Adaptive Monte Carlo Localization in ROS

Deepak Trivedi

Abstract—Adaptive Monte Carlo Localization (AMCL) is demonstrated two robot models using the Robot Operating System (ROS) platform. The technique is demonstrated in simulation on a baseline robot, and larger scale robot. Both robots utilize wheeled locomotion, and use a Hukoyo laser scanner for detecting obstacles. A camera is also mounted for observation, but is not used as input for localization. Simulations are performed in the Gazebo environment. The effect of many parameters in AMCL is studied in order to arrive at robust and efficient localization. Finally, possible future work is recommended.

Index Terms—Robot, Localization, Particle filter, Kalman filter

1 Introduction

Autonomous navigation is an important aspect of all mobile robots and devices. This includes mapping, path planning, localization, and obstacle avoidance. Robot localization denotes the robot's ability to establish its own position and orientation within the frame of reference [1]. This is achieved through the use of sensors such as laser scanners, cameras, inertial measurement units (IMUs), and other sensors for distance, velocity and acceleration measurement. However, all these sensors introduce noise. Additionally, we often have physics-based models for robot kinematics and dynamics. These models are often imperfect, and do not model all of the physics, such as wind gust, wheel slip, geometric tolerances, etc. Also, there are inherent uncertainties in parameters that are used to evaluate these models, such as friction coefficients. Moreover, initial conditions are also not perfectly known. Therefore, relying solely on these models introduces biases in state estimation that grows with time. Both pure model-based and pure sensor-based approaches are impractical for effective localization. One of the first effective algorithms for localization was the Kalman Filter [2]. The papers establishing the mathematical foundations of Kalman type filters were published between 1959 and 1961 [3]. Kalman filters have been effectively used in the navigation systems of nuclear ballistic missile submarines, cruise missiles, and guidance systems of various spacecraft and launch vehicles. An alternative heuristic-like approach is the use of particle filters. Both these approaches will be explained in the following sections.

The Robot Operating System (ROS) platform provides a library for Adaptive Monte Carlo Localization (AMCL). Gazebo is an open source robotics simulator that allows realistic simulation of robots that could be controlled via ROS. In this project, AMCL is used to localize robots and guide them to a navigation goal. The technique is demonstrated first on a baseline robot, and then on a scaled up robot. Both robots utilize wheeled locomotion, and use a Hukoyo laser scanner for detecting obstacles. A camera is also mounted for observation, but is not used as input for localization. The effect of many parameters in AMCL is studied in order to arrive at robust and efficient localization.

Fig. 1. Rudolf Kalman 1930-2016 [4], of Kalman filter fame.

2 Background

Localization from the use of noisy sensors and models is a part of a class of problems known as the filtering problem. The filtering problem is a mathematical model for a number of state estimation problems in signal processing and related fields. The general idea is to establish a ”best estimate” for the true value of some system from an incomplete, potentially noisy set of observations on that system [5]. Two of the most commonly used techniques for localization are Kalman filters and particle filters. While both of these are Recursive Bayesian Estimators, these use different approaches for estimation. The Kalman filter uses a dynamical system model and sensor observations to estimate current state from previous states by propagating the state and uncertainties forward in time, with the assumption of Gaussian distribution. Particle filter uses random sampling to generate different system states and then assign high weights to those states that are supported by observation.

Particles with low weights are discarded and new random samples are generated instead. Both approaches have their advantages and drawbacks. Kalman filters are much more computationally efficient compared to particle filters (with a large number of particles.) However, Kalman filters have several disadvantages. The basic Kalman filter assumes that both the system model and the observation model (sensor physics) are linear and the noise is Gaussian distributed. The limitation for the model to be linear is overcome by the extended Kalman filter. Particle filters do not have any limitations regarding the nature of the dynamical system or the noise. An additional benefit is that the computational cost of the model is tunable by tuning the number of particles. The model could be made a lot faster and memory intensive (possibly at cost of accuracy), simply by reducing the number of particles used.

In the current project, due to the above mentioned reasons such as less restrictive model requirements and computational flexibility, a particle filter-based (AMCL) was chosen for localizing the robot, [6]

2.1 Kalman Filters

The Kalman filter algorithm has two steps: the prediction step and the measurement step. In the prediction step, estimates of the current state variables and their uncertainties are produced. Once a noisy measurement is performed by sensors, these estimates are updated using a weighted average. A higher weight is given to estimates with higher certainty. The algorithm marches in time recursively. This is illustrated in Figure 2.

