Localization in Simulation

Kevin Fructuoso

Abstract—The purpose of this report is to apply the principles of Localization in order to determine a robots pose. There are many different methods of localization with the two most popular methods being Kalman Filters and Particle Filters (otherwise known as Monte Carlo). This report will discuss these two methods and apply Monte Carlo Localization (MCL) in simulation. Two simulated robots one designed by Udacity and the other designed by the author were configured to demonstrate the application of localization tuning parameters. These localization parameters were tuned for each in order to successfully localize the robots pose and navigate to a target position.

1 Introduction

Localization is an integral component of robotics design for mobile robots. Localization is the method for a robot to determine its pose within its environment. There are three main challenges to robot localization - position tracking (local localization), global localization (within an environment), and the kidnapped robot (moving a robot to a new location without the robot knowing). These challenges are addressed using mathematical probability algorithms and various sensors to calculate the position and orientation of the robot. Implementing localization is an important concept as a robot may not always know its starting location and will have to deduce where it is within its environment. There are many methods to apply localization in robots the four most popular are the Extended Kalman Filter (EKF), Markov Localization, Grid Localization, and Monte Carlo Localization. This report will discuss the Extended Kalman Filter and Adaptive Monte Carlo Localization (AMCL) algorithms.

1.1 Project Goal

The goal of this project was to apply Adaptive Monte Carlo Localization (AMCL) in simulation with two different robot configurations in order to successfully determine the robots pose within the map, as shown in Figure 1. The simulations were visualized in the Gazebo and RViz software applications. The simulation was created using the Robot Operating System (ROS) using multiple packages. With these packages, the localization hyper-parameters can be adjusted to modify the localization performance of the robot. The performance is visualized by the spread and direction of the green arrows shown in Figure 1. The directions should point in the forward direction of the robot and the arrows should be converging on a location to denote that the position of the robot is known with high confidence.

2 Background

As mentioned in the Introduction, there are three major localization challenges - local, global, and the kidnapped robot. This report focuses on the EKF and AMCL localization methods and how they operate. Both methods have their own advantages over the other. The AMCL implementation was chosen for the map localization task mainly due to the EKF algorithm's limitation in being unable to globally localize a robot within the map.

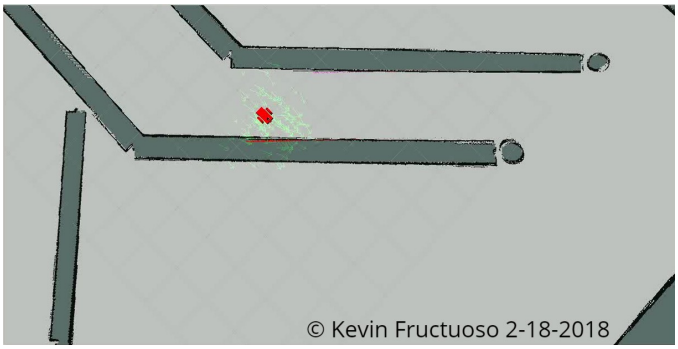

Fig. 1. RViz simulation robot (red), map, and particles (green arrows)

2.1 Kalman Filter (KF)

The Kalman Filter is an estimation algorithm used to approximate the true value of a measurement in real time as data is being collected. It can take data with a lot of noise and produce a very accurate estimate very quickly. The KF algorithm is a cyclic process. First makes an initial estimate of the true value. This starts the cycle between State Prediction and Measurement Updates. After the initial estimate, the algorithm predicts the new state of the robot. Then the measurement is taken to update this new state. The result of this is used to then create another state prediction and so on to continue the process.

This method assumes measurements are linear and the state space can be represented by a uni-modal Gaussian distribution. However, not all models can realistically be analyzed as such. This is wheret the EKF method begins to shine. The EKF method can be used in situations where the measurements are non-linear and the state space is not uni-modal. EKF is capable of this feat by using local linear approximations of infinitesimally small sections of the transfer function. In these small slices, the linear approximation is sufficient in estimating the outcomes. The KF may also be improved by sensor fusion - combining data from multiple sensors to obtain a more accurate estimate of the true value.

The KF cannot be used in global localization. This is the main reason it was not chosen for this project. The KF solves the local challenge and may also be extended to solve the kidnapped robot challenge (outside the scope of this report).

2.2 Monte Carlo Localization (MCL)

Monte Carlo localization (otherwise known as Particle Filter Localization) is the most popular localization algorithm. This algorithm implements many particles as guesses of where the robot is located and its orientation with then environment. As the robot moves and data is collected over time, the algorithm will update each particle and calculate the probabilities that the particle is correct. Based on the probability, the algorithm decides to keep or remove specific particles. This can be extended to what is known as AMCL. AMCL dynamically adjusts the number of particles over time. This may save valuable computation time.

