Localizing two types of mobile robots

Christopher T. Angell

Abstract—Localization and path planning for robotics is essential for getting a robot to where it needs to be for it to execute the actions that make robotics an increasingly important part of society. To localize the robot several ways are possible. Here we employ the adaptive Monte Carlo localization (AMCL) algorithm to locate a robot in a model world using the Gazebo simulation environment. Two different robot models are used, a two wheel model with differential drive, and a four wheel model that uses skid steering. The robots are able to in the simulated environment to navigate a corridor to a final goal state.

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

1 Introduction

Localization of a robot consists of two distinct problems, path tracking and global localization. In path tracking the robots position, already known in a map, is updated accurately as it moves based both on odometry and measurements. For global localization the robots position in a map is determined out of all possible alternatives. That is, determining where the robot is without *a priori* knowledge of where it could be. Once the robot is correctly located within a map, then path planning algorithms can be used to plan trajectories for the mobile robot.

Here, we focus on the localization algorithms, utilizing an adaptive Monte Carlo localization algorithm (AMCL) in the ROS [1] framework to localize a simulated robot. The Gazebo simulation package was used [2]. Two different robots models were implemented, one a two wheel differential drive robot, and the other a four wheel skid-steered model. First we discuss the AMCL algorithm in the context of alternate choices, and then present the result of the simulation implementation of the AMCL, then parameter tuning of the models will be discussed. Next, the results will be discussed, and finally, ideas for future work will be presented.

2 Background

Of path tracking and localization, the approaches include Kalman filters, Markov localization, and particle filters techniques including Monte Carlo localization.

2.1 Kalman Filters

The Kalman filters combines both odometry information with measurement information to provide an updated position of a robot. With precise odometry information measurement could be neglected, but given uncertainty in the gearing of the robot, traction on the terrain, resistance etc., odometry may have uncertainty in it. Measurements can be successfully combined to provide a more accurate update of the robot position. The principle limitation of the Kalman filter is that it must assume that the uncertainties are Gaussian. The extended Kalman filter makes the assumption of locally Gaussian uncertainties for non-linear problems. It is also limited in that it can only update the location of the robot, it cannot locate the robot globally.

2.2 Markov localization

Markov localization methods divide the map up into a grid and uses Bayesian filtering to update the probability of the robot being located in each grid cell. It suffers from needing to use a fine grid to accurately locate the robot which consumes immense computational resources. Using a coarse grid speeds up localization computation, but makes the result too imprecise to be used for updating the robots position accurately.

2.3 Particle Filters

To overcome the computational requirements of Markov localization, particle filters use a distributed set of particles to localize the robot. In Monte Carlo localization [3], [4], the particles are randomly distributed (see Fig. 1), and then selectively updated based on the probability that the particle position accurately reflects the given measurement. Low-probability states are rejected, and high-probability states are replicated. Successively, the set of particles becomes increasingly more representative of most likely position of the robot. Because fewer particles are needed to locate the robot as the knowledge of the position becomes more accurate, the Adaptive Monte Carlo Localization (AMCL) algorithm reduces the number of particles as the algorithm proceeds.

2.4 Comparison / Contrast

Of path tracking and localization, the first problem can be handled successfully with traditional Kalman filter, which is extemely fast computationally, while it fails in localizing the robot globally as it focuses on updating a single state. Multi-state updates, from which the most likely of considered options can be considered as the position of the robot is more or less necessary to globally locate the robot. Of the two approaches considered here, Markov localization is resource intensive because it considers all possible states in a grid of states at each update, and fails at the tracking problem if the grid size is too large. Monte Carlo localization

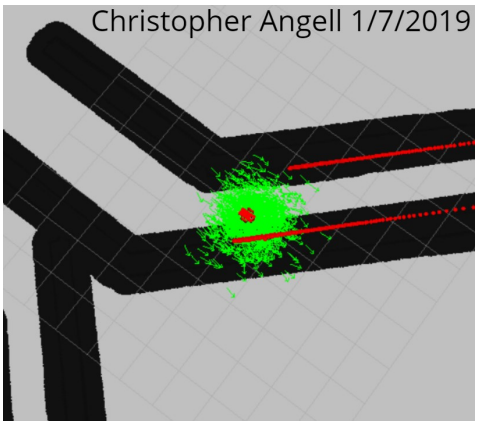

Fig. 1. Particle cloud from AMCL surrounding the robot.

