Monte Carlo Localization

Kenny Kang

Abstract—The main objective of this project reduces to 1) building a robot model from scratch for a Gazebo environment and 2) using a known map and relevant sensor data to accurately localize the robot using the Monte Carlo Localization algorithm. In order to measure the accuracy of said localization, a navigation component is added to the project to validate the practicality of the results.

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

1 Introduction

The process of localization helps determine a robot's pose with respect to a known environment. The key obstacle in this problem is uncertainty. Due to the inevitability of imperfect sensors, dealing with measurement noise is necessary as these errors have the potential to compound with time. In order to deal with uncertainty, popular localization algorithms implement probabilistic methods. Another issue that must be addressed is the computational limitation of these algorithms. With an unlimited amount of computational resources, many of the problems in modern localization algorithms would be negligible. Unfortunately, compute power is limited, especially in mobile robots where localization is primarily used.

2 Background

The field of robotics contain many subtopics which include perception and planning. In order for an autonomous system to navigate, it must know its environment as well as its specific location. These are the goals of Mapping and Localization. Localization assumes the map of the environment is already known. Algorithms without this assumption lie in the domain of SLAM Algorithms (Simultaneous Localization and Mapping). The importance of localization lie in its usage in mobile robotics. Many practical applications such as self-driving cars or automated home vacuums rely on being able to find it's own location in its respective environment.

Two popular localization algorithms are Extended Kalman Filters(EKF) and Monte Carlo Localization(MCL). Within this project, Monte Carlo Localization is tested due to its ease of implementation and overall robustness. Unlike EKF, MCL is not restricted to a linear Gaussian state space and is able to approximate any distribution. This allows the algorithm to be used in a much larger variety of environments.

2.1 Kalman Filters

The Kalman Filter is an algorithm which allows for state estimation with the presence of noise. This algorithm makes the assumption that the final state following a measurement or motion is modeled as a gaussian distribution as seen in Figure 1.

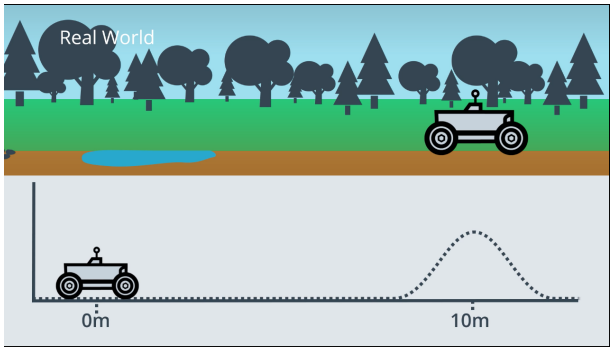

Fig. 1. Gaussian Distribution of states

This Gaussian consists of a mean and variance which are interpreted as an estimated state and uncertainty respectfully. In order to minimize uncertainty, the current measured state is compared to the prior state with the motion applied to it. Averaging these two Gaussians results in a new mean with a variance that represents more certainty than the previous two estimates. While intuitive, this algorithm is limited to environments with linear Gaussian state spaces.

The Extended Kalman Filter solves this limitation by allowing the use of non-linear functions. While a nonlinear function can update the mean of a Gaussian, it cannot update the variance as it will result in a non-Gaussian distribution. Since these distributions are much more computationally expensive to work with, this must be avoided. Fortunately, a non-linear function can be linearized to produce a Gaussian which is the core idea behind the EKF.

2.2 Particle Filters

A Particle Filter, also known as Monte Carlo Localization, is an algorithm which uses the history of a robots measurements to localize its pose with repsect to its environment. This is done by comparing the poses of simulated robots with sensory information from the main robot over time to filter the most likely pose of the main robot.

This is an iterative algorithm which starts with the initialization of many robots, or particles, in the environment with randomized poses and orientations. These particles represent the possible poses for the main robot. As the robot receives input, the same movements are simulated with

each particle. At the end of the motion, these particles are given weights based on their distances to landmarks that are sensed by the main robot. These steps can be visualized in Figure 2. Once particles are weighted, they are re-sampled based on these weights to filter out the most unlikely poses. The next iteration follows with a new motion.

