Robot Localization using AMCL

Navin Rahim, navin.rahim@gmail.com

Abstract—This paper discusses the localization and navigation of two custom build robots in a predefined map. The entire process is performed using Gazebo simulation. AMCL was used for localization and ROS navigation stack was utilized for autonomous navigation through the map. After tuning the parameters for the packages, the robots were able to accurately localize itself and reach the goal position defined.

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

1 Introduction

One of the important tasks many of the robots has to perform for taking any of its actions is to know where it is in the environment. Locating the current position and orientation of a robot in a map is called localization. It is one of the most important tasks because inaccuracies in localization can cause serious damages. If a self driving car localizes itself and says its in another lane, it can cause damage to both the car and its surroundings.

Many approaches are available for the localization problem. Adaptive Monte Carlo Localization(AMCL) was used to localize robots in a predefined map in this project. The project was run in simulation environment and packages in ROS was used.

2 Background

There are mainly four popular localization algorithms - Extended Kalman Filter, Markov Localization, Grid Localization and Monte Carlo Localization. Kalman Filter and Monte Carlo Localization(MCL) are the commonly used ones. Also, a version of MCL is implemented for localizing the robot in this project.

2.1 Kalman Filters

Kalman Filters estimates a state of a system by taking in noisy sensor data and then filtering out the uncertainties in them. Thus, its a popular algorithm used for the localization task in robotics. Robots have noisy sensor data. Kalman filter takes these as input and based on the noises in them, quickly estimates the accurate pose of the system. It provides this estimate even after a few sensor inputs.

Kalman filters works on the assumption that the system is a linear one and that it can be represented using a unimodal Gaussian distribution. But, this is not always the case in real world systems. This is where a variation of the Kalman Filter, called the Extended Kalman Filter(EKF) comes in to the picture. EKF approximates the non-linear systems to a linear system using Taylor Series expansion. Since, it can handle non-linear systems, it is popularly used for localization of robots.

2.2 Monte Carlo Localization

MCL is another popular localization technique widely used in robotics. MCL uses particles to localize a robot and is popularly called as particle filters. Particles can be thought of as guesses of where the robot is and as the robot moves, the particles are also moved in the same way and the sensor measurements are compared. The particle with the most similar measurements represent the location of the robot. After each motion and prediction, particles are resampled with particles with higher similarity to the robot having higher probability.

AMCL is a variation of the particle filter algorithm in which the number of particles are reduced when the position of the robot is more certain. This helps in computational efficiency and is thus implemented for localization in this project.

2.3 Comparison / Contrast

Although EKF and MCL provide accurate estimates of the pose of a robot in a map, each has its own advantages and disadvantages. While EKF ensures efficiency in memory and time, it requires landmark measurements and is comparatively difficult to implement. MCL is easier to implement and works well for non-linear systems and measurement noises. AMCL further helps in bringin in the efficiency in computation. Thus, AMCL is implemented for the localization of the robot in this project.

3 Simulations

Two robots were built using the Gazebo simulator for the purposes of this project. Also a map provided by Udacity(Fig. 1) was used. AMCL and Navigation Stack provided in ROS was used for localization and navigation respectively.

Launch files were created to launch the robots. Rviz was used to visualize various ROS topics and is launched in the same launch file. AMCL and navigation stack are launched using a separate launch file. Also, an additional launch file is provided to give the goal location the robot needs to navigate to.

