Map My World Robot Project Starter Code

Abhay Luna

The objective of this project is to create a 2D occupancy grid and 3D octomap from a provided simulated environment. In addition, a simulated environment will be created to map as well.

How to run the program with your own code

Go to Desktop and open a terminal.

For the execution of your own code, we head to the Project Workspace. For this setup, catkin_ws is the name of the active ROS workspace. If your workspace name is different, change the commands accordingly.

If you do not have an active ROS workspace, you can create one. You can launch it by running the following commands first.

```
cd /home/workspace
mkdir -p catkin_ws/src
cd /home/workspace/catkin_ws
catkin_make
source devel/setup.bas
```

Clone the required repositories to the `/home/workspace/catkin_ws/src` folder

```
cd /home/workspace/catkin_ws/src
git clone https://github.com/Abhaycl/RoboND-Map-My-World-Robot-2P3.git
```

Now install missing dependencies using rosdep install:

```
rosdep install -i slam_bot
```

Build the project:

```
cd /home/workspace/catkin_ws
catkin_make
```

Download the proper kitchen dining room `*.world` file that includes collision models required to map properly:

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

Open three terminals to the `/home/workspace/catkin_ws/src/slam_bot/launch` folder and run each command in one terminal:

```
roslaunch slam_bot world.launch world_name:=kitchen_dining.world
roslaunch slam_bot teleop.launch
roslaunch slam_bot mapping.launch
```

Commands to be executed for the test section

To generate the robot transform tree frames, you have to execute the following command:

```
rosrun tf view_frames
```

To use the database created by the Kitchen World you have to copy the .db file below:

```
cp /home/workspace/catkin_ws/src/slam_bot/db/kitchen/rtabmap.db ~/.ros
```

To use the database created by the Custom World you have to copy the .db file below:

```
cp /home/workspace/catkin_ws/src/slam_bot/db/custom/rtabmap.db ~/.ros
```

Execute the following command to explore the database:

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

The summary of the files and folders within repo is provided in the table below:

File/Folder	Definition
config/*	Folder that contains all the parameters and some values defined for you to help you get started rviz.
db/*	Folder that contains all the databases generated from the maps.
launch/*	Folder that contains all the launch files in ROS that allow us to execute more than one node simultaneously.
meshes/*	Folder that contains all the parameterization of the sensors.
scrips/*	Folder that contains all the scripts for the execution of the different necessary tools.
urdf/*	Folder that contains all the robot's URDF description.
worlds/*	Folder that contains all the Gazebo worlds.
misc_images/*	Folder containing the images of the project.
CMakeLists.txt	Contains the System dependencies that are found with CMake's conventions.
frames.gv	Contains the robot transform tree frames.
frames.pdf	Contains the robot transform tree frames in PDF format.
package.xml	Contains the slam_bot package.
README.md	Contains the project documentation.
README.pdf	Contains the project documentation in PDF format.

Steps to complete the project:

1. You will develop your own package to interface with the rtabmap_ros package.
2. You will build upon your localization project to make the necessary changes to interface the robot with RTAB-Map. An example of this is the addition of an RGB-D camera.
3. You will ensure that all files are in the appropriate places, all links are properly connected, naming is properly setup and topics are correctly mapped. Furthermore you will need to generate the appropriate launch files to launch the robot and map its surrounding environment.
4. When your robot is launched you will teleop around the room to generate a proper map of the environment.
5. You then will build your own simulated environment and apply the code you used to map the supplied environment on this environment.
6. Finally you will create a report and organize the required files for submission.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Abstract

This project aims to successfully perform SLAM leverage GraphSLAM to generate 2D occupancy grids and 3D octomaps of two simulated environments in Gazebo. A two-wheel robot equipped with RGB-D and 2D Lidar sensors (configured as a camera and a Hokuyo lidar, respectively) was simulated to traverse two Gazebo world environments, one provided in this practice, and another that was custom built. The robot maps the environment using the "Real Time Appearance Based Mapping" (RTAB-Map) GraphSLAM approach. After the robot traversed both environments, maps were evaluated on accuracy.

Introduction

For a robot to be fully autonomous it must adapt and understand its environment as it changes and be able to recover when it obtains noisy data from sensors. Simultaneous Localization and Mapping (SLAM) provides a robot with the capability to dynamically map its environment as it relates to it. When implemented correctly, SLAM solves the localization problem to map the environment, and maps the environment to solve the localization problem. This is all done as an iterative cycle.

In this project, two 3D world environments are to be simulated, tested, and mapped using SLAM. The environment provided, and set to be the benchmark, is called kitchen dining. Another environment shall be designed and developed in Gazebo and saved in world format; this one is called my world. The two environments shall be mapped in both 2D and 3D by using Gazebo and Rviz simulation frameworks.

Background / Formulation

Given that a map is needed for localization, but at the same time the robots location and orientation is needed for mapping, this has often been considered the chicken or the egg problem. SLAM is a challenging problem, as it has to quickly solve two related problems simultaneously and back-to-back. There are many approaches to perform SLAM such as:

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

The two most commonly used approaches in robotics are FastSLAM and GraphSLAM. Both of these, solve the SLAM problem well, although in different ways.

FastSLAM

The FastSLAM algorithm can solve the full SLAM problem with known correspondences. Using a custom particle filter approach along with a low dimensional Extended Kalman Filter, FastSLAM is able to estimate the trajectory of the robot which will allow it to map the environment concurrently.

