Localization Project

Snigdha Dongre

Abstract— The paper aims at describing the building of two mobile robots with URDF and the implementations of Adaptive Monte Carlo Localization package in ROS to localize these mobile robots pose and navigate to a pre-defined position inside a provided map in a simulated Rviz and Gazebo environment. The paper will also discuss and compare Kalman and particle filters.

Index Terms—Robot, Udacity, deep learning.

F

1 INTRODUCTION

In order for a robot to navigate to the desired position, it is necessary to determine the exact position of the robot relative to its frame of reference using sensors information. To reach the goal, the robot has to be equipped with sensors suitable to localize the robot throughout its path. Sensors noise greatly restricts the consistency of sensor's readings in the same environment, so in order to get more accurate data to localize a robot, it is suggested to use more sensors and readings. Localization is done by using a probabilistic algorithm to filter sensor's noisy data and track the robot's position. There are 4 popular algorithms which are used for localization, Extended Kalman Filter, Markov Localization, Grid Localization and Monte Carlo Localization. In this paper, Extended Kalman filter and Monte Carlo Localization will be discussed.

In this project localization is performed using ROS packages and the Adaptive Monte Carlo localization algorithm to determine the position of two robots, one is provided by udacity which is known as the udacity_bot and the second one is designed from scratch for the localization project and known as my_robot. The main objective of this project is to tune the specific parameters corresponding to each package for both the robots to best localize the robots and to help them navigate to a pre-defined goal position in the Gazebo and RVIZ simulated environment.

2 BACKGROUND / FORMULATION

Localization is one of the most important tasks for a mobile robot. For a mobile robot to navigate to the desired goal position it has to know its exact starting position to plan a path to reach the desired destination. In order to do so, we use some probabilistic algorithm which was mentioned above. This paper will be mainly focusing on the Kalman Filter and Monte Carlo Localization.

2.1 Kalman Filter

Kalman filter uses linear quadratic estimation to estimate unknown variables by taking the input with statistical noise and inaccuracies from more than one sensor (sensor fusion) and estimate a joint probability distribution to accurately estimate the variables in real time. Since most systems are nonlinear, later extended Kalman filters were introduced which is the non-linear version of the Kalman Filter.

The first step is the initial state prediction. The robot then collects data from the sensors and then during the observation step all the relevant features are extracted. Simultaneously, based on the initial state prediction on the map, the algorithm generates a measurement update which identifies all the relevant elements that the robot expects to find and their position within the map. The algorithm then gives the best-optimized solution by matching the features extracted during the observation and the expected features during the measurement update. These two steps repeat itself over and over again and finally, we get the location of the robot relative to the map. For a non-linear system, the Jacobian is evaluated with current state prediction and then used with the Kalman filter to linearize the non-linear function around the current estimate.

2.2 Particle Filter

Monte Carlo localization is known to be a particle filter as it uses particles to localize the robot locally or globally. In this method, particles are distributed uniformly throughout the map and assigned weights which is the difference between the actual position of the robot and the predicted pose of the robot. These weights represent the probability of the particle being sampled from the probability density function, eliminating the particles with less weight. Then, when the robot moves around, particles are again resampled based on their current state and the action of the robot using Bayesian estimation. The particles with negligible weights are replaced by new particles which are nearer to the particles with higher weights. That means the particles with higher weights have more chances of survival.

2.3 Comparison

The Extended Kalman filter and Monte Carlo filter are briefly compared below in the table.

	MCL	EKF
Measurement	Raw Measurements	Landmarks
Measurement noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency (memory)	Ok	Good

Efficiency (time)	Ok	Good
Ease of implementation	Good	Ok
Resolution	Ok	Good
Robustness	Good	Poor
Memory and resolution control	Yes	No
Global localization	Yes	No
State space	Multimodal Discrete	Unimodal Continuous

3 Model Configuration

In this project, ROS packages are used to launch the robot models in the gazebo world and packages like AMCL and navigation stack to localize the robots accurately and navigate to a pre- defined goal by tuning the parameters corresponding to these packages. For this two mobile robots are built udacity_bot and my_robot with URDF description. For the first robot the udacity_bot link, inertial, collision, visual and joints are defined for box chassis, spherical back and front caster, cylindrical left and right wheels, cubical camera, and laser rangefinder. For the second robot my_robot link, inertial, collision, visual and joints are defined for cylindrical chassis, spherical back and front caster, cylindrical left and right wheels, cubical camera and cylindrical laser rangefinder. The table below shows the details of the components of both the robots.

