

 README.md

# RoboND-MapMyWorld-Project-P7

Localization and mapping using RTAB-Map

## Abstract

SLAM or Simultaneous Localization and Mapping is an important topic within the Robotics community. It is not a particular algorithm or piece of software, but rather it refers to the problem of trying to simultaneously localize (i.e. find the position/orientation of) some sensor with respect to its surroundings, while at the same time mapping the structure of that environment.

In this project we evaluate the usage of RTAB-Map to localize and map an autonomous rover in two different environments:

- Kitchen
- Corridor

## Introduction

SLAM is central to a range of indoor, outdoor, in-air and underwater applications for both manned and autonomous vehicles. It is known to be a difficult problem because it is a chicken-or-egg problem where a map is needed for localization and a pose estimate is needed for mapping.

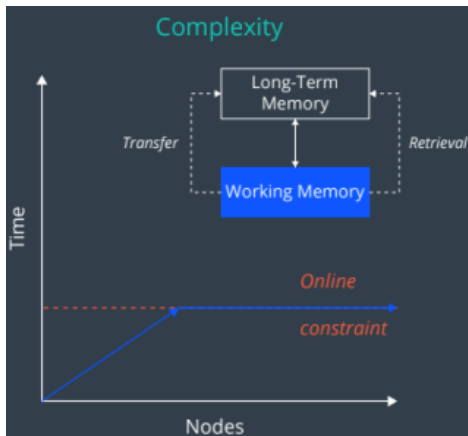
RTAB-Map (Real-Time Appearance-Based Mapping) is a RGB-D Graph SLAM approach based on a global Bayesian loop closure detector. The loop closure detector uses a bag-of-words approach to determinate how likely a new image comes from a previous location or a new location. The following image details how the features are extracted, feature descriptor are created and clustered; a vocabulary of words is set.



When a loop closure hypothesis is accepted, a new constraint is added to the map's graph, then a graph optimizer minimizes the errors in the map. RTAB-Map supports 3 different graph optimizations:

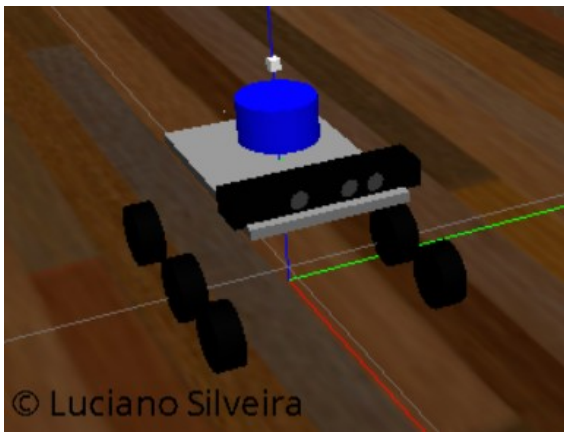
- Tree-based network optimizer, or TORO.
- General Graph Optimization, or G2O.
- GTSAM (Smoothing and Mapping).

A memory management approach is used to limit the number of locations used for loop closure detection and graph optimization, so that real-time constraints on large-scale environments are always respected.

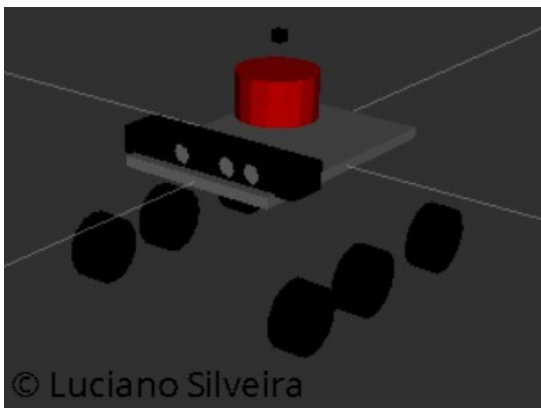


Graph-SLAM complexity is linear, according to the number of nodes, which increases according to the size of the map. By providing constraints associated with how many nodes are processed for loop closure by memory management, the time complexity becomes constant in RTAB-Map.

In our simulation environment a rover called `1s_bot` is equipped with a RGB-D camera and a Lidar sensor and is driven around two different environments. The objective is to create a 2D and 3D representation of its surroundings.



