Map my Location - using SLAM Bot

Nakkeeran K

Abstract—In this project, objective is to create mapping structure of the given local environment using SLAM technique. Simultaneous Localization and Mapping (SLAM) algorithms are useful techniques that helps a mobile robot to navigate and map (create or update) itself in its current location. Graph SLAM algorithm is one of popular technique applied here to solve full SLAM problem by building a graph of poses and features with constraints related to motion or sensory measurements so an optimized map can be obtained as a result. This paper illustrates an experiment using a specific Graph SLAM technique called Real Time Appearance Based Mapping (RTAB-Map) using RGB-D Sensor to build three dimensions (3D) and two dimensions (2D) maps of two different environments.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Mapping, localization, navigation, mobile robots, SLAM.

1 Introduction

MApping is one of the key feature that determines the degree of autonomous a robot can achieve. However, mapping is also a very challenging problem because of the huge hypothesis space and environment around a robot can dynamically change. For example, objects around the map may shift or adjusted by other external forces, rendering the preconceived mapping would lead to incorrectness. This is the primary reason mapping is a critical feature in robotic applications. A valid map is very important for applications which involves motion planning especially navigational robots. This report will focus on the use RTAB-Mapping using RGB-D camera.

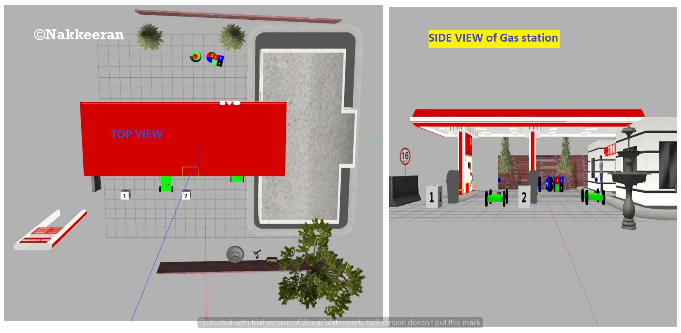

©Nakkeeran
TOP VIEW
SIDE VIEW of Gas station
Protected with trial version of Visual Watermark. Full version doesn't put this mark

Fig. 1. Custom-World

The RTAB-Map library provides the functionalities to transform robot current pose and sensors data into visible 3D and 2D maps. This project covers mapping of two different environments. In the first environment, ***"Udacity_bot"*** was used which was part of the "WhereAmI" udacity class project, while the second environment was created independently using Gazebo simulation tool. The maps produced from the experiment able to closely describe the correct properties of the environment.

2 Background / Formulation

Mapping is a challenging problem and vital for autonomous navigation without collision. Generally, poses of robot cannot be preconceived for every situation. If the poses of robot

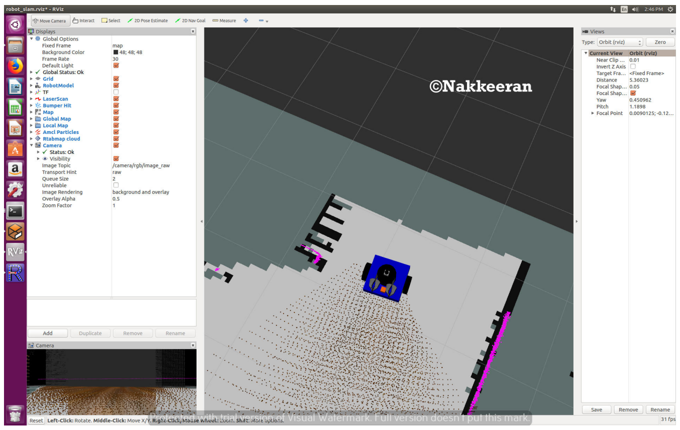

©Nakkeeran
Protected with trial version of Visual Watermark. Full version doesn't put this mark

Fig. 2. custom_bot

are known throughout mapping environment, the problem become easier as we can apply occupancy grid mapping algorithm to produce the map. Although 2D map can be easily obtained by plotting the result of occupancy grid mapping algorithm, it is 3D mapping which would help give us the most reliable collision avoidance, and motion and path planning, especially for flying robots or mobile robots with manipulators. In order to generate 3D maps, robot needs to be equipped with advanced sensors like: 3D lidar, 360-rotated 2D Lidar and RGBD camera.

