Map My World Robot

Pramod Kumar

Abstract—In this project, SLAM(Simultaneous Localization and Mapping) is used for generate a map in the given environment. the RTAB (real time appearance based mapping) methodology has been used to complete this project. Firstly, 2d occupancy grid and 3d octomap have been generated from a given simulated environment. In next step, individual simulated environment has been created for mapping too. a procedure called Loop Closure is utilized to decide if a robot has seen a location previously or not in appearance based technique. In this process, it is seen that RTAB-Map is improved for expansive scale and long term SLAM by utilizing various methodologies to take into consideration loop closure to be done continuously and the outcomes portray that it very well may be a magnificent answer for SLAM to create robots that can outline condition in both 2d and 3d.

Index Terms—Robot, SLAM, RTAB-Map.

1 Introduction

In SLAM, simultaneously a robot need to localize itself with this map when it generating the map. This is more challenging problem compare than mapping or localization while neither the robot poses nor map are given. The measurements of robots motion might be have noise, and due to this noise robots pose and map will have uncertainty and this uncertain error will be correlated between robots pose and map. The map will be more accurate if localization is more accurate. the SLAM issue is to process an estimate of the agents location and a map of the environment over discrete time steps using observations through multiple data of multiple sensors. In this project, two simulation environments is generated for SLAM perforation. The robot is able to localize itself and map the 3d world. The first simulation environment called benchmark environment which is kitchen-dining (Figure 1) and second one is cafeteria called pramod-cafe(Figure 2).

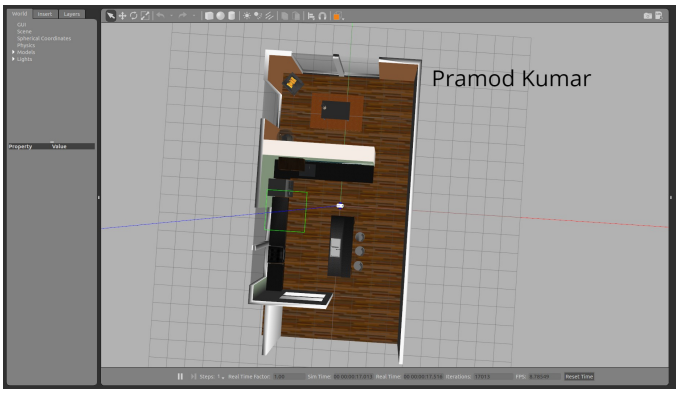

Fig. 1. Kitchen-Dining World

2 Background

SLAM algorithms for the most part fall into 5 classifications:

1) Extended Kalman Filter SLAM (EKF)
2) Sparse Extended Information Filter (SEIF)
3) Extended Information Form (EIF)

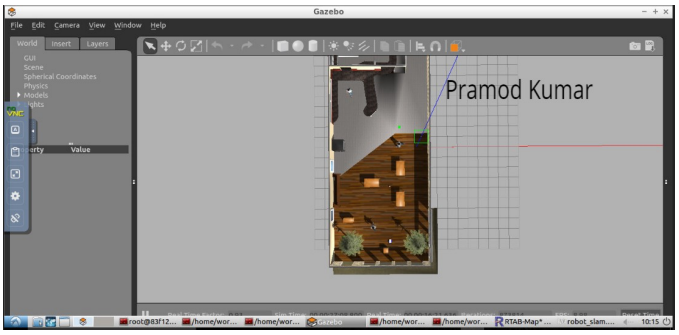

Fig. 2. PramodCafe World

4) FastSLAM
5) GraphSLAM

The two most helpful ways to deal with SLAM are Grid based FastSLAM and GraphSLAM furthermore, these two algorithms will be talked about here.

2.1 Grid based FastSLAM

The FastSLAM algorithm uses a custom particle filter approach to solve the full SLAM problem with known correspondence. Using particles, FastSLAM estimates a posterior over the robot's path along with the map. Each of these particles hold the robot's trajectory which gives an advantage to SLAM to solve the problem of mapping with known poses. In addition to the trajectory, each particle holds a map and each feature of the map is represented by a local Gaussian.

