Project: Where Am I?

Luis F. W. Batista

Abstract—This report presents a theoretical comparison between Extended Kalman Filters and Adaptative Monte Carlo localization, two widely-used algorithms for localization in robotics. For evaluation, two robot models, are implemented in a ROS package. Using AMCL and Navigation packages, the robots transverses through a simulated map with obstacles and reach a specified position. An outline of the components used is explained along with an overview of how the parameters were adjusted to enable the robots to successfully reach the target.

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

1 Introduction

AUTONOMOUS robots must be able to navigate through space based on information provided by sensors. In order to plan the path and control the actuators, the robot must be able to estimate its position on a given map. This task is known as Localization and is fundamental competence required by a robot. Knowing its own location is an essential precursor to deciding future actions [1].

Several approaches exist to solve the localization problem. Two popular localization algorithms are Extended Kalman Filter Localization Algorithm (EKF) [2] and the Adaptive Monte Carlo Localization Algorithm (AMCL) [3].

The main goal of this project is to experiment with the localization algorithms. In order to achieve that, AMCL and Navigation Stack packages form ROS are used to localize and move the robot in a simulated environment using Gazebo.

Two robot models are used for evaluation. The box-shaped Udacity Bot and the R2D2 Bot, loosely resembling the iconic robot from Star Wars. By observing the results in RViz, the parameters of each algorithm are tuned to enable the robots to navigate through the map and reach a given position.

2 Background

Localization is fundamental for developing autonomous mobile robots. In order to decide where to move, the robot first needs to estimate its position. In the real world, sensors are noisy and actuators might not be very precise. Moreover, external influences such as wind, slippage and many others. This combination increases the challenges of implementing a precise localization.

The localization problem can be separated into three different types: local localization, global localization, and kidnapped robot. The local localization is the simplest, where the robot's pose is known and it is necessary to update its position as it moves despite the sensors noise. The global localization is when the initial pose of the robot is unknown. The last problem is known as the kidnapped robot, where at any given time the robot can be moved to another location in the map.

In this paper, Kalman Filters and the Monte Carlo Localization algorithms are compared, and the Adaptative Monte Carlo Localization algorithm is evaluated in a simulated environment.

2.1 Kalman Filters

The Kalman Filter (KF) is one of the most practical algorithms to estimate the state of a system based on uncertain information such as noisy sensors. It was first introduced by Rudolf E. Klmn and used in trajectory estimation for the Apollo program. It can achieve accurate results without a lot of data and does not require many computational resources.

Kalman filters have a probabilistic approach to improve the accuracy of the estimations and can be used to solve local localization problems. First, an initial prediction of the state is made. Next, starts a cycle of state prediction and measurement update. This cycle is continuously repeated using sensor data provided in real time.

Traditional Kalman Filters assume that the system is linear. However, it is not the case in many applications. One solution is to approximate a non-linear equation using some terms from the Taylor Series. This is known as Extend Kalman Filters and they are widely used in the field of Robotics.

2.2 Particle Filters

Particle filters are another common approach to solve to Localization problem in Robotics. The particles are virtual elements that represent possible positions and orientations of the robot. The algorithms can estimate the actual pose of the robot by randomly distributing the particles across the map and filtering the ones that are more likely to be correct by comparing with sensor information.

Opposed to the EKF, the MCL algorithm can be successfully used in Global Localization problems. Moreover, the measurement noise is not restricted to Gaussian distribution which makes it a good option in several applications.

The computational power required proportional to the number of particles used. The Adaptative Monte Carlo Localization algorithm adjusts the number of particles according to the overall accuracy of the estimation, making it possible to lower the computational resources required and the particle distribution converges.

2.3 Comparison / Contrast

Given their intrinsic characteristics, each algorithm can be more suitable for different applications. The Table 1 presents a summary of the key differences between each other.

amcl parameter	Udacity Bot	R2D2 Bot
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

TABLE 1: MCL and EKF Comparison

Kalman Filter would be a good choice to be used in a system with high computational restrictions and where the measurement noise has a Gaussian distribution. If solving global localization is important. MCL is a better alternative. While it requires higher computational resources, it can be controlled by adjusting the number of particles.

The work presented in this report will focus only on the particle filter approach. More specifically using the AMCL implementation available in ROS.

3 Simulations

The Gazebo simulation environment was used to test and experiment with the AMCL algorithm. Two robot models were used for comparison.

