Where Am I

Nicholas Zuiker

Abstract—This project uses the Monte Carlo Algorithm in a Gazebo/RViz environment to solve the problem of robot localization, the ability for robots to understand their state and make a more informed action decision. Using the URDF standard and the ROS framework, a robot was designed to navigate through a provided map and was compared to another baseline robot provided by Udacity. The developed robot uses AMCL packages to process and handle the back-end localization of the robot. Through careful tuning of different costmap and base local parameters, the designed robot was able to successfully localize itself and reach a chosen destination goal.

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

1 Introduction

Localization is a major component of modern robotics. This is the process by which robots identify their location so that they can perform effective path planning towards their objective. In global localization, the robot's initial pose is unknown, and the robot tries to determine its pose relative to a ground truth map. In this project, a particle filter is used to perform global localization to help two robots navigate a given map.

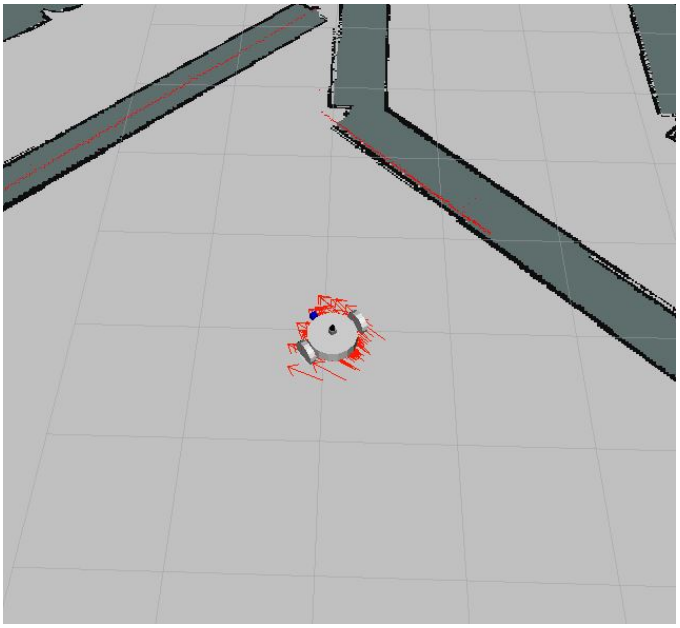

Fig. 1. Custom robot navigating towards the goal.

2 Background

Localization can be done using a variety of methods depending on the parameters and context. Within the scope of this project, the focus is on Kalman Filters and Monte Carlo Localization (particle filtering).

2.1 Kalman Filters

Kalman filters estimate the value of a variable in real time by measuring data. Kalman filters can take data with significant uncertainty or noise in the measurements and still provide an accurate and efficient estimate of the real value. However, the Kalman filters generally assume that measurement models are linear and can be represented uni-modally. While this would severely limit the practical applications of Kalman filters, there exists an "extended" Kalman filter (EKF) that solves this problem by linearizing non-linear problems to provide a Gausssian distribution.

2.2 Particle Filters

Monte Carlo Localization (MCL) is effectively a particle filter algorithm and is widely used in robotics. MCL uses particles to localize the robot. Each particle is given a random pose as a best guess of where the robot is located. These particles are then continuously resampled with replacement every time the robot moves and measures the surrounding environment. Over the course of many iterations, these particles converge with the robot. This provides effective localization.

2.3 Comparison / Contrast

EKF and Monte Carlo Localization can both be effective means of localizing a robot, but there are a few key differences. One advantage of a Kalman filter (extended or not) is that it will typically converge on a robot's location much faster than a particle filter will. However, Kalman filters are typically more difficult to program and implement. Additionally, particle filters can be used for non-linear models without having to modify the measurement inputs. Lastly, particle filters provide a bit more control over computational resources in that the amount of particles being generated can be increased for better accuracy or decreased for faster performance and less memory usage.

3 Simulations

Simulation for this project was completed in Gazebo and RViz within Udacity's online workspace. A launch file was

created for ROS that would launch both the world and the robot within the world. Additionally, the navigation stack and packages for AMCL were installed inside the catkin workspace to provide the robots with localization capabilities. The code written to navigate the robot to its eventual goal was written in C++ and also included in the catkin workspace.

