Autonomous Mobile Robot

Jerrin Bright[1]
[1]School of Mechanical Engineering
Vellore Institute of Technology, Chennai, India

Abstract: The work is a mini project built for learning the basics of SLAM using ROS scripts in Gazebo and RViz for visualization of our script codes. Initial testing was done using the Turtlebot Burger robot, followed by using a custom built URDF model. In this work, we will start with the introduction of the Turtlebot, how to install it, how to simulate in ROS environments. Then we will experiment different types of Lidar based SLAM techniques followed by making a custom URDFs differential robot and experimenting Gmapping with it.

Keywords: Lidar-SLAM; Gmapping; AMCL; ROS; URDF

I. Introduction, Installations

Turtlebot is a small programmable mobile robot used for educational purposes. There are a lot of types of Turtlebot available. Some of them includes burger, waffle pi, etc. Its core tech includes SLAM, Navigation and Manipulation. It comes with an inbuilt Lidar.

Installation of Turtlebot:

git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git

We will use Burger model here. So, to set burger model, do the following,
export TURTLEBOT3_MODEL=burger

Or can be set via,
gedit ~/.bashrc
export TURTLEBOT3_MODEL=burger (Add this at the end of the script)
Then reload the bash file via,
source ~/.bashrc

Two types of SLAM techniques are used here- Gmapping and Karto SLAM.

In **Gmapping**, particle filter is used and data from the robot and the pose of the robot is fused to create the 2D grid map. ROS is leveraged here using the slam_gmapping node to generate the 2d occupancy grid map using the fusion of laser and the estimated pose data. As long as there are enough estimated particles and the real position error corresponds to the covariance of the input odometry, the particle filter converges to a solution which represents well the environment, particularly for GMapping when there are loop closures. Wheel Odometry is used to test if new scans are required and also as an initial guess for the 2D mapping. **Karto SLAM** is generally used to map and navigate in unstructured environments. Hector_mapping node is used in generate the map here. One major advantage of hector SLAM is that odometry is not required whereby we

can prevent errors as a result of slippage of the wheels. While mapping, they create sub-maps that are linked by constraints in the graph. When a loop closure is detected, the position of the sub-maps is re-optimized to correct errors introduced by noise of the sensor and scan matching accuracy. Unlike Hector SLAM, external odometry can be provided to get more robust scan matching in environments with low geometry complexity.