Component	**Udacity_bot**	**My_robot**
Chassis	Geometry-box Position- xyz = (0,0,0) and rpy = (0,0,0) Dimension- l=0.4 , b=0.2 , h= 0.1 Mass- 15 Joint position -xyz = (0,0,0) and rpy = (0,0,0)	Geometry-cylindrical Position-xyz = (0,0,0) and rpy = (0,0,0) Dimension- r =0.2, L= 0.12 Mass- 15 Joint position - xyz = (0,0,0) and rpy = (0,0,0)
Back caster	Geometry-spherical Position- xyz = (-0.15,0,-0.05) and rpy = (0,0,0) Dimension-- r = 0.05	Geometry- spherical Position- xyz = (-0.15,0,-0.05) and rpy = (0,0,0) Dimension- r = 0.05
Front caster	Geometry- spherical Position- xyz = (0.15,0,-0.05) and rpy = (0,0,0) Dimension- r = 0.05	Geometry- spherical Position- xyz = (0.15,0,-0.05) and rpy = (0,0,0) Dimension- r = 0.05
Right wheel	Geometry-cylindrical Position-xyz = (0,0,0) and rpy =(0,1.5707,1.5707) Dimension- r = 0.1, L= 0.05 Mass- 5 Joint position - xyz = (0,-0.15,0) and rpy = (0,0,0)	Geometry- cylindrical Position- xyz = (0,0,0) and rpy = (0,1.5707,1.5707) Dimension- r = 0.1, L= 0.05 Mass- 5 Joint position - xyz = (0, -0.2, 0) rpy=(0, 0, 0)
Left wheel	Geometry-cylindrical Position- xyz = (0,0,0) and rpy = (0,1.5707,1.5707) Dimension-- r = 0.1, L= 0.05 Mass- 5 Joint position - xyz = (0,0.15,0) and rpy = (0,0,0)	Geometry- cylindrical Position - xyz = (0,0,0) and rpy = (0,1.5707,1.5707) Dimension- r = 0.1, L= 0.05 Mass- 5 Joint position - xyz = (0, 0.2, 0) rpy=(0, 0, 0)
Camera	Geometry - box Position- xyz- (0,0,0) and rpy-(0,0,0) Dimension- l= 0.05 , b = 0.05 , h = 0.05 Mass- 0.1 Joint position - xyz = (0.2,0,0) and rpy = (0,0,0)	Geometry - box Position - xyz- (0,0,0) and rpy-(0,0,0) Dimension- l = 0.05 , b= 0.05, h = 0.05 Mass- 0.1 Joint position – xyz = (0.2, 0, 0) rpy=(0, 0, 0)
Laser rangefinder	Geometry - box Position- xyz- (0,0,0) and rpy-(0,0,0) Dimension- l= 0.1 , b = 0.1 , h = 0.1 Mass- 0.1 Joint position - xyz = (0.15,0,0) and rpy = (0,0,0)	Geometry - cylindrical Position- xyz- (0,0,0) and rpy-(0,0,0) Dimension- r = 0.03, L= 0.1 Mass- 0.1 Joint position – xyz = (0.15, 0, 0.1), rpy= (0, 0, 0)

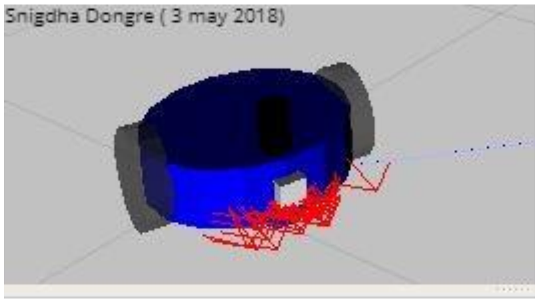
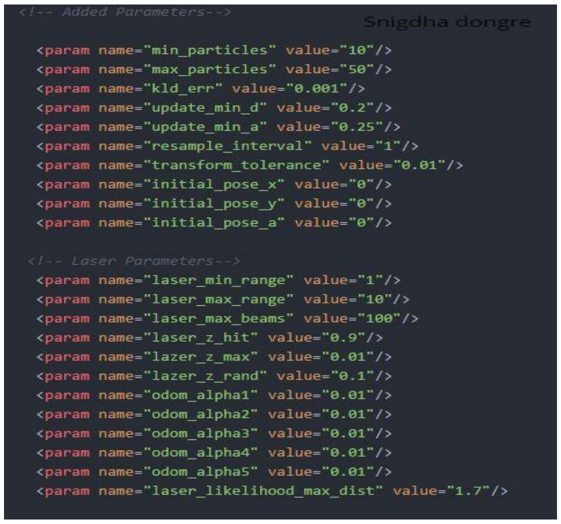


