Project : Where Am I ?

Goutam Gupta

Abstract—The problem of localisation is about finding the robots actual position w.r.t to its environment.In this paper, we will investigate different methods to solve robot's localisation problem and utilise ROS packages to accurately localise a mobile robot.

Index Terms—Robot, IEEEtran, Udacity, Localization.

1 Introduction

The problem of localisation is about finding the robots actual position w.r.t to environment within 2-10 cm of accuracy. Localisation of robots can be done using images from GPS but that incurs errors of up to 10m, which causes applications like self-driving cars to fail. There are different localisation problems in real word scenario. The easiest localisation problem is called position tracking or local localisation. In this case, robot knows its initial pose and robot needs to localise itself as it moves. This becomes difficult because of uncertainties involved in the environment. Global localisation problem involves situations where the robot does not know its initial pose and it must identify its position w.r.t to ground truth map as it moves. The uncertainties involved in global localisation is much more than local localisation problem. The most challenging localisation problem involves the localisation of kidnapped robot. This problem is like global localisation except that robot can be kidnapped at any time and moved to a new location. Real world problem deals with dynamic environment which is more challenging to localise in. In this project, we will focus on static environment where the environment always matches the ground truth map. In this project, we will learn to utilise ROS packages to accurately localise a mobile robot inside a provided map in the Gazebo and RViz simulation environments. Major task in this project includes the following:

- Building a mobile robot for simulated tasks.
- Creating a ROS package that launches a custom robot model in a Gazebo world and utilizes packages like AMCL and the Navigation Stack.
- Exploring, adding, and tuning specific parameters corresponding to each package to achieve the best possible localization results.

2 Background

There are four most popular localisation algorithms: Extended Kalman filter, Markov localisation, Grid localisation and Monte Carlo localisation. Kalman filter is very robust algorithm to filter noisy filter data which can be used for localisation. Localisation is performed by sense and move cycles. Every time the robot moves, it loses information and every time it senses it gains information. There are there ways for environmental tracking. Kalman filter technique estimates continuous state and hence gives uni-modal distribution. Monte Carlo localization estimates discrete states and it gives multi-modal distribution. Particle filters also address localisation with continuous states and multi modal distribution. There are three common types of Kalman filter: KF, EKF and UKF. KF is mainly used for linear systems whereas EKF and UKF is used for non-linear and highly non-linear systems. On the other hand, MCL uses particle filters to track the robot pose and present many advantages over EKF.

2.1 Kalman Filters

It can take data with lot of uncertainties/ noise and provide a very accurate estimate of the real value. Kalman filter is a continuous iteration of two step process. First step is a measurement update where we use the recorded measurement to update our state. The second step is a state prediction where we use the information about current state to predict the future state. We use an initial estimate. Kalman filter helps to consider the uncertainties in both movements and sensory measurement to solve localisation problems. Measurement update is the weighted sum of prior belief and the measurement. State prediction is the addition of prior beliefs mean and variance to the motions mean and variance. Posterior belief is the updated belief/probability of localisation after the measurement has been taken. Error in localisation is mostly due to error in measurement because of faulty sensors, which is represented by probability distribution (posterior distribution: probability given the measurement z). Probabilistic localisation becomes even difficult with inaccurate robot motion or uncertainty in robot motion. Variables of a Kalman filter can be separated into two: the observable and the hidden. For the state, we need a state transition function and for the measurement, we need a measurement function.

2.2 Particle Filters

Particle filter have continuous state space representation. Each of the particle is a discrete guess of the robots position. It can be represented by arbitrary multi modal probability distribution. Monte Carlo localisation follows particle filter approach. Particle filter makes them survive in proportional how consistent one of the particles is with the sensor measurement. Place of high probability will club more particles.

Probability of survival is proportional to their weights. The algorithm basically follows following algorithm:

- In the beginning, thousands of particles each representative of robots position and orientation is uniformly spread across the environment. This distribution can be a Gaussian spread as well.
- Add noise to each of the particle to incorporate the noise associated with the environment.
- Move each particle by certain amount in a specific orientation.
- Assign importance weight to each of the particle based on the difference between the actual measurement and the predicted measurement of the distances between particles and landmark position. Predicted measurement is calculated based on sensory measurement.
- Perform re-sampling on the sets of particles so that in each stage of re-sampling the particles with higher importance weights survives which eventually collapses at the actual robots position.
- The entire loop is run several times by moving the robot in a specific orientation. Orientation of particle matters because the predicted measurement changes as the robot moves in a particular orientation. Eventually, after running through several iterations of this loop, most of the particle converges to robots position and acquires a single orientation, thus solving robots localisation problem. In the beginning particles which are representative of robots position is uniformly spread

2.3 Comparison / Contrast

Particle filter can deal with non-Gaussian noise distribution whereas kalman filter can deal only with noise having Gaussian distribution.While Kalman filter can used for linear system,the particle filter can be used for non-linear systems as well.Hence particle filter will be a good choice for localisation provided the system has appropriate computation ability.We will be using adaptive MCL (particle filter approach) in ROS environment for this project.

