Adaptive Monte Carlo Localization for Mobile Robots

Rohan Paleja

***Abstract*—Local and global localization are of utmost importance in mobile robotics and play a crucial role in a robot going from remote-controlled to completely autonomous. 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 experiment, this algorithm is used to localize a simulated mobile robot in the Gazebo environment. After deploying the AMCL and tuning its parameters to a specific mobile robot, the particles quickly converge on the pose. These parameters were then tested on a similar robot in the same environment and a similar convergence resulted. These results show that the AMCL is a viable localization algorithim and performs well in these conditions.**

***Index Terms*—Robot, IEEEtran, Udacity, Monte Carlo Localization.**

I. Introduction

A. Motivation for Study

Mobile robots play in key role in society today, and will have an even greater role in the future. With the ongoing work with autonomous driving cars, Unmanned Aerial Vehicles (UAVs), and autonomous underwater vehicles (AUVs), estimating the robot's pose (location, orientation) is pivotal to safe operation and proper functionality. High accuracy local localization (initial pose is known) can lead to the robot acquiring a high level of intelligence, and this alongside perception can lead to accurate awareness of environments. These maps can be useful in learning about unknown environments and allows for ease in future operations. As a simple example, imagine one opening a refrigerator and the difference between knowing exactly where the item he or she is looking for versus spending time searching. Applied on a global scale, localization in robotics can lead to massive gains in autonomy and thus, much time saved for human beings.

B. Problem Description

The job of localization of a mobile robot comes with numerous options concerning algorithms, parameters, and properties (speed, computation). The task to localize a simulated robot in the Gazebo physics engine environment given a robot with a hokuyo laser rangefinder and a camera requires choosing a specific algorithm, implementing it, and tuning it for high performance. In the coming sections, background knowledge will be presented that will assist in the choice of algorithm, a discussion of simulation formulation will be presented, and the results of this experiment will be explained.

II. Background

Two localization algorithms will be described and compared below: the Extended Kalman Filter (EKF) and the Monte Calro Localization (MCL). Some general definitions to help with the understanding of the below analysis are:
Local Localization: Localization where the initial pose of the robot is known
Global Localization: Localization where the initial pose of the robot is not known
Robot Kidnapping: Event where robot is teleported to another location in the map

A. Kalman Filters

Kalman filters is an estimation algorithm prominent in controls. It has the property of taking measurements with uncertainty or Gaussian noise and providing an accurate estimate. It does this by a continuous cycle of two steps, the measurement update, and state prediction. The measurement update is when a new set of measurements are taken by the robot sensors and replace the current measurements (note they are replacing, and this means there is no memory of data). The state prediction is the prediction of the future state using the current state and measurements. This cycle is done in real time and can be a great tool in position tracking. A slightly more mathematical definition can also be presented: The measurement update is a weighted sum of the prior belief and the measurement. The state prediction is the addition of the prior belief's mean and variance to the motion's mean and variance.
There are three types of types of Kalman filters, the KF (linear), the EKF (nonlinear), and the UKF (highly nonlinear). Advantages of this algorithm include that it can quickly converge to true values, and be used for sensor fusion provided that the noise and posterior are Gaussian. The Kalman filter utilizes the mean and covariance of data and uses the previously stated cycle to continuously locate the true mean of the data. Some drawbacks of the linear Kalman filter is that it cannot take nonlinear inputs. In any case where the output is not proportional to the input, this algorithm will fail. A simple example of this can be the relation between acceleration and position. As acceleration increases, the position of a vehicle increase in the order of a quadratic and the linear KF would fail to converge on the true position. This is where the EKF is used. The EKF is able to take nonlinear inputs and produce outputs disproportional to the input. It does this by approximating the nonlinear function as a linear function at each iteration using a Taylor series. One powerful aspect of the Kalman filter is

that its convergence is not very dependent on the initial guess of the state. Other benefits include very time and memory efficient and high resolution. Kalman filters can only be used for the local localization task.

B. Particle Filters

A particle filter uses a group of particles to keep track of the position and orientation of a robot. A particle is a virtual element that samples the robot. Each particle has a pose and is a guess of where the robot is located. These particles will be updated each time the robot takes new measurements. The power of using this technique is that this algorithm can solve the global localization problem given a ground truth map. This type of localization reduces the likelihood that the robot will ever get lost. The most popular particle filter used for localization is the Monte Carlo Localization algorithm. It has several advantages including it is easy to program, can represent any type of noise, and is very robust. At the start of MCL, particles are initially spread in a random manner around the robot. Each particle has a standard pose and a weight. These weights represent the difference between the actual position of the robot and the predicted. The higher this weight, the more likely the robot pose matches that of the particle and the more likely this particle remains after a resample. As time goes on, the particles will converge to the actual pose of the robot. During usage of this algorithm, there are two steps similar to the KF. There is a motion and sensor update, and then a re sampling process. This is done for each particle, so this algorithm can quickly become very computationally heavy. A brief summary of this process can be seen as: start with the previous belief, perform a motion update, perform a measurement update, resample, and compute the new belief. This process will repeat every movement.

