Robot Localization with ROS AMCL Package

Abstract *- This paper uses implementation of particle filter localization algorithm in ROS called Adaptive Monte Carlo Localization (AMCL). This algorithm was used to estimate and track poses of the two (2) different simulated mobile robots relative to a known map of an environment while they navigate through the map..*

Key words: Robot, Localization, AMCL, ROS, Kalman Filter

1 INTRODUCTION

The robot localization is the process of determining where the robot is with respect to its environment. There are three (3) types of localization viz position tracking, global localization and kidnapped robot problem.

In position tracking robot knows its initial pose and the localization challenge entails estimating its pose as it moves out in the environment.

In the case of global localization the robot initial pose is unknown and the robot must determine its pose relative to the ground truth map.

Finally, the kidnapped robot problem is similar to the global localization problem except that the robot may be kidnapped at any time and moved to a new location on the map. The challenge is that, the robot has to recover from such experience and once again correctly localize itself on the map.

This paper is attempting to solve global localization problem where two different simulated mobile robots are programmed to localize themselves relative to the known as they navigate through the map.

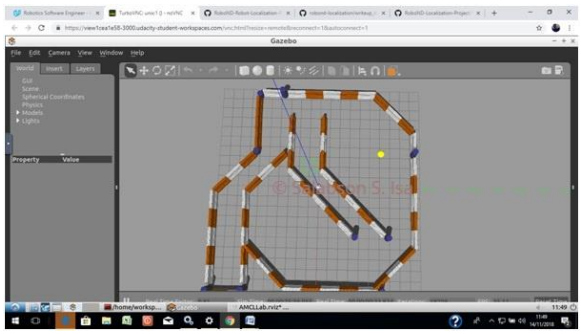

Figure 1: Gazebo world

2 BACKGROUND

In a real world, robot movement is imprecise due imperfection in the terrain, slip in robot wheel, and other adverse environmental factors. Therefore, a robot needs a way to sense its own action, however, the sensory data is often uncertain as sensor measurement contains some noise.

Since both robot movement and sensory measurements are uncertain various localization algorithms were developed enable us filter noise and uncertainty in order to make better estimate of the robot pose:

2.1 **Kalman Filter (KF)**

Kalman filter is an amazing estimation algorithm in order to estimate predicted values. It is an iterative mathematical process that uses a set of equations and consecutive data inputs to quickly estimate the true value, position, velocity etc of the object being measured, when the measured values contain unpredicted or random error, uncertainty or variation. Kalman filter uses Gaussian distribution to model the uncertainty with assumption that motion and measurement models are linear. Therefore Kalman filter can only be applied to linear problems.

2.2 **Extended Kalman Filter (EKF)**

Extended Kalman Filter (EKF) algorithm addresses the limitation of Kalman Filter(KF) by handling non-linear motion and measurement models by taking a local linear approximation and use this approximation to update covariance of the estimation.

2.3 **Monte Carlo Localization (MCL)**

Monte Carlo Localization also referred to as particle filter. It uses particles to localize a robot. Each particle has a position and orientation which represent a guess of where the robot might be located. These particles are resample each time the robot moves and sense its environment. MCL is limited to only local and global localization, therefore, it cannot handle the kidnapped robot problem.

2.4 Compare and contrast between Kalman Filter and Monte Carlo Localization

	MCL	**EKF**
Measurement	Raw Measurement	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency (Memory)		
Efficiency(Time)		
Ease of implementation		
Resolution		
Robustness		
Memory & resolution Control		
Global localization		
State space	Multimodal Discrete	Unimodal Gassuian

Table 1: Comparison: MCL and EKF

3 SIMULATIONS: ROBOT MODELS

In this project ROS packages are used to accurately localize two different mobile robot models namely udacity_bot and my_bot in Gazebo and Rviz with the provided map of an environment.

3.1 **Benchmark Model: udacity_bot**

3.1.1 Model Design

This is simple mobile robot with a rectangular chassis having two differential driven wheels one at the left side and the other at right side. Two caster wheels are added one at the front and the other at back to give it a balance. A Hokuyo laser is

located at top front while camera is fixed at front side.

