Mobile Robot Localization: Where Am I?

Carlos E. Sanoja

Abstract—By using ROS as framework, a mobile robot was developed to navigate through a provided map, and to be compared with a provided robot. To solve the problem of global localization, it is applied the Monte Carlo Algorithm on a simulated environment (Gazebo and Rviz) which involved a non-standard procedure. By adjusting the parameters of the planner, the robot accurately localized itself and managed to successfully navigate to the designated target.

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

1 Introduction

ROBOT localization is the process of determining where a mobile robot is located with respect to its environment. Localization is the knowledge of the robot's own location as an essential precursor to making decisions about future actions. In a typical robot localization scenario, a map of the environment is available and the robot is equipped with sensors that observe the environment as well as monitor its own motion. The localization problem then becomes one of estimating the robot position and orientation within the map using information gathered from these sensors.

The localization problem can be divided into two parts, pose tracking and global localization. In local Localization, the initial estimate of the robot pose is known. During the execution of a task, the robot must update this estimate using measurements from sensors. Using only sensors that measure relative movements, the error in the pose estimate increases over time as errors are accumulated. Therefore external sensors are needed to provide information about the absolute pose of the robot.

In Global Localization, the robots initial pose is unknown and the robot must determine its pose relative to the ground truth map. The amount of uncertainty is much greater than in Position Tracking, making it a much difficult problem. Robot localization techniques need to be able to deal with noisy observations and generate not only an estimate of the robot location but also a measure of the uncertainty of the location estimate.

In this project, ROS packages are utilized to solve the localization problem inside of a provided map (Fig.1). There are several modules created to accomplish the objectives. Those include building a mobile robot with sensors (Camera and laser), creating ROS packages to launch a robot model in Gazebo, Rviz and utilizes the most common packages like navigation stack and AMCL. Finally, the main goal is add and tune the parameters of this package to achieve the best possible localization results.

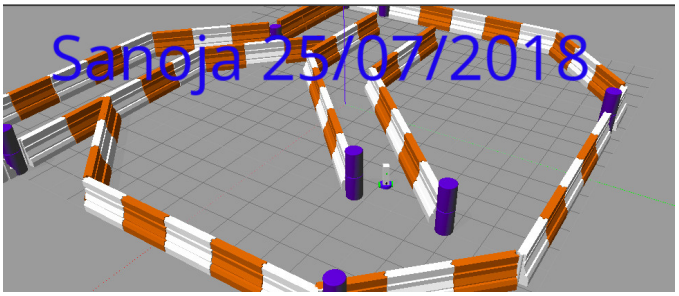

Fig. 1. Localization Map with a mobile robot.

2 Background

To localize itself, robot uses sensor data and movements inputs. Sensors are the fundamental robot input for the process of perception, and therefore the degree to which sensors can discriminate world state is critical. Sensor noise induces a limitation on the consistency of sensor readings in the same environmental state and, therefore, on the number of useful bits available from each sensor reading. Often, the source of sensor noise problems is that some environmental features are not captured by the robots representation and are thus overlooked.

Sensor noise reduces the useful information content of sensor readings. Clearly, the solution is to take multiple readings into account, employing temporal fusion or multi-sensor fusion to increase the overall information content of the robots inputs. The challenges of localization do not lie with sensor technologies alone. Just as robot sensors are noisy, limiting the information content of the signal, so robot effectors are also noisy. In particular, a single action taken by a mobile robot may have several different possible results, even though from the robots point of view the initial state before the action was taken is well-known. In short, mobile robot effectors introduce uncertainty about future state. Therefore the simple act of moving tends to increase the uncertainty of a mobile robot. There are, of course, exceptions.

Because of that, localization algorithms plays important role in calculating approximate position of the robot. The two most commonly approaches to localization are Extended Kalman Filters and Monte Carlo Localization.

