Robotics Software Engineer Nanodegree: Localization and Navigation Project

Lincoln Stein

Abstract—This project focuses on the implementation of localization for a mobile robot. Adaptive Monte Carlo Localization is implemented using 2D LIDAR scan data. This algorithm is also compared with Kalman Particle Filters. Two robot models are compared regarding physical configuration and optimization of the AMCL parameters.

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

1 Introduction

FROM the autonomous vacuum cleaning your floors to the navigation system in your car, localization serves an important role to enable path planning around your home and around the world. The hardware and sensors used for localization takes many forms: encoders to track distance for dead reckoning systems, GPS receivers that rely on a constellation of satellites, and various laser and imaging systems that allow perception of the surrounding environment. The software used in these tasks is also quite varied with implementations of geometry and calculus to track a path from a known point to statistical methods and pattern matching algorithms. The primary goal is enabling the robot to successfully localize itself within the environment using sensor data and a provided map and then navigate to the goal location. The robots in this project use probability based Adaptive Monte Carlo localization to localize itself then navigate to the goal location.

2 Background

2.1 Kalman Filters

Kalman filters are an algorithm to generate a state space estimation. The Kalman filter In terms of mobile robot localization, a Kalman filter is an algorithm that represents the a robots location as a Gaussian probability distribution. This distribution is created by forming an initial estimate of the robots pose and updating it based on sensor readings. Both the initial estimate and the sensor readings are formulated as a probability distribution with a variance to account for noise and randomness present in sensor readings. A Kalman filter utilizes the information present in the two probability distributions to make a pose estimate that is more confident than either the initial estimate or the sensor reading by using a weighted average based towards those distributions with less uncertainty. The Kalman filter is utilized for localization because it is computationally efficient to use in systems with limited hardware as it uses linear state transitions and Gaussian probability distributions. These assumptions and the accuracy of the filter are acceptable in certain cases. However, attempts have been made to extend the traditional Kalman filter to address its inability to handle non linear systems by creating and Extended Kalman Filter. The extended Kalman filter uses a linear approximation of the non linear function over a small section to update the variance of a state prediction or sensor measurement. These approximations allow the extended Kalman filter to be applied to non linear problems, such as converting from polar coordinates of a sensor measurement to a Cartesian representation of state space.

2.2 Particle Filters

A particle filter uses random sampling of a state space, such as a robots possible pose, and represents each sample as a particle. These particles are then filtered as sensor data is gathered to remove those with a lower probability of matching and new particles are generated based on the updated state space. It is possible to obtain an accurate estimate of the actual state by recursively applying this method. A particle filter is able to account for non Gaussian distributions in the state space where other methods may fail. In terms of mobile robot localization, a particle filter is an algorithm that represents the a robots location as a probability distribution using a set of particles or samples. This distribution is created by forming an initial estimate of the robots pose and updating it based on sensor readings. Both the initial estimate and the sensor readings are formulated as a probability distribution with a variance to account for noise and randomness present in sensor readings. A particle filter utilizes the information present in the two probability distributions to make a pose estimate that is more confident than either the initial estimate or the sensor reading by using a weighted average based towards those distributions with less uncertainty. The nature of the algorithm allows us to compensate for highly variable transforms between states and very noisy sensor data to obtain an accurate pose estimate. Particle filters are an efficient method to model non linear systems such as a mobile robot and converge on a solution to the pose estimation problem.

2.3 Comparison / Contrast

The Kalman filters primary advantage is computational efficiency. For systems that are linearly related or able to be approximated sufficiently as such, the Kalman filter can provide efficient filtering. The particle filter can handle non linear systems through its use of random sampling within a state space eventually converging on a solution. This flexibility comes at a higher computational load and also carries the possibility that the correct state space estimate will be lost in the filtering. However, because of its ability to handle non linear systems effectively, the work presented here will focus on the use of particle filters.

3 Simulations

3.1 Achievements

Both the benchmark and the personal robot model were able to successfully localize within the given environment and navigate to the goal position. The benchmark model is able to perform this in approximately 60 seconds where the personal model requires approximately 120 seconds.