II. SLAM Implementation

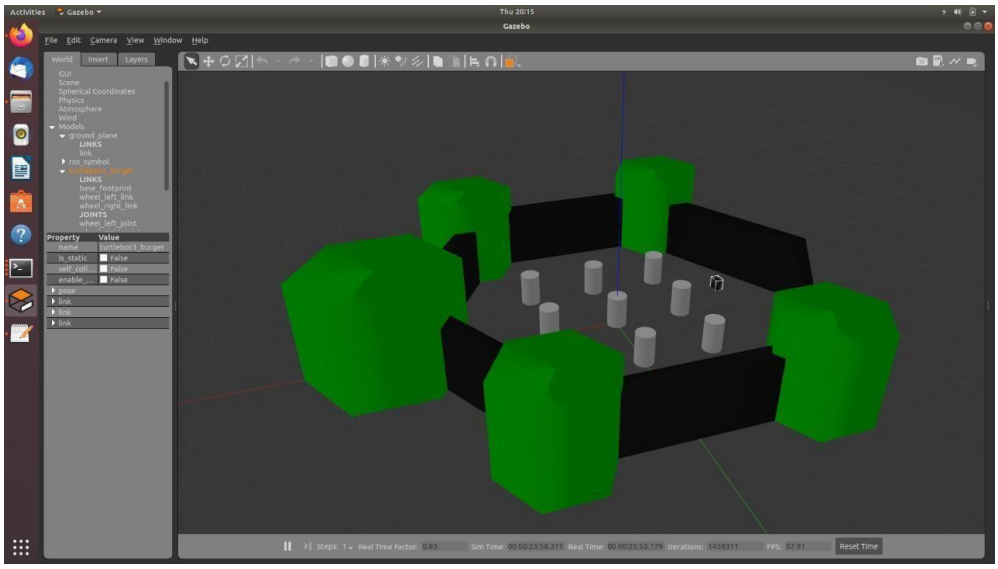

Figure 1. Turtlebot in Gazebo Simulator

Start Gazebo,

roslaunch turtlebot3_gazebo turtlebot3_world.launch

Start SLAM,

roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping

For Autonomous Mapping,

roslaunch turtlebot3_gazebo turtlebot3_simulation.launch

For Manual Teleop Mapping,

roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

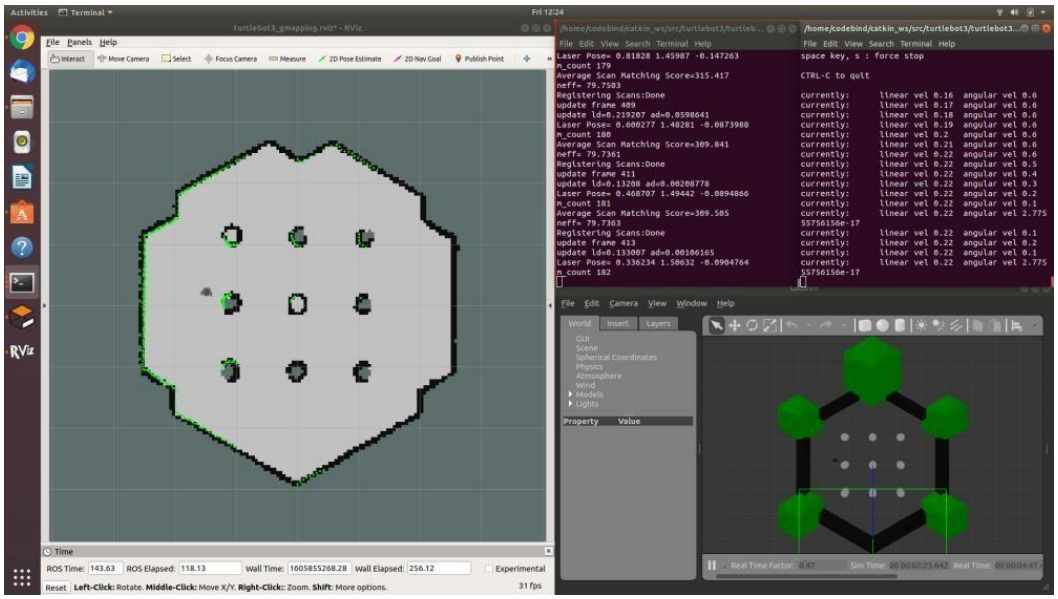

Figure 2. Mapping and Visualization of the Turtlebot world

To save the map,

rosrun map_server map_saver –f ~/map

The YAML and PGM file of the map generated after the above-mentioned command.

