Project: Where am I?

Rastri Dey

Abstract—This report elucidates the most intriguing setback in the field of autonomous robots, "the mobile robot navigation in a given state environment". Localization, is a precursor to the cognitive actions and motion controls pursued by a robot, through the perception abilities of its sensors. A number of probability based localization algorithms such as Kalman Filter, Particle Filter/ Monte Carlo localization are introduced in this paper and a comparison has been drawn on their ability to determine a precise robot pose in a given mapped environment. ROS packages are used in conjunction with the localization algorithms to formulate the Robot motion in a simulated environment of Rviz and Gazebo. A new robot architecture for the application of Medical assistance in disaster management, is proposed and an insight to its modelling, sensor configuration and actuator layout is provided. Localization performance utilizing the ROS package parameters, is evaluated under the two case studies of robot, the one that has been provided and the one that is built.

Index Terms—Mobile robots, Localization, Position estimate, Monte Carlo localization, kidnapping detection

I. Introduction

Localization of a robot significantly depends on the aptness of the robot in estimating its location in a mapped environment. A typical measurement state, comprises of position (x,y) and orientation (ɵ) of the robot, gathered from the onboard sensors. Key to a successful localization technique is not only in its ability in predicting the pose estimates, but also in measuring the uncertainty in those estimates owing to the noisy data observations. A robust approximation of a robot position entails in the identification of associated probabilities of calculated positions. Several methodologies from the past has been proposed as way out to the localization issues with algorithms such as Bayesian filter, Kalman Filter, Particle Filter, Markov, Grid etc. Mathematical models extracting the features of fused sensor data from the mapped environment coupled with their respective probabilities is in core to such algorithms. This report incorporates detailed analysis on a few of such algorithms, namely Kalman Filter, Extended Kalman Filter, Monte Carlo Localization (Particle Filter) and Adaptive Monte Carlo Localization. A test bench setup for localization on a supplied robot is then analysed with adaptive monte carlo localization.

Further on, inspired from the concept of ICARUS's search and rescue operations (SAR) in human crisis management [1], a robotic model is designed for medical assistance in disaster prone areas, named as 'Robo-Med Assist'. This robot is equipped for applications such as delivery of medical supplies, searching and evacuation and getting access to the otherwise unreachable areas of disasters. Such a robot would be highly helpful in areas of explosions, for instance where a reduced number of personnel requirements replaced by robots, would prevent costing more lives. The robotic design is executed on Rviz and Gazebo platforms through URDF and launch files in a simulated environment. Localization and tracking of the proposed robot along with the Udacity provided robotic model is carried out on a software based platform for developing robotic applications namely ROS[2] through its localization package of AMCL. Other ROS packages like, move_base from navigation stack and trajectory planner ROS, are also utilized to provide the robot, a local cost map in relation to the global cost map as the robot traverses through the simulated map of Clearpath Robotics.

II. Background & Formulation

A mobile robot without knowledge of its location, is incapable in making any cognitive decision to command its controllers for actuator actions. This indirectly suggests a state of immobility in mobile robots, which contradicts the fundamental purpose to the existence of mobile robots. The measurements obtained from sensors such as GPS are prone to uncertainties to a few meters, a localization algorithm can provide a more accurate estimate of these measurements by processing the fused data from multiple sensors. Tracking of a robot helps in processing decisions and taking actuator commands based on the current state of the robot. On a broader scale, there are three aspects of the localization issue: Local Localization, Global Localization and Kidnapped Robot problem. Local localization is about tracking a robot given its initial position. This is the most basic position tracking problem, solution to which forms the basic blocks of a localization system. Global localization on the other hand, is an advanced challenge to predict the pose of the robot without a prior knowledge on the robot's initial position. The prediction in a Global localization problem, is only relative to the ground truth map of the environment. Uncertainty in measurements and action responses of robot is inherent to this problem. The third and the most challenging of the localization issue is the Kidnapped robot problem, where the robot gets instantly transported to a different location without

being told during its operation. The mechanical and sensor faults, erroneous noise in measurement data are some of the chief reasons to a typical kidnapping condition, a check on this event is highly useful for fault detection in robot state. Solution to all these concerns in an autonomous robotic system perspective will be to have a robust localization approximation that can adaptively manoeuvre in a real world dynamic environment.

Various localization algorithms such as Grid, Markov, Kalman or Particle Filter are founded to ascertain a safe robotic navigation system. These techniques relies on a probabilistic approach, wherein the incoming measurement data, accompanied with its noisy and partial observations, is used to calculate a state estimate on a maximum likelihood hypothesis. Fused sensor measurements encompassing the extracted feature information from the environment is used in updating the likelihood of position and orientation estimate at every iteration. The crux to such probabilistic framework is to maintain a recursive probability distribution called belief over all the positions in a state space environment, iteratively over time.

Localization based on Kalman and Particle Filters forms the underlying basis to many of the advanced localization techniques. A localization based on AMCL (Adaptive Monte Carlo Localization), a variant of Monte Carlo Localization is preferred in this project due to its significant advantages over Kalman in various aspects. The real world measurements gathered from the sensor data could be of any nature for instance, contrasting the assumption of linear Gaussian distribution of data in Kalman. MCL allows for wide variance in measurement data with no assumptions. It also allows for multimodal likelihood distribution caused under the circumstances of random environment clutter, occlusion, sensor failure or collisions. Unlike Kalman, MCL is also effective in Global Localization, wherein a robot can predict its pose relative to the ground truth map without any previous knowledge of its position. Owing to the above stated reasons, MCL is considered to be the best fit localization technique for both the robots designed in this project.

Further insights into the algorithmic design of Kalman Filter, Particle Filter and their advanced variants are discussed in the following sections.

a. Kalman Filter

Kalman filter is an estimation technique that uses a sequence of measurements observed over time, to provide estimates to the robot pose through a joint probability distribution of the states for each timeframe. The measurement data is usually accompanied with noise and other inaccuracies, which brings in the use of sensor fusion that relies on data from multiple sensors instead of focusing on a single measurement alone. The Kalman filter works on a weighted average called Kalman Gain responsible for generating more precise sates than the ones from the measured data. The weights are calculated from the covariance, a measure of the estimated uncertainty of the prediction on system's state.