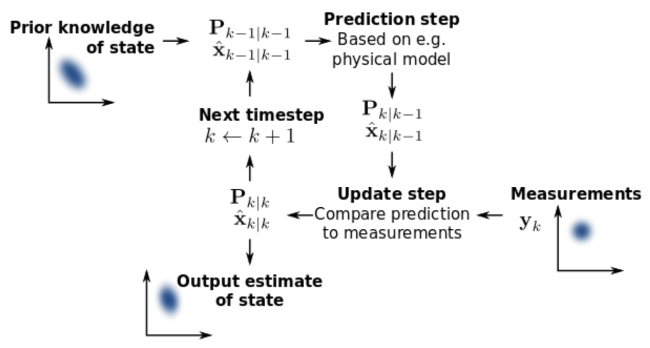


Fig. 2. Basic concept of Kalman filtering

The key assumption made in the basic Kalman filter algorithm is that the dynamical system and the measurement process are both linear, and the noise distribution is Gaussian. The extended Kalman filter (EKF) is the nonlinear version of the Kalman filter. The EKF works by simply linearizing the system and measurement about an estimate of the current mean and covariance, and marching in time. This is effectively done using the Taylor series. EKF methods of higher order could be generated simply by retaining more terms of the Taylor series. The EKF is extensively used today in GPS and other navigation systems. However, it has a few disadvantages. Unlike the basic Kalman filter, the EKF is not necessarily an optimal estimator. Also, some issues can emerge due to linearization. If the initial estimate is wrong, the filter could quickly diverge due to linearization about the wrong point.

2.2 Particle Filters

The term ”particle filters” was first coined in 1996 by Del Moral [7] in reference to mean field interacting particle methods used in fluid mechanics since the beginning of the 1960s.

Particle filters solve the filtering problem through a genetic type mutation-selection approach. Samples from the distribution (called particles) are generated. Each particle has a likelihood weight assigned to it that represents the probability of that particle being sampled from the probability density function. In the resampling step, the particles with negligible weights are replaced by new particles in the proximity of the particles with higher weights. Over a few iterations, particles converge to the actual state of the system.

Monte Carlo localization is a localization technique that uses particle filters. In this approach, the particles 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, 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. [8]

The key benefit of particle filters is that it does not require Gaussian distribution of noise, nor does it require the physics to be linear. Additionally, the number of particles provide a knob that could be adjusted to trade accuracy with speed. This is because the particle filter’s time complexity is linear with respect to the number of particles, while accuracy also increases with number of particles. Since the number of particles is such a key parameter, it is important to optimize it. This is the objective of Adaptive Monte Carlo Localization (AMCL), which uses the KullbackLeibler divergence (KLD) sampling. The idea behind this sampling is that in the initial timestep, it is necessary to use a large number of particles due to the need to cover the entire map with a uniformly random distribution of particles. However, when the particles have converged around the same location, maintaining such a large sample size is computationally wasteful. Therefore, the number of particles is reduced as uncertainty decreases. [9].

2.3 Comparison / Contrast

As evident from the previous discussion, each of the two approaches, i.e., the Kalman filter and the particle filter, has its own benefits and disadvantages. These are summarized below.

Advantages of Kalman filters over Particle filters

- Computationally efficient (fast)
- Low memory requirement

Advantages of Particle filters over Kalman filters

- Does not require Gaussian distribution of noise
- Does not require model and measurement physics to be linear
- Computational complexity is tunable as function of number of particles

- Robot physics does not need to be well known.

For the purpose of this work, particle filters are found more suitable and will therefore further discussion will focus on particle filters.

3 Simulations

This section discusses the performance of robots in simulation with particle-filter based adaptive Monte Carlo localization (AMCL.) First, the robot model design is described. Then, the simulation environment is described, including the packages used. This is followed by a discussion of the parameters chosen for the robot to properly localize itself. Details for both the benchmark robot, and the modified robot are provided.

3.1 Benchmark and Modified Model

3.1.1 Model design

Figure 3 shows a visual illustration of both (A) the benchmark model and (B) the modified model. Not visible in the illustration are two caster wheels mounted on the forward and distal locations on the robot chassis, to provide at least a three-point contact between the ground nad the robot. The two models are similar in their kinematics and primarily differ in scale. This difference makes them differ in their dynamics. Additionally, larger scale also limits the maneuverability and the reachable workspace of the modified robot. In addition to standard odometry, the robot includes a top-mounted Hokuyo laser scanner and a front-mounted camera. The laser scanner should be mounted high enough such that the robot geometry is not obstructing its view. Table 1 lists the geometric and inertial parameters of the benchmark robot model.