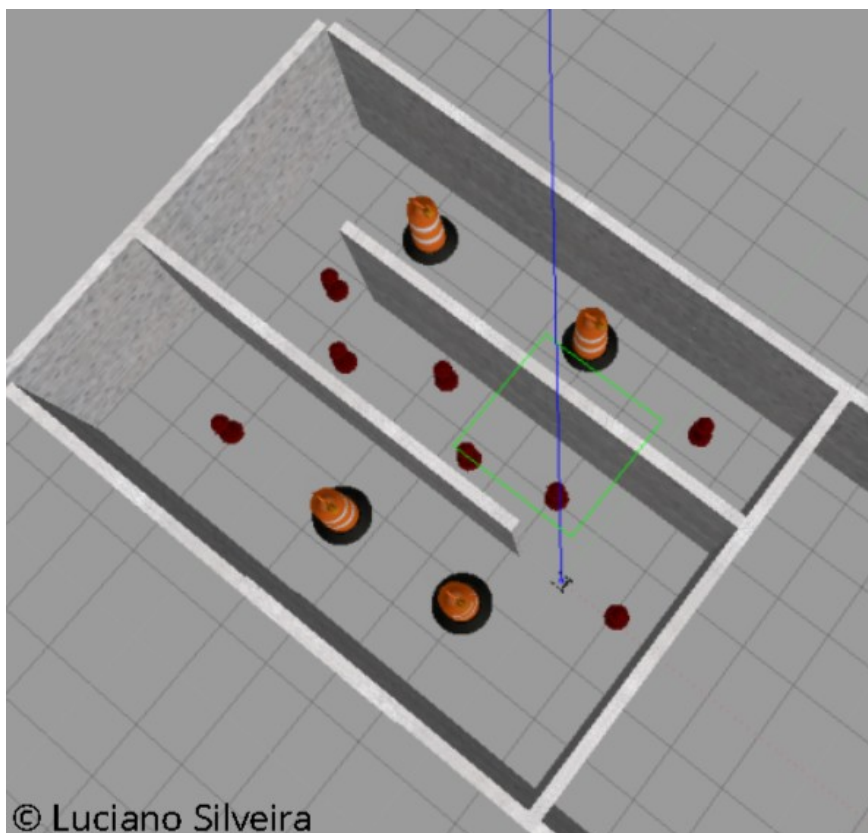
The `1s_bot` is an extension from the [rocker-bogie project](#). Several changes were needed to make to compile using the ROS Kinetic Kame default infrastructure. The autonomous rover includes a RGB-D camera, and Lidar sensors. The sensory information was added using the [Gazebo plugins reference](#). The visualization in [Rviz](#) is as follows:



using the provided environment:



The Corridor example:



The objective is to create a 2D/3D mapping of those environments using the RTAB-Map package.

## Background

SLAM is considered a fundamental problem for robots to become truly autonomous. A large variety of different SLAM approaches have been developed since mid-eighties where the majority uses probabilistic concepts.

## RTAB

**RTAB-Map**, for Real-Time Appearance-Based Mapping is a complete graph-based SLAM approach using real-time constraints. In our case we use the `rtabmap_ros` open source ROS library available since 2013. It implements `loop closure` detection with a `memory management` approach combining other techniques to ensure that the `loop closure` process happens in real time.

`Loop closure` detection is the process used to determine if the current observation comes from a previously visited location or a new one; if it recognizes an already mapped area, the robot `closes a loop`. As the size of the internal map increases, so does the time required to compare new observations with all stored locations, eventually limiting online processing. By revisiting already mapped areas, uncertainties in robot and landmark estimates are reduced.

The `memory management` approach runs on top of graph management modules. It is used to limit the size of the graph so that long-term online SLAM can be achieved in large environments. Without memory management, as the graph grows, processing time for modules like `Loop Closure` and `Proximity Detection`, `Graph Optimization` and `Global Map Assembling` can eventually exceed real-time constraints; meaning that the processing time can become greater than the node acquisition cycle time.

RTAB-Map's memory is divided into a Working Memory (WM) and a Long-Term Memory (LTM). When a node is transferred to LTM, it is not available anymore for modules inside the WM. When RTAB-Map's update time exceeds the fixed time threshold `Rtabmap/TimeThr`, some nodes in WM are transferred to LTM to limit the size of the WM and decrease the update time. Similarly to the fixed time threshold, there is also a memory threshold `Rtabmap/MemoryThr` that can be used to set the maximum number of nodes that WM can hold.

The structure of the map is a graph with nodes and links. After sensor synchronization, a Short-Term Memory (STM) module creates a node memorizing the odometry pose, sensor's raw data and additional information useful for next modules. There are three kinds of links: `Neighbor`, `Loop Closure` and `Proximity` links. `Neighbor` links are added in the STM between consecutive nodes with odometry transformation. `Loop Closure` and `Proximity` links are added through loop closure detection or proximity detection, respectively. All the links are used as constraints for graph optimization. When there is a new loop closure or proximity link added to the graph, graph optimization propagates the computed error to the whole graph, to decrease odometry drift.

## Results

---

Initially the provided `teleoperation` utility was used to move the rover but it proved difficult to control, so the strategy changed to use the [steering plugin] ([http://wiki.ros.org/rqt\\_robot\\_steering](http://wiki.ros.org/rqt_robot_steering)) from the `rqt` ROS utilities. The rover was moved around the environment to generate a map of the environment.