This method may be used for a wide range of state spaces that are discrete or continuous. The MCL algorithm can be used for local and global localization. However, it is not suited for the kidnapped robot challenge.

2.3 Comparison / Contrast

Each localization method has its own set of advantages and disadvantages. Table 1 details a comparison between the MCL and EKF algorithms. AMCL was selected for this report for the following reasons:

- Solves global localization
- Can take any type of state space models
- Provides memory and resolution control
- More robust algorithm

TABLE 1
MCL vs. EKF Comparison

	MCL	EKF
Measurements	Raw Measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency	+	++
Ease of Implementation	++	+
Resolution	+	++
Robustness	++	X
Resolution Control	Yes	No
Global Localization	Yes	No
State Space	Multi-modal Discrete	Uni-modal Continuous

3 Simulations and Model Configuration

Two robot configurations were implemented for testing in this report. Figure 2 shows each configuration.The left image shows the robot specified in the Udacity lessons. It uses a camera on the front of the robot with a laser-range finder on top. The right image shows the custom configuration created by the author. The shape and size were modified and the camera and laser-range finder were raised to the dome section of the robot. This configuration was specified to be larger and lighter.

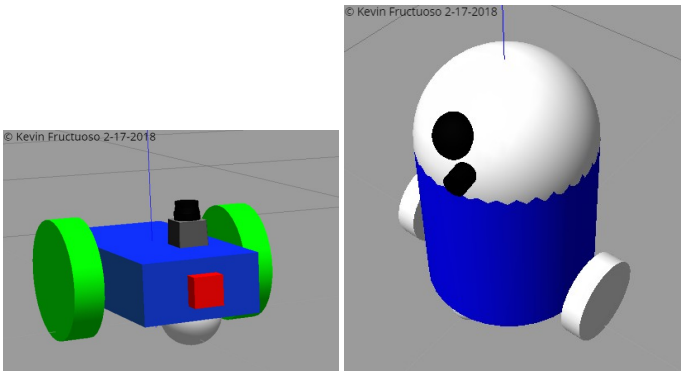

Fig. 2. Udacity lesson configured robot (left). Custom-defined robot (right).

3.1 ROS Packages used

Using multiple ROS packages, hyper-parameters were defined and tuned for each robot model. These packages use a variety of ROS topics to receive measurement data and publish motion commands. Both models implement the same sensors and use the same ROS topics to achieve their results. The ”amcl” package uses data from the laser range-finder from the ”udacity_bot/laser/scan” topic. [1] Navigation of the robot is handled by the ”move_base” package that receives information from the ”odom” and ”udacity_bot/laser/scan” topics and publishes commands to the ”cmd_vel” topic. [2] Map information is sent to the ”map_server” package and transformed using the ”tf” package.

3.2 Achievements

Both robots were tuned to be able to localize and reach the goal position. Table 2 shows the simulation performances of each robot. There were several key parameters that required tuning for each robot. These will be discussed in each models section individually. Others were able to be kept the same across both models - these can be found in the raw data files within the Github project repository. [3]

TABLE 2
Simulation Performance

Robot	Localization Convergence (min)	Reach Target (min)
Udacity	3	25.5
Custom	1.5	10

3.2.1 AMCL Parameters

The AMCL parameters were able to be kept the same between the two robot configurations. The minimum and maximum particles were set to 5 and 100 respectively and were chosen to minimize computational burden on the development environment. Increasing the max particles will increase the amount of operations performed each cycle and results in slower overall performance. Similarly, the transform tolerance was set to 0.25 seconds to reduce computational overhead on the development environment. This parameter effectively controls the period of time of which

the AMCL mathematical transforms are valid. Larger values will result in more error and lead to longer convergence times. However, reducing computational overhead also results in accuracy decrease trade-offs. Additionally, all of the expected odometry noise was reduced to 0.001 in order to improve convergence time.

