Robotic Localization Project

Chuong Dao
riochuong@gmail.com

Abstract

Localization is the challenge for a mobile robot to determine its pose in a mapped environment. Various probabilistic localization algorithms have been invented to optimize robot motions such as the classic Extended Kalman Filter (EKF) which is a special case of Markov Localization. One of the most popular and easy to implement localization algorithm in recent years is Monte Carlo Localization (MCL) which has proved to work well with both global localization and local localization. In order for robot to navigate well in a specific environment, there are some fine tuning works have to be done on both MCL and robot related parameters to guarantee good performance. In this presentation, there are two localization experiments related to applying Adaptive Monte Carlo Localization (AMCL) and adjusting Robot Operating System (ROS) navigation stack parameters to two different robot models in order for each robot to be able to navigate itself through a same given environment to a same predefined goal pose.

Introduction

There are two experiments related to robot navigation illustrated in this paper. First, hte provided robot model from Udacity Robotic need to be fine tuned with parameters related to ROS navigation stack in order for the robot to navigate itself through a known environment provided to a predefined goal using the Adaptive Monte Carlo Localization (AMCL) algorithm. Second experiment describes a whole process of building a new robot model with substantial design differences from the provided one and also adjust its parameter to help it successfully navigate through the same given map to the same predefined goal with AMCL guidance.

Background

A mapped environment is described with global coordinate system which is not relevant to robot pose. Through localization process, mobile robot can correlate its current pose to the global map system using the data acquiring from different kind of sensors like: odometry sensor, laser sensor, lidar, 3D depth camera. However, all these sensors measurements have some certain level of noise that requires to fine tune the robot parameters related to navigation module to be able to use multiple data points overtime to have a good estimate of robot pose. One of the classic localization algorithm is Extended Kalman Filter (EKF) which has two important changes related to the original Kalman Filter (KF) algorithm[1]. First, EFK uses nonlinear generalizations instead of linear relations in KF. Second, EFK also applies Jacobian matrices instead of linear system matrices. EFK assumes that the provided map is presented by a number of selected features. Therefore, it does not work well with map that have too many similar objects and also EKF cannot be applied to solve global localization problem which means robot will be placed initially at an unknown position in its world. MCL is a relatively new algorithm which is surprisingly easy to implement. It can be used to solve both local and global localization. MCL use particles

to represent the belief. These particles are randomly and uniformly spread over the whole robot pose. Continuous measurements from sensors trigger weight update for each particles, and only important particles with high weights will survive the selection process. As a result, MCL can be used to approximate any distribution which is not the case for EKF.

