about the implementation and evaluation of the odometry of Rover. We compare the trajectory generated by our odometry with the trajectory published by hector_slam which shows that our odometry is good enough. To some extent, the error of our odometry reflects the imperfection of Rover.

1 Introduction

SLAM(simultaneous localization and mapping) is central to a range of indoor, outdoor, in-air and underwater applications for both manned and autonomous vehicles. Because Rover is imperfective, our project aims to implement an odometry on Rover, which is a crutial base stone of SLAM. Our endeavor to implement the good enough odometry will be meaningful to the further job.

2 State of the Art

2.1 Wang Zhiwei

Present and cite [1]This paper is about a SLAM algorithm called tinySLAM based on IML tech- nique (Incremental maximum likelihood), which allows the Simultaneous Localization and Mapping using data from a laser sensor and using parrallel localization loops, which runs SLAM algorithm with several parameters simultaneously and choose the best position estimation.

[2]This paper is about a SLAM algorithm called vinySLAM method that extends tinySLAM with a cell model based on the Transferable Belief Model (TBM) and an updated scan matching function to improve robustness, which supposed to be run on low-cost platforms.

[3]This paper is about some metrics for evaluation of 2D SLAM, including Cartographer, tinySLAM, gmapping, hectorSLAM, tinySLAM and vinySLAM, which presents the approach for comparison of SLAM algorithms that allows to find the most accurate one.

Further The further is about the second paper.

This paper introduces the vinySLAM method that enhances tinySLAM with the Transferable Belief Model to improve its robustness and accuracy, which provide information about proximities to nearby obstacles with a laser scanner and supposed to be used in an indoor environment.

TinySLAM may fail if a function that determines a correlation between a scan obtained at a given robot pose and a map doesn't have a single narrow peak. To deal with these issues, this paper uses multiple hypotheses tracking and a graph-based map representation to alter already estimated transformations which implies a robust solution for the loop closure subproblem.

Beside, there are several alternatives for low-cost hardware.The TBM-based cell model can be used to handle noisy laser data and to make a scan matcher more robust. And a cache-friendly way to handle graph-map optimization[4] can be used to speed up a graph-based SLAM.

The evaluation on publicly available datasets shown that vinySLAM outperforms tinySLAM. Although it is less robust than GMapping and Cartographer due to its single hypothesis tracking nature, it's able to process sensor data at the 10 Hz rate on a low-cost hardware unlike GMapping.So considering the accuracy and possible optimizations, it may be reasonable to use vinySLAM on low-cost platforms including a life-long SLAM case.

ROS Package slam_constructor

The package provides implementation of several 2D laser-based simultaneous localization and mapping (SLAM) algorithms (tinySLAM, vinySLAM, GMapping) created with the SLAM constructor framewor, which provides common functionality and classes that may be used to create custom SLAM algorithms.

- The gmapping SLAM provides laser-based SLAM, as a ROS node called slam_gmapping, which can create a 2-D occupancy grid map (like a building floorplan) from laser and pose data collected by a mobile robot.
- The tinySLAM is one of the most simplest and lightweight laser-based SLAM methods.
- The vinySLAM is the enhanced tinySLAM with the Transferable Belief Model.

Current implementation requires odometry data and laser scans to be provided by the ROS topics. It also supposes that a laser scanner is fixed in (0, 0) of a robot and mounted horizontally.

ROS Package hector_slam

The hector_slam metapackage that installs hector_mapping, hector_geotiff and hector_trajectory_server packages.

- hector_mapping is a SLAM approach that can be used without odometry as well as on platforms that exhibit roll/pitch motion (of the sensor, the platform or both). It leverages the high update rate of modern LIDAR systems and provides 2D pose estimates at scan rate of the sensors. While the system does not provide explicit loop closing ability, it is sufficiently accurate for many real world scenarios.

– hector_geotiff provides a node that permits saving of map and robot trajectory data.
– hector_trajectory_server keeps track of tf trajectories extracted from tf data and makes this data accessible via a service and topic.

2.2 Yan Yihui

Present and cite The graph-based SLAM is a method introduced by Lu and Milios[5]. [6] has used this method in urban environments and mapped a number of urban sites. [7] presented a tutorial on graph-based SLAM with sufficient details and insights to allow for an easy implementation of the proposed methods. With the aim to reduce execution time, [4] proposed an efficient implementation and tried to speed up a graph-based SLAM.