is vastly more efficient as it uses a set of continuously distributed particles, sampling only where the robot is likely positioned. Because the particle positions are continuous it overcomes the limitations of grid size being able to be used also for tracking the robot. Thus it provides a rapid way to locate the robot globally, and then track its position. The AMCL algorithm then provides an even more efficient way to update the position of the robot as the set of particles is trimmed in size as fewer particles are needed to locate the robot when the position is precisely known. In summary, Kalman filter quickly updates a robots position, but for an efficient algorithm to locate the robot globally, Monte Carlo localization should be used.

3 Results of Simulations

The robot simulations were based on the Gazebo package in ROS. This allowed for realistic physics simulations of the robot along with simulation of sensor inputs and odometry, as well as the environment. Two different robots were modeled, a benchmark and a personal model, both described below. The robots will be introduced along with achievements while parameters used to tune the simulations will be discussed in Section 4.

3.1 Achievements

In this project, we achieved the simulation of a robot in a realistic environment demonstrating the use localization and path planning packages in ROS. The robot was able to locate accurately to a new position within a few minutes.

3.2 Benchmark Model

The benchmark model was based on the description given in the project. The purpose was to demonstrate the capabilities of ROS navigation stack, and provide a platform for testing the localization routines.

3.2.1 Model design

The robot was box design with dimensions of 0.4 m long, by 0.2 m wide by 0.1 m tall, and had two 0.1 m radius wheels, one at each side at the center of the robot 0.15 m from center (see Fig. 2). Two casters were placed under the robot

Fig. 2. The benchmark robot.

at 0.15 m and -0.15m. The two wheel design with casters allowed for a differential drive to be used. Two sensors were included, a camera placed at the front of the robot, and a Hokuyo laser range finder placed near the front on top. Only the range finder was used for localizing the robot.

3.2.2 Packages Used

The ROS nodes and messages used are shown in Fig. 3. This robot used the `joint_state_publisher` node to publish joint states to the `robot_state_publisher`, which would then send transforms to AMCL module and the `move_base`. The `map_server` node was then launched which sends map information to the `move_base` node. The gazebo module would send odometry information back to `move_base`, and laser scan information to the AMCL node.

In gazebo, the `differential_drive_controller` plugin was used for the drive controller, along with the `camera_controller` plugin for the camera, as well as `gazebo_ros_head_hokuyo_controller` plugin for the laser scanner.

3.3 Personal Model

A personal robot model was developed based on adding two additional wheels to the stock design and removing the casters. In addition to this design, a tracked robot was attempted off of a SDF design [5], but it was proved to difficult to convert the SDF file format to the URDF format necessary for simulation. Because of this the four wheel design was adopted.

3.3.1 Model design

The casters were removed, and the two wheels were moved forward to 0.15 m, and two additional wheels were added at -0.15 m in the y direction (see Fig. 4). The addition of two wheels meant that the differential drive had to be abandoned, and a `skid_steer_drive_controller` plugin in Gazebo was used in stead of the differential drive. The two wheels in the forward direction were used as the drive wheels.

3.4 Results

The benchmark robot was able to localize its position using the AMCL package, and then find its way to the goal position in a matter of minutes. The localization particles rapidly

