# RoboND Where Am I

Amay Yadav

**Abstract**—In this project, two mobile robots are that navigate to the target location in a simulated gazebo environment. One of the mobile bots is already provided the benchmark one and the other mobile bot is built from scartch using Unified Robot Description Format (URDF). Each robot model has camera, laser rangefinder and wheels to move around. In simulation environment, Gazebo plugins were incorporated to mimic the behaviors of camera sensor, the hokuyo sensor and the differential drive controller. Standard ROS packages like AMCL (adaptive monte carlo localization) are utilized and their parameters are tuned to improve the localization results. Parameters are configured for each robot model i.e. benchmark and personal, and both robots are able to reach the goal with reasonable localization accuracy.

**Index Terms**—Robot, AMCL, ROS, URDF, Udacity, Localization.

## 1 Introduction

Robot localization is the process of determining where a mobile robot is located with respect to its environment. Localization is one of the most fundamental competencies required by an autonomous robot as the knowledge of the robot's own location is an essential precursor to making decisions about future actions. Some of the localization algorithms include variations of Kalman filters and particle filters. Localization for the robot needs to be accurate so that we do not crash into walls and we are always driving towards the goal position.

Two common types of localization are:

- Local localization is determining robot pose and orientation based on odometry and perception sensors data before any matching to the map.
- Global localization, it combines output from local localization and matches it to the known map so to place the robot in exact position and orientation on the map. In other word, it is describing the translation and rotation from odom frame to the map frame.

In this project, we are building the basic URDF model of a robot, which represents the full links geometry descriptions along with inertial, visual and connection parameters. We will also buil a ROS package to solve global localization problem in a static environment. Two different simulations are run on two different robots on the same static environment. The goal is to use standard ROS packages for navigation and localization (amcl) on the bots to reach the target destination.

## 2 Background

Localization using Kalman filters takes two things into account:

1) The robots' moves (actuators) from one state to another continuously.
2) The data from the sensors such as cameras, lidar etc.

The problem is that both sets of data can have negligible amount of error. Kalman filter works really well for small

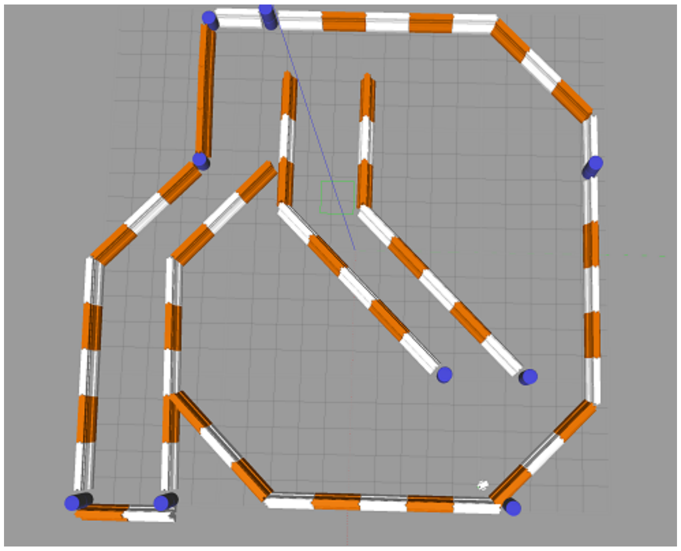


Fig. 1. Gazebo Static Map of the environment.
[1]

incremental errors that arise due to the sensors errors and motion models discrepancies (noise representation, wheel slippage etc.) like odometry estimation and local robot pose. Kalman Filtering allows us to combine the uncertainties regarding the current state of the robot (i.e. where it is located and in which direction it is looking) and the uncertainties regarding its sensor measurements and to ideally decrease the overall uncertainty of the robot. Both uncertainties are usually described by a Gaussian probability distribution, or Normal distribution. A Gaussian distribution has two parameters: mean and variance. The mean expresses, what value of the distribution has the highest probability to be true, and the variance expresses how uncertain we are regarding this mean value. However, Gaussian assumption employed by Kalman filters is restrictive for global localization problem. Hence, the Kalman filter is very well-suited for localization for this instance.

