Where Am I?

Carl Sosa-Rivera

Abstract—This project aims to successfully solve the robot localization problem by applying the Monte Carlo Algorithm on a Gazebo/Rviz simulated environment. Using ROS as the framework, a two-wheeled robot was designed and developed to traverse through a provided map, and to be compared with a provided baseline robot.The developed robot, leverages the navigation stack move base and amcl packages to process and handle the localization of the robot. By tuning the different costmap and base local planner parameters, the robot accurately localized itself and managed to successfully navigate to the designated goal.

Index Terms—Robotics, Localization, Monte Carlo Algorithm, Udacity, IEEEtran.

1 Introduction

In the field of robotics, localization is one of the main capabilities a robot should possess, this allows the robot to identify its location and be able to move to a certain goal with precision. In localization, there are three main challenges: local localization, global localization, and the kidnapped robot problems. In the local localization, also known as position tracking, the robot knows its initial pose and the problem is to keep track of the robot's pose as it moves. In global localization, the robot's initial pose is unknown, and the robot tries to determine its pose relative to the ground truth map; the uncertainty for this type of localization is greater than for local localization. In the kidnapped robot problem, just like in global localization, the robot's initial pose is unknown, however the robot maybe kidnapped at any time and moved to another location of the map. Solving for the latter challenge also helps the robot recover in the event that it loses track of its pose, due to either being moved to other positions, or even when the robot miscalculates its pose.

In this project, the global localization challenge will be tackled, via ROS packages, to help the two robots traverse a known map (see Fig. 1).

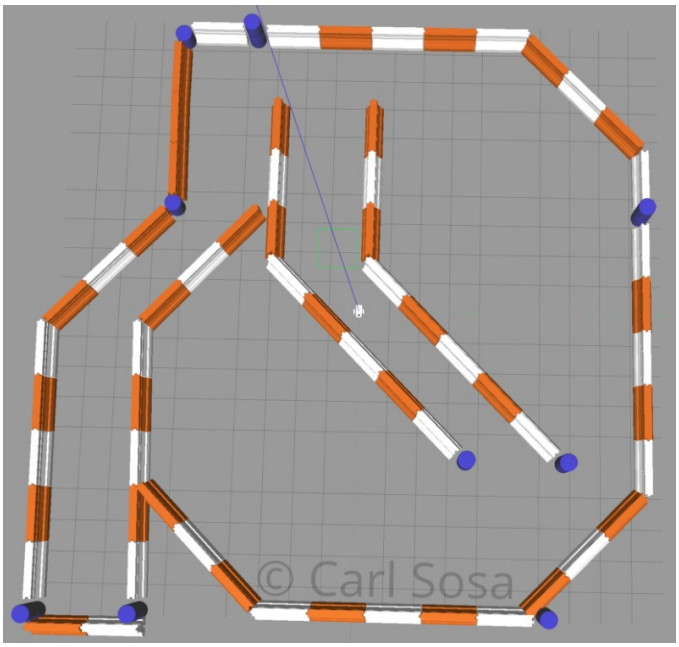

Fig. 1. Map to be traversed by the mobile robot.

2 Background

There are several approaches for a robot to perform localization. The four (4) most popular are Markov localization, Grid localization, Kalman Filter localization, and Monte Carlo Localization. However, for the purpose of this project, two approaches will be further discussed: Kalman Filter and Monte Carlo Localization (Particle filter).

2.1 Kalman Filters

The Kalman filter is, primarily an estimation algorithm, prominent in controls. Normally used to estimates the value of a variable in real time by collecting data. The most important aspect of the Kalman filter is that it can take data with a lot of uncertainty or noise in the measurements and provide an accurate estimate of the real value, efficiently and quickly. Using this algorithm, allows robots to perform sensor fusion, which is pulling and combining data from multiple sensors to obtain an accurate estimate of the measured value.

