Where AM I?

Beshari Jamal

Abstract—Two different wheeled robot models were considered for performance evaluation. The robots were different in size, weight and wheel diameters. Using the Robot Operating System (ROS) and its packages, the robots were virtually constructed and launched in a simulation, then had its sensors and links finetuned for localization and navigation with a predefined map. The robots were lastly evaluated to reach a predefined goal position and goal orientation.

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

1 Introduction

ROBOT localization is one of the essential functions of a mobile robot. It is the ability to ascertain its position and orientation in a frame of reference. A robot uses that information for path planning. Without accurate localization information, a robot path planning is always faulty and could be fatal and costly.

There are three different kinds of localization problem: position tracking also know as local localization, global localization where the initial pose is not known, and the kidnapped robot, where the robot is moved somewhere else without it knowing. Probabilistic techniques such as Kalman Filters, Grid localization and, Monte-Carlo Localization are used to localize the robot using a sensor rangefinder, and a predefined map. The map provided by Clearpath Robotics in Gazebo simulation environment. In this project, Adaptive Monte-Carlo localization (AMCL) method and extended Kalman Filter (EKF) are discussed.

The move_base package is used as part of the navigation stack. The package produces a local cost map using the range finder sensor, as the robot progress, ,making the local cost map move to another global cost map (Clearpath Robotics map) to implement a smooth path.

2 Background

While the first industrial robot was invented in 1961 [6], ROS only started serving the public in early 2007, developed since the mid-2000s in Stanford University, then finally made public by help from Willow Garage, a visionary robotics incubator [2].

2.1 ROS

ROS is a robotics middleware. A middleware is a software that mediates between a piece of software or an application and a network. It manages the interaction between the various applications across the complex computing programs. ”Robotic middleware is designed to manage the complexity and heterogeneity of the hardware and applications, promote the integration of new technologies, simplify software design...” [3]. ROS is composed of a set of tools and libraries made by scientists and roboticists, that aims to simplify the task of creating or recreating complex and robust robot behavior across a wide variety of robotic platforms.

2.2 ROS Navigation Stack

ROS Navigation Stack is a powerful ROS tool for mobile robots to move from one place to another. The navigation stack, given by processing data from odometry, sensor streams, and environment map, plans a global path and local path manifested as velocity commands to the mobile base. The ROS Navigation Stack can be imported through ROS, however, to maximize the performance of the navigation stack the probabilistic parameters need some finetuning.

2.3 Localization Methods

The main two localization methods considered for the robots are the Kalman Filters and particle filters.

2.3.1 Kalman Filters

Kalman Filters is an estimation algorithm that is prominent in controls. It is used to estimate a variable such as speed or location in real-time as other data including noise is being collected. Kalman filter gets its popularity from being able to make very accurate estimations given a considerable amount of noise input. It works on a two-step process: measurement update and state prediction. First, an initial guess is used as a first state prediction, measurement update happens, then lastly a control action may happen in such as a manifestation of movement, then the cycle starts again at state prediction. Given the assumptions that motion and measurement models are linear and state space can be represented by a Gaussian distribution, the Kalman filter converges to an accurate solution. Unfortunately, most mobile robots execute non-linear motion such as moving in a curve. This fact makes KF unsolvable in closed-form - in a finite number of operations - and more computationally intensive. A linear approximation to any nonlinear function can be drawn using Taylor Series on the mean, and then the resulted linear function can be used in KF, this methodology is known as Extended Kalman Filter.

2.3.2 Particle Filters

Particle Filters use particles to localize the robots. Each particle represents a guess of where the robot may be located and has a position and orientation. These particles are assigned probability weights and re-sampled each time the robot makes a measurement or control update. The unlikely particles are then weeded out.

2.3.3 *Comparison / Contrast*

While both Kalman Filters and Particle filters accurately localize robots after few iterations. KF is used mainly in linear measurement and motion models, that is why KF is mostly used in guidance, navigation, and control in vehicles notably aircraft and spacecraft. KF can, for example, predict (value and covariance) using dead reckoning (information from various sensors such as gyro sensor and accelerometer) and measurement update with GPS data. However, particle filters are widely used because it can approximate almost any state space distribution, easy to implement and robust against noise. Particle filters can also be changed to use fewer particles, hence providing control over memory and resolution. Because our robots move with a differential controller, therefore moves in curvy lines, adaptive Monte Carlo localization (amcl), an example of particle filters, is used for localization.

3 SIMULATIONS

The simulation was carried out using the Robot Operating System (ROS), Gazebo and Rviz (3D visualization tool for ROS). Rviz data shows sensor data and custom visualization model such point cloud data, map, robot models.

