Where Am I ?

Bruno Santos

Abstract—In this project, the main goal is building a mobile robot for simulated tasks using Gazebo as simulated environment. For this purpose, it was developed a ROS package that launches a custom robot model in a Gazebo world and utilizes packages like AMCL and the Navigation Stack. Relying on the previous packages, parameters such obstacle range and inflation radius will be tuned to obtain a better localization in the provided map.

Index Terms—Robot,Monte Carlo, Localization.

1 Introduction

Localization in robotics is a hot research topic since it is the step-stone to autonomous mobile robots. The most used localization algorithms are Monte Carlo and Kalman Filter.The main difference between them is that Kalman Filter assume measurement distributions to measurement variables.In this simple 2D problem, it is going to be used a version of Monte Carlo localization algorithm called Adaptative Monte Carlo. For this project, we really on loaded map to navigation, and that is the reason to use Monte Carlo localization method.

2 Background

There are two main type of localization algorithms:

- Kalman Filter and its extensions
- Monte Carlo and its extensions

The Kalman Filter has long been regarded as the optimal solution to many tracking and data prediction tasks. Its use in the analysis of visual motion has been documented frequently. It is based on measurement, update and prediction of certain state variables such as position, speed,etc. It usually assume a gaussian state variable distribution. Monte Carlo localization algorithm relies on a set of particles (similarly to particle filter) and a offline map to find the current robot pose. It iterates several times and base on an estimated error the particles converge with higher probability will be the robot current pose. The Adaptative Monte Carlo algorithm was chosen due to two main factors:

- the robot pose is in 2D
- The variable number of particles allow to estimate more precisely the robot pose

In addition, localization is a difficult problem because relies on a good quality map as well less noisy measurements.

2.1 Kalman Filters

The Kalman Filter has long been regarded as the optimal solution to many tracking and data prediction tasks. Its use in the analysis of visual motion has been documented frequently. It is based on measurement, update and prediction of certain state variables such as position, speed,etc. It usually assume a gaussian state variable distribution, which is a drawback from this algorithm. In addition, baseline Kalman Filter relies on linear transformations, which is also limitation. For that reason, it was introduced Extended Kalman Filter which uses Taylor approximation to describe the transformation between state and measurements.

2.2 Particle Filters

Particle Filter algorithm relies on a set of particles and a offline map to find the current robot pose. It iterates several times and base on an estimated error the particles converge with higher probability will be the robot current pose. The great advantage is the non assumption of a distribution for approximating the state variables.

2.3 Comparison / Contrast

In this section it will be explained the difference between Kalman Filter and Particle Filter like algorithms such as Monte Carlo.

- Both algorithms have measurement and update stages
- Both algorithms can assum gaussian distributions to variables despite not being mandatory to Particle Filter one.
- Particle filters rely on a set particles and probability weight update to find the robot pose
- Kalman filter assume a gaussian distribution and perform linear approximation transforms between measurements and state variable. In addition, the update relies on the covariance of gaussian which is the variable distribution uncertainty.
- As an error approximation, Particle Filter rely on euclidean distance between robot measurements and particles ones.

3 Simulations

3.1 Achievements

In the project,I changed the mass of robot as well the wheels. Those changes lead to an improvement in turning around obstacles in U-shape. In addition, the robot configuration designed by me speed up the time performance by 2 doing the same path.

3.2 Benchmark Model

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 figure 1 for udacity bot visual. See table 1 for detailed specifications).