Further [4] presented a practical approach to implement a 3D graph-based SLAM algorithm on an OMAP embedded architecture, which is a widely used open multimedia applications platform. provided an optimized data structure and an efficient memory access management to solve the nonlinear least squares problem related to the algorithm. The algorithm takes advantage of the Schur complement to reduce the execution time. It takes advantage of the multi-core architecture to parallelize the algorithm. This work aims to demonstrate how optimizing data structure and multi-threading can decrease significantly the execution time of the graph-based SLAM on a low-cost architecture dedicated to embedded applications.

ROS Package move_base

The move_base package provides an implementation of an action that, given a goal in the world, will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task. It supports any global planner adhering to the nav_core::BaseGlobalPlanner interface specified in the nav_core package and any local planner adhering to the nav_core::BaseLocalPlanner interface specified in the nav_core package. The move_base node also maintains two costmaps, one for the global planner, and one for a local planner that are used to accomplish navigation tasks.

ROS Package pointcloud_to_laserscan

Converts a 3D Point Cloud into a 2D laser scan. This is useful for making devices like the Kinect appear like a laser scanner for 2D-based algorithms.

2.3 Problems

Odometry The odometry has not been implemented in the Rover. Many packages are unable to do mapping without odometry, thus, we decide to implement the odometry of Rover firstly.

3 System Description

3.1 Motor Control

The rover has 10 motor, including 6 motors in profile velocity mode, setting velocity to control rover moving at a constant speed and 4 motors in profile position mode, setting the position of the wheels to control rover turning to target degree, as shown in figure 1. We choose Maxon, a Swiss manufacturer and supplier of high-precision drive systems, as the rover's motor provider. According to the Command library of EPOS Positioning Controllers, we can set every wheel's velocity and degree based on the Ackerman steering [8].

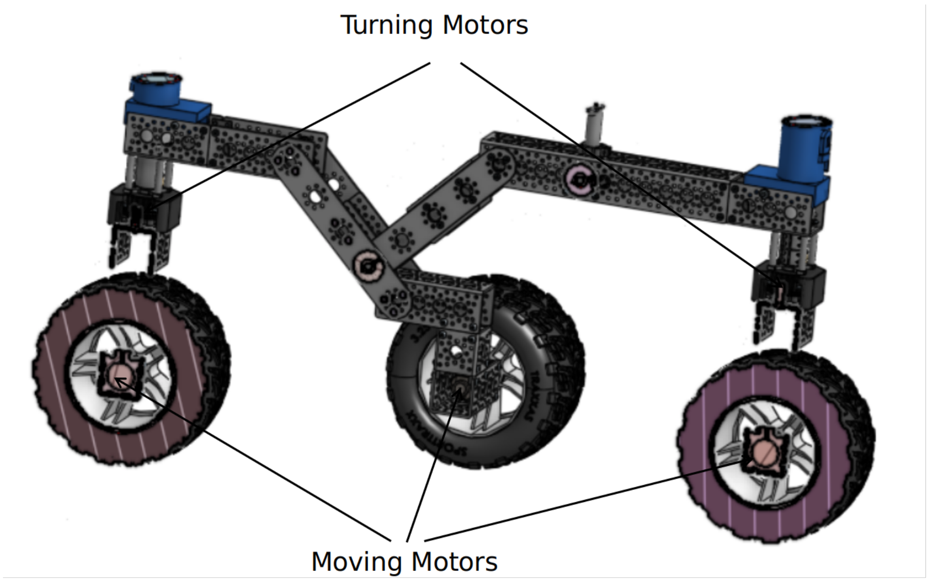


Fig. 1. Section view of 6-wheels Ackerman steering rover.

