Rtabmap for SLAM in ROS

Deepak Trivedi

Abstract—Real-Time Appearance-Based Mapping (Rtabmap) is demonstrated two simulated worlds using the Robot Operating System (ROS) platform. The technique is demonstrated using an RGBD camera for depth detection. The worlds are simulated in the Gazebo environment. Teleoperation is used to navigate the worlds as the ROS Rtabmap package performs SLAM. Finally, possible future work is recommended.

Index Terms—Robot, SLAM, Mapping, Localization, Rtabmap, ROS

1 Introduction

MAPPING of the environment for navigation is an important aspect of all mobile robots and devices. Simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it [1]. Several algorithms are available to achieve this goal through the use of sensors such as laser scanners, cameras, inertial measurement units (IMUs), and other sensors for distance, velocity and acceleration measurement. GraphSLAM is a family of SLAM algorithms which uses sparse information matrices produced by generating a factor graph of observation interdependencies (two observations are related if they contain data about the same landmark) [2]. Another family of algorithms is FastSLAM [3].

Real-Time Appearance-Based Mapping (Rtabmap) is a GraphSLAM 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. 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 [4].

A bigger list of SLAM methods is available at [5].

The Robot Operating System (ROS) platform provides a library for the Rtabmap algorithm [4]. Gazebo is an open source robotics simulator that allows realistic simulation of robots that could be controlled via ROS. In this project, Rtabmap is used to perform SLAM on a rover. The rover uses wheeled locomotion, and use an RGBD camera for depth measurement. A Hukoyo scanner is also mounted for observation, but is not used as input for SLAM. As a part of this project, the ROS package for the rover and performing SLAM is developed from scratch. This process involves many debugging steps and the tools provided by ROS for debugging are extensively used. The algorithm is demonstrated on two simulated worlds. The first world is provided as part of the assignment, while the second world is created as a part of the project in Gazebo.

2 Background

The ability to simultaneously localize a robot and accurately map its surroundings is a key prerequisite of truly autonomous robots. Localization from the use of noisy sensors and models is a part of a class of problems known as the filtering problem. In a previous project, a solution to this problem was demonstrated using Monte Carlo localization. Mapping addresses the problem of building a map of an environment from a sequence of landmark measurements obtained from a moving robot. Since all measurements and robot motion are subject to error, the mapping problem necessarily induces a robot localization problem hence they are often solved simultaneously, and the name SLAM is appropriate. SLAM is one of the fundamental problems in robotics. Even if an environment, such as a factory floor is known a priori, mapping is important since the environment is dynamic and may change over time. Challenges in SLAM can be broadly classified into two categories: convergence of the built map and computation requirement for real world application [6].

One key issue of the convergence of the built map is the loop closure problem. Loop-closure is the key concept of SLAM, and without it, SLAM reduces to odometry. A robot performing odometry and neglecting loop closures interprets the world as an infinite corridor in which the robot keeps exploring new areas indefinitely. A loop closure event informs the robot that this corridor keeps intersecting itself. By finding loop closures, the robot understands the real topology of the environment, and is able to find shortcuts between locations [7]. Correlating the places already visited with new information to identify whether the robot is back to the same place could be challenging due to noisy sensors, repetitive elements in scenery, strong appearance changes due to illumination, weather or seasons, and so on.

As the environments get larger, and especially as the dimensionality of the problem goes from 2D to 3D space, computational complexity increases rapidly. As an example, if a grid-based approach is used and the 3D space is voxelized, doubling space resolution naively will increase memory use by 8X, and computational time by perhaps even more. Performing SLAM in real time could therefore be very challenging. One related issue is map representation. Even when memory is not a tight constraint, e.g. data is

stored on the cloud, raw representations as point clouds or volumetric maps are wasteful. Also, how often is it optimal to update the the map and how to decide when this information becomes outdated and can be discarded is a question that depends on the size of the environment. Optimal coordination between swarms of robots cooperating on SLAM is also an open problem.

A short history of SLAM is provided in [7], and consists of a classical age (1986-2004), which introduced probabilistic formulations such as approaches based on Extended Kalman Filters, Particle Filters, and maximum likelihood estimation; The subsequent period is what we call the algorithmic-analysis age (2004-2015), and it saw the study of fundamental properties of SLAM, including observability, convergence, and consistency. In this period, the key role of sparsity towards efficient SLAM solvers was also understood, and the main open-source SLAM libraries were developed. For an extensive discussion on the evolution of SLAM methodologies, and open challenges associated with each of these, the reader is referred to [7].