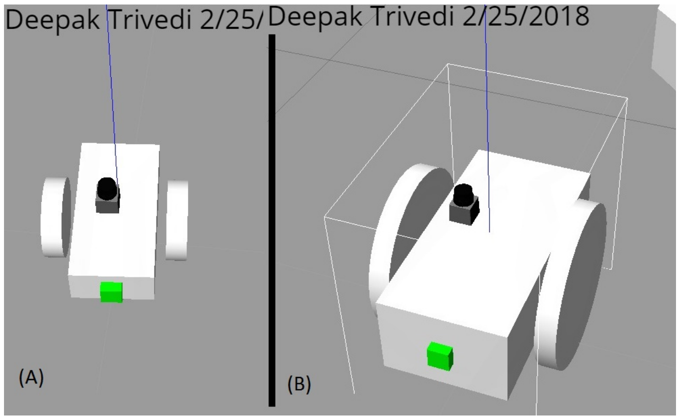


Fig. 3. (A) Benchmark and (B) modified robot models

3.1.2 Packages Used

The robots are operated via ROS and simulated in the Gazebo environment. The move_base ROS package is used for trajectory planning. This package generates a cost map based on the available obstacle information, and then does trajectory planning based on the map of obstacles, and localized current position and the goal. Based on the planned trajectory, move_base publishes the cmd_vel topic, which is subscribed by the Gazebo package. The AMCL package is used to perform Adaptive Monte Carlo Localization. AMCL takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates. On startup, AMCL initializes its particle filter according to the parameters provided. [10]. AMCL is subscribed to the following topics: scan, tf, initialpose and optionally, map. It publishes amcl_pose, particlecloud, and tf.

A full description of the topics and services is presented in 4.

TABLE 1
Comparison of Benchmark robot model and modified robot model

Parameter	Benchmark	Modified
Chassis dimension	0.4 x 0.2 x 0.1	0.6 x 0.3 x 0.2
Chassis mass	15.0	20.0
Chassis inertias	0.1	0.15
Caster wheel radius	0.0495	0.0995
Camera location	(0.2,0,0)	(0.3,0,0)
Laser scanner location	(0.15,0,0.15)	(0.15,0,0.15)
Wheel radius	0.1	0.2
Wheel mass	5.0	7.0
Wheel inertia	0.1	0.15

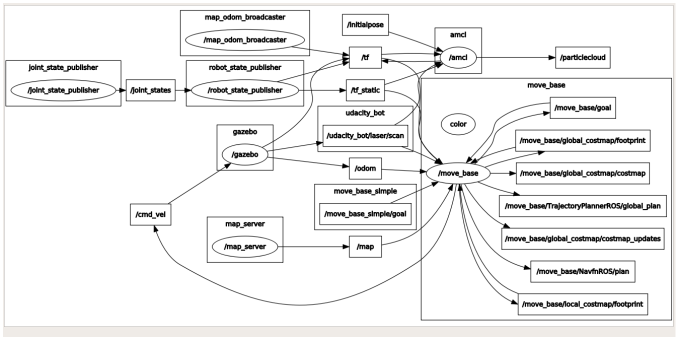


Fig. 4. Layout of the ROS services and topics for the simulation

3.1.3 Parameters

A full description of all tunable parameters in AMCL is presented in [10]. In this subsection, a few localization parameters in the AMCL node relevant to this effort, as well as the move_base parameters in the configuration file are described. Suboptimal specification of these parameters significantly affect the performance of the localization algorithm. As an example, Figures 5 and 6 show poor robot localization leading to large uncertainty in robot position and orientation.

A comparative summary of these parameters for the model is presented in Table 2.

A brief description of key parameters is provided here.

The costmap is essentially an occupancy grid that stores and updates the location of obstacles in the environment. The update and publish frequencies of local and global costmaps determines how often these costmaps are updated and published. The width, height and resolution of these costmap determines the size of the costmap. Too high a resolution or updating too frequently not only leads to a computationally inefficient localization implementation, but could also lead to poor results if the robot is not able to get

