Map-My-World Robot

Stephan Hohne

Abstract—A ROS package containing a model of a mobile rover and two indoor environments is created. The robot model is equipped with a Kinect RGB-D camera and a Hokuyo laser range finder. Launch files for two Gazebo worlds and RViz visualization are provided. A RTAB-Map node is configured to let the rover perform simultaneous localization and mapping, and to detect loop closures. The resulting databases are used to generate 2D occupancy grid maps and 3D point clouds of both environments.

Index Terms—GraphSLAM, RTAB-Map, loop closure detection, 2D occupancy grid mapping, 3D point cloud.

1 Introduction

For robotic applications like logistics and maintenance, a mobile robot has to navigate an indoor environment without being supplied with an a priori map. Even when an initial map of such an environment is available, the locations of objects shift over time. Therefore the robot can not rely on the correctness and accuracy of the given map, and it should continuously map both discovered and undiscovered areas. For the discussion of probabilistic approaches to mapping, the textbooks [1], [2] haven been used as a reference.

In a mapping run, the robot drives through the environment and collects sensor and odometry data. The space of all possible maps that are consistent with this input is huge. The robot must recognize places it has visited before. There will be cycles in the robot path, and the corresponding loop closure should be detected and incorporated into the map. This combination of discrete correspondences and a large continuous hypothesis space makes estimating the full posterior of the map particularly challenging. While driving through the world, the robot odometry accumulates inaccuracy over time, which must be corrected using information gathered from measurements. The robot must handle perceptual ambiguity when there are repetitive patterns in the environment. Furthermore, the noise in sensor measurements must be filtered out, and mapping is computationally expensive because of the large amount of image data that needs to be processed.

When the robot path is known, mapping can be achieved with the occupancy grid mapping algorithm. A discrete approximation of the map is generated by dividing the global map into grid cells. The state of a cell can be occupied, free or unknown. Even with this binary restriction and discrete approximation, the number of possible maps is exponential. The occupancy grid mapping algorithm is a probabilistic approach that uses binary Bayes filtering to estimate a posterior belief for the map, taking noisy sensor measurements and known poses as an input.

In many real-world scenarios a system for global positioning is not available (e.g. there are no predefined beacons), so the robot has to rely on its on-board sensor equipment for localization and mapping. This adds another level of complexity, because a unique reference frame for assigning poses to camera images is not available. Additionally, the drift caused by non-deterministic control actions creates an inherent localization problem. The mapping without known poses problem is therefore referred to as simultaneous localization and mapping (SLAM).

S2H.Mobile@gmail.com

The task of the algorithm is to find the most likely combination of poses x and map m, given measurements y and control commands u. The online variant of the SLAM problem asks to find the map and the instantaneous pose x_k at time step k. Stated as a probability equation, the goal is to determine the posterior probability distribution $P\left(x_k, m|y_{1:k}, u_{1:k}\right)$, where the sequence of measurements $y_{1:k}$ and controls $u_{1:k}$ from the beginning to the current time step are given. In contrast, the full SLAM problem asks for the entire path $x_{1:k}$. The corresponding posterior is given by

$$P\left(x_{1:k}, m|y_{1:k}, u_{1:k}\right) \ . \qquad (1)$$

Integrating over the consecutive full SLAM posteriors for each time step up to k recovers the online SLAM posterior.

Among the probabilistic approaches to solve the SLAM problem are the Kalman filter (see chapter 8 in [1]), the binary Bayes filter and the particle filter. For instance, the Rao-Blackwellized approach to uses a particle filter to estimate a posterior over the trajectory and a low-dimensional extended Kalman filter to solve for independent Gaussian features in the map. The Grid-based FastSLAM algorithm combines three estimation techniques. It uses a particle filter to sample motion and an occupancy grid to estimate the map. For each particle, the importance weight method estimates the likelihood of the measurement, given the pose and map associated with the particle. The GraphSLAM algorithm solves the full SLAM problem, as discussed in section 2.2. Real-time appearance based mapping is based on GraphSLAM and uses data collected from vision sensors for localization and mapping [4]. Loop closures are detected by comparing image features in order to determine whether the robot has seen a location before [5].

In this project, the RTAB-Map ROS package is used to perform SLAM in two simulated environments. A mobile rover operates on a 2D ground plane within two 3D worlds, see images 1 and 2. It collects RGB-D camera images and laser range scans. The RTAB-Map node creates 2D occupancy grid maps. The RTAB-Map database viewer is used to reconstruct 3D point clouds from the recorded mapping data.

This report is organized as follows. In section 2, some background is given on probabilistic approaches to mapping and visualization in two and three dimensions. In section 3, the ROS package for the robot and world models is described. The results of the mapping runs are presented in section 4 and discussed in section 5. An outlook to future work is given in section 6.

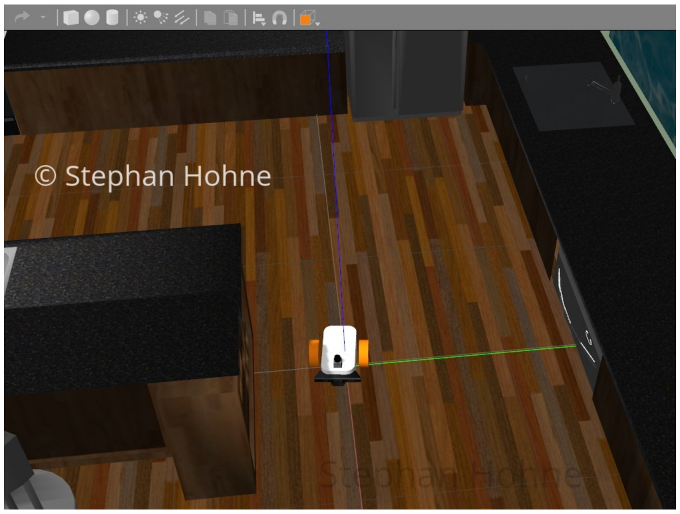


Fig. 1. Rover model in the kitchen dining world viewed from the top. Screenshot taken in Gazebo.

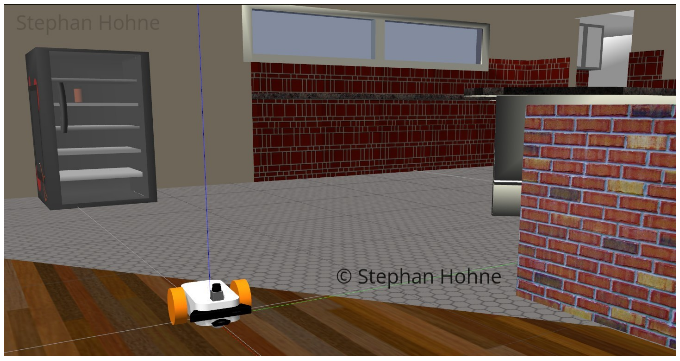


Fig. 2. View of the rover mapping the `cafe.world`. Screenshot taken in Gazebo.

2 Background: Mapping

2.1 Binary Bayes Filtering

Occupancy grid mapping is a probabilistic method to generate a map with known locations of the robot. The robot and the environment are represented mathematically by a dynamical system. The robot path is given by the sequence of poses $x_{1:k}$ up to the current time step k, and the noisy measurements are represented by y_k. The goal is to implement an algorithm that determines the posterior probability distribution

$$P\left(m|x_{1:k}, y_{1:k}\right) . \tag{2}$$

The posterior is conditioned on the trajectory and all measurements up to the current time step.

The occupancy grid mapping algorithm aims at determining the map m that maximizes the posterior probability 2 using binary Bayes filtering. The 2D environment is uniformly partitioned into binary grid cells m_l enumerated by the index $l = 1, \ldots, n$. Each cell stores the probability $P(m_l)$ of being occupied. The main assumption of occupancy grid mapping is that the cells can be considered independently so that the posterior turns into a product of marginals,

$$P\left(m|x_{1:k}, y_{1:k}\right) = \prod_{l=1}^{n} P\left(m_l|x_{1:k}, y_{1:k}\right) . \tag{3}$$

The remaining task is to find a posterior $P\left(m_l|x_{1:k}, y_{1:k}\right)$ for each of the 2^n cells and to describe how to update a single cell upon sensory input. The binary state of the map is static, it does not change during measurement, and the map states $m_l \in \{0, 1\}$ are much less complex than the measurements (for example a camera images). Therefore an inverse measurement model is applied to compute the belief $P\left(m_l|y_{1:k}\right)$. To represent the posterior belief mathematically, the log odds representation of probabilities is used. For the event e with probability $p(e)$, the log odds representation is given by

$$\mathcal{O}\left(e\right) := \log\left(\frac{p\left(e\right)}{1 - p\left(e\right)}\right) . \tag{4}$$

The advantage of this representation is that factorized probabilities are turned into sums and numbers close to zero or one can be handled with better accuracy.

The binary Bayes filter algorithm computes the log odds of the posterior belief l_k recursively according to

$$l_k = l_{k-1} + \mathcal{O}\left(m|y_k\right) - \mathcal{O}\left(m\right) , \tag{5}$$

where m is the binary state of the map and y_k is the sensory input at step k. The second to last term represents the inverse measurement model and the last term represents the initial belief.

The occupancy grid algorithm assumes that the grid cells are independent of each other, that the measurements are independent, and that the environment has a binary structure, where cells can be either occupied or free. These assumptions reduce the original high-dimensional estimation problem into an independent set of one-dimensional problems, but they limit the range of applicability and the correctness of the resulting maps. In the real world, consecutive measurements and neighboring grid cells are typically correlated.

The maps generated by the algorithm can be visualized using RViz for 2D maps and the OctoMap software [3] for 3D maps. The occupancy grid mapping algorithm is used as a post-processing step for SLAM. It takes as input the poses filtered from SLAM and noisy measurements and generates a refined map that can be used for path planning and navigation.

2.2 GraphSLAM

The GraphSLAM algorithm aims at solving the full SLAM problem 1. The goal is to find the most likely robot path and map. The idea is to organize information in a graph. A node in the graph represents either a robot pose x_t at a specific time step t or the location of a feature in the environment denoted as $m^{(i)}$ with $i = 1 \ldots \alpha$. An edge in the graph represents either a measurement constraint between a pose and a feature or a motion constraint between two successive poses. Since the spatial constraint are soft, they can be considered as springs connecting two masses. In this analogy, the full SLAM problem can be solved as a global graph optimization problem. The optimal graph configuration is the one where the springs are relaxed, and the forces on each of the nodes are minimized.

