localization using AMCL and Navigation ROS Packages

Shweta Ruparel

Abstract—Mobile robot localization is a challenge of determining a robot's pose from sensor data in a mapped environment.Often the robots are required to be able to navigate and get the information about its surrounding. Autonomous navigation or along the way-points the robot must be able to perceive its surrounding and make localization decision.
Probabilistic Robotics is a new approach to robotics that pays tribute to the uncertainty in robot perception and action. Probabilistic approaches are typically more robust in handling sensor limitations, sensor noise, environment dynamics, and so on. This paper describes in detail about probabilistic algorithms like **Extended Kalman Filter and Adaptive Monte Carlo localization** that can filter noisy sensor measurements and track the robot's position and orientation.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, localization ,Monte Carlo,Particle filters.

1 Introduction

Mobile robot localization is the problem of determining the pose of a robot relative to a given map of the environment.

Let's consider a home service robot that can navigate inside a home and keep a safety check in the house by navigating across different rooms and localizing areas of safety check.

The amount of information present and the nature of environment that a robot is operating in determine the difficulty of the localization task.Three most common localization problems are mentioned below-

- Local Localization,also known as position tracking. In this problem robot knows its initial pose and the robot estimates its pose as it moves around the environment.Here , uncertainty in the robot motion is limited to the regions surrounding the robot.
- Global localization, in this case the robots initial pose is unknown and the robot must determine its pose relative to the ground truth map.The amount of uncertainty in global localization is much greater than local

localization
- Kidnapped Robot problem, is the most challenging localization problem. This is just like global localization except that robot may be kidnapped at anytime and moved to a new location on the map.

So far we have characterised the three most important localization problems.In the real world we still have to consider whether environment is static, meaning unchanging or dynamic where objects may shift over time.Dynamic environments are more challenging to localize in. In this paper we will focus only on static environments where the environment always matches the ground truth map.

In this paper, we will discuss two most common localization algorithms.

- Extended Kalman Filter - A very robust algorithm for filtering noisy sensor data and can localise multidimensional, non linear motion.
- Adaptive Monte Carlo localization - **amcl** is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map.

2 Background / Formulation

Localization is one of the most fundamental competencies required by an autonomous robot. In a typical robot localization use-case, a map of the environment is available and the sensors equipped robot observes the environment as well as monitor its own motion. The localization techniques are used to estimate the robot position and orientation within the map using information gathered from these sensors.

Each of these sensors has limitations that manifest as noise and error.Three most common types of mobile robot sensors are IMU(inertial Measurement units), rotary encoders, range finder sensors(LIDAR) and vision senors.

Robot localization techniques need to be able to deal with noisy observations and generate not only an estimate of the robot location but also a measure of the uncertainty of the location estimate. Two of the most common probabilistic techniques, **the extended Kalman filter and the particle filter(AMCL)**, that can be used to combine information from sensors to compute an estimate of the robot location are discussed and compared in this paper.

2.1 Kalman Filter

The Kalman filter was invented in the 1950s by Rudolph Emil Kalman, as a

technique for filtering and prediction in linear systems. The Kalman filter implements belief computation for continuous states. It is not applicable to discrete or hybrid state spaces.

2.1.1 Linear Guassian Systems

At the basis of the Kalman Filter is the Gaussian distribution, sometimes referred to as a bell curve or normal distribution. The role of a Kalman Filter is that after a movement or a measurement update, it outputs a unimodal Gaussian distribution. This is its best guess at the true value of a parameter.

A Gaussian is characterized by two parameters - its mean and its variance. The mean is the most probable occurrence and lies at the centre of the function, and the variance relates to the width of the curve. The term unimodal implies a single peak present in the distribution.

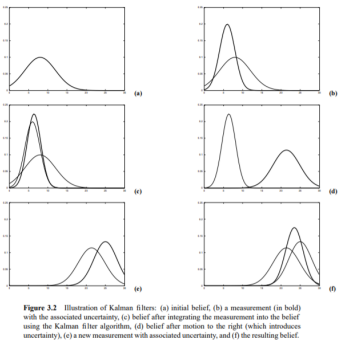

Figure 3.2 Illustration of Kalman filters: (a) initial belief, (b) a measurement (in bold) with the associated uncertainty, (c) belief after integrating the measurement into the belief using the Kalman filter algorithm, (d) belief after motion to the right (which introduces uncertainty), (e) a new measurement with associated uncertainty, and (f) the resulting belief.

Figure 1. Kalman Filter illustration

2.1.2 Illustration

Kalman Filter is an iterative algorithm that starts with an initial estimate then calculates measurement update which is the sum of prior belief and measurements from sensor and then state prediction which is the addition of prior belief's mean and variance to the motion's mean and variance.

