SUMMER RESEARCH INTERNSHIP

Particle Filter-based localization of a Mobile Robot by Using a Single Lidar Sensor under SLAM in ROS Environment

Instructor: Professor Seul Jung
Dept. Of Mechatronics Engineering
Intelligent system and emotional Engineering(ISEE)
Chungnam National University

Project done By-
Dhruv Talwar
Mechanical Dept.
IIT Delhi

Acknowledgements

I would like to express my heartiest gratitude to Prof. Seul Jung for giving me this opportunity to work as a summer research intern in his lab ISEE (Intelligent Systems and Emotional Engineering), Chungnam National University, Daejeon Korea. This was a life-time opportunity for me as not only I gained experience about different experiments, equipment's and devices I also gained a lot from the culture that Korea has to offer. I am also thankful to Prof. SK Saha, Department of Mechanical Engineering, IIT Delhi, for providing me with such an opportunity to be a part of the JNC-R (Joint Network Center-Robotics) Program between India and Korea and Prof. SV Modak, Department of Mechanical Engineering, IIT Delhi, for his supervision throughout the internship.

ABSTRACT

One of the most popular issues in autonomous robots is mapping, localizing and autonomous navigation in an indoor environment. A probabilistic localization algorithm known as the Adaptive Monte Carlo Localization (AMCL) uses a set of weighted particles to approximate the position and orientation of a robot. In this paper we focus on Adaptive Monte Carlo Localization or particle filters method and how effectively it localizes our mobile robot. Bayesian algorithm was used to building grid map in which our robot was to be localized. The robot moving path was computed by the path planner algorithm. During the experiments, we try to localize and autonomously navigate our mobile robot in gazebo and Rviz environment. After deploying the AMCL and tuning its parameters to our specific mobile robot, the particles quickly converge on the pose and the robot was successfully able to follow the path to reach its goal position. The robot was successful in reaching its goal every time. These results thus point out that AMCL is a practical localizing algorithm and performed effectively in these simulations. Different simulations were done with static map as well as dynamic maps.

We chose the ROS platform for mapping and localizing the mobile robot this is because compared with conventional software development platform, ROS have strong focus on usability and ease of installation and have the advantages of free and open source. Moreover, ROS has a lot of plugin and libraries which were very useful to realize real world objects, sensors and scenarios in simulation. The idea was to demonstrate that it I very easy and useful to use ROS platform which would greatly shorten the development cycle of the robot and the SLAM could easily realize on ROS, the robot can realize autonomous moving.

TABLE OF CONTENTS

1. Introduction

With a rapid development in the field of robots, it is now expected of the upcoming robots to have the capability of mapping and localization. Without knowing the current pose and orientation the robot cannot take any intelligent decisions and actions on its own. Localization of the robot in an environment plays a vital role to solve many problems related to automatic navigation.

This is where we introduce SLAM whose main goal is to acquire the map of the unknown environment while simultaneously localize the robot's pose in the map. SLAM is a necessity if one wants to achieve obstacle avoidance global and local path planning and ultimately autonomous navigation. In the last 20 years, a lot of progress has been made with SLAM and a lot of its challenging tasks have been tackled some of them including computation complexity, data association and loop closure in a single robot.

Much research is done in the field of autonomous navigation. Grisetti *et al.* [1] presented many innovative methods to reduce the number of particles in the Rao-Blackwellized particle filter. This algorithm takes in account the movement as well as the sensor updates to get the distribution of the robot in an environment. Imthiyas [2] did research on indoor localization, they collected large data sets from the indoor environment and produced a Gaussian Process(GP) model and using this model they localized the robot in the indoor environment. Dieter Fox[3] proposed the Kullback-Leibler distance sampling(KLD sampling) which uses a small number of samples to represent the belief of the robot. It generated samples until the number of particles is enough to guarantee that the KL distance between the robot's actual position and estimate does not exceed a certain bound error. This technique uses adaptive statistical method based on particle filters. A new development in autonomous navigation and SLAM is the Fast-Slam [4] where a particle filter is used and the paths for the robot and data associations are now represented as samples for the particle filter. Much work has been done on this recent technique Fast-Slam. Montemerlo *et al.* [5] proposed a FastSLAM2.0 algorithm which gives more weightage to the most recent measurement in the process of pose estimation. Perez *et al.* [6] did a comparison of the performance of localization for a mobile robot based on different sensors like vision systems and laser rangefinders. They calculated the precision of the different localizations and compared them. They concluded saying a laser scanner is a more robust solution than vision systems.

Motivated by the AMCL algorithm we will try to localize a robot model in a simulated environment. Once an occupancy grid or map of the environment is formed we will try to get the pose and orientation of our robot in the map frame of reference. For this work we will just use a single hokuyo lidar and implement SLAM in ROS.

1.1 Robot Operating System (ROS)

Robot Operating System [7] is a Linux based software, has many prebuilt libraries and packages which can be modified and changed accordingly to build robot functions' its framework, it used nodes, packages, messages topic and services. Any executable code written which takes data from any of the robot's sensor and passes the same or the output value to another code is termed as a ROS Node. The ROS system is like a send and receive system, the data from the sensors, which are called messages are sent to nodes from ports called as topics. A node that sends messages on a topic is called a publisher node and the node which subscribe to the messages sent via another topic is termed as the subscriber node. All nodes are combines in a ROS package which can very easily be combined on any Linus software with ROS installed [8].

1.2 System Hardware and Software

For the experiments conducted in Simulation we used Ubuntu 16.6 and ROS Kinetic version. For the real-life Simulation, we used Ubuntu 14.04 and ROS Indigo.

The Lidar used in this project was the Hokuyo URG-04LX-UG01. It is a laser class 1 safety. The Laser beam diameter is less than 20mm at 2000mm with maximum divergence 40mm at 4000mm [9]. But due to the small range of this lidar we later changed our lidar to UTM-30LX. The specifications are given below.