3.1 Achievements

The included Udacity model and the custom robot model both reached the navigation goal after several rounds of trial and error tuning the parameters. Each model is discussed in more detail below.

3.2 Benchmark Model

3.2.1 Model design

The benchmark Udacity model has a rectangular prism-shaped chassis with a width of 0.2, length of 0.4, and height of 0.1. There are two spherical casters with identical radiuses of 0.0499 to provide balance and two wheels with a cylindrical shape with radius of 0.1 and a length of 0.05. The two wheels are connected to the chassis via continuous joints so that they are free to rotate. There are also two sensors, a camera, and a laser range-finder.

3.2.2 Packages Used

The ROS package used for Udacity's provided bot contained the following:

- meshes
- urdf
- worlds
- launch
- maps
- rviz
- src
- config

This ROS package comes in addition to the Navigation stack and AMCL packages.

TABLE 1
Udacity Bot Specs

Model		
Part	Geometry	Size
Chassis	Cube	0.4 x 0.2 x 0.1
Casters	Sphere	0.0499 (rad)
Wheels	Cylinders	0.1 (rad), 0.05 (len)
Equipment		
Camera Sensor	Link origin	[0,0,0,0,0,0]
	Shape, Size	box, 0.05^3
	Joint Origin	[0.2,0,0,0,0,0]
	Parent Link	chassis
	Child Link	camera
Hokuyo Sensor	Link origin	[0,0,0,0,0,0]
	Shape, Size	box, 0.1^3
	Joint Origin	[0.15,0,0.1,0,0,0]
	Parent Link	chassis
	Child Link	hokuyo

3.2.3 Parameters

To obtain the most accurate localization results, several parameters were added and tuned in an iterative process of trial and error. In the AMCL node, the most important parameters were 'min particles' and 'max particles', which were respectively set to 10 and 200. An increase of particles would mean an increase in accuracy from having more particles being filtered and therefore a better idea of the robots real position. However, more particles would also have a negative impact by increasing computational load, slowing processing and taking up more memory.

Several other parameters were tuned in the different config files. The transform tolerance (delay in transform data that is tolerable), inflation radius ("inflates" obstacles so that the robot will avoid getting too close and risk getting stuck), robot radius, and obstacle range (distance at which obstacles begin to be perceived) were optimized after several iterations of testing. Increasing the inflation radius has a positive impact on the costmap by eliminating risk of collision. Detecting obstacles' distances related to the where the robot would navigate ensures more reliable navigation. Robot radius represents the radius of the robot as it relates to its environment. Having a larger robot radius decreases the likelihood of the robot getting stuck on obstacles because it cannot fit into riskier smaller areas, but this leads the robot to believe it is bigger than some spaces it may need to pass through. Additionally, since a "diff-corrected" odom model is used, odom alphas (odometer measurement noise) needs to be provided. Due to the smooth environment, very small values of 0.01 were chosen for these alphas (meaning the odometer is trusted to be very precise).

TABLE 2
Udacity Bot Costmap Parameters

Costmap Parameters		
Parameter	Global	Local
global frame	map	odom
robot base frame	robot footprint	robot footprint
update frequency	15.0	0.15
publish frequency	15.0	15.0
width	20.0	5.0
height	20.0	5.0
resolution	0.05	0.05
static map	true	false
rolling window	false	true

3.3 Personal Model

3.3.1 Model design

The personal mobile robot, named zuik bot, was created in a new URDF file within the same udacity bot package with remapped subscribers and publishers. In this file, the size and geometry of the robot is specified, and a physical configuration is setup with inertial and collision parameters. The zuik bot has a cylindrical shape chassis with a radius of 0.2 and length of 0.1 to give it the look of a Roomba vacuum robot. There are two spherical casters with radiuses of 0.0499 to provide balance and two tow wheels with a cylindrical shape of 0.1 radius and a length of 0.05 that are represented as individual links. The two wheels are

