# Mapping: Map my world

Mohit Chaturvedi

**Abstract**—A discussion is presented on the implementation of a Simultaneous Localization and Mapping (SLAM) algorithm towards generating a map of an indoor environment by a mobile robot while it navigates though the setting. The robot model attached with a Hokuyo laser range finder and a Kinect RGB-D camera uses a graph based SLAM algorithm - Real Time Appearance Based Mapping (RTAB-Map) to map the surroundings. Performance in mapping the environment is first analyzed on a pre-set kitchen dining Gazebo world and later it is tested on a user built room Gazebo world. Finally, the 2D occupancy grid map and the 3D octomap generated by RTAB-Map, for both scenarios, are examined for loop closures and are evaluated based on their accuracy in portraying actual environment characteristics.

**Index Terms**—Robot, Mapping, SLAM, ROS, RTAB-Map, Graph-SLAM, IEEEtran, Udacity, LaTeX

## 1 Introduction

In the field of robotics, localization is defined as the challenge in determining the pose of a mobile robot in a mapped environment. In situations where map of the environment is not available, the task becomes more complex as the robot has to localize itself while creating the map. Noisy sensor data results in inaccurate robot's motion and measurements readings leading to erroneous estimates of robot poses which in turn make mapping process inaccurate too. An approach to solve such problem is to implement SLAM (Simultaneous Localization and Mapping) algorithms. The accuracy in mapping depends on the accuracy in localization and vice-versa which makes successful SLAM implementation a challenge. Out of five different SLAM algorithms available, a graphSLAM approach called RTAB-Map (Real Time Appearance Based Mapping) is chosen for this project. RTAB-Map provides several advantages over other SLAM approaches in terms of speed and memory management, custom development tools for information analysis and quality of documentation.

## 2 Background

There area two major challenges with the mapping process:

- Unknown Map and Poses.
  Estimating the map on unknown poses is a challenging task because of the high number of variables but is solvable by applying occupancy grid mapping algorithm. Estimating the map and also the poses relative to it is an even bigger challenge solvable by using SLAM.
- Huge Hypothetical Space.
  The space of all the possible maps that can be formed during mapping is called hypothetical space. If the robot is deployed in an open environment which provides an infinite number of objects to be detected, the maps get defined in a continuous space creating a highly dimensional hypothetical space. Occupancy grid algorithm can provide a discrete approximation of the map. However, that would still result in the number of possible maps to be very high. The base filtering approach used in localization to estimate the posterior pose can be extended to be used for accommodating the huge hypothetical space in mapping.

Other challenges include intensive on-board computations, filtering noisy sensory data, perpetual ambiguity and resolving incremental odometry errors accumulated over time when the robot travels in a cyclic manner.

The following two SLAM algorithms are the most popular to solve these problems.

### 2.1 Grid-based FastSLAM

As a landmark-based algorithm, FastSLAM extends the use of custom particle filter algorithm along with a Extended Kalman Filter to solve the full SLAM problem. The posterior over the robot's path along with the map is estimated using a particle filter. Each of these particles hold the robot's trajectory which provides known poses and makes the mapping process easier. These particles also hold the map with their independent features that are solved using a low dimensional Extended Kalman Filter.

Grid-based FastSLAM is an extension of FastSLAM using occupancy grid mapping which outputs the map in grid cells representing occupied, free and, unknown areas. With the environment being modeled in grid maps, the necessity of having pre-defined landmarks can be avoided making it possible to use this approach in any arbitrary environment. The algorithm estimates robot's trajectory using a custom particle filter and estimates the map with now known poses using the occupancy grid mapping algorithm. Adapting FastSLAM to Grid-based FastSLAM requires following steps:

- Sampling Motion: Current pose estimation with current controls and previous particle pose given.
- Map Estimation: Current map estimation with previous particle map, current measurements and current particle pose given.
- Importance Weight: Current likelihood of measurement estimation with current particle pose and current particle map given.

Sampling motion and importance weight steps can be solved by particle filter algorithm whereas the map estimation step can be solved using occupancy grid mapping.

### 2.2 GraphSLAM

GraphSLAM algorithm solves the full SLAM problem by recovering the entire path and map instead of the most recent pose and map. It uses graphs to represent robot's poses and the environment. This graph contains nodes that represent robot poses or landmarks at different time steps and the edges that connect these nodes represent the motion and measurement constraints. Motion constraints are between the two robot poses and measurement constraints are between a robot pose and a feature in the environment. One of the major advantages of using GraphSLAM is its improved accuracy over FastSLAM. With particle filter approach, at any point in time, there's always a possibility that there is no particle at the most likely location. Also, a limited amount of information stored by a finite number of particles can lead to errors in estimation. GraphSLAM on the other hand solves the full SLAM problem by working with all its data at once to find the most optimal solution.