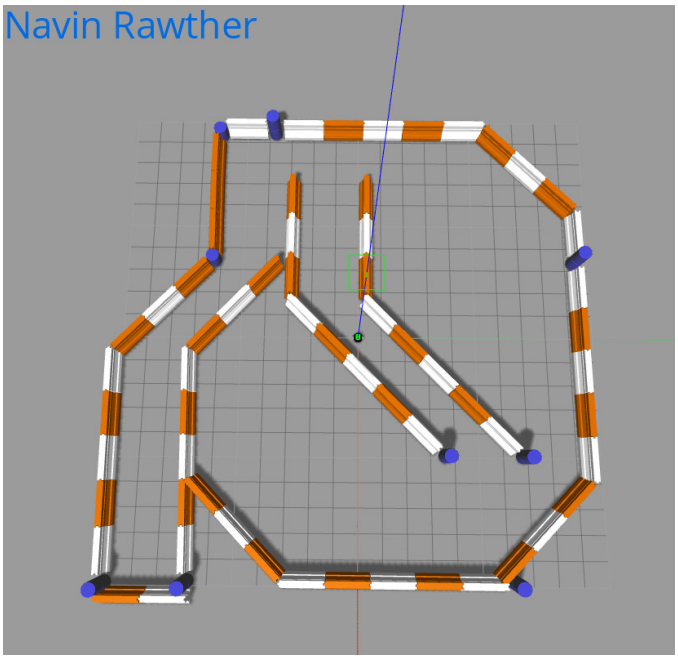

Fig. 1: Map used in the project

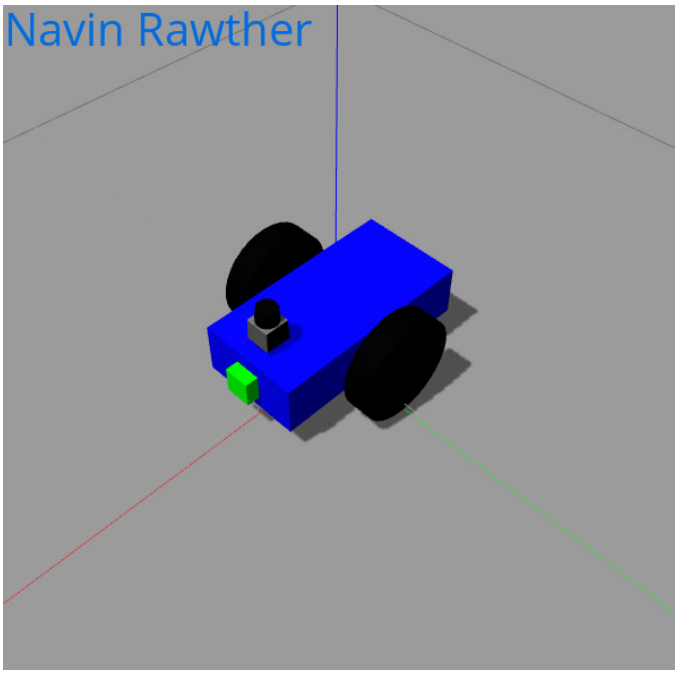

Fig. 2: Udacity bot

3.1 Achievements

After trying out tuning of different parameters for both the AMCL and navigation stack, the robots were able to localize themselves and navigate to the goal locations by avoiding obstacles.

3.2 Benchmark Model

3.2.1 Model design

The benchmark model was developed using design parameters provided by Udacity. The bot had a cuboid base with two sensors. The camera sensor was mounted in the front and a Hokuyo 2D laser was mounted on top of the base. The Udacity bot is a differential bot with two wheels and two caster wheels for stability. The parameters for building the robot is provided in TABLE 1.

Link	Shape	Specification	Remarks
Chassis	Box	0.4 x 0.2 x 0.1	
Casters	Sphere	Radius = 0.05	Front and Back
Wheels	Cylinder	Radius = 0.1 Length = 0.05	Left and Right
Camera	Box	0.05 x 0.05 x 0.05	Connected to chassis at [0.2,0,0,0,0,0]
Hokuyo Laser	Box	0.1 x 0.1 x 0.1	Connected to chassis at [0.15,0,0.1,0,0,0]. Mesh file used for visualization.

TABLE 1: Udacity bot specs

3.2.2 Packages Used

A package was created for incorporating the launching of the robot and localization and navigation purposes. The package is named udacity_bot. It contains various folders as listed below.