3.1 Achievements

Two robots were written in Unified Robot Description Format (URDF). The corresponding ROS packages were written to launch two custom robots in Gazeboo world. AMCL ROS packages and other plugins were utilized to connect the sensors, odometry, and control for the end goal of localization and navigation. Both robots were able to localize themselves and navigate to predefined goals.

3.2 Benchmark Model

The benchmark robot as seen on figure 1, is a simple robot with two driving wheels on the sides and two caster wheels at the front and the back of the robot. The range sensor and camera are stacked on top of each other at the front of the robot.

3.3 Personal model

The personal model robot is a similar robot as seen on figure 2, with two driving wheels on the sides two caster wheels at the front and the back of the robot. The range sensor and camera are in the middle of the front half of the robot. The robot is lighter, has a broader and thinner base and smaller wheels in diameter.

3.3.1 *Packages Used*

The same packages were used for both robot models.

The move_base package implements some kind of action that, given a goal, will attempt to reach it with a mobile base. The move_base node connects a global and local planner to accomplish its global navigation task move_base publishes on "cmd_vel" a stream of velocity commands

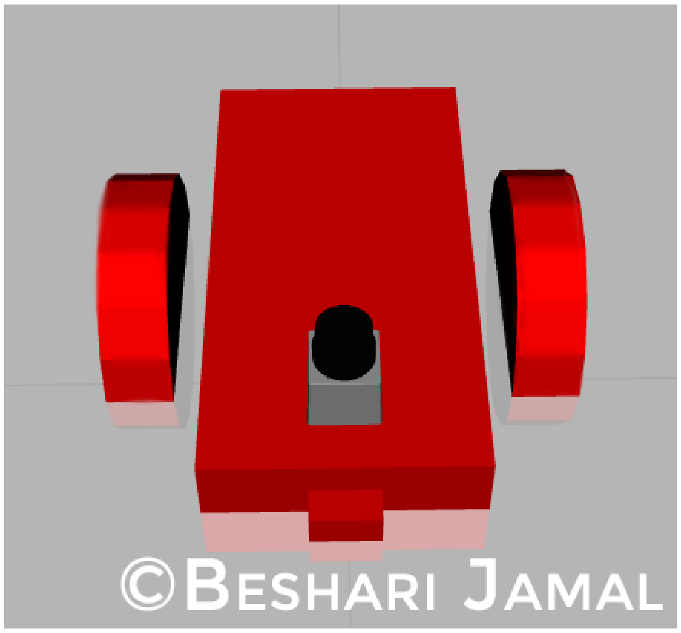


Fig. 1. Bench Mark Model

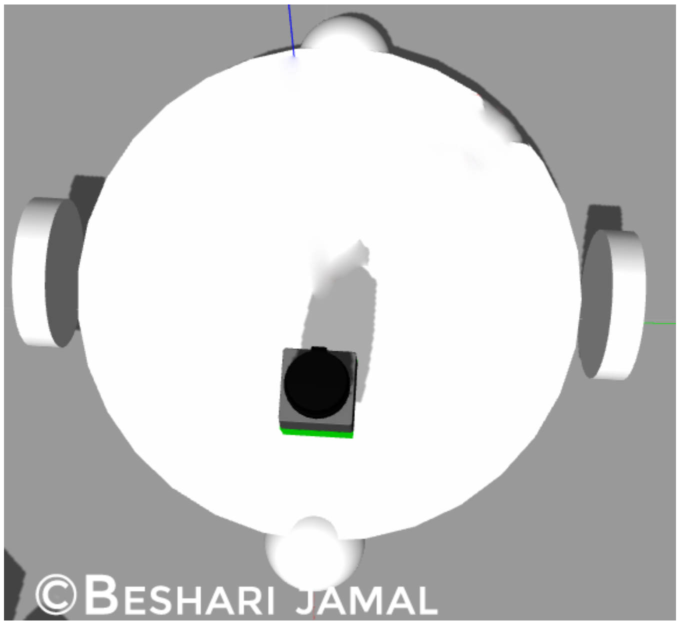


Fig. 2. Personal Model

meant to be executed. Amcl package also has a make_plan clear_unknown_space and clear_costmaps services.

The amcl package is probabilistic localization system to track the pose of a robot against a known map. Amcl publishes a pose and a point cloud of poses on "particlecloud." Amcl has global_localization which initializes a space set with uniformly distributed random pose particles, and request_nomotion_update, a service to manually refresh and publish updated particles.

3.3.2 *Parameters*

In the amcl package, there are three types of parameters: "min_particles" and "max_particles", unmistakably from their names, control the number of virtual particles used in the amcl package.