GraphSLAM

GraphSLAM is a SLAM algorithm that solves the Full SLAM Problem. This algorithm recovers the entire path and map, which allows it to consider dependencies between current and previous poses. GraphSLAM has a few advantages over FastSLAM. GraphSLAM improves upon the need of onboard processing capability, while still improving accuracy over FastSLAM. Since GraphSLAM is able to retain information from past locations, it proves an advantage over FastSLAM which uses less information and has a finite number of particles.

RTAB-Mapping

The Real Time Appearance Based Mapping algorithm is a GraphSLAM approach and will be used in this project to perform SLAM. This algorithm uses data collected from sensors to localize the robot and map the environment. In RTAB-Map a process called loop closure is used to allow the robot to determine if the location has been observed before. While the robot continues to traverse through its environment the map continues to grow. For other Appearance-Based methods, the robot continues to compare new images to past images to identify whether it has been at that location before. This, over time, increases the number of images for comparison, causing the loop closure process to take longer, proportionally increasing complexity along the way. However, RTAB-Map is optimized for large-scale and long-term SLAM, allowing loop closure to be processed fast enough to be known in real-time without dramatically affecting performance.

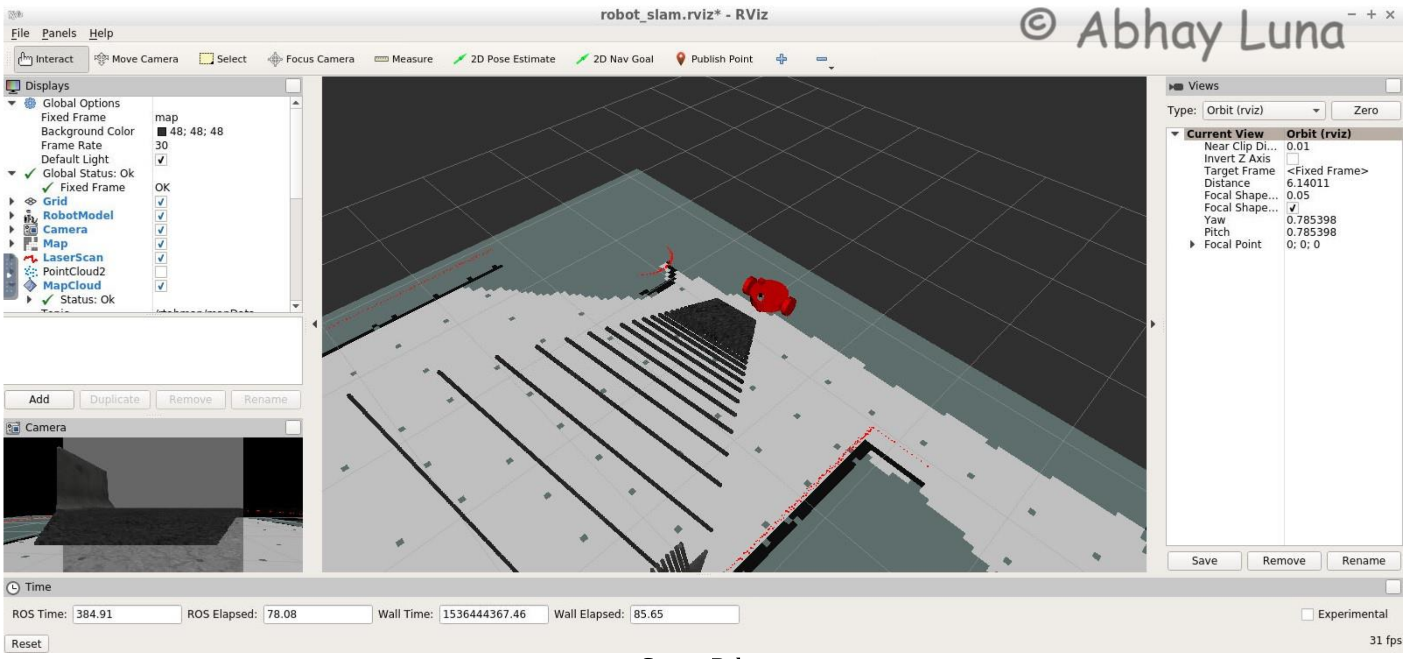
There are numerous challenges for SLAM algorithms to address. As with any robotics application, large amounts of noise may introduce irreparable error into the algorithm. Specifically for SLAM applications, algorithms will tend to have difficulty with large environments that have multiple areas that appear similar.

These produce incorrect consolidations of the created map. Proper tuning of algorithm parameters will mitigate this issue.Additionally, some algorithms may reach memory management issues. This is more prevalent in larger environments. The required memory to accurately map the environment may become costly for the algorithm to maintain. This issue can be addressed architecturally by selecting the most suited SLAM method for the robot and environment. Environments with unique landmarks and smaller spaces are mapped more easily.

Scene and robot configurations

Robot Model

The robot used for this project It's new build to solve the localization problem in a previous project, which has a cylindrical shape, with two cylindrical wheels, and to spherical casters. This robot has been repurposed to perform SLAM by changing its regular RGB camera for an RGB-D sensor camera, which allows it to detect the depth of its environment. This, along with its existing 2D laser rangefinder (Hokuyo) sensor, allows the robot ROS package slambot to leverage the rtabmap-ros package to perform SLAM.