Using the AMCL and Navigation packages and the ROS environment, the robots had to navigate through the provided map shown in Figure 1, and reach a given position. The performance was compared by observing the convergence of the particles displayed in RViz and the time needed to reach the final pose.

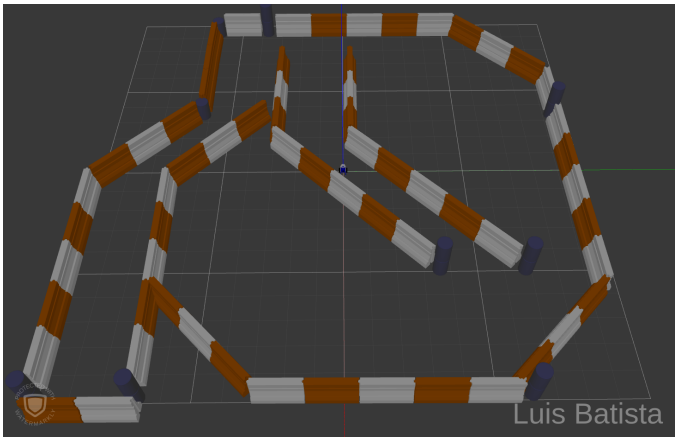


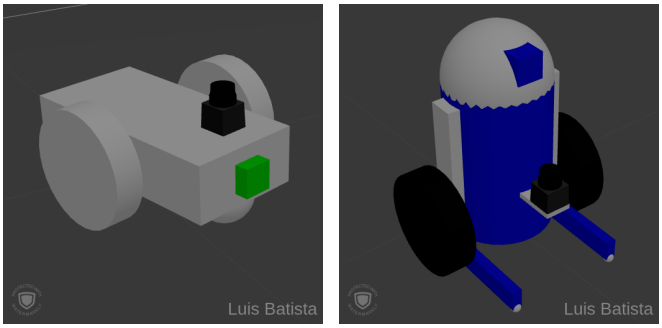
Fig. 1: Jackal Race Map

3.1 Achievements

After adjusting the parameters, the calculated particles converged and allowed the robots to navigate through the map. Both models were able to successfully reach the goal position in less than 65 seconds.

3.2 Model design

The simulation was done with two different models. A benchmark model, referred as Udacity Bot (Figure 2a), and a personal model, referred as R2D2 Bot (Figure 2b). The benchmark model was provided by Udacity along with the course materials. The R2D2 Bot kept the same wheels but its chassis was redesigned to resemble the R2D2 robot from the Star Wars movie.



(a) Udacity Bot (b) R2D2 Bot

Fig. 2: Robot Models

The mass, inertia, footprint, and wheel design were kept as similar as possible to minimize the variables that could impact on simulation results. The goal was to enable a better comparison of the AMCL and navigation parameters. For detailed information about the robot models, please refer to the implementation code available [4].

Both robots have a camera and a laser sensor. The camera is not used for localization and in the R2D2 Bot, it was repositioned in the head of the robot. The location of the Laser sensor was kept in the same position to guarantee similar measurements results. If the hight was changed, it would be expected that the measured distance would vary given that the base of the obstacles has an angled surface.

3.3 Packages Used

The robot models were implemented in a ROS package called *udaicty-bot*. In order to spawn the correct robot model, it is only necessary to adjust which robot description is included in the *world.lauch* file.

To implement localization and movement of the robots, the following packages were used:

- **amcl** - Package used for localization. By adjusting a set of parameters it is possible to customize the overall behavior of the filter. It can also be adjusted based on the laser and odometry specifications.
- **move base** - Package used to calculate the path and control the robot to reach the goal.
- **xacro** - Package used to describe the robot model. It parses a simplified description and generates the full urdf XML description of the robot.

3.4 Parameters

For comparing the localization performance, different parameters were used for each robot. The Table 2 shows the final parameters of the amcl package and the parameters used in the move base package are listed in Table 3. Note also that the costmap parameters, described in Table 4, remained unchanged in final tests.

amcl parameter	Udacity Bot	R2D2 Bot
min_particles	10	20
max_particles	100	500
update_min_d	0.05	0.001
update_min_a	$\pi/6$	0.001
resample_interval	2	1
trasnform_tolerance	0.1	
laser_min_range	0.1	
laser_max_range	10.0	
laser_model_type	likelihood_field	
odom_model_type	diff-corrected	
base_frame_id	robot_footprint	

