Robot Localization Report

Anil Patel and Cedric Kim

October 11th, 2018

1 Executive Summary

The mission at its highest level was to implement the particle filter algorithm for localizing in a known map. We wanted to be able to use a sensor, in this case the laser scanner, and a motor, the vehicle's differential drive, and a predetermined lap and combine them to estimate the position of our robot in the map.

In the context of ROS we are essentially building a coordinate transform that can be used for a variety of different functions. Being able to map the robot's "true" position in terms of the map allows for controls such as path planning and waypoint definition to easily be added, so long as the coordinate transformation is communicated in a clear, portable way.

2 Problem Solving

We started by understanding the algorithm from a high-level architecture and broke the challenge down into a few components. We started with the necessary operations that needed to take place and in their execution order. Next we decided on any nodes, classes, and functions that would be required to achieve our plan. A block diagram of our strategy is shown in Figure 2.1.

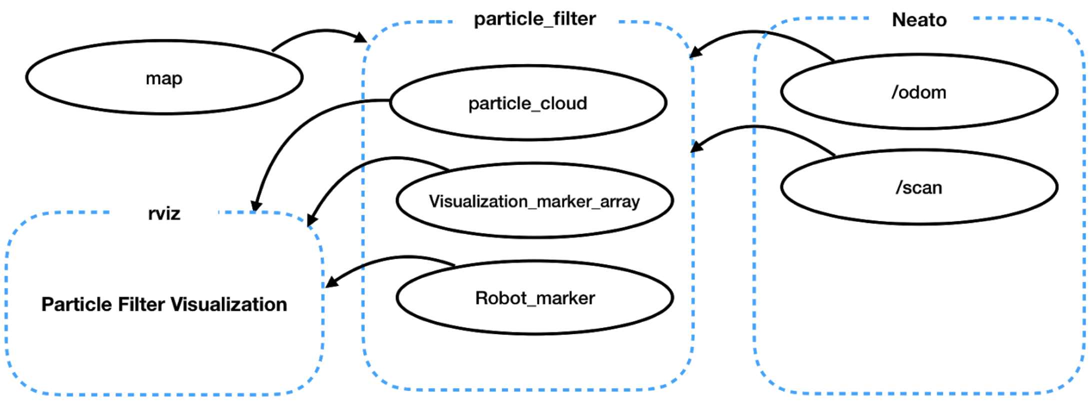

Figure 2.1: Particle Filter Block Diagram

2.1 Particle Initialization

Our particle filter is initialized in a 5x5 grid around the robots odom frame shown in figure 2.2.

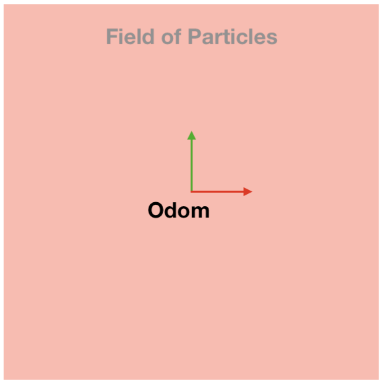

Figure 2.2: robot random location zone

The particles are randomly initialized, and given a random initial direction to start.

2.2 Laser Scan

The laser scan records the x and y values of all points within the robot’s laser scan. The robot is subscribed to laserscan. The laserscan runs in the background and is called every time it is published by the neato robot

2.3 Particle Filter

The particle filter updates each particle's position and weighs them according to how well the laser scan data matches up to the map. The location of the particles all converge to the robot’s estimated position.

2.3.1 Calculate Particle Weight

Particles weight is based off of the closest distance of each particles projected laser scan points to points of the map. Higher correlation = higher weight. The particle array is then sorted by highest to lowest weight

2.3.3 Resample Particles

Each particle goes through a resampling process outlined below in Section 3.1. The resampling function adds the robot’s odom movement to each particle and adds noise based on a distribution. The resampling is only done every half seconds to allow for some odom movement between resampling

2.3.4 Visualize Particles

Ten particles of the highest weight are shown in the visualization. The highest particle weight shows the heading as a blue arrow, as well as the laser scan transformed to the particles map frame. The actual robot heading is shown as a red arrow. The map frame updates as the robot moves through the map

2.3.5 Odom Transform

Before resampling particles, the odom coordinate frame is transformed to the maps coordinate frame to ensure that the particle is reference properly with the laser scan for the weight calculations.

