Robotics Nanodegree Localization Project

Sebastian Castro — sebas.a.castro@gmail.com

Abstract—This paper outlines the design and implementation of the Adaptive Monte Carlo Localization (AMCL) algorithm. This is tested using two simulated mobile robots – a benchmark robot and a personal robot – navigating a known mapped environment from a start to a goal position. Some additional topics are discussed, such as simulated robot design, parameter choices and tradeoffs, and application and deployment of AMCL in the real world.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization.

1 Introduction

Localization is an important component of autonomous system behavior. Usually, localization must be accurate and robust, since many software components in the software stack of a robot rely on a good estimate of a robot's location in the environment to make decisions.

Depending on the environment in which a robot operates, different sensors may be available for localization. Common sensors include global positioning system (GPS), inertial measurement units (IMU), cameras, wheel encoders, and line-of-sight sensors such as radar and lidar [1]. In practicality, all sensors have advantages and disadvantages, and it performing *sensor fusion* to combine localization estimates from multiple sources can lead to better results.

Many of these sensors require additional information to be used in localization. Knowledge of the robot dynamics is needed for proprioceptive (self-sensing) sensors such as wheel encoders, and knowledge of the environment – most commonly, a map – is needed for exteroceptive (externally sensing) sensors such as lidar.

In this paper, we will explore one such combination of robot models, environment maps, and sensor types and layout, to robustly perform robot localization and navigation in a known environment.

It is also worth noting that other techniques exist to use this same sensory information in situations where the environment is unknown and must be determined at the same time as localization. This type of problem is referred to as Simultaneous Localization and Mapping (SLAM) [2].

2 Background

This paper specifically deals with the class of localization problems where the robot dynamic model and a map of the environment are known. This means that all sensor information can directly be used towards localization.

The two most common localization approaches are Kalman filters [3] and particle filters [4] (also referred to as Monte Carlo localization (MCL)).

Both of these techniques rely on a model of the robot to perform a prediction step and sensor measurements to perform an update step. They also can both operate using noisy measurements from multiple sensors and improving localization results by fusing these results to produce a less uncertain estimate compared to using any individual sensor.

2.1 Kalman Filters

Kalman filters have been successfully implemented for over half a decade. Intuitively, a Kalman filter defines an optimal weighting of predicting the robot dynamics and correcting those predictions with sensor measurements. This weighting is principally defined by a priori knowledge of the process and sensor noise.

Traditional Kalman filters are computationally inexpensive because they rely on linear models and Gaussian, zero-mean, noise distributions. Therefore, Kalman filters require only simple matrix computations. However, these assumptions are often insufficient to adequately represent real, complex robotic systems. Variations such as the Extended Kalman filter (EKF) [5] and Unscented Kalman filter (UKF) [6] have been devised to better deal with nonlinear dynamic models and non-Gaussian uncertainty distributions.

2.2 Particle Filters

Particle filters, unlike Kalman filters, are a sampling-based localization approach. Intuitively, a particle filter creates an initial distribution of pose estimates (or *particles*). Each particle is moved using knowledge of the robot dynamics and actuator commands, and is then assigned an importance weight based on the probability of each particle producing the observed sensor measurements. Since more probable pose estimates are more likely to be resampled at the next time step, the distribution of particles ideally converges near to the true pose of the robot.

In theory, a larger number of particles will yield a better pose estimate. Statistically, localizing with an infinite number of particles would converge towards the absolute true pose of the robot. In practice, the number of particles must be limited due to computational resources. Algorithms such as Adaptive Monte Carlo Localization (AMCL) [7] have been designed to vary the number of particles during execution to combine the advantages of large number of particles with the computational efficiency of a small number of particles.

2.3 Comparison / Contrast

To summarize the two approaches, the following comparisons are offered.

- Kalman filters are computationally more efficient than particle filters, since they rely on a single mean and covariance estimate rather than tracking a large number of individual estimates.
- Kalman filters traditionally rely on linear models with Gaussian, zero-mean noise. Although variations exist to relax these constraints, particle filters tend to yield better results if the real system is highly nonlinear with non-Gaussian noise distributions.