3 Simulations

Two robot models have been used in Rviz simulation environment to test the performance of AMCL package.Two wheels are added to the existing Udacity bot to create a car like bot. Following images show both the models.

3.1 My Robot Model

As mentioned in the project, a udacity bot is created having a chassis and two wheels.Colors are added to the wheels and chassis by adding material property in the xacro file.A simple race car bot model is created by adding two more wheels in the existing udacity bot model.This robot can skid steer or it can also perform controlled planner motion. We can use object controller gazebo plugin for this model.

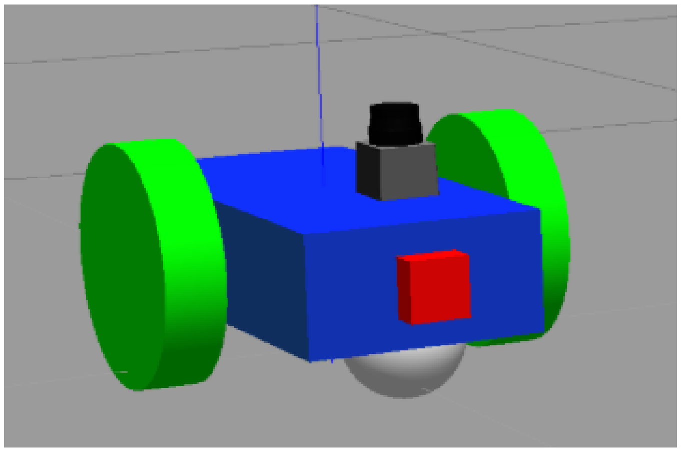


Fig. 1. Udacity bot model.

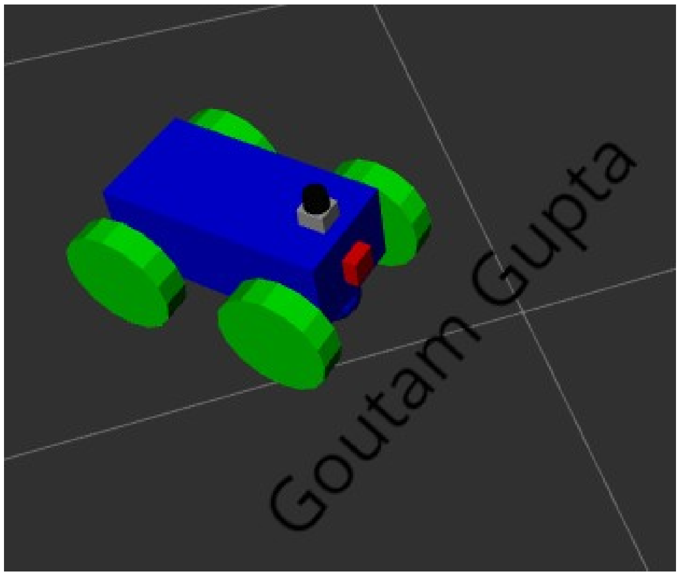


Fig. 2. My robot model.

3.2 Achievements

Both the robot models were tested using AMCL algorithm in Rviz environment. After fine tuning the parameters, the bencmark robots was able to reach the set goals. Personal robot uses four wheel for driving using omni drive mode whereas the udacity bot drives in differential drive mode using its two wheel. The sensors model: Camera and LIDAR were same in both the models. The custom model's localisation was improved by trying different drive mode.

3.3 Benchmark Model

The udacity bot provided in the project is used as the benchmark model.