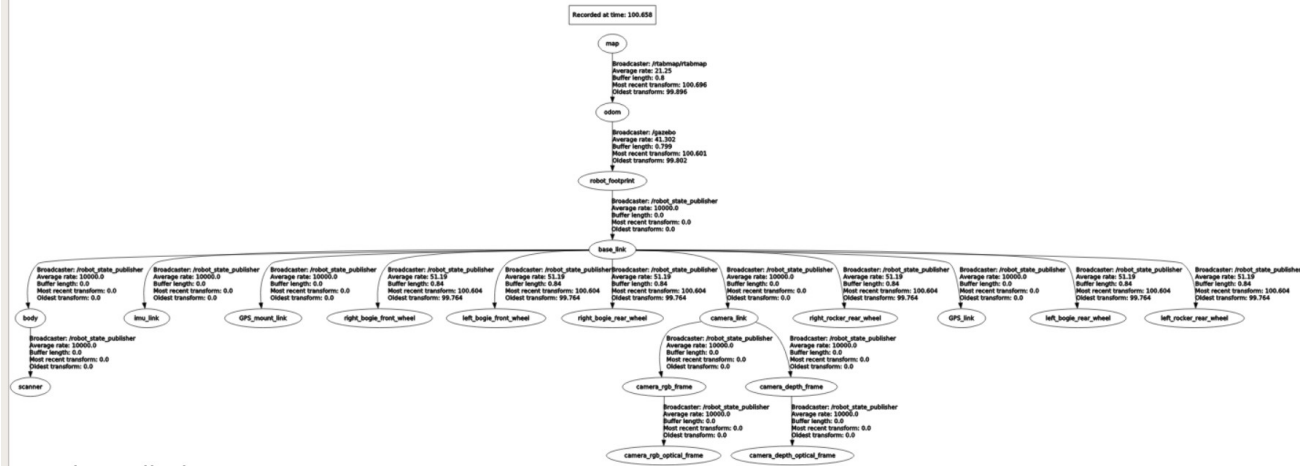
## Validation

### Frames

In order to validate all links are properly connected; the `rqt_tf_tree` utility was used:

```
roslaunch rqt_tf_tree rqt_tf_tree
```

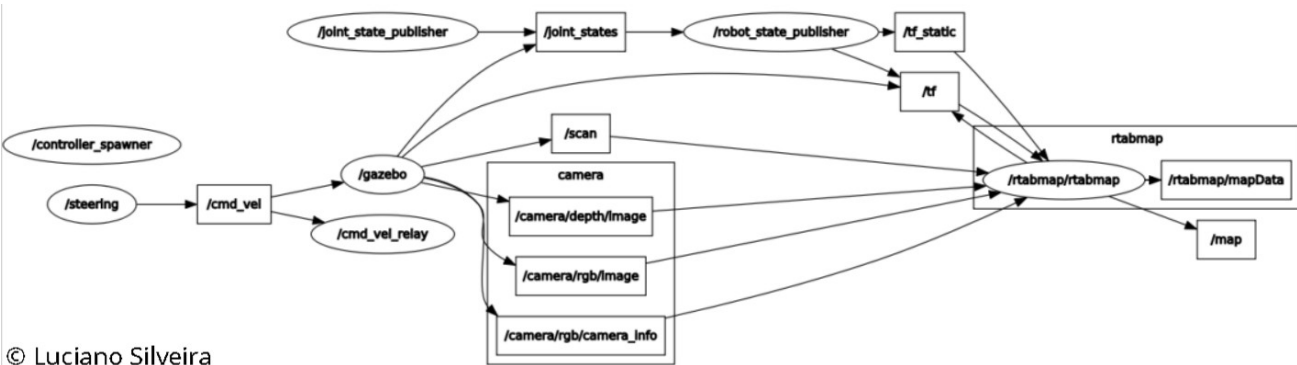
The result is as follows:



© Luciano Silveira

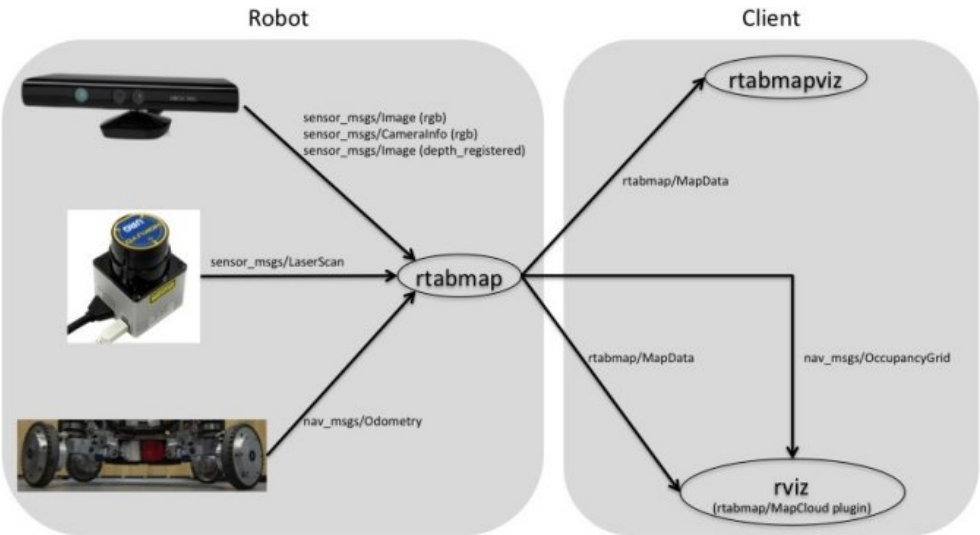
Nodes Graph

To validate topic names and connection between nodes; the [rqt\\_graph](#) utility was used with the following result:



© Luciano Silveira

The configuration is based on the following sensor configuration (already discussed):



roswtf

The [roswtf](#) utility successfully validated the configuration, the warnings and error detailed can be discarded:

```
user@machine:~/catkin_ws/ros$ roswtf
Loaded plugin tf.tfwtf
Loaded plugin openni2_launch.wtf_plugin
No package or stack in context
```

```

=====
Static checks summary:
No errors or warnings
=====
Beginning tests of your ROS graph. These may take awhile...
analyzing graph...
... done analyzing graph
running graph rules...
... done running graph rules
running tf checks, this will take a second...
... tf checks complete
Online checks summary:
Found 2 warning(s).
Warnings are things that may be just fine, but are sometimes at fault
WARNING The following node subscriptions are unconnected:
* /rtabmap/rtabmap:
  * /rtabmap/user_data_async
  * /rtabmap/initialpose
  * /rtabmap/move_base/status
  * /rtabmap/goal
  * /rtabmap/global_pose
  * /rtabmap/goal_node
  * /rtabmap/move_base/feedback
  * /rtabmap/move_base/result
* /gazebo:
  * /gazebo/set_link_state
  * /gazebo/set_model_state
* /rviz:
  * /move_base/TrajectoryPlannerROS/global_plan
  * /map_updates
  * /move_base/TrajectoryPlannerROS/local_plan
  * /rtabmap/octomap_grid_updates
WARNING These nodes have died:
* urdf_spawner-5
Found 1 error(s).
ERROR Different number of openni2 sensors found.
* 0 openni2 sensors found (expected: 1).

```

## Parameter Tuning

A valuable utility to modify node parameters is the [Dynamic Reconfigure](#) node:

```
roslaunch rqt_reconfigure rqt_reconfigure
```

## Visualization Tools

The [rtabmap-databaseViewer](#) utility was used to explore the database generated when the mapping process is finished.

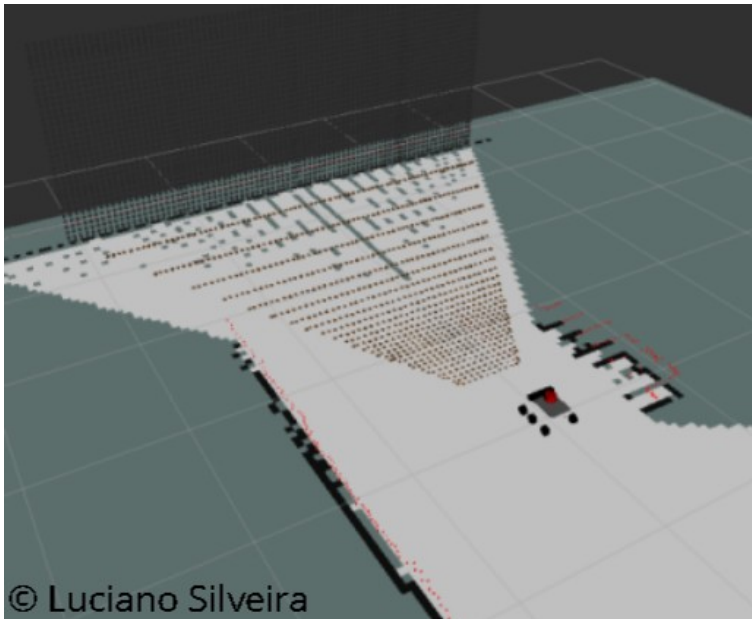
```
rtabmap-databaseViewer ~/.ros/rtabmap.db
```

Notice the detailed information for: Neighbor, Neighbor Merged, Global Loop closure, Local loop closure by space, Local loop closure by time, User loop closure, and Prior link.

