Localization Project: Where Am I?

Dilmuratkyzy Zerde

Abstract—This report studies two models of robots created using URDF in a Gazebo/RViz environment, that are assigned to navigate within a given map to reach a predefined goal position. Robots have two sensors: a camera and a laser scanner, to process data from sensors and move the robot models ROS navigation stack is used.Both robots use AMCL (Adaptive Monte Carlo Localiation) ROS package to perform localization . The parameters of packages are tuned to optimize the performance of both robot models.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization, Kalman Filter, Particle Filter, ROS.

1 Introduction

In robotics the problem of localization is to determine robot's position and pose within a specific environment using sensor data and control inputs. In real world sensor data is uncertain and often not exact with noises from the environment robot is in. And moving commands sent to robot may not be executed perfectly due to texture of the ground and other external factors. Thus some localization techniques are needed to establish the current position of robot and navigate through the map. This paper showcases a successfully implementation of robot localization using the AMCL (Adaptive Monte Carlo Localization) algorithm. Two robots were created to navigate through a given map in a simulation environment. Definition of one of the robots was

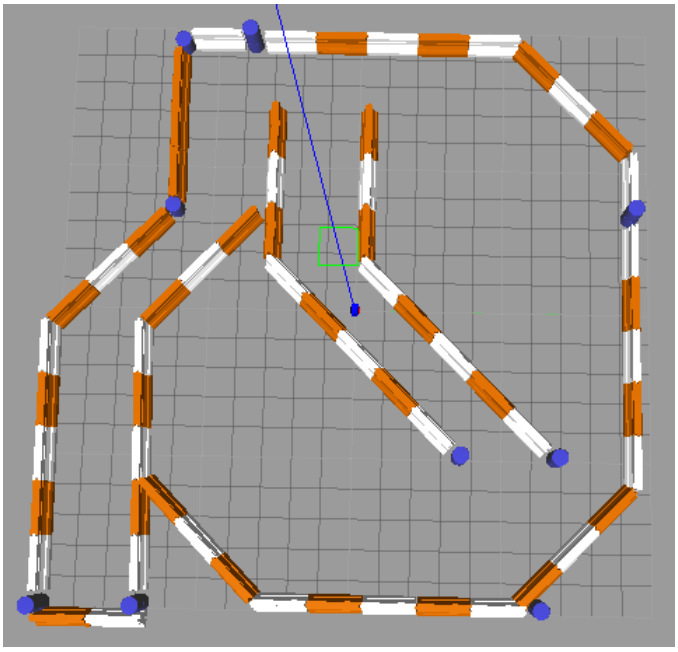

Fig. 1. Given localization map and robot at its initial position

provided as part of the project, and the second was created independently.Both robots successfully localized themselves and reached the end goal position.

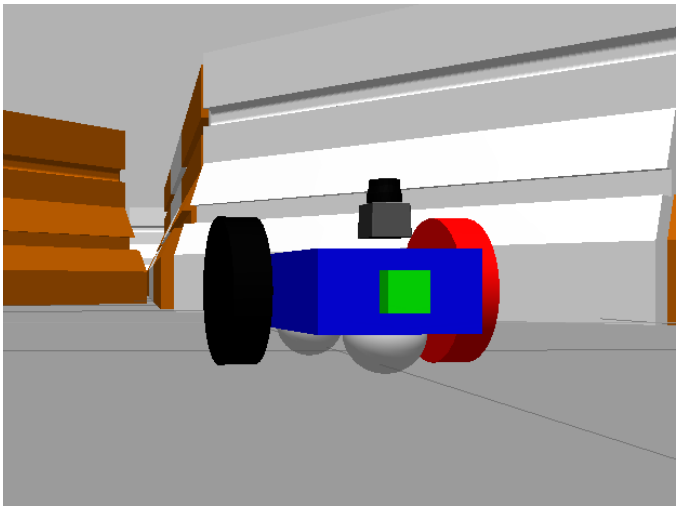

Fig. 2. udacity_bot

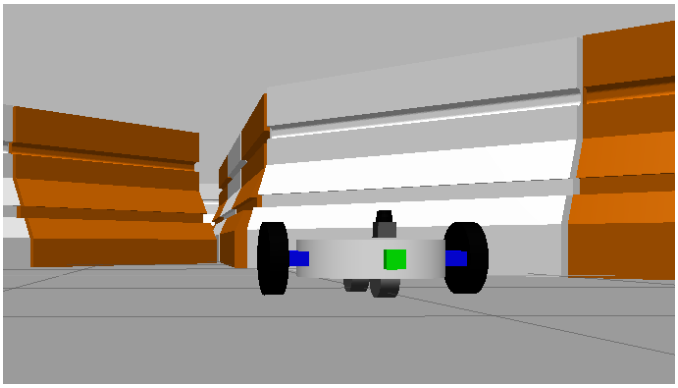

Fig. 3. ziz_bot

2 Background

Localization is one of the fundamental task for an autonomous mobile robot. To localize itself robot uses various sensor data and movement inputs. However,in the real world these data are often noisy and uncertain, so localization algorithms plays important role in calculating approximate position of the robot. The two most commonly used approaches to localization are Extended Kalman Filters

and Monte Carlo Localization.