3.3.1 Model design

Benchmark model Udacity bot includes the following components:

3.3.2 Packages Used

Following packages were used in this project:

Item	Type	Size
robot_footprint	Link	NA
robot_footprint_joint	joint-fixed	NA
chassis	Link	box: 0.4x0.2x0.1
back_caster_visual	visual	sphere: r=0.05
front_caster_visual	visual	sphere: r=0.05
left_wheel	Link	cylinder: r=0.1 l=0.05
left_wheel$_hinge$	joint-continuous	NA
right_wheel	Link	cylinder: r=0.1 l=0.05
right_wheel_hinge	joint-continuous	NA
camera	Link	box: 0.05x0.05x0.05
camera_joint	joint-fixed	NA
hokuyo	Link	mesh: 0.1x0.1x0.1
hokuyo_joint	joint-fixed	NA

TABLE 1
udacity_bot design parameter

map_server : It provides the map_server ROS node which offers map data as a ROS service.
move_base : This package links together a global and a local planner to accomplish its global navigation task. It maintains two cost maps,one for the global planner and one for a local planner that are used to accomplish navigation task.it helps to navigate the robot to the goal position by creating or calculating a path from the initial position to the goal.
joint_state_publisher : It helps in publishing joint state values for a given robot model as mentioned in urdf file.
robot_state_publisher : It publishes the state of robot to tf. state variables of the robot is available to all the components which uses tf once it gets published.
amcl : This package implements adaptive Monte Carlo localisation approach using particle filter for solving localisation problem of a robot.
gazebo_ros : It offers plug-ins for messages and services for interfacing Gazebo through ROS
PoseArray : It depicts a certain number of particles, represented as arrows, around the robot.The position and the direction the arrows point in, represent an uncertainty in the robots pose.Based on the parameters and what values you select for them, as the robot moves forward in the map, the number of arrows should ideally reduce in number.
tf : It helps to keep track of multiple coordinate frames, such as the transforms from the maps generated while launching amcl package(World map,global costmap and the local costmap), along with any transforms corresponding to the robot and its sensors. Both the amcl and move_base packages or nodes require that this information be up-to-date and that it has as little a delay as possible between these transforms.

3.3.3 Parameters

Exploring, adding, and tuning parameters for the amcl and move_base packages are some of the main goals of this project.
transform_tolerance : It defines the maximum amount of latency allowed between transforms from various maps generated from amcl package.This parameter has to be tuned for both the amcl node in amcl.launch file and for the move_base node in the costmap_common_params.yaml file. Tuning the value of this parameter is usually dependent on the system being used to run the simulation environment.
update_frequency : This parameter helps to make sure that the world maps, global costmap and local costmap gets updated fast enough, even on a slower system.
Above two parameters make sure that our robot and maps are responsive. There are few other parameters mentioned in config files which helps the robot to follow the right path.
parameters which are mentioned in costmap_common_params.yaml file are very important in defining how our costmaps get updated with obstacle information and how our robot might respond to those while navigating. These are :
obstacle_range : Tuning this parameter can help with discarding noise, falsely detecting obstacles, and even with computational costs.
raytrace_range : This parameter is used to clear and update the free space in the costmap as the robot moves.
inflation_radius : This parameter determines the minimum distance between the robot geometry and the obstacles.An appropriate value for this parameter can ensure that the robot smoothly navigates through the map, without bumping into the walls and getting stuck, and can even pass through any narrow pathways.
Following approach should be followed while tuning these parameters:

– Try setting a high value for inflation radius first, and have the robot navigate to the goal position. Then repeat the same with a low value.
– set the values for the obstacle and ray-tracing ranges and have the robot navigate to a position, again. Try to see if it can effectively navigate through the narrow passage and then take a turn.
– Add an object, such as the Brick Box in Gazebo. Try to observe how previously selected parameter values behave with an object blocking part of the map.

The amcl package has a lot of parameters to select from. Different sets of parameters contribute to different aspects of the algorithm. Broadly speaking, they can be categorized into three categories - overall filter, laser, and odometry.
Overall filter : The range of number of particle filters, initial pose of the robot and check the values of parameters upon receiving a laser scans needs to tuned through following parameters

– **min_particles and max_particles**
– **initial_pose**
– **update_min_a and update_min_d**

Laser : There are two different types of models to consider under this - the likelihood_field and the beam. Each of these models defines how the laser rangefinder sensor estimates the obstacles in relation to the robot.The likelihood_field model is usually more computationally efficient and reliable for an environment such as the one the project has. Following parameters should be tuned for this environment:

- **laser_*_range**
- **laser_max_beams**
- **laser_z_hit and laser_z_rand**

Odometry:

- **odom_frame_id**
- **odom_model_type**
- **odom_alpha***

3.4 Personal Model

A four wheeler race car type bot is created to test AMCL parametrs.