With the FastSLAM algorithm, the problem is now divided into two separate independent problems, each of which aims to solve the problem of estimating features of the map. To solve these independent mini-problems, FastSLAM will use the low dimensional Extended Kalman Filter. While map features are treated independently, dependency only exists between robot pose uncertainty. This custom approach of representing the posterior with particle filter and Gaussian is known by *Rao-Blackwellized particle filter approach* [**?**]. The Grid based FastSLAM is really an extension of FastSLAM and it adapts FastSLAM to grid maps.

With grid mapping algorithm, the environment can be modeled using grid maps without predefining any landmark position. So by extending the FastSLAM algorithm to occupancy grid maps, the SLAM problem can now be solved in an arbitrary environment. While mapping the real world environment, mobile robots equipped with range sensors can be used and the FastSLAM algorithm can be extended to solve the SLAM problem in terms of grid maps.

$$P(x_{0:t}, m|z_{1:t}, u_{1:t}) = P(x_{0:t}|z_{1:t}, u_{1:t}) * P(m|x_{1:t}, z_{1:t}) \quad (1)$$

The first term in the RHS represents the robot trajectory where just as in FastSLAM, with the grid based FastSLAM, each particle holds a guess of the robot's trajectory.

The second term represents a map where each particle maintains its own. The grid based FastSLAM algorithm will update each particle by solving the mapping with known poses problem using the *Occupancy grid mapping algorithm*.

2.1.1 Grid based FastSLAM techniques

Adapting the FastSLAM algorithm to grid maps is altered in the grid based FastSLAM algorithm. Since the grid based FastSLAM algorithm uses a particle filter approach and represents the world in terms of grid maps, both MCL (Monte Carlo Localization) and Occupancy Grid Mapping algorithm are combined. Now three different techniques are needed which are represented by 3 probability functions to adapt FastSLAM to grid mapping. These techniques are known as:

1) Sampling Motion $P(x_t|x_{t-1}^{[k]}, u_t)$: Estimates the current pose given the k^{th} particle's previous pose and controls u (MCL)
2) Map Estimation $P(m_t|z_t, x_t^{[k]}, m_{t-1}^{[k]})$: Estimates the current map given the current measurements, the current k^{th} particle's pose and the previous k^{th} particle map (use Occupancy Grid Mapping)
3) Importance Weight $P(z_t|x_t^{[k]}, m^{[k]})$: Estimates the current likelihood of the measurement given the current k^{th} particle pose and the current k^{th} particle map (MCL).

The sampling motion, map estimation and importance weight techniques are the essence of the grid based FastSLAM algorithm. Grid based FastSLAM implements them to estimate both the map and the robot's trajectory, given the measurements and the control. The grid based FastSLAM algorithm looks very similar to Monte Carlo localization algorithm with some additional statements concerning the map estimation.

Algorithm 1 Grid based FastSLAM

procedure GRID BASED FASTSLAM(X_{t-1}, u_t, z_t)
  $\bar{X}_t = X_t = \phi$
  for k = 1 to M **do**
    $x_t^{[k]}$ = sample-motion-model($u_t, x_{t-1}^{[k]}$)
    $w_t^{[k]}$ = measurement-model-map($z_t, x_t^{[k]}, m_{t-1}^{[k]}$)
    $m_t^{[k]}$ = updated-occupancy-grid($z_t, x_t^{[k]}, m_{t-1}^{[k]}$)
    $\bar{X}_t = \bar{X}_t + < x_t^{[k]}, m_t^{[k]}, w_t^{[k]} >$
  for k = 1 to M **do**
    draw i with probability $\propto w_t^{[i]}$
    add $< x_t^{[i]}, m_t^{[i]} > to X_t$
  Return X_t

2.2 Graph SLAM

Graph SLAM is a SLAM algorithm that solves the full SLAM problem. This means that the algorithm recovers the entire path and map, instead of just the recent pose and map. This difference allows it to consider dependencies between current and previous poses. One of the benefits of graph SLAM is the reduced need for significant on-board processing capability. Another is graph SLAM's increased accuracy over fast SLAM. Fast SLAM uses particles to estimate the robot's most likely pose. However, at any point in time, it is possible that there is not a particle in the most likely location. In fact, chances are slim to none especially in large environments. Since graph SLAM solves the full SLAM problem, this means that it can work with all of the data at once to find the optimal solution.