C. Comparison / Contrast

The choice of algorithm between KF and MCL is very dependent on its usage. Some general differences between the KF and MCL include that a KF uses landmarks while the MCL uses raw sensor data. The KF is limited to sensors with Gaussian noise and local localization. The MCL provides more robustness, and can cater to any sensor. Most importantly, it can perform global localization and thus, is chosen for this experiment. More so, figure 3 depicts the map of the test environment. Clearly, there are not many landmarks; thus, the EKF would perform poorly in this scenario. Other reasons why the MCL outperforms EKF in this case is that it is not safe to assume that the noise is strictly Gaussian. In most cases, real world noise and the posterior are not Gaussian. In the coming simulation, the algorithm used will be the Adaptive Monte Carlo Localization (AMCL). This is very similar to MCL, but the number of particles change based on parameters set by the user.

III. Simulations

To localize a robot in Gazebo using the AMCL package, several steps were conducted. A mobile robot, gazebo environment, and corresponding ROS package were built. Then, these parameters were tuned and tested. The formulation will be presented below. Note that two mobile robots were created and tested on, and the discussion will switch back and forth between them, describing and comparing them.

A. Building the Mobile Robot and Gazebo Environment

The benefits of creating mobile robots in simulation are that one has the freedom to quickly prototype different robots with different sensors and one does not have to worry about parts breaking due to crashes (if the localization algorithm or sensor fails, this will result). To create a mobile robot, a Unified Robot Description Format (URDF) file is created. This URDF specifies the visual, inertial, and collision properties including the location, shape, and origin. Both robots, the first named the Udacity Bot and the second named User-Created Vehicle both have castor wheels to stabilize the vehicle from tipping in the z-axis. In both vehicles, the castors are defined as part of the chassis but in the User-Created Vehicle the castors have friction and interfere with the motion of the vehicle (as they would in reality). 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. Each link in the Udacity Bot are of radius .1 m and length of .05 m. The wheels of the User-Created Vehicle have a .08 m radius and length of .05. Both vehicles have a wheel mass of 5 kg. Sensor are then added to the vehicles. Both vehicles have a hokuyo laser and a camera. 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. Note the positions of the rangefinder and the camera are different for the two vehicles. This leads to different measurements being sensed as the robot moves, and thus, may result in different outcomes in performance of AMCL. Note for the wheeled robot, a differential drive controller is used. This type of controller specifies many of the upcoming parameters. The last important step to creating the vehicle 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. Below are images of both vehicles in the environment.

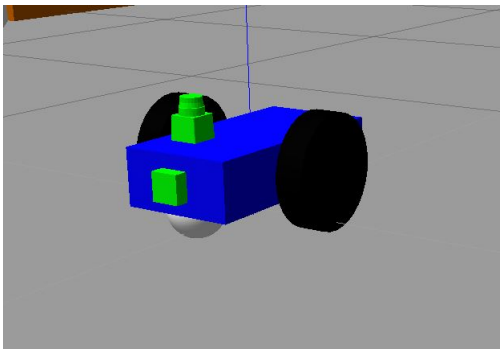

Fig. 1. Udacity Bot in the Gazebo Environment

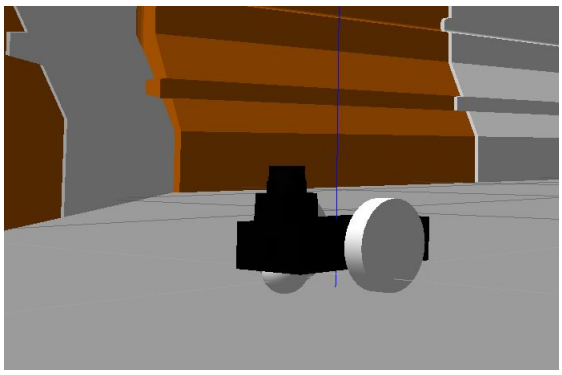

Fig. 2. User-Created Vehicle in Gazebo Environment

The Gazebo environment or ”world” is also defined in the XML format. In this file, all the models and elements of the world are added to it, including the vehicle model for the chosen simulation. The environment for this experiment was kindly given by Udacity and is presented below. This

