Autonomous navigation and mapping of RoboCup Rescue Competition

A project of the 2017 Robotics Course of the School of Information Science and Technology (SIST) of ShanghaiTech University

`https://robotics.shanghaitech.edu.cn/teaching/robotics2017`

Instructor: Prof. Sören Schwertfeger

Xiting Zhao
ShanghaiTech University
zhaoxt@shanghaitech.edu.cn

Yang Zhou
ShanghaiTech University
zhouyang@shanghaitech.edu.cn

Jiadi Cui
ShanghaiTech University
cuijd@shanghaitech.edu.cn

***Abstract*— We designed and implemented perception system and navigation system of rescue robot of undergraduate RoboCup Rescue team for 2018 RoboCup Rescue Competition. We used RGB-D sensor and 2D LIDAR as our sensors of perception system which are cost-effective. We can provide 2D and 3D map and navigate in unseen environment for exploration task in RoboCup Rescue Competition.**

I. Introduction

The RoboCup Rescue Project is intended to promote research and development in disaster rescue which is a significant domain at various levels. The goal of the RoboCup Rescue competitions requires robots to show abilities of mobility, sensory perception, planning, mapping, and practical operator interfaces on searching for simulated victims in unstructured environments.

What we are participating is RoboCup Rescue Robot League (RRL) competitions, and we are focusing on solving exploration tasks in this project. The exploration tasks need teams to create 2D and/or 3D map of a dark Labyrinth while traversing, recognize objects including QR codes, fire extinguishers, doors, simulated victims, and other items, avoid amorphous negative obstacles along a robot's path, drive and map while avoiding amorphous terrain obstacles without enclosing walls. What we focused on is 2D and 3D mapping, navigation and obstacles avoiding.

As members of 2018 undergraduate RoboCup Rescue Team, we designed and implemented our perception and navigation system on our robot which was built by mechanical engineer of our team. We used Kinect 2.0 as our RGB-D sensor and YDLIDAR X4 as our 2D laser LIDAR. To make the problem statement clear, we will describe our task in more details. The problem for 2D/3d mapping is described as below. The dark labyrinth is constructed by wooden walls without significant visual features. The labyrinth is narrow which makes it hard to get some good feature like edge and corner to match. In many cases, the camera is capturing only a wall without any edge information since the camera is too close to the wall. It is quite hard to get decent 3D map only using visual SLAM algorithm. Since most walls are vertical with respect to the ground, the 2D SLAM algorithm can provide a quite good 2D map which is also the 2D projection

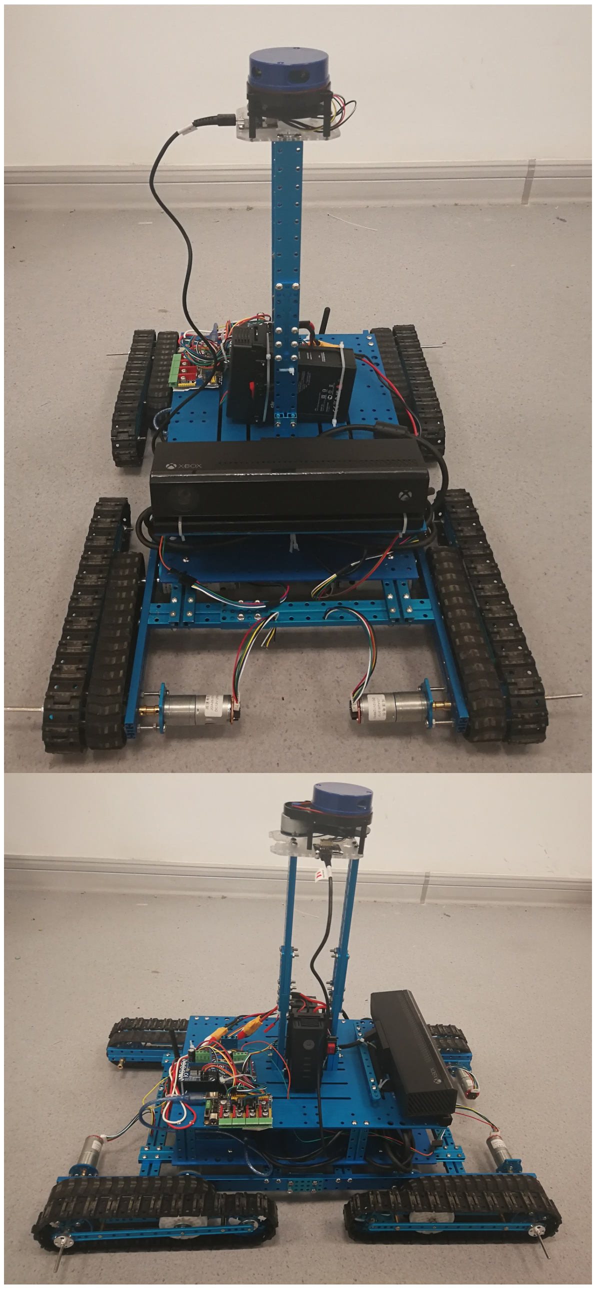


Fig. 1. Our Rescue Robot

of walls using 2D laser LIDAR. We combine 2D SLAM and 3D SLAM together and get good 3D map.

The problems for obstacle avoiding and navigation are described as below. Since there are some obstacles in the labyrinth. To avoid complicated control of going over these obstacles, we need to avoid them. These obstacles cannot be captured by 2D laser LIDAR since we mount the 2D laser LIDAR on the top of our robot. We used RGB-D sensor to get accurate location of these obstacles and projected these obstacles onto 2D map. The obstacles avoiding problem is simplified to 2D navigation problem. Since we can have 2D map of walls and projections of obstacles. We can just use standard 2D navigation algorithm to navigate.

II. State of the Art

A. Summaries of Some Related Works

3D Capturing Performances of Low-Cost Range Sensors for Mass-Market Applications [1]

This paper compared performances of different range sensors, including Kinect1, ASUS Xtion, Kinect2, Realsense F200, which concluded that the Kinect 2 has a lower random error with a long detection distance.

Calibration of Kinect-type RGB-D Sensors for Robotic Applications [2]

This paper described how to use chessboard to calibrate the Kinect. And principles of calibrating RGB and IR sensor.

Advancing the State of Urban Search and Rescue Robotics Through the RoboCupRescue Robot League Competition[3]

This paper described the importance of Robocup Rescue Competition and introduced Robocup Rescue Competition in details.

Unsupervised Learning of Depth and Ego-Motion from Video [4]

It presented an unsupervised learning framework for the task of monocular depth and camera motion estimation from unstructured video sequences. It used an end-to-end learning approach with view synthesis as the supervisory signal, requiring only monocular video sequences for training.

Parallel Tracking and Mapping for Small AR Workspaces[5]

This paper presented a method of estimating camera pose in an unknown scene. The result showed that this system can produce detailed maps with thousands of landmarks which can be tracked at frame-rate, with an accuracy and robustness rivalling that of state-of-the-art model-based systems.

A volumetric method for building complex models from range images [6]

This volumetric representation consists of a cumulative weighted signed distance function. It can integrate a large number of range images (as many as 70) yielding seamless, high-detail models of up to 2.6 million triangles.

ORB-SLAM: a Versatile and Accurate Monocular SLAM System[7]

This paper presented ORB-SLAM, a feature-based monocular SLAM system that operates in real time, in small and large, indoor and outdoor environments. The system is robust to severe motion clutter, allows wide baseline loop closing and re-localization, and includes full automatic initialization. It also presented a novel system about SLAM tasks in this paper: tracking, mapping, re-localization, and loop closure. ORB-SLAM achieves unprecedented performance with respect to other state-of-the-art monocular SLAM approaches.