3.2.2 Common Move_base Parameters

A number of parameters were kept common between the two configurations. The ”meter_scoring” parameter was enabled in order to denote that distances should be expressed in the meters unit. The ”obstacle_range” was set to 1.0 meters and denotes that only objects detected within one meter are obstacles that need to be avoided. Increasing this parameter allows the robot to detect obstacles from a larger distance but may result in poor navigation. The ”raytrace_range” was set to 3.0 meters and signifies that the robot will attempt to clear out space in front of it up to three meters away when given a sensor reading. [1] The ”transform_tolerance” the costmaps tolerable delay in transform data and was set to 0.25 seconds. This parameter should be set to a proper value depending on the performance speed of the development environment. Both the local and global costmap update and publish frequencies were set to 5.0 and 2.0 Hz, respectively. These control the rates at which the costmaps are updated and publish to the ROS topics. Another important rate parameter is the ”controller_frequency” which was set to 3.0 Hz. This controls the rate that the ”move_base” package will publish motion commands to the robot. The costmap update/publish rates and the controller frequency were chosen to relieve the development environment from warning messages during runtime. It should be noted that reduced the controller and update/publishing rates result in accuracy decreases. The parameters should be tuned accordingly to minimize this trade-off.

3.3 Udacity Model

3.3.1 Model design

The chassis is a box shape that is 0.2 meters wide, 0.4 meters long, and 0.1 meters tall. This robot has two wheels set 0.4 meters apart making the robot have an effective square collision shape. It also has two casters to prevent from tipping over frontwards or backwards. The camera is aligned on the same plane as the chassis origin and is located on the front face of the body. The laser range finder is located on top of the bot and 0.05 meters away from the front edge. This model was specified with a weight of 15 kilograms

3.3.2 Parameters

There are a handful of distinct navigation parameters for this robot model. They are listed in Table 3. The ”pdist_scale” parameter determines how strictly the robot follows the defined global path to the target position. The robots maximum velocity in the x-direction was specified as well. Although the robot was able to reach the goal, the speed in the y-direction should be specified as well in order to ensure more consistent motion. These two parameters led to the robot straying away very widely from the given global path. The robot radius parameter is important to set properly to have reliable translation of motion from the navigation commands to the simulated model. If this model is not set properly, the model does not move in the expected fashion which leads to poor navigation. The inflation radius is the distance at which the planned trajectories will avoid known obstacles. This value should be properly set in accordance with the robots size and maneuverability. Some robots need more space while others do not.

TABLE 3
Udacity Robot Parameters

Parameter	Value
pdist_scale	1.2
max_vel_x	2.0 m/s
robot_radius	0.2 m
inflation_radius	0.3 m

3.4 Custom Model

3.4.1 Model design

The chassis is a combination of a cylinder and with a dome on top. The cylinder radius is 0.2 meters wide and 0.4 meters tall. The dome was created with a sphere of the same 0.2 meter radius with its origin placed at the top of the cylinder such that only the top half of the sphere is visually exposed. This robot has two wheels set 0.6 meters apart. Similar to the Udacity robot, it also has two casters to help prevent from tipping over frontwards or backwards. The camera and laser range finder were moved to the dome of the robot. The camera is located (0.15, 0.0, 0.3) meters in the xyz-planes from the origin of the cylindrical chassis. The laser-range finder is located (0.2, 0.05, 0.25) meters in the xyz-planes from the origin of the cylindrical chassis. This model was specified with a weight of 2 kilograms.

3.4.2 Parameters

The hyper-parameters for the Custom Robot model are given in Table 4. Some are discussed from the Udacity Robot model and others were added specifically for this model. The maximum y-direction velocity was specified here. Additionally, the x and y-direction accelerations were limited. The tolerances for the position (xy_goal_tolerance) and orientation (yaw_goal_tolerance) were specified in order to allow the navigation to account for small differences in the target pose. If these are not set appropriately, the robot may reach close to the final position but will continue to oscillate and search for a very small acceptable range of poses. The ”pdist_scale” was modified in order to help the robot more accurately follow the determined path.

4 Results

Both robot models were able to localize and reach the desired locations as detailed in Table 2. Figure 3 shows the localization results of each robot.

TABLE 4
Custom Robot Parameters

Parameter	Value
pdist_scale	0.75
max_vel_x	0.4 m/s
max_vel_y	0.4 m/s
yaw_goal_tolerance	0.05 rad
xy_goal_tolerance	0.15 m
acc_lim_x	0.5 m/s^2
acc_lim_y	0.5 m/s^2
robot_radius	0.3 m
inflation_radius	0.5 m

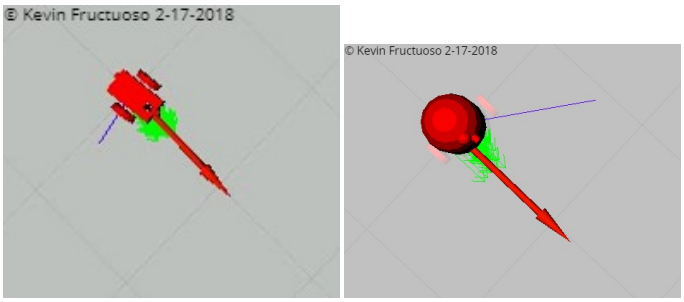

Fig. 3. Localization results for the Udacity lesson configured robot (left) and Custom-defined robot (right).