Table I: URG-04LX-UG01 Specifications

Parameter	Value
Light Source	Infrared rays
Wavelength	785nm
Scan Area	240 deg
Maximum Radius	4000 mm
Pitch Angle	0.36 deg

Table I: UTM-30LX Specifications

Parameter	Value
Light Source	Infrared rays
Wavelength	785nm
Scan Area	270 deg
Maximum Radius	30000 mm
Pitch Angle	0.25 deg

2. Mapping and Localization of mobile Robot

2.1 Iterative Closest Point

For the mapping of the environment both in real life as well as in gazebo environment we have implemented the ICP algorithm to match the point values generated by the lidar to get an transformation for the new point cloud with respect to the previous point cloud. The ICP algorithm is the most used matching algorithm in the 3D or 2D point cloud registration, which optimizes the matrix through iteration. In each iteration, for each point in the target point set, the algorithm will find its nearest point in the reference point set. Then these corresponding points will be used to calculate the corresponding rotation matrix and the translation vectors, which are applied to the target point set to get the new target point set and enter the next iterative process. Finally, it will get the excellent conversion matrix to achieve the accurate registration of the two-point sets. The implementation steps of ICP algorithm are summarized as follows two-point set P1 , P2 :

(a) Screening point pairs: for the point in the P1, search out its nearest point in the P2, and make up a point pair; Find all the point pairs in the two-point sets. The point pair collection is the equivalent of two new point sets that are calculated efficiently;

(b) According to these point pairs, namely two new points set, calculate two centers of gravity;

(c) According to the new point sets, calculate the rotation matrix R and the translation vector t(in fact, according to the difference of center of gravity);

(d) Get the rotation matrix R and the translation vector T, we can calculate the new point set P2 through the rigid body transformation of the point set P2. Then calculate the sum of the distance square between P2 and P2 , regard the absolute value of the difference between the two successive distance squares as the basis of convergence, if the absolute value is less than the threshold, the algorithm converges and stops the iteration.

(e) Repeat the steps (a)-(e), until the algorithm converges or reaches the given number of iterations

Input: Two-point clouds: A = $\{a_i\}$, B = $\{b_i\}$
An Initial transformation: T_o
Output: The correct transformation, T, which aligns A to B

Table II ICP Pseudo code -Algorithm

1: $T \leftarrow T_o$

2: **while** not converged **do**

3: **for** i ← 1 **to** N **do**

4: $m_i \leftarrow \text{FindClosestPointIn}(T.b_i)$;

5: **if** $\|m_i - T.b_i\| < d_{max}$ **then**

6: $w_i \leftarrow 1$;

7: **else**

8: $w_i \leftarrow 0$;

9: **end**

10: **end**

11: $T \leftarrow \text{argmin}\{\sum w_i . \|m_i - T.b_i\|^2\}$;

12: **end**

Iteratively repeating these two steps typically results in convergence to the desired transformation. Because we are violating the assumption of full overlap, we are forced to add a maximum matching threshold $d_{max.}$ This threshold accounts for the fact that some points will not have any correspondence in the second scan (e.g. points which are outside the boundary of scan A). In most implementations of ICP, the choice of d_{max} represents a tradeoff between convergence and accuracy. A low value results in bad convergence (the algorithm becomes "short sighted"); a large value causes incorrect correspondences to pull the final alignment away from the correct value. Thus the ICP algorithm is used by us for the purpose of mapping the environment.

2.2 Particle Filter

In probabilistic robotics belief is represented by conditional probability distributions. It assigns for every possible state a probability. To determine the belief of state $bel(x_t)$ we need to consider all sensor and action data that happened before time t. This is denoted in the following equation:

$$bel(x_t) = p(x_t, |z_{1:t}, u_{1:t})$$

Here $z_{1:t}$ is all the sensor data from timestep 1 until time step t. The control data, actions by which the robot changes the world, is denoted as $u_{1:t.}$ The key idea of the particle filter is to represent the belief $bel(x_t)$ by a set of samples drawn from $bel(x_{t-1})$. The samples of a distribution are called particles and are denoted as:

$$X_t = x_t^{[1]}, x_t^{[2]}, \ldots., x_t^{[M]}$$

In equation SO SO each particle $x_t^{[m]}$ with $(1<m<M)$ is a hypothesis to what the true world state may be at time t. X_t is the set of all M particles. Because a particle filter is to approximate the belief $bel(x_t)$ by the set of particles X_t, the equation is as follows:

$$x_t^{[m]} \sim p(x_t, |z_{1:t}, u_{1:t})$$

The particle filter algorithm constructs the belief $bel(x_t)$ recursively from the belief $bel(x_{t-1})$ one timestep earlier. So, since the belief is represented by the particle set as in equation 9 the particle filter construct a new set of particles X_t from the set X_{t-1}. The particle filter has two important parts. The first part generates a hypothetical state $x_t^{[m]}$ on time t based on the particle $x_{t-1}^{[m]}$ and control u_t. This step involves sampling from the next state distribution
$p(x_t, |u_t, x_{t-1})$. Then the importance factor $w_t^{[m]}$ is calculated based on measurement z_t and the calculated particle $x_t^{[m]}$. The importance factor is interpreted as the weight of each particle. The resulting $\bar{X}_t$ is a temporary set with weighted particles. The second part is called the importance re-sampling and redraws a new set X_t with replacement from the temporary set $\bar{X}_t$ and contains only the particles with a high probability. So over time the set is converge to the hypotheses that produces the most likely output. The particle approximation is the core part of the MCL algorithm, which itself came by with some new improvements. The advantages of the algorithm is that it copes well with noisy data, it concentrates only in interesting space regions[16].

2.3 **Adaptive Monte Carlo Localization**

In this study we have used the most popular localization technique which is by the Adaptive Monte Carlo approach. Monte Carlo localization (MCL), also known as particle filter localization, is an algorithm for robots to localize using a particle filter. Given a map of the environment, the algorithm estimates the position and orientation of a robot as it moves and senses the environment. The algorithm uses a particle filter to represent the distribution of likely states, with each particle representing a possible state, i.e., a hypothesis of where the robot is. The algorithm typically starts with a uniform random distribution of particles over the configuration space, meaning the robot has no information about where it is and assumes it is equally likely to be at any point in space. Whenever the robot moves, it shifts the particles to predict its new state after the movement. Whenever the robot senses something, the particles are resampled based on recursive Bayesian estimation, i.e., how well the actual sensed data correlate with the predicted state. Ultimately, the particles should converge towards the actual position of the robot.

These particles each have their own coordinate values and orientation just like the actual robot along with a given weight. The weight value (w_t) is defined as the absolute difference between the actual pose of the robot and the predicted pose by that specific particle. The bigger or larger the weight of the particle the more accurately it defines the pose of the robot. Whenever the robot moves in the environment and gives new sensor data the particles are re-sampled. With each re-sample, particles that have low weights perish and the particles with high weights survive. After many iterations of the AMCL algorithm, the particles will converge and evaluate an approximate of the robot's pose This algorithm estimates the robot's orientation and position based on the sensor's input to the algorithm[16].

