Where am I? Project Write-up

1. Abstract

In this report, we discuss robot localization problem. A comparison between two localization paradigms are made. Then we address our work to apply one of these paradigms, which is Monte Carlo localization, to estimate where a mobile robot is. We discuss how to build a model for a robot on simulator and how to test an algorithm in certain environment. Algorithm parameter tuning, to give proper response, is discussed later with comments on the results and suggestions for performance enhancement.

2. Introduction

In this project we build a model for a differential drive mobile robot and design a localization algorithm for robot navigation. The robot model is built using URDF file and simulated in gazebo and Rviz. For localization, we use "AMCL" ROS package implementing Monte Carlo Localization algorithms based on particle filter. Our task is to properly tune algorithm parameters. For robot motion we use "move_base" ROS package. For system component integration and operation, we made two launch files for all required nodes: "udacity_world.launch" and "amcl.launch".

3. Localization Background

Robot localization is one of the most important problems to be studied in any robotic system so that the robot can achieve the required tasks. As known that any robot have a goal to be reached in its environment, it is necessary that the robot always keeps track of its actual state, which is the core concept of feedback control. Due to measurement noise and model uncertainty it is not feasible to completely depending on feedback or state prediction. So, the concept of stochastic state estimation is introduced to combine the predicted information based on control action and actual measurements gathered by robot sensors to estimate the real state of the robot (i.e. robot pose). Thus the localization problem can be formally described as stochastic state estimation problem which is solved using estimation filters based on Bayes inference.

Localization Algorithms can be classified into two main categories:

1. Parametric Filters (ex. Extended Kalman Filter)
2. Nonparametric Filters (ex. Particle Filter)

Extended Kalman Filter:

A parametric filter which models model uncertainty and sensor noise as Gaussian noises. It differs from original Kalman filter that it can deal with nonlinear system dynamics which is the case of any robotic system. EKF consists of two main parts: prediction and correction. In prediction part state mean and covariance is updated according to our knowledge of previous state and the

applied control action. Then, the predicted state and corresponding covariance is corrected by a Kalman gain multiplied by the error between predicted and measured values. The algorithm is classified as parametric filter since it assume noise to be modeled by parametric equations. EKF is fast in computation and efficient in calculation but have limitations with noise types.

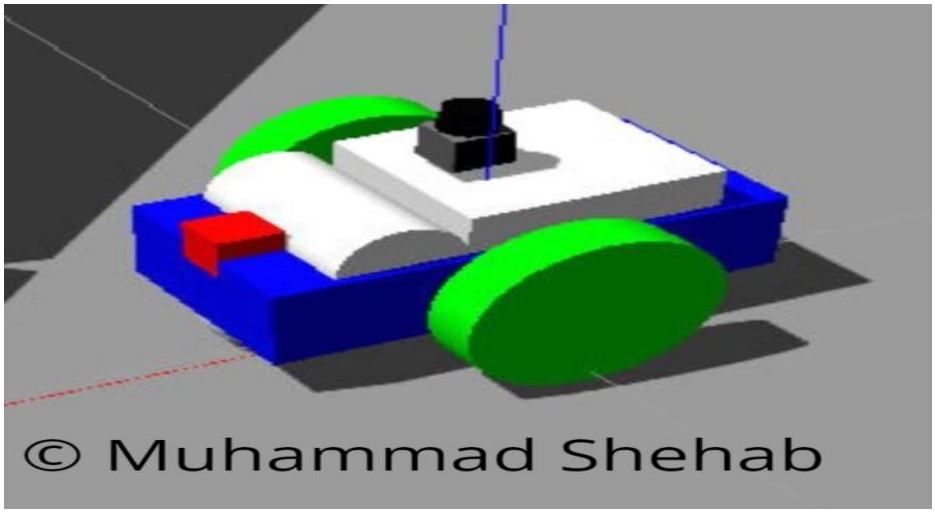
Particle Filter:

A nonparametric filter that can deal with any type of noise. It is the basis of Monte Carlo Localization (which is implemented in this project). The idea of that filter is to propose multiple solutions (represent the particles) of the suggested robot state. The control action is then applied to each solution and the predicted state is calculated. A measurement is then taken and the error is evaluated for each particle. Particles with smaller error have high probability to represent the real state. Thus, the particles with high weights are passed to the next iteration. When the algorithm converges, all the particles are approximately in the same location which is robot state. The advantage of this algorithm is that it doesn't assume any restrictions on noise. However, it is time and memory inefficient as it needs to make computations for each particle. Another great aspect of MCL is that it can be used for global localization by choosing large number of particles and initializing them through the entire environment, which can't be applied in EKF, since EKF needs only one initialization of robot state and it should be close to the real state. EKF can only be used for local localization.