Figure 1, illustrates the Kalman filter algorithm for a simplistic one-dimensional localization scenario. Suppose the robot moves along the horizontal axis in each diagram,in Figure 1. Let the prior belief over the robot location be given by the normal distribution shown in Figure 1a.After gathering the data from sensors, the measurement update returns a measurement that is centered at the peak of the bold Gaussian in Figure 1b. Its peak is the value predicted by the sensors, and its width (variance) corresponds to the uncertainty in the measurement. Combining the prior with the measurement, yields the bold Gaussian in Figure 1c. This belief's mean lies between the two original means, and its uncertainty radius is smaller than both contributing Gaussians. Next, assume the robot moves towards the right. Its uncertainty grows due to the fact that the next state transition is stochastic. This Gaussian is shifted by the amount the robot moved, and it is also wider.Robot receives a second measurement illustrated by the bold Gaussian in Figure 1e,which leads to the posterior shown

in bold in Figure 1f.As this example illustrates, the Kalman filter alternates a measurement update step, in which sensor data is integrated into the present belief, with a prediction step (or control update step), which modifies the belief in accordance to an action.

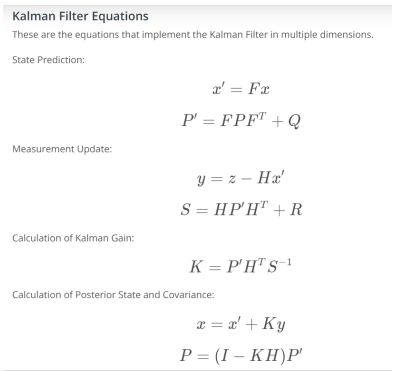

Figure 4. Kalman Filter Equations

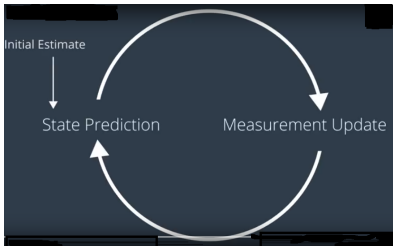

Figure 2. Kalman Filter

2.1.3 The Extended Kalman Filter

The Kalman Filter is applicable to problems with linear motion and measurement functions as shown in the *figure 5*. This is limiting, as much of the real world is nonlinear.

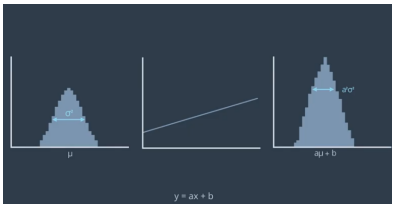

Figure 5. Kalman Filter

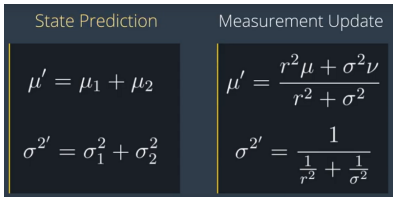

Figure 3. Kalman Filter Algorithm

A nonlinear function can be used to update the mean of a function,but not the variance, as this would result in a non-Gaussian distribution , refer *figure 6.* ,which is much more computationally expensive to work with. To update the variance, the **Extended Kalman Filter** linearizes the nonlinear function f(x) over a small section and calls it F.The mean can continue to be

updated with the non-linear function f(x) as in *figure 7*, but the co-variance must be updated by the linearization of the function f(x).*Figure 8,* shows the change in calculation of mean and co-variance in extended kalman filter.

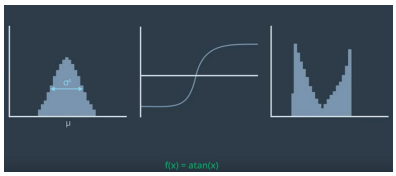

Figure 6. Kalman Filter with a non-linear function

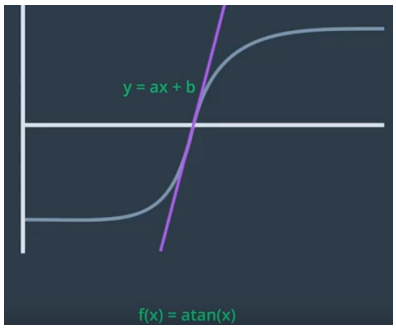

Figure 7. non-linear function f(x)

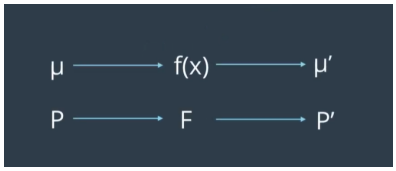

Figure 8. linearization over non-linear function

The linear approximation can be obtained by using the first two terms of the Taylor Series of the function centered around the mean.

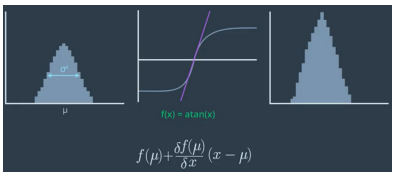

Figure 9. Extended Kalman filter using First two terms of Taylor Series

2.1.4 Extended Kalman Filter Equations

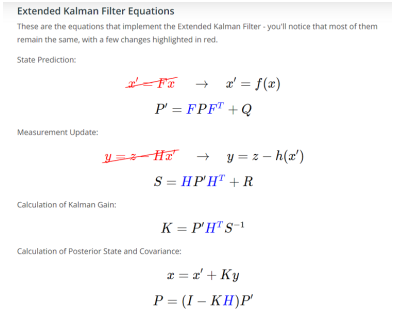

Figure 10. Extended Kalman filter equation using Jacobians

Highlighted in blue are the Jacobians that replaced the measurement and state transition functions.