Formally, the maximum likelihood estimation (MLE) principle is applied to optimize the graph. The robot poses and map features are represented by random variables. The MLE technique tries to estimate the pose and map parameters that best explain the given measurements. Define a measurement update at time step t as

$$\overline{z}_t \coloneqq x_t + m_t^{(i)} \,, \tag{6}$$

which represents for instance a laser range finder measuring the distance to the landmark $m^{(i)}$. Equivalently, define a motion update as

$$\overline{x}_t \coloneqq x_{t-1} + u_t \,, \tag{7}$$

which could be realized as a control command instructing the robot to move a certain distance u_t. The update are assumed to have Gaussian noise. The corresponding probabilty distributions are given by

$$p_m\left(z_t\right) = \frac{1}{\sqrt{2\pi}\sigma_m} \exp\left(-\frac{1}{2}\left(\frac{z_t - \overline{z}_t}{\sigma_m}\right)^2\right) \tag{8}$$

$$p_u\left(x_t\right) = \frac{1}{\sqrt{2\pi}\sigma_u} \exp\left(-\frac{1}{2}\left(\frac{x_t - \overline{x}_t}{\sigma_u}\right)^2\right) \tag{9}$$

In some simple cases it is possible to find an analytical solution to MLE by converting the target function to the negative log-likelihood form

$$J_{\text{GraphSLAM}} = \sum_t \left(\frac{z_t - \overline{z}_t}{\sigma_m}\right)^2 + \sum_t \left(\frac{x_t - \overline{x}_t}{\sigma_u}\right)^2 \,,$$

trying to minimize the sum of all constraints. In more complex realistic scenarios, approximate numerical solutions are needed, for instance by applying gradient descent techniques.

States and measurements in the real world are multi-dimensional, and must be expressed as vectors $\mathbf{x}_t$ and $\mathbf{z}_t$, respectively. The constraints are given by

$$\mathbf{v}_t \coloneqq \mathbf{z}_t - h\left(\mathbf{x}_t, m_t\right) \,, \tag{10}$$

$$\mathbf{w}_t \coloneqq \mathbf{x}_t - g\left(\mathbf{x}_{t-1}, u_t\right) \,, \tag{11}$$

where h is the measurement model and g is the motion model. Noise is represented by covariance matrices Q for the sensor model and R for the motion model. The estimation function is given by

$$J_{\text{GraphSLAM}} = \mathbf{x}_0^{\mathrm{T}} \Omega_0 \mathbf{x}_0 + \sum_t \left(\mathbf{w}_t^{\mathrm{T}} R_t^{-1} \mathbf{w}_t + \mathbf{v}_t^{\mathrm{T}} Q_t^{-1} \mathbf{v}_t\right) \,.$$

To store constraint information efficiently, the information matrix Ω and the information vector

$$\xi = \left(x_0, \ldots, x_t, m_1, \ldots, m_\alpha\right)$$

are introduced. The information matrix is inverse to the covariance. Once all the information is collected, the path and map are inferred from the information matrix and vector by the inversion operation $\Omega^{-1}\xi$. The resulting vector contains the best guess estimate for all poses and features.

The GraphSLAM algorithm solves the full SLAM problem. It has an improved accuracy over FastSLAM and reduces the need for on-board processing. Since it recovers the full path of the robot and the entire map, it can detect correlations between current and previous poses.

RTAB-Map is a graph based SLAM approach that uses image data collected from vision sensors like a RGB-D camera to localize the robot, map the environment, and detect loop closures [4], [5]. Image features are compared in order to determine if the robot has seen a location before. During a mapping run, the number of images to be processed increases linearly, and so does the complexity of the loop closure algorithm. RTAB-Map is optimized for long-term mapping and can do loop closure in real time. In contrast to the landmark based constraints discussed above, RTAB-Map considers only odometry constraints and loop closure constraints. RTAB-Map is able to generate 2D and 3D occupancy grid maps as well as 3D point clouds.

3 Model Configuration

The ROS package `slam_rover` defines two Gazebo worlds and a rover model that employs the RTAB-Map package to perform SLAM of both environments. For ease of reuse of the components, the package is organized in folders containing the robot model URDF files, Gazebo SDF files, meshes, scripts, configuration files and launch files.

The `urdf` folder contains the file `rover.xacro` defining the links and joints of the robot model used for physics simulation and visualization as well as the file `rover.gazebo` specifying the Gazebo plugins for differential drive, RGB-D camera and laser range finder. See section 3.1 for details. Besides the environments provided as student material, the `worlds` directory includes the file `cafe.world` defining a custom indoor environment in SDF markup, see section 3.2 for details. The image and mesh files necessary to model the Hokuyo laser and Kinect camera are downloaded from the Gazebo model database and stored in the `meshes` folder. The `launch` folder contains four ROS node launch configurations, as detailed in section 3.3. The `config` directory contains the RViz configuration file, and a script for tele-operating the rover can be found in `scripts`.

3.1 Robot Model

The rover model is shown in figures 1 and 2. The URDF specification can be found in `rover.xacro`. The transform tree associated to the robot geometry is shown in figure 3.

To create the robot model, the mobile rover from [6] is taken as a starting point. The chassis is left as in the original model, with two parallel wheels at the sides. The wheels are driven differentially. Caster wheels in the front and back stabilize the body while allowing the robot to rotate in place. To enable the robot to perform appearance based mapping using visual odometry, the generic RGB camera of the original model is upgraded to a Kinect RGB-D

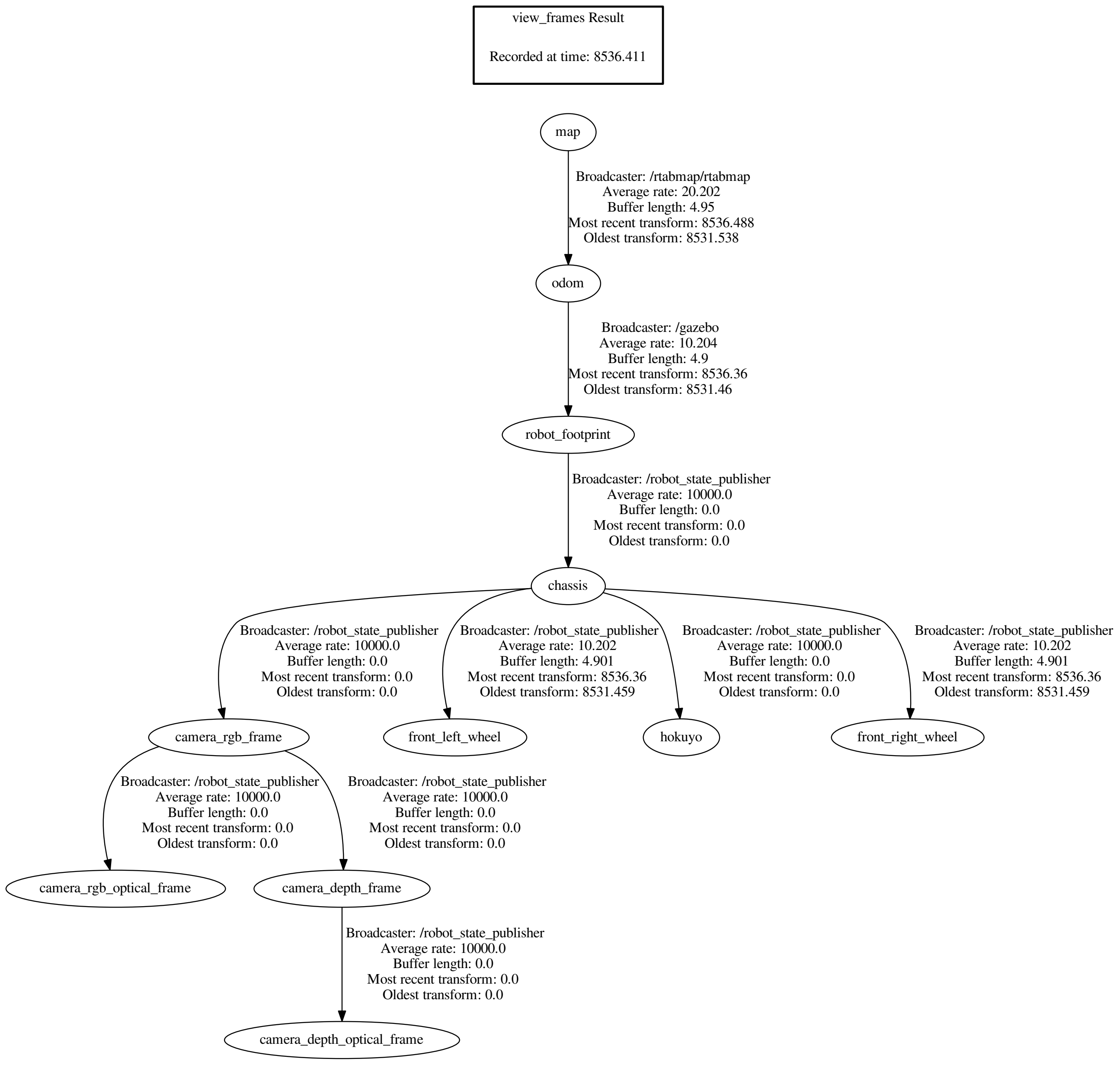

view_frames Result
Recorded at time: 8536.411
map
Broadcaster: /rtabmap/rtabmap
Average rate: 20.202
Buffer length: 4.95
Most recent transform: 8536.488
Oldest transform: 8531.538
odom
Broadcaster: /gazebo
Average rate: 10.204
Buffer length: 4.9
Most recent transform: 8536.36
Oldest transform: 8531.46
robot_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
chassis
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10.202
Buffer length: 4.901
Most recent transform: 8536.36
Oldest transform: 8531.459
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10.202
Buffer length: 4.901
Most recent transform: 8536.36
Oldest transform: 8531.459
camera_rgb_frame
front_left_wheel
hokuyo
front_right_wheel
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_rgb_optical_frame
camera_depth_frame
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
camera_depth_optical_frame

Fig. 3. Transform tree of the rover model. Output from `rosrun tf2_tools view_frames.py`.

camera. The camera is mounted to the front of the chassis to allow for unobstructed view, facing in forward direction. The mesh files for the Kinect camera model are downloaded from the Gazebo model database and included in the `slam_rover/meshes` folder. Like the original model, the rover is fitted with a Hokuyo 2D laser range finder. The corresponding `hokuyo` link is mounted with a fixed joint on the top of the chassis, to let the laser beans rotate without hitting any part of the robot. the laser range finder provides more precise localization and thereby refines geometric constraints

The differential drive plugin is configured in the `rover.gazebo` file to publish control commands to the `/cmd_vel` topic and odometry messages to the `/odom` topic. The camera plugin is configured to publish raw RGB images to `/camera/rgb/image_raw` and raw depth images to `/camera/depth/image_raw`. The laser plugin is configured to publish messages of type `LaserScan` to the `/slam_rover/laser/scan` topic. A graphical view of the ROS topics and nodes is shown in figure 4.