Table III AMCL Pseudo code -Algorithm

1: **procedure** AMCL(x_{t-1}, v_t, s_t)

2: $X_t \leftarrow \varphi$

3: for n =1 to N loop:

4: $x_t^{[n]} \leftarrow$ Motion Update(v_t, $x_{t-1}^{[n]}$)

5: $w_t^{[n]} \leftarrow$ Sensor Update (s_t, $x_t^{[n]}$)

6: $X_t \leftarrow X_t + < x_t^{[n]}, w_t^{[n]} >$

7: end for

8: for n=1 to N loop:

9: draw $x_t^{[n]}$ with probability $\propto w_t^{[n]}$

10: $X_t \leftarrow (X_t + x_t)$

11: end for

12: return X_t

The algorithm can be broken down into 2 steps Motion movement and sensor update and resampling Process. In this algorithm the robot's pose is represented by a belief factor (X_t). Initially, all the particles have equal weight (w_t)and the belief state are estimated by randomly generating N particles. The Algorithm takes into consideration of the previous belief state values X_{t-1}, the movement command V_t, and the sensor measurement

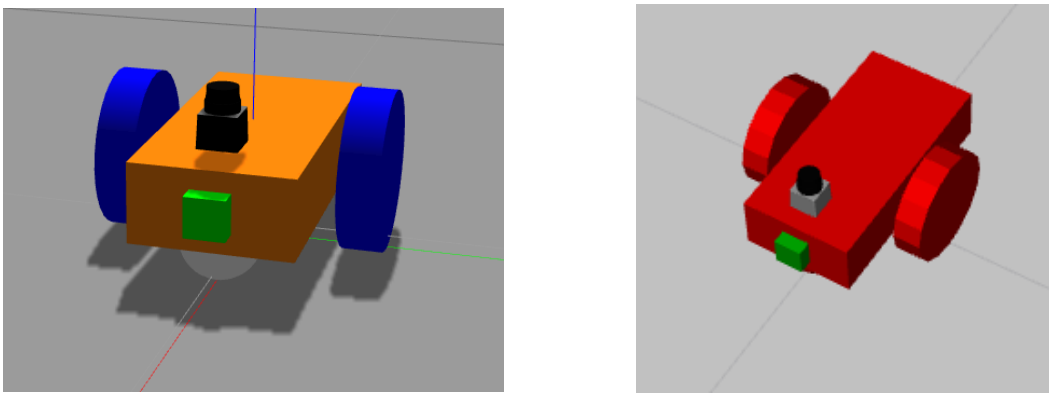
S_t as the initial starting input. In the first iteration of the AMCL algorithm, a hypothetical state is calculated as the initial position of the robot, As the robot's position is updated the laser scanner sends different and updated measurements to the ROS nodes, using these measurements, the particles' weight is recalculated. The result of this iteration are added to the previous belief state and thus getting a new belief state which corresponds to the location of the robot. [14][16]

In the other part of the algorithm, resampling of the particles are done, as each belief state has a weight value associated with it, after each resampling iteration the particles that have a high weight value survive and are then used in the next iteration while those particles with a low weight value perish during the resampling. Finally, with the new sensor measurements the algorithm estimates the position of the robot by calculating the belief state. [16]

3. Simulation in ROS

3.1 Robot Model

A simulated robot and environment is always useful while performing experiments. As in a simulation even if the robot collides or crashes no actual harm will be done to the sensors or the robot and then the experiment can be repeated. Similarly, by having a simulated environment we can change it according to our specific needs. For this study we created a Unified Robot Description Format (URDF) file. This URDF specifies the visual, inertial, and collision properties including the location, shape, and origin. Thus, we made a two-wheel differential drive robot to simulate in ROS. The robot had two caster wheels to stabilize the robot from tipping in the z-axis. Also, note since URDF is being used instead of the Gazebo standard of sdf, the gazebo ROS package is used to spawn the robot model. Robot wheels are links added by using a continuous joint between the chassis and the wheels. After making the hardware of the robot we then added a hokuyo laser and a camera to the robot body. The positioning of the lidar is such that it remains unobstructed by the chassis of the robot. Gazebo does have preexisting sensors, but since ROS is being utilized, plugins for each sensor must be used. Thankfully, many of these plugins can be found online and thus, one is created for each wheel joint for motion, one for the camera, and one for the rangefinder. For the robot, a differential drive controller is used. This type of controller specifies many of the upcoming parameters. The last important step to creating the robot is to add color. This can be done by defining RGB values in the. xacro file or by defining the actual color given by Gazebo in the .gazebo file. Fig. 1 shows the robot model in both the Gazebo(a) as well as the RVIZ(b) environment. In both the figures we can see the Lidar(a black box on the top of the robot) and the camera sensor (a green cuboid in front of the robot).



(a) URDF Model in Gazebo (b) URDF Model in RVIZ

Fig. 1. The robot URDF model in different simulating environments

3.2 Gazebo Environment

The environment we used for this experiment was an empty world with a few obstacles placed. We chose this environment as it was quite easy to map, also we can at any given time add or remove obstacles according to our wish. Moreover, simulations in this environment did not require heavy computations making the simulations consume less time.

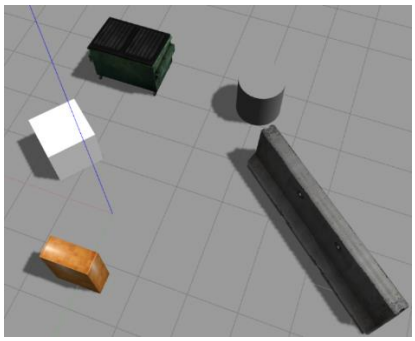


Fig. 2. Gazebo environment

3.3 AMCL and the ROS Navigation Stack

The navigation stack in ROS uses a move_base package to simulate the model of the robot to an end goal position. The move base package uses a cost map, in which the obstacles are the occupied black areas and the areas where the robot can move without colliding with an obstacle is denoted by free gray space. The navigation stack in ROS helps the robot to get out of a not so favorable position by rotation. By Tuning the AMCL parameters we can define how well we can localize our robot. [13]