3.2 Benchmark Model

3.2.1 Model design

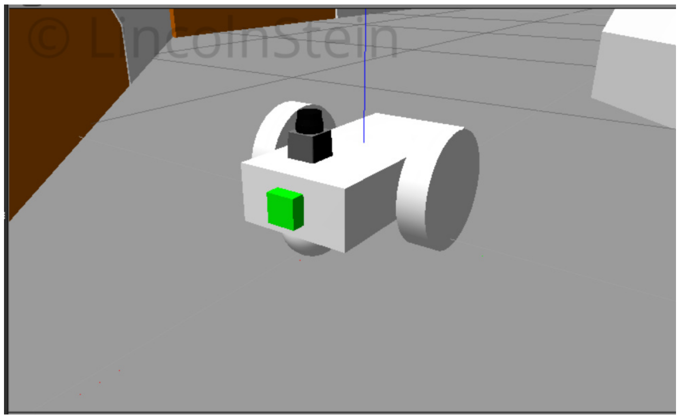

Fig. 1. Benchmark Robot Model for Localization

TABLE 1
Benchmark Model Design

Parameter	Value
Body Size	0.4m x 0.2m x 0.1m
Wheel Radius	0.1m
Laser Scanner on top of robot	30m, +- 30mm at <10m
Camera on front of robot	RGB Camera

3.2.2 Packages Used

The robot operation depends primarily on two packages, move_base and amcl. The ROS nodes and topics are shown in Figure 2. As shown in the graph in Figure 2, the move_base package takes in map, laser scan, sensor transform, odometry, and goal pose topics. The move_base package performs navigation by calculating a path from the initial location to the goal position utilizing the information it receives. It sends commands to the base controller to attempt to navigate along this path. The AMCL package is used to perform localization using transform data, laser scan data and a provided map. AMCL uses the static map service to retrieve map data to be used for localization.

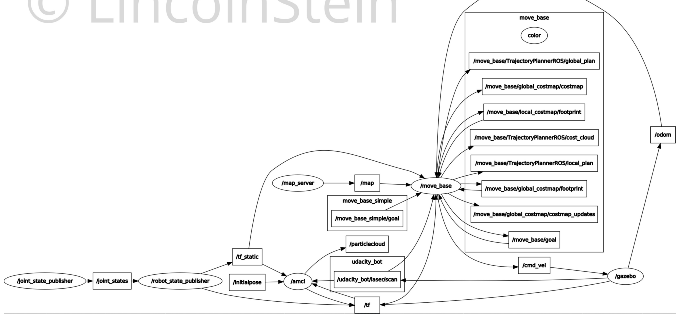
move_base
color
/move_base/TrajectoryPlannerROS/global_plan
/move_base/global_costmap/costmap
/move_base/local_costmap/footprint
/move_base/TrajectoryPlannerROS/cost_cloud
/move_base/TrajectoryPlannerROS/local_plan
/move_base/global_costmap/footprint
/move_base/global_costmap/costmap_updates
/move_base/goal
/odom
/map_server
/map
/move_base
move_base_simple
/move_base_simple/goal
/particlecloud
/tf_static
udacity_bot
/udacity_bot/laser/scan
/cmd_vel
/gazebo
/joint_state_publisher
/joint_states
/robot_state_publisher
/initialpose
/amcl
/tf

Fig. 2. ROS Nodes and Topics Graph for Localization Robot

3.2.3 Parameters

The AMCL parameters affect how the robot processes information to update the estimates of pose. The primary values that required tuning are shown in table 2. Min and max particles determine how many different pose estimates are sampled from the state space and should be tuned based on system computing available as higher values cause long update delays and diminishing returns in terms of capabilities. The transform tolerance sets how long a particular transform, such as sensor data, is valid and able to be used. The update minimums set the threshold for calculating a new filter update based on translational and rotational movement. These minimums need to be tuned to so updates are not performed too frequently which can waste processor time. Finally, the controller frequency is tuned based on how quickly the robot needs to be able to react and is dependent on how fast updates to the filter are calculated.

