Localizing a Mobile Robot Using ROS

Bahadır Özkan

May 8, 2018

Abstract

The purpose of the paper is to describe how a mobile robot is localized in the RViz and Gazebo simulation environments with the help of ROS packages. Adaptive Monte Carlo Localization (AMCL) and navigation stack packages are essential for the task therefore they will be elaborated further. Parameter tuning is crucial to localize the robot and bring it to the target position hence this paper explains the parameter choices and their effects.

1 Introduction

Safe path planning is a critical task when working with mobile robots however before that the robot needs to be sure of its position, in other words it needs to be localized. This paper discusses the performance of two mobile robots with different configurations that are sent to the same goal position in Rviz and Gazebo simulation environments. First robot that is referred as udacity_bot and provided by Udacity as a part of Robotics Software Nanodegree Program. The second robot mod_bot is the modified version of the Udacity robot. Once the actuators and size of the robot has been changed, new set of parameters for the navigation stack needed to be set for the mod_bot.

This article begins with discussing the background of the Robot Operating System (ROS) packages that are used. Thereafter the results and model configurations of the robots are discussed. This includes the parameter selection for the different packages as well as the size, actuator and sensor selections for the robots.

2 Background

Before diving into the technical details and challenges, one needs to understand the ROS packages that are essential to the scope of this project.

2.1 Navigation Stack

Navigation stack package in ROS takes information from odometry, sensor data and destination and provides a safe path by sending velocity commands to the base of the robot. More information on navigation stack can be find on http://wiki.ros.org/navigation.

2.2 AMCL

Monte Carlo Localization (MCL) is also known as particle filter localization algorithm is a probabilistic method that uses filters to localize the robot. The algorithm receives map as input and outputs position and orientation of the robot continuously. Each particle has a position and orientation thus by re-sampling particle information while robot moves and senses, provides certainty where the robot is located. AMCL is the adaptive version of the Monte Carlo Localization where number of particles dynamically changes over a period of time which provides a computational advantage over MCL. ROS has an amcl package that is used to achieve results described in this article.

2.3 Alternative Localization Methods and Challenges

There are other approaches to the localization problem other than the AMCL. Kalman Filter is one of them. Kalman Filter, after taking noisy measurements and initial assumptions, outputs desired states. Advantage of Kalman Filter is that it is; fast and can work with noisy sensor information. Extended Kalman Filter (EKF) is superior to the normal version by its ability to be applied on non-linear systems which is the case in most robotics projects. EKF is a strong algorithm that can be applied to variety of cases however it has its limits. MCL is not restricted with linear Gaussian states-based assumptions like EKF. MCL, with this feature can represent more models than EKF. Further, MCL can control the memory and resolution by changing the number of particles. This allows memory and time efficiency over EKF.
The utmost challenge on mobile robot applications is safely achieving a goal position. This involves the question where is the robot at a given time. This may seem like an easy question however it is not always unlabored to attain the location of a robot. Localization methods that are used for a specific robot may not be applicable to other robots. Two robots that are described here needed different navigation stack parameters, due to their designs. Mapping a known environment will not be sufficient for a robot to move from point A to B safely in real life scenarios since an obstacle easily can be placed along the path of the robot. If robot cannot sense its environment at all times, it will not work safely. Luckily modern-day localization techniques that are described here solves these issues.

3 Results

Figure 1 shows the robot that is provided by Udacity and Figure 2 shows the modified robot which is smaller and has four wheels at the goal position. Launch files and other necessary supporting files can be accessed from https://github.com/bahadirozkan/whereAmIRobot.git.
The udacity_bot moves smoothly after setting the caster wheel dimensions to 0.0499 on its xacro file. Its localization is a cut above the mod_bot. Localization of the mod_bot is effectual until it reaches the goal position. On the goal position mod_bot rotates to position itself which causes a slight deterioration on its localization. (See Figure 2)