Dense visual slam for RGB-D cameras[8] In this paper, a dense visual SLAM (DVO-SLAM) method for RGB-D cameras was proposed. The DVO-SLAM by Kerl et al. optimizes a pose-graph where keyframe-to-keyframe constraints are computed from a visual odometry that minimizes both photometric and depth error. And for keyframe selection and loop closure detection, the paper presented an entropy-based similarity measure. The evaluation about the approach extensively on publicly available benchmark datasets, and it performs well in scenes with low texture as well as low structure, which yields a significantly lower trajectory error.

B. Details of Some Related Works

1) Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots: [9]

In this paper the author described an autonomous rescue robot system which actually used in RoboCup Rescue German 2014 competition. They determined the situation and task which need to be done in rescue. And separate them into different parts, including SLAM, Planning and Exploration, Simulation, Detection and Manipulation. They have a ROS package *robocup_rescue*, which including some sub-package like *hector_slam*, *hector_exploration_planner*,*hector_vision*. These paper and packages can be a good reference for our RoboCup Rescue competition.

2) ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras: [10]: It presented a complete SLAM system ORB-SLAM(2), which is a real-time SLAM library for Monocular, Stereo and RGB-D cameras that computes the camera trajectory and a sparse 3D reconstruction.It is able to detect loops and re-localize the camera in real time on standard CPUs. The system works in a wide variety of environments (indoor and outdoor environments). The results on the trajectory estimation with metric scale are accurate. And the system includes a lightweight localization mode that leverages visual odometry tracks for unmapped regions and matches to map points that allow for zero-drift localization.

We tried the RGB-D part of ORB-SLAM2. ORB-SLAM2 processes RGB-D inputs to estimate camera trajectory and build a local map of the environment.

The ORB-SLAM2 system has 3 main parallel threads

- Tracking: There are 3 methods to track. First one is tracking with the motion model, which uses the information of the last frames to get the information of rotation an translation. Second one is tracking with the last key frame, similarly, which uses the pose of the last key frame. Third one is tracking with the local map. It uses the many map points to constraint the current frame. Then tracking guides matching of points that have been tracked in last frame. Finally, it optimizes the pose with motion-only bundle adjustment.

- Local mapping: The system processes new keyframe with computing "BoW" representation for efficient matching, connecting the current keyframe with the map points in the local map, and updating covisbility graph and spanning tree. Then, it culls the bad map points , creates and merges new map points using epipolar constraint. At last, optimize the local map performing local bundle adjustment, and cull the redundant keyframes.
- Loop closing: There are 2 steps in this thread: loop detection and loop correction. In loop detection, it computed similarity between query frame and immediate neighbours in the covisibility graph, retain the lowest similarity, and find the alternative frames through the consistency check. In loop correction, it stops the insertions of key frames, corrects them by preforming a pose-graph optimization.

We also tried DVO-SLAM in this project. In contrast to sparse, feature-based methods, this allows us to better exploit the available information in the image data which leads to higher pose accuracy.

In the RoboCup Rescue competition, the environment is easier than real disaster situation, minimize re-projection error can be faster. And we can get all keyframes after a full bundle adjustment in time, and get an accurate 3D model of the whole scene.

3) 3-D Mapping With an RGB-D Camera: [11]

In this paper, it extracts visual keypoints from the color images and uses the depth images to localize them in 3D, using RANSAC to estimate the transformations between associated keypoints and optimizing the pose graph using non-linear optimization. This paper generates a volumetric 3D map of the environment. The 3D map can be used for many tasks like robot localization, navigation, and path planning. It model the environtment using a beam-based environment measurement model To improve the reliability of the transformation estimates which can evaluate the quality of a frame-to-frame estimate. It can robustly deal with highly challenging scenarios by rejecting highly inaccurate estimates.This paper released all source code and they are easy to use.

4) IMU aided RGB_D SLAM: [12]

SLAM in complex indoor environment is quite challenging, but we cannot get very decent 3D map only using visual information. In this paper, it combines RGB-D information with inertial information in a loosely-coupled framework for SLAM which can be run in real-time. Since it utilizes inertial information, it can handle degenerate cases in RGB-D image and texture-less/planar environmental ambiguities. This IMU aided RGB-D navigation system can obtain consistent metric maps in an on-line probabilistic framework.

The inertial sensor is used as predictor in the EKF and feature observations from RGB-D camera is used in the EKF update. The pose estimation is based on RGB-D approach which used SURF descriptors to observe features. A map of 3D features is maintained to reduce the effect of drift. And the paper uses a ring buffer to maintain with constant memory limit and behave as a short term memory.

For pose estimation part, since RGB-D may fail meeting featureless or unstructured environment and inertial sensor, the paper use inertial sensor to help predicting the state. And it just uses RGB-D innovation constraints for EKF-update step then pose estimation of RGB-D are available. To handle measurement delay due to computation cost of vision processing, the paper maintain a ring buffer of past states of EKF prediction with time information.

For mapping part, This paper uses a method based on pose graph optimization. The optimization is carried out on the pose graph until convergence and the EKF state vector is updated with the optimized pose vector.

In conclusion, this system has real-time front-end and off-line back-end for ego-motion and map estimation. The result produced by this paper has promising results when evaluating against accurate ground truth. And the approach described in this paper can be generalized to more cases to combine more sensors together.

5) Long-term online multi-session graph-based SPLAM with memory management: [13]

This is the newest paper of RTAB map (Real-Time Appearance-Based Mapping). The RTAB map is designed to solve online loop closure detection problem. Since the limitation of real-time situation, loop closure detection can only use limited number of locations. And the algorithm should be able to access to all locations when they are needed. When the number of point to match has reached limitation of computation, RTAB map will move unlikely-loop-closing locations from Working Memory to Long Term Memory which means that these locations will not participate in next loop closure detection.

Since locations in Long Term Memory will not participate in loop closure detection, it is important to choose proper locations to move from Working Memory to Long Term Memory. RTAB map will choose low-weight locations that are less-likely be visited by loop closure detection to move into Long Term Memory.

Other than Long Term Memory, there is another memory called Short Term Memory. Short Term Memory is used to observe temporal similarity between consecutive images to update weights of their map locations. Working Memory is used to loop closure detection. When the amount of locations stored in Short Term Memory reached the limitation, RTAB map will move locations which stored for longest time in Short Term Memory to Working Memory.

RTAB map use Bayesian filter method to estimate the probabilities of loop closure, and compare new locations to the locations stored in Working Memory. A loop is detected when there is a high probability that new locations and old locations can close a loop. The RTAB map has some ways to keep the amount of locations stored in Working Memory can be handled by Bayesian filter. Identifies locations that should remain in a Working Memory for online processing from locations that should be transferred to a Long Term Memory. When revisiting previously mapped areas that are in Long Term Memory, the mechanism can retrieve these locations and place them back in Working Memory.

C. Related ROS Package

1) ROS package: iai_kinect2*:* The package provides a driver to receive data from the Kinect 2 sensor. It will provide point-clouds, depth-clouds and images for us, and we can use these data to do our 3D SLAM. It basicly contain a bridge to connect the libfreenect2 driver and ROS. It also includes a calibration tool to calibrate the IR sensor, RGB sensor and the depth. It also includes a library for depth registration which gives an registered depth image.

2) ROS Package:rgbdslam: rgbdslam (v2) is a SLAM solution for RGB-D cameras. It provides the current pose of the camera and allows to create a registered point cloud or an octomap. It features a GUI interface for easy usage, but can also be controlled by ROS service calls, e.g., when running on a robot.