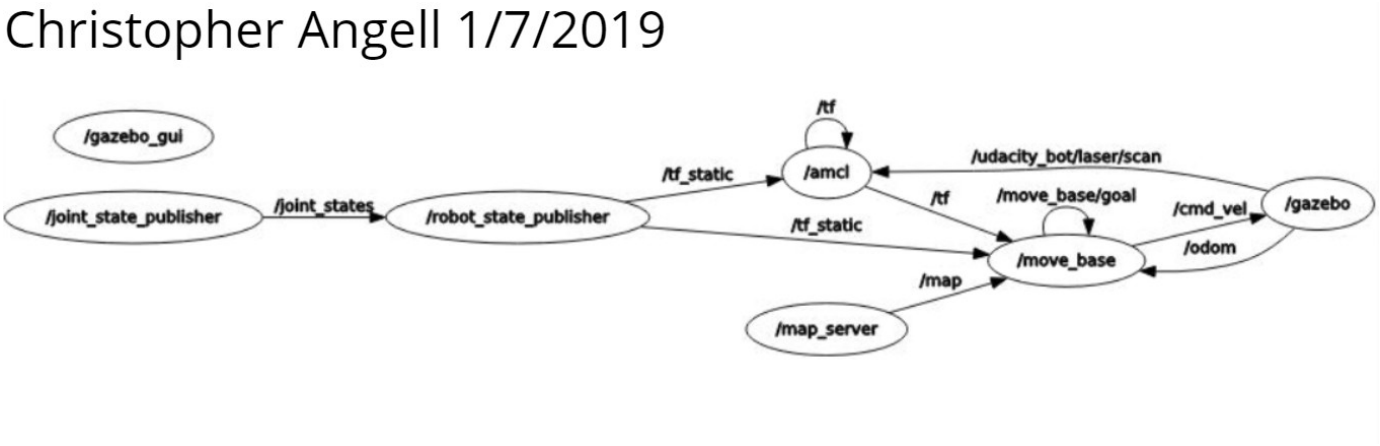

Fig. 3. ROS nodes and topics.

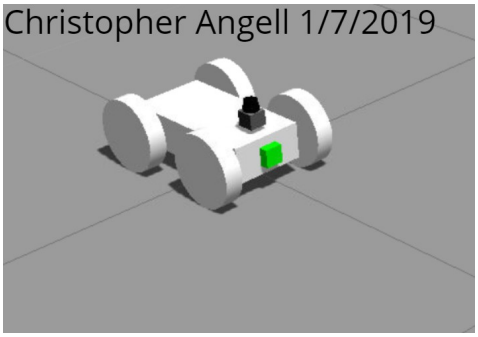

Fig. 4. The personal model robot. The wheels do not turn so a skid steering mechanism was used.

Fig. 5. Final goal state of the benchmark robot. The AMCL particle cloud is barely visible underneath the robot.

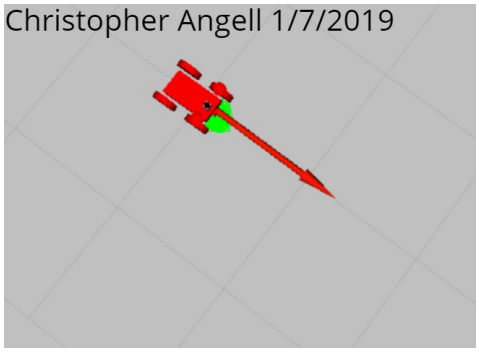

Fig. 6. Final goal state of the personal model robot. The AMCL particle cloud is barely visible underneath the robot. The large red arrow is the target final goal state for the robot. The robot has succesffully arrived at the goal state.

converged within seconds to the position of the robot (see Figs. 5 and 6). The principle issue with convergence of the AMCL algorithm is the sensor uncertainty. Given the high precision of the Hokuyo laser range finder, little ambiguity was left for the robot position.

The path planning algorithm was more problematic. When it got close to high cost regions in the cost map, the robot would get stuck and implement avoidance behavior such as turning around and backing up. Sometimes it would bounce back and forth between being stuck in different parts of the high cost regions because it would extricate itself perpendicularly from the high cost region, not necessarily in the direction of the calculated path to goal, which would lead straight to another wall. Occasionally, because of this the robot would completely fail to reach its stated goal.

3.5 Technical Comparison

The principle difference in the benchmark robot and the personally implemented robot was the number of wheels. Because the personal robot used skid steering with four wheels, it was less maneuverable than the differential drive benchmark robot resulting it getting stuck more often along the corridor in the map, and it needing more space to turn. As for localization, because the robots used identical sensors in identical locations, the behavior was identical.

