RoboND SLAM Project

Abhishek Kathpal

Abstract—In this project, the task is to implement Graph SLAM based approach to create 2D occupancy grid and 3D octomap of a given kitchen and dining environment and also a custom gazebo environment. The task is accomplished by a two wheeled robot with a RGB-D camera and hokuyo LIDAR sensor. Real-Time Appearance based mapping approach is implemented using rtab_ros package. The final 3D octomaps for both the environments are generated by performing tele-operation via keyboard and real-time applications of such maps are discussed. A custom ROS package is written to complete the above defined task.

Index Terms—ROS, RTAB-Map, SLAM, Graph SLAM, Gazebo, Udacity, LaTeX, Mapping, RGBD, LIDAR.

1 Introduction

In robotics, Simultaneously localizing and mapping is the most important and common tasks to explore and work in an unknown environment. For this project, Real-time Appearance based Mapping is used which is a Graph-based approach to do SLAM. It is among the best available algorithms for doing Full SLAM with optimizations for speed and memory management. A ROS wrapper rtab_ros is used for creating and saving rtab map database. RTAB map is created for two gazebo environments and parameters are tuned in the launch files.

The steps for this project can be divided as follows-
1. Integrating RGBD camera in the urdf file of the robot and checking the tf tree
2. Teleop the robot to create a map for the kitchen and dining world
3. Create custom gazebo world and create RTAB map

2 Background

In this project, no predefined map is supplied to a robot. So for creating a map, we need localization and mapping simultaneously. SLAM can be divided into categories based on forms and nature as shown in Figure 1.

Based on forms, SLAM can be online SLAM which gives current pose and map as output and Full SLAM which gives trajectory and map. Full SLAM is much more useful and accurate. Grid-based FastSLAM and Graph based SLAM can be used for doing FullSLAM. Gmapping package in ROS uses Grid-based algorithm which uses occupancy grid mapping and extended Kalman filter. Graph-based SLAM algorithms are usually faster and requires less memory in comparison to Grid-based 3d mapping. Different SLAM strategies are discussed in this section.

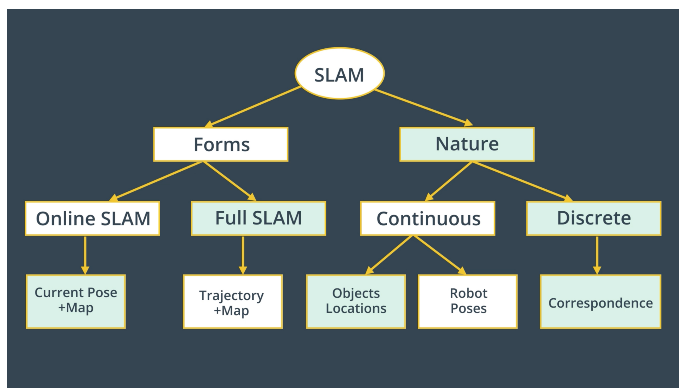

Fig. 1. SLAM categories

2.1 Grid-based FastSLAM

The FastSLAM algorithm solves the Full SLAM problem with known correspondences. It combines extended kalman filter with Monte Carlo Localization technique to perform SLAM. gmapping ROS package is based on this algorithm. The algorithm can be divided into two sections-

1. Trajectory Estimation- A posterior is estimated over the trajectory using a particle filter approach. This will give an advantage to SLAM to solve the problem of mapping with known poses.

2. Map Estimation- Low dimensional Extended Kalman Filter is used to solve independent features of the map which are modeled with local Gaussian.

2.2 Graph-based SLAM Approach

GraphSLAM also solves full SLAM problem. It recovers complete path along with the map, considering dependencies between current and previous poses. It crates graph of all robot poses and features in the environment and then finds most likely path and map of environment. It consists of two sections-

1. Front-End It focus on the construction of graph using the odometry and sensory measurements collected by the robot. It keeps of adding nodes to the graph as robot explores the environment.