Besides, we should get current positions and velocities of ten motors in order to implement the odometry of rover. From the motor controllers library, we use the $VCS\_GetPositionIs$ function to obtain the current position actual values of the ten motor. $VCS\_GetPositionIs$ returns the position actual value, $pPosition$ and requires Handle for port access $KeyHandle$ with Node ID of the addressed device $NodeId$ as the parameters, as shown in the following C++ code.

```
int Motor::getPosition(unsigned short usNodeId) {
    int pPosition;
    int lResult = MMC_SUCCESS;

    if (VCS_GetPositionIs(g_subkeyHandle_, usNodeId, &
       pPosition,
                                                &ulErrorCode_) == 0) {
        lResult = MMC_FAILED;
         LogError('VCS_GetTargetVelocity', lResult,
            ulErrorCode_);
    }

    return pPosition;
}
```

Then, we transform the position value to degree using *tick*2*deg* function. Besides, we also get current $time_stamp$ from ros system. For the 6 moving motors, we can calculate the actual moving distances of the six wheels from last sampling , combing with the wheel radius. For the 4 turning motors, we can calculate the current degrees of the front and rear wheels from the zero points. Then, we public the topic of motor, called $'motor'$, including the moving distances, turning degrees and time stamp for the calculation of rover odometry.

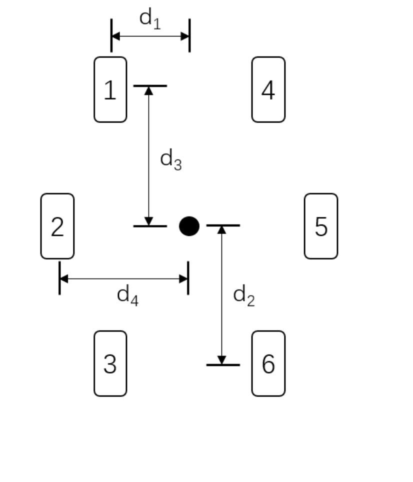
3.2 Odometry Calculations

After obtaining the status of the motors, the directions of corner motors and the moving distances of drive motors, we can calculate the displacement and rotation of the rover. The rover moves in two ways, moving straightly and turning. In this subsection, we will describe them respectively.

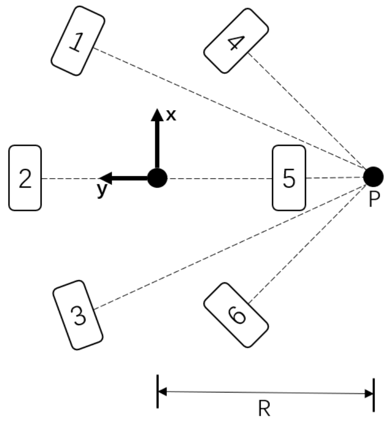
Moving Straightly As figure 2(a) showing, calulating the displacement of Rover is easy, just averaging the movement of six wheels:

$$\Delta x = \frac{1}{6}\sum_{i}^{6} s_i,\ \ \Delta y = 0,\ \ \Delta\theta = 0$$

Turning As figure 2(b) showing, Rover it is turning about point P. What we know is the directions of corner motors (α_1, α_3, α_4, α_6), the moving distances of drive motors (s_1, s_2, s_3, s_4, s_5, s_6) and the horizontal/vertical distance between the center of Rover and each wheel (d_1, d_2, d_3, d_4). The moving_direction (forward: 1, backward: -1, still: 0) and turning_direction (right: -1, left: 1, straight: 0) is set according the sign of the motor status values.



(a) Model of straightly moving Rover.



(b) Model of turning Rover.

Fig. 2. Models of Rover.

Firstly, we need to calculate the rotation radius of the center of Rover (r).

$$r_1 = \frac{d_3}{tan(\alpha_1)} - d_1, \quad r_3 = \frac{d_3}{tan(\alpha_3)} - d_1$$

$$r_4 = \frac{d_2}{tan(\alpha_4)} + d_1, \quad r_6 = \frac{d_2}{tan(\alpha_6)} + d_1$$

$$r = \sum_{i=1,3,4,6} r_i$$

Then we need to know how much degrees Rover rotates ($\mathit{\Delta}\theta$).

$$\mathit{\Delta}\theta_1 = \frac{s_1 * sin(\alpha_1)}{d_3}, \quad \mathit{\Delta}\theta_3 = \frac{s_3 * sin(\alpha_3)}{d_2}$$

$$\mathit{\Delta}\theta_4 = \frac{s_4 * sin(\alpha_4)}{d_3}, \quad \mathit{\Delta}\theta_6 = \frac{s_6 * sin(\alpha_6)}{d_2}$$

$$\mathit{\Delta}\theta = \sum_{i=1,3,4,6} \mathit{\Delta}\theta_i$$

After calculating r and $\mathit{\Delta}\theta$, we can get the translation of Rover.

$$\mathit{\Delta}x = (moving\_direction) * r * sin(\mathit{\Delta}\theta)$$

$$\mathit{\Delta}y = (turning\_direction) * r * (1 - cos(\mathit{\Delta}\theta))$$

Now, we have the transform($\mathit{\Delta}x$, $\mathit{\Delta}y$, $\mathit{\Delta}\theta$) of Rover in each time slot($\mathit{\Delta}t$). It is easy to get the odometry.