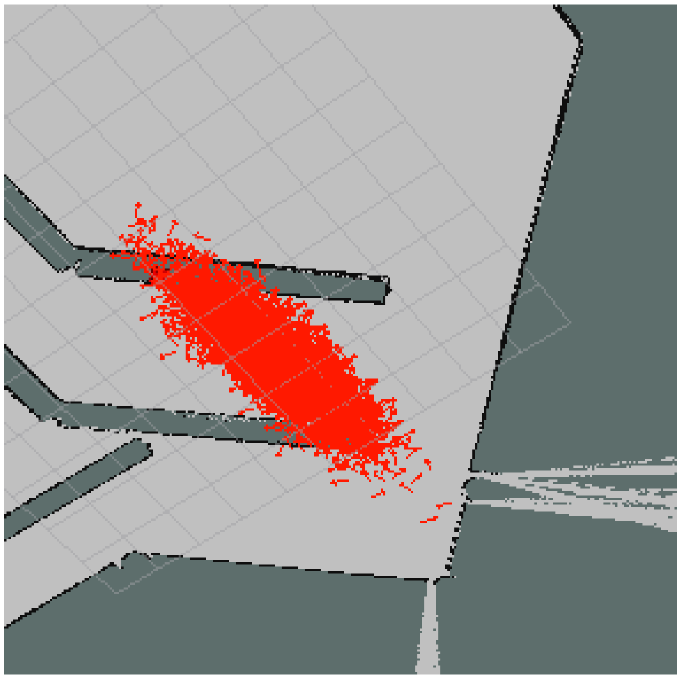


Fig. 5. An example of a particle filter with mistuned parameters

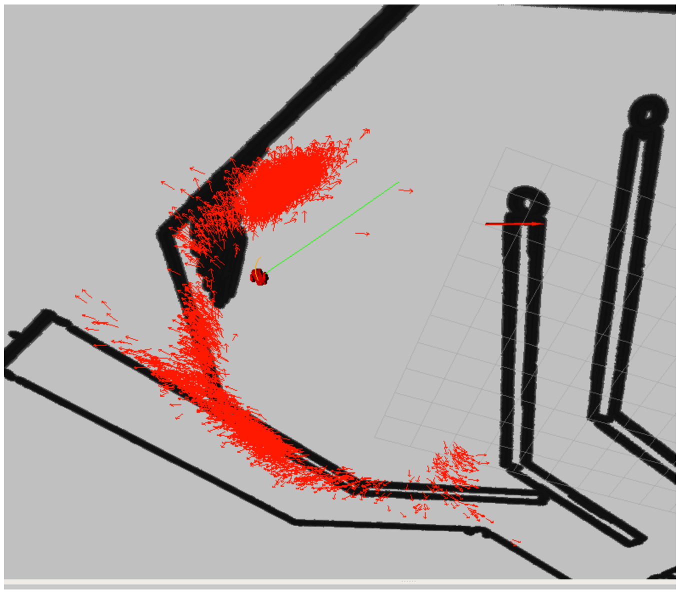


Fig. 6. An example of a particle filter unable to localize near a wall

updates when expected. Choosing too coarse a resolution or too infrequent updates may cause the robot to hit obstacles.

The obstacle_range is the maximum range in meters at which to insert obstacles into the costmap using sensor data. The robot_radius specifies the footprint of the robot. This is important for obstacle avoidance during path planning. Technically, this parameter should only be used for circular robots, all others should use the ”footprint” parameter. However, if the workspace is not too tight, using robot_radius with a radius that inscribes the entire robot footprint is equally effective. The inflation_radius is the radius in meters to which the map inflates obstacle cost values. This inflation is done at the cost of navigable

TABLE 2
Comparison of localization parameters in the AMCL node and the configuration file.

Parameter	Benchmark	Modified
Local costmap update frequency	10.0	10.0
Local costmap publish frequency	10.0	10.0
Local costmap resolution	0.15	0.15
Local costmap transform tolerance	0.2	0.2
Obstacle range	1.5	1.5
Robot radius	0.2	0.3
Inflation radius	0.2	0.25
Global costmap update frequency	5.0	5.0
Global costmap publish frequency	5.0	5.0
Global costmap width and height	40.0	40.0
Global costmap resolution	0.5	0.5
odom_alpha_1-4	0.2	0.2
max_particles	500	500
pdist_scale	1.2	1.2
controller_frequency	25.0	25.0
transform_tolerance	0.2	0.2

workspace. Choosing a value too high will over-constrain the robot and it may not be able to find feasible trajectories to the goal. Choosing too low a value will lead to the robot colliding with obstacles due to imperfect localization. The value of inflation_radius should be at least equal to a couple standard deviations of the expected localization error.

Max_particles is the maximum number of particles allowed by AMCL. Since AMCL is an adaptive algorithm, the number of particles at any timestep will change according to the current estimate of localization error. However, in order to keep computational cost low, a limit is provided by this parameter. In the current simulation, a maximum number of 500 particles was found to be adequate for localization, although numbers as high as 5000 (default) were tested.

There are four relevant odom_alpha parameters that can be used to specify the expected noise in measurement. These are:

- odom_alpha1: expected rotation estimate noise from rotation.
- odom_alpha2: expected rotation estimate noise from translation.
- odom_alpha3: expected translation estimate noise from translation.
- odom_alpha4: expected translation estimate noise from rotation.

The controller_frequency parameter determine how often does the controller sends instructions to the hardware. Too high a number is inefficient. Too low a number can lead to instablity, or oscillating behavior where the robot keeps overshooting the target.

The transform_tolerance parameter post-dates the localization transform, making them valid slightly longer into the future. This is really useful if the update frequencies are low. If the transform_tolerance value is too small, and the controller does not update the robot, the robot may have an unstable, jerky motion, since there will be periods of time when the robot does not have any valid instructions to carry out. A value of 0.2 was found to work well in the current simulations.

