Robot Localization with AMCL

Dennis Forward

Abstract—In this research, Adaptive Monte Carlo Localization (AMCL) of a wheeled robot was implemented in simulation. A simulation environment was built and run in ROS using RViz, Gazebo, and several other tools. For this environment nodes for movement and AMCL Localization were created and added to a launch file. A robot was then created using .urdf files, AMCL was implemented and tuned on that robot, and the robot was then able to successfully navigate a test course and reach a goal position.

Index Terms—Robot, IEEEtran, Udacity, Robot Localization, Adaptive Monte Carlo Localization.

1 Introduction

Localization is the challenge of determining a robots location in an environment. This is an incredibly important piece of Mobile Robotics, because if a robot needs to be mobile and make decisions to move to different points in an environment, it needs to know where it currently is first. To do this, mobile robots use different types of sensors. Some examples of common sensors are laser scanners, cameras, and motor encoders. A big challenge is that all sensor data has noise. Because of this noise and uncertainty, most common localization techniques use probabilistic algorithms to determine a robots location. In this project the Adaptive Monte Carlo Localization technique is explored on the Jackal Race World Map.

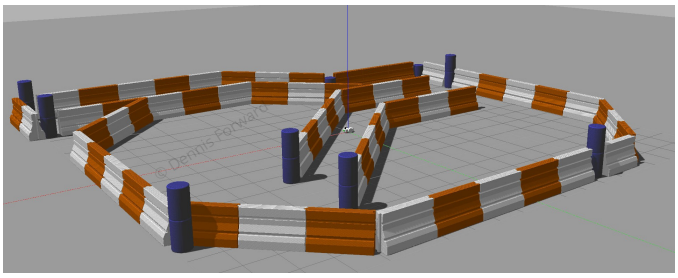

Fig. 1. Jackal Race World Map.

2 Background

There are three main types of localization problems. The first is Position Tracking, or Local Localization, where the robot knows its initial position and needs to track where it is when it moves. A second is Global Localization, which is a more difficult problem. In Global Localization, the robots initial pose is unknown, and it needs to determine where it is on a map. The third and most difficult problem is the Kidnapped Robot problem. This is similar to the Global Localization, but in this case the robot could be picked up and moved to a different place on the map at any point in time, and must be able to relocalize itself. The Kidnapped Robot problem is not a common problem, but it is one of the worst case scenarios in localization that robots should be designed for. All of these localization problems become even more complex when the environment is changing and not static, but for the purposes of this project, the environment will be static. There are many algorithms for localization, and four common ones are Extended Kalman Filters (EKF), Markov Localization, Grid Localization, and Monte Carlo Localization (MCL). In this course we explored using the EKF and MCL algorithms for localization. EKF is a method for filtering noisy sensor data, and the Monte Carlo method uses particle filters to track a robots position.

2.1 Kalman Filters

Kalman Filters are estimation algorithms that are used to estimate the value of a variable in real time, as data is being collected. They are common because they capable of quickly providing accurate estimates from data with large uncertainties. Kalman filters use a two step process where a measurement is taken, and then a state prediction, where the algorithm uses information from the current state, as well as the expected uncertainty in sensor readings, to predict what the future state would be. After iterating through these two steps the algorithm quickly converges near the actual values. Extended Kalman Filters are a more complex version of Kalman Filters, that are able to deal with non-linear systems. One of the biggest restrictions with EKFs is that they are only able to deal with Gaussian distributions of noise and uncertainty. The algorithm breaks down when the uncertainty distribution is not Gaussian.

2.2 Monte Carlo Localization

Monte Carlo Localization (MCL) uses particles to localize the robot. Each particle has a position and orientation and is an estimate of a robots position and orientation. During each iteration of sensor data acquisition, the particles are upsampled, during each iteration. MCL is common because it is easier to program and can deal with non-Gaussian distributions of noise, making usable for a much wider array of robot and sensor types. In the MCL algorithm, particles are spread randomly around the robots map. Each particle is an estimation of the robots pose. During the resampling process of the MCL algorithm each particle on the map is assigned a weight, based on how close the pose of the particle matches with the robots pose. During each iteration, only the particles with the highest weights survive, and

end up converging on the robots actual pose. After several iterations of the algorithm, the particles will converge, and accurately estimate the robots pose. An extension of Monte Carlo Localization, is Adaptive Monte Carlo Localization (AMCL). In AMCL the number of particles is adjusted over time, which offers big computational advantages over standard MCL. In this project, AMCL was used for localization. Figure shows further information on a comparison of EKF and MCL.

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

Fig. 2. MCL vs EKF Localizaion.