One of the drawbacks from the Kalman filter is that it operates under the assumption that motion and measurement models are linear and that the state can be represented by a unimodal Gaussian distribution. Normally, this would be very limiting given that most mobile robots will execute non-linear motions. Therefore, the Kalman filter cannot be applied to most robotics problems. However, by implementing an Extended Kalman Filter (EKF), one can linearize non-linear problems, in order to convert the result into a Gaussian distribution.

2.2 Particle Filters

The Monte Carlo Localization (MCL) also known as the Particle filter algorithm is the most popular localization algorithm in robotics. This algorithm uses particles to localize the robot. Each particle has a position and orientation,

representing the guess of where the robot is positioned. These particles are re-sampled every time the robot moves by sensing the environment through range-finder sensors, such as lidars, sonars, RGB-D cameras, among others. After a few iterations, these re-sampled particles eventually converge with the robots pose, allowing the robot to know its location and orientation. The MCL algorithm can be used for both Local and Global Localization problems, and is not limited to linear models.

2.3 Comparison / Contrast

Although you can estimate the pose of almost any robot with accurate sensors with the Extended Kalman Filter (EKF) algorithm, the Monte Carlo Localization (MCL) algorithm has many advantages over the EKF algorithm. The MCL algorithm is easier to program and setup than the EKF algorithm, making it better for robot implementation, debugging, and community support. The EKF algorithm is normally restricted by a Linear Gaussian state space assumption; whereas MCL can be used to represent any model. This makes the MCL algorithm helpful since the world cannot always be modeled by Gaussian distributions. Furthermore, with the MCL algorithm you can control the computation memory and resolution of the solution by tuning the particle quantity distributed randomly around the map (See Fig. 2).

	MCL	EKF
Measurements	Raw Measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency(memory)	✓	✓✓
Efficiency(time)	✓	✓✓
Ease of Implementation	✓✓	✓
Resolution	✓	✓✓
Robustness	✓✓	x
Memory & Resolution Control	Yes	No
Global Localization	Yes	No
State Space	Multimodel Discrete	Unimodal Continuous

Fig. 2. EKF VS MCL

3 Simulations

In order to simulate mobile robot localization, robots have to be built in the ROS framework, in a way that can be launched and displayed on Gazebo and Rviz. Therefore a ROS package was created to launch the robot and its world configuration. Additionally, the Navigation Stack and AMCL packages were installed on the catkin workspace, to provide the robots with localization capabilities, along with a navigation goal C++ node.

The simulation was in part developed in Udacitys VM workspace, and later finished and tuned in a Jetson TX2 development kit. This helped out reducing lag and performance issues encountered on the VM workspace.

3.1 Achievements

Both the baseline and the custom robots were able to localize themselves and reach their goal by tuning costmap and base local planner parameters. The movement was driven by a C++ node that provided navigation goal. The tuning process was a long and iterative one, but ultimately it enabled the robots with the localization capability.

3.2 Benchmark Model: udacity bot

3.2.1 Model design

The provided benchmark mobile robot, named udacity bot, was developed by creating a Unified Robot Description Format file, also known as a URDF file. In this file, the size and geometry of the robot is designated and set to have physical configuration such as inertial and collision parameters.

The udacity bot has a rectangular cube shape chassis with a size of [0.4, 0.2, 0.1], two casters to provide balance with spherical shape with a radius of 0.0499, and two 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 connected to the chassis via continuous joints, meaning they are free to rotate around the joint axis. The udacity bot is also equipped with two on-board sensors, a camera and a laser range-finder. (See Fig. 3 for udacity bot visual. See TABLE 1 for detailed specifications)