”transform_tolernece” is the time with which to postdate the localization transform. It only needs to be high enough to cover the lag in the system.

Finally: ”odom_alpha1”, ”odom_alpha2”, ”odom_alpha3”, and ”odom_alpha4” are noise parameters in differential drive mobile robots. They correspond to expected noise in odometry translational or rotational estimate from each of the robot’s translational or rotational motions.

TABLE 1
amcl Parameters Table

Parameter name	Benchmark	personal
min_particle	15	10
max_particle	100	70
transform_tolerence	0.4	0.8
odom_alpha1	0.01	0.001
odom_alpha2	0.01	0.01
odom_alpha3	0.05	0.05
odom_alpha4	0.05	0.05

The ”move_base” package interfaces with sensory information, cost maps, local and global planner. Hence it is expected to have parameters that are involved with all of these.

There are three types of parameters of characteristic parameters in the local map and global map package:

Ranger finder related: ”obstacle_range” and ”raytrace_factor” parameters determine the radius in which the local map of the robot is cleared. The ”obstacle_range” is the distance obstacles can be sensed within.

”robot_radius” and ”inflation_radius” parameters are used by the ”move_base” package to plan paths far enough from obstacles and walls.

”move_base” has a local planner that has its parameters. Local planner parameters are mostly self-explanatory and control the kinematics of the robot; ”max_vel_x” is the maximum velocity in the x-direction and ”acc_lim_theta” is the rotation acceleration limit.

some useful parameters to mentions are: ”holonomic_robot” is a boolean that is true if the robot degrees of freedom is the same as the maximum possible degree of freedom. ”sim_time” is the amount of time to forward-simulate trajectories.

”vx_samples” and ”vtheta_sampples” are the numbers of samples to use when exploring the x velocity space or the theta

4 Results

As seen in Both robots reached their end goal in a localization and navigation challenge. It is found that when the robots are in the radius of inflation of the walls, they escape very slowly. Besides, both robots may start by going the other direction until there is enough space for the local planner to plan a U-turn. Nevertheless, reaching their goals and following their global plan conclusively.

TABLE 2
move_base Parameters Table

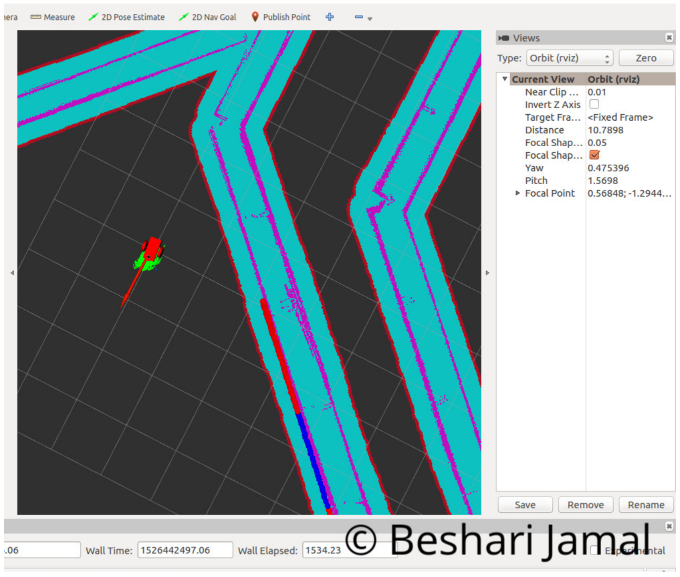
Parameter name	Benchmark	Personal
obstacle_range	3	2.5
raytrace_range	3.5	3.0
transform_tolerence	0.4	0.5
robot_radius	0.3	0.3
inflation_radius	0.2	0.3
coast_scaling_factor	2.5	3
max_vel_x	0.6	0.75
min_vel_x	0.1	0.1
max_vel_theta	1.0	1.0
acc_lim_theta	2.5	2.7
acc_lim_x	2	2.5
acc_lim_y	0	0
holonomic_robot	false	false
sim_time	5	5
controller_frequency	6.0	7.0
vx_samples	15	20
vtheta_samples	30	40
yaw_goal_tolerence	0.2	0.2
xy_goal_tolerence	0.1	0.1

4.1 Localization Results

Both robots are able to localize themselves once able to follow the global plan.