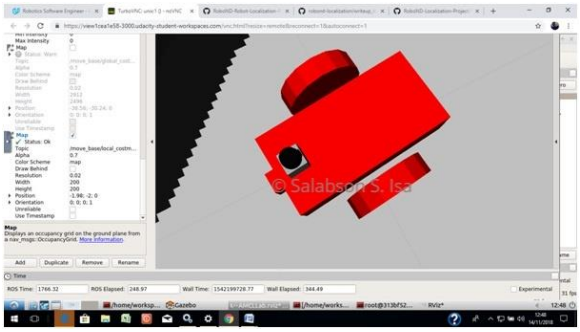

Figure 2: udacity_bot model

Parts	Type	Size
Chassis	Link	Box:0.4x0.2x0.1
Left wheel	Link	Cylinder r=0.1 l=0.05
Left wheel hinge	Continuous joint	N.A
Right wheel	Link	Cylinder r=0.1 l=0.05
Right wheel hinge	Continuous joint	Not applicable
camera	Link	Box: 0.05x0.05x0.05
Camera joint	Fixed joint	Not applicable
Hokuyo laser	Link	Mesh:0.1x0.1x0.1
Hokuyo joint	Fixed joint	Not applicable
Front caster		
Back caster		

Table 2: udacity_bot model design

3.1.2 **Packages used**

ROS navigation packages which include AMCL and Move Base are utilized to move the robot to the goal location while localizing itself.

3.1.3 **Parameters**

Table 3 shows localization parameters for AMCL for udacity_bot.

The minimum and maximum numbers of particles are set to 10 and 300 respectively. Considerable number of particles are needs for better accuracy. However, larger the number of particles higher the computing power required. While odom_alpha from 1 to 4 are all set to 0.2 which represent expected noise in odometry rotation.

While Table 4 show path planning parameters for move_base node. Global costmap parameters help to generate a long-term path for the robot while local costmap parameters help to generate a short-term path for the robot.

AMCL	
General Parameters	
Parameter name	**Parameter value**
min_particles	10
max_particles	300
transform_tolerance	0.2
initial_pose_x	0.0
initial_pose_y	0.0
initial_pose_a	0.0
Laser Parameters	
laser_model_type	Likelihood_field
laser_z_hit	0.997
laser_z_rand	0.003
laser_max_beam	90
Odometry Parameters	
odom_frame_id	odom
odom_model_type	diff
odom_alpha_1	0.2

odom_alpha_2	0.2
odom_alpha_3	0.2
odom_alpha_4	0.2

Table 3: udacity_bot AMCL parameters

Move Base	
Base local planner	
Parameter name	Parameter value
holomonic_robot	false
Meter_scoring	true
yaw_goal_tolerance	0.10
xy_goal_tolerance	0.10
occ_dist_scale	5.0
sim_time	1.0
pdist_scale	0.6
Common costmap	
map_type	costmap
obstacle_range	3.0
raytrace_range	2.5
inflation_radius	0.2
transform_tolerance	0.2
footprint	[[0.2,0.2], [0.2,-0.2],[-0.2,-0.2],[-0.2,0.2]]
observation_source	laser_scan_sensor
laser_scan_sensor	
Global costmap	
global_frame	map
robot_base_frame	robot_footprint
udate_frequency	10.0
publish_frequency	10.0
width	50.0
height	50.0
resolution	0.02
static_map	true
rolling_window	false
Local costmap	
global_frame	odom
robot_base_frame	robot_footprint
udate_frequency	10.0
publish_frequency	10.0
width	4.0
height	4.0

resolution	0.02
static_map	false
rolling_window	true

Table 4: udacity_bot move_base parameters

3.2 **Customized Model: my_bot**

3.2.1 Model design

This robot is built upon the udacity_bot. Two more wheels were added to make it four-wheeled mobile robot while the two caster wheels are removed. The location of Hokuyo laser and camera are unchanged. Finally the chassis is slightly adjusted to fit the four wheels.