TABLE 3
AMCL and Other Parameters, Udacity Bot

AMCL Node Parameters	
min particles	10
max particles	200
initial pose x	0
initial pose y	0
initial pose a	0
odom model type	diff-corrected
odom alpha 1	0.010
odom alpha 2	0.010
odom alpha 3	0.010
odom alpha 4	0.010
Costmap Common Parameters	
obstacle range	2.5
raytrace range	3.0
transform tolerance	0.3
robot radius	0.25
inflation radius	0.5
Base Local Planner Parameters	
holonomic robot	false
yaw goal tolerance	0.05
xy goal tolerance	0.1
sim time	1.0
meter scoring	true
pdist scale	0.5
gdist scale	1.0
max vel x	0.5
max vel y	0.1
max vel theta	2.0
acc lim theta	5.0
acc lim x	2.0
acc lim y	5.0
controller frequency	15.0

connected to the chassis via continuous joints, so they are free to rotate. The udacity bot is also equipped with two on-board sensors, a camera and a laser range-finder.

TABLE 4
Zuik Bot Specs

Model		
Part	Geometry	Size
Chassis	Cylindrical	0.2 (rad), 0.1 (len)
Casters	Sphere	0.0499 (rad)
Wheels	Cylinders	0.1 (rad), 0.05 (len)
Equipment		
Camera Sensor	Link origin	[0,0,0,0,0,0]
	Shape, Size	box, 0.05^3
	Joint Origin	[0.25,0,0,0,0,0]
	Parent Link	chassis
	Child Link	camera
Hokuyo Sensor	Link origin	[0,0,0,0,0,0]
	Shape, Size	box, 0.1^3
	Joint Origin	[0,0,0.1,0,0,0]
	Parent Link	chassis
	Child Link	hokuyo

3.3.2 Packages Used

Existing within the same udacity bot package, there were no major changes that needed to be made to launch zuik bot other than creating a seperate launch file.

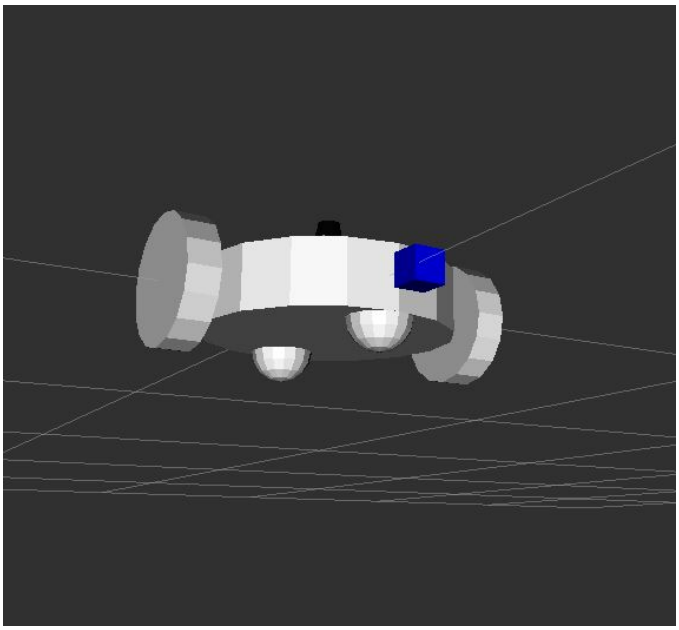

Fig. 2. A closer look at zuik bot.

3.3.3 Parameters

Several of the costmap common parameters were slightly modified to accomodate the the larger body of zuik bot. The robot radius was increased to 0.35 due to the larger model. Obstacle range was increased slightly, but this parameter doesn't seem to have large effect on the navigation time. Lastly, inflation radius was increased to 0.6 as a precaution with the robot's larger radius. If walls on either side of an alley are close enough though, this can be a dangerous value to change due to the robot potentially believing there is no navigable path.

TABLE 5
Costmap Common Parameters, Zuik Bot

Costmap Common Parameters	
obstacle range	3.0
raytrace range	3.0
transform tolerance	0.3
robot radius	0.35
inflation radius	0.6

4 RESULTS

After simulating with the eventual parameters, the results show both robots (benchmark and personal models) were able to reach the designated goal with relative ease. On both models the particle filters converged approximately a few seconds after launching. However, some of the iterations behaved slightly different than others, taking a bit longer to converge. Overall, both robots traversed the map with only light interruptions. Both models struggled with collision when turning around an obstacle. Both mobile robots reached their designated goal successfully, solving the localization problem in the process.

