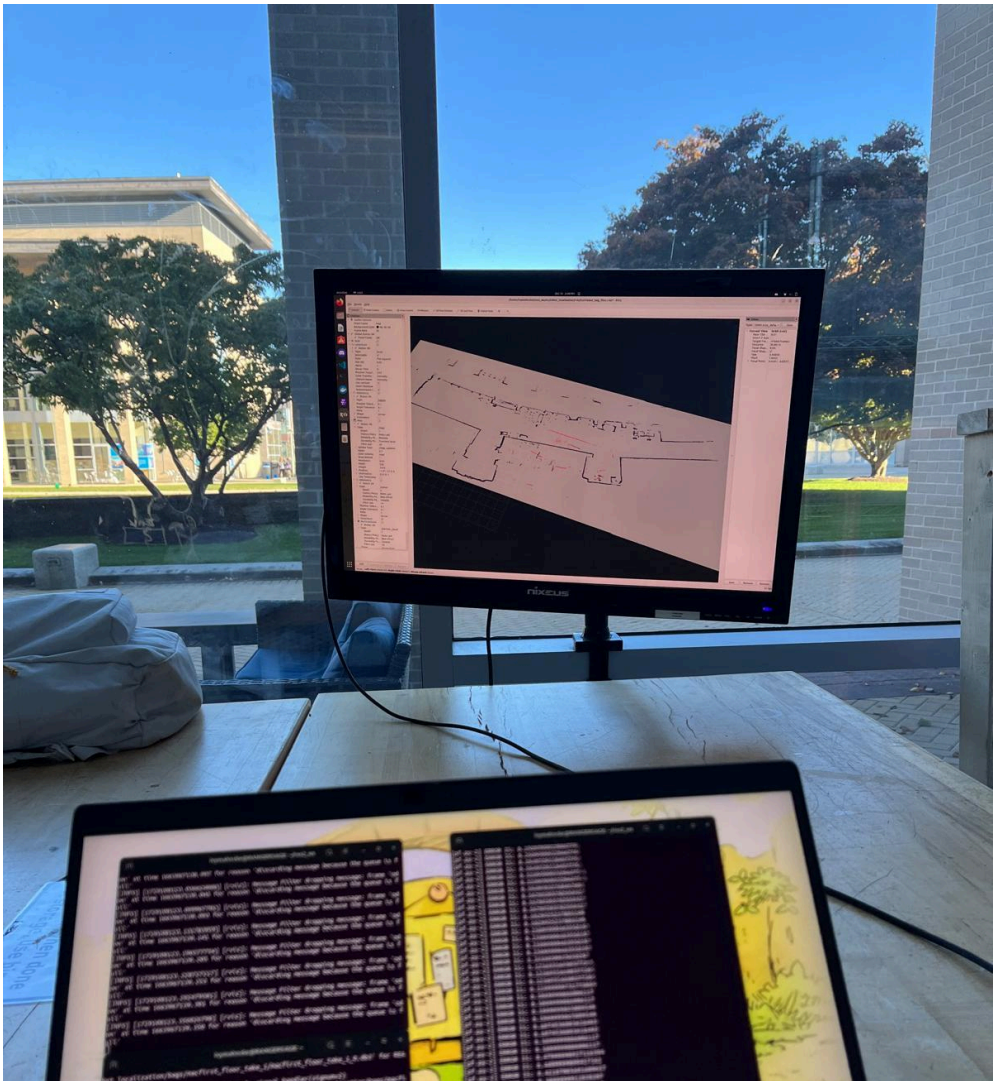
Ivy Mahncke
Vivian Mak

Localization Project

Introduction to Computational Robotics
Fall 2024



Running the particle filter algorithm on the 1st floor MAC in the 1st floor MAC.

1 Project Goals

The robotics field is a very complex field as it integrates many components to have a robot do more complex tasks. Having access to higher-level environment information and decision-making capabilities become critical to making sure robots complete the things we intend them to. Localization is what we are diving into for this project, implementing the particle filter.

The problem to solve is to determine the Neato's location in a given recorded environment with laser scans of nearby objects. The particle filter comes into play for the prediction aspect of this problem. A particle filter generates *n* amount of particle, each with a position and heading (*x,y,theta)* attributes, as an estimate of the Neato's location. In addition, a weight is assigned to each particle determining how likely that particular estimate is the real location. The bigger the weight, the larger the probability the Neato is in that position. Just like in any other system in real life, nothing is completely accurate, so noise is introduced to mitigate this. The resampling of the particles causes them to converge and eventually determine final coordinate and direction of the Neato. The approach to sampling is very open ended as there are an infinite amount of ways to solve this problem. Section 2 provides a more detailed explanation of how we approach the particle filter.