3.4.1 Model design

my_bot is created to test the localisation problem in Rviz environment with amcl package. This bot has following components:

Item	Type	Size
robot_footprint	Link	NA
robot_footprint_joint	joint-fixed	NA
chassis	Link	box: 0.4x0.2x0.1
back_caster_visual	visual	sphere: r=0.05
front_caster_visual	visual	sphere: r=0.05
front_left_wheel	Link	cylinder: r=0.1 I=0.05
front_left_wheel$_hinge$	joint-continuous	NA
front_right_wheel	Link	cylinder: r=0.1 I=0.05
front_right_wheel_hinge	joint-continuous	NA
back_caster_visual	visual	sphere: r=0.05
back_left_wheel	Link	cylinder: r=0.1 I=0.05
back_left_wheel$_hinge$	joint-continuous	NA
back_right_wheel	Link	cylinder: r=0.1 I=0.05
back_right_wheel_hinge	joint-continuous	NA
camera	Link	box: 0.05x0.05x0.05
camera_joint	joint-fixed	NA
hokuyo	Link	mesh: 0.1x0.1x0.1
hokuyo_joint	joint-fixed	NA

TABLE 2
my_bot design parameter

3.4.2 Packages Used

Packages used for localisation of my_bot is same as used in the benchmark model.Few additional packages are used for the localisation of my_bot model as mentioned below:
object_controller : Gazebo plug-in which is used to test planner move (also omni directional move)
libgazebo_ros_camera.so : provides ROS interface for camera simulations.
libgazebo_ros_laser.so : it simulates laser range sensor by broadcasting Laserscan messages.

3.4.3 Parameters

In addition to parameter changes done to the udacity_bot,few parameters like obstacle_range, inflation_radius and raytrace_range were further modified to improve the localisation of my_bot model.

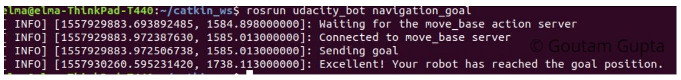
4 RESULTS

Localisation simulation in Rviz environment is performed using amcl package. Below section demonstrates and compares the results obtained for each robot model.

4.1 Localization Results

4.1.1 Benchmark

Navigation goal was sent to the simulated environment by running navigation_goal code using rosrun. Below image shows the command prompt output :



```
elma@elma-ThinkPad-T440:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1557929883.693892485, 1584.898000000]: Waiting for the move_base action server
[ INFO] [1557929883.972387630, 1585.013000000]: Connected to move_base server
[ INFO] [1557929883.972506738, 1585.013000000]: Sending goal
[ INFO] [1557930260.595231420, 1738.113000000]: Excellent! Your robot has reached the goal position.
```

Fig. 3. Console output for udacity_bot.

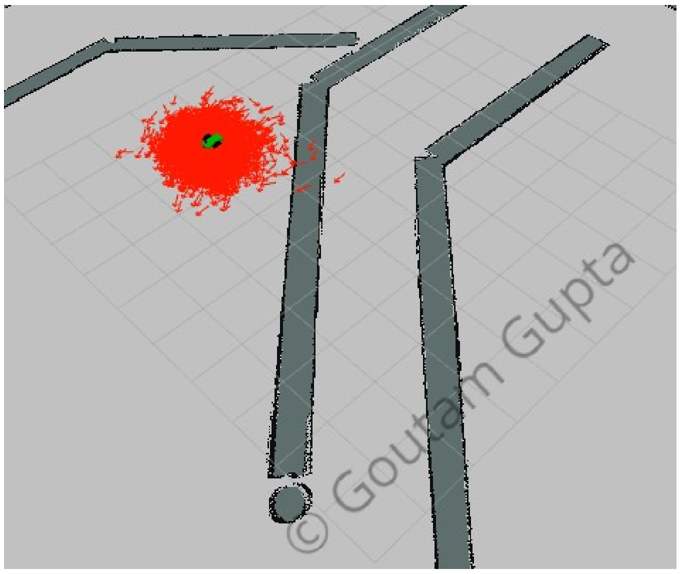


Fig. 4. Rviz environment for udacity_bot.

Robot was able to navigate all of the way to the goal properly avoiding walls and obstacles.