For localization, there are scenarios where readily available map of the environment can be used by algorithms such as Monte Carlo (MCL) to navigate the bot to the goal/location point. But SLAM problem is highly visible when one perform computing the full posterior composed of the robot pose, the map and the correspondence under SLAM poses a big challenge in robotics mainly due to the continuous and discrete portion.

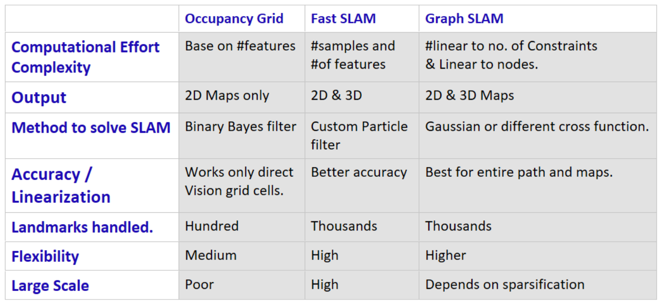
There are two types of SLAM problems:

- On-line SLAM - It happen when try to estimate robot pose and map at a specific point in time.
- Full SLAM - It needs to solve the posterior over the entire path up to the current time.

To resolve, the two popular SLAM algorithms are

1 Fast-SLAM
2 Graph-SLAM

2.1 Comparison of different SLAM(s) algorithm



	Occupancy Grid	Fast SLAM	Graph SLAM
Computational Effort Complexity	Base on #features	#samples and #of features	#linear to no. of Constraints & Linear to nodes.
Output	2D Maps only	2D & 3D	2D & 3D Maps
Method to solve SLAM	Binary Bayes filter	Custom Particle filter	Gaussian or different cross function.
Accuracy / Linearization	Works only direct Vision grid cells.	Better accuracy	Best for entire path and maps.
Landmarks handled.	Hundred	Thousands	Thousands
Flexibility	Medium	High	Higher
Large Scale	Poor	High	Depends on sparsification

Fig. 3. Comparison of different SLAM Algorithms

However, This report focuses on the Graph-based SLAM approach with RTAB-Mapping for performing 3D SLAM.

2.2 Graph SLAM

Two most popular SLAM algorithms are grid-based Fast-SLAM (GFSLAM) and Graph SLAM (GRSLAM). GFSLAM first solves robot trajectory problem using filter particle in which each particle hold an estimate of the robot poses. GFSLAM then applies the occupancy mapping algorithm to solve each particle mapping problem with known pose. The limitation of GFSLAM is that it relies on particle for pose estimation which can cause some problems if particle is not available at the important location on the map.
As a result, GRSLAM has this issue covered with a unique approach to improve accuracy while reducing computation complexity. GRSLAM is divided into two main pieces: the frontend and the backend. The frontend of GRSLAM focus on obtains sensory data to build a graph of constraints among robot poses as well as features of the environment. For example, RTAB-Map algorithm frontend takes in odometry data and images data from RGBD camera and lidar in order to detect loop closure which marks a location as already visited. By applying long-term and window memory management techniques, RTAB-map can support detecting loop closure in real time. On the other hand, The backend of GRSLAM will execute the map optimization algorithm in order to correct the poses of the robot along with all its links transformations. There are various of map optimization algorithms that can be applied such as Tree-based Network Optimizer, General Graph Optimization. As a result, RTAB-Map can produce 2D occupancy grid map, 3D grid map or 3D point cloud