TABLE 2
Benchmark Model AMCL Parameters

Parameter	Value
min_particles	50
max_particles	500
transform_tolerance	0.2
update_min_d	0.1
update_min_a	0.25
controller_frequency	10

The common costmap parameters set values used by both the local and global costmaps. This includes obstacle and raytrace range, which sets the maximum distance at which the robot will add an obstacle to the costmap and the max distance that it will try to clear the costmap based on the laser scan data. The footprint is used to specify the size of the robot to be used in the costmap and the inflation radius sets the minimum distance from obstacles when calculating the costs.

The global costmap is used by the global planner. The global costmap parameters set the size of the global map and frequency of update and publishing. The update and publish frequency are tuned to balance processing capabilities and how fast the costmap needs to be updated. In general, the global costmap can be updated less frequently, especially in the static environment of the maze.

The local costmap parameters used for the costmap centered around the robot used by the local planner. The

TABLE 3
Benchmark Model Common Costmap Parameters

obstacle_range	4.00
raytrace_range	5.00
min_obstacle_height	1.0
transform_tolerance	0.2
footprint	[[0.2, 0.1], [-0.2, 0.1], [-0.2, -0.1], [0.2, -0.1]]
inflation_radius	1.0

TABLE 4
Benchmark Model Global Costmap Parameters

update_frequency	5.0
publish_frequency	5.0

global frame sets the reference for the local costmap. As with the global costmap, the update and publish frequency need to balance processing power and speed. In general, the local costmap should be updated more frequently to allow collision free navigation.

TABLE 5
Benchmark Model Local Costmap Parameters

global_frame	odom
robot_base_frame	robot_footprint
update_frequency	10.0
publish_frequency	10.0

The local planner parameters determine how the robot moves locally. The differential drive robot for the benchmark model requires holonomic set to false. Sim time sets the length of time the potential trajectories are simulated before scoring. Pdist and gdist scale adjust the weights to favor trajectories that follow the global path or the local goal, respectively. A large pdist factor forces the robot to follow the path more precisely but can cause issues in local pathing, such as running into obstacles. The gdist factor increases the weight of trajectories that keep the robot close to the local goal as well as controlling speed. The publish cost grid is set to true which is helpful for debugging as it allows the engineer to visualize the cost map scoring to tune parameters better. Heading lookahead sets the distance that the planner uses to score in place rotations and helps determine the robots behavior regarding these moves.

TABLE 6
Benchmark Model Base Local Planner Parameters

holonomic_robot	false
sim_time	3.0
meter_scoring	true
pdist_scale	0.7
gdist_scale	0.8
publish_cost_grid_pc	true
heading_lookahead	0.50
occdist_scale	0.1

3.3 Personal Model

3.3.1 *Model design*

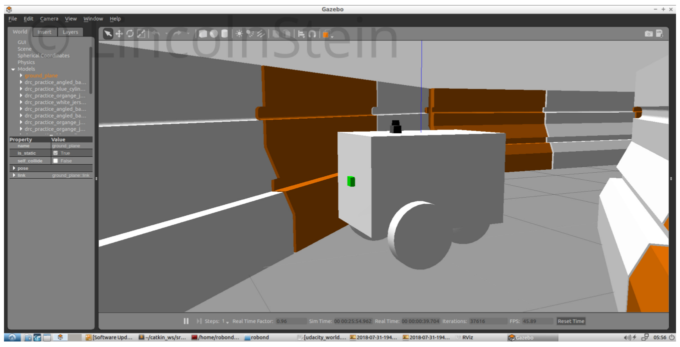

Fig. 3. Personal Robot Model for Localization

TABLE 7
Personal Model Design Layout

Parameter	Value
Body Size	0.8128m x 0.508m x 0.508m
Wheel Radius	0.171m
Laser Scanner on top of robot	30m, +- 30mm accuracy at <10m
Camera on front of robot	RGB Camera

3.3.2 *Packages Used*

The robot operation depends primarily on two packages, move_base and amcl. The ROS nodes and topics are shown in Figure 4. As shown in Figure 4 above, the move_base