4 Model Configuration

The robot model was configured using options in the launch files where the nodes are launched, and in yaml files which are called from launch files. Configuration was split into that for the AMCL module and that for the `move_base` module. Due to the sheer number of parameters, it was difficult to isolate which parameters were causing which behavior. After spending several hours adjusting parameters, and asking questions in the class channel, the work of previous students on Github was referred to [6] and specific parameters affecting performance issues were isolated.

TABLE 1
The AMCL parameters that were tuned.

Parameter name	value
min_particles	100
max_particles	1000
update_min_d	0.1
update_min_a	0.25
odom_model_type	diff-corrected
odom_alpha1	0.0001
odom_alpha2	0.0001
odom_alpha3	0.0001
odom_alpha4	0.0001

Christopher Angell 2019/01/07

TABLE 2
The move_base parameters that were tuned. If only one parameter was used, it was common to both the local and global cost maps.

Parameter name	Global	Local
update_frequency	3.0	5.0
publish_frequency	3.0	5.0
controller_frequency	5.0	
width	40.0	5.0
height	40.0	5.0
resolution	0.05	0.05
transform_tolerance	1.0	
robot_radius	0.5	
inflation_radius	0.4	
TrajectoryPlannerROS Parameters		
pdist_scale	0.8	
gdist_scale	0.8	
yaw_goal_tolerance	0.05	
xy_goal_tolerance	0.10	

Christopher Angell 2019/01/07

4.1 AMCL parameters

The AMCL module controls how the cloud of Monte Carlo particles are distributed and updated. The short list of parameters used are summarized in Table 1. Parameters that weren't adjusted or did not impact performance are not listed.

- min_particles and max_particles affect the number of particles are used in the AMCL algorithm. Increasing the values would decrease performance, and decreasing the values would increase performance, with max_particles affecting the initial distribution, and min_particles the distribution after enough time has passed to accurately locate the robot.
- update_min_d and update_min_a determine how far the robot must move in distance or turn in angle, respectively, before the particle cloud is updated. Increasing these value will improve performance as fewer updates would be done per unit movement, and decreasing these values would decrease performance.
- odom_model_type and the odom_alpha values affect how the particle cloud is updated. odom_alpha1 and odom_alpha2 specify the noise in the odometry's rotation update from the rotational and translational motion of the robot, while odom_alpha3 and odom_alpha4 are the respective noise in the odometry's translation update. These parameters determine how rapidly the particle cloud converges on the position of the robot, with small values converging rapidly, and large values converging slowly, or never converging.

4.2 move_base parameters

The move_base parameters affect the path planning of the robot both locally and globally. Some parameters affected both scales while others were only tuned for one of the scales. Some parameters further affected how often the movements were updated. Other parameters were for the base local planner. A list of pertinent parameters is given in Table 2

- update_frequency, publish_frequency, and controller_frequency all affect different aspects of how frequently different components of the move_base module works. These parameters were adjusted until error messages concerning certain rates not being met disappeared.
- width, height and resolution affect the map that move_base worked with to calculate a path. For the local cost map, it turned out tuning the height and width to a small value was essential to get the robot to move correctly.
- transform_tolerance specifies the delay time in the transform that is tolerable in seconds.
- robot_radius specifies the radius of the robot, and inflation_radius specifies by how much objects should be inflated by to create the cost map. This way the robot is guaranteed to avoid objects as the center of the robot is compelled to avoid obstacles by some margin.
- pdist_scale is the weighting for how much the robot should stay on the path it is given, while gdist_scale is the weighting for how much the robot should strive to reach its local goal (as opposed to the path to the global goal).
- yaw_goal_tolerance and xy_goal_tolerance specify how closely the robot has to get to the goal position for robot yaw and position respectively before the goal is considered accomplished.

5 Discussion

Performance of the system largely depended on a small number of parameters that varied little between the robots. Performance can be split between that for the particle cloud localization (AMCL), and that for the path planning.

