SLAM using RTAB-Map

Christian Rabus

Abstract—This report introduces RTAB-Map (Real-Time Appearance-Based Mapping) which is a RGB-D, Stereo and Lidar Graph-Based SLAM approach based on an incremental appearance-based loop closure detector. An existing robot model will be extended by a Kinect Camera and will perform SLAM in two different environments using the MTAB-Map approach. The user manually controls the robot using the keyboard (teleop integration) in one of the two gazebo world environments. One simulated environment is provided, the other is freely selectable and is individually created by the user in gazebo. The SLAM algorithm will create a 2D occupancy grid and 3D octomap from both simulated environments.

Index Terms—SLAM, RTAB-Map, Localization, Mapping, loop closure, RGB-D, Laser, Kinect, Gazeboo, RViz, ROS, URDF.

1 Introduction

This report focuses on *Simultaneous Localization and Mapping* (SLAM) performed by a remotely controlled (or teleoperated) robot. SLAM has to be used, when neither a map of the environment nor a pose of the robot is available. By using the robots movement and sensor data, simultaneously a map of the environment is created and the pose of the robot in it is localized.
There are different solutions for SLAM - in this report the *RTAB-Map* approach is used and is discussed in more detail. Briefly, RTAB-Map is a RGB-D, Stereo and Lidar Graph-Based SLAM approach based on an incremental appearance-based loop closure detector [1]. Furthermore, other SLAM approaches are discussed in the next section.
In the following project, an existing robot model will be equipped with an RGB-D camera (Microsoft Kinect like camera), a 2D Laser rangefinder sensor (Hokuyo laser range finder) and provides data about its odometry. In order to move the robot in *gazebo*, it is remotely controlled (teleoperated) via keyboard control. The robot receives velocity commands that are split and sent on the two wheels of a differential drive wheel system. Odometry is computed from the feedback from the hardware, and published [2]. To use the RTAB-Map approach for SLAM, the *RTAB-Map ROS wrapper* is used and interfaced with the published information (odometry and sensor data) provided by the robot. The SLAM algorithm will create a 2D occupancy grid and 3D octomap from two simulated environments. The two simulated environments are the provided **kitchen_dining.world** environment and the other **office.world** environment created by the student. During mapping the tool *rtabmapviz* is used for real time visualization of feature mapping, loop closures, and more. After generating the maps (2D and 3D) of each environment, the data is stored in a mapping database. Using the *rtabmap-databaseViewer* tool allows for complete analysis (check for loop closures, generate 3D maps for viewing, extract images, check feature mapping roch zones, etc.) of the mapping. The results of mapping both environments are presented in a later section.

2 Background

As already briefly explained in the introduction, SLAM stands for *Simultaneous Localization and Mapping*. There, a robot must create a map of an unknown environment and at the same time locate itself relative to that map. It combines the challenges of Localization and Mapping and is therefore more difficult than the two tasks separately, since neither the map nor the robots poses are provided. Due to the noisy motions and sensor measurements of the robot, the generated map and poses of the robot are uncertain, resulting in error estimates for following robot positions and mapping. The risk of increased error propagation is therefore very high, which may lead to poor results. The accuracy of the map depends on the accuracy of the localization and vice versa. SLAM is also referred to as a chicken or egg problem because the map is required for localization and the robot's pose for mapping. SLAM is therefore a fundamental challenge for mobile robots.
In general, the SLAM problem can be divided into two key features, *form* and *nature*. The *form* addresses how the posterior is estimated. There are two types of *form*:

- **Online SLAM** problem: Robot estimates its *current pose* and the *map* using current measurements and controls.
- **Full SLAM** problem: Robot estimates its *entire trajectory* and the *map* using all the measurements and controls.