2.3 RTAB-Map

RTAB-Mapping is a real-time method that uses appearance based SLAM algorithms with input from vision-based sensor inputs to perform mapping to perform a process called loop closure. This process determines if the robot has viewed the location before and adjusts the map accordingly based on the path loop it completed. This concept is critical to performing proper mapping of the environment. As the robot travels to new areas of the environment, the map is expanded. The more areas of the map that have been explored, the more comparisons the algorithm must make to determine whether or not a loop closure should occur. This causes the computations to take longer as the algorithm loses efficiency.
RTAB-Mapping is optimized for large-scale and longterm SLAM by using multiple strategies to allow for loop closure to be done in real-time. Image loop closure occurs fast enough to produce results before the next image is obtained. From this, the algorithm assembles an occupancy grid to generate 2D and/or 3D maps. RTAB-Mapping uses global loop closure where new images are compared to all previously viewed locations. If no match is found, it is added to memory. As the map grows with new locations being added, the amount of time to perform loop closure checks increases linearly. When the time to perform the comparison becomes longer than the acquisition time of a new image, the map starts to become ineffective. However, RTAB-Map implements a special memory management technique to ensure that the loop closure process happens in real-time. In this project, RTAB-Map relies on the use of RGBD Camera to perform localization and mapping.

In this implementation we use RTabmap. [1]

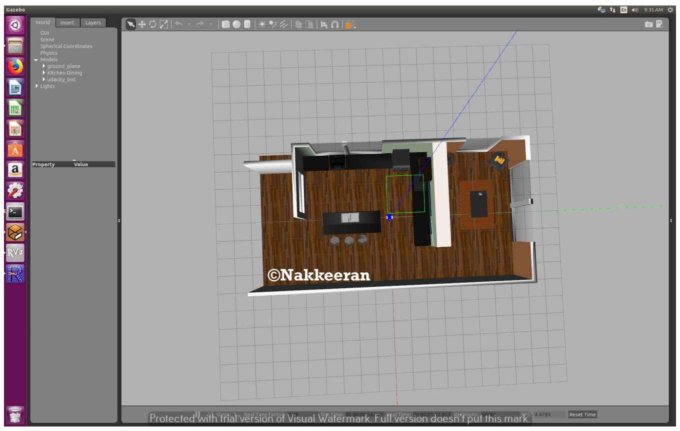

©Nakkeeran
Protected with trial version of Visual Watermark. Full version doesn't put this mark

Fig. 4. Kitchen world in Gazebo

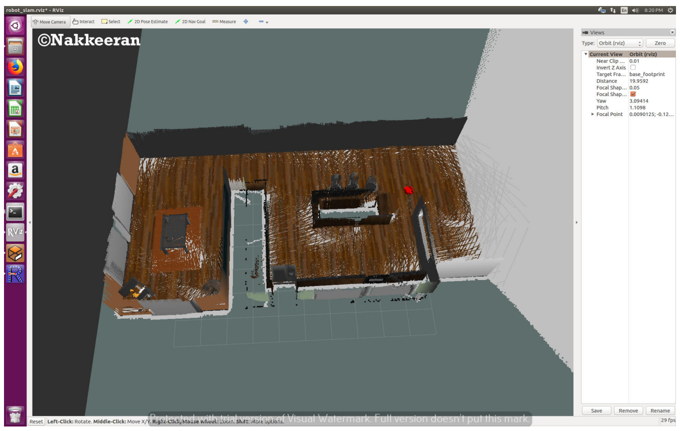

©Nakkeeran
Protected with trial version of Visual Watermark. Full version doesn't put this mark

Fig. 5. Kitchen Env in Rviz

3 About Scene and configuration of Robot

3.1 Robot Model

Since main aim of the project is to 3D Map generation for two environment, robot used is same as ”udacity_bot”. However, tried with customized nk_bot also.

3.2 Worlds

The first environment, found in worlds/kitchen dining.world, was provided for the project. The environment can be seen in figure 4.