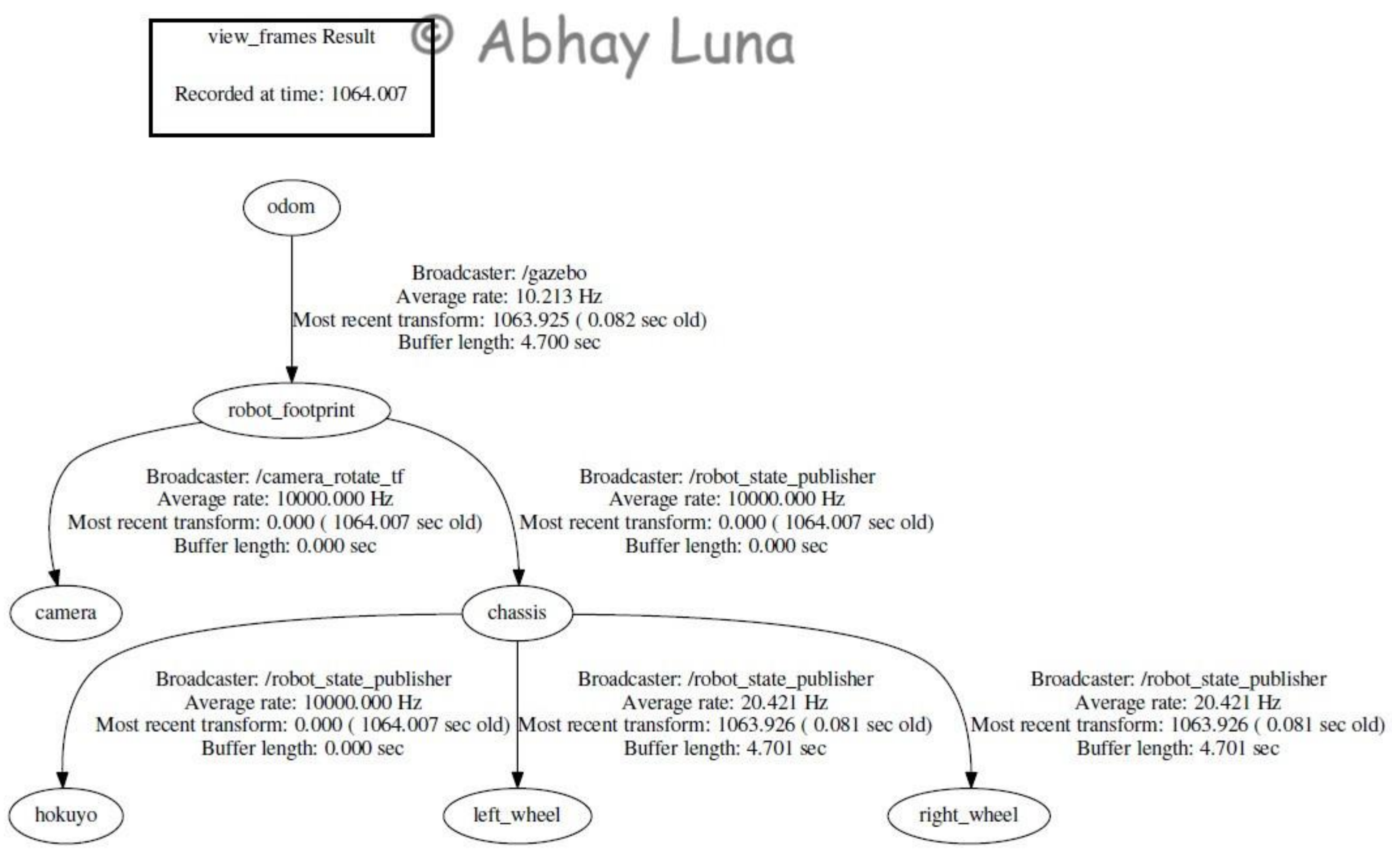


```
Global Options
  Fixed Frame          map
  Background Color     48; 48; 48
  Frame Rate           30
  Default Light        ✓
Global Status: Ok
  Fixed Frame          OK
Grid                   ✓
RobotModel             ✓
Camera                 ✓
Map                    ✓
LaserScan              ✓
PointCloud2
MapCloud               ✓
  Status: Ok

Type: Orbit (rviz)
Current View           Orbit (rviz)
  Near Clip Di...      0.01
  Invert Z Axis
  Target Frame         <Fixed Frame>
  Distance             6.14011
  Focal Shape...       0.05
  Focal Shape...       ✓
  Yaw                  0.785398
  Pitch                0.785398
  Focal Point          0; 0; 0

ROS Time: 384.91   ROS Elapsed: 78.08   Wall Time: 1536444367.46   Wall Elapsed: 85.65
31 fps
```

Custom Robot

Since manual mapping is to be performed the slam-bot package communicates with a teleop package that allows the user to move and steer the robot.