3.3 System Overview

Hardware

The system overview of rover is shown in figure 3, including lidar sensor, motors and main computer.

Lidar Sensor : We mount a RoboSense LiDAR $RS-LiDAR-16$ on the top of our rover. The compact housing of $RS-LiDAR-16$ mounted with 16 laser/detector pairs rapidly spins and sends out high-frequency laser beams to continuously scan the Surrounding environment. Advanced digital signal processing and ranging algorithms calculate point cloud data and reflectivity of objects to enable machine to 'see' the world and providing reliable data for localization, navigation and obstacle avoidance. We use the LiDAR to run hectorSLAM for odometry evaluation and mapping.

Motors : We use Brushless DC Motors from Maxon Motors. The electronically commutated maxon EC motors stand out with excellent torque characteristics, high power, an extremely wide speed range, and an outstandingly long life span. The outstanding controllability of the motors makes high-precision positioning drives possible. It has been optimized for high speeds (up to 120,000 rpm) and withstands up to 2000 autoclave cycles.

Main Computer and Motor control : We use raspberry pi 3b+ as the robot brain which is a small single-board computers with a 1.2 GHz 64-bit quad core processor, on-board 802.11n Wi-Fi, Bluetooth and USB boot capabilities, communicating with the motor control via Universal Serial Bus (USB) port and running SLAM algorithms. We use maxon EPOS as the motor control. EPOS is a modular, digital positioning controller by maxon. The wide range of operating modes, as well as various command interfaces, make it versatile for use in many different drive systems in the fields of automation technology.

ROS-based Software The software of rover is developed based on Robot Operation System(ROS), which is is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. We use ROS to connect our rover in minutes and implement remote control and SLAM.

First, we public the $'motor'$ topic about the position information from motor control. Based on the $'motor'$ topic, we calculate and public the odometry and transform(tf) of rover from Ackerman steering. Besides, we get rslidar_point from RoboSense LiDAR and use pointcloud_to_laserscan to obtain *scan* topic preparing for hectorSLAM. The ROS computation graph visualized by *rqt_graph* is shown in figure 4.

4 System Evaluation

Hector slam uses the hector_mapping node for learning a map of the environment and simultaneously estimating the platform's 2D pose at laser scanner