State Prediction:

$$x' = Fx$$

$$P' = FPF^T + Q$$

Measurement Update:

$$y = z - Hx'$$

$$S = HP'H^T + R$$

Calculation of Kalman Gain:

$$K = P'H^T S^{-1}$$

Fig. 1. Kalman Filter equations

The result of the weighted average is a new state estimate that lies between the predicted and measured state, and has a better estimated uncertainty than either alone. This process is repeated at every time step, with the new estimate and its covariance updating the prediction used in the following iteration. Kalman filter works recursively this way to estimate the next state based on the prediction covariance update from the last best estimate of history.

Calculation of Posterior State and Covariance:

$$x = x' + Ky$$

$$P = (I - KH)P'$$

Fig. 2. Kalman Filter equations (Contd.)

Kalman Filter even though being a great estimator, possess a few drawbacks. Firstly, the estimation with Kalman filter is limiting to its linear operations on motion and measurement models with white Gaussian noise. This restricts the robot from executing any non-linear motion like for a movement on a curve. This clearly is highly idealistic for a real world scenario where a motion can never be truly linear. Second, the state space using Kalman is a unimodal Gaussian distribution which carries out the state update probabilistically, as a whole under a Gaussian belief. This limits the behaviour of Kalman Filter to recover from a collision situation, where the uncertainty accounted, can deviate from its assumed unimodal nature.

The stated issues of Kalman Filter can be resolved with a better estimation technique called Extended Kalman Filter (EKF). EKF is an advanced variant of Kalman Filter, wherein a local linear approximation is used to update the covariance of the estimate. The linear approximations are made using Taylor Series or Jacobian matrices with partial derivatives for multidimensional state space. This technique helps in alleviating a few of the drawbacks of Kalman Filter assumptions and allows the underlying state transition or measurement function to be nonlinear.

b. Particle Filter

Particle filter (PF) is another estimation technique, that are sequential Monte Carlo methods under the Bayesian estimation framework where the key is to represent the next

probability density function (PDF) of the states by a set of random samples or particles with their associated weights. The issue of weight disparity and collapse as encountered by many estimation techniques, can be resolved with the resampling step in Particle Filter. In the resampling step, the particles with negligible weights are replaced in every iteration by a new set of particles in the proximity of the particles with higher weights. The resampling process of keeping beliefs with higher weights and neglecting the rest, keeps improving the pose estimate with every iteration.

The below code snippet shows the MCL algorithm in two parts of: Motion and Sensor Update (outlined in Red) & Resampling (outlined in Yellow). The state estimates from the motion and sensor update through a defined mathematical modelling undergoes a state change in every iteration. Each state represents a robot's belief by a set of weighted hypotheses (samples), which gets updated through random resampling after the motion and sensor updates, in every iterating cycle.

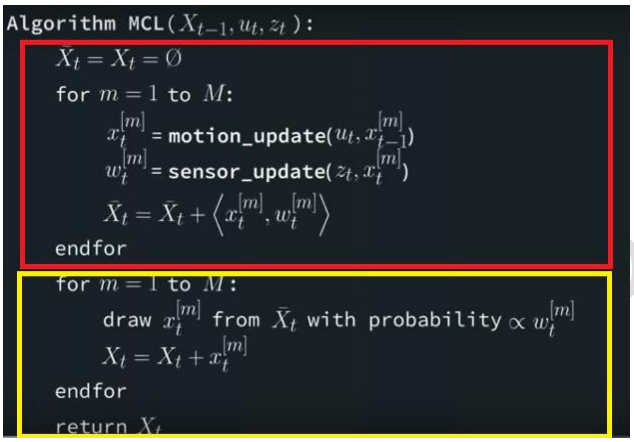

Fig. 3. Particle Filter Algorithm

PF performs at par where Kalman fails. It works even with the complex stochastic characteristics with inherent nonlinearity and randomness in sensor measurements and state environment. The performance of PF depends on the number of particles and its initialization. A higher number of particles generates a better pose approximate though at the expense of higher computational effort. This allows for memory and resolution control with MCL. It is also an apt technique to address the Global Localization issue, where the robot can localize itself without any prior knowledge of its initial position. Moreover, MCL allows for a multimodal discrete state space, enhancing the tracking performance even at the extreme conditions of collision or sensor failures. Overall, Particle filter proves to be an extremely efficient algorithm in terms of accuracy of the state vector estimate, robustness, tolerance to measurement noise and ease of implementation.

c. Compare & contrast

An effective and robust filtering technique is key to a mobile robot tracking application. Several factors that are crucial in comparing an estimation technique and weighing one above another typically involves: estimation accuracy, adaptability to environmental randomness and noise, robustness, efficiency, memory requirements and ability to handle localization challenges such as Robot kidnapping or multirobot localization problem. In this report emphasis is laid on two of the estimation techniques namely: Kalman Filter and Particle Filter. A comparison has been drawn for the same, in this section.

Kalman Filter is an estimation technique employed for the localization of a robot by fusing data incoming from various onboard sensors. This is an optimal filter for a linear model with Gaussian noise. In contrast, Particle Filter contains no underlying assumption of linearity in sensor measurements or state environment. EKF on the other hand approximates the state vector with a Gaussian, which then undergoes a first order linearization of the nonlinear system with the help of a Jacobian. The series approximation in EKF however, leads to a poor depiction of the nonlinear functions and their associated probability distributions. This at times, engenders a diverging filter nature in the estimation cycles of EKF. The unimodal state space in EKF poses another of its distinguishable disadvantages over Particle Filter leading to a further restriction of keeping a collision-free state environment which is impractical in a real world state. These comparisons draws in clear reliability of Particle Filter over Kalman Filter, in terms of accuracy, tolerance and robustness.

Moreover, Kalman Filter can address only one of the basic mobile robotics localization problems on position tracking called local localization. The absence of an initial position estimate or the measurement data and an appropriate prediction covariance limits Kalman Filter for Global Localization. Particle Filter on the other hand can solve both the local and global localization problems with much ease, owed to its random resampling technique of particles generated over space on weighted hypotheses of Bayesian framework. But similar to Kalman Filter, Particle Filter is not suitable for addressing the kidnaped robot problem. This is due to the nature of particle filter where the convergence of particle hypotheses over time causes the degeneration of particles in some areas, causing a localization failure for the robot kidnapped in that area. Particle filter holds the internal state belief generated over iterations and fails to account for any sudden change in robot location.