Fig 3.1 (my_robot)

3.1 Parameter tuning

AMCL is responsible for the localization of the robots as it adjusts the number of particles over a time period when the robot is navigating in the map. The fig shows the parameters added to the amcl file –



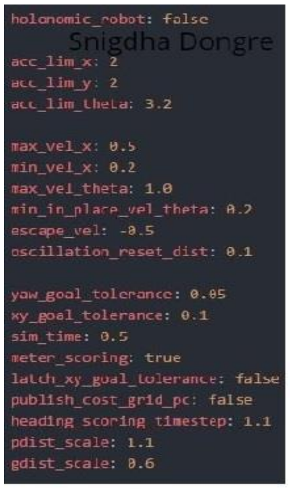
```
<!-- Added Parameters-->

  <param name="min_particles" value="10"/>
  <param name="max_particles" value="50"/>
  <param name="kld_err" value="0.001"/>
  <param name="update_min_d" value="0.2"/>
  <param name="update_min_a" value="0.25"/>
  <param name="resample_interval" value="1"/>
  <param name="transform_tolerance" value="0.01"/>
  <param name="initial_pose_x" value="0"/>
  <param name="initial_pose_y" value="0"/>
  <param name="initial_pose_a" value="0"/>

 <!-- Laser Parameters-->
  <param name="laser_min_range" value="1"/>
  <param name="laser_max_range" value="10"/>
  <param name="laser_max_beams" value="100"/>
  <param name="laser_z_hit" value="0.9"/>
  <param name="lazer_z_max" value="0.01"/>
  <param name="lazer_z_rand" value="0.1"/>
  <param name="odom_alpha1" value="0.01"/>
  <param name="odom_alpha2" value="0.01"/>
  <param name="odom_alpha3" value="0.01"/>
  <param name="odom_alpha4" value="0.01"/>
  <param name="odom_alpha5" value="0.01"/>
  <param name="laser_likelihood_max_dist" value="1.7"/>
```

Fig 3.2

AMCL node can be configured using three categories of ROS parameters: overall filter, laser model, and odometery model. The min and the max particles are the minimum or maximum particles allowed and here they are set to a small value to minimize the computational burden on CPU and to improve the speed of the process. kld_err is the maximum error between the true distribution and the estimated distribution which should be minimum to get a good result. transform_tolerance is the time with which to post-date the transform that is published, to indicate that this transform is valid into the future was also kept low to reduce computational time. Laser_max range is the maximum scan to be considered. Laser_max_beams is the number of evenly-spaced beams in each scan to be used when updating the filter and its value is kept higher than the default value to increase the accuracy. Remaining values like for laser_z_max which is the mixture weight for the z_hit part of the model and laser_z_rand which is mixture weight for the z_rand part of the model were chosen by trial and error method. Odom_alpha1 specifies the expected noise in odometry's translation estimate from the rotational component of robot's motion. Odom_alpha2 specifies the expected noise in odometry's rotational estimate from the translation component of robot's motion. Odom_alpha3 specifies the expected noise in odometry's translation estimate from the translational component of robot's motion. Odom_alpha4 specifies the expected noise in odometry's rotational estimate from the rotational component of robot's motion. Odom_alpha5 translation- related noise parameter. The value for all the odom_alpha value lower than the default value gave better results.

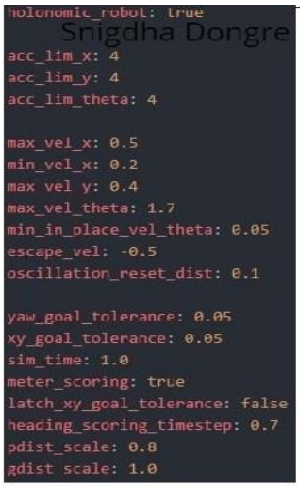
Navigation stack is used to produce the best path to navigate to a required destination by processing the data from odometry, sensors, and map of the environment. For this, the following topics are to updated are tuned- base_local_planner, global_planner, local_planner, costmap_param