3.2 World Creation

As part of the project, the custom `cafe.world` is created in Gazebo. This indoor environment is based on the cafe model from the Gazebo database. The base model consists of a large room with a bar reaching into the room, and a separate kitchen. The kitchen can not be entered by the robot. Several items are added to the base model to serve

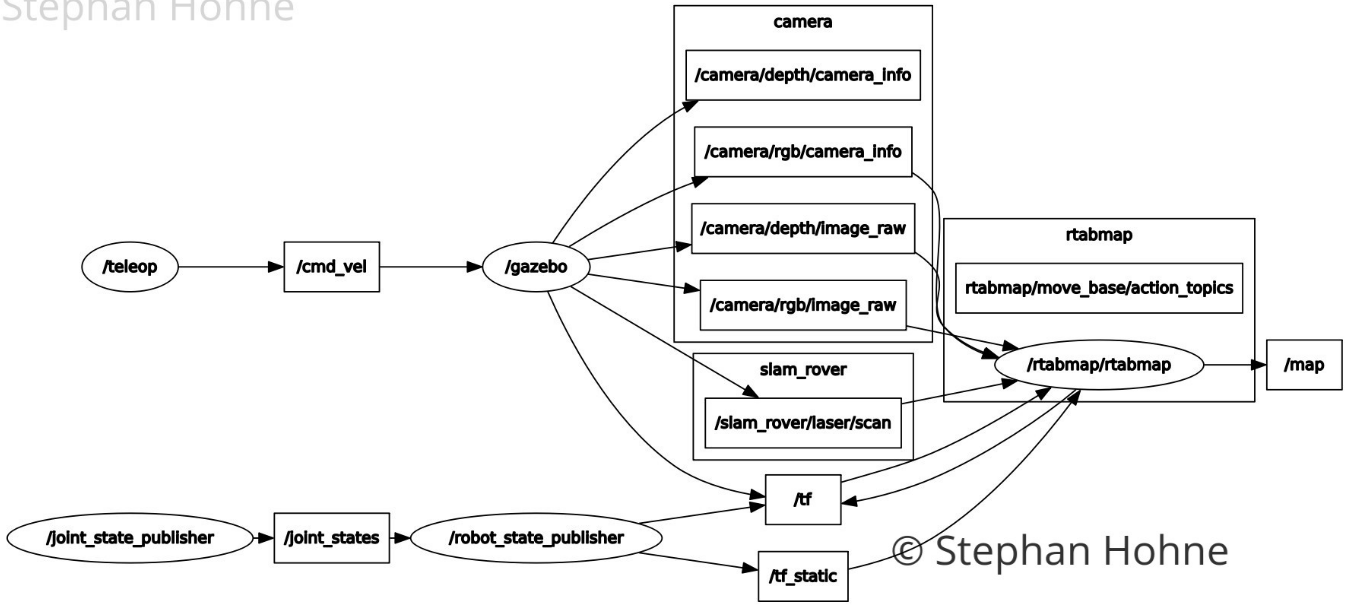


Fig. 4. Active topics when all nodes are launched. Output from `rqt_graph`.

as distinctive elements in the environment, such as tables, a ladder, a fountain, and a block made of red bricks. The large room is separated in two parts of approximately equal size by a white wall oriented in x-direction. A bird's eye view of the cafe world which also serves as the ground truth for mapping is shown in figure 5. An image of the rover navigating the cafe world is shown in figure 2.

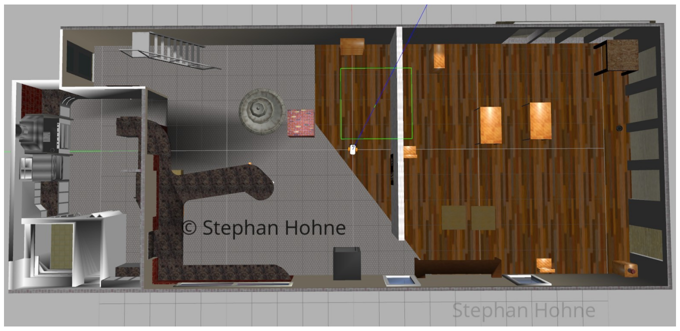
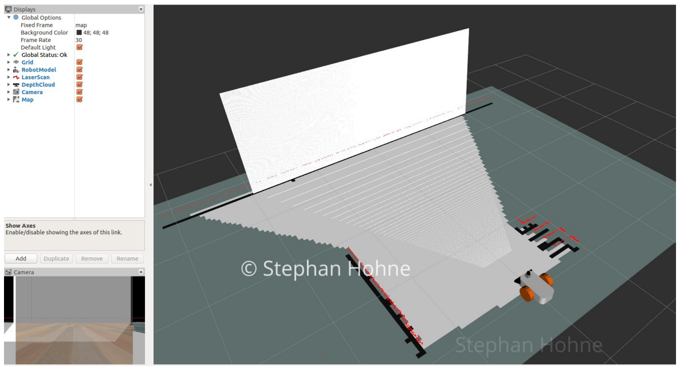


Fig. 5. View of the rover in the `cafe.world` from above. Screenshot taken in Gazebo.

3.3 ROS Launch Configuration

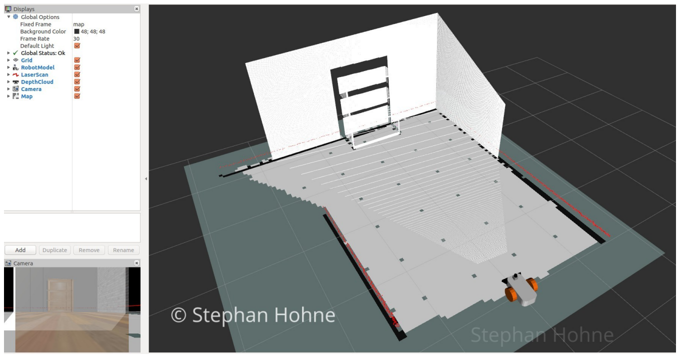
Four ROS nodes need to be launched in order to let the robot perform a mapping run in one of the environments. The `world.launch` file starts the Gazebo physics simulation of the rover in one of the environments. The world file, which can be either `kitchen_dining.world` or `cafe.world`, is specified inside the launch file. The RTAB-Map node is started with the `mapping.launch` file. The node is configured for visual loop closures using the SURF detection method. A minimum of 15 visual features must coincide for loop closure to be accepted. The `rviz.launch` file starts visualization of the rover, sensor data, as well as map and camera topics in RViz. See images 6 and 7 for screenshots taken during mapping of each world. The `teleop.launch` file starts the teleoperation script provided as student material. The script lets the user remote control the robot model.



Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Default Light ✓

Show Axes
Enable/disable showing the axes of this link.

Fig. 6. View of the rover mapping the `kitchen_dining.world`. Screenshot taken in RViz.



Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Default Light ✓

Fig. 7. View of the rover mapping the `cafe.world`. Screenshot taken in RViz.

When all the four aforementioned nodes are launched, the robot is able to perform mapping of the given environment. During the mapping run, the map data are stored in the `rtabmap.db` database. The full set of active topics and nodes during a mapping run is shown in figure 4. The `localization.launch` file can be started in order to localize the robot during the run.

4 Results

Remote controlled mapping runs were performed in both worlds. In order to collect enough images for comparison, the rover was steered 2 or 3 times across each room in each environment. During the runs, visualization of the generated 2D occupancy grid maps was achieved using RViz, as shown in figures 6 and 7. When the mapping runs were finished, the resulting map databases were evaluated using the RTAB-Map Database Viewer.

4.1 Kitchen Dining World

Mapping run 1 in the kitchen dining world resulted in a graph with 140 global loop closures. The RTAB-Map database created during this mapping run can be downloaded as a shared file (approx. 155 MB download, file name is `rtabmap_kitchen_dining_run1.db`).

The database was evaluated using RTAB-Map Database Viewer. Instances of loop closures were examined and visualized, see figures 11, 12, 13, 14. Yellow circles indicate image features, and pink circles indicate image features that match different points on the robot trajectory.

The robot trajectory while traversing the kitchen dining world and the graph view of the resulting 2D occupancy grid are shown in figure 8. The 2D map overlaid on the occupancy grid is shown in figure 9.

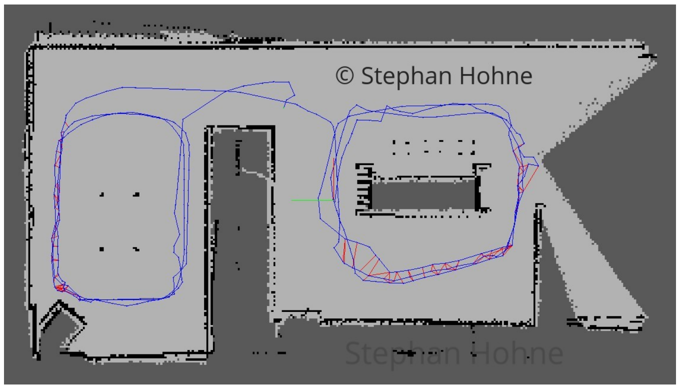


Fig. 8. Robot path and 2D map of kitchen dining world created during mapping run 1. Screenshot taken in RTAB-Map Database Viewer.

The *Export 3D Clouds* functionality was then used to reconstruct 3D point clouds. In the first export, the default settings were kept. Figure 16 shows the resulting reconstruction of the kitchen dining world. The features in the environment like tables and chairs are recognizable, see the further discussion in section 5.