The Extended Kalman Filter requires us to calculate the Jacobian of a nonlinear function as part of every single iteration, since the mean (which is the point that we linearize about) is updated.

2.1.5 Summary

- The Kalman Filter cannot be used when the measurement and/or state transition functions are nonlinear, since this

would result in a non-Gaussian distribution.

- Instead, we take a local linear approximation and use this approximation to update the covariance of the estimate. The linear approximation is made using the first terms of the Taylor Series, which includes the first derivative of the function.
- In the multi-dimensional case, taking the first derivative isn't as easy as there are multiple state variables and multiple dimensions. Here we employ a Jacobian, which is a matrix of partial derivatives, containing the partial derivative of each dimension with respect to each state variable

2.2 Adaptive Monte Carlo localization

The powerful Monte Carlo localization algorithm estimates the posterior distribution of a robot's position and orientation based on sensory information. This process is known as a recursive Bayes filter.

The goal of Bayes filtering is to estimate a probability density over the state space conditioned on the measurements. The probability density, or also known as posterior is called the belief.

Monte-Carlo localization uses a Bayes filter; it has a prediction step and a measurement update step. The knowledge that the robot holds about its environment (also known as belief) is represented by a set of particles, where a particle represents a 'guess' location. If the starting position of the robot is unknown, these particles are spread across the map used for localization, with each particle representing a potential position and orientation for the robot. Alternatively, a known starting location can be given as an input, around which the particles spread.

Lets consider a 2D mapped environment and a robot has no idea where it is located. Since its initial stage is unknown, the robot will estimate its pose by solving the global localisation problem.Lets consider that the current robot pose is determined by x coordinate, y coordinate and an orientation theta with respect to global coordinate frame.

With The MCL algorithm particles are initially spread randomly and uniformly all over the map.Just like robot each particle has an x coordinate , y coordinate and an orientation theta . Each of these particles represents a hypothesis of where the robot might be. In addition to a 3D vector , particles are each assigned a weight. The weight of a particle is the difference between the robot's actual pose and the particle's actual pose.The importance of the particle depends on its weight and bigger the particle more accurate it is.The partcles with more weight are more likely to survive during a resampling

process. After the resampling process , the particles with more weight are more likely to survive and the particles with less weight die.Finally after several iterations of MCL algorithm and after different stages of resampling particles will converge and estimate the robots pose.

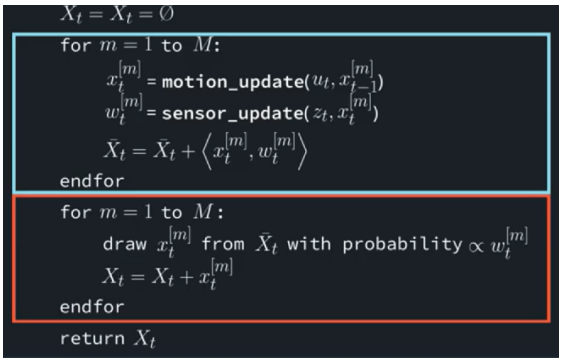

Figure 11. MCL Algorithm

The MCL Algorithm mainly consist of two for loops , first for loop for motion and measurement update and second for loop for resampling process. At each iteration the algorithm takes the previous belief , actuation command and sensor measurement as input. Initially the belief is obtained by randomally generating M Particles, then in the first for loop hypothetical state is computed whenever the robot moves, following the particle's weight is computed using the sensors measurement.Motion and measurement are then added to the prior state.Moving on to the next for loop where resampling process happens. Here the particles with high probability survive and reconsidered in the next iteration while the other die.Finally algorithm outputs the new belief.Refer *figure 11* for MCL ALgorithm

MCL solves the global localization and kidnapped robot problem in a highly robust and efficient way. It can accommodate arbitrary noise distributions (and non-linearities). Thus, MCL avoids a need to extract features from the sensor data.