Generally, Kalman filters are more effective when the number of individual readings is relatively small – for example, if there is no map, or there are a few markers or beacons to keep track of. For a robot with hundreds of lidar range measurements and a set of obstacles and walls that cannot easily be expressed in closed form, it is easier to instead use particle filters and Bayes' theorem to estimate the probability of an estimated pose producing the measured lidar scans given the known map.

For the work presented in this paper, particle filter localization was selected – specifically, the AMCL algorithm. The following section describes the robot model and sensor configurations used to perform localization.

3 Approach

3.1 Model Design

To test localization and navigation capabilities, two robot simulation models were created – a benchmark robot and a personal robot. Both models were built using URDF files and plugins (camera, lidar, odometry, base controllers) for the Gazebo simulator.

3.1.1 Benchmark Model

The benchmark robot is a differential drive robot with a front-facing camera and lidar. Figure 1 shows the robot in RViz at its initial position and Table 1 lists some key robot parameters.

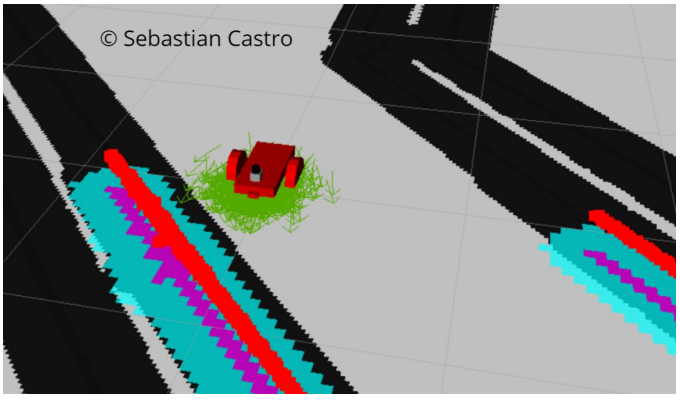

Fig. 1. Benchmark model at initial position

TABLE 1
Benchmark Model Parameters

Total Mass	25.2 kg
Wheel radius	10 cm
Wheelbase	40 cm
Camera location [XYZ]	[0.2 0 0] m
Camera field of view	80 deg
Lidar location [XYZ]	[0.5 0 0.1] m
Lidar range	0.2 - 30 m
Lidar angles	720 samples, -90 to +90 deg

3.1.2 Personal Model

The personal model was developed as an extension to the benchmark model. The main difference is the redesign of the base to support a full 360 degree lidar scanner view. The list of changes is below:

- Lidar angular range doubled to 360 degrees, but with the same number of scans, so angular resolution was halved.
- Wheel radius decreased by 25%, wheels lowered, and lidar raised to prevent lidar collisions
- Lidar was moved to be directly above the center of the robot chassis to make coordinate transformations simpler
- Camera was moved to the top of the robot chassis instead of in front of the robot with safety against frontal collisions in mind. Since the camera was moved further towards the center of the robot, field of view was decreased by 5 degrees.
- Cosmetic changes to URDF materials: body was colored black, wheels gray, and camera red

Figure 2 shows the personal robot in RViz at its initial position and Table 2 lists some key robot parameters.