As is clearly evident there are a numerous advantages of Particle Filter over Kalman Filter. Therefore, the robot localization in this project is executed with Particle Filter. A variant of Particle Filter in MCL known as AMCL (Adaptive Monte Carlo Localization), used as a ROS package in the localization implementation, has an edge over Monte Carlo Localization. AMCL dynamically adjusts the number of particles over a period of time, as the robot navigates around, in the map. This adaptive process of AMCL, offers the robot a significant computational advantage in a simulated mapped environment.

III. MODEL CONFIGURATION

Two robot models has been setup in this project for executing localization process in a provided world map. The benchmark robot is a differential drive mobile robot with a rectangular box shaped body. Either side of the robot is connected to two base wheels to perform the actuator movements on controller commands. The front and back side of the vehicle body is supported on omnidirectional Caster wheels, preventing any fall during its movement. A typical robotic movement accompanied with differential drive wheels in cartesian world frame, can be represented as:

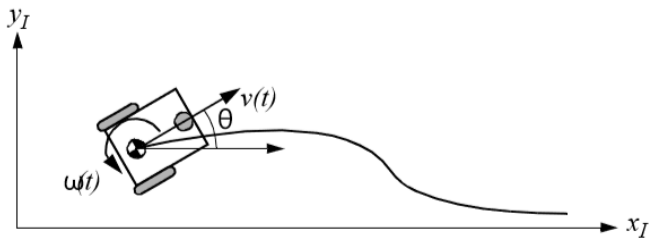

Fig. 4. Differential drive controller in cartesian frame

The layout of the world map, on which both the robotic model traverses till its destination, following a certain trajectory, looks like:

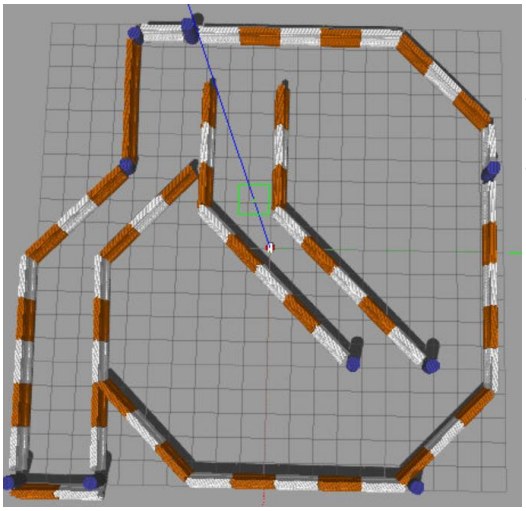

Fig. 5. Simulated map of Clearpath Robotics

The robotic model design, sensor configuration and localization parameters are stated in the following sections. The Udacity supplied benchmark model is discussed first and following that, the custom designed model is illustrated.

a. Benchmark Model

i. Model Design:

A robot design can be configured with its corresponding URDF file. Unified Robot Description Format or urdf, is an XML format used in ROS to define a robot model, its kinodynamic properties, visual elements and model sensors. It describes the robot's rigid links connected by joints in a chain or tree-like structure.

The Udacity supplied benchmark robot looks like:

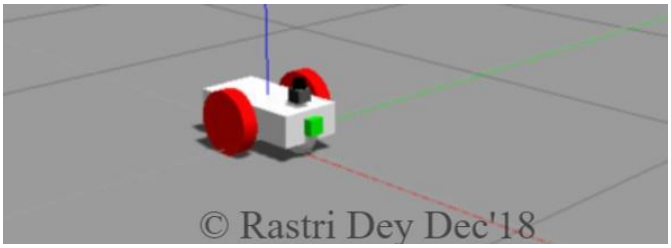

Fig. 6. Representation of supplied Robot model in Gazebo

A clear understanding of the robot can be visualized with the below indications on its length, width and height:

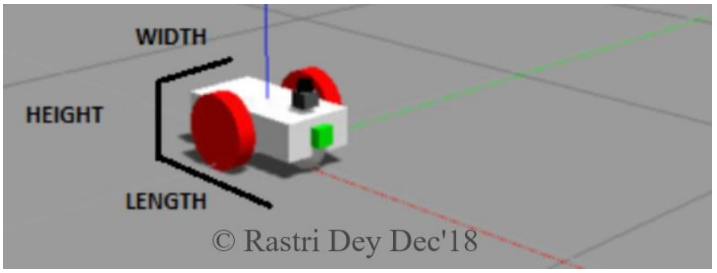

Fig. 7. Representation for Physical dimensions of Robot model

The robot's design consideration, sensor and actuator layout can be represented with the following table:

Model Component	Element Type	Element Dimension	Reference Origin in XYZ world frame		
			X- axis	Y-axis	Z -axis
chassis_visual	box	Length=.4 Width=.2 Height=.1	0	0	0.1
sback_caster_visual	sphere	radius=0.05	0.15	0	0.05
front_caster_visual	sphere	radius=0.05	-0.15	0	0.05
left_wheel_hinge	cylinder	length=0.05 radius=0.1	0	0.15	0.1
right_wheel_hinge	cylinder	length=0.05 radius=0.1	0	-0.15	0.1
camera_joint	box	Length=.05 Width=.05 Height=.05	0.2	0	0.1
hokuyo_joint	box	Length=.1 Width=.1 Height=.1	0.15	0	0.2
Net Robot Dimensions (Length X Width X Height) = 0.425 X 0.35 X 0.25			0.4 + (0.05/2)	0.3 + 0.05	0.2 + 0.05
			0.425	0.35	0.25

Table. 1. Design layout of benchmark model

The table above states the measurements of different model components, their respective geometry and their position with

respect to origin (0,0,0) in Gazebo frame. The net length towards x-, y- and z- axes are to represent the net physical dimension of the robot body. The names in Model Component are in reference to the urdf file used in configuring the robot design.