In the second export, all available algorithms for enhancement of the reconstruction were enabled. The options are cloud filtering, cloud smoothing, gain compensation,

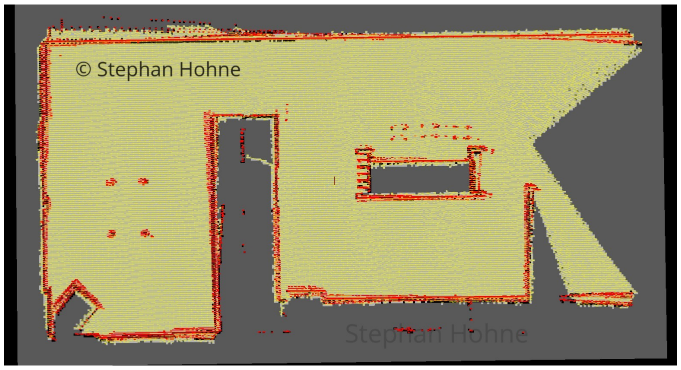


Fig. 9. 2D map and occupancy grid of kitchen dining world created during mapping run 1. Screenshot taken in RTAB-Map Database Viewer.

and meshing. The full export configuration can be downloaded as a shared file. The resulting 3D point cloud of the kitchen dining world is shown in figure 17, for a discussion see section 5.

4.2 Cafe World

Mapping run 2 in the cafe world resulted in a graph with 90 global loop closures. The RTAB-Map database created during this mapping run can be downloaded as a shared file (approx. 105 MB download, file name is `rtabmap_cafe_run2.db`). The resulting database was again evaluated in RTAB-Map database viewer. A 2D map and occupancy grid of the cafe world is shown in figure 10.

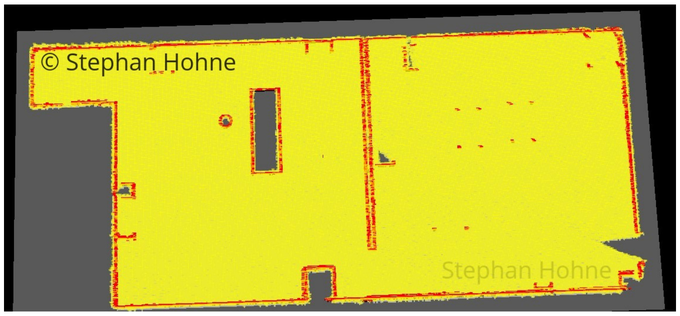


Fig. 10. 2D map and occupancy grid of cafe world created during mapping run 2. Screenshot taken in RTAB-Map Database Viewer.

The robot path traversing the cafe world is shown in figure 21 below. For this run, 90 global loop closures were detected. An example loop closure between index 43 and 55 is shown in figure 15. The RTAB-Map tool was used to export 3D reconstructions as before. Figure 18 shows the exported 3D point cloud using the default settings. Characteristic features of the environment like the ladder, tables and brick column are recognizable, see further discussion in section 5. More exports were done with enhanced reconstruction settings, equal to those for the kitchen dining world. The resulting 3D views of the cafe world are shown in figures 19 and 20.

5 Discussion

The overall goal of generating 2D occupancy grid maps and 3D visualizations of both the `kitchen_dining.world` and the custom `cafe.world` has been achieved. The generated 2D and 3D maps contain characteristic features, so that the ground truth environments can be recognized.

There were instances of apparent mapping errors, where the spatial relationships of elements could not be reproduced correctly. The most notable example is the chair in the lower left corner of the dining room in the kitchen dining world, see image 16 and 17. In the reconstruction, the chair seems to be rotated by $\pi/2$ around the z-axis when compared to the ground truth world.

In the cafe world, the robot sensors did not recognize the bar reaching into the left room. Therefore this feature is missing in the map, see for instance figure 10. The most probable explanation is that the walls of the bar are made of glass and therefore transparent. The footprint of the brick column is quadratic in Gazebo, see image 5, but rectangular in the 2D map, see image 10.

The generated 2D and 3D maps can be improved in quality and detail by doing mapping runs that cover the environment completely, and by further inspecting and optimizing the loop closures detected by RTAB-Map.

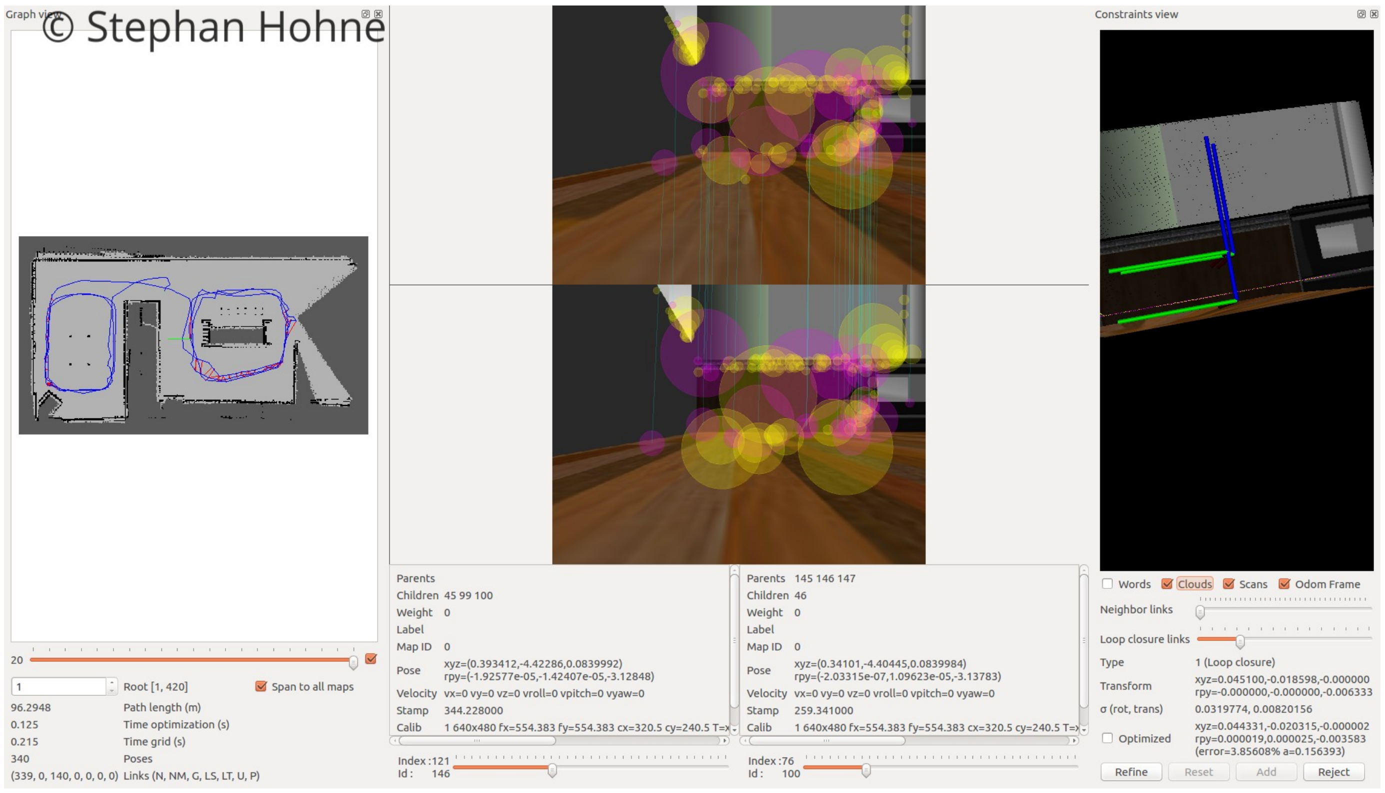
6 Future Work

It is desirable to explore more of the visualization options of the RTAB-Map software. For instance, it will be interesting to create 3D occupancy grid maps by enabling the `Grid/3D` and `Grid/FromDepth` parameters in the RTAB-Map ROS package. Another interesting 3D visualization option is to install the Octomap package.

A physical mobile robot based on the Jetson TX2 is under construction. The plan is to deploy the mapping stack developed here to it. A reasonable first task for RTAB-Map is to create a map of the Wifi coverage of the environment, following the ROS Wiki tutorial.

References

[1] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki and Sebastian Thrun, *Principles of Robot Motion*, The MIT Press, 2005
[2] Sebastian Thrun, Wolfram Burgard and Dieter Fox, *Probabilistic Robotics*, The MIT Press, 2005
[3] Armin Hornung and Kai M. Wurm and Maren Bennewitz and Cyrill Stachniss and Wolfram Burgard, *OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees*, Autonomous Robots,2013, Software available at GitHub
[4] Labbe, M. and Michaud, F., *Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM*, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2661-2666, Sept 2014
[5] Labbe, M. and Michaud, F., *Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation*, IEEE Transactions on Robotics Vol. 29 No. 3, 734-745, 2013
[6] Stephan Hohne, *Robot Localization – Where Am I?*, project on GitHub, 2018