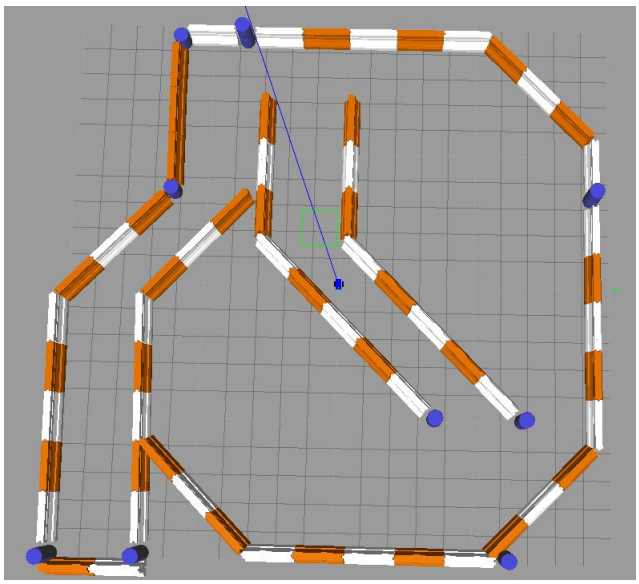

Fig. 3. Gazebo Environment

environment has some color data that the camera can use for localization. It also has some bends and texture which both the camera and rangefinder can make use of. There are not many ”landmarks” in this environment and this is a reassurance the AMCL was a preferable choice compared to the EKF.

B. AMCL and the Navigation Stack

As previously stated, the AMCL dynamically adjusts the number of particles over a period of time. This provides a computational advantage over the traditional algorithm. The AMCL has a node that will define its behavior in RVIZ and parameters that will determine how effectively it can localize itself. In this stage, topics are remapped to fit the AMCL specifications and the truth map is published to the RVIZ program. The navigation stack uses the move_base package to move the robot to a goal position specified by the user. This goal can be expressed in RVIZ through a click interface or through a node publishing a pose. The move_base package utilizes a cost map, where the maps has parts that are occupied, such as obstacles, and areas where the robot is allowed to move through known as free space. As the robot move, the local cost map is continuously updated. Some special features within this package include the ability to get out of a still position by rotation. This navigation stack also has its own sets of parameters that help it function efficiently.

C. Parameter Tuning

Parameter tuning is vital to make sure the AMCL and navigation stack packages work correctly. The goal is to tune the parameters in a way such that all the particles converge onto the true pose of the robot.

1) AMCL Parameters: The AMCL package has parameters that relate to the filter, laser, and odometry. The first two parameters that relate to the filter are min_particles and max_particles. The maximum number of particles starts off at the initial and then AMCL dynamically decreases the number of particles toward the preset minimum. One important aspect of these parameters is that if the maximum is too high, it might be too computationally extensive and lead to a laggard system. The transform_tolerance parameter determines the longevity of the transforms being published for localization purposes. A good value removes all lags from the system. Another parameter known as kld_err is the maximum error between the true distribution and the estimated distribution. The default for this value is set to .01 but lowering it should increase accuracy. Update_min_a and Update_min_d are defined as the translational movement required before performing a filter update and the rotation movement required before performing a filter update, respectively. By default, these are set to .2 and .52, respectively. Lowering these values would result in more updates and thus more iterations which will have the affect of increasing accuracy, while also increasing computation.

2) Laser Parameters: Parameters for the laser can be changed to increase the amount of incoming data from this sensor. The parameter laser_max_beams determine the amount of beams in each scan of laser to be used when updating the filter. The value is very dependent on the sensor. The laser_max_range parameter describes the maximum scan range of the laser. The laser_likelihood_max_dist parameter determines the maximum distance to do obstacle inflation on map. Note this parameter is for the likelihood_field model which is chosen to due its computational efficiency. Two other important parameters used to increased the accuracy of localization are the laser_z_hit and the laser_z_rand which are weights for the z_hit and z_rand part of the model.

3) Odometry Parameters: Odometry parameters describe the movement of the robot and provide the AMCL information about this movement. The odom_model_type is the diff_corrected type. This is due the sensor plugins making this robot a differential drive mobile robot. There are also 4 odom_alpha parameters. Each (in order) specifies the expected

noise in odometry's rotation estimate from the rotation component of the robot's motion, the expected noise in odometry's rotation estimate from the translational component of the robot's motion, the expected noise in odometry's translation estimate from the translation component of the robot's motion,and the expected noise in odometry's translation estimate from the rotation component of the robot's motion, respectively. Note the default values must be lowered for proper use.

D. Move_base Parameters

The move_base parameters are located in 4 configuration files. Each files adds a specific functionality that aids in the package.