A 2D representation of the robot layout in FRONT view looks like:

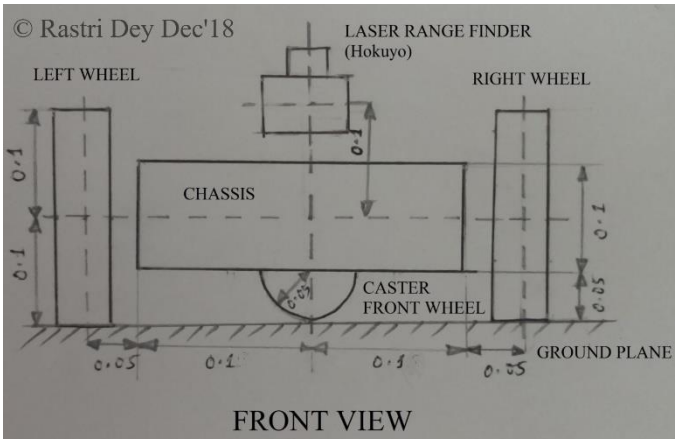

Fig. 8. FRONT VIEW of benchmark robot layout

Similarly, a 2D representation of the robot layout in SIDE view can be shown as:

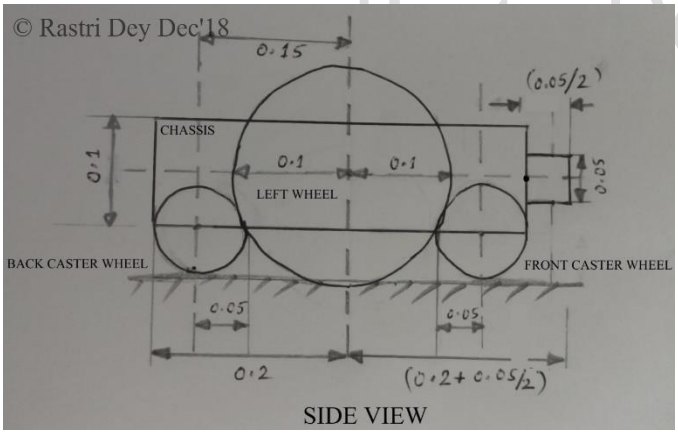

Fig. 9. SIDE VIEW of benchmark robot layout

The above figures and the table gives a clear picture of robot layout and sensor configurations. It should be noted that the figures shown are schematic representation of 2D version of 3D diagrams which means the circles are spheres and cylinders respectively for the (front & back) caster wheels and (right & left) robotic wheels. Also, as it may seem from SIDE view, the three wheels are not aligned in a line, this can be figured out from the front view of the robot model.

ii. Packages Used :

A number of ROS packages used for navigation and motion control are :

- TrajectoryPlannerROS-
  - base_local_planner_params.yaml
  - costmap_common_params.yaml
  - global_costmap_params.yaml
  - local_costmap_params.yaml
- ROS built in localization package-
  - amcl.launch
- ROS navigation stack – move_base

Few of the topics in ROS:

- Differential drive controller command topic: cmd_vel
- odometry topic: odom
- camera topic: image_raw
- hokuyo sensor topic: /udacity_bot/laser/scan

iii. Parameters configuration:

The three categories of ROS parameters configured under amcl package are as follows:

Overall filter parameters:

update_min_a = 0.02
update_min_d = 0.02

The value of rotational movement required before performing a filter update(update_min_a) and the value of translational movement required before performing a filter update(update_min_d) are reduced from their default values of 0.2 meters to a very low value 0.02. The magnitude of default values are considerably high for a slow moving robot like this. A reduction in this values is highly useful in the robot's state update. Its observed that a considerable reduction in this value reduces the randomness in particle update to its lowest minimum, thus reducing the uncertainty in robot pose estimation to a significant extent.

min_particles = 500

The minimum number of particles are set to a value of 500. The higher the number of particles the better is the approximation, though at a cost of increase in computational power. In this scenario a higher number of particles was not a mandatory option and even the default value would have sufficed. Yet for a better localization approximation throughout the robot movement, the minimum number of particles are set to 500. The higher value resulted in a very accurate localization as the robot traversed towards its goal position, with no significant change in computational power.

recovery_alpha_slow = 0.001
recovery_alpha_fast = 0.1

The exponential decay rate for the slow average weight filter and fast average weight filter, used in deciding when to recover by adding random poses are set to 0.001 and 0.1

respectively. These values are tuned as per the recommended values of ROS package summary.

Odometry model parameters:

odom_alpha1 = 0.5
odom_alpha2 = 0.6
odom_alpha3 = 0.4

A higher range odom_alpha's are tuned to overcome noise from the odometry data. A higher value of odom_alpha1 improves the localization while the robot is executing a turn. This value compensates for the expected noise in odometry's rotation estimate from the rotational component of the robot's motion. Similarly a high value of odom_alpha2 reflects the nature of covariance in noise due to the dependency of translational component on rotational component. This parameter helps in robot localization, during transition from translational to rotational movement. Finally a higher odom_alpha3 is to compensate for noise in the translational movement of robot as it goes along and reaches the destination. It's been observed that noise in odometry translational component is lesser than the noise due to rotational components. Due to this reason, the values of odom_alpha1 and odom_alpha2 are tuned to 0.5 and 0.6 and odom_alpha3 is tuned to 0.4.

Laser model parameters:

laser_sigma_hit = 0.001

The laser model parameter of standard deviation for Gaussian model has been given a low value suggesting the laser rangefinder sensor to be a better sensor with high accuracy data outputs. This also makes the robot localization more dependent on the laser sensor than odometry.

laser_max_beams = 45

An increase in number of evenly-spaced beams in each scan, gives a higher resolution data. This helps in avoiding any loss in information due to absence of laser beam hit in the azimuth range of the laser sensor. More data also aids in better estimation.

laser_z_rand = 0.01

The mixture weight of z_rand is reduced from its default value of 0.05. These mixture weights for laser sensor are tuned with rigorous localization testing.