4.1.1 *Benchmark*

The Udacity robot was able to localize itself shortly after starting moving. The move_base package was therefore able to plan the right trajectory to its end goal position as shown in figure 3



Type: Orbit (rviz) Zero

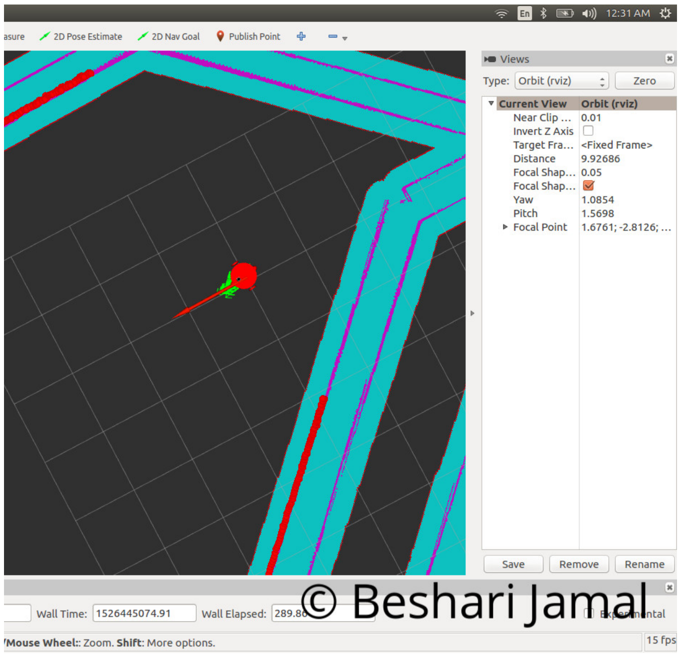
Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	10.7898
Focal Shap...	0.05
Focal Shap...	☑
Yaw	0.475396
Pitch	1.5698
Focal Point	0.56848; -1.2944...

Wall Time: 1526442497.06 Wall Elapsed: 1534.23

Fig. 3. Benchmark Model Reaching Final Goal

4.1.2 *Student*

The personal robot was also able to localize itself. The robot reaching its final goal position can be seen in figure 4



Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	9.92686
Focal Shap...	0.05
Focal Shap...	✓
Yaw	1.0854
Pitch	1.5698
Focal Point	1.6761; -2.8126; ...

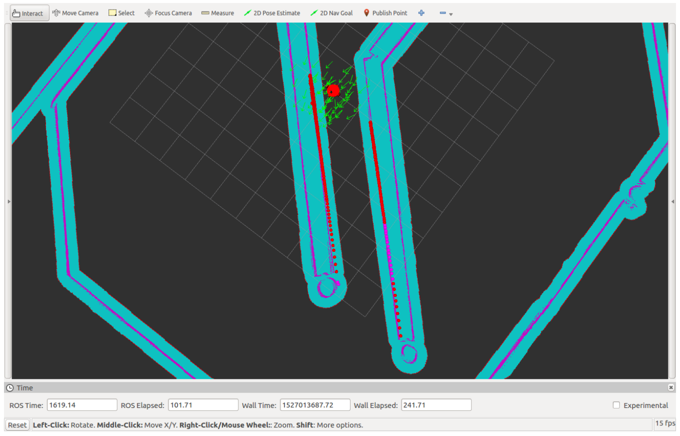
Wall Time: 1526445074.91 Wall Elapsed: 289.86

Fig. 4. Personal Model Reaching Final Goal

4.2 Technical Comparison

The two robots are similar in architecture but different in size, wheel diameters, the locations of sensors, caster wheels, and finally the base's shape and weight. Both robots as seen in figures 1 and 2 were close in performance, with the bench mark being more stable. Yet the personal model reached the goal faster.

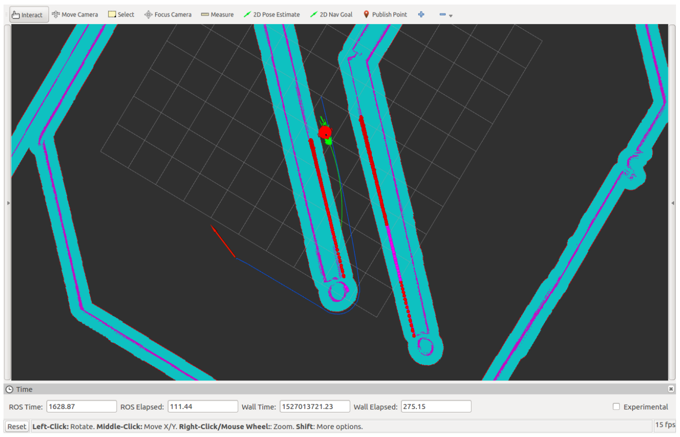
5 Discussion



ROS Time: 1619.14 ROS Elapsed: 101.71 Wall Time: 1527013687.72 Wall Elapsed: 241.71