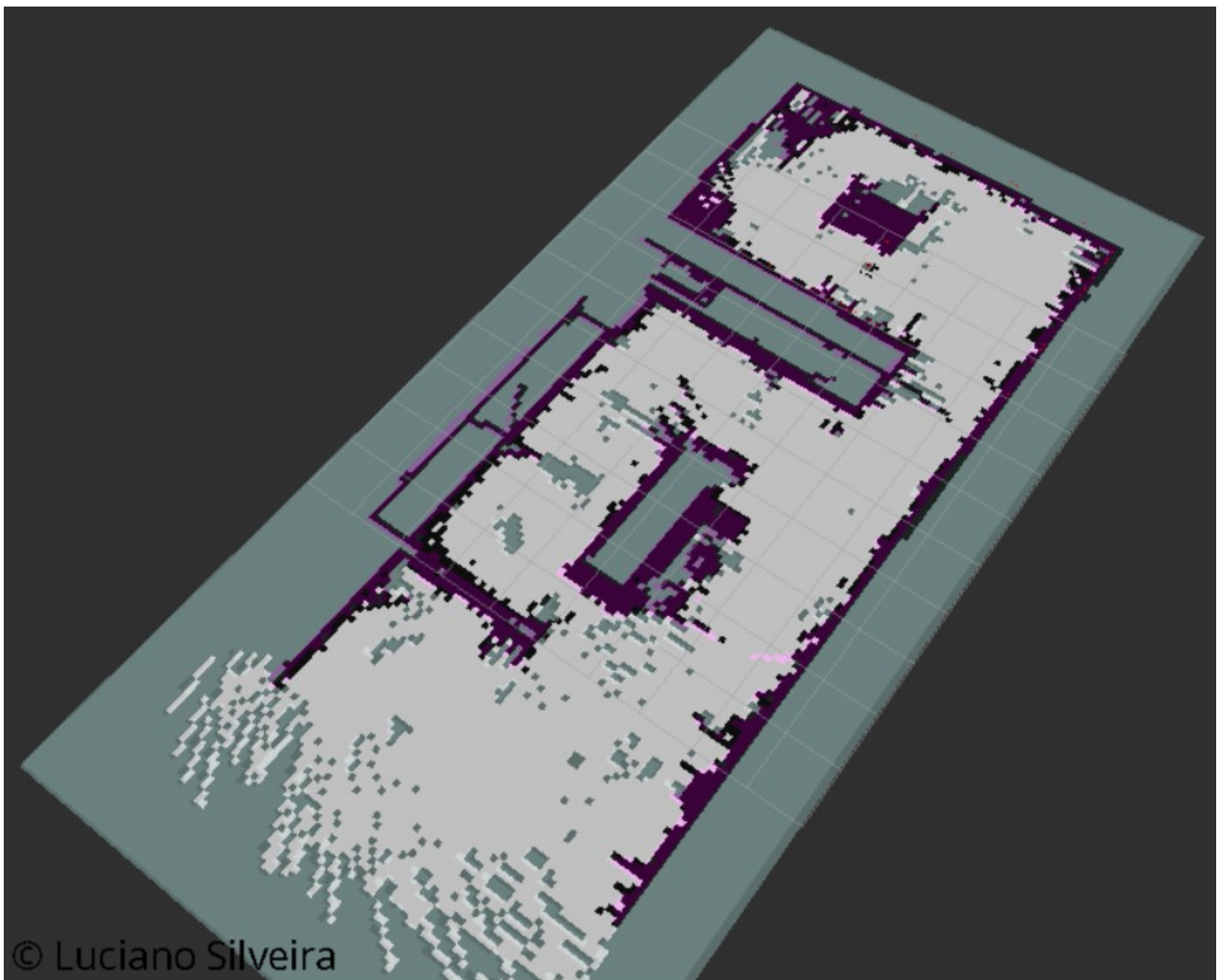
## Kitchen Sample

Once the Simulation starts, the rover detects the following environment:

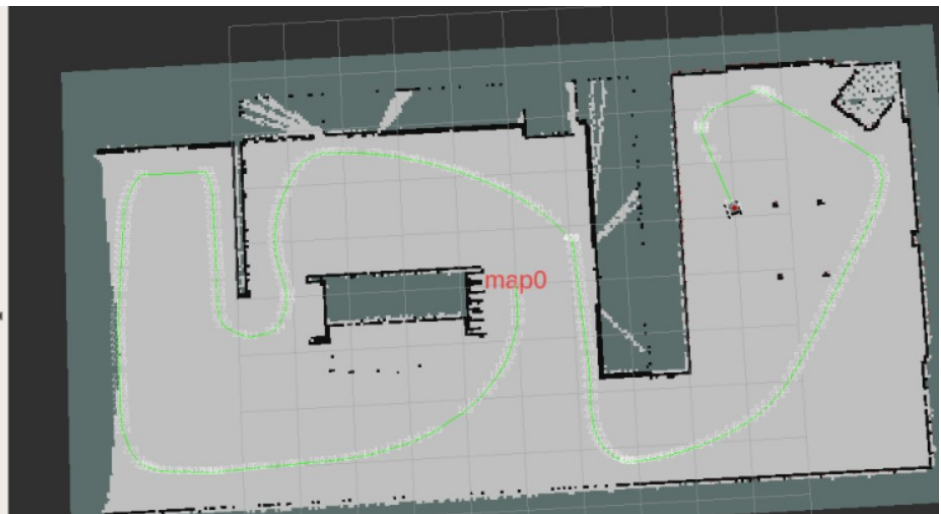
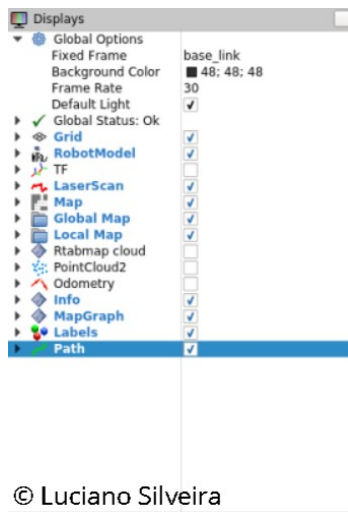




Once the Kitchen was traversed, a 2D representation is shown as follows:



Using Rviz including the traversed path:



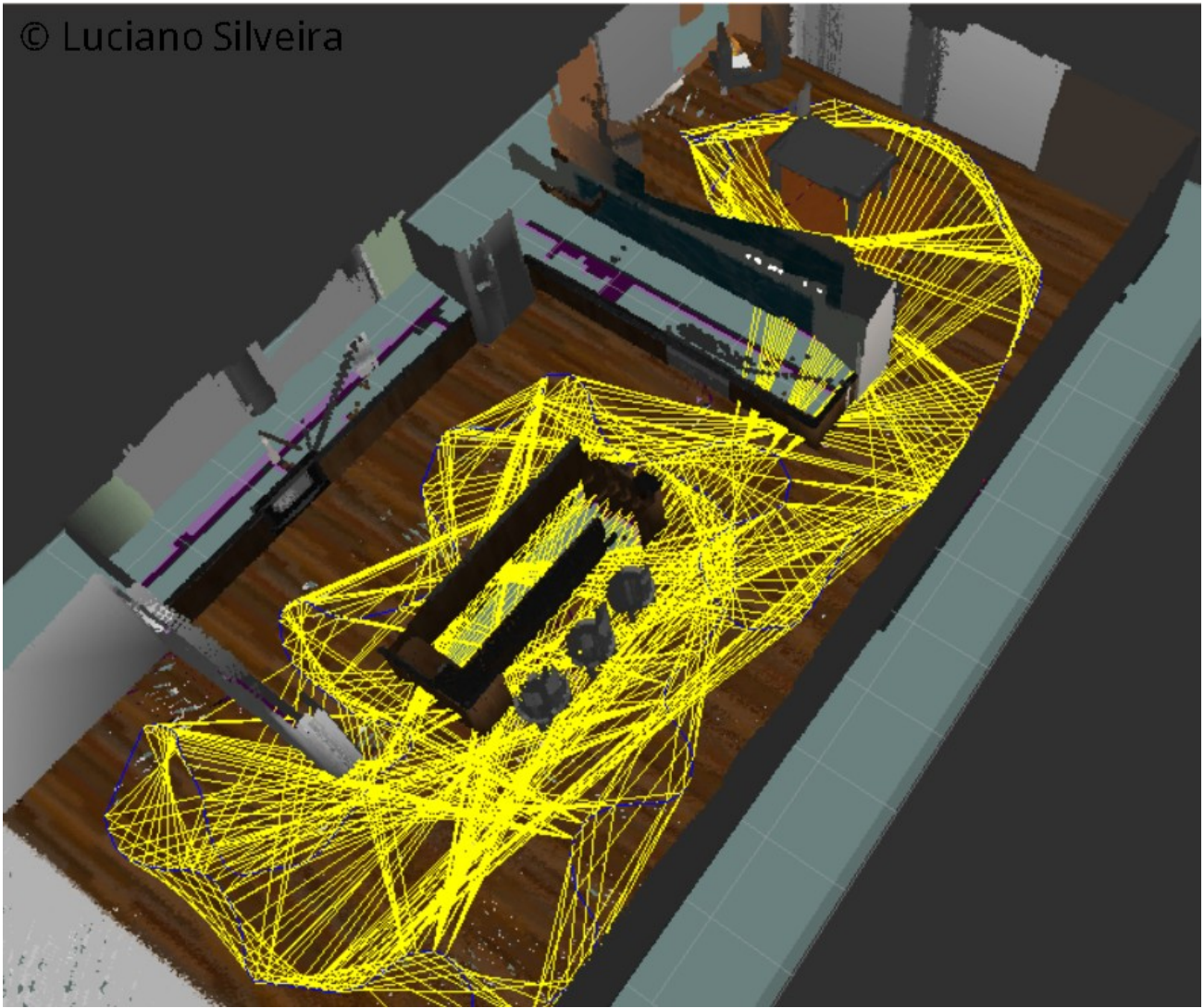
© Luciano Silveira

3D representations:





© Luciano Silveira



More detail here:

- [2D graph](#)
- [Database viewer 2D](#)
- [Rviz mapped world with path](#)

#### Database information

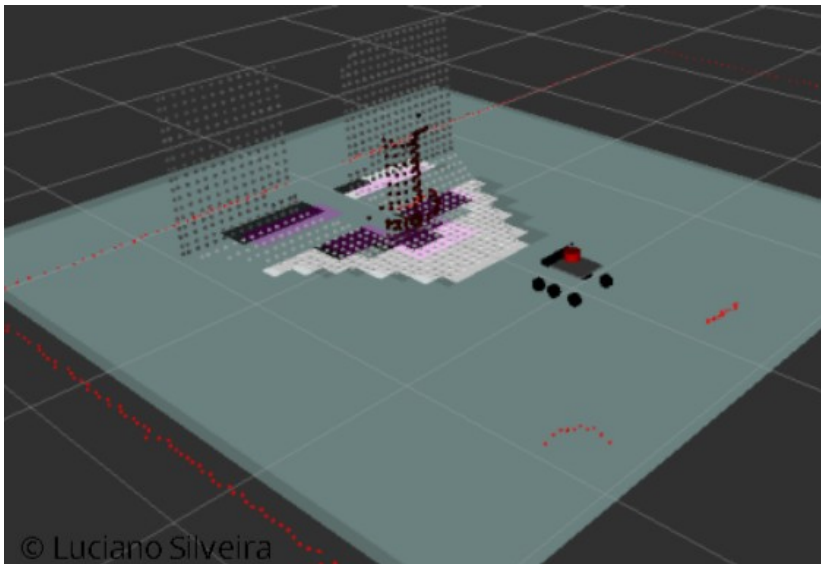
```
Version:          0.17.6
Sessions:         1
Total odometry length: 55.1824 m
Total time:       00:08:53.510
LTM:             337 nodes and 10903 words
WM:             337 nodes and 10903 words
Global graph:    337 poses and 1008 links
Ground truth:    0 poses
GPS:             0 poses
```

```
Database size: 163 MB
Nodes size:    42 KB   0.03%
Links size:    730 KB  0.45%
RGB Images size: 11 MB  6.85%
Depth Images size: 127 MB 78.12%
Calibrations size: 24 KB 0.01%
Grids size:    1 MB   1.14%
Scans size:    833 KB  0.51%
User data size: 0 Bytes 0.00%
Dictionary size: 3 MB   1.87%
```