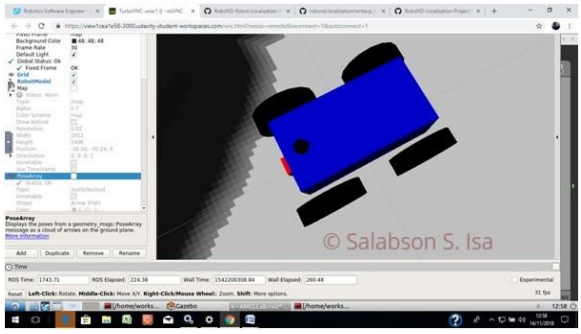

Figure 3: my_bot model

Parts	Type	Size
Chassis	Link	Box:0.4x0.2x0.15
Front left wheel	Link	Cylinder r=0.1 l=0.05
Front left wheel hinge	Continuous joint	Not applicable
Front right wheel	Link	Cylinder r=0.1 l=0.05
Front right wheel	Continuous joint	Not applicable

hinge		
Back left wheel	Link	Cylinder r=0.1 l=0.05
Back left wheel hinge	Continuous joint	Not applicable
Back right wheel	Link	Cylinder r=0.1 l=0.05
Back right wheel hinge	Continuous joint	Not applicable
camera	Link	Box: 0.05x0.05x0.05
Camera joint	Fixed joint	Not applicable
Hokuyo laser	Link	Mesh:0.1x0.1x0.1
Hokuyo joint	Fixed joint	Not applicable

Table 5: my_bot model design

3.2.2 **Packages used**

ROS navigation packages which include AMCL and Move Base are utilized to move the robot to the goal location while localizing itself.

3.3.3 **Parameters**

Table 6 shows localization parameters for AMCL node my_bot.

As shown in Table 6 most of the AMCL parameters used were from udacity_bot with few changes. The odom_model_type is changed to "omni" to fit 4-wheel robot while odom_alpha_5 is added as it use by omni odom model type.

In the same vein, few move base parameter are changed to accommodate slight change in model design where inflation rate is set to 0.3 and footprint to [[0.2,0.25], [0.2,0.25],[0.2,0.25],[0.2,0.25]] and laser scan topic to "my_bot/laser/scan"

AMCL	
General Parameters	
Parameter name	**Parameter value**
min_particles	10
max_particles	200
transform_tolerance	0.2
initial_pose_x	0.0
initial_pose_y	0.0
initial_pose_a	0.0
Laser Parameters	
laser_model_type	Likelihood_field
laser_z_hit	0.997
laser_z_rand	0.003
laser_max_beam	90
Odometry Parameters	
odom_frame_id	odom
odom_model_type	omni
odom_alpha_1	0.2
odom_alpha_2	0.2
odom_alpha_3	0.2
odom_alpha_4	0.2
odom_alpha_5	0.2

Table 6: my_bot AMCL parameters

Move Base	
Base local planner	
Parameter name	Parameter value
holomonic_robot	false
Meter_scoring	true
yaw_goal_tolerance	0.10
xy_goal_tolerance	0.10

occ_dist_scale	5.0
sim_time	1.0
pdist_scale	0.6
Common costmap	
map_type	costmap
obstacle_range	3.0
raytrace_range	2.5
inflation_radius	0.2
transform_tolerance	0.2
footprint	[[0.2,0.25], [0.2,-0.25],[-0.2,-0.25],[-0.2,0.25]]
observation_source	laser_scan_sensor
laser_scan_sensor	
Global costmap	
global_frame	map
robot_base_frame	robot_footprint
udate_frequency	10.0
publish_frequency	10.0
width	50.0
height	50.0
resolution	0.02
static_map	true
rolling_window	false
Local costmap	
global_frame	odom
robot_base_frame	robot_footprint
udate_frequency	10.0
publish_frequency	10.0
width	4.0
height	4.0
resolution	0.02
static_map	false
rolling_window	true

Table 7: my_bot move_base parameters

4 **RESULT**

Using provided C++ node, both the udacity_bot and my_bot achieved the localization result by successfully navigating from the start position to goal position as shown in figure 6 and 7. It took udacity_bot about two minutes to reach its goal while it took my_bot more than five minutes to reach the same goal.