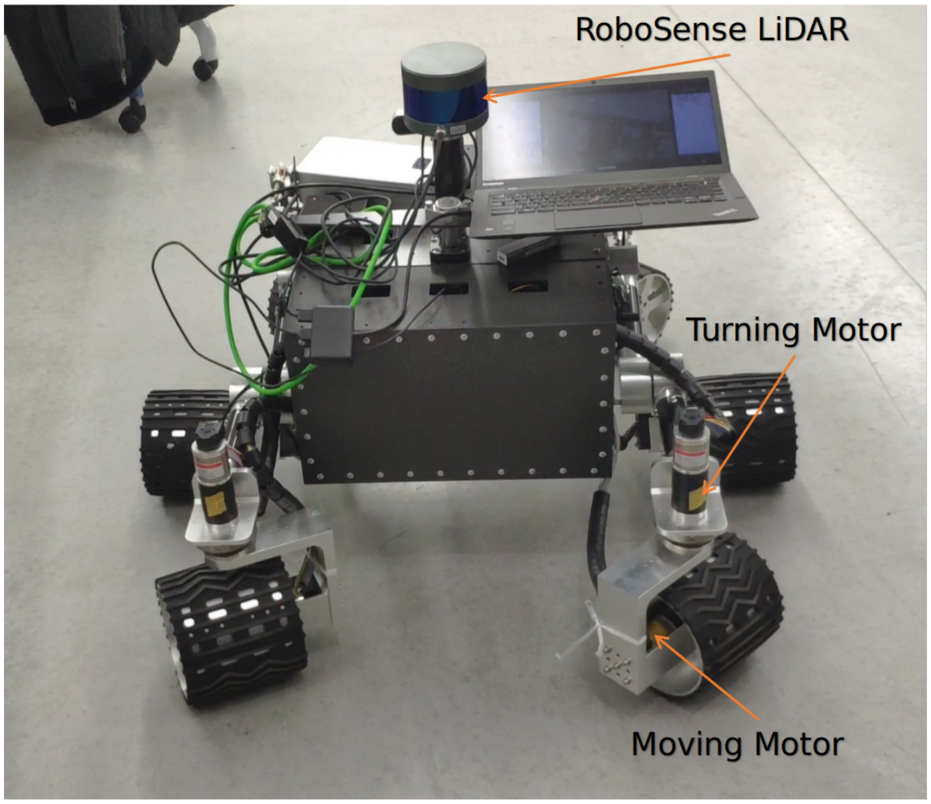


Fig. 3. The Overview of Our Rover.

frame rate without odometry. We input the laser_scan topic provided by pointcloud_to_laserscan node in the hector_mapping node and set the the current estimate of the robot's pose within the map frame, called 'pub_map_odom_transform'. Then, from hectorSLAM, we can get the estimated 2D pose of rover. While the system does not provide explicit loop closing ability, the hectorSLAM is sufficiently accurate for many real world scenarios. So we consider the estimated 2D pose as the ground truth.

Figure 5, 6 show the trajectory from odometry and 2D pose estimated by hectorSLAM. The green trajectory is generated by odometry and the yellow one is generated by hectorSLAM. It can be seen from the two figures that our odometry succeeds in determining the position and orientation of rover. However, some deviations exist between the trajectory from odometry and 2D pose estimated by hectorSLAM, partly due to the offset from the zero point of turning motor. Figure 7 shows the comparison of total trajectory and figure 8 shows the 2D map generated by rover with hector_slam.

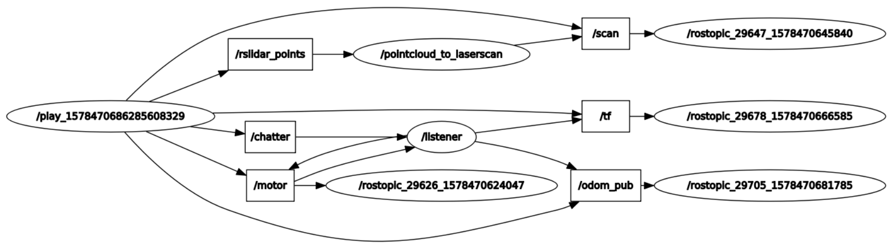

/play_1578470686285608329
/rslidar_points
/pointcloud_to_laserscan
/scan
/rostopic_29647_1578470645840
/tf
/rostopic_29678_1578470666585
/chatter
/listener
/motor
/rostopic_29626_1578470624047
/odom_pub
/rostopic_29705_1578470681785

Fig. 4. Visualizing the ROS computation graph.

5 Conclusion

Our project is mainly about the odometry of Rover, the crutial base stone of mapping and navigation. We obtain the motor status with maxon's library, and then implement the odometry on Rover using the motor status. We compare the trajectory generated by our odometry with the trajectory published by hector_slam. The result shows that our odometry is good enough, although there are some errors caused by the accuracy of motor status and the initial position of corner wheels. The motor status converting from the encoder is not perfectlly equal to the real value. The initial position of corner wheels is bias from the straight forward direction.

6 README

6.1 Relevant Packages

motor_control This package is responsble for Rover controlling with remote control handle and publish the odometry of Rover.

ros_rslidar This is the lidar's driver of RoboSense.

pointcloud_to_laserscan This package converts a 3D point cloud into a 2D laser scan.

hector_slam This package can do mapping without odometry and publish the trajectory of lidar.

6.2 Usage

– Modify the path of configure file in motor_control/src/listener.cpp:
  std::string file_name = "/home/rov/motor/src/motor_control/config/default.yaml";

– launch file: motor_control/launch/rover_slam.launch

NOTE: If there are errors like ”VCS_***”, you can modify the value of can_baudrate in ”/motor_control/config/default.yaml” from 125000 to 50000, or from 50000 to 125000.