The backend configuration of the robot can be better appreciated in the Transform tree.


view_frames Result
Recorded at time: 1064.007
© Abhay Luna
odom
Broadcaster: /gazebo
Average rate: 10.213 Hz
Most recent transform: 1063.925 (0.082 sec old)
Buffer length: 4.700 sec
robot_footprint
Broadcaster: /camera_rotate_tf
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1064.007 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1064.007 sec old)
Buffer length: 0.000 sec
camera
chassis
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1064.007 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 20.421 Hz
Most recent transform: 1063.926 (0.081 sec old)
Buffer length: 4.701 sec
Broadcaster: /robot_state_publisher
Average rate: 20.421 Hz
Most recent transform: 1063.926 (0.081 sec old)
Buffer length: 4.701 sec
hokuyo
left_wheel
right_wheel

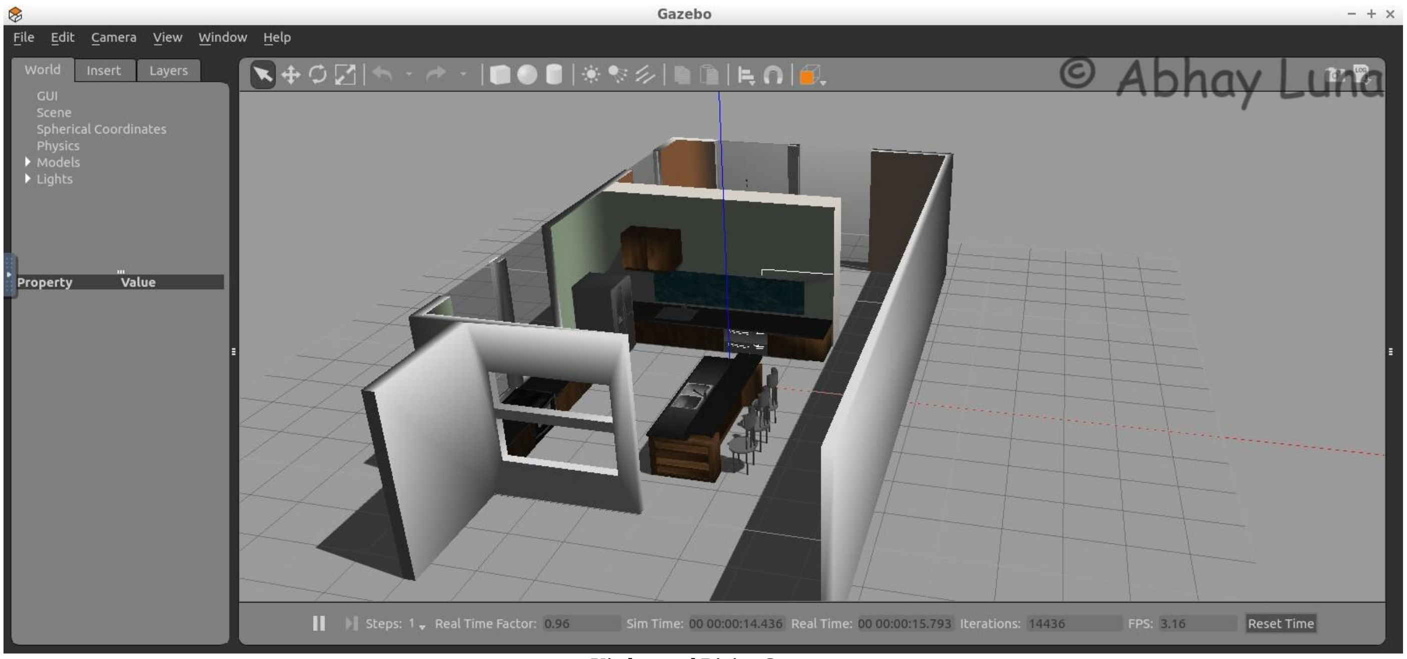
Robot TF Frames

Scene Configuration

In order for the robot to perform SLAM, it must have a World to perform it in. For this reason, two environments will be used to test and simulate a robot performing SLAM and to generate 2D and 3D maps from them.

Kitchen and Dining Scene

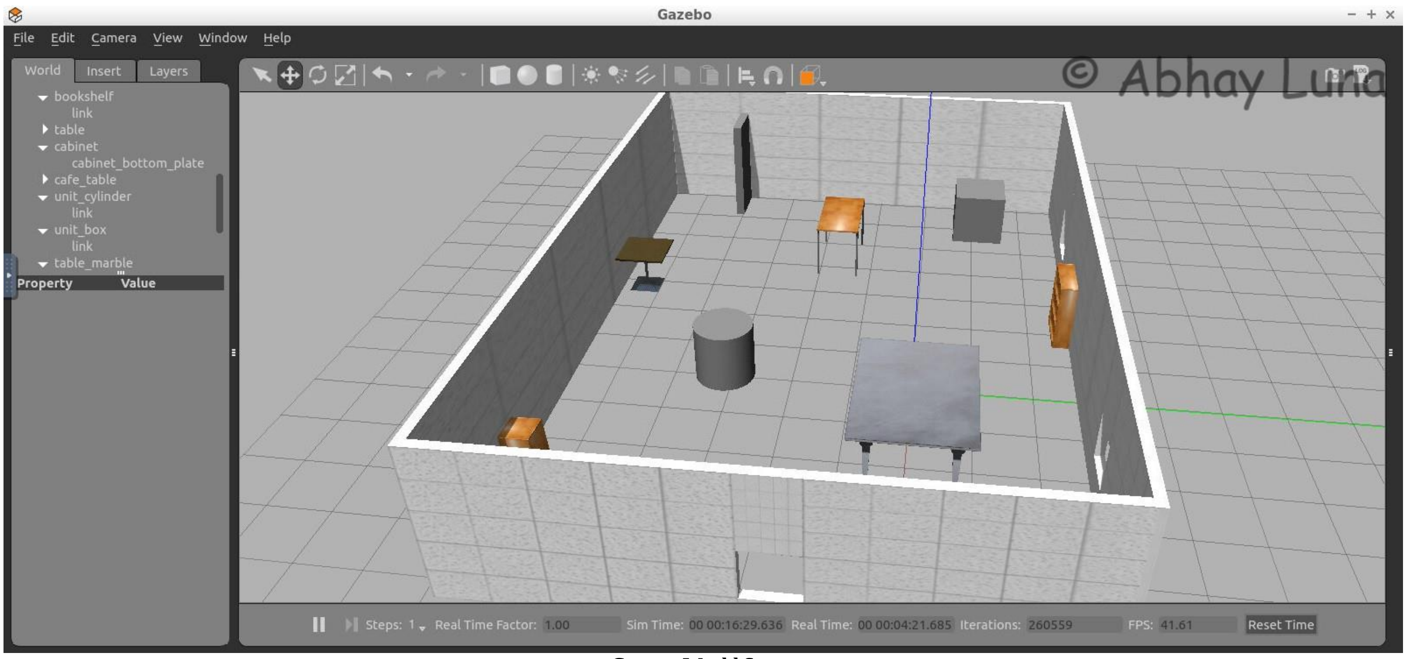
The Kitchen and Dining Scene world was provided and will be used as a benchmark for experimentation purposes in the image below.