- config - Contains the tunable parameters for navigation stack
- launch - Contains files for launching various nodes and robot
- maps - Contains the map files
- meshes - Contains file for Hokuyo design
- src - Contains C++ file for providing goal location
- urdf - Contains the robot specifications and design
- worlds -Contains the Gazebo world where robot has to be displayed

3.2.3 Parameters

Different parameters were tested to obtain accurate localization and to get the robot move to the goal location. The ROS wiki page was used to obtain understanding on the parameter usage. The final parameters obtained apart from the default values are listed below

- AMCL parameters
  - min_particles = 10
  - max_particles = 100
  - initial_pose_a = 0.062 - Since, the robot always started at the same position
  - update_min_d = 0.01 - Added, so that particles were updated for change in robot translation
  - update_min_a = 0.01 - Added, so that particles were updated for change in robot rotation
- Common Costmap Parameters
  - obstacle_range = 3.0
  - raytrace_range = 3.0
  - transform_tolerance = 0.2
  - inflation_radius = 0.25
- Global Costmap Parameters
  - update_frequency = 5.0
  - publish_frequency = 5.0

- Local Costmap Parameters
  - update_frequency = 5.0
  - publish_frequency = 2.0
  - width = 5.0
  - height = 5.0
- Base Local Planner Parameters
  - pdist_scale = 5
  - publish_cost_grid_pc = true
  - heading_score = true

3.3 Personal Model

3.3.1 Model design

The personal model was developed trying to emulate a Roomba robot design. The bot had a cylindrical base with two sensors. A box was kept on the base to attach the sensors. The camera sensor was mounted in the front of the box and a Hokuyo 2D laser was mounted on top of the box allowing it to observe the environment. The personal bot, named Navbot is also a differential bot with two wheels and two caster wheels for stability. The parameters for building the robot is provided in TABLE 2.

Link	Shape	Specification	Remarks
Chassis	Cylinder	Radius = 0.2 Length = 0.05	
Casters	Sphere	Radius = 0.05	Front and Back
Wheels	Cylinder	Radius = 0.1 Length = 0.05	Left and Right
Top Base	Box	0.2 x 0.2 x 0.3	Connected to Chassis at [0,0,0.15,0,0,0]
Camera	Box	0.05 x 0.05 x 0.05	Connected to Top Base at [0.1,0,0,0,0,0]
Hokuyo Laser	Box	0.1 x 0.1 x 0.1	Connected to Top Base at [0.07,0,0.185,0,0,0]. Mesh file used for visualization.

TABLE 2: My caption

3.3.2 Packages Used

The Navbot was built in the same package as of the Udacity bot. A different config folder for tuning the navigation stack, separate launch files and URDF file was used apart from the packages of the udacity bot.

3.3.3 Parameters

The parameters used for the Navbot is listed below.

- AMCL parameters
  - min_particles = 10
  - max_particles = 100
  - initial_pose_a = 0.062 - Since, the robot always started at the same position
  - update_min_d = 0.01 - Added, so that particles were updated for change in robot translation
  - update_min_a = 0.01 - Added, so that particles were updated for change in robot rotation
- Common Costmap Parameters
  - obstacle_range = 3.0