The Monte Carlo Localization (MCL) method takes a new approach to representing uncertainty in mobile robot localization: instead of describing the state space by a probability density function, we represent it by maintaining a set of samples that are randomly drawn from it. By using a sampling-based representation we obtain a localization method that has several key advantages with respect to earlier work. In contrast to Kalman filtering based techniques, it is able to represent multimodal distributions and thus can globally localize a robot. It drastically reduces the amount of memory required compared to grid-based Markov localization, and it can integrate measurements at a considerably higher frequency. It is more accurate than Markov localization with a fixed cell size, as the state represented in the samples is not discretized.

### 2.1 Kalman Filters

Kalman filter is an estimation filter which uses imperfect measurements (measurement with noise) observed over time and produces the estimates of required variables. The idea of Kalman filter is simple; given a physical model of a system, e.g., how location and velocity change over time and theres a way to measure these variables that are dependent on the state of the system. In addition, we need an initial belief of the state of the system in the form of a probability density function with a normal distribution. The Kalman filter is iterative approach wherein every step includes two phases i.e. the prediction phase and measurement update. Prediction phase is a calculation of an estimate of system state in the next time step based on previous state estimate and process noise; Its like any extrapolation applied in a mathematical model. Any model of a system is imprecise, the prediction wont be perfect. For example, when predicting a location of a car, we cant predict the impact of wind. Thats why the predicted state will have covariance larger than the previous state of the system.

Then we measure some variables that depend on the state of the system and update the estimate of the state with information from the measurement. The measurement is also inaccurate, so we need to consider the measurement noise. When we update the estimate of a state, we combine information from the prediction and the measurement. In a simple example of one variable, the update is just a multiplication of two Gaussian functions.

This filter is very prominent in control systems due to its accuracy and computational efficiency. It is good at taking in noisy measurements in real time and providing accurate predictions of the actual variables such as position.

Kalman filter algorithm is based on three assumptions:

1) The State transition probability is linear with added gaussian noise.
2) The measurement update probability is linear with added gaussian noise.
3) The initial state belief i.e. represented by the mean and covariance is normally distributed.

These assumptions limit the applications of Kalman filters because the robots in the real world will have extremely complicated measurements and they might end up having nonlinear motions, and the Kalman Filter cannot handle that. Hence, we need to use a variant form of Kalman Filter called the Extended Kalman Filter (EKF) which addresses the one major limitation of Kalman Filter i.e. it linearizes the general non-linear motion or measurement with multiple dimension using multi-dimensional Taylor Series.

### 2.2 Particle Filters

Particle filter is localization technique that uses particles that are spread across the map. Formally, a particle is a discrete guess of where a robot may be located. Particle filters use particles to represent the state of the robot. On 2D map each particle has its own x position, y position, orientation and weight based on some robot measurement made by a sensor like the RGB-D camera for example or the laser range finder.

Particle filters operate by uniformly distributing particles throughout the entire state space and then removing those particles that least likely represent the current state of the robot. Particles are compared to robots current position on each iteration. If the position of the particle on the map corresponds to the robots current position/detected landmarks, particle gets higher weight. Particles with higher weights are more likely to be chosen for the next iteration, which results in all particles merging near the robots current position. The particle filter uses Monte Carlo simulation on an even distribution of particles to determine the most likely position value. Monte Carlo Localization (MCL) is not subject to the limiting assumptions of Kalman Filters. The basic steps for MCL algorithm are listed as follows:

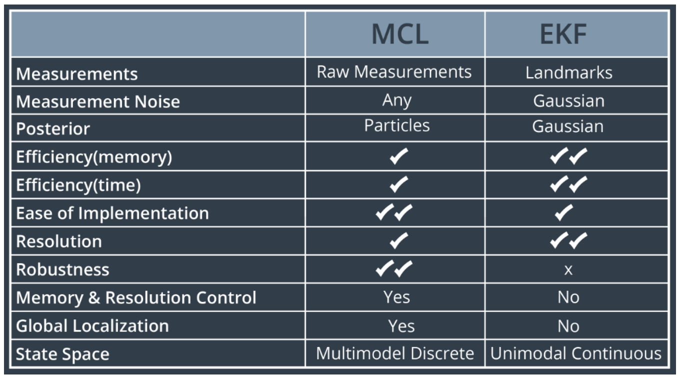
- Populate the space with random samples of where the robot might be.
- See if the random samples are consistent with sensor and movement readings.
- Keep samples that are consistent over samples that are not consistent.
- Randomly select M particles based on weights (same particle may be picked multiple times).
- Move all particles according to movement model with noise.
- Integrate sensor readings into a weight for each sample by making a prediction about the sensor readings likelihood given this particles location. Update weight on the particle accordingly.