4.1 Localization Results

4.1.1 Benchmark Model

When the simulation starts, the particles are wide spread (See Fig. 3).

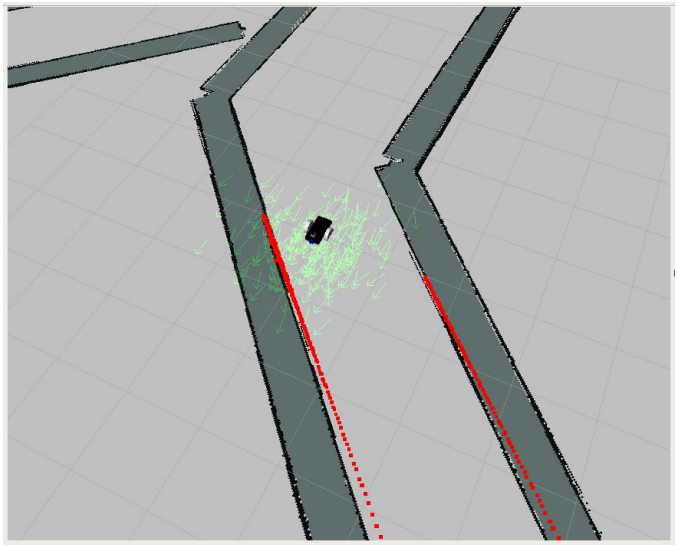

Fig. 3. Starting position for Udacity robot and particle poses.

After just a few iterations through MCL, the particles begin to converge much tighter on the robot (See Fig. 4).

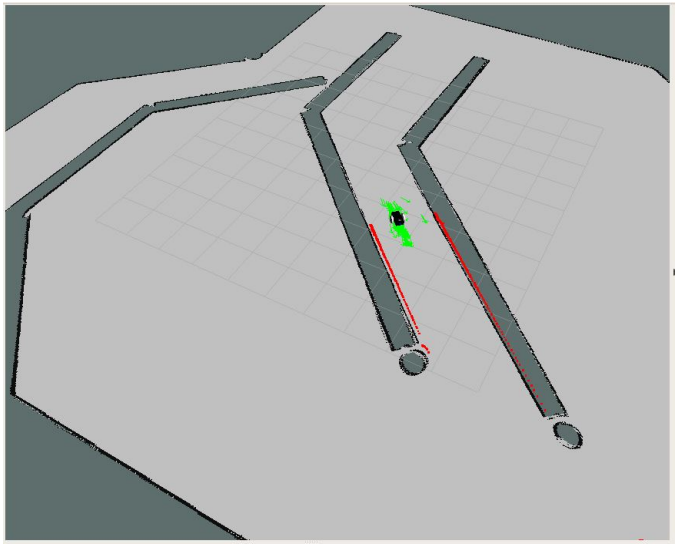

Fig. 4. Particles begin to converge.

After turning around an obstacle, the particles have converged enough to be almost entirely under the robot model (See Fig. 5).

Finally, the robot reaches its goal destination and orients correctly (See Fig. 6).

4.1.2 Zuik Bot

Through a similar process of slowly converging over time, zuik bot also eventually reaches the destination (See Fig. 7).

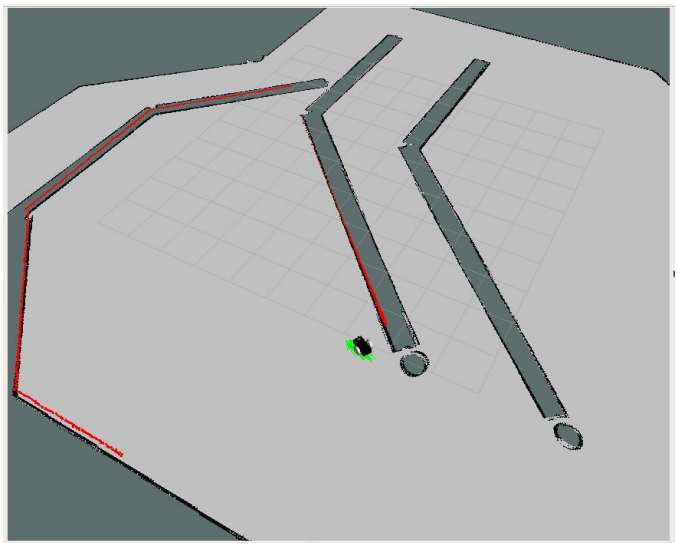

Fig. 5. Tight convergence as robot nears destination.

