AMCL Localization Project with ROS

Nicholas Stepanov

Abstract—Two robot models for simulated environment that perform AMCL localization on a known map. The paper describes the models specifics, ROS packages used as well as their parameters. Two different robots are tuned separately to successfully be able to localize. The results are reported as well as possibility for further improvement and deployment on a physical model.

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

1 Introduction

Localization of a robot in a given environment is a tedious task. Robot has to use its sensors to correctly synchronize the data it receives with a known map. There are several approaches to localizing a robot - among them the most used are: Extended Kalman Filters(EKF) and Particle Filters (aka. Monte Carlo Localization(MCL). While Kalman Filters' probabilistic algorithms are easier to implement and need less computing power, their performance is not as good compared to MCL which has several position vectors (particles) being localized at the same time. Possible applications to such algorithms would be company local delivery or warehouse robots in which the environment is easily mapped out on detail. Described package incorporates AMCL algorithms to localize a simulated robot in a known map and can be modified to control an actual robot. The project is contained in the following github repository:

https://github.com/renowator/AMCL_Localization

2 Background

Adaptive Monte Carlo Localization(AMCL) algorithm was chosen because the map is specifically known, and it would provide the best results for such localization. Kalman filter would eventually loose track of the robot and would not be able to localize successfully. However, AMCL does perform really well and has a maximum error of robot's size.

2.1 Kalman Filters

Kalman Filters use Gaussian distribution probabilistic approach to determining robot's position and orientation in a known environment. Various sensor readings could be used to be compared against a map and determine the probability of robot being in a particular place. After movement is executed Kalman Filter uses this information to update its probability values, takes new measurements and repeats the process. Extended Kalman Filter improves the performance of standard Kalman Filters by allowing non linear inputs which would otherwise change the probability range to a non Gaussian. This is achieved by linearizing non linear functions using Taylor Series.

2.2 Particle Filters

Particle filters are initially spread on the map uniformly. As robot explores its environment the particles reposition themselves, according to odometer and sensor information. After some time these particles should converge under the robot's position.

2.3 Comparison / Contrast

While Kalman filters are great for certain simple tasks like localizing in a one dimensional world, more complex environment will make it loose track of the robot's position. MCL advances its understanding of the 2d map as a whole to determine position more precisely and be able to keep track of the robot even with big odometer and sensor uncertainty.

3 Simulations

This section describes the performance of robots in simulation. There are two models included in submission with named udacity_bot.xacro and my_bot.xacro as well as their respective .gazebo files.

3.1 Achievements

Both robots were able to reach the navigation goal in a decent amount of time. However, there is room for improvement, especially with the move_base parameters that set local navigation goal. AMCL algorithm works perfectly and pose array converges under robots actual pose after several movements.

3.2 Benchmark Model

3.2.1 Model design

Udacity bot is a simple model with box of size .4 .2 .2 It has 2 wheels centered on its sides, a camera and a LIDAR hokuyo sensor that publishes information to the topic udacity bot/laser/scan

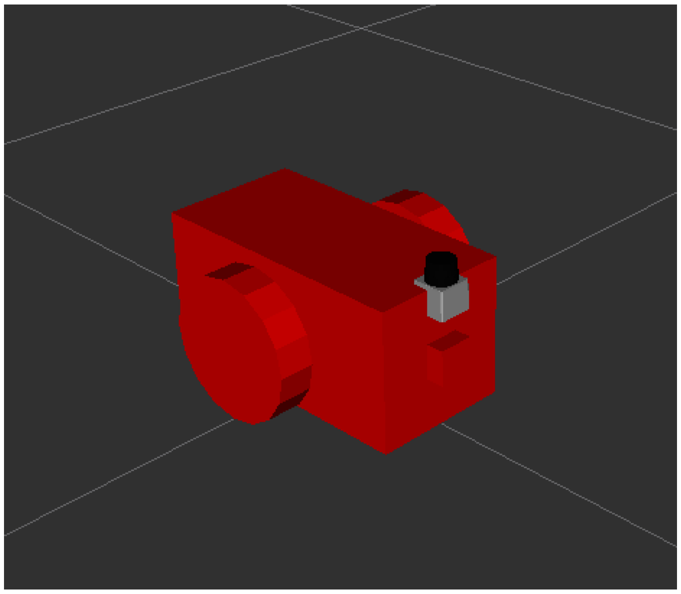


Fig. 1. udacity bot.