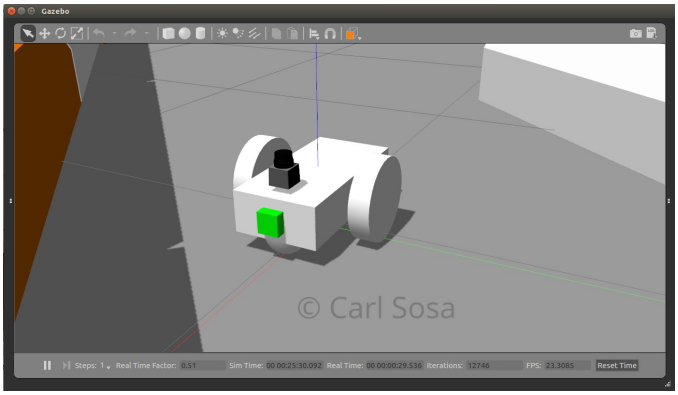

Fig. 3. Udacity Bot Visual

3.2.2 Packages Used

In order to launch the mobile robots, a ROS package was created. The robot package structure was designed as shown below. Udacity Bot Package:

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

This robot package, along with the Navigation Stack and AMCL packages were crucial for a complete simulation of a mobile robot performing and successfully solving the localization problem.

TABLE 1
Udacity Bot Setup Specifications

Udacity Bot Body		
Part	Geometry	Size
Chassis	Cube	0.4 x 0.2 x 0.1
Back and Front Casters	Sphere	0.0499(radius)
Left and Right wheels	Cylinders	0.1 (radius), 0.05(length)
Udacity Bot Equipment		
Camera Sensor	Link origin	[0,0,0,0,0,0]
	Shape-Size	Box - 0.05 x 0.05 x 0.05
	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 x 0.1 x 0.1
	Joint Origin	[0.15,0,0.1,0,0,0]
	Parent Link	chassis
	Child Link	hokuyo

3.2.3 Parameters

To obtain most accurate localization results, several parameters were added, tested and tuned. The parameter values obtained for the udacity bot were tuned in an iterative process to see what values worked best. In the AMCL node, the most prominent parameters were the min particles and max particles which were set to 10 and 200, respectively. These, tuned the accuracy of the localization process. An increase of particles would mean an increase in accuracy, however, it would also have impact on computational efficiency, making processing slower. (See Table ?? for detailed parameter specifications) Several other parameters were tuned in the different config files. The transform tolerance, inflation radius, robot radius, and obstacle range were obtained after several iterations of testing and tuning. Increasing the inflation radius would have an impact on the costmap while detecting obstacles and their distance related to the robot. Whereas robot radius represents the radius of the robot as it relates to its environment. Meaning that having a lower robot radius value would increase chances of it getting stuck around obstacles, and a higher value would prompt the robot to think it was bigger than the space it had to pass by. Albeit briefly summarized here, the ROS Wiki Page) provides in-depth descriptions of each and every one of the parameters used for this project as well as other parameters that can be explored further. See TABLE 1 and TABLE 3 for detailed parameters specifications used for the udacity bot.

TABLE 2
Global and Local Costmap Parameters: Udacity Bot

Costmap Parameters		
Parameter	Global	Local
global frame	map	odom
robot base frame	robot footprint	robot footprint
update frequency	15.0	15.0
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

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

3.3 Personal Model

3.3.1 Model design

The personal mobile robot, named csosa bot, was developed by creating a new Unified Robot Description Format file (URDF) all within the same udacity bot package with remapped subscribers and publishers. In this file, the size and geometry of the robot is designated and set to have physical configuration such as inertial and collision parameters. The csosa bot has a cylindrical shape chassis with a radius of 0.2 and length of 0.1, two casters to provide balance with spherical shape with a radius of 0.0499, and two 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 connected to the chassis via continuous joints, meaning they are free to rotate around the joint axis. The udacity bot is also equipped with two on-board sensors, a camera and a laser range-finder. (See Fig. 4 for csosa bot visual. See TABLE 4 for detailed specifications)

3.3.2 Packages Used

Just like the udacity bot, the csosa bot uses the same packages for simulation and to perform successful localization. As it is hosted within the udacity bot package, for the csosa package no ROS package had to be created; it leverages most of the udacity bot package.