3 Design Decisions

Two of our key design decisions were related to how we determined a particle's weight and sampled desirable points from a series of weight-sorted points.

3.1 Weight calculation

To decrease the time it took for our particles to converge on a location, we implemented an algorithm that changed the standard deviation of the normal distribution curve we used to sample our points.

Our particles were first sorted by weight, so that the highest weight was the first index. We then sampled from a normal distribution to generate an index to pull from the sorted array. We populated a new array of particles by repeatedly generating indices and indexing our sorted particle list. This returned a new array that contained duplicates of high-weighted particles and omitted low-weighted particles. The probability density function that represents the normal distribution is shown below in Figure 3.1.

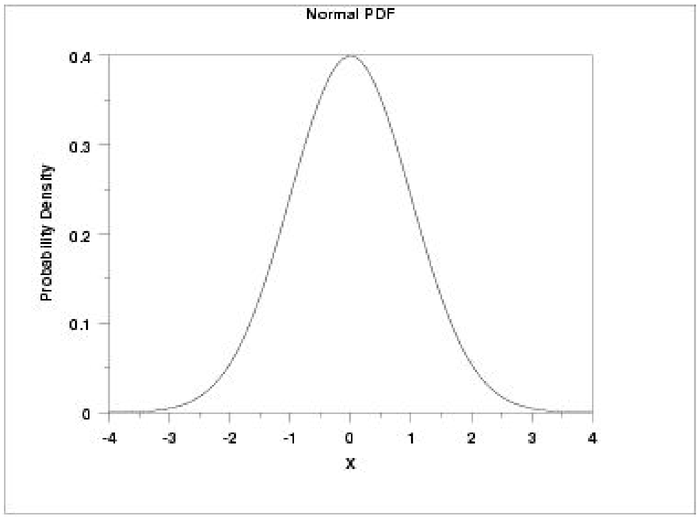

Figure 3.1: Probability Density Function

However in certain scenarios we might have a weighted list with a few very high particles, and many that were fairly low. Our sorted weights might like something in Figure 3.2

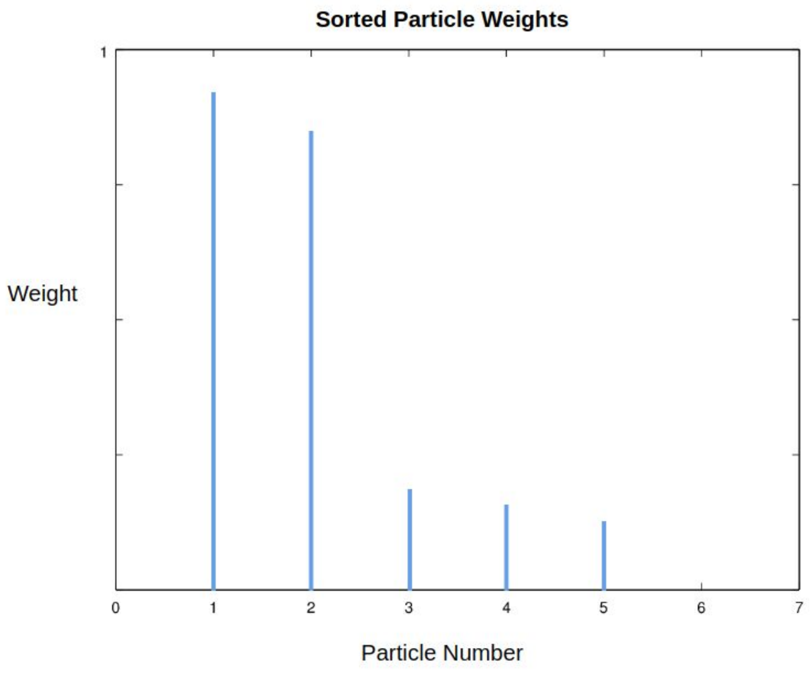

Figure 3.2: Particle Number vs. Weight

In this case if we kept the same normal distribution sampling method, we would end up with more samples from lower weighted particles. This is a result of the sharp drop in weight between the first few particles and the majority. To rectify this we can decrease the standard deviation of the probability density function that we used to generate our indices. Below shown in Figure 3.3 is a graphical depiction of how changing the standard deviation (rho) affects the normal distribution.