1) Local Costmap Parameters: The local costmap uses the odom frame as the global frame since it updates as the robot moves. Other parameters include the width and height of the local map. The update frequency and publish frequency determine how often the map is updated, and how often the map will publish to display information. Note these values are closely related with the AMCL parameters as computation determines how fast the system can update. Also, the resolution of the map can be chosen. The rolling_window parameter is set to true in this case. This means a rolling window version of the costmap will be used.

2) Global Costmap Parameters: Many of these parameters are the same as the local costmap parameters including the width, height, resolution, update frequency and publish frequency. Some differences include the global_frame is now the static global map. Also, the rolling_window parameter is now set to false.

3) Common Costmap Parameters: The navigation stack uses costmaps to store information about obstacles and the world. For this, the costmap must know which sensor topics they must listen to, and must know the parameters that define the costmap. The parameters defined are obstacle_range, raytrace_range, transform_tolerance, robot_radius, inflation_radius, xy_goal_tolerance, and yaw_goal_tolerance. The obstacle_range and raytrace_range define the maximum range sensor reading that will result in an obstacle being put into the costmap and the range to which free space is given to a sensor reading, respectively. The inflation_radius is the maximum distance from obstacles at which a cost should be incurred. The xy_goal_tolerance is the position tolerance for the controller when achieving a goal. This can be lowered to increase the system accuracy, but will undoubtedly increase the time to reach the goal destination. the yaw_goal_tolerance is the tolerance in orientation when achieving a goal. Again, this can be lowered to increase system accuracy, but will undoubtedly increase the time to reach goal destination. The observation_sources parameter describes the sensors that will pass information to the costmap. Here, the topic to listen to, data type, and frame name are also defined.

E. Base Local Planner

This configuration file helps the navigation stack calculate a path to the robot. The parameters used in this file are acc_lim_theta, acc_lim_x, acc_lim_y, holonomic_robot, max_vel_x, min_vel_x, max_vel_theta, and min_in_place_vel_theta. The acc_lim_x and acc_lim_y parameters correspond to the acceleration limit of the robot in the corresponding direction given in meters/sec2. The acc_lim_theta is the rotational acceleration limit given in radians/sec2. The max_vel_x and min_vel_x correspond to the maximum and minimum forward velocity allowed for the base. The max_vel_theta parameter is the maximum rotational velocity of the base in radians/sec. The last parameter min_in_place_vel_theta is the minimum rotational velocity allowed for the base when performing in place rotations given in radians/sec.

F. Udacity Bot Model

1) Model design: The Udacity Bot is a mobile robot with a rectangular base as the chassis, two caster wheels, and two wheels for driving. This mobile robot has a hokuyo laser and a camera. Information about pose etc. is given in the table below. Note the mass of the chassis is 15 kg, wheels are each 5 kg, and sensors are .1 kg.

Robot Item	Pose	Excess Information
Chassis	(0 0 .1 0 0 0)	Prism: L: .4 W: .2 H: .1
Front Caster	(.15 0 -.05 0 0 0)	Sphere: radius-.05
Back Caster	(-.15 0 -.05 0 0 0)	Sphere: radius-.05
Left Wheel	(0 .15 0 0 1.57 1.57)	Cylinder: radius-.1 length-.05
Right Wheel	(0 -.15 0 0 1.57 1.57)	Cylinder: radius-.1 length-.05
Camera	(.2 0 0 0 0 0)	Cube: S-.05
Laser	(.15 0 .1 0 0 0)	Mesh

Table 1: Udacity Bot Model Configuration

This robot was used as a base model for testing a "standard" mobile robot in a gazebo environment.

2) Packages Used: Several packages are used for this experiment including the main packages of AMCL and move_base. Alongside this, RVIZ and gazebo programs are launched for physics and visual simulation. Several nodes are also launched to publish and subscribe to different topics to transport camera data (image_raw), hokuyo laser data(/udacity_bot/laser/scan) and others to manage the publishing of the ground truth map and local and global maps generated by AMCL.

3) Parameters: Previously stated were the definitions of many parameters for the AMCL and move_base package. The actual chosen values alongside reasoning will be presented here. First, the values for the move_base package will be

presented. All these values are constant for both models, the Udacity Bot and User-Created. The parameters used in this file are

Parameter	Value
max_vel_x	.45
min_vel_x	.1
max_vel_theta	1.0
min_in_place_vel_theta	.4
acc_lim_x	2.5
acc_lim_y	2.5
acc_lim_theta	3.2

Table 2: Base Local Planner Parameters

These base local parameters are very similar to the default parameters defined by the package. Some difference include a lowered maximum velocity in the x direction.

