Where Am I – An implementation of adaptive Monte Carlo localization using ROS

Yisi Zhang

Abstract—In this project, two designed robots use adaptive Monte Carlo localization algorithm (AMCL) to localize themselves in a predefined map. The robots manage to align with the target vector within reasonable time. The particle filters converge relatively fast. Relevant parameters in the ROS amcl package and navigation package are tuned to achieve the task.

Index Terms—AMCL, Robot, ROS.

1 Introduction

Monte Carlo localization (MCL) uses a particle filter for robots to localize. MCL does not assume the belief of the robot is Gaussian, like the Kalman filter and its varients, and it can deal with the situations where the belief is multimodal. Compared to other non-parametric methods, such as the grid-based localization, MCL has better accuracy as the states are not discretized. Another advantage of MCL is that it can localize globally, which is not possible for the Kalman filter. This method thus also prevents the robot from being kidnapped to a random position. Adaptive Monte Carlo localization (AMCL) dynamically adjusts the number of particles as the robot navigates in the map. It reduces the number of particles as the pose of the robot is more certain and thus is computationally more efficient.

In this project, we applied the ROS AMCL package to localize a custom robot in a 2D environment in the Gazebo and RViz simulation. A variety of parameters were tuned to optimize the performance of navigation and localization.

2 Background

Both AMCL and Kalman filter have been applied to solve the localization problem, in which the robot navigates in an environment with noisy measurements from the sensors and it tries to estimate its accurate position based on that information. The AMCL has a number of advantages in performing localization compared to the Kalman filter and its variants. We discuss in the follows their differences.

2.1 Kalman Filters

The Kalman filter is an algorithm that tracks the internal state of linear systems in noisy environment. It has two steps, the prediction step and the update step. In the first step, a prediction of the state of the current time step is made based on the previous state and the dynamical model of the system. In the update step, the current observation is taken into account and the posterior estimate is a weighted average of the prediction and the measurement implied state. The weights are determined by the noises of the process and the observation. This algorithm assumes that the noises are Gaussian.

In a lot of cases, the system is not linear and linear Kalman filter is not applicable. The extended Kalman filter (EKF) was developed to deal with those situations. In the EKF, the equations are linearized by computing the Jacobian matrix of the equations expanded locally at the previous state. The process and observations noises are also assumed to be Gaussian.

2.2 Particle Filters

The particle filters adopt Monte Carlo techniques to estimate the state of the system. The particle filter has two steps, a prediction step and a resampling step. In the first step, the algorithm generates a motion update of the particles based on the previous belief and then the corresponding weights of the particles are calculated based on the measurement. In the second step, the posterior distribution of the belief is approximated using importance sampling. The weights of the particles are normalized so they add up to 1. Then, the particles are randomly drawn with replacement based on their weights. The new set of particles represents the posterior distribution of the belief. The particle filter does not assume the noises to be Gaussian.

2.3 Comparison / Contrast

The linear Kalman filter is optimal for linear system with Gaussian noise; and in cases of nonlinear systems, the EKF would be the choice. In general, the Kalman filter methods are more computationally efficient than the particle filters. However, in many situations the Gaussian noise assumption is not satisfied, and the particle filters deal with this issue due to its nonparametric nature. The posterior estimate of Kalman filters is Gaussian; the posterior estimate of particle filters, however, is represented by the distribution of particles and thus can be arbitrary. When using for localization, Kalman filters are good for tracking the robot position, while particle filters can localize both locally and globally as it updates the particles of the entire map. Particle filters can also prevent the robot from being kidnapped to a random location.

3 SIMULATIONS

We designed a robot using Gazebo. The robot has a disk-shaped base and two wheels. To stabilize the robot, there are also two caster wheels in the front and in the back. On top of the base, there is a camera facing the front and a laser sensor at the center of the vehicle (Fig. 1). The laser is raised to avoid reflections from the robot itself. To launch the robot, two nodes are included in the launch file: one publishes joint state messages for the wheels using the `joint_state_publisher` package, one publishes the robot's state to tf (transform tree) using the `robot_state_publisher`. The robot is spawned in a Gazebo world, as well as a predefined map. The ROS amcl package is integrated with the robot to localize it in the map. The parameters of the amcl package are tuned. The `move_base` package is applied for the robot to navigate in the map. It utilizes global and local costmaps to find a continuous path for the robot.

The costmap parameters are crucial for the robot to move around in the map. The `obstacle_range` parameter defines the maximum range of obstacles for the robot to consider. It is set to be 2. The `raytrace_range` parameter determines the space in front of it the robot attempts to clear out and is set to 1.5. The `robot_radius` is important for the robot to evaluate whether it can pass. It is set to be the radius of the robot base plus the width of the wheel, which is 0.2. The `inflation_radius` is the inflated distance from the actual obstacle and is set to 0.35. A relatively large inflation is helpful for the robot to avoid hitting the walls. For the control loop to have enough time to update to publish the command, the `update_frequency` and the `publish_frequency` in both local and global costmap setting are reduced to 2.5Hz. In addition, the size of the local costmap is reduced to 5x5. In the base_local_planner_params.yaml file, the `controller_frequency` is rediced to 10Hz (thus the sim_time is 0.1s).

