Robotics Software Engineer Nanodegree Program: Term 2: Project 2: Localization Project - Where Am I?

Li Feng,
acumenta@yahoo.com

ABSTRACT

This report presents robotic localization project through the creation and testing of two robots (one is introduced in the lessons and the other is created by the student) in the ROS, Gazebo, and RViz simulation environment. The Adaptive Monte Carlo Localization algorithm is used to localize the poses of the robots in the provided map. The move_base package and navigation_goal program were used to navigate the robots to the defined goal pose. Parameters tuning on the (global and local) costmaps and planner were practiced to improve robots performance.

1 Introduction

Localization of a robot to find its poses according to the known map is a very big challenge in robotic world. There are four localization alogorithms (i.e. Extended Kalman Filter (EKF), Markov Localization, Grid Localization, and Monte Carlo Localization (MCL)) can be used to filter noisy sensor measurements and track the robot's poses. This project uses Adaptive Monte Carlo Localization (AMCL) package (a variant of MCL) in the ROS environment. It dynamically adjusts the number of particles, uses raw measurements, is more efficient in memory usage and speed performance., and can perform global localization.

Two robot models (i.e. udacity_bot and li_bot) were created in the Gazebo, ROS, and RViz environment. The udacity_bot was given and used as an example guidance in this project. The li_bot was built based on udacity_bot with the addition of two skinny cylinders in the back, and one rectangle box in the front on top of the chassis. The laser rangefinder was moved to the top of the rectangle box. Both were equipped with camera and laser rangefinder. The provided map (jackal_race) was developed by Clearpath Robotics (https://www.clearpathrobotics.com). AMCL package was used to determine the position of the robots in the provided map and move_base package and navigation_goal program were used to navigate the robots to the specified goal position with the correct pose.

2 Background

A mobile robot has to know where it is located in its environment, so that it can move around, avoid obstacles, and accomplish tasks. This process of figuring out the location is called localization. The two most common approaches covered in the classroom for localization are: (1) (Extended) Kalman Filter (EKF), (2) Monte Carlo Localization (MCL) algorithm which is a Particle Filter.

2.1 Kalman Filters

Kalman Filter is very popular and is used in many industries. It is very good at taking in noisy measurements in real time and providing accurate predictions of the position, but it limits its appicability to only linear model of the motion and measurement and unimodal Gaussian distribution of the state space.

The Extended Kalman lter (EKF) solves these limitations through linearizing a nonlinear motion or measurement function using multidimensional Taylor series. (Reference: lesson 10, section 17 EKF) However, EKF's complex mathematics comsumes extensive computational resource.

2.2 Particle Filter

The Monte Carlo Localization (MCL) (or Particle Filter) distributes virtual particles uniformly and randomly over the environment. Measurements are taken and weights of importance are assigned to the particles, particles with least likely robot positions are eliminated during the iterating resampling process.

AMCL is used in this project. AMCL adaptively alters the number of particles used while navigating. Hence, reduces the computational overhead.

2.3 Comparison/Contrast

MCL has several advantages over KF and EKF. For example: MCL is easier to program and uses raw measurements (ie from lasers), is unrestricted by a linear Gaussian states-based assumption, is memory and time efficient, and can perform global localization.

For a comparison chart between MCL and EKF, see Lesson 12, section 3, Power of MCL, MCL vs EKF comparison table.

3 Simulations

The ROS, Gazebo, and RViz simulation environment was used in this project.

3.1 Achievements

udacity_bot and li_bot both achieved the project requirement of reaching the end goal. The goal was achieved by udacity_bot in about 3 minutes and 53 seconds (i.e. average of 3 runs) and the goal was achieved by li_bot in about 4 minutes and 33 seconds (i.e. average of 3 runs). (Reference: 4.1.1 and 4.1.2)

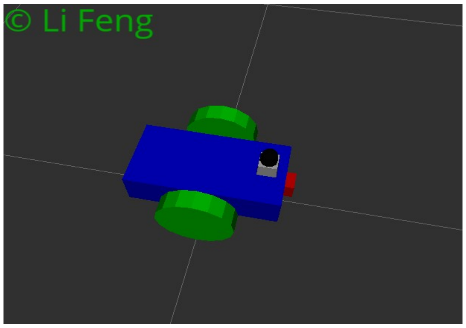
3.2 Benchmark Model - udacity_bot

The udacity_bot was used as a benchmark model.