3.2.2 Packages Used

The ROSpackages used for the project are: gazebo, robot state publisher and rviz for visualization; map server for holding the global map; tf for synchronization of robot's perception and map information; amcl for localization; move base for trajectory planning. AMCL node receives hokuyo sensor information on the relevant topic, information from state publisher, as well as the map server to perform localization and publish pose array.

3.2.3 Parameters

The following parameters were tuned for the robot to be able to localize on the map successfully and reach its goals. A lot of these parameters could be system specific, and these are designed to perform with small amount of memory. More general parameters of amcl and base local planner can be left the same.

Parameter List		
Name	Value	Comment
amcl:min particles	10	lower value improved performance
amcl:max particles	100	lower value improved performance
amcl:transform tolerance	1.2	was just enough for my system
amcl:odom alpha1245	0.05	barely any noise(simulation)
amcl:odom alpha3	0.1	barely any noise(simulation)
blplanner:all velocities	0.7 or -0.7	slower speed helped navigation
blplanner:escape velocity	-0.3	how to deal with obstacles
blplanner:sim granularity	0.05	increased performance
blplanner:controller frequency	1.0	limited memory in VM
costmap common params:obstacle range	5.0	within the range of local costmap
costmap common params:robot radius	0.1	actual robot radius
costmap common params:transform tolerance	0.7	was just enough for my system

The local costmap was decreased in size to solve the problem of robot going back to where it started.

3.3 Personal Model

3.3.1 Model design

My bot is created based on Udacity bot. This robot model is noticeably larger with chassis box size .5 .25 .35. The robot has larger wheels and a sensor mount. Mounting a sensor above the robot made it easier for him to determine real local map.

3.3.2 Packages Used

my_bot and udacity_bot use identical packages.

3.3.3 Parameters

Most of my_bot parameters did not require change from udacity_bot. Among those that were changed are speed and yaw tolerance(due to different size robot is little more unstable). I also had to further decrease local cost map to be able to navigate correctly. One of the primary changes was tuning occdist(obstacle avoidance) and pdist(follow global plan) parameters in base local planner for the robot to follow an optimal path.

4 Results

Both robots are able to successfully localize in the environment and reach the goal in reasonable amount of time. The AMCL pose array does converge seamlessly after several turns. However, robots local path keeps going off which needs to be investigated more. This does not seem

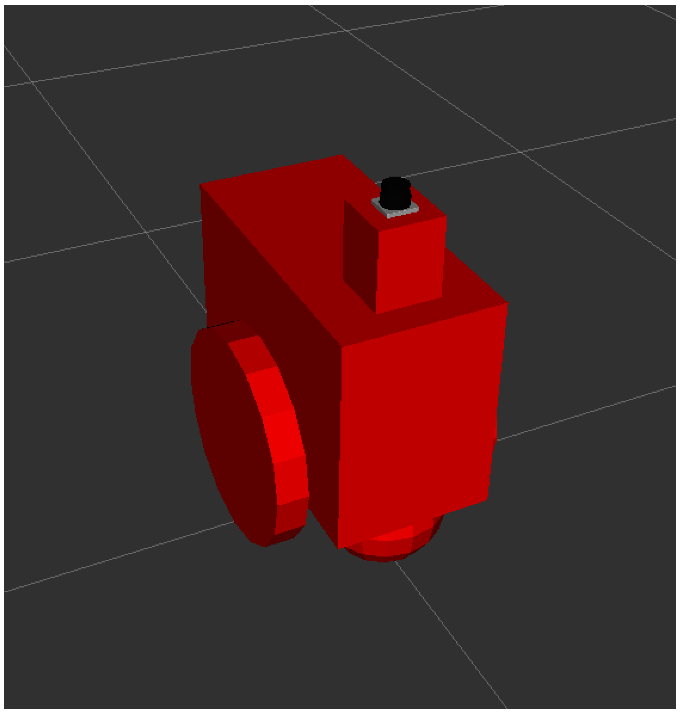
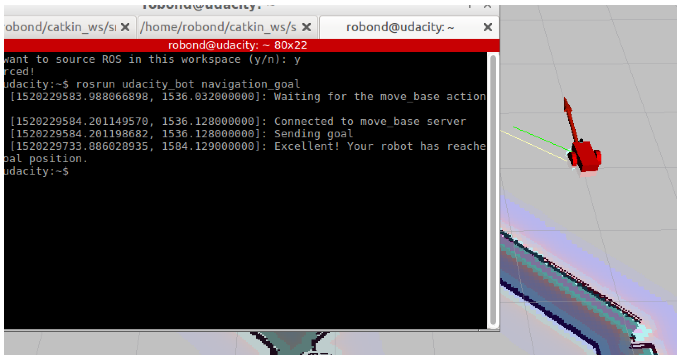