In graph SLAM, 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. . . . . 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.

The Maximum Likelihood Principle (MLE) is used to optimize the graph. When applied to SLAM, likelihood tries to estimate the most likely configuration of state and feature locations given the motion and measurement observations. The measurement update at time step t is given by

$$\bar{z^t} := x_t + m_t^{(i)} \quad (2)$$

which represents for instance a laser range finder measuring the distance to the landmark m (i) . Equivalently, a motion update can be defined as

$$\bar{x^t} := x_{t-1} + u_t \quad (3)$$

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 probability distributions are given by

$$p_u(x_t) = \frac{1}{\sigma_m \sqrt{2\pi}} e^{-(z_t - \bar{z_t})^2 / 2\sigma_m^2} \quad (4)$$

$$p_m(z_t) = \frac{1}{\sigma_u \sqrt{2\pi}} e^{-(x_t - \bar{x_t})^2 / 2\sigma_u^2} \quad (5)$$

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

$$J_{GraphSLAM} = \sum_t \left(\frac{z_t - \bar{z_t}}{\sigma_m} \right)^2 + \sum_t \left(\frac{x_t - \bar{x_t}}{\sigma_u} \right)^2 \quad (6)$$

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.

In real world, most systems are multi-dimensional and to tackle such scenarios, matrices and covariances must be used. The state and measurement are given by x_t and z_t. The constraints are given by

$$v_t := z_t - h(x_t, m_t) \tag{7}$$

$$w_t := x_t - g(x_{t-1}, u_t) \tag{8}$$

where h() and g() represent the measurement and motion functions and Q_t and R_t are the covariances of the measurement and motion noise. The multidimensional formula for the sum of all constraints is given by

$$J_{GraphSLAM} = x_0^T \Omega x_0 + \sum_t \left(w_t^T * R_t^{-1} * w_t + v_t^T * Q_t^{-1} * v_t \right) \tag{9}$$

The goal of graph SLAM is to create a graph of all robot poses and features encountered in the environment and the most likely robot's path and map of the environment. This task can be broken down into two sections. The *front-end* and the *back-end*.

2.2.1 Front End vs Back End

The front end of graph SLAM looks at how to construct the graph, using the odometry and sensory measurements collected by the robot. This includes interpreting sensory data, creating the graph and continuing to add nodes and edges to it as the robot traverses the environment. Naturally the front end can differ greatly from application to application depending on the desired goal, including accuracy, the sensor used and other factors e.g. the front end of a mobile robot applying SLAM in the office using a Laser Range finder would differ greatly from the front end of a vehicle operating on a large outdoor environment and using a stereo camera. The front end of graph SLAM also has the challenge of solving the data association problem. In simpler terms, this means accurately identifying whether features in the environment have been previously seen.

The back end of graph SLAM is where the magic happens. The input to the back end is the completed graph with all of the constraints and the output is the most probable configuration of robot poses and map features. The back end is an optimization process that takes all of the constraints and find the system configuration that produces the smallest error. The back end is a lot more consistent across applications. The front end and the back end can be completed in succession or can be performed iteratively, with a back end feeding an updated graph to the front end for further processing.

2.2.2 Using RTAB-Map for 3D Graph SLAM

RTAB-Map (Real Time Appearance Based Mapping) [**?**] is a graph based SLAM approach. Appearance based SLAM means that the algorithm uses data collected from vision sensors to localize the robot and map the environment. In appearance based methods, a process called *Loop Closure* is used to determine whether the robot has seen a location before. As the robot travels to new areas in its environment, the map is expanded and the number of images that each new image must be compared to increases. This causes the loop closure to take longer with the complexity increasing linearly. RTAB-Map is optimized for large scale and long term SLAM by using multiple strategies to allow for loop closure to be done in real time. Figure 3 shows the block diagram of the front end and the back end.