The second key feature of the SLAM problem is related to its *nature* and consists of two elements:

- **Continuous**: Robot continuously estimates its *pose* and senses the environment to estimate the *location of objects* or landmarks.
- **Discrete**: During SLAM, the Robot also has to identify any relations between newly detected objects and previously detected ones. It has to check each moment, if it has been at this location before, which leads to a discrete decision: yes or no. This is also known by *correspondences*.

To solve these challenges, several SLAM algorithms exist and can be divided into 5 categories [3]:

- **Extended Kalman Filter SLAM (EKF)**
- **Extended Information Filter (EIF)**
- **Sparse Extended Information Filter (SEIF)**
- **FastSLAM**
- **GraphSLAM**

Extended Kalman Filter SLAM (EKF) utilizes the Extended Kalman Filter which is typically feature based, and use the maximum likelihood algorithm for data association. EKF SLAM was considered the state of the art until it was replaced by FastSLAM [4].
Extended Information Filter (EIF) is the dual of the EKF. The key difference to the EKF is the way the Gaussian belief is represented. The EIF represent Gaussians in their canonical parameterization, which is comprised of an information matrix and an information vector [5].
Sparse Extended Information Filter (SEIF) relies on the EIF and is its sparse representation. SEIFs represent maps by graphical networks of features that are locally interconnected, where links represent relative information between pairs of nearby features, as well as information about the robots pose relative to the map [6].
FastSLAM uses a particle filter (MCL) approach to solve SLAM problems. There are three different known algorithms:

- **FastSLAM 1.0**: uses large number of particles and low-dimensional Extended Kalman Filter to solve Online and Full SLAM problem; known landmark positions are required, therefore modelling an arbitrary environment is not possible
- **FastSLAM 2.0**: more efficient than FastSLAM 1.0 due to low number of particles; despite also known landmark positions are required; modelling of arbitrary environment not possible
- **Grid-based FastSLAM**: uses particles for trajectory and pose estimation; instead of low-dimensional Kalman Filter, Occupancy Grid Mapping algorithm is used for map estimation; therefore modelling of arbitrary environment is possible; advantage ofer FastSLAM 1.0 and 2.0, no predefined landmark positions are required

GraphSLAM solves the Full SLAM problem, so it is able to estimate the robot's entire path and map. To do this, it uses *soft* spatial constraints to represent relationships between robot poses (*motion constraint*: between two successive poses) and the environment (*measurement constraint*: constraint between robot pose and a feature). They are called *soft*, because the robot's motion and the measurements are always uncertain. A *feature* can be a landmark or an identifiable element of the environment (e.g. edge or corner). All these constraints are stored in a *sparse* information matrix and information vector. It is called *sparse*, because as most off-diagonal elements of the matrix are zero, as no information between the constraints is (yet) available. The algorithm can be separated in two sections, the *frontend* and *backend*. In the *frontend* the graph is created using odometry and sensor data by by continuously adding nodes (poses and features) and edges (constraints). Due to the uncertainty of the motion and measurement, constraints conflict each other and is leading to errors. In the *backend*, the graph is optimized by minimization of errors in all the constraints by using MLE (Maximum Likelihood Estimation).
Due to its data structure, GraphSLAM is not bound to a finite number of variables and is able to hold thousands of nodes and features. Furthermore, the data sparsity reduces the need for significant onboard processing capability, which makes it suitable for a large environment. It also has an improved accuracy over FastSLAM, because FastSLAM uses particles to estimate poses and most likely does not exist in all positions.
In this project, RTAB-Map is used and is an implementation of GraphSLAM. It fuses sensor data from an RGB-D camera and a LIDAR (laser detection and ranging) and uses odometry to generate a 2D occupancy map, 3D octomap and 3D point cloud. It uses the process of *loop closure* to check if the robot has been visited the location before. The more the robot moves to new areas, the more images have to be compared. Therefore, the loop closure takes longer and the complexity increases linearly. RTAB-Map is optimized for large-scale and long-term SLAM, so that the loop closure can be performed in real-time. It uses the approach of *Bag-of-Words*: SURF (Speeded Up Robust Features) method is used to extract features and their feature descriptors are clustered together to represent the vocabulary. Each feature descriptor can be seen as a word and therefore an image consisting of many words is a *bag-of-words*. Loop closure is now detected when a new image has reached a certain amount of words, and these words have already been found in a previously saved image.
Whether only 2D maps or also 3D maps and point clouds shall be generated during SLAM depends on the later use case. If only 2D maps of the environment are generated during mapping, problems (collisions) could occur. Larger robots that use the same maps for localization, for example, could no longer pass under certain small objects (e.g. chair). Therefore generating 3D maps seems to make sense. However, it should be noted that 3D maps and point clouds not only require more processor capability, but also additional sensors.