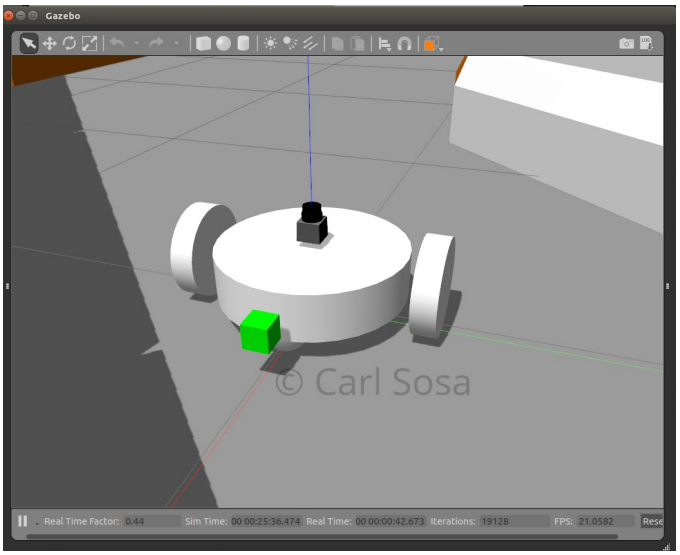

Fig. 4. Csosa Bot Visual

TABLE 4
Csosa Bot Setup Specifications

Csosa Bot Body		
Part	Geometry	Size
Chassis	Cylindrical	0.2 (radius), 0.1(length)
Back and Front Casters	Sphere	0.0499(radius)
Left and Right wheels	Cylinders	0.1 (radius), 0.05(length)
Csosa Bot Equipment		
Camera Sensor	Link origin Shape-Size Joint Origin Parent Link Child Link	[0,0,0,0,0,0] Box - 0.05 x 0.05 x 0.05 [0.25,0,0,0,0,0] chassis camera
Hokuyo Sensor	Link origin Shape-Size Joint Origin Parent Link Child Link	[0,0,0,0,0,0] Box - 0.1 x 0.1 x 0.1 [0.0,0,0.1,0,0,0] chassis hokuyo

3.3.3 Parameters

For the csosa bot, most of the parameters used for the udacity bot were reused in order to successfully solve the localization problem. However, the costmap common parameters used was slightly different as the body and shape of the csosa bot was dramatically different. (See TABLE 5 to for the different parameters)

TABLE 5
Costmap Common Parameters: Csosa 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, testing, and tuning parameters, the results provided at the end were definitely valuable as 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 some taking a bit longer some taking a bit less to converge. Overall, both robots traversed adequately with minimal interruptions, albeit, sometimes collision with obstacles was observed, specially 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: Udacity-Bot

At the start of the simulation for the udacity bot one can see the particles spread out around the vicinity of the robot (See Fig. 5).

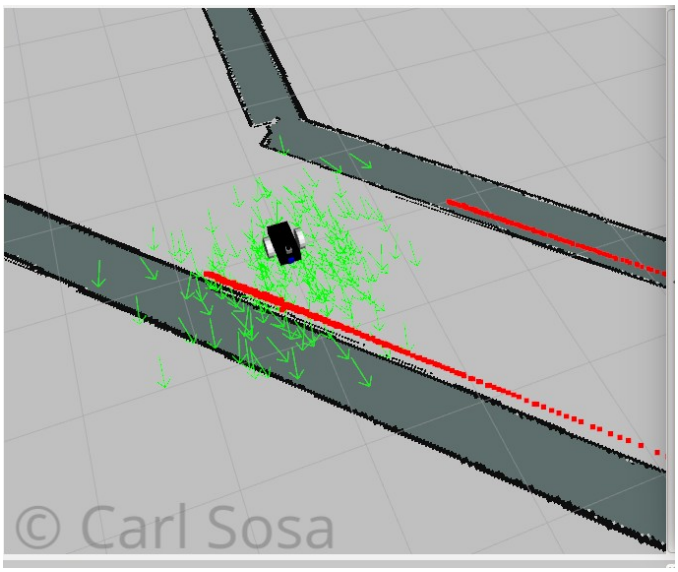

Fig. 5. Udacity Bot: Starting Simulation

After, a few iterative steps, one can see the particles start to converge on the robot (See Fig. 6).

