Where Am I?

Lorenzo Bermillo

Abstract – In this paper, the topic of localization is discussed and applied to two robots in simulation using ROS, Gazebo, and RViz to successfully traverse through a given maze and reach the desired goal pose and orientation. The localization technique specifically used in the simulation is the Adaptive Monte Carlo Localization, or AMCL, a Particle filter-based algorithm, which is compared to another algorithm, the Kalman filter. The resulting performance of each robot is then compared in terms of efficiency, and the design of the created robot is discussed along as to why certain parameters were chosen to improve accuracy of localization for each robot.

1. Introduction

The goal of this project is to accurately localize a robot's pose and orientation as the robot navigates through a maze and successfully reaches its desired destination in simulation. This is a challenge, however, due to noisy sensor readings from the robot caused by flawed motors or undesired environment conditions. The use of localization algorithms helps the robot filter out the noise read from its sensors allowing the robot to correctly localize itself within a given environment, assuming that the algorithm was tuned properly.

To properly test this goal, the Adaptive Monte Carlo Localization was applied to two robots in simulation. One that is predefined for the project, *udacity_bot,* therefore setting the benchmarks for performance, and the other was designed to meet or exceed the standards set by the predefined robot, this robot shall be known as *mini_r2*.

2. Background

Localization, as described in the book ***Probabilistic Robotics***, "is the problem of estimating a robot's coordinates relative to an external reference frame." Another way to put it, localization is the "process of establishing correspondence between the map coordinate system and the robot's local coordinate system." The importance of this problem is vital because almost all tasks in robotics involve the knowledge of the location of objects being controlled.

Localization problems are mainly caused by noisy sensors and flawed actuators, this creates a challenge to directly measure the robot's pose and orientation. Instead, to determine the pose, the data from the robot is deduced. This can be possible through the use of a localization algorithm to filter the noise away from the sensor readings and the combine data over time to establish the robot's pose. The two most commonly known algorithms used in robotics are the Kalman Filter and the Particle Filter.

2.1 Kalman Filter

The Kalman Filter is an estimation filter technique used to estimate a value of a variable in real time as the data is being collected such as a robot's pose. This filter is very prominent in control systems due to its accuracy and computational efficiency. However, for systems to achieve an accurate estimate of a variable, these three assumptions must be met:

1. The **state transition probability** must be **linear** with added Gaussian noise.
2. The **measurement probability** must also be **linear** with added Gaussian noise.
3. The **initial belief**, represented by the mean and covariance, must be **normally distributed**.

However, most mobile robots execute nonlinear motions, and this poses a problem for the Kalman Filter. It is crucial for the correctness of the Kalman Filter that

observations are linear for all states, so for nonlinear problems we extend the Kalman Filter by linearizing the nonlinear function.

The Extended Kalman Filter, or EKF, linearizes this nonlinear problem via Taylor Expansion. This approximates the nonlinear function by a linear function tangent at the mean of the Gaussian. The linear estimate is only valid for a small section of the nonlinear function, however, if the linear function is placed at the best estimate, the mean, and updated at every step, the resulting estimates becomes truer. Thus, the EKF inherits from the Kalman Filter's fundamental belief representation but the EKF differs in that it's certainty is only an estimate rather than correct, as the case with the Kalman Filter.

2.2 Particle Filter

The Particle Filter is another localization algorithm that randomly and uniformly spreads particle within the entire state space. Each particle represents a guess of the robot's pose and orientation. In addition to the pose and orientations, each particle also contains a weight which is the difference between the actual and estimated. The value of the weight for each particle corresponds to the accuracy of each guess. The Particle Filter works by sampling from the set of particles, with particles with larger weights having a better chance of survival during the resampling process and particles with smaller weights are more likely to die in the process.

The Monte Carlo Filter, or MCL, is a particle-based filter that represents each belief with a set of particles. Each iteration takes the previous set of particles, but for the initial belief, it is obtained my randomly generating particles. For each particle, the motion update is computed by calculating the particle's hypothetical state, then the measurement update by calculating the weight of the particle using the latest sensor readings. Both updates are then added to the previous state. The next part of the MCL is when the resampling process occurs. This is where particles with higher weights tend to have a higher chance of surviving while ones with lower weights die off. After this, a new belief is created, and the process starts again.