The move_base package is tuned for controller frequency of value:

controller_frequency = 5

The rate at which the control loop runs and sends command for cmd_vel to the robot base has been reduced from its default value of 20 to 5 Hz. The higher default value was difficult to get processed in run time. Lowering the controller frequency helped in smooth running of controller without throwing any error. This value impacts the robot velocity in run time, lower frequency avoids any abrupt velocity update in robot's state.

The parameters with TrajectoryPlannerROS is tuned with the following config files:

base_local_planner_params.yaml-

pdist_scale: 1.0
gdist_scale: 0.4

The parameters influencing the robot movement towards the goal point are namely, gdist_scale and pdist_scale ,where pdist: the weightage for the controller to stay close to the path is increased to 1 and gdist: the weightage for the controller to reach its local goal is decreased to 0.4. This prevents the robot from a high deviation on its planned trajectory and attempts to reach the goal location by slowly following the locally planned path.

xy_goal_tolerance: 0.1
yaw_goal_tolerance: 0.1

The tolerance for the controller in the x & y distance when achieving the goal is kept at 0.1 meters for a precise end goal position. The tolerance for the controller in yaw/rotation is increased to 0.1radians due to unnecessary disturbance in controller loop to adjust for the required yaw value. A reduction in these values ensures a lower error in robot's location at its final destination. Though a very small tolerance might lead to endless controller loop to adjust for a very precise value which increases the settling time and computational costs. Hence an optimized value of xy_goal_tolerance and yaw_goal_tolerance is set to achieve high accuracy localization results in optimal time.

costmap_common_params.yaml:

obstacle_range: 4
raytrace_range: 5

The threshold set for obstacle_range is increased from its default value of 2.5 meters. This is done to acquire more sensory information in cost_map. A high value would include the information available from distant objects as well, which thereby helps in better controller action and path planning. The "raytrace_range" parameter determines the range to which the robot will attempt to clear out free space in front of it. To avoid any misleading information of previous obstacle data, the robot attempts to clear out a range 5 meters ahead of it. It's

been observed that a higher set value of this with respect to obstacle_range helps in better cost_map optimization.

inflation_radius: 1.4
transform_tolerance: 0.2

A higher value of inflation_radius helps in ensuring any collision avoidance of the robot with the environment surrounding it. This narrows down the free space available for the robot to move and sets an equal obstacle cost to any data more than the inflation_radius.
The value for transform_tolerance is kept at its default value of 0.2 as the system is comfortably able to handle that amount of delay.

global_costmap_params.yaml:

update_frequency: 5.0
publish_frequency: 5.0

The rate at which the cost_map gets updated and publish visualizing information is reduced to a lower value based on the performance of VM and the CPU processing speed. The update_frequency is the frequency in Hz at which the cost_map will run its update_loop. Lowering this will lower the update speed of cost_map to match the system processing speed.

width: 30.0
height: 30.0
resolution: 0.03

The width and height of global_costmap is reduced to 30 meters. The width and height are tuned as per the map environment provided. Lowering this value also helps in enhancing the processing speed of run-time for robot in map environment. The grid resolution is set to an optimal value such that no valuable information from the environment is lost while maintaining a considerable processing speed.

local_costmap_params.yaml:

update_frequency: 5.0
publish_frequency: 5.0

The update_frequency and publish_frequency of local cost_map is set similar to global_costmap parameters.

width: 6.0
height: 6.0
resolution: 0.03

The width and height of local_costmap are one of the crucial parameters that influenced the localization process for both the robots in this project. Reducing these values to considerable numbers has helped in improving the robotic performance to a greater extent. A lower width and height values, allows the robot to move to its specific planned path very precisely. In contrast, the higher set values for these resulted in the robot taking multiple turns and navigating in random directions, away from goal.

b. Personal Robot – 'Robo-Med Assist'

i. Model Design:

The robot designed in this project is named as Robo-Med Assist. This robot is a three-storeyed disk platform robot with camera and laser sensors. As the name suggests, this robot is for the application of medical assistance during disaster recovery. In a real world scenario, the utilities related to medical assistance can be kept in these platforms which the robot will transport from one place to another. As similar to the previously described robot, a urdf file in xacro is created for this robot as well.

The customized, Robo-Med Assist looks like:

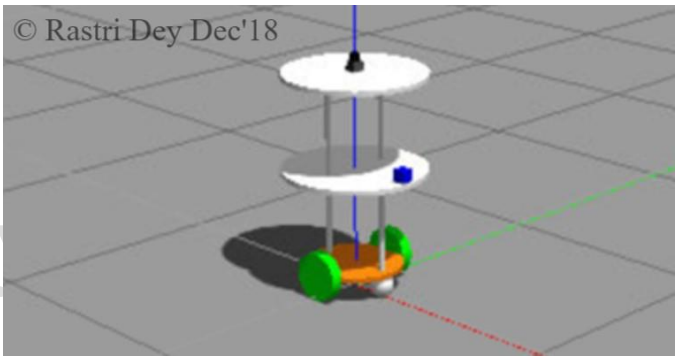

Fig. 10. Representation of designed Robot model in Gazebo

As opposed to the previous robot, the structure for this robot is almost cylindrical and not cuboid. The overall physical dimensions should therefore be parametrized with radius and height. Yet for similar reference of comparison, the physical dimensions are presented in (length, width and height) along with (radius and height), where length and width are equal to the radius of the overall structure.

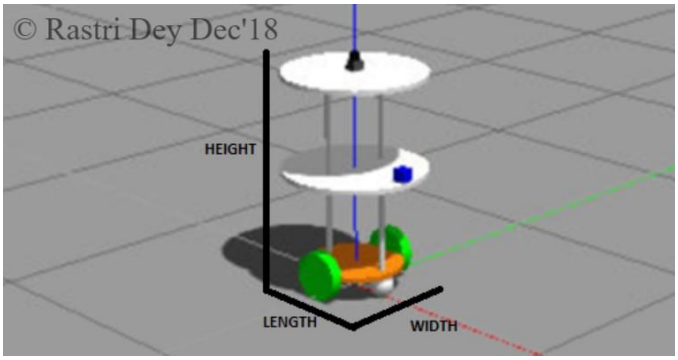

Fig. 11. Representation of physical dimensions of designed Robot model

This robot is having two left and right wheels, two caster wheels, three disks, six supporting legs and two sensors. A complete description on individual component position, dimension, sensor layout and structure is illustrated in the following table:

Model Component	Element Type	Element Dimension	Reference Origin in XYZ world frame		
			X -axis	Y-axis	Z -axis
chassis_visual	cylinder	radius=0.2 length=0.02	0	0	0.1
left_leg_visual	cylinder	radius=0.01 length=0.4	0	-0.15	0.3
right_leg_visual	cylinder	radius=0.01 length=0.4	0	0.15	0.3
front_leg_visual	cylinder	radius=0.01 length=0.4	0.15	0	0.3
back_leg_visual	cylinder	radius=0.01 length=0.4	-0.15	0	0.3
disk2_visual	cylinder	radius=0.3 length=0.02	0	0	0.5
left_leg2_visual	cylinder	radius=0.01 length=0.4	0	-0.15	0.7
right_leg2_visual	cylinder	radius=0.01 length=0.4	0	0.15	0.7
disk3_visual	cylinder	radius=0.3 length=0.02	0	0	0.9
back_caster_visual	sphere	radius=0.05	0.15	0	0.05
front_caster_visual	sphere	radius=0.05	-0.15	0	0.05
left_wheel_hinge	cylinder	length=0.05 radius=0.1	0	0.2	0.1
right_wheel_hinge	cylinder	length=0.05 radius=0.1	0	-0.2	0.1
camera_joint	box	Length=.05 Width=.05 Height=.05	0.27	0	0.54
hokuyo_joint	box	Length=.1 Width=.1 Height=.1	0	0	0.95
Net Robot Dimensions (Length X Width X Height) =0.6 X 0.6 X 1 (Radius X Height) = 0.6 x 1			0.6 Length = 0.6	0.6 Width = 0.6	1 Height = 1

Table. 2. Design layout of customized robot model

The robot geometry having 4 legs are namely- Front Leg, Back Leg, Right Leg and Left Leg in level 1 in between Disk 1 and Disk 2. On top of Disk 2 there are two more legs namely, Right Leg 2 and Left Leg 2 in between Disk 2 and Disk 3. Disk 1 is surrounded by 2 right and left cylindrical wheels and front and back caster wheels. 'camera_joint' as mentioned in the table refers to the camera sensor and 'hokuyo_joint' refers to the laser sensor. The measurements of different model components, their respective geometry and their position are given with respect to origin (0,0,0) in Gazebo frame. The net length towards x-, y- and z- axes are to represent the net physical dimension of the robot body. The names in Model Component are in reference to the urdf file used in configuring the robot design in the project.

A further analysis on shape, size and position can be inferred from table 2 and Fig 12.

A schematic diagram of the robot front view can be shown as:

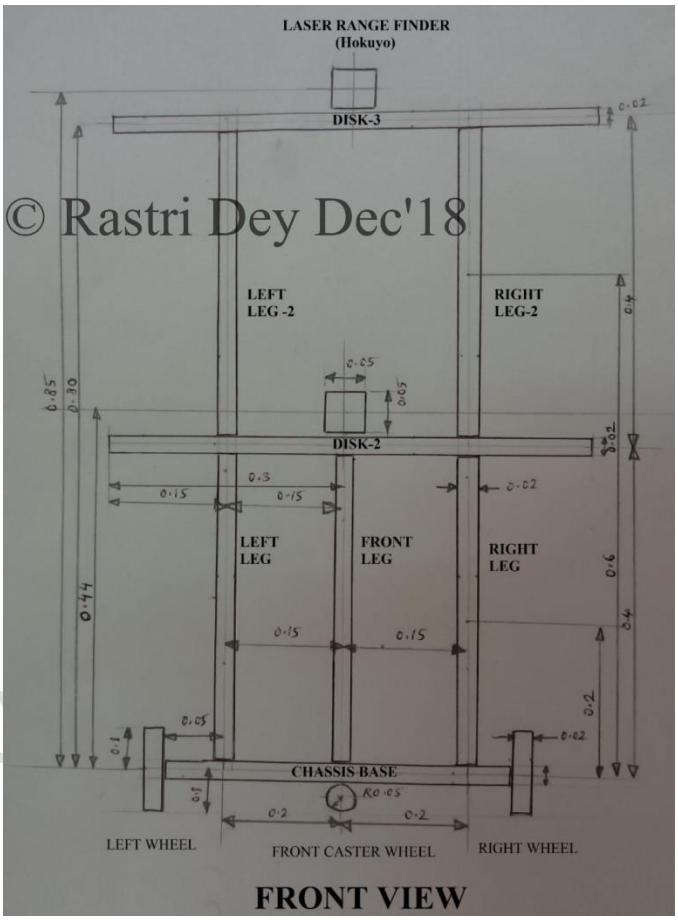

Fig. 12. FRONT VIEW of custom designed Robot

It should be noted that the rectangles in the above figure represent cylinders, and circles represent spheres in a front view 2D perspective. The dimension values, type and positions for sensors and robot footprint are also mentioned in similar ways in the urdf file.

ii. *Packages Used :*

The packages used in Robo-Med Assist are similar to the ones in benchmark model and hence not repeated in this section.

iii. *Parameters configuration:*

The ROS parameter configuration for Robo-Med assist is similar in many aspects to the benchmark model. Here the parametric change is in amcl.launch where the odometry parameters are not optimized and the pdist_scale and gdist_scale are not set in base_local_planner_params.yaml. Its observed that the

robot model performs best with the default values only.

All other parameters with respect to cost_map, move_base etc are set similar to the parameters of the supplied robot.

IV. RESULTS

Two case studies of robot designing and tracking are presented in this project. Both the benchmark model and the customized model are designed in Gazebo world through urdf files. The resulting models are further tested on the localization package of ROS called AMCL. The trajectory planner from ROS navigation stack and amcl are tuned to attain the best fit localization performance on the mobile robots. Due to a very different structural architecture in robot designing, the localization performance attained between the two robots is quite noticeable.

The small dimension and mass value in the supplied robot, allows for the robot to navigate quite rapidly on track following the trajectory very closely. The figure below shows the robot moving towards its goal location. The robot initially starts with a very high random number of particles, which narrows down as it moves along the planned trajectory.