A key problem with particle filter is maintaining the random distribution of particles throughout the state space, which goes out of hand if the problem is high dimensional. Due to these reasons it is much better to use an adaptive particle filter which converges much faster and is computationally much more efficient than a basic particle filter. In this project we have utilized the Adaptive Monte Carlo Localization (AMCL). The AMCL algortithm dynamically adjusts the number of particles over a period of time, as the robot navigates the map.

### 2.3 Comparison / Contrast

As discussed in the Kalman Filter section Extended Kalman Filter (EKF) is great for position tracking problems and works really well if the position uncertainty is small and one major advantage of the Kalman filter is the computing and memory efficiency. However, EKF performs really poorly in general global localization problems. On the other hand

Particle filter is much easier to implement. MCL its more robust as it can work with any type of measurement noise without need for workarounds or transformations. Hence, its much better suited for the Global Localization problem. The accuracy and computational costs for the MCL can be improved by setting the size of particles[1].



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

Fig. 2. MCL vs EKF Tabular comparison.
[1]

Based on all the stated reasons particle filter has been chosen for this project. Also, particle filter can solve the kidnapped robot problem by adding random particles in the particle set.

## 3 Simulations

A ROS package is created and configured to use other standard Ros packages to perform 2D global localization and navigation in a simulated Gazebo environment. Two different robot models, a benchmark model and a personal model are specified in the Unified Robot Description Format (URDF). RViz is used to visualize the ROS topics and the also to visualize the pose of the robot by displaying the PoseArray which is displayed in the form of arrows in Rviz and it can been seen in images below. PoseArray depicts a certain number of particles.

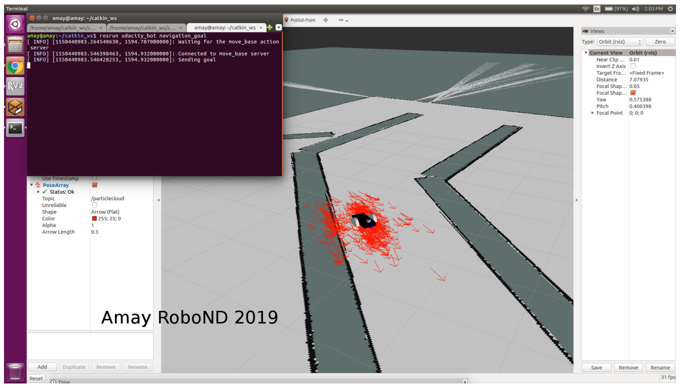
### 3.1 Achievements

In this project we have achieved localization for both the benchmark and the personal bot. We have successfully demonstrated the AMCL and the images from Rviz shows green arrows that represent particles created by the amcl package. The results of this project are in the result section and the discussion of those results is in the discussion section. We have only really created the urdf model for the personal model and all of the other components already existed in ROS and its truly spectacular how well the libraries and packages fucntioned. They are most certainly worth the study and we can expand on them for more complex problems.