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Property Value
© Abhay Luna
Steps: 1 Real Time Factor: 0.96 Sim Time: 00 00:00:14.436 Real Time: 00 00:00:15.793 Iterations: 14436 FPS: 3.16 Reset Time

Kitchen and Dining Scene

My Custom World Scene

The my custom world scene was designed in Gazebo to be used as the experimental environment to deploy a robot for performing SLAM. This scene was developed with the RTAB-Map Appearance Based feature in mind. Therefore, this scene is dramatically feature-rich to help the robot easily identify where it is based on appearance, and map the environment more accurately. This scene includes a tables, boxes, closet and even floor colors that allow the robot to detect more features.


Gazebo
File Edit Camera View Window Help
World Insert Layers
bookshelf
link
table
cabinet
cabinet_bottom_plate
cafe_table
unit_cylinder
link
unit_box
link
table_marble
Property Value
© Abhay Luna
Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:16:29.636 Real Time: 00 00:04:21.685 Iterations: 260559 FPS: 41.61 Reset Time

Custom World Scene

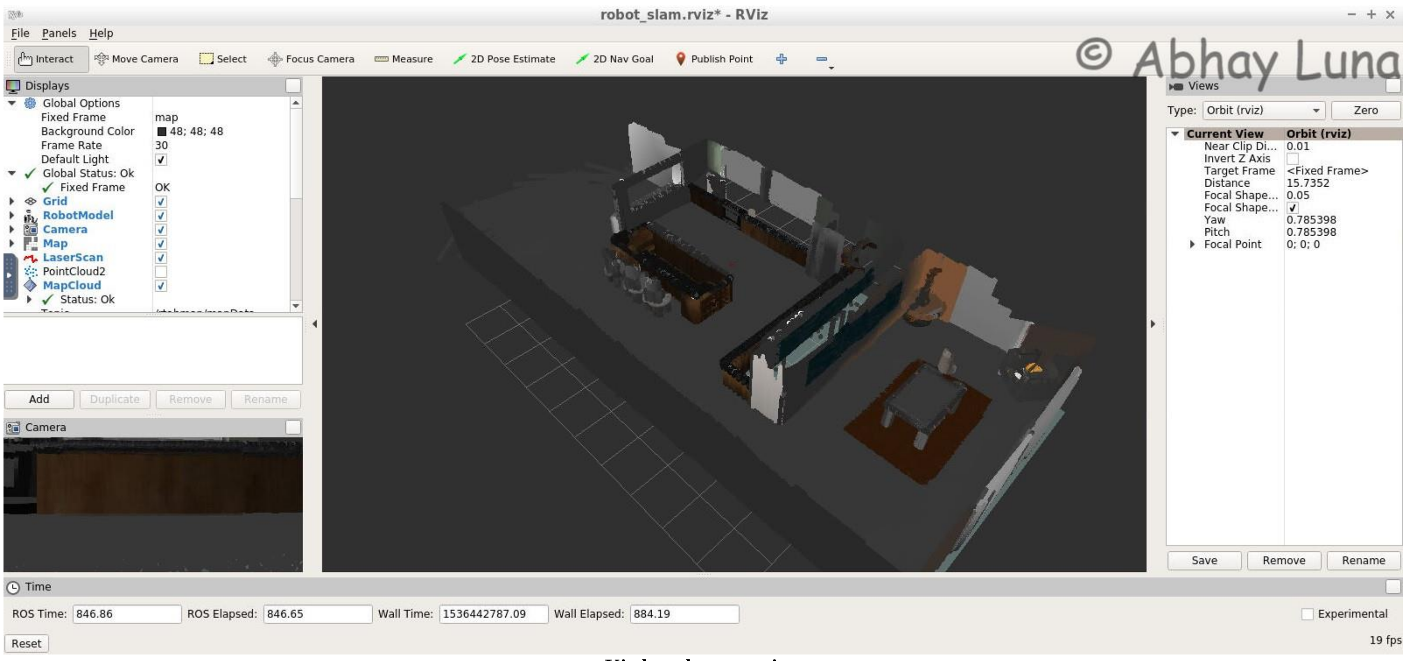
Results

Kitchen and Dining Scene

Upon launching the Kitchen and Dining scene simulation, and launching the mapping node, the slam-bot started mapping the environment in its immediate vicinity.

After traversing and navigating around the entire world, the slam-bot was able to generate an accurate map of the world that was provided.

As can be seen from the rtabmap database, several loop closures were observed, and bot a 2D occupancy grid map and a 3D octomap were generated for the provided environment.