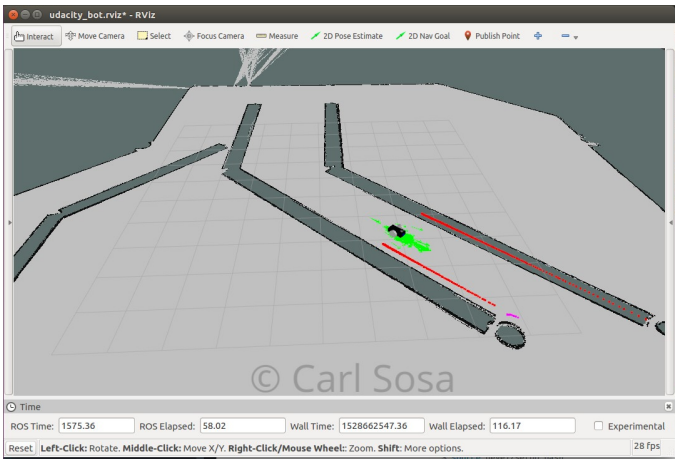

Fig. 6. Udacity Bot: Particles start to converge

The udacity bot reaches the designated goal in the end (See Fig. 7).

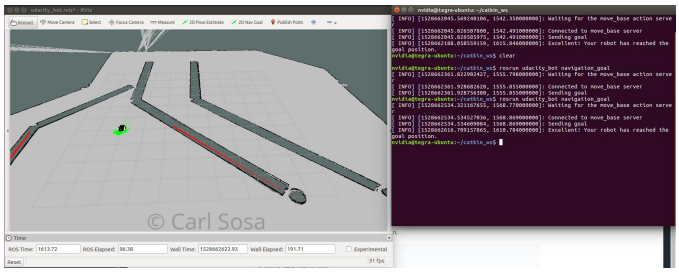

Fig. 7. Udacity Bot: Reaching Designated Goal

4.1.2 *Personal Model: Csosa-Bot*

At the start of the simulation for the csosa bot one can see the particles spread out around the vicinity of the robot (See Fig. 8).

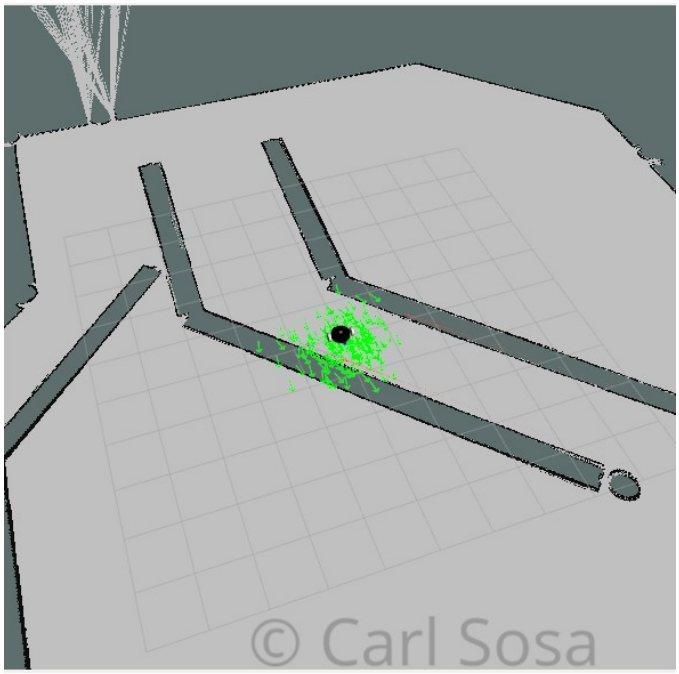

Fig. 8. Csosa Bot: Starting Simulation

After, a few iterative steps, one can see the particles start to converge on the robot (See Fig. 9).