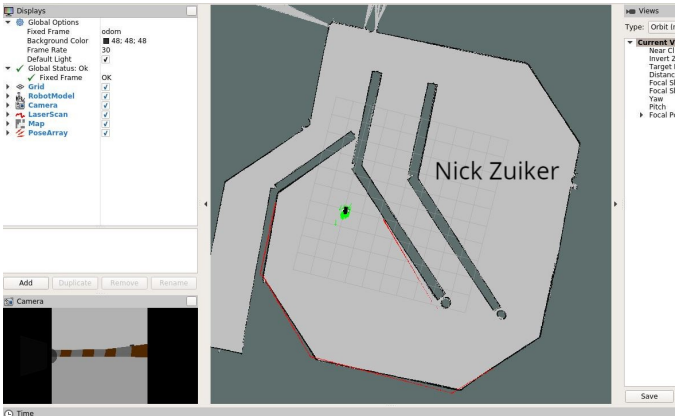

Fig. 6. Udacity bot successfully arrives at destination.

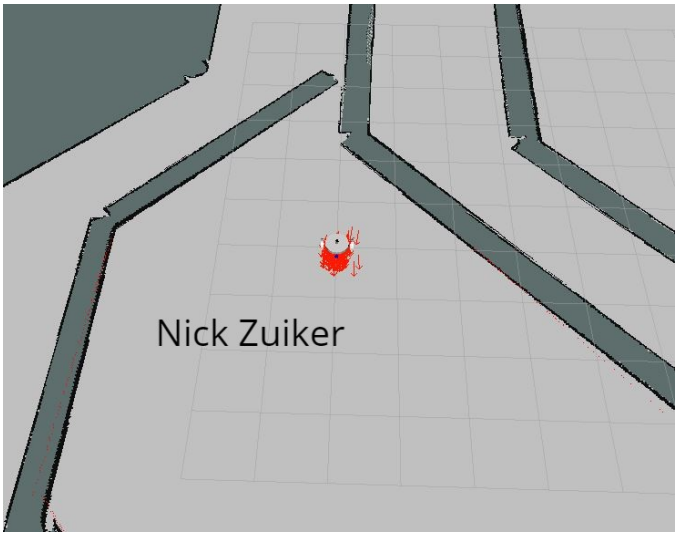

Fig. 7. Zuik bot successfully arrives at destination.

4.2 Technical Comparison

The major difference between the Udacity bot and Zuik bot was the physical shape of each robot. This affected the turning and navigation of the larger Zuik bot and made its navigation a bit slower compared to the benchmark model. Overall, both models were able to reach the destination in less than a minute.

5 Discussion

- Overall, the udacity bot performed better and navigated the map faster. While Zuik bot was able to navigate to the goal, it needed to make much larger turns to avoid obstacles due to its larger size. Perhaps Zuik bot could be a more reliable robot in terms of consistent completion to different goal locations, but that hypothesis would need to be tested.
- It can be seen that the kidnapped robot problem could actually be solved almost the exact same way as in this project. The starting particles would need to more spread out to represent all the potential locations that the robot may be located, and there may need to be more starting particles, but eventually the particles would still converge to the robot.
- It seems that MCL/AMCL could have a fairly extensive amount of utility for various industries. However, when being applied to large spaces with unknown maps, the amount of particles required for adequate performance would require an enormous amount of computational resources.

6 Conclusion / Future work

After all the iterations and trial by error, it can be concluded that AMCL is a powerful tool for solving the localization problem for robots. MCL's ease of implementation and immense community support and popularity make this algorithm a strong candidate for developing autonomous and location-aware robots. However, consideration must be given to the number of particles chosen to be used for a good filter. A larger number of particles results in a more accurate model that is more likely to represent the robot's true position, but there is considerable trade-off in that the filter will require more processing time to iterate through every particle. In future work, implementing AMCL for 3-Dimensional localization problems seems like the next logical step. You could use this drones or factory actuation accuracy via a robotic arm. An additional improvement for this project might come in the form of added sensors and sensor fusion for additional accuracy.