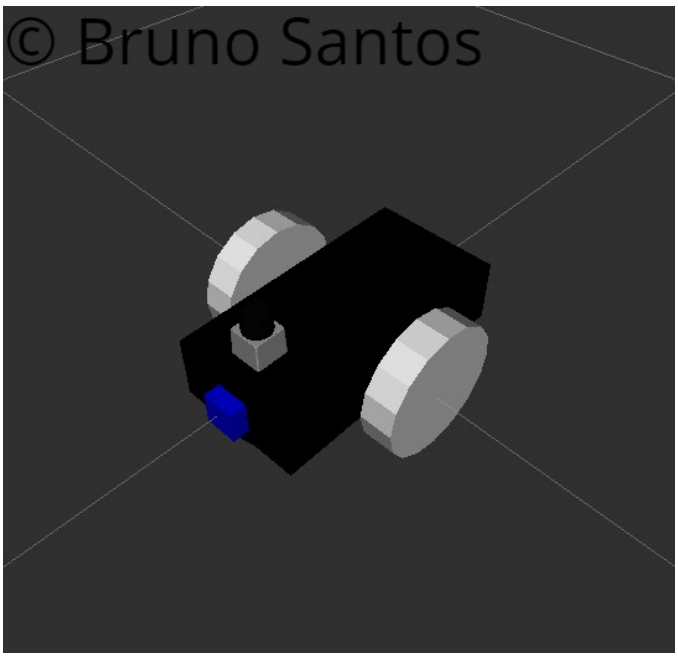

Fig. 1. Udacity Robot design

TABLE 1
Udacity bot setup specifications.

Part	Geometry	Size
Chassis	Cube	0.4x0.2x0.1
Back and front casters	Sphere	0.049 (radius)
Left and Righ wheels	Cylinders	0.1 (radius) ,0.05(length)
Camera Sensor	Link origin Shape-Size Joint Origin Parent Link Child link	0,0,0,0,0,0 Box - 0.05X0.05X0.05 0.2,0,0,0,0,0 chassis camera
Hokuyo Sensor	Link origin Shape-Size Joint Origin Parent Link Child link	0,0,0,0,0,0 Box - 0.1X0.1X0.1 0.2,0,0,0,0,0 chassis hokuyo

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.

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 2 , TABLE 3, TABLE 4 and TABLE 5, for detailed parameters specifications used for the udacity bot.

TABLE 2
Global and Local Costmap Parameters: Udacity Bot

Parameter	Global	Local
global frame	map	odom
robot base frame	robot footprint	robotfootprint
update frequency	15	15
publish frequency	15	15
width	20.	5.
height	20.	5.0
resolution	0.05	0.05
static m	true	false
rolling window	false	true

TABLE 3
AMCL Parameters: Udacity bot

minparticles	10
maxparticles	200
initialposex	0
initialposey	0
initialposez	0
odommodeltype	diff-corrected
odomalpha1	0.010
odomalpha2	0.010
odomalpha3	0.010
odomalpha4	0.010

TABLE 4
Cost Common parameters: Udacity bot

obstaclerange	2.5
raytracerange	3.0
transformtolerance	0.3
robotradius	0.25
inflationradius	0.5

TABLE 5
Base Local Planner Parameters: Udacity bot

holonomicrobot	false
yawgoaltolerance	0.05
xygoaltolerance	0.1
simtime	1.0
meterscoring	true
pdistscale	0.5
gdistscale	1.0
maxvelx	0.5
maxvely	0.1
maxveltheta	5.0
acclimtheta	5.0
acclimx	2.0
acclimy	5.0
controllerfrequency	15.0

3.3 Personal Model

3.3.1 Model design

The personal mobile robot, named brunosantos 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 brunosantos bot has a cylindrical shape chassis with a radius of 0.15 and length of 0.05, 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. 2 for brunosantos bot visual. See TABLE **??** for detailed specifications).

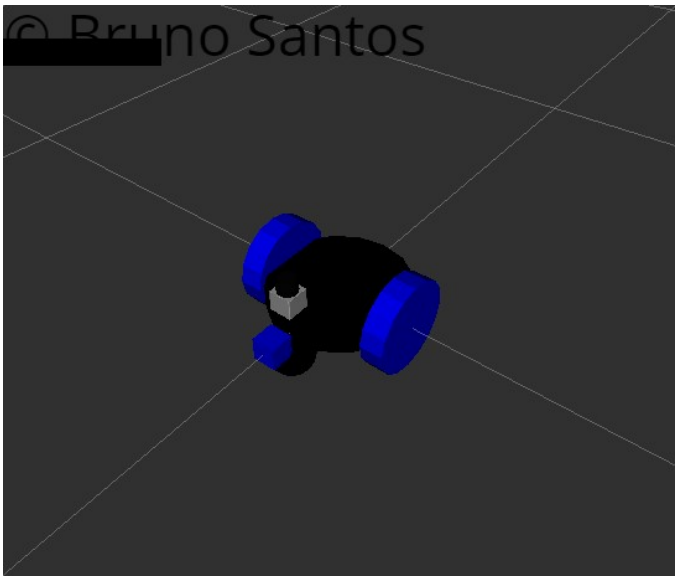

Fig. 2. Personal Robot design

3.3.2 Packages Used

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

3.3.3 Parameters

For the brunosantos 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 brunosantos bot was dramatically different. (See TABLE 7 for the different parameters)

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

At the start of the simulation for the udacity bot one can see the particles spread out around the vicinity of the robot (See figure 3).