4. Results

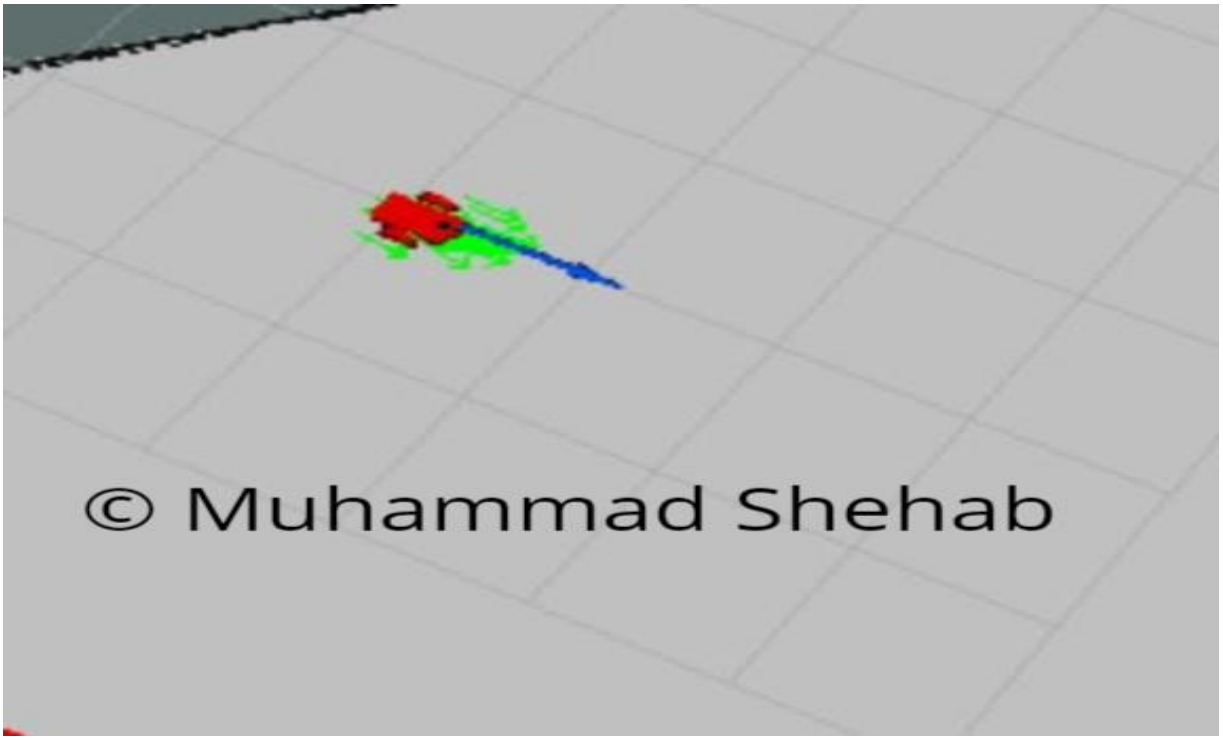
4.1 Robot Model

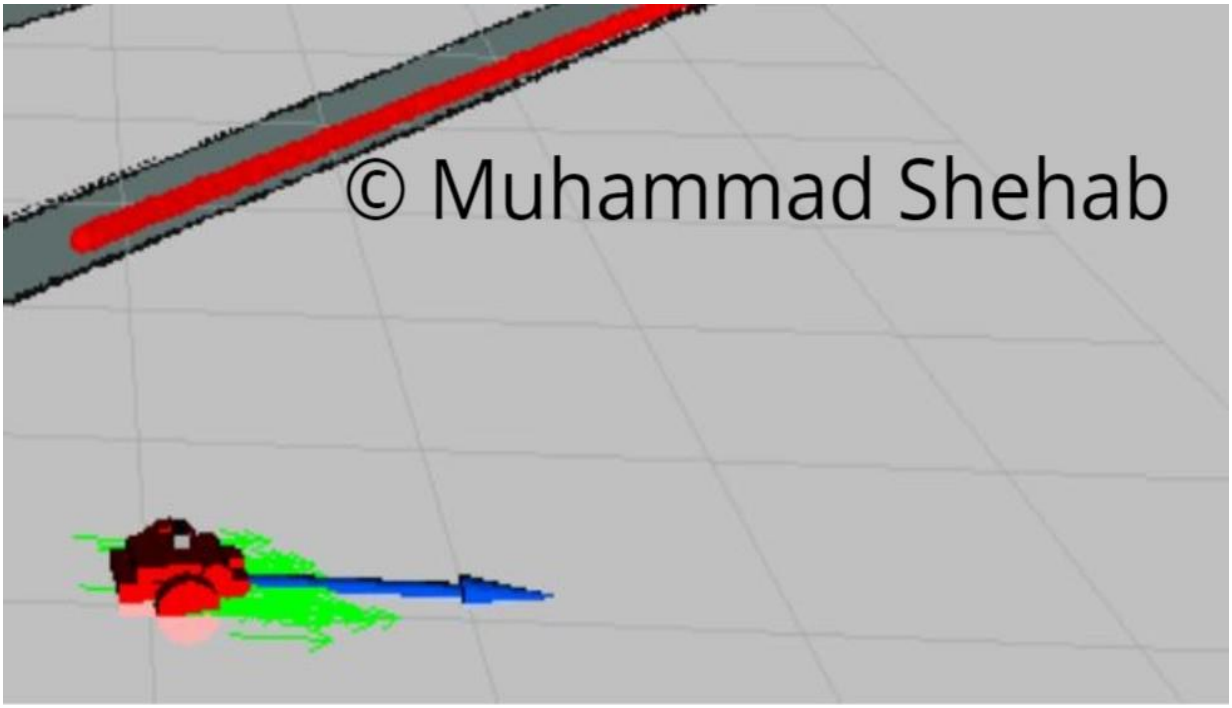
Based on the given design, we made slight modification by adding a driving cabinet and a tower above which the laser range finder is located. Thus laser range finder location in our model is different from udacity_bot which gave a chance to compare amcl results between both cases. When tower height increases above certain value, the algorithm doesn't behave in a good manner. Also, we changed camera location to be on front of the cabinet directly.



4.2 Achieving Requirements

Both models: udacity_bot and our model have reached the goal as shown in the figures below. The robot followed the planned path with some error between the local path and the global path in the beginning (and the particles of amcl algorithm were relatively spread). Having reached the goal, the particles converged to smaller area.





5. Model Configuration

5.1 Robot Model

The pose of laser range finder in my_bot is at the center of the robot exactly, the robot may be considered as a circle whose radius is a parameter for the localization algorithm. The camera is also lift up from the ground by making it above the base of the robot so that the captured scene contains more information about the environment.

5.2 Map Update and Size

Due to computation limitations, we have to choose suitable rate of map update and use only necessary size of the map to decrease space and time required for map update. We use:

- 10 Hz for update and publish frequencies.
- 20 m for width and height of global costmap.
- 5 m for width and height of local costmap.

5.3 Robot Motion

For computation minimization, we explore the necessary space required for robot short term motion. Also So that the robot doesn't crashed into walls we make a safety distance of 0.4 m to be considered between detected walls and robot path. If we increase this distance the robot suffers when passing through narrow corridors. For these purposes we use:

- 1.5 m for obstacle range.
- 1.5 m for raytrace range.
- 0.4 m for inflation radius.