Base_local_planner – It is responsible for providing a controller that drives a mobile robot base in the plane and helps path planner to connect to the robot.



```
holonomic_robot: false

acc_lim_x: 2
acc_lim_y: 2
acc_lim_theta: 3.2

max_vel_x: 0.5
min_vel_x: 0.2
max_vel_theta: 1.0
min_in_place_vel_theta: 0.2
escape_vel: -0.5
oscillation_reset_dist: 0.1

yaw_goal_tolerance: 0.05
xy_goal_tolerance: 0.1
sim_time: 0.5
meter_scoring: true
latch_xy_goal_tolerance: false
publish_cost_grid_pc: false
heading_scoring_timestep: 1.1
pdist_scale: 1.1
gdist_scale: 0.6
```

base_local_planner_param



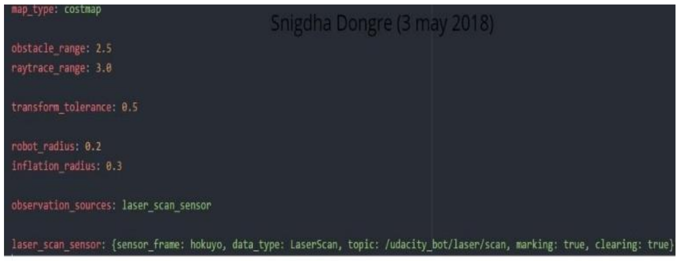
```
holonomic_robot: true

acc_lim_x: 4
acc_lim_y: 4
acc_lim_theta: 4

max_vel_x: 0.5
min_vel_x: 0.2
max_vel_y: 0.4
max_vel_theta: 1.7
min_in_place_vel_theta: 0.05
escape_vel: -0.5
oscillation_reset_dist: 0.1

yaw_goal_tolerance: 0.05
xy_goal_tolerance: 0.05
sim_time: 1.0
meter_scoring: true
latch_xy_goal_tolerance: false
heading_scoring_timestep: 0.7
pdist_scale: 0.8
gdist_scale: 1.0
```

my_robot_base _local_planner_param

Fig 3.3

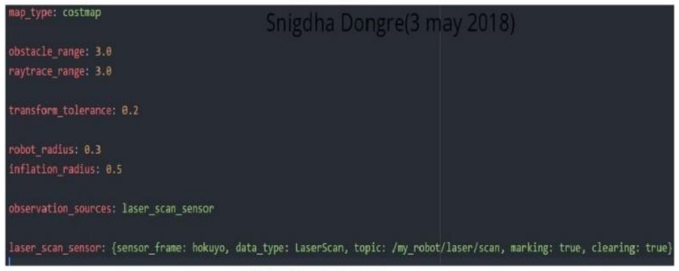
The holonomic_robot for udacity_bot was given to be false which means that no strafing velocity commands will be issued, while in case of my_robot default status 'true' is chosen which means strafing velocity commands may be issued to the base. In navigation the translational and rotation velocity and acceleration is important. After experimenting with different values of these parameter for both robots, the values near to the default values were chosen. In case of my_robot higher value of acceleration, parameters increased the ease of motion for this robot. Escape velocity (negative velocity) was also provided for the robot to move in the backward direction if stuck. The xy_goal_tolerance and yaw_goal_tolerance were specified to make up for any small difference in the target position. These parameters are specified so that the robot does not oscillate near to the goal position, as it is confused and is unable to find the goal position. This happens when the tolerances are not set in the appropriate range. Pdist_scale helps the robot to stay close to the path to follow and gdist_scale is the weighting for how much the controller should attempt to reach its local goal, both of these values are set higher than the default values to get better results.

Costmap_param – costmap is composed of 3 main layers – static map layer, obstacle map layer, and inflation layer. Static map layer interprets the given map provided to the navigation. Obstacle map layer includes the 2d and 3d obstacles and inflation layer calculates the cost for each 2d costmap cell after inflating the obstacles.



```
map_type: costmap

obstacle_range: 2.5
raytrace_range: 3.0

transform_tolerance: 0.5

robot_radius: 0.2
inflation_radius: 0.3

observation_sources: laser_scan_sensor

laser_scan_sensor: {sensor_frame: hokuyo, data_type: LaserScan, topic: /udacity_bot/laser/scan, marking: true, clearing: true}
```

costmap_common_param



```
map_type: costmap

obstacle_range: 3.0
raytrace_range: 3.0

transform_tolerance: 0.2

robot_radius: 0.3
inflation_radius: 0.5

observation_sources: laser_scan_sensor

laser_scan_sensor: {sensor_frame: hokuyo, data_type: LaserScan, topic: /my_robot/laser/scan, marking: true, clearing: true}
```