The csosa bot reaches the designated goal in the end (See Fig. 10).

4.2 Technical Comparison

Given the body difference between the two mobile robots, the turning and steering performance seems to be slightly different. Making the udacity bot have slightly better performance traversing through the map. However, both robots were able to reach the desired location in less than 2 minutes.

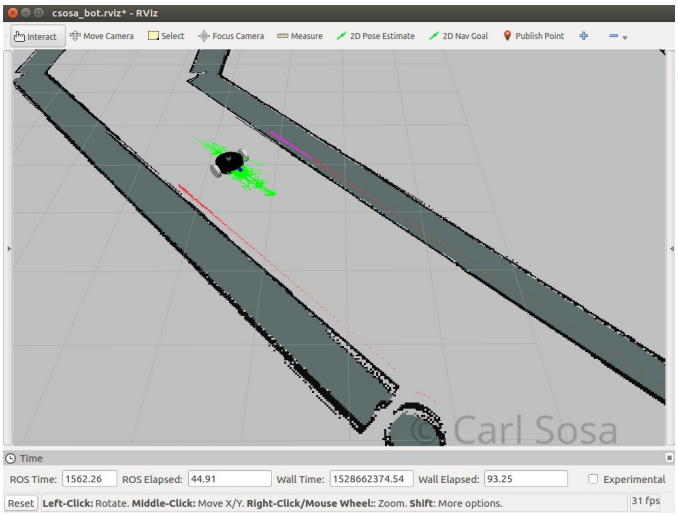

Fig. 9. Csosa Bot: Particles start to converge

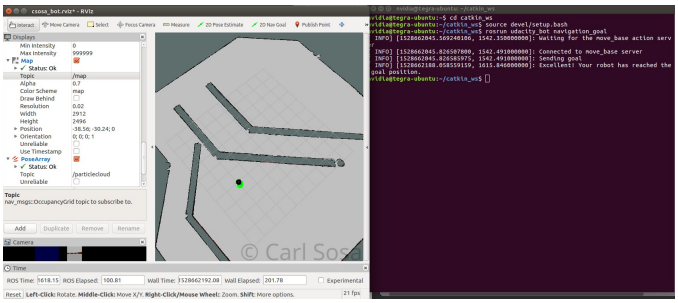

Fig. 10. Csosa Bot: Reaching Designated Goal

5 Discussion

- Overall, the udacity bot benchmark model, was the most effective, possibly theres a physical configuration that allows it to traverse faster through the map. However, the personal model csosa bot performs well throughout its trajectory to the goal. In the end, both models reached the designated goal successfully and solved the localization problem.
- It can be understood that these robots would be able to solve the kidnapped robot scenario. Due to the fact that the particles can be initialized randomly throughout the map, allows the robots to localize themselves after re-sampling of the particles. In the end, they would eventually converge on the robot, albeit slightly slower but nonetheless end up solving the problem for multiple scenarios.
- The MCL/AMCL would probably work best in known, closed spaces. Understanding that the particles have to be spread out throughout the environment. It seems very computationally intensive to have that many particles spread out if the space is an open one or a rather immense space.

6 Conclusion / Future Work

After all the iterations and tuning, it can be concluded that AMCL is a powerful tool to use when trying to solve the localization problem for robots. The fact that its very easy

to implement and the immense community support for debugging purposes due to it popularity, this algorithm will probably be the first choice when developing autonomous and location-aware robots. In future work, implementing AMCL for 3-Dimensional localization problems would be taking robot localization to next step.

This would be used for drones, and arm actuator robots that probably need to realize the end-effector location and orientation to reach specific goals with a greater accuracy. This could probably be done with a 3D Laser range-finder instead of the 2D Hokuyo sensor.