The udacity bot reaches the designated goal in the end (see figure 4).

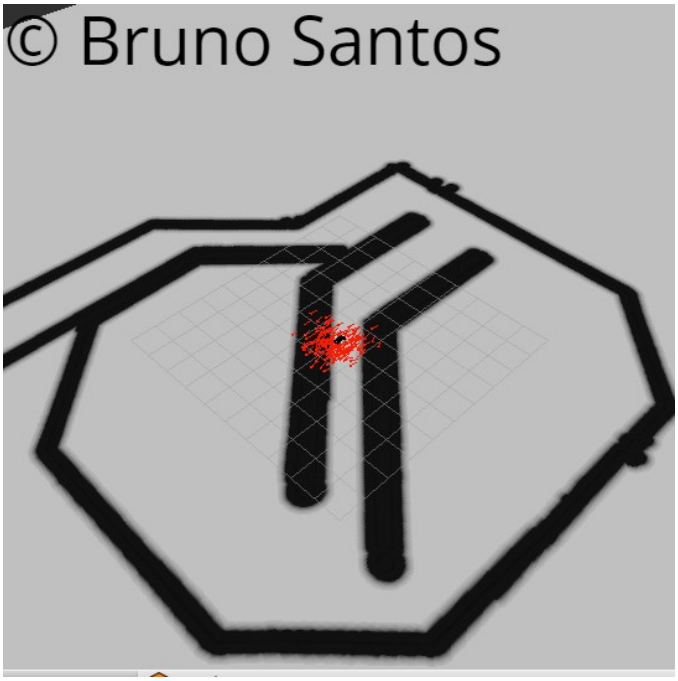

Fig. 3. Start Robot pose.

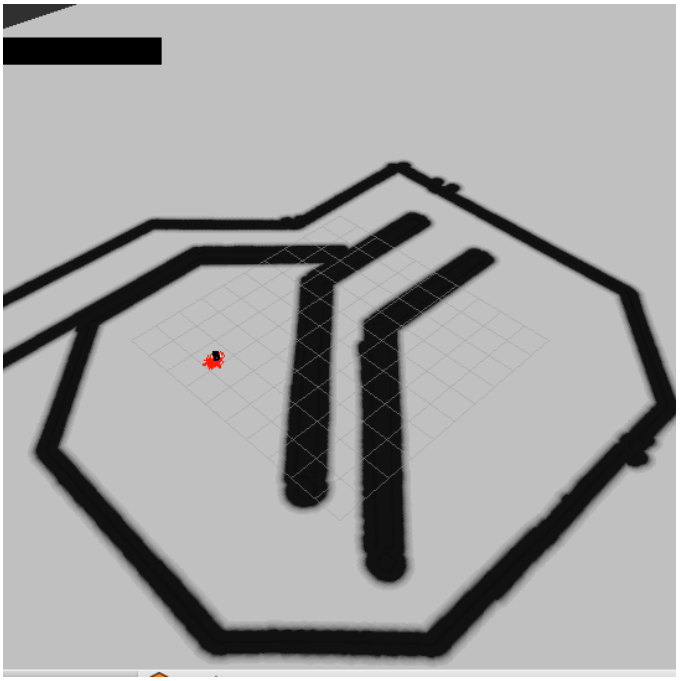

Fig. 4. Final Robot pose.

TABLE 6
Udacity bot setup specifications.