TABLE 2: amcl parameters

base_local_planner parameters	Udacity Bot	R2D2 Bot
controller_frequency	10	
holonomic_robot	false	
yaw_goal_tolerance	0.05	0.1
xy_goal_tolerance	0.1	
meter_scoring	true	

TABLE 3: base_local_planner parameters

Udacity and R2D2	global_costmap	local_costmap
map_type	costmap	
obstacle_range	3.0	
raytrace_range	8.0	
transform_tolerance	0.1	
footprint	[[0.2, 0.2], [0.2, -0.2], [-0.2, -0.2], [-0.2,0.2]]	
inflation_radius	0.6	
observation_sources	laser_scan_sensor	
global_frame	map	odom
robot_base_frame	robot_footprint	
update_frequency	10.0	
publish_frequency	10.0	
width	40.0	5.0
height	40.0	5.0
resolution	0.10	0.05
static_map	true	false
rolling_window	false	true

TABLE 4: costmap parameters

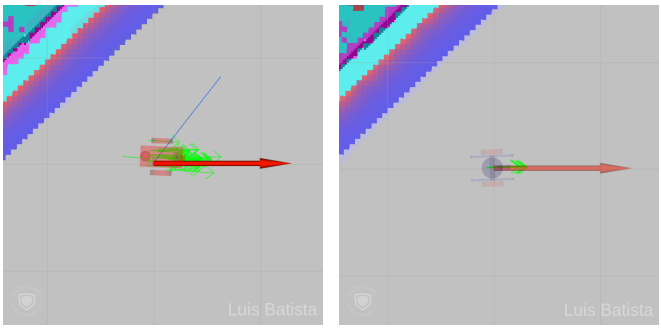
To explore the results in the localization perfomance, different values were selected for parameters such as *min_particles*, *max_particles*, *udate_mind_d*, *update_min_a*, and *resample_interaval*.

Other parameters including *trasnform_tolerance*, *controller_frequency*, *update_frequency*, and *publish_frequency* had to be empirically adjusted based on the computer used for simulation. Increasing the frequency gives better result but require more computation capability.

The remaining parameters, namely, *odom_model_type*, *base_frame_id*, *holonomic_robot*, *observation_sources*, and *robot_base_frame* were selected according to the robot model definition.

4 Results

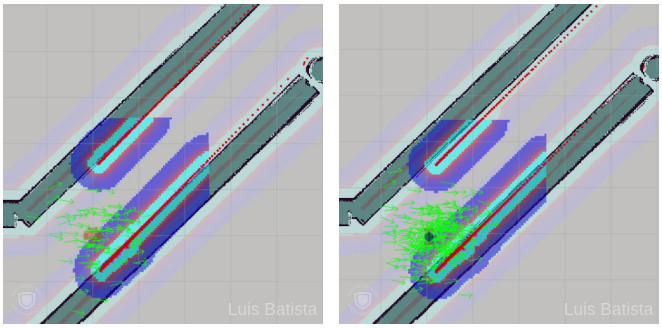
By running the simulation several times, it is possible to observe that both robots were able to reach the final position successfully. The elapsed time required the reach the goal always remained under 65 seconds and varies on each execution. The distribution of the particle cloud when reaching the final position can be observed in the Figure 3.



(a) Udacity Bot (b) R2D2 Bot

Fig. 3: Robots in Final Position

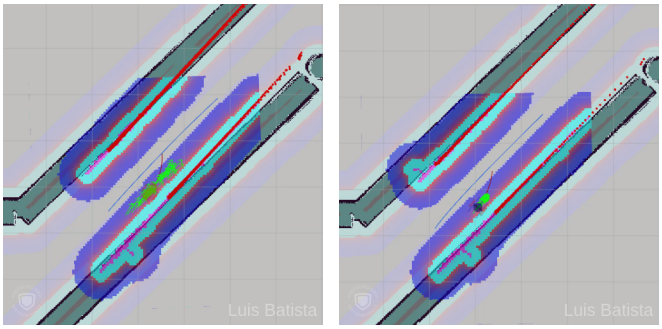
Figure 4 shows the initial distribution of the particles. In the case of the R2D2 robot, the higher number of particles incurs in a higher probability that some particles are correctly placed close to the actual location of the robot.