There are a number of trajectory scoring parameters that evaluate the quality of the planned trajectory. The pdist_scale is one such parameter. It determines how much the controller should stay close to the path it was given. A large pdist_scale weight will encourage the robot to stay close to the assigned trajectory. However, it may come at the cost of other metrics such as avoiding obstacles. The default value for this parameter is 0.6. However, in the current simulations a value of 1.2 was found to work well.

4 Results

4.1 Localization Results

Once the parameters set above were tuned, AMCL was able to effectively localize the robots and enabled them to move to their goal positions. Complete planned trajectories for both the benchmark robot and the modified robot have been recorded and are available at [11] and [12] respectively. It can be seen in the video that in both cases, the initial few timesteps have a relatively large uncertainty in the robot position and orientation. However, in a few timesteps, the uncertainty drops significantly. A third example is provided [13] as contrast, for a case where the uncertainty does not drop over time due to mistuned parameters.

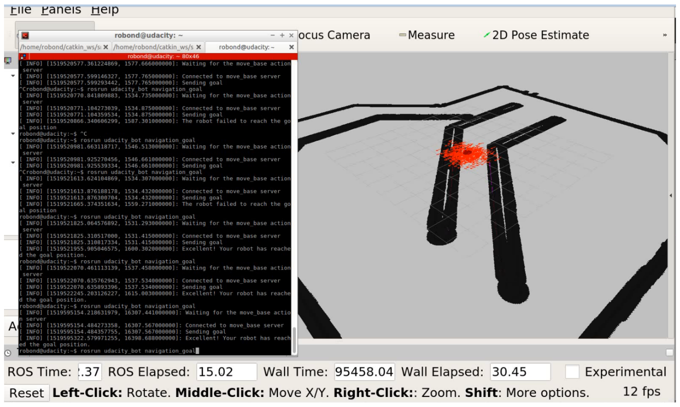
Next, screenshots at the initial state and the end state of the robot are provided. Figure 7 shows the initial state of the benchmark robot, while Figure 8 shows the final state. Likewise, Figure 9 shows the initial state of the modified robot, and Figure 10.

Following are observations that could be made:

- There is large uncertainty in the state of the robot at the initial few time steps, This uncertainty drops quickly after a few time steps.
- Once the parameters were tuned, the robot was always able to reach the goal. This was robust to the initial position. Some cases were run where the initial position of the robot was different from the default. The algorithm performance was unaffected.
- The trajectories planned may not be optimal. It appears that shorter/quicker solutions might sometimes be available.
- In a few runs, the robot diverges significantly from the planned trajectory before joining it back. However, it appears to be reasonably robust in terms of ultimately being able to merge with the planned global trajectory.
- It is possible for the robot to sometimes get stuck. The robot is not comprehensively robust. Further tuning of parameters could alleviate some of these issues.
- The robot takes about 70 "ROS Elapsed" seconds to reach the goal. It is able to stay very close to the planned trajectory without oscillating, and is able to go straight at a uniform speed. This was not the case in the initial trials. One modification that helped was slightly decreasing the size of the caster wheels, in order to not kinematically overconstrain the robot.