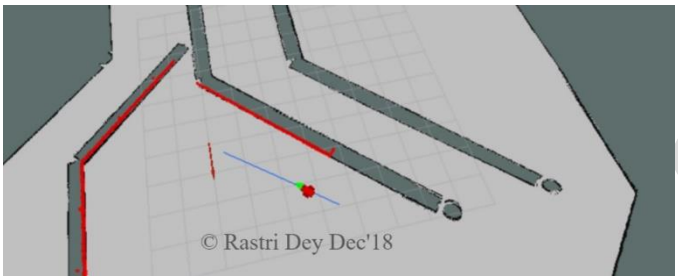

Fig. 13. Robot (in Red) navigating on the assigned trajectory (blue) in the simulated environment of Rviz

Here the arrow in Red shows the final destination for robot to reach, and path in blue is its current local trajectory plan. In green are the arrows for robot pose array depicting the robot localization performance. The position of the arrows denotes the robot identified position and the orientation of the arrows denote the yaw for the robot at its current instant.

The robot once reaching the goal location looks like the following figure in Rviz window:

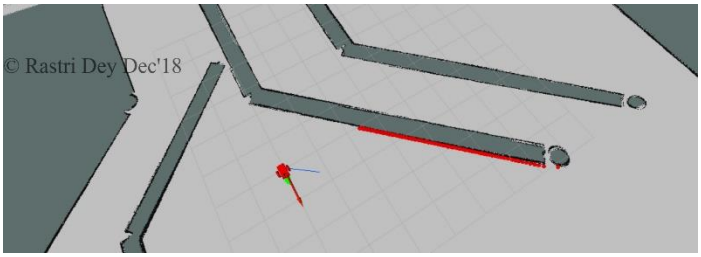

Fig. 14. Robot (in Red) on reaching its Goal location (Red Arrow) in simulated environment of Rviz

sThe small white blob is the actual robot at its goal location.

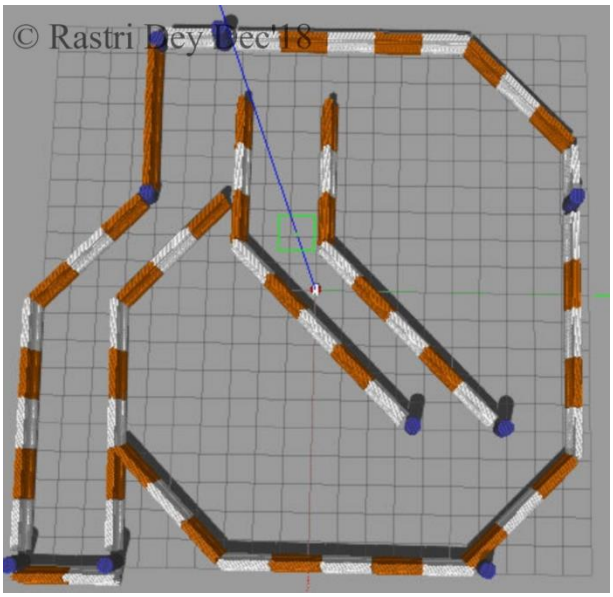

Fig. 15. Robot (in white blob) on reaching its Goal location in simulated environment of Gazebo

The estimate of the robot pose as pose array is shown in Green in the following figure. It can be observed that the Green arrow is overlaid very accurately on the Red arrow, signifying a very precise localization performance.

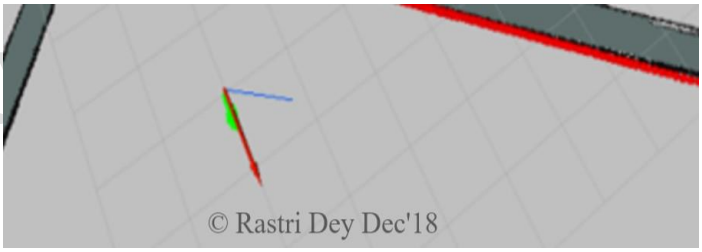

Fig. 16. Robot localization performance, Pose array (Green) & Goal location (as Red Arrow) in Rviz

The robot performance in the custom made robot is different in marginal aspects from the benchmark model. For instance the sensor positioning in this robot is different to the supplied robot, which changes the incoming data information from sensors. Moreover, the robot is structurally taller, owing to change in the robot's overall inertia and velocity vector as it moves along the path. The following figure shows the robot movement on its local trajectory:

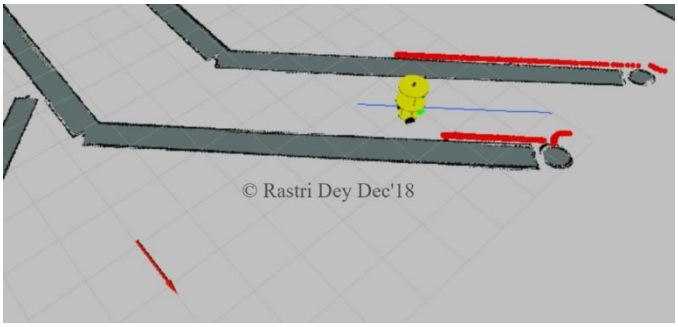

Fig. 17. Custom designed Robot (in Yellow) navigating on the assigned trajectory (blue) in the simulated environment of Rviz

Here the continuous red dots represents the laser hits from the hokuyo sensor implanted on the robot head. Following figure represents the robot on reaching its destination location:

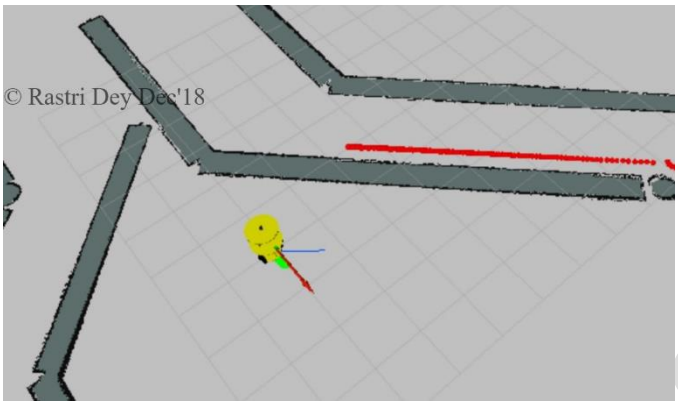

Fig. 18. Custom designed Robot (in Yellow) on reaching its Goal location (Red Arrow) in simulated environment of Rviz