- example

1) example

3 Model Configuration

In the amcl.launch file, the minimum and maximum number of particles was set. The number I chose for the maximum was based on the computing resources available inthe Udacity workspace environment, and 250 seemed to work to navigate the robots well, without slowing down the system significantly. The minimum number was chosen with the assumption that less than 20 particles may result in a less accurate localization. The odomalpha parameters are initially set to a default of 0.2, but because this project was in a simulated environment, there was no noise in the odometry data. Because of that i reduced all four odomalpha params to 0.01, which cause the AMCL particles to converge much tighter around the robots actual pose. The obstaclerange determines how close of an obstacle detected should be added to the cost map. The raytracerange is the distance away from the robot where the robot clears the costmap. This is useful for preventing errors and noise from adding up and affecting the navigation. The inflationradius was set to 0.5 m. This value is the minimum distance the robot will allow between itself and obstacles. This is a balance between the robot being able to navigate through tight spaces and how critical it is that the robot never bump into anything. If you want the robot to navigate through tight spaces easily this should be small, and if you want the robot to be more conservative with how far from obstacles it stays, this should be larger. The global and local costmap updatefrequency and publishfrequency determine how quickly the system updates the costmaps. These need to be tuned to the system capabilities to make sure updates and publishes arent missed, causing data mismatches. The Model Configuration that was used for both robots in this project can be seen in Figure 3.

Model Configurations	
amcl.launch	
min_particles	20
max_particles	250
initial_pose_x	0
initial_pose_y	0
initial_pose_a	0
odom_alpha1	0.01
odom_alpha2	0.01
odom_alpha3	0.01
odom_alpha4	0.01
costmap_common_params	
obstacle_range	5.0 m
raytrace_range	8.0 m
inflation_radius	0.5 m
transform_tolerance	0.2
global_costmap_params	
update_frequency	0.5
publish_frequency	0.5
local_costmap_params	
update_frequency	0.5
publish_frequency	0.5

Fig. 3. Model Configuration

4 Results

Both the Classroom robot, and the custom robot were able to successfully navigate the map and make it to the goal position. At the beginning of the simulation the particles are spread around the robot, and as both robots moved, the particle array shrunk significantly and quick as the particles moved towards converging. The robot then navigated very cleanly towards the goal position in both attempts. Figures 4 and 5 show the initial and goal positions of the Udacity Bot, and Figures 6 and 7 show the initial and goal positions of the Dennis Bot.

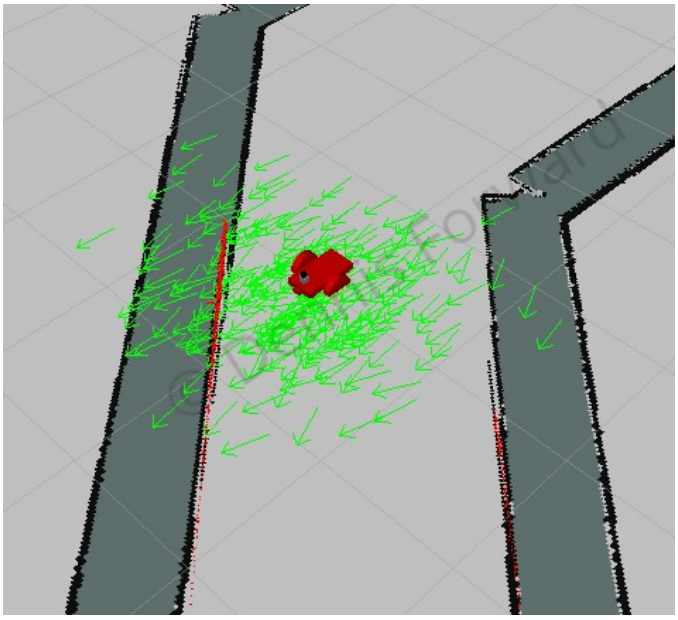

Fig. 4. Udacity Bot Initial Position.

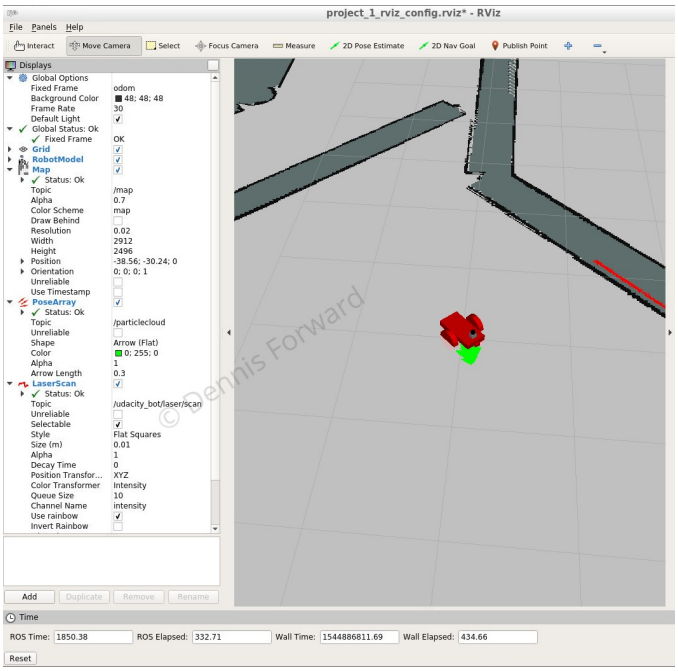

Fig. 5. Udacity Bot Goal Position.