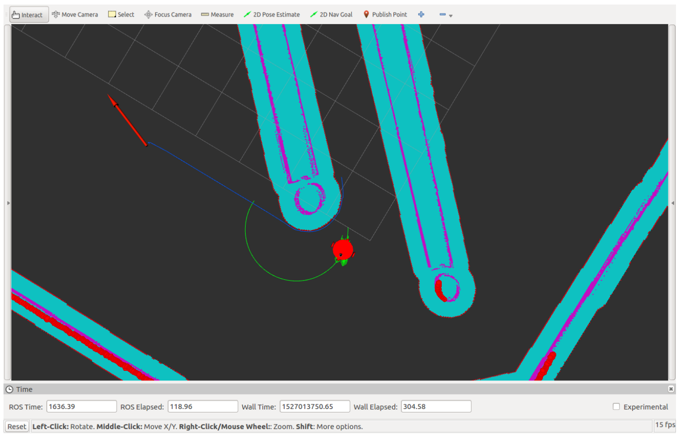
Fig. 5. Personal Model at start of simulation

The robots performed satisfactorily localizing themselves and reaching their end goal positions. Both robots are experts in going on straight lines or following smooth curves, yet under harsher tests, it is found that both robots get lost not finding any control solution, and failing to make U-turns sometimes. the robot also gets slow when they are in the a higher-cost area, especially at beginning during their discovery phase, where the Monte Carlo localization



ROS Time: 1628.87 ROS Elapsed: 111.44 Wall Time: 1527013721.23 Wall Elapsed: 275.15

Fig. 6. Personal Model completing a straight path



ROS Time: 1636.39 ROS Elapsed: 118.96 Wall Time: 1527013750.65 Wall Elapsed: 304.58

Fig. 7. Personal Model turning around a corner

is determining the robots initial location. However, within the project scope, both robots perform well.

The typical implementation of AMCL is inadequate in solving the kidnapped robot problem on its own, however superior in the global localization challenge. The difference is that when the robot is kidnapped, it believes it is somewhere that is not, and there may bot surviving amcl particles in the new location. At the time, this could be solved by resting the particles and starting again. There are also many alternate proposed solutions such as continuously augmenting the particles with another uniform particles or introducing more noise than there is. Still, the first used method is the mixture Monte Carlo Localization. The difference between the two is that the MCL algorithm first guesses using odometry and then assigns importance weights using sensor data, the Mixture-MCL algorithm, besides, guesses new poses based on sensor data and assigns weights based on odometry. Bringing a disadvantage of the mixture Monte Carlo Localization algorithms being a requirement for a sensor model that allows fast sampling of poses. Kd-trees is usually utilized as a data structure solution for such rangefinder sensor data.

ROS can be deployed to any ARM-based microcontroller with Linux installed, sensors can be interfaced using Rviz and the robot directly. When deploying this type of projects to real hardware, mobile robots require to be light with limited power. This limitation means that efficient use of processing is of utmost importance as it is directly correlated to power usage. The higher the processing, the less time a robot is active.

Therefore, while adding more sensors and resolution can produce a high-quality map, it also employs more sensors and processors to unnecessary extremes. Careful attention should be paid to how the robot is sensing and processing the surroundings, and whether a high-quality map is needed or a robot (such as the Romba) does just fine with a minimalist 2D map.

6 Conclusion / Future work

After successful fine tuning, localization and navigation. The project can be deployed to hardware.

For future work,The robot current camera can be used as a depth camera to take in more sensor data, consequently an obstacle detection algorithm can be implemented. in addition, More work on software would make the robot able to map the environment with SLAM.

References

[1] David L. Anderson, Jeremy Gottlieb, 2013 *Introduction to Robotics, ROBOTS: IN THE BEGINNING*. http://www.mind.ilstu.edu/curriculum/medical_robotics/robots_in_beginning.php Accessed: June 18, 2018.

[2] Open Source Robotics Foundation, 2017 *ROS: History*. http://www.ros.org/history/ Accessed: June 18, 2018.

[3] Ayssam Elkady, Tarek Sobh, 2012 *Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography*. https://www.hindawi.com/journals/jr/2012/959013/ Accessed: June 18, 2018.

[4] Udacity, 2017 *Robotics Software Engineer Nanodegree Program: Term2: Localization*. https://www.udacity.com/ Accessed: June 18, 2018.

[5] Open Source Robotics Foundation, 2017 *ROS: amcl*. http://wiki.ros.org/amcl Accessed: June 18, 2018.

[6] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaer, 2001 *Robust Monte Carlo Localization for Mobile Robots*. http://robots.stanford.edu/papers/thrun.robust-mcl.pdf Accessed: June 18, 2018.