RGB-D SLAM has many options to customize its behavior. You can speed it up significantly by

- enabling concurrency on multi-core machines. The three corresponding parameter names start with "concurrent ..."
- using SIFTGPU features, if you have a GPU, ORB otherwise
- decreasing the "max keypoints" parameters
- setting the kinect driver to QVGA (There's a bug in the camera info of the driver. See/use the launchfile qvga-kinect+rgbdslam.launch for a workaround)

However, the last three options and using ORB will probably decrease the accuracy

3) ROS Package: orb_slam_2_ros*:* *orb_slam_2_ros* package integrates ORB_SLAM2 into ROS in a more user friendly way. The package has been tested in Ubuntu 12.04, 14.04 and 16.04, but it should be easy to compile in other platforms.

With the help of this ROS package(*orb_slam_2_ros*), we can process the live input of a monocular, stereo or RGB-D camera using ROS(a version Hydro or newer is needed) by ORB_SLAM2, and in this project, we use the RGB-D Example to process RGB-D input.

For an RGB-D input from topics /camera/rgb/image_raw and /camera/depth_registered/image_raw, run node *ORB_SLAM2/RGBD*. We will need to provide the vocabulary file and a settings file.

4) ROS Package: rtabmap_ros*:* *rtabmap_ros* package integrates rtabmap into ROS. This package is a ROS wrapper of RTAB-Map (Real-Time Appearance-Based Mapping) which is a RGB-D SLAM approach based on a global loop closure detector with real-time constraints. This package can be used to generate a 3D point clouds of the environment and/or to create a 2D occupancy grid map for navigation.

5) ROS Package: hector_slam*:* [14] *hector_slam* package contains 2D mapping package based on matching method and exploration package which can make the robot navigate in 2D space knowing 2D map which can be provided by Hector SLAM itself or cost mapping produced by other algorithms.

hector_slam provide the building blocks for a system capable of exploration in USAR environments. In RoboCup Rescue and other applications, these modules have been proofed to be efficient and have good performance. These modules are made both by Team Hector (Heterogeneous Cooperating Team of Robots) of TU Darmstadt and numerous other international research groups.

III. System Description

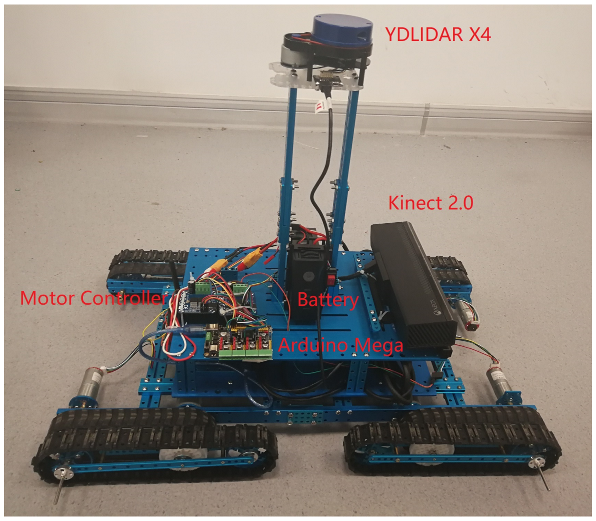


Fig. 2. The overview of our rescue

A. Hardware

1) The Main Computer and Motor control: We use an mini PC of Intel i7-7500u as the main computer which needs low power and satisfy the computation demand of using Kinect 2.0. We use the Arduino Mega and DC motor driver to drive the motor. The Arduino Mega is connected to the mini PC and communicate with the ROS by rosserial package,and use *diff_drive_controller* package to control the movement of robot.

2) Kinect 2.0 Sensor and Calibration: We put Kinect 2.0 Sensor in front of the robot. The RGB camera has 1920x1080 resolution, the depth camera has 640x480 resolution, the FOV is 70x60, the detection range is 0.5m-5m. We use the calibration tool to calibrate Kinect 2.0. This tool use the chessboard patterns to calibrate all the camera including RGB camera, IR camera and depth. We captured a lot of pictures to calibrate the camera about 100 pictures for each camera. After that we get some camera matrices including rotation, translation, essential and fundamental matrix. The depth quality changed a lot after calibration which is illustrated as below. The Kinect 2.0 is used for 3D mapping and serves as the main camera for human interaction in teleoperation control mode.

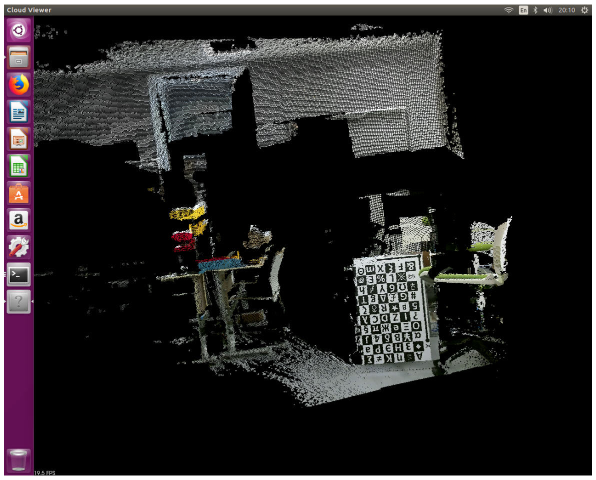


Fig. 3. Before the calibration

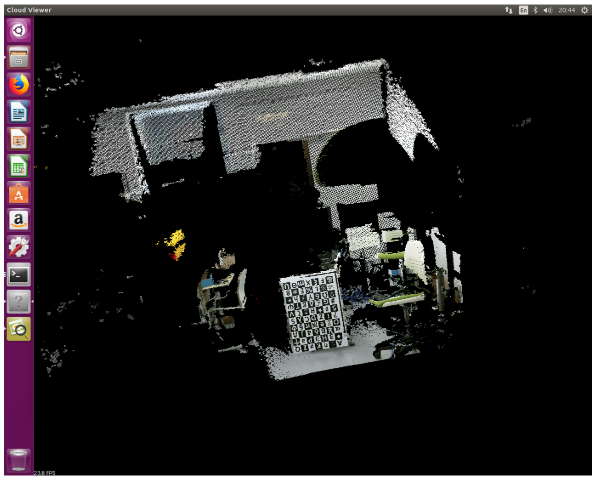


Fig. 4. After the calibration

3) LIDAR and other sensors: We mount a YDLIDAR X4 on the top of our robot. This LIDAR is really cheap and is highly competitive comparing with other expensive LIDAR. The sample rate is 5000 samples per second, the original spinning frequency is 7Hz which we hacked the electronic board to raise it up to 12 Hz, the scanning distance is 0.12m-10m. The LIDAR is used for 2D mapping and improve 3D mapping by combine 2D SLAM and visual slam together.

4) Multi sensor calibration and tf relation: To calibrate the relative location between Kinect 2.0 and 2D laser Lidar . We use the matlab toolbox for kinect calibration.[15] After calibration, we get the relative position between two sensors, and use the vertical point as the base_link and create the tf relationship between them.

B. ROS-based Software

Our software is developed based on Robot Operation System(ROS) which is very popular software architecture. We use a lot of ROS package to finish the project. The pipeline is discribed in the figure.
Firstly, we use *iai_kinect2* and calibrated camera yaml to get the rgb image and depth image from Kinect 2.0. As the compressed image and 1920*1080 resolution need huge amount of computation, which cause the slam is very slow. As performance limit, we choose 960*540 resolution raw image, which can let us get an process rate at about 20 Hz. We also collect the laser scan data from YDLidar X4. While get the sensor data, we also publish tf of the sensors: *laser_link* and *camera_link*, according to the calibrated relative location. Then we use *hector_slam* to get the scanmatch odometry. As we didn't finish the hardware motor encoder now, we have to use the scanmatch odometry for the *rtabmap_ros* in the next step.
In the *rtabmap_ros*, which is the main part of the SLAM, we use combined 2D and 3D SLAM, so we can get the pointcloud, the 3D projection map, the 2D map and the SLAM odometry. We use the rtabmapviz the visualize the map and the point cloud. After that we transform the 3D projection map to costmap to avoid obstacle in 3D. And we use the *hector_exploration_plannar* to plan the exploration path. Use the *hector_exploration_controller* the convert the path to robot velocity command *cmd_vel*. Use the *rosserial_python* to send the twist to arduino and use PWM signal and Motor driver to control the motor.

