Where Am I

Shreyas Chandra Sekhar

Abstract—Two different simulated robot models were build using ROS. They were integrated with AMCL and Navigation packages to sense and navigate through a custom created map. Both the robots used Camera, Laser Range finder sensors and AMCL packages for Localization. The simulation was studied and analyzed using Gazebo and Rviz.
The Robots considered were of different physical configurations and sizes. The latter was 10 percent bigger and heavier than the former. The location of the sensors was modified accordingly.
Both the models were instructed to navigate through the map between the same starting, ending point and their performance were studied. Due to signification differences in their configurations, AMCL and Navigation Stack parameters had to tuned with different values.
After conducting multiple trials and tuning both the robot models reached the desired end location and orientation however the path and time taken were quite different.
The smaller robot took little more than 7 minutes however the heavier model took close to 10.5 minutes.

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

1 Introduction

AUTONOMOUS robots have wide range of indoor and outdoor applications. The first step for an autonomous system to work, is to identify its current pose (location and orientation) in the given environment and never get lost. This field of cartography is known as Localization.

Localization has a deep history of beyond 40 years of research which has been completely described in Durrant-Whyte and Bailey in two surveys [7, 69]. There were many research conducted between 1986 - 2004 on probabilistic formulation which includes approaches like Extended Kalman Filters, Maximum likelihood estimation, particle filters. The following period between 2004 - 2015 was a period of algorithmic analysis on localization algorithms. This included analysis on observability, convergence and consistency of Localization algorithms.

Autonomous systems used different types of sensors depending upon their applications and usage to accomplish Localization.

As mentioned, there are enormous applications of Localization, which are not limited to the few mentioned below.

An automotive navigation system used in a self-driving car gets its current location and position from satellites.

Robotics industry is growing fast in indoor auto nous robots. Robotics vacuum cleaners and floor washing robots are used to clean the floor and cloths.

Outdoor robots are used as lawn mowers in huge uneven lawn. They are further used as a security system for patrolling a highly secured environment.

An assistive domestic system focuses on making elderly and disabled close to independent. These robots help them move around independently.

The present paper covers the research and tuning of AMCL and Navigation Stack ROS packages on a simulated environment. There were two different robots considered to navigate through a given map and their tuning parameters have been compared to make them successful navigate from the same starting and ending point in the map.

2 Background

The robots used for this paper had camera and Laser Rangefinder sensors on board. It uses sensor fusion and uses the information to the localization algorithm. Initially, two localization algorithms were considered, namely, Monte Carlo Localization (MCL) and Extended Kalman Filter(EKF). However, considering the merits for the given use case MCL was used for this experimentation.

MCL has many merits over EKF. Some of the key consideration for choosing MCL was due it is ease of implementation and a lower memory foot print as compared to EKF. MCL memory consumption can easily be controlled with the number of particle filters being exposed. MCL is also capable of measuring noise with any kind of motion distribution as compared to just a Gaussian with EKF.

Choosing the right Localization algorithm was very critical for this project as the Robot would always needs to know its pose in the map even during hijacked situations. This experiment was conducted on Virtual machine with 8GB Ram and 4 CPU cores, which makes it important to consider memory foot print of the localization algorithm. EKF satisfied all the criteria mentioned above. [1]

2.1 Kalman Filters

Kalman filter are usually used in controls which needs to deal with real time information. Kalman filter are very effective in converging close to accurate value with lot of noise in the data. It further converges to very accurate estimates very fast with a little raw measurement data. Kalman filters does this by considering the uncertainty in the sensor data which can vary with the sensor hardware and the environment.

Kalman filter is a two-step iterative process, which are of ”Measurement Update” and ”State Prediction”. The sensors update the state of the robot in the ”Measurement Update” step and these information with the current state are used to predict the new state in the ”State Prediction” step.

Initially, an estimated guess of the state is used and iterates through the two steps to converge and predict the current state.

Kalman filter expects the output to be linear to the input, which may not be the case in many real time applications. For example, the actual pose of a robot in a noise environment can be anywhere around the predicted estate following a Gaussian distribution. The Gaussian distribution would be much narrow for an environment with lower terrain disturbances but much wider otherwise. On top of this the sensors would have internal noise which adds to this uncertainty. This is overcome by using multiple sensors onboard to provide inputs to localization algorithm. This is called "sensor fusion".

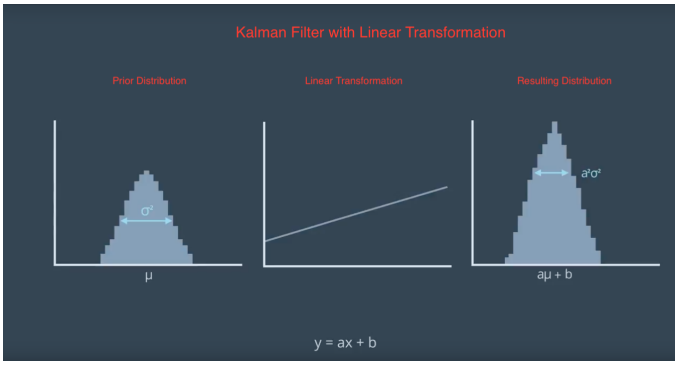

Fig. 1. Kalman Filter Model.

Kalman filters was used for trajectory estimation of the Apollo program by NASA. It has a wide range of applications in controls engineering with high amount of noise. It can be used to measure the fluid level of a tank, position of a mobile robot. It can be further used to estimate share market performance or GDP of a country.

