Where am I?

Jeremy Hale

Abstract—For the Udacity Robotics Nanodegree Where Am I? localization project, two different robot models were constructed in URDF and made to navigate a simple map. Both robots use a camera and a laser scanner as inputs and use differential drive for locomotion, however, their configuration is significantly different. The simulation was carried out using ROS, Gazebo, and RViz.

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

1 Introduction

THE project is an exercise in defining robots within ROS, but more importantly, an exercise is localization. Localization is a key topic in robotics and important for student understanding. Localization is the problem of determining where a subject is on a map. While there are several types of localization problems, this project deals with localization when the map is known but the location must be determined using sensors. To solve the localization problem a technique known as adaptive monte carlo localize will be emloyed. This technique is a particle filter that dynamically adjusts the number of particles in order to improve computation efficiency when the location is confidently known.

2 Background

As per the project instructions, Adaptive Monte Carlo Localization module in ROS was used as well as the ROS Navigation stack.

2.1 Kalman Filters

At a high-level, a Kalman filter takes multiple, related, noisy sensor readings and combines them into one estimation that has less randomness than any of the individual readings.

Linear Kalman filters are based on gaussian distributions, and their validity is limited to linear state transitions. Extended Kalman filters use a linear approximation (Taylor series) to allow non-linear functions.

2.2 Particle Filters

A particle filter consists of two stages: a transformation/measurement stage and a resampling stage. Initially, particles are randomly distributed. The particles are transformed based on some input (motion for example), then based on the measurement, particles with a high probability of having the same measurement have their weight increased. During the 2nd stage, resampling, particles with large weights are retained while those with low weights are discarded.

After several cycles, the particles converge on the estimated location.

2.3 Comparison / Contrast

MCL is easy to implement, is not restricted to Gaussian sensor noise, and is more flexible in terms of computational resources.

3 Simulations

Both the benchmark robot and the personal robot reached the navigation goals successfully using the same parameters.

The personal model is shown below. The robot is supposed to look like R2D2. Compared to the Udacity robot, the student robot is larger and tilted back at 30 degrees. The camera is no longer horizontal, however the laser scanner was kept horizontal.

Some inspiration was taken from the udacity xacro file as well as the ROS URDF tutorials (https://github.com/ros/urdf_tutorial/tree/master/urdf).

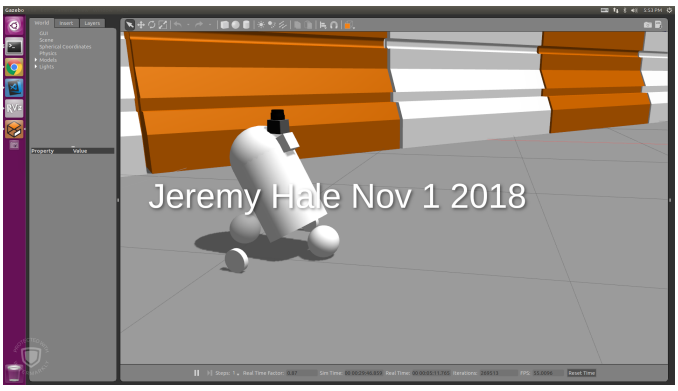

Fig. 1. My Robot Gazebo

The Udacity provided robot successfully navigates to the goal.

The student-defined robot successfully navigates to the goal.

The command to launch the student robot world is:

```
roslaunch udacity_bot my_world.launch
```

3.1 Achievements

Both models were able to navigate to the goal successfully. The benchmark model reached the goal in 66 seconds.

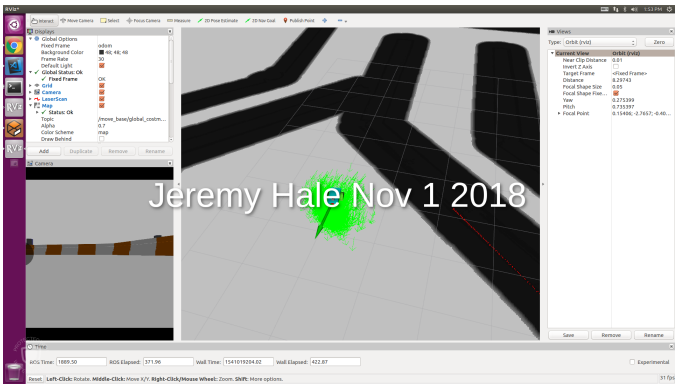

Fig. 2. Udacity Robot

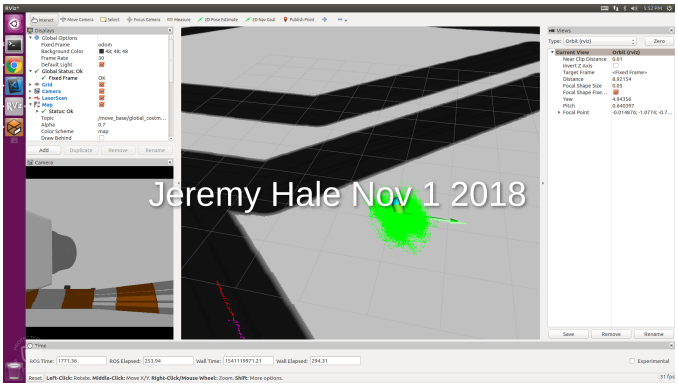

Fig. 3. My Robot

3.2 Benchmark Model

3.2.1 Model design

The benchmark model is a simple box with two driven wheels and two wheels for stability. The camera is front-mounted as is the laser scanner.

3.2.2 Packages Used

The packages used in the project should be specified as well as the topics received and published; the services it used and provided should also be addressed.