C. 2D/3D SLAM

SLAM (Simultaneous Localization and Mapping) is an important technique to get location of robots and build a map of the unknown environment. SLAM algorithms working in 2D place are called 2D SLAM which will build 2D map. We use 2D laser LIDAR as the sensor for 2D SLAM. We apply Hector SLAM[9] as our 2D SLAM approach since it is a method based on robust scan matching. Comparing method like Gmapping[16] which is based on odometry and scan matching, Hector SLAM is more suitable since odometry is unreliable in rescue scene. We have tested Hector SLAM and it performed very well on flat environment.

Besides 2D map, we also provide 3D map of the unseen environment. SLAM algorithms working in 3D space are called 3D SLAM. We use Kinect 2.0 as the sensor for our 3D SLAM. We have tried several 3D SLAM algorithm including RGB-D SLAM 10, ORB SLAM 9 and RTAB SLAM 8. And at last we choose RTAB SLAM as our 3D SLAM algorithm. After testing for several times, we realized that the 3D map built only by 3D SLAM is not accurate enough, the angle between walls will have some errors caused by drift and miss matching which often happens in rescue scene which is lack of feature.In the figure, the RGB-D SLAM get lost after some times. The ORBSLAM2 didn't matches well and drift a lot. The rtabmap use only rgbd camera is a bit better than last two but also drift near the exit.

At last, we decide to hybrid 2D SLAM and 3D SLAM together since we found that the 2D map built by Hector SLAM is good enough to guide 3D SLAM algorithm. We mounted 2D laser LIDAR on the top of our robot to avoid the influence of obstacles which will get a 2D map that is exactly the 2D projection of walls in 3D map. Hector SLAM estimate the odometry by ICP, and 3D SLAM estimate the

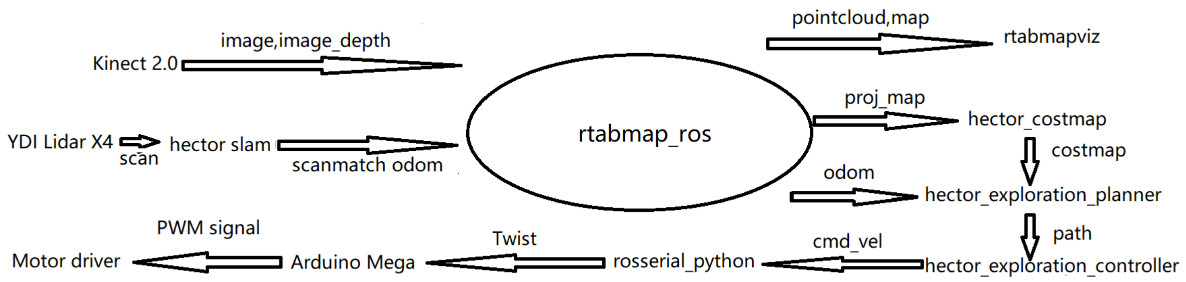


Fig. 5. ROS pipeline

odometry by visual approach. We use the odometry which is estimated by combining the information of 2D odometry and 3D odometry to guide 3D SLAM which gives us really good mapping result. 12 13

We overcame four problems when we were working on 2D/3D SLAM. Firstly, the difficulties to understand the details of different SLAM algorithms is pretty big. We spent a lot of time to study these SLAM algorithms. Secondly, the dataset is quite hard to collect. We can only collect one ROS bag each time due to limitation of disk storage and limitation of battery. Collecting dataset of LIDAR and RGBD running in the labyrinth is time consuming. Thirdly, the electronics parts of our robot is complex and cost us a lot of time to fix it after experiments. At last, the parameters of SLAM algorithm need to be fine-tuned to get good performance. In the figure, you can find that the wall aligned well and nearly no drift

D. Navigation

We achieved navigation using ROS Package *hector_navigation*. Using the 2D projection map of 3D mapping produced by SLAM algorithm, the navigation algorithm first build cost map for the exploration task, then *hector_exploration_planner* which is based on exploration transform approach presented in [17] will start navigation. 14

To prevent a lot of frequent turning of robot, the exploration planner prefers to weight frontiers that are towards the front of the robot. The planner using follow wall strategy to generate trajectory. In case the situation that the environment has been explored totally, the planner has a inner exploration mode. The hector trajectory server node will store traversed path of the robot, which will be retrieved. A list of position which are sampled based on distance will pass to exploration transform algorithm as a list of goal. The point which has the highest value of exploration transform cell is safe to reach for the robot.

IV. System Evaluation

For the evaluation, we want to use the method in this paper[18], which fit the situation of Robocup Rescue. However, since we didn't have the ground truth, for evaluation, we compare it with the origin rtabmap. 8 We can find that if just using the rtabmap, the map will be very bad, the same wall overlap many times and is very thick. And at the last the map is mismatch. However, in combined slam, the map is far more better. Most walls are straight and no mismatch. 12 So we can find that for both 2D map and 3D map, the combined slam far more better than origin.

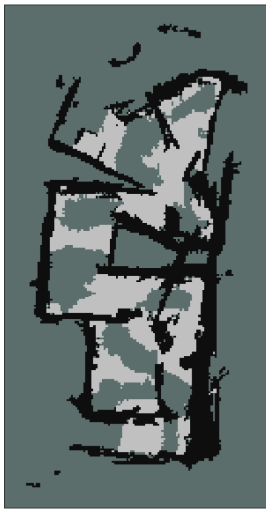


Fig. 6. rtabmap only projection map

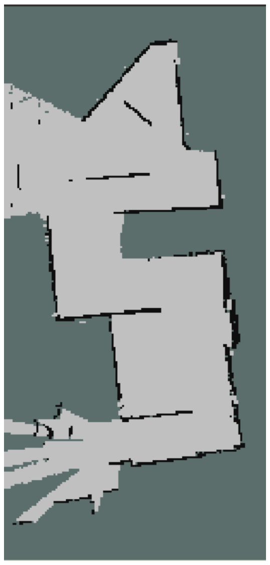


Fig. 7. lidar rtabmap combine projection map

V. Conclusions

We designed and implemented the perception system and navigation system of our RoboCup Rescue Robot. The rescue robot can provide high quality 2D and 3D map and finish simple navigation task. Furthermore, we will implement object recognition, increase the speed of robot to build the map without lost of tracking and try to avoid the influence of pure rotation which can cause drift.

References

[1] G. Guidi, S. Gonizzi, and L. Micoli, "3d capturing performances of low-cost range sensors for mass-market applications." *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 41, 2016.

[2] B. Karan, "Calibration of kinect-type rgb-d sensors for robotic applications," *Fme Transactions*, vol. 43, no. 1, pp. 47–54, 2015.

[3] R. Sheh, A. Jacoff, A.-M. Virts, T. Kimura, J. Pellenz, S. Schwertfeger, and J. Suthakorn, "Advancing the state of urban search and rescue robotics through the robocuprescue robot league competition," in *Field and service robotics*. Springer, 2014, pp. 127–142.