Parameter	Value
global_frame	map
update_frequency	20
publish_frequency	20
width	40
height	40
resolution	.05
rolling_window	false

Table 3: Global Costmap Parameters

The global costmap parameters have been altered from the default values. The update and publish frequency were set to 20 Hz, as my computer was averaging around .05 seconds per cycle. Thus, I defined the update frequency as 20 Hz. I chose a publish frequency to match this as this was ensure maximizing the amount of information from RVIZ. The width and height of this map was set to 40 m by 40 m which is the size of a start large office. This was chosen to be twice the size of the local cost map.

Parameter	Value
global_frame	odom
update_frequency	20
publish_frequency	20
width	20
height	20
resolution	.05
rolling_window	true

Table 4: Local Costmap Parameters

The local cost map parameters are very similar to the global parameters. Some things that have changed are the global frame, the width, and the height. The width and height are halved. This makes sense because the local frame should be much smaller than the global frame. Reducing this even more would be required on the Jetson, but as a virtual machine is used for this experiment, memory was not a limitation.

Parameter	Value
obstacle_range	2.5
raytrace_range	3.0
transform_tolerance	.3
inflation_radius	.4
xy_goal_tolerance	.05
yaw_goal_tolerance	.05

Table 5: Costmap Common Parameters

The parameters chosen for the costmap were very similar to the sample costmap given on ROS wiki. An obstacle range of 2.5 meters is fairly lengthy, and provides a good estimate of where the robot of this size can travel. After a gap of .5 meters, the costmap will mark the area as free space the robot can travel in. The transform tolerance chosen is slightly higher than the default value of .1 seconds. This means a transform is valid farther into the future and this can be done when sensor data is very confident. The inflation radius is set to a standard value of .55 meters. All obstacles past this point have an equal obstacle cost. Lastly, the pose goal tolerance is reduced to ensure high accuracy. This may not provide better results in the real world (if the system has high uncertainty) but in the case of simulation where data is almost perfect, lowering the tolerance results in more accurate AMCL implementation. The AMCL parameters are specific for this vehicle. These are presented in the table below.

Parameter	Value
min_particles	10
max_particles	50
kld_err	.001
update_min_d	.25
update_min_a	.25
resample_interval	1
transform_tolerance_a	.01
laser_max_beams	50
laser_max_range	12
laser_likelihood_max_dist	1.5
laser_z_hit	.8
laser_z_rand	.2
odom_alpha1	.05
odom_alpha2	.05
odom_alpha3	.05
odom_alpha4	.05

Table 7: AMCL Parameters

The number of particles was chosen based on computing power. 500 and 1000 were attempted but caused huge lags due to computation limits. Decreasing the kld_err results in a much closer end result to the goal pose. Increasing the update values from their default of .2 meters results in a more smooth decline and convergence of particles. Changing the resample interval to 1 instead of its default value of 2 results in a higher number of updates throughout a run. This parameter and the previous two have some correlation. The transform tolerance was set to a very low value of .01. This decreases the longevity of

a transform and can result in higher robustness. Moving onto the laser parameters, the max_beams for the laser was set to 50 rather than its default of 30 to increase the amount of data coming in. The choices for the laser hit and rand values were chosen by iteration and the combination of .8 and .2 seemed to produce the best results. For the odom_alpha values, a low value of .05 was chosen. This values were found by iterating and also looking on Wiki ROS pages.

G. User-Created Model

This model was created very similar to the Udacity bot. The most significant changes are introducing friction on the casters, a smaller chassis, smaller wheels, and different poses for the sensors. Changes in mass were also attempted but seemed to cause disturbance in motion.

1) Model design: Below is a table for the model configuration similar to table 1.

Robot Item	Pose	Excess Information
Chassis	(0 0 .1 0 0 0)	Prism: L: .2 W: .2 H: .05
Front Caster	(.075 0 -.025 0 0 0)	Sphere: radius-.025
Back Caster	(-.075 0 -.025 0 0 0)	Sphere: radius-.025
Left Wheel	(0 .075 0 0 1.57 1.57)	Cylinder: radius-.05 length-.025
Right Wheel	(0 -.075 0 0 1.57 1.57)	Cylinder: radius-.05 length-.025
Camera	(.1 0 0 0 0 0)	Cube: S-.025
Laser	(.1 .1 .1 0 0 0)	Mesh

Table 8: User-Created Model Configuration

2) Packages Used: The packages used were the same as the ones for the Udacity Bot.

3) Parameters: As stated above, the parameters for the configuration files remained the same. The parameters in the AMCL.launch file were altered slightly. Here, the differences will be noted and the other parameters can readily be assumed the same as the Udacity Bot. Computation seemed to increase during testing this vehicle, and thus the number of maximum particles was lowered to 50. Also, decreasing the odom_alpha values to .005 seemed to have a large effect and increase performance. It was expected that too many parameters would not need changes as the vehicle was fairly similar. One surprising aspect was that the heightened friction of the castor wheels did not seem to have any effect.