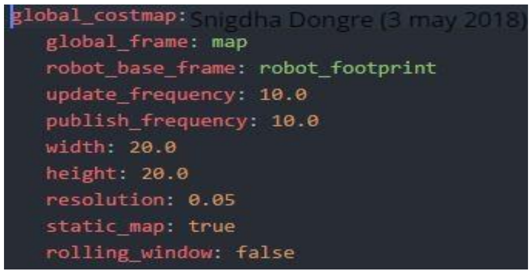
my_robot_costmap_common_param

Fig 3.4

Obstacle layer is responsible for marking 2d and 3d obstacles in the costmap by interpreting the data obtained from laser scans and other sensors. Ray tracing is used in obstacle layer to determine different types of obstacles. For this, parameters like obstacle_range which update its map with information about obstacles that are within value provided in meters of the base and raytrace_range which attempt to clear out space in front of it up to set value in meters away given a sensor reading, are set. These parameters can be over-ridden on a per-sensor basis. Inflation_radius controls how far away the zero cost point is from the obstacle and therefore helps to determine inflation. The best inflation is obtained when the robot is as far as possible from the obstacle from both sides. Transform_tolerance is maximum latency allowed between the transforms and is therefore responsible to update tf tree. The tf tree should be updated at the required rate, if not the navigation stack will stop the robot.

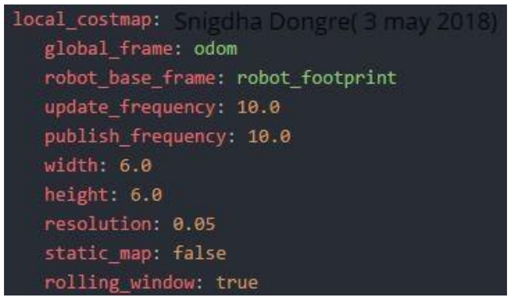
Then there is global costmap which is generated by inflating the obstacles on the provided map and local costmap is generated by inflating obstacles detected by the robot's sensors in real time. The global costmap parameters define the coordinates in which the costmap should run in and used for global planning. While the local costmap is used to avoid obstacles. In both costmaps following parameters are specified-

Update_frequency which is the frequency in Hz at which the costmap will run its update loop. Publish_frequency which is the frequency in Hz at which the costmap will publish visualization information. The "width," "height," and "resolution" parameters set the width (meters), height (meters), and resolution (meters/cell) of the costmap. It is enough to keep the resolution same as the resolution of the map provided to navigation stack. Static_map parameter determines whether or not the costmap should initialize itself based on a map served by the map_server. rolling_window parameter to be true means that the costmap will remain centered around the robot as the robot moves through the world.



```
global_costmap:   Snigdha Dongre (3 may 2018)
  global_frame: map
  robot_base_frame: robot_footprint
  update_frequency: 10.0
  publish_frequency: 10.0
  width: 20.0
  height: 20.0
  resolution: 0.05
  static_map: true
  rolling_window: false
```

Fig 3.5

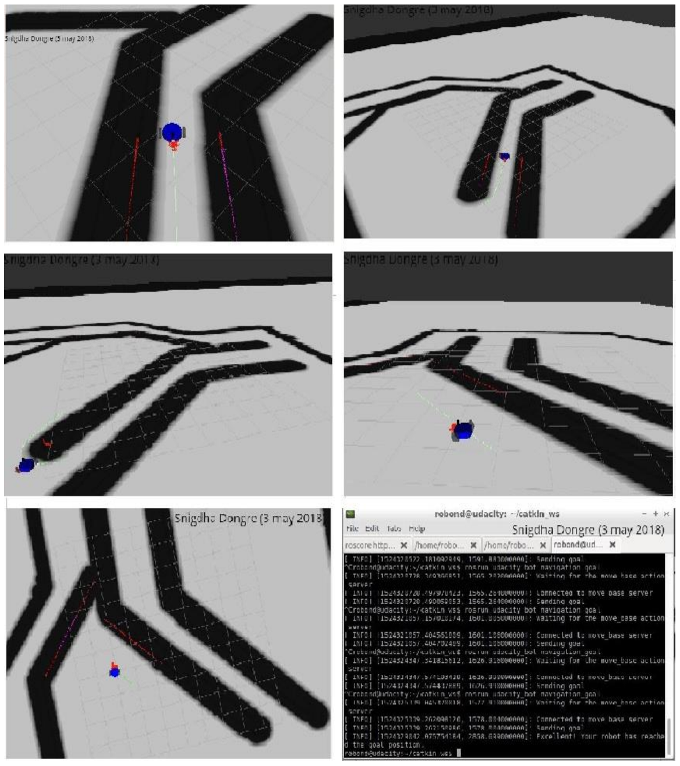


```
local_costmap:   Snigdha Dongre( 3 may 2018)
  global_frame: odom
  robot_base_frame: robot_footprint
  update_frequency: 10.0
  publish_frequency: 10.0
  width: 6.0
  height: 6.0
  resolution: 0.05
  static_map: false
  rolling_window: true
```