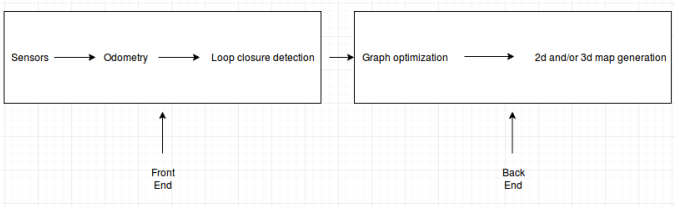

Fig. 3. RTAB-Map Front end and Back End block diagram

The front end of RTAB-Map focuses on sensor data used to obtain the constraints that are used for feature optimization approaches. Although landmark constraints are used for other graph SLAM methods like the 2d graph SLAM, RTAB-Map does not use them. Only odometry constraints and loop closure constraints are considered here. The odometry constraints can come from wheel encoders, IMU or visual odometry. Visual odometry is accomplished by 2d features such as Speeded Up Robust Features (SURF). RTAB-Map is appearance based with no metric distance information. It can use a single monocular camera to detect loop closure. For metric graph SLAM, RTAB-Map requires an RGB-D camera or a stereo camera to compute the geometric constraint between the images of a loop closure. A laser range finder can also be used to improve or refine this geometric constraint by providing a more precise location. The front end also involves graph management, which includes node creation and loop closure detection using **Visual Bag of Words**.

The back end of RTAB-Map includes graph optimization and assembly of an occupancy grid from the data of the graph. Loop closure detection is the process of finding a match between the current and previously visited locations in SLAM. There are two types of loop closure detection:

a) **Local Loop closure detection**: More probabilistic SLAM methods use Local loop closure detection, where matches are found between a new observation and a limited map region. The size and location of this limited map region is determined by the uncertainty associated with the robot's position. This type of approach fails if the estimated position is incorrect.

b) **Global Loop closure detection**: In this approach, a new location is compared with the previously visited locations. If no match is found, the new location is added to memory. As the map grows and more locations are added to the memory, the amount of time to check whether the location has been previously seen increases linearly. If the time it takes to search and compare new images to the one stored in memory becomes larger than the acquisition time, the map becomes ineffective. RTAB-Map uses a global loop closure

approach combined with other techniques like Visual bag of words, quantization, graph optimization techniques like Tree Based Network Optimizer (TORO) or General Graph Optimization (G2O) to ensure that the loop closure process happens in real time.

3 Scene and Robot configurations

The ROS package `slam_project` deploys the RTAB-Map to perform SLAM on two environments. The first environment is an environment provided by Udacity as a part of this research project and is named *Kitchen-Dining*. The second environment is a custom made environment of a cafeteria named *pramodr_cafe*. Just like other robotics project, this project has been organized into different folders containing the meshes, Gazebo SDF files, scripts, robot model URDFs, configuration files and launch files.

The URDF folder contains the files pramod_bot.xacro defining the links and joints of the robot model used for physics simulation and visualization as well as the file pramod_bot.gazebo specifying the Gazebo plugins for differential drive, RGB-D camera and laser range finder. Section 3.1 goes into more details about the robot model. Besides the provided kitchen-dining world, the *worlds* directory contains a file named **pramod_cafe.world** defining a custom indoor cafeteria environment in SDF format. Section 3.2 discusses this in more 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

Figure 1 and 2 depicts the robot model inside the Gazebo simulation environments of both the provided and the custom made one. The URDF specification can be found in the file pramod_bot.xacro. The transform tree associated with the robot is shown in Figure 4.

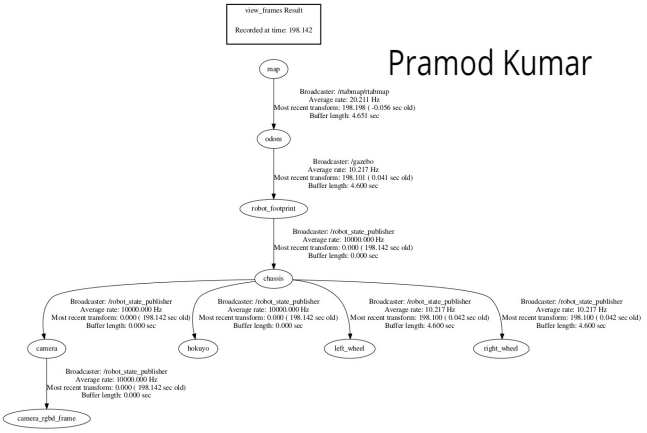

Fig. 4. Transform Tree of the Robot - pramod_bot

For this project, the robot from [**?**] was taken as a starting point. The xacro file provides the shape and size of the robot in macro format. For the pramod_bot, a fixed base is used. A single link, with the name defined as ”chassis”, encompassed the base as well as the caster wheels. Each link has specific elements, such as the inertial or the collision elements. The chassis is a cuboidal (or box), whereas the casters are spherical as denoted by their ”geometry” tags. Each link (or joint) has an origin (or pose) defined as well. Every element of that link or joint will have its own origin, which will be relative to the link's frame of reference.