6.3 Code Instruction

– motor_control/src/listener.cpp: publish motor status, subscribe the moving order for controlling motors and subscribe the motor staus for updating odometry.
– motor_control/src/odometry.cpp: the code of odometry.

References

1. O. E. Hamzaoui and B. Steux, “SLAM algorithm with parallel localization loops: TinySLAM 1.1,” in *2011 IEEE International Conference on Automation and Logistics (ICAL)*, Aug. 2011, pp. 137–142.
2. A. Huletski, D. Kartashov, and K. Krinkin, “VinySLAM: An indoor SLAM method for low-cost platforms based on the Transferable Belief Model,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 6770–6776.
3. A. Filatov, A. Filatov, K. Krinkin, B. Chen, and D. Molodan, “2D SLAM quality evaluation methods,” in *2017 21st Conference of Open Innovations Association (FRUCT)*, Nov. 2017, pp. 120–126.
4. A. Dine, A. Elouardi, B. Vincke, and S. Bouaziz, “Graph-based SLAM embedded implementation on low-cost architectures: A practical approach,” in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 4612–4619.
5. F. Lu and E. Milios, “Globally Consistent Range Scan Alignment for Environment Mapping,” p. 31.
6. S. Thrun and M. Montemerlo, “The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures,” *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, May 2006. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364906065387
7. G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, “A Tutorial on Graph-Based SLAM,” *IEEE Intell. Transport. Syst. Mag.*, vol. 2, no. 4, pp. 31–43, 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5681215/
8. Michael Cox, Eric Junkins, Olivia Lofaro, “Open source Rover: Software controls,” https://github.com/nasa-jpl/open-source-rover/blob/master/Software/Software%20Controls.pdf, (accessed 2020-1-8).

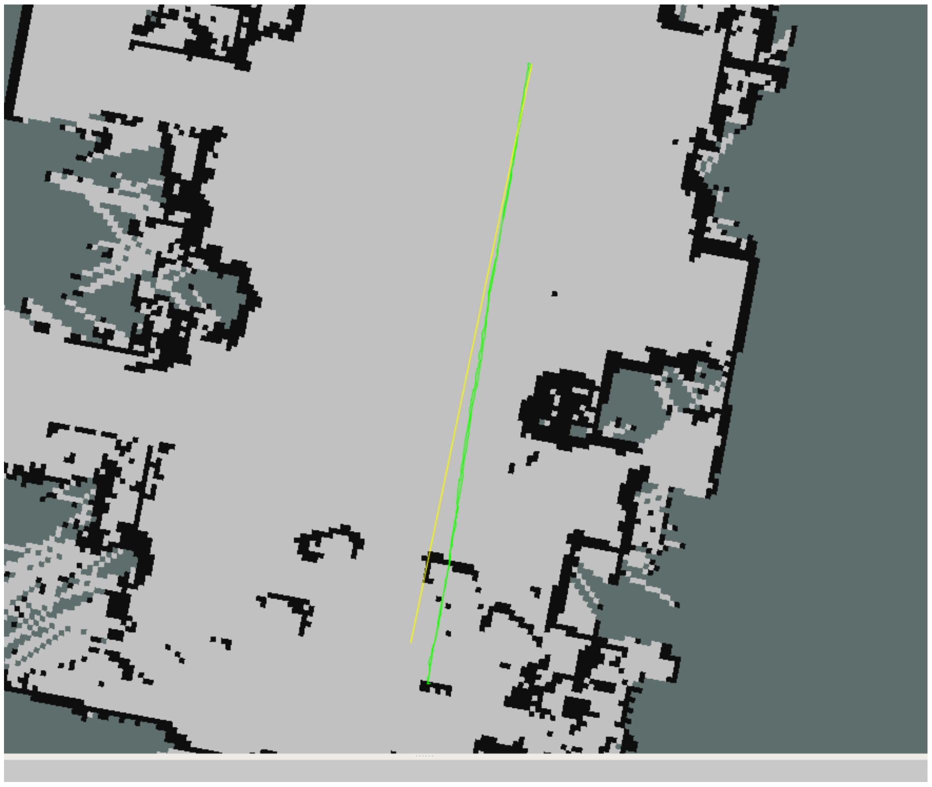


Fig. 5. Comparison of straight trajectory.