4.2 Technical Comparison

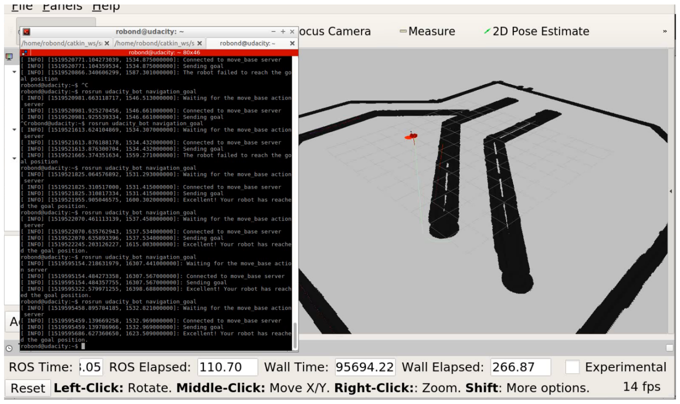
As can be seen in the videos [11] and [12], the performance of the two robot, with slightly modified localization parameters, as documented in Table 2 is comparable.



```
[ INFO] [1519520577.361224869, 1577.666000000]: Waiting for the move_base action
 server
[ INFO] [1519520577.599146327, 1577.765000000]: Connected to move_base server
[ INFO] [1519520577.599293442, 1577.765000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519520770.841809883, 1534.735000000]: Waiting for the move_base action
 server
[ INFO] [1519520771.104273039, 1534.875000000]: Connected to move_base server
[ INFO] [1519520771.104359534, 1534.875000000]: Sending goal
[ INFO] [1519520866.340606299, 1587.301000000]: The robot failed to reach the go
al position
robond@udacity:~$ ^C
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519520981.663118717, 1546.513000000]: Waiting for the move_base action
 server
[ INFO] [1519520981.925270456, 1546.661000000]: Connected to move_base server
[ INFO] [1519520981.925539334, 1546.661000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519521613.624104869, 1534.307000000]: Waiting for the move_base action
 server
[ INFO] [1519521613.876188178, 1534.432000000]: Connected to move_base server
[ INFO] [1519521613.876300704, 1534.432000000]: Sending goal
[ INFO] [1519521665.374351634, 1559.271000000]: The robot failed to reach the go
al position
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519521825.064576892, 1531.293000000]: Waiting for the move_base action
 server
[ INFO] [1519521825.310517000, 1531.415000000]: Connected to move_base server
[ INFO] [1519521825.310817334, 1531.415000000]: Sending goal
[ INFO] [1519521955.905046575, 1600.302000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519522070.461113139, 1537.458000000]: Waiting for the move_base action
 server
[ INFO] [1519522070.635762943, 1537.534000000]: Connected to move_base server
[ INFO] [1519522070.635893396, 1537.534000000]: Sending goal
[ INFO] [1519522245.203126227, 1615.003000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519595154.218631979, 16307.441000000]: Waiting for the move_base actio
n server
[ INFO] [1519595154.484273358, 16307.567000000]: Connected to move_base server
[ INFO] [1519595154.484357755, 16307.567000000]: Sending goal
[ INFO] [1519595322.579971255, 16398.688000000]: Excellent! Your robot has reach
ed the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal

ROS Time: .37  ROS Elapsed: 15.02  Wall Time: 95458.04  Wall Elapsed: 30.45
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.
```

Fig. 7. Initial state of the benchmark robot



```
[ INFO] [1519520771.104273039, 1534.875000000]: Connected to move_base server
[ INFO] [1519520771.104359534, 1534.875000000]: Sending goal
[ INFO] [1519520866.340606299, 1587.301000000]: The robot failed to reach the go
al position
robond@udacity:~$ ^C
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519520981.663118717, 1546.513000000]: Waiting for the move_base action
 server
[ INFO] [1519520981.925270456, 1546.661000000]: Connected to move_base server
[ INFO] [1519520981.925539334, 1546.661000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519521613.624104869, 1534.307000000]: Waiting for the move_base action
 server
[ INFO] [1519521613.876188178, 1534.432000000]: Connected to move_base server
[ INFO] [1519521613.876300704, 1534.432000000]: Sending goal
[ INFO] [1519521665.374351634, 1559.271000000]: The robot failed to reach the go
al position
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519521825.064576892, 1531.293000000]: Waiting for the move_base action
 server
[ INFO] [1519521825.310517000, 1531.415000000]: Connected to move_base server
[ INFO] [1519521825.310817334, 1531.415000000]: Sending goal
[ INFO] [1519521955.905046575, 1600.302000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519522070.461113139, 1537.458000000]: Waiting for the move_base action
 server
[ INFO] [1519522070.635762943, 1537.534000000]: Connected to move_base server
[ INFO] [1519522070.635893396, 1537.534000000]: Sending goal
[ INFO] [1519522245.203126227, 1615.003000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519595154.218631979, 16307.441000000]: Waiting for the move_base actio
n server
[ INFO] [1519595154.484273358, 16307.567000000]: Connected to move_base server
[ INFO] [1519595154.484357755, 16307.567000000]: Sending goal
[ INFO] [1519595322.579971255, 16398.688000000]: Excellent! Your robot has reach
ed the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519595458.895784185, 1532.821000000]: Waiting for the move_base action
 server
[ INFO] [1519595459.139669258, 1532.969000000]: Connected to move_base server
[ INFO] [1519595459.139786966, 1532.969000000]: Sending goal
[ INFO] [1519595686.627360650, 1623.509000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$ 

ROS Time: .05  ROS Elapsed: 110.70  Wall Time: 95694.22  Wall Elapsed: 266.87
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.
```

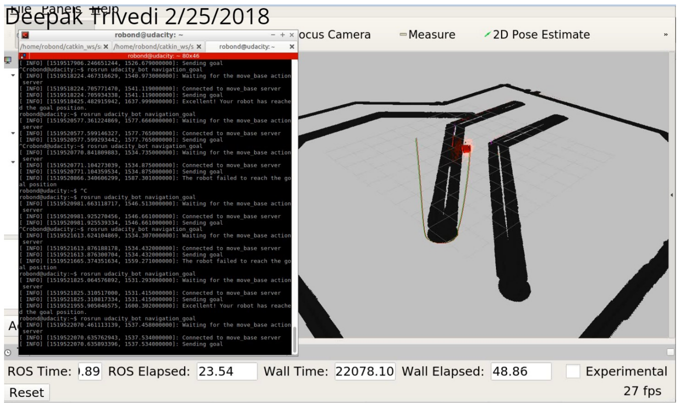
Fig. 8. Final state of benchmark robot

The benchmark robot takes about about 70 "ROS Elapsed" seconds to reach its target, while the modified robot takes about 80 "ROS Elapsed" seconds. The trajectory taken by the modified robot is slightly longer because of the large size of the robot. However, the performance of the localization algorithm is quite comparable.

