Where Am I?

Dale Cassidy

Abstract—Utilized ROS packages to accurately localize a mobile robot inside a provided map in the Gazebo and RViz simulation environments. Built a mobile robot to follow a path toward a goal in Gazebo using ROS packages AMCL and the Navigation Stack. Tuned specific parameters within the packages to achieve the best possible localization results.

Index Terms—Robot, Gazebo, RViz, Localization.

1 Introduction

Localization is the challenge of determining the robot's pose in a mapped environment. In this project, the robot is operating in simulated 2 dimensions and the pose consists of an x and y coordinate as well as a theta for orientation.

This problem is important because the robot must know where it is in a mapped environment before it can accurately navigate (without hitting obstacles for example) to achieve some goal in a different location within the mapped area.

There are different methods to localize a robot and some are better than others for various reasons. In this approach, Adaptive Monte Carlo Localization (AMCL) is used. AMCL is more effective than other methods such as using Kalman filters. AMCL due to it's use of particles can map any type of distribution. It does not have to be a Unimodal Gaussian distribution as is the case when using the Kalman filter for example.

Using AMCL and the Navigation Stack within the simulated environments of Gazebo and RViz, the plan was to navigate the robot to it's goal as quickly as possible.

2 Background

Localization is an important topic in Robotics since the robot needs to know where it is before it can figure out how to reach it's goal. There are 3 types of localization- positional, global and the kidnapped robot problem. Positional is where the robot's initial position and orientation are known. Global is where the robot's position is not known adding more uncertainty to the localization problem than with positional. And for the kidnapped robot problem the robot can be taken from any location at any time and moved to a new location adding much more uncertainty than in the global localization problem.

For this project, there was a choice between three filters- the Kalman filter, the Extended Kalman filter and Adaptive Monte Carlo Localization or Particle Filter Localization. The sections below describe each in detail and the reason the particle filter was chosen.

The Kalman filter is a filter used to estimate the value of a variable in realtime. It can take a lot of noisy sensor data and provide an accurate estimate of the value quickly. It's applied to linear systems where the output is proportional to the input.

The EKF is more applicable in robotics because it can work with nonlinear systems. Real world systems are often more nonlinear.

The Adaptive Monte Carlo Localization can represent any distribution not just Gaussian distributions. Monte Carlo Localization estimates the robot pose using particles. Each particle represents a guess as to the robot's location and orientation. AMCL can model a greater variety of environments because of this. You can also control the resolution of your solution by changing the number of particles generated. For this reason particle filters was chosen for this project.

3 Results

The following two images show the final results of both robots reaching the goal. Fig. 1 is the robot from the classroom and Fig. 2 is the custom developed one.

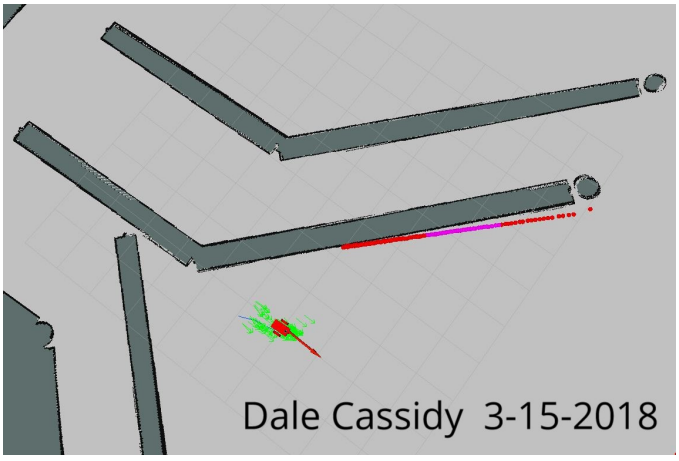

Fig. 1. Final Pose Array for Classroom Robot.

4 Model Configuration

The following is a description of the custom robot parameters in their own respective files:

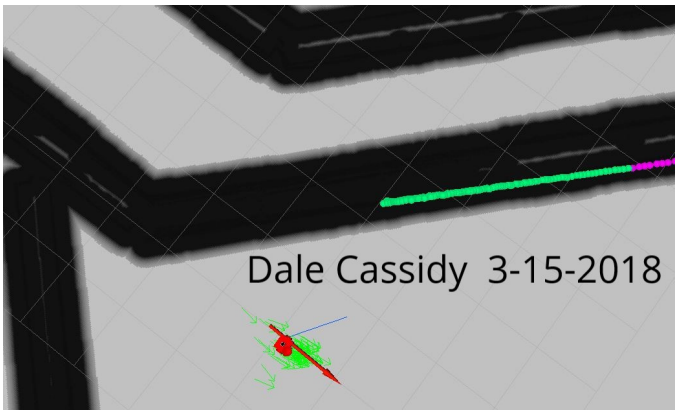

Fig. 2. Final Pose for Custom Robot.