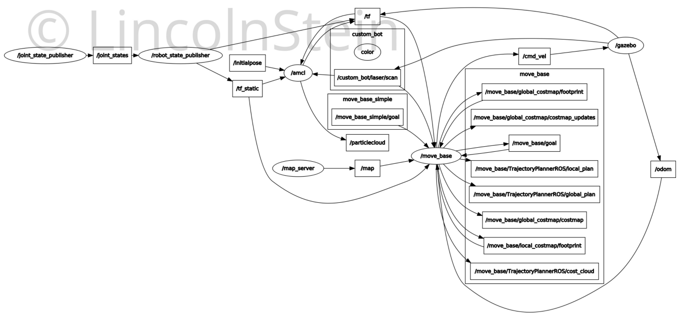
/joint_state_publisher
/joint_states
/robot_state_publisher
/tf
custom_bot
color
/initialpose
/amcl
/custom_bot/laser/scan
/tf_static
move_base_simple
/move_base_simple/goal
/particlecloud
/map_server
/map
/move_base
/cmd_vel
/gazebo
move_base
/move_base/global_costmap/footprint
/move_base/global_costmap/costmap_updates
/move_base/goal
/move_base/TrajectoryPlannerROS/local_plan
/move_base/TrajectoryPlannerROS/global_plan
/move_base/global_costmap/costmap
/move_base/local_costmap/footprint
/move_base/TrajectoryPlannerROS/cost_cloud
/odom

Fig. 4. ROS Nodes and Topics Graph for Personal Localization Robot

package takes in map, laser scan, sensor transform, odometry, and goal pose topics. The move_base package performs navigation by calculating a path from the initial location to the goal position utilizing the information it receives. It sends commands to the base controller to attempt to navigate along this path. The AMCL package is used to perform localization using transform data, laser scan data and a provided map. AMCL uses the static map service to retrieve map data to be used for localization.

3.3.3 *Parameters*

The personal model common costmap parameters were updated to reflect the larger size of the robot footprint.

TABLE 8
Personal Model AMCL Parameters

Parameter	Value
min_particles	50
max_particles	500
transform_tolerance	0.2
update_min_d	0.1
update_min_a	0.25
controller_frequency	10

TABLE 9
Personal Model Common Costmap Parameters

obstacle_range	6.00
raytrace_range	5.00
min_obstacle_height	1.0
transform_tolerance	0.2
footprint	[[0.42, 0.36], [-0.48, 0.36], [-0.48, -0.36], [0.42, -0.3
inflation_radius	0.3

TABLE 10
Personal Model Global Costmap Parameters

update_frequency	5.0
publish_frequency	5.0

In tuning the personal robot model parameters, a much smaller local costmap was used. This was done to ensure the goal location was not within the local costmap upon initialization and created less conflict between trying to reach global versus local goals.

TABLE 11
Personal Model Local Costmap Parameters

global_frame	odom
robot_base_frame	robot_footprint
update_frequency	10.0
publish_frequency	10.0
width	5.0
height	5.0

TABLE 12
Personal Model Base Local Planner Parameters

holonomic_robot	false
sim_time	3.0
meter_scoring	true
pdist_scale	0.6
gdist_scale	0.8
publish_cost_grid_pc	true
heading_lookahead	1.0
occdist_scale	0.1
yaw_goal_tolerance	0.15
xy_goal_tolerance	0.3

The heading lookahead and goal tolerances were adjusted to account for the larger robot base in case a goal is placed too close to a obstacle, the robot can still reach a close point to that goal.

4 Results

4.1 Localization Results

4.1.1 Benchmark

The benchmark robot model localizes itself well within the environment and successfully navigates to the goal in approximately a minute. Initially, the robot may move away from the global path over the first few seconds as its localization filter converges and then it corrects to follow the global path mostly smoothly to the goal. This initial deviation can be attributed to the size of the local costmap which is large enough to encompass the goal location and may lead to conflict between local and global goal seeking. The successfulness of the localization is shown in Figure 5

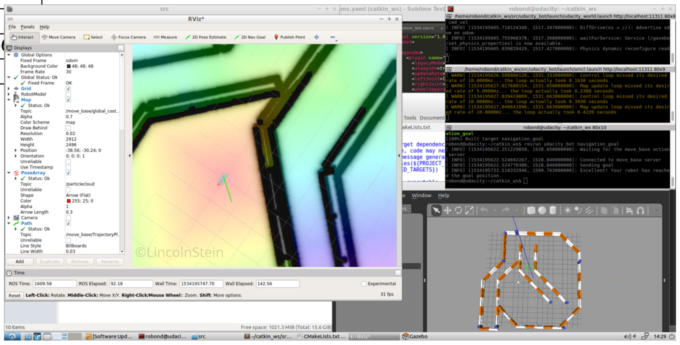
Fig. 5. Benchmark Model At Goal Position