IV. Results

Adaptive Monte Carlo Localization performed well in a Gazebo simulated environment on a mobile robot with a camera and hokuyo laser. Quick convergence of particles and reaching the goal accurately were both aspects that provide proof for the previous statement. It was seen in several simulations (look at pictures presented below) that halfway through a given trajectory, majority of the particles had already converged to the robot's position. While due to the parameters honing in on accuracy, the simulations had increased duration. This assured that the critical goal of reaching a destination pose was achieved. Slight misbehavior can be viewed at the start of following a trajectory. More misbehavior is seen when the robot is stuck. Overall, on linear corridors, or corridors with slight turns, the AMCL algorithm works well to localize a mobile robot.

A. Localization Results

Below is the presentation of the AMCL algorithm in the gazebo environment. The images depict an RVIZ environment that shows the ground truth map in the background, alongside camera readings, laser readings, and a local/global map. The images below represent the mobile robot tasked to go to one destination pose and then to another.

1) Udacity Bot: Below is a depiction of the robot at the initial launch of the RVIZ environment. Note the particles are spread uniformly. After this, a 2D NAV goal is supplied by the

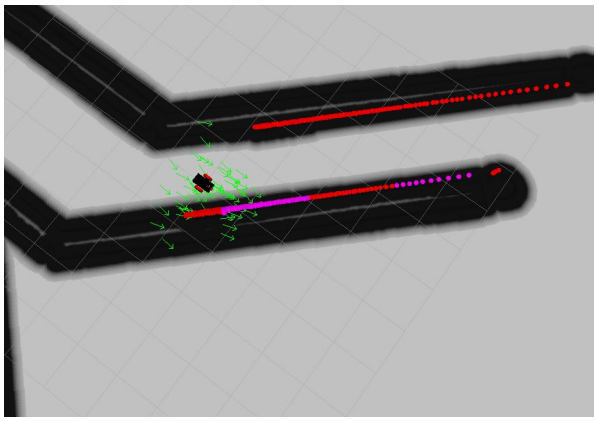

Fig. 4. Udacity Bot in the RVIZ Environment at Launch

user through the RVIZ environment or by a node. A trajectory will be generated that the robot must follow. A good indication that the AMCL algorithm is working properly is shown by the pictures below taken at the quarter and halfway point to the destination. It can easily be seen that the particles have

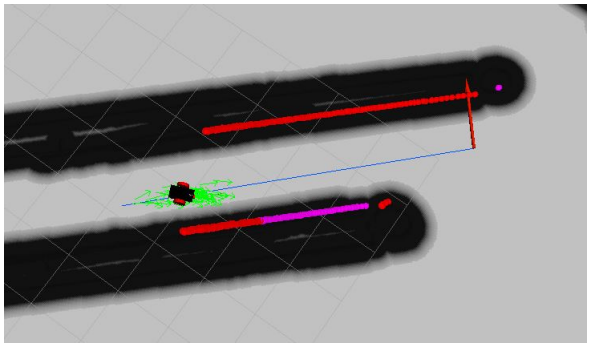

Fig. 5. Udacity Bot Quarterway to the First Destination

converged a great amount. The picture at end goal also shows

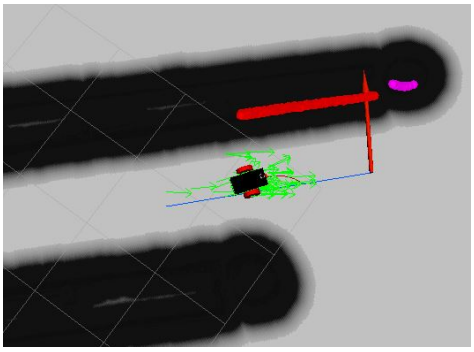

Fig. 6. Udacity Bot Halfway to First Destination

similar results. A depiction of the Udacity Bot to the second goal is also shown. This second destination is slightly longer