### 3.2 Benchmark Model

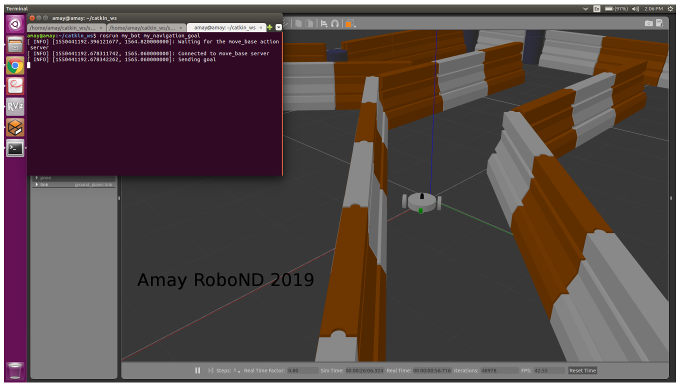
#### *3.2.1 Model design*

This robot design has already been provided. The robot model is designed using URDF files which defines the shape of the robot. The benchmark model bot has a square chassis



```
amay@amay:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1550440983.264540630, 1594.707000000]: Waiting for the move_base action
 server
[ INFO] [1550440983.546398463, 1594.932000000]: Connected to move_base server
[ INFO] [1550440983.546428253, 1594.932000000]: Sending goal
```

Fig. 3. Benchmark Model at Start position.



```
amay@amay:~/catkin_ws$ rosrun my_bot my_navigation_goal
[ INFO] [1550441192.396121677, 1564.820000000]: Waiting for the move_base action
 server
[ INFO] [1550441192.678311742, 1565.060000000]: Connected to move_base server
[ INFO] [1550441192.678342262, 1565.060000000]: Sending goal
```

Fig. 4. Personal Model at Start position.

and it consists of two actuators(wheels), caster, camera and Hokuyo sensor(laser sensor). Chassis has the shape of a box with the Hokuyo sensor on the top and the camera on the front side.

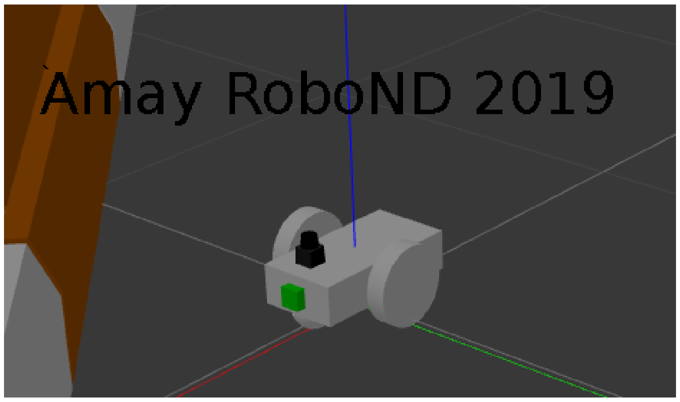


Fig. 5. Benchmark Model Robot.

#### *3.2.2 Packages Used*

The main packages used for this project are move_base, amcl, map_server, differential drive controller, camera and Hokuyo controller. There functioning is listed below

- Differential drive controller was used to simulate the bots' movement by turning left and right wheels. Differential drive controller publishes odom_topic and receives cmd_vel topic.

- Camera being a sensor was used to take pictures of the environment. Camera node published to the image_raw and camera_info topics.
- Hokuyo sensor was used for the laser scan of the environment. Hokuyo controller publishes scan_topic.
- amcl (Adaptive Monte Carlo Localization package) package is used for the localization was. This package receives the scan_topic from Hokuyo controller, the odom_topic from differential drive controller and the map_topic from the map_server.
- move_base package was used for path planning and robot movement. It receives an estimation from the amcl package on a robots position and combines it with the map_topic to perform global and local planning. Then it publishes cmd_vel topic to differential drive controller according to a planned path.

#### *3.2.3 Parameters*

The initial pose parameters for x, y and yaw are all set to 0.

Adaptive Monte Carlo localization (amcl) algorithm uses variable number of particles depending on certainty which can save on computational power as it can scale up or down base on certainty. As the task is simple and the static map doesn't exactly have a lot of obstacle a small number of particles is sufficient. Hence, the minimum number of particles is set to 100 and it works well and maximum number of particles is set to 1000.

The Odometry model parameters are as follows:

1) odom_alpha1 : Specifies the expected noise in odometry's rotation estimate from the rotational component of the robot's motion.
2) odom_alpha2 : Specifies the expected noise in odometry's rotation estimate from translational component of the robot's motion.
3) odom_alpha3 : Specifies the expected noise in odometry's translation estimate from the translational component of the robot's motion.
4) odom_alpha4 : Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion.