3.3.1 AMCL Parameter Tuning

The main goal for the AMCL parameter tuning is that all the particles in the filter after a certain iteration must converge in on a pose of the robot which is very close to the actual pose. The AMCL package has parameters related to the following subheading particle filter, laser scanner and the odometry. In this study we are localizing our robot without any source of odometry and using just a single laser scanner. The tuning is done just for the first two sub heading-

1. Particle Filter- These are the consequential parameters along with their definition which effect the particle filter and ultimately the localization.
   (a) Min particles – Minimum number of particles to be considered by the filter while estimation of the pose of the robot.
   (b) Max particles – Maximum number of particles to be considered by the filter while estimation of the pose of the robot.
   (c) Transform tolerance – Time with which to pose-date the transform that is published, to indicate that this transform is valid into the future
   (d) Kld err - Maximum error between the true distribution and the estimated distribution
   (e) Update min_d – Translational movement required before performing a filter update.
   (f) Update min_a – Rotational movement required before performing a filter update.
   (g) Resample interval – Number of filter updates required before resampling

Increasing the particle numbers increase the accuracy of the estimate of the pose but in turn increases the computational power required. The transform_tolerance determines that for how long will the current transform will be accepted for localization. A good transform_tolerance removes any kind of lag in the system. Lowering the kld error value shall increase the accuracy at the cost of computation time and power. Less value for update_min_d means more updates for a given distance which means more accuracy which in turn will result in a high accuracy of localization. Similarly, for update_min_a, lowering these values will increase the iterations which can result in a high accuracy localization but with more iterations time and computation power will increase. Increasing the resample interval will also put a lot of stress for the processing.

2. Laser scanner Parameters - These parameters are directly related to the amount of data we want from the sensor.
   (a) laser_max_beam- How many evenly-spaced beams in each scan to be used when updating the filter
   (b) laser_max_range – Minimum scan range to be considered.
   (c) laser_likelihood_max_dist – Maximum distance to do obstacle inflation on map.
   (d) laser_z_hit- Mixture weight for the z_hit part of the model
   (e) laser_z_rand- Mixture weight for the z_rand part of the model

Laser_max_beams and laser_max_range values are dependent on the lidar used.

TABLE IV: AMCL PARAMETERS

AMCL PARAMETERS	**VALUE SET**
MIN_PARTICLES	4000
MAX_PARTICLES	6000
TRANSFORM_TOLERANCE	0.3
KLD_ERR	0.01
UPDATE_MIN_D	0.01
UPDATE_MIN_A	0.1
LASER_MAX_BEAM	50
LASER_MAX_RANGE	-1.0
LASER_LIKELIHOOD_MAX_DISTANCE	4.0
LASER_Z_HIT	0.9
LASER_Z_RAND	0.5

3.4 Path Planners

In this section we will discuss how does the simulated robot will find obstacles and then recalculate its global path from its starting to the goal point while avoiding obstacles in the path.

3.4.1 Global Path Planner

For the Global path, the ROS navigation package uses the Dijkstra's algorithm It is made according to the static map that we have. The global cost map does not get updated with the real time. Dijkstra finds a network of paths from the start position to the goal position. The path only includes the places where the robot can go that is the free spaces in the map. It avoids the spaces that has an obstacle in the map. Each free space it reaches is given a cost value. The more the cost value the longer the path. The

minimum cost associated with a path to the goal is the shortest path. The global planner tries to divide the map into many nodes for each free cell available on the map, but the result is not smooth, and some points and nodes are not suited for the vehicle geometry [12][10].

3.4.2 Local Path Planner

The local planner breaks the long global path into smaller achievable targets or waypoints for the bot to follow. The Local cost map considers dynamic obstacles while generating a small local path. The Local Cost map is directly updated by the sensor which is feeding in the data. The local planner just uses the map which at that time the sensor can comprehend and according to the output of the sensor it generates a local path. This local path is the one which avoids the obstacles in between and tries to follow the path as closely as possible to the global path. The Local path planner follows the DWA Dynamic Window Approach. In this algorithm the robot has a sample of parameters (delta x, delta y, heading angle). For a sample velocity, the algorithm will iterate a series of possibilities, if that velocity is applied to the robot's current state or pose for a short time for different heading direction. The algorithm will then give a score to each trajectory that the algorithm finds, depending upon the feasibility of the path. The trajectory that has the highest score is associated with the robot base and it becomes the Local Path[12][10].

3.5 Move_base Parameters

There are 4 configuration files for the move_base package which helps in the generation of path.

1. **Local Cost Map Parameters**: The frame of the local cost map is the odometry frame as we wish the local cost map to be updates as our robot moves. We have also set the width and height of the local map. The update frequency gives at what rate the local map is to be updated and the publish frequency gives at what frequency the local map shall display the information. The resolution of the local map is chosen according to the resolution of the map formed. We have used a rolling window version of the cost map. The static_map is set to false as we want to continuously update the local map with the new founds sensor data [10]

TABLE V: LOCAL COST MAP PARAMETERS

Local Cost Map Parameters	**Values Set**
Global_frame	Odom
Robot_base_frame	Chassis
Update_frequency	10.0
Publish_frequency	1.0
Static_map	False
Rolling window	True
Width	6.0
Height	6.0
Resolution	0.05

2. **Common Cost Map Parameters**: These are the parameters that the ROS navigation package uses to store information about the world and the obstacles in it. The obstacle_range and raytrace_range parameters will tell the maximum distance that the laser scanner will read and add information into the cost map. Obstacle_range detects if there is any obstacle closer to the laser scanner than 2.5 meters (as set) it will update the cost map and put an obstacle in it. raytrace_range does quite the opposite it, clears out the cost map and updates the free space. The footprint parameter indicates the actual geometry of the robot. It helps in keeping a good estimated distance so that we know if the robot can fit through a window or door. inflation radius helps to have a minimum distance between the obstacle and the geometry of the obstacle. We have also set the tolerance for the position of the robot, that is the xy_goal_tolerance. Lowering the position tolerance will give us more accuracy, similarly we set the yaw_goal_ tolerance which is the tolerance in the heading of the robot when it reaches near its goal [10].

TABLE VI: COMMON COST MAP PARAMETERS

Common Cost Map Parameters	**Value Set**
Obstacle_range	2.5
Raytrace_range	3.0
Robot_radius	0.2
Inflation_radius	0.1
XY_goal_tolerance	0.05
Yaw_goal_tolerace	0.05

3. **Global Cost Map Parameters**: The parameters of the Global Cost map almost the same as those of the local cost map. The width and height are set higher than the local cost map. The static_map parameter is set to true as in this case we need a static map [10].