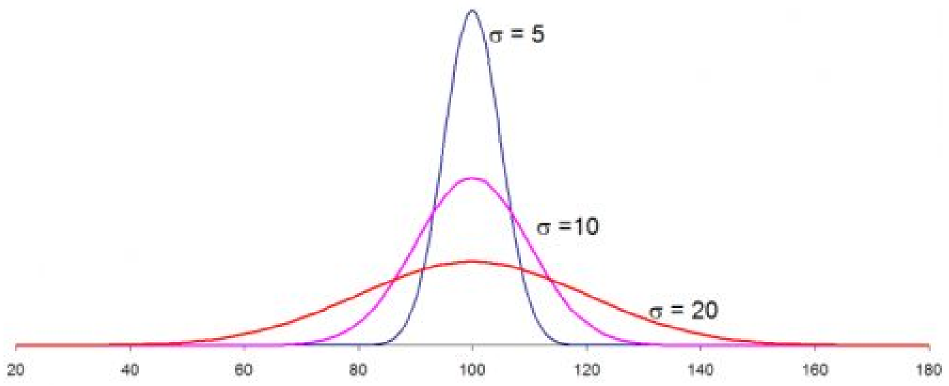

Figure 3.3: Standard Deviation

A lower standard deviation meant that we would bias even further towards our highest weights, meaning our particles would converge much more quickly. Conversely when we initialize the particle filter, and there is a large spread of lower weights, the algorithm maintains this spread until a favorable match is found. This avoids a rapid convergence on a false positive particle.

We experimented with a few different methods of defining the standard deviation in terms of the sorted weights. A function that saw significant improvement in convergence behavior was:

$$standard\ deviation = 1 - {weight_0}^{1/4} \qquad \textit{equation 1}$$

Where $weight_0$ is the highest normalized particle weight. With more time, we would define quantitative metrics for convergence timing and evaluate different definitions of STD.

4 Challenges

One challenge we had was visualizing all of our particles and particles angle. We were unable to debug our functions without knowing the exact locations and angles of each particle. Once we were able to visualize our particles, we were able to double check that our coordinate transform was working as intended and all of our particles were independently moving along the odom track. In addition, we had a hard time mapping our laser scan on top of each particle to debug our code. Once we figured that our, we were able to make sure our weight function was working as intended

Another challenge was building our particle functions without a solid understanding of our coordinate frames. One example of how this slowed us down was our interpretation of the /odom topic. When were propagating our particles using the /odom topic, but were transforming the motion to each respective particle as if the /odom messages were defined in map frame. This led us down a rabbit hole of debugging that was founded on incorrect assumptions. Once we properly understood /odom it was simple to correctly transform to each particle’s position and location.

5 Results

We visualized the results in Rviz. Figure 5.1 shows the initial particle locations for the first 10 particles.