2.1 Kalman Filters

The Kalman filter keeps track of the estimated state of the system and the variance or uncertainty of the estimate. The estimate is updated using a state transition model and

measurements (Fig. 2). Where X denotes the state and P the uncertainty. It follows from theory that the Kalman filter is

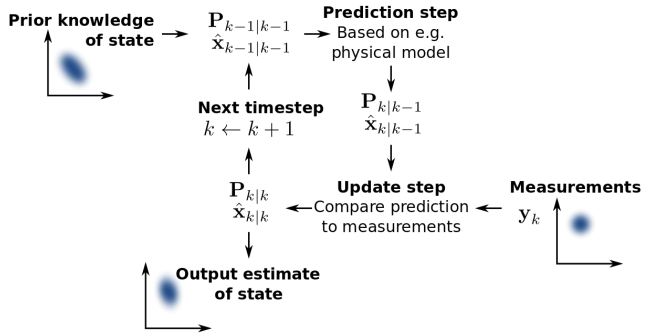

Fig. 2. Kalman Filter Map Algorithm.

the optimal linear filter in cases where

- the model perfectly matches the real system,
- the entering noise is white (uncorrelated) and
- the covariances of the noise are exactly known.

The Kalman Filter is applicable to problems with linear motion and measurement functions. This is limiting, as much of the real world is nonlinear. A nonlinear function can be used to update the mean of a function, but not the variance, as this would result in a non-Gaussian distribution which is much more computationally expensive to work with. To update the variance, the Extended Kalman Filter linearizes the nonlinear function f(x) over a small section and calls it F. This linearization, F, is then used to update the state's variance. The linear approximation can be obtained by using the first two terms of the Taylor Series of the function centered around the mean

2.2 Particle Filters

A particle is a discrete guess of where a robot may be located. Also, regularly, these particles are re-sampled with replacement from the distribution so that the particles consistent with the measurements survive. After successful localization, the particles are collected in a region of high probability of the robot.

Monte Carlo localization (MCL), also known as particle filter localization, is an algorithm for robots to localize using a particle filter. Given a map of the environment, the algorithm estimates the position and orientation of a robot as it moves and senses the environment. The algorithm uses a particle filter to represent the distribution of likely states, with each particle representing a possible state, (i.e., a hypothesis of where the robot is). The algorithm typically starts with a uniform random distribution of particles over the configuration space, meaning the robot has no information about where it is and assumes it is equally likely to be at any point in space. Whenever the robot moves, it shifts the particles to predict its new state after the movement. Whenever the robot senses something, the particles are re-sampled based on recursive Bayesian estimation (Fig. 3) (i.e., how well the actual sensed data correlate with the predicted state). Ultimately, the particles should converge towards the actual position of the robot.

MCL algorithm $(\chi_{t-1}, u_t, z_t, m)$

$\overline{\chi}_t = \chi_t = \emptyset$

for $p = 1$ *to* P

$\mu_t^{[p]} =$ ***Motion model***$(u_t, \mu_{t-1}^{[p]})$

$\mu_{t,belief}^{[p]} =$ ***Measurement model***(z_t, μ_t^p, m)

$\mu_t^{[p]} =$ ***Resampling***$(\{(\mu_t^{[p]}), \mu_{t,belief}^{[p]})|\ p = 1,\ldots, P\})$

end for

return χ_t

Fig. 3. MCL Algorithm.

2.3 Comparison / Contrast

Particle Filter presents many advantages over EKF. Some of the important ones are:

- MCL is easy to program compared to EKF
- MCL can approximate any other practical important distribution. This means MCL is unrestricted by a linear Gaussian states based assumption as is the case for EKF. This allows MCL to model the real world who can not always be modeled with Gaussian distributions.
- In MCL, you can control the computational memory by changing the number of particles distributed uniformly and randomly throughout the map.

3 Simulations

The simulations are done using Gazebo and RViz tools for visualization. As previously stated, two different robot models were built using URDF format. The first benchmark robot model was provided and is called udacity bot. The second robot model, explo bot is based on the first one.

3.1 Achievements

Both designed robots were capable to navigate to a specific goal position while localizing itself in the map. Also, the robot successfully reach the goal under the acceptance margin probabilities for position and orientation.

3.2 Benchmark Model

3.2.1 Model design

In Fig.4 is shown the udacity bot. A mobile robot created to navigate and localize itself through a know map. Keeping ROS as framework, the model was developed using the Unified Robot Description Format (URDF). Its structure consists in:

- main body which is 0.4x 0.2 x 0.1 box
- two wheels with 0.1 radius attached to left and right sides
- front and back spherical caster (with radius 0.049999) underneath the robot body for stability.

The robot is also equipped with a front facing camera mounted to the front of the robot and a Hokuyo LIDAR attached to the top.