Kalman filters comes with three different variations, namely Kalman filter (KF) for linear systems where output is linear to input, Extended Kalman filters (EKF) are applied to non-linear systems and Unscented Kalman filter (UKF) are used for highly non-linear systems where EKF doesn't converge.

2.2 KF vs EKF

KF works well only under linear Motion, measurement models and when the State prediction is a unimodal Gaussian distribution.

However, the robots following a curvy path or circle, leads to non-linear motion and non-unimodal Gaussian distribution. This defeats the prerequisite of KF and it can no longer be applied for this model.

To overcome this limitation, KF model is tweaked a little bit to form EFK. With reference to the below diagram, a closer look of the non-linear function approximates to linear function in a small region. EKF uses this small region of non-linearity for state prediction.

Repeated samples of the robot's pose at every step as it moves and narrowing down the transformation function in the linear region, provides sufficient information for a close enough pose prediction.

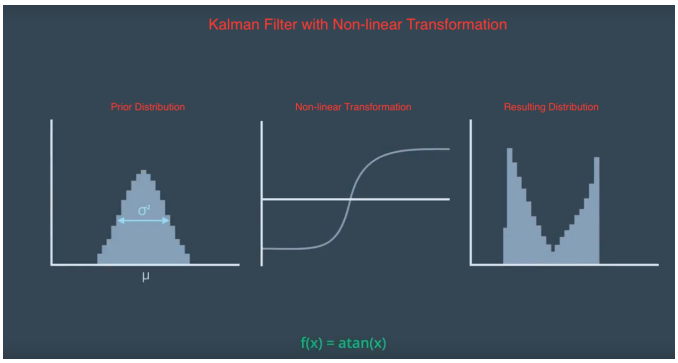

Fig. 2. Extended Kalman Filter Model.

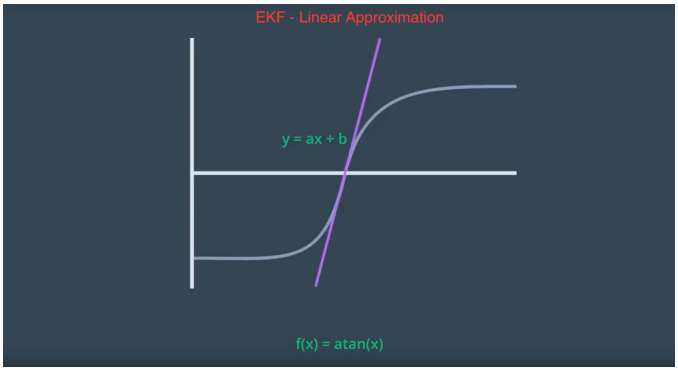

Fig. 3. Extended Kalman Filter Approximation of Non-Linearity

The mean of the prediction can use the non-linear function; however, the covariance requires just a linear model for state prediction.

To accomplish this, first two terms of Taylor series are used for linear approximation to form Unimodal Gaussian distribution. This would induce some noise however, provides close approximation for desired results. This makes the calculation simple and efficient.

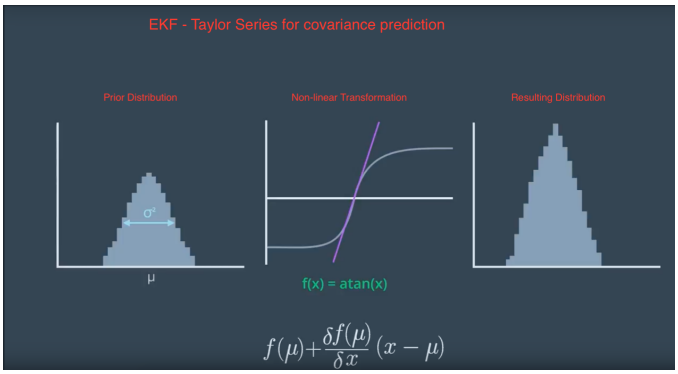

Fig. 4. Extended Kalman Filter Approximation Taylor Series Approximation

2.3 Particle Filters

Monte Carlo Localization (MCL) is also known as particle filters as it uses particles for localization. Each particle is a virtual element which resembles the robot. Each particle has its pose in the given map. These particles are depicted

from the on board sensors about the likelihood of its current location.

However, the limited to local and global localization, it would lose its track on hijack situation.

MCL estimates the distribution of a robots pose based on information from the sensors, which is termed as recursive Bayes filter.

In a MCL, the robots start with attempting to identify its pose in the given map by solving global identification. It uses its on board sensors and estimates its local coordinates and orientation (local coordinate frame) with respect to global coordinate frame.

The name ”Particle filter” comes from the virtual particles spread across the map representing the robot. Each particle has its own coordinate and orientation with respect to the global coordinate frame. Each particle also has its weights which represents the difference between robot actual pose and the predicted pose by the particle. Particle with higher weights survives during the re sampling process, rest of the fade away. After multiple re sampling, the particle with highest weights survives and converge to represent the robots correct pose.

2.4 Comparison / Contrast

MCL has many advantages over EKF. Unlike the need of Gaussian State space distribution in EKF, MCL works with any kind of distribution. This enable EKF to be applied for wider range of environments. Memory consumption by MCL can easily be controlled by the number of particles exposed for localization. MCL is also easier to implement and much time efficient as compared to EKF. Below is a detailed comparison between them.