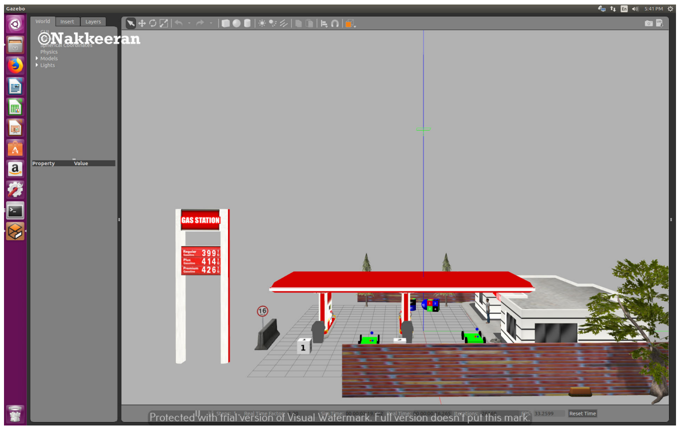

©Nakkeeran
GAS STATION
Protected with trial version of Visual Watermark. Full version doesn't put this mark.

Fig. 6. Gas-Station-Environment.

The second, a customized environment worlds/gas-station.world, was created using gazebo’s built-in Building/Model editor for the project. Its a simple gas-station open environment. A number of objects from the Gazebo Model Database were imported and placed. The environment can be seen in figure 6.

3.3 Robot configuration

During development of the mapping project, it became apparent that the RGB-D camera frame required rotation and translation to properly re-orient the measured point cloud axes for mapping purposes. The robot was modified to include a RGB-D camera. It was mounted on top of the robot to maximize the height, making use of the vertical field of view. The LIDAR was moved to the front of the robot since it only scans in a 2D plane and therefore its output is less affected by height. The robot configuration is defined in the urdf/udacity bot.xacro and urdf/udacity bot.gazebo files of the package. Also, a specific transform node was created in the robot description.launch file. The TF tree frames can be found in Figure 7.

Note: The links and frames found under the camera rgb frame were essential to orient the cameras axes correctly. These link and frame definitions were adapted from the turtlebot description ROS package.

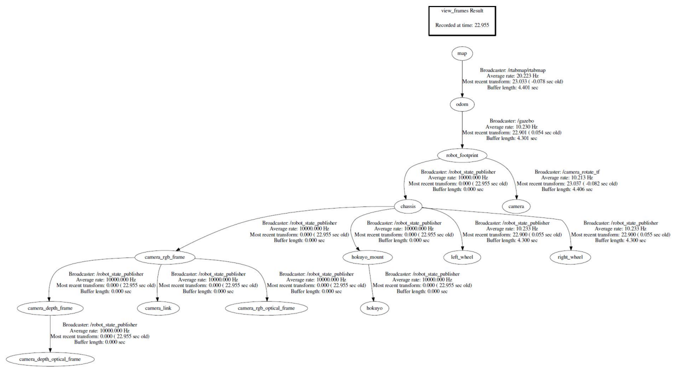


Fig. 7. Transform Tree Frame view

3.4 Packages

Required files for Navigation

All project files were contained in a single ROS package called mapmylocal project. The package contained all the necessary files to run the simulation in both worlds. The robot definition and world files are located in the urdf and worlds directories as mentioned previously. The package also contains a number of launch files in the launch directory, which are used to launch the various nodes required for the project. Two separate launch files exist for the two different worlds. udacity_world.launch launches the kitchen dining world while gasstn.launch launches the custom my world. In addition, the teleop.launch, mapping.launch and rviz.launch files launch the teleop node to control the robot, the rtabmap ros node to implement the RTAB-Map algorithm, and the rviz node to launch Rviz (optional - anyway it is called in udacity_world.launch itself), respectively. Once all nodes were launched, rqt graph and tf viewframes was used to ensure that all nodes were publishing and subscribing to the correct topics.

- **mapmylocal/launch/udacity_world.launch:** The udacity world.launch file spawns the robot (details found in the xacro file) into the specific world. It also launches the saved Rviz configuration. It includes the robot description launch file, the gazebo kitchen and dining world, spawns the robot in gazebo world, and launches RViz.

- **mapmylocal/launch/mapping.launch:** The mapmylocal/launch/mapping.launch file contains various parameter settings of RTabmap and is used with udacity world as well as custom world files. It launches the map server, the odometry frame, updates the mapping node and the trajectory planner server. It need to be updated with RTabmap parameters example the ”Frame-id” is same as mentioned in ”robot-description.launch”. Similarly, update the remap ”scan” parameter is same as in udacity_bot.gazebo file.

