ECE 276A W21: PR2 Particle Filter SLAM

Patrick Youssef
University of California, San Diego
psyoussef@ucsd.edu

1 INTRODUCTION

In this paper we will discuss an approach to solving the simultaneous localization and mapping (SLAM) problem for a mobile vehicle and attempt to understand the shortcomings of the given implementation. Possible improvements will be suggested along the way alongside the state of the current results.

1.1 Importance of SLAM

The problem of localizing a mobile robot within an unknown map is one that has many novel uses in modern robotics. In many applications we don't have the luxury of a predetermined map, but the usage of landmarks and correspondence can help to maintain a correct localization. A solution to this problem falls into the family of problems called SLAM. A utility of this method is for navigation and creation of a map for any unknown or changing space where a map is either not available or needs to be updated realtime.

1.2 Summary of Approach

There are a variety of ways to solve this family of problems but we are tasked to solve this using a particle filter for state prediction and lidar correspondence for landmarks within the map. The process of creating the map and maintaining a position within the map is as follows:

(1) Initialize map with first lidar scan and start coordinate system there.
(2) Initialize particles with the same state as the vehicle $[0, 0, 0]$ and equal weights.
(3) Predict the state of a set of dispersed particles and measure the degree of correspondence of each particle against the map up to the point.
(4) Normalize the weights of the particles such that the sum of all weights is 1.
(5) Predict new robot state with a weighted average of the particle states.
(6) Update the map with a new prediction of the robot state given the best approximation of state. Repeat 3-6.

2 PROBLEM FORMULATION

Here we will discuss more precisely the problem we are trying to solve and the associated mathematical quantities that will help us.

2.1 Problem Statement

In this problem we are given sensor data from a mobile vehicle and are tasked with predicting the state of the vehicle over time using a map of its surroundings that we generate along the way. We are given the data of 3 different sensors: a fiber optic gyroscope (FOG), wheel encoder readings, and a front-scanning lidar.

2.1.1 Localization Problem. Localization deals with understanding the position of a robot in a given map.

- Input: A given map m and a sequence of controls u
- Objective: Obtain a sequence of states x of the robot's path

2.1.2 Mapping Problem. Mapping deals with logging the landmarks of a robot's surroundings.

- Input: A sequence of states x and observations z
- Objective: Obtain a map m of the robot's surroundings

2.1.3 SLAM Problem. SLAM is doing both localization and mapping with a limited understanding of both.

- Input: A sequence of controls u and observations z
- Output: A sequence of states x and a map of surroundings m

2.2 General Methods

Here I will discuss generally how you might solve the SLAM problem mentioned prior.

2.2.1 Bayes Filter. A particle filter is a Bayes Filter that has two steps, a prediction step shown in equation 1 and an update step shown in equation 2.

$$p_{t+1|t}(x) = \int p_f(x|s, u_t) p_{t|t}(s) ds \tag{1}$$

$$p_{t+1|t+1}(x) = \frac{p_h(z_{t+1}|X) p_{t+1|t}(x)}{\int p_h(z_{t+1}|s) p_{t+1|t}(s) ds} \tag{2}$$

A particle filter is a special kind of bayes filter that does not assume anything about the world.

2.3 Metrics

2.3.1 Lidar Correlation. This metric is used to determine a score for how well a projected lidar set matches with a given map. This is later used to determine scores for the particles based on lidar projection from their state. It's as follows:

$$corr(y, m) = \sum_i \mathbf{I}(y_i = m_i) \tag{3}$$

Where y is the set of projections, m is the map, and $\mathbf{I}$ is the indicator function.

2.3.2 What I Would Like. Other than our intuition there is no real way to check against the true path or true map. I would have liked to have a ground truth path to gather an MSE of the filtered path.