4.1 Localization Results

4.1.1 Udacity Robot

Although the Udacity robot is capable of localizing and reaching the target position, the tuning is far from optimized. The localization parameters operate fairly well as the robot is able to reliably localize without having to detect very much of the map. This can be seen in the videos located in the Github repository. [3] At the target position shown in Figure 3, the direction of the particles is correct and the spread has converged nicely. The navigation parameters cause the overall performance to be very poor. The robot begins with an initial discovery period where it is oscillating in position around for a few minutes in order to find a local path to take. The robot also does not follow the given path tightly leading to the incredibly long time to reach the target position.

4.1.2 Custom Robot

The Custom robot tuning performs much better than the Udacity robot. The localization is accurate and fast as it converges twice as fast due to being able to move immediately after launching. From the final result shown in Figure 3, it can be seen that the localization is as accurate as the Udacity robot, if not more. The directions are correct and the spread appears to be slightly more converged. Similar to the Udacity robot, the Custom robot has an initial discovery period. However, instead of oscillating in a single position, it first begins moving in the opposite direction of the global trajectory plan. This happens due to the robot's original orientation and position along with the "obstacle_range" parameter. The robot starts by facing a wall within the obstacle range and attempts to navigate to avoid the obstacle. This navigation directs the robot in the opposite direction until it can self-correct. After this discovery period, the robot followed the global path very closely due to the "pdist_scale" and speed/acceleration settings.

4.2 Technical Comparison

Discuss the difference of the layout, parameters, performance etc. between the benchmark robot and your robot. It is acceptable for your custom robot to perform worse than the provided robot. The focus is on learning and understanding, not performance.

5 Discussion

From the discussed results, it is clear that the Custom robot model performed better than the Udacity robot model. More effort is required to optimize the Udacity robot tuning as defined in this report.The Custom robot performed better due to its incredibly light mass in comparison to the Udacity robot. This allowed it to be more responsive to motion commands. Additionally, more hyper-parameters defining motions were configured for the Custom robot. Both require some tuning for better obstacle avoidance to remove their initial discovery periods. The Custom robot really shined in its ability to follow the global plan very closely. It did not stray significantly from the path allowing it to reach the navigation target much faster. The localization for the Custom robot was twice as fast as well due to it's improved motion over the Udacity robot. While the Udacity robot simply kept turning in place before moving, the Custom robot started with motion and was able to collect more useful data to apply towards localization.

The normal implementation of AMCL is incapable of solving the kidnapped robot problem on its own. In order to address this, additional verifications can be added to the sensor readings in order to detect if the motion executed may happen naturally. If it determines that the motion was not natural, the obtained localization thus far would be restarted from scratch in order to begin the process anew.

Localization is often used in scenarios where system state measurements are noisy. In addition to position tracking of a mobile robot as discussed in this report, it may be applied to detecting the fluid level of tank. It may also be used outside of the field of Engineering as a whole. Economists may use localization to predict economic trends as well. Within industry, the AMCL method specifically may be used in an automated factory setting similar to an Amazon warehouse. AMCL may be used to initialize a new robot item picker after boot up or after being worked on in case of faulting equipment or sensors.

6 Conclusion / Future work

Both robots were capable of achieving the desired results of localization and navigation to the final position using AMCL. The hyper-parameter trade-offs of accuracy versus processing time are acceptable to meet the project requirements.

Future work to continue this report may focus on the addition of different types of sensors. For example, the camera, which was not used at all for localization or navigation, may be modified to pass data to determine if an obstacle is

blocking the robots path. Additional effort may determine that the robot would benefit from different motion control methods.

6.1 Modifications for Improvement

As mentioned on numerous occasions, both robot tuning models can be further optimized for improvement. Improvements for the Udacity robot model include:

- Executing planned navigation path more closely
- Obstacle avoidance to remove initial discovery period
- Speed settings to improve overall speed to reach target position

Improvement suggestions for the Custom robot model include:

- Obstacle avoidance to remove initial discovery period
- Speed settings to improve overall speed to reach target position

REFERENCES

[1] Open Source Robotics Foundation, "Amcl package summary," 2017.
[2] Open Source Robotics Foundation, "Move base package summary," 2016.
[3] Kevin Fructuoso, "Udacity_bot localization project repository," 2018.