5 Discussion

Several factors contributed to a successful implementation of the localization scheme on the two robots. First, the mechanical design of the robot has to be sound. Several

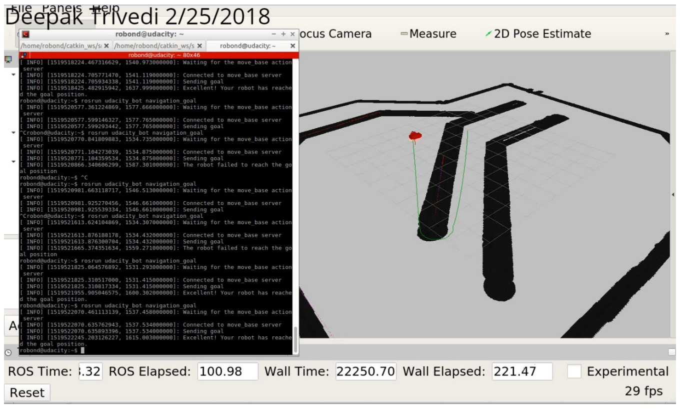
Deepak Trivedi 2/25/2018



```
[ INFO] [1519517906.246651244, 1526.679000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519518224.467316629, 1540.973000000]: Waiting for the move_base action
 server
[ INFO] [1519518224.705771470, 1541.119000000]: Connected to move_base server
[ INFO] [1519518224.705934338, 1541.119000000]: Sending goal
[ INFO] [1519518425.482915942, 1637.999000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519520577.361224869, 1577.666000000]: Waiting for the move_base action
 server
[ INFO] [1519520577.599146327, 1577.765000000]: Connected to move_base server
[ INFO] [1519520577.599293442, 1577.765000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519520770.841809883, 1534.735000000]: Waiting for the move_base action
 server
[ INFO] [1519520771.104273039, 1534.875000000]: Connected to move_base server
[ INFO] [1519520771.104359534, 1534.875000000]: Sending goal
[ INFO] [1519520866.340606299, 1587.301000000]: The robot failed to reach the go
al position
robond@udacity:~$ ^C
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519520981.663118717, 1546.513000000]: Waiting for the move_base action
 server
[ INFO] [1519520981.925270456, 1546.661000000]: Connected to move_base server
[ INFO] [1519520981.925539334, 1546.661000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519521613.624104869, 1534.307000000]: Waiting for the move_base action
 server
[ INFO] [1519521613.876188178, 1534.432000000]: Connected to move_base server
[ INFO] [1519521613.876300704, 1534.432000000]: Sending goal
[ INFO] [1519521665.374351634, 1559.271000000]: The robot failed to reach the go
al position
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519521825.064576892, 1531.293000000]: Waiting for the move_base action
 server
[ INFO] [1519521825.310517000, 1531.415000000]: Connected to move_base server
[ INFO] [1519521825.310817334, 1531.415000000]: Sending goal
[ INFO] [1519521955.905046575, 1600.302000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519522070.461113139, 1537.458000000]: Waiting for the move_base action
 server
[ INFO] [1519522070.635762943, 1537.534000000]: Connected to move_base server
[ INFO] [1519522070.635893396, 1537.534000000]: Sending goal

ROS Time: .89  ROS Elapsed: 23.54  Wall Time: 22078.10  Wall Elapsed: 48.86
```