2.4 Assumptions

2.4.1 Markov Independence. We assume Independence in the joint distribution $p(x_{0:T}, z_{0:T}, u_{0:T-1})$.

$$Joint = p(x_0) \prod_{t=0}^{T} p_h(z_t|x_t) \prod_{t=1}^{T} p_f(x_t|x_{t-1}, u_{t-1}) \prod_{t=0}^{t-1} p(u_t, x_t) \tag{4}$$

3 TECHNICAL APPROACH

Here I will discuss in depth how the methods mentioned prior are implemented to solve the particle filter SLAM problem.

3.1 Data Cleaning

A good first step is to process the data into a form that makes this problem more tractable and allow for easier correspondence between sensors. We'll go through the step taken for each sensor's data.

3.1.1 Fiber Optic Gyroscope. The data from the FOG is in $rad\backslash s$ which is not immediately useful to us. It's also sampled 10x faster than both the lidar and encoders. I chose to compute $\Delta\theta$ as $\tau\dot{\theta}$ and summed the changes over every 10 samples to bring the samples inline with the other sensors.

3.1.2 Wheel Encoders. Wheel encoders were more straightforward than the FOG. Given $\frac{ticks}{rev}$ and the sample time it was easy to compute τ and $\Delta Ticks$ to determine the wheel linear velocity at each sample time as:

$$\dot{x} = \frac{\Delta Ticks\, \pi\, D_{wheel}}{\Delta\tau\, TPR} \tag{5}$$

3.1.3 Lidar. Lidar did not have much preprocessing as much of the processing happens during the mapping steps. All I did was reform the data structures here to be inline with how I present the data from before.

3.1.4 Across All Sensors. Once I had all sensors cleaned up and at the same sampling rate I trimmed the data such that I had the largest data set where all sensors were sampled and times were inline.

3.2 Dead Reckoning

Although dead reckoning is not necessary for the particle filter SLAM, it's a good step to build some intuition of the data and many of the steps translate well into the SLAM predict step. The integration steps done here are the base of updating the particle states in the prediction step.

3.2.1 Motion Model. The motion model used to describe the rates of the states of the vehicle is a differential drive model. The motion model is as follows:

$$\dot{X} = \left[v\cos(\theta), v\sin(\theta), \dot{\theta}\right] \tag{6}$$

Where v is the centerline velocity (average wheel velocity) and theta is the heading of the robot.

3.2.2 Euler Integration. Once the rates were determined at each time I compute the changes in the 3 states over each sample and applied a cumulative sum to get the estimated state over time. The changes were computed as an euler integration:

$$X_{t+1} = X_i + \tau\dot{X} \tag{7}$$

where X is a vector of states and $\dot{X}$ are the state derivatives from the motion model.

3.3 Particle Filter SLAM

Here we will go into detail about how the implementation for particle filter SLAM was formed. I will relay the steps that map well to function defined in my code that I determined as distinct steps in this process.

3.3.1 Initialize Particles. First step taken is to initialize N particles to the initial state of the vehicle $[0, 0, 0]$ with weights of $\frac{1}{N}$. As we are not trying to localize the robot in a given map it does not make sense to disperse the particles initially but to instead start them on top of the robot.

3.3.2 Lidar Projection. This helper function makes it easier to project the lidar scan into the robot coordinate system. it utilizes the rotation and translation matrix as given by the geometry of the robot. The primary equation used is:

$$X_{car} = R\,X_{lidar} + T \tag{8}$$

Where X_{car} is the 3D coordinate of a lidar scan in the car coordinates, R is the rotation matrix from the lidar coordinates to the car, X_{lidar} is the position of the scan in lidar coordinates, and T is the translation from lidar to car.

3.3.3 Initialize Map. We initialize a map of all zeros and apply the first lidar scan where endpoints increase the value in the grid by 4 and open space decreases the value of the grid by 4. The sign conventions of these changes are based on applying a sigmoid to the map later to determine the probabilities that a space if occupied.