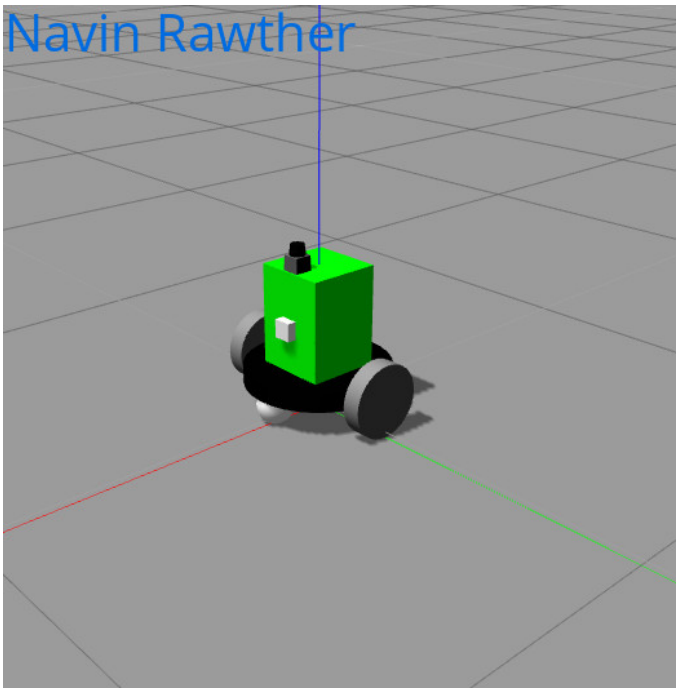

Fig. 3: Navbot

  - raytrace_range = 3.0
  - transform_tolerance = 0.2
  - inflation_radius = 0.25
- Global Costmap Parameters
  - update_frequency = 5.0
  - publish_frequency = 5.0
- Local Costmap Parameters
  - update_frequency = 5.0
  - publish_frequency = 2.0
  - width = 5.0
  - height = 5.0
- Base Local Planner Parameters
  - pdist_scale = 2
  - publish_cost_grid_pc = true
  - acc_lim_theta = 10 - Increased as robot kept moving towards the obstacle and stopped near it without rotating away from it.

4 Results

4.1 Localization Results

4.1.1 Benchmark Model

The particles were distributed around the robot in the initial stage. As soon as the robot started moving, the particles converged and localized the robot. Also, the robot took a smooth path to reach the goal. This was due to the tuning of parameters of the AMCL and navigation stack.

4.1.2 Personal Model

The Navbot required modifications in the parameters obtained for the udacity bot. After few fine tuning, the particles for Navbot also converged as the robot started moving and followed a smooth path to the goal.

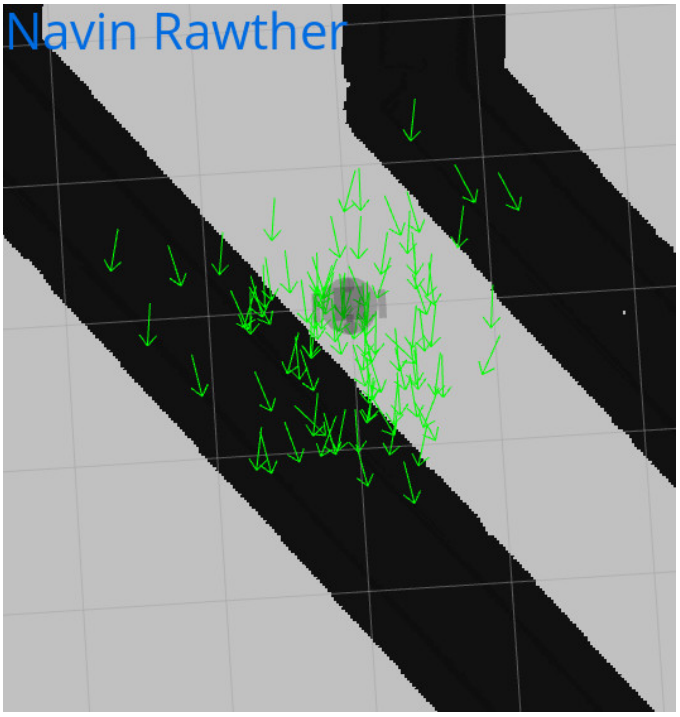

Fig. 4: Initial distribution of particles around Navbot. The distribution was similar for udacity bot as well.