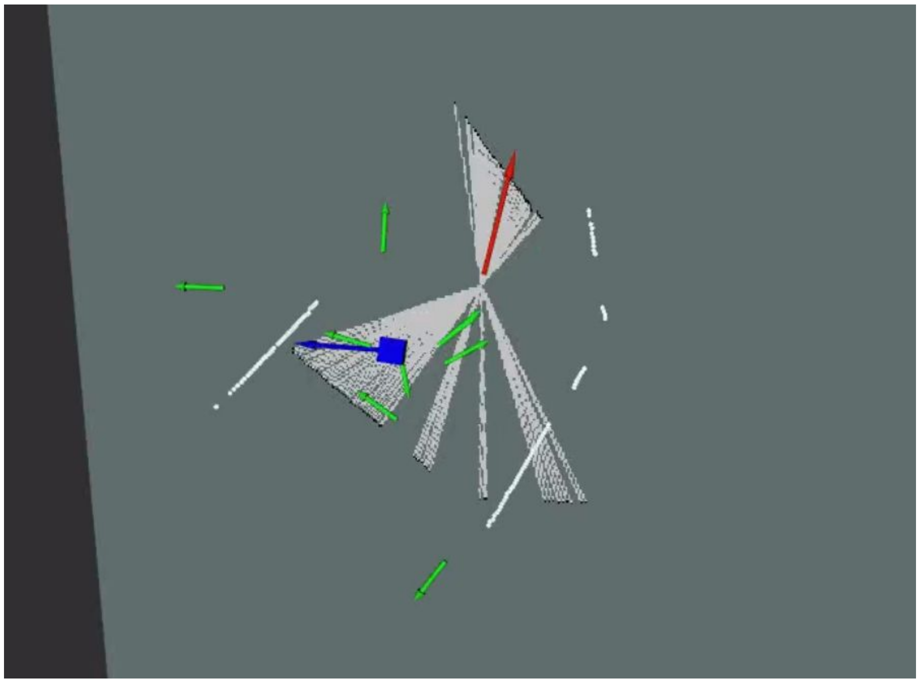

Figure 5.1: Initial Particle Locations

Figure 5.2 shows the converged particle locations after the robot moves for a certain amount of time. The blue arrow is very close to the red arrow.

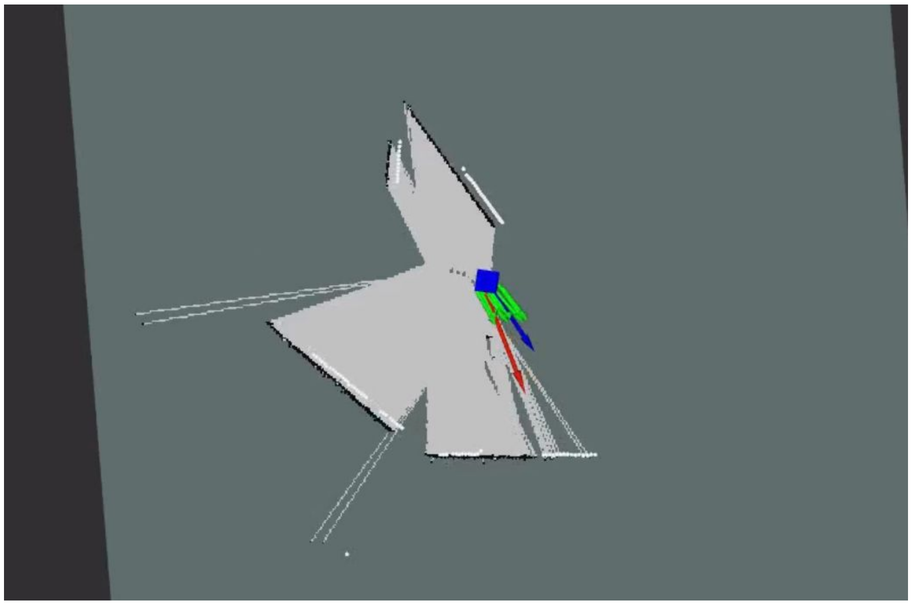

Figure 5.2: Particle Convergence

6 Improvements

The biggest area of improvement would be our weighting calculations and our convergence timing. Even with large clouds of scattered particles initialized it can sometimes take multiple seconds to properly converge on the correct point, which would be unacceptable in some use cases. Additionally, we were subjected to some false-positive convergence, which could be an artifact of how we developed our weighting algorithm.

If we had more time we would have liked to build more quantitative evaluation of our algorithms. Measuring convergence timings, decreasing latency between pose updates, and measuring error would have allowed more informed design decisions in some of our calculations. Our latest implementation was mostly experimentally and qualitatively tuned. While particle filtering behavior is effective and functional, there may be certain use-cases where higher precisions and update frequencies would be required.

7 Lessons Learned

We learned the power of building robust and correct visualization tools as a tool for developing and debugging code. Developing tools that were easily manipulated and adjusted for tuning and fixing code would have alleviated many errors in the making. ROS has many tools that we did not use to their fullest capacity early enough in the development process. It was only until we got our laser scans properly transformed and displayed that we could effectively understanding errors in our weighting calculations.

Additionally, we learned the value of writing code that is aligned with ROS standards. The more formally we adhered to conventions in coordinate transformation, visualization topics, and debugging tools, the easier it would be to add additional ROS tools into our code, use included functionality for solving our issues, and develop our code to be more understandable and portable for future projects.