Packages:

- amcl
- joint_state_publisher
- move_base
- robot_state_publisher
- tf

Topics (brief):

- /amcl
- /joint_states
- /map
- /move_base/TrajectoryPlannerROS/cost_cloud
- /move_base/TrajectoryPlannerROS/global_plan
- /move_base/TrajectoryPlannerROS/local_plan
- /move_base/current_goal
- /move_base/global_costmap/costmap
- /move_base/goal
- /move_base/local_costmap/costmap

TABLE 1
AMCL Parameters

transform_tolerance	1.0
update_min_d	0.1
update_min_a	0.0872665
min_particles	20
max_particles	100
laser_min_range	-1
laser_max_range	-1
laser_max_beams	30
laser_z_hit	0.95
laser_z_rand	0.05
odom_alpha	0.2

TABLE 2
Common Costmap Parameters

obstacle_range	2.5
raytrace_range	3.0
transform_tolerance	0.2
inflation_radius	0.55

- /move_base_simple/goal
- /odom
- /particlecloud
- /tf
- /udacity_bot/camera1
- /udacity_bot/laser/scan

3.2.3 Parameters

For tuning both the costmap and the navigiation stack, the ROS documentation at the following URIs were used:

http://wiki.ros.org/costmap_2d

http://wiki.ros.org/navigation/Tutorials/RobotSetup

The tuning parameters were set to the documentation defaults as a starting point and then modified to increase performance.

The update rate for the local costmap was set to 40 Hz which is the same frequency as the laserscanner. The size of the map was significantly reduced (6x6) to limit computation time. The global map, which does not change, was set to update at 5 Hz. A local costmap smaller than 3x3 was found to be insufficient for navigation. In order to make the robot responsive when turning, the update_min_a was reduced to 5 degrees so it would perform filter updates more frequently.

The obstacle and raytrace range were set to the defaults of 2.5 and 3.0, respectively. As long as the raytrace range was set higher than the obstacle range, the robot was able to navigate.

The inflation radius was also set to 0.55 as per the example and proved to large enough that the robot would not become stuck on the wall but could still fit through narrow passages.

The number of particles was dramatically reduced as little as 20 particles could be used to successfully reach the navigation goal.

The transform tolerance was increased to allow delays in processing.

TABLE 3
Local Costmap Parameters

update_frequency	40
publish_frequency	40
width	6.0
height	6.0
resolution	0.05

TABLE 4
Global Costmap Parameters

update_frequency	5
publish_frequency	5
width	40
height	40
resolution	0.05

The parameters related to the laser scanner were left to defaults. Changing the number of beams did not make an observable difference to performance. Reducing the maximum range made localization worse. Adjusting laser_hit and laser_rand within a fairly wide range did not impact performance as well.

3.3 Personal Model

3.3.1 Model design

The student-defined model is an R2D2-type robot: 2 wheels drive the robot while 2 undriven wheels provide fore/aft stabilization. The camera is attached to the ”head” of the robot and the laser scanner is floating above the head in order to keep it horizontal.

3.3.2 Packages Used

Same as benchmark.

3.3.3 Parameters

Same as benchmark.

4 Results

The robot meets the performance objective as it is only to reach the goal. No time limit was suggested or imposed.

The particle filter converges within a few second for the benchmark. The personal robot has a subjectively slightly worse localization. This is believed to be due to the non-smooth, jerky motion of the robot. The simulator causes jerky motion when in the physical world this would not happen.

In either case, the robot reaches the goal in about 1 minute. There is a tendency for the robots to perform an ”Austin Powers” manoeuvre where it goes back and forward between the two walls of corridor instead of heading straight down.

4.1 Localization Results

4.1.1 Benchmark

The images above show the clustering of green arrows indicating localization result.

4.1.2 Student

The images above show the clustering of green arrows indicating localization result.

4.2 Technical Comparison

The simpler benchmark robot performs better because it has smoother motion. The personal robot bounces fore/aft as it moves causing jitter in the laser scanner readings.

The parameters for both robots are the same and their times to the goal are not dissimilar. The R2D2-configuration is mechanically quite different from the benchmark, but due to the limits of the simulator, a rear stabilizing wheel had to be added. This made the two configurations more similar.

5 Discussion

The personal robot performed adequately reaching the goal slightly slower than the benchmark.

5.1 Topics

- Which robot performed better? Benchmark
- Why DID it performed better? (opinion) The personal robot had a mechanical stability issues.
- How would you approach the ’Kidnapped Robot’ problem? AMCL would work well for the kidnapped robot problem. Once deposited in the new location, the particle cluster would spread out randomly until the robot is able to re-localize itself. This was not tested but is the author’s opinion.
- IN What types of scenarios could localization be performed? AMCL would work well in real life for situations where a well defined map exists.
- Where would you use MCL/AMCL in an industry domain? An industry example could be a warehouse. A warehouse is a fairly static well-defined environment that could be mapped. This technique would not work well for self-driving cars where the environment is constantly changing and large.

6 Conclusion / Future work

This project served as a toy example of tuning AMCL in a simulated environment. To that point, it was helpful to learn basic AMCL parameter tuning.

The techniques described above could be implemented easily on a hardware platform that supports Ubuntu linux. The hardware laser scanner and drive system would need plugins for ROS.

For future work, other sensors and drive systems could be explored such as the four wheel steering controller, etc.

6.1 Modifications for Improvement

Examples:

- Base Dimension No change necessary.
- Sensor Location No change necessary.
- Sensor Layout Consider adding IMU.
- Sensor Amount Consider additional sensors.

6.2 Hardware Deployment

1) What would need to be done? Mechanical construction, motors & motor controllers, power supply and battery management, sensor hardware, single board computer.
2) Computation time/resource considerations? There is a tradeoff in robot responsiveness (how fast it can process the update loop) and processing power.