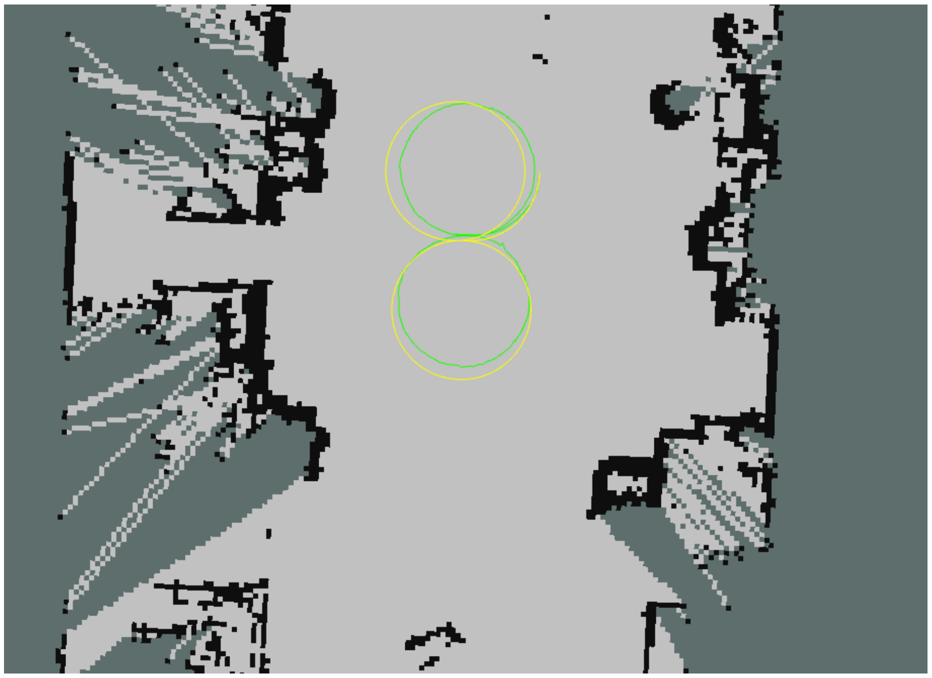


Fig. 6. Comparison of turning trajectory.

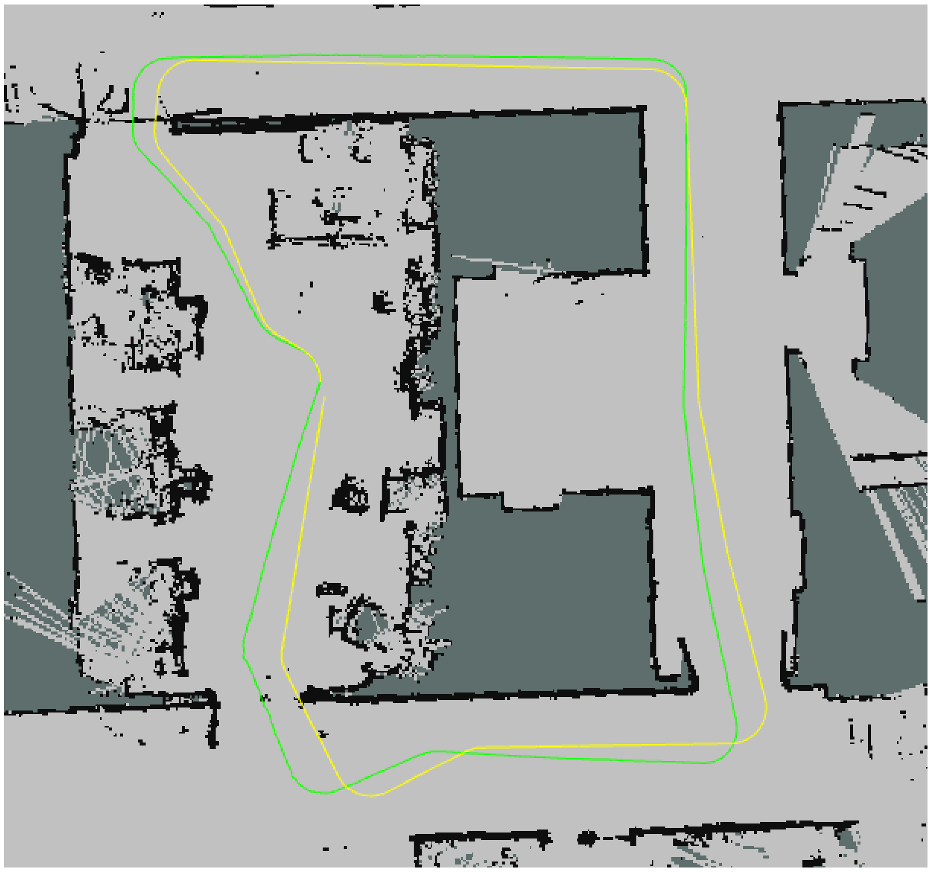


Fig. 7. Comparison of total trajectory.