Odometry alphas 1 to 4 are the only parameters tuned since the robot uses a differential drive system. These parameters specify the noise level in the sample motion model odometry algorithm [1]. All four odom alpha parameter values are set to 0.002 to produce a lower noise and tighter bound on the location. If we keep decreasing the odom alpha parameter values is proportional to tighter bound due to decreased noise and inversely proportional to robustness. Hence, 0.002 is an optimal value in the trade-off between robustness and accuracy.

The transform_tolerance parameter is set to 0.3 seconds. This parameter helps select the durability of the transforms(s) being published for localization reasons and should account for any lag in the system being used.

The robot_radius is set to 0.25 which is approximate the size of the robot.

The update_frequency, publish_frequency and the map size for both local and global costmaps were reduced drastically to release computational resources.

The resolution is increased from 0.05 to 0.1 to reduce computational load.

The inflation_radius was set to 0.50 in costmap common params to avoid hitting the barriers in the simulation, this ensures that the robot will keep a distance of half a meter away from the barrier.

TABLE 1
Table

| Parameters | Values |
|---|---|
| initial_pos_x | 0 |
| initial_pos_y | 0 |
| initial_pos_a | 0 |
| min_particles | 100 |
| max_particles | 1000 |
| odom_alpha1 | 0.002 |
| odom_alpha2 | 0.002 |
| odom_alpha3 | 0.002 |
| odom_alpha4 | 0.002 |
| transform_tolerance | 0.3 |
| robot_radius | 0.25 |
| inflation_radius | 0.5 |
| update_frequency | 20.0 Hz |
| publish_frequency | 5.0 Hz |
| width | 5.0 |
| height | 5.0 |
| resolution | 0.1 |

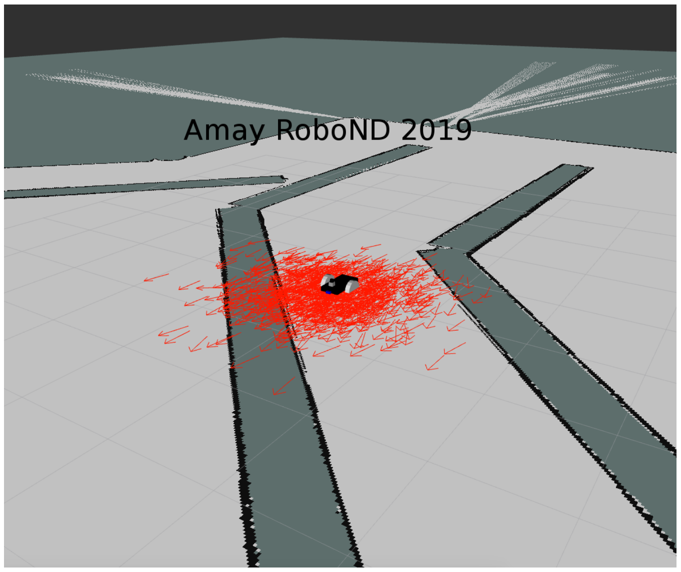


Fig. 6. Benchmark Model Rviz with PoseArray.

### 3.3 Personal Model

#### *3.3.1 Model design*

Personal Model bot has a round chassis with two wheels and one caster. The laser scanner is placed at the center of the chassis, so when the bot turns, it is still at the center. Camera is placed in the front just like in the benchmark model.

#### *3.3.2 Packages Used*

Personal model was able to operate with the same packages as the benchmark model.

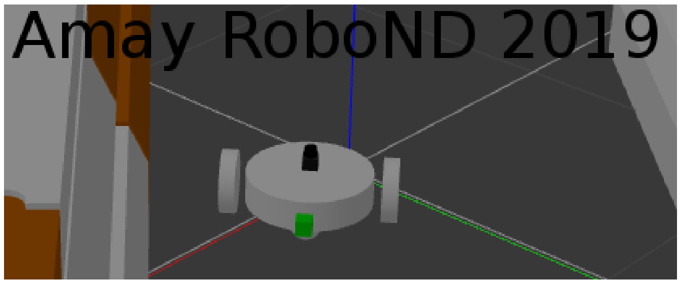


Fig. 7. Personal Model Robot.