Fig 3.6

4 RESULTS

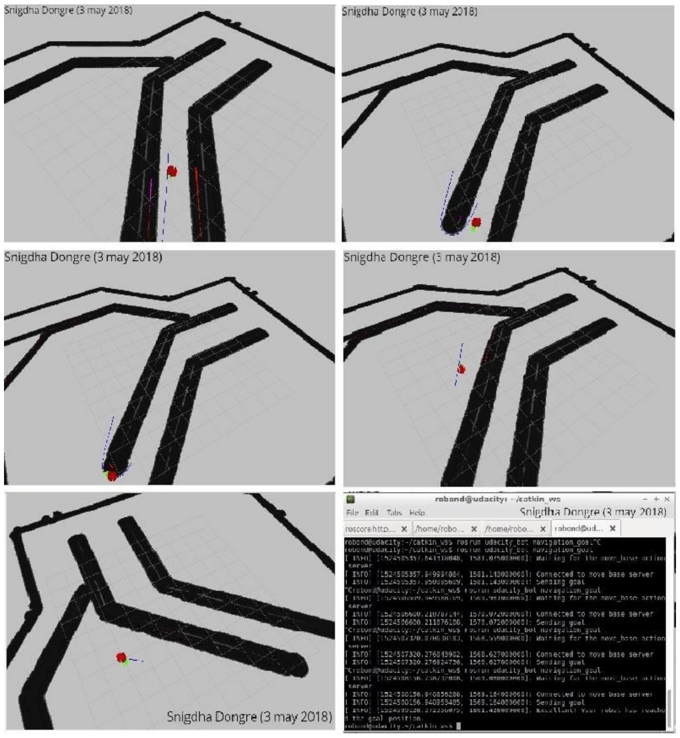
Localization was successfully implemented on both robots - udacity_bot, and my_robot using ROS packages. The robots successfully localized itself in the provided map and navigated to the pre-defined goal position. Both robots closely followed the path and after reaching the goal position successfully generated the message "Excellent! Your robot has reached the goal position". Below images shows the successful navigation of both robots towards the pre-defined goal position and the message generated for both the robots on reaching the goal pose.


Snigdha Dongre (3 may 2018)
robond@udacity: ~/catkin_ws
File Edit Tabs Help
roscore http... /home/robo... /home/robo... robond@ud...

```
[ INFO] [1524328522.181092849, 1591.883000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1524328728.369306851, 1565.282000000]: Waiting for the move_base action
 server
[ INFO] [1524328720.497970123, 1565.264000000]: Connected to move base server
[ INFO] [1524328720.490052853, 1565.264000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1524321057.157018174, 1601.085000000]: Waiting for the move_base action
 server
[ INFO] [1524321057.404561089, 1601.188000000]: Connected to move_base server
[ INFO] [1524321057.404702489, 1601.188000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1524324347.341815812, 1626.818000000]: Waiting for the move_base action
 server
[ INFO] [illegible]: Connected to move_base server
[ INFO] [1524324347.574417885, 1626.[illegible]]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [illegible]: Waiting for the move_base action
 server
[ INFO] [1524325339.262098120, 1578.884000000]: Connected to move base server
[ INFO] [1524325339.262158086, 1578.884000000]: Sending goal
[ INFO] [1524328842.075754184, 2858.698000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~/catkin_ws
```