TABLE VII: COST MAP PARAMETERS

Global Cost Map Parameters	**Values Set**
Global_frame	Map
Robot_base_frame	Chassis
Update_frequency	1.0
Publish_frequency	0.5
Resolution	0.05
Static_map	True
width	10.0
height	10.0

4. **Base Local Planner :** This configuration file helps the ROS navigation stack calculate a local path for the robot using the parameters set. In this we have set the max and min velocity in the x and y direction with which the robot will move while following the path. We also have set the max and min angular velocity, max and min acceleration and minimum in place angular velocity. Setting these values too high can result in the robot collide with an obstacle as the speed of the robot might be faster than the map or path update rate. We have assumed a non-holonomic model of the robot. We also have pdist_scale which is with what weight shall the controller try to stay close to the global path. Setting this too high can cause the robot being very close to the obstacle. gdist_scale which is with what weight shall the controller try to reach the local goal independent of the global path. Increasing this will make the robot stay away from obstacles but in turn make the path the robot take quite long. There must be a balance between the pdist_scale and gdist_scale parameter. [12][10]

TABLE VIII: BASE LOCAL PLANNER PARAMETERS

Base Local Planner Parameters	**Value Set**
Max_vel_x	0.5
Min_vel_x	0.01
Min_vel_y	0.0
Max_vel_y	0.0
Max_vel_theta	1.5
Min_in_place_vel_theta	0.02
Acc_lim_x	0.5
Acc_lim_y	0.0
Acc_lim_theta	1.5
Pdist_scale	0.80
Gdist_scale	1.5
Holonomic_robot	False

4. RESULTS

4.1 Simulation Results

The result of the AMCL was seen in two scenarios, one is in simulation in Gazebo and RVIZ and the other using a lidar in real life. In both results the AMCL performed reasonably well and it was able to localize the robot in both the cases. During the simulation the particles converged to give the pose and location of the robot in the given map. Other than this simulation were carried out to autonomously navigate the model of the robot in gazebo using the ROS navigation stack

4.1.1 Mapping of the simulated Environment

We can see the map of the environment as made by the laser scanner. It can be clearly seen that the boundaries of the obstacles are marked as black and the free space is colored white. The unexplored region by the robot is marked in gray color.

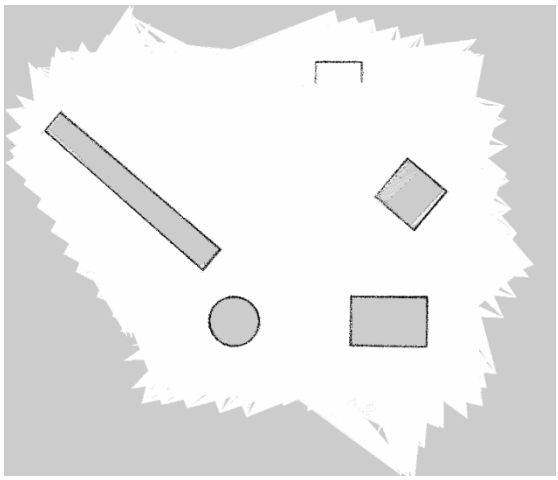


Fig. 3. Map of the Gazebo Environment

We can compare this map to Fig. 2. which shows the original environment. The map clearly shows all the obstacles present and clearly defines the accessible and non-accessible parts of the map.

4.1.2 Localization in RVIZ environment

Below we can see the results as given by the AMCL algorithm in the RVIZ environment. The images below show the static map, laser reading, and particles in the AMCL algorithm. The robot is localized according to the map frame as shown in Fig. 4 which is fixed in the global map, and relative to its position we can get the coordinates of the robot

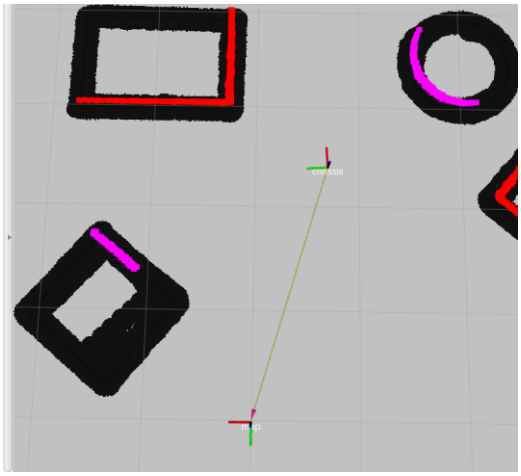


Fig. 4. Bot chassis with respect to the origin of the map.

When we first spawn our bot in the environment we can see that the particles are uniformly spread. This shows that the probability of locating the or localizing the robot is distributed in the map. These are those particles which are randomly generated by the AMCL algorithm as shown in Fig. 5(a). Now we give a goal to the robot so that it follows a pre-defined trajectory, visualizing the pose array of particles we can now see that as the bot moves the particles have converged to the real estimate of the position of the robot as shown in Fig. 5(b).

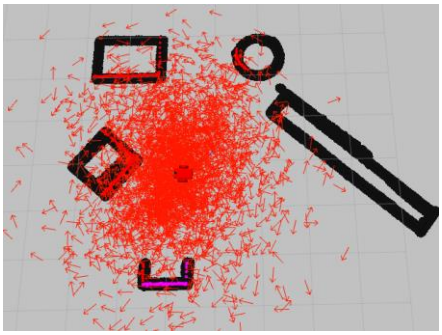


Fig. 5(a). Particle array when Bot is spawned

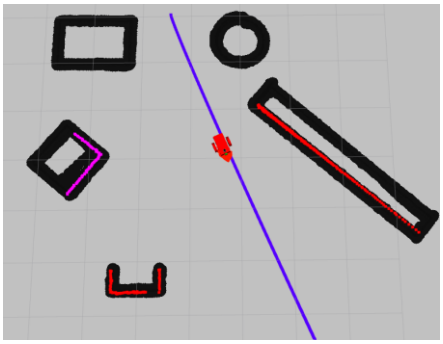


Fig. 5(b). Particle array in between goal

Finally, when the robot reaches its destination we can see the particles now also forming a cluster near the robot showing the position of the robot. The particles now generated by the particle filter are now only showing those particles with high weights. This can be seen in Fig. 5(c). given below