3.2.1 Model design - udacity_bot

A urdf file (udacity_bot.xacro) was created which defined the layout of the robot. The udacity_bot consisted a cuboidal base (chassis) with two caster wheels. The caster wheels helped stabilize the model. The chassis was a cuboidal (or box), whereas the casters were spherical. Two wheels (left wheel and right wheel) were added to the sides. A camera was attached to the front and a laser rangefinder was arranged on the top of the robot chassis. Three plugins (camera sensor, hokuyo sensor, differential drive controller) were used to bring those devices to work. Below are a list of components in udacity_bot and a picture of the udacity_bot. (reference:./images/udacity_bot/udacity_bot_view.png).

name	shape	size (meter)
chassis	box	0.4 0.2 0.1
back_caster	sphere	0.05 radius
front_caster	sphere	0.05 radius
left_wheel	cylinder	0.1 radius, 0.05 length
right_wheel	cylinder	0.1 radius, 0.05 length
camera	box	0.05 0.05 0.05
laser rangefinder	box	0.1 0.1 0.1



3.2.2 Packages Used - udacity_bot

The following packages were used for the udacity_bot testing environment:

amcl,	move_base,	rviz,
robot_state_publisher,	joint_state_publisher,	gazebo_ros,
ros-kinetic-navigation,	tf (transform tree),	map_server,
camera sensor plugin,	hokuyo sensor plugin,	differential drive controller plugin
navigation_goal,	udacity_bot,	

3.2.3 Parameters - udacity_bot

In the config directory: (applied to both bots)

base_local_planner_params.yaml: meter_scoring was set to true to ensure pdist_scale used meters. pdist_scale was set to 0.1, so that the robot followed the planned path closely.

costmap_common_params.yaml: transform_tolerance was set to 1.2, obstacle_range was set to 1.0, raytrace_range was set to 3.0, robot_radius was set to 0.5, and inflation_radius was set to 0.6. Both robot_radius and inflation_radius were set larger to avoid robot stuck in the crack.

global_costmap_params.yaml: update_frequency was set to 5.0 and publish_frequency was set to 2.0.

local_costmap_params.yaml: update_frequency was set to 5.0, publish_frequency was set to 2.0, width was set to 2.0, and height was set to 2.0. Reduced width and height to eliminate robot from running far from the planned path.

In the launch directory: (applied to both bots)

amcl.launch: min_particles was set to 10 and max_particles to 200. A higher max_particles did not improve the robot to find itself with certainty, only slowed it down. odom_alpha1 to odom_alpha4 were set to 0.001. This smaller number, i.e. 0.001 worked better.

In the urdf directory: (only applied to udacity_bot)

The following modifications only applied to udacity_bot, because udacity_bot was built lower and moved slower due to higher friction with the surface. After the following modifications, udacity_bot moved swiftly. (Note: The li_bot was built higher and heavier and front and back caster provided about the right amount of friction.)

udacity_bot.xacro: the sphere radius of back_caster_collison and front_caster_collison were modified to 0.048 from 0.05 to reduce the friction.
udacity_bot.gazebo: the hokuyo's min range was increased to 1.0 from 0.1 to view better.

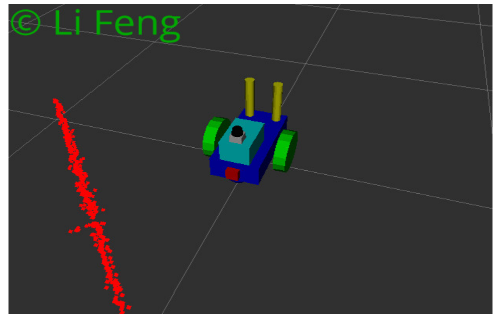
3.3 Student Model - li_bot

The student model of the robot is named as li_bot.