The goal of GraphSLAM is to find an optimal configuration of nodes such that the errors represented by the constraints binding them together are minimized. The whole process is split into two parts: the front-end and the back-end. The front-end deals with constructing the graph by adding nodes and edges given the odometry and sensory measurements collected by the robot as it traverses the environment. The back-end takes in the completed graph with all the constraints and outputs the most probable configuration of robot poses and map features. It's the optimization process that finds the system configuration with the smallest error. The back-end process is usually more consistent across various applications while the front-end can change greatly depending on the environment. The algorithm used in this project is RTAB-Map (Real Time Appearance Based Mapping) which is a variant of Graph-based SLAM approach that collects data from a vision sensor to localize the robot and map the environment. RTAB-Map is optimized for large-scale and long-term SLAM and offers several benefits over a traditional GraphSLAM in terms of speed, memory management and, information analysis.

### 2.3 RTAB-Map

Just like the two parts of a traditional 2D Graph-Based SLAM approach, RTAB-Map also has a front-end and a back-end. The front-end focuses on vision sensor data used to obtain the constraints that are used for feature optimization approaches. Landmark constraints used by other Graph-Based SLAM approaches are ignored by RTAB-Map and it only considers odometry and loop closure constraints. Since it's an appearance based algorithm with no metric distance information, the visual odometry is computed using 2D features such as Speeded Up Robust Features (SURF). A single monocular camera can be used to detect loop closures however, for a metric graph SLAM an RGB-D camera or a stereo camera is required to compute the geometric constraints between the images of a loop closure. In this project, a Kinect sensor is used along with Hokuyo laser range finder that refines geometric constraints by providing a more precise localization. The front-end also handles graph management by nodes creation and loop closure detection using bag-of-words approach.

The back-end acts by adding a new constraint to the map's graph whenever a loop closure hypothesis is accepted and then optimizes the graph by minimizing errors in the map. There are three graph optimization algorithms that are supported by RTAB-Map: Tree-based network optimizer (TORO), General Graph Optimization (G2O), or Georgia Tech Smoothing and Mapping (GTSAM). These optimizations use node poses and link transformations as constraints. As a loop closure is detected, errors induced by the odometry propagate to all links, correcting the map.

## 3 Scene and Robot Configuration

### 3.1 Scene

Testing and analysis of RTAB-Map is done on two different environments. The first scene is a pre-loaded Gazebo world called `kitchen_dining` as shown below.

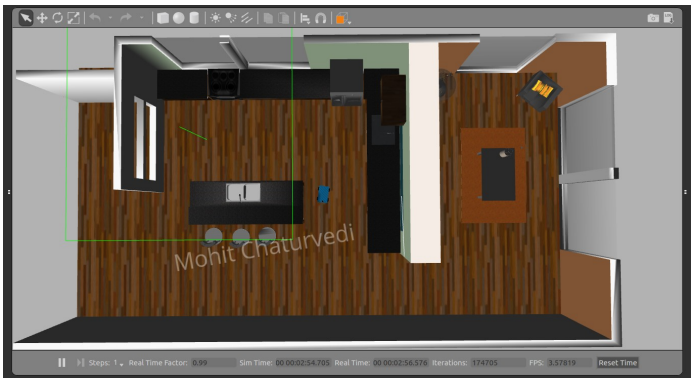



Fig. 1: Top view of `Kitchen_Dining` world

The other scene is a custom build world called `room` which consists of a number of objects, obtained from the Gazebo library, placed inside the walls. The placement of objects provide enough room for the robot to run around and explore the environment.