Results

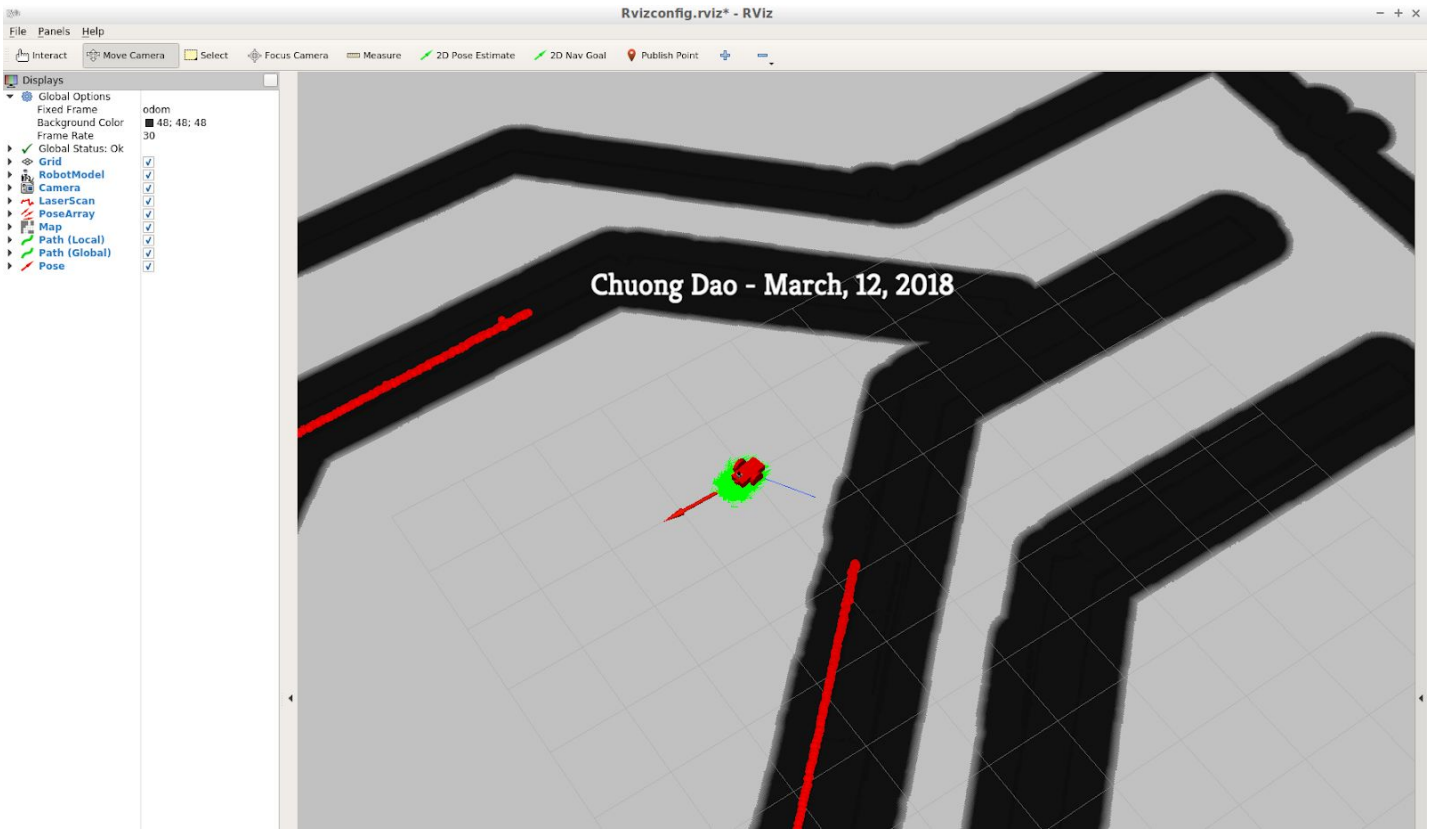

Figure 1: Provided Robot at goal position

Figure 2: New design robot at goal position

Model Configurations

1. **Provided Robot Model from Udacity**

   Costmap_common_params.yaml:

   transform_tolerance = 0.5 (s) - This helps to accomodate for the delay in data moving between odometry to navigation stack as the simulation is running on virtual machine on a slow computer.

   robot_radius = 0.4 (m) - This is the longest side of the robot chassis which can give good margin for movement.

   inflation_radius = 0.5 (m) - This help robot to have better movement at a safe distance with the wall.

   obstacle_range = 1.0 (m) - Keep this small as the robot largest side is only 0.4m. This would help save some computation time to update the cost map.

   raytrace_range = 1.0 (m) - Similar as above, robot only need to clear enough space right in front of it with new sensor readings.

   base_local_planner_params.yaml:

   max_vel_x = 4.0 (m/2) - the provided robot is moving too slow so increasing this to help itm move faster. However, this does not dramatically enhance the speed of the robot.

   max_vel_theta = 3.0 (rad/s) - this change allows robot to have higher angular velocity to make steep turn

escape_velocity = -0.5 (m) - This help robot to have better movement at a safe distance with the wall.

pdist_scale = 1.5 - Increasing this value help the robot local path stay closer to its global path to have higher probability of getting to goal

gdist_scale = 1.0 - Also increasing this value helps robot actually escaping some repeated back and forth movements due to **pdist_scale's** influence especially at the turn.

amcl.launch

min_particles = 10 - Reduce number of min particles so the pose array would not be to scattered out of the robot pose when the localization belief of the system is high.

max_particles = 2000 - Keep this number not too high to avoid high computation requirement.

odom_alpha1, odom_alpha2, odom_alpha3, odom_alpha4 = 0.008 - through trials and errors in order to select the value that help the particles converge nicely within the robot pose.

2. New Robot Model:

Robot Design :

Robot design is inspired from the design of monster trucks with big wheels so it can overcome complex obstacles out in the nature. Therefore, the new robot has four wheels. The differential motor is placed between the front wheels. The Hokuyo laser sensor is placed 0.07 (m) from the center of the chassis toward the back. This new location will help the laser sensor has better coverage while the robot is in reversed mode. The RGB camera is still placed in front of the new robot.