2. Back-End The input of this is the completed graph with all the constraints and the output would be the most probable robot poses and features of the map. It is an optimization process to find a configuration of system which lead to produce smallest error.

2.3 RTAB-Map

Real-Time Appearance based Mapping is a graph based approach with speed and memory optimizations. In Apperance based mappings, Loop closures are used which helps in determine if robot has seen the location before or not. RTAB Map is optimized for long term SLAM by using various strategies so that loop closures can be completed in real-time. For metric based raph SLAM, RTAB Map requires a RGBD camera to compute geometric constraints between the images of loop closure. In the next section , the process of adding a RGBD camera is described.

RTAB map also uses visual bag of words to link the features of environment to word. RTAB-Map uses a memory management technique to limit the number of locations considered as candidates during loop closure detection. This technique is a key feature of RTAB-Map and allows for loop closure to be done in real time.

3 SCENE AND ROBOT CONFIGURATION

This section describes the addition of RGBD camera in our robot model and the environments for which RTAB map is created.

3.1 Robot Configuration

Modifying the robot urdf is crucial, so that all the right information is being published on the correct topics for the RTSB map package. The robot urdf is modified in accordance with the figure shown below-

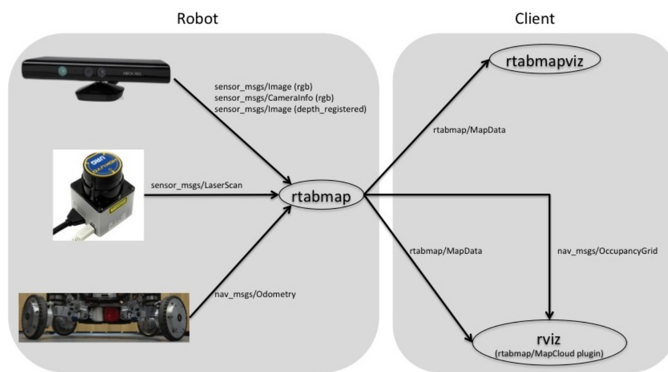

Fig. 2. rtab_map configuration

The snippet shown in figure is added in robot .gazebo file to integrate the RGBD camera-

```
<!-- RGBD Camera -->
<gazebo reference="camera">
  <sensor type="depth" name="camera1">
      <always_on>1</always_on>
      <update_rate>45.0</update_rate>
      <visualize>true</visualize>
      <camera>
          <horizontal_fov>1.047</horizontal_fov>
          <image>
              <width>640</width>
              <height>480</height>
              <format>R8G8B8</format>
          </image>
          <depth_camera></depth_camera>
          <clip>
              <near>0.1</near>
              <far>20</far>
          </clip>
      </camera>
      <plugin name="camera_controller" filename="libgazebo_ros_openni_kinect.so">
          <alwaysOn>true</alwaysOn>
          <updateRate>45.0</updateRate>
          <cameraName>/camera</cameraName>
          <frameName>camera_rgbd_frame</frameName>
          <imageTopicName>/camera/rgb/image_raw</imageTopicName>
          <depthImageTopicName>/camera/depth/image_raw</depthImageTopicName>
          <pointCloudTopicName>/camera/depth/points</pointCloudTopicName>
          <cameraInfoTopicName>/camera/rgb/camera_info</cameraInfoTopicName>
          <depthImageCameraInfoTopicName>depth/camera_info</depthImageCameraInfoTopicName>
          <pointCloudCutoff>0.4</pointCloudCutoff>
          <hackBaseline>0.07</hackBaseline>
          <distortionK1>0.0</distortionK1>
          <distortionK2>0.0</distortionK2>
          <distortionK3>0.0</distortionK3>
          <distortionT1>0.0</distortionT1>
          <distortionT2>0.0</distortionT2>
          <CxPrime>0.0</CxPrime>
          <Cx>0.0</Cx>
          <Cy>0.0</Cy>
          <focalLength>0.0</focalLength>
      </plugin>
  </sensor>
</gazebo>
```