[4] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," *CoRR*, vol. abs/1704.07813, 2017. [Online]. Available: http://arxiv.org/abs/1704.07813

[5] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 1–10.

[6] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Conference on Computer Graphics and Interactive Techniques*, 1996, pp. 303–312.

[7] M. J. M. M. Mur-Artal, Raúl and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[8] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013, pp. 2100–2106.

[9] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, and O. V. Stryk, "Hector open source modules for autonomous mapping and navigation with rescue robots," *Lecture Notes in Computer Science*, vol. 8371, pp. 624–631, 2013.

[10] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[11] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2017.

[12] U. Qayyum, Q. Ahsan, and Z. Mahmood, "Imu aided rgb-d slam," in *Applied Sciences and Technology (IBCAST), 2017 14th International Bhurban Conference on*. IEEE, 2017, pp. 337–341.

[13] M. Labbé and F. Michaud, "Long-term online multi-session graph-based splam with memory management," *Autonomous Robots*, pp. 1–18, 2017.

[14] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[15] J.-C. Devaux, H. Hadj-Abdelkader, and E. Colle, "A multi-sensor calibration toolbox for Kinect : Application to Kinect and laser range finder fusion." in *16th International Conference on Advanced Robotics (ICAR 2013)*, Montevideo, Uruguay, Nov. 2013, p. (to appear). [Online]. Available: https://hal.archives-ouvertes.fr/hal-00913178

[16] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[17] S. Wirth and J. Pellenz, "Exploration transform: A stable exploring algorithm for robots in rescue environments," in *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*. IEEE, 2007, pp. 1–5.

[18] S. Schwertfeger and A. Birk, "Map evaluation using matched topology graphs," *Autonomous Robots*, vol. 40, no. 5, pp. 761–787, 2016.

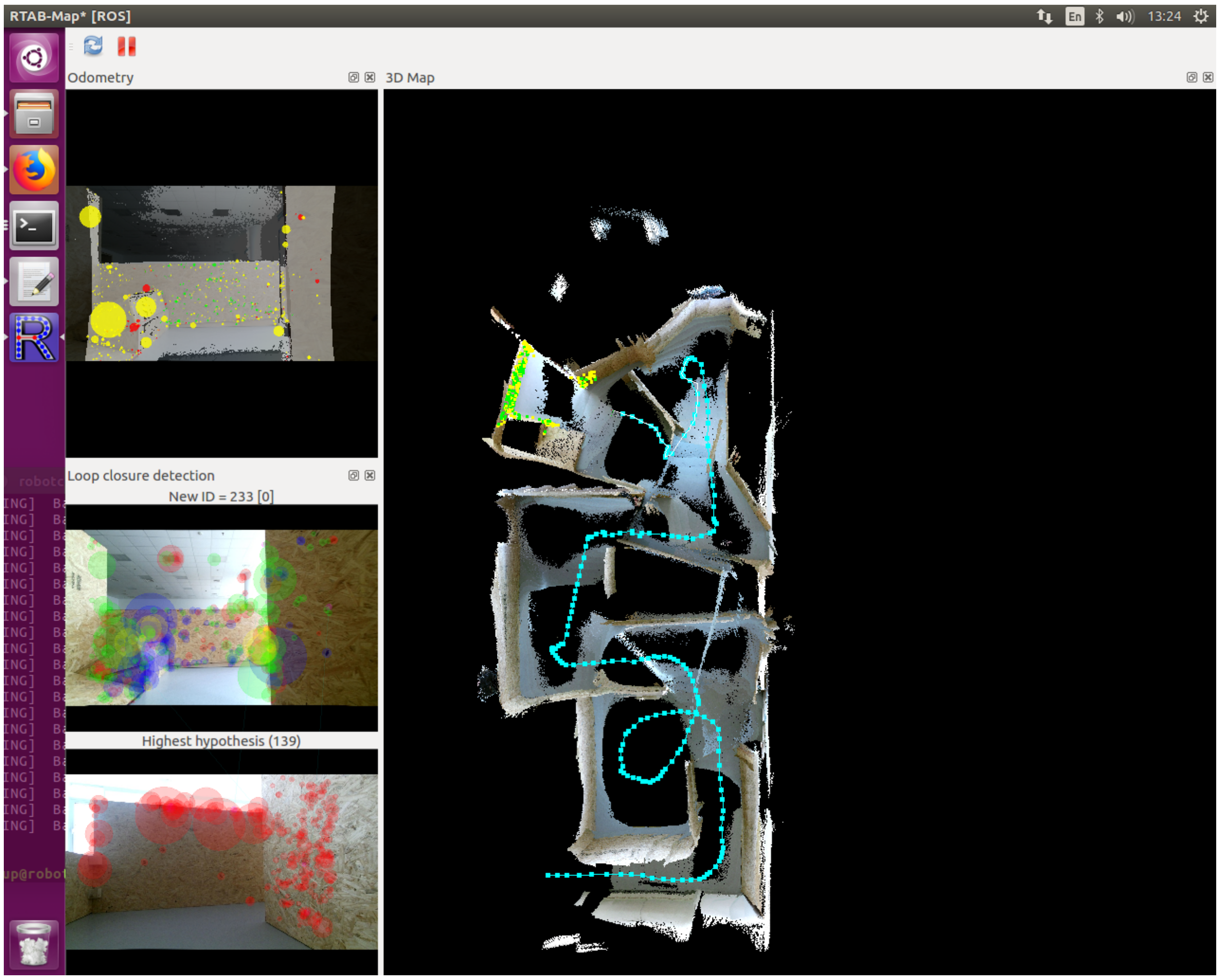
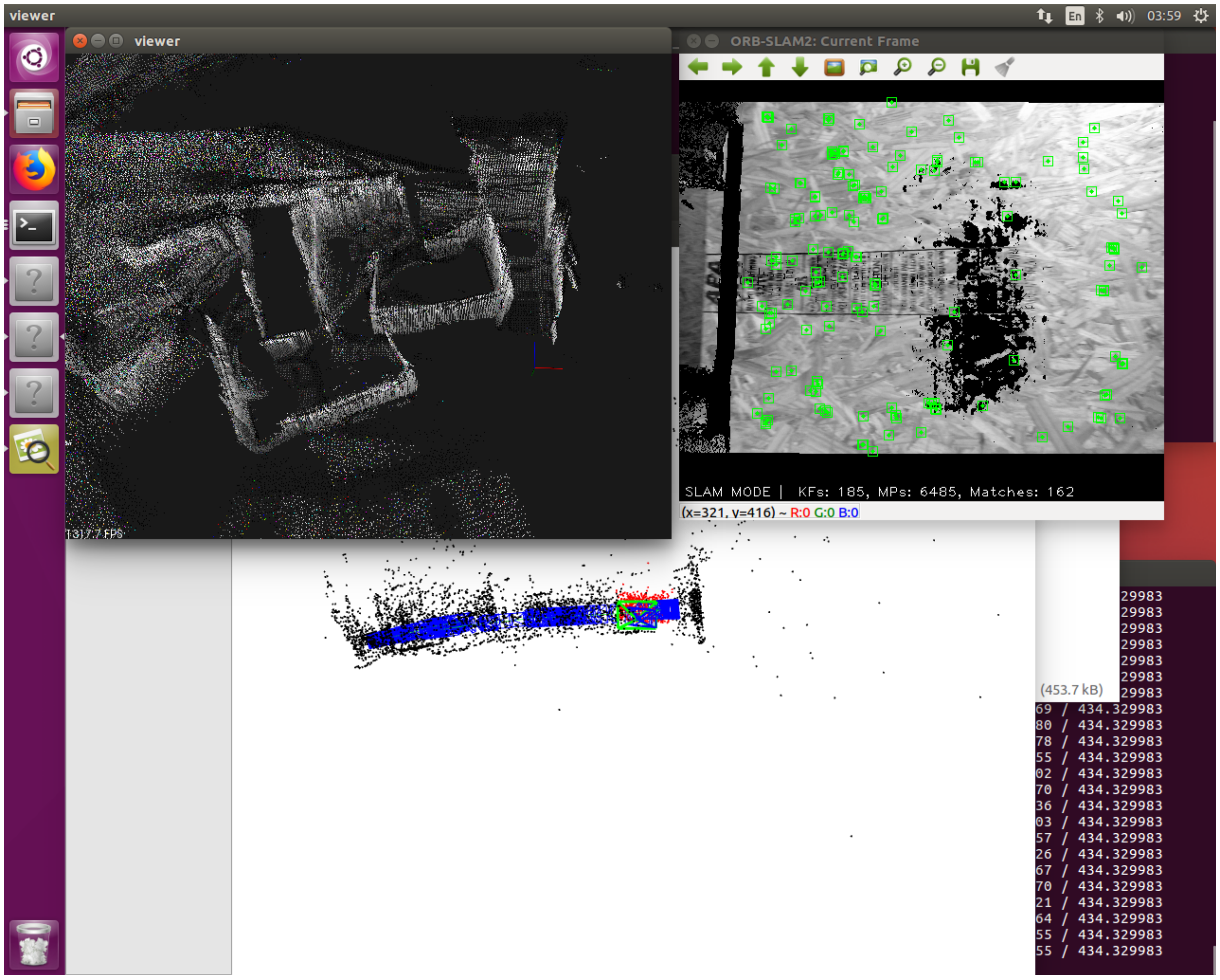