Fig. 9. Initial state of modified robot



```
[ INFO] [1519518224.467316629, 1540.973000000]: Waiting for the move_base action
 server
[ INFO] [1519518224.705771470, 1541.119000000]: Connected to move_base server
[ INFO] [1519518224.705934338, 1541.119000000]: Sending goal
[ INFO] [1519518425.482915942, 1637.999000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519520577.361224869, 1577.666000000]: Waiting for the move_base action
 server
[ INFO] [1519520577.599146327, 1577.765000000]: Connected to move_base server
[ INFO] [1519520577.599293442, 1577.765000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519520770.841809883, 1534.735000000]: Waiting for the move_base action
 server
[ INFO] [1519520771.104273039, 1534.875000000]: Connected to move_base server
[ INFO] [1519520771.104359534, 1534.875000000]: Sending goal
[ INFO] [1519520866.340606299, 1587.301000000]: The robot failed to reach the go
al position
robond@udacity:~$ ^C
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519520981.663118717, 1546.513000000]: Waiting for the move_base action
 server
[ INFO] [1519520981.925270456, 1546.661000000]: Connected to move_base server
[ INFO] [1519520981.925539334, 1546.661000000]: Sending goal
^Crobond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519521613.624104869, 1534.307000000]: Waiting for the move_base action
 server
[ INFO] [1519521613.876188178, 1534.432000000]: Connected to move_base server
[ INFO] [1519521613.876300704, 1534.432000000]: Sending goal
[ INFO] [1519521665.374351634, 1559.271000000]: The robot failed to reach the go
al position
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519521825.064576892, 1531.293000000]: Waiting for the move_base action
 server
[ INFO] [1519521825.310517000, 1531.415000000]: Connected to move_base server
[ INFO] [1519521825.310817334, 1531.415000000]: Sending goal
[ INFO] [1519521955.905046575, 1600.302000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$ rosrun udacity_bot navigation_goal
[ INFO] [1519522070.461113139, 1537.458000000]: Waiting for the move_base action
 server
[ INFO] [1519522070.635762943, 1537.534000000]: Connected to move_base server
[ INFO] [1519522070.635893396, 1537.534000000]: Sending goal
[ INFO] [1519522245.203126227, 1615.003000000]: Excellent! Your robot has reache
d the goal position.
robond@udacity:~$
```

ROS Time: .32 ROS Elapsed: 100.98 Wall Time: 22250.70 Wall Elapsed: 221.47

Fig. 10. Final state of modified robot

hours were spent debugging the parameters of localization and path planning. However, it was found that there were issues related to the geometric dimensions of the Caster wheels. Once the issue was resolved, the rest of the tuning was simplified. Additionally, several other factors played a role, including computational complexity. Reducing the maximum number of particles from 5000 to 500 reduced computational cost and helped achieve the desired localization speed.

The benchmark robot was able to reach the goal early, since it had to take a shorter path due to its less constrained kinematics. Neither of the two robots, with their current sensors will be able to find their way out of the ”kidnapped robot” problem, since knowledge of initial position is important for both methods. Adding a GPS sensor is an example of an approach that could be used to solve the ”kidnapped robot” problem. Other than this, the particle filter approach is quite robust since it does not make any constraining assumptions about the nature of errors or the robot physics. This makes the particle filter approach very useful in industry. It is possible to find examples where this approach will not work. For example, in presence of periodic structures such as a forest with many similar trees, or a room with many similar pillars or windows, particle filters may fail to resolve the state of the robot accurately. Humans may find this understandable since it is also easy for humans to get lost in such environments. If any tags or markers could be added in such an environment, that will significantly help with localization.

6 Conclusion / Future work

Two popular approaches for robot localization are (a) the Kalman filter and (b) the particle filter. The particle filter-based AMCL approach was successfully implemented and demonstrated on two robots - a benchmark robot, and a modified robot. Videos and snapshots document the successful implementation of the approach. The simulation environment is reasonably close to hardware environment.

6.1 Modifications for Improvement

The current rover design could be improved in the following ways.

- Base Dimension: The robot could be made more stable by properly designing the base, thereby moving the center of gravity lower.
- Sensor Location: The laser scanner should not be occluded by the body of the robot. Robust design will eliminate this problem.
- Sensor Amount: Additional sensors for redundancy should help with localization.

6.2 Hardware Deployment

With a few additional steps, the filter could be deployed on hardware. For example, ROS could be set up on a Raspberry Pi, or another microcontroller. ROS could be installed on the microcontroller, and it could be used to control motor drivers and interface with laser scanner. This was performed by the author on a Raspberry Pi, with Kalman filters being used for localization [**?**]. Given fewer time constraints, it would be interesting to deploy this on a Jetson TX2. Additionally, being able to do this localization in 3D, with the ability to control a robot in 3D would be very interesting.

References

[1] “https://en.wikipedia.org/wiki/robot_navigation.”
[2] “https://en.wikipedia.org/wiki/kalman_filter.”
[3] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
[4] “https://en.wikipedia.org/wiki/rudolf_e._k
[5] “https://en.wikipedia.org/wiki/filtering_problem_(stochastic_processes).”
[6] L. Lamport, *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley, 1994.
[7] P. Del Moral, *”Non Linear Filtering: Interacting Particle Solution”*. Markov Processes and Related Fields., 1996.
[8] “https://en.wikipedia.org/wiki/particle_filter.”
[9] “https://en.wikipedia.org/wiki/monte_carlo_localization.”
[10] “http://wiki.ros.org/amcl.”
[11] “https://youtu.be/bvvht9xcqrm.”
[12] “https://youtu.be/3jyqn5_uipm.”
[13] “https://youtu.be/xaohh0p3nx8.”