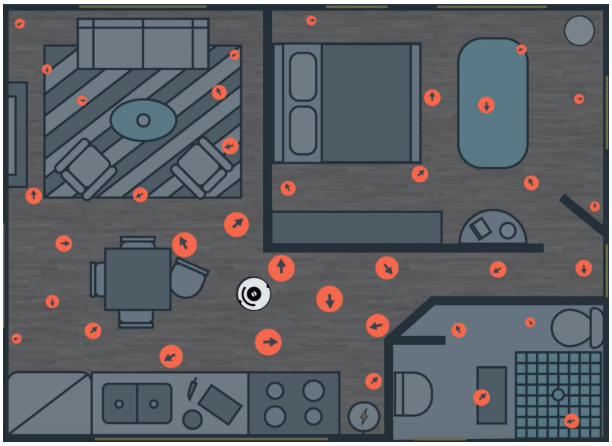

Fig. 2. Particle Filter Visualization

2.3 Comparison / Contrast

Kalman filters and Particle filters are tools that have their respective advantages. Kalman filters are more memory and time efficient, and as a result are better suited for real time systems. On the other side, particle filters are not limited by linear Gaussian measurements. Therefore, it is a more robust algorithm. In addition, it allows for control over memory and resolution unlike the kalman filter which is helpful when computational resources are a limited. Due to these reasons, the particle filter was chosen as the localization algorithm used in this project. More specifically, a variation of MCL called Adaptive Monte Carlo Localization(AMCL) is used. This algorithm dynamically adjusts the number of particles over time, significantly reducing computation.

3 Simulations

In order to test the localization algorithm, Gazebo was used to simulate the robot and it's environment. The Robot Operating System(ROS) was used to create components of the robot and allow communication across nodes and sensors. The Rviz package was used to visualize this data.

3.1 Achievements

Using two different robot models, each robot was able to localize itself using AMCL and navigate through an environemnt to a predetermined goal destination.

3.2 Benchmark Model

3.2.1 Model design

The benchmark model consisted of a rectangular chassis and two wheels. This model included a camera sensor and a laser scanner located on the front and top of the chassis respectfully.

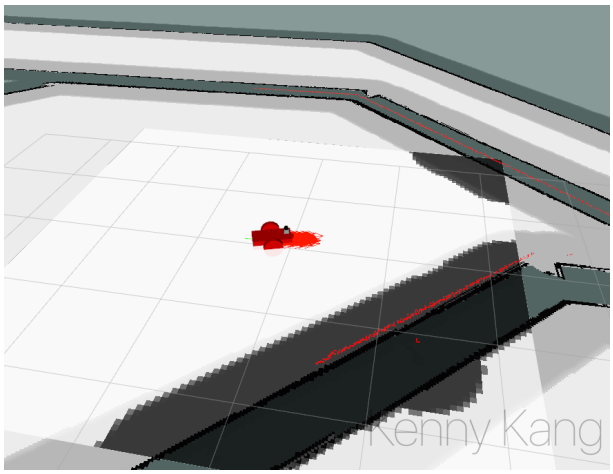

Fig. 3. Benchmark Model reaching goal destination

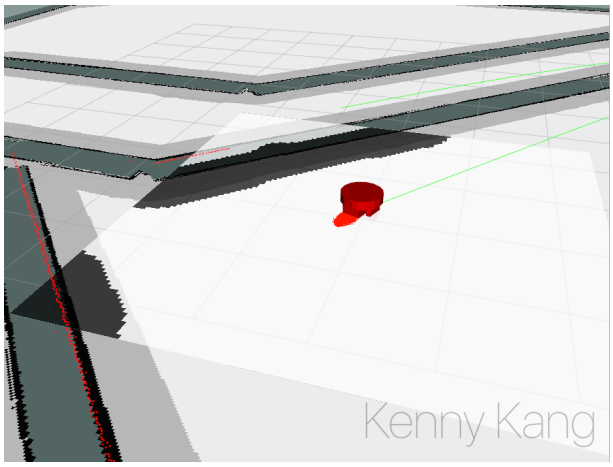

Fig. 4. Personal Model reaching goal destination

3.2.2 Packages Used

The simulation primarily utilized the ROS Navigation stack which includes packages for AMCL, local planning, and cost maps. The AMCL package subscribed to the hokuyo scanner, the map, and the initial robot pose. The package then publishes the estimated pose along with the estimated filter poses. The local planner package subscribes to the odom and publishes a global plan, local plan, and a cost grid of the environment. Finally the cost maps subscribes to the robot footprint and publishes a costmap of the environment.