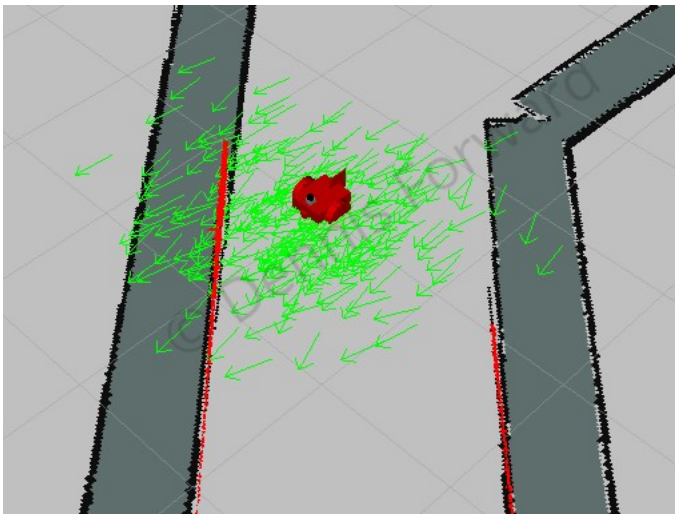

Fig. 6. Dennis Bot Initial Position.

5 Discussion

Both the classroom robot and the custom robot made it to the goal positions multiple times without having issues using Adaptive Monte Carlo Localization in the simulated environment. The AMCL particle clouds in the simulation for both robots converged very tightly after less than 10 seconds. This helped the robots easily avoid the walls on the map and make it to the goal positions in a clean and not erratic nature. It is suspected that one of the reasons the convergence happened so quickly in these simulations was that there was no noise in the odometry data. For AMCL to work with the kidnapped robot problem, there are methods where more complex algorithms can be used, that can introduce random particles or have more complex weighting methods. In order to implement this, further research would need to

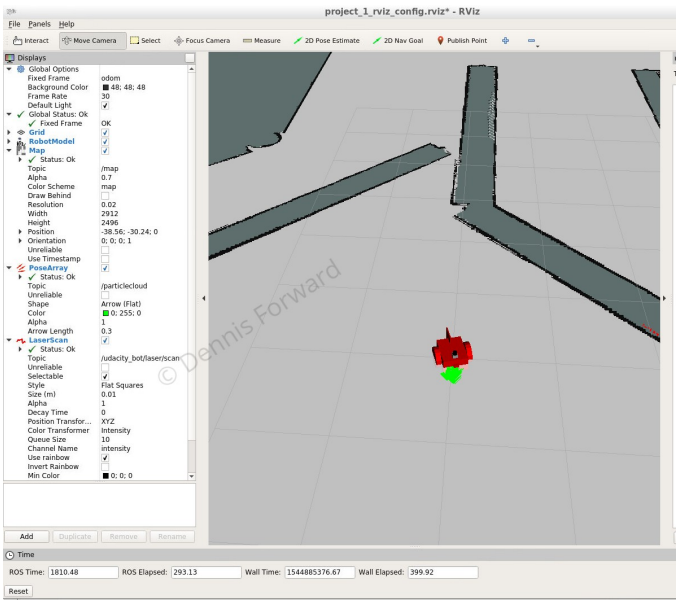

Fig. 7. Dennis Bot Goal Position.

be done and more parameters would have to be defined and tuned. A perfect example of this kidnapped robot problem and MCL/AMCL in industry is with an iRobot Roomba or other brand robotic vacuum, where during operation a person picks it up and moves it to a different room that they want cleaned. In this scenario the robot needs to be able to relocalize itself after this kidnapping, so it can find its way back to the charging station after it has completed the cleaning.

6 Future Work

The functionality and learning from this research could be improved in several ways if more work is done in the future. In order to better tune the AMCL algorithm and parameters that would translate to the physical world, noise could be randomly introduced into the odometry data.This would make the simulated environment more closely match a real world application. More parameters could also be defined that would help the robots used recover from the kidnapped robot problem. This would further the robustness of the localization of these robots when they are put in real world type scenarios. More sensors or different sensor placement could also be used. On both robots used for this research, the 2D laser scanners were at a certain height above the ground, but the walls in the maps were wider at the bottoms than they were at the height of the laser sensor. This creates an error in the actual distance that the robot is away from the wall versus how far the robot thinks it is away. This can be tuned using some of the parameters above, if all the walls it encounters are the same, but if it encounters different shape walls, the distance would still be off. To fix this more sensors could be used, or other types of assumptions about walls and the environment could be made. Overall in this research a great baseline AMCL localization algorithm was tuned and tested, and could be easily transferred to a robot in the real world.