3 Robot Model and Scenes

In order to perform SLAM, the existing robot (*udacity_bot*) from the previous project (*Where Am I?*) is used and teleoperated via keyboard control within two simulated environments in Gazebo. The first environment is the provided **kitchen_dining.world**, Fig. 3 and the second one, called **office.world** Fig. 4, is new created with Gazebo.

3.1 Robot Model

Fig. 1 shows the Robot Model (*udacity_bot*) and was already used in the previous project. It consists of a cuboid chassis (blue), two spherical casters (white, bottom side of chassis), two cylindrical wheels (black), a visual for the rgbd camera (red box at frontend) and a visual for the laser range finder (grey/black mesh on top side of chassis).
In addition to the visual design (links, joints), plugins are also implemented so that the robot can drive and sensor data from camera and laser is available. For driving the **Differential Drive Plugin** (libgazebo_ros_diff_drive.so) is

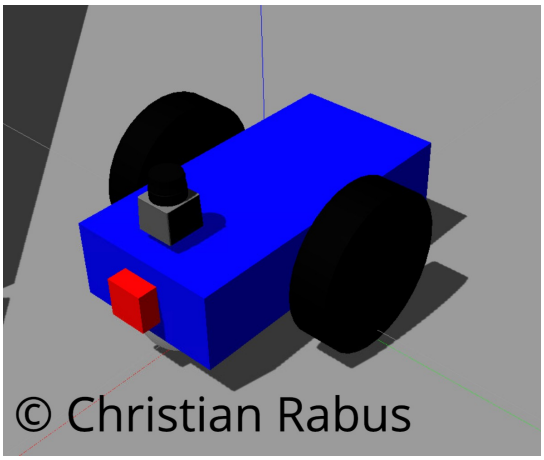

Fig. 1. Robot Model named udacity_bot

used and provides also data about velocity and odometry. The RGB-D Camera is implemented by the **Depth Camera / Openni Kinect Plugin** (libgazebo_ros_openni_kinect.so) and provides rgb-image and depth-image data. Also a LIDAR sensor (Hokuyo) is used and implemented by the **Laser Plugin** (libgazebo_ros_laser.so). It provides horizontal 2D laser scan data.
In order to move the robot in *gazebo*, it is remotely controlled (teleoperated) via keyboard control. The robot receives velocity commands that are split and sent on the two wheels of a differential drive wheel system.
The transformation tree in Fig. 2 shows how the links of the Robot Model are connected.