Fig. 4. Udacity Bot.

3.2.2 Packages Used

1) The move base package: with this package is possible to define a goal position for the robot in the map, and the robot will navigate to that goal position. The move base package is a very powerful tool. It utilizes a costmap - where each part of the map is divided into which area is occupied, like walls or obstacles, and which area is unoccupied. As the robot moves around, a local costmap, in relation to the global costmap, keeps getting updated allowing the package to define a continuous path for the robot to move along.
What makes this package more remarkable is that it has some built-in corrective behaviors or maneuvers. Based on specific conditions, like detecting a particular obstacle or if the robot is stuck, it will navigate the robot around the obstacle or rotate the robot till it finds a clear path ahead. In Fig.5 are expose the topics (subscribers and publishers) needed to work.

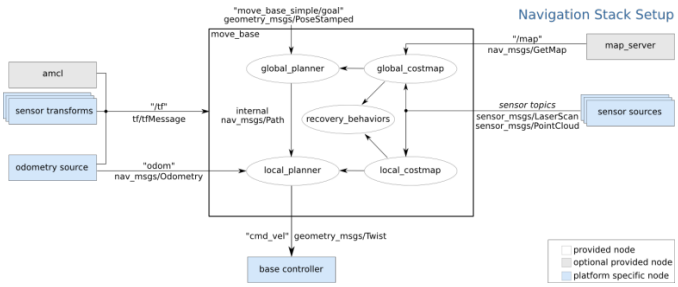

Fig. 5. Navigation package setup .

2) Adaptative Monte Carlo Localization: dynamically adjusts the number of particles over a period of time, as the robot navigates around in a map. This adaptive process offers a significant computational advantage over MCL. AMCL is a probabilistic localization system for a robot moving in 2D. In Fig. 6 are shown the main topics related to this package.

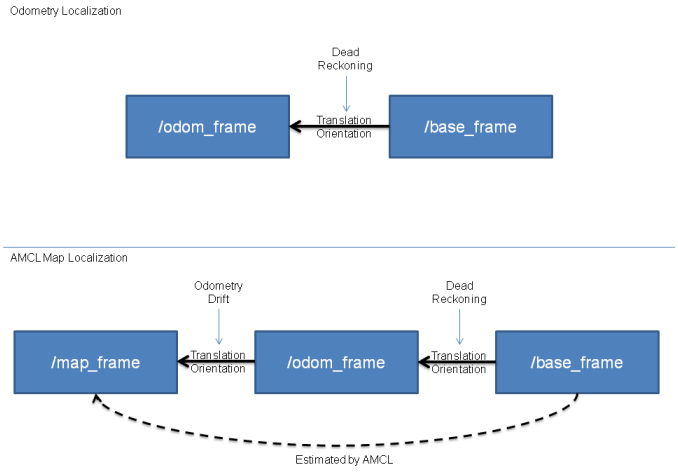

Fig. 6. AMCL package setup .

3.2.3 Parameters

Localization parameters in the AMCL node were described as follow:

- min_particles: 15.
- max_particles: 200.
  Initial values 100 and 5000 were too large for simulation environment and required larger computational cost.
- transform_tolerance: 0.35
  Time with which to post-date the transform that is published, to indicate that this transform is valid into the future. Initial value 0.0 was not enough to avoid mistakes while simulation.
- odom_alphaX: odometry model noise parameters odom_alpha1 to odom_alpha4 were set to the experimental values found it. By correcting setting those parameters AMCL algorithm improved and particles converged better.

Move base parameters in the configuration file are described as follow:

- update_frequency: 10.0
- publish_frequency: 5.0
- controller_frequency: 7.0
  All of the frequency parameters were tuned until warnings disappeared. These parameters also helped with following the local path.
- obstacle_range: 4.5
- raytrace_range: 9.0
- transform_tolerance: 0.3
- inflation_radius: 0.55

”Obstacle range” and ”raytrace range” set thresholds on obstacle information put into the costmap. The ”obstacle_range” parameter determines the maximum range sensor reading that will result in an obstacle being put into the costmap. The ”raytrace_range” parameter determines the range to which we will raytrace freespace given a sensor

reading. Setting it to 9.0 meters as we have above means that the robot will attempt to clear out space in front of it up to 9.0 meters away given a sensor reading.