For this base, as the casters are included as part of the link (for stability purposes), there is no need for any additional links to define the casters, and therefore no joints to connect them. The casters do, however, have friction coefficients defined for them, and are set to 0, to allow for free motion while moving.

Two wheels were attached to the robot. Each wheel is represented as a link and is connected to the base link (the chassis) with a joint. For each wheel, a ”collision”, ”inertial” and ”visual” elements are present. The joint type is set to ”continuous” and is similar to a revolute joint but has no limits on its rotation. It can rotate continuously about an axis. The joint will have it's own axis of rotation, some specific joint dynamics that correspond to the physical properties of the joint like ”friction”, and certain limits to enforce the maximum ”effort” and ”velocity” for that joint. The limits are useful constraints in regards to a physical robot and can help create a more robust robot model in simulation as well. 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 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_project/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 pramod_bot.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 /scan topic. A graphical view of the ROS topics and nodes is shown in figure 5 and a closeup of the robot model is depicted in Figure 6.

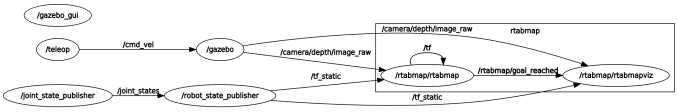

Fig. 5. RQT graph of the topics after all the nodes are launched

3.2 Design of the World

As the second part of the project, the custom world is created in Gazebo. This world is based on the cafe model

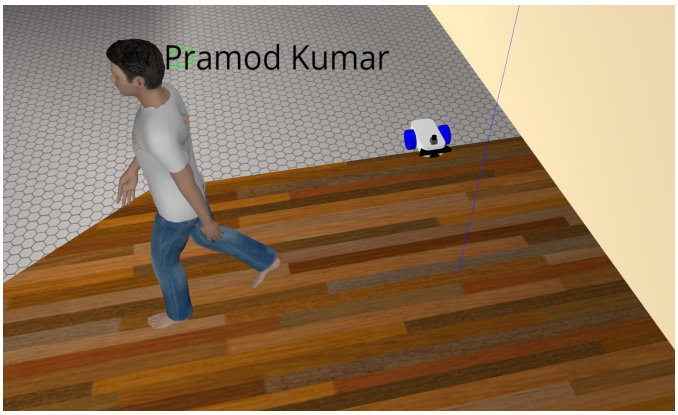

Fig. 6. Close up view of the robot model

inside Gazebo database. The base model is customized with different objects like tables, beer can, people, trees etc. These objects serve as distinctive elements in the base world for the robot to distinguish and map. In this world, the kitchen cannot be entered by the robot. A bird's eye view of this world is provided in Figure 7.

Fig. 7. Bird's eye view of pramodcafe

3.3 Launch file configuration

Four launch files are required for a successful mapping of the environments in simulation. The gazebo simulation environment is (kitchen-dining or cafe) specified in the **world.launch** file. The **teleop.launch** file launches the teleop keyboard which is required for moving the robot in the simulation world. The **mapping.launch** file is used to start the RTAB-Map node. This node is used for loop closure detection using the ORB-SLAM algorithm. ORB-SLAM is a versatile and accurate Monocular SLAM solution able to compute in real-time the camera trajectory and a sparse 3D reconstruction of the scene in a wide variety of environments, ranging from small hand-held sequences to a car driven around several city blocks. It is able to close large loops and perform global re-localization in real-time and from wide baselines. Finally, the **rviz.launch** file starts visualization of the rover, sensor data, as well as map and camera topics in RViz.Figure 8 depicts RViz view of the world at the starting point.