Fig. 3. RGBD integration

The tf tree after integration is shown below-

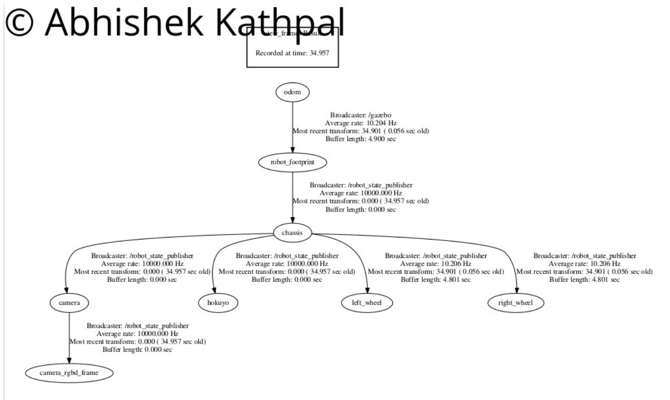

Fig. 4. Robot tf tree

3.2 Environments

The robot is tele-operated in two simulated environments. One is kitchen and dining environment which was provided by Udacity. The other one is created by me. The simulated environment I created has various kinds of artifacts, I played around with them to understand which kind of environment is difficult for RTAB map. This issues I observed are disucssed in discussion section.

