Project: Where Am I?

Oleksii Zhurbytskyi

Abstract—The purpose of this report is to describe the principles of working with simulation in robotics. In particular, develop a mobile robot model for Gazebo, and integrate the AMCL (Adaptive Monte Carlo Localization) and Navigation ROS packages for localizing the robot in the provided map. To improve localization results the parameters tuning was used. After achieving the desired results for the robot model introduced in the lesson, the custom robot model was implemented.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization.

1 Introduction

Localization is the challenge of determining a robot's pose in a mapped environment. This is done by implementing the probabilistic algorithm to filter noisy sensor measurements and track the robot's position and orientation. There are four very popular localization algorithms: EKF (Extended Kalman Filter), Markov Localization, Grid Localization, and MCL (Monte Carlo Localization). The project utilizes AMCL (Adaptive Monte Carlo Localization) algorithm. AMCL is an MCL that dynamically adjusts the number of particles.

2 Background

There are three different types of localization problems. The amount of information present and the nature of the environment that a robot is operating in determine the difficulty of the localization task. First, and the easiest problem, is called Position Tracking (Local Localization). In this problem, the robot knows its initial pose, and the localization challenge entails estimating the robot's pose as it moves out on the environment. This problem is not as trivial since there is always some uncertainty in robot motion.

Second, a more complicated localization problem is called Global Localization. In this case, robot's initial pose is unknown and the robot must determine its pose relative to the map. The Global Localization problem is more difficult since the error in the robots estimate cannot be assumed to be small.

Third, the most challenging localization problem is Kidnapped Robot problem. This problem is just like Global Localization except that the robot may be kidnapped at any time and moved to a new location on the map.

2.1 Kalman Filters

The Kalman Filter is an estimation algorithm that is very prominent in controls. It used to estimate the value of a variable in real time as the data is being collected. This variable can represent the position or velocity of a robot. The reason that the Kalman Filter is so noteworthy is that it can take data with a lot of uncertainty or noise in the measurements, and provide a very accurate estimate of the real value. Filtering is a two-step process and the Kalman Filter continuous iteration of these two steps. The first step is a measurement update. We use the recorded measurement to update our state. The second step is a state prediction. We use the information that we have about the current state to predict what the future state sill be. At the start, we use initial guess. We continue to iterate through these two steps and it does not take many iterations for our estimate to converge on the real value.

However, the assumptions under which the Kalman Filter operates are that

- Motion and measurement model are linear
- State space can be represented by unimodal Gaussian distribution

These assumptions are very limited and would only suffice for very primitive robots; most mobile robots will execute nonlinear motions. Extended Kalman Filters (EKFs) helps to resolve some of these. EKF is using a local linear approximation of small sections of the transfer functions. In these small slices, the linear approximation is sufficient in estimating the outcomes.

2.2 Particle Filters

Monte Carlo Localization algorithm, also known as Particle Filter, is the most popular localization algorithm in robotics. A robot can navigate in sites known and collect sensory information using rangefinder sensors. MCL will use these sensors measurements to keep track of a robot pose. Monte Carlo Localization algorithm uses particles to localize the robot. Each particle is a virtual element that resembles a robot and has a position and orientation and represents a guess of where a robot might be located. These particles are resampled each time robot moves and sense its environment. MCL is limited to Local and Global Localization problem only.

2.3 Comparison / Contrast

MCL presents many advantages over EKF. Firstly, MCL is easy to program as compared to EKF. Second, MCL represents non-Gaussian distributions and can approximate any other practical important distribution. This means that MCL is unrestricted by a linear Gaussian states-based assumption as is the case of EKF. This allows modeling a much greater

variety of environment. Third, in MCL, we can control computational memory and resolution of the solution by changing the number of particles distributed throughout the map.

TABLE 1
MCL vs EKF

	MCL	EKF
Measurements	Raw measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency(memory)	+	++
Efficiency(time)	+	++
Ease of Implementation	++	+
Resolution	+	++
Robustness	++	x
Memory and Resolution Control	Yes	No
Global Localization	Yes	No
State Space	Multimodal Discrete	Unimodal Continuous

Adaptive Monte Carlo Localization was chosen for this project because it solves Global Localization problem, provides memory and resolution control can take any State Space models.

3 Results

3.1 Model Configuration

Figure 1 shows the custom bot. The shape and the size modified. Was added tower which carries a camera and the laser range-finder. The driving wheels are moved closer to the front of the robot, and the front caster is removed. The model became heavier and less maneuverable.

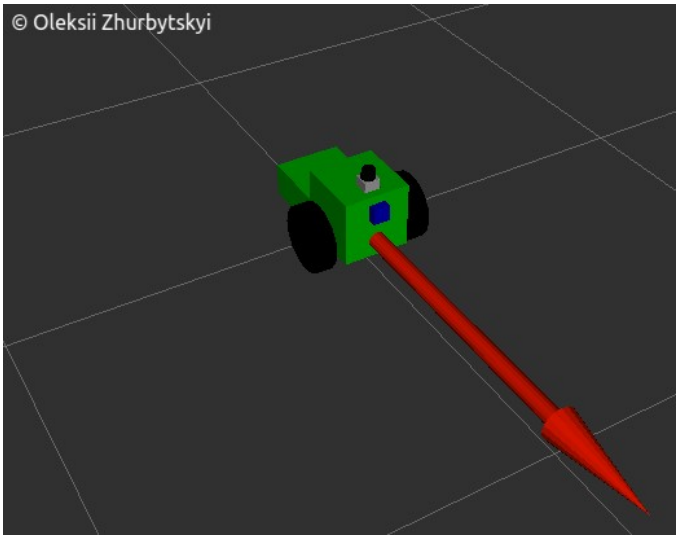

Fig. 1. Custom bot.