During the mapping of the environment, the mapping data is saved in the rtabmap.db database. The localization.launch file can be started in order to localize the robot during the run.

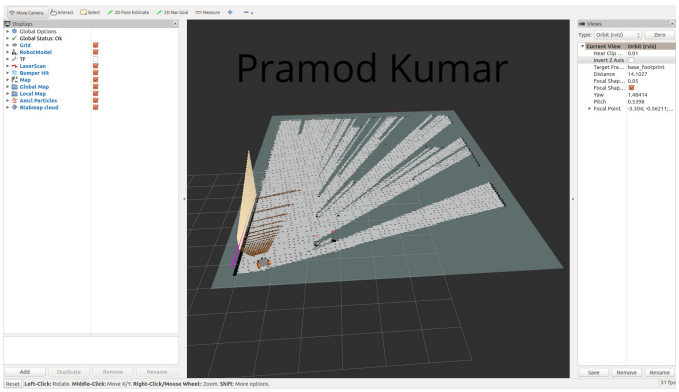

Fig. 8. Starting point of the robot as seen in RViz

4 Results

The mapping was done by the robot controlled by the teleop keyboard. In order to be able to have more than 3 loop closure detection, which was the project's benchmark, the robot was navigated through the full environment of both the worlds so that it could collect more images.

4.1 Kitchen-Dining World

The mapping run in the provided world ended with 66 global loop closures. This file has a size of 315 MB and is named as rtabmap_kitchen_dining.db. Figure 9 shows the robot's trajectory as well as the 2d occupancy grid map of the kitchen_dining world.

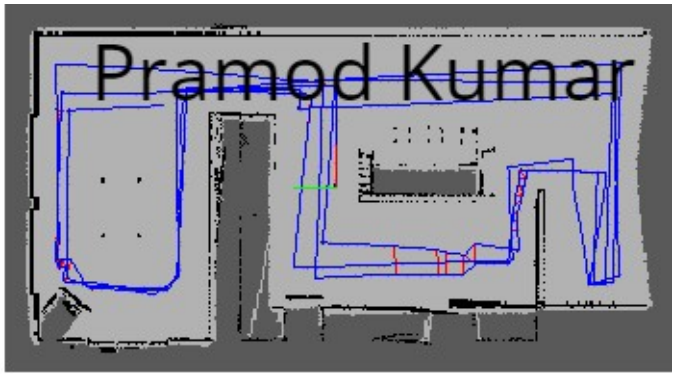

Fig. 9. Robot's trajectory and 2d occupancy grid map of the Kitchen-Dining world

At the end of the multiple passes, a well structured 3d point cloud map was created by using the *Export 3d Clouds* functionality. Figure 10 depicts the reconstructed point cloud data. It can be seen that most features in the world like the chairs and tables are reconstructed properly and are distinctive. Figure 11 shows the RViz result of the same world after the end of the mapping task.

At the end of the map, the loop closures can be seen in the rtabmap_kitchen_dining.db. Figure 12 shows one of them.

4.2 pramodcaf World

In the custom made world - **pramod_cafe.world**, the robot performed well. As the robot is very short, some of the

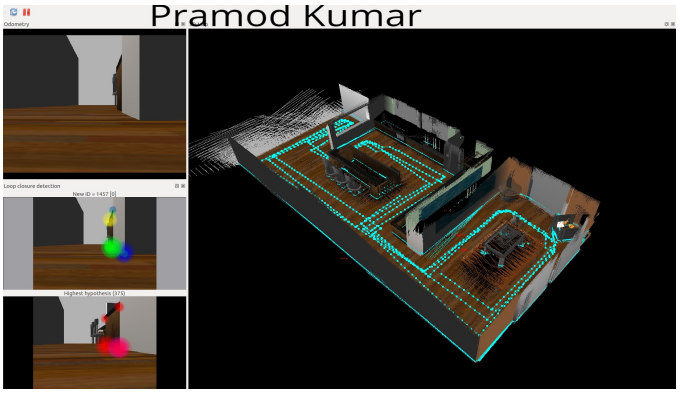

Fig. 10. Reconstructed Point cloud data in RTAB-Map viewer of the Kitchen-Dining world

Fig. 11. Rviz view of the Kitchen-dining world

taller objects like the people, trees are not fully mapped. The kitchen also could not be traveled by the robot. Figure 13 and 14 depicts the RTAB-map view and the RViz view of this world respectively at the end of the mapping.