The inflation radius should be set to the maximum distance from obstacles at which a cost should be incurred. For example, setting the inflation radius at 0.55 meters means that the robot will treat all paths that stay 0.55 meters or more away from obstacles as having equal obstacle cost.

The base_local_planner is responsible for computing velocity commands to send to the mobile base of the robot given a high-level plan.

To set the velocity of the robot:

- max_vel_x: 0.4
- min_vel_x: 0.1
- max_vel_theta: 1.0
- min_vel_theta: -1.0
- min_in_place_vel_theta: 0.35

To set the acceleration of the robot:

- acc_lim_x: 2.0
- acc_lim_y: 2.0
- acc_lim_theta: 2.8

Velocity and acceleration play an important role because the rate of change of the particles is different to this movement. For that, it is important to ensure that the movement of the robot do not interfere with the time of processing of the data to re-sampling the particles.

Finally, to know how much uncertainty is the final position and orientation of the robot, it is possible to set an margin of error tolerated when reaching the goal.

- yaw_goal_tolerance: 0.05
- xy_goal_tolerance: 0.05

3.3 Personal Model

3.3.1 Model design

In Fig.7 is shown the explo bot. A mobile robot created to navigate and localize itself through a know map. Also, the idea is to give him an humanoid looking to simulated an assistant mobile robot in house.

Fig. 7. Explo Bot.

Keeping ROS as framework, the model was developed using the Unified Robot Description Format (URDF). Its structure consists in:

- main body which is cylinder with radius: 0.2 and length: 0.1 .
- two wheels with 0.1 radius attached to left and right sides
- front and back spherical caster (with radius 0.049999) underneath the robot body for stability.
- Body. It is composed of 3 rectangular boxes.

The robot is also equipped with a facing camera mounted on the front of the robot's body and a Hokuyo LIDAR attached to the body. To avoid interference, the hokuyo is place under the arms to properly acquire the data from environment.

3.3.2 Packages Used

The packages used for navigation and path planning is the same as packages used for udacity bot.

3.3.3 Parameters

Localization parameters in the AMCL node were described as follow:

- min_particles: 15.
- max_particles: 200.
- transform_tolerance: 0.35
- odom_alphaX: odometry model noise parameters odom_alpha1 to odom_alpha4 were set to the experimental values found it. By correcting setting those parameters AMCL algorithm improved and particles converged better.

Move base parameters in the configuration file are described as follow:

- update_frequency: 10.0
- publish_frequency: 5.0
- controller_frequency: 7.0
  All of the frequency parameters were tuned until warnings disappeared. These parameters also helped with following the local path.
- obstacle_range: 4.5
- raytrace_range: 8.5
- transform_tolerance: 0.3
- inflation_radius: 0.6

To set the velocity of the robot:

- max_vel_x: 0.4
- min_vel_x: 0.1
- max_vel_theta: 1.0
- min_vel_theta: -1.0
- min_in_place_vel_theta: 0.4

To set the acceleration of the robot:

- acc_lim_x: 2.4
- acc_lim_y: 2.4
- acc_lim_theta: 3.4

Finally, to know how much uncertainty is the final position and orientation of the robot, it is possible to set an margin of error tolerated when reaching the goal.

- yaw_goal_tolerance: 0.05
- xy_goal_tolerance: 0.05

4 Results

Both robots completed satisfactorily the navigation to the goal location. It is important to note the measure of the errors. The arrows deployed are in some way, our descriptors to evaluate results. In Fig.8 is shown the final position for the udacity_bot. It's visible how the arrows are concentrated in a point (final position) and show the exactly same direction as was desired (RED arrow). By the other hand, In Fig.9

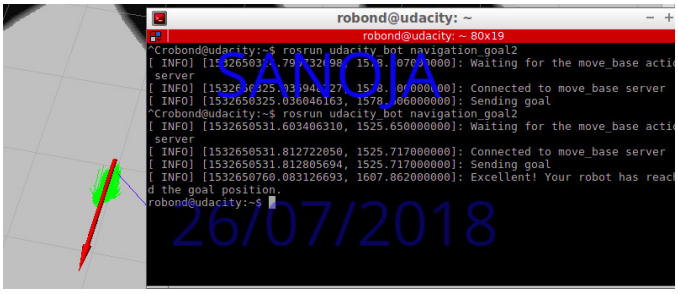

Fig. 8. Udacity_bot's arrows at final position.

is shown the final position for the explo_bot. This time, an error was detected, the final orientation of the robot have an error. However, the position was reach and the arrows are concentrated at the surroundings of the main arrow (RED arrow).