- **mapmylocal/launch/teleop.launch:** The mapmylocal/launch/teleop.launch file contains commands mapped to keyboard which helps manually navigate the bot in a given world environment. It launches on the terminal with 9 keys which one can be used to increase or decrease the speed of linear/angular movements.

4 RESULTS

4.1 Kitchen Dining Area

The generated mapping of the Kitchen Dining area resulted in 26 global loop closures. This is quite high which highlights that the mapping parameters are not fully optimized. The 2D map, 2D occupancy grid, and 3D visualization can be found in Figures 8-12.

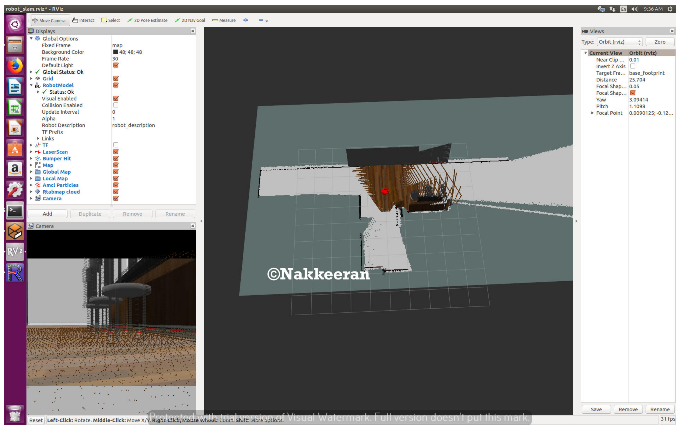


Fig. 8. Kitchen-and-Dining world-1st step

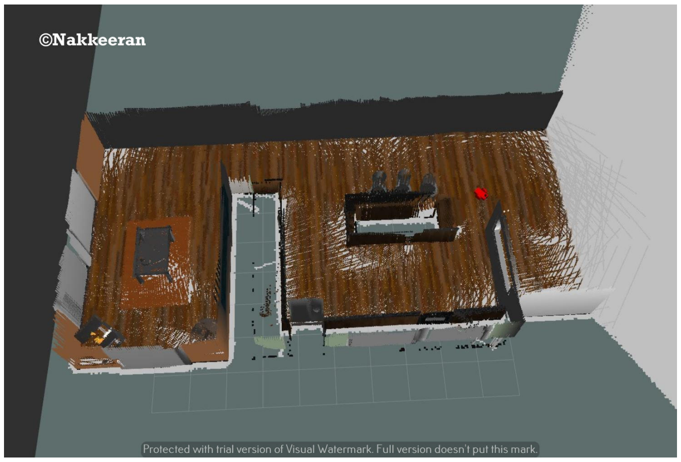


Fig. 9. Kitchen world - Mapping complete in Rviz

The robot was able to correctly map the boundaries very well. The missing chunk on the right side is likely due to that side of the room being unbounded. From the the 2D map, one can see where the kitchen island and chairs lay as well as the chair in the corner of the next room over. However, it should be pointed out that the robot failed to map 2D boundaries for the tables in the next room over. This is due to the height of the laser scanner - it was not level with the horizontal portions of the tables and could not map them as boundaries.

One way to improve this might be to use the RGBD camera and use the ROS depthimage to laserscan package.