3.3.1 Model design - li_bot

The li_bot was built based on udacity_bot with the addition of two skinny cylinders on the back, and one rectangle box in the front on top of the chassis. The laser rangefinder was moved to the top of the rectangle box. Below are a list of components in li_bot and a picture of the li_bot (reference:./images/li_bot/li_bot_view.png).

name	shape	size (meter)
chassis	box	0.4 0.2 0.1
back_caster	sphere	0.05 radius
front_caster	sphere	0.05 radius
left_wheel	cylinder	0.1 radius, 0.05 length
right_wheel	cylinder	0.1 radius, 0.05 length
camera	box	0.05 0.05 0.05
laser rangefinder	box	0.1 0.1 0.1
topcube	box	0.18 0.12 0.1
leftpole	cylinder	0.02 radius, 0.3 length
rightpole	cylinder	0.02 radius, 0.3 length



3.3.2 Packages Used - li_bot

The following ROS packages were used for the li_bot testing environment:

amcl, move_base, rviz,
robot_state_publisher, joint_state_publisher, gazebo_ros,
ros-kinetic-navigation, tf (transform tree), map_server,
camera sensor plugin, hokuyo sensor plugin, differential drive controller plugin
li_navigation_goal, li_bot,

3.3.3 Parameters - li_bot

The parameters settings for the four files (i.e. base_local_planner_params.yaml, costmap_common_params.yaml, global_costmap_params.yaml, and local_costmap_params.yaml) in the config folder were updated with the same values as in udacity_bot package. (Reference: 3.2.3)

The parameters settings in the li_amcl.launch file were updated with the same values as in amcl.launch file for the udacity_bot. (Reference: 3.2.3)

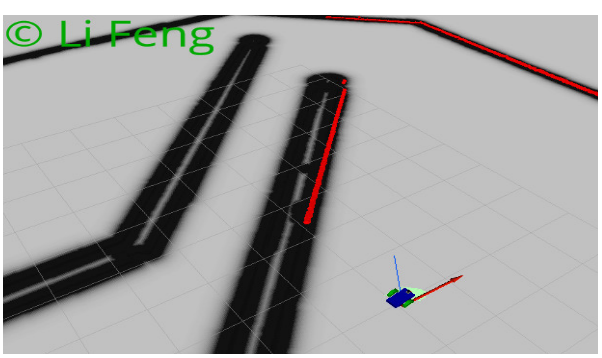
4 RESULTS

4.1 Localization Results

After parameter tuning, both udacity_bot and li_bot run smoothly and swiftly. They made a loop turn at the beginning to align their head to the direction of the path. Then, they just slided along the path. They might deviate from the path a little bit, but switched bak quickly. They didn't go to the wall or stuck at crack. Those wonderful behaviors attributed to these parameters settings: (1) local_costmap: width:2.0, height: 2.0 (2) pdist_scale:0.1. They made the robots following the global path very closely and limited local area of the robot to relatively small size, hence eliminated robots from roaming around aimlessly.

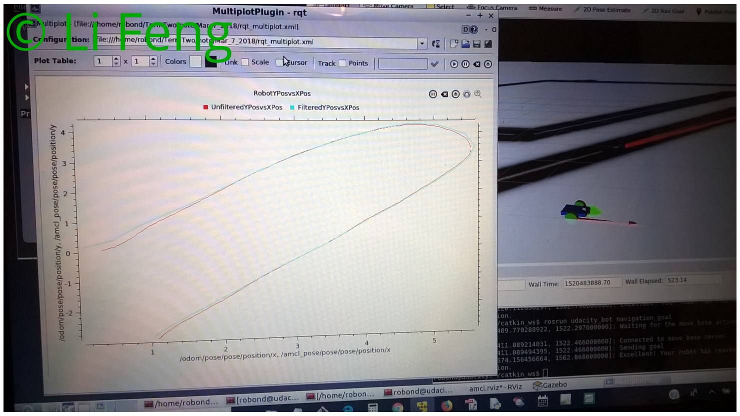
4.1.1 Benchmark - udacity_bot - time taken to reach the goal