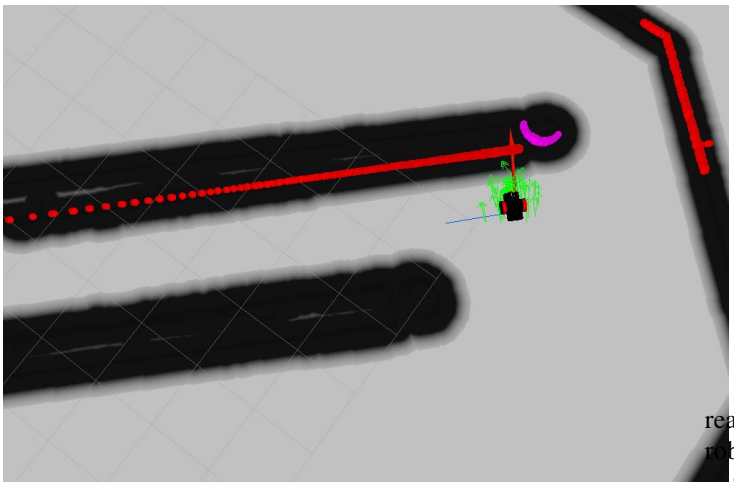

Fig. 7. Udacity Bot at End Goal

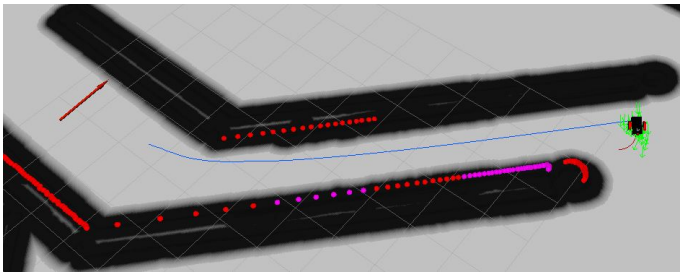

Fig. 8. Udacity Bot Traveling to Second Goal

than the first and has a slight end. The depiction below shows the particles near the bend. Notice that the position of the particles is less accurate but the orientation of the particles are almost all identical. The ending of the second goal is shown below. This had better convergence than the first goal. This is also expected. Over time, the AMCL algorithm should keep improving. Overall, the AMCL algorithm was very successful on the Udacity Bot. This algorithm was tested 15-20 times and

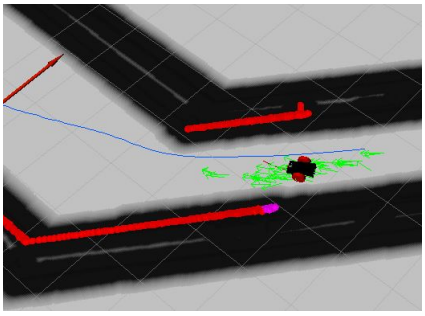

Fig. 9. Udacity Bot Halfway to Second Destination

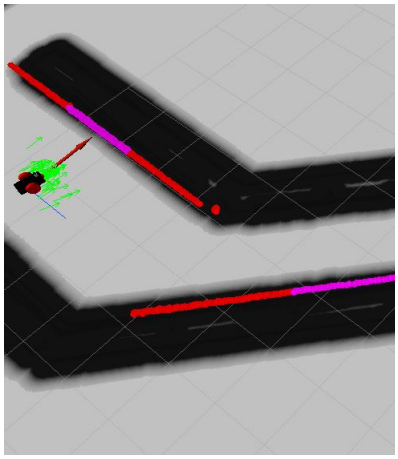

Fig. 10. Udacity Bot at Second Goal

reached the correct position all trials except for one where the robot got stuck in a position.

2) User-Created Vehicle: At the initialization of the user-created vehicle, the particles were uniformly spread. A depiction is shown below. This vehicle traveled to its goal in a

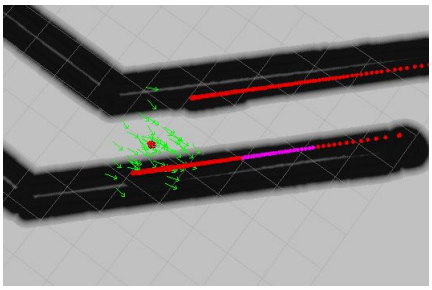

Fig. 11. User-Created in the RVIZ Environment

much smoother fashion. The result of the first goal is depicted below. Then, a second goal was given to this vehicle to repeat the same experiment as the Udacity Bot. The results are shown below. The results are slightly less convergent if compared visually to the first robot. Parameter tuning iteratively proved that this set of parameters performed best on this vehicle. Overall, these results prove AMCL is viable for localization. Majority of the particles represent the true pose of the vehicle.