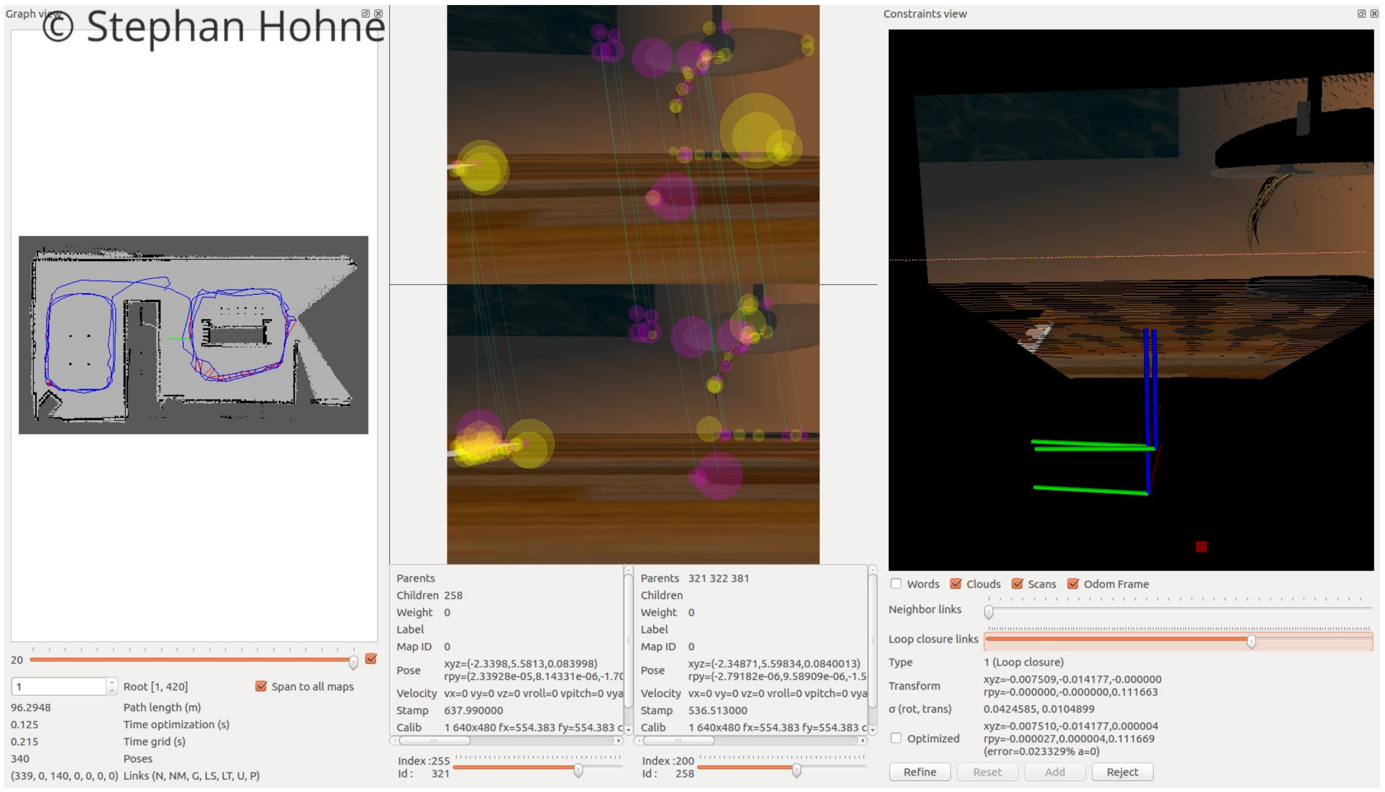
```
20
1        Root [1, 420]          Span to all maps
96.2948  Path length (m)
0.125    Time optimization (s)
0.215    Time grid (s)
340      Poses
(339, 0, 140, 0, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)
```

```
Parents
Children  45 99 100
Weight    0
Label
Map ID    0
Pose      xyz=(0.393412,-4.42286,0.0839992)
          rpy=(-1.92577e-05,-1.42407e-05,-3.12848)
Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp     344.228000
Calib     1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=240.5 T=x
Index :121
Id :   146
```

```
Parents   145 146 147
Children  46
Weight    0
Label
Map ID    0
Pose      xyz=(0.34101,-4.40445,0.0839984)
          rpy=(-2.03315e-07,1.09623e-05,-3.13783)
Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp     259.341000
Calib     1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=240.5 T=x
Index :76
Id :   100
```

```
Type            1 (Loop closure)
Transform       xyz=0.045100,-0.018598,-0.000000
                rpy=-0.000000,-0.000000,-0.006333
σ (rot, trans)  0.0319774, 0.00820156
Optimized       xyz=0.044331,-0.020315,-0.000002
                rpy=0.000019,0.000025,-0.003583
                (error=3.85608% a=0.156393)
```

Fig. 11. Loop closure in the map database `rtabmap_kitchen_dining_run1.db`. Screenshot taken in RTAB-Map Database Viewer.



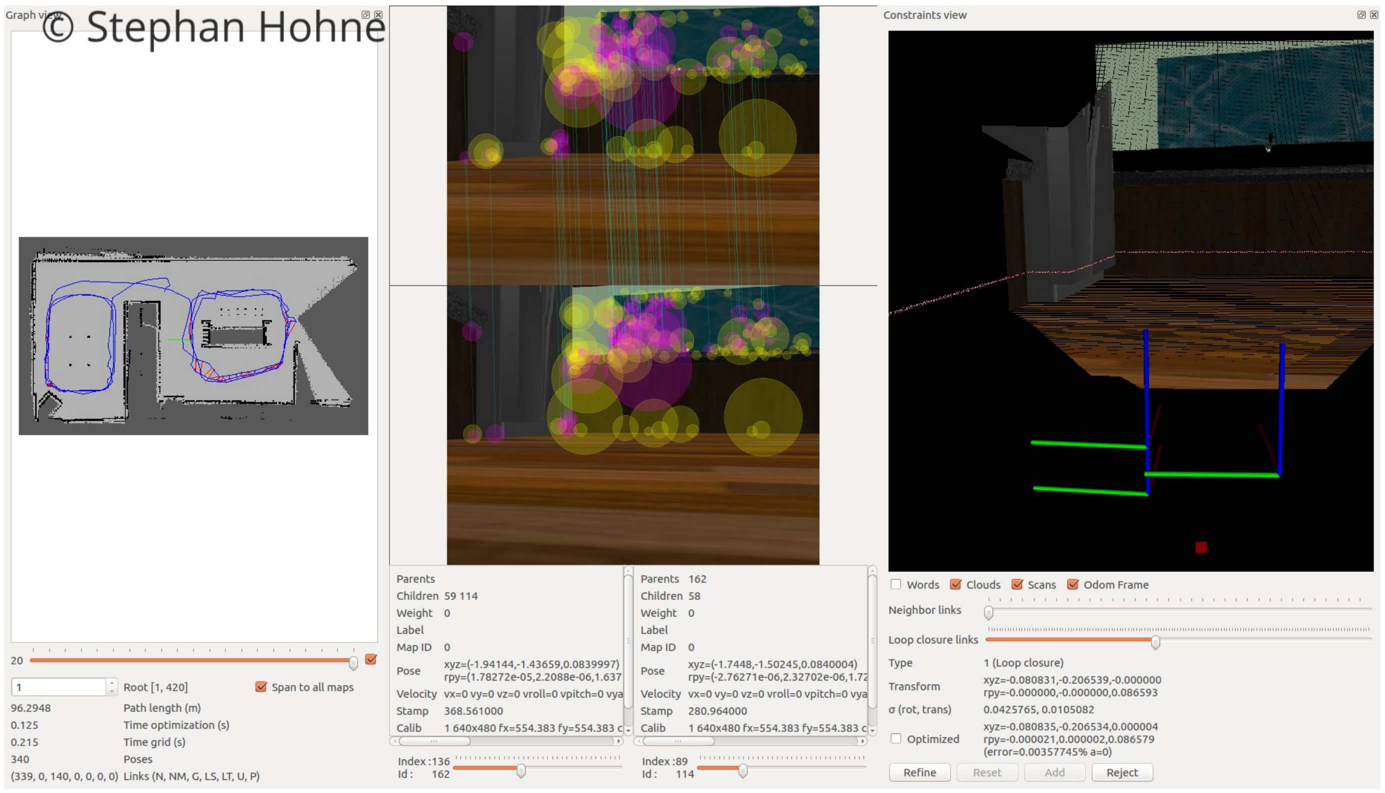
```
20
1        Root [1, 420]          Span to all maps
96.2948  Path length (m)
0.125    Time optimization (s)
0.215    Time grid (s)
340      Poses
(339, 0, 140, 0, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)
```

```
Parents
Children  258
Weight    0
Label
Map ID    0
Pose      xyz=(-2.3398,5.5813,0.083998)
          rpy=(2.33928e-05,8.14331e-06,-1.70
Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vya
Stamp     637.990000
Calib     1 640x480 fx=554.383 fy=554.383 c
Index :255
Id :   321
```

```
Parents   321 322 381
Children
Weight    0
Label
Map ID    0
Pose      xyz=(-2.34871,5.59834,0.0840013)
          rpy=(-2.79182e-06,9.58909e-06,-1.5
Velocity  vx=0 vy=0 vz=0 vroll=0 vpitch=0 vya
Stamp     536.513000
Calib     1 640x480 fx=554.383 fy=554.383 c
Index :200
Id :   258
```

```
Type            1 (Loop closure)
Transform       xyz=-0.007509,-0.014177,-0.000000
                rpy=-0.000000,-0.000000,0.111663
σ (rot, trans)  0.0424585, 0.0104899
Optimized       xyz=-0.007510,-0.014177,0.000004
                rpy=-0.000027,0.000004,0.111669
                (error=0.023329% a=0)
```

Fig. 12. Loop closure in the map database `rtabmap_kitchen_dining_run1.db`. Screenshot taken in RTAB-Map Database Viewer.