2.3 Comparison

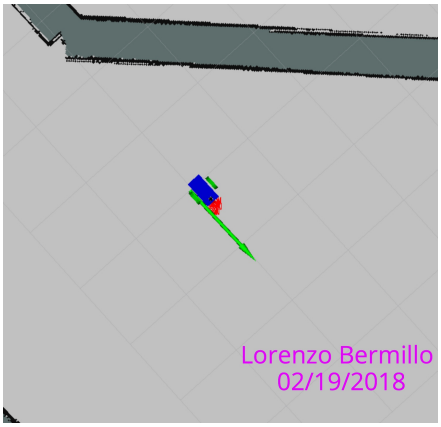
EKF is great for position tracking problems and work well if the position uncertainty is small. If the EKF is implemented for the right system, it is more time and memory efficient compared to the MCL. However, the EKF performs really poor in more general global localization problems. Global localization is where the MCL shines, making it more robust than the EKF. By also adding random particles in the particle set, it also solves the kidnapped robot problem. The accuracy and computational costs for the MCL can be tweaked through setting the size of the particles. EKF, on the other hand, has great accuracy but is heavy in resources due to the matrix computations in the algorithm. The table below summarizes the comparison of the two implementations:

	Extended Kalman Filter (EKF)	Monte Carlo Localization
Measurements	Landmarks	Raw measurements
Measurement noise	Gaussian	Any
Posterior	Gaussian	Particles
Efficiency (memory)	++	+
Efficiency (time)	++	+
Ease of Implementation	+	++
Resolution	++	+
Robustness	-	++
Global localization	No	Yes

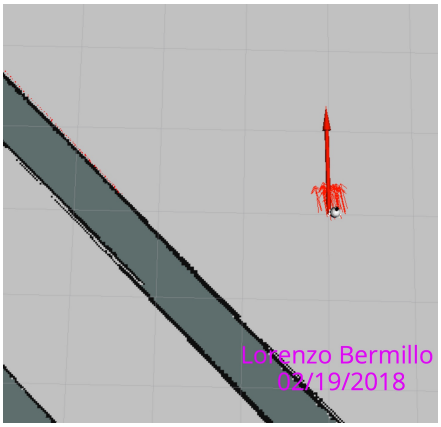
*Comparison Table 8.6 from *Probabilistic Robotics*

3. Results

Both robots reached the goal after tweaking parameters. Both robots' particles were uniformly spread at the start of the simulation and eventually narrowed down to a small group of particles as the robots proceeded to the goal. At the goal, the robots' particles had a tight grouping pointing towards the orientation of the goal.

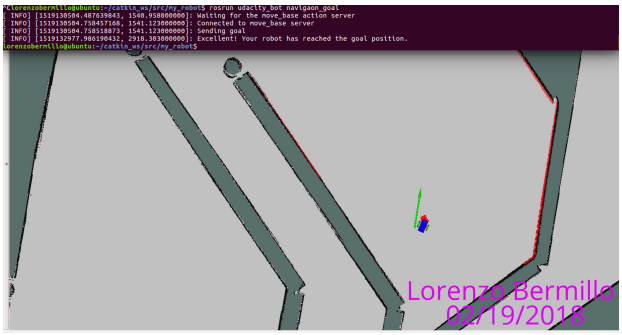


udacity_bot at goal

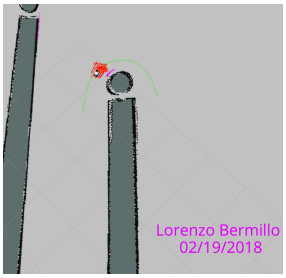


mini_r2 at goal

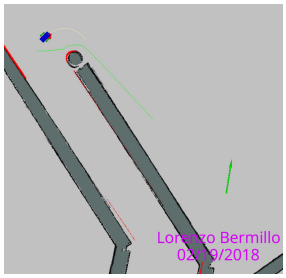
As seen from the images above, the *udacity_bot* had a tighter group of particles at the goal and are oriented better compared to the *mini_r2*, indicating that the *udacity_bot* was tuned for accuracy. Although the *mini_r2* wasn't as accurate as the *udacity_bot*, it did reach the goal significantly faster than the *udacity_bot*. The *mini_r2* successfully completed the maze and arrived at its destination in about 70 seconds compared to the *udacity_bot* which took 1377 seconds (~23 min).