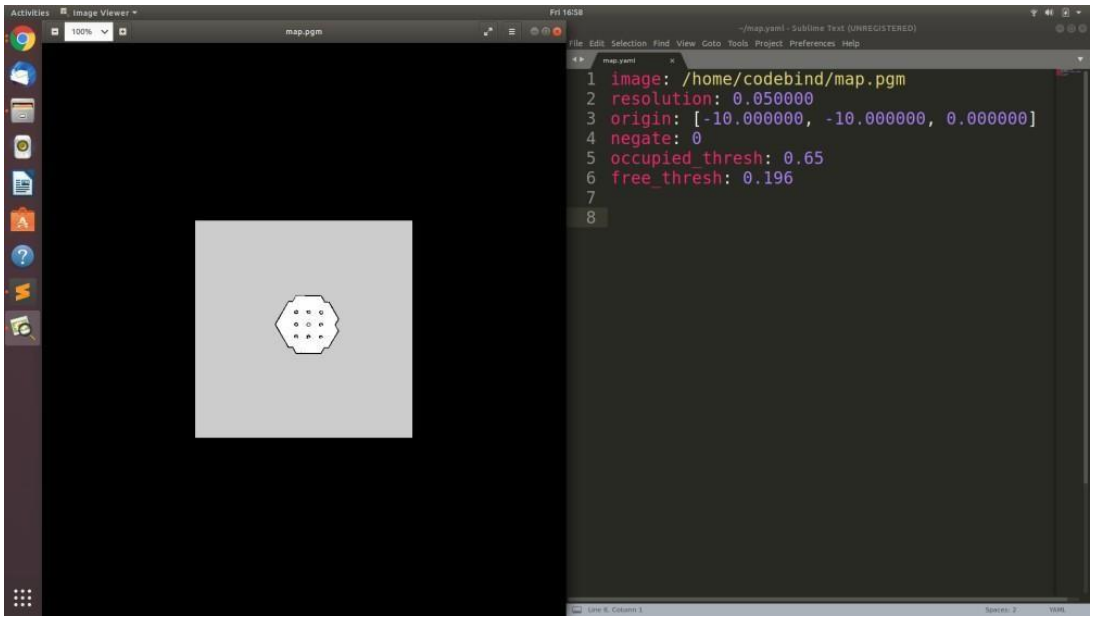

Figure 3. Map generated after mapping

After saving, to launch for navigation,

roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml

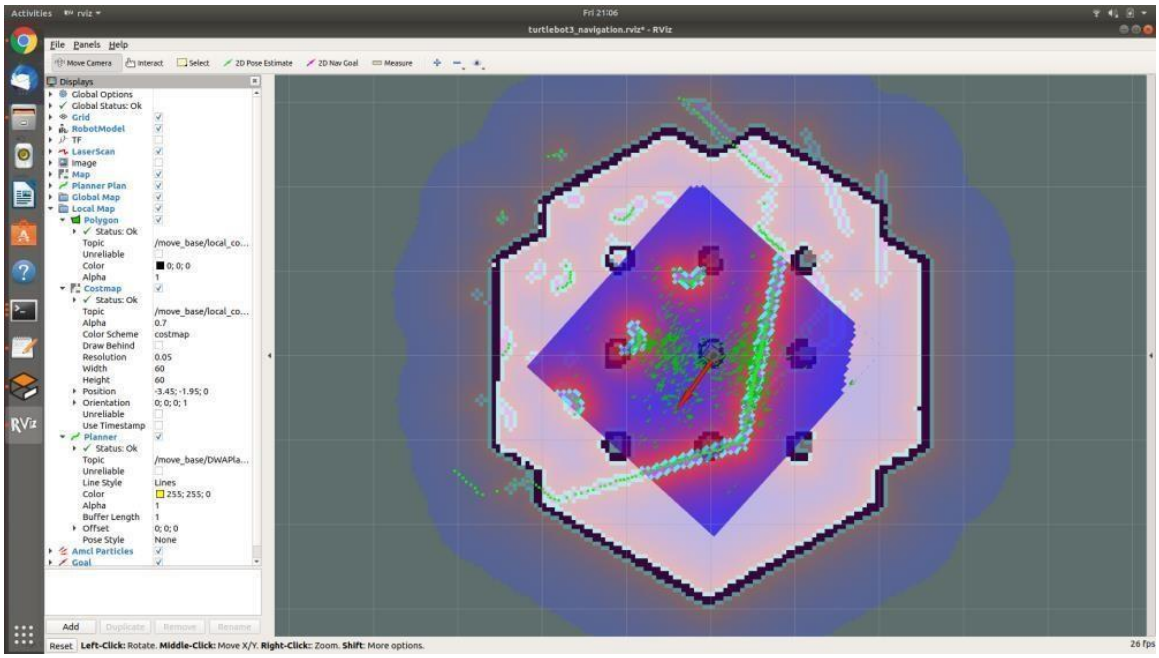

Figure 4. Navigation using AMCL

Fig. 4 is the navigation RViz window, where navigation will take place avoiding the obstacles autonomously when a particular 2D point is marked.