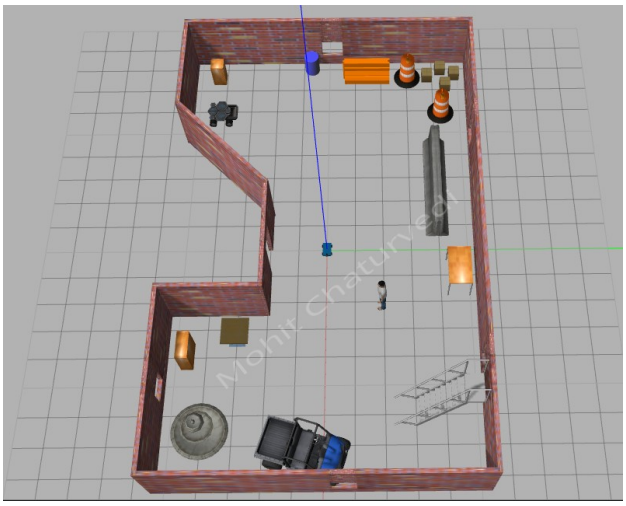



Fig. 2: Top view of `room` world

### 3.2 Robot Configuration

The custom `slam_bot` design has a cuboid chassis with a flatbed on top. There are two pair of wheels in the front and the back to drive the robot. A Kinect camera sensor is attached to front of the chassis under the flatbed. A Hokuyo laser sensor is installed at the top. A `skid_steer_drive_controller` Gazebo plugin is used to provide mobility as it is a four wheeled model. An additional optical link is added to the xacro file of the robot model so as to provide rotation to the camera frame. Without this, the camera points upwards and not in the direction of travel. An image of the transform tree is provided at the last page of this document.

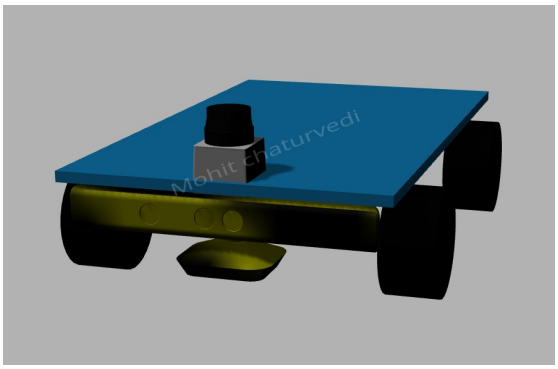


Fig. 3: `slam_bot` robot model

## 4 Results

### 4.1 RTAB-Map Loop Closure Detection

For both `kitchen_dining` world and the `room` world, the robot captured data about the environment by making 3 loops around it. After the 2D and 3D maps were generated, the visual analysis of loop closures was done with the help of rtabmap-databaseViewer tool. The yellow circles indicate features detected on objects from the detection algorithm. Purple circles indicate where the two images have features in common which leads to loop closures.

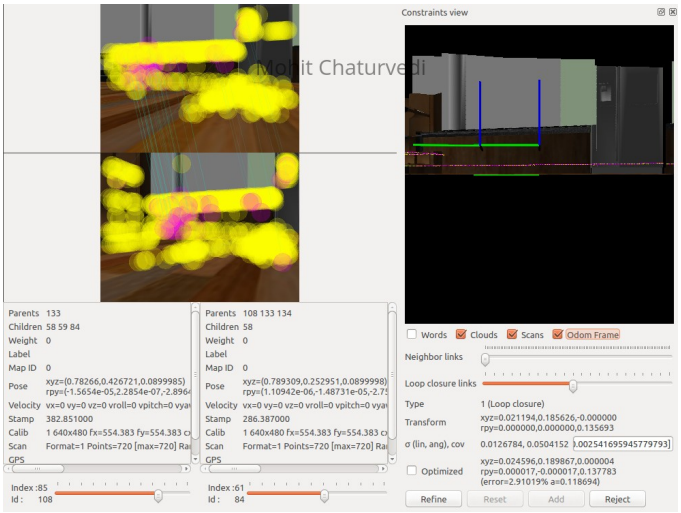


Fig. 4: Loop closures from the `kitchen_dining` world

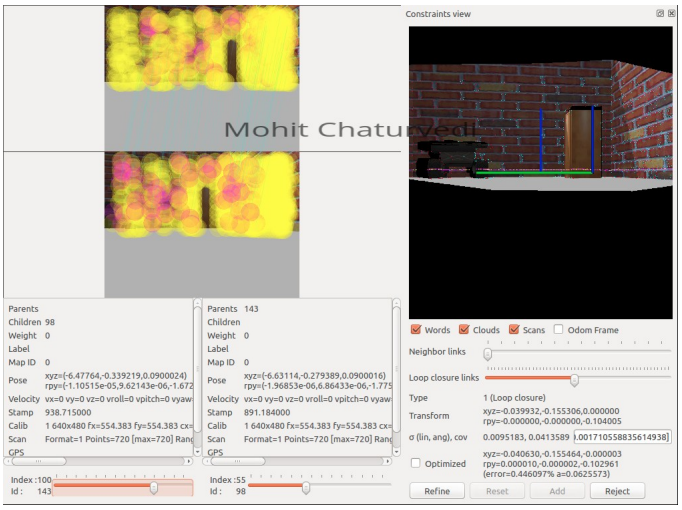
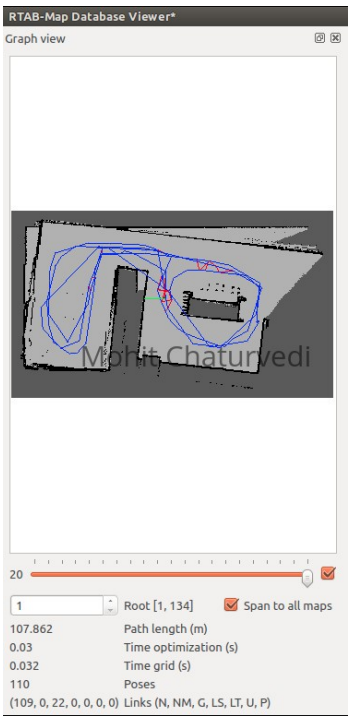