The *udacity_bot* veered of the path often and frequently stopped. When it stops, it will turn in place for several minutes before it continues back to its original path. This behavior is the main reason the *udacity_bot* took a while to reach its goal. It was also observed that the *mini_r2* followed the global path much closer than the *udacity_bot* as seen below when the robots turn the corner:



mini_r2



udacity_bot

4. Model Configuration

4.1 Parameters

The parameter choices used for the robots were based on the ROS Navigation Tuning Guide. The first parameters that were tweaked were in the amcl.launch file. Here the initial pose parameters for x, y, and z were all set to 0, and initially the min and max particles were set to 100 and 1000 respectively. However, the number of particles was unnecessarily too large for the map provided. After several iterations, max particles were reduced to 500 instead. This not only reduced unnecessary particles in the simulation, but it also freed up computational resources.

The next parameters modified, and the most important in terms of localization accuracy, were the odom_alpha parameters. These parameters specify the expected noise in odometry's rotation estimate. Odom_alphas 1-4 are the only parameters tuned since the robot uses a differential drive system. These were initially all set to their default values, 0.2, and after every iteration each odom_alpha parameter was incremented by 0.1. However, it was observed that as the odom_alphas increased, the particles seemed to spread more as the robot moved instead of coming closer together. After noticing this behavior, the odom_alphas were decremented first by 0.1 from their default values. Several iterations later, the particles weren't accurate enough. After consulting the tuning guide, it seemed that the step size of 0.1 used to increment or decrement was too large and the default value of 0.2 was only used for older robot models. For newer models, the values needed to be significantly smaller. This suggestion led to the final values of the odom_alphas 1-4 as follows:

- odom_alpha1 = 0.0005
- odom_alpha2 = 0.001
- odom_alpha3 = 0.0005
- odom_alpha4 = 0.0005

Setting the parameters to these resulted to tighter packed particles as the robot moved towards the goal.

To reduce computational load, the transform_tolerance parameter was adjusted in both amcl.launch and costmap_common_params.yaml files. This parameter helps select the durability of the transform(s) being published for localization reasons and should account for any lag in the system being used. Following the tuning guide, final value of the transform_tolerance led to 0.3 to meet the system needs. Furthermore, the update and publish frequency and the map size for both local and global costmaps were reduced drastically to release computational resources. The resolution was also increased from its default value 0.05 to 0.1, this was a good enough value to reduce computational load while keeping the preferred accuracy.

To avoid hitting the barriers in the simulation, the inflation_radius parameter in costmap_common_params.yaml was set to 0.5. This ensures that the robot will keep a distance of half a meter away from the barrier to prevent the robot colliding with the barrier. Additionally, the pdist parameter in base local planner was also set to 3.5 so that the robot will follow the global path, but it'll give the robot enough freedom to recover from any mistakes.

4.2 Mini_r2 Design

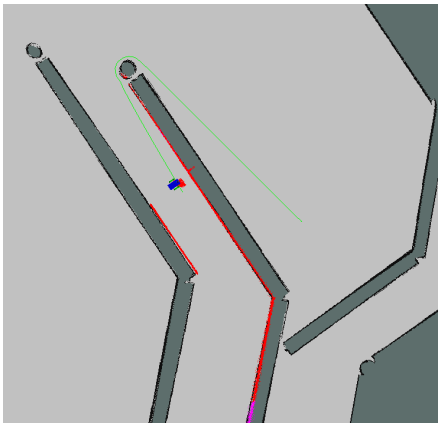
The design of the robot was inspired from Star War's R2D2. This platform worked for what was

needed so that it was possible to reposition the sensors to somewhere higher from the *udacity_bot*. To stabilize the robot, 4 wheels were used. All wheels are actuated to give the robot better handling when turning corners. Lastly, the inertial mass of the robot was much heavier compared to the *udacity_bot* due to being more upright and having a higher center of gravity. This will safeguard the robot from falling over when accelerating or decelerating.