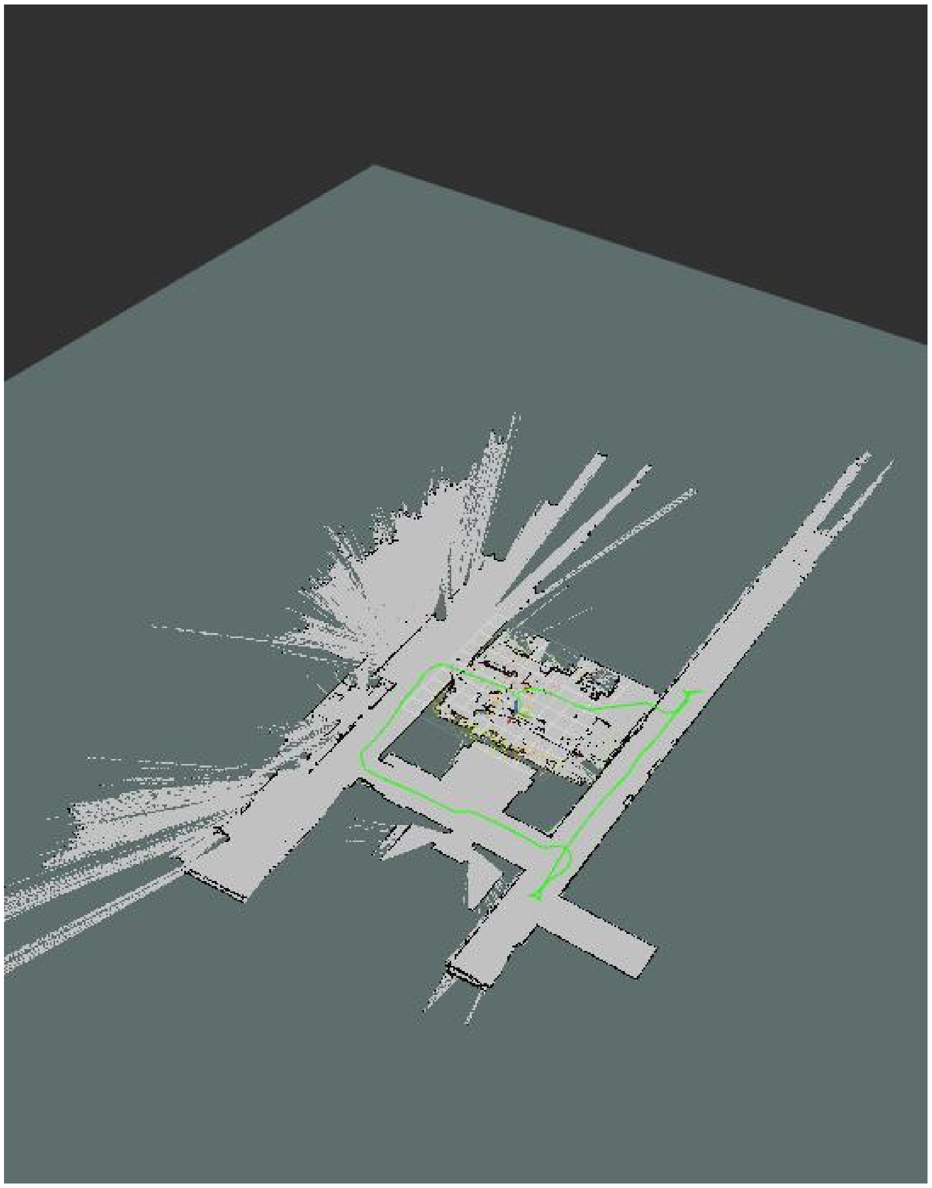


Fig. 8. 2D map generated by hector_slam