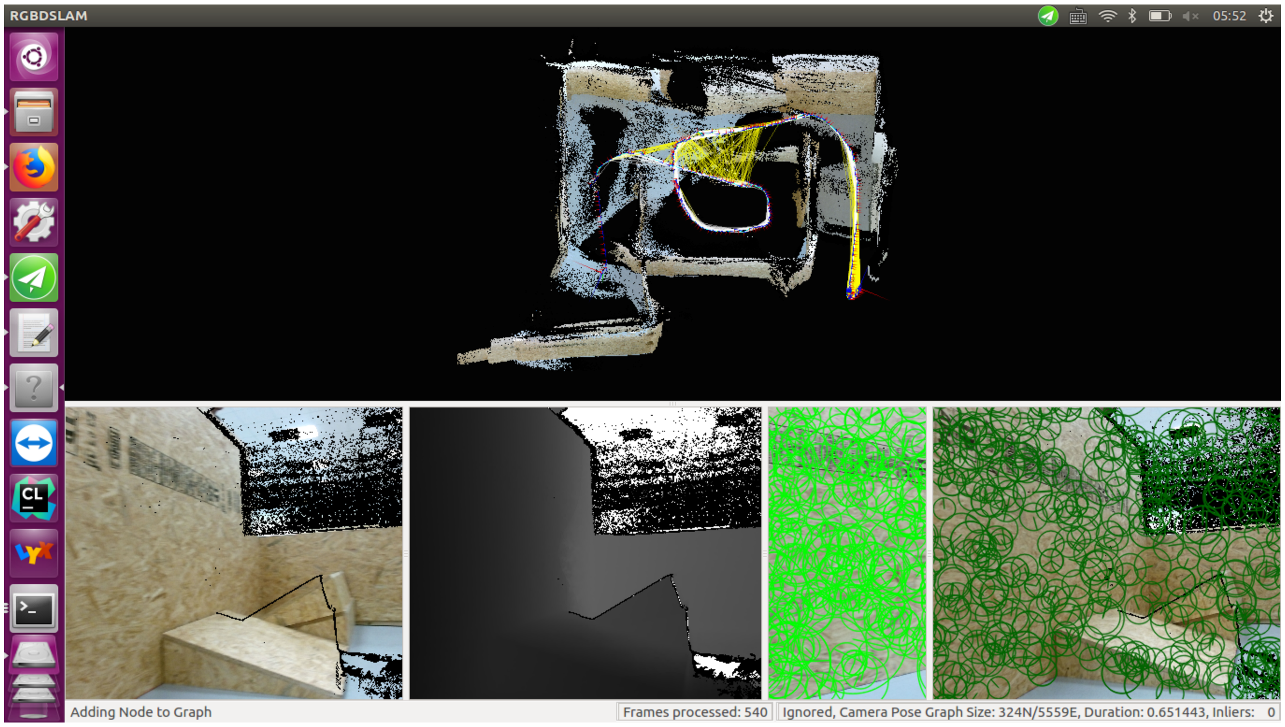
Fig. 8.   Failure case of 3D SLAM



```
SLAM MODE |  KFs: 185, MPs: 6485, Matches: 162
(x=321, y=416) ~ R:0 G:0 B:0
```

```
29983
29983
29983
29983
29983
29983
29983
69 / 434.329983
80 / 434.329983
78 / 434.329983
55 / 434.329983
02 / 434.329983
70 / 434.329983
36 / 434.329983
03 / 434.329983
57 / 434.329983
26 / 434.329983
67 / 434.329983
70 / 434.329983
21 / 434.329983
64 / 434.329983
55 / 434.329983
55 / 434.329983
```

