Udacity Localization Project

Abstract

This project explores local localization of a robot using the Adaptive Monte Carlo Localization algorithm. Given a map and a initial position, the robot should be able to quickly discover its pose and navigate to a given goal position. The map contains multiple obstacles and paths which the robot must consider while traveling to its goal. We also explore the effects of the multiple parameters required to tune both navigation and AMCL. Additionally, we will be trying different robot bases and sizes to gauge its effect on both algorithms.

Introduction

The goal of this project is to create and design a robot that can localize itself within a given environment. The map and initial position are given and the robot must accurately calculate its pose and orientation over time. There are 3 main types of localization: local, global, and global with kidnapped robot localization. In local localization, the map and initial pose are known so the main challenge is to incorporate uncertainty from sensors and movements. In global localization, only the map is known, so the uncertainty increases initially and the robot locks in on its true position over time. In the kidnapped robot localization, the robot can be moved to a random new location at any time. This requires the localization algorithm to be robust and flexible. The type of solution addressed in this project is local localization since initial pose and map are kept consistent.

Background

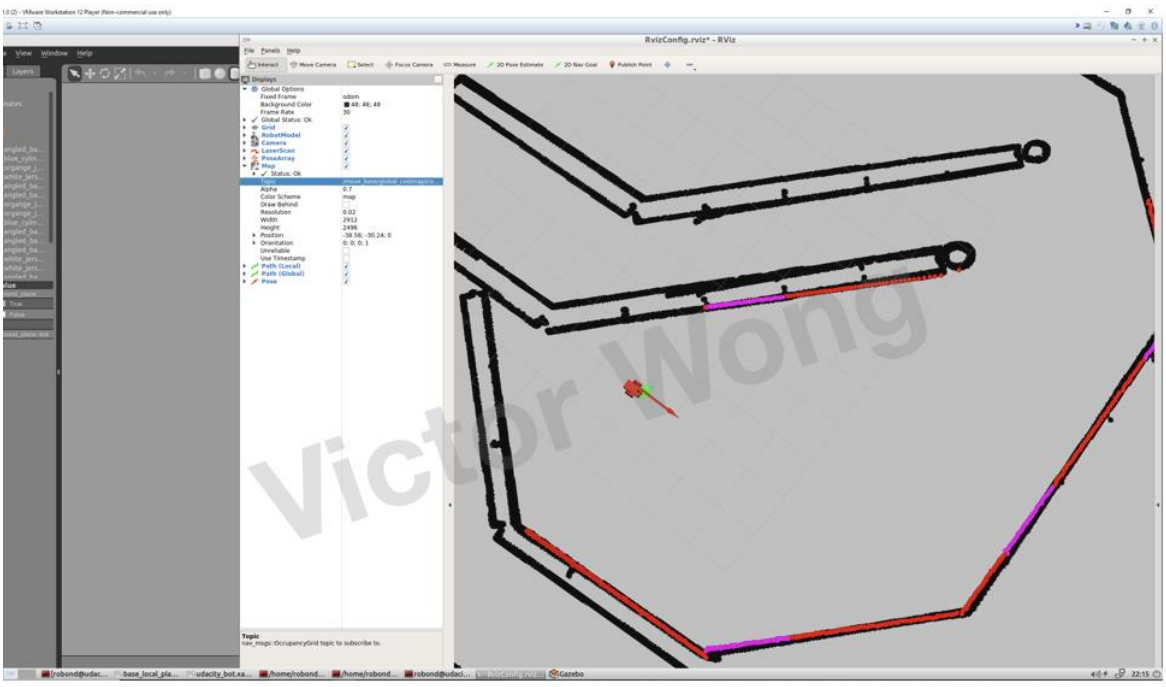
For most interesting tasks, the first question a robot must ask itself is "Where am I?" Herein lies the base of the decision tree for robots that are untethered and need to navigate around its environment. In order for a robot to perform the task required of it, it first needs to be at the right place. Some of the challenges for localization are memory and computational limitations on the size and granularity of the map, sensor noise, actuation noise, and dynamic environments. The 2 most common algorithms used for localization are the Kalman filter and the Monte Carlo Localization.