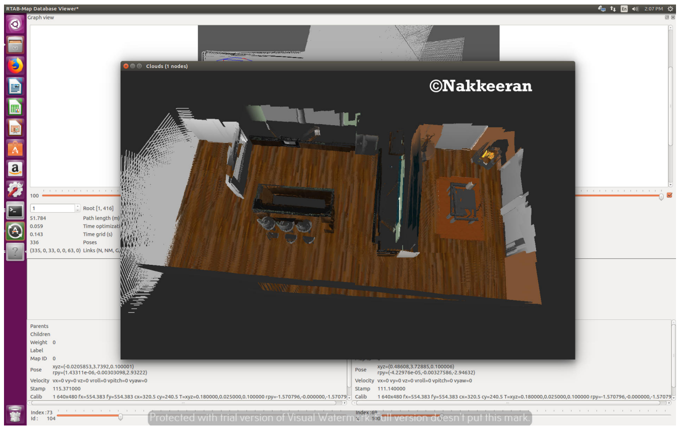


Fig. 10. 3D cloud-mapping

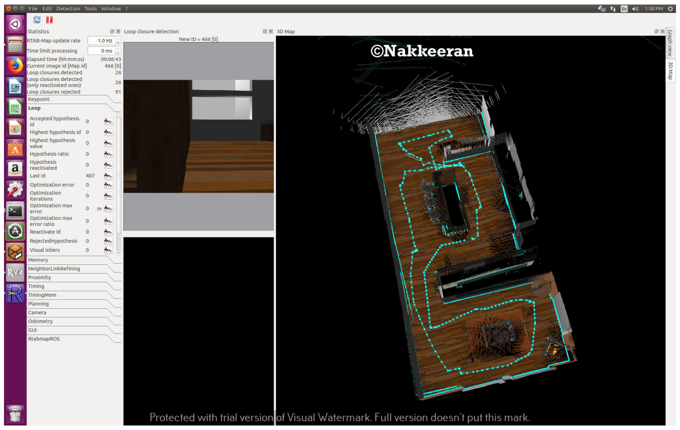


Fig. 11. GrapeView-Trajectory for Kitchen Dining world

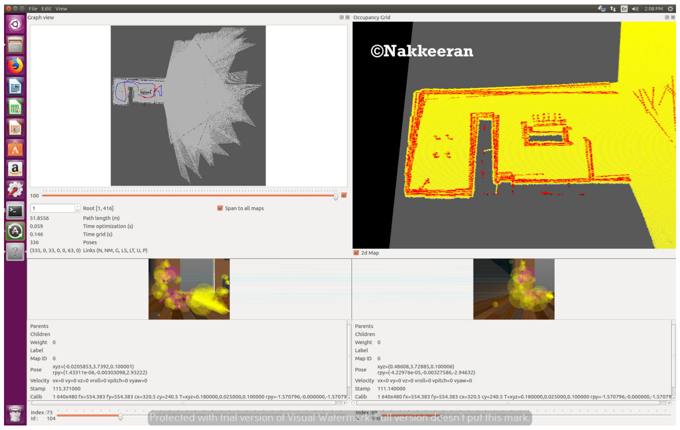


Fig. 12. Graphview and (2D) Occupany Grid for Dining world

The generated 3D map very clearly resembles the external surfaces of the rooms and objects within them. The robot had trouble filling the insides of the table and kitchen island. This is expected as the camera images can only perceive external surfaces of the objects.

4.2 Custom World - GAS STATION

rtabmap-databaseViewer reported 12 loop closures for this custom world which can be seen in figure 9, along with the 2D map showing the path followed by the robot.

The 2D map, 2D occupancy grid, and 3D visualization can be found in Figures below. As seen in Figure-9, the boundaries are mapped very well and the flat portion marble table was too high to map in 2D. From the 2D occupancy grid, it can be seen in the upper left corner (near the jersey barrier) that objects in the corners affect ability to generate a totally accurate map in that section.

The 3D map shows some difficulties mapping high regions of the walls. This is presumably due to the short path not having scanned the entire section of the wall. Also, due to the objects in the corners, the remaining walls could not be seen.

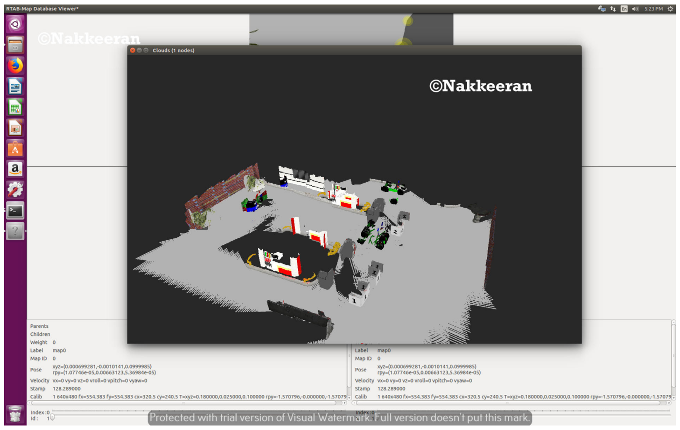

©Nakkeeran
Protected with trial version of Visual Watermark. Full version doesn't put this mark.