3 Simulations

This section discusses the performance of robots in simulation with the Rtabmap algorithm. First, the robot model design is described. Then, the simulation environment is described, including the packages used. This is followed by a discussion of the parameters chosen for the robot to properly localize itself. Details for the robot and the package created are provided.

3.0.1 Model design

Figure 1 shows a visual illustration of the rover. Not visible in the illustration are two caster wheels mounted on the forward and distal locations on the robot chassis, to provide at least a three-point contact between the ground nad the robot. In addition to standard odometry, the robot includes a top-mounted Hokuyo laser scanner and a front-mounted depth camera. The laser scanner should be mounted high enough such that the robot geometry is not obstructing its view. Table 1 lists the geometric and inertial parameters of the benchmark robot model.

TABLE 1
Geometry of the robot

Parameter	Benchmark
Chassis dimension	0.4 x 0.2 x 0.1
Chassis mass	15.0
Chassis inertias	0.1
Caster wheel radius	0.0495
Camera location	(0.2,0,0)
Laser scanner location	(0.15,0,0.15)
Wheel radius	0.1
Wheel mass	5.0
Wheel inertia	0.1

Figure 2 shows how the various frames of the robot connect to each other. The 'map' frame is created by 'rtabmap', and it is connected to the 'odom' frame. These two frames are fixed to the ground. The 'robot_footprint' is a moving frame that is connected to the robot. The 'chassis' frame is

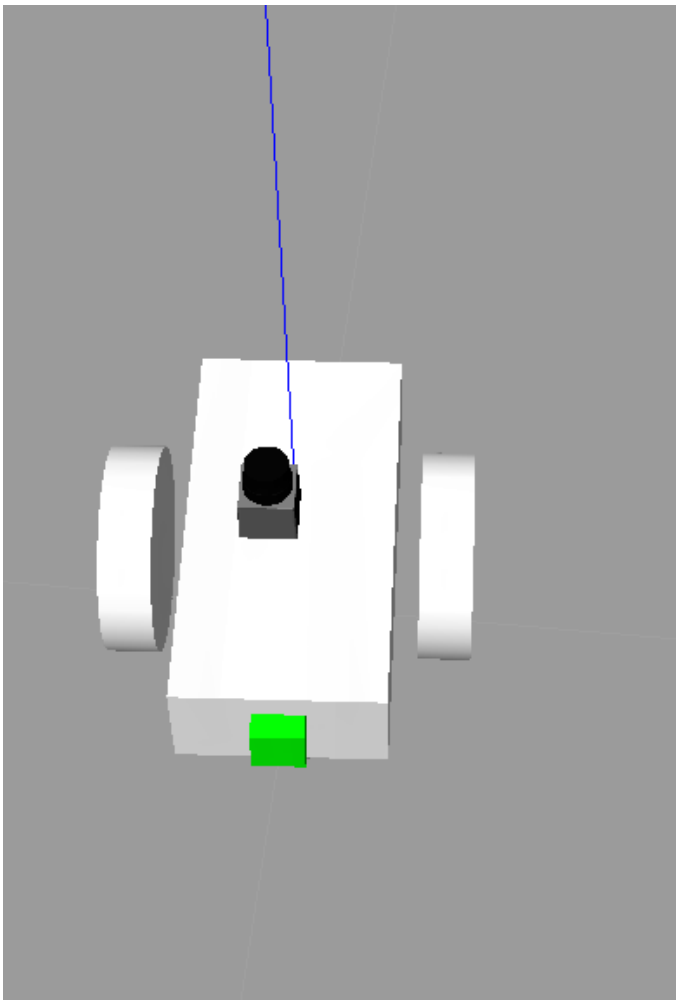

Fig. 1. Robot geometry

fixed relative to the 'robot_footprint'. The frames for the 'left wheel' , 'right wheel', the 'depth camera' and the 'Hukoyo' laser scanner are defined relative to 'robot_footprint'. The depth camera uses the 'camera_rgbd_frame' for correctly orienting the output of the camera. The roll-pitch-yaw rotation of this frame is $(-\pi/2, 0, -\pi/2)$ with respect to the 'camera' frame in order to correct for the differences in the reference frames of the two axes.