5 Discussion

In this experiment, a robot was navigated using a keyboard around two environments. One common problem was that the robot would not move forward, and occurred during loop closure. To fix this, the number of max features could be reduced and the minimum inliers could be reduced. It was also seen that rotating the robot resulted in regaining the ability to move. The results in the kitchen environment was considerably better than the cafe environment. This was due to the excess amount of features in the kitchen world. With chairs, tables, and windowed walls, the robot was able to calculate features with ease and thus have much better visual odometry. The cafe world did have some objects but gaps were left in some areas to test how the robot would perform when features were lax. Also, the cafe had some areas were the colors of the surfaces were similar and this seemed to also deteriorate the results. It was seen that the kitchen model performed better and portrayed that rich features are vital in the operation of RTABMap. It was

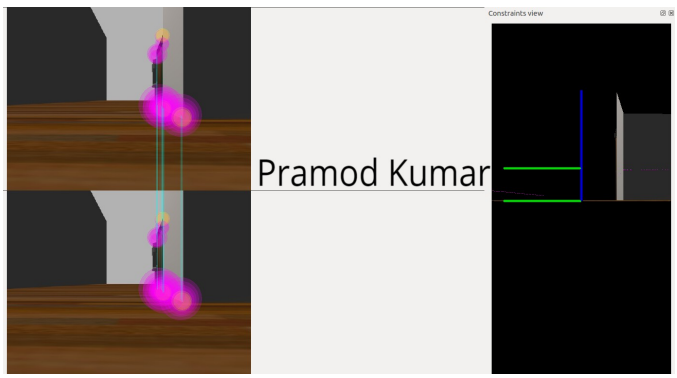

Fig. 12. Loop Closure detection

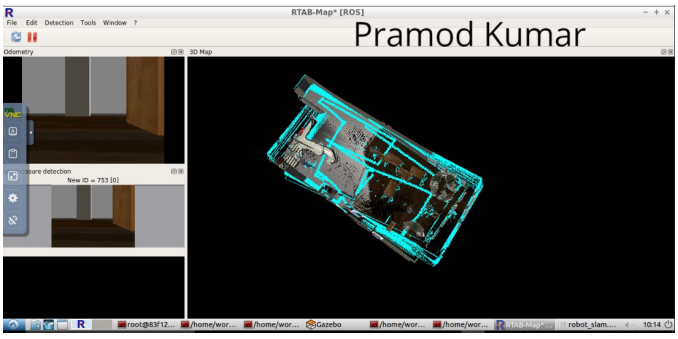

Fig. 13. Reconstructed Point cloud data in RTAB-Map viewer in the pramodcafe world

