Where Am I? - ROS AMCL and Navigation Stack

Snehal Gor

Abstract—Mobility is one of the basic need not only for humans or animals, but also for robots. Mobile robot requires localization of itself in an environment. Author here has explored ROS AMCL (Adaptive Monte Carlo Localization) and Navigation stack. Further Author has also explored building own robot using URDF and tuning different parameters of ROS packages. Author here is documenting his learning, findings and results. Author has successfully build two mobile robots - one with 4 wheels and another with 3 wheels, created packages that launches a custom robots in a Gazebo world and utilized packages like AMCL and the navigation stack to localize and navigate the robots towards goal location on a map.

Index Terms—Robot, Udacity, LaTeX, Monte Carlo Localization, Navigation, ROS, Gazebo, Differential Drive.

1 Introduction

As age of data, artificial intelligence and robots dawn upon us, robots which can handle complex environment autonomously without any human interventions is becoming everyday reality. We see robots in every sphere of life, be it in industrial automation, entertainment, space exploration or Autonomous vehicle (and many more). Robots are making our world safer, faster, greener, smarter and enjoyable.

For robot to be mobile and autonomous in an environment, localization is one of the key. Mobile robot localization is the problem of determining robot's pose (location and orientation) in an environment from sensor data. In this report, Author is documenting his exploration of robot localization in Gazebo environment using different ROS packages. In subsequent section Author will document

- Background
- Project brief and Model Configuration
- Results
- Discussion
- Future Work

2 Background / Formulation

Mobile robot localization comes in many flavors in particular below Author has listed prominent three

- Position Tracking - Here initial robot position is known, and goal is to compensate errors in robot's odometry
- Global localization - This is more difficult problem, as initial robot pose is not known and goal is to determine it from scratch from multiple distinct hypothesis.
- Kidnapped robot problem - It is the most challenging problem, where robot (can be well localized) is teleported to some other place without being told. The kidnapped robot localization problem often used to test Robot's ability to recover from catastrophic localization failure.

There many existing algorithms to tackle position tracking problem. One of the prominent being Kalman tracker. Kalman filter estimates posterior distributions of robot's poses conditioned on sensor measurements. Although kalman filter is an elegant algorithm, Gaussian belief representation makes classic kalman filter inapplicable for global localization problem. There are algorithms like Markov localization and multi-hypothesis kalman filter, which can handle global localization (although with restrictive beliefs or approximations).

To robustly solve global localization and kidnapped robot problem, probabilistic localization algorithm called Monte Carlo Localization (Adaptive MCL / Mixture MCL [1] for kidnapped robot problem) is very popular. Below Author has provided comparison of Kalman filter and Monte Carlo localization (Table 1).

TABLE 1
Kalman vs. MCL

	Kalman	MCL
Measurements	Features	Raw Data
Measurement Noise	Gaussian	Any
Posterior	Gaussian	Particles
Efficiency	Good	Normal
Robustness	No	Yes
Global Localization	No	Yes
State Space	Uni-modal Continuous	Multi-modal Discrete
Ease and Control	Normal	Good

3 Project brief and Model Configuration

Author used Gazebo and different ROS packages to implement a package each for two robots. Each package launches a custom robots in a Gazebo world and utilizes packages like AMCL and the navigation stack to localize and navigate the robots towards goal location on a map. Below Fig 1 shows overall setup [2]

Below table 2 provides brief about two robot setups, while Fig 2 shows simple graphical representation of the same.

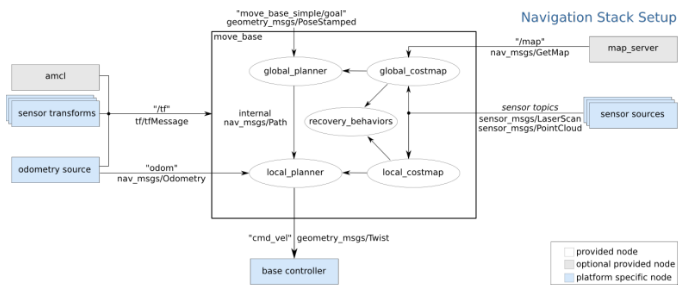

Fig. 1. Overall Setup with tf.