2.2.1 MCL vs EKF

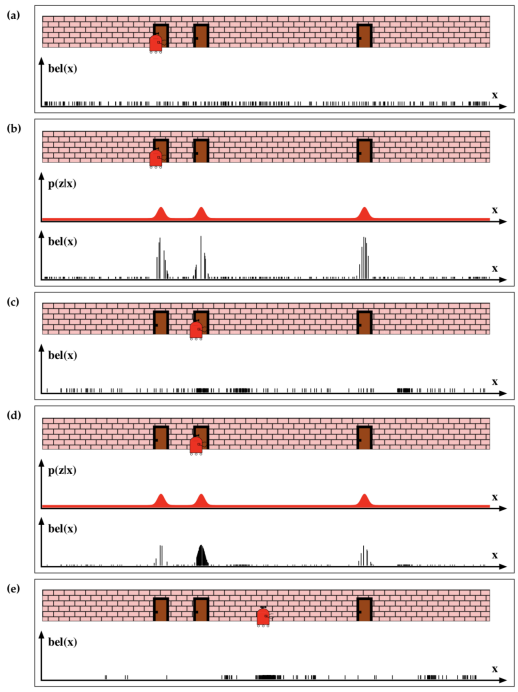

Figure 12. MCL illustration

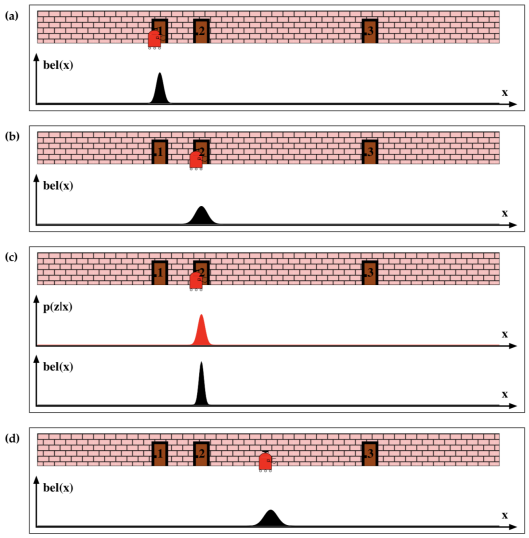

Figure 13. EKF illutration

At time:

- t=1, Particles are drawn randomly and uniformly over the entire pose space.
- t=2, Measurement is updated and an importance weight is assigned to each particle.
- t=3, Motion is updated and a new particle set with uniform weights and high number of particles around the three most likely places is obtained in resampling.
- t=4, Measurement assigns non-uniform weight to the particle set.
- t=5, Motion is updated and a new resampling step is about to start.

At time:

- t=1, Initial belief represented by a Gaussian distribution around the first door.
- t=2, Motion is updated and the new belief is represented by a shifted Gaussian of increased weight.
- t=3, Measurement is updated and the robot is more certain of its location. The new posterior is represented by a Gaussian with a small variance.

- t=4, Motion is updated and the uncertainty increases.

MCL has many advantages over Kalman Filter , refer *figure* 14

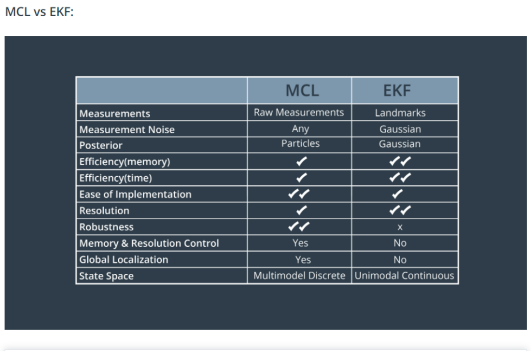

Figure 14. MCL vs EKF

map. This adaptive process offers a significant computational advantage over MCL.

The **ROS amcl package** implements this variant and it is used with the robot to localize it inside the provided map.

The **move_base** package is a very powerful tool. It utilizes a costmap - where each part of the map is divided into which area is occupied, like walls or obstacles, and which area is unoccupied. As the robot moves around, a local costmap, in relation to the global costmap, keeps getting updated allowing the package to define a continuous path for the robot to move along.

3 RESULTS

3.1 Test Environment

Udacity workspace was used to test the robot in simulation.This online tool includes GPU access when needed as well as ROS, Gazebo, and other packages.While Gazebo is a physics simulator, RViz can visualize any type of sensor data being published over a ROS topic like camera images, point clouds, Lidar data, etc. This data can be a live stream coming directly from the sensor or pre-recorded data stored as a bag file.

Adaptive Monte Carlo Localization (AMCL) dynamically adjusts the number of particles over a period of time, as the robot navigates around in a

3.1.1 Udacity Bot test results

Udacity supplied robot was created and tested in Gazebo and rviz was used to visualise the sensor readings , path mapping and amcl algorithm in action.

Following figures shows the different scenarios tested for udacity_bot.

- **udacity_bot tested for the navigation goal set by the supplied file**.