Both the environments are shown below-

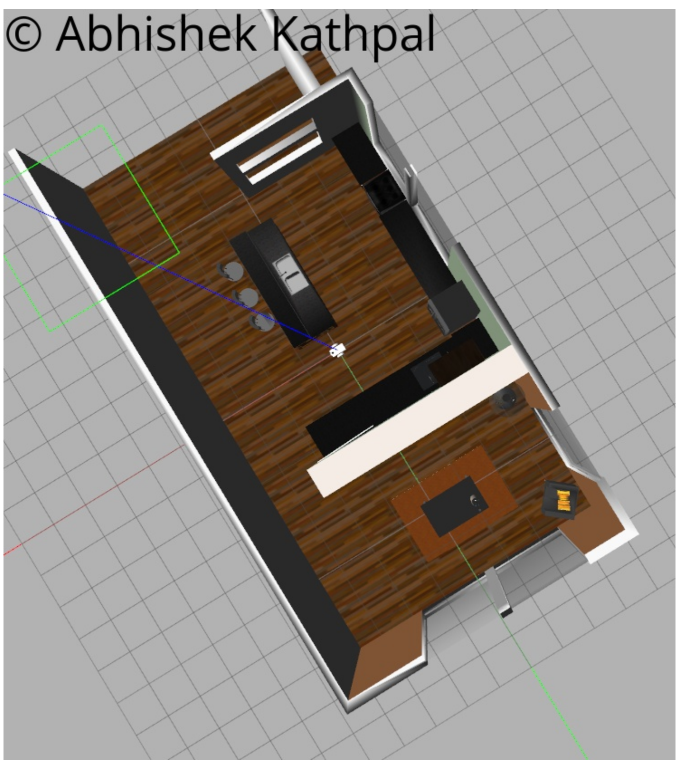

Fig. 5. Kitchen and Dining Environment

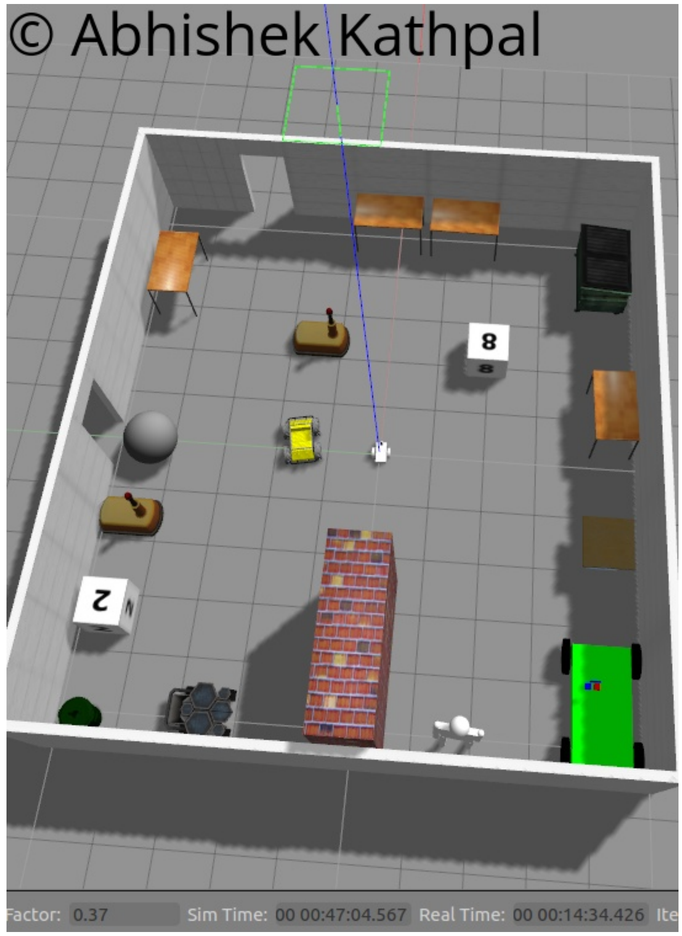

Fig. 6. My simulated Environment

4 Results

In this section, the final RTAB map results are shown for both the environments-

4.1 Kitchen and Dining Environment

This environment was easier to map in comparison to the custom environment I created. It has less visible shadows and features are easily extracted. The initial configuration is hsown below-