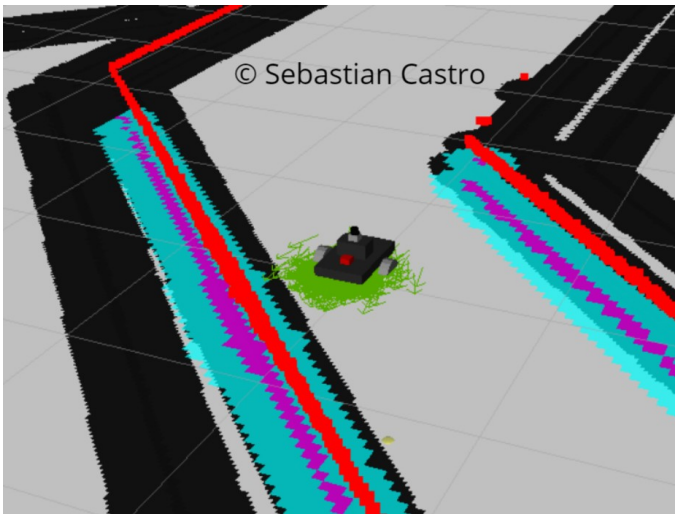

Fig. 2. Personal model at initial position

3.2 Packages Used

This project implementation is named `udacity_bot`, which can be downloaded from this GitHub repository. It depends mainly on the ROS packages `amcl` for localization

TABLE 2
Personal Model Parameters

Total Mass	25.2 kg
Wheel radius	75 cm
Wheelbase	45 cm
Camera location [XYZ]	[0.1 0 0.05] m
Camera field of view	75 deg
Lidar location [XYZ]	[0 0 0.15] m
Lidar range	0.2 - 20 m
Lidar angles	720 samples, -180 to +180 deg

and `move_base` for navigating in a mapped environment using local and global costmaps.

For both robots, the list of ROS nodes and topic names can be found in Table 3 and in the `rqt_graph` snapshot in Figure 3.

TABLE 3
ROS Topic Names

Odometry	`\odom`
Camera image	`\udacity_bot\camera1\image_raw`
Laser scan	`\udacity_bot\laser\scan`
Localization particles	`\particle_cloud`
Localization pose	`\amcl_pose`
Velocity command	`\cmd_vel`
Navigation (action server)	`\move_base`
Global costmap	`\move_base\global_costmap\costmap`
Local costmap	`\move_base\local_costmap\costmap`

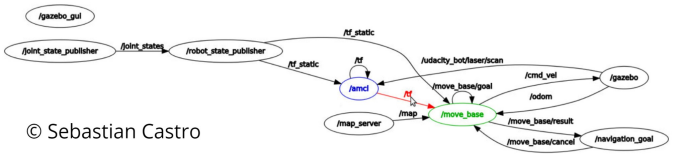

Fig. 3. ROS graph of simulated robot, AMCL, `move_base`, and `navigation_goal` nodes

3.2.1 *Benchmark Model*

Bringing up the experiment for the benchmark robot requires the following commands:

- `roslaunch udacity_bot udacity_world.launch`
- `roslaunch udacity_bot amcl.launch`
- `rosrun udacity_bot navigation_goal`

3.2.2 *Personal Model*

Bringing up the experiment for the personal robot requires the following commands:

- `roslaunch udacity_bot udacity_world_sebastian.launch`
- `roslaunch udacity_bot amcl_sebastian.launch`
- `rosrun udacity_bot navigation_goal`

3.3 Parameters

The navigation parameters for AMCL and mobile base control were selected with the intent of being general enough to localize and control both benchmark and personal robots so they effectively reach the goal position within the provided map. A few parameter values were different for both robots, which will be discussed below.

The AMCL parameters were chosen as shown in Table 4.

- The range of particle numbers was reduced from the default 100-5000 to 100-1000 to reduce computation and memory usage while still localizing the robot in the known map.
- Initial X and Y pose covariance was reduced significantly to $0.1 * 0.1 = 0.01m^2$ since quick convergence was useful to more reliably navigate around the wall directly in front of the robot starting position, while the estimated pose is still converging. However, localization results still converges with larger initial covariance.
- Update distances – both linear and angular – were reduced from their default values so the filter updated more often to avoid changes in localization results.
- Lidar range and hit standard deviation were chosen to match the simulated lidar parameters, and therefore differ between the benchmark and personal robots.

TABLE 4
AMCL Parameters

Particle numbers	100 - 1000
Update minimum distance	5 cm
Update minimum angle	0.2 rad
Initial std. dev. [X, Y, theta]	[0.1 0.1 $\frac{\pi}{12}$]
Laser range (benchmark)	0.2 - 30 m
Laser range (personal)	0.2 - 20 m
Lidar location [XYZ]	[0 0 0.15] m
Lidar range	0.2 - 20 m
Max laser beams	720
Laser hit std. dev.	0.1

The mobile base control parameters were chosen as shown in Table 5.