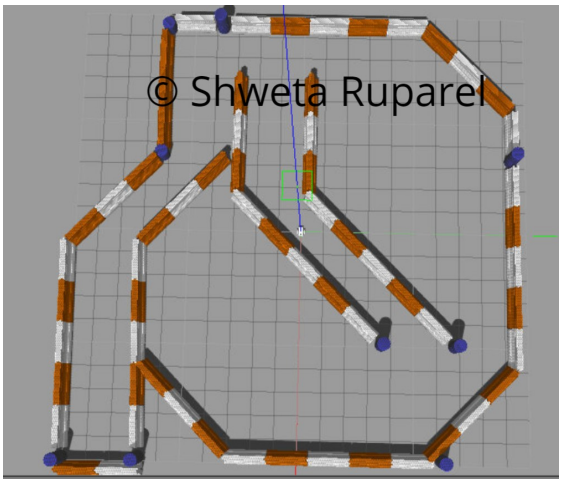

Figure 15. Udacity_bot initialised in Gazebo

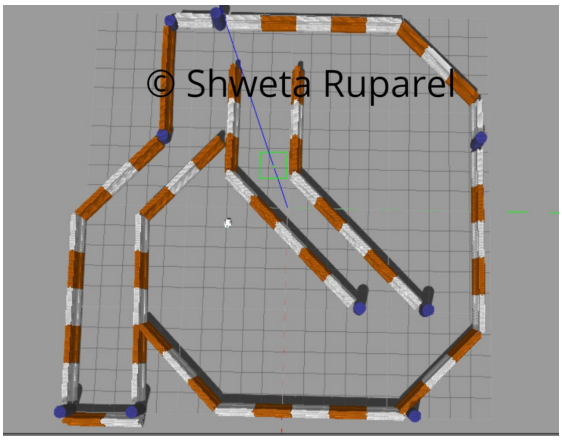

Figure 17. Udacity_bot reached the goal set by the navigation_goal file in Gazebo

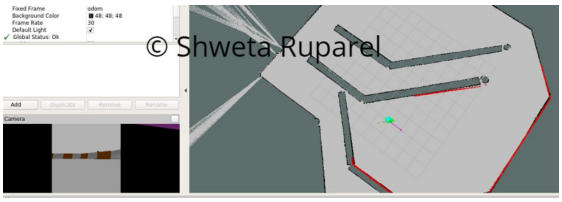

Figure 18. Udacity_bot reached the goal set by the navigation_goal file in rviz

- **udacity_bot tested for the 2d nav goal set using rviz tool.**

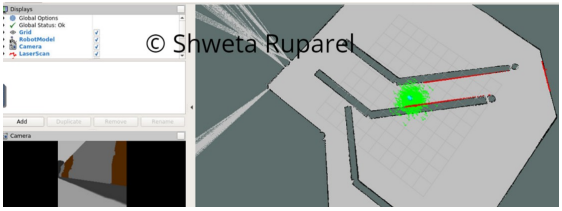

Figure 16. Udacity_bot initialised in Rviz

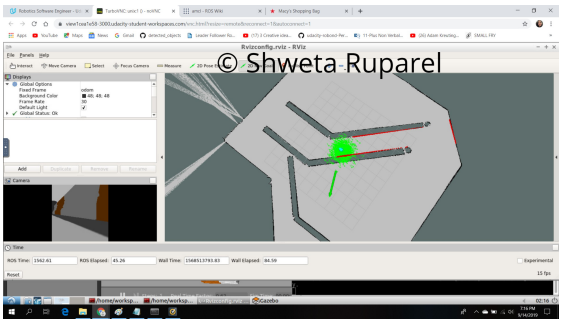

Figure 19. Udacity_bot initialised in Rviz. Shows the navigation Goal by a green arrow

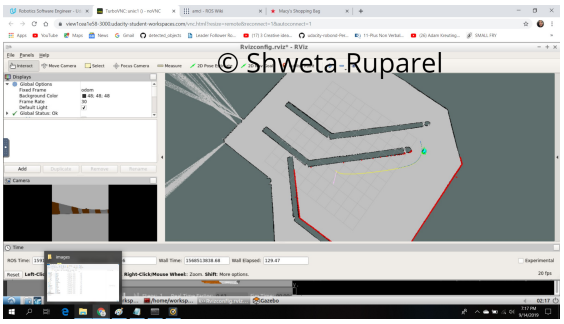

Figure 20. Udacity_bot The path to be followed

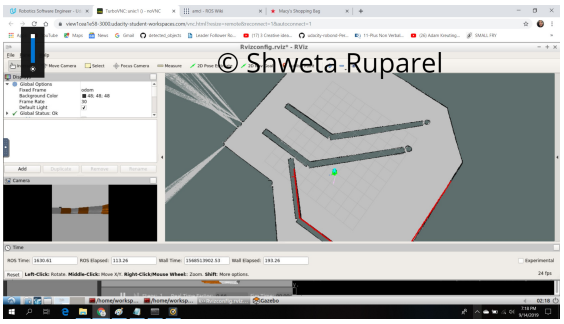

Figure 21. Udacity_bot reached the goal set by 2D nav Goal tool

3.1.2 Safety Bot test results

Safety bot with different base and configuration was created and the parameters tuned for udacity bot were used to check the performance.