3.3.4 Predict New Particle States. Here the particles are updated to new states following the integrated motion model output at the given time. The difference compared to dead reckoning is that we add noise to disperse the particles around the dead reckoned state to capture other possible states. The update for a given particle is as follows:

$$X_{t+1} = X_i + \tau\dot{X} + \mathcal{N}(0, \sigma) \tag{9}$$

As you can see, normal noise is added to the particles with a different standard deviation for translational noise (x and y) and rotational noise (θ).

3.3.5 Update Particle Weights. Here the particle weights are updated with the aforementioned lidar map correlation metric. This is done by applying the $t + 1$ lidar scan to all particles and seeing how well they correspond with the prior map. Weights are stored as this correlation score with the particles.

3.3.6 Normalize Weights. For the following step, we need to normalize the weights such that they sum to 1. This is done using the softmax activation method as follows:

$$W = \frac{e^{w_i}}{\sum_{j=1}^{N} e^{w_j}} \tag{10}$$

Where W is the vector of weights for all particles being overwritten by the softmax activation over each prior weight (Note: overwriting occurs after all new elements have been computed), and N is the number of particles.

3.3.7 Estimate Robot State. We now have a set of N particles and their respective weights. We approximate the new state of the robot as a weighted average of the particle states. This is as follows:

$$X_{robot} = \sum_{i=1}^{N} X_i \, W_i \tag{11}$$

We can now also compute the weighted variance of the estimation as:

$$Var = \sum_{i=1}^{N} (X_i - X_{robot})^2 \, W_i \tag{12}$$

Note: The states and variance are vectors in the prior equations.

3.3.8 Update Map. This step is very similar to the initialize map step but instead of a matrix of zeros we have the matrix of the prior map. We apply the same ±4 with the current lidar scan applied to the estimated robot state. To ensure we don't get numerical overflow and overly confident probabilities for cell occupancy we cap the values of grid sums to ±100.

3.3.9 Resample. Lastly, if need be we resample the particles via sample important resampling. This method repositions particles relative to the weights of the prior particles. This way we focus our efforts on particles that performed well.

3.3.10 Filter Map. The prior steps, starting with predict new particles, are repeated for the robot runtime but in post if you want to save the map as probabilities you can apply the sigmoid function to convert the log-odds grid sums to occupancy probabilities. This is done as follows:

$$p(m_{i,j}) = \frac{e^{m_{i,j}}}{1 + e^{m_{i,j}}} \tag{13}$$

Where $m_{i,j}$ is the sum of a particular grid element. This normalizes the sums in the grid elements to probabilities that they are occupied. This also explains why we start the matrix for the map as all zeros. When passed into the sigmoid, grids of 0 would yield $p_{i,j} = 0.5$. In other words, we are equally uncertain about whether the grid space if occupied or not.

3.4 What is The Right Noise

Given that we apply a noise in the particle prediction step we must determine what the correct amount of noise. I determined the correct amount of noise by rationalizing the purpose of the noise and looking at the data.

The noise is used to disperse the particles states to encompass states that may be a better fit with the map than the dead reckoned state. This can be due to sensor noise, motion model inaccuracies, and other reasons that would make the dead reckoning incorrect. Extending this idea, I applied noise that allowed me to encompass a fairly small error band relative to the motion model state change. Figures 1 and 2 below show the relative scale of the noise on the motion model state deltas. Table 1 lists the chosen standard deviations for the noise addition.

3.5 Texture Mapping

Although I did not get to applying texture mapping to the code stack, I wanted to discuss the means to applying this to my code.

3.5.1 What is Texture Mapping. Texture mapping is the process of applying color to the endpoints of the lidar scans to colorize the map. This can be done by matching the physical position of the lidar scan to the object color from the camera view.