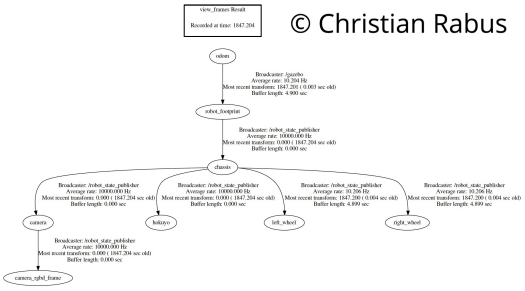

Fig. 2. transformation tree of udacity_bot

3.2 Scenes

There are two simulated environments in which the robot shall run SLAM. The first environment is already provided and is shown in Fig. 3. The second environment, shown in Fig. 4, was new created in Gazebo. At first, the size of the ground plane was reduced to 10x7m and the texture (material) was changed to *Gazebo/WoodFloor*. After that, a building was created with the 'Building Editor'. The textures of the walls were changed accordingly. Different wall textures were deliberately chosen, so that features can be extracted better. The changes were made directly in the file with a standard text editor.
The building has two rooms and one hallway and shall represent an office or storage. Therefore several items have been placed, that can generally be found in such environment: card boards, table, cabinet, euro pallet, trash can, a person, cinder blocks, etc. Also other items (e.g. calibration plane on floor), that normally not belong to an office or storage, have been placed, so that more features can be found during mapping.

Fig. 3. kitchen_dining.world in gazebo

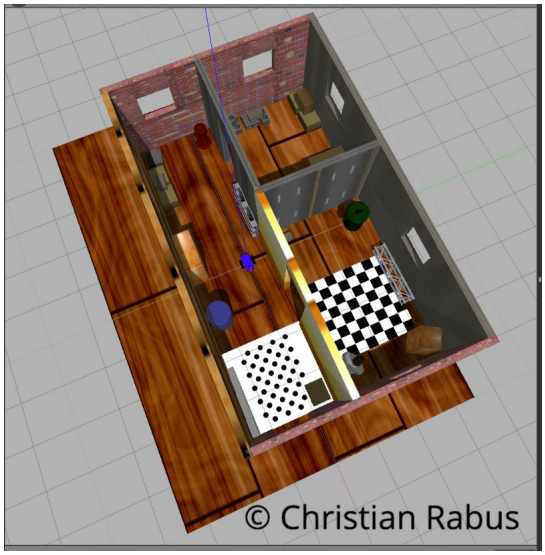

Fig. 4. office.world in gazebo

4 Results

4.1 Kitchen and Dining World

After launching the robot within the Kitchen and Dining world in Gazebo, and then starting the teleop and mapping nodes, data was immediately gathered Fig. 5, Fig. 6. The robot was manually navigated through the entire environment several times (3 times). The results of the accurate mapping are shown in Fig. 7: Mapping with path of robot, Fig. 8: 3D octomap, Fig. 9: 2D occupancy grid map.
With the *rtabmap-databaseViewer*, the rtabmap database (rtabmap.db) can be evaluated, Fig. 10. The results are:

Neighbor (535), Neighbor Merged (0), **Global Loop closure (34)**, **Local loop closure by space (828)**, Local loop closure by time (0), User loop closure (0), and Prior link (0).

4.2 Office World

After mapping the Kitchen and Dining world, the new created Office world was mapped, using the same robot with the same settings for the RTAB-Map algorithm. Fig. 11 and Fig. 12 show the initial mapping position right before navigating the robot through the building. The robot was manually navigated through the entire environment several times (2 times). The results of the accurate mapping are shown in Fig. 13: Mapping with path of robot, Fig. 14: 3D octomap, Fig. 15: 2D occupancy grid map.
The stored mapping database was evaluated with the *rtabmap-databaseViewer*, Fig. 16. The results are: **Neighbor (450)**, Neighbor Merged (0), **Global Loop closure (87)**, **Local loop closure by space (752)**, Local loop closure by time (0), User loop closure (0), and Prior link (0).

5 Discussion

The mapping was finally successful for both environments. Initially, the results of the mapping were poor. No meaningful 3D reconstruction could be achieved with the default settings. It therefore took a few attempts until the correct settings for the sensors and the mapping algorithm were found.
For the sensors and the differential drive the *updateRate* was set to 10. Otherwise, no further settings were made.
For the mapping algorithm the *Rtabmap/DetectionRate* was set to 5 Hz, so that the input images are filtered more often. Furthermore, the *Reg/Strategy* was set to 1, so that the data from the laser is also used for the loop closure. In addition, the number of required *Vis/MinInliers* has been set to 20, so more features are needed in an image to accept loop closure. With the new settings it is now possible to achieve an accurate mapping for both environments. Both the results of the 2D Occupancy Grid Map and the 3D Map are satisfactory for both worlds.
Especially the consideration of the laser data for loop closure takes an important role. This can also be determined from the data of the respective rtabmap database evaluations:

- **Kitchen and Dining world**
  - Global Loop closure (34)
  - Local loop closure by space (828)
- **Office world**
  - Global Loop closure (87)
  - Local loop closure by space (752)

The amount of global loop closure in the Office world is more than that in the Kitchen and Dining world. An explanation for this could be that the Office world has more features due to the many different existing objects and different textured walls as well as textured floor. Features are essential for a successful loop closure.
Another difference detected during mapping was the performance of the simulation environment. While during the mapping of the Office world, the robot could be navigated smoothly through the environment, the Kitchen and Dining world showed a clear judder. A possible cause could be the size of the environment. The Kitchen and Dining area is larger in size.
In conclusion, RTAB-Map is a very good algorithm to perform SLAM and obtain 2D/3D maps in a wide range of environments. It was possible to achieve good results with the adjusted settings. In order to optimize the results even further, the RTAB-Map algorithm offers many more options that have not yet been fully leveraged in this project.

6 Future Work

The RTAB-Map algorithm is a powerful tool to perform SLAM successfully. Alone the tutorial page [7] provides 10 different ideas how this SLAM approach can be used in different scenarios.
In order to use the RTAB-Map algorithm in practice and to implement the idea of the Office world, a robot with a differential drive, equipped with the nvidia Jetson TX2 and a Microsoft Kinect Camera, is to be developed. The first task is to autonomously create a map of a large office building. Then specific areas are labeled on the map (e.g. offices are assigned the names of employees). The labeled map can now be used for indoor navigation. New employees or guest can use their smart phone cameras to observe environment and get navigated with Augmented Reality to their desired location. The map can also be used for mobile service robots that are supposed to transport objects from A to B.

References

[1] RTAB-Map, http://introlab.github.io/rtabmap/.
[2] ROS diff_drive_controller, http://wiki.ros.org/diff_drive_controller.
[3] Lesson15: Introduction to Mapping and SLAM, Udacity Robotics Software Engineer Nanodegree Program.
[4] EKF SLAM, https://en.wikipedia.org/wiki/EKF_SLAM.
[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT press, 2005.
[6] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.
[7] rtabmap_ros, http://wiki.ros.org/rtabmap_ros.