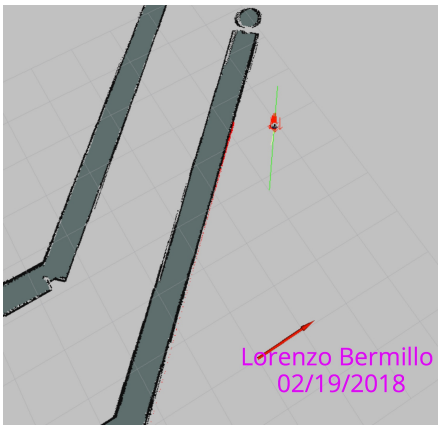
5. Discussion

Although both robots accomplished the goal, the *mini_r2* performed better than the *udacity_bot* after taking all considerations. The *mini_r2* completed the course in a much faster time while keeping the localization accurate enough to successfully navigate through the maze. The parameters used on the *mini_r2* were similar to the *udacity_bot* except for the following parameters:

These were modified to tailor to the design of the *mini_r2*. Changing the global path of the *mini_r2* to have a smaller map size proved to be very beneficial, because the *mini_r2* showed the same behavior as the *udacity_bot* of pausing and turning in place for several minutes as soon as the global path reaches the goal like shown below:



To combat this, the map size for both local and global costmaps were reduced significantly to prevent the *mini_r2* from locating the goal early. Doing so improved the *mini_r2's* time significantly compared to having a larger map size. The added benefit of also reducing the map size for both global and local costmaps was the reduced computational complexity.



Compared to the mass of the *udacity_bot*, the *mini_r2* was far heavier. This was to prevent the *mini_r2* from toppling over, but as far as performance, the mass had no effect in either of the robots' performances. There was also no significant effect in repositioning the camera and the rangefinder on the *mini_r2* since the height of the barriers were still higher than the rangefinder. Adding an extra pair of actuated wheels, however, helped *mini_r2* turn much faster and follow the global path considerably closer compared to the *udacity_bot*. This was also due to the *mini_r2*'s cylindrical design, making turning an ease compared to the *udacity_bot's* long base.

The AMCL did not do well solving the kidnapped robot problem at all. When Gazebo resets the robot's position to the robot's initial position after it has reached its goal, the AMCL algorithm was not able to locate the robot. This is something that this algorithm needs to take account for to create a robust localization algorithm. The MCL algorithm will be well suited in industries such as distribution centers or warehouses such as the Amazon Warehouse. These have flat surfaces and are easily mapped for the robot to use for navigation and localization purposes. The shelves also act as

barriers to help the robot navigate through the given path.

6. Future Work

Using the AMCL algorithm, both robots were able to accomplish the goal of this project successfully. The particles initially were spread out around the robots indicating uncertainty, but as the robot navigated through the maze, the algorithm became more confident in locating the robot's pose thus the particles began to group closer and closer. When the robot reached the goal, the particles were tightly grouped under the robots and on the goal's location, pointing in the goal's direction. In this project, it was observed that accuracy and processing time have an inverse relationship as the results proved comparing the two robots. Compared to the *udcacity_bot,* the *mini_r2* was less accurate but reached the goal significantly faster. There may be little to no adjustments needed to further enhance the accuracy of the localization software wise in the current system. In terms of hardware however, a more powerful CPU or an addition of a GPU would be helpful to ease the localization and navigation computations. The next steps to this project is to implement the software on a physical robot using the Jetson TX2. This will be more involving since it now involves hardware, software, and physical properties, such as friction and gravity, which will all be taken into account in designing a new robot or modifying the existing design and tweaking configurations.

Works Cited

Dieter, Fox, Sebastian Thrun and Wolfram Burgard. *Probabilistic Robotics*. 2005.
ROS. *Basic Navigation Tuning Guide*. n.d. 24 February 2018.