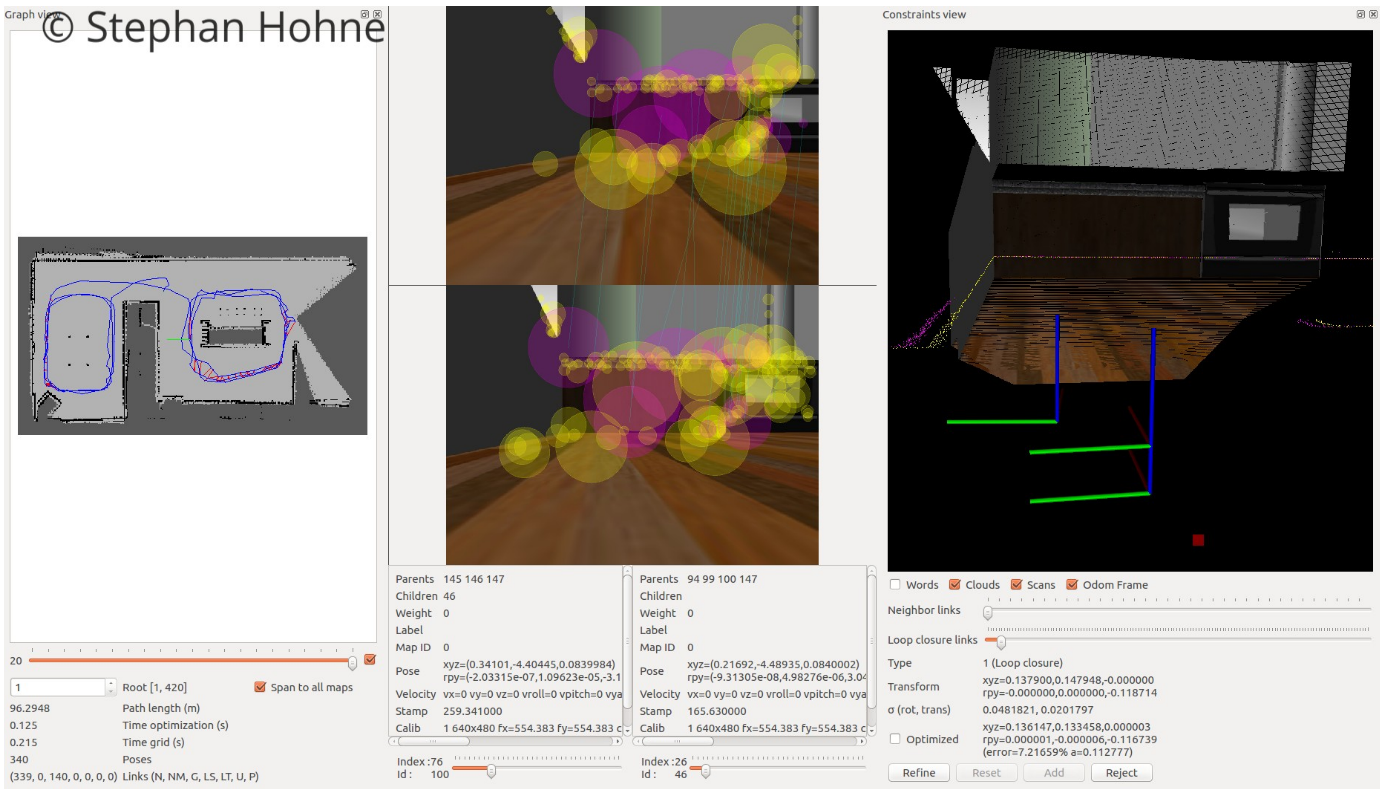
```
20
1        Root [1, 420]        Span to all maps
96.2948                  Path length (m)
0.125                    Time optimization (s)
0.215                    Time grid (s)
340                      Poses
(339, 0, 140, 0, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)
```

```
Parents
Children 59 114
Weight   0
Label
Map ID   0
Pose     xyz=(-1.94144,-1.43659,0.0839997)
         rpy=(1.78272e-05,2.2088e-06,1.637
Velocity vx=0 vy=0 vz=0 vroll=0 vpitch=0 vya
Stamp    368.561000
Calib    1 640x480 fx=554.383 fy=554.383 c
Index :136
Id :   162
```

```
Parents  162
Children 58
Weight   0
Label
Map ID   0
Pose     xyz=(-1.7448,-1.50245,0.0840004)
         rpy=(-2.76271e-06,2.32702e-06,1.72
Velocity vx=0 vy=0 vz=0 vroll=0 vpitch=0 vya
Stamp    280.964000
Calib    1 640x480 fx=554.383 fy=554.383 c
Index :89
Id :   114
```

```
Words  Clouds  Scans  Odom Frame
Neighbor links
Loop closure links
Type            1 (Loop closure)
Transform       xyz=-0.080831,-0.206539,-0.000000
                rpy=-0.000000,-0.000000,0.086593
σ (rot, trans)  0.0425765, 0.0105082
Optimized       xyz=-0.080835,-0.206534,0.000004
                rpy=-0.000021,0.000002,0.086579
                (error=0.00357745% a=0)
Refine  Reset  Add  Reject
```

Fig. 13. Loop closure in the map database `rtabmap_kitchen_dining_run1.db`. Screenshot taken in RTAB-Map Database Viewer.



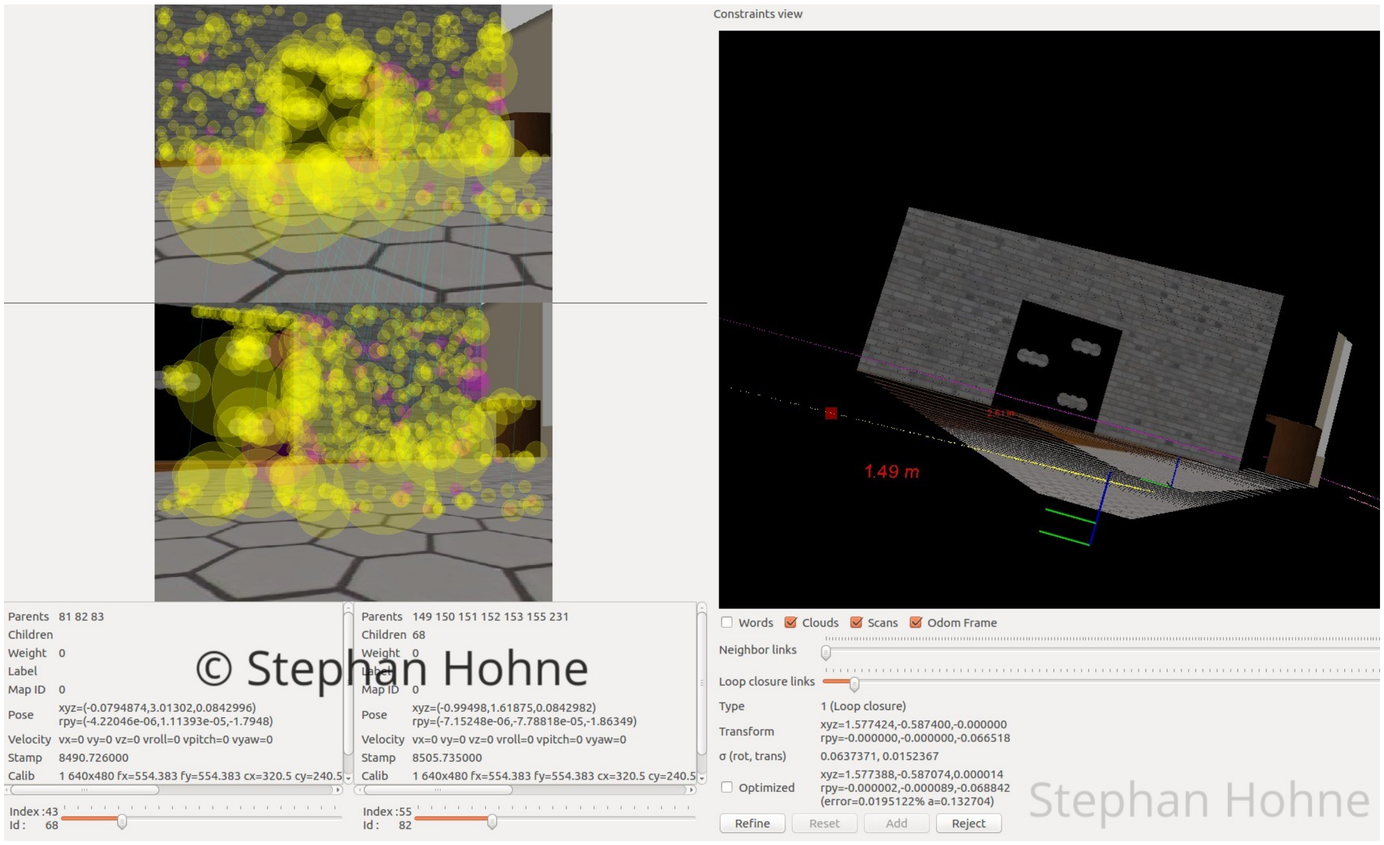
```
20
1        Root [1, 420]        Span to all maps
96.2948                  Path length (m)
0.125                    Time optimization (s)
0.215                    Time grid (s)
340                      Poses
(339, 0, 140, 0, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)
```

```
Parents  145 146 147
Children 46
Weight   0
Label
Map ID   0
Pose     xyz=(0.34101,-4.40445,0.0839984)
         rpy=(-2.03315e-07,1.09623e-05,-3.1
Velocity vx=0 vy=0 vz=0 vroll=0 vpitch=0 vya
Stamp    259.341000
Calib    1 640x480 fx=554.383 fy=554.383 c
Index :76
Id :   100
```

```
Parents  94 99 100 147
Children
Weight   0
Label
Map ID   0
Pose     xyz=(0.21692,-4.48935,0.0840002)
         rpy=(-9.31305e-08,4.98276e-06,3.04
Velocity vx=0 vy=0 vz=0 vroll=0 vpitch=0 vya
Stamp    165.630000
Calib    1 640x480 fx=554.383 fy=554.383 c
Index :26
Id :   46
```

```
Words  Clouds  Scans  Odom Frame
Neighbor links
Loop closure links
Type            1 (Loop closure)
Transform       xyz=0.137900,0.147948,-0.000000
                rpy=-0.000000,0.000000,-0.118714
σ (rot, trans)  0.0481821, 0.0201797
Optimized       xyz=0.136147,0.133458,0.000003
                rpy=0.000001,-0.000006,-0.116739
                (error=7.21659% a=0.112777)
Refine  Reset  Add  Reject
```

Fig. 14. Loop closure in the map database `rtabmap_kitchen_dining_run1.db`. Screenshot taken in RTAB-Map Database Viewer.



Parents	81 82 83
Children	
Weight	0
Label	
Map ID	0
Pose	xyz=(-0.0794874,3.01302,0.0842996) rpy=(-4.22046e-06,1.11393e-05,-1.7948)
Velocity	vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp	8490.726000
Calib	1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=240.5

Index :43
Id : 68

Parents	149 150 151 152 153 155 231
Children	68
Weight	0
Label	
Map ID	0
Pose	xyz=(-0.99498,1.61875,0.0842982) rpy=(-7.15248e-06,-7.78818e-05,-1.86349)
Velocity	vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp	8505.735000
Calib	1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=240.5

Index :55
Id : 82

Type	1 (Loop closure)
Transform	xyz=1.577424,-0.587400,-0.000000 rpy=-0.000000,-0.000000,-0.066518
σ (rot, trans)	0.0637371, 0.0152367
Optimized	xyz=1.577388,-0.587074,0.000014 rpy=-0.000002,-0.000089,-0.068842 (error=0.0195122% a=0.132704)

Fig. 15. Example of a loop closure in the map database for the `cafe.world`. Screenshot taken in RTAB-Map Database Viewer.