Time stamps were captured on udacity_bot reaching goal three times as below:
1st Run: start time: 3/4/2018 16:45:10 end time: 3/4/2018 16:48:57 elapsed time: 0:3:47
2nd Run: start time: 3/4/2018 21:49:09 end time: 3/4/2018 21:53:04 elapsed time: 0:3:55
3rd Run: start time: 3/4/2018 21:55:28 end time: 3/4/2018 21:59:24 elapsed time: 0:3:56
The average elapsed time is 3 minutes and 53 seconds.
Below is a snapshot of the udacity_bot reached the goal.
(reference:./images/udacity_bot/udacity_bot_reached_goal.png).



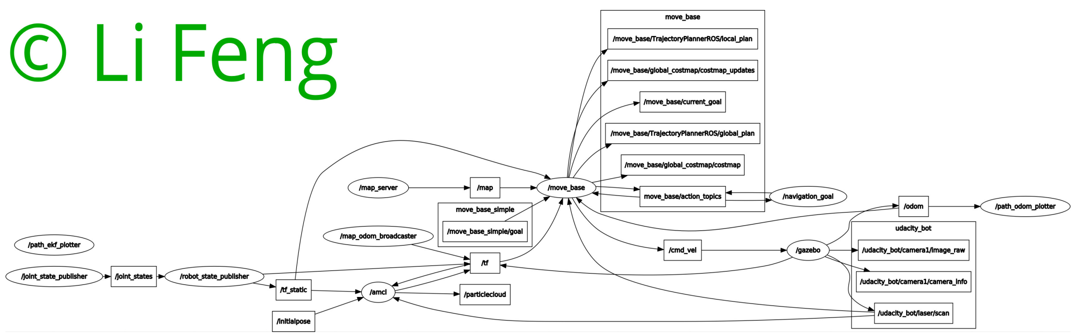
4.1.1 Benchmark - udacity_bot - using rqt_multiplot to plot the filtered and unfiltered robot poses

The "rosrun rqt_multiplot rqt_multiplot" was executed and the resulting graph shown below:
(Reference: ./images/udacity_bot/udacity_bot_plot_path_un_filter.jpg)
The configuration settings were:
(UnfilteredYPosvsXPos - Topics: /odom - Red color) and (FilteredYPosvsXPos - Topics: /amcl_pose - Green color)
At beginning, the filtered and unfiltered poses were further apart. Then both poses (or paths) were closely aligned. But they were slightly apart when adjusting poses at the goal area.



4.1.1 Benchmark - udacity_bot - using rqt_graph to visulize the connections between topics and nodes

The "rosrun rqt_graph rqt_graph" was executed and the resulting graph of active notes and topics were shown below:
(Reference: ./images/udacity_bot/rosgraph_u_note_topic_active.png)



4.1.1 Benchmark - udacity_bot - list of topics

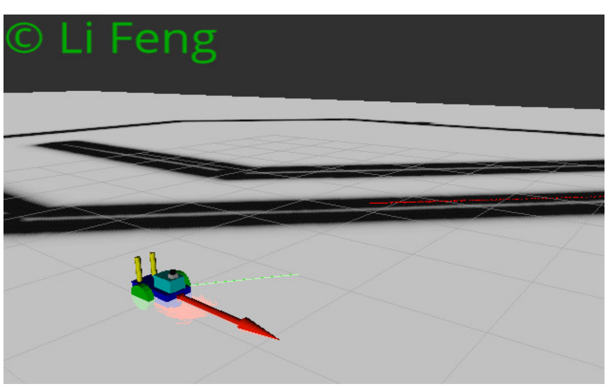
The "rostopic list" was executed and the resulting topic list was in (./images/rostopic_2_bots.docx).