MCL vs EKF

	MCL	EKF
Measurements	Raw Measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency(memory)	✓	✓✓
Efficiency(time)	✓	✓✓
Ease of Implementation	✓✓	✓
Resolution	✓	✓✓
Robustness	✓✓	x
Memory & Resolution Control	Yes	No
Global Localization	Yes	No
State Space	Multimodel Discrete	Unimodal Continuous

Fig. 5. MCL vs EKF

3 Simulations

There were two different models used for simulation. The benchmark model called ”udacitybot” was provided by Udacity. The second model called ”shreyasbot” was customized with many changes. It was ten percent bigger and heavier. Both the models had camera, Lazar sensors and used ROS navigation stack to move around in the map.

ROS navigation stack takes in information from sensors, odometry, goal pose through `tf/Message`, `nav_msg/Odometry` topics. It constantly provides velocity command to control the robot.

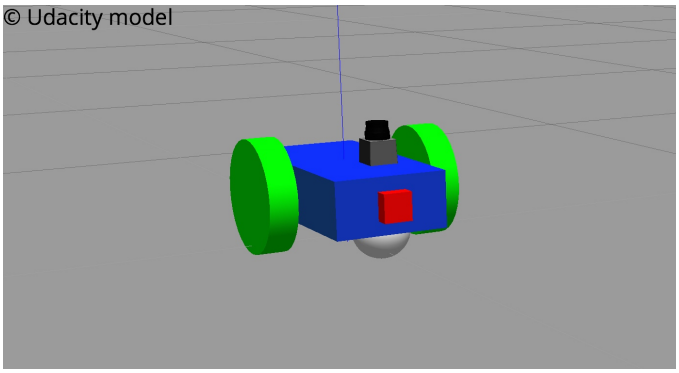

Fig. 6. Udacity Benchmark Model

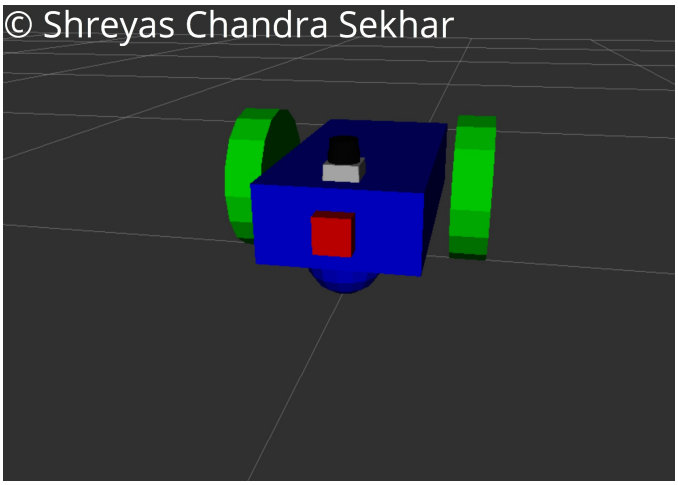

Fig. 7. Shreyas Model

ROS `move_base` package uses navigation stack to move the robots in the map from current pos to end pose without collusion. It takes information from the sensors and provides to `global_planner` and `local_planner` through `geometry_msgs/PoseStamp` topic.

`global_planner` uses navigation algorithms like A*, Dijkstra for path planning of the robot from current to goal pose. `local_planner` communicates with `global_planner` constantly and produces velocity commands to the robot for navigation. `global_planner`, `local_planner` comes as a pair with `global_costmap`, `local_costmap` which provide current location in the map and sensor information.

They further interact with `recovery_behaviors` which attempts to rescue the robot from struck condition. `clear_costmap_recover` provides navigation stack that attempts to clear the space by altering the cost map to static map out side of the given area. `rotate_recovery` package attempts to clear space by 360-degree rotation of the robot.

3.1 Achievements

Both the udacitybot and shreyasbot were made to tunned to navigate from start to goal location in the map. Due the significant variations between the robots configurations,

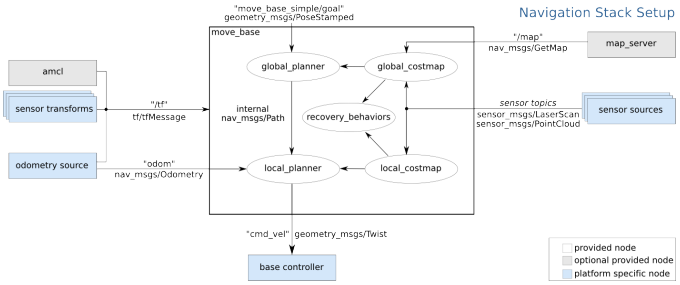

Fig. 8. Navigation stack setup

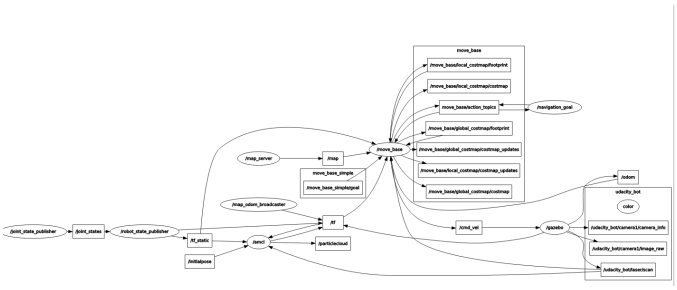

Fig. 9. udacitybot packages

AMCL and move base was tunned separately to achieve the desired results.