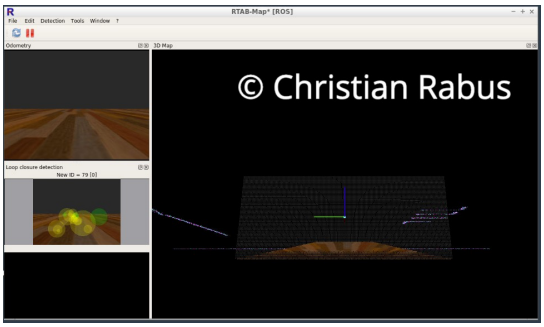

Fig. 5. rtabmapviz: kitchen dining start position mapping

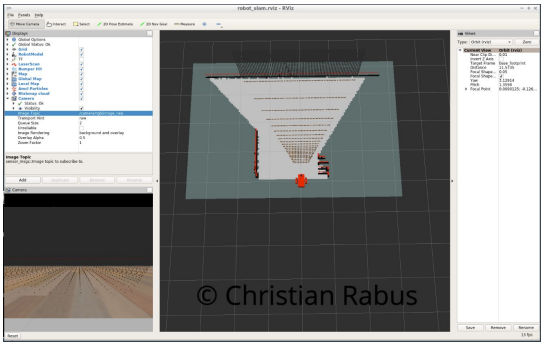

Fig. 6. rviz: kitchen dining start position mapping

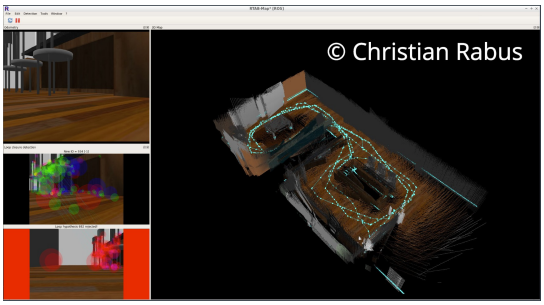

Fig. 7. rtabmapviz: kitchen dining after mapping

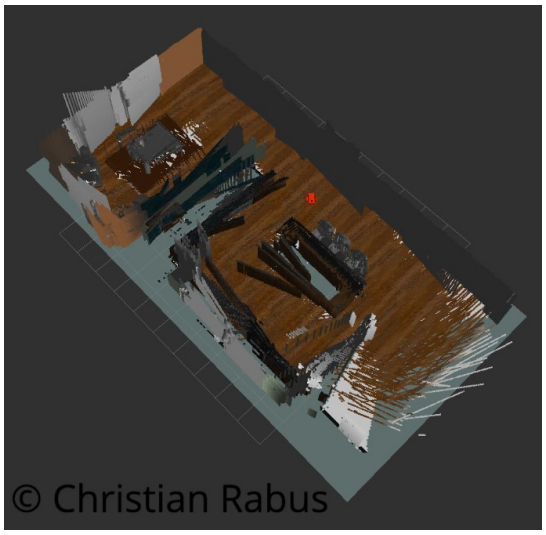

Fig. 8. rviz: kitchen dining after mapping

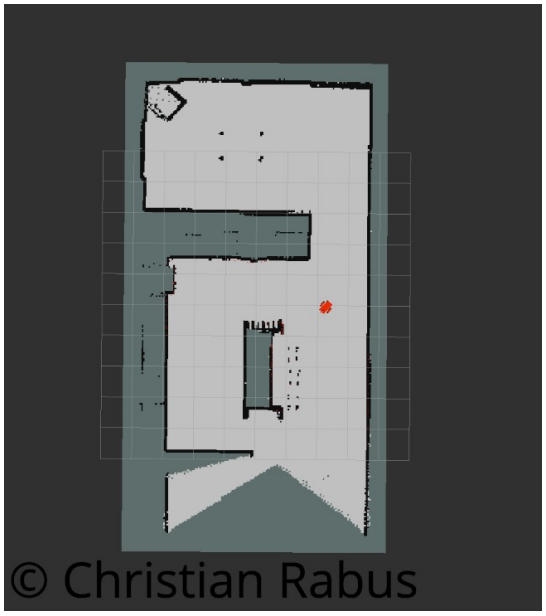

Fig. 9. 2D occupancy map of kitchen dining after mapping

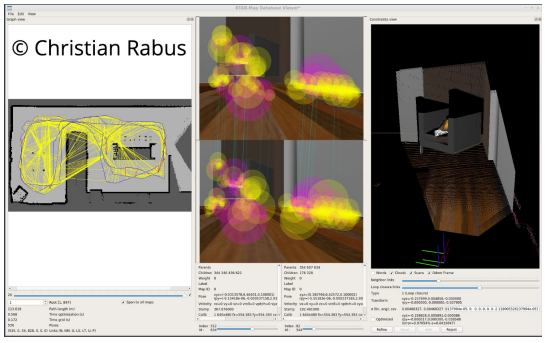