#### 3.3.3 *Parameters*

The personal model bot is quite similar to the benchmark model bot. Hence, the parameters are all pretty much the same

The benchmark model robot got stuck sometimes at the u-turn with the same odom alpha parameter values of 0.002( used inbenchmark model). Hence, all the 4 odometry alpha parameters were increased from 0.002 to 0.01. This inceases the noise and the particles end up having a wider spread but the bot doesnt get stuck on the u-turn anymore.

The max number of particles were decreased to 300 to reduce computational cost even further.

The robot_radius and inflation_radius are changed from the benchmark model because the shape of the personal model is different

The robot_radius is set to 0.3 which is approximately the size of the robot.

The inflation_radius is set to 0.6 respectively to avoid getting into walls.

Additionally, the parameters of TrajectoryPlannerROS for base local planner are tuned. By changing max velocity to 1.0 from 0.5, the bot can reach a higher speed and thus finish the task quicker. Additionally, the pdist parameter in base local planner was also set to 0.5 so that the robot will follow the global path, but it will give the robot enough freedom to recover from any mistakes.

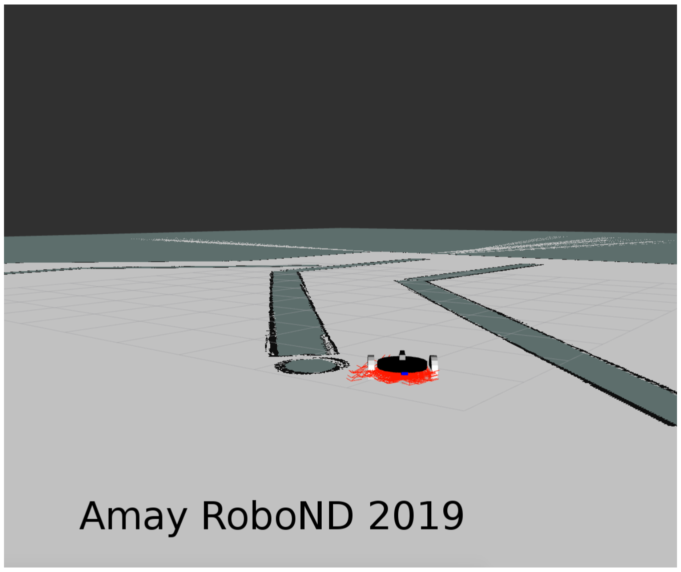


Fig. 8. Personal Model Rviz with PoseArray.

TABLE 2
Table

| Parameters | Values |
|---|---|
| initial_pos_x | 0 |
| initial_pos_y | 0 |
| initial_pos_a | 0 |
| min_particles | 100 |
| max_particles | 300 |
| odom_alpha1 | 0.001 |
| odom_alpha2 | 0.001 |
| odom_alpha3 | 0.001 |
| odom_alpha4 | 0.001 |
| transform_tolerance | 0.3 |
| robot_radius | 0.3 |
| inflation_radius | 0.6 |
| update_frequency | 20.0 Hz |
| publish_frequency | 10.0 Hz |
| width | 5.0 |
| height | 5.0 |
| resolution | 0.1 |
| holonomic_robot | false |
| meter_scoring | true |
| pdist_scale | 0.5 |
| gdist_scale | 1.0 |
| max_vel_x | 1.0 |
| min_vel_x | 0.1 |
| max_vel_theta | 2.0 |
| acc_lim_theta | 5.0 |
| acc_lim_x | 5.0 |
| acc_lim_y | 5.0 |
| controller_frequency | 15.0 |

## 4 Results

Both robots i.e. the benchmark and the personal model were able to reach the target location. The localization results look reasonable. For both cases, robots particles were uniformly spread at the start of the simulation and eventually narrowed down to a small group of particles as the robots proceeded to the goal. At the goal, the particles had a tight group pointing towards the orientation of the goal.

### 4.1 Localization Results

#### 4.1.1 *Benchmark*

Benchmark model is able to reach the target location in 18 seconds. The robot got stuck in the U-turn in one of the simulation and it needed a restart. Once restarted the robot was able to take the U-turn slowly but smoothly. The robot also slows down right before it reaches the target position and then it turns very slightly and then moves slowly.