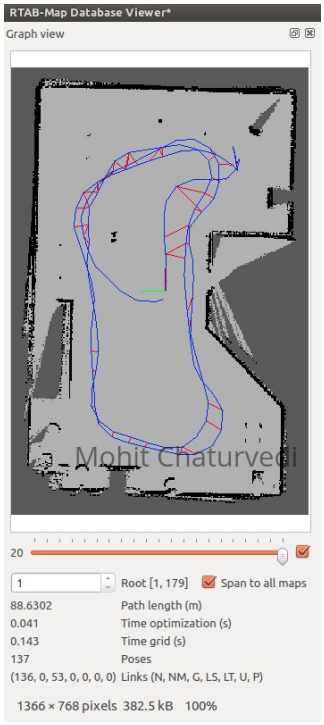
Fig. 5: Loop closures from the `room` world

### 4.2 Mapping

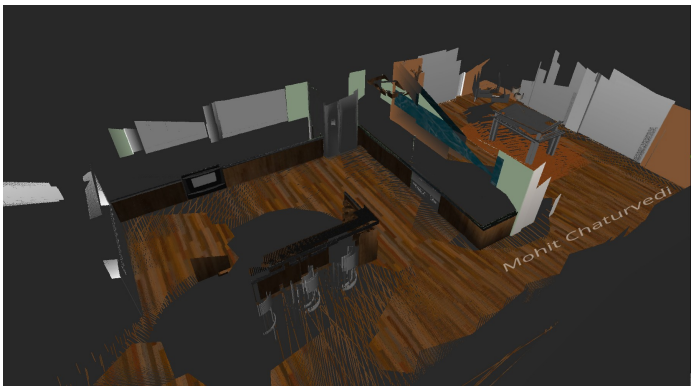
The 2D grid maps of both environments are based on the scans of laser range finder and show the navigation path of robot whereas 3D maps are visualized as point clouds from the RGB-D camera and portray environmental characteristics. The following images obtained from rtabmap-databaseViewer visualization tool show 2D and 3D maps for both `kitchen_dining` and `room` Gazebo worlds.