Our project goals were to build a strong understanding in one of the many algorithms to solve the robotics localization problem, the particle filter, and get more exposure to ROS2 and debugging strategies.

2 Design Decisions

2.1 Update Particles with Odom

Given a fresh odometry update, we have to update the particles' position on the map to reflect the robot's movement.

For this function, we chose to update the particles' position with some noise added in addition to the position change supplied by the odom message. We separated the heading noise rate from the x and y noise rate because it operated on a fairly different scale. When tweaking the noise rates, we found that higher noise rates caused the particles to converge much slower. This is good in a sense, because particle death does not happen so rapidly that a faulty estimation gets chosen early on. However, a higher noise rate didn't lead to a more accurate convergence later on. We ended up choosing an xy noise rate of 0.1 and a heading noise rate of 0.5, which delayed particle convergence somewhat.

2.2 Update Particles with Laser

Given a laser scan, we have to update the particles' weights given the error of the laser scan to the closest obstacle distance.

From the odom frame, the pose of the Neato is given along with its 360 laser scan in the lists *r, theta*, with r being values of distances of objects from the current position, and *theta* being corresponding angles of the scan. With the polar coordinates, we want to turn them into cartesian coordinates to be able to transform the scans onto each particle. This is done with the following equations:

- $x = r * cos(\theta)$ for the x cartesian coordinate, and
- $y = r * sin(\theta)$ for the y cartesian coordinate

Once the scan points are in terms of *x,y,theta,* the projection to each particle is just a simple rotation-translation transformation with the following equations:

- $x = (x * cos(p.\theta)) - (y * sin(p.\theta)) + p.x$
- $y = (x * sin(p.\theta)) - (y * cos(p.\theta)) + p.y$

The projection of the scans allows us to compare the current Neato's scan with the environment for each particle. The helper function *get_closest_obstacle_distance( )* returns the error of each point of the scan with the closest point on the map. We emitted *nan* and *infinite* values of *r* to solve division by zero issues when computing the total error. We used a simple reciprocal which allows large errors to give a smaller weight and small errors to give a larger weight. The following equation was used to determine the weight of each particle:

$$p.w = \frac{1}{min(1000, (sum(p_{error})/len(p_{error})^2)^2}$$,where p_{error} is the error list

Using this to determine each weight gives us a more amplified number to more accurately determine our robot's pose. If the error list is empty, meaning the particle is off the map, the weight is assigned to *0.001* meaning the probability of the particle is as low as possible. After all the computation of updating particle weights, we normalize the weights to ensure all sum to one.

2.3 Update Robot Pose

To update the robot pose, we have to use our weighted particles to determine a best estimate for where our robot is.

As an MVP, we first simply selected the particle with the highest weight. This worked somewhat well, but didn't account for the situation in which multiple particle clusters with similar weight distributions formed.

We updated this MVP with an alternate solution, in which we chose the three particles with the best weights and found the median particle position across their poses. We chose to do median instead of mean such that the estimate, when faced with multiple clusters, would end up in one cluster rather than in the empty space between. This solution resulted in an improved estimate

of robot pose within our distribution of particles; however, it did not solve the issue of consistently poor estimates.

2.4 Resampling Particles

Once the robot pose is updated, we want to resample the particles while adding noise to them. In the first few iterations, the particles are spread across the map but as time passes and weights get updated, the particles eventually converge to a more accurate estimate of the robot's location. To resample the particles, we used the helper function *draw_random_sample( )* which takes in the particles in the particle cloud and their corresponding weights and generates a new set of particles. Gaussian noise is added to each particle to add more variance to the particle cloud. After some trial runs, we figured out that the *theta* noise should be higher than the *x,y* noise because even though the projected laser scan overlapped with the map, the heading suddenly changed to a random angle.