2.1 Kalman Filters

The Kalman Filter is an estimation algorithm that uses measurements with noises and other inaccuracies, and produces estimates of unknown variables like position of a robot. It can take data with a lot of uncertainty in the measurements, and provide very accurate estimate of the real value. Kalman Filter works by iterating through these two steps:

- Measurement Update
- State prediction

It can run in a real time, as the data is being collected. Kalman Filter is often unsuitable for real life scenarios, because it can only work in linear systems. The Extended Kalman Filter fixes this issue by linearizing non-linear inputs using Taylor Expansions. But it is computationally expensive and needs considerable amount of CPU resources.

2.2 Particle Filters

Particle Filters works by distributing particles uniformly throughout the known map, and then based on data from robot's sensors, the weight of each particle is resampled depending on the likelihood of the particle pose to robot pose. As the robot moves around particles converge on estimated robot pose. The Monte Carlo Filter is a type of particle filter that uses Monte Carlo simulation for its location likelihood calculation.

2.3 Comparison / Contrast

Particle Filter presents many advantages over EKF. Some of the important ones are:

- MCL is easier to implement
- MCL is unrestricted by a Linear Gaussian states-based assumption
- in MCL it is possible to control computational memory and resolution

In this project only one type of filter, particle filter - AMCL ,is used.

3 Simulations

The simulations are done in an environment using Gazebo and RViz tools for visualization. As previously stated, two different robot models were built using URDF format and localization exercise was performed. The first benchmark robot model was provided for the project and is called udacity_bot. The second robot model, ziz_bot, is based on udacity_bot with some changes which will be discussed in the following sections.

3.1 Benchmark Model:udacity_bot

3.1.1 Model design

The benchmark model udacity_bot shown in Fig. 2., is consisted of main body which is 0.4x 0.2 x 0.1 box, two wheels with 0.1 radius attached to left and right sides, and front and back spherical caster(with radius 0.049999) underneath the robot body for stability. The robot is also equipped with a front facing camera mounted to the front of the robot and a Hokuyo LIDAR attached to the top. Full robot model configuration can be found in the udacity_bot.xacro and udacity_bot.gazebo files.

3.1.2 Packages Used

The packages used for navigation are *amcl* and *move_base* . *amcl* node applies AMCL algorithm to perform localization and define robot position in the map, *move_base* creates creates local and global costmaps, as well as calculates local and global trajectories to the goal position. The map used in this project is jackal_race that was created by Clearpath Robotics.

3.1.3 Parameters

Parameters in the AMCL node , as well as move_base parameters were tuned to help robots to navigate through the map and reach the end goal. Here is list of some main parameters that were tuned and had effect on robot's performance:

- min and max particles where set to 15 and 200 respectively. Initial values 100 and 5000 were too large for simulation environment and required larger computational cost
- transform_tolerance was set to 0.3. As it is maximum amount of delay or latency allowed between transforms, setting this value to a too small will result in transform timeout and if too large will cause system to process outdated data.
- odometry model noise parameters odom_alpha1 to odom_alpha4 were set to the recommended values found in [1]. By tuning this parameters amcl algorithm improved and particles converged better.
- update and publish frequencies were set to 7.0 to match the system limitations. controller_frequency was also lowered to 8.0. All of the frequency parameters mentioned were tuned until warnings disappeared or significantly decreased. These parameters also helped with following the local path.
- obstacle_range and raytrace_range were tuned to 5.0 and 8.0 respectively to give robot further observation range
- the robot footprint was set to [[-0.1, 0.2], [0.1, 0.2], [0.1, -0.2], [-0.1, -0.2]], it determines the footprint of a rectangular robot.The corner points are specified in clockwise order.
- xy_goal_tolerance is set to 0.05 and yaw tolerance is set to 0.03.This parameters helped robot to get closer to the goal position, but setting them too low will result robot circling around in one place when it gets closer the end goal.
- meter_scoring is set to true, distances is measured in meters.

- robot velocity and acceleration parameters were set to default values.

The full parameter values can be found in amcl.launch and .yaml files in the /config folder

3.2 Personal Model: ziz_bot

3.2.1 Model design

The custom model ziz_bot shown in Fig. 3., is consisted of base_cylinder whith radius 0.2 and length 0.1, base_box with size 0.15x 0.5 x 0.04 and same origin as the base_cylinder, two wheels with 0.1 radius attached to left and right sides of base_box, and front and back wheels with radius 0.0499 length 0.05 underneath the robot body for stability. Sensors of the robot is placed in similar positions as for udacity_bot. Full robot model configuration can be found in the ziz_bot.xacro and ziz_bot.gazebo files.