- **safety_bot tested for the navigation goal set by the supplied file.**

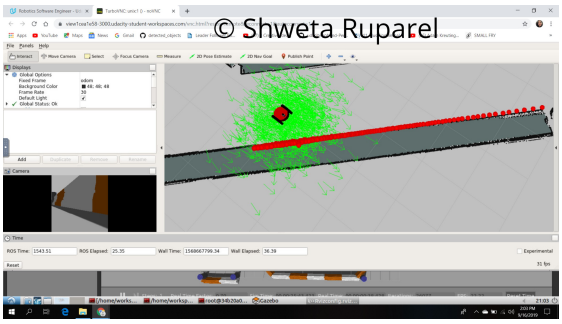

Figure 22. safety_bot initialised in rviz

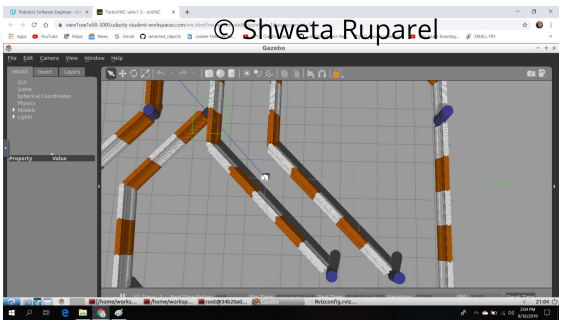

Figure 23. safety_bot initialised in Gazebo

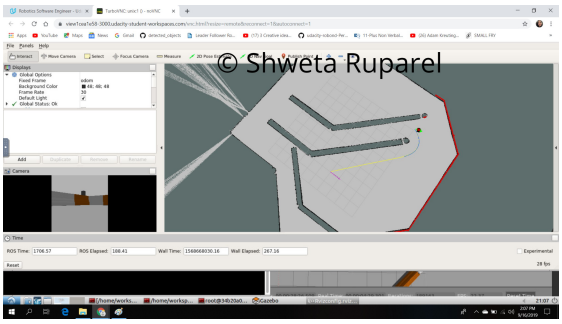

Figure 24. safety_bot path calculated for navigation goal

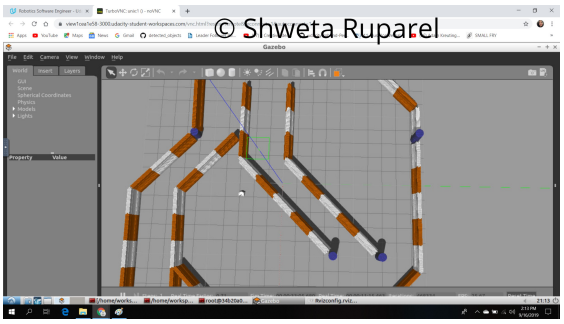

Figure 25. safety_bot reached the goal set by the navigation_goal file in Gazebo

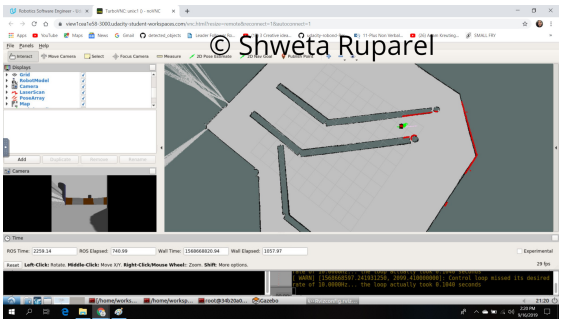

Figure 28. safety_bot reached the goal set by 2D nav goal tool

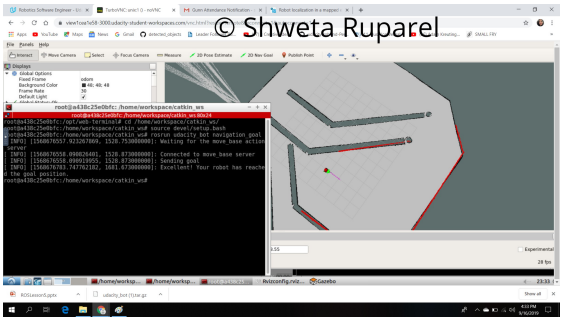

Figure 26. safety_bot reached the goal set by the navigation_goal file in rviz

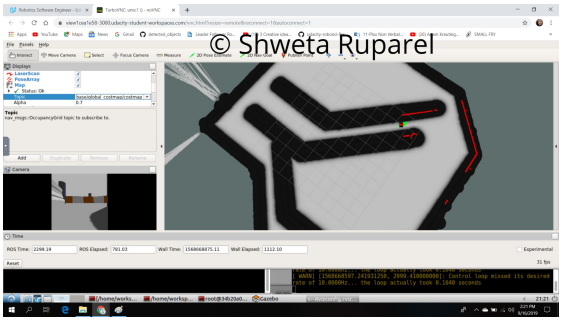

Figure 29. safety_bot shown in global map

- safety_bot tested for the 2d nav goal set using rviz tool.