#### 4.1.2 *Student*

Personal model takes 20 seconds to reach the target location. There are no problems in the U-turn at all. It is able to reach the goal but at the very end the robot spins around as if it correctly find the target and then move towards the target location. Hence, at the very end is when it loses time. The path taken is almost identical to the benchmark robot.

### 4.2 Technical Comparison

The main differences between the benchmark model and the personal model are the chassis shape and the Hokuyo laser sensor position. Because the Hokuyo sensor positioned

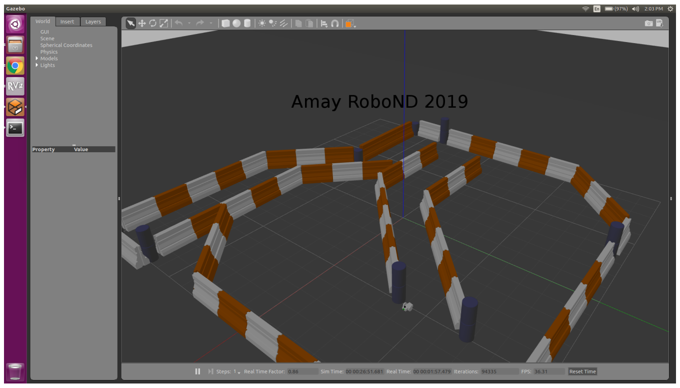
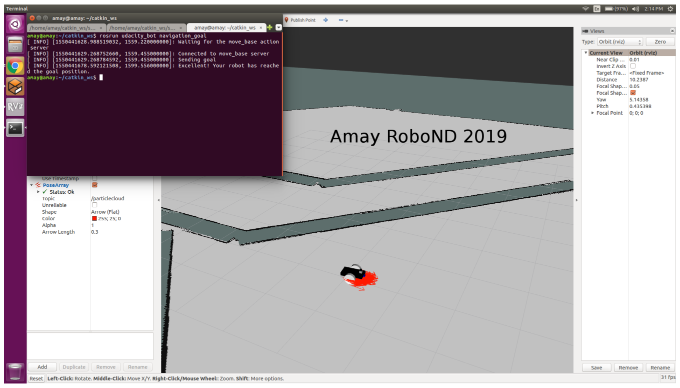


Fig. 9. Benchmark Model at U-Turn.



```
amay@amay:~/catkin_ws$ rosrun udacity_bot navigation_goal
[ INFO] [1550441628.988519032, 1559.220000000]: Waiting for the move_base action
 server
[ INFO] [1550441629.268752660, 1559.455000000]: Connected to move_base server
[ INFO] [1550441629.268784592, 1559.455000000]: Sending goal
[ INFO] [1550441678.592121508, 1599.556000000]: Excellent! Your robot has reache
d the goal position.
amay@amay:~/catkin_ws$
```

Fig. 10. Benchmark Model at Goal Position.

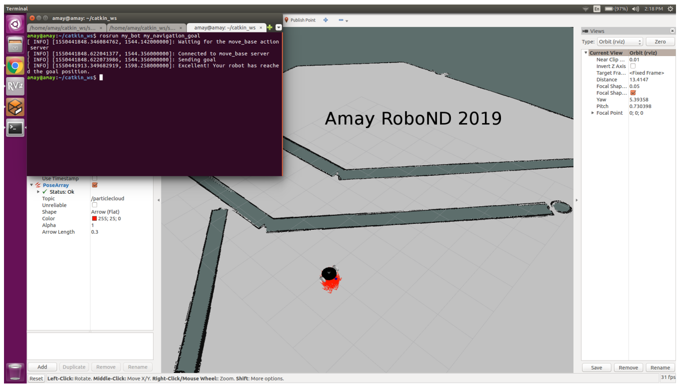
in the middle of the personal model, laser readings are slightly worse. This results in slightly worse Monte Carlo Localization at the very end when the benchmark model is near the target position . The benchmark model robot is still able to reach the goal location.