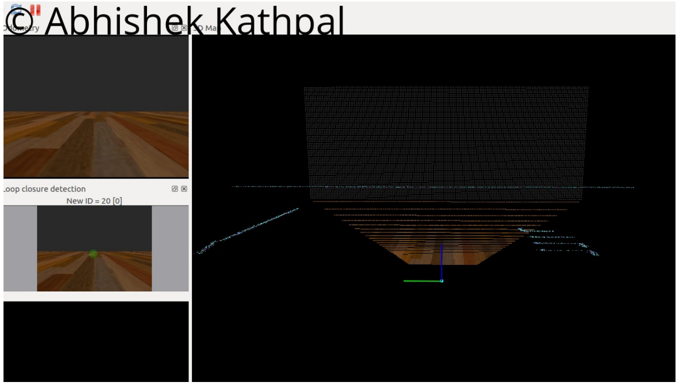

Fig. 7. Initial configuration RT

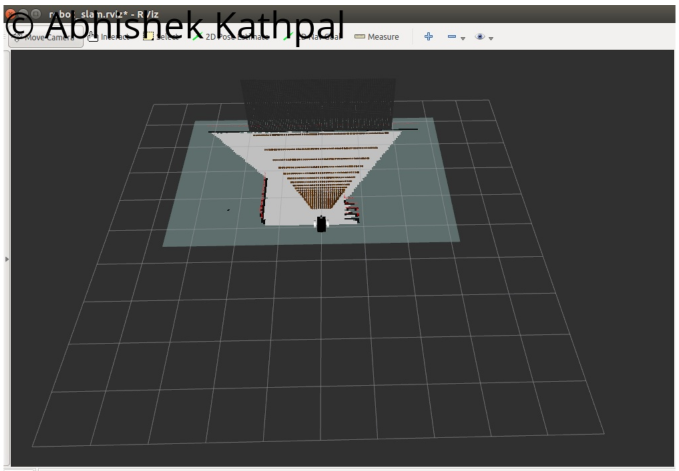

Fig. 8. Initial configuration RVIZ

The final RTAB map for this environment is shown below-

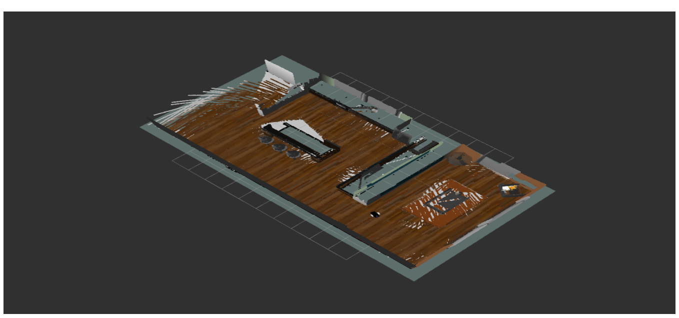

Fig. 9. RTAB Map Kitchen world

4.2 My Gazebo environment

The number is little high as sometimes the robot is rotated at its location couple of times to get the features. It took a lot of time for this custom environment as there are shadows the some wierd textures are there. The initial config is shown below-

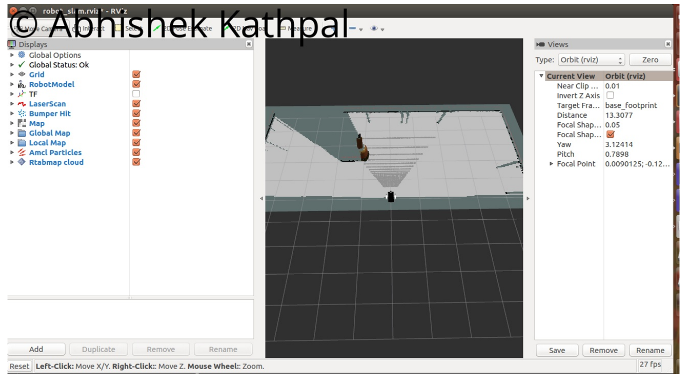

Fig. 10. Initial configuration

Loop closures are shown in figure below (in the bottom the G parameter represents global loop closures). This is done after generating the map and creating rtab map database.

closures.png

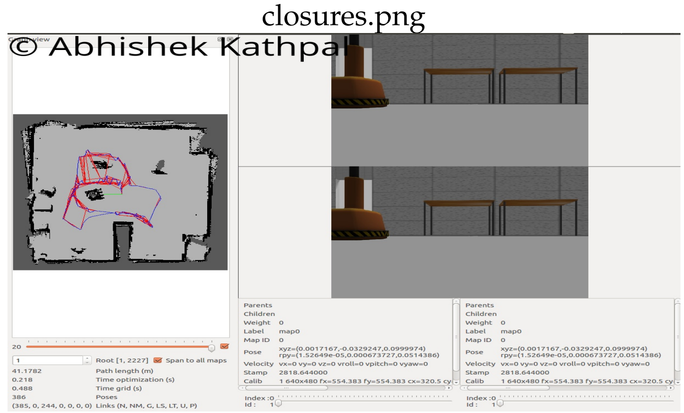

Fig. 11. Loop CLosures

The final RTAB map for this environment is shown below-

Fig. 12. RTAB Map

5 DISCUSSION

The first issue I faced when I tried increasing speed of the robot a lot during the tele-operation, the map started shifting a lot. The mapping should be done with very slow speeds. Another thing, initially I created a very open environments, then RTAB map is difficult to create as it depends on feature extractions to find the loop closures. ANother thing I notices is, the textures of the objects also has effects in the map quality, as you can see in my environment the sphere is not represented correctly because the texture is quite similar to the background and environment. Another issue is caused by lighting, the shadows sometimes causes false loop closures. Because of these issues, I have to traverse the robot many times to the same location to make sure, I have appropriate loop closures. ANother thig, I notices is rotating the robot should be done at very slow speeds in comparison to traslational velocities.

6 FUTURE WORK

The next goal is to implement and test this mapping technique on an autonomous drone using NVIDIA Jetson board and ZED RGBD camera. To improve the output, may be try to use a better feature extractor in the back end. I will also test this technique with faster speed and how it can work better with drones which relatively moves at higher speeds in comparison to mobile robots.. I would also like to test this mapping technique on turtlebot 2 as it already comes with a RGBD camera and it will be easiest to map with as it has very slow speeds.