3.5.2 How to Match Points. To match points between the lidar scans and the camera, we need to transform the data from each to a common coordinate system. The most convenient coordinate system for this is the robot coordinates. This is fairly trivial for the lidar as you only need to apply the translation and rotation from the given matrices, but for the camera you need to estimate depth and use projective geometry to infer the world coordinates of the image

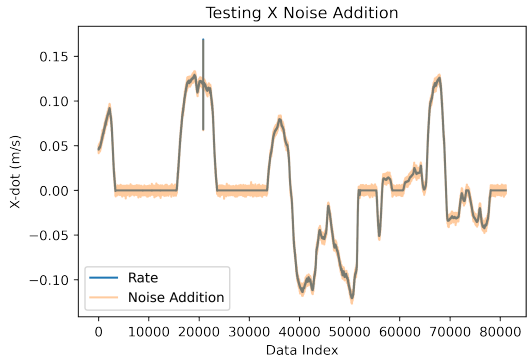

Figure 1: x motion model deltas with noise

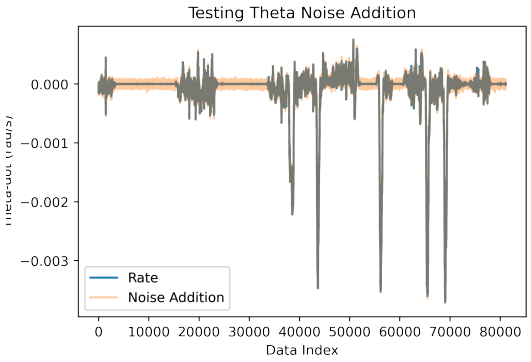

Figure 2: θ motion model deltas with noise

Table 1: Chosen Noise Standard Deviations

State	Noise Std. Dev.
x	0.002
y	0.002
θ	0.00003

points. Depth can be inferred from a disparity map on a stereo set of calibrated images. Once you have brought both sensors into a common coordinate system you merely need to match the points from the lidar to the color of the matching image point and apply said color to the map.

4 RESULTS

The results section serves to present the performance of the solution. Note that there is a video of mapping done with the dead reckoned path. My method for making the video could be applied to the particle filter mapping easily but it's quicker to produce the dead reckoned video and proves the same idea.

4.1 Summary of Parameters

To reiterate, the parameters used moving into results are as follows in table 2. Also the data was subset as the computation time was extremely long for me and I wanted to ensure that the map crossed over its own path at least once.

Table 2: Particle Filter Parameters

State	Noise Std. Dev.
x	0.002
y	0.002
θ	0.00003
Number of Particles	
N	30
Grid Resolution	
dx/dy	1 meter

4.2 Summary of Results

You will see more details moving forward but overall the particle filter SLAM showed itself to outperform the dead reckoned MAP due to its ability to correspond and correct relative to landmarks in the map.

4.3 Dead Reckon

Here we will discuss the results of the dead reckoning for robot state and how the applies when mapping.

4.3.1 Wheel Velocities. Figures 4 and 5 show the linear velocities of the left and right wheels determined from the encoder readings. Looking closer at figure 5 we can see that there is a short period of incorrect velocity measurements where we see a velocity peak higher than all the rest of the readings and almost instantly reduce to nearly 0. This could be due to a wheel slip or other factors, but these kinds of errors are why we need a filter.

4.3.2 Yaw Rates. For the robot yaw rate, it's clear to use the better catered sensor (FOG) to do the measurements, but I was curious to see how different the measurements from a yaw rate determined from the wheel velocities would be compared to the gyroscope. When comparing the FOG yaw rate in figure 7 to the wheel determined yaw rate in 6 we can see that the aforementioned discontinuity in velocity from figure 5 translates into a similar discontinuity in the yaw rate estimate. Moving forward I will only use the FOG yaw rate.