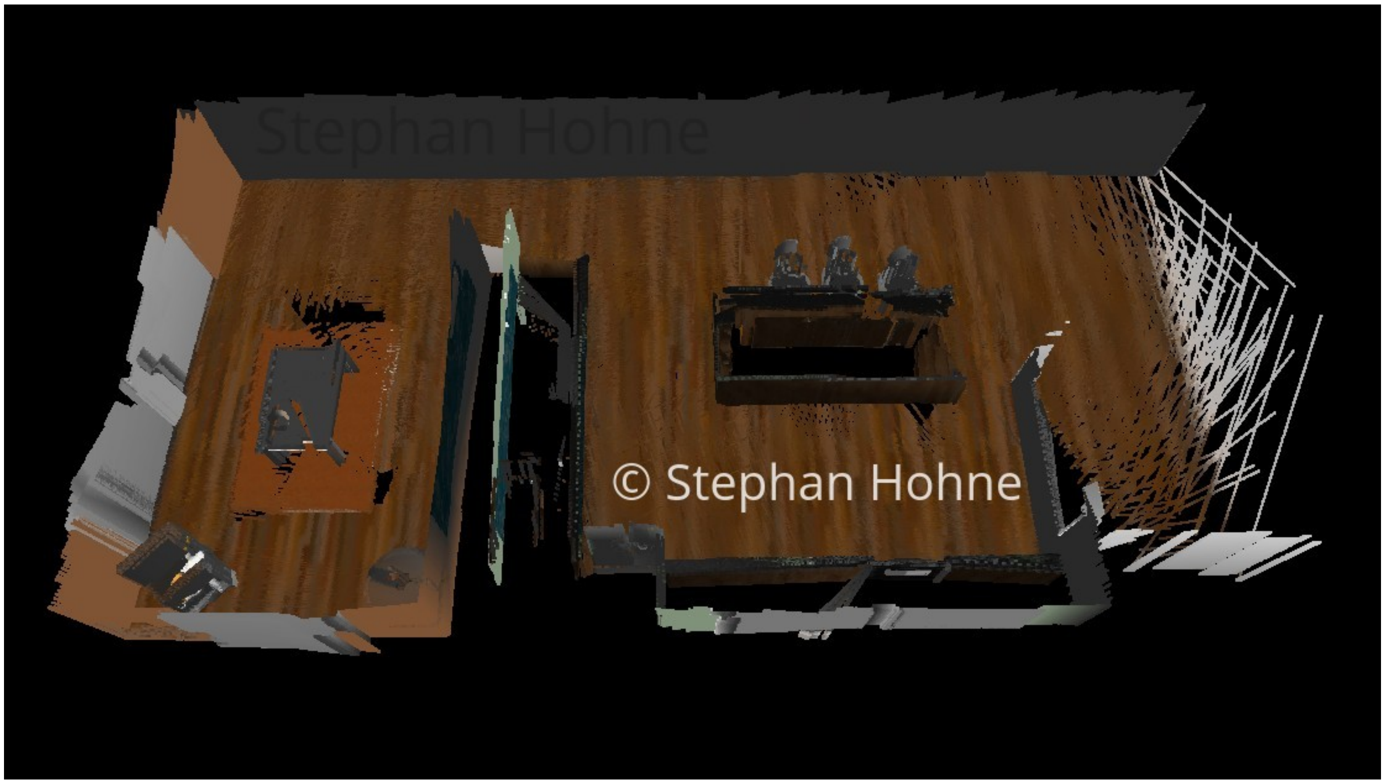


Fig. 16. Reconstructed 3D point cloud of kitchen dining world. Default settings were used for reconstruction. Screenshot taken in RTAB-Map Database Viewer.

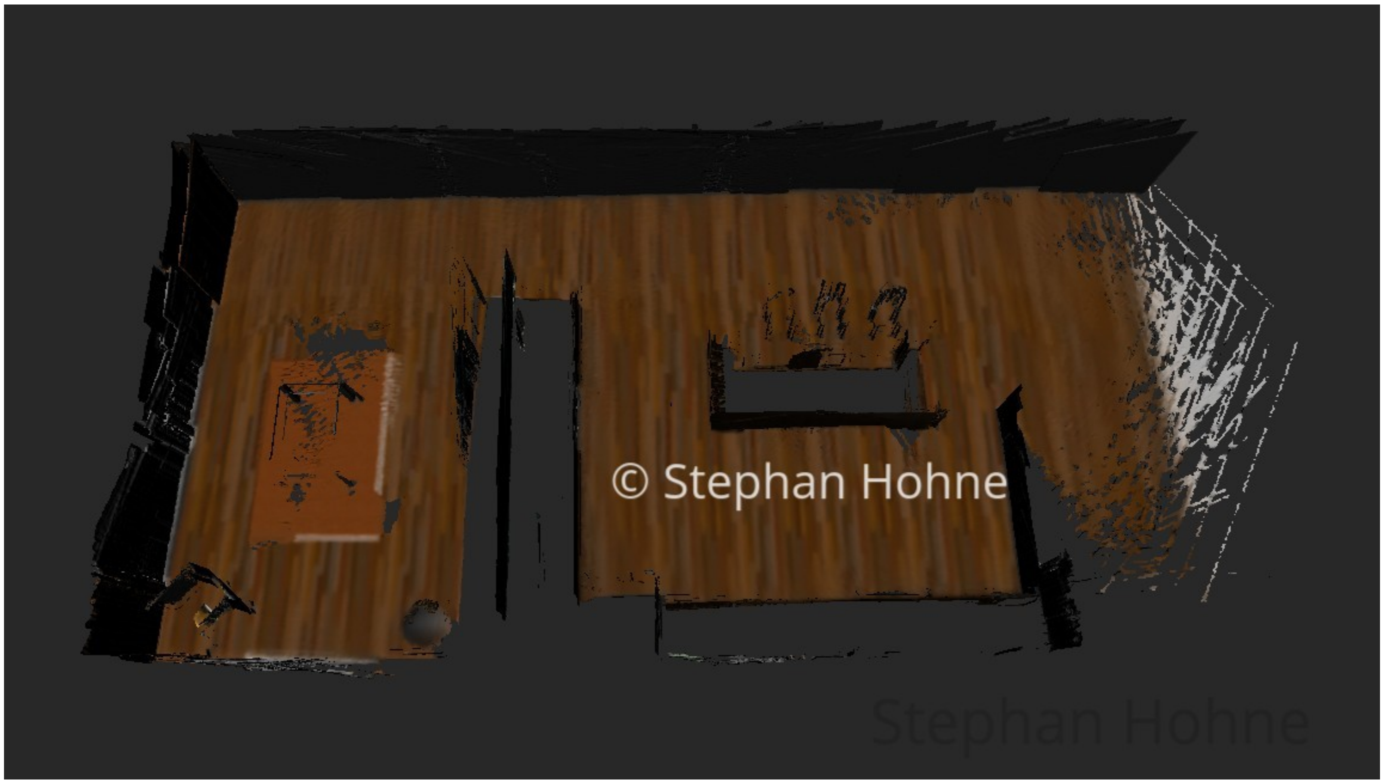


Fig. 17. Reconstructed 3D point cloud of kitchen dining world. Enhancement algorithms were applied for reconstruction. Screenshot taken in RTAB-Map Database Viewer.

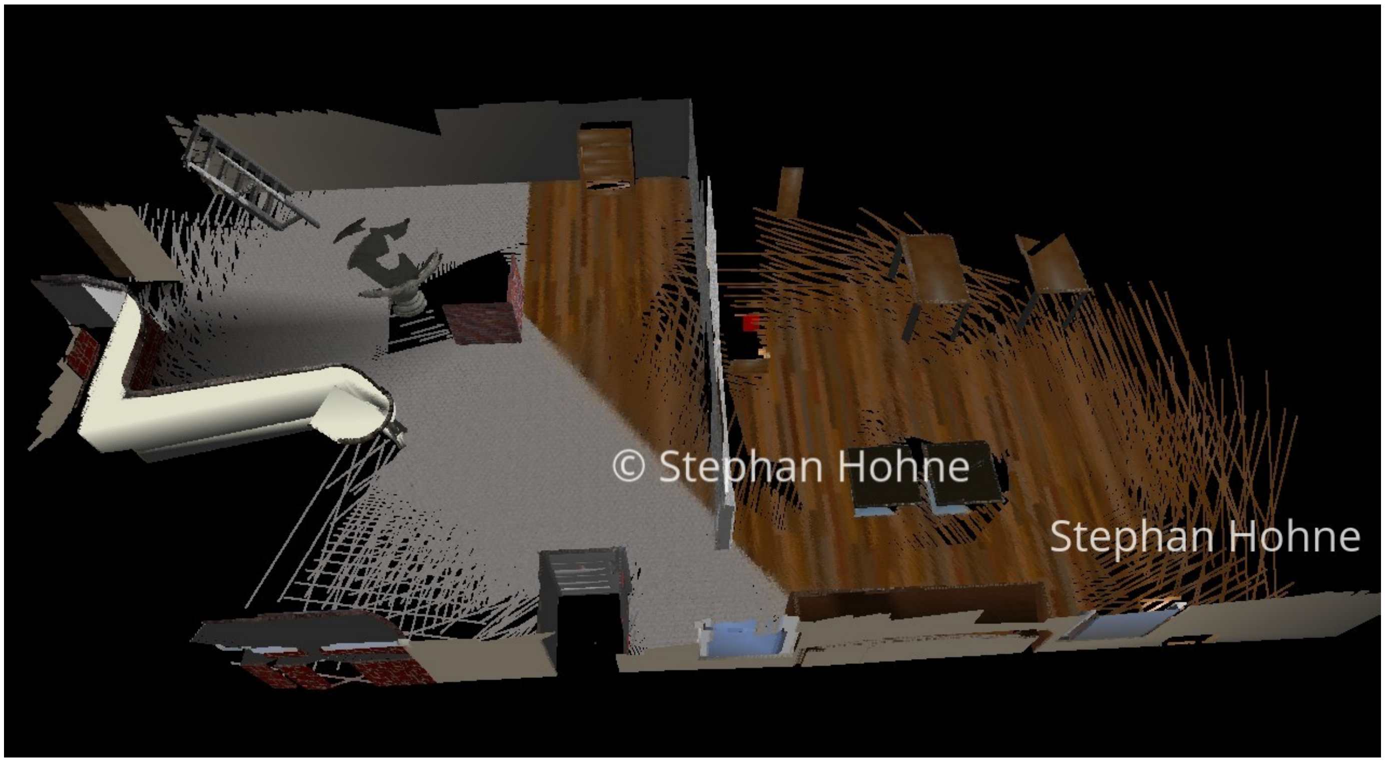


Fig. 18. Reconstructed 3D point cloud of cafe world. Default settings were used for reconstruction. Screenshot taken in RTAB-Map Database Viewer.