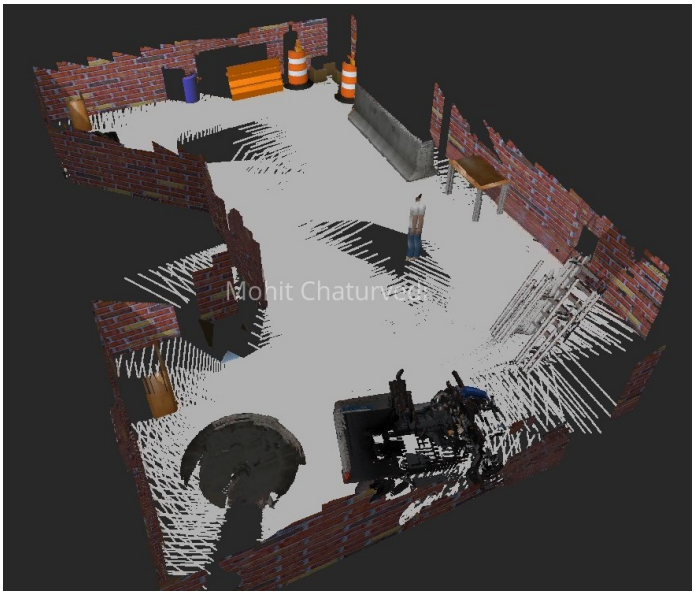
(a) `kitchen_dining` 2D map



(b) `room` 2D map




(a) `kitchen_dining` 3D map



(b) `room` 3D map

## 5 Discussions

Overall, both the environments were mapped in 2D and 3D to a decent degree. The Kitchen-dining environment was more accurately mapped by the robot as it contains more feature rich objects as compared to the room world. A certain number of false positives of loop closures were detected in the initial setting of the room environment because of the position of some of the objects and the use of duplicate objects. Results were better when these objects were moved to a different location and duplicate objects were removed. A more accurate map of these worlds can be attained with proper navigation of the robot around the environment and with the use of more on-board strategically placed sensors.
One of the major challenges in setting up the project in ROS was to map the various topics to its subscribers correctly. ROS tools such as `roswtf`, `rqt_graph`, `rqt_image_view` were essential in debugging and setting up the information pipeline.

## 6 Future Work

Since both the environments in simulation used static objects, it'll be interesting to test the performance of RTAB-Map in dynamic settings were objects can change their position any time. It is expected that changing object locations might produce erroneous constraints resulting in less accurate 3D maps. Use of an advanced object detection algorithm with neural networks might be solution space for such a task. It might also be helpful in resolving erroneous loop closure detection because of duplicate object present in the scene.
The performance of this SLAM approach can be tested on actual hardware running Jetson TX2. Along with path planning algorithms, it'll be fascinating to deploy simulation solutions on a real robot hardware platform and to test their performance in indoor and outdoor environments.

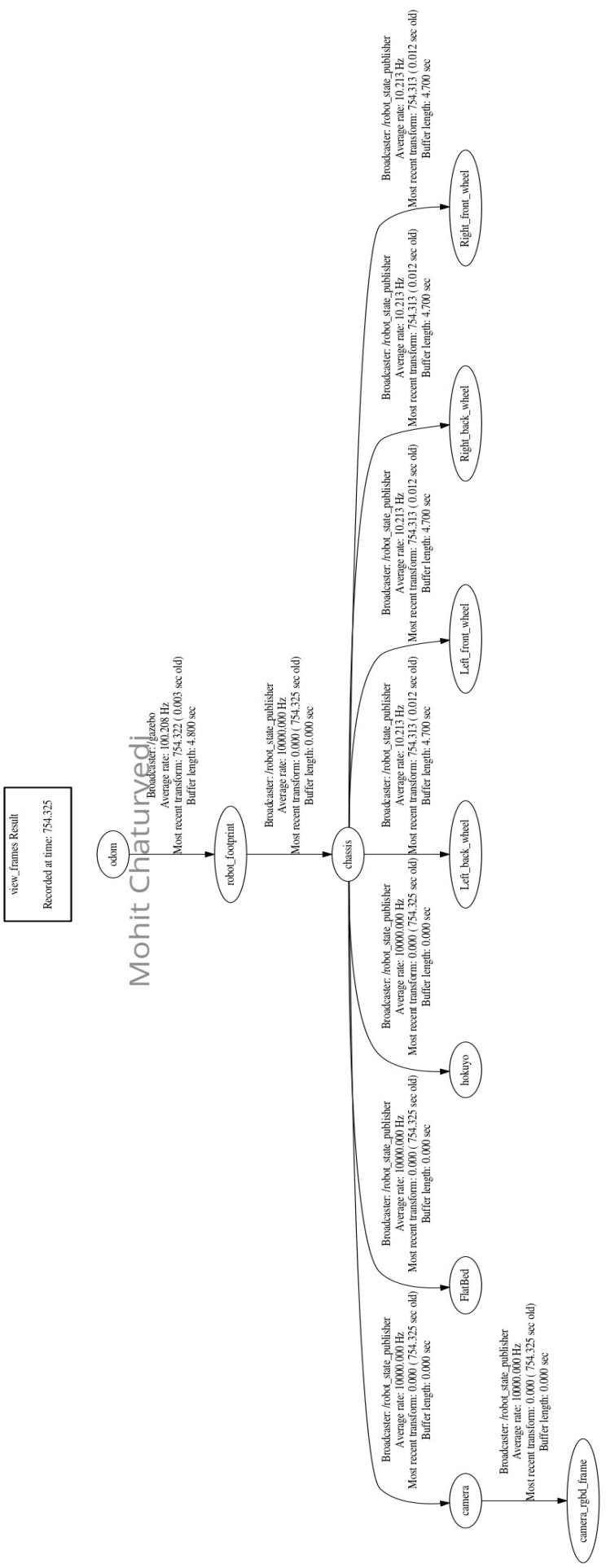



Fig. 8: Transform Tree for `slam_bot` robot model