In this project, we convert the output of an RGBD camera to mimic a laser scanner. This is achieved using the $depthimage\_to\_laserscan$ package of ROS [8]. This package takes a depth image and generates a 2D laser scan based on the provided parameters. The output of $depthimage\_to\_laserscan$ need to be accomplished using 'camera_rgbd_frame' as the in order for the orientation of the output to be correct.

In order to navigate the environment, we use the $teleop$ package, which broadcasts the cmd_vel topic. This topic then allows Gazebo to move the robot. The nodes of the $depthimage\_to\_laserscan$ use various topics published by the camera module to publish a $scan$ topic, which is used by $rtabmap$ for creating the
map. topic. For more details, please look at Figure 3 shows the ROS computation graph of the 'slam_project' package. This provides a graphical representation of how various

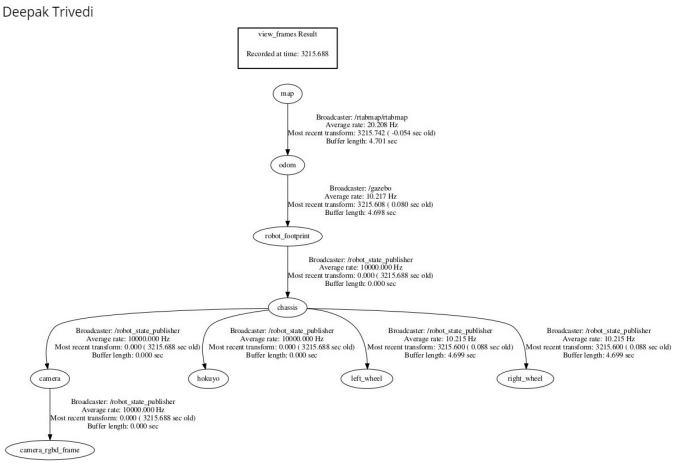

Fig. 2. TF Frames

nodes interface.

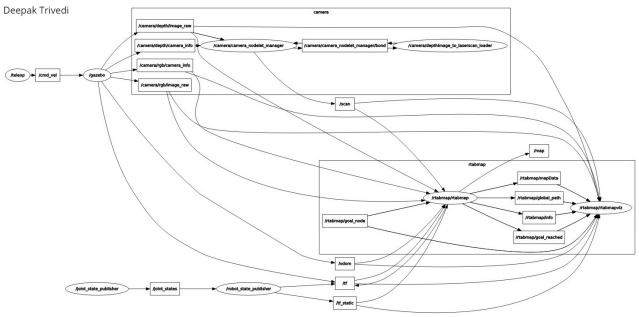

Fig. 3. RQT Graph showing nodes in the slam_project package

3.1 Scene configuration

Figure 4 is a top view of the Gazebo rendering of the Kitchen Dining World. The world contains two rooms with a number of obstacles including furniture and walls.

Fig. 4. Gazebo rendering of the Kitchen Dining World

Figure 5 is a top view of the Gazebo rendering of the Outdoor world created in Gazebo as part of this exercise. The world contains a number of objects as obstacles. A number of other worlds were also created and experimented on as part of this project, with a number of different configurations and objects.