3.2 Benchmark Model

3.2.1 Model design

There were multiple difference between the physical configurations of udacitybot and shreyasbot. shreyasbot was ten percent bigger and heavier as compared to base model. As it was bigger, it had an higher inertia and the location of the sensors were changed correspondingly

Below Tables compares the configurations of the Robot Models

TABLE 1
udacitybot vs shreyasbot physical configuration

configuration	udacitybot	shreyasbot
chassis box size	0.4 0.2 0.1	0.44 0.22 0.11
chassis mass	15.0	16.5
chassis inertia	0.1	0.11
caster sphere radius	0.05	0.055
back caster sphere origin	-0.15 0 -0.05	-0.165 0 -0.055
front caster sphere origin	0.15 0 -0.05	0.165 0 -0.055
wheel length, radius	0.05, 0.1	0.055, 0.11
wheel mass	5.0	5.5
wheel inertia	0.1	0.11
wheel effort	10000	11000
camera joint	0.2 0 0 0	0.22 0 0
laser rangefinder joint	0.15 0 0.1	0.165 0 0.07

3.2.2 Packages Used

The robots starts from its stating location by providing its initial pose to amcl which further communicates with move_base topic through tf. The goal location is also provided to move_base through its goal topic. move_base provides velocity as output to the robot in gazebo through odom topic. The robots executes and collects in current location through the sensors and provides it amcl. This cycle follows in a loop until it reaches the goal location in the map.

3.2.3 Parameters

amcl launch file has pointers to amcl localization and move_base configuration files. move_base has four configuration files:

- base_local_planner_params.yaml:
  base_local_planner_params Provides velocity commands to the mobile base of the robot given high level plan.
- costmap_common_params.yaml: The robot gets to know about the obstacles in the map with the help of costmaps. There are three types of cost maps, namely, global, local and common.
  costmap_common_params are used to get information from the sensors and are used by both global and local costmaps.
- global_costmap_params.yaml:
  global_costmap_params is used for long term planning of the robot's navigation over the entire environment.
- local_costmap_params.yaml local_costmap_params is used for short term planning, mainly for obstacle avoidance.

There are many configuration parameters for both amcl as well as move_base which plays a critical part on the robots performance and functionalities. Here are explanations of some of parameters the play a key role in successful navigation of the robot.

- min_particle and max_particle: As mentioned earlier, MCL is also called particle filters. They use virtual particles to represent the robot with weights to find the location of the robot in the map. the range between min_particle and max_particle controls the number of particle used by MCL. A narrow range could impact the accuracy of the localization and a high number would consume lot of memory and CPU resources for commutation.
- transform_tolerance: It is the tolerable tf transformation latency time allowed between the links in seconds. It allows in accepting latency in command propagation through the link however gauges from updating the robot from outdated command instruction. The robot goes to a stand still on crossing this latency.
- controller_frequency: The rate at which the command are generated to the robot. An higher number would need higher computation power and a lower number may not update the robot with a sufficient rate for successful navigation through the map.
  As seen from the explanations controller_frequency would have to be smaller than transform_tolerance.
- laser range: laser_min_range and laser_max_range provided the range of the laser . Setting laser_max_range to -1 will enable its maximum range
- velocity range: min_vel_x, max_vel_x controls the linear velocity range of the robot. min_vel was set to

negative number to enable going backward during struck conditions.
max_vel_theta limit the angular velocity

- sim_time: Its the maximum time allowed to follow the same odometry command. Having it too high makes the robot oscillate.
- pdist_scale: It controls how close should the robot be to the planned path. Setting it low leads to struck conditions in curvy paths.
- robot_radius: It is the radius of the robot steering. Larger robots needs higher radius to avoid collusion during turns.
- inflation_radius: It is the distance from obstacle from which the obstacle cost in the costmap is set same as that of the obstacle.

3.2.4 Errors and resolution

This session describes the list of errors in the logs and the tunning performance to rectify them.

- loop missed its desired rate: [WARN] [1518612926.122964016, 1528.110000000]: Map update loop missed its desired rate of 10.0000Hz... the loop actually took 0.1830 seconds
  Resolution: Increase transform_tolerance and decrease controller_frequency
- Clearing costmap to unstuck robot: [WARN] [1518884983.613627023, 1543.530000000]: Clearing costmap to unstuck robot (3.000000m). Resolution: Increase inflation_radius of global_costmap
- Timed out waiting for transform: [WARN] [1519022368.615851302, 1522.258000000]: Timed out waiting for transform from robot_footprint to map to become available before running costmap, tf error: . canTransform returned after 1522.26 timeout was 0.1
  Resolution: amcl was not receiving laser data from scan topic. used rosrun tf view_frames for debugging

3.3 Personal Model

3.3.1 Model design

shreyasbot was ten percent bigger and heavier than udacitybot, refer benchmark model design session for more details.

3.3.2 Packages Used

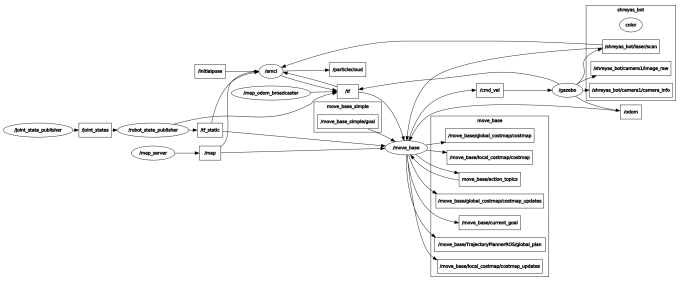

Fig. 10. shreyasbot packages

3.3.3 Parameters

Same as benchmark model

4 Results

udacitybot took close to 7 minutes to navigate from the start to goal location in the map where us shreyasbot took close to 10 minutes. Navigation of both the robots were smooth and they never went to struck condition.