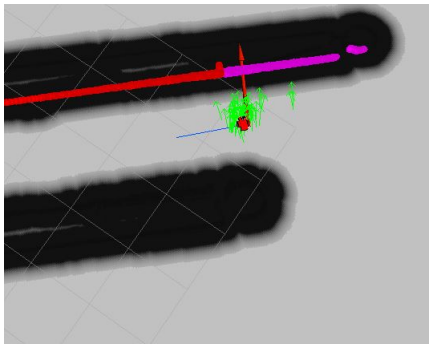

Fig. 12. User-Created Vehicle at First Goal

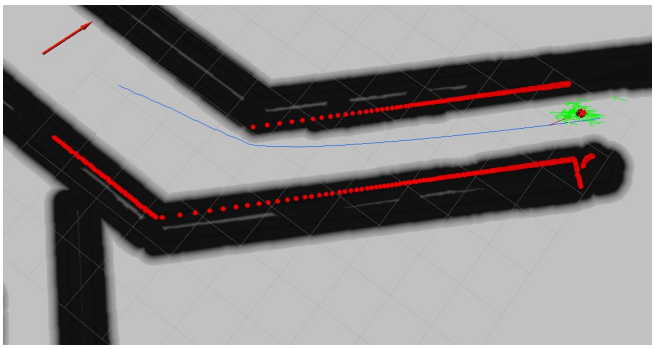

Fig. 13. Path to Second Goal for the User-Created Robot

V. Discussion

The experiment conducted on both vehicles was a first destination set (randomly chosen) and then a second destination chosen around a bend. Areas of focus were rate of convergence, particle convergence at first goal, trajectory following, particle convergence near the bend, and particle convergence at the second goal. In both cases, the robots did converge well at each goal. The User-Created Robot outperformed the Udacity Bot on movement and trajectory following. This may be due to the size difference between the vehicles. The first vehicle had tighter convergence at both goals. This can be explained slightly due to a bigger body, and thus, a higher likelihood that a particle will fall on that body. Near the bend, both vehicles showed slight inaccuracy which is to be expected when the environment is changing drastically. The User-Created Vehicle and its smaller size seemed to allow for a reduction in some parameters that increased accuracy while the larger vehicle did not. The Udacity Bot overall did perform better using the AMCL algorithm. A slight explanation for this may be the locations of the sensors were higher off the ground and thus pick up more data. More data would results in better performance and faster convergence.

A. Topics

- Which robot performed better?
  The Udacity Bot outperformed the User-Created Vehicle.
- How would you approach the 'Kidnapped Robot' problem?

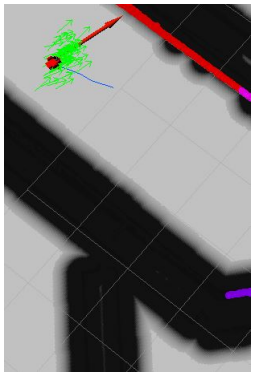

Fig. 14. User-Created Robot at Second Goal

  A viable method to approach the kidnapped robot model is if the robot could first travel through the environment and create a global map of it. If the robot also has a ground truth map, and the environment has enough features, the 'Kidnapped Robot' problem can be reduced to a matching of landmarks problem.
- What types of scenario could localization be performed?
  AMCL works in local and global localization. This would work in cases where a robot is deployed onto a street, or in a building. Scenarios where the environment does not have features such as the desert or where high sunlight is present (interference of laser or camera) may cause poor localization.
- Where would you use MCL/AMCL in an industry domain?
  A viable use is warehouse usage. Robots can travel to designated spots to carry an item and bring it to a packing location. Amazon has already implemented this, but it is unsure if they are using AMCL or some other localization algorithm.

VI. Conclusion / Future work

Adaptive Monte Carlo Localization is a very viable solution for position tracking of mobile robots. Experiments in a ROS-Gazebo environment resulted in evidence that the particles in the AMCL algorithm converge quickly to the correct pose of the robot. Moreover, the robot was able to travel to several positions quickly while performing this algorithm in real time. To deploy this algorithm to hardware, not many steps are required. Other than tuning the parameters to the computational needs of the device (reducing number of particles, etc.), sensors must be setup and remapped to the correct topics. After that, much of the visualization and analysis can be done in a similar manner. This can be applied to a simple household robot like the Roomba and can have the goal of localizing itself in the hopes of re-finding its charging station. AMCL would perform well in applications similar to this. Many different steps must still be conducted to test for robustness of the AMCL algorithm. Different drive controllers and sensors must be tested. Different levels and types of noisy data should be tested to ensure that it is robust to noise. Different environments that

have different obstacles should be tested to ensure capabilities in different settings. Lastly, deployment is very important in testing if the algorithm is viable in reality. An addition of sensors may prove the extensibility of AMCL. Sensors that are common to most mobile robots and should be added include GPS, wheel encoders, and a depth camera. Running this algorithm and an EKF would also be a nice comparison in real time. Even though computationally heavy, a combination of these two might prove to localize well.

REFERENCES

[1] Amcl Package Summary. (n.d.). Retrieved March 16, 2018, from http://wiki.ros.org/amcl#Parameters.

[2] Costmap Parameters. (n.d.). Retrieved March 16, 2018, from http://wiki.ros.org/costmap_2d#costmap_2d.2BAC8-layered.Parameters

[3] Thrun, S. (2000). Robust Monte Carlo localization for mobile robots. Pittsburgh, PA: School of Computer Science, Carnegie Mellon University.