4.1.2 Student - li_bot - time taken to reach the goal

Time stamps were captured on li_bot reaching the goal three times as below:
1st Run: start time: 3/4/2018 15:28:33 end time: 3/4/2018 15:32:53 elapsed time: 0:4:20
2nd Run: start time: 3/4/2018 21:33:44 end time: 3/4/2018 21:37:56 elapsed time: 0:4:12
3rd Run: start time: 3/4/2018 21:40:33 end time: 3/4/2018 21:45:39 elapsed time: 0:5:06
The average elapsed time is 4 minutes and 33 seconds.

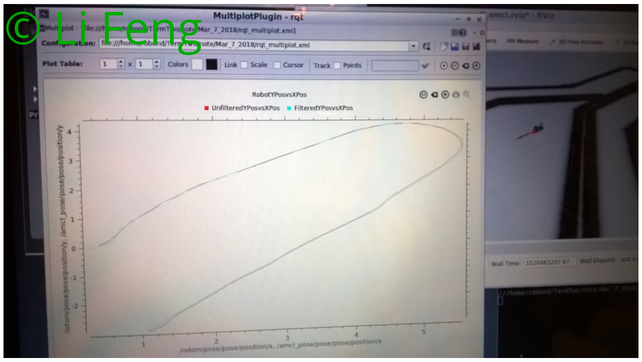
Below was a snapshot of the li_bot reached the goal:./images/li_bot/li_bot_reached_goal.png.

There were 12 snapshots of the li_bot travelling along the path at: ./images/li_bot/path/ folder.



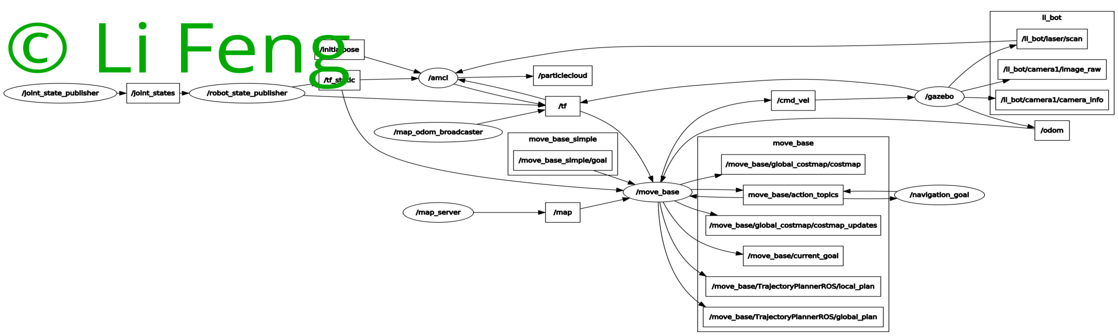
4.1.2 Student - li_bot - using rqt_multiplot to plot the filtered and unfiltered robot poses

The "rosrun rqt_multiplot rqt_multiplot" was executed and the resulting graph shown below:
(Reference: ./images/li_bot/li_bot_plot_path_un_filter.jpg)
The configuration settings were:
(UnfilteredYPosvsXPos - Topics: /odom - Red color) and (FilteredYPosvsXPos - Topics: /amcl_pose - Green color)
The poses of filtered and unfiltered were closely aligned most of the way, except at the end (close to the goal area) slightly apart.



4.1.2 Student - li_bot - using rqt_graph to visulize the connections between topics and nodes

The "rosrun rqt_graph rqt_graph" was executed and the resulting graph of active note and topic were shown below:
(Reference: ./images/li_bot/rosgraph_l_note_topic_active.png)



4.1.2 Student - li_bot - list of topics

The "rostopic list" was executed and the resulting topic list was in (./images/rostopic_2_bots.docx).

4.2 Technical Comparison

Modifications were made on udacity_bot.xacro to creae li_bot.xacro. It added 3 joints (i.e. topcube_joint, leftpole_joint, rightpole_joint) and 3 links (i.e. topcube, leftpole, rightpole). The hokuyo_joint and hokuyo link were updated accordingly to the higher z position. So, li_bot was built higher and heavier and front and back caster values provided about the right amount of friction (i.e. no adjustment needed). Although udacity_bot moved about one minute faster than li_bot.