Displays

Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	
Fixed Frame	OK
Grid	✓
RobotModel	✓
Camera	✓
Map	✓
LaserScan	✓
PointCloud2	
MapCloud	✓
Status: Ok	

Views

Type: Orbit (rviz) Zero

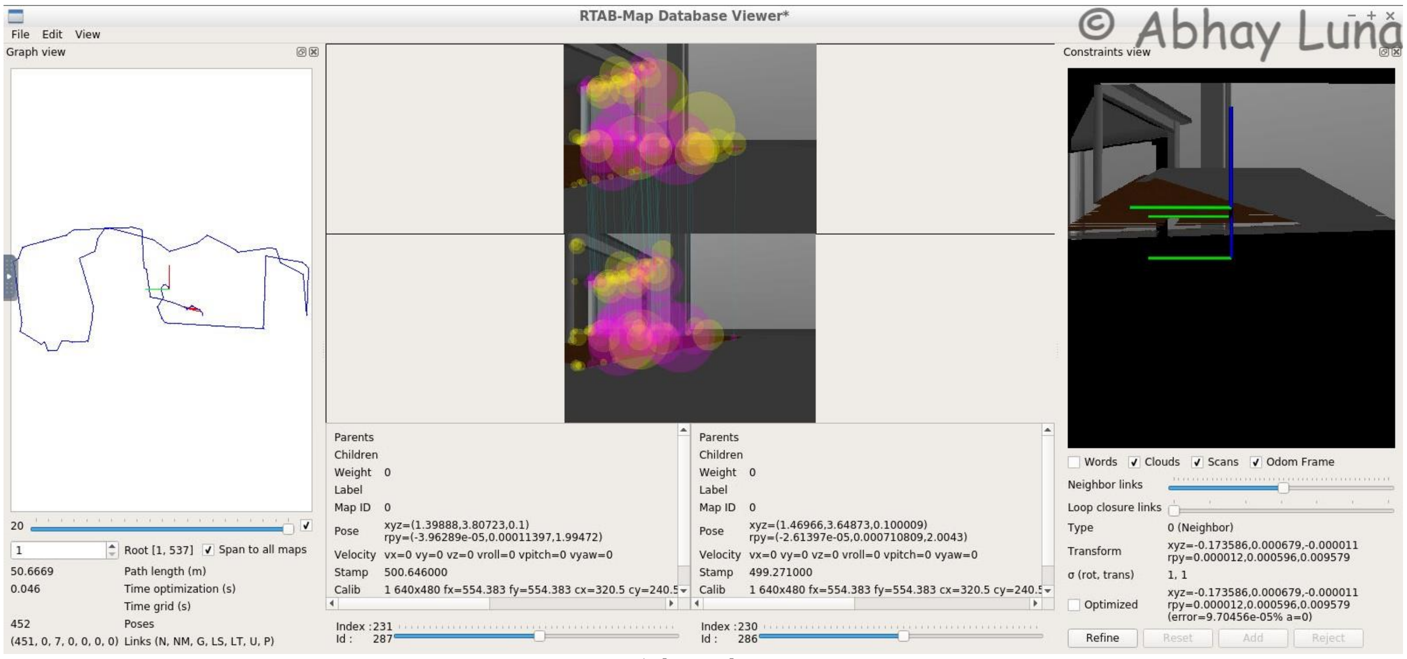
Current View	Orbit (rviz)
Near Clip Di...	0.01
Invert Z Axis	
Target Frame	<Fixed Frame>
Distance	15.7352
Focal Shape...	0.05
Focal Shape...	✓
Yaw	0.785398
Pitch	0.785398
Focal Point	0; 0; 0

Time

ROS Time: 846.86 ROS Elapsed: 846.65 Wall Time: 1536442787.09 Wall Elapsed: 884.19 Experimental

19 fps

Kitchen slam mapping



20 ✓

1 Root [1, 537] ✓ Span to all maps

50.6669	Path length (m)
0.046	Time optimization (s)
	Time grid (s)
452	Poses
(451, 0, 7, 0, 0, 0, 0)	Links (N, NM, G, LS, LT, U, P)

Parents	
Children	
Weight	0
Label	
Map ID	0
Pose	xyz=(1.39888,3.80723,0.1) rpy=(-3.96289e-05,0.00011397,1.99472)
Velocity	vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp	500.646000
Calib	1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=240.5

Index :231
Id : 287

Parents	
Children	
Weight	0
Label	
Map ID	0
Pose	xyz=(1.46966,3.64873,0.100009) rpy=(-2.61397e-05,0.000710809,2.0043)
Velocity	vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp	499.271000
Calib	1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=240.5

Index :230
Id : 286

Words ✓ Clouds ✓ Scans ✓ Odom Frame

Neighbor links

Loop closure links

Type	0 (Neighbor)
Transform	xyz=-0.173586,0.000679,-0.000011 rpy=0.000012,0.000596,0.009579
σ (rot, trans)	1, 1
Optimized	xyz=-0.173586,0.000679,-0.000011 rpy=0.000012,0.000596,0.009579 (error=9.70456e-05% a=0)

