TKNLS Report

Osama Awad

July 2023

1 fore-note

This isn't meant to be a rigorous technical guide by any mean. This document just gives a general overview and describes some important points when designing the robot. For a more detailed explanation behind the math and APIs that make this work you can checkout my final internship report(as of the moment it is a work in progress)

2 Introduction

In this report I will outline the design process I followed to create a 2d occupancy map (occ_map) using a 3D lidar model. The idea is to be able to generalize this process to any sensor.

3 The Sensor

First one needs to find a model of the sensor that he/she wants to use or design the sensor from scratch. Designing a sensor model from scratch can be a bit intimidating as this requires writing a gazebo plug-in which is not very easy to do. Luckily for most commonly available sensors, a model probably already exists online somewhere. For the 3D Lidar I found a model created by the toyota research institute. There is also another velodyne lidar model that is provided by gazebo.

3.1 Plugins

For a sensor to function properly, one should first download the gazebo plugin for the specific sensor. After that the plugin is compiled (using cmake and make usually) and then the path to the compiled plugin must be exported in bashrc. Instead of exporting, I like to put the plugin I compiled with the default gazebo plugins. The required termainal command for doing so can be found in the github readme.

3.2 Packages

Some sensors might have ros packages associated with them. The velodyne lidar im using has a maintained package that performs useful functions, however for achieving the required task I didn't find a need for using it.

3.3 SDF and URDF

I am using the ready made turtlebot3 as a custom base for the sensor. This comes with some issues such as needing to modify the sdf and urdf files separately due to the way the launch scripts are written. If you are designing a robot from scratch however, this shouldn't be an issue since gazebo auto handles the conversion between the two as needed.

3.3.1 SDF

In the SDF part we need to add the relevant tags provided in the package for our desired sensor. If we installed the library correctly everything should work properly. If using a robot base like me, you need to check for conflicting joint names, joints that arent going to be used anymore and need to be deleted etc...

3.3.2 URDF and Joint State Publishing

whether you are designing your own custom base or building your own base from scratch, you need to publish the joint states using the relevant plugin otherwise you wont be able to move your robot or do any proper visualizations using rviz. If you are building your robot from scratch you need to add the plugin to your urdf file. If you are using a ready built robot base chances are the plugin is already added and configured, however changes might still need to be made since we added some joints in the SDF of the robot.

4 PointClouds and Laserscans

ROS2 (and ROS) uses a message type called PointCloud2 to represent the data coming from most 3D sensors. This format can be useful if we want to write our own custom algorithms for doing stuff in 3D. At the moment however, NAV2 stack only supports 2D occ_maps, 2D costmaps, and only 2D planning algorithms are implemented. Thus to get any useful functionality out of our sensor, we need to convert the pcd to a 2d laser scan. This can be achieved using the pointcloud_to_laserscan package. Please check the github page of the package and my own github page to see the parameter definitions and appropriate topic remappings. Please also check the custom launch file I created to see how to make this conversion automatically happen whenever the robot is launched.

5 slam_toolbox and NAV2

After setting up the robot and the sensors, we can finally proceed to the mapping step. This step requires installing the slam_toolbox and reconfiguring the parameter files. An example can be found in my github page. For simple applications, one can just reconfigure the topic name and leave the rest of the parameters to be the same.

For this application I chose async mapping because I dont know if my computer will be able to handle sync mapping. Moreover I kept the default amcl and solver configurations since I don't have a good grasp on what changing those parameters in terms of real-life applications. These topics are really complicated and require deeper investigation. The interested might want to look at the book Probabilistic Robotics for an introduction and then move on to more rigorous resources for a deeper understanding of mapping and localization topics. To make the setup easier, I have also created a custom launch file that handles rviz viewing and all mapping related stuff.

To save the maps, the normal mapsavercli package that comes with nav2 can be used.