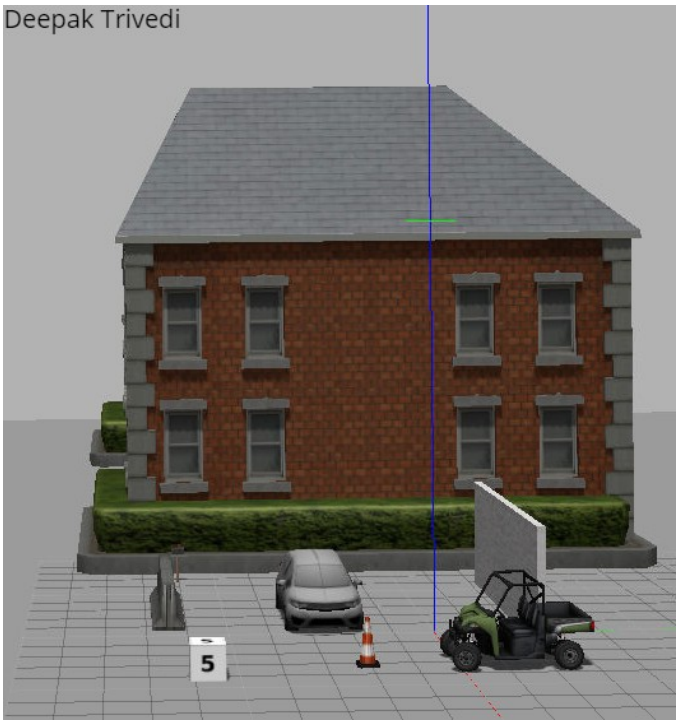

Fig. 5. Gazebo rendering of the custom world

4 Results

4.1 Map Results

In the following, we explain some of the key elements of rtabmap with the help of example outputs in 'rtabmap_databaseViewer'. As mentioned earlier, rtabmap works on incremental appearance-based loop closure detection, using a bag-of-words approach to determinate how likely a new image comes from a previous location or a new location. When a loop closure hypothesis is accepted, a new constraint is added to the maps graph, then a graph optimizer minimizes the errors in the map [4]. Using 'rtabmap_databaseViewer', it is possible to look at various constraints in any frame of the rtabmap database. Figure 6 is an example of constraint in one of the frames in the database.

Figure 7 shows a view of the graph of the map generated by the rtabmap algorithm along with the path taken by the robot.

In order to find constraints, rtabmap first needs to detect features in images and match them to each other. This process should be robust to image transformations such as rotation, scale, illumination, noise and affine transformations (distortions.) Fast and robust image matching is a very important task rtabmap needs to perform, and various algorithms are available for it, such as SIFT, SURF, and ORB. A full discussion of these algorithms is available in [9]. Once feature detection is performed, the algorithms creates keypoints that could be matched among images. Figure 8 shows an example of keypoints in one of the frames.

Figure 9 shows some statistics of the database. The codes stand for the following: Neighbor, Neighbor Merged, Global Loop closure, Local loop closure by space, Local loop closure by time, User loop closure, and Prior link [10].