5. DISCUSSION

The student had encountered many problems when the project was first started to run, for example: VMware crashed repeatedly when the robots were around the round turn pillar (corrected with the right .rviz file), or the robots kept going to the wall and stayed there forever (corrected with pdist_scale and the width, height of localmap), or slow moving robots took forever to get to the goal (corrected with the sphere radius slightly on the front and back casters collision tags, and min laser range in .gazebo file for the udacity_bot), ...etc. So, based on these experience, every little detail mattered and they all integrated together for the robots to work correctly, smoothly, and swiftly.

Both robots performed equally well. It took udacity_bot about 3 minutes to reach goal and took li_bot about 4 minutes to reach the goal. Although the li_bot was about 1 minute slower than udacity_bot, but li_bot was heavier and did not have adjustment in the caster radius.

MCL/AMCL can be used in robotic vacuum cleaner industry domain. It can use AMCL to localize itself and either with the provided map or do the mapping in runtime.

In this project, the student learned the basic skills of being a robotist. They are: how to create a robot model from scratch in the Gazebo, ROS, and RViz environment, how to integrate various nodes, parts, sensors, packages, and program to build a functioning robot, and how to tune parameters in the global and local costmaps and planner.

5.1.1 Topics - Kidnapped Robots

The student had changed robot's positions (x, y) through Gazebo user interface several times. The robots jumped up and down, then settled down at the specified new locations, an updated path drawn, and robots had moved to the goal positions easily. Therefore, according to those observations, AMCL is capable of handling kidnapped robot problem within the same or provided map environment.

5.1.2 Topics - Color of the Robots

The color of the robot in the Gazebo window was specified through gazebo package's predefined colors in the udacity_bot.gazebo. For example:

```
<gazebo reference="chassis">
  <material>Gazebo/Blue</material>
</gazebo>
```

The color of the robot in the rviz window was determined by the color specification in the material tag inside each link visual element. For example:

```
<visual name='chassis_visual'>
  .........
  <material name="blue">
    <color rgba="0 0 0.8 1"/>
  </material>
</visual>
```

5.1.3 Topics - How to solve or remove the yellow Warning messages

Added the following lines in the udacity_bot.gazebo file to resolve the many yellow Warning messages outputted in the terminal window.

```
<publishWheelTF>false</publishWheelTF>
<publishWheelJointState>false</publishWheelJointState>
<rosDebugLevel>na</rosDebugLevel>
<wheelAcceleration>0</wheelAcceleration>
<wheelTorque>5</wheelTorque>
<odometrySource>world</odometrySource>
<publishTf>1</publishTf>
```

6. CONCLUSION/FUTURE WORK

Both robots reached the goals in a relatively speedy fashion (i.e. between 3 to 5 minutes). Both robots followed paths closely and avoided obstacles. These proved that localization (AMCL) was working pretty well in this environment.

There is definitely the trade-offs between accuracy and processing time. Adding sensors and different base sizes could potentially improve the performance.

This project achieved what the student attempted. The student would not deploy it on hardware untill learning further from the class. (Integrating various hardware and software components to work together is probably the main task here.) And it still has a lot more work to do before applying to commercial product.

For future work, the student would like to build a Poppy or Pepper robot in the ROS, Gazebo, Rviz environment.

REFERENCES

[1] Simulation Environment for Mobile Robots Testing Using ROS and Gazebo by K. Takaya, T. Asai, V. Kroumov, and F. Smarandache
[2] ROS Navigation Tuning Guide by K. Zheng
[3] wiki.ros.org/Packages
[4] wiki.ros.org/base_local_planner
[5] wiki.ros.org/move_base
[6] wiki.ros.org/base_local_planner#Parameters
[7] Tools for dynamics simulation of robots: a survey based on user feedback by S. Ivaldi, V. Padois, and F. Nori
[8] http://moorerobots.com/blog and https://www.youtube.com/watch?v=8ckSl4MbZLg by Richard Wang
[9] The vSLAM Algorithm for Robust Localization and Mapping by N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. Munich