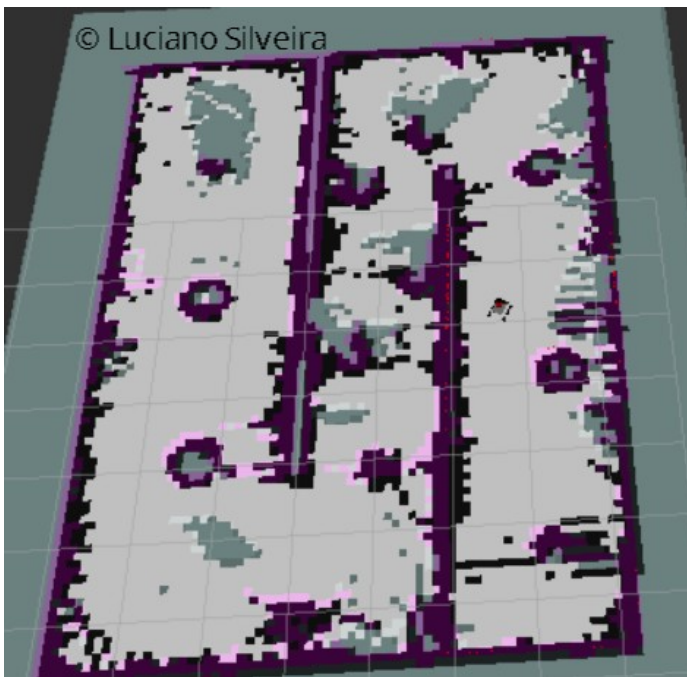
Features size:	12 MB	7.49%
Statistics size:	0 Bytes	0.00%
Other (indexing):	5 MB	3.53%

## Corridor Sample

Once the Simulation starts, the rover detects the following environment:



A 2D representation for the Corridor world once it was traversed:



3D representations:



More detail here:

- [Rviz 3D representation](#)
- [Corridor final mapped world](#)
- [Database viewer 2D representation](#)

Database information



```
Version:          0.17.6
Sessions:         1
Total odometry length: 54.3411 m
Total time:       00:08:40.810
LTM:             389 nodes and 52272 words
WM:             389 nodes and 52272 words
Global graph:    389 poses and 698 links
Ground truth:    0 poses
GPS:            0 poses

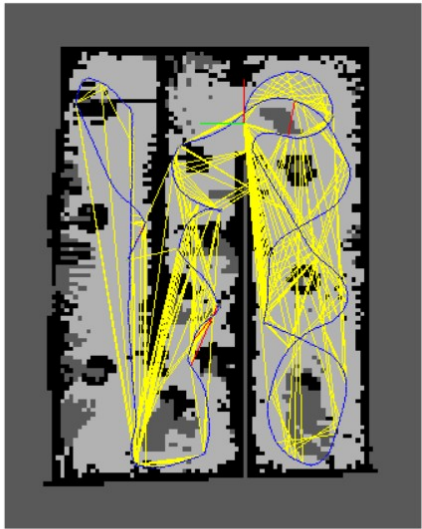
Database size: 342 MB
Nodes size:     49 KB  0.01%
Links size:     496 KB 0.14%
RGB Images size: 18 MB  5.48%
Depth Images size: 184 MB 53.75%
Calibrations size: 28 KB 0.01%
Grids size:     2 MB  0.66%
Scans size:     1 MB  0.32%
User data size: 0 Bytes 0.00%
Dictionary size: 14 MB  4.31%
Features size:  85 MB 24.84%
Statistics size: 0 Bytes 0.00%
Other (indexing): 35 MB 10.47%
```

The final result from the generated database:

© Luciano Silveira

RTAB-Map Database Viewer\*

Graph view



20

1

Root [1, 490]

Span to all maps

54.4136

Path length (m)

0.073

Time optimization (s)

0.011

Time grid (s)

389

Poses

(388, 0, 10, 300, 0, 0, 0)

Links (N, NM, G, LS, LT, U, P)

Parents 39 119 120 121 122 123 124 125 126 127

Children

Weight 0

Label map0

Map ID 0

Pose xyz=(0.000146495,-0.000408508,0.125 rpy=(4.93858e-07,2.64468e-07,-0.0006

Velocity vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw

Stamp 453.630000

Calib 1 640x480 fx=554.383 fy=554.383 cx=

Parents 39 119 120 121 122 123 124 125 126 127

Children

Weight 0

Label map0

Map ID 0

Pose xyz=(0.000146495,-0.000408508,0.125 rpy=(4.93858e-07,2.64468e-07,-0.0006

Velocity vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw

Stamp 453.630000

Calib 1 640x480 fx=554.383 fy=554.383 cx=

Index : 0

Id : 1

Index : 0

Id : 1

Models

The generated databases can be downloaded from [here](#).