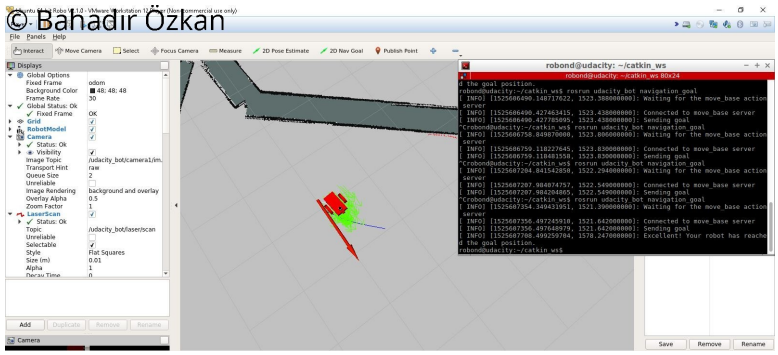

Figure 1: udacity_bot at the goal position

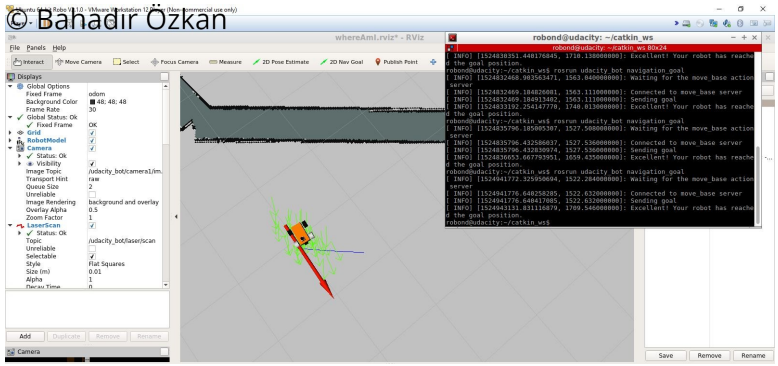

Figure 2: mod_bot at the goal position

4 Model Configuration

The modified robot, mod_bot was chosen mildly smaller than the udacity_bot . The modified version do not have casters, instead it has four smaller wheels. It has a front-wheel drive, a camera and a laser sensor located in front of the robot. Initially, camera and sensor was placed in right justified position. Despite its reasonable performance, it has been found out that, centered camera and sensor have more accurate measurements due to the symmetry of the robot.

4.1 amcl Parameters

Both mod_bot and udacity_bot uses same parameters however different launch files are necessary to link the robot to the navigation stack configuration files. Minimum and maximum particle sizes were set to 100 and 120 respectively to not heavily utilize the CPU. Odom alpha values from one to four were set to 0.05 by trial and error. amcl is a very strong package that with minimum effort can localize a robot inside the ROS environment.

4.2 Navigation Stack Parameters

4.2.1 base_local_planner_params_mod

controller_frequency is set to 10.0 due to *"control loop missed its desired rate"* errors. sim_time is kept as default value 1.0. Decreasing this below 1.0 is inadequate for robot to calculate its forward-simulate trajectory. pdist_scale kept as it is because robot follows a close distance to its path with the default value. gdist_scale value is briefly reduced to 0.7 where robot reaches the local goal and controls its speed sensibly. When setting pdist_scale and gdist_scale to 0.3 robot do not closely follow the given path and takes a longer route to get through the goal position.

4.2.2 costmap_common_params_mod

Changing the transform_tolerance to a non zero number provides a safeguard against losing the link in the tf tree. In this case 1.0 was selected. Obstacle and raytrace ranges were set to 1.0 and 2.0 respectively. These two parameters allows the robot to update its map regarding obstacles within the specified value and clear out space for itself. Robot's radius was set to 0.3 according to its size. Inflation radius was set to 1.5 and cost_scaling_factor to 2 to allow robot to gently navigate around the corners. These parameters are affecting the costmap hence important for smooth navigation. Failing to set the accurate parameters may cause the robot to navigate very closely to obstacles or even to get stuck.

4.2.3 global_costmap_params_mod

update_frequency was set to 1.0 and publish_frequency to 0.5. rolling_window is kept as false thus static_map as true as recommended in the ROS.org documentation. Opposite of this arrangement causes robot to plan a direct path to the goal, ignoring obstacles.

4.2.4 local_costmap_params_mod

On the contrary, rolling window was set to true hence static map to false on the local costmap params. This is recommended in medium or large environments. According to Koubaa, if the exact opposite is chosen; huge number of obstacles may overburden the computational power. Window size was chosen as 5.5 x 5.5 for size to not exceed the local sensor range. [1]

5 Discussion

As a result, both the classroom robot udacity_bot and mod_bot reaches the goal position on all trials. While mod_bot cruises more smoothly, it takes a moment to trace the path that its given in the beginning. Also, as it was mentioned earlier when it arrives to the goal position; it rotates to position itself which slightly deteriorates the particle filter density.The amcl package quickly gains confidence about where the robot is once the robot starts to navigate through its path. The challenge was to tune the navigation stack parameters which would have been done better with devoting more time.

AMCL is an appropriate solution for the kidnapped robot problem where the robot is positioned in an arbitrary location on the map. AMCL does not use landmarks like EKF, and according to the sensor measurements, is able to localize the robot. AMCL takes advantage of dynamically adjusting the number of particles in this kind of cases.

Since probabilistic methods like the AMCL provides a real-time solution to the localization problem, it can be used in variety of cases in the industry. Most common application would be an indoor autonomous mobile robot with or without a known map.

6 Future Work

Parameter tuning of the navigation stack furthermore has affects on the processing time other than the accuracy. This is even more important while working on an actual hardware. The use of rolling window as a local costmap and keeping its size small drastically affects the processing time. Moreover, choosing a very small resolution on the local costmap means that there will be more grids to handle thus will extend the process time.

As might be expected, the size of the robot and the sensors that it uses is also critical on the accuracy and processing time. The mod_bot would navigate more accurately with a all-wheel drive and additional sensors. A four wheel differential drive plugin could be used. A LIDAR sensor would provide a more precise map of the environment however LIDAR is a more expensive choice than the laser rangefinder. A RGBD camera could be used with the laser rangefinder to enhance the map accuracy. This would be a more cost effective solution than the LIDAR solution.

The RViz and Gazebo environments are excellent for testing and iterating nevertheless working on an actual hardware includes additional problems. The power source is one of them, which will affect the design of the robot if a mobile robot is necessary. To minimize the size of the power source, efficient use of CPU and GPU is required. Sensor and actuator selection is another topic that is critical in this sense due to their power consumption. Actuator should be selected according to the total weight, that is to say the power source must be defined in the first place. Lastly, the fact that there would be more obstacles in real life than in simulation environment should be taken into account.

References

[1] Koubaa, A. Robot Operating System (ROS) The Complete Reference (Volume 2). Springer, 2017.