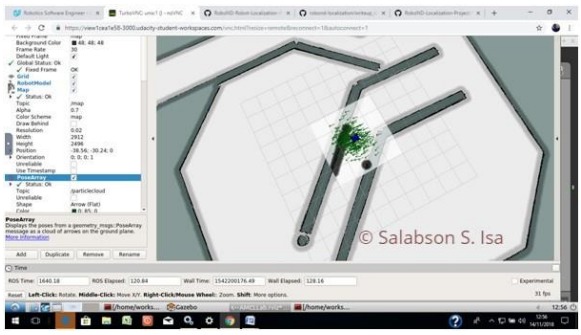

Figure 4: mybot at the start position

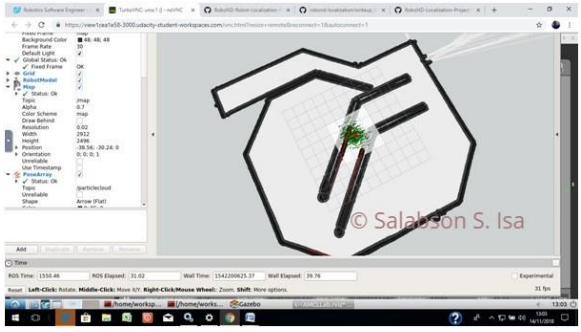

Figure 5: udacity_bot at the start postion

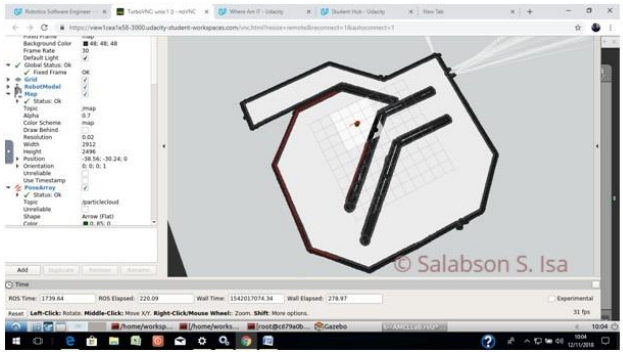

Figure 6:udacity_bot at the goal position

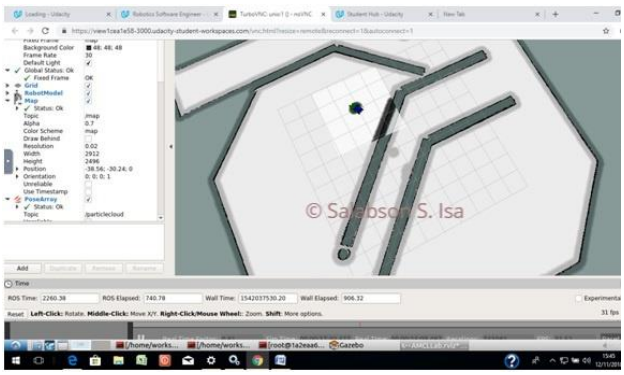

Figure 7: my_bot at the goal position

5. DISCUSSION

The performance of the udacity_bot was better than that of my_bot, as my_bot get closer to the goal it took longer time to adjust its pose. This is because it uses the omni drive which is not as flexible as differential drive.

6 CONCLUSION/FUTURE WORK

In conclusion both robots were successfully able to reach the goal position. The Parameters could be future tuned to improve their performances. In future this will be deploy on the real hardware to the performance in the real world.

REFERENCES

[1] Udacity, Robotics software Engineer classroom, lesson 9: Localization. 2018

[2] *YoonSeok, HanCheol RyuWoon, TaeHoon. "ROS Robot Programming", ROBOTIS Co., Ltd, 2017.*

[3] www.lecturesonline.com, "Special lecture: kalman filter"