4.1.2 Student

Same simulation was performed with my_bot by running my_navigation_goal command using rosrun. Initially with the same set of parameters,my_bot was not able to reach the goal position. It got stuck at the obstacles and walls. One more parameter ”alpha5” was added for omni drive controller mode and the simulation environment was launched again. Following images shows Rviz environment in this case. The robot is able to move up to certain distance and it gets stuck to the wall.

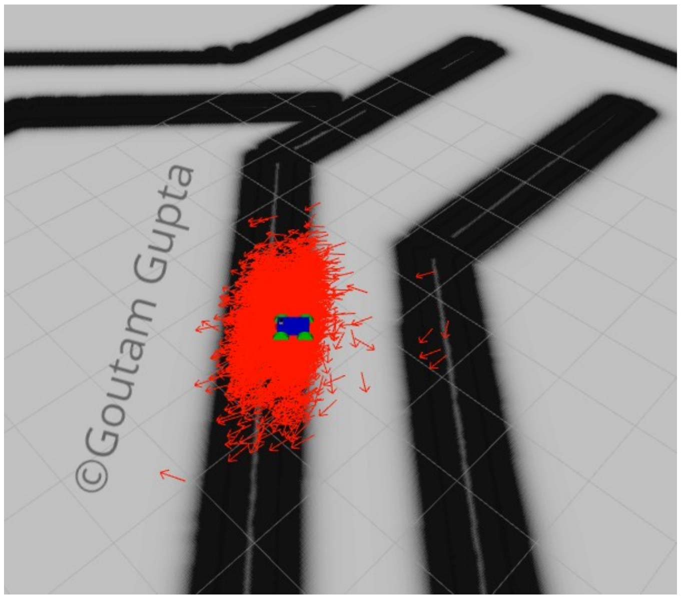


Fig. 5. Rviz environment for udacity_bot.

4.2 Technical Comparison

My_bot having four wheels performed worse than the benchmark robot. The custom bot gets stuck to the wall and not able to reach its desire goal position. We may need to change the drive mode to improve this.

5 Discussion

5.1 Topics

- Which robot performed better? : The benchmark robot performed better as it was able to reach its goal position within a short span of time. The custom bot gets stuck to the wall and was not able to move forward towards its goal position.
- Why it performed better? : Benchmark model worked better because of the differential drive mode controller. The differential drive mode doesn't work better for a four wheeled bot.
- How would you approach the 'Kidnapped Robot' problem? : In mobile robotics localization, the kidnapped robot problem is defined as a condition when the robot is instantly moved to other position without being told during the operation of the robot.In order to solve this problem, we should recover from kidnapping which can be done either by increasing the number of particles or enhancing the algorithm like mixed- MCL.
- What types of scenario could localization be performed? : There are lot of use cases where localisation of robot is very critical. Some of the use cases involves home automation, inventory management etc.
- Where would you use MCL/AMCL in an industry domain?: It can be used in industrial robot particularly in manufacturing industry where the accuracy of localisation of robot is very critical.

6 Conclusion / Future work

The benchmark robot was able to reach the desired goal position but the custom created bot model couldn't reach its goal as it gets stuck to the wall. Several parameters like obstacle_range, raytrace_range and inflation_radius were changed to improve the localisation of custom bot. It helped in improving the localisation of benchmark model but couldn't help the custom bot to localise itself to the goal position. The drive mode is changed to omni drive mode and object controller plug in for gazebo was tried for custom bot.

6.1 Modifications for Improvement

Examples:

- Base Dimension : Base design can be improved to build a more realistic robot model. We can use mesh files for this as it was used for hokuyo sensor in the benchmark model.
- Sensor Location : LIDAR sensor location can be changed to detect obstacle clearly. If we change the LIDAR sensor and camera sensor location it might help in localising the custom bot better.
- Sensor Layout : More sensors placed at location which can cover 360 degree field of view will help the algorithm in improved path planning.
- Sensor Amount : Different types of sensor or using higher number of sensor can be placed for capturing distinctive information about the environment, which in turn can be used for improving the global and local cost-map.

6.2 Hardware Deployment

1) What would need to be done?Actual robot hardware can be developed using a platform having very high computation and GPU capabilities like NVidia Jetson TX2 platform. A micro controller can be used for integrating different sensors. The simulated environment can be deployed into the actual hardware by changing the name of nodes appropriate to the environment running in the hardware.
2) Computation time/resource considerations? : A microprocessor like Raspberry Pi is enough for running ROS nodes but we need an efficient platform Nvidia Jetson platform which can run deep learning algorithm for improving the vision intelligence of the robot.