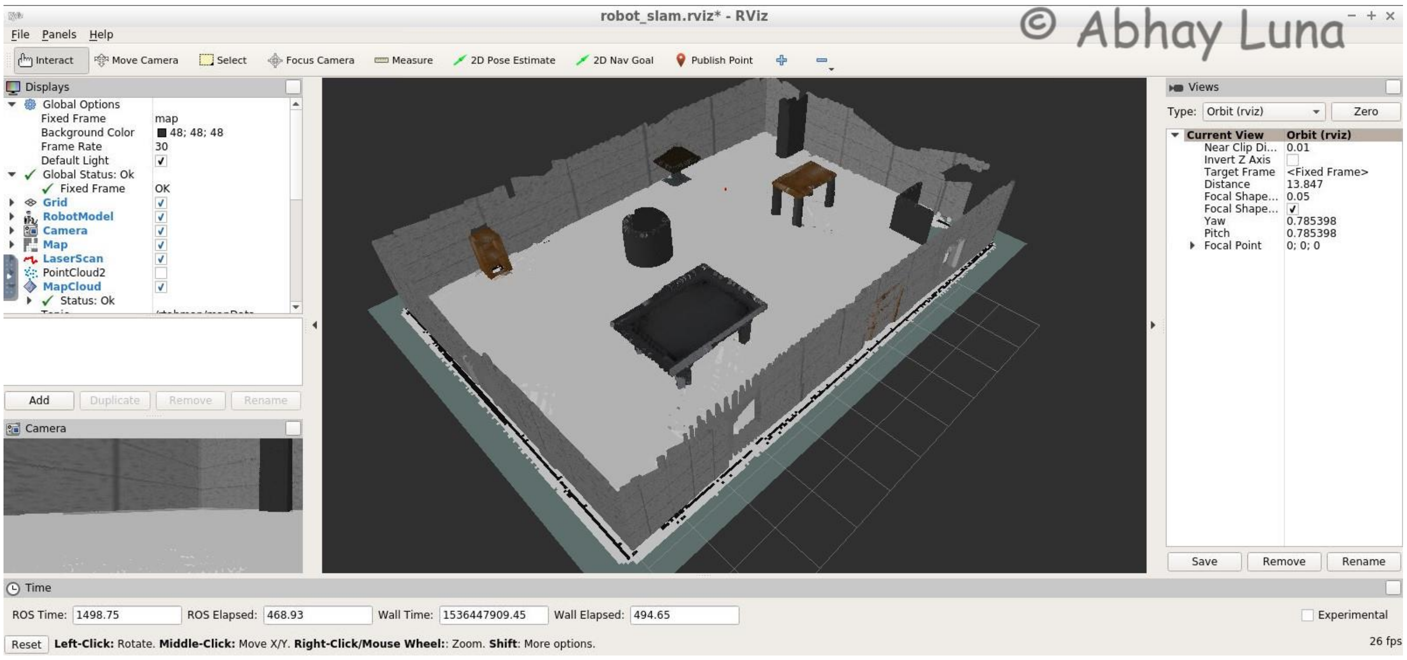
Kitchen Rtabmap

Custom World Scene

Upon launching the my custom world scene simulation along with the mapping node the slam-bot started mapping the immediate environment.

After navigating throughout the entire environment, the slam-bot was able to generate an accurate map of the world that was developed.

As can be seen from the rtabmap database, several loop closures were observed, and bot a 2D occupancy grid map and a 3D octomap were generated for the custom developed environment.



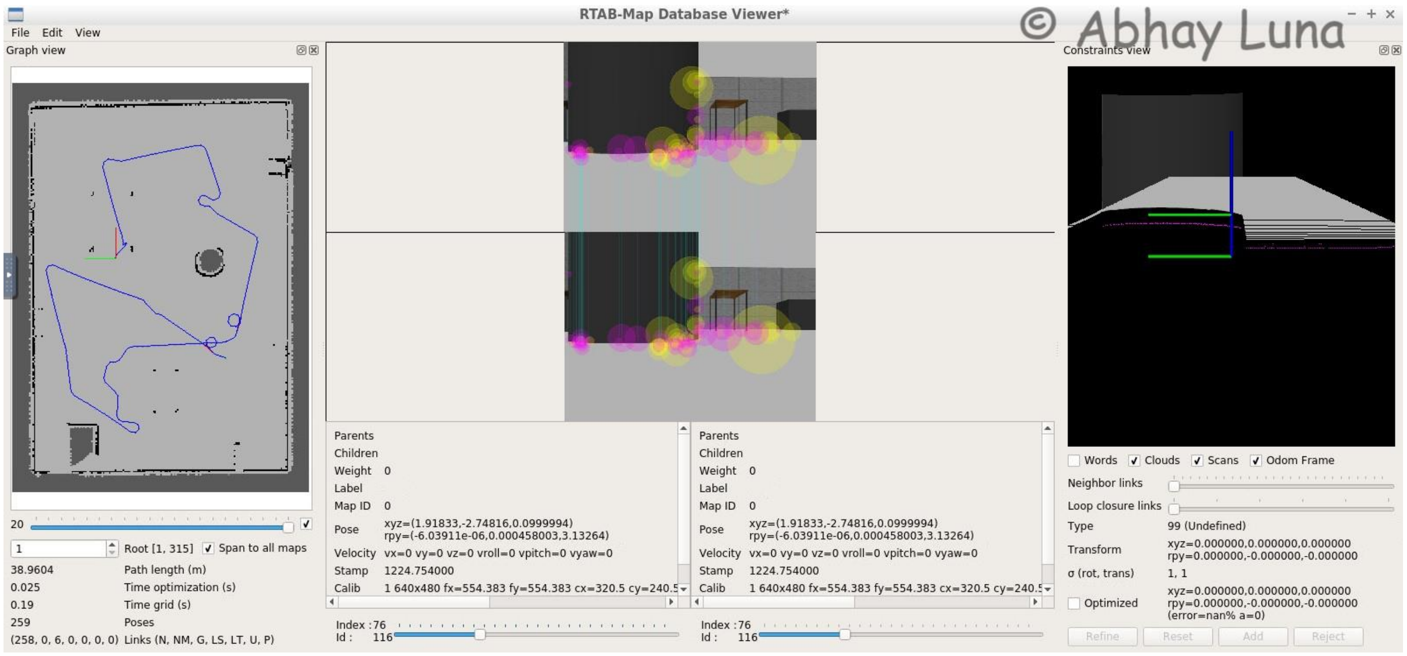
Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Default Light ✓
Global Status: Ok
Fixed Frame OK
Grid ✓
RobotModel ✓
Camera ✓
Map ✓
LaserScan ✓
PointCloud2
MapCloud ✓
Status: Ok