3.2.2 Packages Used

The packages used for navigation and path planning is the same as packages used for udacity_bot.

3.2.3 Parameters

Parameter for ziz_bot is mainly set to same values as for udacity_bot, but some of them are tuned to match the robot model and improve performance:

- footprint is set to [[-0.28, 0.1], [0.28, 0.1], [0.28, -0.1], [-0.28, -0.1]] to match the model, as it is different than udacity_bot
- inflation_radius is set to 0.7, this parameter helped robot to plan its path closer to the middle of the pathway. significantly improved performance when turning around the corner of the hallway.
- odometry model noise parameters were decreased by half,it lowered particle cloud size.

The parameters that differs from udacity_bot can be found in amcl_ziz.launch and costmap_common_params_ziz.yaml files.

4 Results

Both robots was able to accomplish the task and reach the end goal. The udacity_bot completed the task in approximately 5 and half minutes, whereas ziz_bot managed to reach the end goal significantly faster only in 1 and half minutes. udacity_bot sometimes went far away from the global path and wondered around quite a bit, where ziz_bot stayed close to the global path most of the time.

4.1 Technical Comparison

Custom robot model ziz_bot performed better compared to benchmark model udacity_bot due to the tuned parameters disgusted earlier in section 3.2.3 . Larger inflation_radius allowed robot to move more to the center of the pathway and odom parameters helped with localization and reducing particle cloud size. Also changing caster to wheels might have an effect on movement velocity as well.

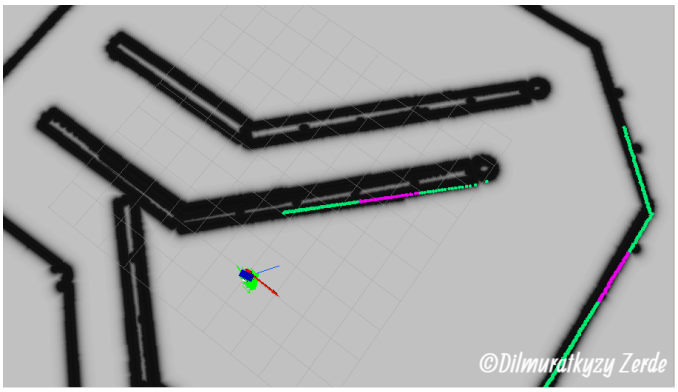

Fig. 4. udacity_bot reached the goal position

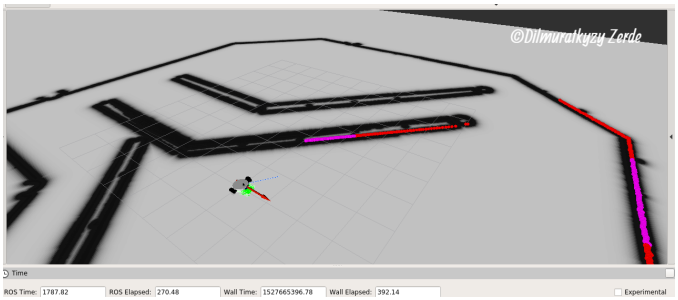

Fig. 5. ziz_bot reached the goal position

5 Discussion

Both robots were able to self-localize in the given map and successfully reach the set target position while avoiding walls and obstacles. Comparing the two tested models custom model ziz_bot performed better and reached the end goal faster due to the better tuned parameters and model specifications discussed earlier. In the kidnapped robot problem robot is picked up and placed in a different location without being notified. This problem differs from global localization problem in that the robot might still think it is in the previous position after being kidnapped. The regular MCL algorithm is not suitable for this problem as there might be no particles nearby the robots new pose after it has been kidnapped. However the AMCL algorithm configurations can be modified to be able to solve the this problem, as the number of particles can change over time. Parameters such as recovery_alpha_slow and recovery_alpha_fast have to be tuned. MCL could be used in an industry domain where robot needs to localize and move itself in a known static map. Examples could be house cleaning sweeping robot, warehouse robot used for moving goods.

6 Future work

To improve performance of robot, further tuning of parameters of the path planner is necessary. Trajectory scoring parameters like pdist_scale, gdist_scale, and occdist_scale should be looked into. With system with higher computational resource increasing amount of particles should also improve the accuracy. Additionally placement of the sen-

sors ,placing them higher above for example, can also be explored.

REFERENCES

[1] Spyros Maniatopoulos. Tuning amcl's diff-corrected and omni-corrected odom models. https://answers.ros.org/question/227811/tuning-amcls-diff-corrected-and-omni-corrected-odom-models/. answered Apr 10 '16.