TABLE 2
Brief details of Author's Robot setup

	Robo1 (Udacity)	Robo2
Sensors	Camera and Range Finder	Camera and Range Finder
Camera Position	In front on chassis	In front on a platform (For better visibility)
Laser Range Finder	In front on chassis	In front on a platform
Wheels	4	3
Number of Joints	4 (2 wheels and 2 sensors)	5 (3 wheels and 2 sensors)
Actuation	Two wheel Differential	Two wheel Differential
Efficiency (Memory and Time)	Good	Normal
Localization	AMCL	AMCL
Map or World	jackal_race	jackal_race

3.1 Differential Drive [3]

One important part of Author's robot setup is Differential drive and corresponding kinematics model. Below Author is sharing his understanding and learning.

A differential drive is based on two separately driven wheels and doesn't require additional steering motion. For balancing the robot with differential drive, additional wheels are required. Author used two caster wheels for Robot 1 and one caster wheel for 2nd robot.

In differential drive, robot's motion vector is sum of independent wheel motions

- Straight line motion is produced by moving both the wheels at the same rate in same direction.
- In place rotation is produced by moving both the wheels at the same rate in opposite direction
- Other variations like turning left / right while forward / backward moving, is obtained varying speed and / or direction of the wheels

3.2 Configurations / Parameter Tuning [2] [4] [5] [6] [7]

Below Author is providing insights into different parameters tuned by Author for both robots.

- AMCL
  - Author kept minimum particles at 200 and maximum particles around 2000. As environment is not complex and static, along with almost no noise due to working in simulated environment. Author observed that particles are doing good job of localizing the robot.
  - Author changed "update_min_d" and "update_min_a", to half the previous number. As Author wanted filter to update itself more frequently against pose changes.
  - Author kept laser model parameters to default, with maximum range to 30. As Author wanted in wider space in a map, robot should be able to localize itself.
  - As we are in simulated environment, Author changed "odom_alpha1...4" parameters and made it 0.05.
- Cost Map
  - Author kept "inflation_radius" to 0.4. Author kept moderate inflation radius to make sure that, Robot can travel in narrow spaces, at the same time it will not go very much near the obstacles.
  - Author kept "obstacle_range" at 2.5 and "raytrace_range" to 3.0. 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 free space given a sensor reading.
  - "transform_tolerence" kept at 0.2, to make sure that tf tree gets updated
  - Author reduced width and height of the "global_costmap" and "local_costmap" to reduce computation load
- Planner - Author used almost default parameters (except for controller frequency) for both the robots. For Robot 2, as Author observed some deviation in Robot's motion from global path, Author increased "pdist_scale" to 4.0.
- Overall - Author updated different frequencies like "controller_frequency", "update_frequency" and "publish_frequency" to make sure that controller and maps are updated regularly. And also simulation running without crashing.

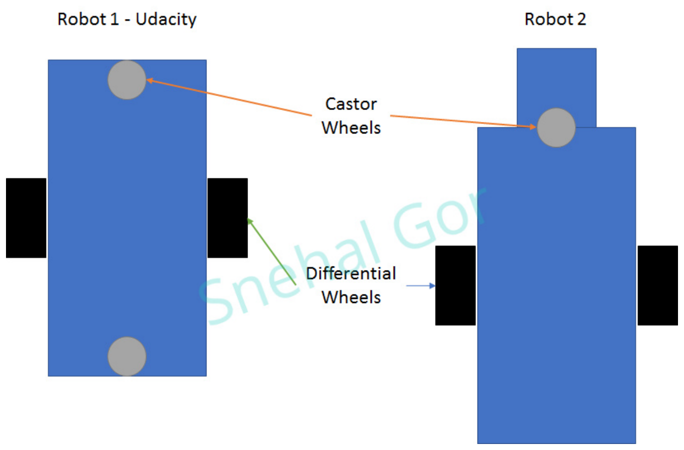

Fig. 2. Author's Robots Setup

4 Results

Author got a good result, with both the robots able to reach the goal. Also localization accuracy is quite good, even in open areas. Snapshots of the output at goal location been provided below in Fig 3, Fig 4.

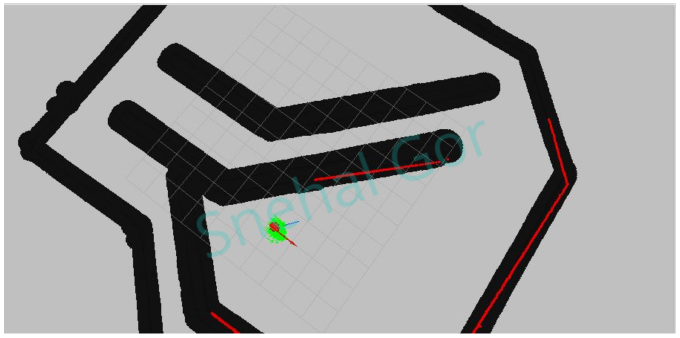

Fig. 3. Output Snapshot - Robot 1 - With Global Costmap.