4.1.2 Student

The personal robot model also localizes itself quite well within the environment, requiring about 2 minutes to successfully reach the goal. The smaller local costmap makes it more likely that the robot follows the global path from the beginning as there is less conflicting scoring since the local goal is aligned on the path to obtain the global goal.

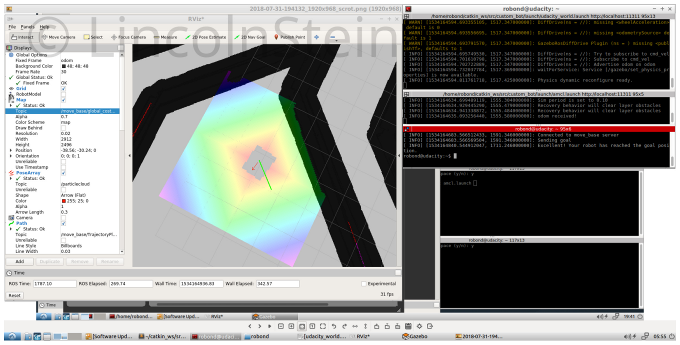
Fig. 6. Personal Model At Goal Position

4.2 Technical Comparison

The primary differences in the design of two models is the size of the robot. The chassis and wheel size was increased and the wheels were shifted forward replacing the front caster for the personal model . The parameters did not require much change from tuning performed for the benchmark model except to account for these changes in size, such as with the robot footprint and inflation radius.

5 Discussion

5.1 Topics

- Which robot performed better? In the current configurations, the benchmark robot was able to successfully localize and travel to the goal location nearly twice as fast as the personal model.
- Why it performed better? (opinion) The benchmark model was likely faster as it had more free space in the maze corridor so while the localization converged it had more free space to move and perform updates compared to the personal model. Once localized successfully, both robots navigated similarly.
- How would you approach the 'Kidnapped Robot' problem? The difficulty in addressing a kidnapped robot greatly increases after the particles have converged as the likelihood of sampling outside of this location is greatly decreased. One method to address this would be to maintain a higher number of particles to increase the chance of sampling near the new location after the kidnapping. Another method would be to add more particles at each iteration to ensure there is always some spread throughout the state space.
- What types of scenario could localization be performed?
- Where would you use MCL/AMCL in an industry domain? Monte Carlo Localization could be effectively used in many different domains such navigation of a vehicle or autonomous system or determining a customer location within a store,

6 Conclusion / Future work

The goal of this project was to implement Adaptive Monte Carlo Localization for a benchmark and personal model in simulation. Given the parameters described earlier, both models were able to successfully localize and navigate to the required goal location. The ability to successfully localize within an environment could be applied to indoor navigation in a warehouse or store for a stocking robot, in a home for a cleaning or service robot, outside the home navigating on roads or even around a park providing transportation for guests. Future work in this area would be focused on adding sensors and refining the accuracy of localization and navigation results. Implementing approaches to address the "kidnapped robot" problem would also be an important area to further develop.

6.1 Modifications for Improvement

Examples:

- Sensor Location - While suitable for the obstacles within this environment, only sensor obstacles on a plane higher than the robot causes large blind regions
- Sensor Layout/Amount - Additional sensors positioned to detect a larger volume of space around the robot would enable it to detect obstacles more accurately and potentially localize more effectively.

6.2 Hardware Deployment

1) What would need to be done? In order to deploy this, it would be necessary to ensure the hardware and software parameters matched as close as possible in addition to the physical configuration of the robot and sensors.
2) Computation time/resource considerations? Depending on the platform for deployment, tuning of the particle count and transform tolerance would be important to tune properly depending on hardware capabilities. Other important factors would be matching controller frequency and costmap updating rates with what the hardware is actually capable.

References

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