Figure 3: Robot Design

Chassis's size:

Width = 0.35 (m), Height = 0.3(m), Thickness = 0.11(m)

Size of each wheel cylinder:

Length = 0.07(m) , Radius = 0.08(m)

In compare with the provided Robot from Udacity tuning parameters, belows are the differences in configuration of the navigation stack:

Costmap_common_params.yaml:

inflation_radius = 0.5 (m) - This help robot to have better movement at a safe distance with the wall.

obstacle_range = 3.0 (m) - Increase this value to help the new robot recognize obstacle earlier.

raytrace_range = 3.5 (m) - Increase this value so robot can help the robot to see more free space ahead to plan smoother turn. t

max_vel_x = 0.5 (m) - During the tuning process, setting the maximum velocity on the x direction too high can lead to poor performance as robot takes longer time to adjust to the correct motion.

max_vel_theta = 6.0 (rad/s) - Increase this value in order to allow robot to escape some steep turns toward the goal

escape_velocity = -2.0 (m/s) - Increase this value in order to help robot to unstuck faster.

pdist_scale = 0.5 - Decreasing this to avoid robot trying to stick too close the global path at the turn and lead to robot stuck moving back and forth without making any progress toward goal.

yaw_goal_tolerance = 0.5 (rad) - Increasing this in order to help robot reach goal a little faster as observing robot was struggling to find the correct orientation while spinning around the goal location for a long time.

xy_goal_tolerance = 0.2(m) - Similar as above, this increment is to help robot to reach goal faster.

The remaining parameters are similar for both robots models.

Discussion

Both of the robot models are able to navigate to the same predefined goal. However, both robot models are subjected to poor performance due to imperfect fine tuning. For example, the robots take quite sometimes to adjust to the correct path and keep moving in to the obstacle direction until it recognized the blocking space and turn around. Also, both robot perform poorly at the U-turn as they both fell off the global path rather than making turn and stick to it. This is also one of the reason that the robots take quite sometimes to recover and follow the new global path.

The AMCL algorithm performs decent in both cases based on the particle array from **Figure 1** and **Figure 2**. Using ROS AMCL package really speeds up the time for implementing robot localization feature. There are still a lot of room for fine tuning improvements to optimize robot performance. AMCL also has some successes in solving the robot kidnapped problem which is often used to verify robot ability to autonomously recover from fatal localization failure[2]. Indeed, the AMCL is able to generate consistent particle representation based on sensor readings. By adjusting the MCL sampling method to use mixture proposal distribution [2], MCL is able to recover quickly from a kidnapped situation. As a result, the MCL family of algorithms can be applied to robot using in big warehouses or large grocery stores. With a predefined map information of the location, robot can easily apply MCL algorithm to navigate around the known environment for task like restocking or curbside pick-up.

Conclusion and Future Works

Although both of the robots in this paper are able to navigate to the predefined goal, observing the whole navigation process points out some weaknesses of the whole navigation stack. The robots have some hard time to select the trajectory path and sometimes stuck due to no trajectory with good scores found. Increasing the number angular samples and reduce the angular granularity can help to improve at the cost of higher processing time. Also, increasing the local map size will also improve the path planning process at the cost of more computation. It is also tricky if the robot can go in the reversed direction, because the laser sensor might not be able to fully capture the blind spot behind the robot. It can lead to robot is totally stuck colliding with obstacle but the local planner still thinks there are more room behind it for backing up. Obviously, adding one more laser sensor can obviously helps in this case or we can use a lidar spinning on top of the robots to obtain even more information at longer range of the environment. On the other hand, trying different local planner like Dynamic Window Approach (DWA) would be a good experiment to see if the performance can be improved. Also, dynamic reconfigure, which applies a specific set of local planner parameters to a specific area of the map, can help to optimize planning performance also.

References

[1] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Cambridge, Mass: The MIT Press, 2006. Print.

[2] Dieter Fox, Wolfram Burgard, Sebastian Thrun, Frank Deallert. *Particle Filters for Mobile Robot Localization.* In: Doucet A., de Freitas N., Gordon N. (eds) Sequential Monte Carlo Methods in Practice. Statistics for Engineering and Information Science. Springer, New York, NY