my_robot

Fig 4.1


Snigdha Dongre (3 may 2018)
robond@udacity: ~/catkin_ws
File Edit Tabs Help
roscore http... /home/robo... /home/robo... robond@ud...

```
robond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal^C
robond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [illegible]: Waiting for the move_base action
 server
[ INFO] [1524585357.849994064, 1581.143000000]: Connected to move base server
[ INFO] [1524585357.950895609, 1581.143000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [illegible]: Waiting for the move_base action
 server
[ INFO] [1524586600.210787144, 1578.072000000]: Connected to move base server
[ INFO] [1524586600.211876100, 1578.072000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1524587320.070638183, 1568.590000000]: Waiting for the move_base action
 server
[ INFO] [1524587320.276043982, 1568.627000000]: Connected to move base server
[ INFO] [1524587320.276924736, 1568.627000000]: Sending goal
^Crobond@udacity:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [illegible]: Waiting for the move_base action
 server
[ INFO] [1524588156.940056200, 1569.184000000]: Connected to move base server
[ INFO] [1524588156.940953405, 1569.184000000]: Sending goal
[ INFO] [illegible]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~/catkin_ws$
```

Udacity_bot

Fig 4.2

5 DISCUSSION

The localization performed on both robots successfully helped the robot to localize and navigate to the desired goal position but then the results have shown that even though the chosen parameters have effectively worked to achieve the desired results, the parameters can further be optimized to get smooth navigation and better accuracy. If we compare both robots my_robot works better than udacity_bot. Udacity_bot still divert a little from its path while navigating which can be fixed by tuning the parameters further. While my_robot closely follow the given path to reach the goal position and navigates more smoothly than udacity_bot.

AMCL alone is not sufficient to solve the kidnapped robot problem. Although if the robot can somehow detect the abnormal activity like kidnapping and then restart the process of localization from scratch for the new environment by taking in the new data from the sensors.

MCL/AMCL can be used for pick and drop task in the industry where the map from the initial to the final position will be fixed and path can be pre-defined.

6 CONCLUSION / FUTURE WORK

Both robots successfully localized itself in the given map using Adaptive Monte Carlo Localization and were able to navigate to the pre- defined goal position following the path closely in Gazebo and Rviz simulated environment. The accuracy of localization and navigation of the robots can further be improved by tuning the parameters and adding more sensors (sensor fusion).

When the robot pose is highly uncertain, the increase in particles can increase the accuracy at the cost of longer processing time and when the robots position is somewhat well determined, the number of particles can be decreased to decrease processing time at the cost of accuracy. This enables the robot to make trade-offs in accuracy and processing time.

In future, this localization technique can be used in multi-robot scenarios, where robots will be able to synchronize with one another and navigate to their respective required position without colliding to perform their respective tasks.

REFERENCES

[1] ROS wiki, “Navigation tuning guide” http://wiki.ros.org

[2] Wikipedia, “Kalman filter”, https://en.wikipedia.org/wiki/Kalman_filter

[3] Wikipedia, “Extended Kalman filter”, https://en.wikipedia.org/wiki/Extended_Kalman_filter

[4] Wikipedia, “Particle filter”, https://en.wikipedia.org/wiki/Particle_filter

[5] Wikipedia, “Robot navigation”, https://en.wikipedia.org/wiki/robot_navigation

[6] Wikipedia, “Monte Carlo Localization”, https://en.wikipedia.org/wiki/Monte_carlo_localization

[7] ROS wiki, “navigation/ Tutorials/ RobotSetup”, http://wiki.ros.org/navigation/Tutorials/RobotSetup#Costmap_Configuration_.28local_costmap.29_.26_.28global_costmap.29

[8] Luc Alexandre Bettaieb, “A Deep Learning Approach To Coarse Robot Localization”, https://etd.ohiolink.edu/!etd.send_file?accession=case1493646936728041&disposition=inline

[9] Github, "amcl odom_aplha parameters have wrong description in wiki.ros.org”, https://github.com/ros-planning/navigation/issues/542

[10] Kaiyu Zheng, “ROS Navigation Tuning Guide”, http://kaiyuzheng.me/documents/papers/ros_navguide.pdf

[11] AMRobot5, “Mobile Robot Localization”, http://www.cs.cmu.edu/~rasc/Download/AMRobots5.pdf

[12] Robotics knowledgebase, “Adaptive Monte Carlo Localization”, http://roboticsknowledgebase.com/wiki/state-estimation/adaptive-monte-carlo-localization/