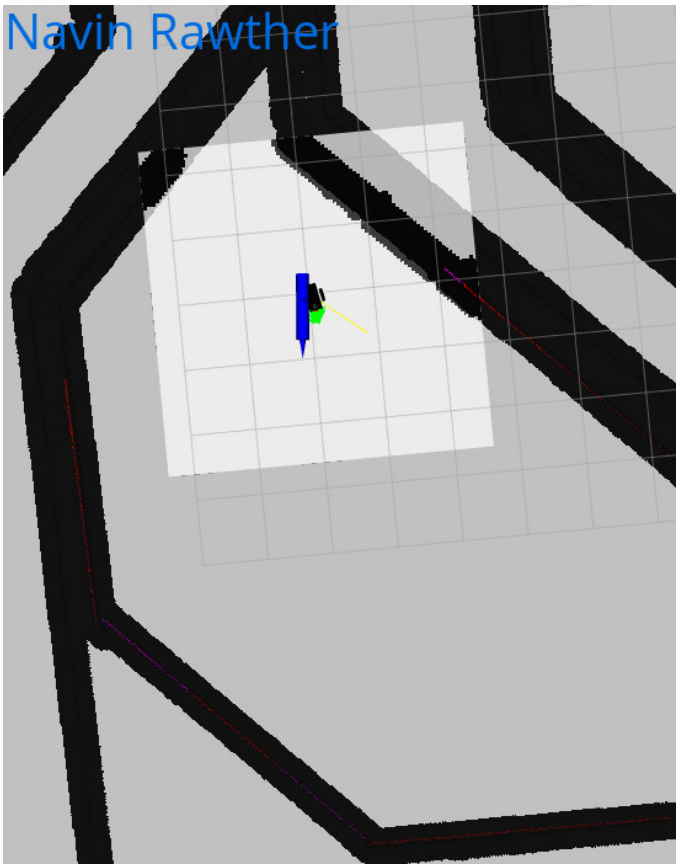

Fig. 5: Udacity bot after reaching the destination. Blue arrow points to the goal pose. Green lines are the converged particles.

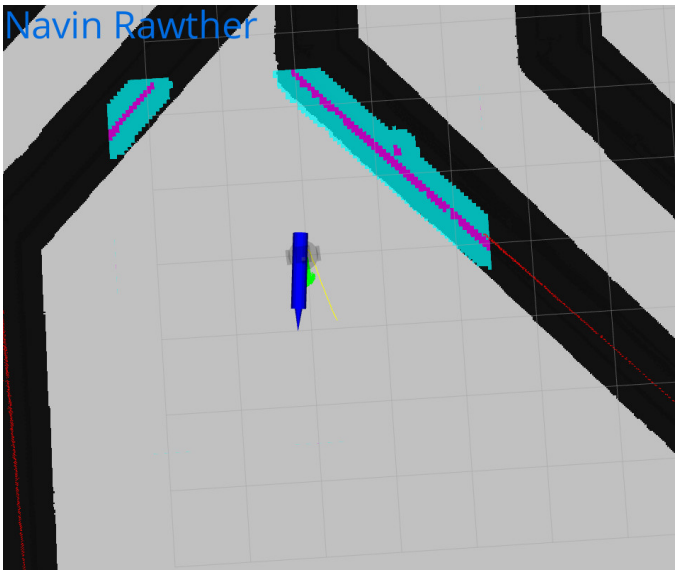

Fig. 6: Navbot after reaching the destination. Blue arrow points to the goal pose. Green lines are the converged particles.

4.2 Technical Comparison

Both the robots performed well in the localization problem as they were able to converge as soon as they started moving. Although the udacity bot could follow the path well, the Navbot took a few deviations in the curves. Still, both the robots were able to move smoothly, avoiding obstacles, localizing itself perfectly and reaching the destination under two minutes.

5 Discussion

The udacity bot performed better than the Navbot in terms of following the path to the destination. Both the robots performed equally in case of the localization. The udacity bot performed better since it followed the path almost perfectly and took a smooth path to reach the destination. Navbot took a few turns away from the global path in the initial stage before starting to follow the global path and reaching the destination.

For the Kidnapped robot problem, the number of particles has to be increased as it can be moved to any location in the map. Localizing it again would take more time and computations depending on the map size.

Localization is important module in robots and could be used for robots moving items in a warehouse to self-driving vehicles. MCL/AMCL packages are ideal candidates for localization in indoor environments although a lot of effort has to be utilized in tuning the parameters.

6 Conclusion / Future work

Two robots were developed in Gazebo and ROS packages were used to accurately localize these robots and navigate them to a goal location. AMCL was used for localization and ROS navigation stack for autonomous navigation. The robots were equipped with a 2D Lidar and a camera. Fine tuning of parameters were done to accomplish the task.

The ROS packages can be implemented in actual hardware. The model of the robot could be built in Gazebo simulation and parameters can be tuned there. Once good results are obtained in simulation, it can be deployed in the robot and could be fine tuned according to the performance.

The implementation could be improved by using more sensors such as ultrasonic sensors and utilizing the camera to perform a few vision tasks. This would be helpful in getting accurate understanding of the obstacles and the environment around the robot.

The design of the robot depends on the use case. Also, computation power and battery requirements vary based on the hardware available. Based on these considerations, benchmarks could be defined and algorithms could be tuned to attain good performances.