The amcl parameters are important for the particles to converge. We found that the `odom_alpha1-4` parameters are crucial. The `odom_alpha1-2` are reduced to 0.001 and the `odom_alpha3-4` are set to 0.01. The number of particles is set to between 100 and 1000. The maximum laser range is increased to 50 and the maximum number of beams is increased to 60.

The launch files were combined in a single main.launch file.

3.1 Achievements

Both the original model and the modified model reached target within reasonable time. The particle cloud reduced in size as the robot moved towards the target (Fig2).

3.2 Model design

3.2.1 Benchmark model

The benchmark model is comprised of a box-shaped base (0.4x0.2x0.1). There are two wheels with 0.1m radius on the two sides of the base. In addition, there are two caster wheels to stabilize the robot. Each of the caster wheel is a sphere with radius 0.05m. The radius of the front caster wheel was reduced by 0.01m to reduce friction so as to increase velocity. The camera is placed on the front side of the base and the laser sensor is placed on the top of the base close to the front edge. To improve the laser sensing, the laser is raised higher above the base.

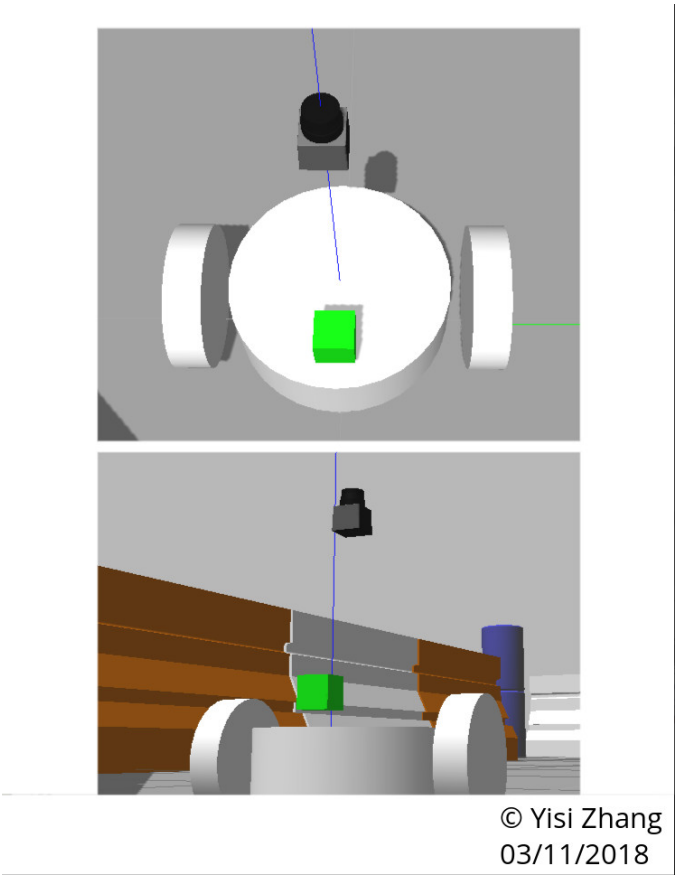

Fig. 1. Robot design

3.2.2 Personal model

The modified model base is a cylinder with 0.15m radius and 0.1m height. Two wheels, 0.1m radius and 0.05m width, are placed on both sides of the robot. Two caster wheels are placed in the front and in the back of the base bottom. The front caster wheel is a sphere of 0.049m radius, and the back caster wheel has a radius of 0.05m. A camera is placed on top of the base towards the front and facing the front. A laser sensor is placed at the center of the robot.

3.3 Packages Used

The `joint_state_publisher` and `robot_state_publisher` packages are used to receive and publish the topics. The ROS amcl package was used for localization and the `move_base` package was used for navigation.

3.4 Parameters

The navigation and amcl parameters for the personal model are listed in Table 1-5.