Fig. 13. 3D: Gas Station rendered with 3DCloud-maps

5 DISCUSSION

5.1 General Localization Results

In both Environment, robots were able to successfully complete the mapping generation and performed equally well. udacity bot was able to map both environments reasonably well using RTAB-Map. There were however some problems that can be observed. From the generated map of the kitchen dining world shown previously (Fig6), we can see that the left side of the world, the kitchen area, was mapped reasonably well, while the dining area on the right has more repeated, choppy artifacts. These artifacts are usually due to no loop closures being detected and therefore the graph not being optimized. The map had 26 loop closures, however, it can be seen that most of the loop closures occurred in the kitchen area with only a few in the dining area. This would explain the better quality map in the kitchen area as opposed to the dining area. The same phenomenon can be seen in the 3D map of the custom world environment (Fig12). This may be solved by changing some of the RTAB-Map parameters such as the feature detector strategy or loop closure parameters however this was not explored in this project. Similarly, The path taken and the distances from other objects often affected loop closure detection such that it would incorrectly close the loop prematurely. More work needs to be done to tune the mapping parameters to optimize loop closure detection.

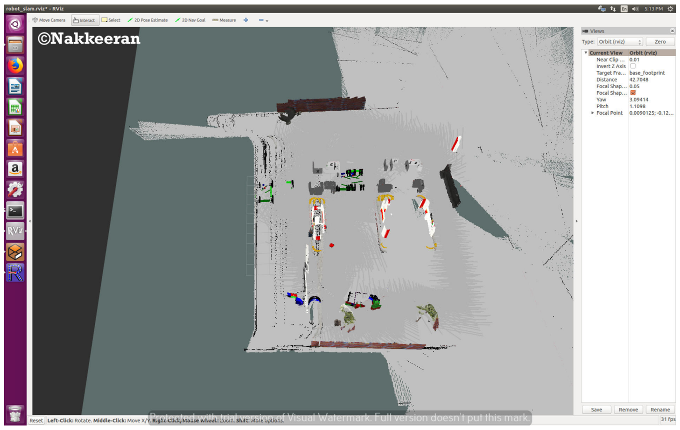

©Nakkeeran
Protected with trial version of Visual Watermark. Full version doesn't put this mark.

Fig. 14. Mapping complete on Custom world

Another observation was that tall objects were not fully mapped as can be seen in the point clouds. This was due to the fact that the RGB-D sensor is relatively low down, close to the ground since the robot is small. In order to overcome this, the sensor can be mounted at an angle such that it is looking slightly up. This will allow more of the environment to be mapped and may increase the number of loop closures, since more features will be detected.