Fig. 9. Result of ORB_SLAM

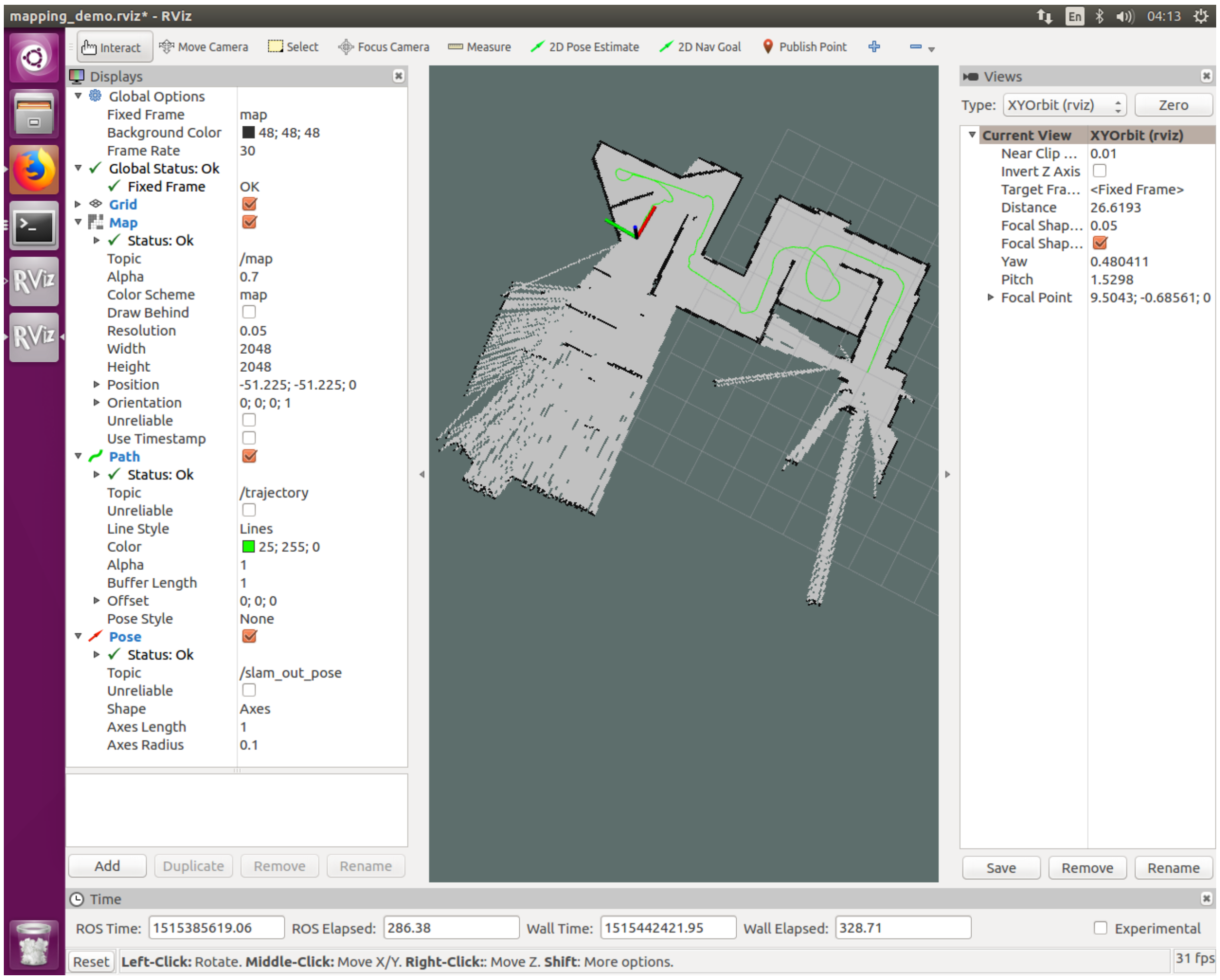


Adding Node to Graph

Frames processed: 540

Ignored, Camera Pose Graph Size: 324N/5559E, Duration: 0.651443, Inliers: 0

Fig. 10. Result of RGBD_SLAM



Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	☑
Map	☑
✓ Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	☐
Resolution	0.05
Width	2048
Height	2048
Position	-51.225; -51.225; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐
Path	☑
✓ Status: Ok	
Topic	/trajectory
Unreliable	☐
Line Style	Lines
Color	🟩 25; 255; 0
Alpha	1
Buffer Length	1
Offset	0; 0; 0
Pose Style	None
Pose	☑
✓ Status: Ok	
Topic	/slam_out_pose
Unreliable	☐
Shape	Axes
Axes Length	1
Axes Radius	0.1

Type: XYOrbit (rviz)

Current View	XYOrbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	26.6193
Focal Shap...	0.05
Focal Shap...	☑
Yaw	0.480411
Pitch	1.5298
Focal Point	9.5043; -0.68561; 0

Time

ROS Time: 1515385619.06 ROS Elapsed: 286.38 Wall Time: 1515442421.95 Wall Elapsed: 328.71 ☐ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Move Z. **Shift**: More options.