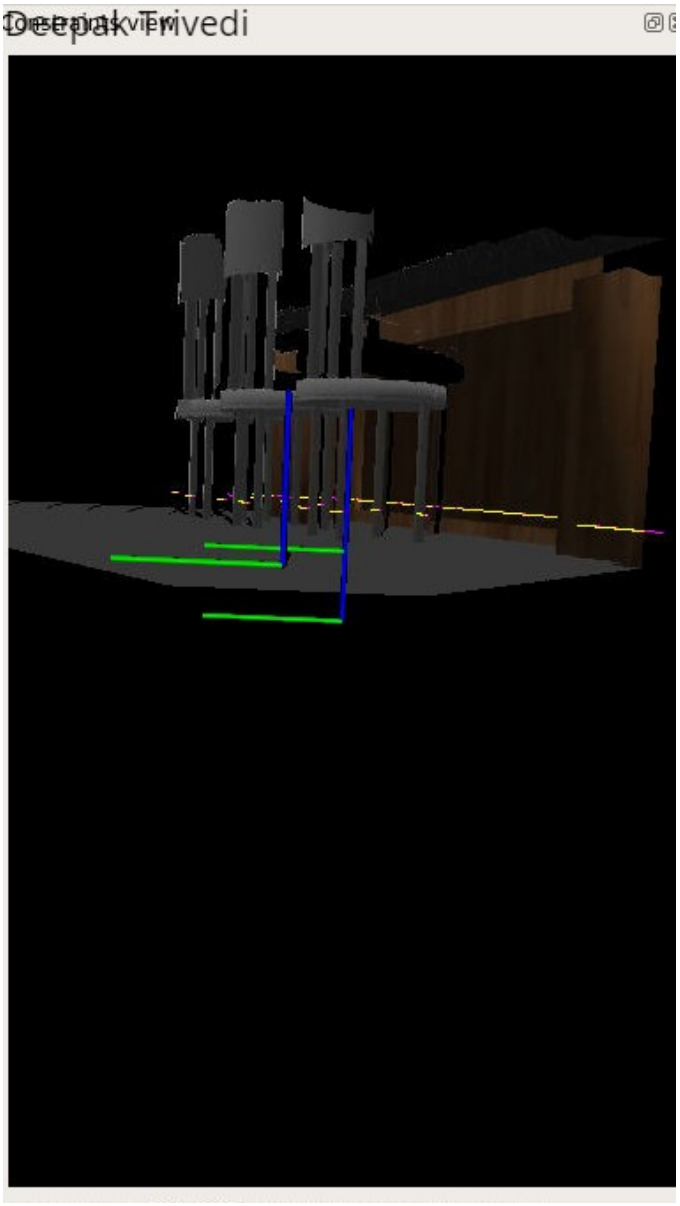

Fig. 6. An example constraint view of the Kitchen Dining World

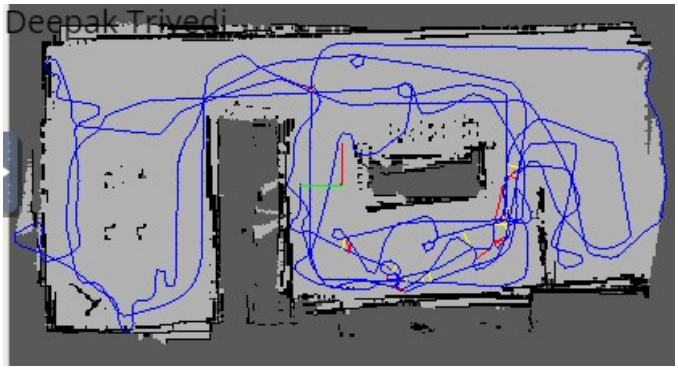

Fig. 7. An example of graph view of the Kitchen Dining World

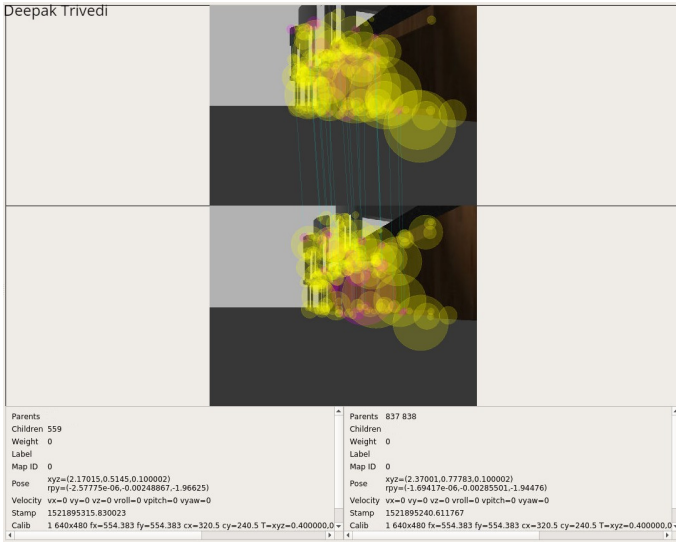

Fig. 8. An example of keypoints view of the Kitchen Dining World

Neighbor links are added between the current and the previous nodes with their odometry transformation. Loop closure links are added when a loop closure detection is found between the current node and one from the same or previous maps. Local and Global refers to the local graph and the global graph [11].

For the map generated for the Kitchen Dining World, we could identify 925 neighbor links and 27 loop closures. There are also 20 loop closures by space.

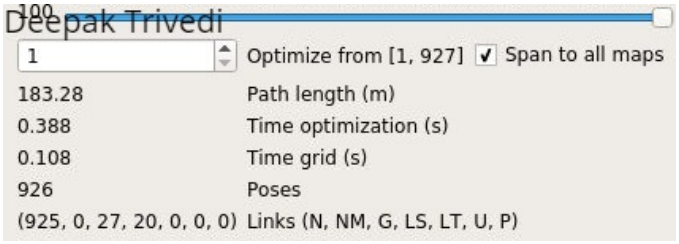

Fig. 9. An example of loop closures in the Kitchen Dining World

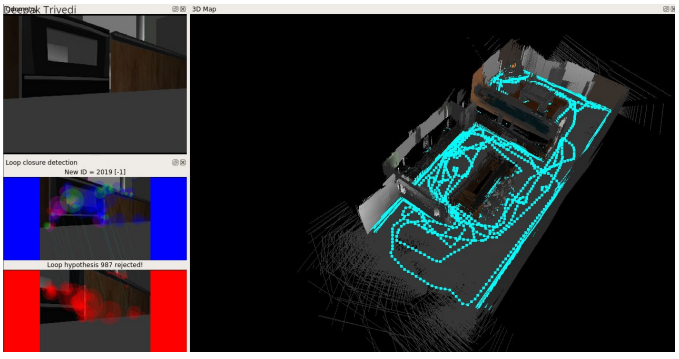

Fig. 10. Rtabmap 3D map for custom world

The rtabmap database output for the customized Outdoor World is shown in Figure 11. In the map generated for this world, we identify 191 neighbor links and 20 loop closures.