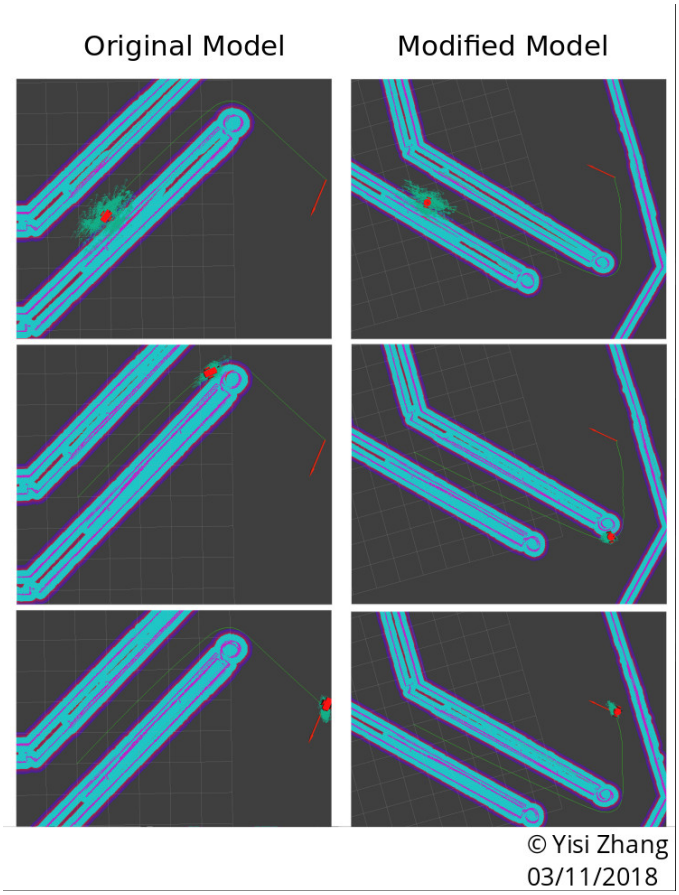

Fig. 2. Localization performance

TABLE 1
base_local_planner_params.yaml

holonomic_robot	false
controller_frequency	10.0

TABLE 2
costmap_common_params.yaml

obstacle_range	2
raytrace_range	1.5
transform_tolerance	0.25
robot_radius	0.2
inflation_radius	0.35

TABLE 3
global_costmap_params.yaml

update_frequency	2.5
publish_frequency	2.5
width	40.0
height	40.0
resolution	0.05

4 Results

4.1 Localization Results

For both models, the robot reached target reliably. The position and orientation of the robot overlapped the target vector (Fig.2). It took approximately 15s for the particle filters to converge. The travel velocity was around 0.2-0.3m/s and for the targets set manually around 8-10m away from the robot, it took 30-40s for the robot to reach the target.

The robot performed worse when moving around the poles and the corner of walls. It sometimes got stuck at those locations and turned around to move out of the stuck spot. This was improved by increasing the inflation radius and adjusting the robot radius close to half the width of the robot.

4.2 Technical Comparison

The width of the personal robot is 0.1m wider than the benchmark model. Thus, the robot radius was set differently for these two models. Overall, they performed similarly. The modified model performed slightly better than the benchmark model as the round shape is less likely to get stuck. The particle filter convergence and time to reach the target was comparable in these two cases.

5 Discussion

In this project, the robot successfully reached the target by localizing itself in the map. The AMCL method was useful to move the robot out of kidnapping locations. The robot is more likely to get trapped at the notches between obstacles as the sensor mistakes it for a gap to pass. With AMCL, the robot attempts to move out of the location to reach the final goal.

The MCL/AMCL can be widely used in the navigation and localization of autonomous vehicles such as self-driving car. It is particularly useful in the navigation in an urban environment as the map is usually well defined. It is also useful for the localization in a living area such as in the robotic cleaning system.

6 Conclusion / Future work

In this project, the robot navigates in the map to reach the target. It localizes itself within reasonable time and with good precision using AMCL method, implemented by the ROS amcl package. The AMCL method is powerful when navigating in a complex environment to achieve global localization and to release from kidnapping.

For robots using sensors to navigate, it continuously processes the sensor information to estimate the robot's location. The estimate is usually noisy because of the noise from the measurement and the number of particle used for estimation. Increasing the number of laser beams and scan range can increase the accuracy of measurement, however, it also increases processing time. A greater number of particles also increases precision, but it increases the computational load as well. Thus, there is a trade-off between accuracy and processing time. For this task in which the velocity of the robot is relatively slow, the processing time on the order of 100ms is sufficient for the robot to get updated information of its current location and there is room for improvement in accuracy. However, in the situations where the robot needs to travel faster, the processing time would be a limit factor.

TABLE 4
local_costmap_params.yaml

update_frequency	2.5
publish_frequency	2.5
width	5.0
height	5.0
resolution	0.05

TABLE 5
amcl.launch

transform_tolerance	0.25
odom_alpha1	0.001
odom_alpha2	0.001
odom_alpha3	0.01
odom_alpha4	0.01
min_particles	100
max_particles	1000
laser_max_range	50
laser_max_beams	60

In the future, a better designed robot, such as a four-wheel vehicle can improve the smoothness of the trip. Some parameters, such as the obstacle range, resolution and the inflation radius can be adjusted to reduce the false detection of space. The transform tolerance and the odom alphas can be further tuned to improve the convergence of particle filter. Additional algorithms may be implemented to better plan out the path so the robot does not hit the walls. The algorithm can also take the image input from the camera into account to more precisely localize the robot. However, image processing takes longer time and it may not allow the robot to travel fast. It can be deployed on a mobile robot with sensors and actuators implemented through processors like the NVIDIA Jetson boards.