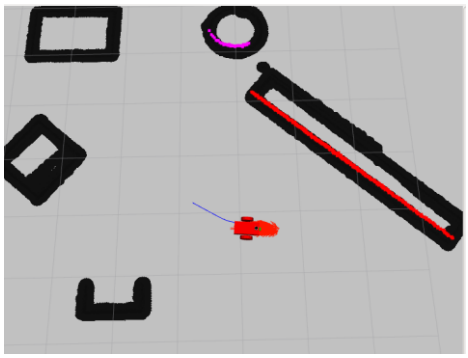


Fig. 5(c). Particle array when the robot reaches its goal

For the final part of the localization of the robot in the environment we now subscribe to the amcl_pose topic which will now show the robot position with its x,y and z coordinates in the map frame, as shown in Fig. 6. Given below

```
header:
  seq: 464
  stamp:
    secs: 528
    nsecs: 110000000
  frame_id: "map"
pose:
  pose:
    position:
      x: -1.54586932062
      y: 0.343360164908
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.858403736905
      w: 0.512974682092
  covariance: [0.0035980093555294523, 0.0015499088637265235, 0.0, 0.0, 0.0,
 0.0015499088637265235, 0.023564238457272554, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.015410630198473357]
---
```

Fig. 6. Coordinates of the robot in the environment

We can also calculate the yaw or heading of the robot by the quaternion, the computational formula of the angle[9]

$$\text{Yaw} = \text{atan2}(2\times(w\times z + x\times y), 1-2(z\times z))\times 57.$$

4.1.3 Path generation

The ROS navigation stack generates the global and local path for the robot to follow. The robot will be able to follow and reach the goal only if its localization in the map is accurate. We can simulate two different cases of the path generated into two different scenarios.

CASE I: Static environment

When the whole map is stationary, there is no movement of obstacles. In this case the global path that is formed will not change and the local path will follow the global path quite closely.

Fig. 7 shows the navigation in the static environment. Fig. 7 (a) shows the initial position, (b) goal point given to the bot. The robot now follows the global path. Fig. 7 (c), (d), and (e) show that the global planner forms the shortest trajectory from the initial point to the goal using the static map. While cornering sometimes it comes close to the obstacle, then the local planner deviates from the global path to avoid it. After avoiding the obstacle, the robot again returns to follow the global path, as now the local and global path coincide. The robot managed to reach the goal point every time as shown in Fig. 7 (f).

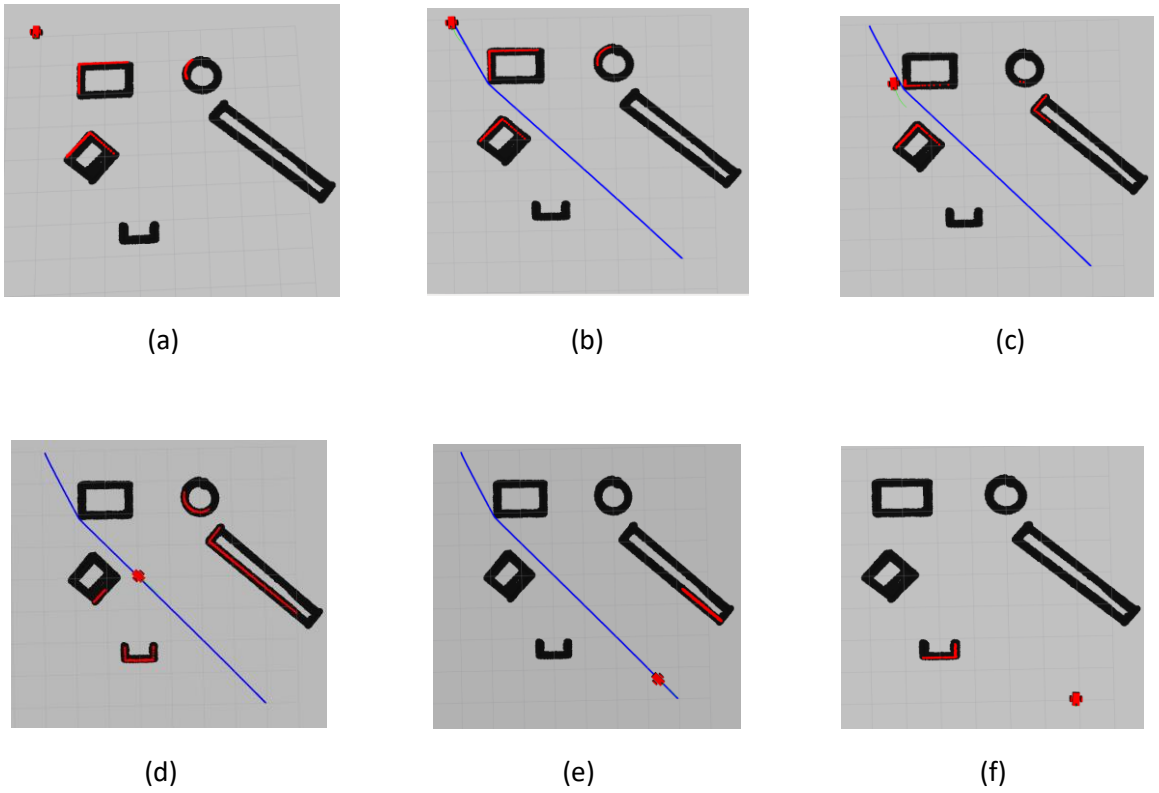


Fig. 7. Static environment

CASE II: Dynamic environment

This is the case when we suddenly introduce an obstacle in the path of the robot while the robot is moving. The aim of this experiment is to localize the robot in the environment with the changes. Simulations were carried out to see if the localization of the robot was accurate enough to guide itself from the obstacle which is not present on the global map. The result shows that the path planning algorithm now calculates an alternative global path and again the local path follows the global path as shown in Fig. 8