Performance for particle cloud localization was dominated by the sensitivity of the odom_alpha parameters. Initially, these parameters were set too high, around 1.0 to 5.0 meters, and the particle cloud failed to converge at all. By lowering these values to something close to or less than the actual sensitivity of the laser range finder, the particle cloud rapidly converged to the position of the robot.

Performance of the path planning was affected significantly by the map size. Initially, the same size map for the

global and local cost maps was used, but the robot would wander aimlessly, and even go in the wrong direction. By lowering the size of the local cost map (`height` and `width`) which affects the size of the considered region for local cost planning, reasonable trajectories were planned that the robot could follow to reach the goal.

The two robots performed similarly with the differential drive robot having a tighter turning radius. Because of the larger turning radius of the skid steering robot, the `inflation_radius` parameter was tuned smaller for that robot so that the robot could approach objects closer as it completed wider turns.

Here, the localization solution adopted could be readily adapted to solve the 'kidnapped robot' problem where the robot is suddenly moved from one location to another with out warning. This could be done by having the particle cloud de-focus and cover the entire map. Then the cloud would be updated on a likelihood basis of where the robot could be based on the range finder data.

The AMCL/MCL algorithm is limited in that it requires an accurate pre-existing map of the environment to which the laser range finder data can be compared to. Building that map could be problematic. Here, we circumvented that problem by using a simulator which uses the same map to generate the laser range finder data as is compared to for calculating the probabilities for the particle cloud. Disparities between the internal model for the world and the external reality could lead to conflicts between where the robot believes it is located and to where it actually is. These conflicts could be resolved by using Simultaneous Localization and Mapping (SLAM).

AMCL/MCL would be readily deployable in an industrial setting where the environment doesn't change compared to the reference map. When obstacles are moving within the setting, errors in localization could occur as they wouldn't be accurately represented in the internal map.

6 Conclusion / Future work

In summary, two robots with different types of steering mechanisms were able to be located accurately and guided correctly to a final destination. The steering mechanism differences proved to not be such a sizable problem with the only correction needed was for the `inflation_radius` to be adjusted to accommodate differences in turning radius between the robots. While this project demonstrated the basics of AMCL with the built-in ROS path planning algorithms, it could be improved in many ways, as well as deployed on hardware.

6.1 Modifications for Improvement

Not explored here were further modifications to the robot physical dimensions. The dimensions affect how the robot turns in the environment and how it locates itself via sensors. A sharp turning radius enables precise positioning, so decreasing the physical size of the robot could improve performance on following planned trajectories. Because the localization depends only on the precision of determining distance of the robot to objects, sensor placement should not significantly affect performance. Nevertheless, because a simulation was developed different robot designs could be readily tested.

The biggest problem in performance was the ability of the robot to follow a planned path. Further work on tuning the parameters of the cost map to improve following behavior could be attempted, beyond the parameters mentioned above. Considering the sheer number of parameters, there wasn't time to test all of them, but hopefully further tuning could yield marginal improvement.

Looking further, different implementations of localization algorithms and path planning software could be tested to find gross differences in performance.

6.2 Hardware Deployment

Because of the simulation developed here, hardware deployment could be done readily by first identifying hardware components (chassis, wheels, motors, sensors, etc.) and then tuning the simulation parameters to match the hardware specification. Because the ROS stack is used, once a physical robot is built, the control software for the robot would need only implement the messages and nodes that gazebo simulates, being the odometry of the robot and the laser range finder data. Problematic however is building an accurate map of the environment that the robot lives in. Significant effort would be needed in all areas of identifying parts, assembling the robot, writing the control software, and building the map. But the main areas of the navigation stack could be fully vetted through simulation before work even begins.

References

[1] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[2] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IROS*, 2004.

[3] D. Fox, W. Burgardy, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," in *AAAI*, 1999.

[4] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *ICRA*, 1999.

[5] A. Synodinos, "progtologist/gazebo-tracks." https://github.com/progtologist/gazebo-tracks, 2018. Accessed: 2019-01-08.

[6] C. Dhingra, "chesh27 / robond-localization-project." https://github.com/chesh27/RoboND-Localization-Project, 2018. Accessed: 2019-01-08.