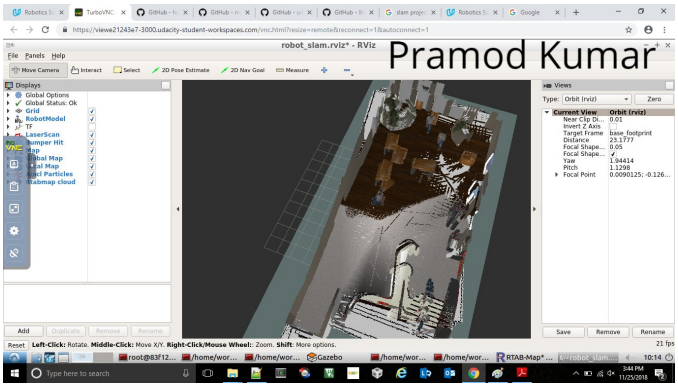

Fig. 14. RViz view of the pramodcafe world

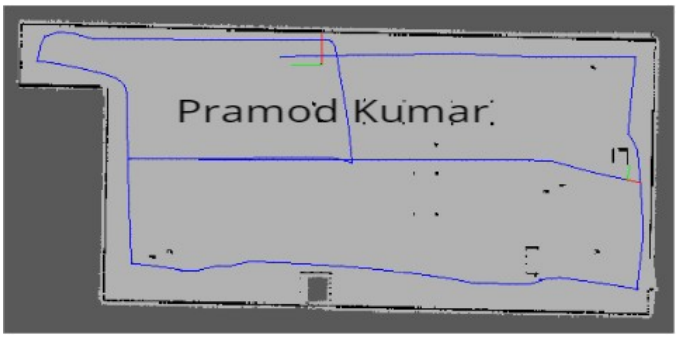

Fig. 15. Robots trajectory and 2d occupancy grid map of the pramodcafe

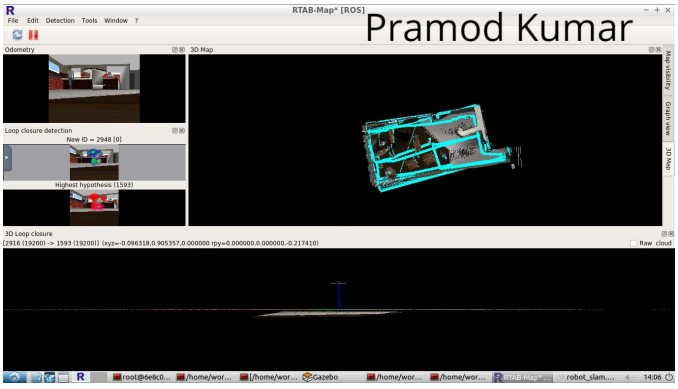

Fig. 16. Loop Closure detection 1

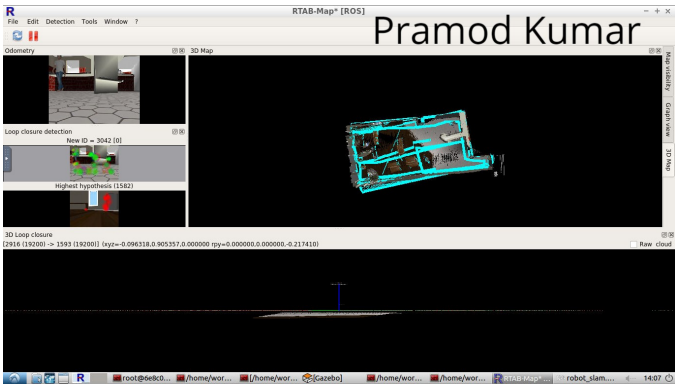

Fig. 17. Loop Closure detection 2

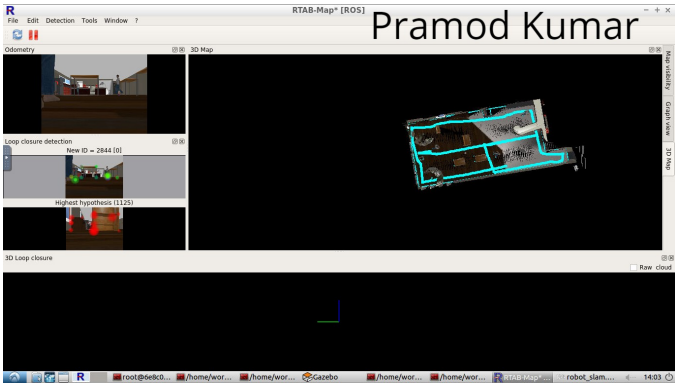

Fig. 18. Loop Closure detection 3

also seen that incorrect loop closure quickly leads to an experiment failure and further uncertainty.

6 Conclusion / Future work

The operation of SLAM is a very difficult and sensitive technique. The environment must fit the criteria of the SLAM algorithm for it to run at its highest performance. RTABMap proved to be very viable in feature rich environments and produce very meaningful results in a robot equipped with a laser scanner, an RGBD camera, and odometry. As previously stated, the laser scanner can be removed, and then this robot would be able to perform SLAM for a very cheap price. This type of SLAM would be very useful in mapping unknown areas for intelligence. Furthermore, much of this data could be processed and tested against Lidar datasets for object recognition as previously done in a Udacity Project. This type of SLAM could also be applied to a real household robot using the NVIDIA Jetson TX2. It was found difficult to implement simulations but it may be a worthwhile experiment to lower the computation and attempt to map a feature rich area.