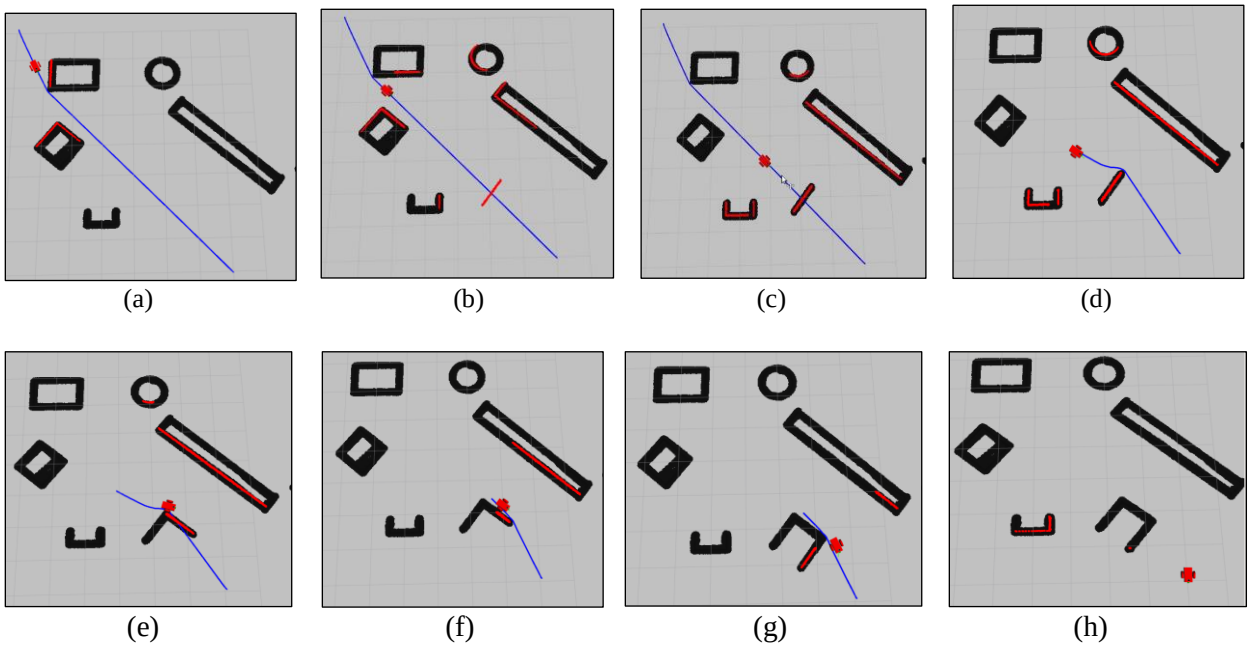


Fig. 8. Dynamic environment

In Fig.8, each plot is described as follows:

(a) Goal point given to the bot. As the global planner uses the static map to form a trajectory to the goal position.

(b) The lidar sensor detects an obstacle at some distance. Map update even as the robot moves is taking place as the area that was free for the robot will now be changed to occupied area where the robot cannot go.

(c) The new obstacle's cost has been updated in the map. The global path will now receive messages simultaneously as the map is being updated using the current local map. The global planner will now again form the shortest path to the goal position by avoiding all the obstacles. The map and the path are simultaneously being updated

(d) After receiving messages about the map update, the global path recalculates the shortest path to the goal from the current position of the robot avoiding all obstacles. The global Path has now changed from the original one.

(e) The robot now follows the new path generated. But again, the new path goes through an obstacle. So, the map and the global path are updated again to account for this change.

(f) A new Global path is generated to avoid the obstacle and simultaneously the map is also updated. The robot now follows this path.

(g,h) The bot now easily follows the path to reach its goal.

4.2 Real Life results

4.2.1 Mapping of the real-life environment

The same algorithm for mapping was used to obtain a map of the corridor of the lab hallway. This environment has a long corridor with doors to rooms on either side, bathroom space, stairway, lift area, glass doors. We started mapping from one end till the other end. Fig. 9(a) and (b) show the real time map of the environment. The same gmapping algorithm was used as before. Map Fig.9.(a) was formed using the URG-04LX-UG01 and the map Fig. 9(b) was made by UTM-30LX.

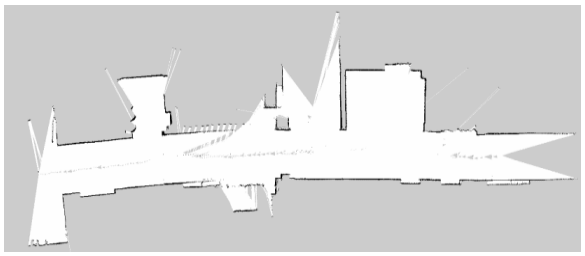


Fig. 9(a) Real time map of the corridor- Run 1

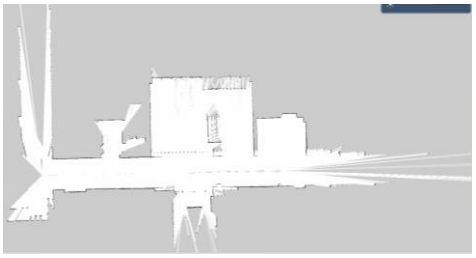


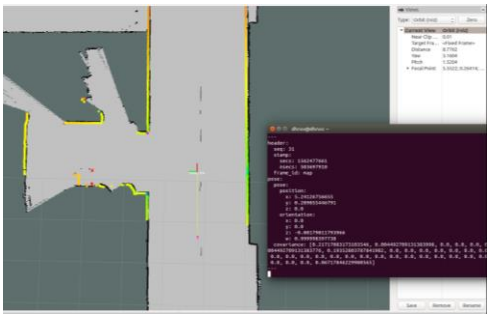
Fig. 9(b) Real time map of the corridor- Run 2

We can see that the single lidar was able to map the environment, but the corridors are not parallel in the map Fig. 9(a). As the URG-04LX-UG01 has a smaller range than the UTM one it catches less particles or less obstacles in its range thus because of the fewer number of particles in the filter while mapping the map does not come parallel and precise.

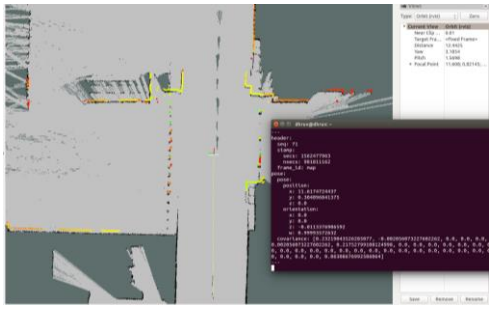
4.2.1 Localization in the real-life environment

After localizing used the map of the corridor that we made with the robot. The AMCL gave correct coordinates of the lidar.