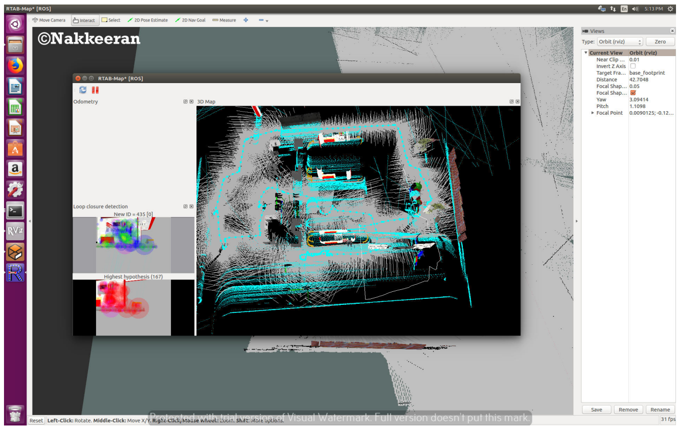
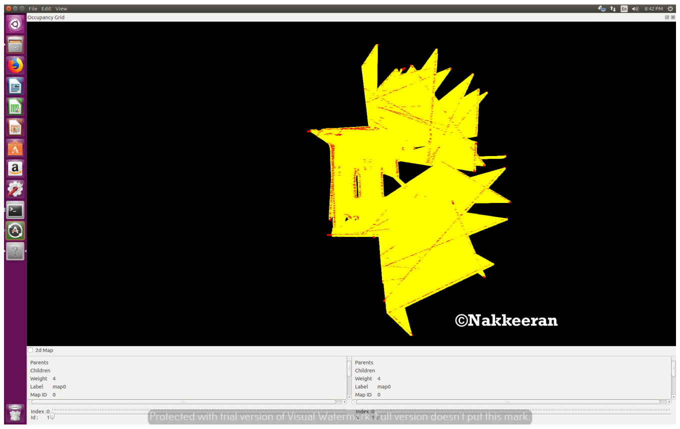

©Nakkeeran
Protected with trial version of Visual Watermark. Full version doesn't put this mark.

Fig. 15. RTabmap-3Dmap view

5.2 comparison betn udacity_world Vs custom_world

Couple of difference noticed while mapping custom world

1 Loop closures are lesser.
2 Maps looks murkier in 2D map alignment, when loop closure rejection happens. Yet to understand the reason.



Occupancy Grid
©Nakkeeran
2d Map
Protected with trial version of Visual Watermark. Full version doesn't put this mark.

Fig. 16. 2D: Occupancy Grid for Custom world

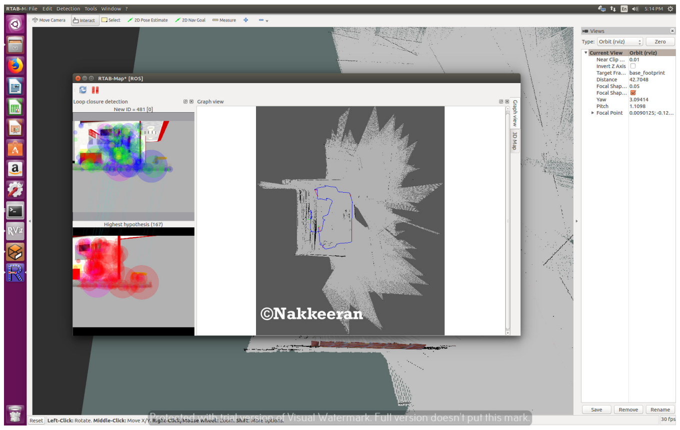


Fig. 17. GraphView Custom world

6 CONCLUSION / FUTURE WORK

In conclusion, this project tackled one of the fundamental building blocks of an autonomous robots, simultaneous localization and mapping. However, for this project, the robot was driven successfully by manual command mapping most of the world environment. An obvious extension to this project would be to use the results from RTAB-Map to localize a robot within an unknown environment and to detect and avoid obstacles when doing path planning and navigating autonomously to a goal position. The improvements and suggestions discussed in the previous section should also be implemented, such as increasing the angle of the kinect sensor. Finally, the algorithm could be implemented in hardware and tested in a real world environment using a mobile robot platform with a kinect sensor.

6.1 Real time Hardware Deployment on Jetson TX2

To conclude, Next plan is move from x86+dGPU and try on a Jetson TX2 board for path planning project. The TX2 prototype board will have a camera which could be connected into the model with suitable drivers. Laser scanner hardware and drivers would have to be integrated in order for a hardware version to operate.

REFERENCES

[1] L. Lamport, *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley, 1994.

1 RTabmap ROS from Scratch. https://wiki.ros.org/rtabmap_ros/Tutorials/SetupOnYourRobo

2 IntroLab, "rTab-map overview. Udacity.com

3 http://www2.informatik.uni-freiburg.de/ stachnis/pdf/grisetti10titsmag.pdf

4 http://robot.cc/papers/thrun.graphslam.pdf