Fig. 2. my bot.

to be a localization problem, but should involve tuning trajectory planning parameters. The results can be recreated by installing the package and running three commands in three distinct terminals: roslaunch udacity_bot udacity_world.launch (or my_world.launch) roslaunch udacity_bot amcl.launch (or my_amcl.launch) rosrun udacity_bot navigation_goal

4.1 Localization Results



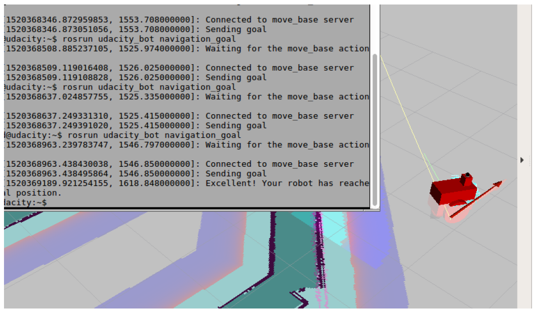
```
want to source ROS in this workspace (y/n): y
rced!
udacity:~$ rosrun udacity_bot navigation_goal
 [1520229583.988066898, 1536.032000000]: Waiting for the move_base action

 [1520229584.201149570, 1536.128000000]: Connected to move_base server
 [1520229584.201198682, 1536.128000000]: Sending goal
 [1520229733.886028935, 1584.129000000]: Excellent! Your robot has reache
oal position.
udacity:~$
```

Fig. 3. Udacity bot reached goal.

4.2 Technical Comparison

The later robot is more unstable and, perhaps, could flipped if controlled incorrectly. But having a sensor mount gives it better sensor reading and solves the problems of mapping own wheels. Both robots have similar localization results, as well as similar trajectories along their global paths.



```
[1520368346.872959853, 1553.708000000]: Connected to move_base server
[1520368346.873051056, 1553.708000000]: Sending goal
udacity:~$ rosrun udacity_bot navigation_goal
[1520368508.885237105, 1525.974000000]: Waiting for the move_base action

[1520368509.119016408, 1526.025000000]: Connected to move_base server
[1520368509.119108828, 1526.025000000]: Sending goal
udacity:~$ rosrun udacity_bot navigation_goal
[1520368637.024857755, 1525.335000000]: Waiting for the move_base action

[1520368637.249331310, 1525.415000000]: Connected to move_base server
[1520368637.249391020, 1525.415000000]: Sending goal
@udacity:~$ rosrun udacity_bot navigation_goal
[1520368963.239783747, 1546.797000000]: Waiting for the move_base action

[1520368963.438430038, 1546.850000000]: Connected to move_base server
[1520368963.438495864, 1546.850000000]: Sending goal
[1520369189.921254155, 1618.848000000]: Excellent! Your robot has reache
al position.
lacity:~$
```

Fig. 4. My bot reached goal.

5 Discussion

While running software in simulation environment is relatively easy, making real robots that would run on the same software is a more complex task. Therefore, a real prototype needs to be considered in order to attempt to tune package parameters for a specific model. Unfortunately, the robotic prototype I am currently working is a boat, that would need a liquid simulator to be run in. In that position it is easier to test localization performance on actual robot instead of simulation environment, especially without access to large computing capabilities.

5.1 Topics

- Which robot performed better? Both robots did a similar job. That is expected given that my_bot parameters were tuned to only accommodate the change in its form, size and weight.
- How would you approach the 'Kidnapped Robot' problem? 'Kidnapped Robot problem' can be approached by slowly placing new particle filters randomly and uniformly throughout the map.
- What types of scenario could localization be performed? The localization described above would do best in predetermined indoor environments.
- Where would you use MCL/AMCL in an industry domain? MCL/AMCL could be best used for indoor warehouse or delivery robots.

6 Conclusion / Future work

For commercial deployment the model should be modified based on actual robot including real sizes, masses and materials, as well as proper electronic equipment. After that is done, parameters can be readjusted to incorporate the new robot.

6.1 Hardware Deployment

1) What would need to be done? A package with proper parameters would need to be installed on the robot's hardware and run in a similar way as in simulation. Given that robot has a true map of the environment it is in, it should start localizing itself correctly.
2) Computation time/resource considerations? This project shows that localization and path planning is not too computationally expensive and can be

run in a VM with 2GB RAM with the simulation environment running. This means that current micro-controllers like Raspberry Pi should be able to perform localization. The better hardware of the robot, the better its performance can be(but keep in mind to update controller and map frequencies).