The robots revolved around two spots which needs further investigations.

4.1 Localization Results

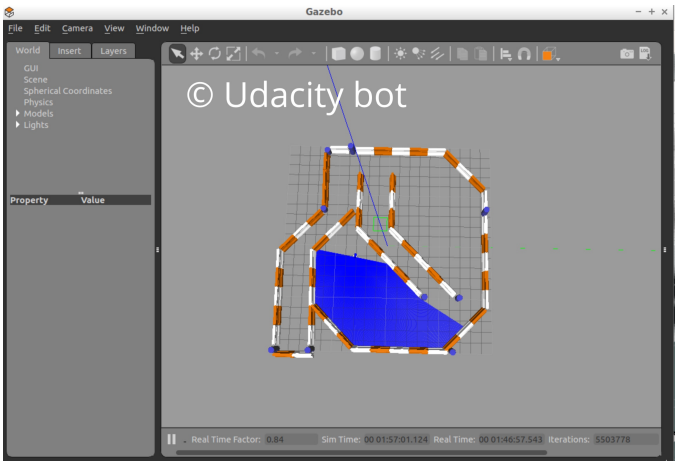

Fig. 11. udacitybot reached goal gazebo

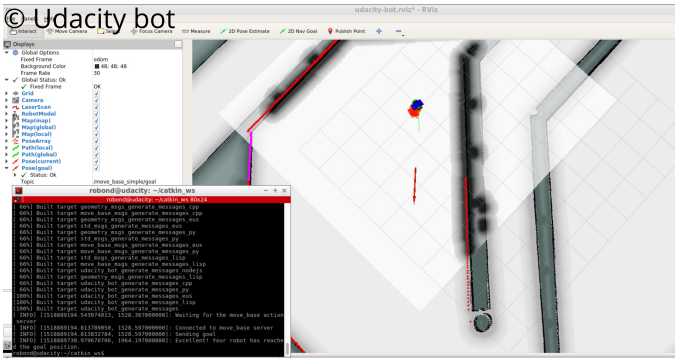

Fig. 12. udacitybot reaches goal location

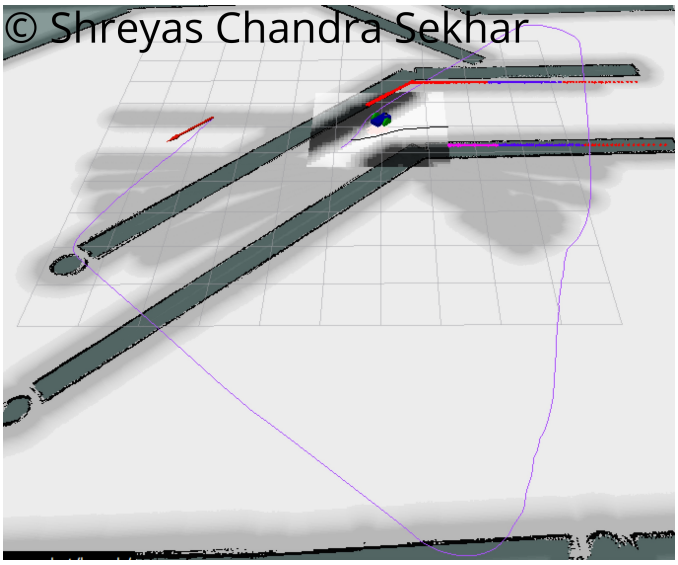

Fig. 13. shreyasbot navigation starting point

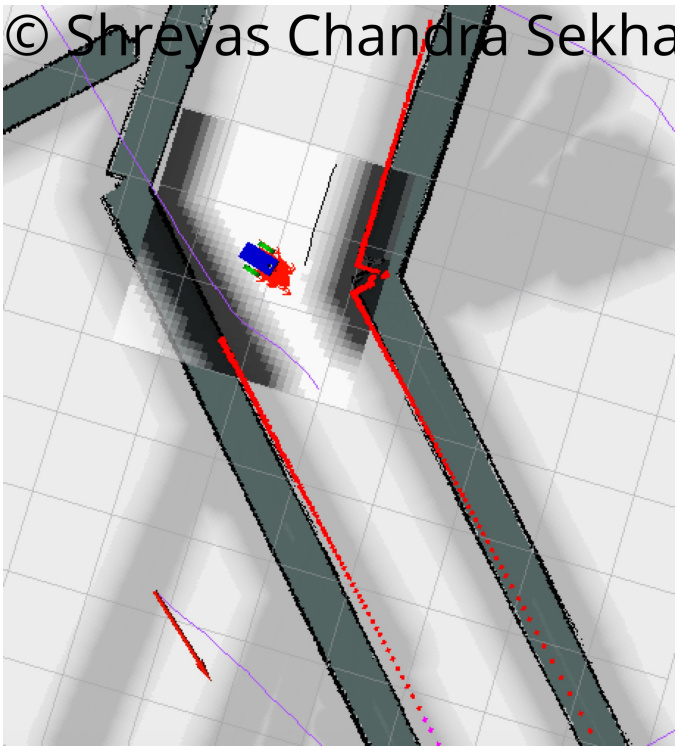

Fig. 14. shreyasbot proceeds from start point

4.2 Technical Comparison

The robots took a different path, shreyasbot took a longer path to reach the same goal. shreyasbot being bigger and heavier was slower due to its size and took longer in curves.

controller_frequency of shreyasbot was increased to increase transformation_tolerance however the limit of cpu and memory was capped as the exercise was ran on Virtual machine with 4 CPU cores, 8GB ram. The robots are sure to perform faster in TX2. The frequency and the max_particle can also be increased which would lead to better performance.