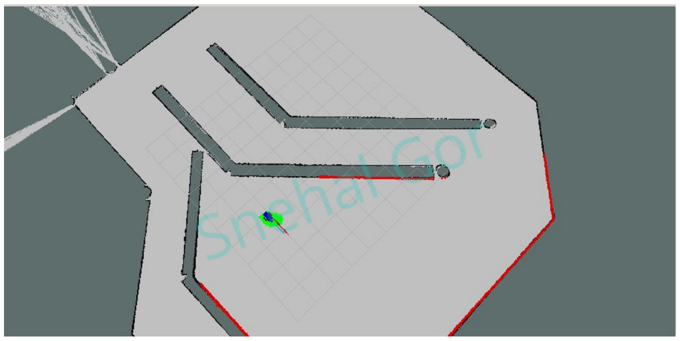

Fig. 4. Output Snapshot - Robot 2.

Below Author is sharing qualitative comparison of both robots (Table 3)

TABLE 3
Qualitative comparison of Robots

	Robo1 (Udacity)	Robo2
Stability	Better	Good
Motion	Better	Good
Localization	Good	Good
Actuation	Two wheel Differntial	Two wheel Differntial
Cost	Good	Better

5 Discussion

Author could successfully localize both robots in Gazebo environment. In Author's opinion following are few of the learning / reasons of getting a good performance

- Simplicity of the environment : Map environment which has been simulated in Gazebo, is static with no moving obstacles or complex reflection or patterns
- Ideal sensor models, where almost no effects of environment or inherent noises
- Monte Carlo Localization (Particularly Adaptive MCL) powerful enough to work in complex and noisier environment
- Author also used ROS provided Navigation stack to handle control and planning part

Author could easily localize and drive Robot 1 in map environment. But for Robot 2, Author faced some challenges. Below Author is sharing learning

- Three wheel robot is more unstable than four wheel. Where Author observed shaky motion with initial design.
- To make sure that Robot is not shaking, Author connected through a fixed joint a caster wheel at front with non-zero mass
- Author also observed inertial lag, due to which robot was not following global path. Author increased "gdist_scale" parameter's weight to make sure that Robot follows global path
- Author reduced AMCL noise parameters, as working in simulated environment in ideal condition
- Author also learned / explored xacro macros, differential drive and kinematics [3], Mixture-MCL [1]
- Author explored / tuned ROS Navigation stack and explored "costmap", "move_base", "base_local_planner". Author do feel need of better documentation and further exploration of the same.

In Author's opinion AMCL can solve kidnapped robot localization problem. Key to it being the ability to recover from unexpected large state changes. As described in excellent paper by Thurn [1], combining regular samples with its dual it can overcomes a range of limitations of MCL.

Author sees MCL / AMCL can be applied to Autonomous car / robot / drone localization, for object tracking, locating a sensor node in wireless sensor networks (WSN), etc.

6 Conclusion / Future work

Author has successfully implemented / experimented with two mobile robots: Building own robot using URDF, Tuning different parameters of ROS packages - AMCL and Navigation stack. With respect to AMCL, Author sees clear trade-off between accuracy and processing time. Whereby increase in number of particles provides better accuracy, but at the cost of processing speed. Also Author sees trade-off between direct utilization of raw data against features, whereby accuracy and processing time can be trade-offed. AMCL is excellent in terms of handling those trade-offs.

Author also sees following further areas of exploration / future work

- Mobile robot kinematics and dynamics exploration
- Exploration of omni-drive robot
- Deeper exploration of different ROS packages

References

[1] S. Thrun, D. Fox, W. Burgard, and F. Dallaert, "Robust monte carlo localization for mobile robots," *Artif. Intell.*, vol. 128, pp. 99–141, May 2001.
[2] *Setup and Configuration of the Navigation Stack on a Robot.*
[3] *Mobile Robot Kinematics.*
[4] *AMCL - Probabilistic Localization System.*
[5] *base_local_planner - Local Robot Navigation on a plane.*
[6] *Inflation Costmap Plugin.*
[7] *ROS move_base packge.*