Unzip the files and execute a command similar to:

```
rtabmap-databaseViewer ~/corridor01.db
rtabmap-databaseViewer ~/kitchen01.db
```

## Mapping

To execute the [mapping](#) step use the [kitchen.launch](#) file:

```
<!-- Kitchen -->
<include file="$(find slam_project)/launch/world.launch" />
<include file="$(find slam_project)/launch/teleop.launch" />
<include file="$(find slam_project)/launch/mapping.launch" />
```

## Localization

In order to use the generated map for [localization](#), the following changes were done:

- Duplicate the mapping.launch file as localization.launch
  - Remove the args="--delete\_db\_on\_start" from the launcher file.
  - Remove the Mem/NotLinkedNodesKept parameter
  - add the Mem/IncrementalMemory parameter of type string and set it to false.

## Extra Launch files

For the evaluation of the map generated, it is possible to execute the [rtabmapviz.launch](#) file:

```
roslaunch slam_project rtabmapviz.launch
```

For evaluation of the database in offline mode once finished the mapping session:

```
rtabmap-databaseViewer ~/.ros/rtabmap.db
```

## Discussion

The [depthimage\\_to\\_laserscan](#) package for the Kinect RGB-D sensor was discarded as the project was run on a Virtual machine. All parameters were modified so as to consume as little processing as needed. Instead a Lidar sensor was added to improve the Odometry provided by the Gazebo simulator.

Nodes are created at a fixed rate `Rtabmap/DetectionRate` of 1 second according to how much data created from nodes should overlap each other.

The initial [Parameters](#) and [Advanced configuration](#) was checked:

### Initial

- Reg/Force3DoF set to true.
- Kp/DetectorStrategy using SURF Loop Closure Detection.
- Kp/MaxFeatures maximum visual words per image (bag-of-words) set to 400.
- SURF/HessianThreshold to extract more or less SURF features to 100.
- Reg/Strategy Loop Closure Constraint tested with Visual and ICP; ICP selected as the map got better.
- Vis/MinInliers minimum visual inliers to accept loop closure set to 15.

- `NotLinkedNodesKept` set to false to avoid saving data when robot is not moving.

## Advanced configuration

- `queue_size` set to 10.
- `wait_for_transform_duration` changed its default value from 0.1 to 0.2 to be more tolerant to problems.
- `param name="RGBD/NeighborLinkRefining` set to `true` to correct odometry using the input laser topic using ICP.
- `RGBD/ProximityBySpace` set to `true`.
- `RGBD/AngularUpdate` set to 0.05.
- `RGBD/LinearUpdate` set to 0.05.
- `RGBD/OptimizeFromGraphEnd` set to `false`.
- `Grid/3D` enabled.
- `Grid/CellSize` set its default value from 0.05 to 0.1.
- `Grid/FromDepth` set to `true`.

## Troubleshooting

### Collisions

The provided world was not detecting collisions and thus the Lidar sensor didn't provide any information, the `scan` topic was empty. The problem was that the default Gazebo model lacked the `collision` tag in its [SDF](#) definition. The solution was found using this [thread](#) on the slack community.

```
<collision name="collision">
  <geometry>
    <mesh><uri>model://kitchen_dining/meshes/kitchen_dining.dae</uri></mesh>
  </geometry>
</collision>
```

and updating the local gazebo folder

```
curl -L https://s3-us-west-1.amazonaws.com/udacity-robotics/Term+2+Resources/P3+Resources/models.tar.gz | tar
```



### Octomap projection

In order to create a map based on Octomap the following properties were enabled:

- `Grid/3D`
- `Grid/FromDepth`

otherwise the following error appears:

```
Octomap projection map is empty! (poses=340 octomap nodes=0). Make sure you activated "Grid/3D" and
"Grid/FromDepth" to true.
See "$ rosrun rtabmap_ros rtabmap --params | grep Grid" for more info.
```

## Discussion

It was possible to generate maps for both generated environments. In some cases it was difficult to get an accurate map as the rover needed to move very slowly particularly trying not to rotate very fast. Several parameters related to resolution were modified to lower the power processing and the memory used.

## Conclusion / Future Work



RTAB-Map is a multi-purpose graph-based SLAM approach which can be used out-of-the-box, it supports different sensor configurations and processing capabilities. It can be used without a deep understanding on its internals. It seems to be a reasonable solution for SLAM to develop robots that can map environments in 2D/3D and low-cost sensors such as RGB-D Camera, stereo cameras or combined with a Lidar.

The drawback found is that the environments must be feature-rich enough to make global loop closures, otherwise the generated maps are not useful.

The following step would be to apply it on a real world environment and validate if the generated map is reasonable and good enough for navigation.

### Links:

- [Initial Resources](#)
- [This repository](#)
- [Project Rubric](#)
- [Gazebo](#)
- [Gazebo Model creation](#)
- [RViz](#)
- [Gazebo plugins](#)
- [rtabmap](#)
- [rtabmap\\_ros](#)
- [SetupOnYourRobot withb rtabmap](#)
- [What is SLAM?](#)
- [rtabmap parameters](#)
- [RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation](#)