4.3.3 Wheel Velocities.

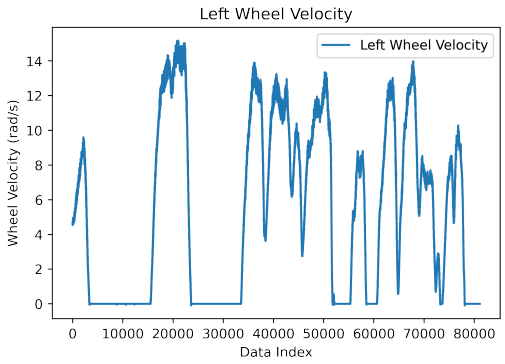

Figure 3: Left Wheel Velocity

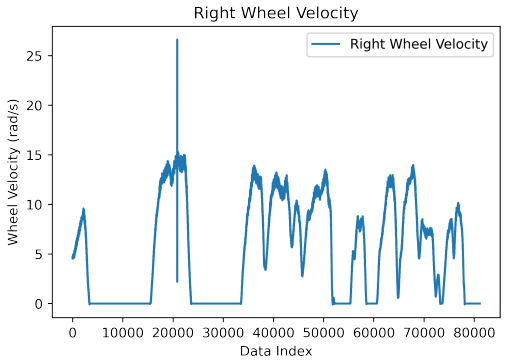

Figure 4: Right Wheel Velocity

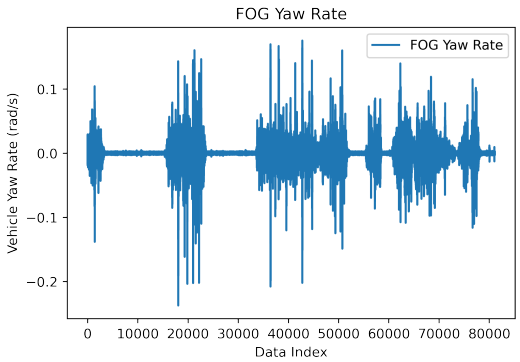

Figure 5: FOG determined yaw rate

4.3.4 Robot Path. The dead reckoned robot path is shown in figure 8. As we can see the path is actually quite good with just the dead reckoned estimates.

4.3.5 Map. With the dead reckoned path we can apply the lidar scans just like we would in the particle filter to get an impression of what the map will look like. Figure 8 shows the result of this lidar scan application.

4.4 Particle Filter

Moving into the particle filter, we have some other facets we can looks at given how the approach varies from dead reckoning alone.

4.4.1 Map. As we can see in figure 9, the particle filter SLAM based map is very similar to the dead reckon based map. We'll dive in deeper to see the differences in a following section but luckily here we're able to show the correct functionality of the particle filter.

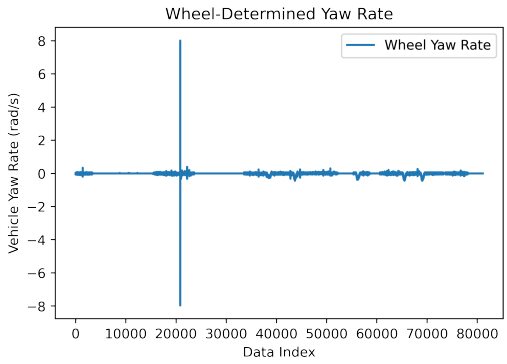

Figure 6: Wheel velocity determined yaw rate

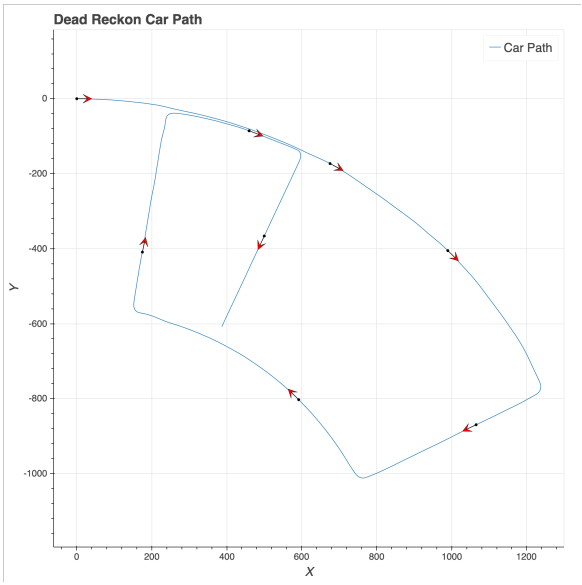

Figure 7: Dead reckoned robot path

4.4.2 Particle State Variance. Given how we use a weighed average of the particle states to determine the new robot state estimate, we also can compute a weighted variance of the particle states. The variance can be looked at as a metric of particle filter uncertainty as the two cases where variance is low correspond to a good estimate.