- Different inflation radius was used for both robots. Specifically, a smaller radius was chosen for the personal robot because of its wider wheelbase requiring more free space for the robot to navigate, and because the lidar sensor was farther towards the back of the robot.
- Obstacle range was chosen to be 5 meters in accordance with the local costmap size of 5 meters. This was sufficient for the robot to navigate obstacles without using too large a local costmap.
- Both local and global costmaps used a resolution of 0.05 meters (5 cm)
- To ensure the algorithms ran on the development machine without significant timing issues, the map was updated at 5 Hz, the base was controlled at 10 Hz, and a transform tolerance of 0.2 was selected.

TABLE 5
Move_Base Parameters

Inflation radius (benchmark)	0.25 m
Inflation radius (personal)	0.2 m
Obstacle range	5 m
Ray trace range	10 m
Costmap resolution	0.05 m
Global costmap size	40 m
Local costmap size	5 m
Costmap frequency	5 Hz
Controller frequency	10 Hz
Transform tolerance	0.2

4 Results

With the design choices described in the previous section, both robots were able to successfully localize and navigate from the starting position to the goal specified in the `navigation_goal` program.

4.1 Localization Results

4.1.1 Benchmark Robot

Figure 4 shows the in-progress and final navigation results for the benchmark robot. As can be seen in the figure, the AMCL particles converge around the true robot pose. In fact, they remain close to the robot but diverge again when the robot changes orientation around the goal position.

Figure 5 shows the time history of the ground truth and localized X and Y positions of the benchmark robot, and that they are consistently close to each other during the navigation task. Navigation for the benchmark robot took approximately 47 seconds.

4.1.2 Personal Robot

Figure 6 shows the in-progress and final navigation results for the personal robot. As can be seen in the figure, the AMCL particles converge around the true robot pose. In fact, they remain close to the robot but diverge again when the robot changes orientation around the goal position.

Figure 7 shows the time history of the ground truth and localized X and Y positions of the custom robot, and that they are consistently close to each other during the navigation task. Navigation for the benchmark robot took approximately 75 seconds.

4.2 Technical Comparison

Although both robots were able to successfully localize in the environment using AMCL, the benchmark robot completed the navigation task approximately 63% faster than the personal robot (47 vs. 75 seconds).

Based on the results, this was primarily due to the personal robot path tracing a significantly wider arc towards the goal after clearing the initial corridor in the map. This can be seen by comparing the plots in Figures 4 and 6, or by seeing that the path leading the goal position in Figure 4 is much closer to the wall than the one in Figure 6.