5.4 AMCL Parameters

We have different sets of parameters that contribute to algorithm performance. These sets are: overall filter parameters, Laser Range Finder parameters, and Odometry parameters.

- Overall filter
    - min_particles = 20
    - max_particles = 100
    - update_min_d = 0.1 m
    - update_min_d = 0.25 rad

We try to use minimum number of particles that give good response. For more robustness we need to increase the number of particles but that require higher computation capabilities. Also we don't make update until certain translational or rotational movement is occurred.

- Laser Range Finder
    - We use 20 beams to capture information about obstacles. This number must be greater in a real world with noisy measurements.
- Odemetry
    - As our model is differential drive robot we use diff-corrected type for odem_moodel_type
    - As odemetry data received from simulation, we needn't tune odom_alpha1-4, but this is necessary step in a real world scenario.

6. Discussion

It is obvious that AMCL requires much computation than other methods based on parametric filters. In our project, decreasing map size and update rate make the algorithm work in an appropriate manner, but this may not be an acceptable solution in a real world scenario. When talking about kidnapping problem, the robot must discover the whole environment so we should use large number of particles and large size of map. Another Issue to be addressed is that rotational accuracy is better than position accuracy at goal for both robots. For more robustness, uncertainty should be modeled by real experiments.

7. Future Work

All work done in this project only represents a first step in localization problem issues to be solved. For real world application this work should be extended to include:

1. Sensor Calibration to estimate measurement noise ranges.
2. Real application of control action to evaluate model uncertainty.
3. Use of more powerful Hardware to perform required computations effectively.
4. Try to tune algorithm parameters, specially odom_alphas, and Laser_hit on a real robot.