(a) Udacity Bot (b) R2D2 Bot

Fig. 4: Initial Position

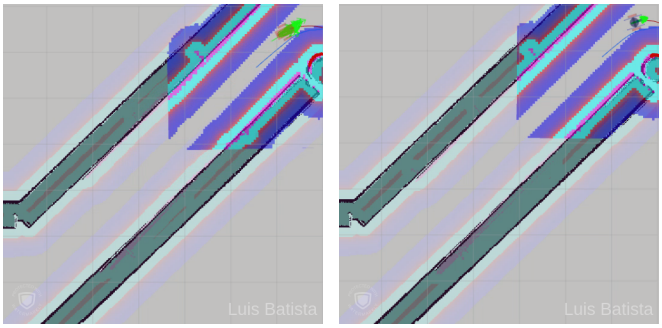
As a result, the particles tend to converge quickly after the R2D2 starts moving. The Figure 5b shows clearly shows the difference in the distribution of the particles after each robot has moved almost the same distance.



(a) Udacity Bot (b) R2D2 Bot

Fig. 5: First Particle Convergence

In the case of the Udacity Bot, the particles start to converge only when it is close to making the first sharp turn to the right at the end of the middle passage (Figure 6a). Despite the longer time to converge, the robot is able to follow a smooth path from the start to the goal position.



(a) Udacity Bot (b) R2D2 Bot

Fig. 6: First Particle Convergence

The R2D2 Bot have achieved better performance using the parameters described on Tables 2 and 3. Nevertheless, as described previously in Section 3.2, the physical characteristics and the sensor placement is very similar on both robots. It allows to use the better-tunned paraments also in the Udacity Bot and get very similar results. The Figure 7 shows the benchmark robot reaching the final position with converged particles by using the same parameters as the R2D2 Bot.

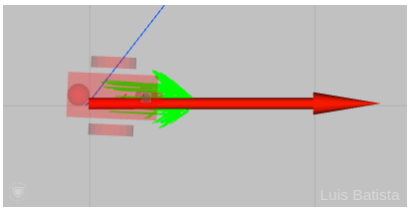


Fig. 7: Udacity Bot with tunned AMC Parameters

5 Discussion

The results clearly indicate that increasing the number of particles and updating the particles in shorter cycles, resulted in a better convergence. This explains why parameters used for the R2D2 Bot performed better when comparing to the parameters used for the Udacity Bot.

It was also noticed that the distribution of the particles becomes less precise when the robot reaches the final position. As the orientation is facing towards an open area, the distance of the objects in larger, increasing the noise and reducing the accuracy of the estimated pose.

The AMCL algorithm is a feasible option to be used in cases where global localization is necessary. If the initial position of the robot was unknown, the particles would have to be evenly distributed across the entire map. The required computational resources would decrease as the particles converged to the actual position of the robot. However, the algorithm would not be sufficient in case of the kidnapped robot problem. Robustness for such situations become mandatory in real-world robots such as autonomous vacuum cleaner for example.

While this report focused mostly on the evaluation of the AMCL algorithm for localization, there are some situations where the EKF would be a better option. This would be the case if the available hardware resources are very limited and the primary goal is to solve local localization.

6 Conclusion / Future work

This project successfully achieved the goal of learning how to use and adjust the parameters for localization using AMCL algorithm. Using the simulated environment, it was possible the explore the localization parameters and quickly observe the results. It was also possible to experiment with the computational impact when trying to optimize the results.

6.1 Modifications for Improvement

Despite reaching the initial goals there are several improvements and further investigation that can be done.

- Adjust amlc parameters according to the expected noise from the sensors.
- Modify the move_base parameters to get better path planning results.
- Make use of additional sensors in different locations of the robot.
- Explore the impact of physical characteristics of the robot, such as mass, inertia, and friction coefficients.
- Repeat the tests using a more realist robot model, such as the TurtleBot 3 available in ROS.

6.2 Hardware Deployment

Exploring the results on simulation is great for learning and also important for initial adjustments. In order to deploy in real hardware, it is very important to readjust the parameters according to the hardware performance. In case of using limited hardware, such as a Raspberry PI, the AMCL could be processed in a more powerful computer. The following steps provide an outline of the required tasks.

1) Recreate simulation model based on a real robot.
2) Adjust the AMCL and Move Base parameters according to the new robot.
3) Deploy and adjust the frequency of update according to computation resources available.
4) Test and tune the parameters.

References

[1] S. Huang and G. Dissanayake, *Robot Localization: An Introduction*, pp. 1–10. 2016.
[2] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, pp. 401–422, March 2004.
[3] S. Thrun, "Probabilistic robotics," *Commun. ACM*, vol. 45, Mar. 2002.
[4] L. Batista, "Project: Where Am I?," May 2019.