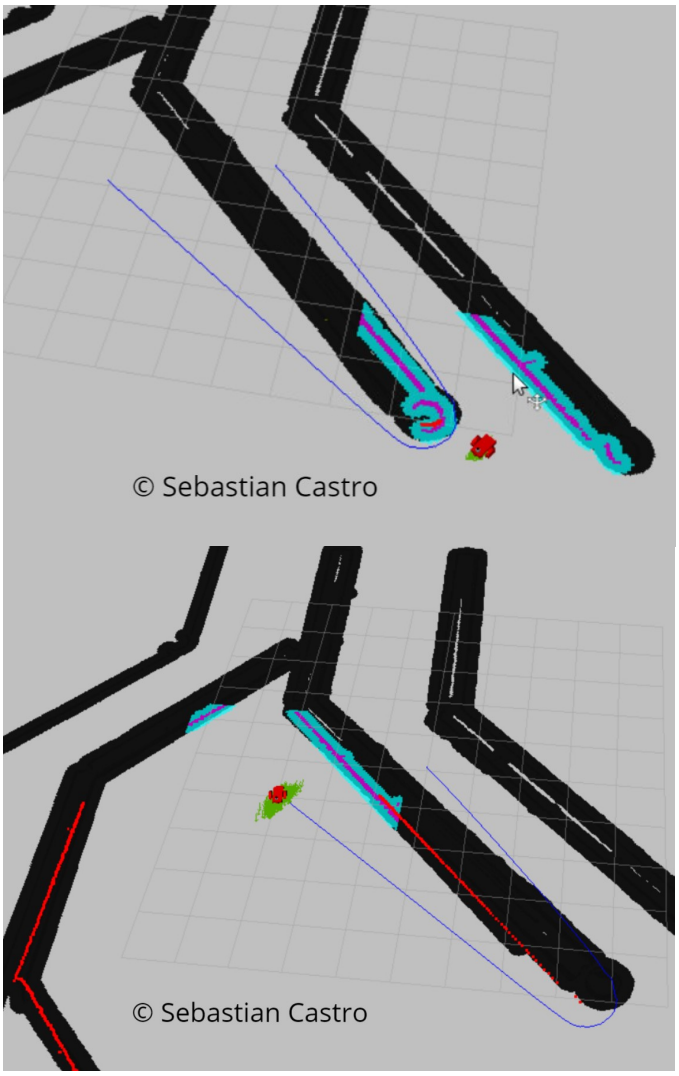

Fig. 4. Benchmark robot during navigation [top] and at the goal [bottom]

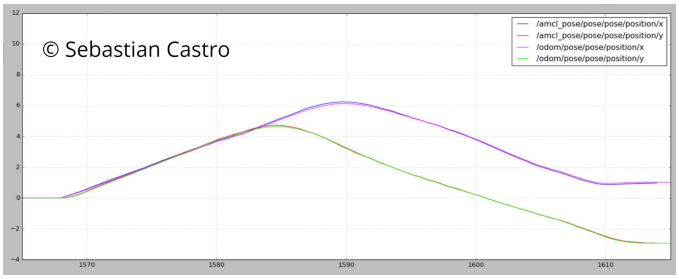

Fig. 5. Benchmark robot localization plot

5 Discussion

5.1 Performance Comparison

In terms of computational load, both robots should have been equal. While the personal robot has a 360 degree lidar scan view, the angular resolution was halved, so both robots have the same number of total beams. Also, all the localization and navigation parameters (number of particles, update thresholds, frequencies, and costmap size/resolution) were selected to the be same. Therefore, this should not be a factor in justifying the differences in performance.