The pose array display of robot's final position relative to the assigned destination can be depicted in the figure below:

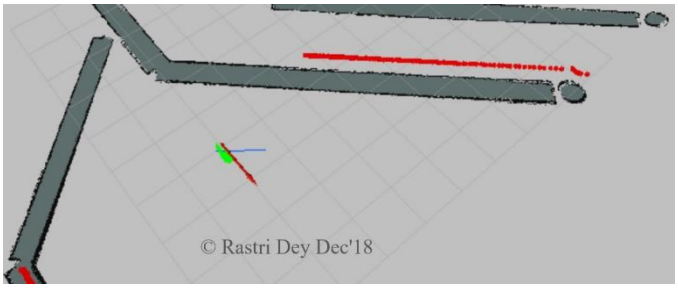

Fig. 19. Custom designed Robot localization performance, Pose array (Green) & Goal location (as Red Arrow) in Rviz

The arrows in green are very close to the assigned goal location of the arrow in Red. The orientation of the arrows in Green are also very near to the arrows in Red depicting a very accurate yaw estimate of the robot. Here, the marginal difference in performance due to the robot structure and sensory positions, with respect to the supplied robot can be neglected. Overall the result analysis on the robot localization signifies an extremely well performance for both the robots. The robots maintain their trajectory path and reaches their destination at a good speed and conduct themselves very well while avoiding any obstacle collision on its way.

V. Discussion

The robot designed in this project is having multiple storage racks for its application to keep medical aids, food, water and other supplies, that can be transported to the areas of emergency. The design is supported by multiple pillars at each level to make the structure stand firmly on to the ground wheels. The omni-directional caster wheels provides additional degree of freedom to the robot movement. The sensors supported for data extraction are fitted to each robot level. A camera is fitted at the second level and a laser range finder at the third. The positioning of laser range finder at the third level provides a future implementation of 360 degree LIDAR sensor at the top, for an enhancement in data collection. The practical implementation of this robot will need a further detailed designing for an adaptive control and a robust hardware.

The localization for both the robots are supported by ROS AMCL and navigation stack package. The parameters tuned in AMCL and trajectory planner of ROS is under a controlled environment in this case and hence the robot localizes itself with quite ease at a very short interval of time even with lower number of particles. This although is opposing to the real world case, where a large number of particles and computational power are bottleneck to a robust design and localization problem. An overall performance for the robot to localize itself worked really well for this project, giving highly precise results.

The two robots are able to localize themselves with high degree of precision and complete the tracking process throughout the trajectory planned. Although due to a lower mass and smaller structure for the supplied robot, it acted better than the custom designed robot. The pose array for the benchmark robot is overlaid exactly on top of the desired goal position and orientation. For the custom designed robot whereas, the robot's goal position is very near to the desired position but not exactly overlaid on each other, the difference is though quite insignificant and only at a sharp focus it is noticeable. This is due to a small structure, mass and chassis inertia in case of supplied robot. Moreover change in sensor layout and positioning are also factors owed to the slight difference in localization results. There is also speed difference noticed between the two robots, the supplied robot moving at a higher speed than the customized robot, the difference again is very minimal.

It can be further inferred that the robot localization would fail with the existing AMCL algorithm i.e., when the robot is moved to a different location. This is because AMCL is a bayesian algorithm that holds an internal belief or likelihood of the world which is difficult to get updated at abrupt changes in the environment. One generalized solution to resolve the robot kidnapping problem would be to reset the AMCL past belief to a random uniform distribution of particles, when placing the robot to a new location. This would give equal

weightage to particles which are no more dependent on the state history as is the case due to bayesian filter.
AMCL is a an adaptive way to change the number of particles over a period of time as the robot navigates around the map and become more precise in its localization process. For an industry where accuracy and speed, both are crucial factors of influence, AMCL provides a better solution as it adapts for the computational speed with accuracy. In other words, when the robot position is very accurate a lesser number of particles are employed to save the additional resources of computing power. This helps in proper resource planning and cost consideration for industrial application of robotics domain.

VI. Conclusion & Future Work

Designing a mobile robot is foremost to its creators to justify the application for which it will be utilized. A robust hardware design accompanied by precise localization are key to mobile robotics. The performance for both the robots in terms of design, planning and execution even though being highly satisfactory in this project, possess some scope of improvement. An autonomous robot as designed here for an application of disaster recovery, should be able to operate on partially unknown or completely unpredictable environments. The robot should be tested under a wide range of uncontrolled environments where the capability of the robotic controllers and actuators could be tested for longer durations. This calls for the requirement of highly robust design and implementation in mobile robots that is flexible to the emergent behaviour of environment, sensor noise or mechanical failures. The robot design implemented here, can have more levels for its application to allow in wider space to carry medical supplies. There could be additional number of sensors for a 360 degree view to access the entire environmental space at every instant, a 360 degree 3D Velodyne Lidar could be a good recommendation in that case. The design of the robot can also be enhanced to box shaped compartments instead of disk shaped surface to allow for safer transports. Better localization algorithms can be implemented for faster convergence of robot's state pose. Robot's accuracy can also be improved with advanced algorithms, though this might lead to more computational time, for instance a higher number of particles can be tuned for particle filter but this may slow down the processing speed. Hence an optimal balance between accuracy and processing speed is required for an efficient robot navigating system. The localization approach of the robot can also be challenged by a regressive testing of robot to come out of difficult situations of robot kidnapping or sudden collision at the risk prone areas. The robot's ability to track dynamic motion profiles with an appropriate mathematical motion model and localization algorithm are key to future enhancements for automation in robotics technology.

VII. References

[1] Armbrust, Christopher & De Cubber, Geert & Berns, Karsten. (2014). ICARUS Control Systems for Search and Rescue Robots.
[2] http://wiki.ros.org/