Fig. 11. Result of 2D SLAM

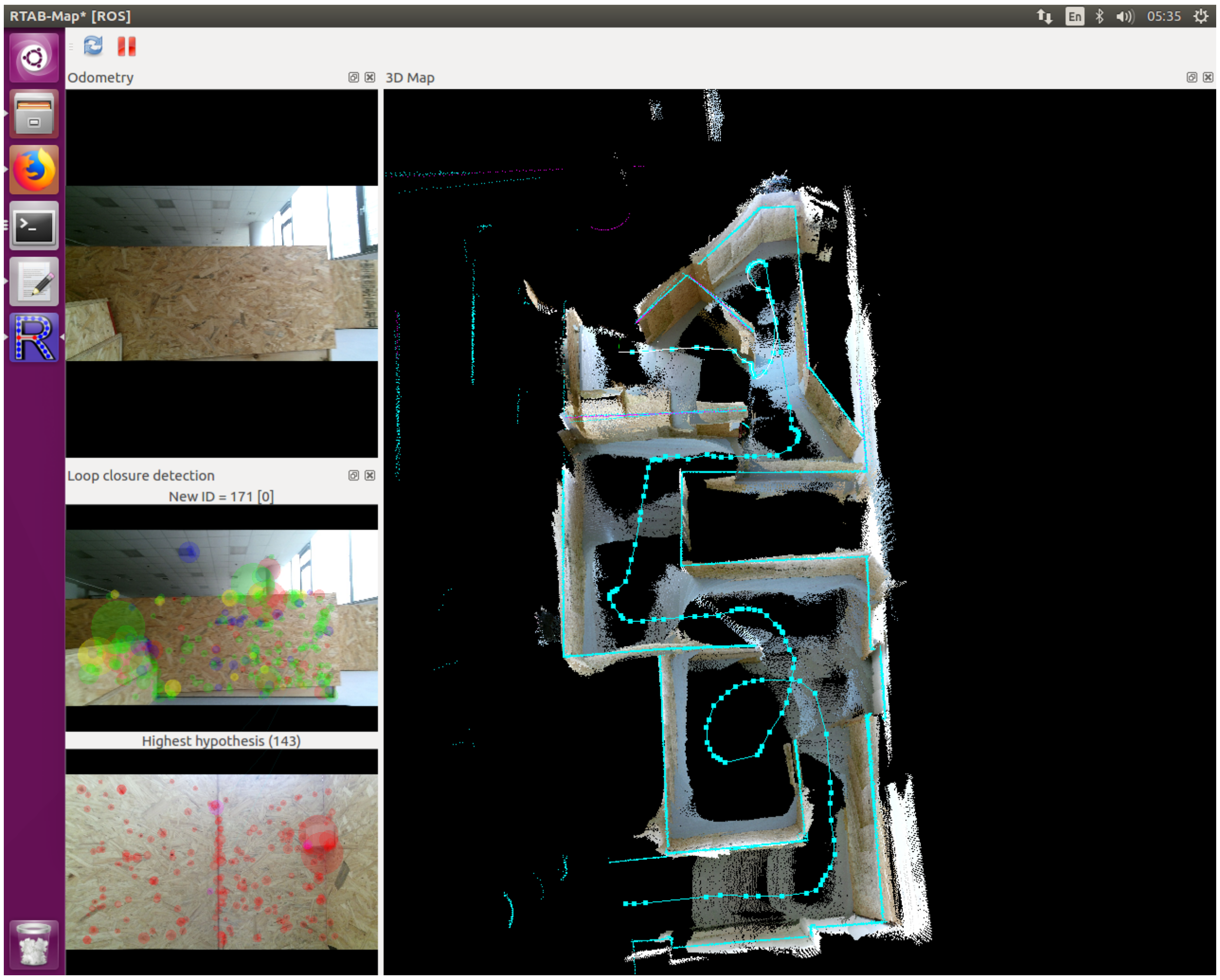


Fig. 12. Result of 2D/3D SLAM

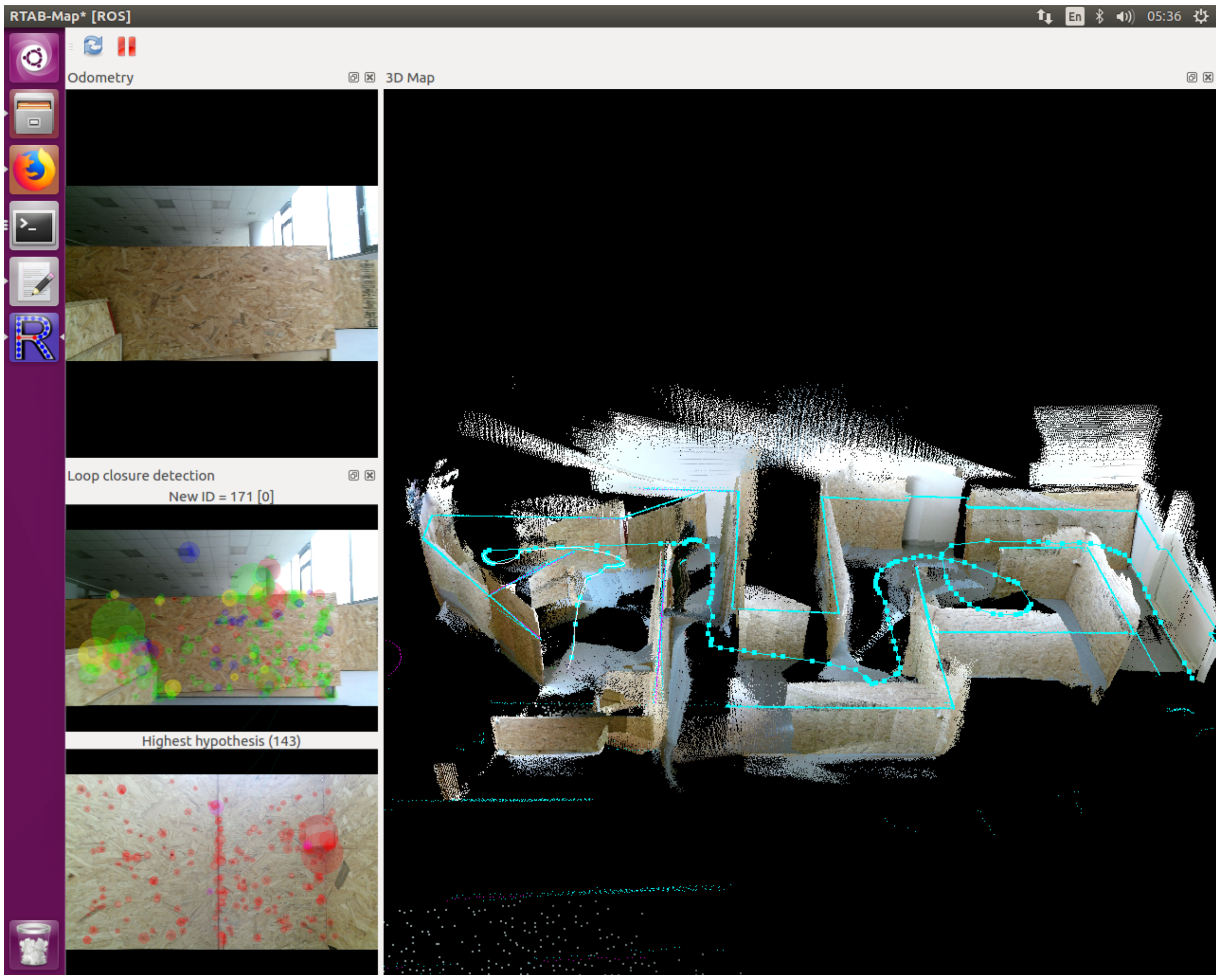
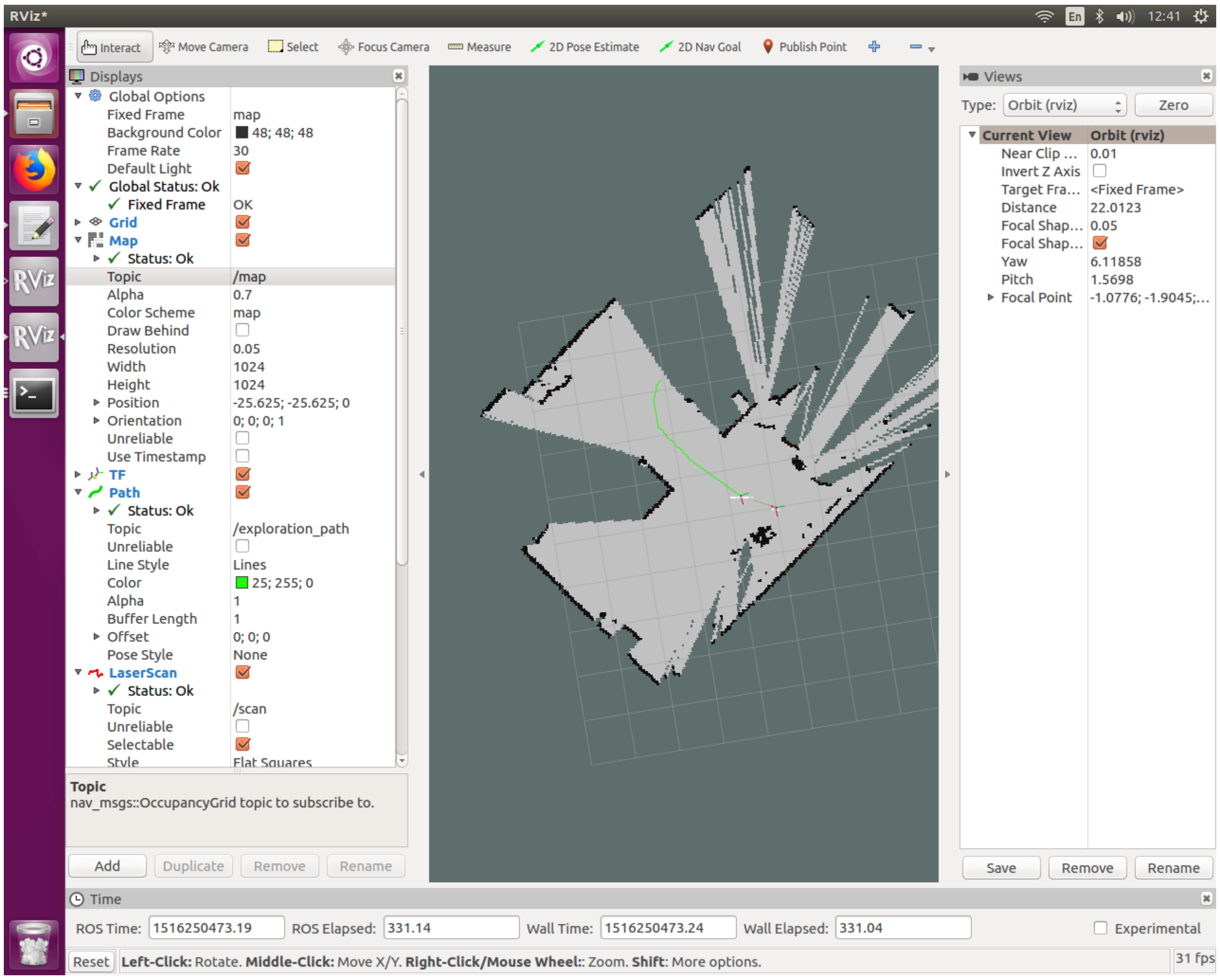


Fig. 13. Result of 2D/3D SLAM



Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
Map	✓
✓ Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	☐
Resolution	0.05
Width	1024
Height	1024
Position	-25.625; -25.625; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐
TF	✓
Path	✓
✓ Status: Ok	
Topic	/exploration_path
Unreliable	☐
Line Style	Lines
Color	🟩 25; 255; 0
Alpha	1
Buffer Length	1
Offset	0; 0; 0
Pose Style	None
LaserScan	✓
✓ Status: Ok	
Topic	/scan
Unreliable	☐
Selectable	✓
Style	Flat Squares

Topic
nav_msgs::OccupancyGrid topic to subscribe to.

Type: Orbit (rviz) Zero

Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	22.0123
Focal Shap...	0.05
Focal Shap...	✓
Yaw	6.11858
Pitch	1.5698
Focal Point	-1.0776; -1.9045;...

ROS Time: 1516250473.19 ROS Elapsed: 331.14 Wall Time: 1516250473.24 Wall Elapsed: 331.04 ☐ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

Fig. 14. Result of Navigation