Type: Orbit (rviz) Zero
Current View Orbit (rviz)
Near Clip Di... 0.01
Invert Z Axis
Target Frame <Fixed Frame>
Distance 13.847
Focal Shape... 0.05
Focal Shape... ✓
Yaw 0.785398
Pitch 0.785398
Focal Point 0; 0; 0

ROS Time: 1498.75 ROS Elapsed: 468.93 Wall Time: 1536447909.45 Wall Elapsed: 494.65 Experimental

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

Custom slam mapping



20 ✓
1 Root [1, 315] ✓ Span to all maps
38.9604 Path length (m)
0.025 Time optimization (s)
0.19 Time grid (s)
259 Poses
(258, 0, 6, 0, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)

Parents
Children
Weight 0
Label
Map ID 0
Pose xyz=(1.91833,-2.74816,0.0999994)
rpy=(-6.03911e-06,0.000458003,3.13264)
Velocity vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp 1224.754000
Calib 1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=240.5
Index :76
Id : 116

Parents
Children
Weight 0
Label
Map ID 0
Pose xyz=(1.91833,-2.74816,0.0999994)
rpy=(-6.03911e-06,0.000458003,3.13264)
Velocity vx=0 vy=0 vz=0 vroll=0 vpitch=0 vyaw=0
Stamp 1224.754000
Calib 1 640x480 fx=554.383 fy=554.383 cx=320.5 cy=240.5
Index :76
Id : 116

Words ✓ Clouds ✓ Scans ✓ Odom Frame
Neighbor links
Loop closure links
Type 99 (Undefined)
Transform xyz=0.000000,0.000000,0.000000
rpy=0.000000,-0.000000,-0.000000
σ (rot, trans) 1, 1
Optimized xyz=0.000000,0.000000,0.000000
rpy=0.000000,-0.000000,-0.000000
(error=nan% a=0)

Custom Rtabmap

Discussion

Since it was a manual navigation, traversing through the two environment was rather tedious. Furthermore, making a close approaches to obstacles threw off the mapping process at times, and was difficult for the robot to recover afterwards. Although the Kitchen and Dining scene seemed to have less noticeable features that the my custom world scene, it was less complicated and faster to map that the latter. In order to map the my custom world scene, navigating around the objects that were placed to serve as both obstacles and features had to be done cautiously. Otherwise, if the robot got too close to objects, it would delocalize the robot, throwing off the mapping process. Whereas not getting close enough, would not allow the robot to map the entire my custom world scene accurately. One thing that was observed was that the laser range-finder had a lower range than necessary for the designed environment. While the Kitchen and Dining scene was more constrained, with solid objects and obstacles, the range-finder had solid surfaces to bounce off to detect distance accurately. However, the my custom world scene has more open space which seemed to have made the mapping slightly more complicated for the robot. Nonetheless, the robot was able to successfully map both environments accurately, solving the SLAM problem in the process.

The performance of the robot’s mapping for each environment have a few similarities. Both were capable of clearly mapping 2D map boundaries, except for the unbounded section of the Kitchen Dining area. The robot was able to generate 3D maps of each world that were very recognizable of the actual environment. However, both struggled with 2D and 3D mapping the tables due to the height and angle of the camera and laser sensors.

The Kitchen Dining Area resulted in full, apparent walls while the custom world had sections missing. This is partially due to failing to scan the entire area methodically.

Future Work

Possible future work includes developing a 4-wheeled robot equipped with the Jetson TX2 and a kinect RGBD camera sensor to map an entire real-life apartment. Possibly this would lead to building a custom Roomba robot to vacuum or somehow clean an apartment. With some improvements, this approach can be implemented in a drone for real estate aerial mapping; among other solutions that can be deployed for aerial or ground robotics.

Mapping is a very valuable feature in many robotics applications, both autonomous and controlled. This can be used in surveillance drones to generate maps of areas of interest as they scan the area. The map can be used for any sort of strategic planning for the area. An autonomous robot can use mapping to learn how to navigate around an environment to complete its goal. This would be useful for a home assistance robot that may also implement pick-nplace perception to grab items the owner may need. These are only two valuable examples where mapping would be invaluable.

A ROS package was generated and successfully performed 2D and 3D mapping using RTAB-Map. The robot was able to generate quality maps of the kitchen dining room and average maps of the custom world. However, there is still much room for improvement in this SLAM implementation. Further work is required to optimize the SLAM package to be more robust and reliable. Also, the robot may be expanded to perform more autonomous behavior. Newer configurations should be tested to attempt to address shortcomings noted in the Discussion section.