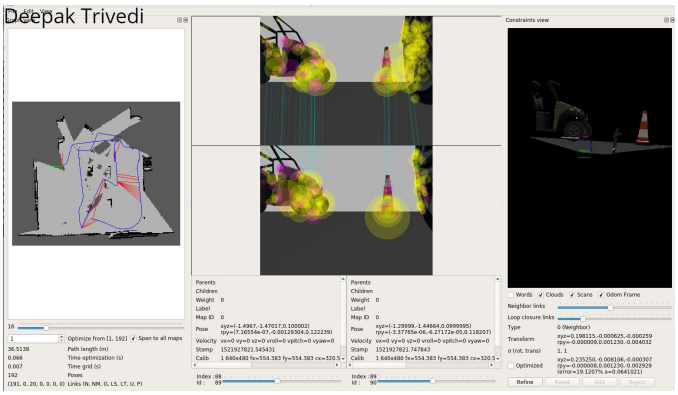

Fig. 11. An example of rtabmap-databaseViewer output for custom world

A 3D rendering of the world map for Outdoor World is shown in 12. It is seen that objects of the world such as the car, wall, plants, mailbox can be clearly identified. However, there is a little smear on some of the object boundaries, showing that the localization has been imperfect. This situation does not seem to improve as more passes are made through the world by the robot, and in fact in many cases deteriorates. Having more feature rich environments may help alleviate some of these issues.

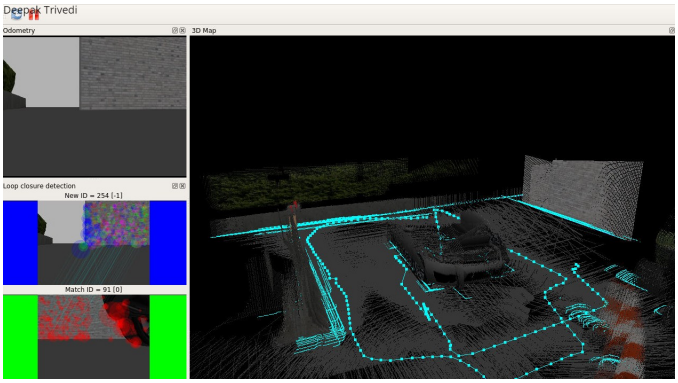

Fig. 12. Map for custom world

A 3D rendering of the world map as seem on rviz for Outdoor World is shown in 13.

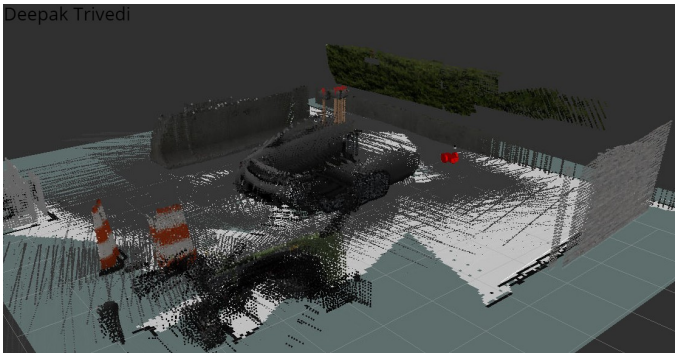

Fig. 13. Rviz view for custom world

Following observations were made:

- During mapping trials on custom worlds, it was found that having repetitive elements, such as the same object repeated multiple times in the world confuses rtabmap causing the map to get corrupted, possibly due to false loop closures.
- Having several distinct objects that make the world feature-rich helps with mapping because keypoints help with finding loop closures.
- The speed of mapping in the GPU environment, though not real-time, was quite impressive. The algorithms will likely do well enough on a Jetson TX2, but may struggle on a Raspberry Pi .
- Often, the map gets corrupted and the algorithm either finds false loop closures, or drifts. There need to be a better understanding of what causes this. It might be due to drift in odometry. It will be useful to understand the causes of these issues and develop a more robust implementation.
- It is possible for the robot to sometimes get stuck due to imprecise teleoperation. In the simulated environment, it was not clear what is a good way to recover from this issue. This led to several failed attempts of mapping the environment.
- There are several issues related to installation of rtabmap, including installation of dependencies. A first attempt on making it work on the Jetson TX2 failed. Therefore, the project was done on the provided GPU environment. The way it was accomplished requires a re-installation of rtabmap dependencies every time the GPU environment gets restarted. There must be a way to avoid this effort, which could be found out given more time.
- Several bugs in the implementation of the package were related to incorrect specification of frames. These were corrected using trial and error and finding references on the Internet. Overall, this was a good, albeit time-consuming way to learn.

5 Discussion / Conclusion

Several lessons were learned from this project that are worth mentioning.

First, as mentioned above, there were challenges involved in installing packages and their dependencies. This had to do with version issues. Several troubleshooting and debugging steps were followed to finally get a working version of rtabmap.

Second, this was an opportunity to create a ROS package from scratch, integrating teleoperation, Gazebo, Rviz, rtabmap, and *depthimage_to_laserscan* together. When several packages were integrated, topics subscribed and broadcast by the packages may need to be remapped in order for these packages to communicate. Several debugging tools are available in ROS to find out errors in communication. Part of the project was focused on ensuring all packages communicate with each other via ROS topics. This was an excellent learning.

Third, there were learnings regarding nuances of mapping, and what is effective. For example Faster robot speed allows more passes to be made around the world - but a lower spatial resolution, which leaves gaps in the measurement. Having similar objects and several location, or a continuous smooth wall confuses the loop closure algorithm

leading to poor results. Once the loop closure algorithm learns wrong information, it seems like making multiple more passes does not help in making it forget and correct the mistakes easily. In these cases, it may be more effective to simply start over. It is unclear how this will work in a practical application without supervision. Perhaps there are parameters in the rtabmap algorithm that could be fine tuned to improve performance in specific environments. For example, in presence of periodic structures such as a forest with many similar trees, or a room with many similar pillars or windows, the algorithm may fail to map correctly. Humans may find this understandable since it is also easy for humans to get lost in such environments. If any tags or markers could be added in such an environment, that will significantly help with mapping.

6 FUTURE WORK

Following are some ideas on things to try in future

- Play with rtabmap parameters to see if loop closures could be improved in different environments.
- Multiple sensors: Is it possible to use more than one depth camera to gain accuracy of mapping?
- Additional dimensions: Implementing rtabmap on a drone will teach how the algorithm performs when extra movement degrees of freedom are present.
- Multiple robots: Implementing rtabmap on a swarm of collaborating robots will be an interesting and useful effort. It would be interesting to learn what is the best policy for the robots to follow to optimize map building.

6.1 Hardware Deployment

With a few additional steps, the Rtabmap could be deployed on hardware. For example, ROS could be set up on a Raspberry Pi, or another microcontroller, and be used to control motor drivers and interface with laser scanner or an RGBD camera. Given fewer time constraints, it would be interesting to deploy this on a Jetson TX2, possibly with one of the Traxxas scaled vehicles as the rover.

REFERENCES

[1] "https://en.wikipedia.org/wiki/simultaneous_localization_and_mapping."
[2] "https://en.wikipedia.org/wiki/graphslam."
[3] "http://robots.stanford.edu/papers/montemerlo.fastslam-tr.pdf."
[4] "http://wiki.ros.org/rtabmap."
[5] "https://en.wikipedia.org/wiki/list_of_slam_methods."
[6] "http://ieeexplore.ieee.org/document/1308008/."
[7] "https://arxiv.org/pdf/1606.05830.pdf."
[8] "http://wiki.ros.org/depthimage_to_laserscan."
[9] "https://arxiv.org/ftp/arxiv/papers/1710/1710.02726.pdf."
[10] "https://classroom.udacity.com/nanodegrees/nd209/parts/dad7b7cc-9cce-4be4-876e-30935216c8fa/modules/aec2781f-e368-4e1e-9aef-d46aeee55354/lessons/0f504827-ab9c-4280-913f-413e4df602be/concepts/333568e8-649e-4548-8337-49b5b36ddd25."
[11] "https://pdfs.semanticscholar.org/533d/cfc1cb34897d3ba2e86c4b2e95fc8778148d.pdf."