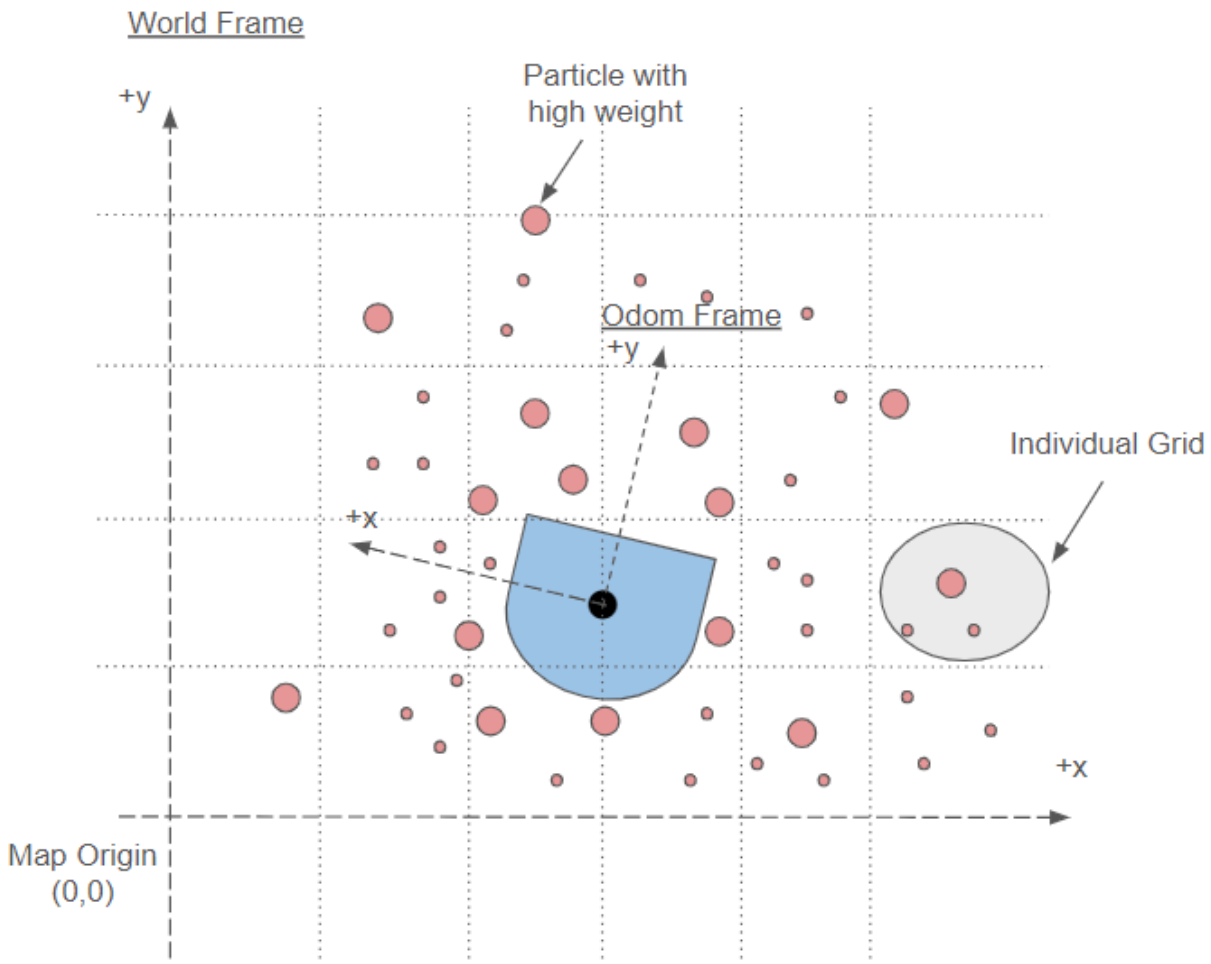
3 Challenges

One challenge we encountered was too-quick particle convergence. We realized that this was likely due to the fact that our unlikely particles were dying too quickly. With a near-instant particle convergence, our filter had no way to improve a faulty estimate, and our particles could never re-converge on a more accurate estimate later.

Another difficulty we encountered was understanding how to properly compare the particles' theoretical laser scans to the actual laser scans. We initially misunderstood the rotation and translation of the particle scan points, which meant we couldn't properly weight our particles. Luckily, we caught this mistake and rewrote the calculations for the particle pose translation such that the particle scans were properly overlaid with the robot scans.

4 Future Improvements

Resampling is one part that can be significantly improved. As the MVP, we decided to resample all of the particles. An algorithm we wanted to test out was to divide the map into a grid and find the location of the robot that way. The main advantage of this is to mitigate concerns of converging at the wrong location. For example if there is a particle with a higher weight further from the actual position, taking the mean of that would alter the result of the guess. Taking the mode would also be challenging because there are an infinite amount of possibilities of numbers from 0 to 1. So this was an algorithm we believe that can avoid these issues. From each square on the grid the mean of the particle weights are calculated. After finding the square of the highest weight, the mean of the *x,y,theta* will then be taken to estimate the final location of the robot.



Resampling algorithm using a grid to determine robot pose

Additionally, we think that a contributing factor to our particle filter's inability to make a proper estimate was the proportion of particles we resampled. Because we resampled 100% of the particles every time, we quickly lost any unlikely particles that were potentially better pose estimates than the initial likely particles. If we resampled only a portion of our particle cloud – like 50%, or even 40%, we could maintain more unlikely floater particles, which would allow more clusters of possibility to compete for weight.