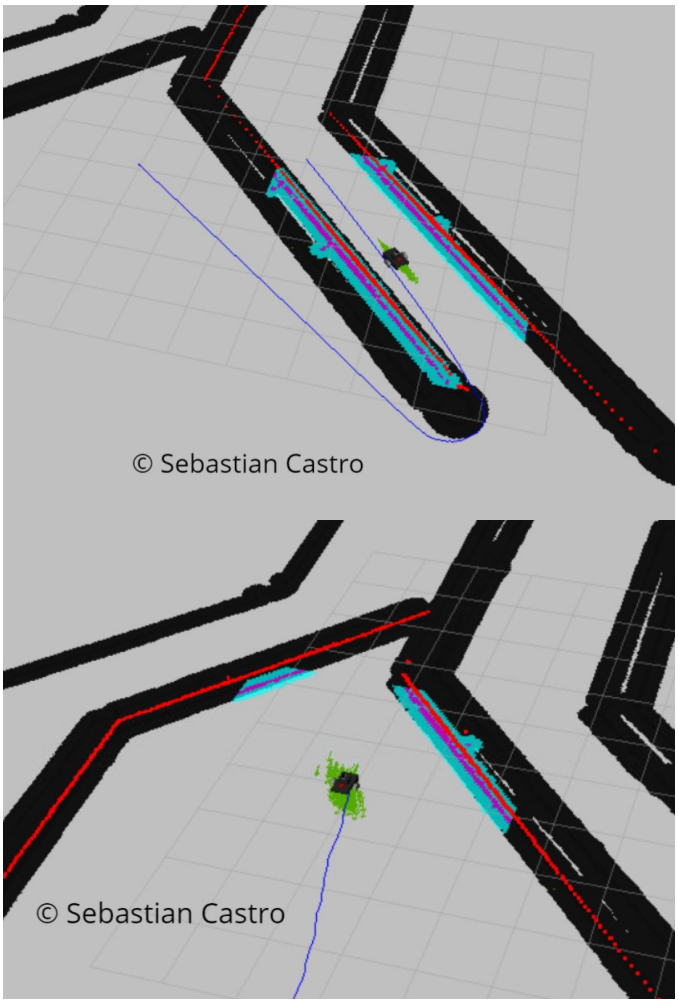

Fig. 6. Personal robot during navigation [top] and at the goal [bottom]

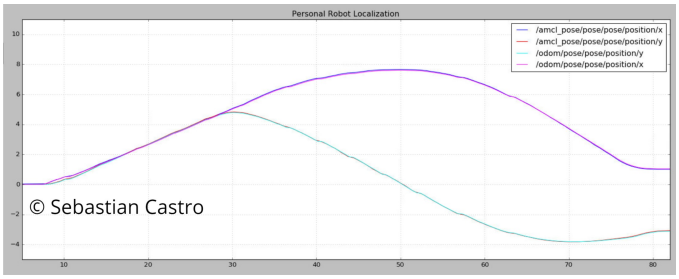

Fig. 7. Personal robot localization plot

It is believed that the robot layout contributed significantly to the discrepancy in performance. These are similar reasons for why the map inflation radius had to be changed for the personal robot.

Mechanical parameters – The personal robot has a larger wheelbase and smaller wheel radius, which made it less responsive to sharper turns. This caused obstacle avoidance in the initial corridor to have a few more oscillations, but also is believed to be a main factor for why the path to the goal after the initial corridor was much wider and continuously needed to be adjusted.

Sensor positioning and controller tuning – The lidar sensor in the benchmark robot was located near the front of the robot, which helped with clearing the wall near, and directly in front of the robot, right around the starting position. More importantly, the `navigation_goal` program was likely tuned for the benchmark robot configuration – that is, with obstacle avoidance thresholds that assumed a lidar near the front of the chassis, and with a control strategy that mostly drove the robot forward unless an obstacle was directly in front of the robot.

With this largely forward-driving controller, having a full 360 degree lidar range is not necessarily an advantage, though having half the angular resolution and a sensor farther from the front of the chassis seemed to be a disadvantage for obstacle avoidance and navigation.

5.2 Global Localization: The Kidnapped Robot Problem

Traditional particle filters sample particles based on the previous pose belief, incremented accordingly by a prediction step using the mobile robot dynamics. To improve convergence of the algorithm, importance weights for resampling are then assigned based on the likelihood that the current measurement can be produced from each particle pose.

However, this approach is not robust to the *kidnapped robot problem* – that is, if the robot is suddenly transported to a significantly different location on the map. Fox, Thrun, et al. [4] have introduced ways to address the kidnapped robot problem by instead sampling directly using the current sensor measurement. This approach would better addressed the kidnapped robot problem, but could cause issues if there are errors in an individual sensor measurement because only the current measurement is used to resample particles. Fox, Thrun, et al. then describe a way to combine both types of sampling for robustness, which could be implemented to solve the kidnapped robot problem.

5.2.1 Real-World Applications

As described briefly in the introduction, localization is important in many robotics applications because a good estimated pose is often a necessary quantity in several downstream algorithms and decision-making processed for autonomous systems.

"Pure" localization – that is, one in which the map is known a priori – can be useful for real-world applications where a constrained environment is adequate. Some examples may include:

- Industrial settings, such as automated assembly lines and warehouses
- Personal service settings, such as home robots, museums, or robot-guided tours. Note that, while these environments have a known map, there needs to be a good obstacle avoidance or map adaptation algorithm to handle dynamic obstacles such as people, furniture, etc.
- Automated driving applications, especially assuming the use of GPS enabled maps where road, intersection, and individual lane locations may be known.