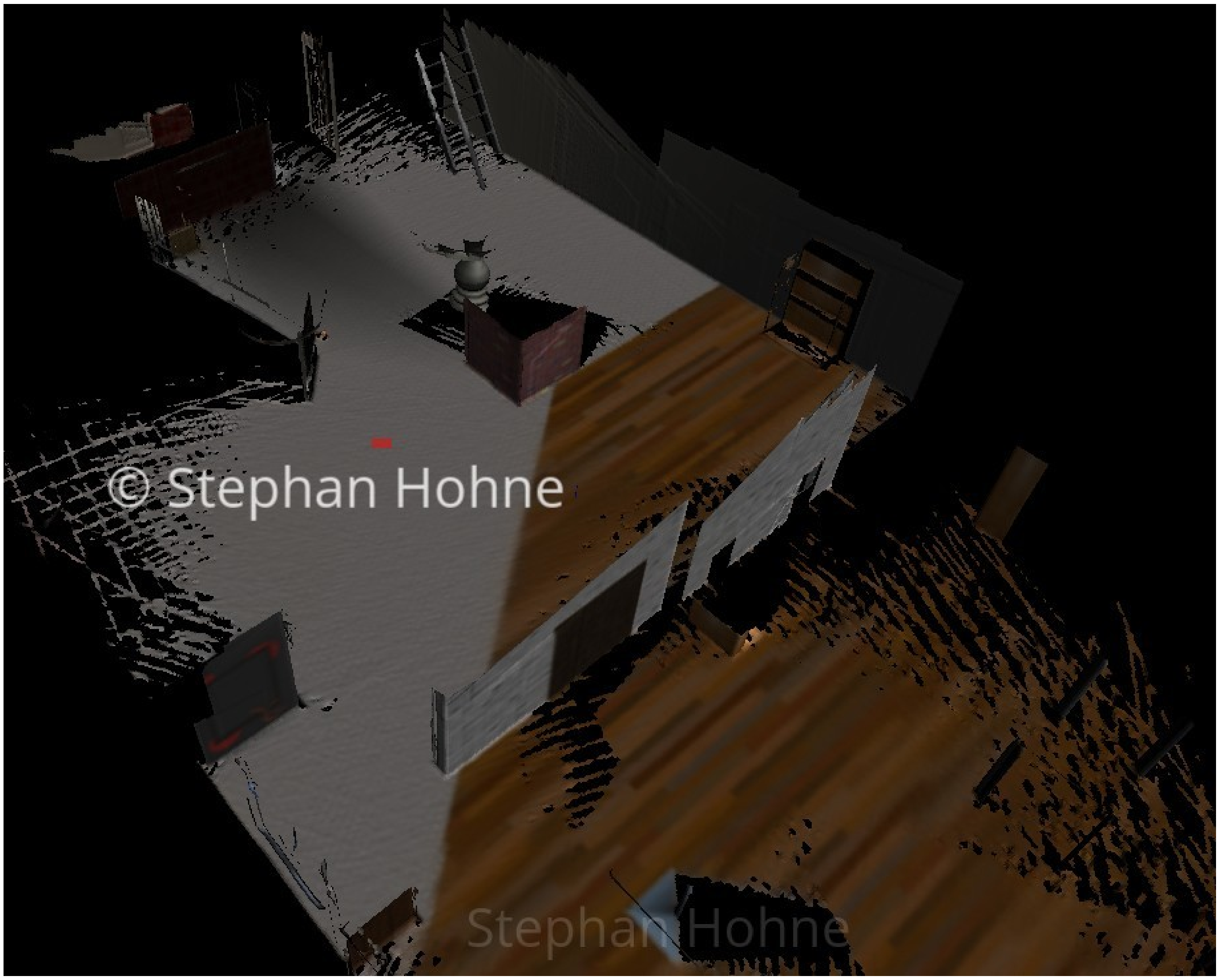


Fig. 19. 3D view of the cafe world with enhanced reconstruction options enabled. Screenshot taken in RTAB-Map Database Viewer.

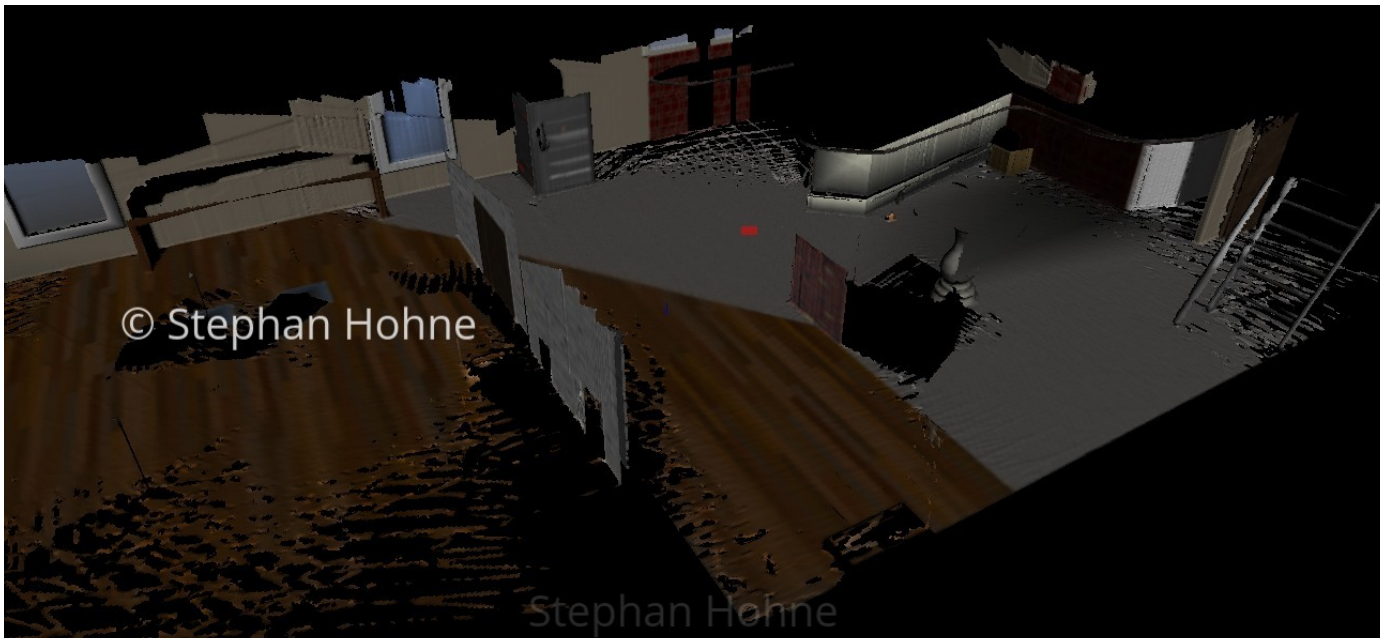
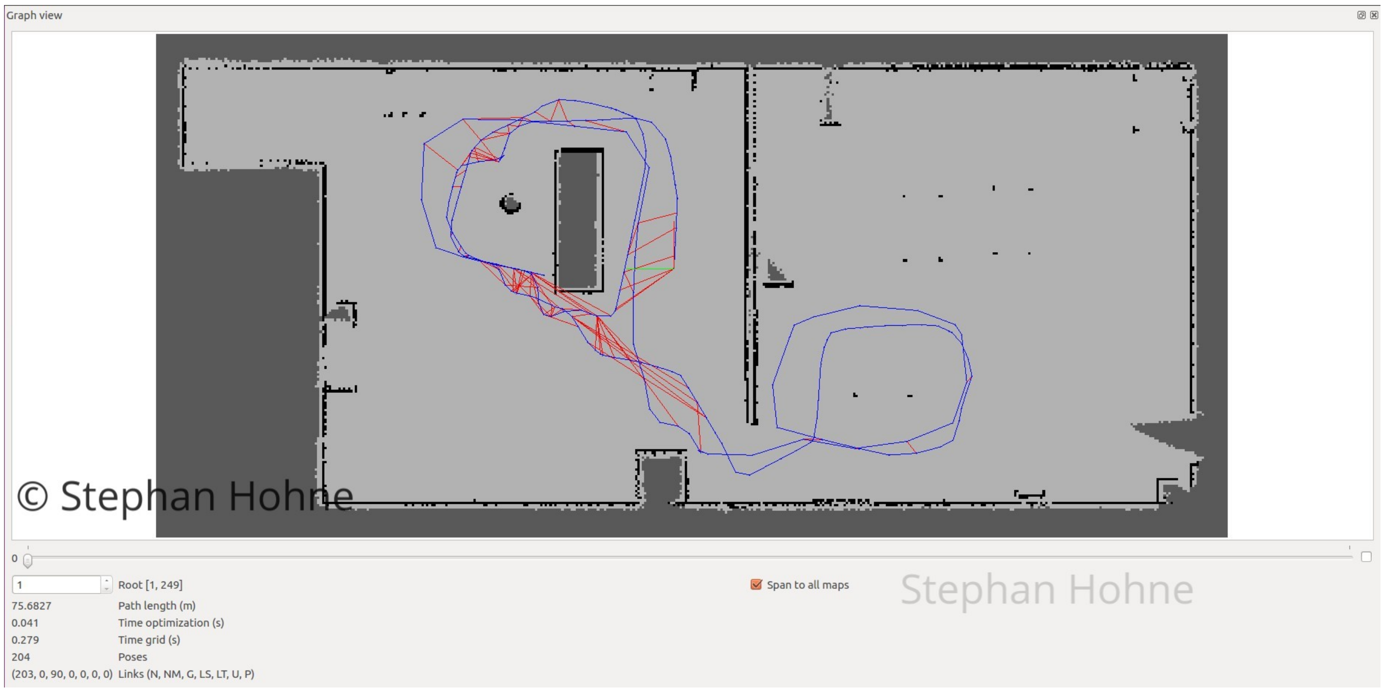


Fig. 20. 3D view of the cafe world with enhanced reconstruction options enabled. Screenshot taken in RTAB-Map Database Viewer.



```
1                   Root [1, 249]
75.6827             Path length (m)
0.041               Time optimization (s)
0.279               Time grid (s)
204                 Poses
(203, 0, 90, 0, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)
```

Fig. 21. Graph view showing the 2D map of the cafe world created during mapping run 2. A total of 90 global loop closures was detected. Screenshot taken in RTAB-Map Database Viewer.