Part	Geometry	Size
Chassis	Cylindrical	0.15 (radius) , 0.05(length)
Back and front casters	Sphere	0.049 (radius)
Left and Righ wheels	Cylinders	0.1 (radius) ,0.05(length)
Camera Sensor	Link origin Shape-Size Joint Origin Parent Link Child link	0,0,0,0,0,0 Box - 0.05X0.05X0.05 0.2,0,0,0,0,0 chassis camera
Hokuyo Sensor	Link origin Shape-Size Joint Origin Parent Link Child link	0,0,0,0,0,0 Box - 0.1X0.1X0.1 0.2,0,0,0,0,0 chassis hokuyo

TABLE 7
Cost Common parameters: Personal bot

obstaclerange	3.5
raytracerange	4.0
transformtolerance	0.3
robotradius	0.35
inflationradius	0.5

4.1.2 Personal Model: brunosantos-bot

At the start of the simulation for the brunosantos bot one can see the particles spread out around the vicinity of the robot (See Fig. 5).
The brunosantos bot reaches the designated goal in the end

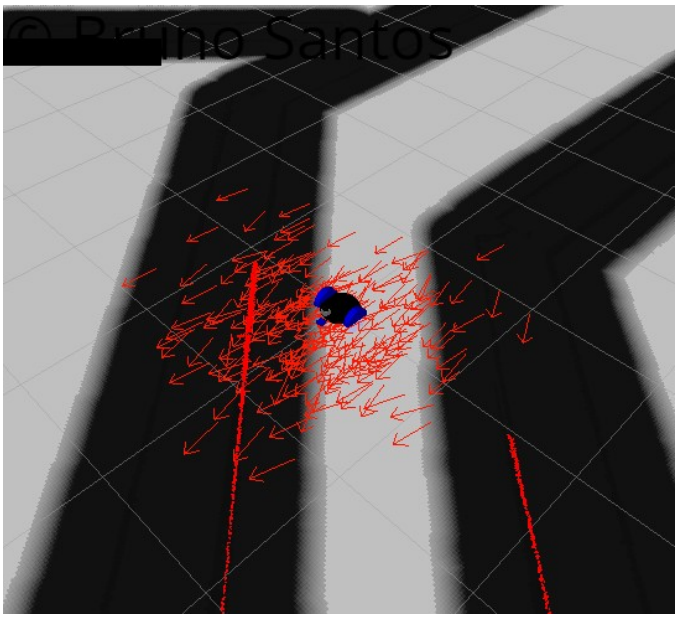

Fig. 5. Start brunosantos Robot pose.

(see figure 6).

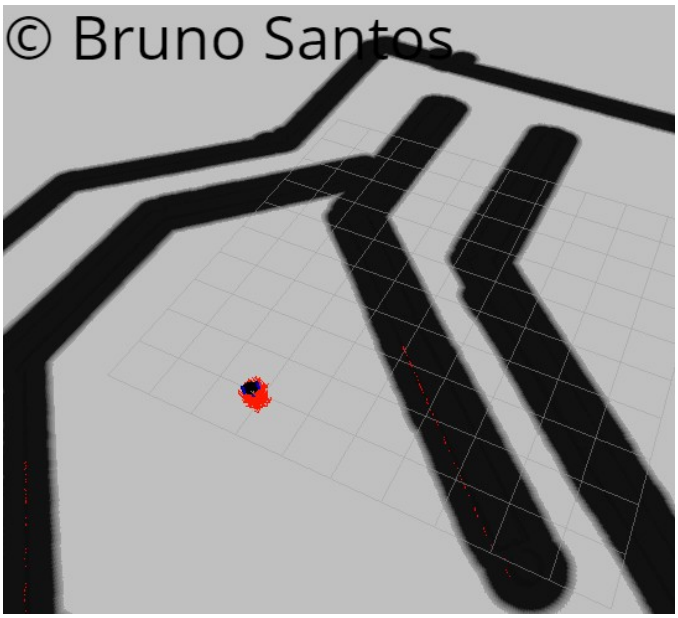

Fig. 6. Final brunosantos Robot pose.

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 brunosantos bot have slightly better performance traversing through the map. However, both robots were able to reach the desired location in less than 2 minutes.

5 Discussion

- Overall, the brunosantos benchmark model, was the most effective, possibly theres a physical configuration that allows it to traverse faster through the map. However, the personal model udacity 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 if the space is an open one or a rather immense space.
- If the number particles increased we would have most likely a better robot pose estimation, although the computational costs would downgrade the time performance. There should a balance between accuracy and computation costs.

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.