In other scenarios such as outdoor, deep-sea, or cave exploration, or disaster relief in damaged environments too risky for human intervention, we often do not have a known map. One option is to send a teleoperated (human-controlled) robot to first map the environment, but another

is to send an autonomous robot equipped with a SLAM algorithm to perform exploration and localization simultaneously.

6 Conclusion / Future work

This paper presented an approach for adaptive Monte Carlo localization (AMCL) for simulated ground robots. Various considerations were explored, such as selection of robot sensor and actuator locations, Kalman filter vs. particle filter localization, and tuning of algorithm parameters. These results were verified by using the estimated pose from localization to navigate two different simulated robots between two points on a known map.

Below, two categories of future work are discussed.

6.1 Modifications for Improvement

Mechanical design – There are Gazebo plugins for several types of robots besides differential drive robots, including "tricycle" robots and four-wheeled robots with wheel steering. However, these plugins do not directly publish odometry data for use with the `amcl` package because odometry is more challenging to calculate compared to a simple differential drive robot. One future area of interest is the use of holonomic robots to navigate paths and obstacles more efficiently – specifically, building a custom model with 3 or more omniwheels.

Sensor configuration – As described in the results comparison section, the 360 degree lidar in the personal robot could potentially be more useful for localization, and also to detect obstacles with navigation planners that permit reverse motion. However, this could also be coupled with a high-resolution, low range front-facing lidar specifically for obstacle avoidance.

Algorithm Design – With the 360 degree lidar, the navigation script could be tuned such that a robot could move backwards and adjust its steering instead of tuning in a wide arc to correct itself along a path. Also, As discussed in the previous section, AMCL is not sufficient to solve the kidnapped robot problem. Some additional global localization techniques could be explored to improve the robustness of our localization algorithm.

6.2 Hardware Deployment

The ROS software architecture provides a good starting point as far as creating a modular software component design that could also work in hardware. Specifically, localization and navigation are in separate nodes that execute on parallel threads. This is important because particle filters can be computationally expensive, especially with a large number of particles, and this cannot block execution of more critical components such as obstacle avoidance and low-level control.

One thing that ROS is not designed for is embedded, real-time systems. If the target robot has sufficient computational power to run an operating system that support ROS, then perhaps the simulation software architecture could be mostly retained. However, targeting an embedded system may require the re-implementation of many software components, particularly the middleware layer that communicates with the real robot's sensors and actuators (which also would not be using ROS in this case).

Just as implementation of these components with Gazebo required tuning of algorithm frequencies, transform tolerances, map sizes, and sensor resolutions, the constraints imposed by the onboard robot computer and real sensor hardware would dictate these in the real-world case. If the simulation was not created with the final robot hardware constraints in mind, this may mean retuning of these parameters on the deployed system to ensure localization and navigation work adequately.

Another consideration is memory usage. As number of particles, map size, and map resolution grow, this requires more memory. On a desktop computer, this was not a significant issue for the scale of this problem. However, depending on the onboard hardware, this may also limit the possible design range of these parameters.

References

[1] J. Borenstein, H. R. Everett, L. Feng, and D. K. Wehe, "Mobile robot positioning: Sensors and techniques," *J. Field Robotics*, vol. 14, pp. 231–249, 1997.

[2] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard, "Simultaneous localization and mapping: Present, future, and the robust-perception age," *CoRR*, vol. abs/1606.05830, 2016.

[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] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Monte carlo localization for mobile robots," in *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*, 1999.

[5] S. Julier and J. Uhlmann, "A new extension of the kalman filter to nonlinear systems," in *Proc. of AeroSense: The 11th Int. Symp. on Aerospace/Defense Sensing, Simulations and Controls*, 1997.

[6] E. A. Wan and R. V. D. Merwe, "The unscented kalman filter for nonlinear estimation," pp. 153–158, 2000.

[7] D. Fox, "Kld-sampling: Adaptive particle filters and mobile robot localization," in *In Advances in Neural Information Processing Systems (NIPS*, 2001.