3.2.3 Parameters

While there were many parameters for each package, there were only a handful that significantly affected the overall results. The width and height of the local cost map was the most significant factor which contributed to an accurate path of the robot. In early iterations, a large local cost map often led to the robot roaming in circles, never following the path which was accurate according to the Rviz visualization. Inflation radius was another parameter that helped in the navigation aspect of the robot. When too low, the robot frequency stopped moving once reaching a turn as the planned path followed the cost map too closely, leading to the robot driving into a wall. However, a high value led

TABLE 1
Benchmark Model Parameters

Component	Geometry	Size
Chassis	Box	0.4 x 0.2 x 0.1
Back and Front Caster	Sphere	0.0499(radius)
Left and Right Wheels	Cylinder	0.1(radius) x 0.05(length)
Camera	Link Origin	[0,0,0,0,0,0]
	Shape Size	0.4 x 0.2 x 0.1 (Box)
	Joint Origin	[0.2,0,0,0,0,0]
	Parent Link	Chassis
	Child Link	Camera
Hokuyo	Link Origin	[0,0,0,0,0,0]
	Shape Size	0.1 x 0.1 x 0.1 (Box)
	Joint Origin	[0.15,0,0.1,0,0,0]
	Parent Link	Chassis
	Child Link	Hokuyo

to the robot being unable to move at all as it determined itself to be surrounded by obstacles with no space to move. Regarding the localization itself, the primary issue was the rapid increase in uncertainty of the robot's pose when turning without any forward movement. In order to minimize this, the angular update frequency of the AMCL node was set to be much lower in order to avoid confusing the robot too much during turns. Forward and backward movements were the only types of movement which decreased uncertainty of the robot's pose, so the frequency of these updates were increased.

TABLE 2
Benchmark Node Parameters

AMCL Node Parameters	min particles	10
	max particles	1000
	update min d	0.01
	update min a	$\frac{3\pi}{2}$
	resample interval	10
	use map topic	True
Local Planner Parameters	holonomic robot	False
	meter scoring	True
	pdist scale	4
	occdist scale	0.3
	yaw goal tolerance	3.14
Common Cost Map Parameters	obstacle range	3
	raytrace range	2
	transform tolerance	1
	inflation radius	0.5
Local Cost Map Parameters	update frequency	10.0
	publish frequency	1.0
	width	5.0
	height	5.0
	resolution	0.05
Global Cost Map Parameters	update frequency	1.0
	publish frequency	1.0
	width	40.0
	height	40.0
	resolution	0.1

3.3 Personal Model

3.3.1 Model design

The updated model attempted to solve two issues with the benchmark: a narrow field of view(FOV) of the laser scanner and the tendency of corners trapping the robot against walls. The resulting robot added an additional hokuyo laser scanner to the rear of the chassis as well as adding a "head" to the robot itself to give it a uniform collision perimeter.

TABLE 3
Benchmark Model Parameters

Component	Geometry	Size
Chassis	Box	0.4 x 0.2 x 0.15
	Link Origin	[0,0,0.05,0,0,0]
Chassis Head	Cylinder	0.23(radius), 0.1(length)
	Link Origin	[0,0,0.15,0,0,0]
Back and Front Caster	Sphere	0.0499(radius)
Left and Right Wheels	Cylinder	0.1(radius) x 0.05(length)
Camera	Link Origin	[0,0,0,0,0,0]
	Shape Size	0.4 x 0.2 x 0.1 (Box)
	Joint Origin	[0.23,0,0.15,0,0,0]
	Parent Link	Chassis
	Child Link	Camera
Hokuyo	Link Origin	[0,0,0,0,0,0]
	Shape Size	0.1 x 0.1 x 0.1 (Box)
	Joint Origin	[0.15,0,0.25,0,0,0]
	Parent Link	Chassis
	Child Link	Hokuyo
Hokuyo2	Link Origin	[0,0,0,0,0,0]
	Shape Size	0.1 x 0.1 x 0.1 (Box)
	Joint Origin	[0.15,0,0.25,0,0,0]
	Parent Link	Chassis
	Child Link	Hokuyo2

3.3.2 Packages Used

The same packages were used as the benchmark model

3.3.3 Parameters

While the majority of the parameters were identical to the benchmark, the only parameter than needed to be changed was the inflation radius of the Common Cost Map. Since the new model was larger than the benchmark, this parameter was too large and the robot found itself in the middle of what it classified as a wall from the start.

4 RESULTS

Present an unbiased view of your robot's performance and justify your stance with facts. Do the localization results look reasonable? What is the duration for the particle filters to converge? How long does it take for the robot to reach the goal? Does it follow a smooth path to the goal? Does it have unexpected behavior in the process?
For demonstrating your results, it is incredibly useful to have some watermarked charts, tables, and/or graphs for the reader to review. This makes ingesting the information quicker and easier.