- A few particles have large weights and are close to each other, there is good consensus amongst a few strong particles.
- Most particles are distributed close to each other and have similar weights, there is good consensus amongst many particles.

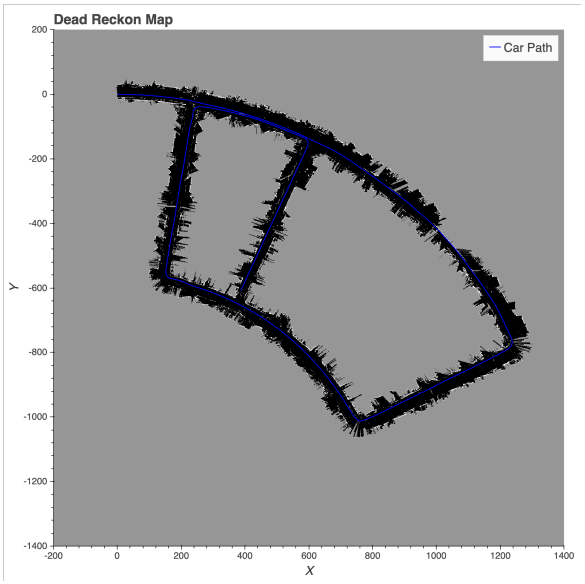

Figure 8: Map following dead reckoned path

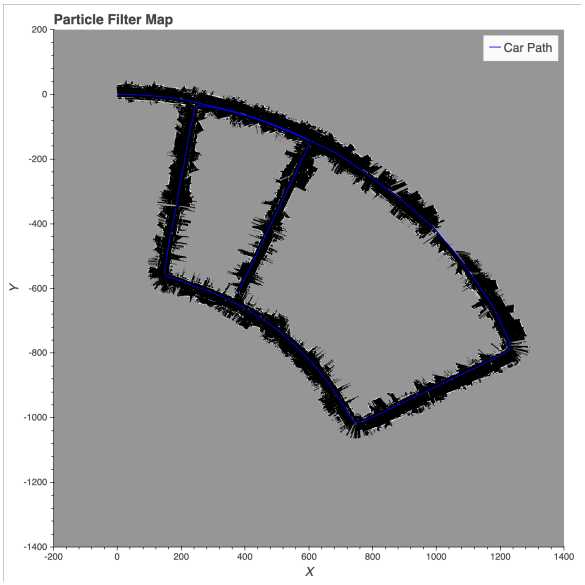

Figure 9: Map from particle filter SLAM

Figures 10, 11, and 12 show the states and variances for x, y, and θ respectively. As we can see, it would seem like the variances for x and y hold pretty consistently which gives us confidence that the estimated state is due to a good degree of consensus. θ on the other hand does not seem to be holding as well, it may be due to too large of noise for a relatively slow state change.