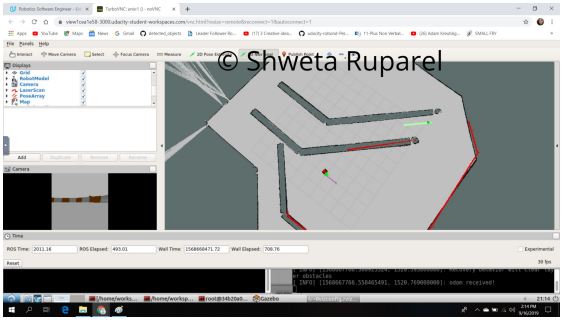

Figure 27. safety_bot initialised in Rviz. Shows the navigation Goal by a green arrow

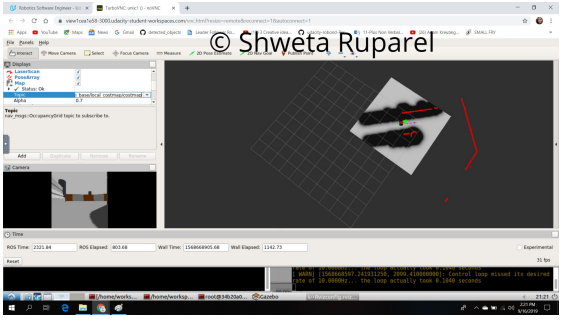

Figure 30. safety_bot shown in local map

3.1.3 Robot Models

Two different models are used to test localization using amcl package and

the results are analysed and compared.

- udacity_bot Udacity bot is a cuboidal base robot with two caster wheels, two wheels on left and right side.It has two sensors , a camera and a laser rangefinder(Hokuyo rangefinder).Refer the figure for the udacity provided robot model.

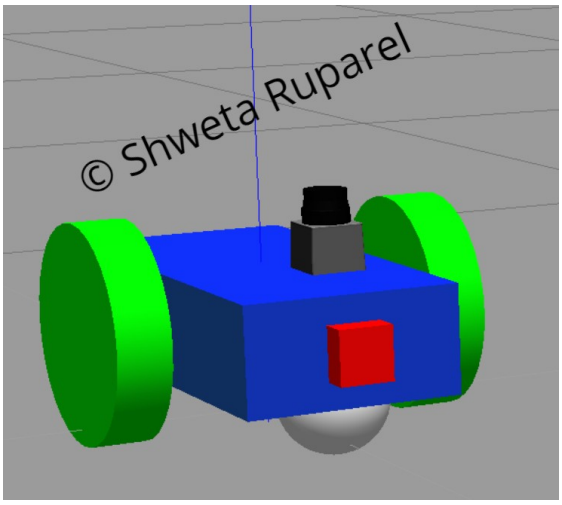

Figure 31. Udacity Bot

- safety_bot Safety bot is a small light weight cylindrical based robot created to test the localization.Safety bot is a small cylindrical based robot with two caster wheels , two driving wheels , a camera and a sensor just like udacity bot. This implementation can be used in applications like safety check inside home or mapped boundary outside home to track any alarming activities.Figure below shows the safety bot .

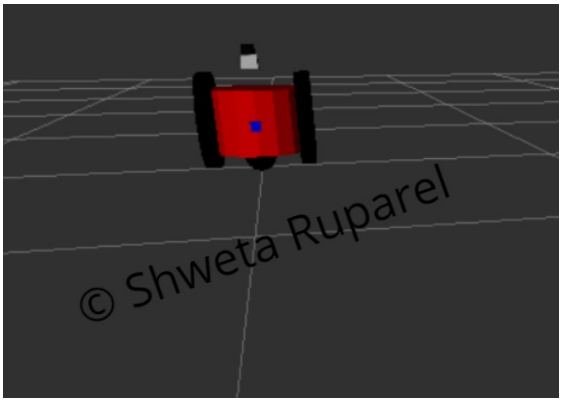

Figure 32. Safety bot

4 Model Configuration

The goal of the navigation stack is to move a robot from one position to another position safely (without crashing or getting lost).It takes in information from the odometry and sensors, and a goal pose and outputs safe velocity commands that are sent to the robot.

The **ROS Navigation Stack** has many different parameters that can be tuned to obtain a good performance. Here we show the most important in our experience. There are 4 files that contain the parameters to set:

- *base local planner params* :The parameters that determine the behavior of the global and local planners.
- *costmap common params*:The parameters common to the two

costmaps: footprint of the robot, the obstacle layer, inflation layer, and static layer;

- *local costmap params*:The parameters of the local costamp.
- *global costmap params*:The parameters of the global costamp.

One of these parameter is max_vel_x :0.85, that is the maximum velocity that the robot can reach, and is kept to a relatively low value. The path that the local planner has to simulate is longer, so there are more possible trajectories to compute,but the time to compute them is the same, so it's more likely that the Navigation Stack will crash the robot with higher values of velocity,since it has little time to decide where to go.

Another parameter that depends on the computing power is sim time , this parameter indicates how long the trajectory is simulated beforehand. In this case it is set to 1.5 seconds.

The update frequencies of the local and global costmaps are strictly related to the computing power, and especially increasing the update frequency of the local costmap can have good effects on the path produced. These parameters are set as 10 updates per second in the local costmap and 2 per second in the global costmap.

These are the default parameters for pdist_scale , the weighting for how much the controller should stay close to the path it was given, and gdist_scale,the weighting for how much the controller should attempt to reach its local goal, while for occdis_scale, the weighting for how much the robot should attempt to avoid obstacles, its value is raised to 0.05 from 0.01,because with the smaller value the robot passes too close to the obstacles and this can cause the robot to crash.

meter_scoring is set to true, that means that instead of using the cells of the grid map as units, the distances computed by DWA are in meters.