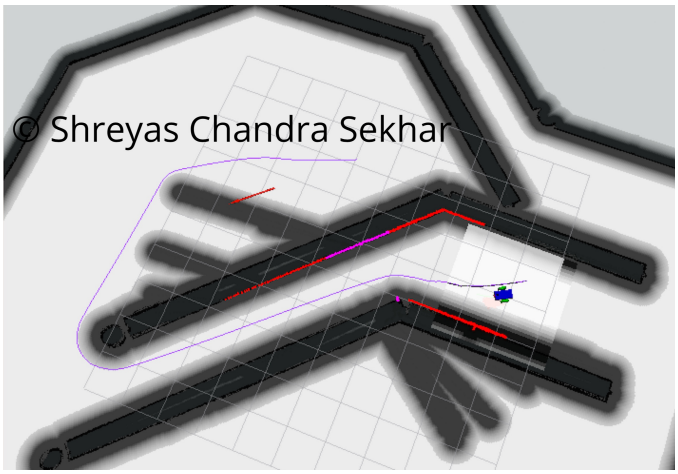

Fig. 15. shreyasbot navigates towards south

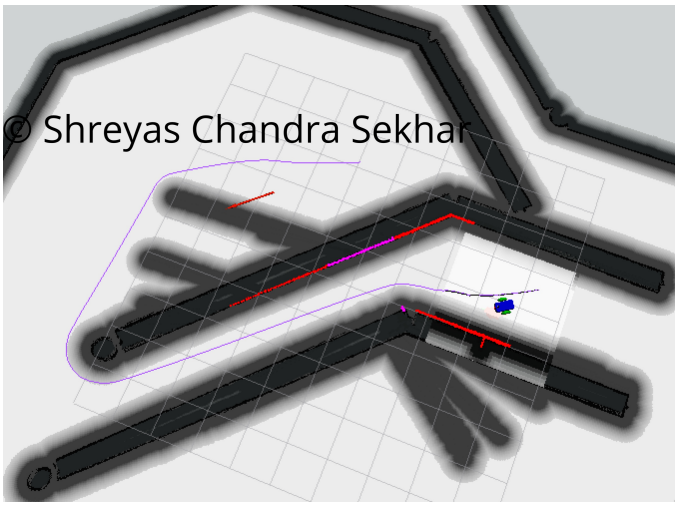

Fig. 16. shreyasbot turns back

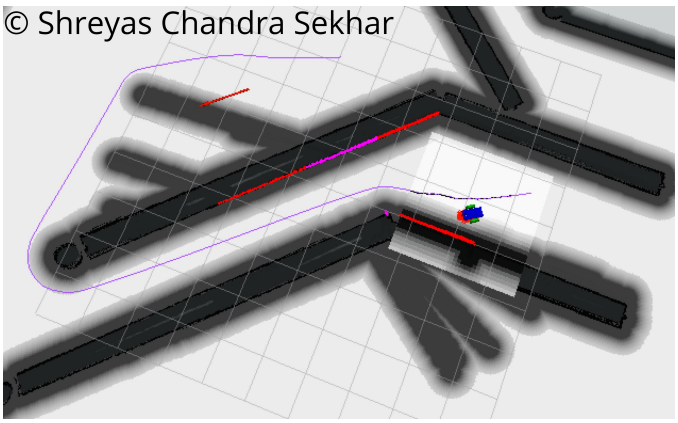

Fig. 17. shreyasbot proceeds further

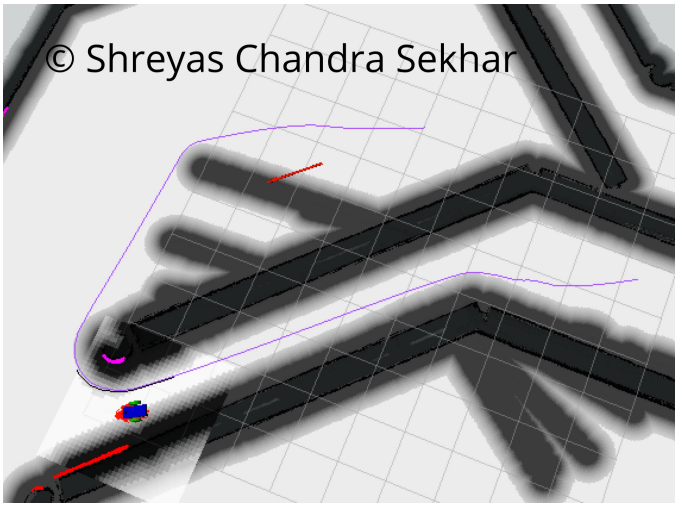

Fig. 18. shreyasbot close to turn

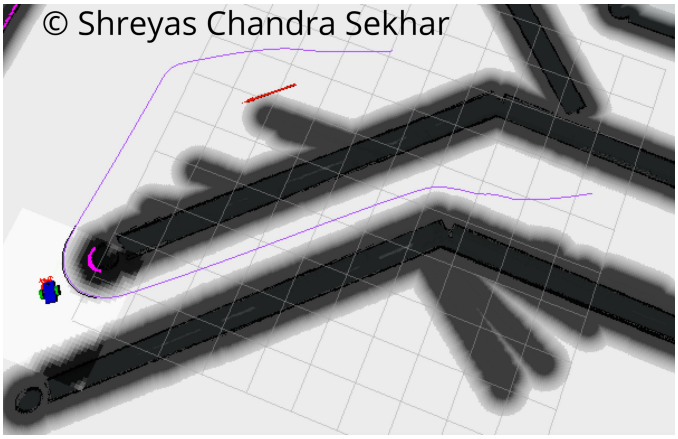

Fig. 19. shreyasbot turning

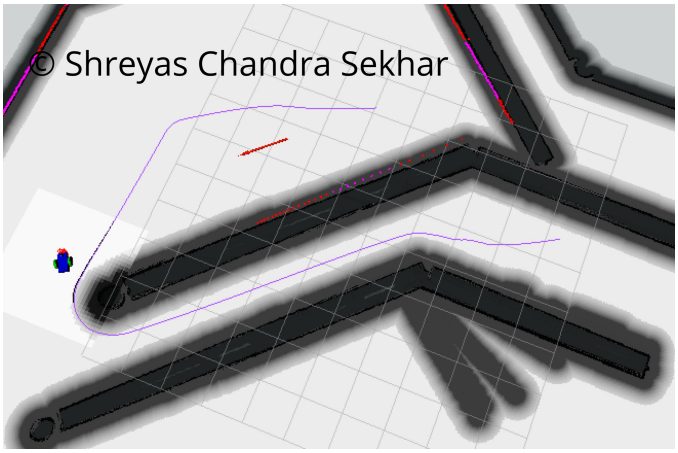

Fig. 20. shreyasbot turning right

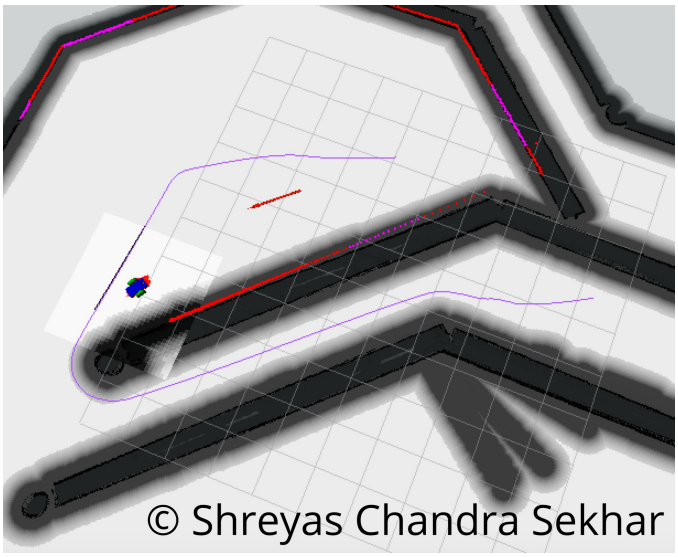

Fig. 21. shreyasbot proceeds further after turning

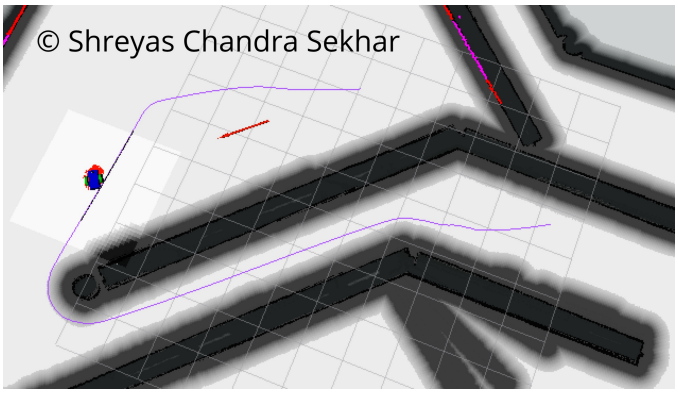

Fig. 22. shreyasbot gets close to global path

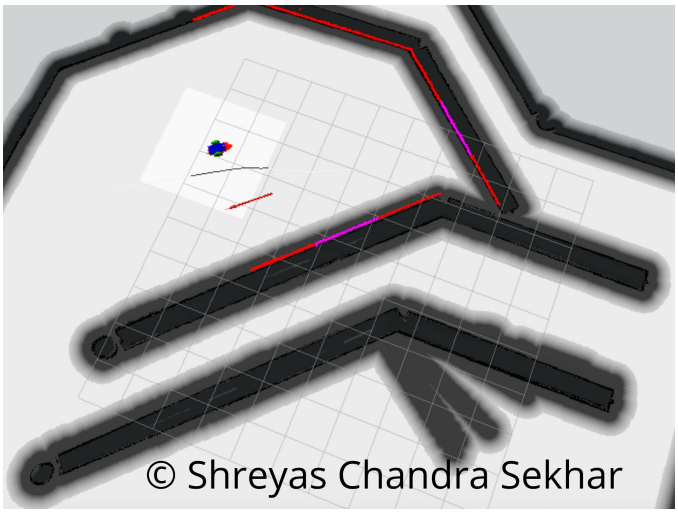

Fig. 23. shreyasbot back on track

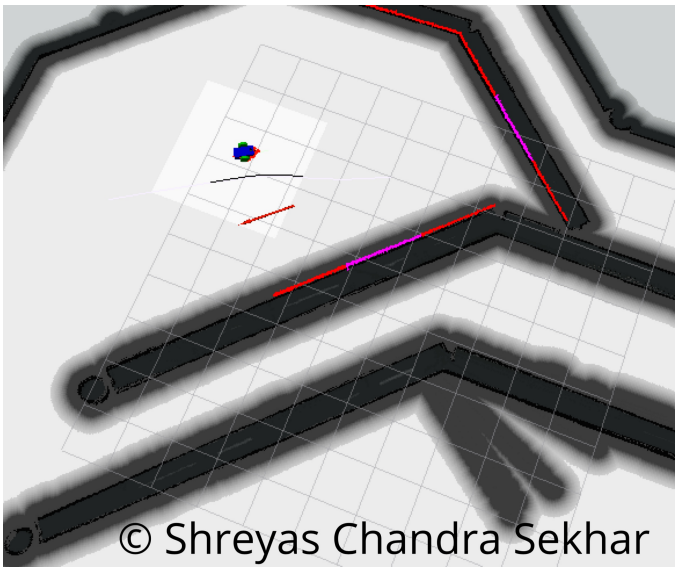

Fig. 24. shreyasbot close to goal

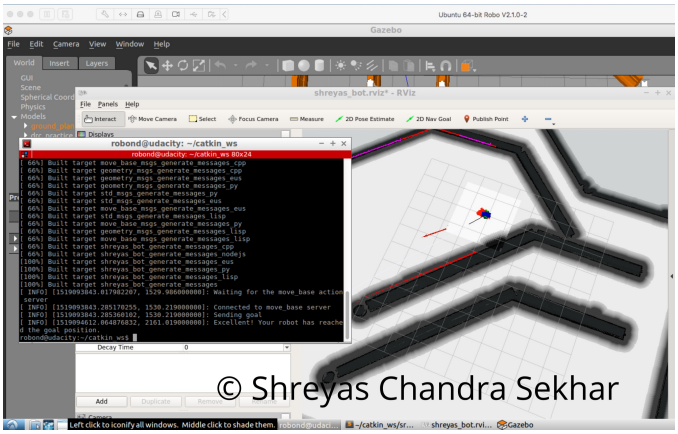

Fig. 25. shreyasbot reached goal rviz

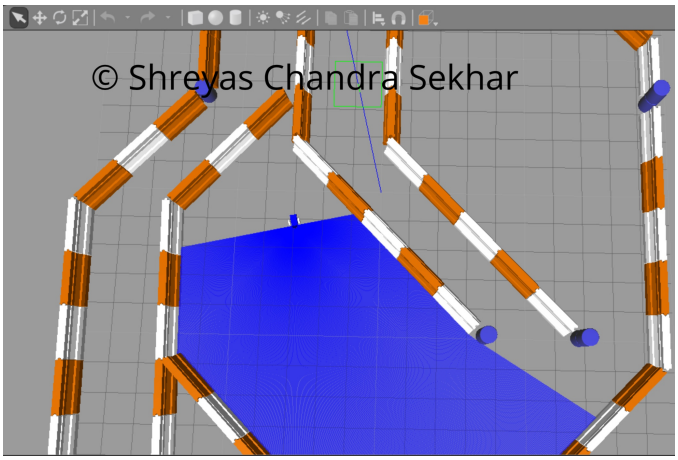

Fig. 26. shreyasbot reached goal gazebo