## 5 Discussion

This is the only section of the report where you may include your opinion. However, make sure your opinion is based on facts. If your robot performed poorly, make mention of what may be the underlying issues. If the robot runs well, which aspects contribute to that? Again, avoid writing in the first person (i.e. Do not use words like ”I” or ”me”). If you really find yourself struggling to avoid the word ”I” or ”me”; sometimes, this can be avoid with the use of the word one. As an example: instead of : ”I think the robot cannot localize itself because the sensor does not provide enough information for localization” try: ”one may believe the localization performance is poor because the sensor layout is not able to provide enough information for localization”. They say the same thing, but the second avoids the first person.

### 5.1 Topics

- Which robot performed better?
  The benchmark model bot performed better
- Why it performed better? (opinion)
  The personal model had the Hokuyo laser sensor in the middle of the bot and in the very end it got into a self-correcting mode where the bot spun around a little to reach to the target.
- How would you approach the ’Kidnapped Robot’ problem?
  The adaptive nature of AMCL would enable the algorithm to function for the kidnapped robot problem. This problem requires not only that the robot localize itself from an unknown initial position, but also that the robot be able to adapt to unexpected relocation during its operation. AMCL can be configured to re-adjust the number of particles in response to uncertainty, such that poor correlation with measurements causes new particles to be created with random pose. These new particles represent new hypotheses that can then guide the filters distribution toward the new location.
- What types of scenario could localization be performed?
  Localization needs to be performed in any autonomous mobile bot to get a sense of its location in respect to its surrounding environment.
- Where would you use MCL/AMCL in an industry domain? In the industry domain the MCL/AMCL algorithm will be well suited in industries for warehouses where there is not lot a randomness and the number of particles are low. These kind of environment can easily mapped to robot to use for navigation and localization purposes.



```
amay@amay:~/catkin_ws$ rosrun my_bot my_navigation_goal
[ INFO] [1550441848.346084762, 1544.142000000]: Waiting for the move_base action
 server
[ INFO] [1550441848.622041377, 1544.356000000]: Connected to move_base server
[ INFO] [1550441848.622073986, 1544.356000000]: Sending goal
[ INFO] [1550441913.349682919, 1598.258000000]: Excellent! Your robot has reache
d the goal position.
amay@amay:~/catkin_ws$
```

Fig. 11. Personal Model at Goal Position.

## 6 Conclusion / Future work

For this project we two robots were designed using URDF to complete a navigation task with ROS navigation packages after the parameters have been tuned. The AMCL algorithm was used where particles represented robot states or pose. The dynamics of the group of particles was updated step by step and re-sampled for localizing the robots.

Possible improvements would be to tune the parameters further to improve the the robot localization problem and test it on more complex simulated environments. We could create another robot model probably with more sensors and better shape etc. If the robot is implemented in real hardware, the noise level will be higher and the motion model has to be more robust for it to work. More particles are needed to localize the robot which requires more computational resources like a powerful CPU and maybe coupled with a GPU. Parameters may need to be tuned again for real hardware and to account for real world variables like friction or dynamic environment.

### 6.1 Modifications for Improvement

Some of the possible improvements could be that the robot can have additional sensors; maybe we could somehow capture a roundabout view of the environment that would make them more reliable and enable AMCL algorithm to work with more information. This should should improve localization for the u-turn most certainly. We could change the sensor location to be on the front and back and on the sides which would allow the robot to move better in different directions currently we can't execute a reverse turn easily if we get trapped in a location.

### 6.2 Hardware Deployment

Since we are able to deploy this code on our machines we don't need heavy processing power. This code can be easily deployed on Nvidia Jetson TX2. We would need to build out the robot with all the actuators (wheels) and sensors (camera and Hokuyo sensor). Topics published by Gazebo should be replaced with topics published by robots sensors.

Since we need to keep the battery consumption for the bot to a low we would have to decrease the number of particles used by AMCL. We can sacrifice accuracy and reliability for more output from the bot if we don't mind the bot bumping into walls in an enclosed space . This will help the bot run longer on the limited battery power it has. Also hardware is a one time cost but batteries need to be recharged and then replaced.

## REFERENCES

[1] U. lecturers, *Udacity RoboND Lectures*. Udacity, 2019.