3.2 ROS packages

For the simulation in RViz, the following ROS packages were used:

- amcl
- move_base

3.3 Parameters

To successfully solve the localization problem and reach the goal position, the parameters for each of the used ROS packages were tuned.

AMCL parameters where tuned to reduce computational load. The $min\_particles$ is set to 10 and the $max\_particles$ is set to 100. After playing with transform_tolerance, it was increased to 0.25. This parameter is a time with which to post-date the transform that is published, to indicate that this transform is valid into the future. Also, the documentation for the package contains information that for diff-corrected odom_model_type all odom_alpha should be greatly reduced. These parameters have been reduced by a factor of 100 compared to the default values.

Move_base parameters where tuned according to the map, the configuration of obstacles and the size of the robot. The maximum range in meters at which to insert obstacles into the costmap using sensor data $obstacle\_range$ is set to 1.0. The maximum range in meters at which to raytrace out obstacles from the map using sensor data $raytrace\_range$ is set to default value 3.0. Also, the footprint of the robot specified in the $footprint$ parameter. Both the Local and Global costmap update and publics frequencies where set to 5 and 2 Hz. The both update and publish rates where chosen to fix the warning messages.

After tuning these parameters, the robot successfully reached the goal, it is shown in Figure 2 for Udacity Bot, and in Figure 3 for Custom Bot.

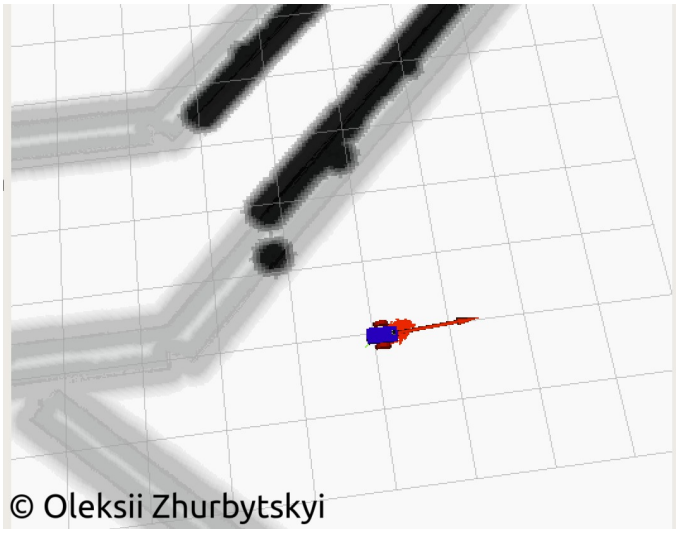

Fig. 2. Udacity bot result.

Since the dimensions of the Custom Bot and its characteristics are slightly different from the Udacity Bot, some parameters have been changed. After long attempts to find the right parameters, it was possible to achieve the maximum result with the following values. The minimum and maximum number of particles is increased to 50 and 300, respectively. The maximum forward velocity allowed for the base in meters/sec max_vel_x is set to 0.5. The delay in transform data is set to 0.5. The weighting for how much the controller should stay close to the path it was given $pdist\_scale$ is set to 1.6. The weighting for how much the

controller should attempt to reach its local goal $gdist\_scale$ is set to 1.8.

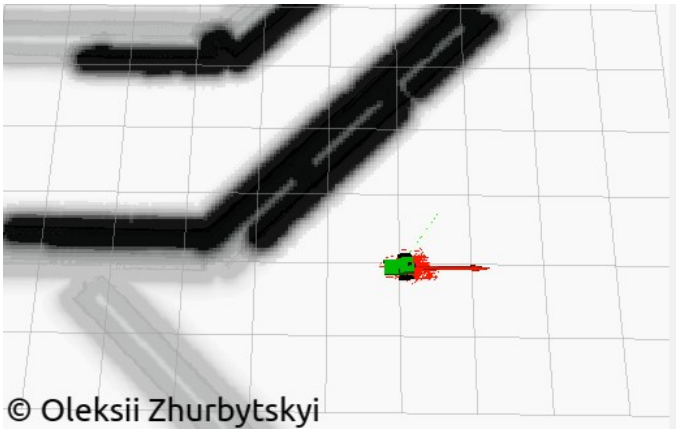

Fig. 3. Custom bot result.

4 Discussion

In the process of working on the project, it became clear that the size, weight, and maneuverability of the robot greatly influence the choice of parameters.

The AMCL itself can not solve kidnapped robot problem. But there is a method to detect the kidnapped robot problem event in Monte Carlo Localization [2]. The method uses the sensor reading of the robot to determine if robots displacement at particular time instance is considered a natural displacement or not. After the determination of such event, global localization task should begin from the beginning.

The AMCL can be used in various areas of robotics. The most relevant area is self-driving cars and warehouse robots.

5 Future work

The most difficult part of the project and the most tedious and frustrating is the part of the tuning the parameters. They are very large set, there are many variants of the combinations, and there are no clear instructions on the tuning and debugging the problems. This part took a very long time.

Despite the difficulties in the selection of parameters, robots successfully reached the goal. Perhaps the configuration of the Custom Bot has not been successful since it acts less efficiently than Udacity Bot. Future work should be aimed at improving the configuration of the Custom Bot, a more detailed study of the remaining parameters and their impact on the behavior of the robot. Adding other sensors, such as a LIDAR, will also positively affect the solution of the problem.

[1] Setup and Configuration of the Navigation Stack on a Robot (http://wiki.ros.org/navigation/Tutorials/RobotSetup)

[2] Detection of kidnapped robot problem in Monte Carlo localization based on the natural displacement of the robot (http://journals.sagepub.com/doi/full/10.1177/1729881417717469)