4.1 amcl.launch

- min_particles: 50 and max_particles: 150- is the number of estimated states representing the robots pose. Controlling this range controlled the speed of system execution. Less particles meant the system could calculate the pose faster. This is discussed more under the Future Work section.
- odom_alpha'a: 0.025- this is how random noise was generated into the particles. Originally the numbers were set too large and made the location unnecessarily uncertain. This is discussed more under the Future Work section.
- controller_frequency: 5- originally number was set too fast and kept giving a missed control loop warning.

4.2 base_local_planner_params.yaml

- meter_scoring: true- the distance in the map was mapped in meters not cells
- sim_time: 2- gave more time for the robot to pick the lowest cost trajectory
- pdist_scale: 1 and gdist_scale: 5- are parameters that fed directly into cost function and were very difficult to tune. Setting the path distance scale much lower than the goal distance scale told the navigator to favor trajectories with lower distance to the navigation path
- yaw_goal_tolerance: 0.05 and xy_goal_tolerance: 0.1- these were doubled from the original value to give the robot an easier yaw and distance tolerance when reaching the goal
- heading_lookahead: 1- sets the distance the robot looks ahead when determining the lowest cost trajectory. Was set higher than the default to allow it to look ahead further and choose a better lower cost trajectory.

4.3 costmap_common_params.yaml

- obstacle_range: 2.5- robot would update the costmap with obstacles 2.5 meters or less from the sensors
- raytrace_range: 4- the distance in meters the robot tried to clear out in front of it
- transform_tolerance: 0.16- depended on the system. This represents the time that signal transforms are valid for. If the number is too big, the system uses old invalid sensor readings and if the number is too small, the navigation system will turn off.
- robot_radius: 0.2- the radius of the robot
- inflation_radius: 0.35- represents how close the center of the robot can get to an obstacle. If this number is too big, there is no navigable pathway in simulation.

4.4 global_costmap_params.yaml

- update_frequency: 1 and publish_frequency: 1- rate at which the robot updated the cost map. If too fast, the system would get missed map update warnings and the costmap would not be updated.

4.5 Custom Robot Specifications

For the custom robot, the base was made into a square .2 by .2 meters. The hakuyo laser was placed directly on the top in the center. The camera was placed at the center of the front side. Two casters of a small 0.02 radius were put on the bottom chassis one up front and one towards the back for stability. There are two wheels. One on the left side and one on the right with a reduced radius of 0.05 meters. Also the wheels were moved down under the line of site of the top part of the chassis so that the hakuyo laser's line of sight was not obstructed.

5 Discussion

The custom robot ran much better than the classroom robot. The robot moved towards the goal much faster than the classroom robot. One may believe the localization performance was much better because the object was smaller and the top of the wheels were lower than the top part of the chassis so that the hakuyo laser had a clear line of site all around it.

AMCL would work well for the kidnapped robot problem. Every time the robot is kidnapped, the particles would be spread out widely across the map and the maximum number of particles would be used before consolidating as more sensor data helped to localize the robot and it became more sure of its location.

AMCL could be used in industry wherever a robot may need to localize itself in a given situation. For example, a company such as Amazon could use AMCL to localize their robots in their warehouse as they are trying to find specific packages in the room. Also, common robots like the Roomba that vacuum the floor can use Adaptive Monte Carlo Localization to know where it's at in the room.

6 Future Work

Overall the custom robot did well. Most of the time it quickly went towards the goal. But there were a couple times it would get stuck at a corner or would turn around and go in the opposite direction. Given more time, adding more particles and adding a little more uncertain noise would probably prevent the robot from getting stuck. The

cost function parameters gdist and pdist could still be tuned better as well.

With a faster system, one could increase the number of particles and lower the transform tolerance and increase the map updates which would give better performance for the robot traveling to its goal.

There is also a Dynamic Window Approach model I would try if I had more time. From comments comparing it with the Navigation Stack, people seemed to think it navigated the robot better.

REFERENCES

[1] L. Lamport, *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley, 1994.