Hector SLAM

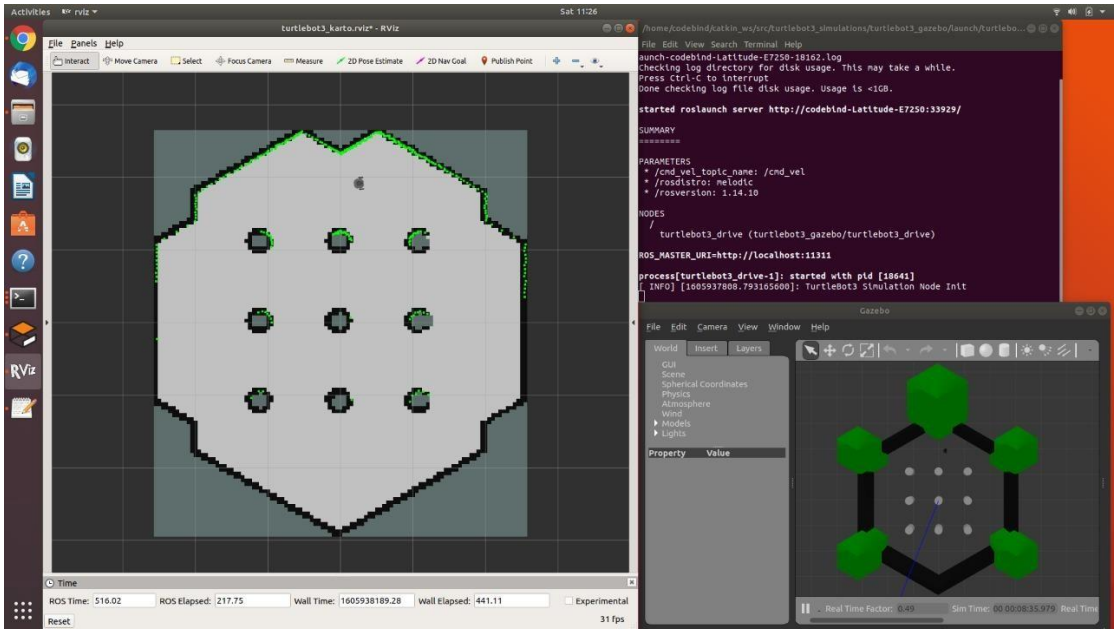

Figure 5. Karto SLAM Mapping

Got very similar results, but surpasses Gmapping technique comparatively in terms of the map quality.

II. Custom SLAM Implementation

Used m2wr robot and did custom mapping with it and saved using the map_server. The steps involved are very similar to the turtlebot3

- Robot with differential drive, wheels and caster is designed.
- URDF files are made from Solidworks using plugins.
- Then, added Lidar plugins and URDF in the custom bot.
- Wrote codes to launch my robot in gazebo and RViz.
- Obstacle avoiding code was written for autonomous mapping.
- Tweaking the Gmapping code based on our topic list.
- Then, launching the Gmapping code along with gazebo env.
- Launching the obstacle avoiding code for navigation.
- Once the whole world is mapped, save it.

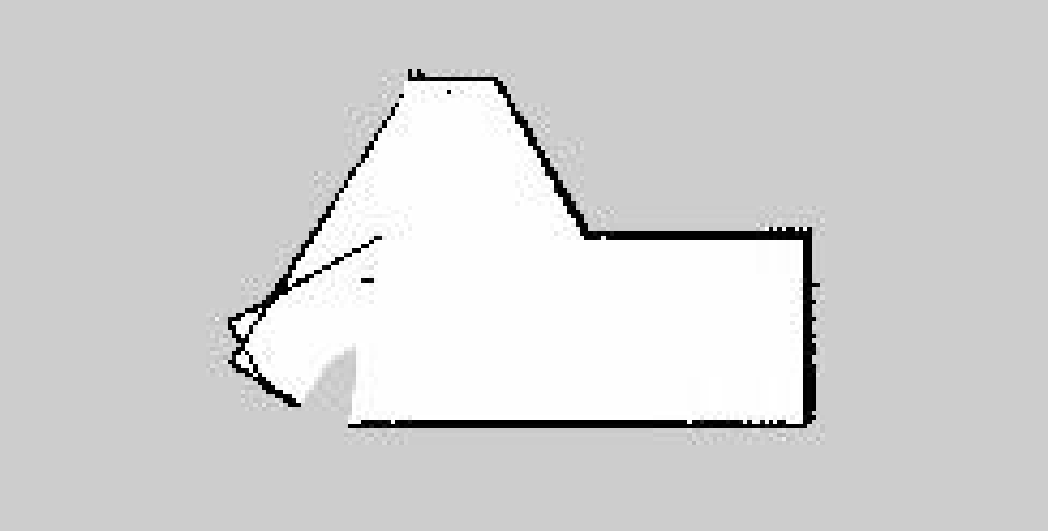

Figure 6. Map obtained from Custom URDF

III. Source Code

GitHub Repository: https://github.com/jerriebright/M2WR-SLAM-ROS

The Repo consists of the m2wr URDF, launch files for Gmapping and spawning the m2wr in our environment, custom worlds and the map folder where all the maps generated will be stored.

IV. References

[1]. https://www.youtube.com/watch?v=tUm6MRGYam8

[2].https://github.com/StevenShiChina/books/blob/master/Mastering%20ROS%20for%20Robotics%20Programming.pdf

[3]. https://www.theconstructsim.com/ros-projects-exploring-ros-using-2-wheeled-robot-part-1/

[4]. http://gazebosim.org/tutorials?cat=build_world&tut=building_editor

[5]. http://wiki.ros.org/urdf/XML