Fig. 9. Explo_bot's arrows at final position.

4.1 Localization Results

4.1.1 Benchmark

In Fig. 10 is deployed the udacity_bot when reaching the final position. The final time that took to navigate to the goal was 3.81 minutes.

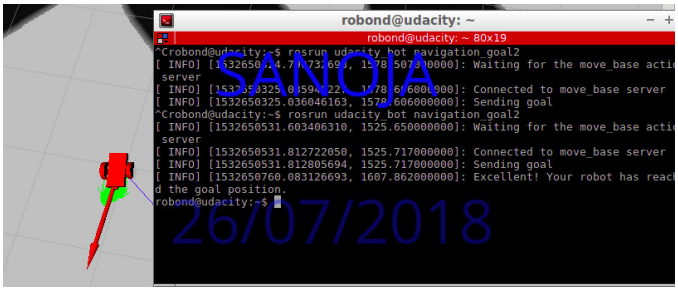

Fig. 10. Udacity_bot at final position.

4.1.2 Student

In Fig. 11 is deployed the explo_bot when reaching the final position. The final time that took to navigate to the goal was 5.18 minutes.

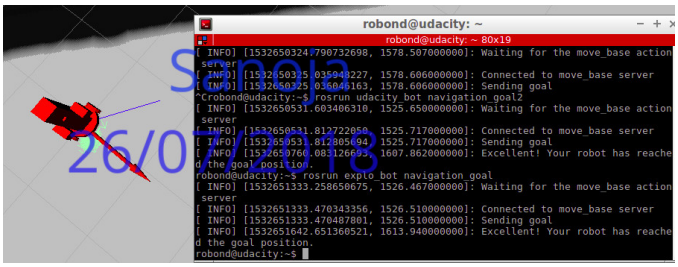

Fig. 11. Explo_bot at final position.

4.2 Technical Comparison

Given the body difference between the two mobile robots, the turning and the guidance performance seems to be slightly different. Making the udacity_bot have better performance.

In addition to, the main difference are in their design. Explo_bot has a circular base with a body which implies more weight. Also, it is important to note the position of the sensors. While in udacity_bot the lidar is ubicated in the front and lower part of the robot, in explo_bot, the lidar is centered in the body under the arms.

5 Discussion

Overall, the udacity_bot benchmark model, was the most effective, possibly there is a physical configuration that allows it to navigate faster through the map. However, the personal model explo_bot performs well through its trajectory to the goal. In the end, both models reached the goal successfully and solved the global localization problem.

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. MCL suppose that the robot has small probability of being kidnapped to a random position in the map. So, it is not recommended this method to approach the kidnapped problem in localization.

Finally, MCL could be used in an industry domain where robot needs to localize and move itself in a known static map.

6 Conclusion / Future work

Both robots where capable to accomplish the desire goal. It was perceptible the efficiency of MCL as an algorithm to localized a robot in a know environment. In addition to, analyzed the existing packages of ROS that allows to perform navigation easily through parameter tunning. However, they were not use all of them. For future works, will be a key point to analyzed the impact of those ones to improve performance during convergence.

On the other hand, study different designs and positions to the sensors is crucial to acquire data. For this work, were analyzed two different approaches with different locations.

Also, it will be interesting to extend this work to a three dimensional world and be more accurately modeling and solving real world problems.

7 REFERENCES

1) https://en.wikipedia.org/wiki/Extended_Kalman_filter
2) https://home.wlu.edu/ levys/kalman_tutorial/
3) http://petercorke.com/wordpress/rvc/mobile-robotics/6-localization/
4) https://medium.com/@ioarun/robot-localization-using-particle-filter-fe051c5d38e2
5) http://wiki.ros.org/navigation/Tutorials/RobotSetup
6) https://en.wikipedia.org/wiki/Monte_Carlo_localization
7) Particle Filters for Mobile Robot Localization. Dieter Fox, Sebastian Thrun, Wolfram Burgard, and Frank Dellaert
8) Approaches to Mobile Robot Localization in Indoor Environments. Patric Jensfelt