Costmap parameters tuning is essential for the success of the planners. The costamp is composed of a static layer, a obstacle layer, and an inflation layer. The obstacle layer tracks the obstacles as read by the sensor data. Inflation layer is an optimization that adds new values around obstacles (i.e., inflates the obstacles) in order to make the costmap represent the configuration space of the robot. To have smooth paths the following values for these two parameters in the inflation layer are good:

cost_scaling_factor is inversely proportional to the cost of a cell. Setting it higher will make the decay curve of the values around the obstacles more steep. infation_radius controls how far away the zero cost point is from the obstacle in meters

ROS navigation has two recovery behaviors. These behaviors will be run when DWA fails to find a valid plan. After each behavior completes, DWA will attempt to build a plan. If plan-

ning is successful, DWA will continue the normal operations. Otherwise, the next recovery behavior in the list will be executed. The behaviors are: clear costmap recovery and rotate recovery. Clear costmap recovery is basically reverting the local costmap to have the same state as the global costmap removing all obstacles outside of the rectangular region in which it can rotate in place. Rotate recovery makes the robot rotate 360 degrees in place. Sometimes rotate recovery will hit an obstacle during the rotation and cause worse problems, so this is not used.

AMCL Parameters

Parameters	values
min_particles	25
max_particles	1000
transform_tolerance	0.2
kld_err	0.05
kld_z	0.99
odom_alpha1	0.005
odom_alpha2	0.005
odom_alpha3	0.010
odom_alpha4	0.005

base local Parameters

Parameters	values
controller_frequency	10
max_vel_x	0.85
max_vel_theta	0.7
acc_lim_theta	3.5
acc_lim_x	2.5
acc_lim_y	2.5
occdist_scale	0.05
meter_scoring	true
sim_time	1.5

common costmap Parameters

Parameters	values
obstacle_range	3.0
raytrace_range_x	5.0
inflation_radius	0.7

5 DISCUSSION

Both the robots were able to localize in the mapped environment.

At the very first time when the parameters were not tuned correctly , the robot was moving very slowly , also getting bumped into the walls. There was no recovery behaviour parameters set to help the robot tackle the situation and come out to move to the goal.

Tuning the parameters for amcl and move_base packages are some of the main and important aspects to obtain accurate trajectories and smooth movements.

The move_base package helps navigate the robot to the goal position by creating or calculating a path from the initial position to the goal, and the amcl package will localize the robot.

The Particle Cloud display shows the particle cloud used by the robot's localization system.The spread of the cloud represents the localization system's uncertainty about the robot's pose As the robot moves about the environment, this cloud shrink in size as additional scan data allows amcl to refine its estimate of the robot's position and orientation.

Both the robots could successfully locate the goal position supplied by

udacity and 2D nav goal position set in rViz using the 2D nav button.

- *The time taken by Udacity bot was better than safety bot. It moved fast and reached the goal in less time that safety bot.*
- *Few parameters were adjusted for safety bot but it did not add in better performance , so for the simplicity of the project , the urdf file to define the configuration of the model is the only file to change to test two different models.*

Due to the nature of particle filter used in MCL, where the convergence process of hypotheses (called particles) causes the absence of particles in some areas, could lead to a localization failure if the robot is kidnapped to that area.This will be an area to be explored and use-cases to be experimented with.

6 Conclusion / Future work

Both the robots localized the mapped environment accurately avoiding the walls . They successfully navigated and localized the mapped environment using particle filter.

It was observed that a well constructed robot model help in smooth mobility and well tuned parameters for trajectories, costmap and amcl resulted in accurate and timely localization.

Few points can be noted for future enhancements.

- At present the only sensor that plays the most important role is Laser range finder Hokuyo.With the IMU sensor and rotary encoders for wheels , the navigation can be more accurate and specific for reaching the safety area targets.The real challenge would be when the robot is implemented in the real world and there is a lot more noise and dynamic environment challenges to be faced.
- Size of the robot also plays an important role. Based on the need of application , the robots could be made big/ small , heavy/light. For example, in case of a safety bot that is required to navigate in the house making sure that it does not bang into anything and checks the safety parameters while navigating along the waypoints or autonomously, it is preferable to build a robot that is small and light weight.
- In future , the parameters can be tuned to increase the speed but not at the cost of accuracy. Both kind of robots could be useful in domestics as well as industrial application.The accuracy is most important in navigation and localization needs.

REFERENCES

[1] Monte Carlo Localization: Efficient Position Estimation for Mobile Robots.
Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999,

[2] Intelligent robotics and autonomous agents by Sebastian Thrun and Wolfram Burgard and Dieter Fox

[3] Probabilistic Robot Localization in Continuous 3D Maps
Bachelor's Thesis, Advisor: Prof. Dr. Joachim Hertzberg, Dr. Thomas Wiemann

[4] ROS Navigation Tuning Guide