4.1 Localization Results

Neither one of these models outperformed the other in all aspects. While each robot successfully reached the goal destination, its different attributes led them to different issues.

4.1.1 Benchmark

The most common issue with the benchmark model was its tendency to "look back". This is caused by its uncertainty of it's environment and the possibility of a quicker path in this direction. While this did not affect it's overall performance, this hindered the overall time in which the robot reached its goal.

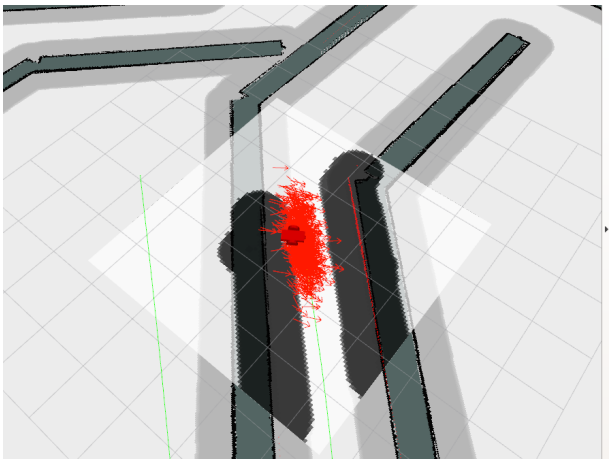

Fig. 5. Benchmark "Look Back" Issue

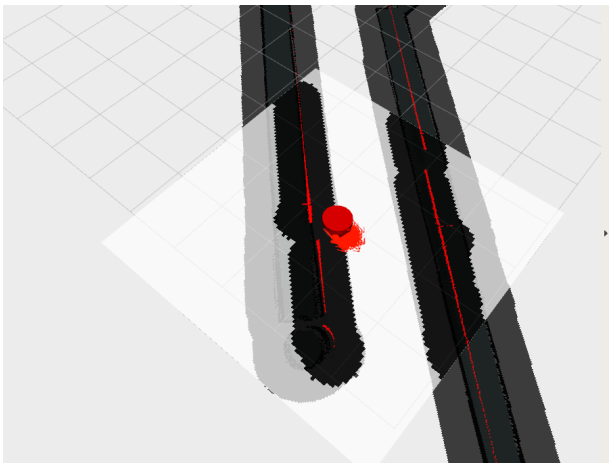

Fig. 6. Updated Model "Stuck" Issue

4.1.2 Student

The updated model encountered an entirely different issue. For the inflation radius parameter, the value needed to stay large enough to create space from the wall, yet small enough not to "trap" the robot in its initial pose. The larger, updated model was inconsistent in that a value any smaller than its current value would lead to it stopping when going around walls. Any larger value led to it being stuck initially. Unfortunately the final value is not perfect and occasionally finds itself trapped while following its path.

4.2 Technical Comparison

In terms of the robot's ability to comprehend the environment and respond to it efficiently, the updated model learned faster with the additional laser scan sensor. However, its size was its primary hindrance regarding consistency. In this aspect the benchmark model was superior.

5 Discussion

While the final results do not display perfect results, there were some underlying parameters of the project which prevented the updated model from performing as consistently as the benchmark model, notably its initial pose being too close to one wall. While this is a parameter that can easily be altered, there were other factors which resulted in clear performance boosts. One such alteration was the addition of the extra laser scanner. This enabled the robot to have a 360 degree view of its environment at all times, never having to turn for the sake of gathering information.

In practical use, this model paired with this sensor will have a much more difficult time localizing itself in empty spaces. The robot is able to localize itself with respect to obvious obstacles that are captured with the laser scanner. However without detecting any obstacles, the robot will quickly become uncertain of its pose.

6 Conclusion / Future Work

The robot model has a significant impact on the final results of the algorithm and therefore needs to be optimized to fit its environment. In the particular environment used here, the small size of the benchmark along with the second laser scanner would have performed the most optimally.

When applying Monte Carlo Localization to a commercial product, it will seem to be limited to industrial use mainly due to its need for a predefined map of the environment and it's limitation of navigating flat 2d areas. Without an accompanying mapping algorithm, this algorithm will fail to localize itself to a global map.

Regarding hardware deployment, it is very plausible to run this algorithm on a variety of devices due to its ability to control memory usage. A real-life implementation would simply need physical sensors, odometer, and a map of the environemnt to localize itself within it.