Fig. 10. databaseViewer: results kitchen dining after mapping

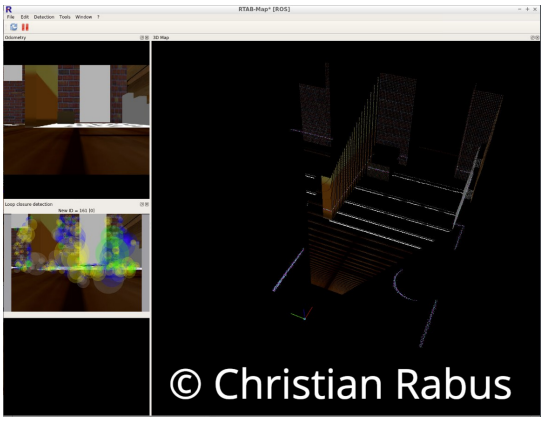

Fig. 11. rtabmapviz: office start position mapping

Fig. 14. rviz: office after mapping

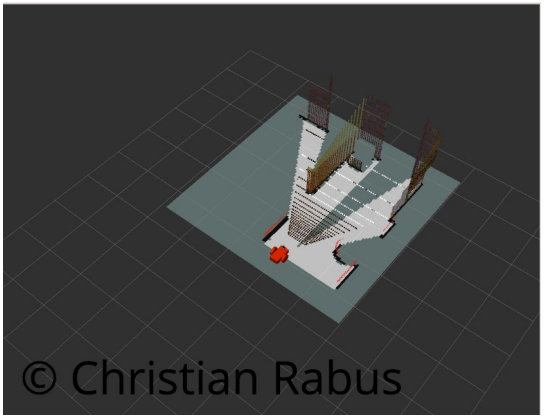

Fig. 12. rviz: office start position mapping

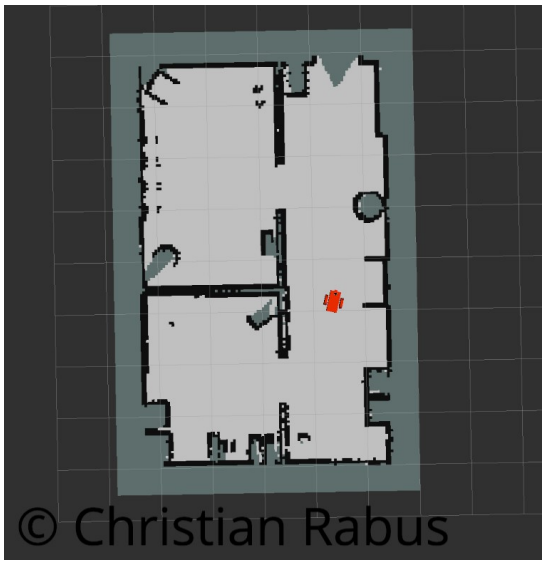

Fig. 15. 2D occupancy map of office after mapping

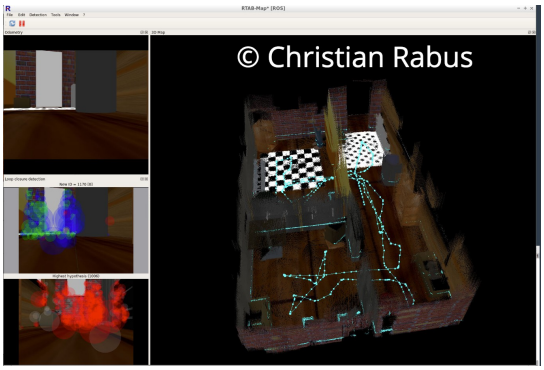

Fig. 13. rtabmapviz: office after mapping

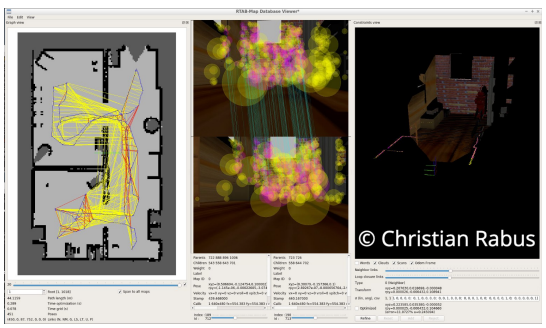

Fig. 16. databaseViewer: results office after mapping