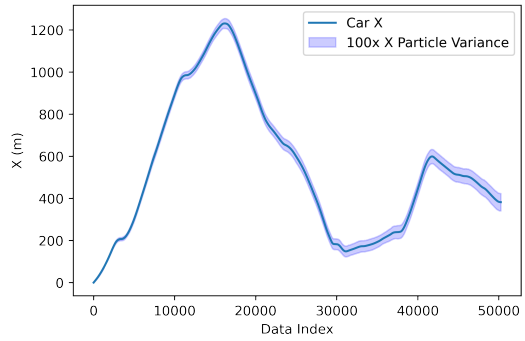

Figure 10: Estimated x state over iterations with variance

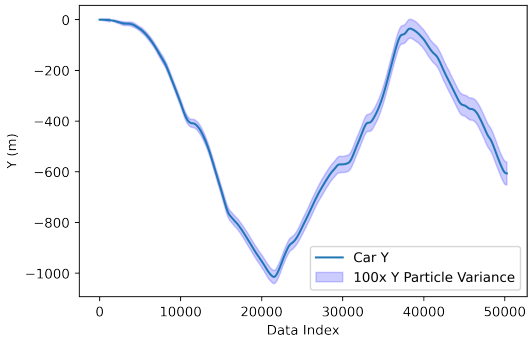

Figure 11: Estimated y state over iterations with variance

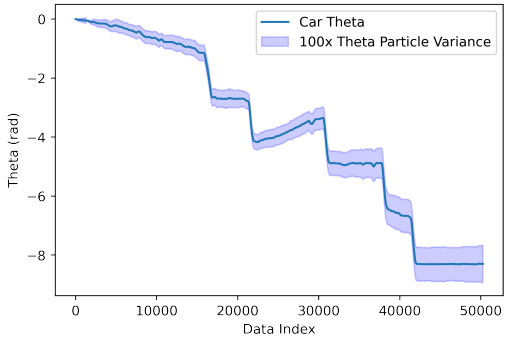

Figure 12: Estimated θ state over iterations with variance

4.5 Comparing Dead Reckoning and Particle Filter

To appreciate the subtle differences between the map from the particle filter SLAM and the dead reckon mapping I thought it would be good to look closer at the maps to see the differences. Figures 13 and 14 show that the particle filter is better able to track back into the same path as when it first passed the road whereas the dead reckon path shows signs of accumulated integration error with no way to correct for it.

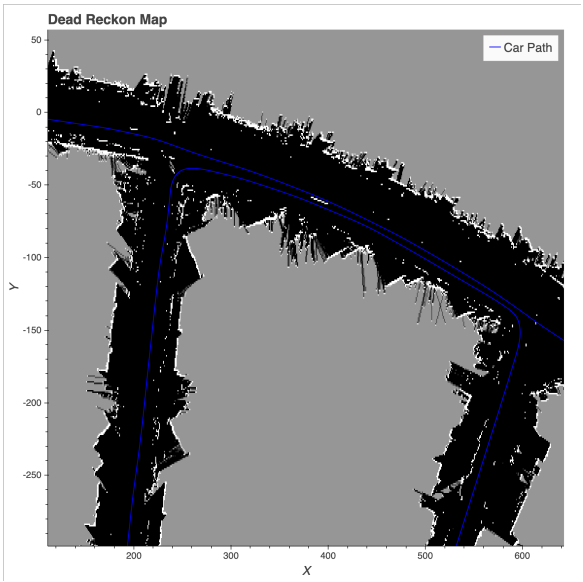

Figure 13: Map closeup from dead reckon mapping

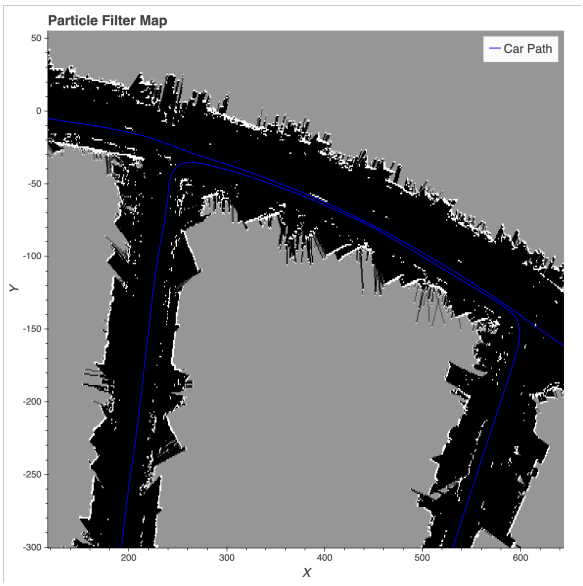

Figure 14: Map closeup from particle filter SLAM