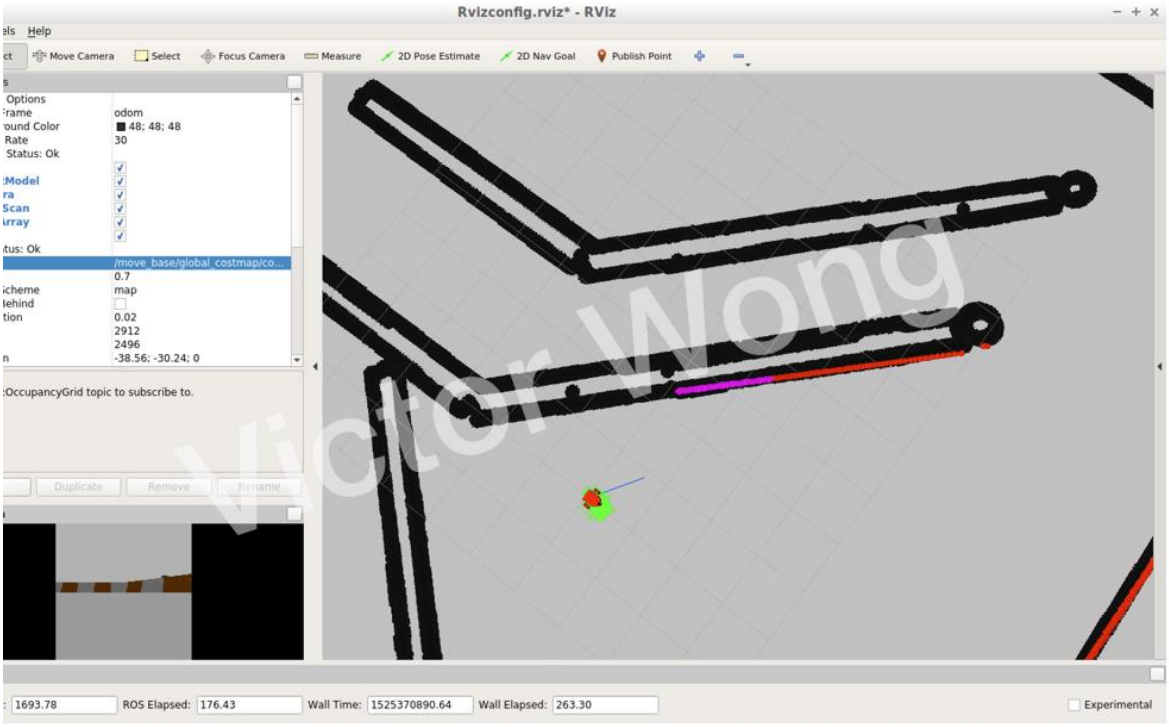
The Kalman filter is a very robust and quick localization algorithm that is ideal for linear systems. The Kalman filter excels in situations where both measurement and actuation are linear systems with Gaussian noise. It can very quickly ascertain in real time an accurate guess of pose and orientation with just a couple data points. The biggest limitation however, is that all

systems involved must have Gaussian noise. The EKF addresses this weakness by using the Taylor series to locally approximate non-linear systems. Also, this can easily be extended to track multiple variables take advantage of multiple sensor inputs.

The Monte Carlo Localization is a simple and flexible solution to more complex systems, nonlinear systems often found in the real world. It uses particle filters to approximate pose and orientation, converging to an accurate belief over time. The advantages of this algorithm lies mainly in its flexibility. Since each particle believes itself to have the correct state, it can use the same method for any problem and still approximate the most complex of systems by slowly weening out incorrect particles. Also, the computational and memory cost can easily be adjusted by tuning the number of particles. For its flexibility, we use Monte Carlo Localization in this project.

Results





As shown in the pictures above, both robots were able to reach the correct pose and orientation. Also, the particles for both have converged since they are densely grouped around the center of the robot. The custom robot, although more jittery, was able to reach its destination in a quicker time.

Model Configuration

After testing multiple geometries for the robot, there seemed to be an advantage in speed and robustness for smaller and rounder robots.Longer robots with wheels off-center tended to get stuck near the walls even with ample allowance on the robot_footprint parameter.

One of the first parameters to be adjusted were the min and max particles. The number was set pretty high relative to the system's capabilities which increased the time for each testing cycle. Reducing the number of particles not only brought down computation time, but also dramatically sped up the process for tuning the other parameters. The goal early on was to

reduce the computation time as much as possible, in order to quickly discern the parameters which had the biggest impact.

The next parameters to be tuned were transform_tolerance and controller_frequency. The navigation initially did not function properly because the control loop was too short compared to the computation time. Transform tolerance allows the computations to be valid for a bigger window, allowing for slower control loops to utilize its value. The controller frequency was also set too high for the system, so it had to be dialed down. The tradeoff was that having longer control loops would cause the robot to overshoot, without having the reaction to respond to upcoming turns or obstacles.

Robot radius and inflation radius were adjusted to give the robot some buffer between obstacles. The robot tended to navigate along the wall, eventually getting stuck. Increasing these parameters allowed to give a wider berth and not get stuck. Sim_time and p_dist_scale were increased to make the robot less jittery, allowing the local path planner to project a smoother route. Another motivation for increasing the p_dist_scale was to account for the big hairpin turn required for this particular map. The global path planning hugged the hairping extremely tight at the curve causing the robot to get stuck. By allowing the local path planning to deviate more from the global, the robot is able to clear the turn with a bigger buffer.

Discussion

Initially, the robot would take a long time to compute the movement step and tend to get stuck along the walls indefinitely, or backtrack for no apparent reason. No amount of parameter tuning seemed to help, even though these same parameters seemed to suffice the provided robot. It was eventually discovered that it was because the custom robot had two large wheels in the front which were obscuring the view of the laser scanner. Switching the wheels to a smaller size solved this problem.

AMCL, while great for the local localization problem, is not optimal for the kidnapped robot problem. The idea behind AMCL is that it the system starts at a very high uncertainty, and slowly converges to a reasonably accurate depiction of pose. The problem is that in order to converge over time, particles with incorrect poses slowly get pruned, leaving only the poses that are correct. Therefore, if the location was changed after the particles have converged, there would no longer be particles spread out to capture the pose in the new location. Although, with noise, the converged particles, could eventually land on the new location, this is pretty inefficient and unpredictable. In the real world, ACML is ideal for static environments

with noisy data. For example, applications can include robots that need to patrol and monitor an area like collecting rock samples on the moon over time.

Future Work

Since noise in the simulate environment is low, we can further reduce the number of particles decrease the computational time. Decreasing the noise parameters will allow the particles to converge faster, although these improvements are only beneficial in the simulated environment. Also, having a smaller bot means that it can squeeze through narrower paths, allowing us to decrease the resolution of the maps which can further push down the processing time. Additional sensors in the back could allow the robot to get a bigger picture of its surroundings, so that it doesn't have to turn around to gauge possible paths to the goal.