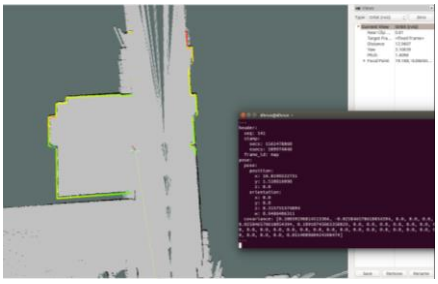
In the Fig. 11(a)(b)(c) given below we moved our lidar to small distance in the map. And it can be clearly seen that the AMCL algorithm can follow the real-life movements of the `lidar while giving the coordinates of the lidar in the map at all given times. The axes of the map frame are fixed and with reference to them our robots' axis are defined. Thus, with a little tuning in the AMCL parameters we can to a high accuracy know the coordinates of our mobile robot in any given map. Even while turning the lidar now matches its scans to the already made map thus it can predict its location in the map.



11(a). Coordinates of the robot in the map



11(b) Coordinates of the robot in the map



11(c). Coordinates of the robot in the map

5. DISCUSSIONS

The simulations conducted on the robot model were to determine if the robot was localized in the given environment. One of the key factors was the convergence of the particles in the filter to get a good and accurate sense of pose of the robot, and every time other than initial spawn of the robot the particles were converged into a small area to return the correct orientation of the robot.

To demonstrate how accurate the AMCL localization in simulation is we compared the distance travelled by the robot when we give it a goal in the map. We calculate the distance given by the pre-built odometry in the robot which gives highly accurate results and we compare it to the results calculated by the AMCL coordinates.

TABLE IX: COMPARISON BETWEEN ODOMETRY AND AMCL

Distance in meter		Absolute Error %
Odometry	AMCL	
1.804954603	1.82345566	1.02501507
2.714149382	2.709495405	0.1714709051
7.76448317	7.728373561	0.4650613347
4.63037165	4.672836185	0.9170869664
7.652827856	7.597781043	0.7193002924
Average Error %		0.6595869138

As we can see from the table above the Average Error percentage is less than 1%. The error can again be reduced if we change the parameters more but with more accuracy the demand for high processing power will increase.

During the path generation and autonomous navigation, our bot had no problems in traversing the static map. But when an obstacle was dynamically placed in its path quite near the bot it took a lot of time to calculate its next trajectory considering the new obstacle.

Also, we calculated the actual and simulated distance as given by AMCL algorithm when we were localizing it.

TABLE IX: COMPARISON BETWEEN ODOMETRY AND AMCL

Distance in meter		Absolute Error %
Real	AMCL	
5.41	5.24	3.14
13.1	11.61	11.37
21.2	18.81	11.2

As we can see from the table that initially the error is very low but as we go further in the map the error increases and remains somewhat constant. This is because until the first point the map made was very accurate. Then the section of the map leading to the second point is incorrect and not accurate and is in fact shorter than the actual length of the corridor. This is a failure of the ICP algorithm which is used to build a map. Again, as we go forward as now there are a lot of features in the map, we assume that the map is between the point 2 and 3 is very accurate but the error reciprocates. Thus, now every reading for every point will be less than the actual value as we have a map error in the beginning only.

6. CONCLUSION AND FUTURE WORK

Adaptive Monte Carlo Localization is a very effective solution for localizing the robot in a given environment. Our robot was successfully localized in the ROS gazebo and Rviz environment to an accuracy of about 0.66%. This clearly shows that the particles in the filter were successfully able to converge quickly and were able to give the exact pose and orientation of the mobile robot. Because of the effective implementation of AMCL our robot successfully was able to reach a goal in the map within a specific time interval. Moreover, the robot was able to navigate to many positions in the map while performing AMCL algorithm in real time. These algorithms along with all the ROS packages demonstrate that it is very much viable to realize autonomous navigation on mobile robots using them. Our AMCL localization performed very well in a completely static environment as well as a environment where the obstacles appeared dynamically. To move on from simulation to real life, not much changes are to be done in the AMCL algorithm, only a few parameters are to be changed (increasing or decreasing the number of particles in the filter etc.) according the computational power of the hardware device, and sensors must be setup. Then again, we can localize our robot in the real environment. We can always add more sensors to the mobile robot as with more sensors the AMCL algorithm can become more extensive. We can add wheel encoders, depth camera etc. to our robot.

In the future we could work to improve the path planning algorithm and make changes in them according to the environment to get the best path available. More than one robot localization can also be considered for the future.

7. REFRENCES

[1] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *IEEE Trans. Robotics and Automation,* vol. 23, no. 1, pp. 34-46, 2007.

[2] M. P. Imthiyas. Indoor Environment Mobile Robot Localization. *International Journal on Computer Science and Engineering*, 2(3):714–719, 2010.

[3] D. Fox, “Adapting the sample size in particle filters through KLD-sampling,” *The international Journal of robotics research,* vol. 22, no. 12, pp. 985-1003, 2003.

[4] M . Montemerlo and S. Thrun. *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics (Springer Tracts in Advanced Robotics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[5] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *International Joint Conference on Artificial Intelligence.* pp. 1151-1156,2003

[6] Pérez, J.A., Castellanos, J.A., Montiel, J.M.M., Neira, J. and Tardos, J.D., 1999. Continuous mobile robot localization: Vision vs. laser. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)* (Vol. 4, pp. 2917-2923). IEEE

[7] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A.Y., 2009, May. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).

[8] Zaman, S., Slany, W. and Steinbauer, G., 2011, April. ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues. In *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)* (pp. 1-5). IEEE

[9] https://www.roboshop.com/files/pdf/hokuyo-urg-04lx-urg01-specifications.pdf

[10] Zheng, K., Sep 2016,“. *ROS Navigation Tuning Guide*.

[11] http://wiki.ros.org/amcl

[12] http://wiki.ros.org/base_local_planner

[13] Zhang, L., Zapata, R. and Lepinay, P., 2012. Self-adaptive Monte Carlo localization for mobile robots using range finders. *Robotica*, *30*(2), pp.229-244.

[14] Lima, O. and Ventura, R., 2017, April. A case study on automatic parameter optimization of a mobile robot localization algorithm. In *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)* (pp. 43-48). IEEE.

[15] An, Z., Hao, L., Liu, Y. and Dai, L., 2016. Development of Mobile Robot SLAM Based on ROS. *International Journal of Mechanical Engineering and Robotics Research*, *5*(1), pp.47-51.

[16] Hiemstra, P. and Nederveen, A., 2007. Monte carlo localization.