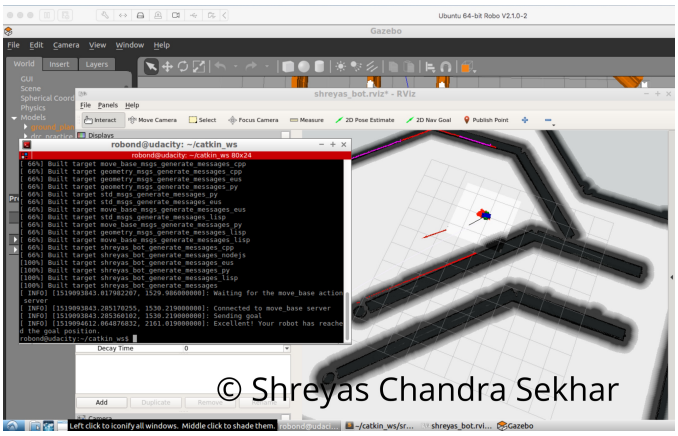

Fig. 27. shreyasbot reaches goal location

5 Discussion

shreyasbot had a bigger robot_radius to avoid collusion. It was almost the same speed in the straight path but was slower and took a longer turn. Increasing its obstacle_range will provide wider room for the bigger robot to turn. This will also lead to longer turning radius.

Considering the size of the robots both reports performed almost the same. The shreyasbot took a longer path and had to cover longer distance. MCL has many real time applications. It can be used for floor cleaning, farming and many more. However, MCL does not handled 'Kidnapped Robot' problem. EKF would be better option.

6 Conclusion / Future work

Two robots with different sizes were used to navigate between the same start and goal location of a map. The second robot was ten percent bigger and heavier. They used laser and camera sensors to get odometry information from the map to MCL. The benchmark model took close to 7 minutes and the second robot took close to 10 minutes.

As a future enhancement, EKF will be tried to compare performance. The experiment would also be run on TX2 for better results.

6.1 Hardware Deployment

This paper covered results of simulated environment. The approach and results can be completely different with real hardware.

The starting point to hardware implementation would be to setup an open loop controller of a differential robot. This can be implemented with two machines connected through a WIFI. Host machine can do all the heavy commutation work and the robot can be loaded with Raspberry Pi to execute the velocity commands. GoPiGo Differential robots and Raspberry Pi USB WiFi Dongle are suggested hardware to start with.

References

[1] W. B. Sebastian Thrun, Dieter Fox and F. Dellaert, *LATEX: Robust Monte Carlo Localization for Mobile Robots*. Elsevier Preprint, 2001.